MovieLens Project

Kiko Núñez

9/8/2019

1. Introduction

1.1. Dataset description

This is the MovieLens 10M movie ratings. It is a stable benchmark dataset accounting with 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. It was released on January 2009.

You can find further information at: https://grouplens.org/datasets/movielens/10m/

The dataset has been partitioned into training set (aka edx) and validation set according to the instructions provided in the introductory exercise to the MovieLens project.

1.2. Goal of the project

To predict movie ratings in the validation set after fitting a model with the edx training set.

1.3. Key steps

1. Initial analysis (first visual inspection).
2. Preprocessing.
3. Understanding the dataset.
4. Building the model to predict ratings.
5. Apply trained model to validation dataset.
6. Conclusions and future work.

2. Methods

2.1. Processes and techniques

2.1.1. Initial analysis

We'll proceed with the initial analysis through a visual inspection of the variables:

```r
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ad
```

We notice that we have six variables:

1. userID: Is the identifier of the user who made the rating.
2. movieID: Is the identifier of the movie which was rated.
3. rating: Is the rating asigned by the user "userID" for the movie "movieID".
4. timestamp: Is the date when the rating was given (expressed in seconds, counting from 01:00:00 1/1/1970).
5. title: Is the title of the movie and the year it was released (between brackets).
6. genres: Is a list of the movie genres.

After this visual inspection, we find out that our model will work better if we translate some of those variables into more suitable ones. For instance, we will need to translate "timestamp" to "yearRating" and extract the year when the movie was releaed from "title".

2.1.2. Preprocessing

Let's convert the numeric timestamp to a date and then extract the year when the rating was made:

```r
edx <- transform(edx, timestamp = as.POSIXct(timestamp, origin = "1970-01-01"))
edx$yearRating <- as.integer(format(edx$timestamp, '%Y'))
```

And now let's extract the year from the variable "title":

```r
edx$yearMovie <- as.integer(sub("\\).*", "", sub(".*\\(", "", edx$title)))
```

Besides, we'll make a bagging of the genres so we can visualize how movie ratings perform for each one of them:

```r
# Split genre value by '|'
GenresBags <- unique(unlist(str_split(edx$genres, "\\|")))
print(GenresBags)
```

```
##  [1] "Comedy"             "Romance"            "Action"
##  [4] "Crime"              "Thriller"           "Drama"
##  [7] "Sci-Fi"             "Adventure"          "Children"
## [10] "Fantasy"            "War"                "Animation"
## [13] "Musical"            "Western"            "Mystery"
## [16] "Film-Noir"          "Horror"             "Documentary"
## [19] "IMAX"               "(no genres listed)"
```

```r
print(paste("Movies in the dataset have",
            length(GenresBags), "different types of genres."))
```

```
## [1] "Movies in the dataset have 20 different types of genres."
```

```r
# Set-up a matrix for bagging genres
GenresEdx <- matrix(, nrow(edx), length(GenresBags))
colnames(GenresEdx) <- GenresBags

## Populate the matrix of genres (this may take a while):
for (i in 1:length(GenresBags)) {
  GenresEdx[grep(GenresBags[i],edx$genres), i] <- 1
}
GenresEdx[is.na(GenresEdx)] <- 0

edx <- cbind(edx, GenresEdx)
```

At this point, we don't need the original columns of "timestamp" and "title" anymore, so let's drop them to improve the speed of the operations with the edx dataset:

```r
edx_clean <- edx %>% select(-c(timestamp, title))
```

3. Understanding the dataset

In this section, we'll gain some insights about the distribution of the variables and how they relate to the ratings given.

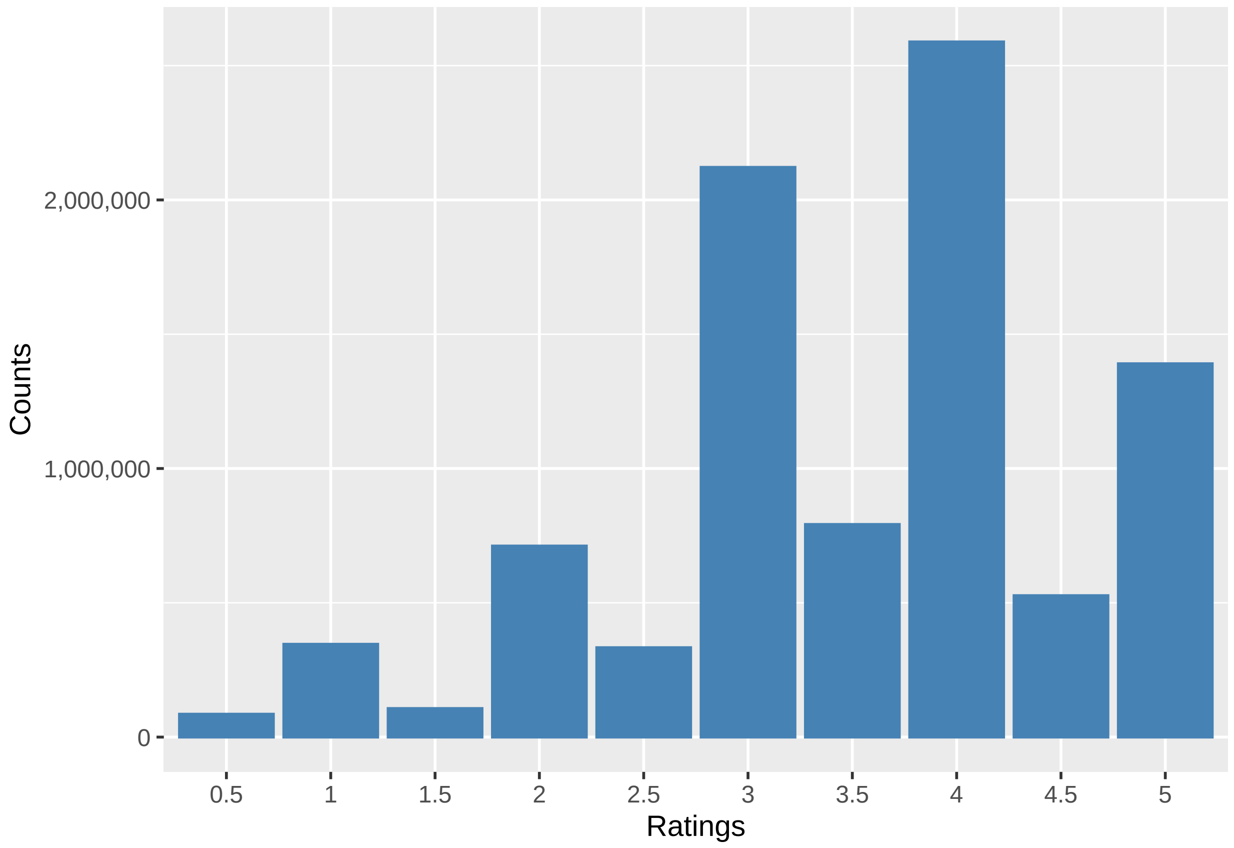
3.1. Mean ratings

First, let's visualize the overall movie ratings distribution in the dataset:

```r
summary(edx_clean$rating)
```

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

```r
edx_clean %>% ggplot(aes(x=factor(rating))) +
  geom_bar(color="steelblue", fill="steelblue") +
  labs(x = "Ratings", y = "Counts") +
  scale_y_continuous(labels = comma)
```

