MovieLens project for Data Science Capstone Course

Shahidul Islam

2024-11-29

Introduction

The purpose of this project is to wrangle the MovieLens dataset, describe its characteristics, and predict movie ratings based on features such as users, movies, and genres. Analytic datasets were created following the instructions provided in the course materials. The MovieLens dataset was downloaded and curated in accordance with the provided guidelines. A final hold-out test dataset was generated by partitioning the main dataset and extracting 10% of the MovieLens data using the "caret" package.

The training dataset was explored and analyzed using tools from the "tidyverse" and "sqldf" packages, focusing on distributions, counts, outliers, and other key features. Several models were developed to predict movie ratings, with complexity introduced gradually. Root Mean Square Error (RMSE) was calculated for each model and recorded in a data frame for reporting. The RMSE for the best model was 0.86445. The algroritthm for the RMSE:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Dataset

The MovieLens 10M datasets was splitted into 90% training and 10% validation datasets. The training dataset included 69878 unique users with 9000055 observations (ratings), and 10677 unique movies. The validation dataset "final_holdout_test" included 999999 ratings from 68534 unique users. This validation dataset only used for prediction, i.e., evaluating RMSE, it was not used to train any algorithm.

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(sqldf)
```

```
## Loading required package: gsubfn
## Loading required package: proto
## Loading required package: RSQLite
```

```
##########################################################
# Create edx and final_holdout_test sets
##########################################################

# 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")

# library(tidyverse)
# library(caret)
# library(sqldf)

setwd("C:/Users/ShahS/OneDrive/EDX.ORG/DataScienceCertificate/CAPSTONE")
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"),
                                   simplify = TRUE), stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

Exploring the edx dataset

To better understand the data set, we can print a few rows, examine its dimensions, inspect the variable types, and perform other exploratory checks.

*******We created the 'edx' and 'final_holdout_test' (Validation) data sets using the code above and saved them locally, as generating these data sets can be time-consuming. By loading the data from the local drive, we were able to significantly improve run time.********

```
setwd("C:/Users/ShahS/OneDrive/EDX.ORG/DataScienceCertificate/CAPSTONE")
load(file="edx.Rdata")
load(file="final_holdout_test.Rdata")

head(edx, n=5)
```

```
##   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)
##                          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
```

```
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
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|Ac
```

```
#69878 unique users have  9000055 observations in the dataset. 10677
sqldf("select count(distinct userId) from edx")
```

```
##   count(distinct userId)
## 1                  69878
```

```
sqldf("select count( movieId) from edx")
```

```
##   count( movieId)
## 1         9000055
```

```
#68534 unique users have 999999 observations
sqldf("select count(distinct userId) from final_holdout_test")
```

```
##   count(distinct userId)
## 1                  68534
```

Top 5 higest rated movies overall

```
edx %>%
  group_by(title) %>%
  summarize(N_ratings = n(), avg_ratings = mean(rating, na.rm = TRUE)) %>%
  arrange(desc(avg_ratings)) %>%
  slice_max(order_by = avg_ratings, n = 5)
```

```
## # A tibble: 6 x 3
##   title                                    N_ratings avg_ratings
##   <chr>                                        <int>       <dbl>
## 1 Blue Light, The (Das Blaue Licht) (1932)         1           5
## 2 Fighting Elegy (Kenka erejii) (1966)             1           5
## 3 Hellhounds on My Trail (1999)                    1           5
## 4 Satan's Tango (Sátántangó) (1994)                2           5
## 5 Shadows of Forgotten Ancestors (1964)            1           5
## 6 Sun Alley (Sonnenallee) (1999)                   1           5
```

##Top 5 highest rated movies with >1000 user ratings

```
edx %>%
  group_by(title) %>%
  summarize(N_ratings = n(), avg_ratings = mean(rating, na.rm = TRUE)) %>%
  filter(N_ratings>1000)%>%
  arrange(desc(avg_ratings)) %>%
  slice_max(order_by = avg_ratings, n = 5)
```

```
## # A tibble: 5 x 3
##   title                            N_ratings avg_ratings
##   <chr>                                <int>       <dbl>
## 1 Shawshank Redemption, The (1994)     28015        4.46
## 2 Godfather, The (1972)                17747        4.42
## 3 Usual Suspects, The (1995)           21648        4.37
## 4 Schindler's List (1993)              23193        4.36
## 5 Casablanca (1942)                    11232        4.32
```

##cleaning the genres variable to parse values of genres–each movie is >=1 genres, let's ensure we get the count by each genres

```
edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(N_ratings = n()) %>%
  arrange(desc(N_ratings)) %>%
  slice_max(order_by = N_ratings, n = 5)
```

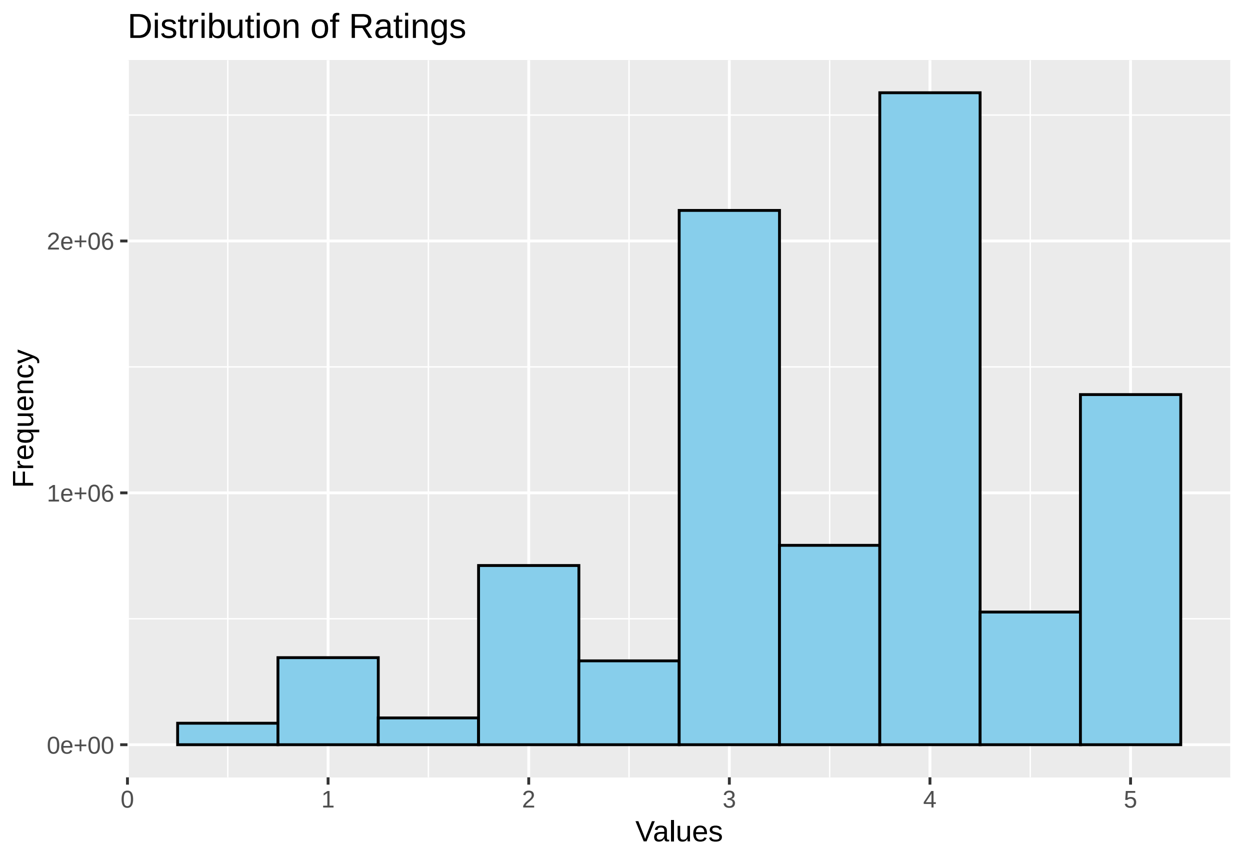
```
## # A tibble: 5 x 2
##   genres    N_ratings
##   <chr>         <int>
## 1 Drama       3910127
## 2 Comedy      3540930
## 3 Action      2560545
## 4 Thriller    2325899
## 5 Adventure   1908892
```

