Capstone Project - Movie Rating Prediction

Steven Wild

06 March, 2022

1 Introduction

This report forms a part of the project submission for the HarvardX PH125.9x course : Data Science: Capstone. The aim of the project is to create a movie recommendation system, making use of the 10M version of the MovieLens dataset. All data was downloaded as per assignment instructions. (Section 1.1)

The report is organised into the following sections:

1. Introduction
    1. Data download and preparation
2. Analysis
    1. Data verification
    2. Naive model
    3. Movie effects model
    4. User effects model
    5. Year effects model
    6. Genre effects model
    7. Penalised least squares / tuning parameters
3. Results
4. Conclusion.

The approach taken follows something similar to the one described in the data science textbook, *Introduction to Data Science* by Rafael Irizarry (link)

Relationships between the variables were considered and their effects were modeled in an incremental fashion to establish a final combined model. The standard deviation of the difference between the real and predicted ratings was was used to determine the accuracy of the predictions. This residual mean squared error or RMSE was calculated from the validation set and minimised to determine the best prediction model.

1.1 Data download and preparation

The code in this section was supplied as part of the assignment instruction with minor modifications to save / retrieve a copy of the dataset to a local folder.

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

```
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)

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

if(!file.exists("MovieLens.RData"))
  {
  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")

  # if using R 4.0 or later:
  movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(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")
  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)

  #remove all temp files
  rm(dl, ratings, movies, test_index, temp, movielens, removed)

} else {
  load("MovieLens.RData")
}
```

The following lines were added to save a copy of the dataset to a local folder to prevent having to download again.

```
if(!file.exists("MovieLens.RData"))
{
save(edx, validation, file = "MovieLens.RData")
}
```

2 Analysis

2.1 Data verification

First we need to examine the layout and nature of the data. Each entry in the dataset represents individual movie ratings with a unique userId for each person making the rating, a movieId matched to the movie titles, a rating from 0-5 (with 0 being poor and 5 being excellent), a genre (or combination of a number of genres) and a timestamp representing the date and time when the rating was given.

```
head(edx)
```

```
##    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)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres year
## 1:                Comedy|Romance 1992
## 2:          Action|Crime|Thriller 1995
## 3:  Action|Drama|Sci-Fi|Thriller 1995
## 4:       Action|Adventure|Sci-Fi 1994
## 5: Action|Adventure|Drama|Sci-Fi 1994
## 6:       Children|Comedy|Fantasy 1994
```

A potential additional useful variable to consider could be the year that the movie was released. We add the extra column by extracting the year from the title.

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

```
##    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)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres year
## 1:                Comedy|Romance 1992
## 2:          Action|Crime|Thriller 1995
## 3:  Action|Drama|Sci-Fi|Thriller 1995
## 4:       Action|Adventure|Sci-Fi 1994
## 5: Action|Adventure|Drama|Sci-Fi 1994
## 6:       Children|Comedy|Fantasy 1994
```

As part of the data cleanup process we should check for incomplete records starting with a check for NAs in the ratings and newly created year columns. There are no NAs in the dataset.

```
sum(is.na(edx$rating))
```

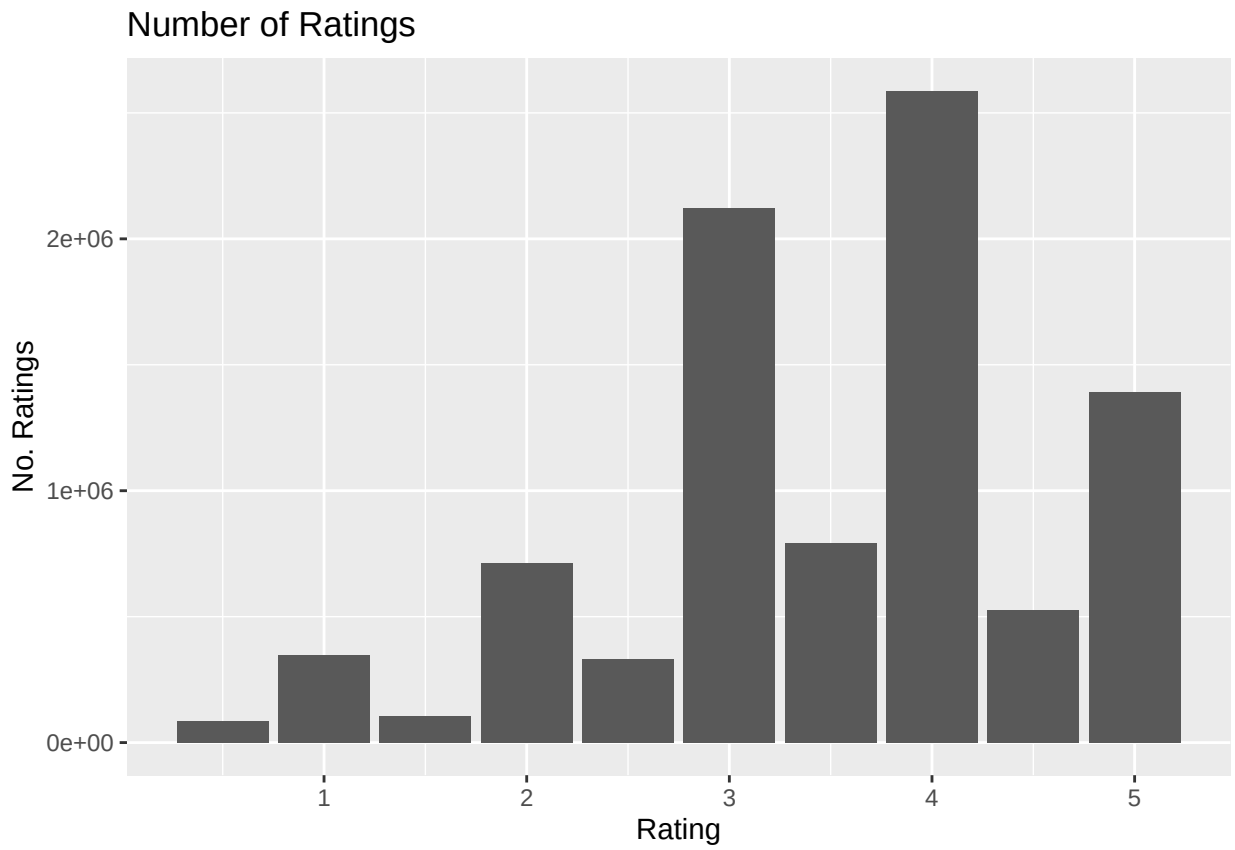
```
## [1] 0
```

```
sum(is.na(edx$year))
```

```
## [1] 0
```