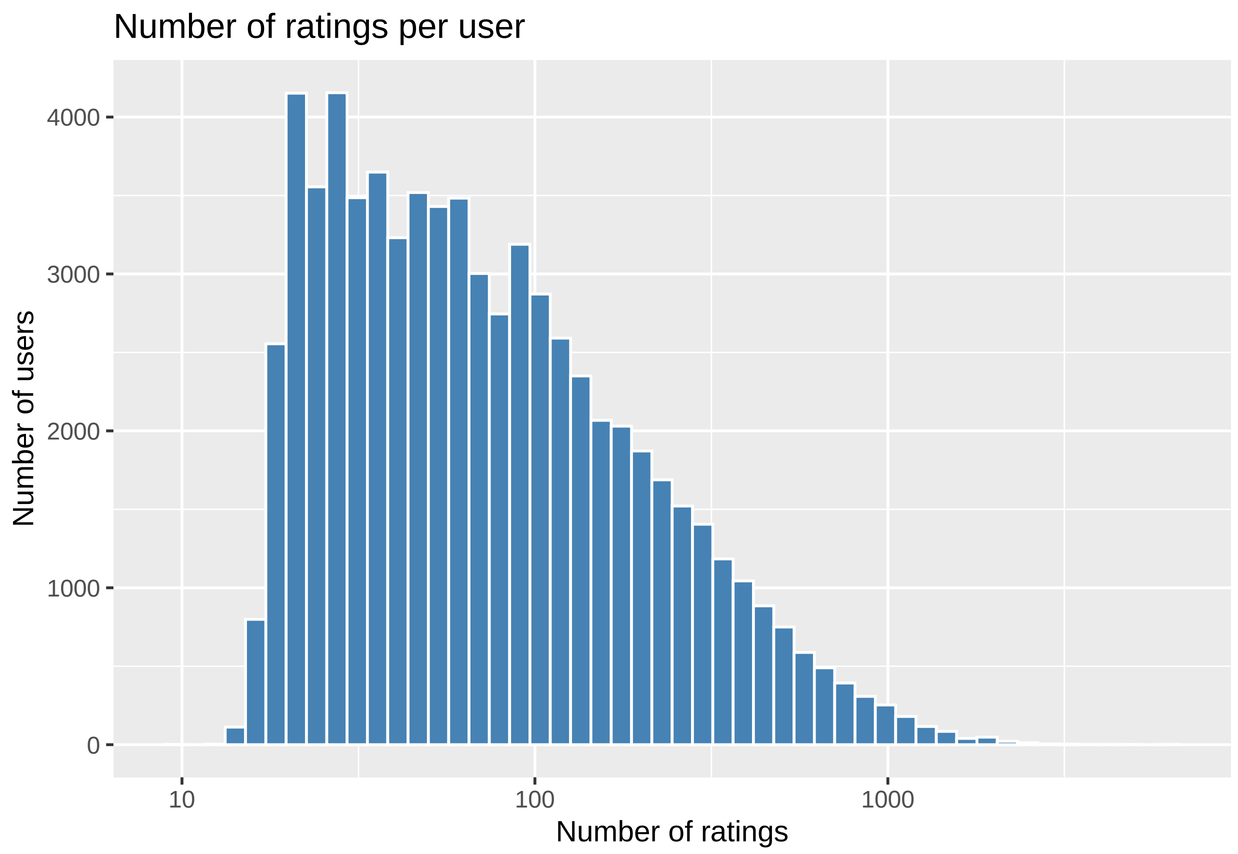


We can see that the distribution is slightly skewed towards the upper ratings and that half point ratings are less usual than the others.

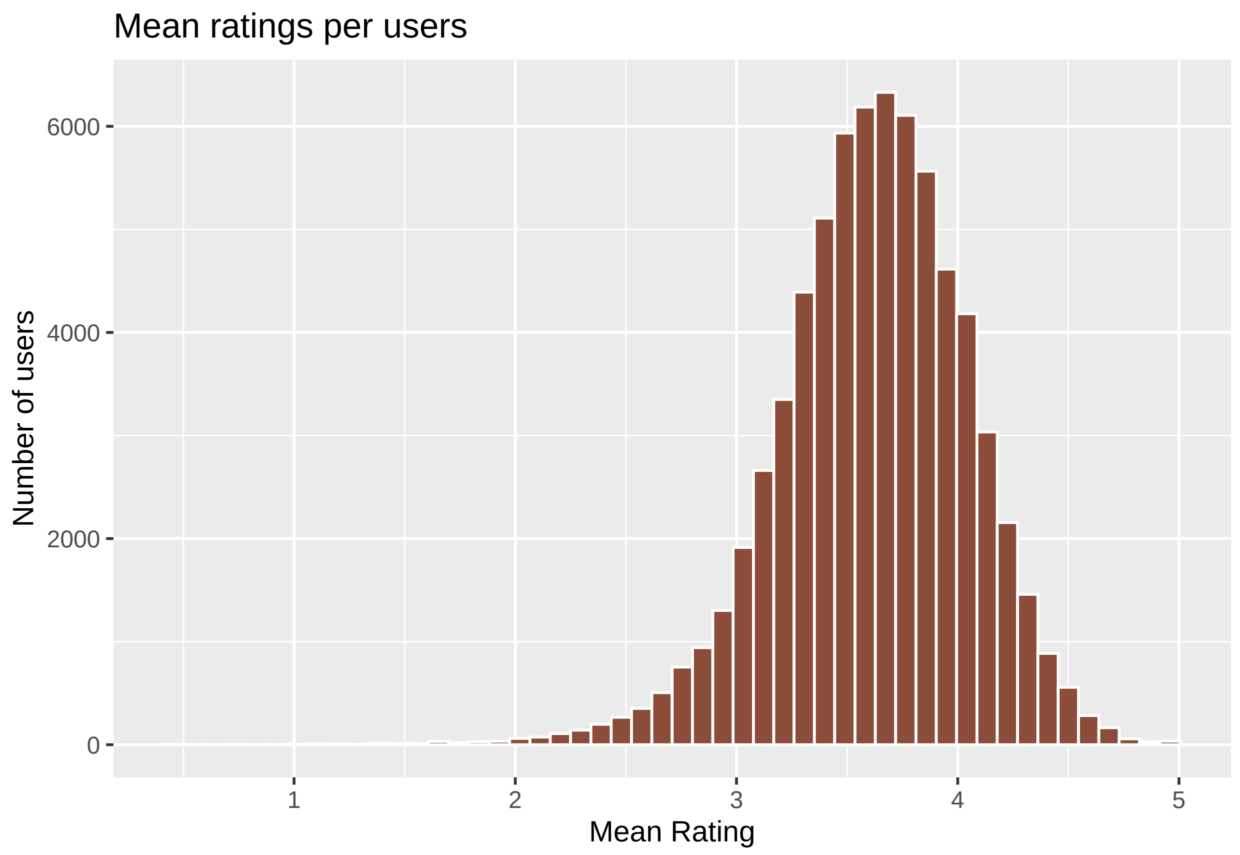
3.2. Users' behavior

Now let's see how users behave when rating the movies:

```r
edx_clean %>% group_by(userId) %>% summarise(Users = n()) %>%
  ggplot(aes(Users)) + geom_histogram(bins = 50, fill = "steelblue", color = "white") +
  scale_x_log10() + labs(title = "Number of ratings per user") +
  xlab("Number of ratings") + ylab("Number of users")
```



```r
edx_clean %>% group_by(userId) %>% summarise(meanRating = mean(rating)) %>%
  ggplot(aes(meanRating)) + geom_histogram(bins = 50, fill = "salmon4", color = "white") +
  labs(title = "Mean ratings per users") +
  xlab("Mean Rating") + ylab("Number of users")
```



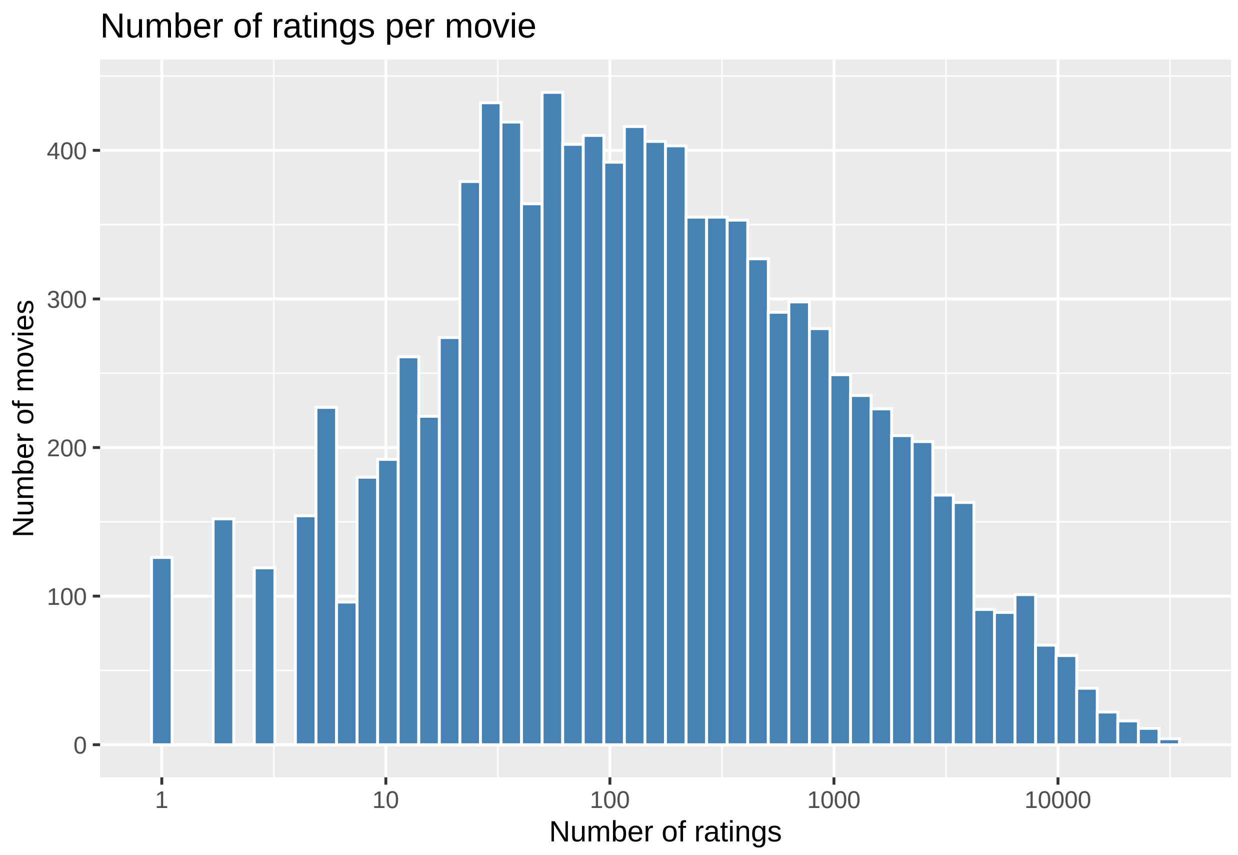
These plots give us the following information:

- Not all users rated the same number of movies. In fact, most of them rated less than 100 movies, while a small fraction of them rated more than 1000 (please note the logarithmic scale in the x axis of the firsst plot).
- The distribution of mean ratings per user is approximately normal (Gaussian) with a mean rating of 3.512, as reported in the summary of the dataset.

