Movielens Capstone Project

Nishaal Ajmera

18/05/2020

Introduction

Many companies especially those involved in e-commerce allow their clients to rate their products. Generally, users rate items in stars where 1 is the lowest and 5 is the highest rating given. These companies can then build recommendation systems from the data obtained to predict ratings that a customer give to a product. Therefore, if the customer is likely to give a high rating to a certain product then it will be recommended to that customer.
In 2006, Netflix offered a challenge to the data science community to improve their recommendation system by 10%. The data analysis and algorithm used in this project is motivated by the winning teams strategy. The Netflix data is not open to public however the GroupLens research lab generated their own database with over 20 million ratings for over 27000 movies by more than 138000 users. This project uses 10M version of the MovieLens dataset.
The key goals of this project are:

- to develop a machine learning algorithm that predicts the ratings that could be given by each user.
- to evaluate several algorithms using various approaches that would eliminate bias presented by the data.
- to produce a machine learning algorithm that gives the maximum accuracy by choosing the algorithm with the lowest root mean squared error (RMSE).

In this project, the data has been split into two sets primarily **edx** and **validation**. **Edx** dataset is used to train, tune and evaluate the models. **Validation** dataset is used to predict the ratings from the final model and calculate the final RMSE.

Data Analysis

First part of the analysis shows how the data can be obtained. The edx and validation data sets are created here.

```
################################
# 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")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ------------------------------------------------------------------------------------ t
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

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

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

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

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

```
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
```

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

```
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# 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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

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

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

Understanding the Data

```
#checking dimensions of edx
dim(edx)
```

```
## [1] 9000055       6
```

```
#checking the 5 rows of edx
head(edx,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
```

```
#number of distinct movies and users in edx
edx %>% summarize(n_movieId=n_distinct(movieId),
n_userId=n_distinct(userId))
```

```
##   n_movieId n_userId
## 1     10677    69878
```

Each row in the data represents a rating given by one user to one movie. When the two numbers ie. the number of distinct movies and users are multiplied, a number larger than 700 million is obtained, yet the data table has about 9000055 rows. Since each row represents a rating, it implies that not every user rated every movie.

Movie Titles with Total Ratings

Checking the movies with highest total ratings

```
#movie titles with total number of ratings
edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                 count
##      <dbl> <chr>                                                 <int>
##  1     296 Pulp Fiction (1994)                                   31362
##  2     356 Forrest Gump (1994)                                   31079
##  3     593 Silence of the Lambs, The (1991)                      30382
##  4     480 Jurassic Park (1993)                                  29360
##  5     318 Shawshank Redemption, The (1994)                      28015
##  6     110 Braveheart (1995)                                     26212
##  7     457 Fugitive, The (1993)                                  25998
##  8     589 Terminator 2: Judgment Day (1991)                     25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10     150 Apollo 13 (1995)                                      24284
## # ... with 10,667 more rows
```

Data Modification

The datasets are modified to allow exploratory analysis. A year and date columns are added.

```
#modifying data sets to add a year column and date column
#year column is taken from title and date is modified from timestamp column
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following object is masked from 'package:base':
##
##     date
```

```
edx<- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)),date= as_datetime(timestamp))
validation<- validation %>% mutate(year = as.numeric(str_sub(title,-5,-2)),date= as_datetime(timestamp))

# Viewing the first five columns of data sets
head(edx,5)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     292      5 838983421               Outbreak (1995)
## 4      1     316      5 838983392               Stargate (1994)
## 5      1     329      5 838983392 Star Trek: Generations (1994)
##                            genres year                date
## 1                  Comedy|Romance 1992 1996-08-02 11:24:06
## 2           Action|Crime|Thriller 1995 1996-08-02 10:58:45
## 3  Action|Drama|Sci-Fi|Thriller 1995 1996-08-02 10:57:01
## 4         Action|Adventure|Sci-Fi 1994 1996-08-02 10:56:32
## 5 Action|Adventure|Drama|Sci-Fi 1994 1996-08-02 10:56:32
```

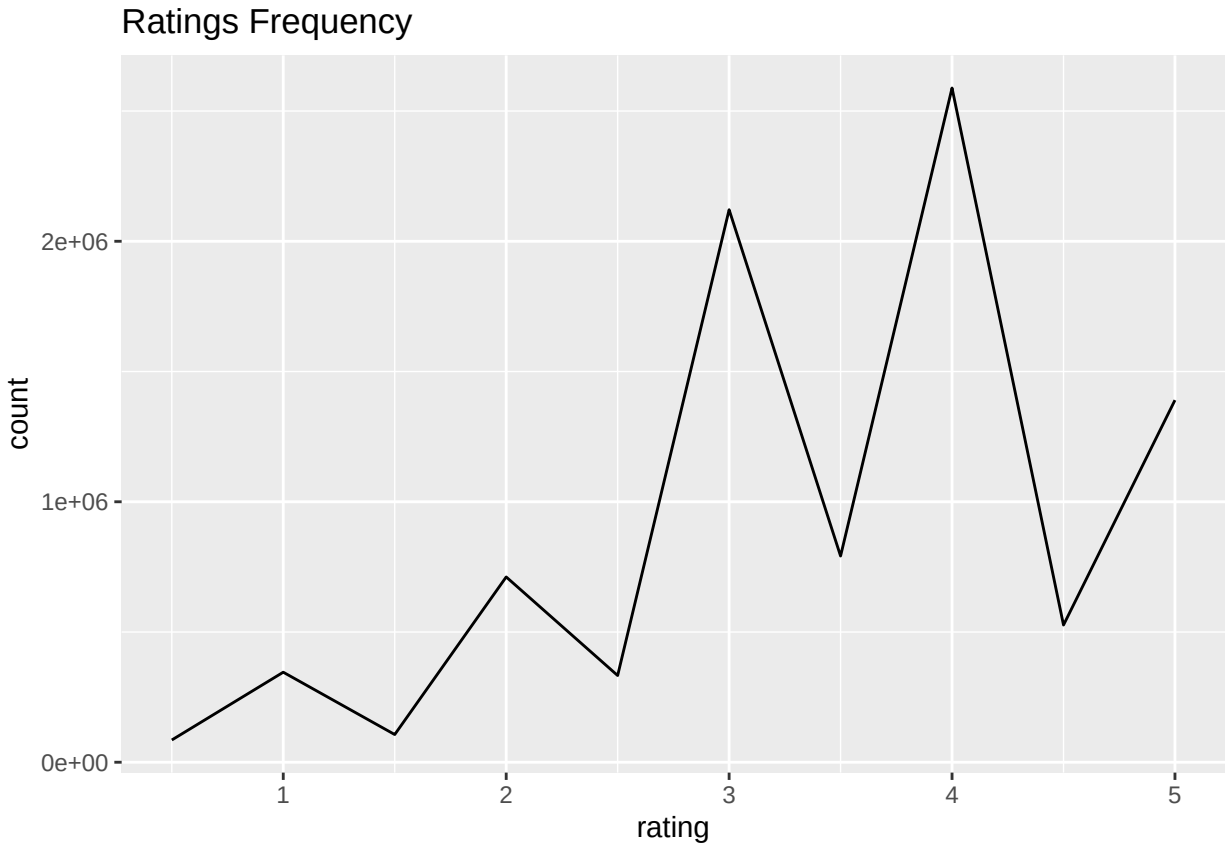
```
head(validation,5)
```

```
##   userId movieId rating timestamp                 title
## 1      1     231      5 838983392  Dumb & Dumber (1994)
## 2      1     480      5 838983653  Jurassic Park (1993)
## 3      1     586      5 838984068     Home Alone (1990)
## 4      2     151      3 868246450        Rob Roy (1995)
## 5      2     858      2 868245645 Godfather, The (1972)
##                              genres year                date
## 1                            Comedy 1994 1996-08-02 10:56:32
## 2 Action|Adventure|Sci-Fi|Thriller 1993 1996-08-02 11:00:53
## 3                  Children|Comedy 1990 1996-08-02 11:07:48
## 4         Action|Drama|Romance|War 1995 1997-07-07 03:34:10
## 5                       Crime|Drama 1972 1997-07-07 03:20:45
```