The distribution of the ratings can also be viewed to check for 0 ratings and it will also give us a feel for the data. We can see that the ratings range from 0-5, in half point increments with 4.0 the most prevalent rating given.

```
edx %>%
  group_by(rating) %>%
  ggplot(aes(rating)) +
  geom_bar() +
  xlab("Rating") +
  ylab("No. Ratings") +
  ggtitle("Number of Ratings")
```



We can also draw a summary of the data with the following code.

```
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              year
##  Length:9000055     Length:9000055     Min.   :1915
##  Class :character   Class :character   1st Qu.:1987
##  Mode  :character   Mode  :character   Median :1994
##                                        Mean   :1990
##                                        3rd Qu.:1998
##                                        Max.   :2008
```

The data contains individual ratings for 69,878 unique users and 10,677 unique movies.

```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

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

the movies in the dataset cover the period of 93 years, from 1915 to 2008

```
min(edx$year)
```

```
## [1] 1915
```

```
max(edx$year)
```

```
## [1] 2008
```

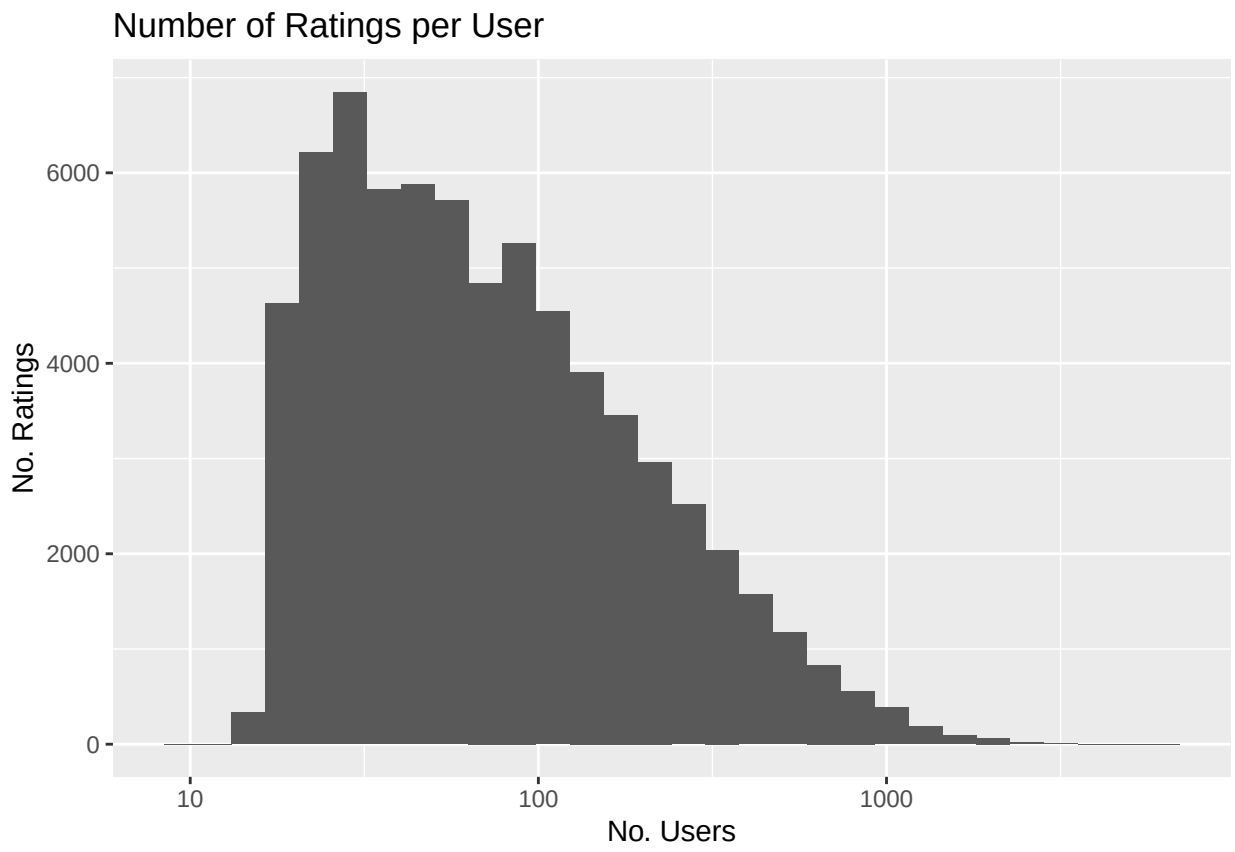
```
max(edx$year) - min(edx$year)
```