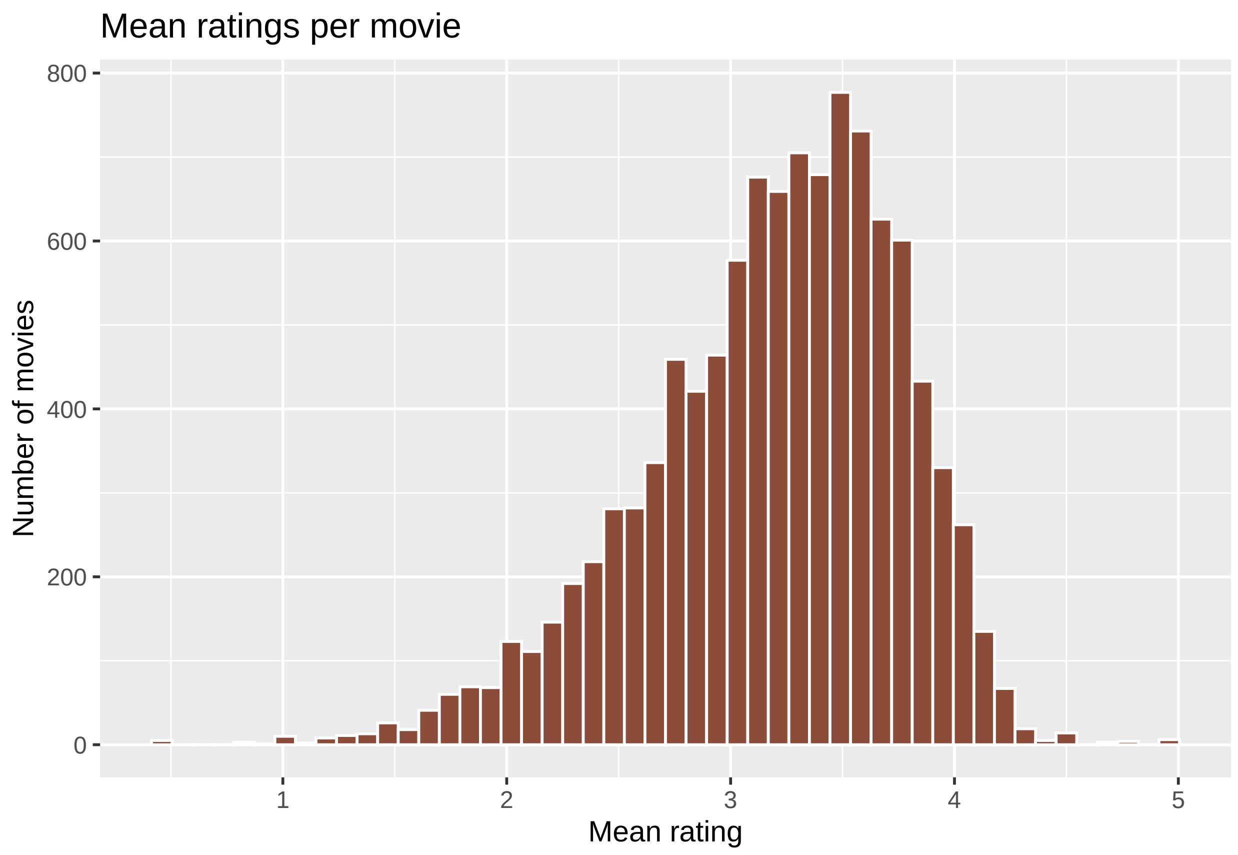
3.3. Movies' behavior

The following step will be to perform a similar analysis, but in this case let's look into the movies' behavior:

```
edx_clean %>% group_by(movieId) %>% summarise(Movies = n()) %>%
  ggplot(aes(Movies)) +
  geom_histogram(bins = 50, fill = "steelblue", color = "white") +
  scale_x_log10() + labs(title = "Number of ratings per movie") +
  xlab("Number of ratings") + ylab("Number of movies")
```



```
edx_clean %>% group_by(movieId) %>% summarise(meanRatingMovie = mean(rating)) %>%
  ggplot(aes(meanRatingMovie)) + geom_histogram(bins = 50, fill = "salmon4", color = "white") +
  labs(title = "Mean ratings per movie") +
  xlab("Mean rating") + ylab("Number of movies")
```



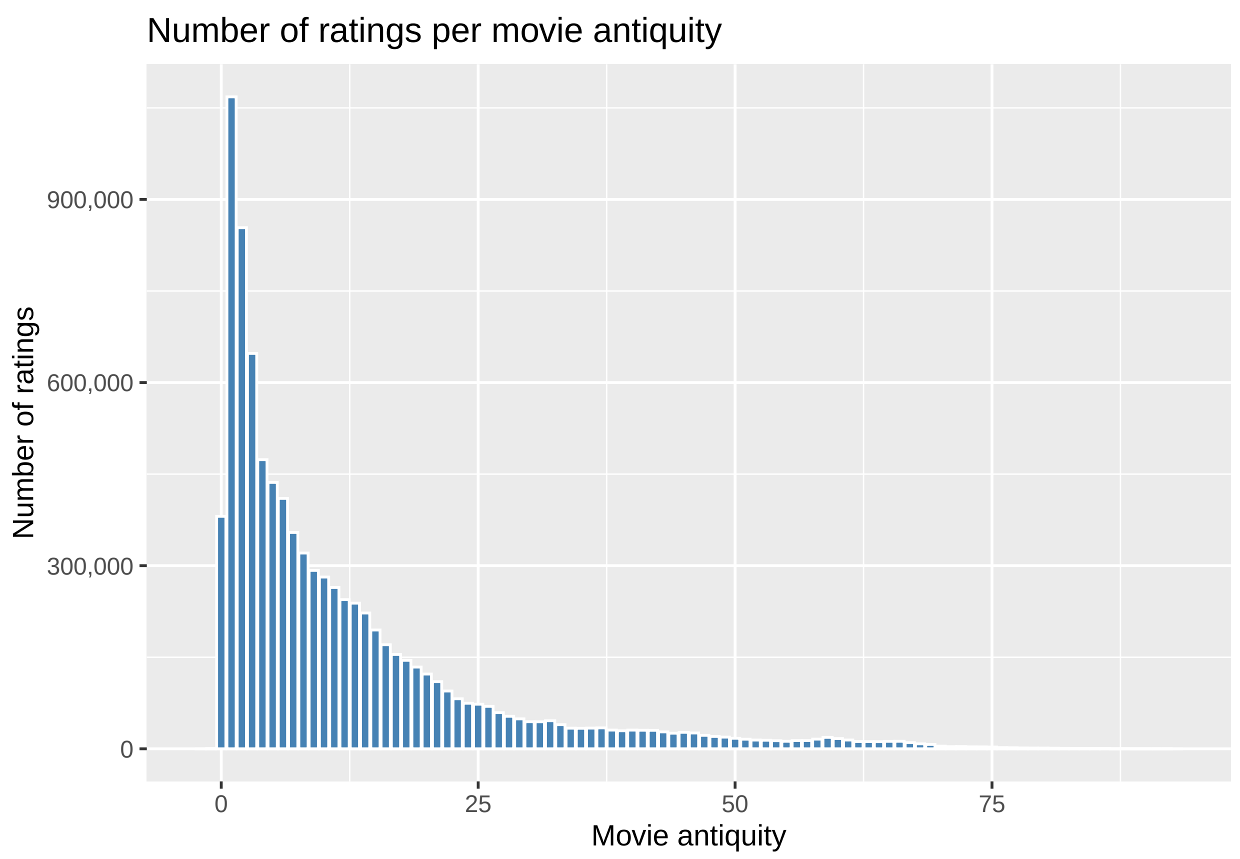
The information we can extract from here is that:

- Not all movies have been rated the same number of times. More than 100 movies were rated only once, most of them were rated around 100 times, and only a few were rated more than 10,000 times (these surely corresponds to blockbusters). (Please note again the logarithmic scale of the x axis in the first plot).
- The distribution of the mean ratings per movie is slightly skewed towards the upper ratings, and this is a bias that should be taken into account when building our predictive model.

