MovieLens 10M Recommendation System

Simon Lee

9/20/2025

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.2
## v ggplot2   3.5.2     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.1.0
## -- 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 errors
```

```
library(caret)
```

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

```
library(lubridate)
```

1. Introduction

The purpose of this report is to build and evaluate a movie recommendation system using the **MovieLens 10M dataset**.

---

2. Loading and Preparing the Data

```
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")
```

---

3. Split Into Training and Testing Set

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

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

```
test_index<-createDataPartition(movielens$rating, p=0.1, list=FALSE)
edx<-movielens[-test_index, ]
temp<-movielens[test_index, ]

final_holdout_test<-temp %>%
```

```
  semi_join(edx, by="movieId") %>%
  semi_join(edx, by="userId")

removed<-anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```
edx<-rbind(edx, removed)
```

---

4. Exploratory Data

4.1: Dataset Overview

```
cat("Number of ratings:", format(nrow(edx), big.mark=","), "\n")
```

```
## Number of ratings: 9,000,055
```

```
cat("Number of users:", length(unique(edx$userId)), "\n")
```

```
## Number of users: 69878
```

```
cat("Number of movies:", length(unique(edx$movieId)), "\n")
```

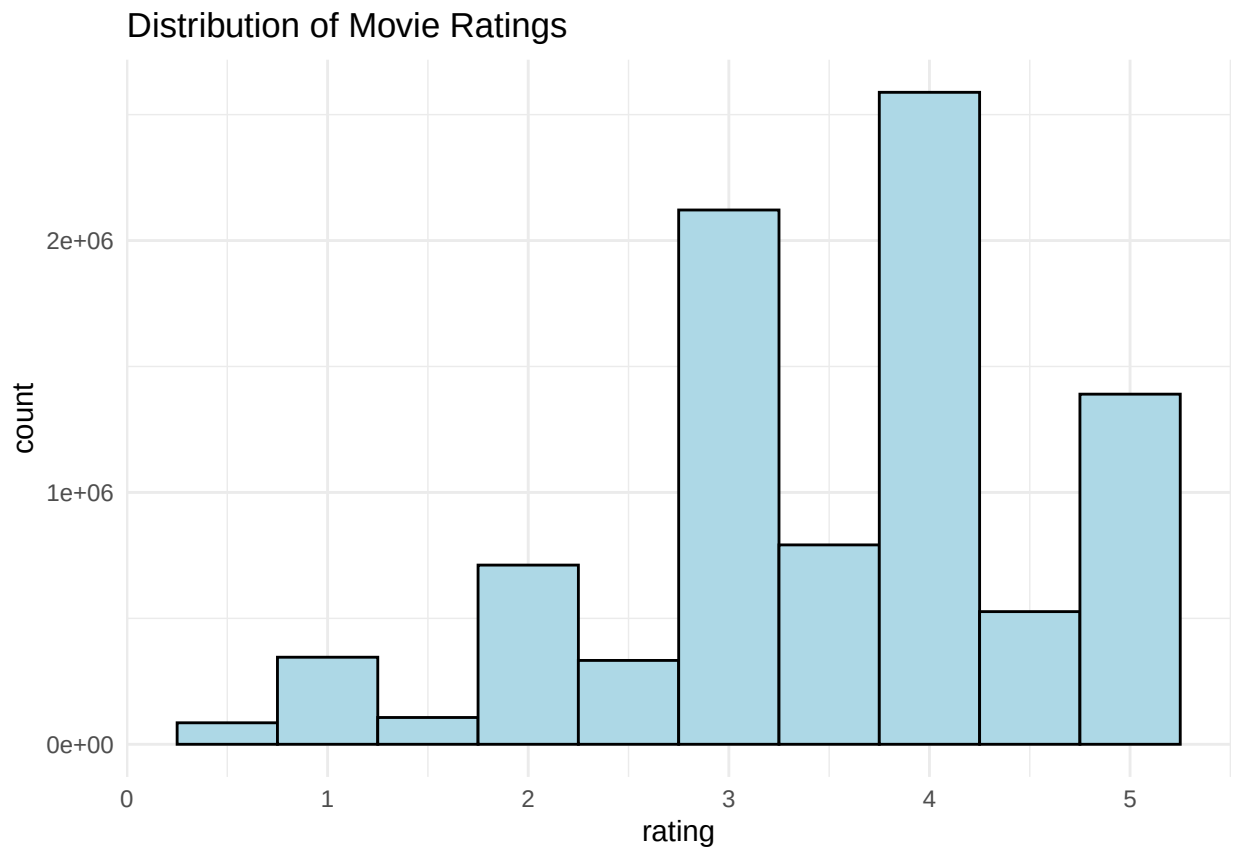
```
## Number of movies: 10677
```