Exploratory Analysis

In this section, various aspects of the data are explored
Ratings

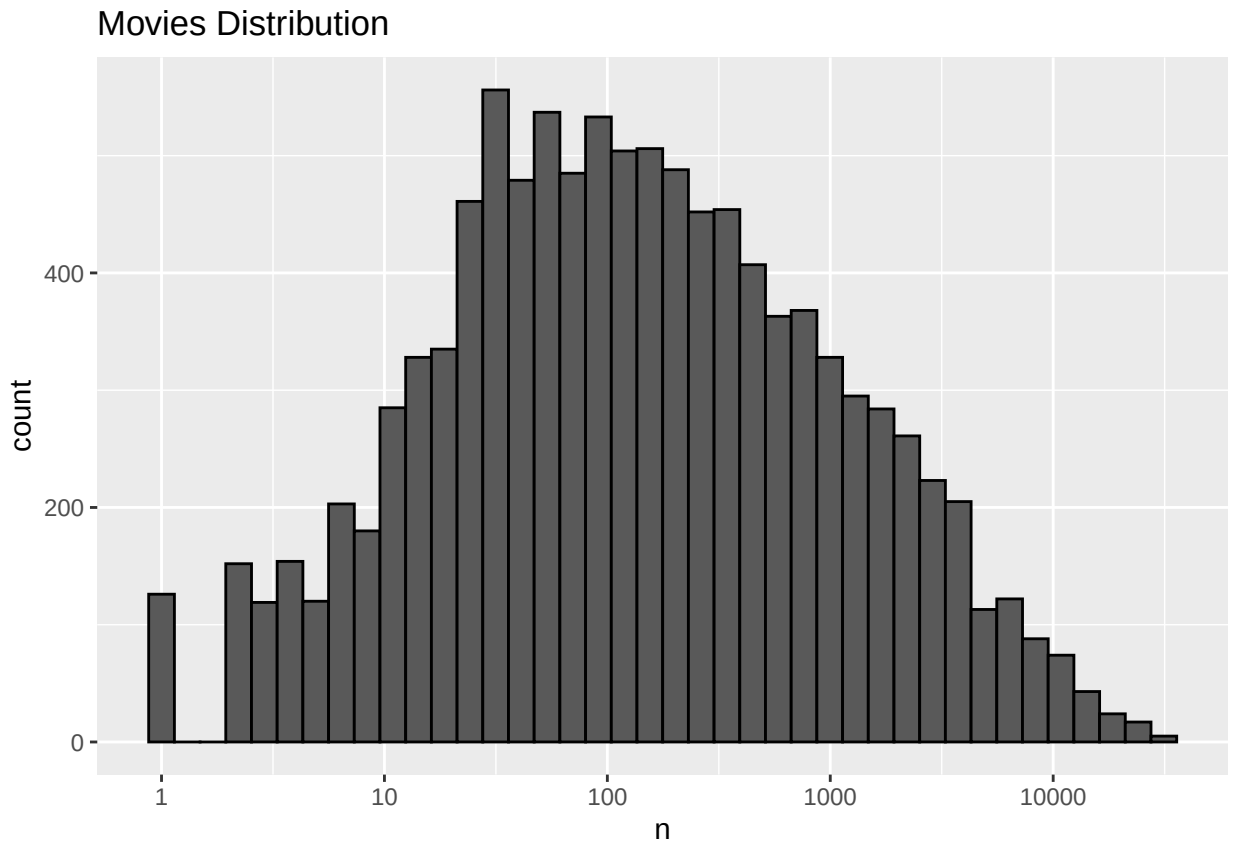
```
edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) + geom_line()+ ggtitle("Ratings Frequency")
```



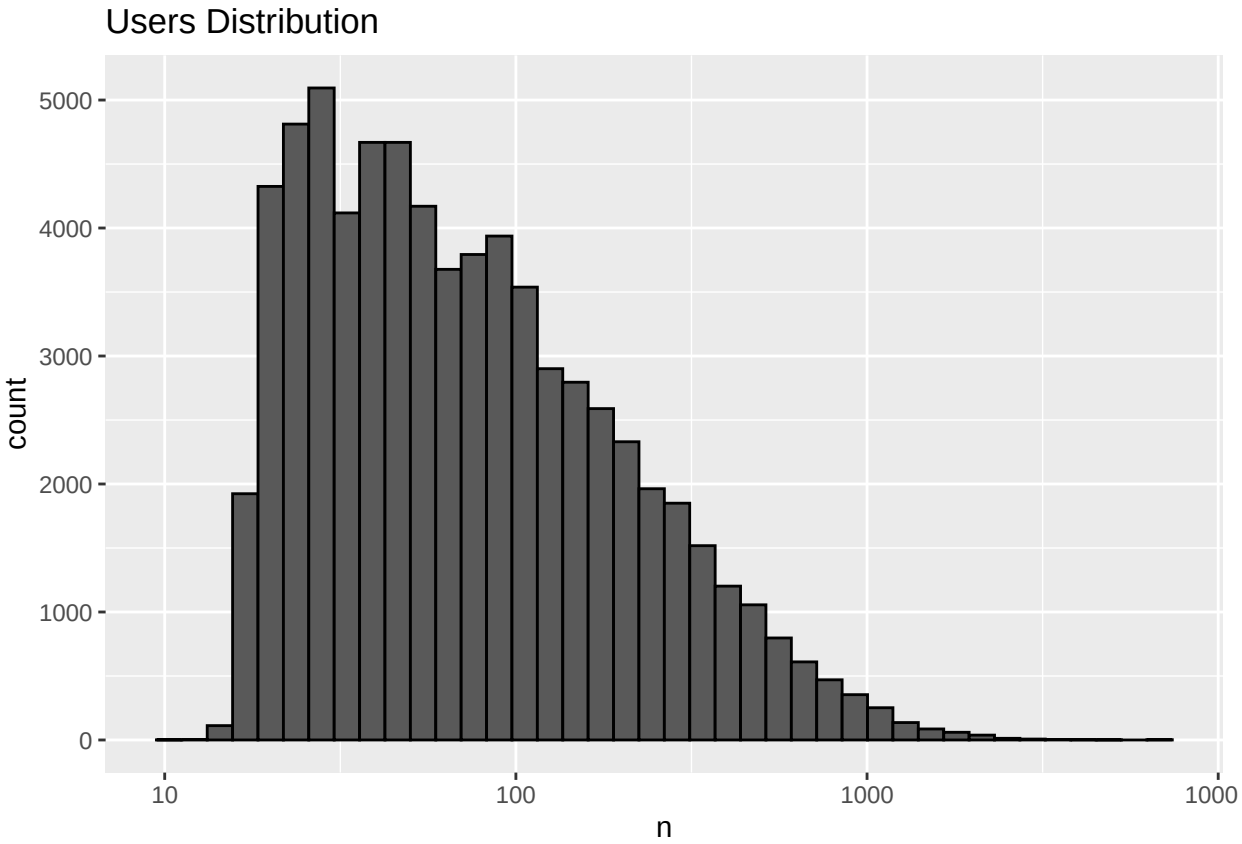
Users tend to rate 3 and 4 more often. Half star ratings are less common than full star ratings

