MovieLens Rating Prediction

Vivek Krishnan

10 March 2020

Introduction

The MovieLens dataset was generated by the MovieLens team (https://grouplens.org/datasets/movielens/10m/) containing over 20 million ratings for over 27,000 movies by more than 138,000 users.

This report is based on a smaller dataset from MovieLens that contains 10 million ratings spanning over 10,000 movies and around 70,000 users.

The objective of this piece of work is to analyse the pattern of how users rate movies, what features influence the movie ratings, and ultimately make predictions on the ratings that specific users give for specific movies.

The analysis will be structured as follows:

1. Method and Analysis for the study
2. Data Wrangling, data exploration and analysis of the rating patterns
3. Identifying influencers for building a prediction model
4. Building a predictive model iteratively based on these influencers using a training set and evaluating against a test set
5. Applying regularisation
6. Building the final model and evaluation against the validation set
7. Results
8. Conclusion and future work that can be done

Please note that the code embedded in this report may take several minutes to run and ideally requires 16 GB RAM or more. While the code can also be run on an 8 GB RAM machine, please note that the regularisation iterations in the code will run for a couple of hours.

Method and Analysis

The total dataset of 10 million ratings was split into a dataset called *edx* consisting of *9 million* records and another dataset called *validation* consisting of *1 million* records.

All of the data exploration, analysis, visualisation and building and testing of prediction models have been done using only the *9 million* record *edx* dataset.

The *validation* set was used only for final evaluation of the predictions made using the best model.

The *edx* dataset itself was split into a training set called *train_set* and a test set called *test_set* with a **training:test split ratio of 80% : 20% (7.2 million training : 1.8 million test)**.

This *training_set* consisting of around *7.2 million* records was used for building prediction models. Each model was tested against the *1.2 million* record *test_set* and iteratively tuned.

The best model based on the evaluation against the *test_set* was used as the final model.

This final model was run against the *validation* set at the end to generate the final result.

The evaluation metric used for the prediction models was **RMSE** or the **root mean squared error** calculated as the square root of the sum of the squared difference between the predicted rating and actual rating for every movie-user combination in the evaluation dataset.

If there are totally N movies and M users, and if the actual rating for the movie i by user u is $R_{i,u}$ and the predicted rating for the same movie is $P_{i,u}$, then the $RMSE$ is calculated as follows:

$$RMSE = \sqrt{\frac{1}{MN}\Sigma_{i=1}^{N}\Sigma_{u=1}^{M}(P_{iu} - R_{iu})^2}$$

The model chosen was the one that minimises the RMSE over the *test_set* consisting of 1.8 million records. **L2 regularisation** was also applied on the model to penalise overfitting on the training set. The optimised L2 regularisation parameter λ was determined by trying various values of λ and identifying the parameter value that gave the least RMSE on the test set.

Once the best model was identified and the optimised L2 regularisation parameter λ determined for that model, the model was then trained on the entire edx dataset of 9 million records with regularisation.

This was the final model that this study came up with. The performance of this final model was evaluated by running the model to generate predictions against the validation set of 1 million records and calculating the final RMSE.

Data Wrangling, Data Exploration and Analysis of the Rating Patterns

Before attempting to build any prediction model, it is imperative to first explore the data and understand the underlying patterns. This section will focus on the data exploration that was done on the *edx* dataset and analysis of the patterns in the data.

Let's first load the libraries that we will need.

```
library(tidyverse)
library(dplyr)
library(anytime)
library(lubridate)
library(ModelMetrics)
library(ggplot2)
library(caret)
library(data.table)
```

Let's look at the structure of the *edx* data set.

```
head(edx)
```

```
##   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)
## 7      1     355      5 838984474       Flintstones, The (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
## 7       Children|Comedy|Fantasy
```

```
num_rows<-NROW(edx)
num_rows
```

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

- We can see that each record in *edx* corresponds to a rating by a user for a movie
- There are totally 9000055 records or ratings in the *edx* set
- We can also see that the timestamp or the date of rating is in epoch format
- The title includes the year of release of the movie within parenthesis
- The various genres that the movie can be classified into is contained within the genres column with each genre separated by a | character
- The rating column contains the actual rating by the user referred to in the userId column for the movie referred to in the movieId column Let's see what is the range of user ratings:

```
unique(edx$rating)
```

```
##  [1] 5.0 3.0 2.0 4.0 4.5 3.5 1.0 1.5 2.5 0.5
```

As can be seen, the rating value ranges from 0.5 to 5.

Let's get the year of movie release into a separate column. We can use regex to identify and extract the *year* from within the title.

```
#Extract 2 groups - the title in the 1st and the year in the 2nd.
temp <- str_match(edx$title, '(.*)[(](\\d{4})[)]$')[,2:3]
edx$title <- str_trim(temp[,1])#trim the trailing whitespace in the extracted title
edx$year <- as.integer(temp[,2])
rm(temp) #remove variable to free up memory
```

Let's look at the data set now and check.

```
head(edx)
```

```
##   userId movieId rating timestamp                 title
## 1      1     122      5 838985046             Boomerang
## 2      1     185      5 838983525              Net, The
## 4      1     292      5 838983421              Outbreak
## 5      1     316      5 838983392              Stargate
## 6      1     329      5 838983392 Star Trek: Generations
## 7      1     355      5 838984474      Flintstones, The
##                         genres year
## 1               Comedy|Romance 1992
## 2         Action|Crime|Thriller 1995
## 4  Action|Drama|Sci-Fi|Thriller 1995
## 5       Action|Adventure|Sci-Fi 1994
## 6 Action|Adventure|Drama|Sci-Fi 1994
## 7       Children|Comedy|Fantasy 1994
```

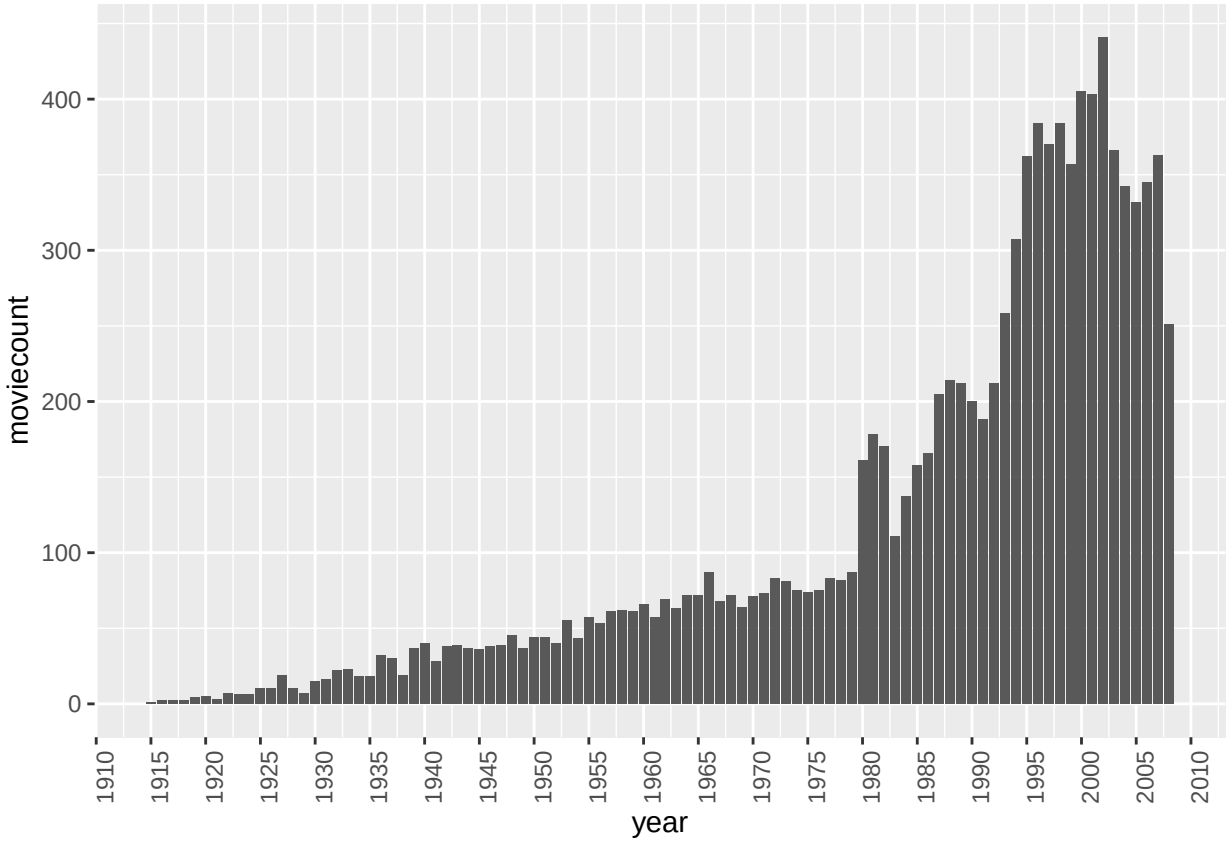
We can see the year has been extracted out into a separate column called *year* from the *title* column.

Data Exploration and Visualisation

Let's explore the trends in the data now.

How about **distribution of movies over release year**?

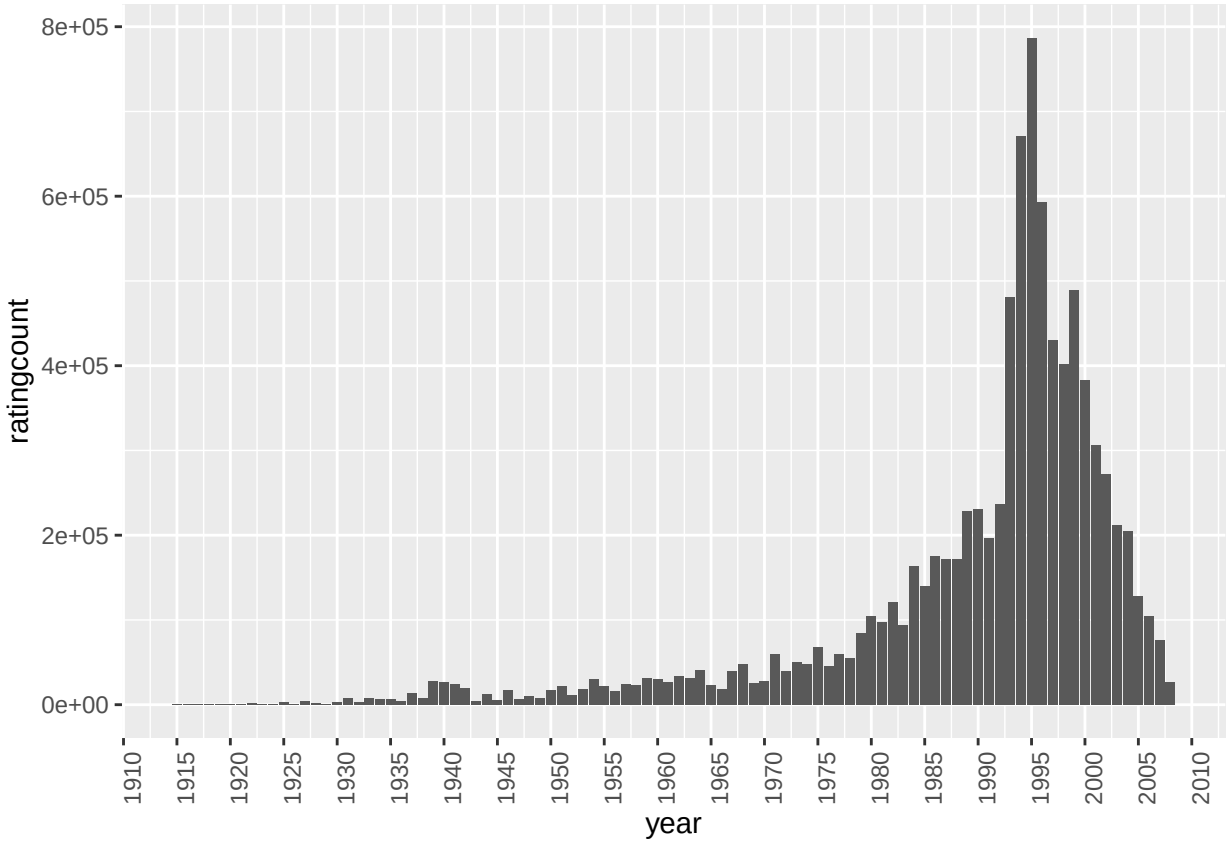
```
edx %>% group_by(year) %>% summarise(moviecount=n_distinct(movieId)) %>%
  ggplot(aes(x=year, y=moviecount)) + geom_bar(stat='identity') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_x_continuous(breaks=seq(1900, 2020, 5))
```



We can see a clear increase in the number of movies released per year with a huge spike around 1980 and again in 1995.

How about the **number of ratings posted by year of release**?