```
summary(edx$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

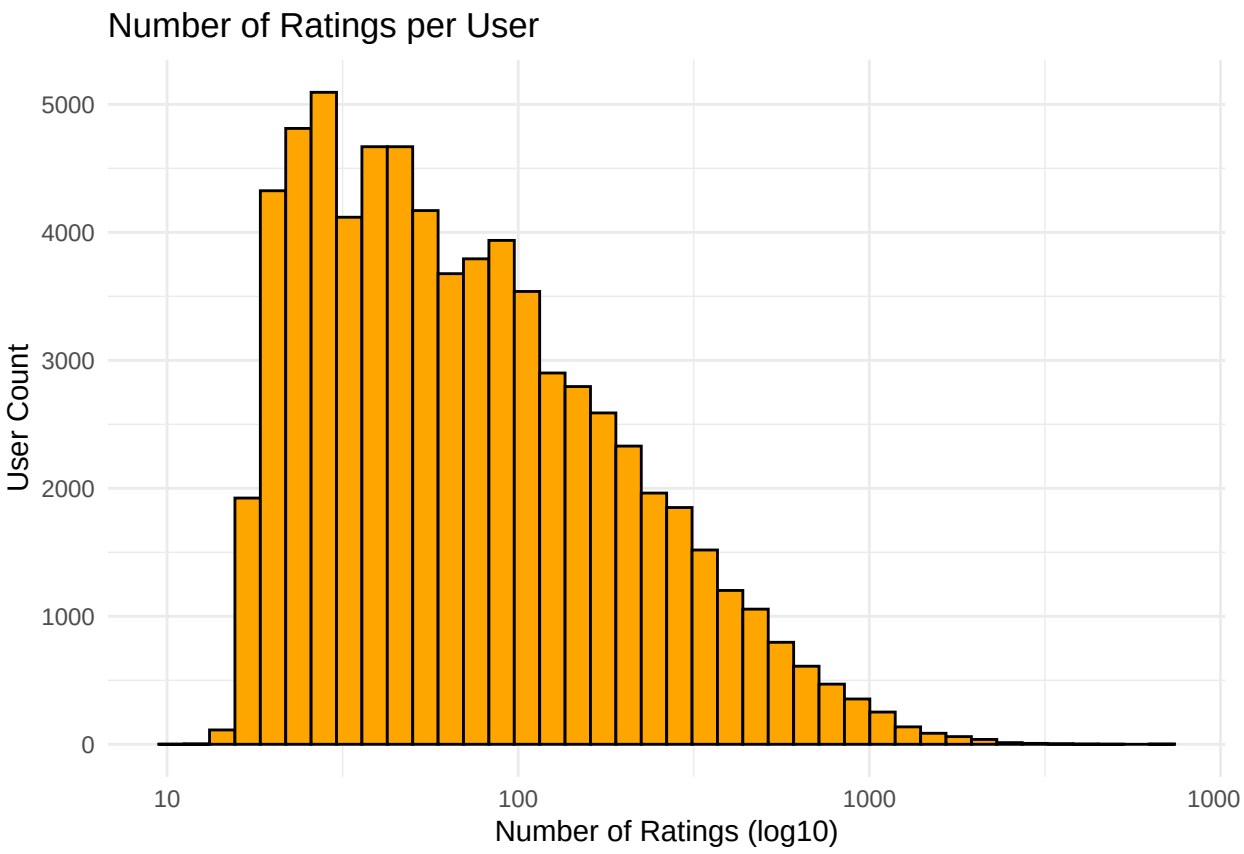
4.2 Rating Distribution

```
ggplot(edx, aes(rating))+
  geom_histogram(binwidth=0.5, fill="lightblue", color="black")+
  theme_minimal()+
  labs(title="Distribution of Movie Ratings")
```