```
## [1] 93
```

In order to determine which potential parameters we might be able to use in our modeling we can investigate the number of ratings per grouping of the variables and their resultant distributions.

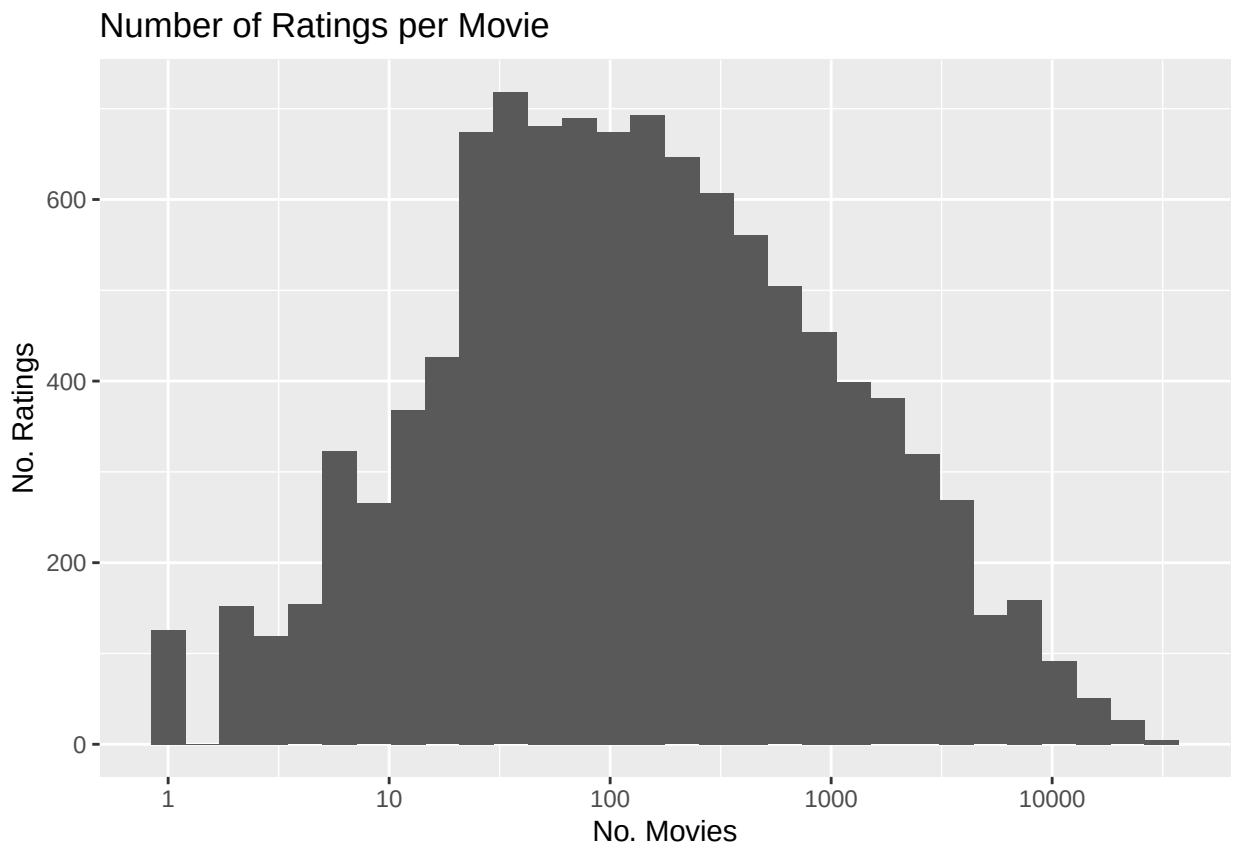
Looking at the count of users plotted against the number of ratings we can see that not all users make the same number of ratings which could skew the predictions we make. In addition we also know that not every user will have the same taste in movies and we will therefore include a bias correction for user.

```
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30) +
  scale_x_log10() +
  xlab("No. Users") +
  ylab("No. Ratings") +
  ggtitle("Number of Ratings per User")
```



If we consider the count of movies plotted against the number of ratings we can also draw the observation that some of the less popular movies are rated very little compared to some of the more mainstream ones. We should therefore consider making some kind of correction based on the number of ratings per movie.

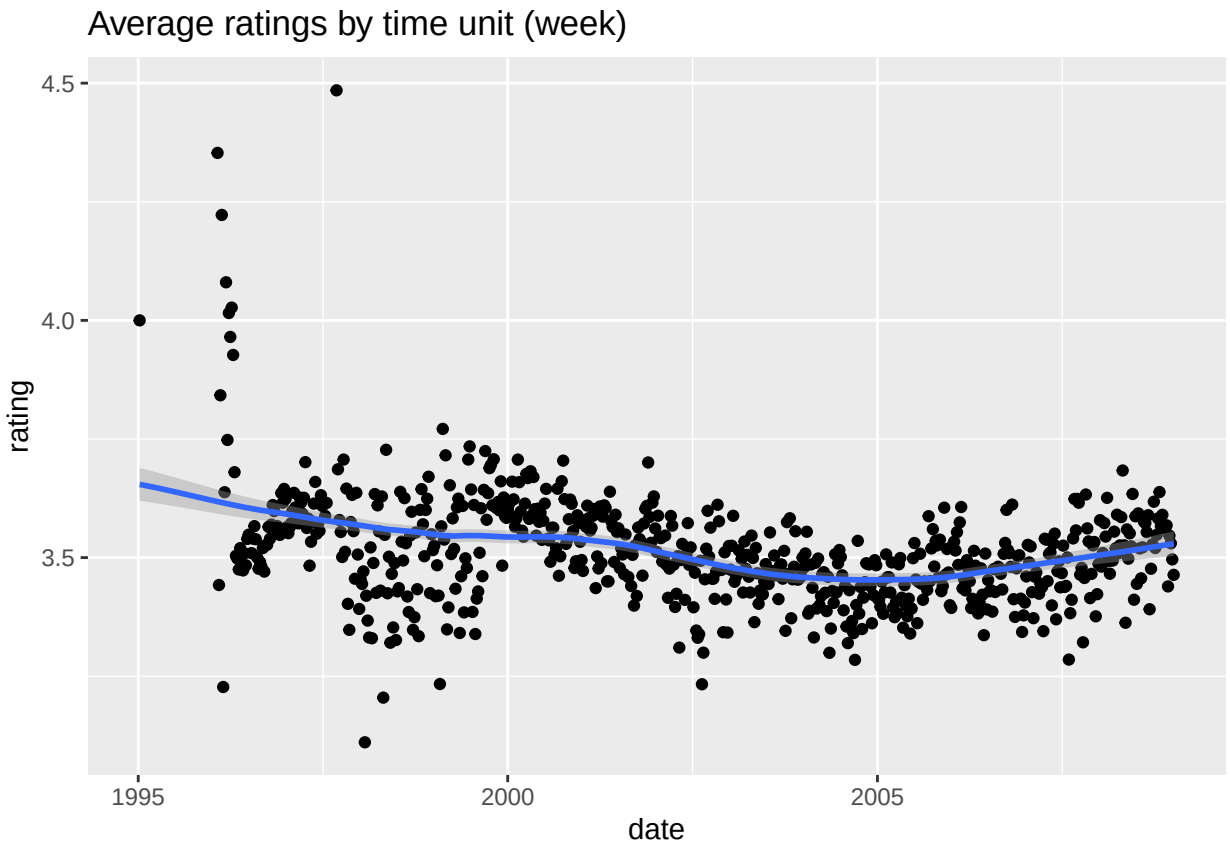
```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30) +
  scale_x_log10() +
  xlab("No. Movies") +
  ylab("No. Ratings") +
  ggtitle("Number of Ratings per Movie")
```



The plot below illustrates the potential effect of time on the rating. It seems that the effect of the timestamp is not that notable to include it as a parameter in the prediction model and it will therefore not be considered further.