Distribution of ratings

Based on the distribution below, a rating of 4 is the most common, followed by 3 and 5. It is evident that some movies are rated by significantly more users than others, while some movies receive very few ratings. This type of non-normal distribution may not generalize well, potentially leading to suboptimal estimates.

To address this issue, we will later implement regularization techniques, which help reduce error by adding a penalty term to the model. This approach mitigates overfitting by preventing the model coefficients from taking on extreme values. As a result, introducing regularization leads to more robust and accurate models.

```
ggplot(edx, aes(x = rating)) +
  geom_histogram(binwidth = 0.5, fill = "skyblue", color = "black") +
  labs(title = "Distribution of Ratings", x = "Values", y = "Frequency")
```



# Checking for outliers

List of movies that received one rating only. These outliers may influence predictions

```
edx %>%
  group_by(movieId) %>%
  summarize(ratings = n()) %>%
  filter(ratings == 1) %>%
  left_join(edx, by = "movieId") %>%
  group_by(title) %>%
  summarize(rating = mean(rating), n_rating = n()) %>%
  #slice(1:100) %>%
  knitr::kable()
```

title	rating	n_rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	2.0	1
100 Feet (2008)	2.0	1

title	rating	n_rating
4 (2005)	2.5	1
Accused (Anklaget) (2005)	0.5	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di…) (1971)	1.5	1
Africa addio (1966)	3.0	1
Aleksandra (2007)	3.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1
Battle of Russia, The (Why We Fight, 5) (1943)	3.5	1
Bellissima (1951)	4.0	1
Big Fella (1937)	3.0	1
Black Tights (1-2-3-4 ou Les Collants noirs) (1960)	3.0	1
Blind Shaft (Mang jing) (2003)	2.5	1
Blue Light, The (Das Blaue Licht) (1932)	5.0	1
Borderline (1950)	3.0	1
Brothers of the Head (2005)	2.5	1
Chapayev (1934)	1.5	1
Cold Sweat (De la part des copains) (1970)	2.5	1
Condo Painting (2000)	3.0	1
Confess (2005)	2.5	1
Confessions of a Superhero (2007)	0.5	1
Cruel Story of Youth (Seishun zankoku monogatari) (1960)	2.5	1
David Holzman's Diary (1967)	4.0	1
Dead Man's Letters (Pisma myortvogo cheloveka) (1986)	3.0	1
Deadly Companions, The (1961)	4.0	1
Death Kiss, The (1933)	2.5	1
Delgo (2008)	2.0	1
Demon Lover Diary (1980)	4.5	1
Deux mondes, Les (2007)	2.5	1
Devil's Chair, The (2006)	2.5	1
Diggers (2006)	3.0	1
Diminished Capacity (2008)	1.5	1
Dirty Dozen, The: The Fatal Mission (1988)	3.5	1
Dischord (2001)	1.0	1
Dog Day (Canicule) (1984)	3.0	1
Dog Run (1996)	1.0	1
Dogwalker, The (2002)	2.0	1
Double Dynamite (1951)	2.0	1
Down and Derby (2005)	3.5	1
Du côté de la côte (1958)	2.5	1
Emerald Cowboy (2002)	3.0	1
Face of a Fugitive (1959)	3.0	1
Fallout (1998)	2.0	1
Family Game, The (Kazoku gêmu) (1983)	4.0	1
Father Sergius (Otets Sergiy) (1917)	3.0	1
Fighting Elegy (Kenka erejii) (1966)	5.0	1
Fireproof (2008)	2.0	1
Fists in the Pocket (I Pugni in tasca) (1965)	4.0	1
Flandres (2006)	3.5	1
Flu Bird Horror (2008)	1.5	1
Fools' Parade (1971)	3.0	1
Forgotten One, The (1990)	3.5	1