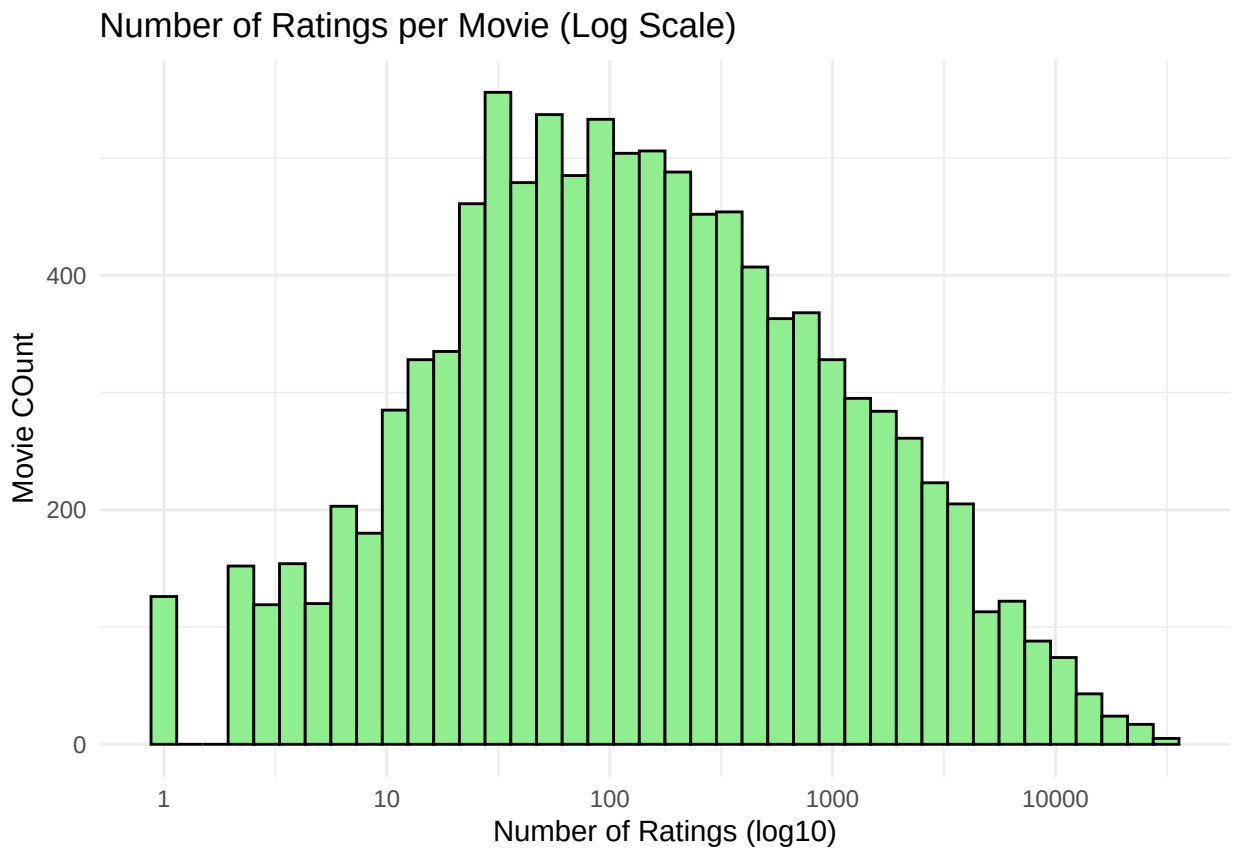
4.3: Number of Ratings Per User

```
user_count<-edx%>%
  group_by(userId)%>%
  summarize(n=n())

ggplot(user_count, aes(n))+
  geom_histogram(bins = 40, color="black", fill="orange")+
  scale_x_log10()+
  ggtitle("Number of Ratings per User")+
  xlab("Number of Ratings (log10)")+
  ylab("User Count")+
  theme_minimal()
```