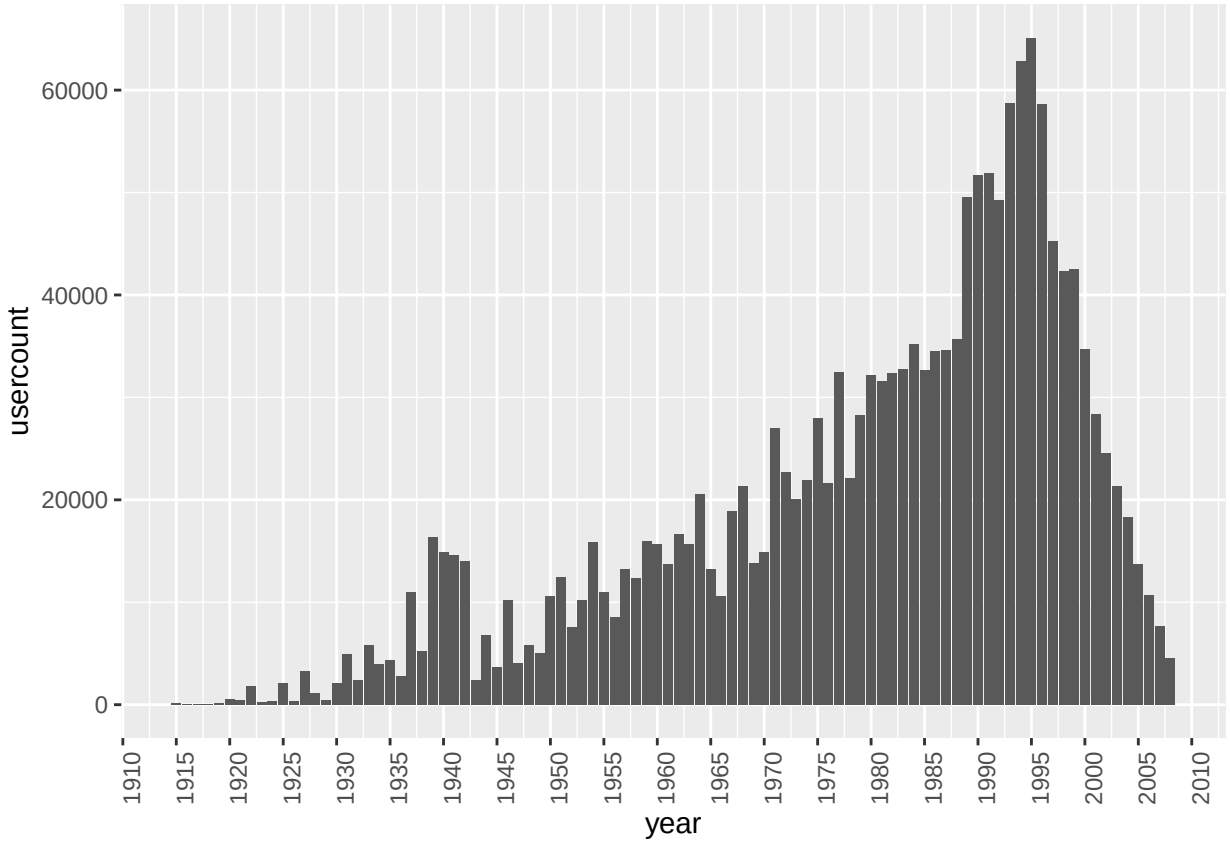
```
edx %>% group_by(year) %>% summarise(ratingcount=n()) %>%
  ggplot(aes(x=year, y=ratingcount)) + geom_bar(stat='identity') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_x_continuous(breaks=seq(1900, 2020, 5))
```



We can see sizeable increase in user rating count for movies released around 1993 onwards.

How about the **number of users posting ratings by year of release**?

```
edx %>% group_by(year) %>% summarise(usercount=n_distinct(userId)) %>%
  ggplot(aes(x=year, y=usercount)) + geom_bar(stat='identity') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_x_continuous(breaks=seq(1900, 2020, 5))
```

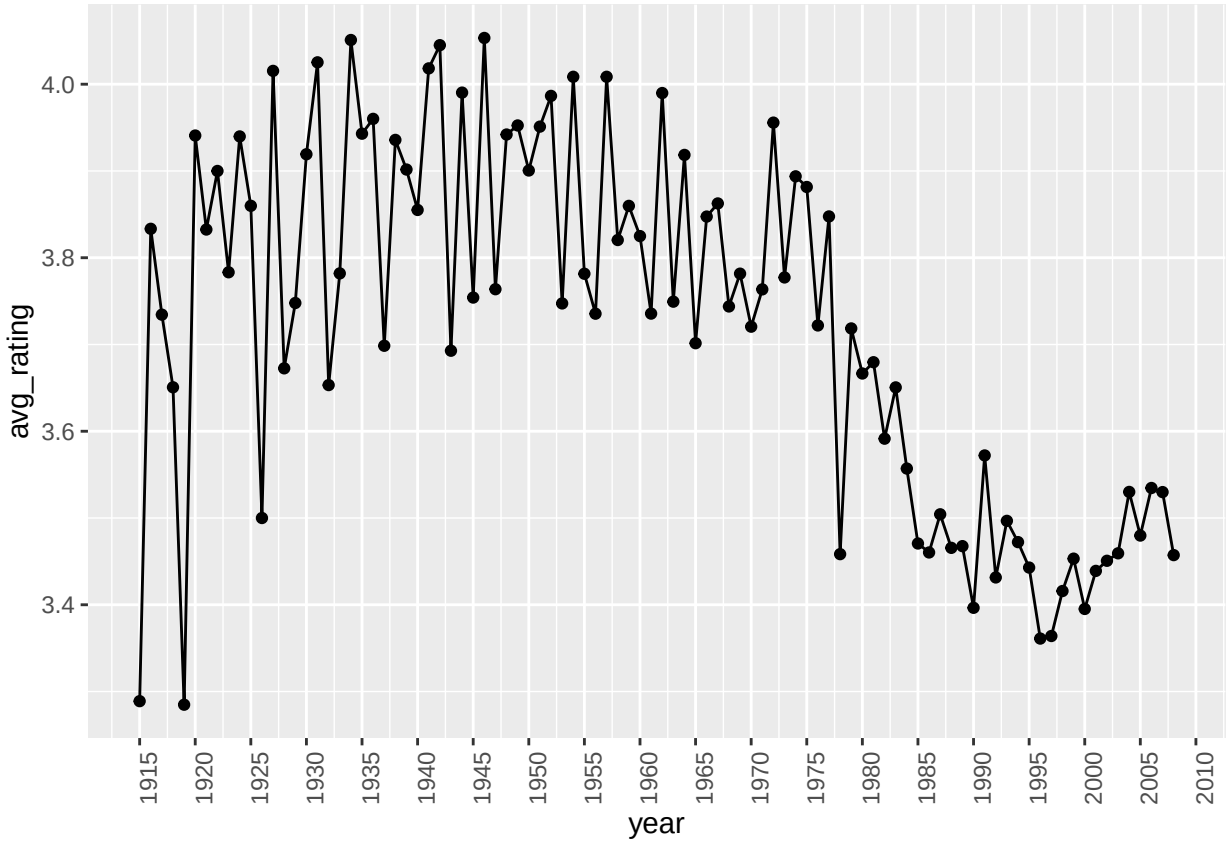


We can see a similar trend here with a spike for movies released around 1990 and again 1993.

Now let's look at the trends in the rating itself:

Average rating by year of release

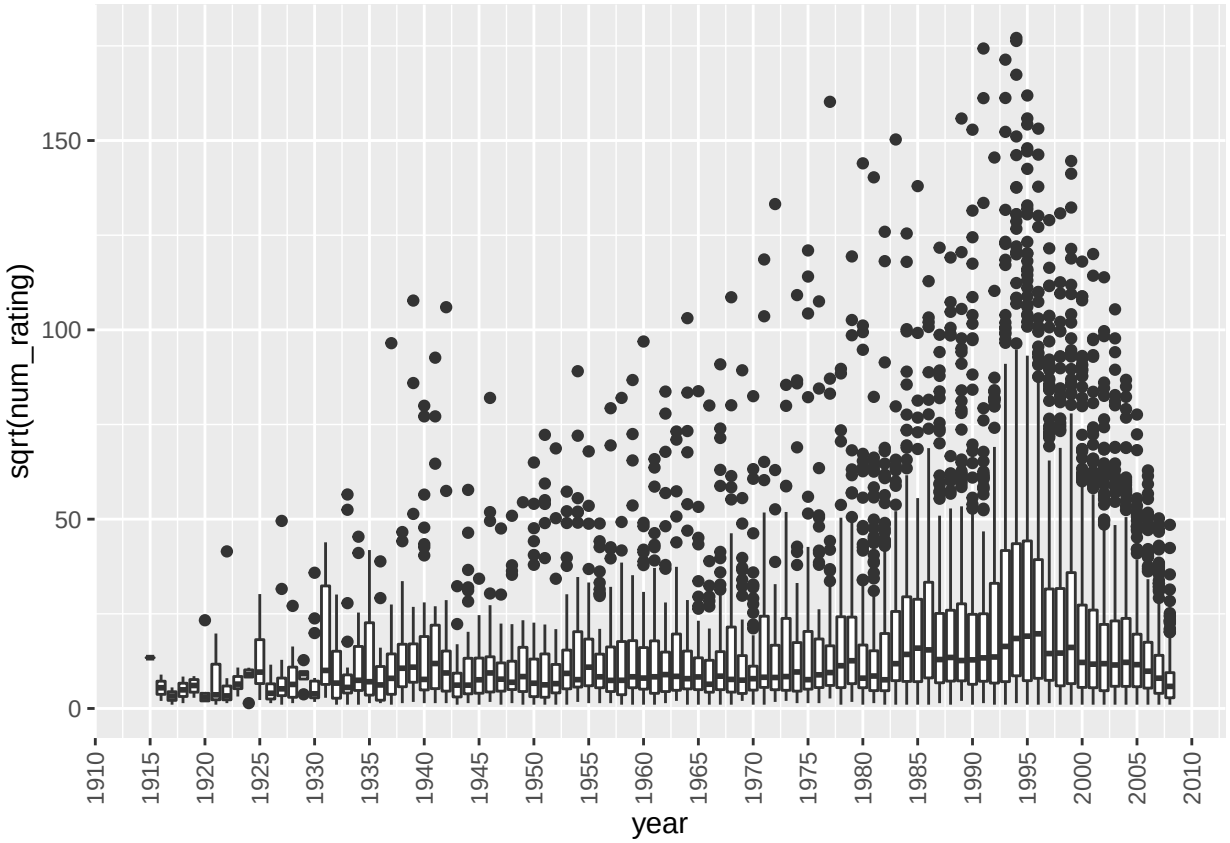
```
edx %>% group_by(year) %>% summarise(avg_rating=mean(rating)) %>%
  ggplot(aes(x=year, y=avg_rating)) + geom_point() + geom_line() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_x_continuous(breaks=seq(1900, 2020, 5))
```



The above graph shows a clear distinction between movies released pre-1990 and post.

Now, let's look at the **number of ratings for a movie plotted against the year it was released**.

```
edx %>% group_by(movieId) %>% mutate(num_rating=n()) %>% sample_n(1) %>% ungroup() %>%
  select(movieId, title, year, num_rating) %>%
  ggplot(aes(x=year, y=sqrt(num_rating), group=year)) + geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5)) +
  scale_x_continuous(breaks=seq(1900,2020,5))
```



From the above, we can see the highest median number of ratings for movies as well as the highest number of ratings for a single movie are for movies released in the years 1994/1995.

Let's have a look at the movies with the highest number of ratings:

```
edx %>% group_by(movieId) %>% mutate(num_rating=n()) %>% sample_n(1) %>% ungroup() %>%
  select(movieId, title, year, num_rating) %>% arrange(desc(num_rating))
```

```
## # A tibble: 10,677 x 4
##    movieId title                                                year num_rating
##      <dbl> <chr>                                               <int>      <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 War~ 1977      25672
## 10     150 Apollo 13                                            1995      24284
## # ... with 10,667 more rows
```

The above plots show that, on average, movies that came out after 1993 get more ratings. We also see that with newer movies, starting in 1995, the number of ratings decreases with year: the more recent a movie is, the less time users have had to rate it.