Movies Distribution and Users Distribution

```
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 40, color = "black") +
  scale_x_log10() +
  ggtitle("Movies Distribution")
```



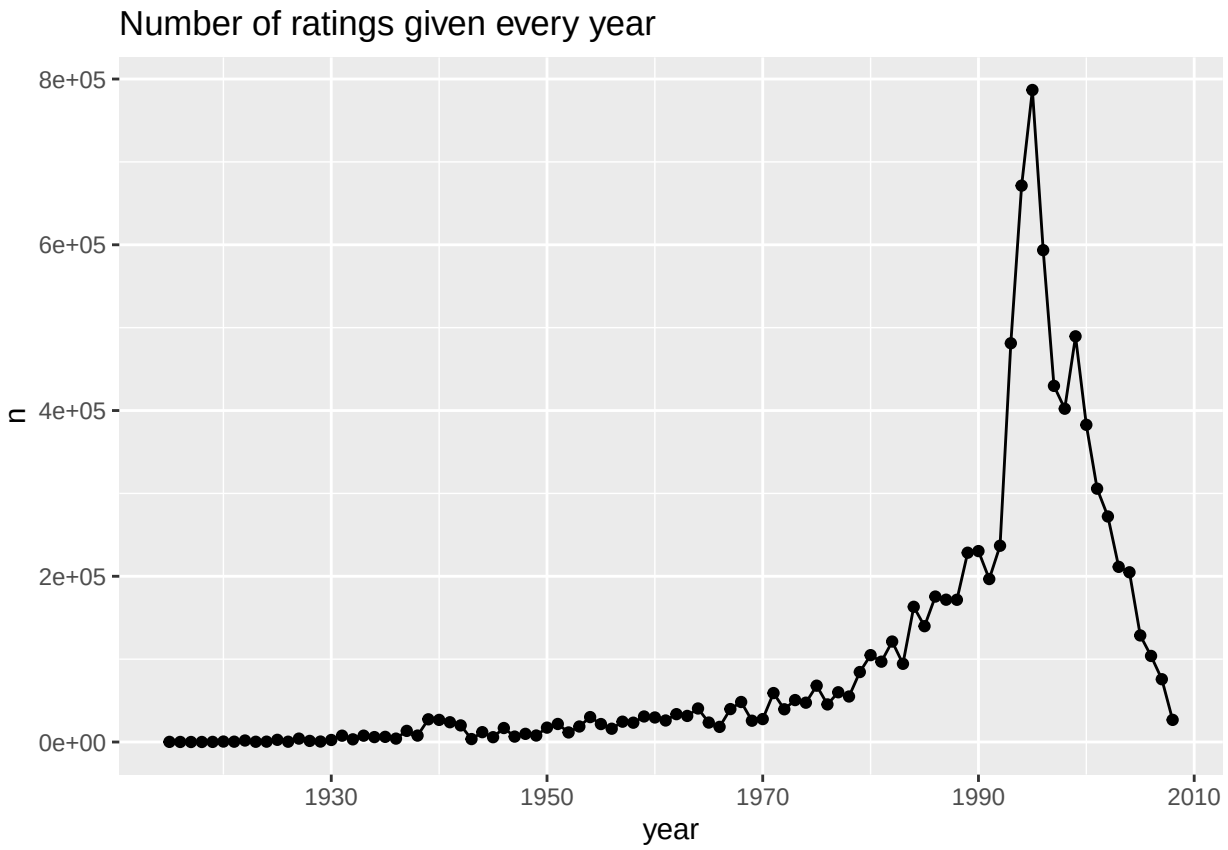

```
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 40, color = "black") +
  scale_x_log10() +
  ggtitle("Users Distribution")
```



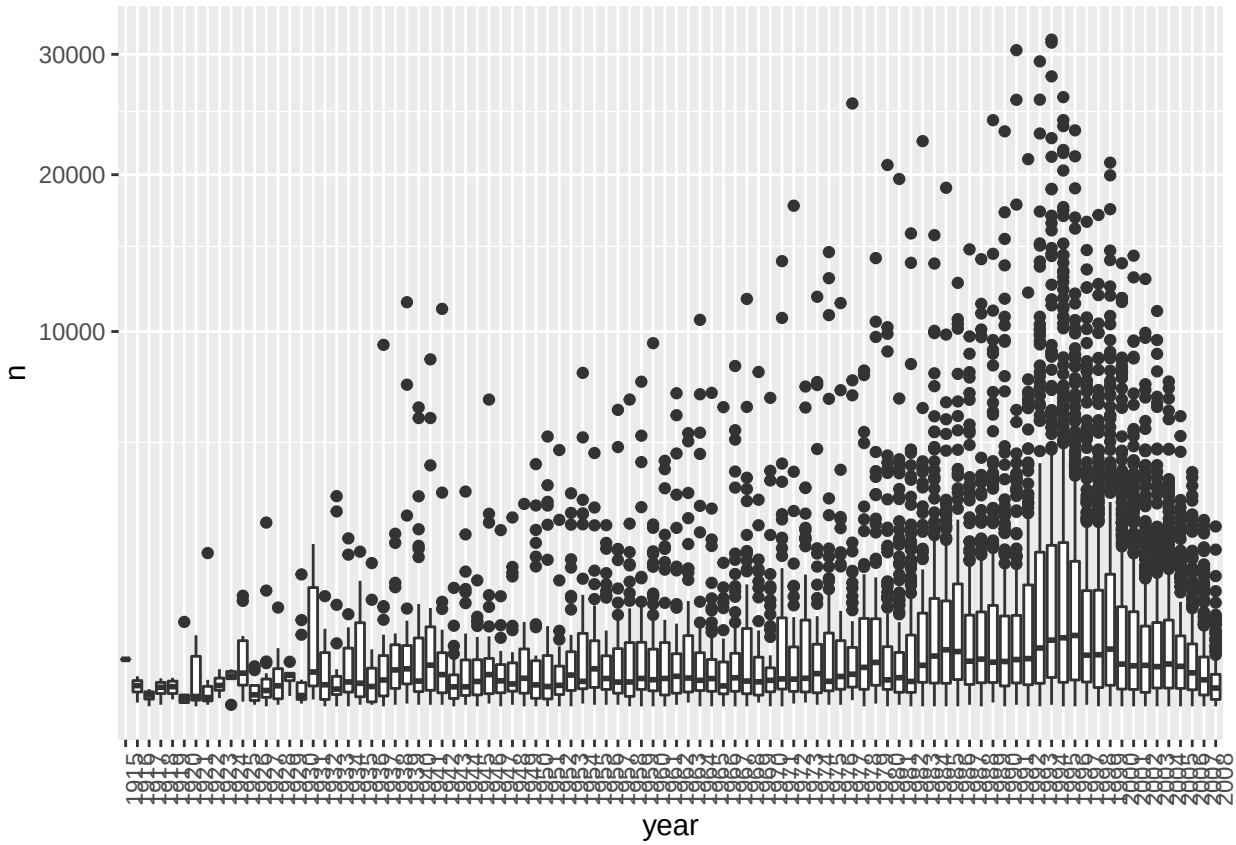
From the movies distribution, it is shown that some movies get rated more than the others. This can be explained because there are blockbuster movies that has many audience and artsy, independent movies watched by just a handful.
From the users distribution, it can be inferred that some users are more active at rating movies than others

Year Effect

```
#plot to show total ratings given by year
edx %>% group_by(year) %>% summarize(n=n()) %>% ggplot(aes(year,n))+ geom_point()+geom_line() +ggtitle(
```



