Capstone Project: MovieLens

Isaiah Lemons

12/28/2019

Introduction

This project is part of the HarvardX:PH125.9x Data Science Capstone project. It is based on a challenge that Netflix offered to the data science community in 2006, to improve Netflix's movie recomendation system by 10% and win a $1,000,000 prize. The goal of this project was to design multiple machine learning algorithms using inputs from a subset of data to predict the ratings in the validation set. The algorithim with the highest accuracy can then be used to more accurately recommend movies to specifc users based on their previous moving ratings.

The dataset used can be found at the link below. (http://files.grouplens.org/datasets/movielens/ml-10m.zip).

This report contains the entire process from data wrangling, exploratory analysis, modeling, results, and conclusion.

##Data Wrangling The following code was used to pull the data and split it into training and validation sets.

```
# 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)
# if using R 3.5 or later, use `set.seed(1, sample.kind="Rounding")` instead
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)
```

##Data Exploration Once data is available we can begin to research the dataset. We can see using the code below this dataset in in tidy format, edx(training set) contains 9000055 rows and 6 columns.

```
# edx data is in tidy format
edx %>% as_tibble()
```

```
## # A tibble: 9,000,061 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|Thriller
##  3      1     231      5 838983392 Dumb & Dumber (199~ Comedy
##  4      1     292      5 838983421 Outbreak (1995)     Action|Drama|Sci-Fi|Thri~
##  5      1     316      5 838983392 Stargate (1994)     Action|Adventure|Sci-Fi
##  6      1     329      5 838983392 Star Trek: Generat~ Action|Adventure|Drama|S~
##  7      1     355      5 838984474 Flintstones, The (~ Children|Comedy|Fantasy
##  8      1     356      5 838983653 Forrest Gump (1994) Comedy|Drama|Romance|War
##  9      1     362      5 838984885 Jungle Book, The (~ Adventure|Children|Roman~
## 10      1     364      5 838983707 Lion King, The (19~ Adventure|Animation|Chil~
## # ... with 9,000,051 more rows
```

```
nrow(edx)
```

```
## [1] 9000061
```

```
ncol(edx)
```

```
## [1] 6
```

```
# Check for missing values
any(is.na(edx))
```

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

```
# Number of unique movies and users
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

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

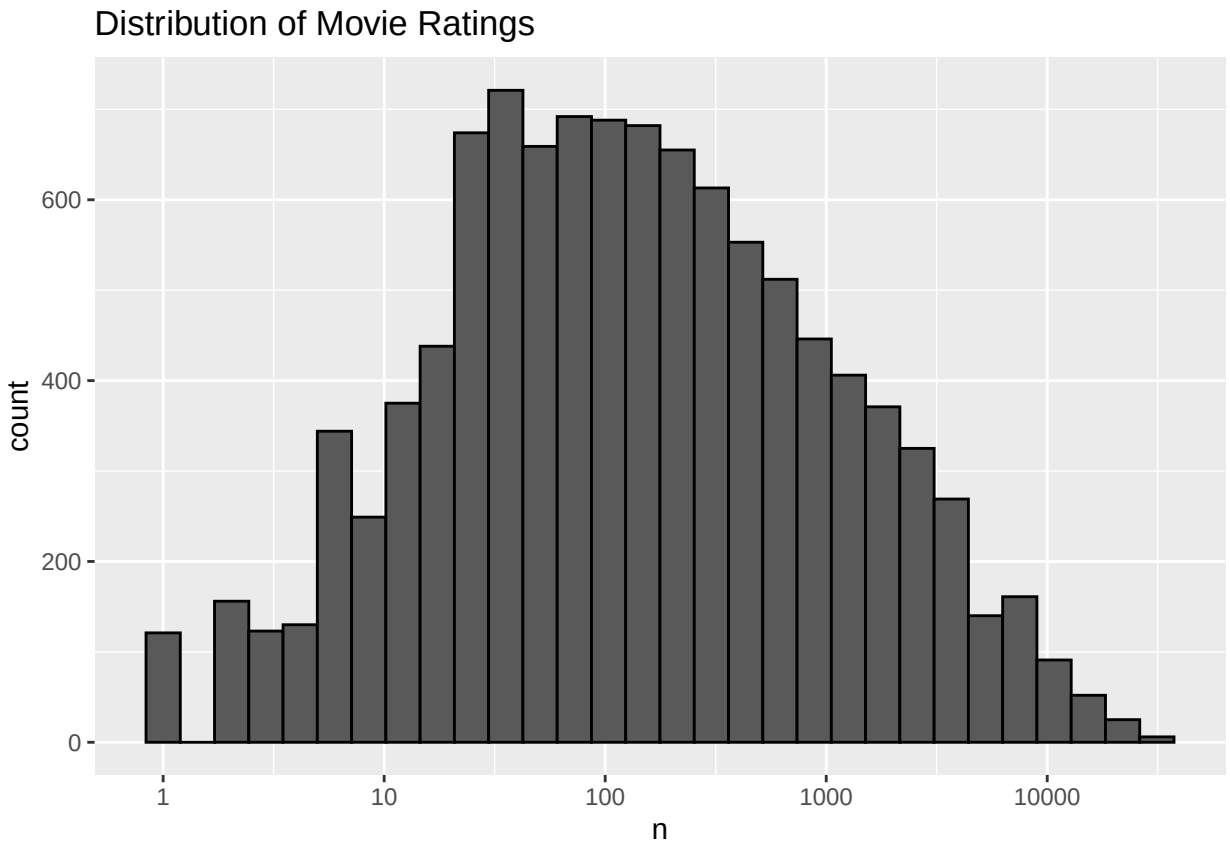
After intial exploration you can see there is not a designated column for year, it has just been apended to the end of the movie titles. The code below modifies the dataset by adding an additional column for year, which will be used for further analysis.