Let's look at movies that came out in 1993 or later, and look at the movies with the highest average number of ratings per year of review and the average rating of each of them.

```
edx %>% filter(year >= 1993 & year <= 2018) %>%
  mutate(review_year=year(anydate(timestamp))) %>%
  group_by(movieId, review_year) %>%
  mutate(avg_num_rating_per_year=n(), avg_rating=mean(rating)) %>%
  sample_n(1) %>% ungroup() %>% top_n(25) %>%
  select(title, year, review_year, avg_num_rating_per_year, avg_rating) %>%
  arrange(desc(avg_num_rating_per_year)) %>% head()
```

```
## # A tibble: 6 x 5
##   title                       year review_year avg_num_rating_per_ye~ avg_rating
##   <chr>                      <int>       <int>                  <int>      <dbl>
## 1 Kids of Survival            1996        1996                      6          5
## 2 Marvin's Room               1996        1996                      6          5
## 3 I Shot a Man in Vegas       1995        1996                      5          5
## 4 Mother and Son (Mat i syn)  1997        2006                      2          5
## 5 Savior                      1998        2008                      2          5
## 6 Edge of Seventeen           1998        2005                      2          5
```

We get a strange result! These are unheard of movies, yet they are the ones with the top average rating per review year=5. However, we can also see that these movies have extremely few user ratings. This is intuitive as movies with a very low rating count usually have high ratings.

One possible reason for this may be because such niche/unheard of movies are watched and reviewed by people close to the film's makers who naturally give it a high rating.

However, without the actual data of who submitted the ratings, we cannot confirm that supposition.

So, let's instead order by number of ratings per year, instead of average rating:

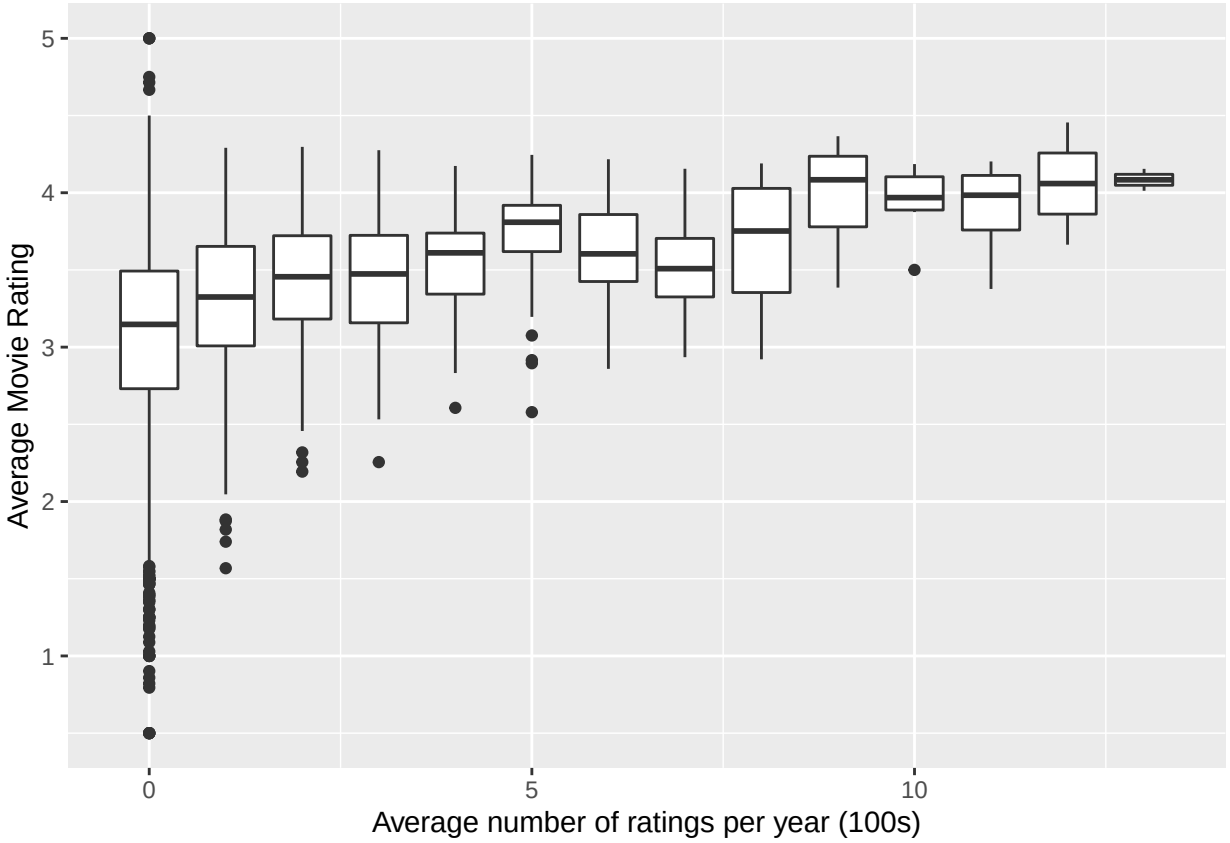
```
edx %>% filter(year >= 1993 & year <= 2018) %>%
  mutate(review_year=year(anydate(timestamp))) %>%
  group_by(movieId, review_year) %>%
  mutate(avg_num_rating_per_year=n(), avg_rating=mean(rating)) %>%
  sample_n(1) %>% ungroup() %>% top_n(25, wt=avg_num_rating_per_year) %>%
  select(title, year, review_year, avg_num_rating_per_year, avg_rating) %>%
  arrange(desc(avg_num_rating_per_year)) %>% head()
```

```
## # A tibble: 6 x 5
##   title           year review_year avg_num_rating_per_year avg_rating
##   <chr>          <int>       <int>                  <int>      <dbl>
## 1 Apollo 13       1995        1996                  11391       3.99
## 2 Pulp Fiction    1994        1996                  10922       4.01
## 3 Fugitive, The   1993        1996                  10898       4.12
## 4 True Lies       1994        1996                  10835       3.57
## 5 Forrest Gump    1994        1996                   9985       4.12
## 6 Batman Forever  1995        1996                   9906       3.13
```

This is much more along expected lines. We can also see that the most frequently rated movies tend to have above average ratings. This is not surprising: more people watch popular movies.

Let's confirm this by stratifying the post-1993 movies by ratings per review year and compute their average ratings.

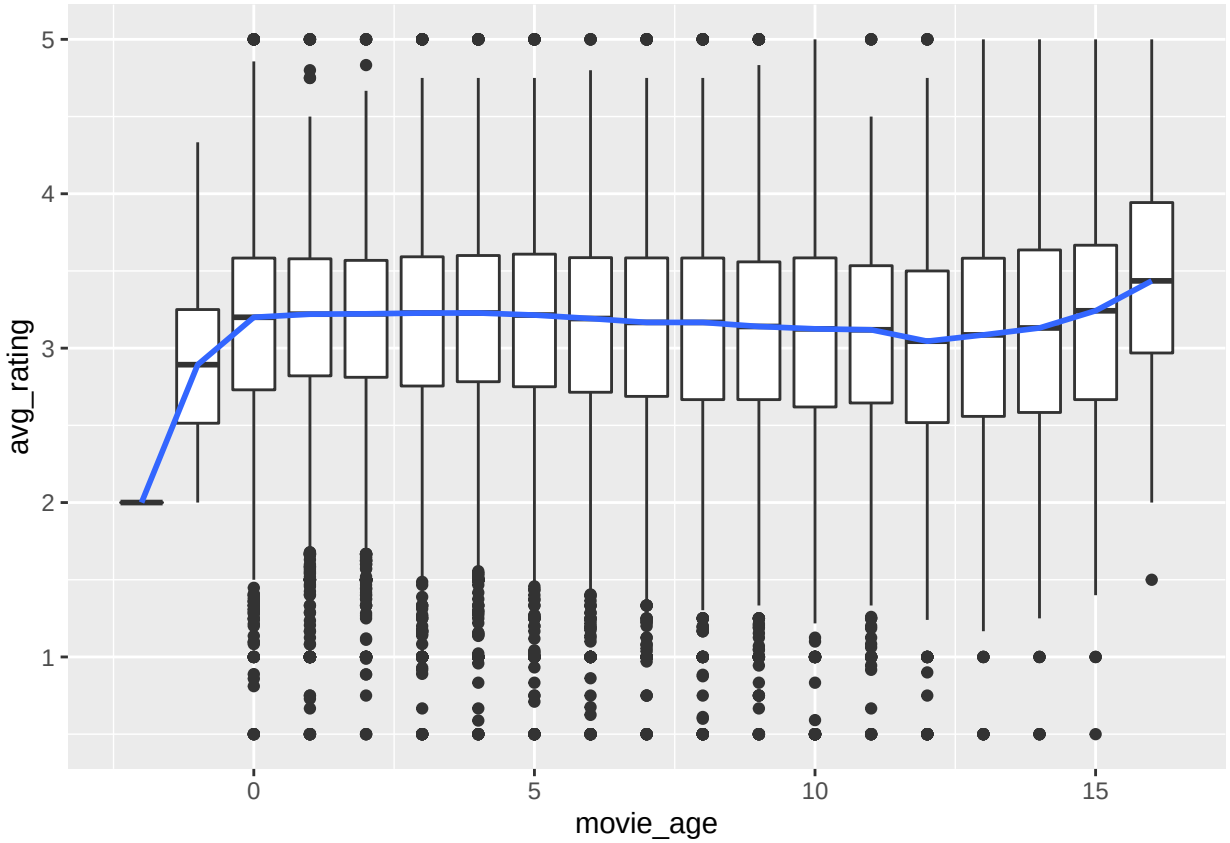
```
edx %>% filter(year >= 1993 & year <= 2018) %>%
  mutate(year_rating=year(anydate(timestamp))) %>%
  group_by(movieId) %>%
  mutate(num_rating=n(), num_years=n_distinct(year_rating),
         avg_ratingcount_per_year=round((num_rating/(2018-year))/100), avg_rating=mean(rating)) %>%
  sample_n(1) %>% ungroup() %>% arrange(desc(avg_ratingcount_per_year)) %>%
  ggplot(aes(avg_ratingcount_per_year, avg_rating, group=avg_ratingcount_per_year)) +
  geom_boxplot() +
  xlab('Average number of ratings per year (100s)') + ylab('Average Movie Rating')
```



From the above boxplot, we can see that the more often a movie is rated, the higher its average rating.

Let's plot the average rating per review year against the age of the movie on the date of review:

```
edx %>% filter(year >= 1993 & year <= 2018) %>%
  mutate(review_year=year(anydate(timestamp))) %>%
  group_by(movieId, review_year) %>%
  mutate(avg_rating=mean(rating), movie_age=review_year-year) %>% sample_n(1) %>%
  ungroup() %>% ggplot(aes(movie_age, avg_rating, group=movie_age)) +
  geom_boxplot() +
  stat_summary(fun.y=median, geom="smooth", aes(group=1), lwd=1)
```



This shows an interesting trend. If a user is reviewing a movie 12 years or more after its release, the user tends to give a higher rating. We also see that movies that are reviewed on the same year of release get a lower score than in later years.

We also notice potential **inaccuracies in the data**! In some cases, the year of movie release is **later** than the year of the user rating!

These are clearly cases of bad data, some examples of which can be seen below:

```
edx %>% mutate(review_year=year(anydate(timestamp))) %>%
  filter(year >= 1993 & year <= 2018 & review_year<year) %>%
  select(-genres) %>% head()
```

```
##   userId movieId rating timestamp            title year review_year
## 1    785     981      3 844464462 Dangerous Ground 1997        1996
## 2   1468     879      2 841308568       Relic, The 1997        1996
## 3   1583     981      1 842861387 Dangerous Ground 1997        1996
## 4   1766     870      5 839441876     Gone Fishin' 1997        1996
## 5   1766     879      5 840812687       Relic, The 1997        1996
## 6   1766     981      3 841947226 Dangerous Ground 1997        1996
```

Let's see how many such cases there are:

```
nCount <- edx %>% mutate(review_year=year(anydate(timestamp))) %>%
  filter(year >= 1993 & year <= 2018 & review_year<year) %>%
  NROW()
nUniqueMovieCount <- length(unique(edx %>%
```

```
                                    mutate(review_year=year(anydate(timestamp))) %>%
                                    filter(year >= 1993 & year <= 2018 & review_year<year) %>%
                                    pull(movieId)))
print(nCount)
```

```
## [1] 175
```

```
print(nUniqueMovieCount)
```