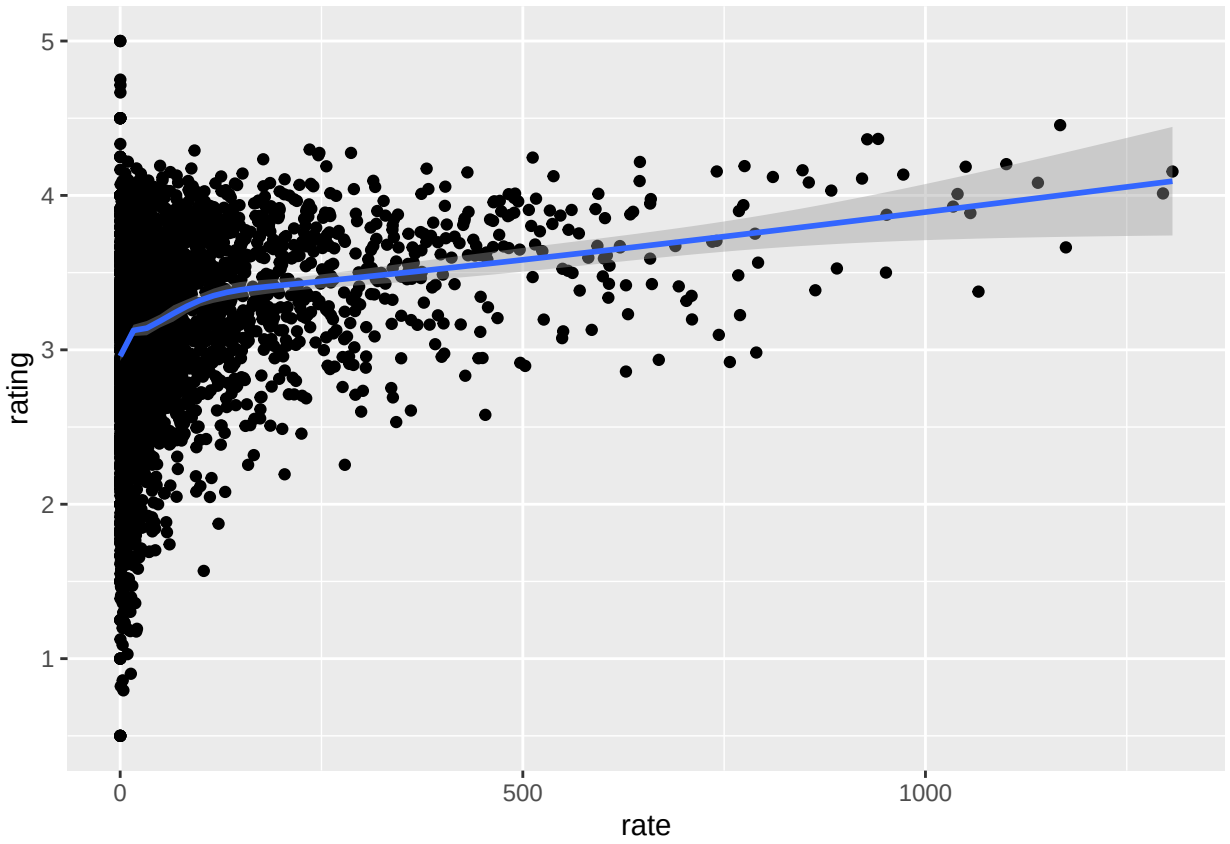

```
edx %>% group_by(movieId) %>%
  summarize(n = n(), year = as.character(first(year))) %>%
  qplot(year, n, data = ., geom = "boxplot") +
  coord_trans(y = "sqrt") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



It is observed that on average, movies that came out after 1993 get more ratings. It is also shown that recent movies,starting with 1993, the number of ratings decreases with year; the more recent a movie is, the less time users have had to rate it.
The plots show that the most frequently rated movies tend to have above average ratings. A rationale for this is more people watch popular movies. The movies released post 1993 can be stratified by ratings per year and the average ratings are computed to confirm the aforementioned.

```
#2018 is used as the end year to calculate the number of ratings per year
edx %>%
  filter(year >= 1993) %>%
  group_by(movieId) %>%
  summarize(n = n(), years = 2018 - first(year),
            title = title[1],
            rating = mean(rating)) %>%
  mutate(rate = n/years) %>%
  ggplot(aes(rate, rating)) +
  geom_point() +
  geom_smooth()
```

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

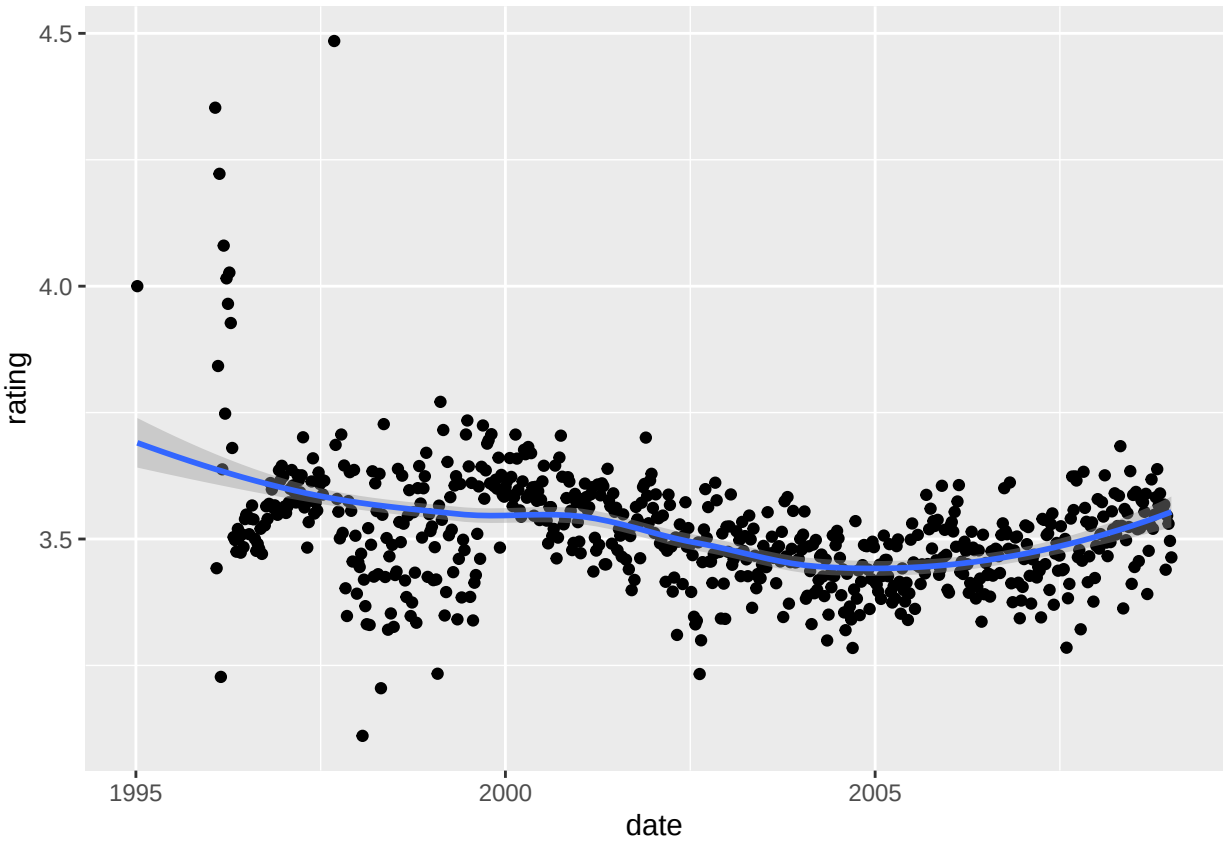


A trend that the more often a movie is rated the higher its average rating is observed.

Date Effect