```
edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth(method = "loess", span = 0.5, method.args = list(degree=1)) +
  ggtitle("Average ratings by time unit (week)")
```

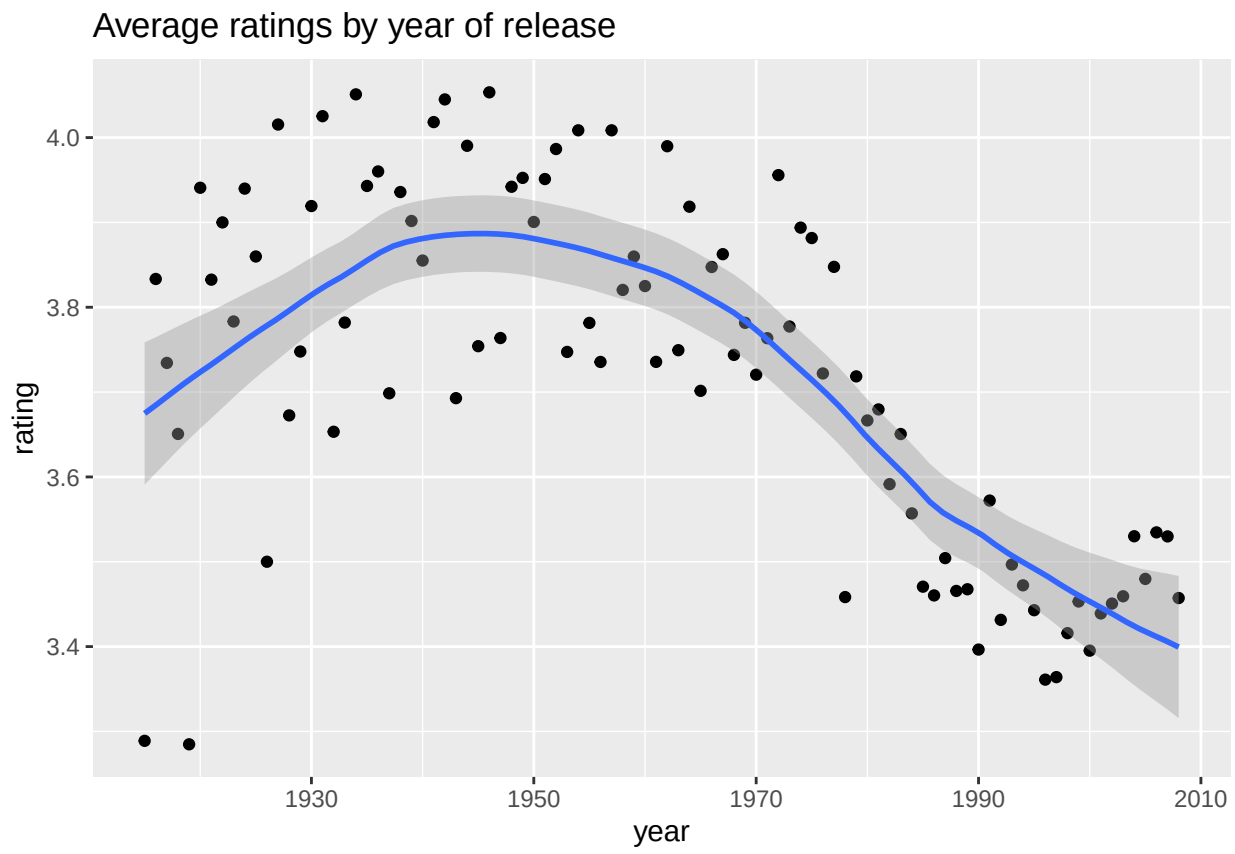
```
## `geom_smooth()` using formula 'y ~ x'
```



If we consider the effect of the year of release on the potential rating then we see from the plot below that there is a marked decrease in the mean rating for more modern movies. For this reason we will include a bias parameter in our model that will adjust for year of release.

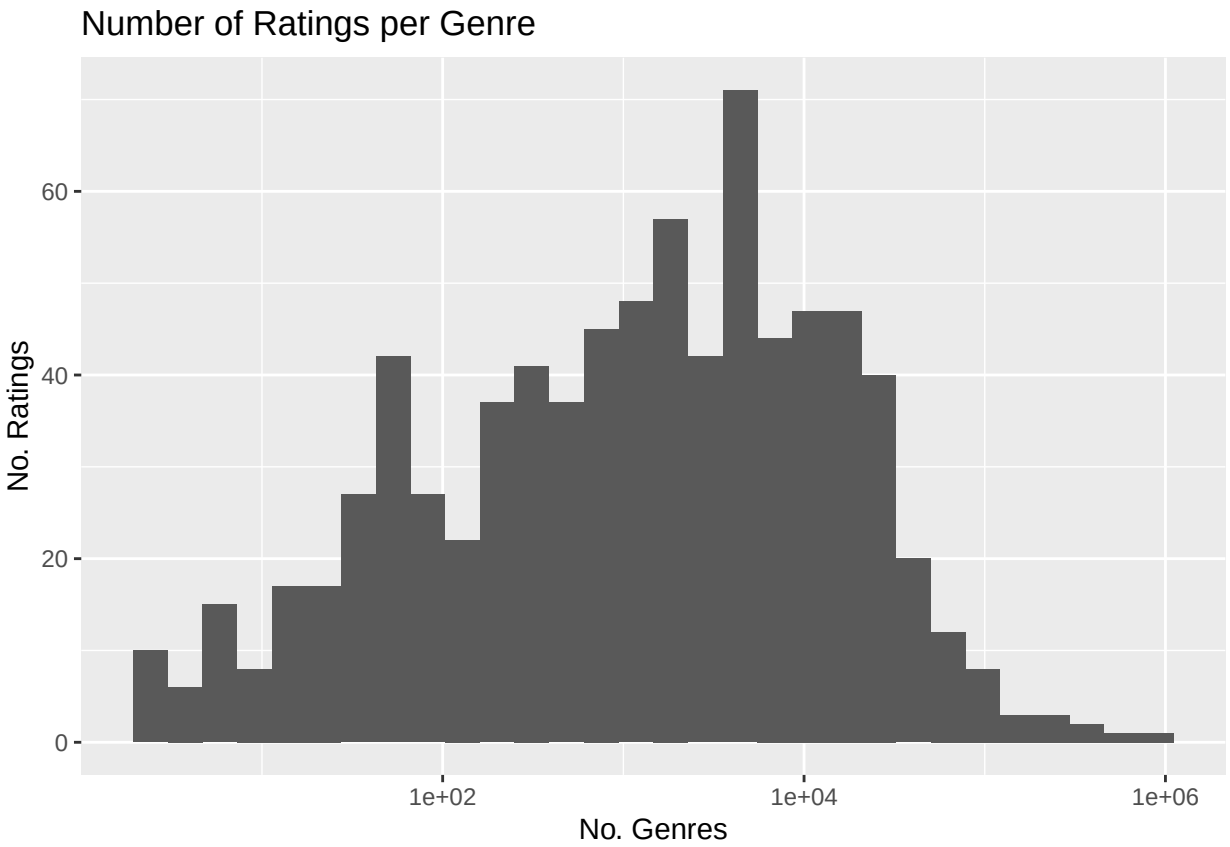
```
edx %>% group_by(year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(year, rating)) +
  geom_point() +
  geom_smooth(method = "loess", span = 0.5, method.args = list(degree=1)) +
  ggtitle("Average ratings by year of release")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Plotting the number of ratings per genre on a histogram also illustrates another potential source of bias on the predicted ratings. We can see that some genres have very few ratings and this is a function of how popular the genre is and not how good the movie may be. For this reason we need to include some form of adjustment in our final model.

```
edx %>%
  dplyr::count(genres) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30) +
  scale_x_log10() +
  xlab("No. Genres") +
  ylab("No. Ratings") +
  ggtitle("Number of Ratings per Genre")
```