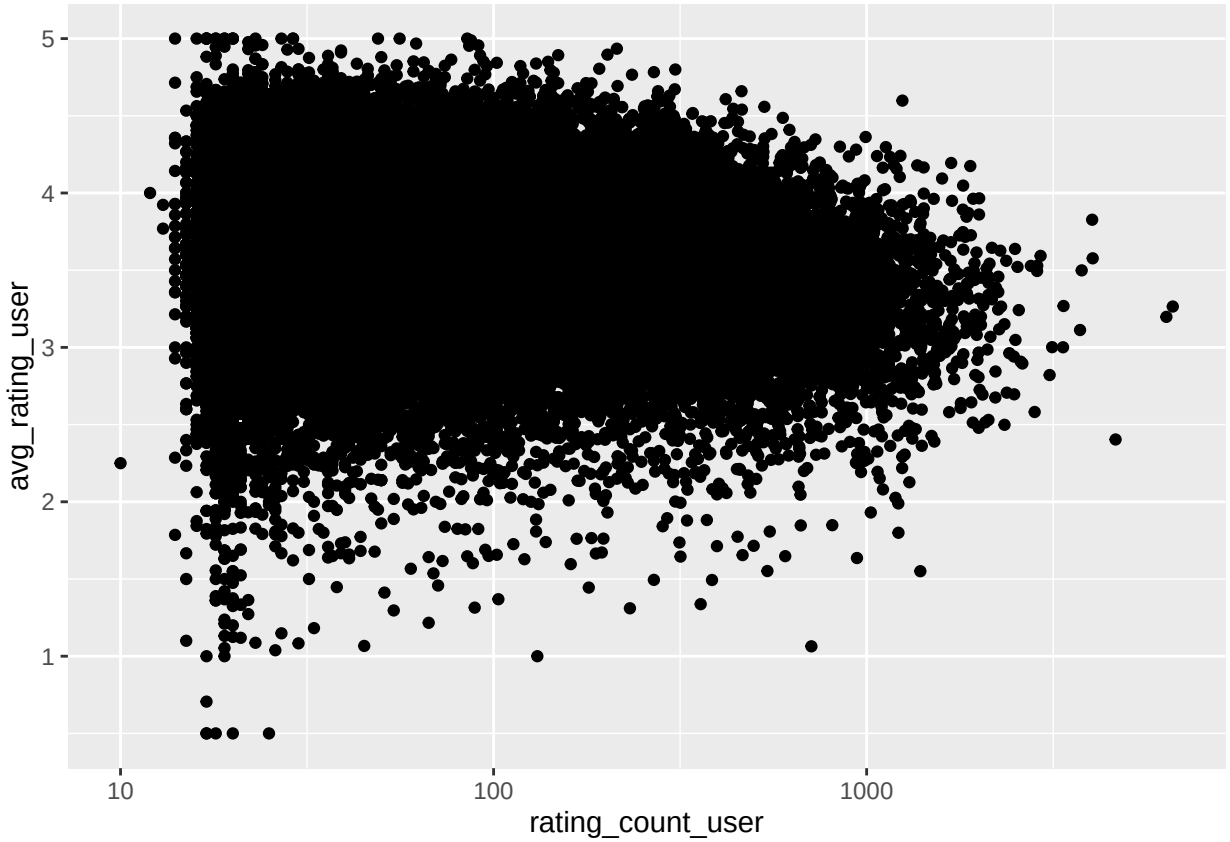
```
## [1] 23
```

There are 23 movies with a total of 175 ratings with this problem where the timestamp for user rating is before the movie was actually released.

Since this is a very small fraction, we can ignore this for the purpose of this study.

Let's now look at the trend of **average rating per user** along with the **number of ratings** they have given:

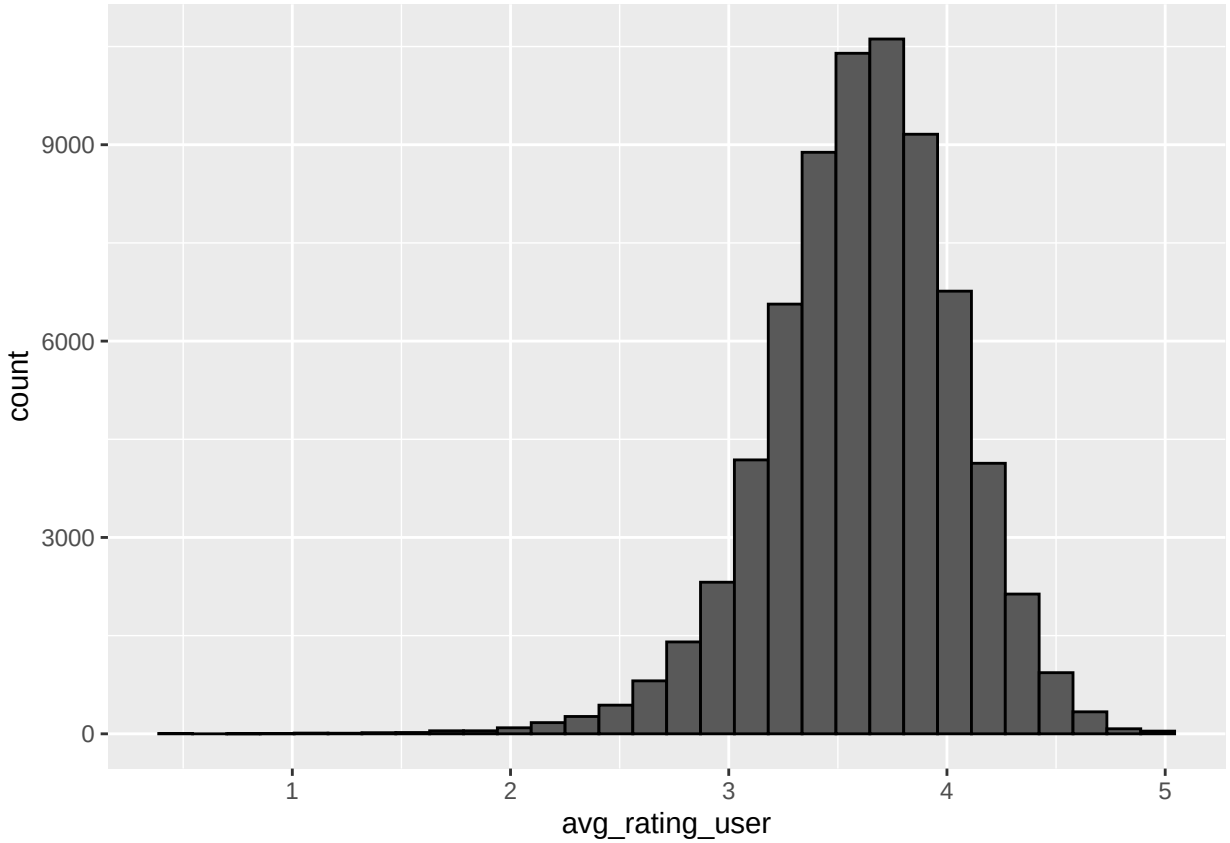
```
edx %>% group_by(userId) %>%
  summarize(avg_rating_user=mean(rating), rating_count_user=n()) %>%
  ggplot(aes(x=rating_count_user, y=avg_rating_user)) +
  geom_point() + scale_x_log10()
```



From the above, we can see that users who post fewer reviews are more likely to give extreme ratings on average compared to users who are more frequent.

Let's look at the histogram of average rating of users who have posted more than 100 ratings:

```
edx %>% group_by(userId) %>%
  summarize(avg_rating_user = mean(rating)) %>% filter(n()>=100) %>%
  ggplot(aes(avg_rating_user)) + geom_histogram(bins = 30, color = "black")
```



We can see there are some users who consistently give low ratings, while some give high ratings.

Let's now look at the **trends by genre**:

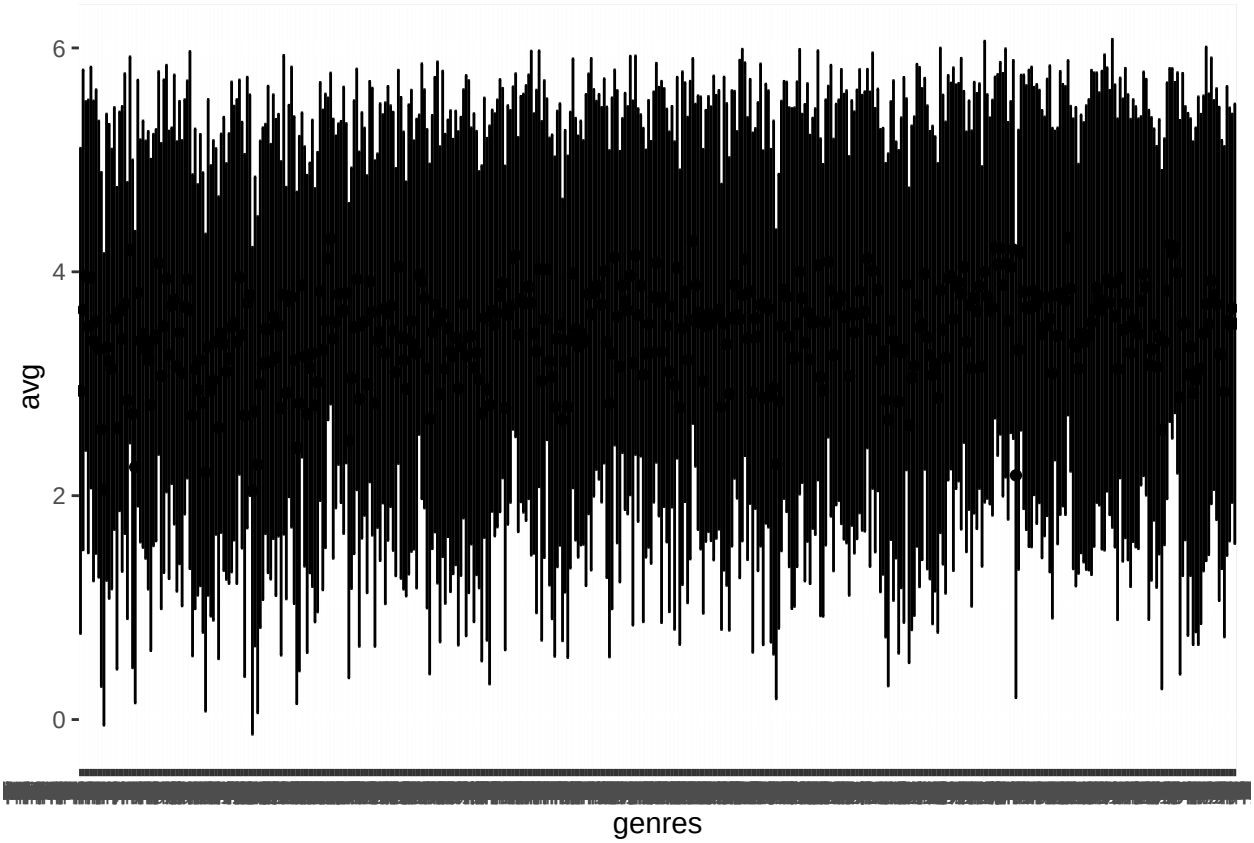
```
nGenreCount <- NROW(unique(edx$genres))
nGenreCount
```

```
## [1] 797
```

There are 797 different genre categories. Let's first explore genre trends keeping the *genres* column as it currently is.

Trend of **average rating by genre** for genres that have at least 1000 user ratings:

```
edx %>% group_by(genres) %>%
  summarize(count=n()) %>% filter(count>1000) %>%
  inner_join(edx, by='genres') %>%
  group_by(genres) %>%
  summarize(avg=mean(rating), sdev=sd(rating)) %>%
  ggplot(aes(genres, avg)) + geom_point() +
  geom_errorbar(aes(ymin=avg-2*sdev, ymax=avg+2*sdev))
```



We can see there is considerable variation based on genre of the movie.

However, this visualisation is practically unreadable as there are 797 genre combinations. As we saw before, the MovieLens dataset contains all the genres that a movie belongs to in the same field with a | separator.

This leads to the 797 genre combinations across the entire dataset which does not make it useful to analyse and derive insights from the genre data.

Let's instead identify the individual genres and flag if a movie belongs to that genre or not (instead of these genre combinations that is present in the *genres* column).

We can extract the individual genres for each movie by separating using the | character delimiter.

```
genres <- data.frame(unique(edx$genres), stringsAsFactors = FALSE)
names(genres) <- 'genregroup'
genres <- unique(genres %>%
                   separate(genregroup, into=paste('genre', 1:9, sep=''),
                            fill="right", sep='[|]') %>%
                   gather(genre, value, paste('genre', 1:9, sep='')) %>%
                   filter(!is.na(value) & value!='(no genres listed)') %>%
                   pull(value))
nGenreCount<-length(genres)
genres
```

```
##  [1] "Comedy"      "Action"      "Children"    "Adventure"   "Animation"
##  [6] "Drama"       "Crime"       "Sci-Fi"      "Horror"      "Thriller"
## [11] "Film-Noir"   "Mystery"     "Western"     "Documentary" "Romance"
## [16] "Fantasy"     "Musical"     "War"         "IMAX"
```

We can now clearly see the 19 individual genres that a film could belong to. It makes a lot more sense to visualise the rating distribution based on these individual genres.