```
# Modify edx by adding a column for year
edx_mod <- edx %>% mutate(year = as.numeric(str_sub(title, -5, -2)))
edx_mod %>% as_tibble()
```

```
## # A tibble: 9,000,061 x 7
##    userId movieId rating timestamp title            genres                  year
##     <int>   <dbl>  <dbl>     <int> <chr>            <chr>                  <dbl>
##  1      1     122      5 838985046 Boomerang (1992) Comedy|Romance          1992
##  2      1     185      5 838983525 Net, The (1995)  Action|Crime|Thriller   1995
##  3      1     231      5 838983392 Dumb & Dumber (~ Comedy                   1994
##  4      1     292      5 838983421 Outbreak (1995)  Action|Drama|Sci-Fi|T~  1995
##  5      1     316      5 838983392 Stargate (1994)  Action|Adventure|Sci-~  1994
##  6      1     329      5 838983392 Star Trek: Gene~ Action|Adventure|Dram~  1994
##  7      1     355      5 838984474 Flintstones, Th~ Children|Comedy|Fanta~  1994
##  8      1     356      5 838983653 Forrest Gump (1~ Comedy|Drama|Romance|~  1994
##  9      1     362      5 838984885 Jungle Book, Th~ Adventure|Children|Ro~  1994
## 10      1     364      5 838983707 Lion King, The ~ Adventure|Animation|C~  1994
## # ... with 9,000,051 more rows
```

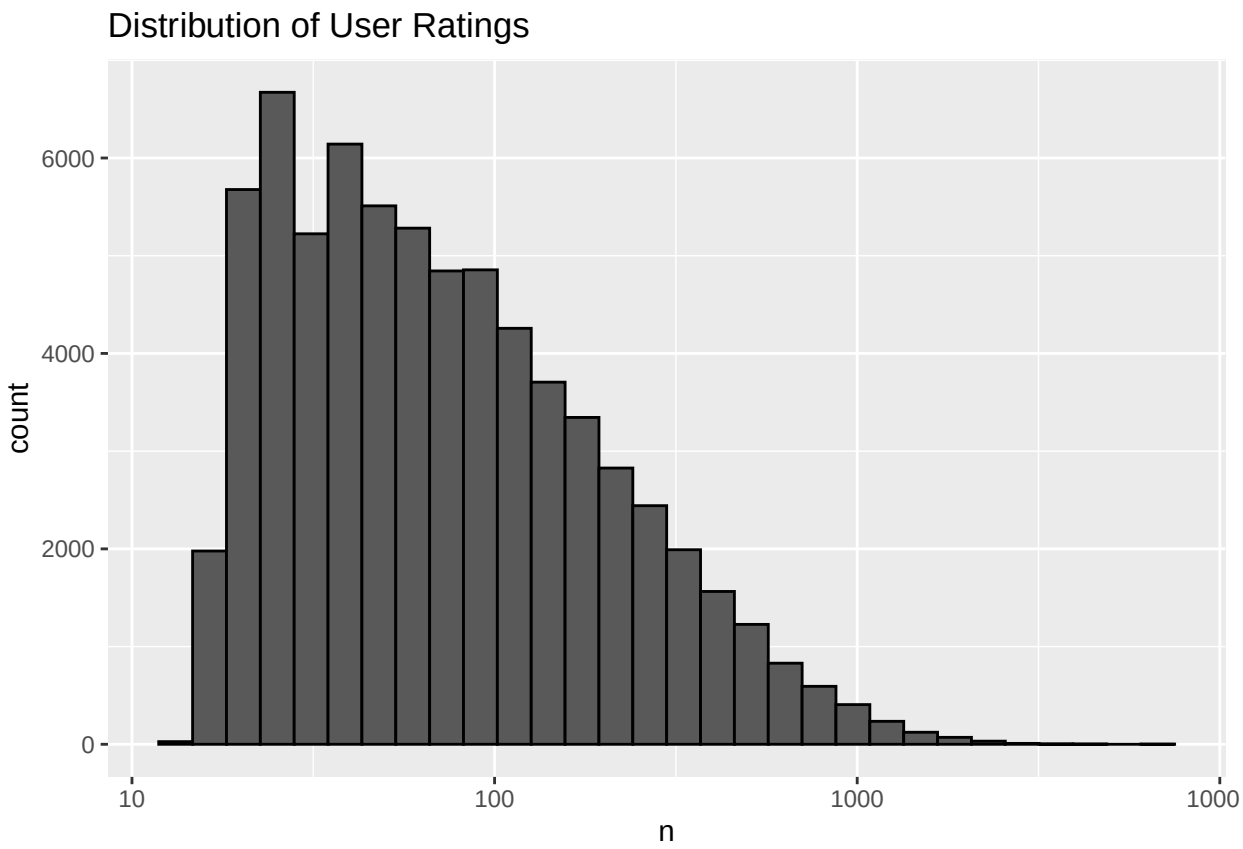
#Distributions Our first observation is probably the easiest thing to notice where some movies get rated much more then others. This can be explained because some are blockbuster movies that get watched and reviewed by millions of people, while independent movies may have only a few views in comparison.