We can use the following function in our modeling to determine the estimate of the error loss in each of our predictions. We will use the residual mean squared error (RMSE) between the actual ratings in the validation dataset and our predictions.

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

2.2 Naive model

To determine a baseline for our prediction model we will take mean of the distribution and use it as our first prediction. This mean of 3.512 is compared with each entry in the validation set and the error loss determined through our RMSE function described above. (Refer to the distribution of ratings histogram and RMSE function in Section 2.1)

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

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

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

For ease of reference we save all the RMSEs we calculate into a table.

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

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

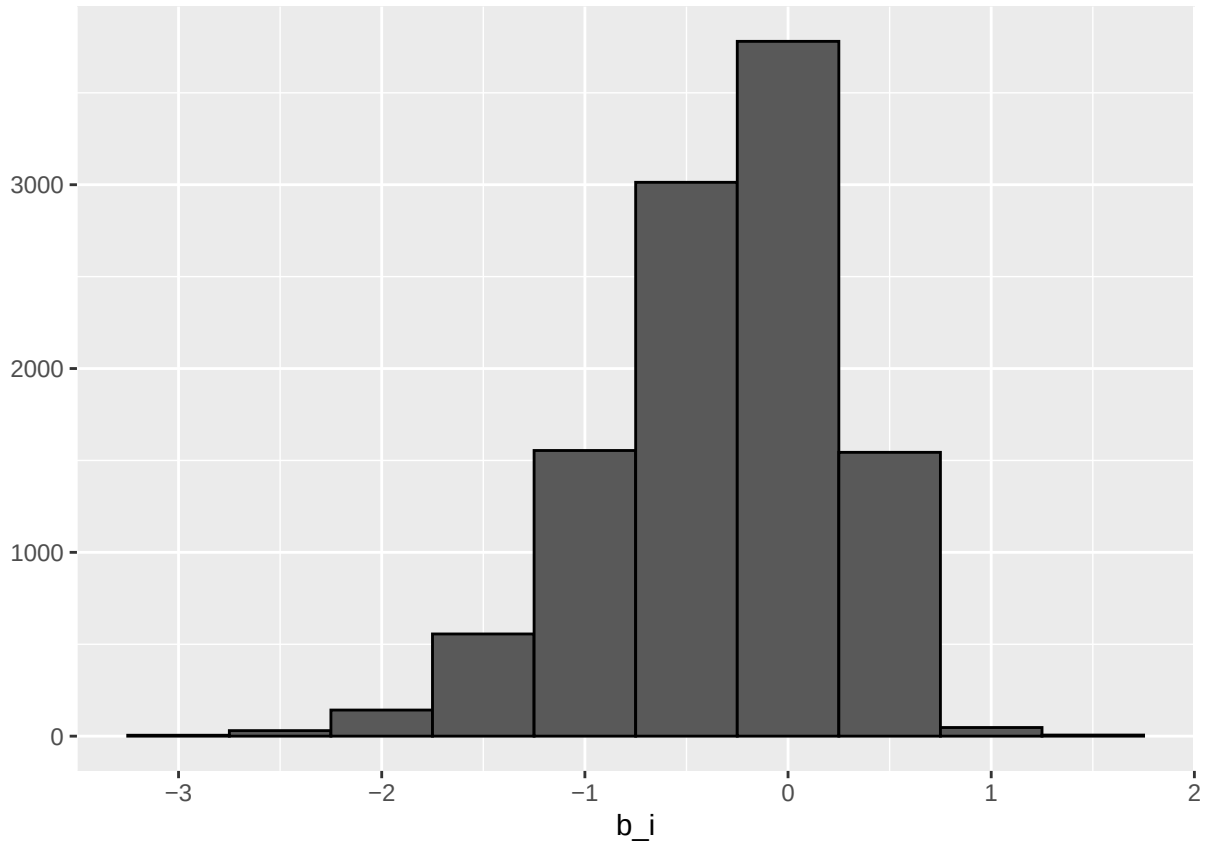
The outcome of this model is an RMSE of 1.061, meaning that our predictions can be out by more than a full point on the scale from 0-5. We now need to see how we can optimise this result.

2.3 Movie effects model

In Section 2.1 above, we plotted the number of ratings per movie and determined that there might be some kind of error introduced if we make the assumption that all ratings are equal. We can clearly see that some movies have very few ratings and others have many, which could well skew the prediction. We could determine a bias parameter and apply it to the mean, thereby correcting our model for what we have called the "movie effect".