Let's extract these individual genres out for each movie and create a mapping dataframe that maps the movies to the individual genres it belongs to. Let's also calculate the number of distinct genres each movie belongs to and add that as a column in the mapping dataframe.

```
movie_genres <-
  edx %>% select(movieId, genres) %>%
  group_by(movieId, genres) %>% sample_n(1) %>% ungroup() %>%
  separate(genres, into=paste('genre', 1:9, sep=''), fill="right", sep='[|]') %>%
  gather(key, genre, paste('genre', 1:9, sep='')) %>%
  filter(!is.na(genre) & genre!='(no genres listed)') %>%
  select(movieId, genre) %>%
  group_by(movieId) %>% mutate(genre_count=n())
```

Let's look at the movieId 1 (Toy Story) and check how the mapping looks.

```
movie_genres %>% filter(movieId==1)
```

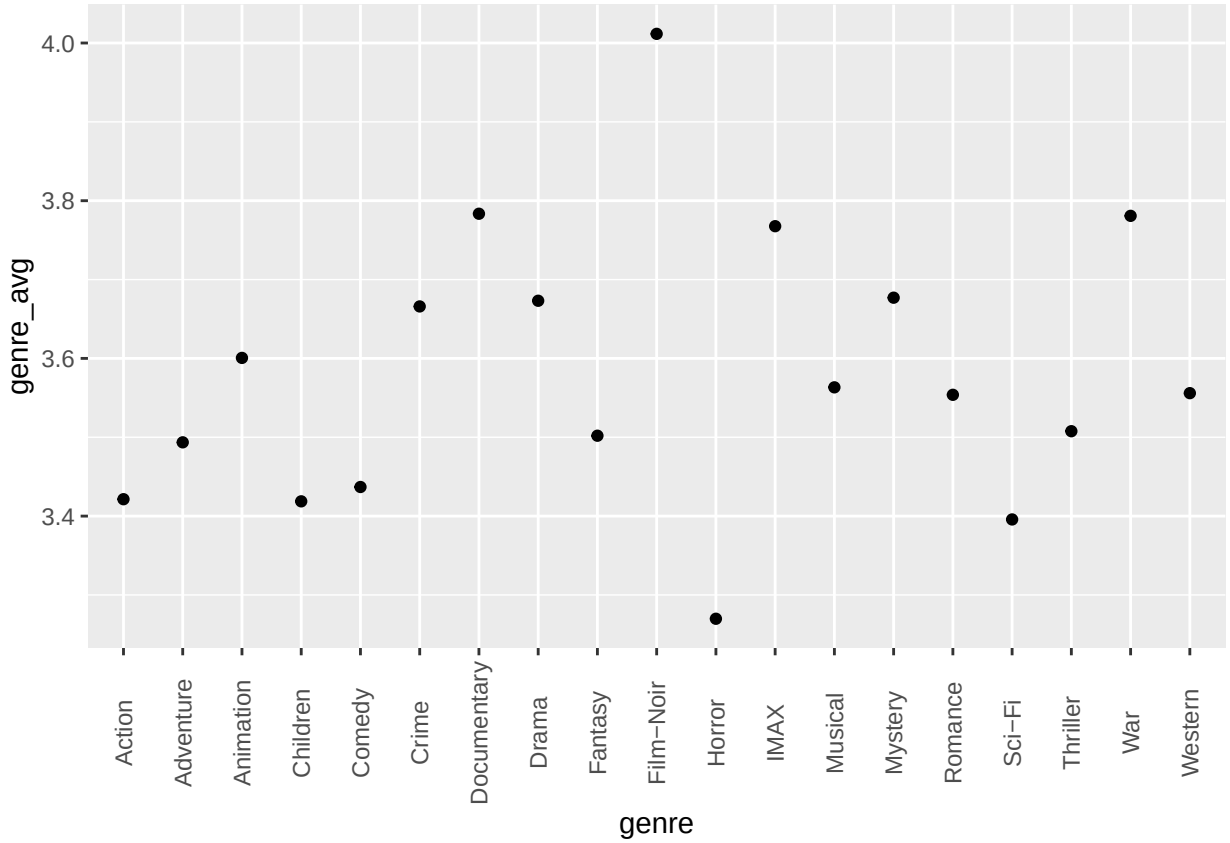
```
## # A tibble: 5 x 3
## # Groups:   movieId [1]
##   movieId genre     genre_count
##     <dbl> <chr>           <int>
## 1       1 Adventure           5
## 2       1 Animation           5
## 3       1 Children            5
## 4       1 Comedy              5
## 5       1 Fantasy             5
```

We don't need the *genres* column anymore as we have extracted the genre information out and created the movie-genre mapping in the $movie_genres$ dataframe.

```
edx <- edx %>% select(-genres)
```

Now, we can more clearly visualise the **average ratings by genre**:

```
edx %>% select(movieId, rating) %>%
  inner_join(movie_genres, by='movieId') %>%
  group_by(genre) %>%
  summarize(genre_avg = mean(rating)) %>%
  ggplot(aes(x=genre, y=genre_avg)) + geom_point() +
  theme(axis.text.x=element_text(angle=90,vjust=0.5))
```



We can now clearly see that some genres have higher median ratings than others. We can also see that Film-noir movies are highly rated.

We can see the average ratings by genre for each genre in decreasing order of average rating:

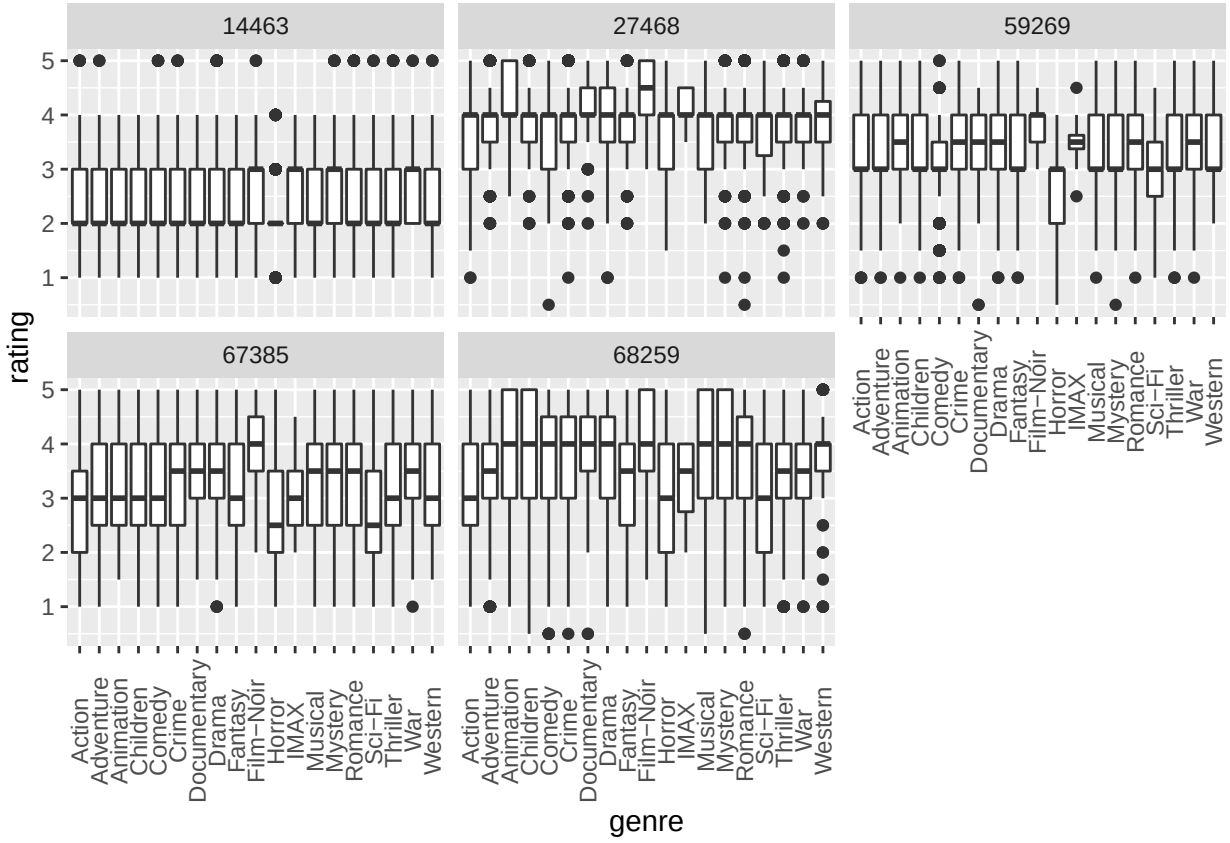
```
edx %>% select(movieId, rating) %>% inner_join(movie_genres, by='movieId') %>%
  group_by(genre) %>% summarize(avg_rating_by_genre=mean(rating)) %>% arrange(desc(avg_rating_by_genre))
```

```
## # A tibble: 19 x 2
##    genre       avg_rating_by_genre
##    <chr>                     <dbl>
##  1 Film-Noir                  4.01
##  2 Documentary                3.78
##  3 War                        3.78
##  4 IMAX                       3.77
##  5 Mystery                    3.68
##  6 Drama                      3.67
##  7 Crime                      3.67
##  8 Animation                  3.60
##  9 Musical                    3.56
## 10 Western                    3.56
## 11 Romance                    3.55
## 12 Thriller                   3.51
## 13 Fantasy                    3.50
## 14 Adventure                  3.49
## 15 Comedy                     3.44
```

```
## 16 Action                    3.42
## 17 Children                  3.42
## 18 Sci-Fi                    3.40
## 19 Horror                    3.27
```

Let's see how users rate specific genres. Let's pick the top 5 most active users and check their genre preferences.

```
edx %>% group_by(userId) %>% summarize(count=n()) %>% top_n(5) %>%
  inner_join(edx, by='userId') %>%
  select(userId, movieId, rating) %>%
  inner_join(movie_genres, by='movieId') %>%
  ggplot(aes(x=genre, y=rating, group=genre)) + geom_boxplot() +
  facet_wrap(~userId) +
  theme(axis.text.x=element_text(angle=90,vjust=0.5))
```

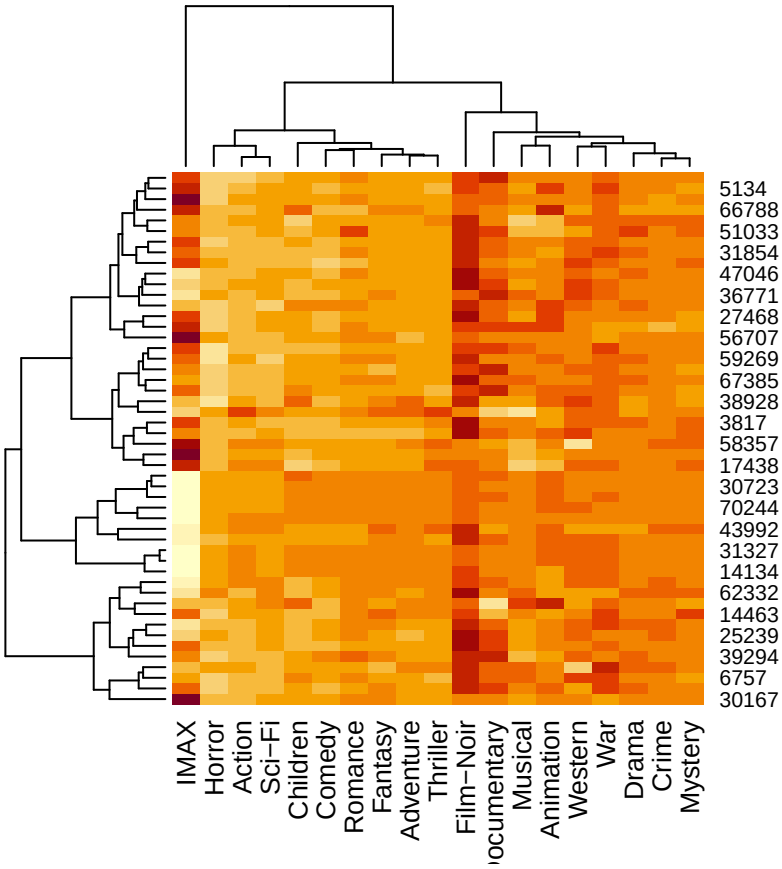


The above plot clearly shows that specific users rate specific genres high and others low. For example, of the 5 most active users, 2 users have a median rate for Drama at 4, while 2 others rate Drama at 3.5 and 1 rates Drama at 2.

Let's try a heatmap for the top 50 users.

```
m <- edx %>% group_by(userId) %>% summarize(count=n()) %>% top_n(50) %>%
  inner_join(edx, by='userId') %>%
  inner_join(movie_genres, by='movieId') %>%
  select(userId, genre, rating) %>%
```

```
  group_by(userId, genre) %>%
  summarize(avg_rating=10*mean(rating)) %>%
  ungroup() %>%
  spread(genre, avg_rating, fill=0) %>%
  as.matrix()
row.names(m) <- m[,1]
m <- m[,-1]
heatmap(m)
```



The above heatmap reiterates the point that users have genre preferences and this shows in their ratings. It also clearly shows that Film-Noir gets higher ratings than other genres.