4.4: Number of Ratings Per Movie

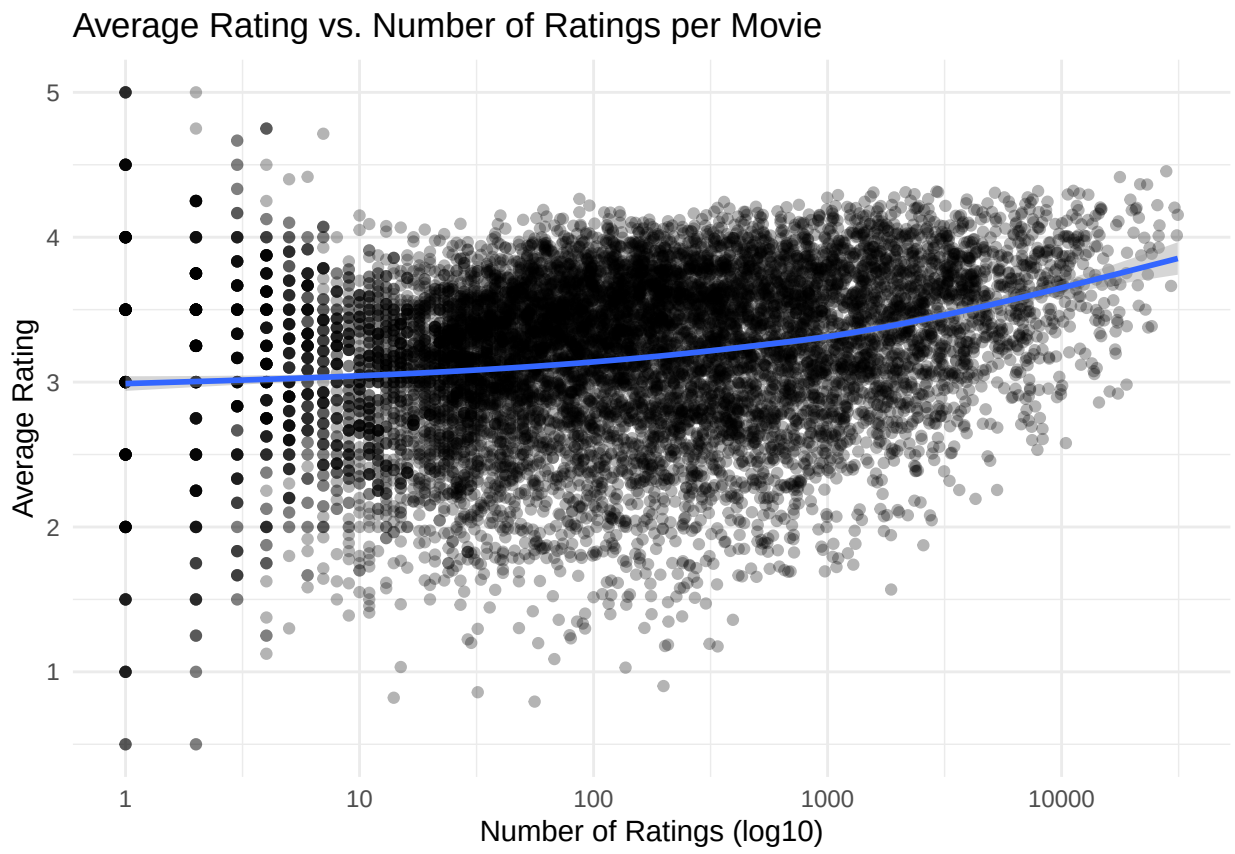
```
movie_count<-edx%>%
  group_by(movieId)%>%
  summarize(n = n())

ggplot(movie_count, aes(n))+
  geom_histogram(bins = 40, color="black", fill = "lightgreen")+
  scale_x_log10()+
  ggtitle("Number of Ratings per Movie (Log Scale)")+
  xlab("Number of Ratings (log10)")+
  ylab("Movie COunt")+
  theme_minimal()
```