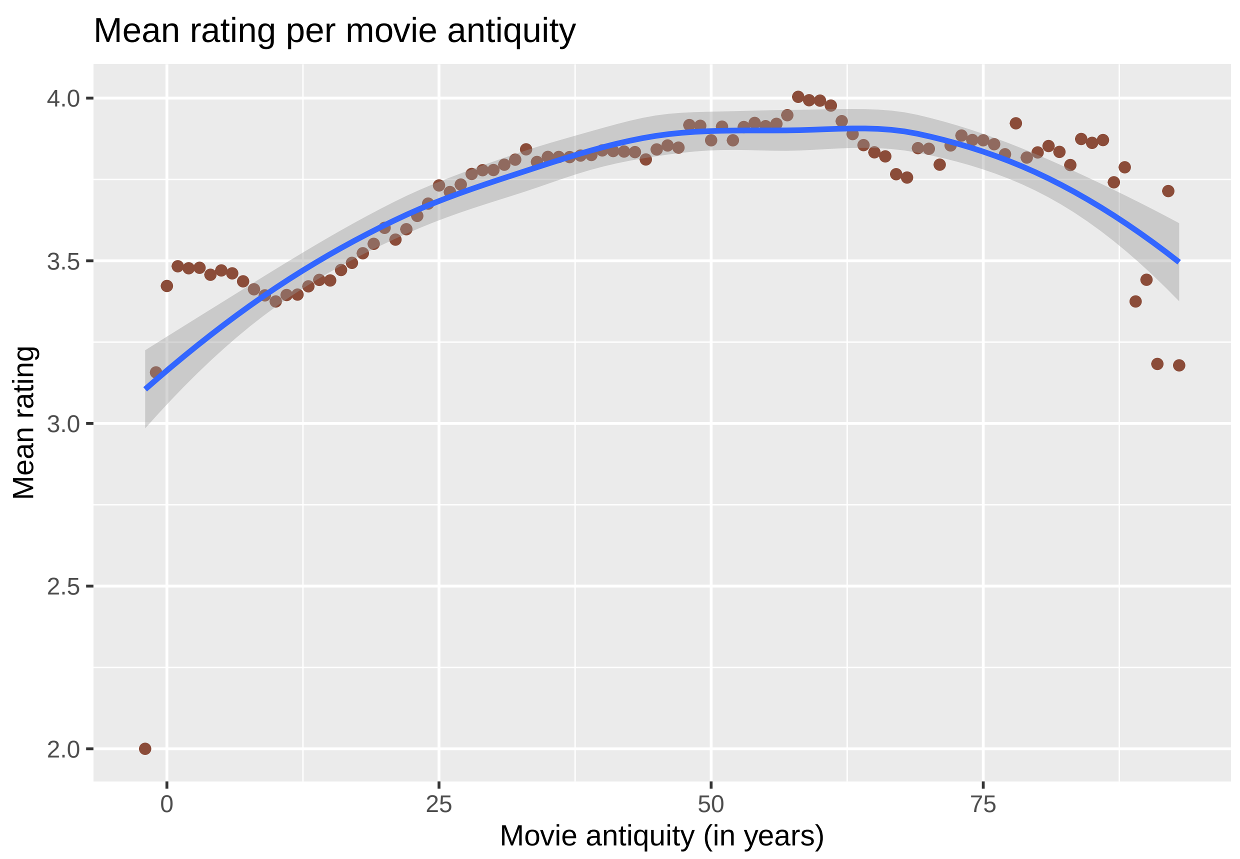
3.4. Ratings and movie antiquity

Next, let's explore the behavior of ratings when compared to movies antiquity:

```
edx_clean %>% mutate(yearDiff = yearRating - yearMovie) %>%
  group_by(yearDiff) %>% summarise(Difference = n()) %>%
  ggplot(aes(yearDiff, Difference)) + geom_bar(stat = "identity", fill = "steelblue", color = "white")
  scale_y_continuous(labels = comma) + labs(title = "Number of ratings per movie antiquity") +
  xlab("Movie antiquity") + ylab("Number of ratings")
```



```r
edx_clean %>% mutate(yearDiff = yearRating - yearMovie) %>%
  group_by(yearDiff) %>% summarise(meanRatingYearDiff = mean(rating)) %>%
  ggplot(aes(yearDiff, meanRatingYearDiff)) + geom_point(color = "salmon4") +
  geom_smooth(method = "loess", formula = y ~ x) +
  labs(title = "Mean rating per movie antiquity",
       x = "Movie antiquity (in years)", y = "Mean rating")
```



What we can see here is that ratings are given more often when the movie is recent (released 1-2 years ago), and that older movies are barely rated. However, the second plot shows that mean ratings for newer movies are usually lower than mean ratings for older movies, but this trend decreases for movies with that were released more than 70 years ago. This fact may also be taken into account by the predictive model.

3.5. Ratings by genre

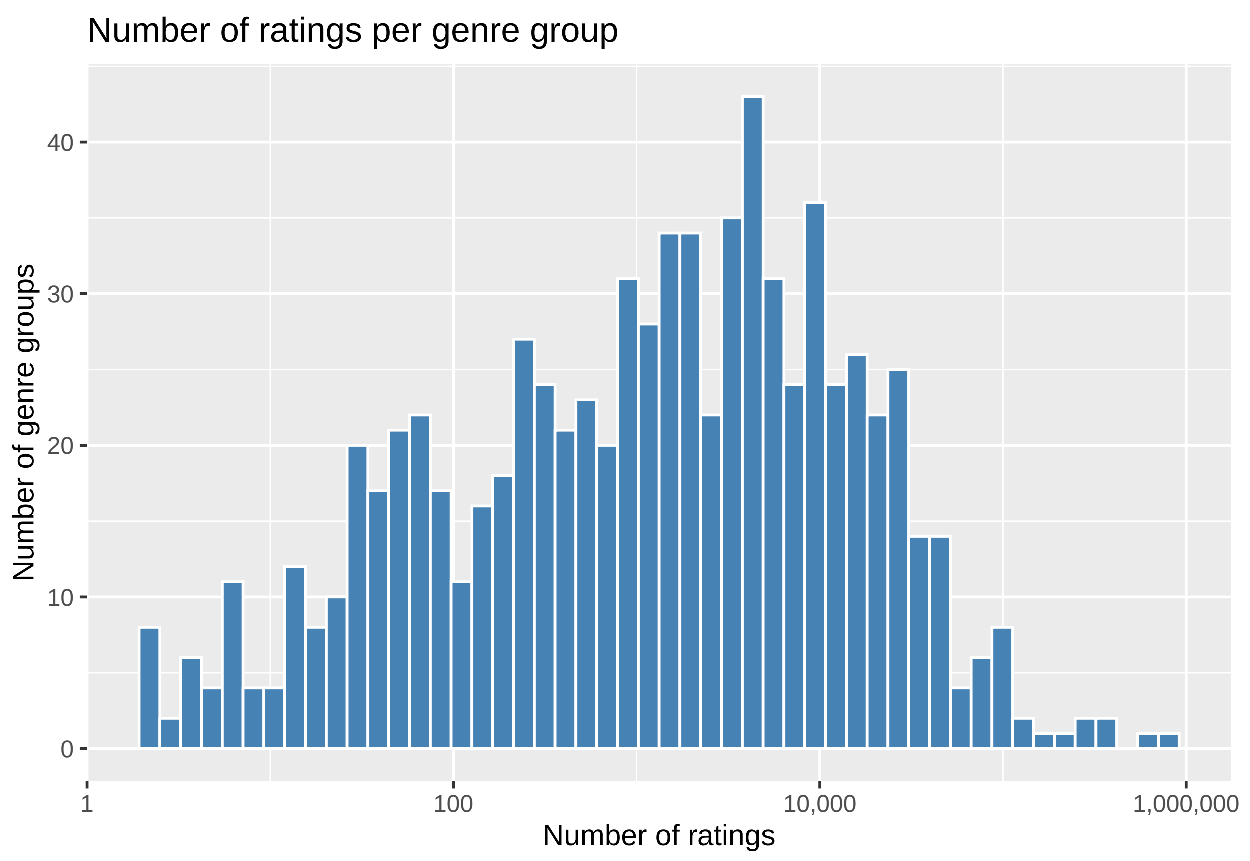
Now let's take a look into the movie genres. First, let's consider the genres "aggregated", this is in the same way they were registered in the original dataset:

```
cat("\n")
```