```
#analysis to check if there is any effect of date on the rating patterns
#for easier analysis date columns in the datasets is rounded to the nearest week
edx<- edx %>% mutate(date = round_date(date, unit = "week"))
validation<- validation %>% mutate(date = round_date(date, unit = "week"))

edx %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth()
```

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



There is seem to be some kind of date effect where there is an increase and decrease in the number of ratings accorfinh to different dates

Training and Test Sets

```
#Creating training set to train the algorithm and test set for assessment from edx data
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Root Mean Squared Error (RMSE) function

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

Model 1: Average

A very basic model is used for the start with just the mean. This is a model that assumes the same rating for all movies and users with all the differences explained by random variation.

```r
mu<- mean(train_set$rating)
mu
```

```
## [1] 3.512574
```

```r
#RMSE with just the average
naive_rmse<- RMSE(test_set$rating,mu)
naive_rmse
```

```
## [1] 1.060704
```

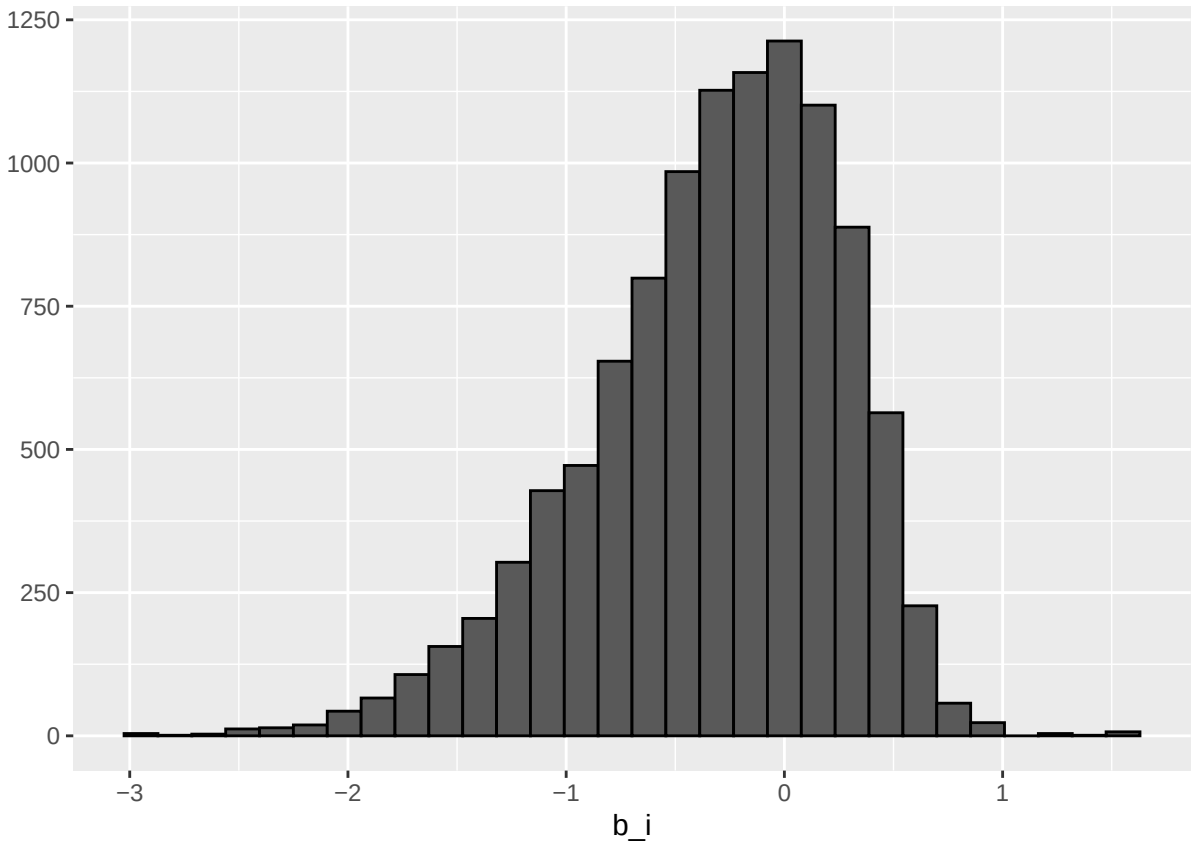
```r
#Tibble to compare RMSE
rmse_results <- tibble(model = "Average", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   model    RMSE
##   <chr>   <dbl>
## 1 Average  1.06
```

Model 2: Movie Effect

As shown previously, some movies are rated higher than others. Here **b_i** represents the average ranking for movie i.

```r
#Movie effect bias (b_i) estimating b_i
movie_avg<- train_set %>% group_by(movieId) %>% summarize(b_i=mean(rating-mu))

qplot(b_i, data = movie_avg, bins = 30, color = I("black"))
```




```
#RMSE for Movie effect
predicted_ratings <- mu + test_set %>%
  left_join(movie_avg, by='movieId') %>%
  pull(b_i)
movie_rmse<- RMSE(test_set$rating,predicted_ratings)

rmse_results<- bind_rows(rmse_results,
                         tibble(model="Movie Effect",RMSE=movie_rmse))
rmse_results
```

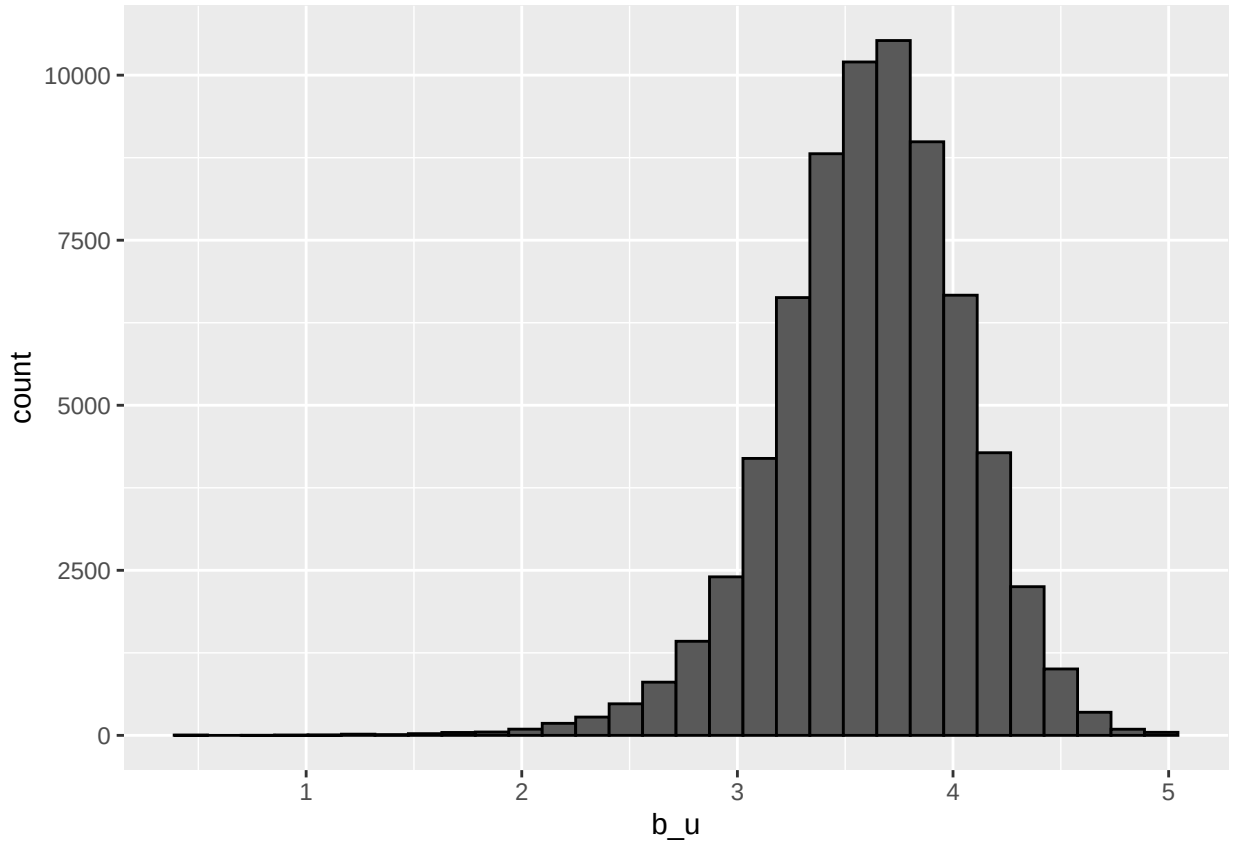
```
## # A tibble: 2 x 2
##   model         RMSE
##   <chr>        <dbl>
## 1 Average      1.06
## 2 Movie Effect 0.944
```