```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Distribution of Movie Ratings")
```



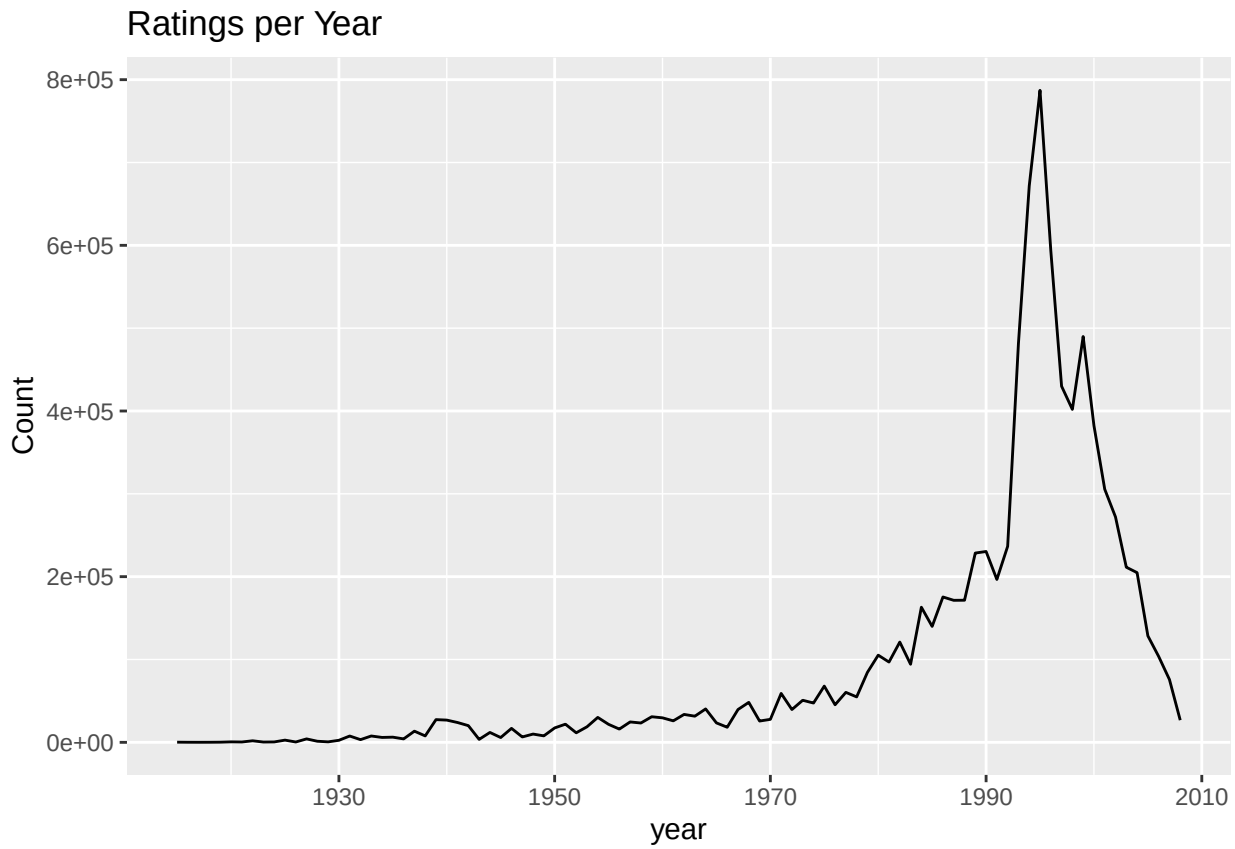
The second observation is that some users rate more movies than others.

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



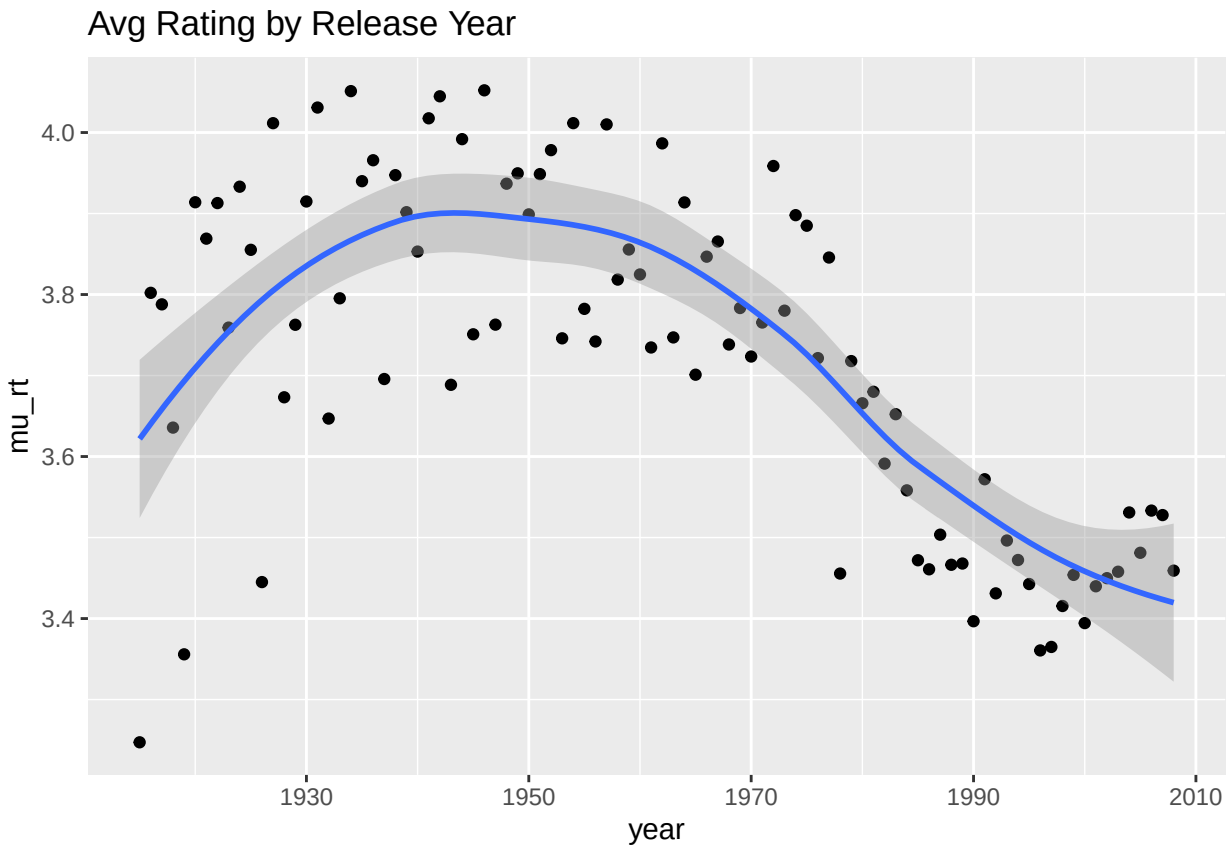
The next distribution is for the total number of ratings per year. As you can see movies that were released around 1995 have higher amount of ratings than more recent years. This can be attributed to those movies being out for a longer time period which has allowed them to have several more views and ratings.