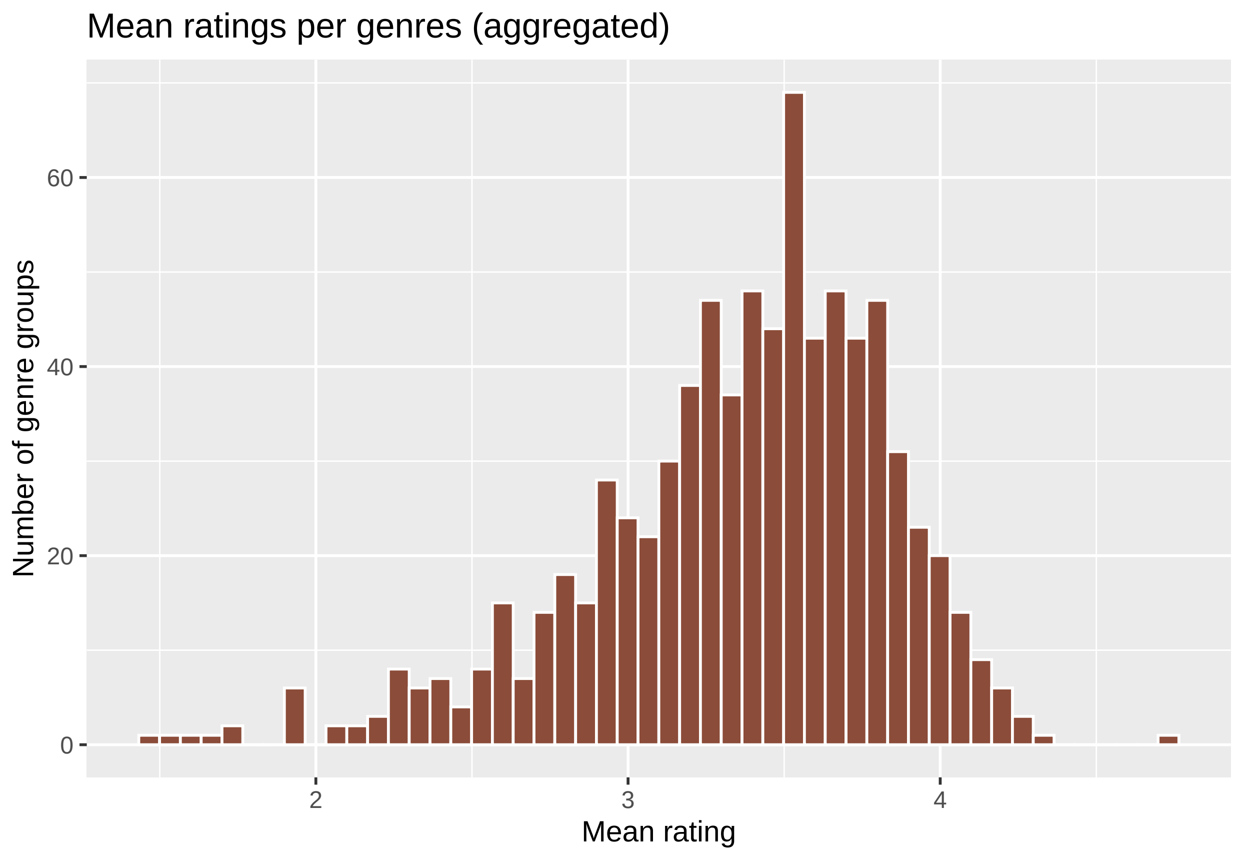
```
print(paste("the number of genres groups found are: ", length(unique(edx_clean$genres))))
```

```
## [1] "the number of genres groups found are:  797"
```

```
edx_clean %>% group_by(genres) %>% summarise(Genres = n()) %>%
  ggplot(aes(Genres)) + geom_histogram(bins = 50, fill = "steelblue", color = "white") +
  scale_x_log10(labels = comma) + labs(title = "Number of ratings per genre group") +
  xlab("Number of ratings") + ylab("Number of genre groups")
```



```
edx_clean %>% group_by(genres) %>% summarise(meanRatingGenres = mean(rating)) %>%
  ggplot(aes(meanRatingGenres)) +
  geom_histogram(bins = 50, fill = "salmon4", color = "white") +
  labs(title = "Mean ratings per genres (aggregated)") +
  xlab("Mean rating") + ylab("Number of genre groups")
```



We can see that there are about 800 genres groups in the original dataset, and that most of them have been rated among 100 and 10,000 times (plese notice the log scale in the horizontal axis in the first plot). Besides, the ratings related to the genre groups follow a similar trend as we have observed previously, with a mean value around 3.5 and slightly skewed to the upper ratings. In any case, we can extract more insigths if we analyze the genres individually.

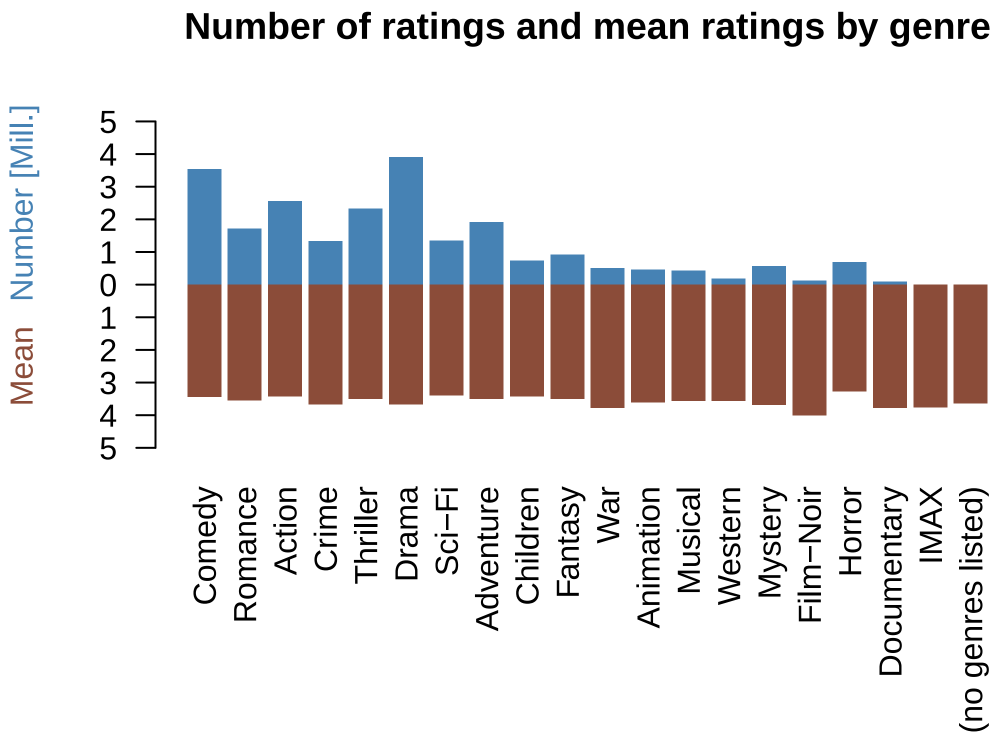
```r
cat("\n")
```

```r
p_countGenres <- NULL
p_ratingGenres <- NULL

for (i in 1:length(GenresBags)) { # Calculate counts and mean ratings per genre
  index <- which(edx_clean[GenresBags[i]] == 1)
  p_ratingGenres <- append(p_ratingGenres, mean(edx_clean[index, "rating"]))
  p_countGenres <- append(p_countGenres, length(index))
}
names(p_ratingGenres) <- GenresBags
names(p_countGenres) <- GenresBags

# Plot the results
par(mai = c(1.8, 1, 1, 1))
barplot(p_countGenres/1000000, ylim = c(-5, 5), axes = FALSE, border = NA,
        col = "steelblue", las = 2, main = "Number of ratings and mean ratings by genre")
barplot(-p_ratingGenres, add = TRUE, axes = FALSE, col = "salmon4", border = NA, names.arg = NA)
axis(2, at = seq(-5, 5, 1),
     labels = c(rev(seq(0, 5, 1)), seq(1, 5, 1)), las = 2)
```

```r
mtext("Mean", 2, line = 3, at = -2.5, col = "salmon4")
mtext("Number [Mill.]", 2, line = 3, at = 2.5, col = "steelblue")
```

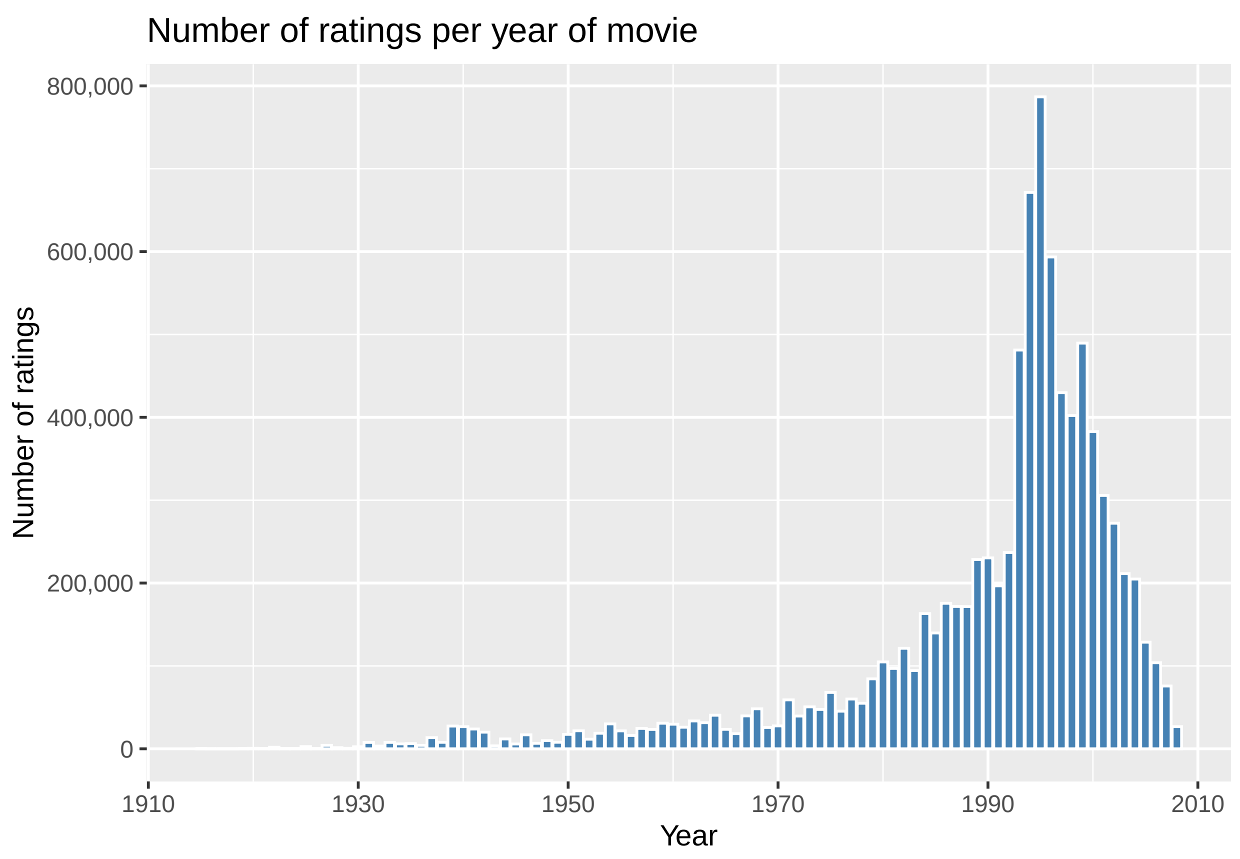


This plot helps us to understand which genres are rated more often, being at the top-3 Drama, Comedy and Action, while the least rated are (no genres listed), IMAX and Documentary ones. We can see that the mean rating for all of them is among 3 and 4 points, being Horror genre the worst rated and Film-Noir the best one (although the low number of samples must prevent us from extracting a conclusion in this case).