Model 3: User Effect

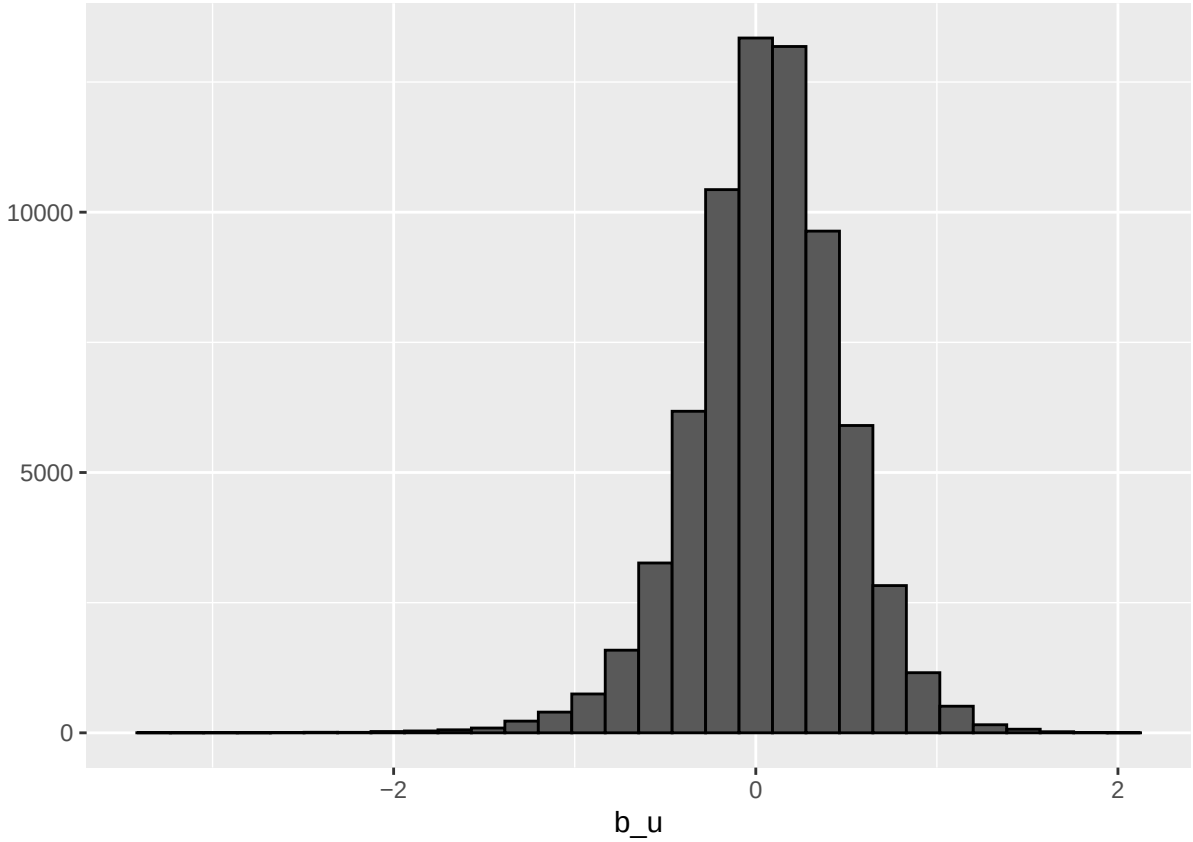
In the plot shown above, it can be seen that some users are cranky giving low ratings to good movies and some seem to love every movie by rating each one. Here **b_u** represents user specific effect

```
#Analysing user effect
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
```

```
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```




```
# User effect bias, estimating b_u
user_avg <- train_set %>%
  left_join(movie_avg, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

qplot(b_u, data = user_avg, bins = 30, color = I("black"))
```




```
predicted_ratings <- test_set %>%
  left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

movieuser_rmse<- RMSE(test_set$rating,predicted_ratings)

rmse_results<- bind_rows(rmse_results,
                         tibble(model="Movie and User Effect",RMSE=movieuser_rmse))
rmse_results
```

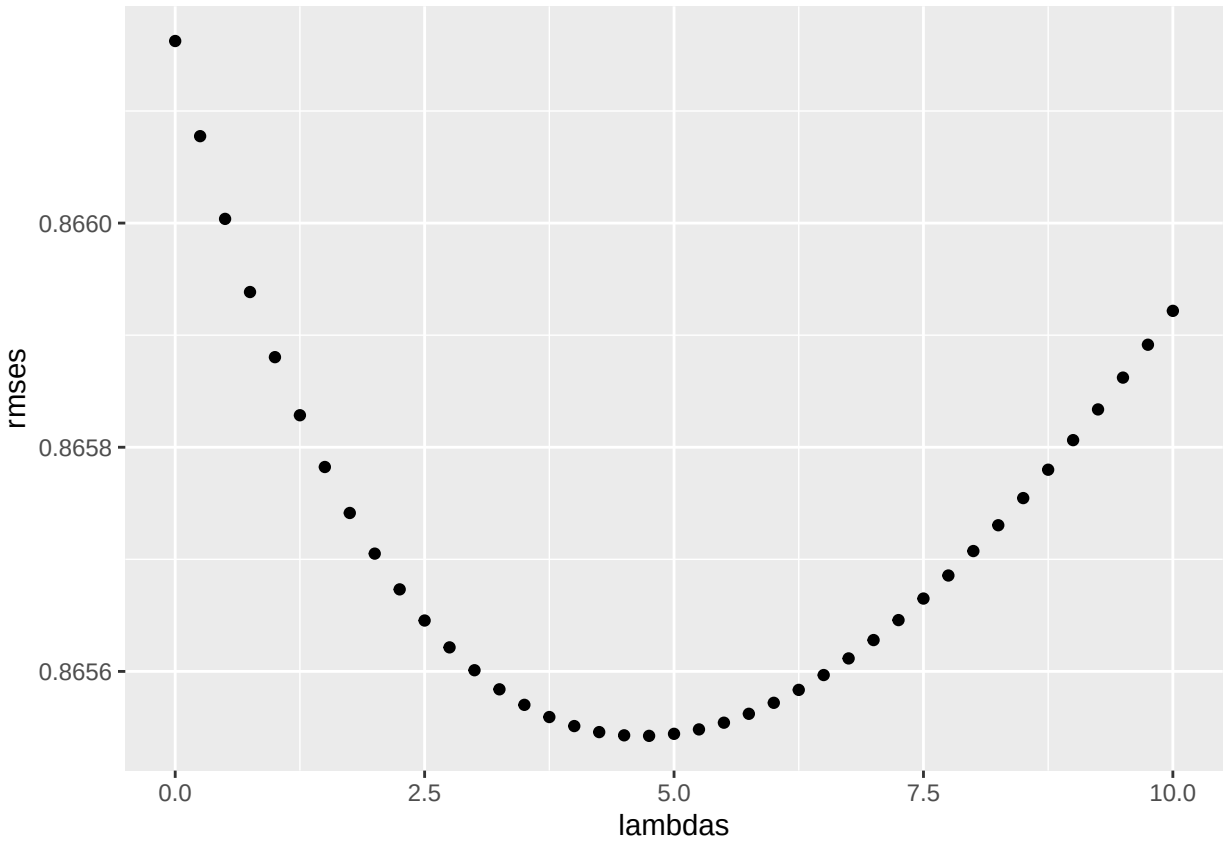
```
## # A tibble: 3 x 2
##   model                  RMSE
##   <chr>                 <dbl>
## 1 Average               1.06
## 2 Movie Effect          0.944
## 3 Movie and User Effect 0.866
```

Regularization

Regularization can reduce noisy results which increase the RMSE. Large estimates that are formed using small sample sizes can be penalized. These penalized estimates can improve the RMSE greatly.

Model 4: Regularized Movie and User Effect