```
rating_years <- edx_mod %>%
  group_by(year) %>%
  summarize(Count = n()) %>%
  arrange(desc(Count))

rating_years %>%
  ggplot(aes(year, Count)) +
  geom_line() +
  ggtitle("Ratings per Year")
```



The code below shows the average rating by release year. This shows an interesting change in the average rating, where there was a sharp decrease in the averages from 1980 forward.

```
edx_mod %>% group_by(year) %>%
  summarize(mu_rt = mean(rating)) %>%
  ggplot(aes(year, mu_rt)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Avg Rating by Release Year")
```



##Modeling Since the RMSE (root mean square errors) will be used multiple times while training models the best approach is to create a function that will simplify our code later. The RMSE calculation is defined as shown in the code below.

```
# Create a funtion for RMSE (root mean squre errors)
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

#Model 1: Naive Approach The simplest recommendation system would be to predict the same rating for all movies regardless of any other factors. This model can be built simply by taking the average rating for all movies.It will act as a baseline which will be improved on by other models.

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

```
## [1] 3.512464
```

We can then use mu to predict the unknown ratings using the RMSE function shown below.

```
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse
```

```
## [1] 1.060651
```

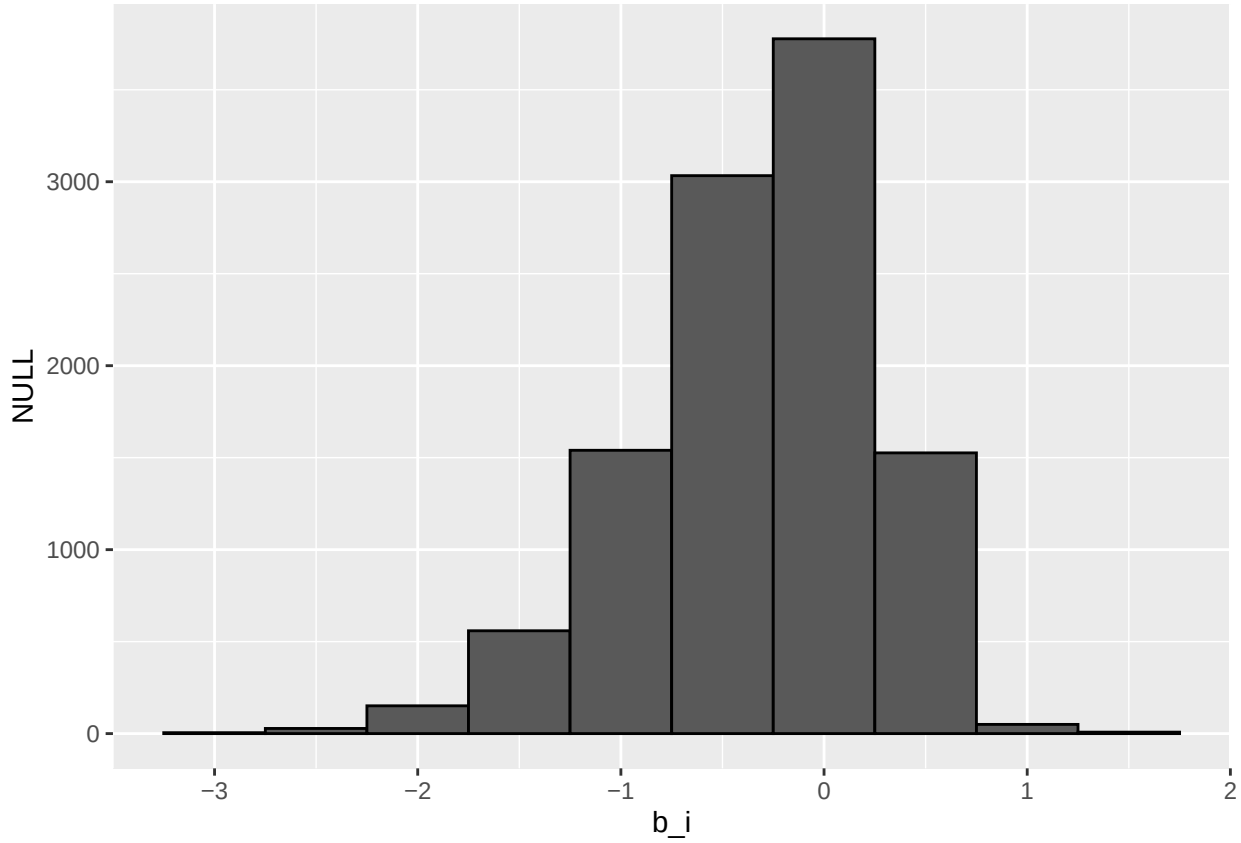
Create table to store model results

```
rmse_results <- data.frame(method = "Naive Approach", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Approach	1.060651

#Model 2: Movie Effect From our analysis we know that different movies tend to have different ratings. This movie effect or bias can be taken into consideration by adding the difference of movie bias and average rating.