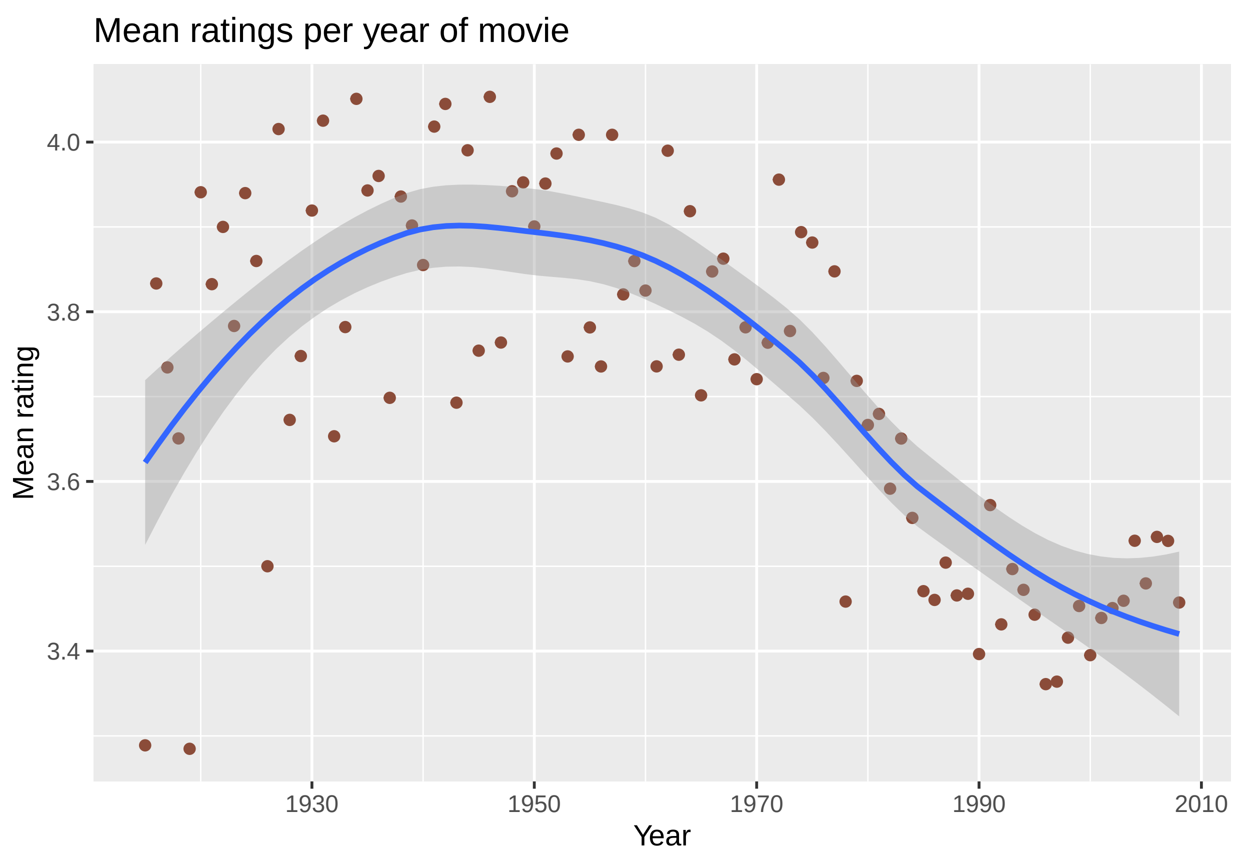
3.6. Ratings by year of release

Finally, let's try to gain some information from the year the movie was released:

```r
edx_clean %>% group_by(yearMovie) %>% summarise(Years = n()) %>%
  ggplot(aes(yearMovie, Years)) +
  geom_bar(stat = "identity", fill = "steelblue", color = "white") +
  labs(title = "Number of ratings per year of movie") +
  xlab("Year") + ylab("Number of ratings") + scale_y_continuous(labels = comma)
```



```
edx_clean %>% group_by(yearMovie) %>% summarise(meanRatingYear = mean(rating)) %>%
  ggplot(aes(yearMovie, meanRatingYear)) + geom_point(color = "salmon4") +
  geom_smooth(method = "loess", formula = y ~ x) +
  labs(title = "Mean ratings per year of movie") +
  xlab("Year") + ylab("Mean rating")
```



In this case, what we can see is that films released in the 90s have been more rated than others. This could be due to several factors, such as the number of blockbusters in that decade and also due to a bias in the dataset, since 54.45% of filmes included in the MovieLens dataset were released in that decade. This rate could be calculated as follows:

```
length(which((edx$yearMovie >= 1990) & (edx$yearMovie <= 2000)))/nrow(edx)
```

```
## [1] 0.5445524
```

From the second plot, the main finding is that movies released between the 30s and 60s are, on average, rated higher than the rest. This could also be helpful when designing the predicition model.

4. Building the model to predict ratings

In this work, we'll make use of the ensemble method to build the predictive model for movie ratings. We'll start from the most basic model (baseline) and then we'll add other effects that can bias the prediction.

4.1. Baseline (mean) model

The most basic predictive model would be to compare to the overall mean of the ratings:

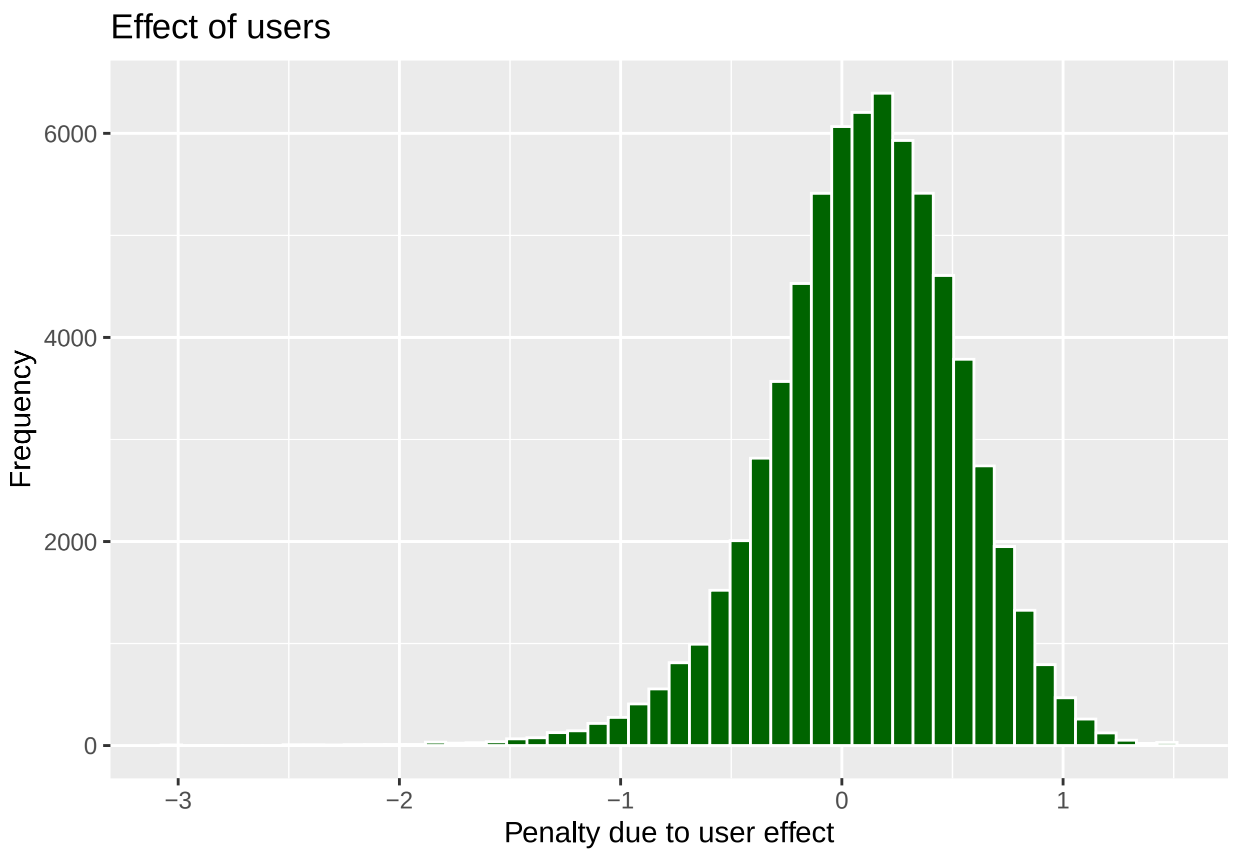
```
Baseline <- mean(edx_clean$rating)
print(paste("Baseline model (average): ", Baseline))
```

```
## [1] "Baseline model (average):  3.51246520160155"
```

