Capstone HarvardX: PH125.9x Movielens Prediction

Marcus Vinicius Goulart Gonzaga Junior

Table of Contents

Introduction 2

Recommender Systems 2

The Project Evaluation 2

The Dataset 3

Methods and Analysis 4

Overview 4

Counting Users and movies 4

Ratings distribution 5

Ratings per member 6

Ratings per movie 7

Movies rated per year 8

Ratings per users 9

Correlations 10

Number of ratings vs avg movie ratings 11

Predictive Model 12

I. Average movie rating model 12

II. Movie effect model 13

III. Movie and user effect model 14

IV. Regularized movie and user effect model 16

V - Apply Model to Validation Set 19

Results 19

Conclusion 20

References 20

Environment 21

Introduction

This project is part of the activities aimed at certification in data science, PH125.9x Capstone. The challenge is to produce a movie recommendation system, using machine learning techniques based on dataset provide by GroupLens Research at the University of Minnesota. The dataset contains approximately 10 Millions of movies ratings, divided roughly in 9 Million for training and one Milion for validation.

Recommender Systems

Recommender Systems is a set of techniques aimed to discover data patterns in the data set by learning consumers choices and produces the outcomes that co-relates to their needs and interests. Algorithms try to provide the most relevant and accurate items to the user by filtering useful stuff from a massive pool of information base.

Recommender systems (Felfernig et al., 2007, 2013; Jannach et al., 2010; Ricci et al., 2011) support users in the process of finding and selecting products (items) from a given assortment. Examples of such items are movies, books, songs, financial services, apartments, and so one.

The Project Evaluation

According EDX Grading Rubric, this project will be evaluated considering three issues:

1. Quality of Report:
   Includes all required sections, is easy to follow with good supporting detail throughout, and is insightful and innovative.

2. Quality of Code Easy to follow, is consistent with the report, and is well-commented

3. Algorithm Performance

The performance will be evaluated using a The Root Mean Square Error (RMSE) that is a frequently used measure and consists in a measure the difference between values predicted by a model and the values actually observed. The RMSE serves to aggregate that computation into a single measure of predictive power.

The performance reference gived by EDX for the grading was: 5 points: RMSE >= 0.90000, 10 points: 0.86550 <= RMSE <= 0.89999 ,15 points: 0.86500 <= RMSE <= 0.86559, 20 points: 0.86490 <= RMSE <= 0.86499 25 points: RMSE <= 0.8649 The lower the RMSE value, the better the performance and consequently the higher number of points obtained.

The function that computes the RMSE for vectors of movie ratings and their corresponding predictors is:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

$y_{u,i}$ as defined as the rating for movie i by user u and denote prediction with $\hat{y}_{u,i}$

In R code it will be:

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

The Dataset

The MovieLens dataset were obtained from these addresses:
https://grouplens.org/datasets/movielens/10m/
http://files.grouplens.org/datasets/movielens/ml-10m.zip EDX provided the code for import, as well as for the creation of two subdatasets, one for training and one for validation

There are three files:

1.movies.dat

Movie information is contained in this file. Each line represents one movie. MovieID - is the MovieLens id. Movie titles - include year of release. They are entered manually, so errors and inconsistencies may exist. Genres - Action,Adventure,Animation,Children,Comedy,Crime,Documentary,Drama,Fantasy,Film-Noir,Horror,IMAX,Musical,Mystery,Romance,Sci-Fi,Thriller,War,Western

2.ratings.dat

All ratings are contained in this file. Each line of this file represents one rating of one movie by one user, and has the following format UserID::MovieID::Rating::Timestamp.

The lines within this file are ordered first by UserID, then, within user, by MovieID. Ratings are made on a 5-star scale, with half-star increments. Timestamps represent the time of rating in seconds since midnight UTC of January 1, 1970.