First we create a table of all the unique movies and then calculate a distance from the mean for each entry in the training set. The plot shows the distribution of these distances around the mean (0)

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

qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



When we make our prediction from the validation set we apply the mean together with the bias parameter (b_i) and then calculate the new RMSE. again, for ease of comparison we append the new results to a consolidated table.

```
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
movie_model_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- rmse_results %>% add_row(tibble_row(method = "Movie effect model", RMSE = movie_model_rm
movie_model_rmse
```

```
## [1] 0.9439087
```

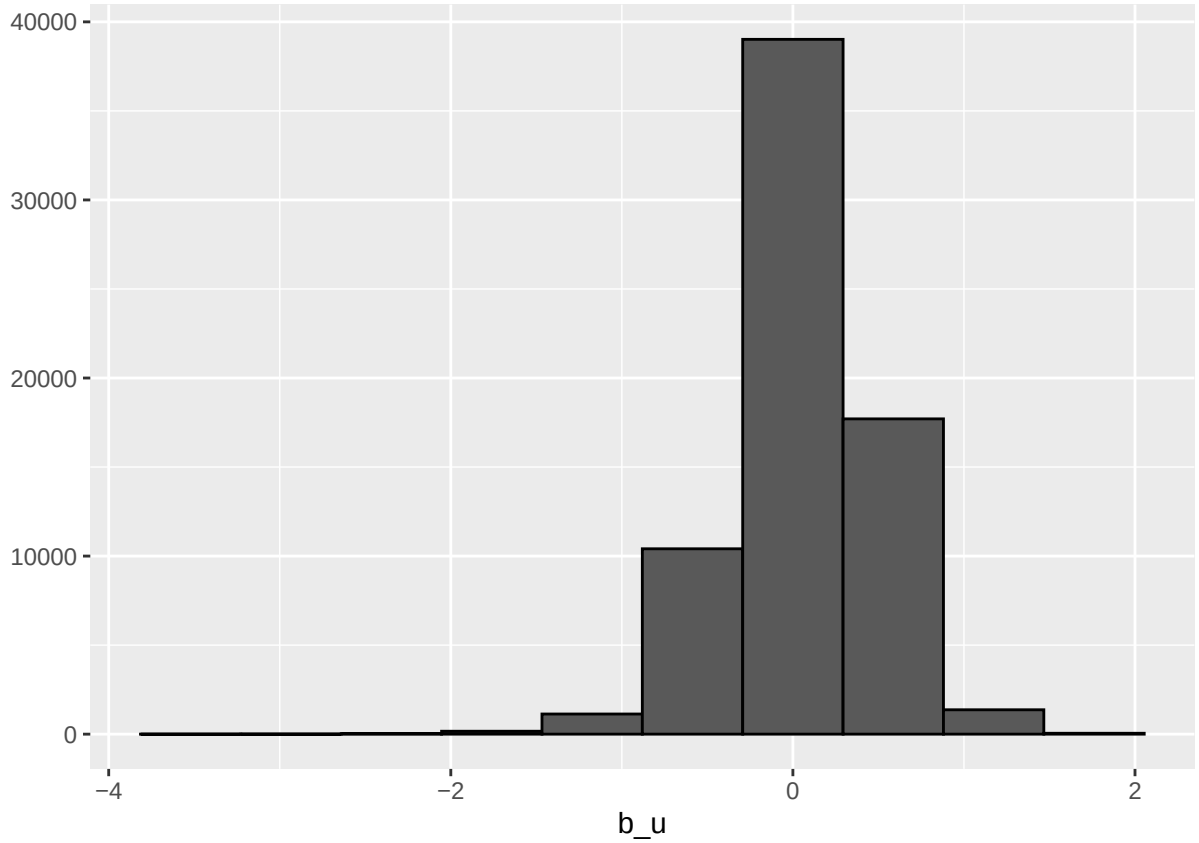
The new RMSE has improved considerably to 0.944 indicating that our predictions are getting closer to the real values in the validation dataset.

2.4 User effects model

If we consider the plot we made in Section 2.1 "No. of Ratings per User" we can again see that some users rated a large number of movies while others only rated a few. Each person also has a specific taste in movies that needs to be accounted for. This can introduce another level of bias in that the users that made very few ratings could well be on some obscure movies and we therefore do not get a fair representation. We will again calculate a bias parameter we can apply to the mean and we'll call it "user effect".

The plot below show the error (b_u) between the real rating and the rating per user which we can then apply to the mean.

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

qplot(b_u, data = user_avgs, bins = 10, color = I("black"))
```



We add it incrementally to the correction we already applied for the "movie effect" (b_i).

```
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
user_model_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- rmse_results %>% add_row(tibble_row(method = "User effect model", RMSE = user_model_rmse
user_model_rmse
```