4.2. Model including user effect

Then we can add a penalty term due to the user effect:

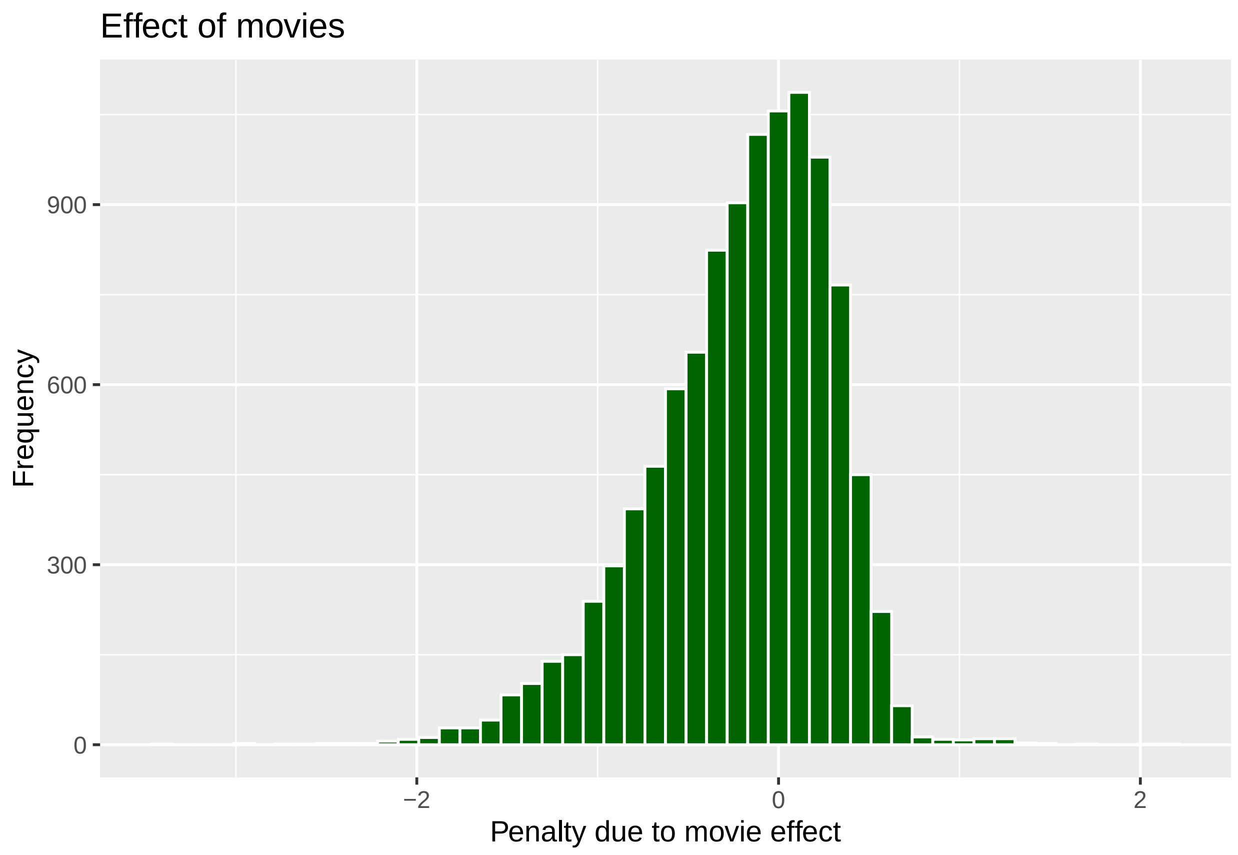
```r
meanUsers <- edx_clean %>% group_by(userId) %>%
  summarise(p_user = mean(rating - Baseline))
meanUsers %>% ggplot(aes(p_user)) +
  geom_histogram(bins = 50, fill = "darkgreen", color = "white") +
  labs(title = "Effect of users") +
  xlab("Penalty due to user effect") + ylab("Frequency")
```



4.3. Model including user and movie effects

Now we can add a new penalty term to incorporate the effect that the movie has on the ratings:

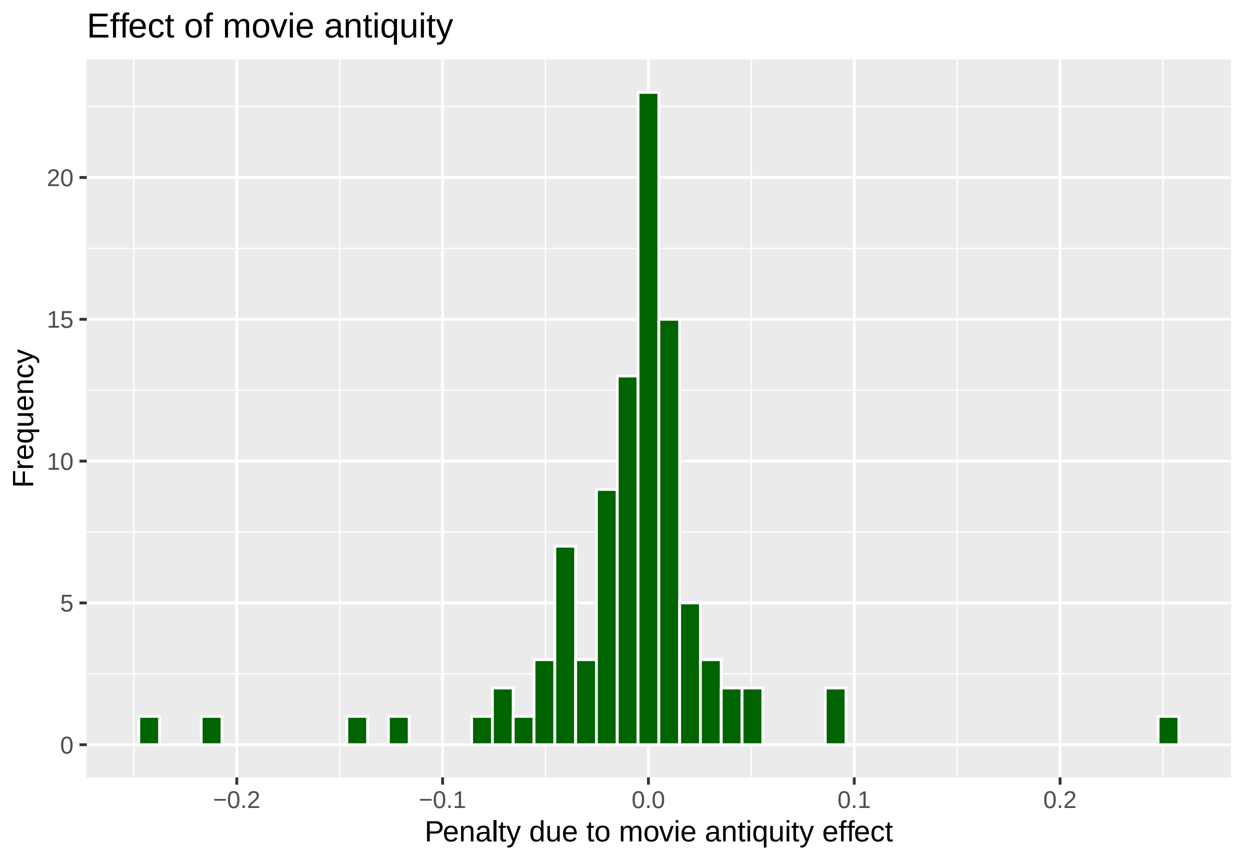
```r
meanMovies <- edx_clean %>% left_join(meanUsers, by = "userId") %>%
  group_by(movieId) %>% summarise(p_movie = mean(rating - Baseline - p_user))
meanMovies %>% ggplot(aes(p_movie)) +
  geom_histogram(bins = 50, fill = "darkgreen", color = "white") +
  labs(title = "Effect of movies") +
  xlab("Penalty due to movie effect") + ylab("Frequency")
```



4.4. Model including user, movie and antiquity effects

We can continue adding other effects to the model. For instance, we can include the bias on the ratings provided by the antiquity of the movie being rated:

```r
meanDiffYear <- edx_clean %>% mutate(diffYear = yearRating - yearMovie) %>%
  left_join(meanUsers, by = "userId") %>% left_join(meanMovies, by = "movieId") %>%
  group_by(diffYear) %>% summarise(p_diffYear = mean(rating - Baseline - p_user - p_movie))
meanDiffYear %>% ggplot(aes(p_diffYear)) +
  geom_histogram(bins = 50, fill = "darkgreen", color = "white") +
  labs(title = "Effect of movie antiquity") +
  xlab("Penalty due to movie antiquity effect") + ylab("Frequency")
```