4.5: Average Rating vs. Number of Ratings per Movie

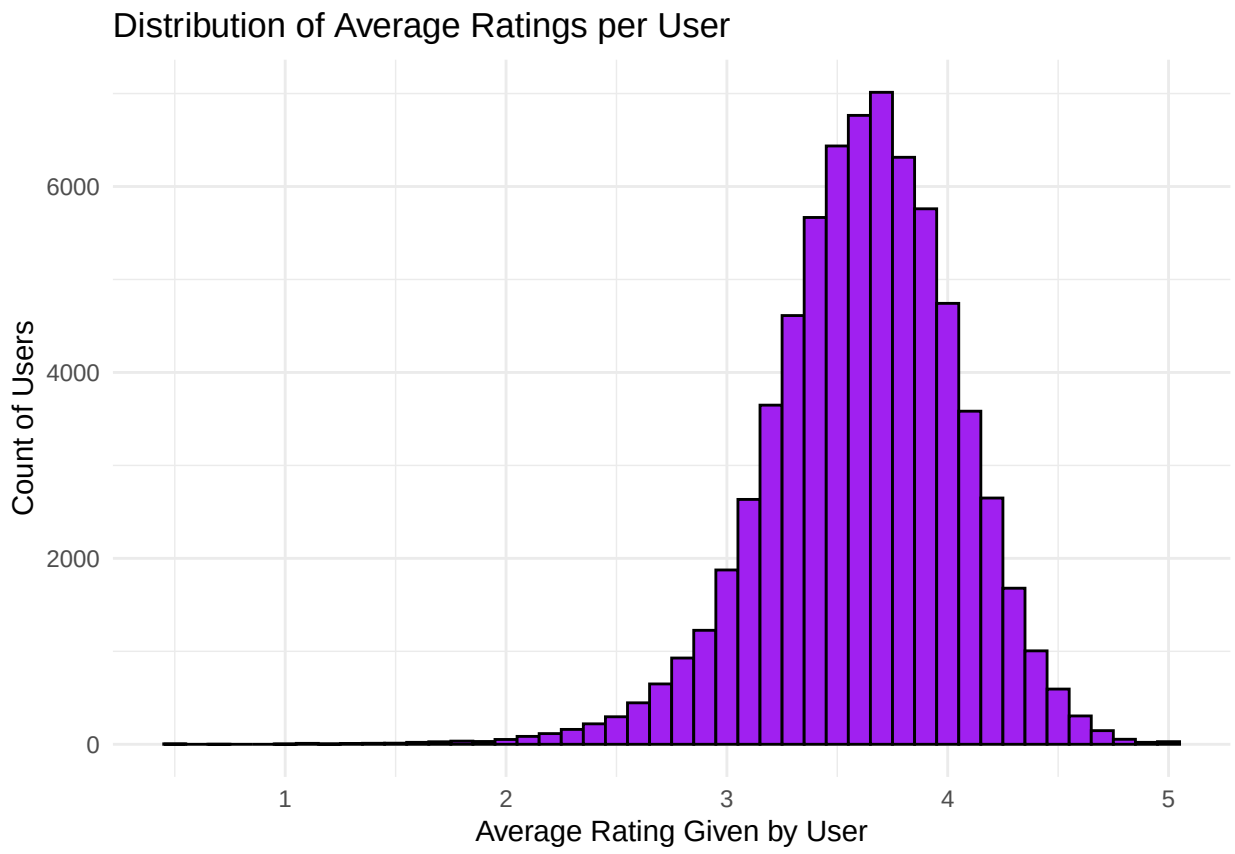
```
movie_avgs<-edx%>%
  group_by(movieId)%>%
  summarize(avg_rating=mean(rating), n=n())

ggplot(movie_avgs, aes(n, avg_rating))+
  geom_point(alpha=0.3)+
  scale_x_log10()+
  geom_smooth()+
  ggtitle("Average Rating vs. Number of Ratings per Movie")+
  xlab("Number of Ratings (log10)")+
  ylab("Average Rating")+
  theme_minimal()
```

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



4.6: Average Rating Per User

```
user_avgs<-edx%>%
  group_by(userId)%>%
  summarize(avg_rating=mean(rating), n=n())

ggplot(user_avgs, aes(avg_rating))+
  geom_histogram(binwidth = 0.1, color="black", fill="purple")+
  ggtitle("Distribution of Average Ratings per User")+
  xlab("Average Rating Given by User")+
  ylab("Count of Users")+
  theme_minimal()
```



---

5. Modeling: Regularized Movie + User Effects

Connecting the Graphs to the Model

Rating Distribution

The histogram shows a slight left skew with median somewhere between a rating of 3 and 4.
Insight: Ratings are not uniform, some confounding variable might be at play.
Model relevance: The global mean alone cannot accurately predict a rating for a movie. Adding **movie bias (b_i)** variable allows the model to adjust predictions for movies that are generally liked or disliked.