```r
# Perform cross-validation to choose optimal tuning parameter, lambda
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) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})

# RMSE versus lambda plot to select lambda at which RMSE is smallest
qplot(lambdas, rmses)
```




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

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

```
# Use lambda to evaluate regularized b_i and b_u
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

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

#apply regularized model to predict ratings on test set
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

#Compute RMSE and add it to results tibble
reg_movieuser_rmse<- RMSE(test_set$rating,predicted_ratings)
rmse_results<- bind_rows(rmse_results,
```

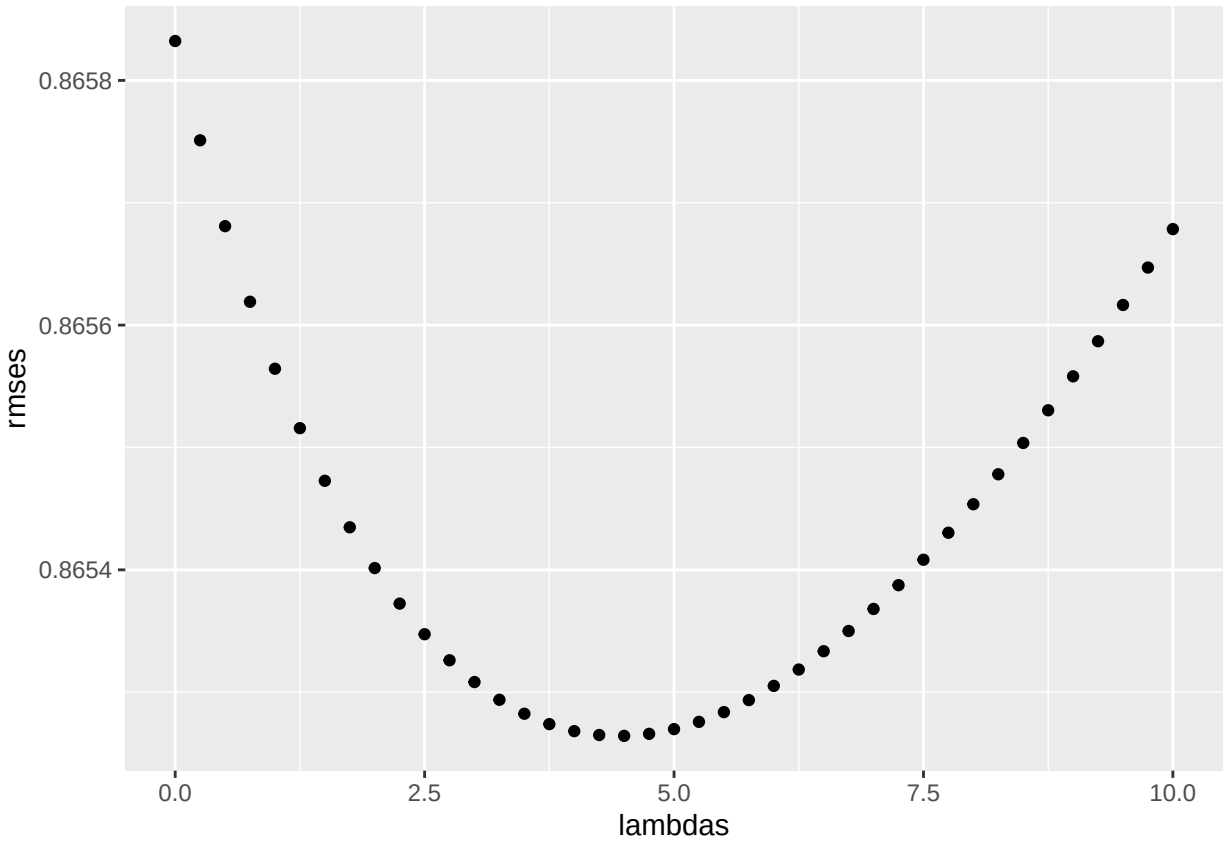
```
                          tibble(model="Regularized Movie and User Effect",RMSE=reg_movieuser_rmse))

print.data.frame(rmse_results)
```

```
##                                model      RMSE
## 1                            Average 1.0607045
## 2                        Movie Effect 0.9437144
## 3             Movie and User Effect 0.8661625
## 4 Regularized Movie and User Effect 0.8655425
```

Model 5: Regularized Movie, User and Year Effect

In this model, the year effect is added as seen above that there is a year effect on the ratings given by the users. The year specific effect is represented by **b_r**.

```
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))

  b_r<- train_set %>%
    left_join(b_i,by="movieId") %>%
    left_join(b_u,by="userId") %>%
    group_by(year) %>%
    summarize(b_r=sum(rating - b_i - b_u - mu)/(n()+l))

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_r,by="year") %>%
    mutate(pred = mu + b_i + b_u + b_r) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})

# RMSE versus lambda plot to select lambda at which RMSE is smallest
qplot(lambdas, rmses)
```




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

```
## [1] 4.5
```

```
# Use lambda to evaluate regularized b_i, b_u and b_r
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

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

b_r<- train_set %>%
  left_join(b_i,by="movieId") %>%
  left_join(b_u,by="userId") %>%
  group_by(year) %>%
  summarize(b_r=sum(rating - b_i - b_u - mu)/(n()+lambda))


#apply regularized model to predict ratings on test set
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
```

```
  left_join(b_u, by = "userId") %>%
  left_join(b_r,by="year") %>%
  mutate(pred = mu + b_i + b_u + b_r) %>%
  pull(pred)