We can also see a few user preferences coming out from the above heatmap. For example, there is significant user variation in the ratings for movies with the IMAX genre.

Identifying Influencers for Building a Prediction Model

Let's quickly recap the key observations from the data exploration done so far.

The important outcome of the data exploration should be to identify key influencers that impact the rating of a movie by a user.

In summary:

- We can see a clear increase in the number of movies released per year with a huge spike around 1980 and again in 1995

- We can see sizeable increase in number of user ratings for movies released around 1993 onwards
- We can see a similar trend with the number of unique users posting ratings with a spike for movies released around 1990 and again 1993
- The average ratings for movies shows a clear distinction between movies released pre-1990 and post
- The highest median number of ratings and the highest number of ratings for a single movie are for movies released in the years 1994/1995. On average, movies that came out after 1993 get more ratings. We also see that with newer movies, starting in 1995, the number of ratings decreases with year: the more recent a movie is, the less time users have had to rate it
- Movies with a very low number of ratings usually have extreme ratings
- The most frequently rated movies tend to have above average ratings
- If a user rates a movie 13 years or more after its release, the user gives a higher rating
- We can clearly see that some genres have higher median ratings than others. Film-noir movies are usually highly rated, while horror movies have the lowest average rating
- There are user dependencies as well - some users tend to give high ratings on average while others give poor ratings
- Also, users who are less active tend to provide extreme ratings on average compared to those who are more frequent

Potential key influencers for use in a prediction model

Based on all of the above, we can infer that a rating by a user u for a movie i depends on:

- the date of review by user u for movie i
- The release year of the movie i
- The time between the release of movie i and the date of review by user u
- The user $u's$ past rating pattern
- The movie $i's$ past ratings
- The number of ratings the movie i has got ie. how popular the movie is
- The genre of the movie i
- The user $u's$ past ratings for the genre of the movie i

Clubbing some of the above key influencers, below is the list of factors that a prediction model would need to include:

- Date factor (movie release year, rating date and the duration between the two)
- User factor (user's preferences, past ratings)
- Movie factor (movie's past ratings, number of ratings)
- Genre factor (the genre that the movie belongs to, the genre popularity)

The predictive models that were built were based on the above 4 factors.

Building a Predictive Model

Let's start and split the *edx* data set into training and test sets:

```
test_index <- createDataPartition(edx$rating, 1, 0.2, list=FALSE)
train_set <- edx[-test_index, ]
test_set <- edx[test_index, ]
```

We need a function for the evaluation metric. Recall that the evaluation metric chosen is RMSE calculated as follows: $RMSE = \sqrt{\frac{1}{MN}\Sigma_{i=1}^{N}\Sigma_{u=1}^{M}(P_{iu} - R_{iu})^2}$

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

Let's first try out a simple model that uses the mean rating alone to make predictions:

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

```
## [1] 3.512291
```

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

```
## [1] 1.05988
```

We can see this naive model has a pretty high RMSE. Let's store the RMSE value for each model that we iteratively build.

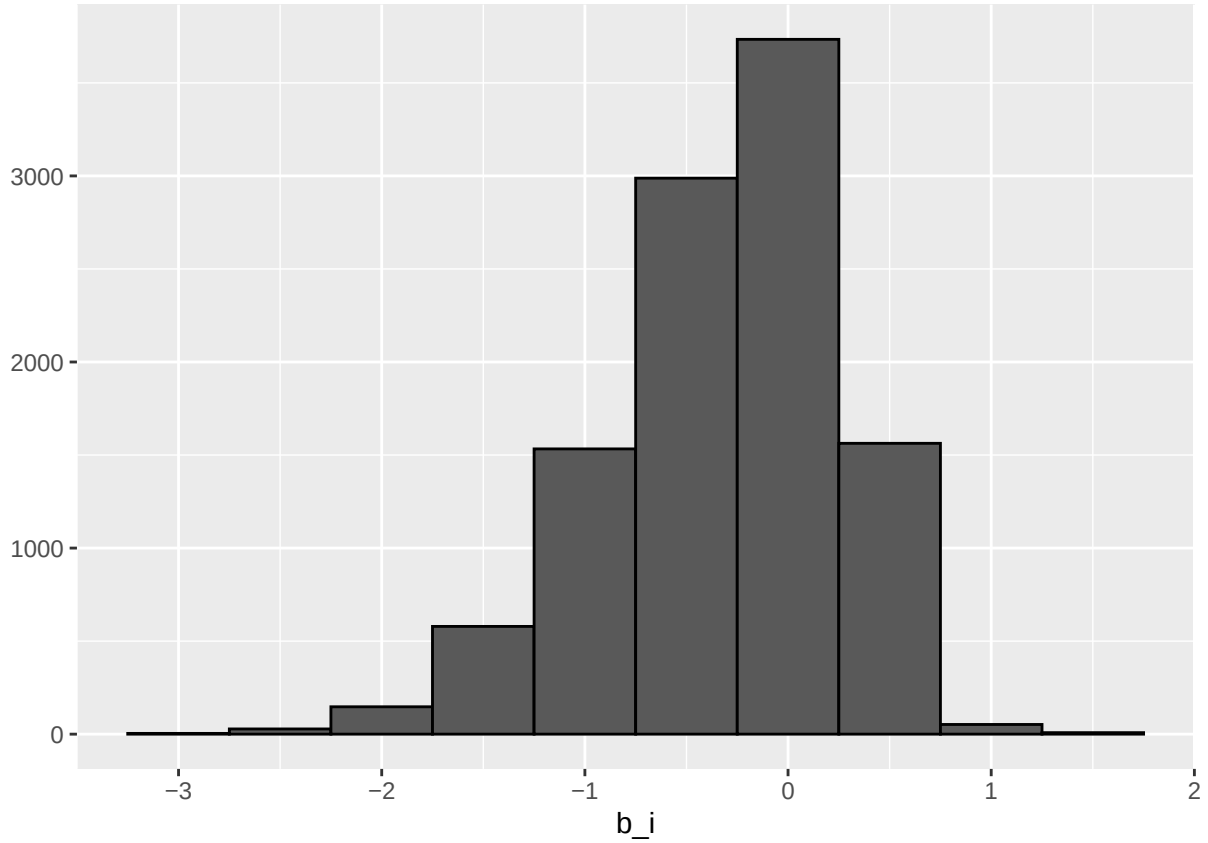
```
rmse_results <- data.frame(method = "Using only average", RMSE = naive_rmse)
```

Let's now try a model based on movie average rating where we take the movie factor or effect into consideration.

Some movies are usually rated higher than others. Let's look at the variation caused by this movie effect:

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

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



We can see from the above histogram that there is a wide variation caused by the movie effect. Adding this term will adjust for movie effects.

So let's use the movie factor in our next prediction model:

```
predicted_ratings_2 <- mu + (test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i)

#Set the predicted rating to the average for any movies in the test set
#that are not present in the training set
predicted_ratings_2[is.na(predicted_ratings_2)] <- mu
model_1_rmse <- RMSE(predicted_ratings_2, test_set$rating)
model_1_rmse
```

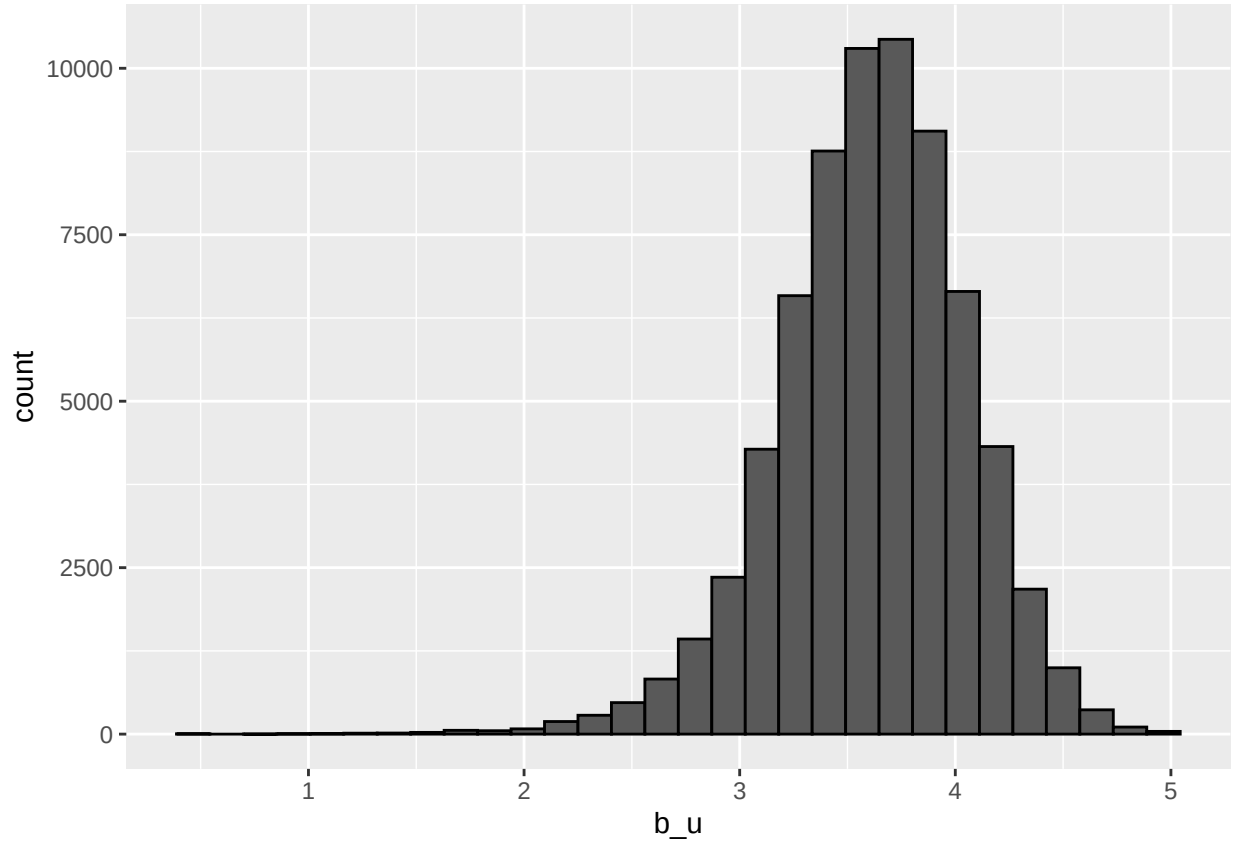
```
## [1] 0.9437469
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))
```

We can see this model gives a much better RMSE than the naive model.

Let's now model the user effect as well. How does the user effect affect ratings?

```r
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



We can see considerable variation here as well. Adding this b_u bias user factor term will improve our model.

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

predicted_ratings_3 <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

#Set the predicted rating to the mean for any users in the test set
#that are not present in the training set
predicted_ratings_3[is.na(predicted_ratings_3)] <- mu

model_2_rmse <- RMSE(predicted_ratings_3, test_set$rating)
```

```
model_2_rmse
```

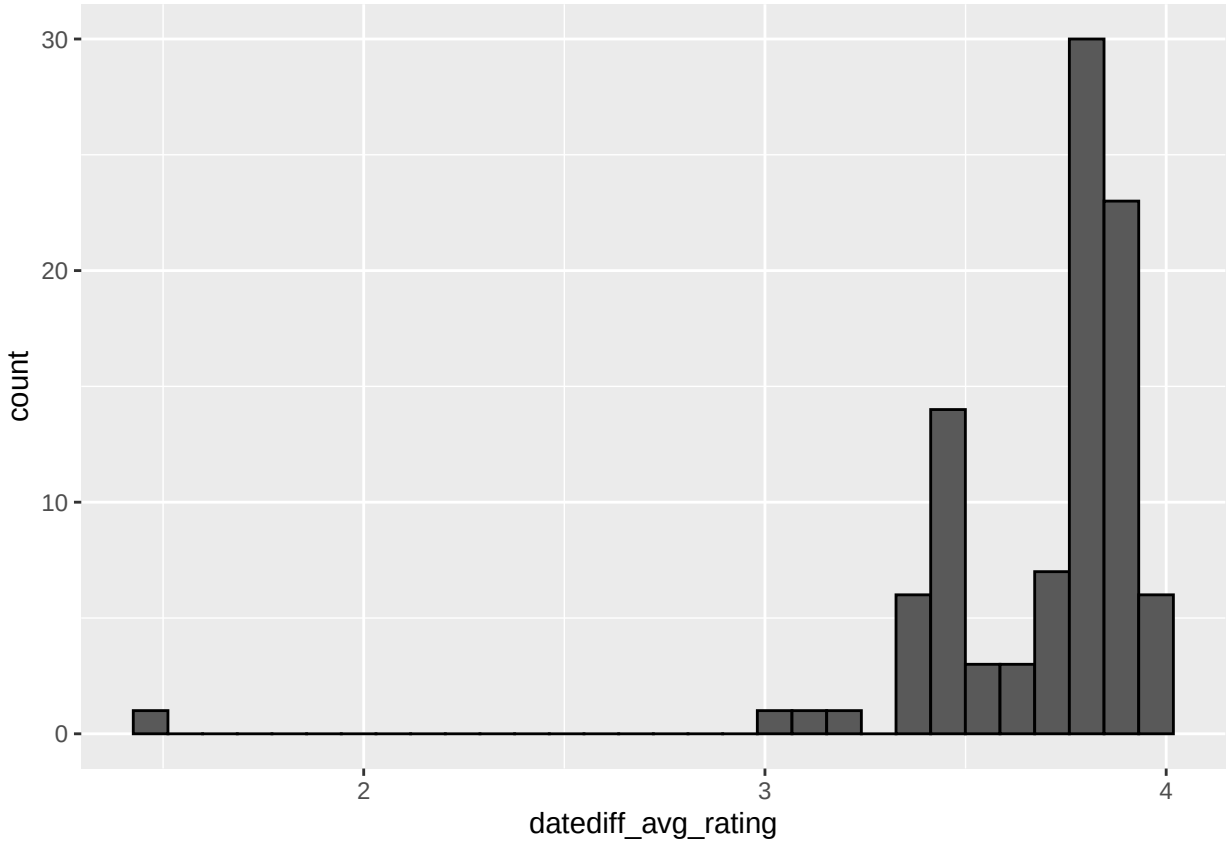
```
## [1] 0.8661863
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_2_rmse ))
```

We can see this is way better than the naive model and the model with just the movie factor.