```
# estimate movie bias b_i
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# plot movie bias
movie_avgs %>% qplot(b_i, geom = "histogram", bins = 10, data = ., color = I("black"))
```



Using the following code we can tell that by adding the movie effect it has improved the predictions.

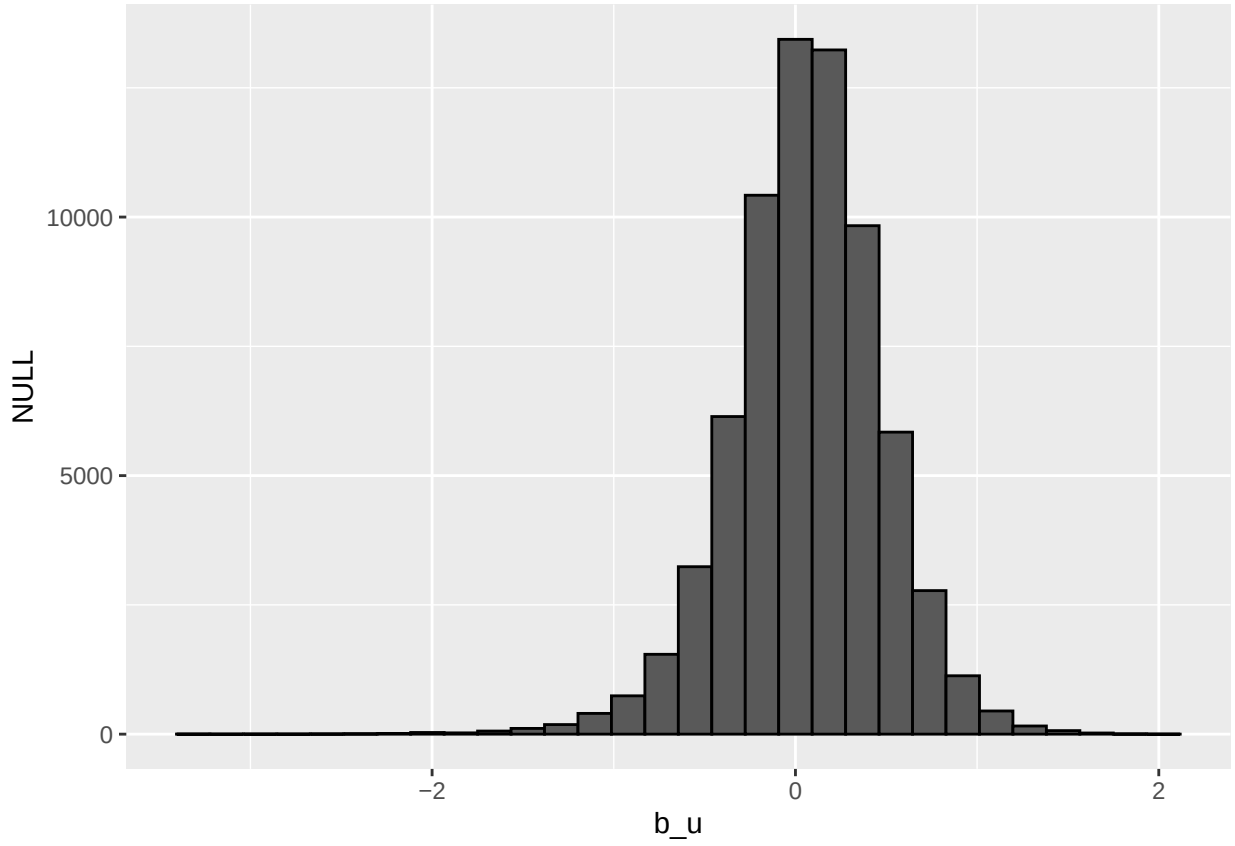
```
# calculate the predictions
predicted_ratings <- mu + validation %>%
```

```
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

# calculate rmse after modelling movie effect
model_2_rmse <- RMSE(predicted_ratings, validation$rating)

# add result to table
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Movie effect model",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Approach	1.0606506
Movie effect model	0.9437046

#Model 3: Movie + User Effect Similar to the movie bias, users will also rate movies differently than other users. For example certain users tend to always give really low ratings, while others may give fives for every movie they rate. We can apply the same concept as the movie bias here but this time take the difference of user bias, average rating, and movie bias.

```
# estimate user bias 'b_u' for all users
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

#plot user bias
user_avgs%>% qplot(b_u, geom="histogram", bins=30, data=., color=I("black"))
```