```
## [1] 0.8653488
```

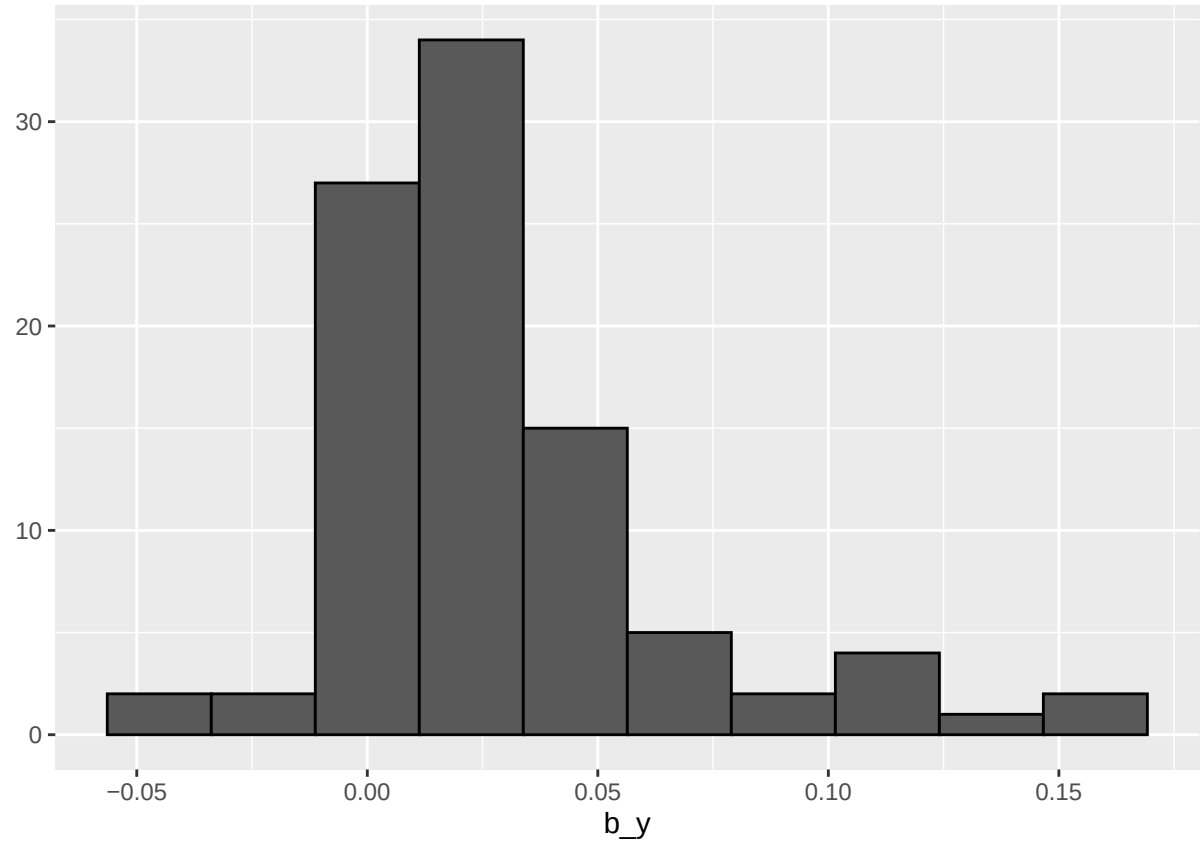
Our incremental RMSE applying both the movie and user effects has improved the RMSE to 0.865, a substantial improvement.

2.5 Year effects model

From our exploration of the data in Section 2.1 above we noted that in the plot "Average ratings by year of release" it seems that mean ratings have indeed changed dramatically over time. In order to apply a fair rating we will use this as an additional bias parameter in our prediction.

First, we calculate the bias and then plot it to see the distribution around the mean. (again, this is incremental to the parameters already applied in the sub-sections above)

```
year_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u))

qplot(b_y, data = year_avgs, bins = 10, color = I("black"))
```



We then need to calculate the new predictions on the validation set and add the calculated error loss (incremental) to the table for comparison.

```
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year') %>%
  mutate(pred = mu + b_i + b_u + b_y) %>%
  pull(pred)
year_model_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- rmse_results %>% add_row(tibble_row(method = "Release year model", RMSE = year_model_rms
year_model_rmse
```

```
## [1] 0.8650043
```

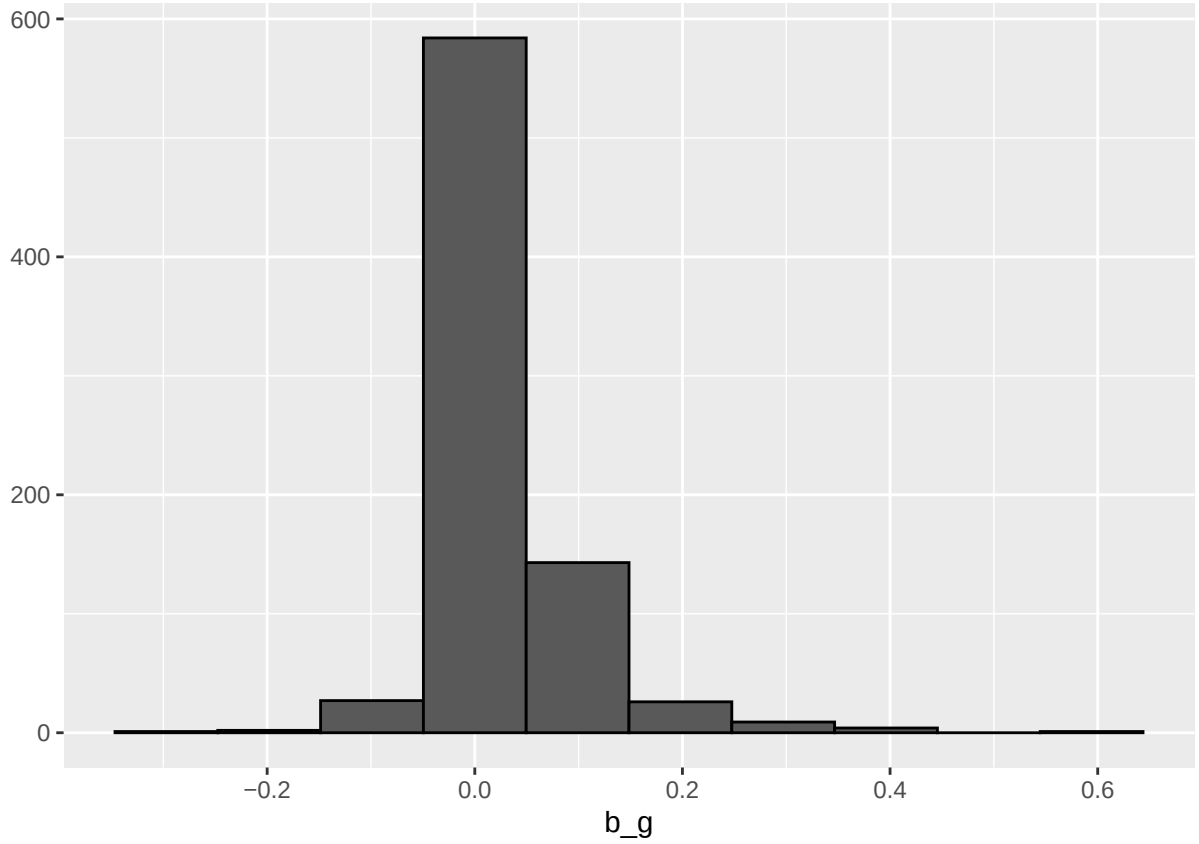
The year of release parameter has made a very small difference to the previous RMSE calculated at 0.865.

2.6 Genre effects model

The data contains a variable for genre, or a combination of different genres, that we viewed in Section 2.1 in the plot "No. of ratings per Genre". We can see that similar to the other variables some genres receive more ratings which could potentially apply unwanted bias to the predicted ratings.

We calculate the bias and then plot it to see the distribution around the mean. (again, this is incremental to the parameters already applied in the sub-sections above)

```
genre_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_y))

qplot(b_g, data = genre_avgs, bins = 10, color = I("black"))
```




```
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='year') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
  pull(pred)
genre_model_rmse <- RMSE(predicted_ratings, validation$rating)
```

```
rmse_results <- rmse_results %>% add_row(tibble_row(method = "Genre model", RMSE = genre_model_rmse))
genre_model_rmse
```

```
## [1] 0.8647135
```

The RMSE has again improved down to 0.8647

2.7 Penalised least squares / tuning parameters

In this section we would like to improve on the least squares estimates by applying a penalty to the parameters we have already included based on the weighting of the number of ratings. We will use cross-validation to choose the tuning parameter that delivers the optimal error loss.

We start with a range of tuning parameters, lambda, to test from 0 to 10 at increments of 0.5.