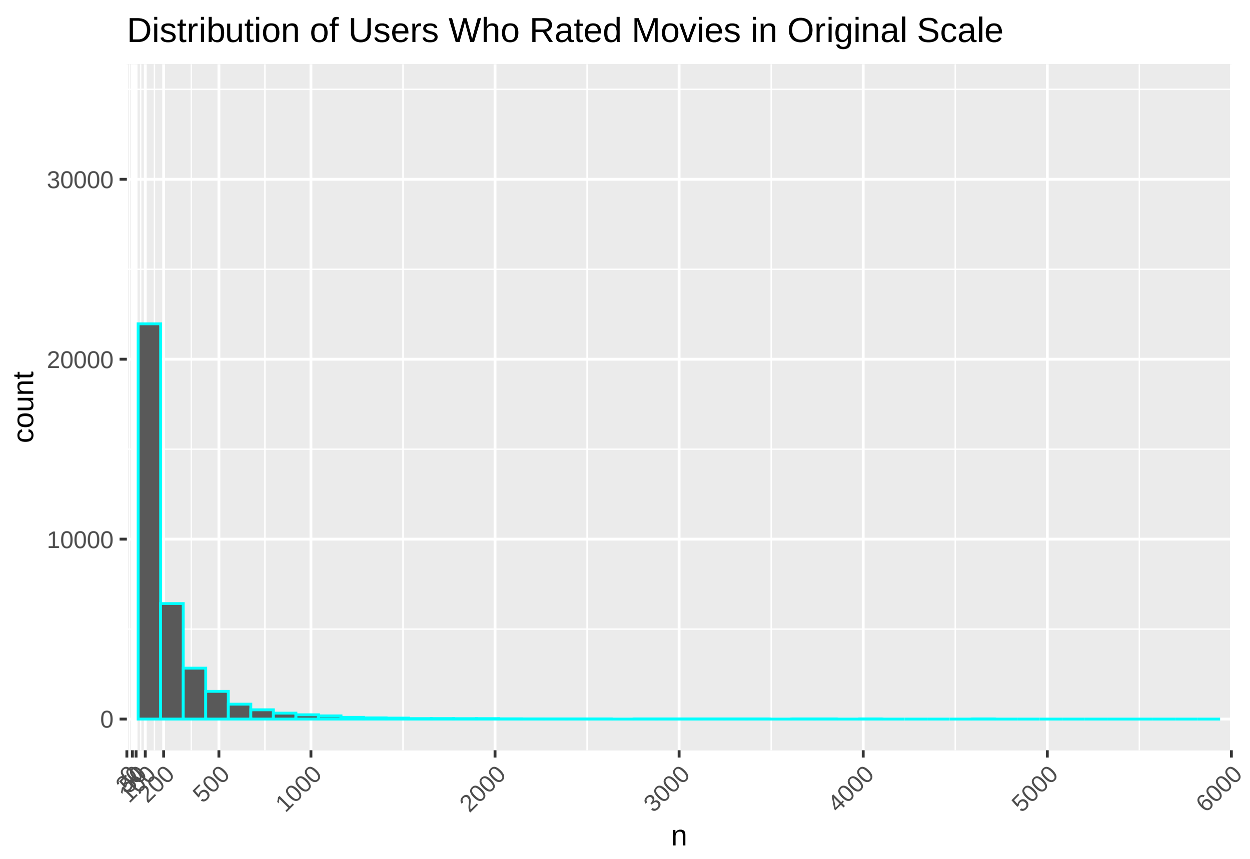
title	rating	n_rating
Forty Shades of Blue (2005)	3.5	1
Gaucho, The (1927)	3.5	1
God’s Sandbox (Tahara) (2002)	3.5	1
Gold Raiders (1951)	2.0	1
Guard Post, The (G.P. 506) (2008)	2.5	1
Hellhounds on My Trail (1999)	5.0	1
Hexed (1993)	1.5	1
Hey Hey It’s Esther Blueburger (2008)	4.0	1
Hi-Line, The (1999)	0.5	1
Hundred and One Nights, A (Cent et une nuits de Simon Cinéma, Les) (1995)	3.5	1
Impulse (2008)	4.0	1
In Bed (En la cama) (2005)	2.5	1
In the Winter Dark (1998)	3.5	1
Jimmy Carter Man from Plains (2007)	4.0	1
Just an American Boy (2003)	2.5	1
Kanak Attack (2000)	4.0	1
Kansas City Confidential (1952)	4.5	1
Krabat (2008)	4.0	1
Ladrones (2007)	4.5	1
Last Time, The (2006)	3.5	1
Lessons of Darkness (Lektionen in Finsternis) (1992)	3.5	1
Living ’til the End (2005)	3.0	1
Long Night, The (1947)	3.0	1
Love (2005)	3.5	1
Love Forbidden (Défense d’aimer) (2002)	2.5	1
Love Life (2001)	3.5	1
Mala Noche (1985)	4.0	1
Malaya (1949)	3.0	1
Man Named Pearl, A (2006)	4.5	1
Man of Straw (Untertan, Der) (1951)	4.0	1
Mesmerist, The (2002)	3.5	1
Mickey (2003)	4.5	1
Monkey’s Tale, A (Les Château des singes) (1999)	1.0	1
Moonbase (1998)	2.0	1
Mr. Wu (1927)	3.0	1
Much Ado About Something (2001)	4.0	1
Music Room, The (Jalsaghar) (1958)	4.0	1
Nazis Strike, The (Why We Fight, 2) (1943)	3.5	1
Neil Young: Human Highway (1982)	1.5	1
Once in the Life (2000)	3.0	1
One Hour with You (1932)	3.0	1
Part of the Weekend Never Dies (2008)	3.5	1
Please Vote for Me (2007)	4.5	1
Quarry, The (1998)	3.5	1
Quiet City (2007)	3.5	1
Relative Strangers (2006)	1.0	1
Ring of Darkness (2004)	3.5	1
Rockin’ in the Rockies (1945)	2.0	1
Shadows of Forgotten Ancestors (1964)	5.0	1
Small Cuts (Petites coupures) (2003)	3.0	1
Splinter (2008)	4.0	1
Stacy’s Knights (1982)	1.0	1

title	rating	n_rating
Stone Angel, The (2007)	2.5	1
Strange Planet (1999)	2.0	1
Sun Alley (Sonnenallee) (1999)	5.0	1
Sun Shines Bright, The (1953)	3.5	1
Symbiopsychotaxiplasm: Take One (1968)	3.5	1
Säg att du älskar mig (2006)	3.0	1
Tattooed Life (Irezumi ichidai) (1965)	3.5	1
Testament of Orpheus, The (Testament d'Orphée) (1960)	4.5	1
Tokyo! (2008)	4.5	1
Train Ride to Hollywood (1978)	3.0	1
Twice Upon a Time (1983)	3.5	1
Uncle Nino (2003)	4.0	1
Valerie and Her Week of Wonders (Valerie a týden divu) (1970)	4.5	1
Variety Lights (Luci del varietà) (1950)	4.0	1
Vinci (2004)	4.0	1
When Time Ran Out... (a.k.a. The Day the World Ended) (1980)	1.0	1
Where A Good Man Goes (Joi gin a long) (1999)	4.0	1
Won't Anybody Listen? (2000)	2.0	1
Young Unknowns, The (2000)	2.5	1
Zona Zamfirova (2002)	4.0	1