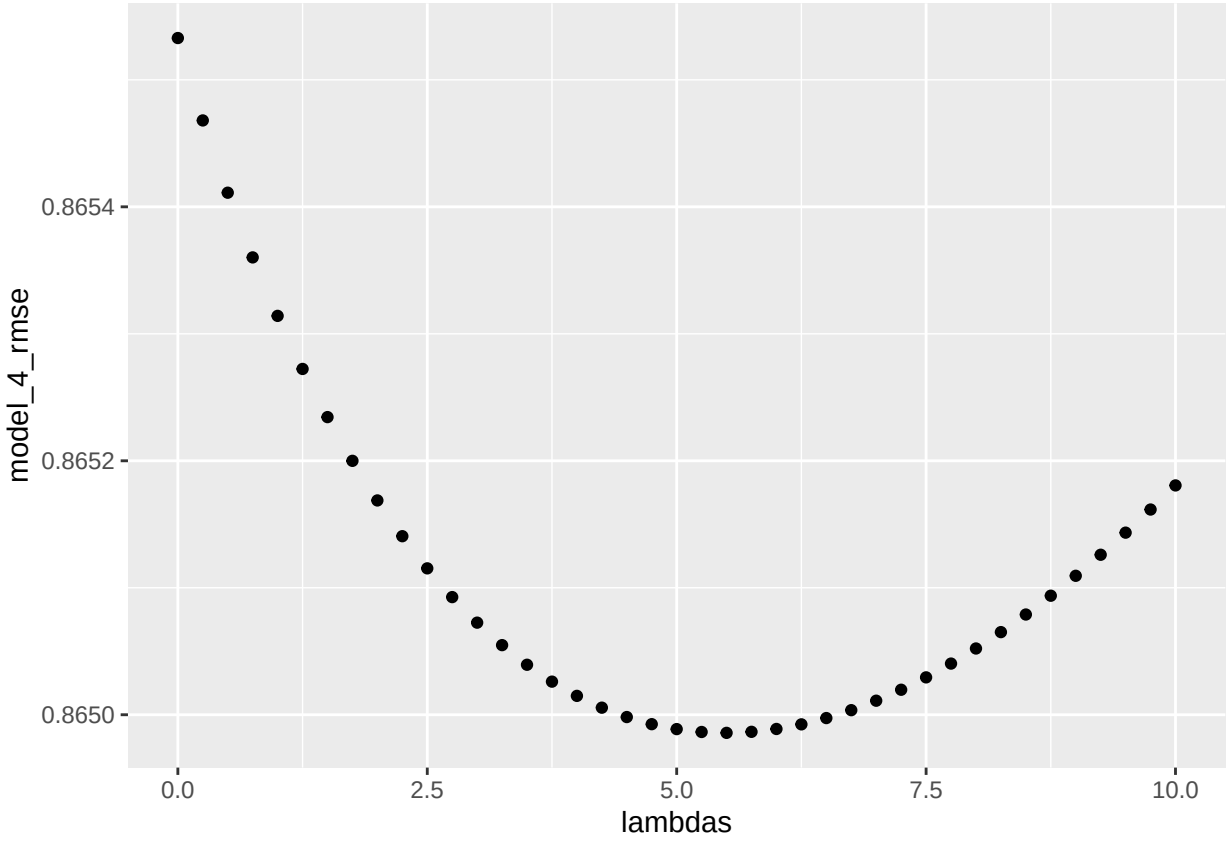



```r
# calculate predictions considering user effects in previous model
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# calculate rmse after modelling user specific effect in previous model
model_3_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Movie + User effects model",
                                     RMSE = model_3_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Approach	1.0606506
Movie effect model	0.9437046
Movie + User effects model	0.8655329

#Model 4: Regularizing The Movie + User Effect The purpose of regularization is to reduce the noise caused by variability in the data. We can do this by adding a penalty term to give less importance to certain data points. This is an imporatant step because some movies have been rated only a few times say 1 or 2 and can throw off the RMSE, which is sensitive to these large errors and having them will increase the residual squared errors.

```
#labdas is a tuning parameter, we can use cross-validation to choose the penalty term
lambdas <- seq(0,10,0.25)

model_4_rmse <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))
})

qplot(lambdas, model_4_rmse)
```



The optimal lambda is chosen by finding the one with the lowest RMSE

```
lambda <- lambdas[which.min(model_4_rmse)]
lambda
```

```
## [1] 5.5
```

```
#calculate and add record to table
rmse_results <- bind_rows(rmse_results,
                          data.frame(method="Regularized Movie + User effect model",
                                     RMSE = min(model_4_rmse)))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Approach	1.0606506
Movie effect model	0.9437046
Movie + User effects model	0.8655329
Regularized Movie + User effect model	0.8649857

Results

After developing the four models we can review and compare the RMSE to see any improvement in the models performance based on the initial Naive Approach.

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

method
Naive Approach
Movie effect model
Movie + User effects model
Regularized Movie + User effect model
Each model shows an improvement when com pared to the previous model. The Regularized Movie + User Effect Model

Conclusion

The Naive Approach which was the simplest model had an RMSE higher than 1 which is not good and means our error is off by one star. By implementing the Movie Effect the RMSE went down to 0.9439087, but by also considering the User Effect the model was able to lower the RMSE to 0.8653488 which is a significant improvement. Regularizing the movie and user effect however allowed us to get an even lower RMSE at 0.864810, which is about an 18.5% improvement from the baseline. For future enhancements an additional model could be created to train a regularization on the genres and years, which could potentially improve the RMSE even more.