```
lambdas <- seq(0, 10, 0.5)
```

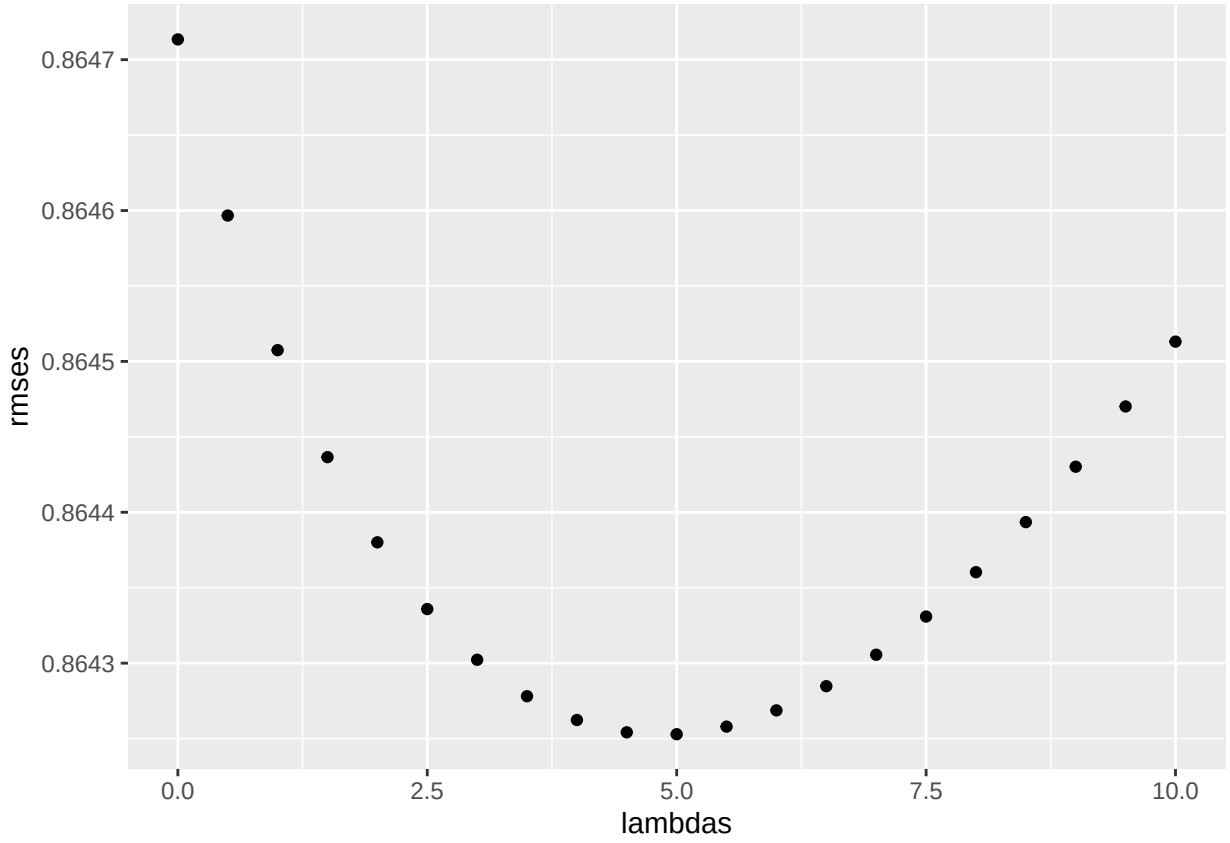
Then we need to set up a function that runs each of the test parameters in turn on our incremental model we defined in Sections 2.3 to 2.6. The tuning parameter, or lambda, is now also included in the determination of each bias which is then added to the prediction.

```
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))

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

  b_g <- edx %>%
    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_i - b_u - b_y - mu)/(n()+l))

  predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year") %>%
```

```
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    .$pred

  return(RMSE(validation$rating,predicted_ratings))
})
```

We can plot the various lambdas to see which one returns the lowest error loss.

```
qplot(lambdas, rmses)
```



In our case 5 is the optimum tuning parameter to use which returns a slightly improved RMSE of 0.8643

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

```
## [1] 5
```

```
rmse_results <- rmse_results %>% add_row(tibble_row(method = "No. ratings opt. weighted model", RMSE = r
min(rmses)
```

```
## [1] 0.8642529
```

3 Results

By applying various techniques starting with a simple mean and then incrementally increasing the complexity of the estimation with the addition of bias we have improved the error loss considerably.

The table below lists the various RMSEs we have calculated in increasing order of complexity until we get the final result of 0.8642529 from the "optimised weighted model" which is as follows:

$$Y_{ui} = \mu + b_{i,n,\lambda} + b_{u,n,\lambda} + b_{y,n,\lambda} + b_{g,n,\lambda} + \epsilon_{u,i,y,g}$$

```
rmse_results
```

```
## # A tibble: 6 x 2
##   method                            RMSE
##   <chr>                            <dbl>
## 1 Just the average                 1.06
## 2 Movie effect model               0.944
## 3 User effect model                0.865
## 4 Release year model               0.865
## 5 Genre model                      0.865
## 6 No. ratings opt. weighted model 0.864
```

4 Conclusion

The assignment was to produce a machine learning model that returned an error loss on the prediction using the validation set of lower than 0.86490. We have managed to reduce the residual mean squared error to 0.8642529 through incremental training of 90% of the dataset. This was achieved through the application of bias parameters as well as tuning parameters to the mean.

Further improvements on the model could be achieved by applying additional methods including various types of cross-validation.