looking at the distribution of users

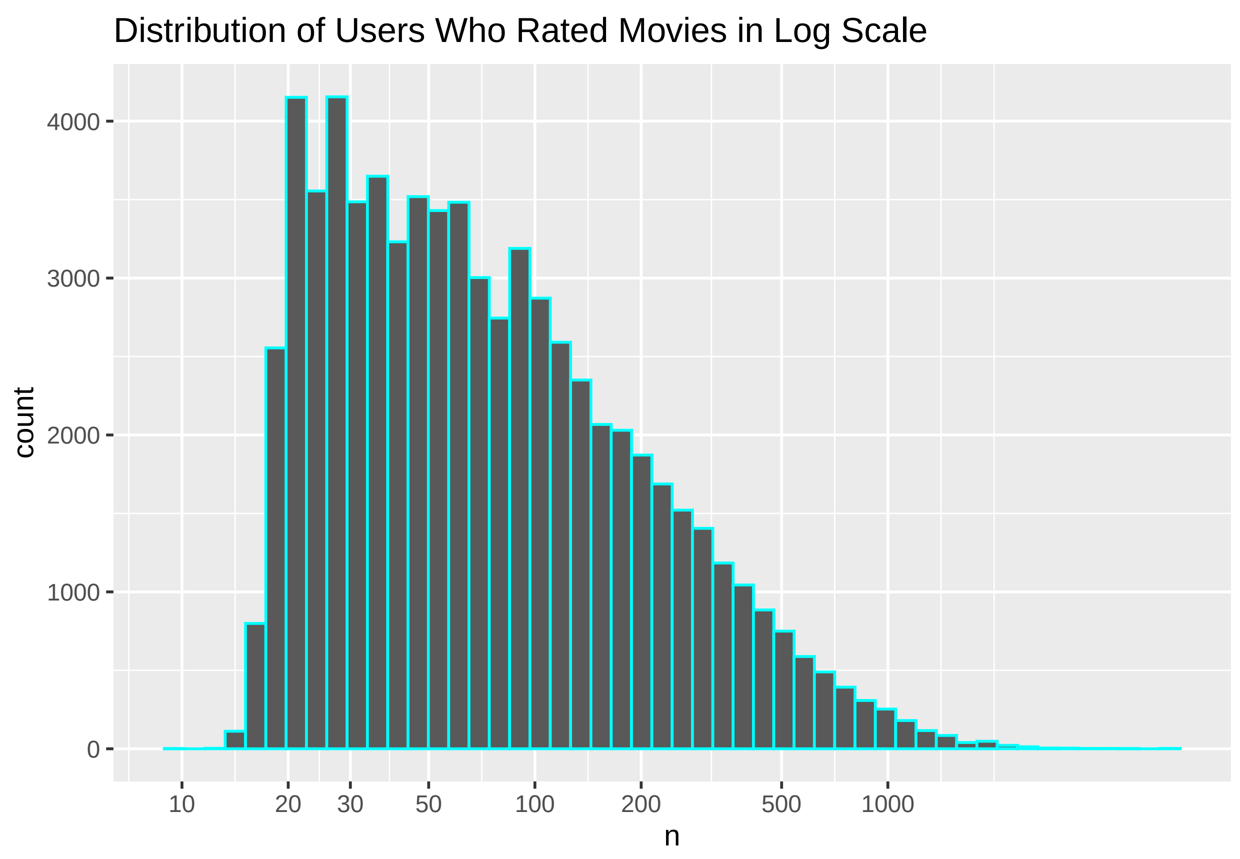
```r
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 50, color = "cyan") +
  ggtitle("Distribution of Users Who Rated Movies in Original Scale") +
  scale_x_continuous(
    breaks = c(0,30, 50, 100, 200, 500, 1000, 2000, 3000, 4000, 5000, 6000), # Exact breakpoints
    labels = c(0, 30, 50, 100, 200, 500, 1000, 2000, 3000, 4000, 5000, 6000), # Matching labels
    limits = c(0, 6000), # Ensure the x-axis range
    expand = c(0, 0) # Remove extra padding around the axis
  ) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) # Rotate x-axis labels if needed
```

```
## Warning: Removed 2 rows containing non-finite values (`stat_bin()`).

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



Distribution of users in a log scale to get better view of the distribution

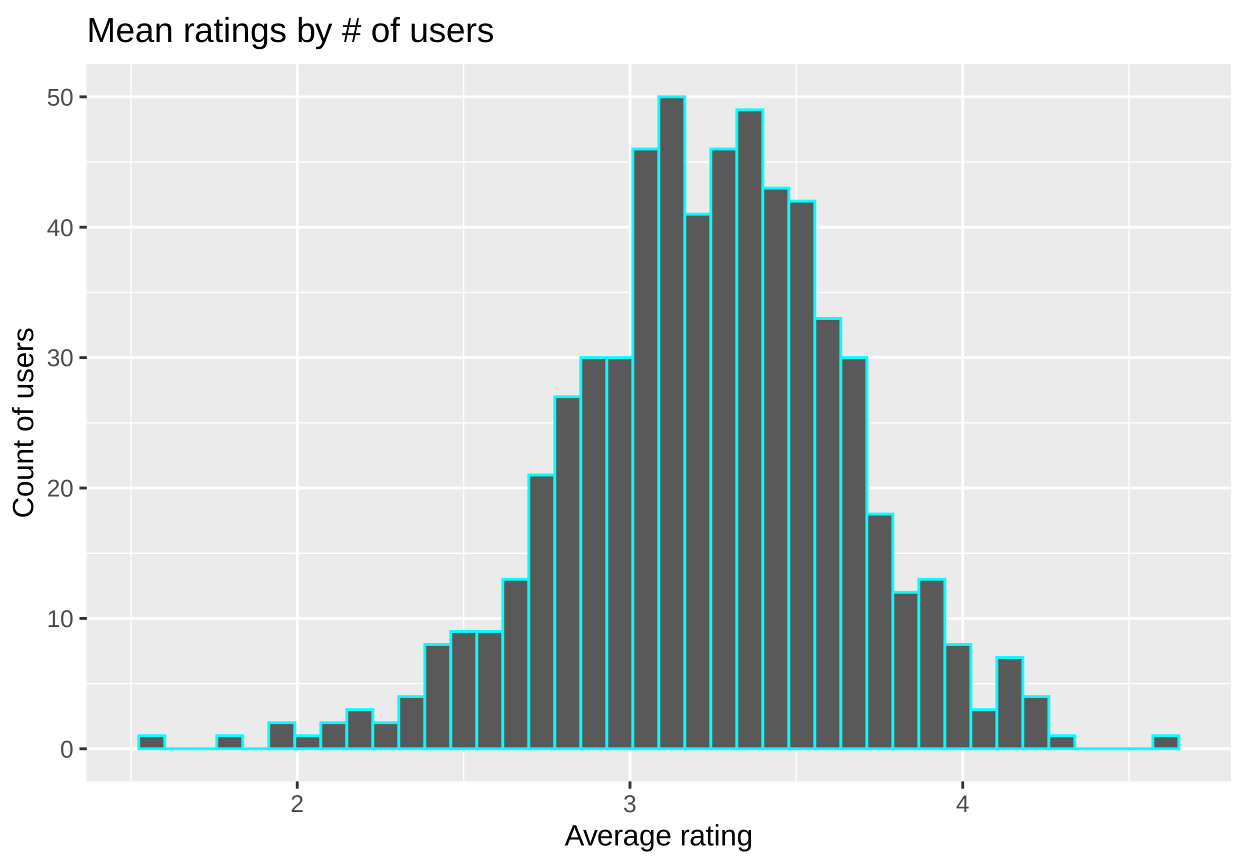
```r
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 50, color = "cyan") +
  ggtitle("Distribution of Users Who Rated Movies in Log Scale") +
  scale_x_log10(
    breaks = c(1, 10, 20, 30, 50, 100, 200, 500, 1000), # Logarithmic breakpoints
    labels = c(1, 10, 20, 30, 50, 100, 200, 500, 1000)  # Matching labels
  )
```



Histogram showing the distribution of average movie ratings

##given by users who have rated more than 1000 movies.