Our next model will make use of the date factor. First, let's look at the variation caused by the date factor.

```
train_set %>% select(userId,rating,timestamp,year) %>%
  mutate(date_diff=round((year(anydate(timestamp))-year))) %>%
  group_by(date_diff) %>%
  summarize(datediff_avg_rating = mean(rating)) %>%
  ggplot(aes(datediff_avg_rating)) +
  geom_histogram(bins = 30, color = "black")
```



We can see considerable variation here as well when we plot the average rating grouped by the age of the review (where age is the number of years between the movie release year and the year of review.

Adding a b_d bias date factor term will improve our model.

```
date_avgs <- train_set %>% select(userId,movieId,rating,timestamp,year) %>%
  mutate(date_diff=round((year(anydate(timestamp))-year))) %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(date_diff) %>%
  summarize(b_d = mean(rating - mu - b_i - b_u))

predicted_ratings_date <- test_set %>% select(userId,movieId,timestamp,year) %>%
  mutate(date_diff=round(year(anydate(timestamp))-year)) %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(date_avgs, by='date_diff') %>%
  mutate(pred = mu + b_i + b_u +b_d) %>%
  .$pred

#Set the predicted rating to the mean for any records in the test set
#that are not present in the training set
predicted_ratings_date[is.na(predicted_ratings_date)] <- mu
#Cut-offs for predicted rating should be 0.5 and 5.
predicted_ratings_date[predicted_ratings_date<0] <- 0.5#any -ve predictions
predicted_ratings_date[predicted_ratings_date>5] <- 5#any predictions > 5

model_date_rmse <- RMSE(predicted_ratings_date, test_set$rating)
model_date_rmse
```

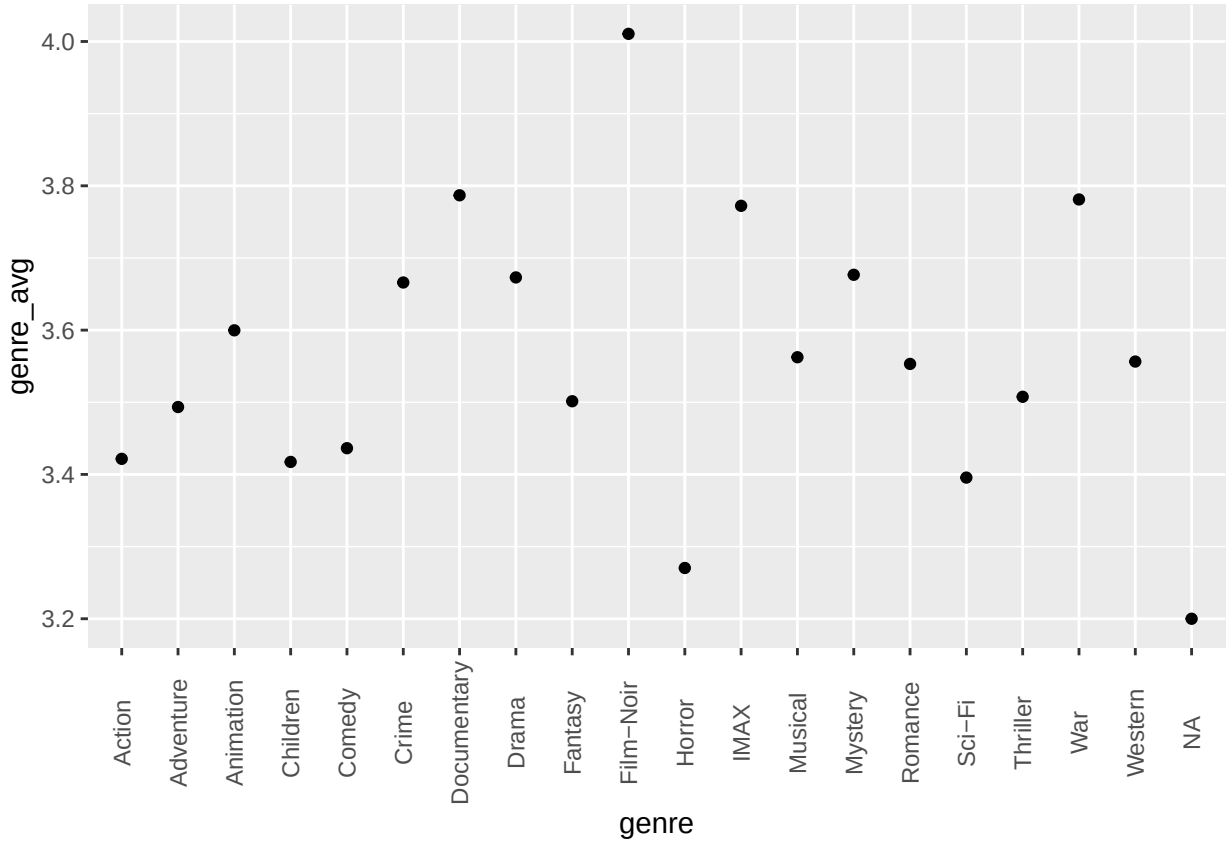
```
## [1] 0.8655267
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + Date Effects Model",
                                     RMSE = model_date_rmse ))
```

We can see the RMSE has further improved by adding the date factor.

Our next model will make use of the genre factor. First, let's look at the variation caused by the genre factor.

```
train_set %>% select(movieId, rating) %>% left_join(movie_genres, by='movieId') %>%
  group_by(genre) %>%
  summarize(genre_avg = mean(rating)) %>%
  ggplot(aes(x=genre, y=genre_avg)) + geom_point() +
  theme(axis.text.x=element_text(angle=90,vjust=0.5))
```



We can see there is considerable variation here as well with some genres having higher ratings on average than others.

Adding a genre bias term will improve our model.

```
genre_avgs <- train_set %>% select(-title) %>%
  left_join(movie_genres, by='movieId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(date_diff=round(year(anydate(timestamp))-year)) %>%
  left_join(date_avgs, by='date_diff') %>%
  mutate(b_g_ui = (rating - mu - b_i - b_u - b_d)/genre_count) %>%
  group_by(genre) %>% summarize(b_g=mean(b_g_ui))
```

Before we build the model including this genre factor, let's take a look at how the genre bias has been quantified, along with the number of ratings per genre:

```
train_set %>% select(movieId, rating) %>%
  inner_join(movie_genres, by='movieId') %>%
  group_by(genre) %>%
  summarize(count=n(), avg=mean(rating)) %>%
  arrange(desc(avg)) %>%
  inner_join(genre_avgs, by='genre')
```

```
## # A tibble: 19 x 4
##    genre         count   avg       b_g
```

```
##    <chr>          <int> <dbl>     <dbl>
##  1 Film-Noir      94834  4.01  0.00713
##  2 Documentary    74430  3.79  0.0577
##  3 War           409384  3.78  0.00123
##  4 IMAX            6479  3.77 -0.00821
##  5 Mystery       454432  3.68  0.00453
##  6 Drama        3127384  3.67  0.00702
##  7 Crime        1061908  3.67  0.00386
##  8 Animation     373503  3.60 -0.00322
##  9 Musical       346063  3.56 -0.00306
## 10 Western       151334  3.56  0.000124
## 11 Romance      1369245  3.55 -0.000475
## 12 Thriller     1860825  3.51 -0.00180
## 13 Fantasy       740258  3.50 -0.000686
## 14 Adventure    1526741  3.49 -0.00525
## 15 Comedy       2832801  3.44 -0.000883
## 16 Action       2048879  3.42 -0.00414
## 17 Children      590064  3.42 -0.00759
## 18 Sci-Fi       1073399  3.40 -0.00437
## 19 Horror        552850  3.27  0.00250
```

Genres that have generally high ratings such as Film-Noir and Documentary have a positive genre bias whereas the ones that are further below have negative genre bias.

Interestingly, Horror which has the least average rating has a positive genre bias - this could be because the movie effect bias for horror movies is overestimated (and may include the genre bias in the movie effect bias, which is why the genre bias itself is positive to compensate)

Similarly, IMAX movies may have a strong movie bias already, and hence the genre itself has a small negative bias.

So let's generate predictions now using a model that includes the genre factor.

```
predicted_ratings_4 <- test_set %>% select(-title) %>% left_join(movie_genres, by='movieId') %>%
  left_join(genre_avgs, by='genre') %>%
  group_by(movieId, userId) %>% mutate(sum_b_g=sum(b_g)) %>% sample_n(1) %>% ungroup() %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(date_diff=round(year(anydate(timestamp))-year)) %>%
  left_join(date_avgs, by='date_diff') %>%
  mutate(pred = mu + b_i + b_u + b_d + sum_b_g) %>%
  select(movieId, userId, pred, rating)

#Set the predicted rating to the mean for any records in the test set
#that are not present in the training set
predicted_ratings_4[is.na(predicted_ratings_4$pred), 'pred'] <- mu#set it to just the mean
#Cut-offs for predicted rating should be 0.5 and 5.
predicted_ratings_4[predicted_ratings_4$pred<0, 'pred'] <- 0.5#any -ve predictions
predicted_ratings_4[predicted_ratings_4$pred>5, 'pred'] <- 5#any predictions > 5

model_genre_rmse <- RMSE(predicted_ratings_4$pred, predicted_ratings_4$rating)
model_genre_rmse
```

```
## [1] 0.8654113
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + Date + Genre Effects Model",
                                     RMSE = model_genre_rmse ))
```

We can see the RMSE has reduced marginally after including the genre effect.

Let's look at the RMSE values for all the models built so far:

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

method	RMSE
Using only average	1.0598803
Movie Effect Model	0.9437469
Movie + User Effects Model	0.8661863
Movie + User + Date Effects Model	0.8655267
Movie + User + Date + Genre Effects Model	0.8654113

Applying Regularisation

How can the performance of our model be improved? For this, let's first check where the model has been going wrong with our predictions:

Let's look at the top 10 movies with the highest movie bias:

```
movie_avgs %>% arrange(desc(b_i)) %>% head(10) %>%
  inner_join(unique(select(train_set, c('movieId', 'title'))), by='movieId') %>%
  select(title, b_i)
```

```
## # A tibble: 10 x 2
##    title                                                                b_i
##    <chr>                                                              <dbl>
##  1 Hellhounds on My Trail                                              1.49
##  2 Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo pe~  1.49
##  3 Satan's Tango (SÃ¡tÃ¡ntangÃ³)                                        1.49
##  4 Shadows of Forgotten Ancestors                                      1.49
##  5 Money (Argent, L')                                                  1.49
##  6 Fighting Elegy (Kenka erejii)                                       1.49
##  7 End of Summer, The (Kohayagawa-ke no aki)                           1.49
##  8 Blue Light, The (Das Blaue Licht)                                   1.49
##  9 Human Condition II, The (Ningen no joken II)                        1.24
## 10 Human Condition III, The (Ningen no joken III)                      1.24
```

Let's look at the 10 movies with the least movie bias:

```
movie_avgs %>% arrange(b_i) %>% head(10) %>%
  inner_join(unique(select(train_set, c('movieId', 'title'))), by='movieId') %>%
  select(title, b_i)
```

```
## # A tibble: 10 x 2
##    title                          b_i
```

```
##    <chr>                          <dbl>
##  1 30 Years to Life               -3.01
##  2 Besotted                       -3.01
##  3 Hi-Line, The                   -3.01
##  4 Altered                        -3.01
##  5 SuperBabies: Baby Geniuses 2   -2.72
##  6 Disaster Movie                 -2.64
##  7 From Justin to Kelly           -2.63
##  8 Roller Boogie                  -2.56
##  9 Criminals                      -2.51
## 10 Mountain Eagle, The            -2.51
```

These are all clearly obscure movies, but have been rated with a very high or a very low movie bias.