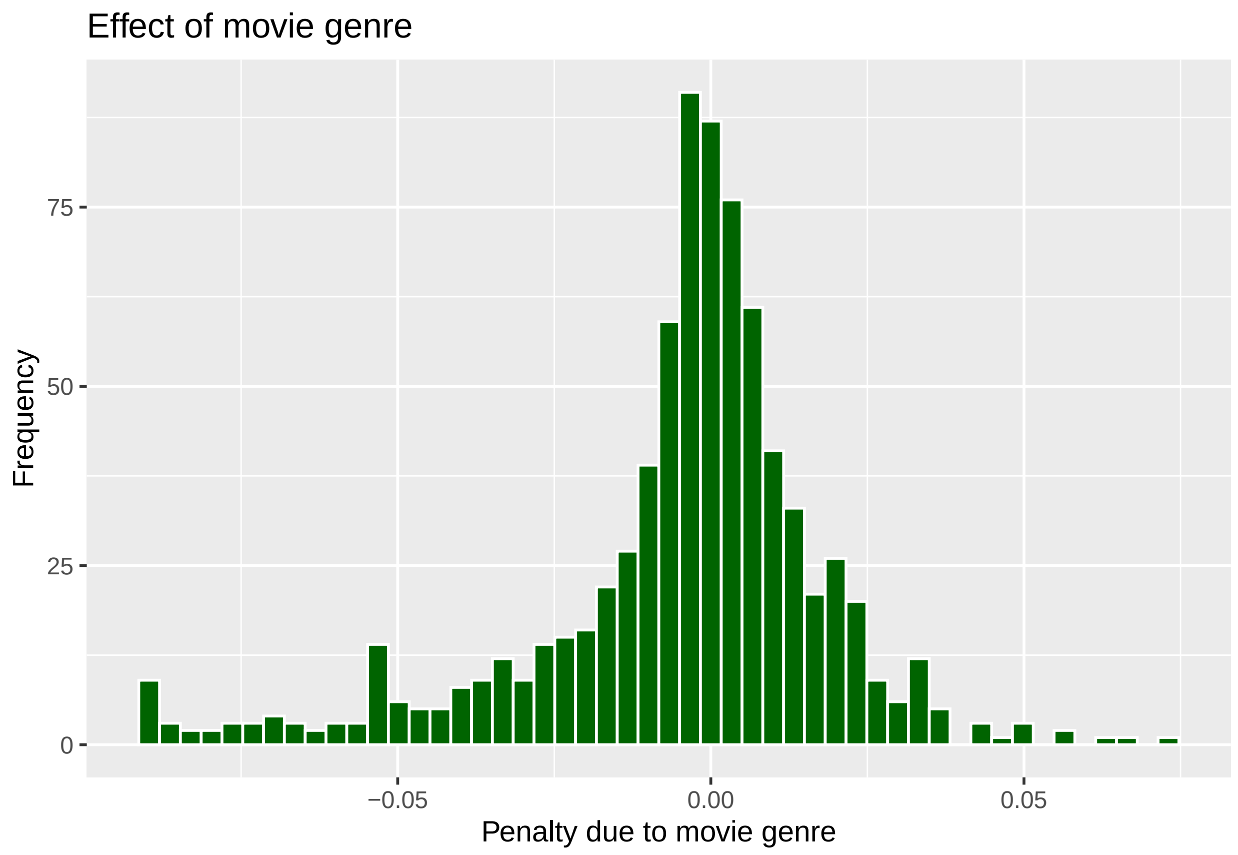


As this distribution is almost centerd at 0, and the differences are in a very short interval [-0.2, 0.2], this penalty will not influence very much to the predicted rating.

4.5. Model including user, movie, antiquity and genre effects

Finally, we can add the genre group effect to the model by including the following lines of code:

```
meanGenre <- edx_clean %>% mutate(diffYear = yearRating - yearMovie) %>%
  left_join(meanUsers, by = "userId") %>% left_join(meanMovies, by = "movieId") %>%
  left_join(meanDiffYear, by = "diffYear") %>% group_by(genres) %>%
  summarise(p_genres = mean(rating - Baseline - p_user - p_movie - p_diffYear))
meanGenre %>% ggplot(aes(p_genres)) +
  geom_histogram(bins = 50, fill = "darkgreen", color = "white") +
  labs(title = "Effect of movie genre") +
  xlab("Penalty due to movie genre") + ylab("Frequency")
```



The same comment as before can apply here, as the interval is even shorter [-0.1, 0.1] and the distribution is almost centerd around 0.

5. Apply trained model to validation dataset

5.1. Preprocessing of the validation dataset

In order to apply the model we've just developed, we need to preprocess the validation dataset in the same way that we did with the edx dataset. In this sense, we'll have to extract year of ratings from timestamp, add year of movie column and perform genre bagging operations. After that, we'll drop useless columns such as timestamp and title:

```r
validation <- transform(validation, timestamp = as.POSIXct(timestamp, origin = "1970-01-01"))

# Add yearRating column to validation dataset
validation$yearRating <- as.integer(format(validation$timestamp, '%Y'))

## Add year of movie column:
validation$yearMovie <- as.integer(sub("\\).*", "", sub(".*\\(", "", validation$title)))

## Genre bagging. Set-up a matrix for bagging genres:
GenresVal <- matrix(, nrow(validation), length(GenresBags))
colnames(GenresVal) <- GenresBags

for (i in 1:length(GenresBags)) {
  GenresVal[grep(GenresBags[i],validation$genres), i] <- 1
}
GenresVal[is.na(GenresVal)] <- 0

validation <- cbind(validation, GenresVal)

## Drop useless columns (timestamp and title):
validation_clean <- validation %>% select(-c(timestamp, title))
```

5.2. Model accuracy (RMSE)

Finally, let's test how good is our model predicting movie ratings at each stage of the ensemble we have built:

```r
RMSE_Baseline <- RMSE(validation_clean$rating, Baseline)
print(paste("RMSE in baseline model: ", RMSE_Baseline))
```

```
## [1] "RMSE in baseline model:  1.06120181029262"
```

```r
y_hat_users <- validation_clean %>% left_join(meanUsers, by = "userId") %>%
  mutate(pred_rating = Baseline + p_user)
RMSE_Users <- RMSE(validation_clean$rating, y_hat_users$pred_rating)
print(paste("RMSE adding user effect: ", RMSE_Users))
```

```
## [1] "RMSE adding user effect:  0.978335971005418"
```

```r
y_hat_movies <- validation_clean %>% left_join(meanUsers, by = "userId") %>%
  left_join(meanMovies, by = "movieId") %>%
  mutate(pred_rating = Baseline + p_user + p_movie)
RMSE_UsersMovies <- RMSE(validation_clean$rating, y_hat_movies$pred_rating)
print(paste("RMSE adding movie and user effects: ", RMSE_UsersMovies))
```

```
## [1] "RMSE adding movie and user effects:  0.881609571352724"
```

```r
y_hat_diffYear <- validation_clean %>% mutate(diffYear = yearRating - yearMovie) %>%
  left_join(meanUsers, by = "userId") %>% left_join(meanMovies, by = "movieId") %>%
  left_join(meanDiffYear, by = "diffYear") %>%
  mutate(pred_rating = Baseline + p_user + p_movie + p_diffYear)
RMSE_UsersMoviesDiffYear <- RMSE(validation_clean$rating, y_hat_diffYear$pred_rating)
print(paste("RMSE adding user, movie and antiquity effects: ", RMSE_UsersMoviesDiffYear))
```

```
## [1] "RMSE adding user, movie and antiquity effects:  0.880923995726298"
```

```r
y_hat_genre <- validation_clean %>% mutate(diffYear = yearRating - yearMovie) %>%
  left_join(meanUsers, by = "userId") %>% left_join(meanMovies, by = "movieId") %>%
  left_join(meanDiffYear, by = "diffYear") %>% left_join(meanGenre, by = "genres") %>%
  mutate(pred_rating = Baseline + p_user + p_movie + p_diffYear + p_genres)
RMSE_UsersMoviesDiffYearGenre <- RMSE(validation_clean$rating, y_hat_genre$pred_rating)
print(paste("RMSE adding user, movie, antiquity and genre effects: ", RMSE_UsersMoviesDiffYearGenre))
```

```
## [1] "RMSE adding user, movie, antiquity and genre effects:  0.880870231465023"
```

6. Conclusions

6.1. Summary of the report

In this work, we have applied the concepts learnt during the edx Data Science course to a well known and validated dataset such as the MovieLens one. In this sense, we have applied several techniques related to: data wrangling with the `dplyr` library, text processing with the `stringr` library and visual inspection of the dataset with the `ggplot` library, among others.

Finally, we have built a prediction model based on the ensemble technique for movie ratings based on several features such as movie, user, genre and movie antiquity, achieving an RMSE of 0.8809.

6.2. Limitations

This work has an important limitation regarding the approach followed to build the prediction model given that more advanced models are not feasible to be implemented in regular laptops given the size of the dataset and the computational needs of them. For instance, we tried to train a support vector machine (SVM), an artificial neural network (ANN) and a gradient boosted decision tree (XGBoost) durting the exploratory stage of this work and decided to stop the computations since none of them yielded a result after several hours of execution. For that reason, we decided to choose the ensemble approach, as it needs less computational resources during its building.

6.3. Future work

Future work could address the inclusion of new penalty terms to the ensemble. For instance, it could be worthy to explore the importance of the year of release to the prediction of the movie rating and see how this impacts on the accuracy of the model.

On the other hand, a subset of the full edx dataset could be used for exploratory purposes to test the accuracy of other more advanced methods as those metioned in the Limitations section.