3.tags.dat

All tags are contained in this file. Each line represents one tag applied to one movie by one user, and has the following format UserID::MovieID::Tag::Timestamp. The lines within this file are ordered first by UserID, then, within user, by MovieID. Tags are user generated metadata about movies. Each tag is typically a single word, or short phrase. The meaning, value and purpose of a particular tag is determined by each user. Timestamps represent the time of tagging in seconds since midnight UTC of January 1, 1970.

Methods and Analysis

Overview

First, a simple summary

```
##      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         
##  Length:9000055     Length:9000055    
##  Class :character   Class :character  
##  Mode  :character   Mode  :character  
##                                       
##                                       
## 
```

Add column year_rated from timestamp

```
edx <- mutate(edx, year_rated = year(as_datetime(timestamp)))
```

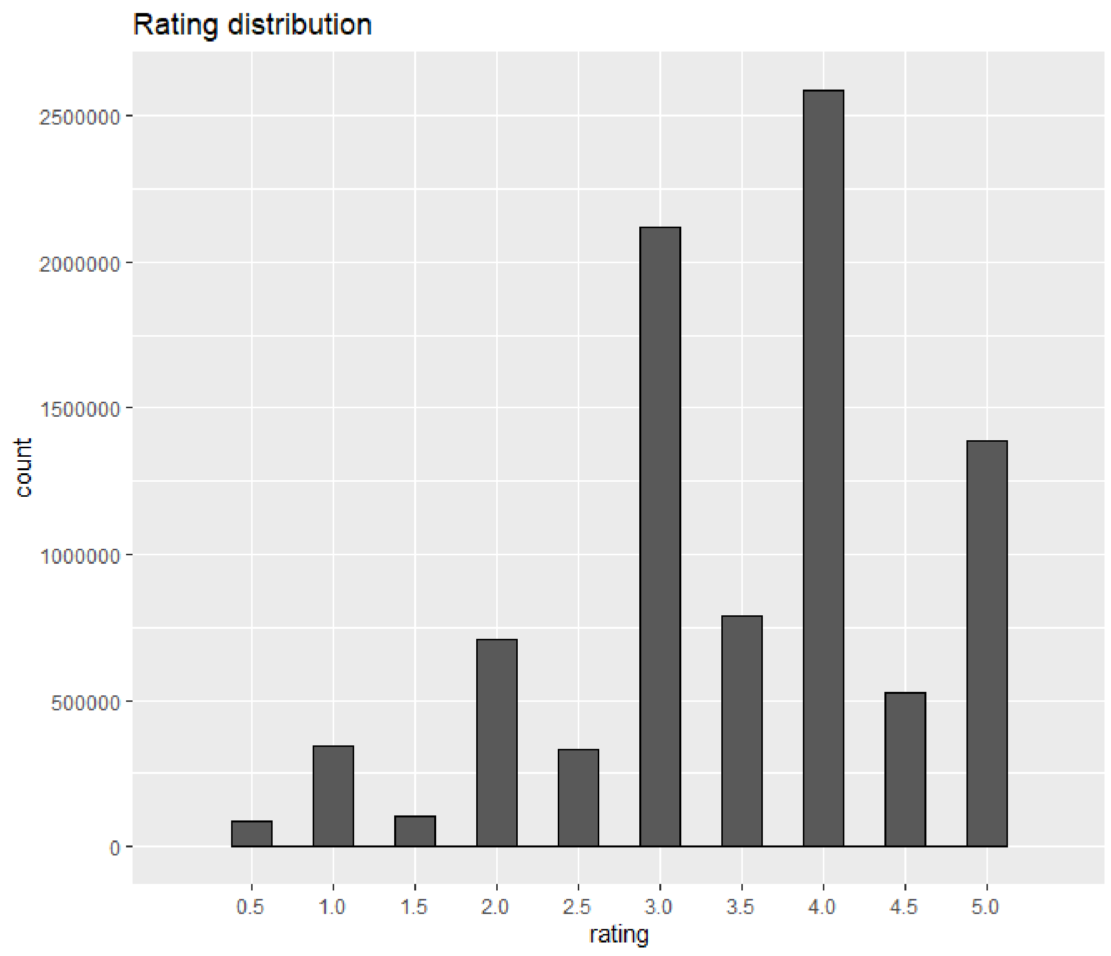
Counting Users and movies

The total of unique movies and users in the edx subset is about 70.000 unique users and about 10.700 different movies:

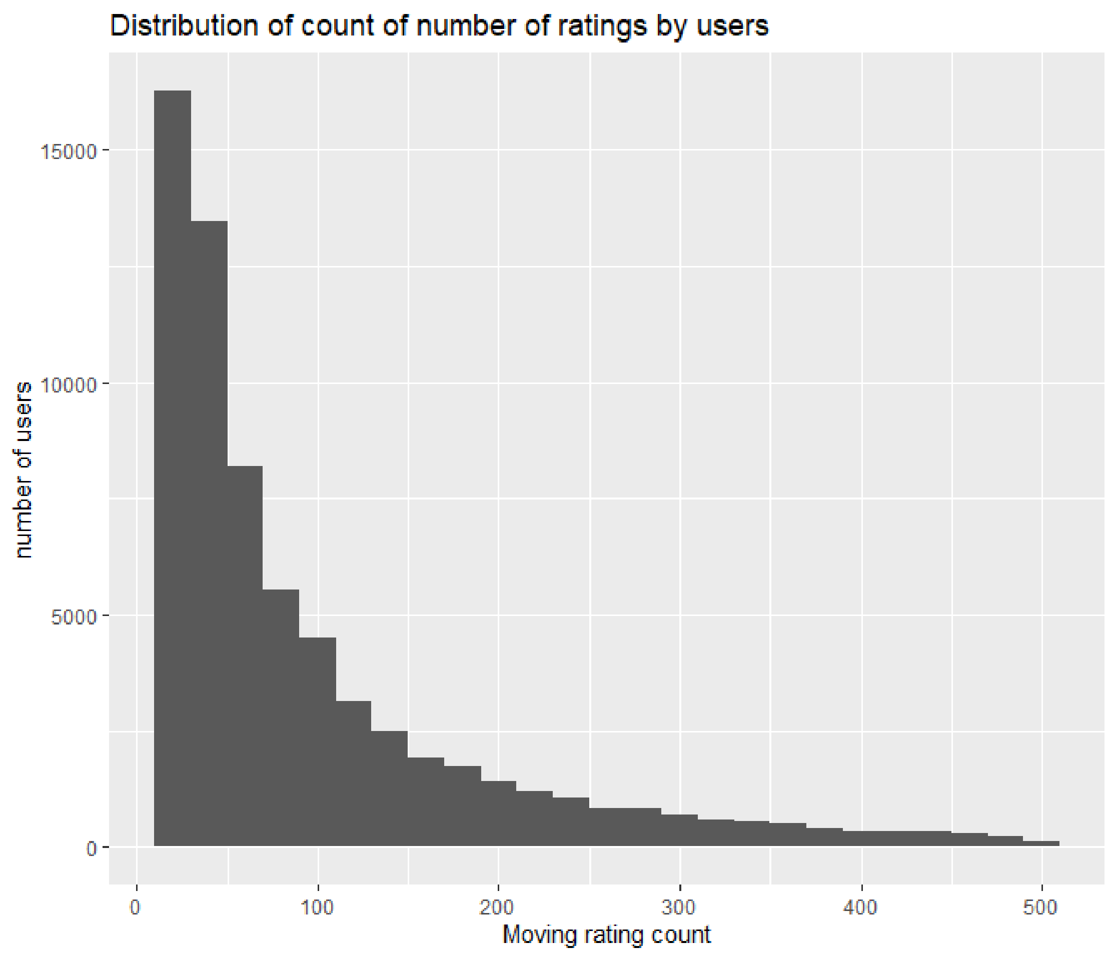
Users	Movies
69878	**10677**

Ratings distribution

Users rate movies rather higher than lower as shown by the distribution of ratings below. 4 is the most common rating, followed by 3 and 5. 0.5 is the least common rating. In general, half rating are less common than whole star ratings.