So, clearly a regularisation factor needs to be applied to reduce the effect of these extreme cases and prevent overfitting.

Let's now apply regularisation to the last prediction model built that included the movie factor, user factor, date factor and genre factor.

The prediction model should be evaluated against the test set for multiple values of the regularisation factor λ.

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

#NOTE: The below code will take a couple of hours to run.
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_d <- train_set %>% select(userId,movieId,rating,timestamp,year) %>%
    mutate(date_diff=round((year(anydate(timestamp))-year))) %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(date_diff) %>%
    summarize(b_d = sum(rating - mu - b_i - b_u)/(n()+l))

#We need not regularise genre effects as these are already averaged over all movies
#and user ratings for that genre. The number of ratings per genre is very high.
#So, these bias quantities will not be affected by L.
  genre_avgs <- train_set %>% select(-title) %>%
    left_join(movie_genres, by='movieId') %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    mutate(date_diff=round(year(anydate(timestamp))-year)) %>%
    left_join(b_d, by='date_diff') %>%
```

```
    mutate(b_g_ui = (rating - mu - b_i - b_u - b_d)/genre_count) %>%
    group_by(genre) %>% summarize(b_g=mean(b_g_ui))

  predicted_ratings <- test_set %>% select(-title) %>%
    left_join(movie_genres, by='movieId') %>%
    left_join(genre_avgs, by='genre') %>%
    group_by(movieId, userId) %>%
    mutate(sum_b_g=sum(b_g)) %>% sample_n(1) %>%
    ungroup() %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    mutate(date_diff=round(year(anydate(timestamp))-year)) %>%
    left_join(b_d, by='date_diff') %>%
    mutate(pred = mu + b_i + b_u + b_d + sum_b_g) %>%
    select(movieId, userId, pred, rating)

  predicted_ratings[is.na(predicted_ratings$pred), 'pred'] <- mu
  #Cut-offs for predicted rating should be 0.5 and 5.
  predicted_ratings[predicted_ratings$pred<0, 'pred'] <- 0.5#any -ve predictions
  predicted_ratings[predicted_ratings$pred>5, 'pred'] <- 5#any predictions > 5

  rmse_val <- RMSE(predicted_ratings$pred, predicted_ratings$rating)
  return(rmse_val)
})
l <- lambdas[which.min(rmses)]
```

Now the optimum λ can be identified which is equal to **5** which gave the least RMSE of **0.8647604**.

Building the Final Model and Evaluation against the Validation set

We have now identified our best model and the optimum regularisation factor.

Now that we have the best model with the optimum regularisation factor, let's use the entire *edx* set to derive our bias effect values and that will then be the final model.

```
rm(train_set, test_set)#delete the train and test set variables to free up memory.

mu <- edx %>% pull(rating) %>% mean()

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_d <- edx %>% select(userId,movieId,rating,timestamp,year) %>%
  mutate(date_diff=round((year(anydate(timestamp))-year))) %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(date_diff) %>%
```

```
  summarize(b_d = sum(rating - mu - b_i - b_u)/(n()+1))

genre_avgs <- edx %>% select(-title) %>% left_join(movie_genres, by='movieId') %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(date_diff=round(year(anydate(timestamp))-year)) %>%
  left_join(b_d, by='date_diff') %>%
  mutate(b_g_ui = (rating - mu - b_i - b_u - b_d)/genre_count) %>%
  group_by(genre) %>% summarize(b_g=mean(b_g_ui))
```

We now have the final model. We can now use this model to make predictions against the *validation* set and evaluate our model's performance on data it has never seen before.

```
#First extract the year out of the validation set
#Extract 2 groups - the title in the 1st and the year in the 2nd.
temp <- str_match(validation$title, '(.*)[(](\\d{4})[)]$')[,2:3]
validation$title <- str_trim(temp[,1])#trim the trailing whitespace in the extracted title
validation$year <- as.integer(temp[,2])
rm(temp) #remove variable to free up memory

predicted_ratings <- validation %>% select(-c(title, genres)) %>%
  left_join(movie_genres, by='movieId') %>%
  left_join(genre_avgs, by='genre') %>%
  group_by(movieId, userId) %>%
  mutate(sum_b_g=sum(b_g)) %>% sample_n(1) %>%
  ungroup() %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(date_diff=round(year(anydate(timestamp))-year)) %>%
  left_join(b_d, by='date_diff') %>%
  mutate(pred = mu + b_i + b_u + b_d + sum_b_g) %>%
  select(movieId, userId, pred, rating)

predicted_ratings[is.na(predicted_ratings$pred), 'pred'] <- mu
#Cut-offs for predicted rating should be 0.5 and 5.
predicted_ratings[predicted_ratings$pred<0, 'pred'] <- 0.5#any -ve predictions
predicted_ratings[predicted_ratings$pred>5, 'pred'] <- 5#any predictions > 5

rmse_val <- RMSE(predicted_ratings$pred, predicted_ratings$rating)
rmse_val
```

```
## [1] 0.8641228
```

Results

As we saw earlier, our initial models without regularisation showed iterative reduction in the RMSE value:

RMSE values for models without regularisation:

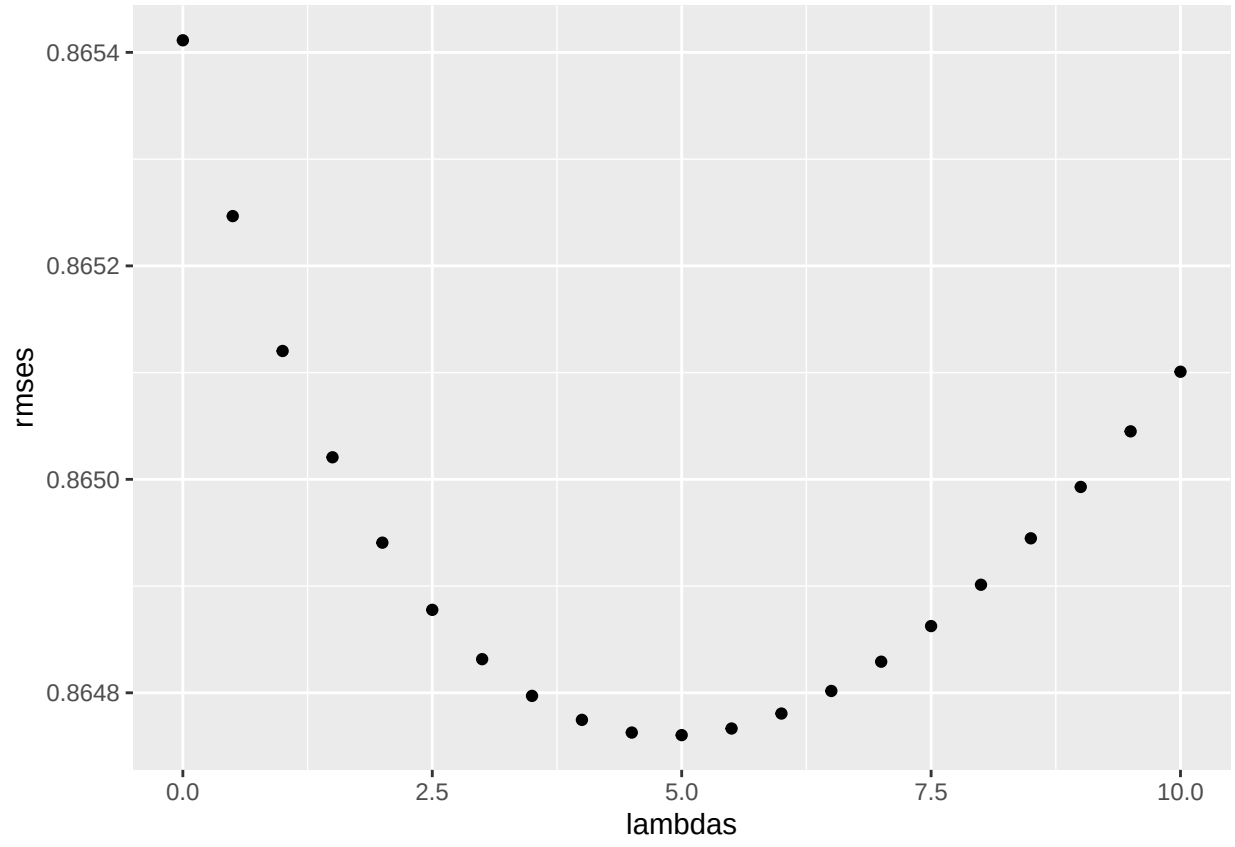
```
rmse_results %>% knitr::kable()
```

method	RMSE
Using only average	1.0598803
Movie Effect Model	0.9437469
Movie + User Effects Model	0.8661863
Movie + User + Date Effects Model	0.8655267
Movie + User + Date + Genre Effects Model	0.8654113

L2 Regularisation was then applied and various values of the regularisation factor λ were tried out and the RMSE evaluated for each value of λ.

Let's take a look at how the RMSE varied for various values of the regularisation factor λ:

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




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

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

We saw that the least RMSE of 0.8647604 was achieved with a value of $\lambda = 5$.

RMSE values for models without regularisation:

We can see how the RMSE improved iteratively and finally improved even further with regularisation:

```
rmse_results <- bind_rows(rmse_results,
                data_frame(method="Movie + User + Date + Genre Effects Model with Regularisation",
                RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Using only average	1.0598803
Movie Effect Model	0.9437469
Movie + User Effects Model	0.8661863
Movie + User + Date Effects Model	0.8655267
Movie + User + Date + Genre Effects Model	0.8654113
Movie + User + Date + Genre Effects Model with Regularisation	0.8647604

We finally ran this model against the *validation* set which contained movie-user combinations that the model had never seen before.

This achieved an RMSE value of

0.8641228

on the *validation* set.

Final Result:

method	RMSE
Using only average	1.0598803
Movie Effect Model	0.9437469
Movie + User Effects Model	0.8661863
Movie + User + Date Effects Model	0.8655267
Movie + User + Date + Genre Effects Model	0.8654113
Movie + User + Date + Genre Effects Model with Regularisation	0.8647604
Final RMSE on validation set	0.8641228

Conclusion and Future Work that can be done

A lot of ground has been covered in this study:

- A large 10 million record set containing user ratings for movies was used to start off with
- This was split into a 9 million record *edx* dataset and a 1 million record *validation* data set
- All analysis, exploration, visualisation, modeling and testing was done on the *edx* dataset
- The data was wrangled and information was extracted that would be useful for further data exploration and modeling
- Data exploration was done along with data visualisation and this led to identification of patterns in the user ratings
- Key influencers were identified that impact the user ratings for movies
- The *edx* dataset was split into a training set called *train_set* and a test set called *test_set*

- The key influencers were quantified in the form of factors or biases using the *train_set* - these were the movie effect factor, user effect factor, date effect factor and the genre effect factor
- Models were iteratively built using these factors one by one and the performance of the model was evaluated against the *test_set* and the RMSE value iteratively improved
- Regularisation was applied and several values of the regularisation factor were tested to identify the optimum regularisation factor value
- The final model that used all the 4 bias factors with the optimum regularisation factor was built on top of the entire *edx* data set
- This final model was finally used to generate predictions for the previously unseen *validation* data set and an RMSE value of 0.8641228 was achieved

So, what else can be done beyond this?

There are additional factors that could be tried out:

- The date factor used in this study only modeled the age of the movie review (the time between the year of the movie release and the year of review). The date factor can further be improved by also modeling:
  - the month of review (as intuitively we know that the ratings during the holiday seasons tend to be higher)
  - the date the movie was released as we saw earlier that movies released in 1993 or later have higher ratings
  - if we had the actual date of the movie release and not just the year, we could model the number of days between the movie release and the rating as users rating movies during the opening weekend or soon after tend to give higher ratings as they are usually fans
- If we have more computing capacity, We could also try out regression models using machine learning. For example, a random forest may work really well on modeling the genre effect as we could have 0/1 genre tags for each movie ie. have a separate column in the data set for each genre having a value of 1 if the movie belongs to that genre and 0 otherwise. A random forest model can use this to split nodes and model the genre effect better than what we have done in this study
- Matrix factorisation can be used to build a matrix with rows for movies and columns for users with the value of the user rating for the movie in individual cells. This can then be used to build a collaborative filtering recommendation model that fills in the empty cells (which are the ratings we want to predict) by identifying the movies that are closest to each other and the users that are closest to each other. This may give a better performance that the models used in this study