#Compute RMSE and add it to results tibble
reg_movieuseryear_rmse<- RMSE(test_set$rating,predicted_ratings)
rmse_results<- bind_rows(rmse_results,
                         tibble(model="Regularized Movie, User and Year Effect",RMSE=reg_movieuseryear_
print.data.frame(rmse_results)
```

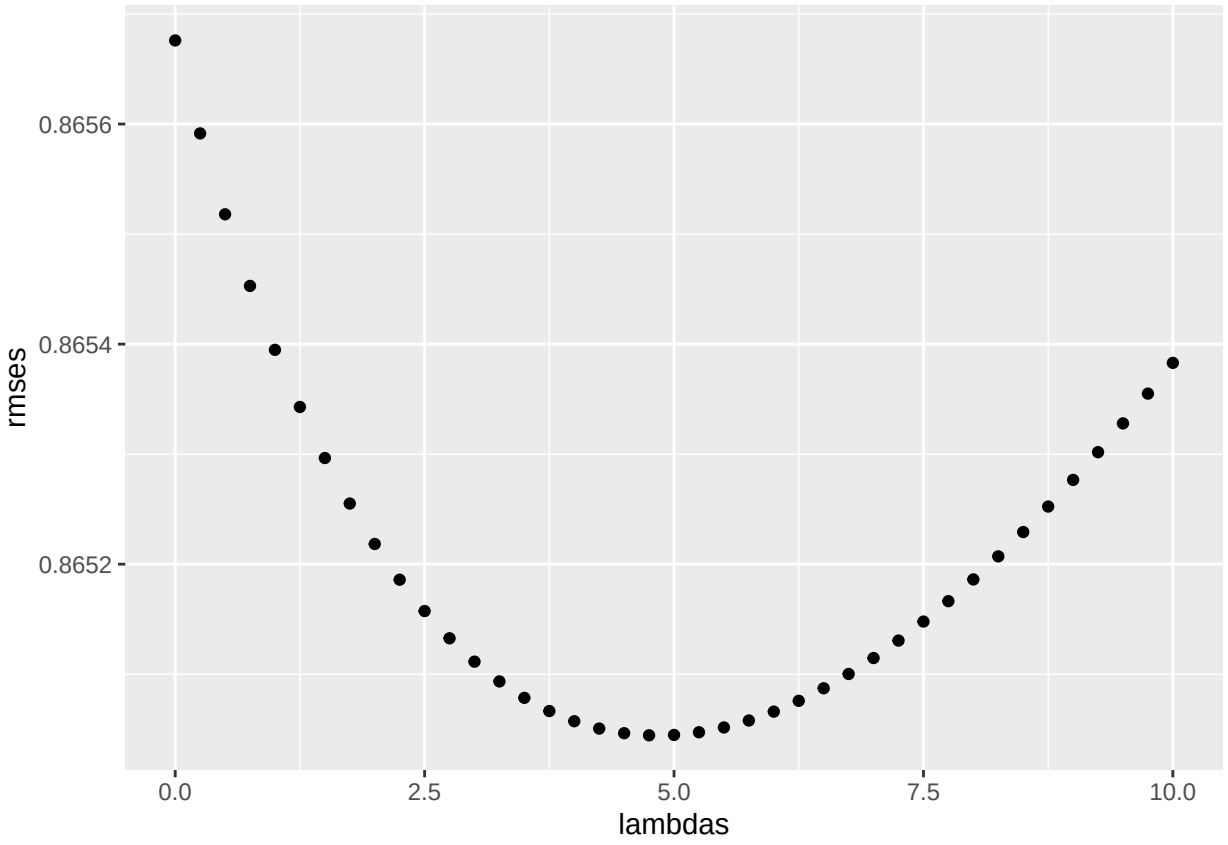
```
##                                       model      RMSE
## 1                                   Average 1.0607045
## 2                             Movie Effect 0.9437144
## 3                      Movie and User Effect 0.8661625
## 4       Regularized Movie and User Effect 0.8655425
## 5 Regularized Movie, User and Year Effect 0.8652643
```

Model 6: Regularized Movie, User, Year and Date to the nearest week effect

As seen in the RMSE results, it has greatly improved. However, the RMSE could be further reduced. In this model, regularized date effect is added to the model. The date specific effect is represented by **b_t**.

```
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))

b_r<- train_set %>%
  left_join(b_i,by="movieId") %>%
  left_join(b_u,by="userId") %>%
  group_by(year) %>%
  summarize(b_r=sum(rating - b_i - b_u - mu)/(n()+l))

b_t<- train_set %>%
  left_join(b_i,by="movieId") %>%
  left_join(b_u,by="userId") %>%
  left_join(b_r,by="year") %>%
  group_by(date) %>%
  summarize(b_t=sum(rating - b_i - b_u - b_r - mu)/(n()+l))

predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
```

```
  left_join(b_u, by = "userId") %>%
  left_join(b_r,by="year") %>%
  left_join(b_t,by="date") %>%
  mutate(pred = mu + b_i + b_u + b_r + b_t) %>%
  pull(pred)

return(RMSE(predicted_ratings, test_set$rating))
})

# RMSE versus lambda plot to select lambda at which RMSE is smallest
qplot(lambdas, rmses)
```




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

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

```
# Use lambda to evaluate regularized b_i, b_u, b_r and b_t
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
```

```
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

b_r<- train_set %>%
  left_join(b_i,by="movieId") %>%
  left_join(b_u,by="userId") %>%
  group_by(year) %>%
  summarize(b_r=sum(rating - b_i - b_u - mu)/(n()+lambda))

b_t<- train_set %>%
  left_join(b_i,by="movieId") %>%
  left_join(b_u,by="userId") %>%
  left_join(b_r,by="year") %>%
  group_by(date) %>%
  summarize(b_t=sum(rating - b_i - b_u - b_r - mu)/(n()+lambda))


#apply regularized model to predict ratings on test set
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_r,by="year") %>%
  left_join(b_t,by="date") %>%
  mutate(pred = mu + b_i + b_u + b_r + b_t) %>%
  pull(pred)


#Compute RMSE and add it to results tibble
reg_movieuseryeardate_rmse<- RMSE(test_set$rating,predicted_ratings)
rmse_results<- bind_rows(rmse_results,
                         tibble(model="Regularized Movie, User, Year and Date Effect",RMSE=reg_movieuse
print.data.frame(rmse_results) #Choose the model with the lowest RMSE
```

```
##                                          model      RMSE
## 1                                      Average 1.0607045
## 2                                Movie Effect 0.9437144
## 3                      Movie and User Effect 0.8661625
## 4           Regularized Movie and User Effect 0.8655425
## 5     Regularized Movie, User and Year Effect 0.8652643
## 6 Regularized Movie, User, Year and Date Effect 0.8650445
```

Results

```
print.data.frame(rmse_results)
```

```
##                                          model      RMSE
## 1                                      Average 1.0607045
## 2                                Movie Effect 0.9437144
## 3                      Movie and User Effect 0.8661625
## 4           Regularized Movie and User Effect 0.8655425
## 5     Regularized Movie, User and Year Effect 0.8652643
## 6 Regularized Movie, User, Year and Date Effect 0.8650445
```

From the above results it is observed that the RMSE reduces when the year(**b_r**) and date(**b_t**) effects are taken into account in the regularized model. Therefore, the final model used to predict the ratings for the **Validation** dataset will be the **Regularized Movie, User, Year and Date effect* model.

Applying Final Model

```
#Predict ratings of Validation
lambda
```

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

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

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

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


#apply regularized model to predict ratings on test set
predicted_ratings <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_r,by="year") %>%
  left_join(b_t,by="date") %>%
  mutate(pred = mu + b_i + b_u + b_r + b_t) %>%
  pull(pred)

#Final RMSE
RMSE(validation$rating,predicted_ratings)
```

```
## [1] 0.8643114
```

Conclusion

In this project various models have been tested to build a recommendation system for users to watch movies. The RMSE results show that by taking the different bias that might exist such as the Movie, User, Year and Date effects in to account the RMSE value has greatly reduced when compared to using the mean only. Regularization greatly helped in minimizing noisy data effects. To conclude the final model can used to give recommendations to users since it has a small RMSE value. In the future some analysis over the genres can be done to investigate if there is any effects of different genres on the ratings given.