```
edx%>%group_by(userId)%>%
  filter(n()>1000)%>%
  summarize(mean_rating=mean(rating))%>%
  ggplot(aes(mean_rating))+
  geom_histogram(bins=40, color="cyan")+
  xlab("Average rating")+
  ylab("Count of users")+
  ggtitle("Mean ratings by # of users")+
  scale_x_continuous(
    breaks = c(0,1, 2, 3, 4, 5, 6), # Exact breakpoints
    labels = c(0,1, 2, 3, 4, 5, 6) # Matching labels
  )
```



## Predictive Models##

Here is equation that computes RMSE, which evaluates the models. Lower RMSE suggests better model.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

##Function to compute RMSE

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

1. Naive Model to estimate mean movie ratings

This is the base model. It simply estimates the average of overall ratings

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

So, here is the evarage of all ratings:

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

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

```r
# RMSE for the naive model
validation<-final_holdout_test
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse
```

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

```r
# create dataframe to store RMSE
rmse_results <- as.data.frame(tibble(Model ="Naive Average movie rating model",
                                     RMSE = naive_rmse))
rmse_results %>% knitr::kable()
```

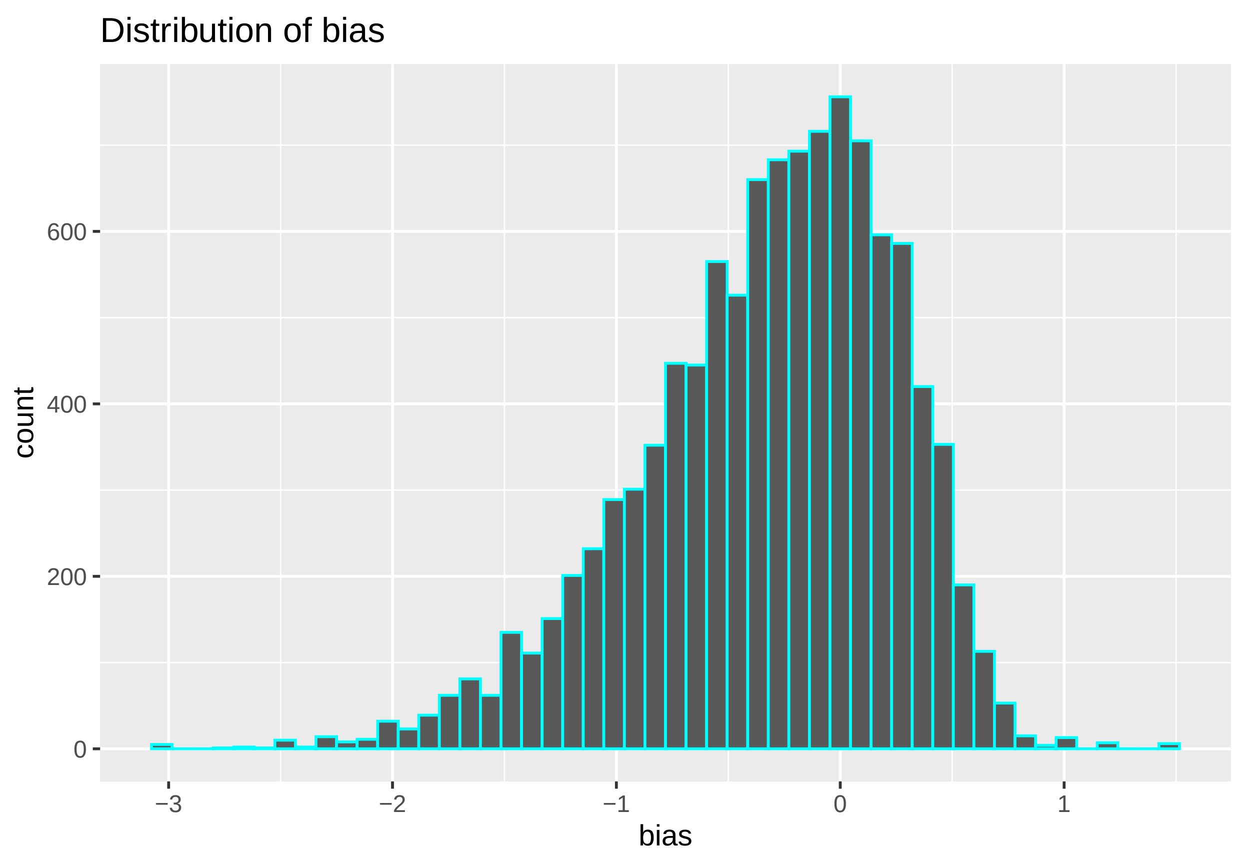
Model	RMSE
Naive Average movie rating model	1.061202

2. Movie effect model

As observed earlier, some movies received high ratings while others were rated much lower. To account for this variability, we compute a bias term, which is the difference between each movie's ratings and the overall mean, and then calculate the average of these bias values. We plotted the distribution of the bias and subsequently incorporated the movie effect (i.e., bias) into the model.

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