Number of Ratings per User

The histogram shows a huge right skew with majority of users rating below 100 movies.
Insight: Users vary in rating activity.
Model relevance: The users who tend to rate more movies usually provide a more consistent pattern for th emodel to predict. Including a **user bias (b_u)** variable accounts for that factor.

Number of Ratings per Movie

The histogram shows that a most movies receive very few reviews.
Insight: Movies with few ratings tend to be noisy and might produce extreme averages.
Model relevance: Regularization ( ) in the calculation of b_i attempts to shrink these noisy predictions towards 0, limiting the effect of noisy data and extreme predictions.

Average Rating vs Number of Ratings per Movie

The scatter plot shows that movies with more ratings tend to cluster around their mean rating while movies with less rating do the opposite.
Insight: Movies with fewer ratings are unreliable.
Model relevance: Further enforces the need for regularization.

Average Ratings per User

The histogram shows the existence of users who might tend to rate movies harshly or generously.
Insight: Users habits can influence rating.
Model relevance: Further enforces the need for a user-bias variable.

From the graphs, the recommender model will follow the formula:

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

Movie Bias

The movie-bias variable (b_i) is defined as:

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

Where:

- b_i = movie bias for movie i
- $r_{u,i}$ = rating given by user u to movie i
- μ = overall mean rating
- n_i = number of ratings for movie i
- λ = regularization parameter

User Bias

The user effect (bias) is defined as:

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

Where:

- b_u = user bias for user u
- $r_{u,i}$ = rating given by user u to movie i
- b_i = movie bias for movie i
- μ = overall mean rating
- n_u = number of ratings made by user u
- λ = regularization parameter

Setting lambda (λ)

We will set λ as 5.

```
lambda<-5
mu<-mean(edx$rating)

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-mu-b_i)/(n()+lambda))

RMSE<-function(true, pred){
  sqrt(mean((true-pred)^2))
}
```

---

6. Model Evaluation

6.1 Training RMSE

```r
pred_train<-edx%>%
  left_join(b_i, by="movieId")%>%
  left_join(b_u, by="userId")%>%
  mutate(pred=mu+b_i+b_u)

rmse_train<-RMSE(pred_train$rating, pred_train$pred)
rmse_train
```

```
## [1] 0.8570452
```

6.2 Final Holdout RMSE

```r
pred_holdout<-final_holdout_test%>%
  left_join(b_i, by="movieId")%>%
  left_join(b_u, by="userId")%>%
  mutate(pred=mu+b_i+b_u)%>%
  mutate(pred=pmin(pmax(pred, 0.5), 5))

rmse_holdout<-RMSE(pred_holdout$rating, pred_holdout$pred)
rmse_holdout
```

```
## [1] 0.8647091
```

---

7. Top 5 Recommendations for a New User

A new user has no rating history → recommend the *highest-scoring movies overall*:

```r
best_movies<-b_i%>%
  left_join(movies, by="movieId")%>%
  mutate(pred=mu+b_i)%>%
  arrange(desc(pred))%>%
  slice_head(n=5)

best_movies
```

```
## # A tibble: 5 x 5
##   movieId   b_i title                              genres                  pred
##     <int> <dbl> <chr>                              <chr>                  <dbl>
## 1     318 0.942 Shawshank Redemption, The (1994) Drama                    4.45
## 2     858 0.903 Godfather, The (1972)            Crime|Drama             4.42
## 3      50 0.853 Usual Suspects, The (1995)       Crime|Mystery|Thriller  4.37
## 4     527 0.851 Schindler's List (1993)          Drama|War               4.36
## 5     912 0.808 Casablanca (1942)                Drama|Romance           4.32
```

---

8. Conclusion

Through exploratory analysis of the data, we show how we can create a recommender model by simply factoring in user and movie bias to predict a user's rating for a certain movie and recommend them the top 5 highest, predicted rating movies. The model is fast and simple. However, most real-world recommendation systems incorporate implicit feedback and data from cookies when making recommendations, which are not considered in this simple model. Future work can extend to incorporating individual user input to tailor recommendations for each individual user in the context of their own data. As such, we could not incorporate the user bias in the final recommendation because it is for a new user in which we have no data on, i.e. they have not rated any movies. Asking a user to rate multiple movies would be tedious for the user as well, hence why we need additional data other than the movielens dataset.