```r
# estimating bias for movie ratings
movie_avg <- edx %>%
  group_by(movieId) %>%
  summarize(bias = mean(rating - mu))

# examine distribution of bias
movie_avg %>%
  ggplot(aes(bias)) +
  geom_histogram(bins = 50, color = "cyan") +
  ggtitle("Distribution of bias")
```



```
# Compute the movie based predicted ratings on validation dataset
rmse_movie_model <- validation %>%
  left_join(movie_avg, by='movieId') %>%
  mutate(pred = mu + bias) %>%
  pull(pred) #Extracts the pred column as a vector.

rmse_movieBased<- RMSE(validation$rating, rmse_movie_model)
rmse_results <- rmse_results %>% add_row(Model="Movie-Based Model",
                                         RMSE=rmse_movieBased)
rmse_results
```

```
##                              Model      RMSE
## 1 Naive Average movie rating model 1.0612018
## 2                Movie-Based Model 0.9439087
```

Adding the movie effect improved the RMSE compared the Naive Model.

3.Movie+User effect model

We wanted to see if the model could be improved further by incorporating user effect as some users rated more movies than others.

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

```
user_avg <- edx %>%
  left_join(movie_avg, by='movieId') %>%
  group_by(userId) %>%
  summarize(bias_user = mean(rating - mu - bias))

# Compute the predicted ratings on validation dataset (movie+user)
rmse_movie_user_model <- validation %>%
  left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  mutate(pred = mu+ bias + bias_user) %>%
  pull(pred)

rmse_movie_userbased <- RMSE(validation$rating, rmse_movie_user_model)
rmse_results <- rmse_results %>% add_row(Model="Movie+User Based Model",
                                         RMSE=rmse_movie_userbased)
rmse_results
```

```
##                               Model      RMSE
## 1 Naive Average movie rating model 1.0612018
## 2                Movie-Based Model 0.9439087
## 3           Movie+User Based Model 0.8653488
```

Adding user effect improved the RMSE again. We now want to see if incoporating differnt genere into to the mdoel further improve RMSE.

4.Movie+user+Genre based model

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

```
genre_pop <- edx %>%
  left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  group_by(genres) %>%
  summarize(bias_user_genre = mean(rating - mu - bias - bias_user))

# Compute the predicted ratings on validation dataset

rmse_movie_user_genre_model <- validation %>%
  left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  left_join(genre_pop, by='genres') %>%
  mutate(pred = mu + bias + bias_user + bias_user_genre) %>%
  pull(pred)

rmse_movie_user_genre <- RMSE(validation$rating, rmse_movie_user_genre_model)
rmse_results <- rmse_results %>% add_row(Model="Movie+User+Genre Based Model",
                                         RMSE=rmse_movie_user_genre)
rmse_results
```

```
##                               Model      RMSE
## 1 Naive Average movie rating model 1.0612018
## 2                Movie-Based Model 0.9439087
## 3           Movie+User Based Model 0.8653488
## 4     Movie+User+Genre Based Model 0.8649469
```

Adding genre reduced RMSE slightly, not a significant improvement.

##Regularization models###
So far, we have been working with simpler models. However, the data shows significant variability in ratings. For instance, many movies were rated by only one user, while some users rated numerous movies. This variability introduces uncertainty into the data, which can make predictions less reliable.

To address this uncertainty and improve the robustness of our predictions, we will use regularization. Regularization is a modeling technique that enhances a model's generalizability and prevents overfitting by penalizing large coefficients. This penalty discourages the model from relying too heavily on any single feature, leading to more balanced and reliable predictions.

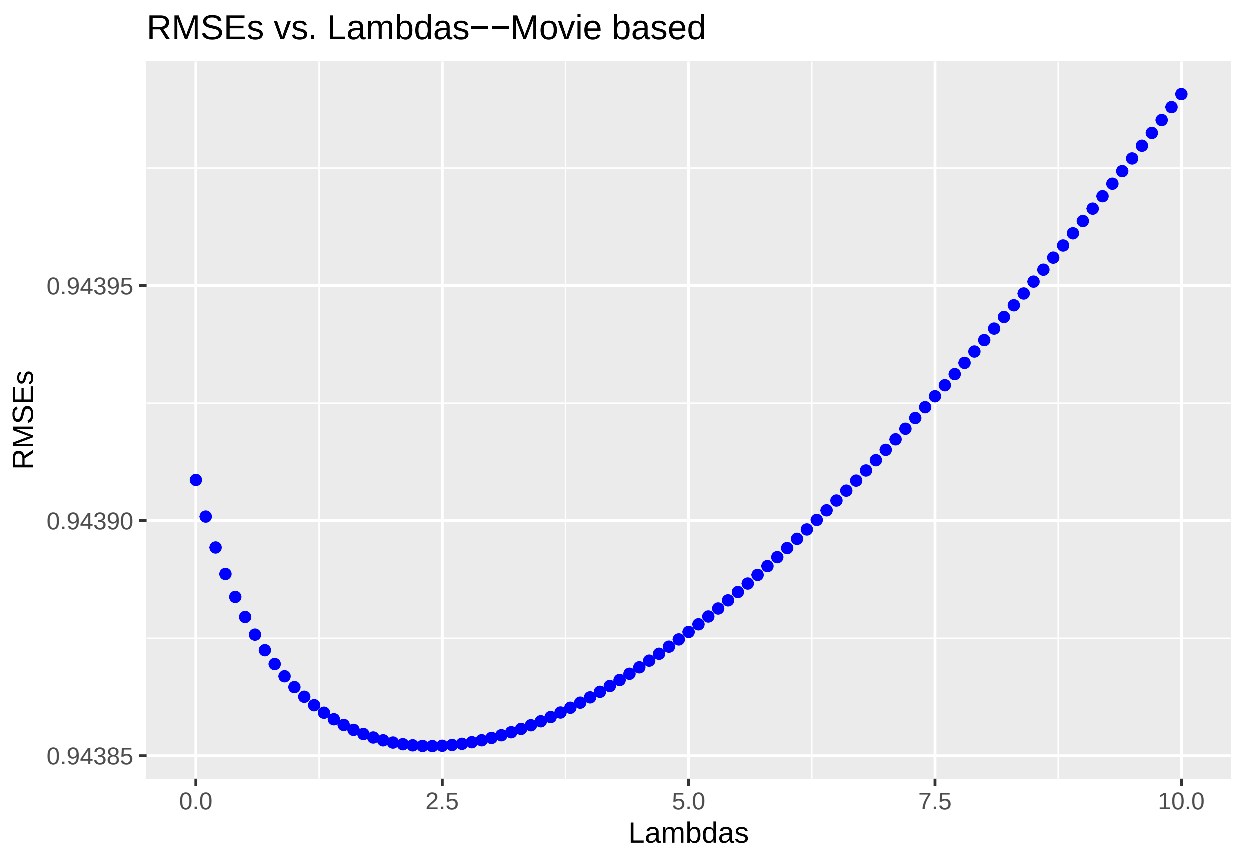
Next, we will apply regularization techniques to all the previously developed models and evaluate their performance using the RMSE metric.

5.Regularization model–movie based

```r
lambdas <- seq(0, 10, 0.1)
# Compute the predicted ratings on validation dataset using different values of lambda
rmses <- sapply(lambdas, function(lambda) {

  # Calculate the average by movieId
    b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + lambda))

  #predicted ratings on validation data set
    predicted_ratings <- validation %>%
    left_join(b_i, by='movieId') %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)

  # Predict RMSE on validation set
    return(RMSE(validation$rating, predicted_ratings))
})

# lambda value that minimizes the RMSE
min_lambda <- lambdas[which.min(rmses)]
# Predict the RMSE on the validation set
rmse_regularized_movie <- min(rmses)
# Add the results to the results dataset
rmse_results <- rmse_results %>% add_row(Model="Regularization Movie-Based Model", RMSE=rmse_regularize
rmse_results
```

```
##                               Model      RMSE
## 1 Naive Average movie rating model 1.0612018
## 2                 Movie-Based Model 0.9439087
## 3           Movie+User Based Model 0.8653488
## 4     Movie+User+Genre Based Model 0.8649469
## 5 Regularization Movie-Based Model 0.9438521
```

```r
ggplot(data = data.frame(lambdas, rmses), aes(x = lambdas, y = rmses)) +
  geom_point(color = "blue") +
  ggtitle("RMSEs vs. Lambdas--Movie based") +
  xlab("Lambdas") +
  ylab("RMSEs")
```



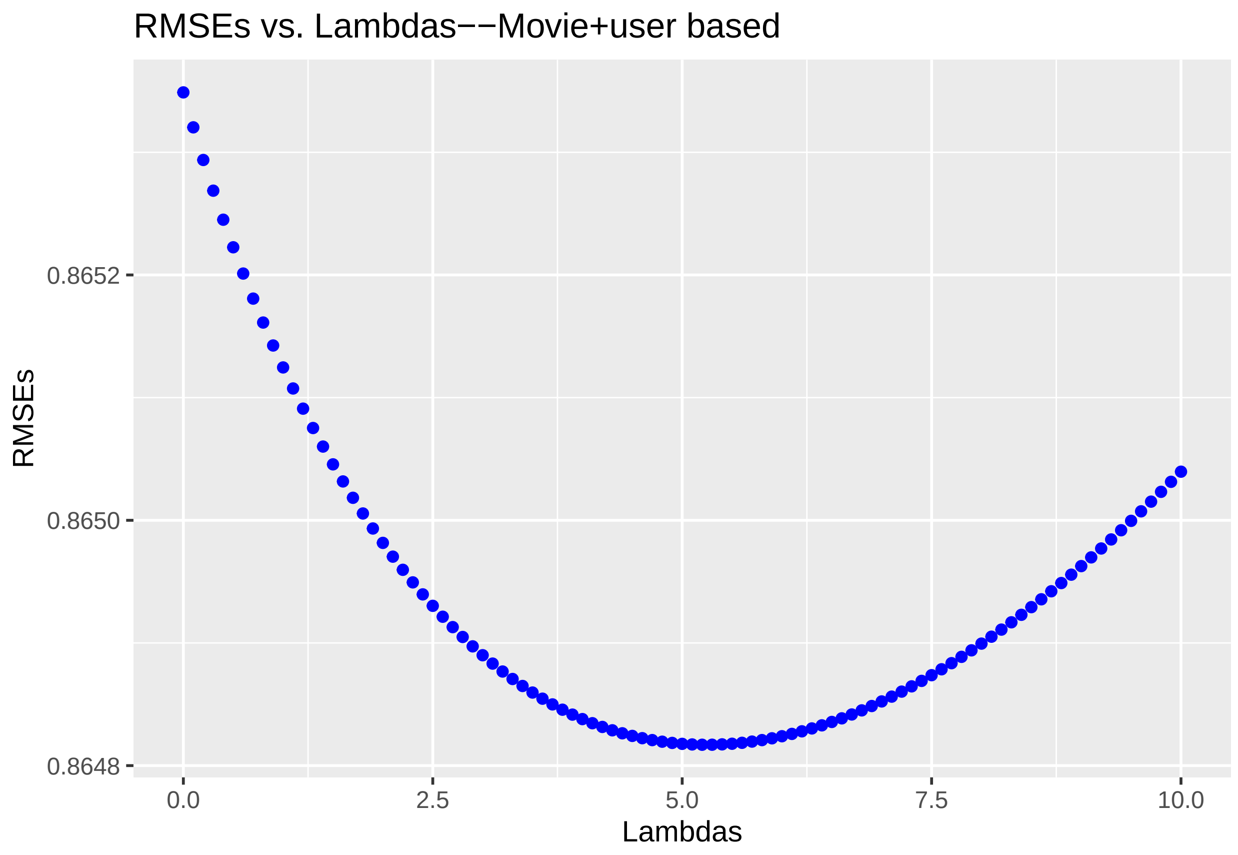
The regularization model considering only the movie effect did not lead to an improvement in RMSE. Let's re-run the model by including the user effect to assess its impact.

###Regularization Movie+user based model###

```r
rmses <- sapply(lambdas, function(lambda) {
    # Calculate the average by user
    b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + lambda))

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

  # predicted ratings on validation dataset
    predicted_ratings <- validation %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  # Predict the RMSE on the validation set
    return(RMSE(validation$rating, predicted_ratings))
})

# lambda value that minimizes the RMSE
min_lambda <- lambdas[which.min(rmses)]

# Predict RMSE on validation set
rmse_regularized_movie_user <- min(rmses)
rmse_results <- rmse_results %>% add_row(Model="Regularization Movie+User Based Model", RMSE=rmse_regula
rmse_results
```

```
##                                    Model      RMSE
## 1      Naive Average movie rating model 1.0612018
## 2                      Movie-Based Model 0.9439087
## 3                 Movie+User Based Model 0.8653488
## 4           Movie+User+Genre Based Model 0.8649469
## 5      Regularization Movie-Based Model 0.9438521
## 6 Regularization Movie+User Based Model 0.8648170
```

```
ggplot(data = data.frame(lambdas, rmses), aes(x = lambdas, y = rmses)) +
  geom_point(color = "blue") +
  ggtitle("RMSEs vs. Lambdas--Movie+user based") +
  xlab("Lambdas") +
  ylab("RMSEs")
```



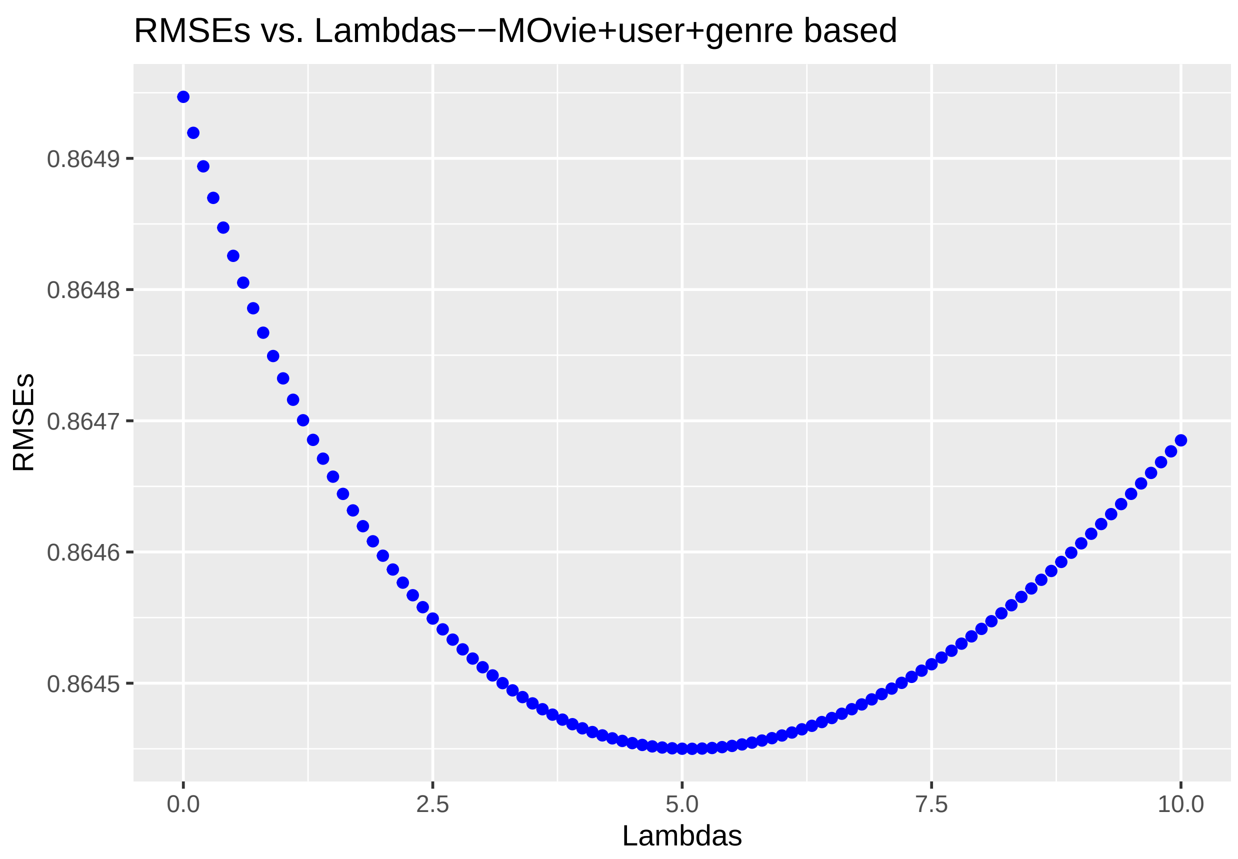
By incorporating the user effect into the movie-only regularization model, we achieved the lowest RMSE to date. Next, let's explore the impact of adding genre as an additional factor to the model.

###MOvie+user+genre based model###

```r
# Compute the predicted ratings on validation dataset using different values of lambda
rmses <- sapply(lambdas, function(lambda) {
  # Calculate the average by user
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + lambda))

  # Calculate the average by user
  b_u <- edx %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + lambda))

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

  #predicted ratings on validation dataset
  predicted_ratings <- validation %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_u_g, by='genres') %>%
    mutate(pred = mu + b_i + b_u + b_u_g) %>%
    pull(pred)
    # Predict RMSE on validation set
    return(RMSE(validation$rating, predicted_ratings))
})

# lambda value that minimizes the RMSE
min_lambda <- lambdas[which.min(rmses)]
# Predict RMSE on validation set
rmse_regularized_movie_user_genre <- min(rmses)

rmse_results <- rmse_results %>% add_row(Model="Regularization Movie+User+Genre Based Model", RMSE=rmse_
rmse_results
```

```
##                                          Model      RMSE
## 1            Naive Average movie rating model 1.0612018
## 2                            Movie-Based Model 0.9439087
## 3                       Movie+User Based Model 0.8653488
## 4                 Movie+User+Genre Based Model 0.8649469
## 5            Regularization Movie-Based Model 0.9438521
## 6       Regularization Movie+User Based Model 0.8648170
## 7 Regularization Movie+User+Genre Based Model 0.8644500
```

```r
ggplot(data = data.frame(lambdas, rmses), aes(x = lambdas, y = rmses)) +
  geom_point(color = "blue") +
  ggtitle("RMSEs vs. Lambdas--MOvie+user+genre based") +
  xlab("Lambdas") +
  ylab("RMSEs")
```



This model produced the lowest RMSE, 0.86445.

Conclusion

The final model:

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

With the following regularization techniques:

Bias Calculation:

$$b_i = \frac{\sum_u (r_{u,i} - \mu)}{n_i + \lambda}$$

$$b_u = \frac{\sum_i (r_{u,i} - b_i - \mu)}{n_u + \lambda}$$

$$b_g = \frac{\sum_u (r_{u,i} - b_i - \mu - b_u)}{n_g + \lambda}$$

Predicted Ratings:

$$\hat{r}_{u,i} = \mu + b_i + b_u + b_g$$

RMSE calculation:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{u,i} (r_{u,i} - \hat{r}_{u,i})^2}$$

We developed seven models, ranging from simple to more complex approaches incorporating regularization techniques. The best-performing model was the regularized version that accounted for movie, user, and genre effects, yielding the lowest RMSE of 0.86445. However, additional variables, such as user demographic characteristics, may be missing and could further enhance the model's performance by potentially reducing the RMSE even further.