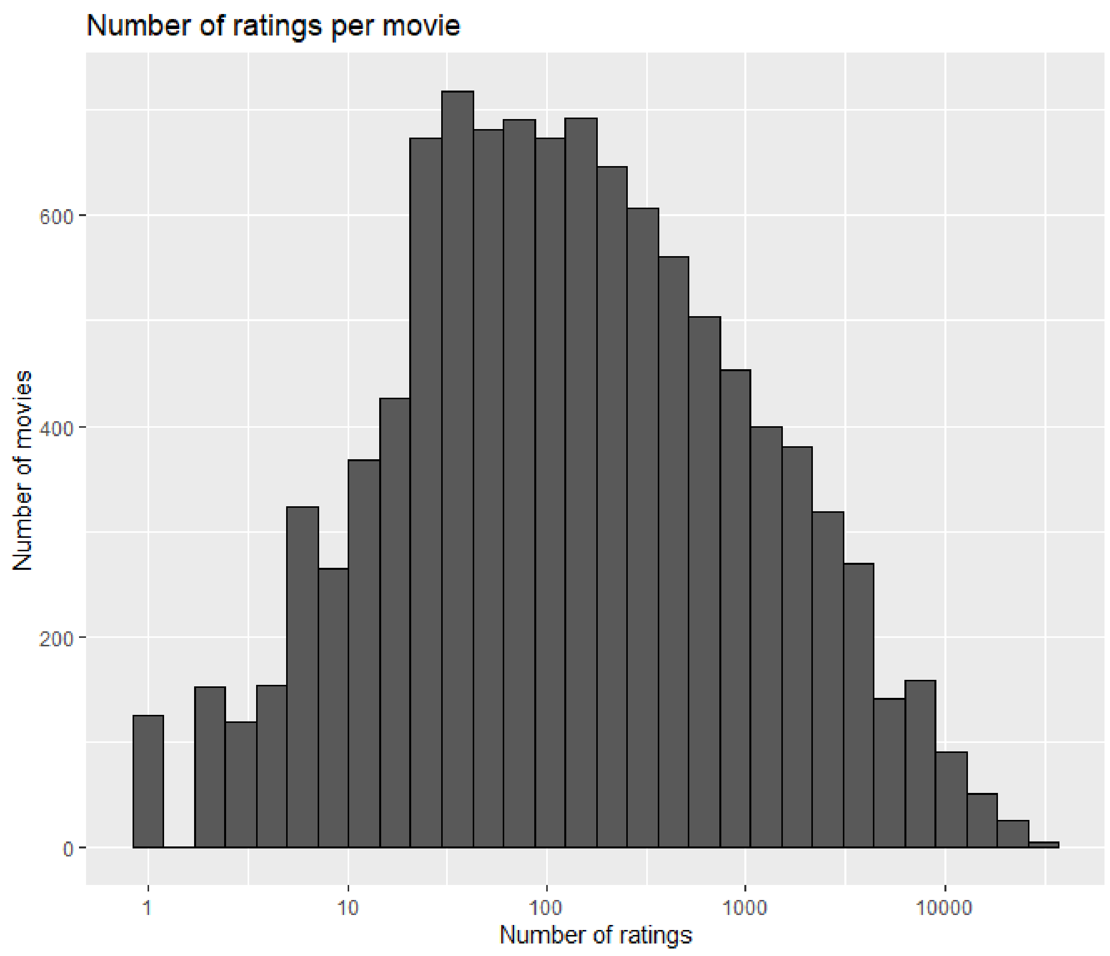
Ratings per member



Ratings per movie

The variation in the amount of films that are rated is very large. This is an important factor because films with few ratings can result in untrustworthy estimates. More than 100 movies had only one rating.

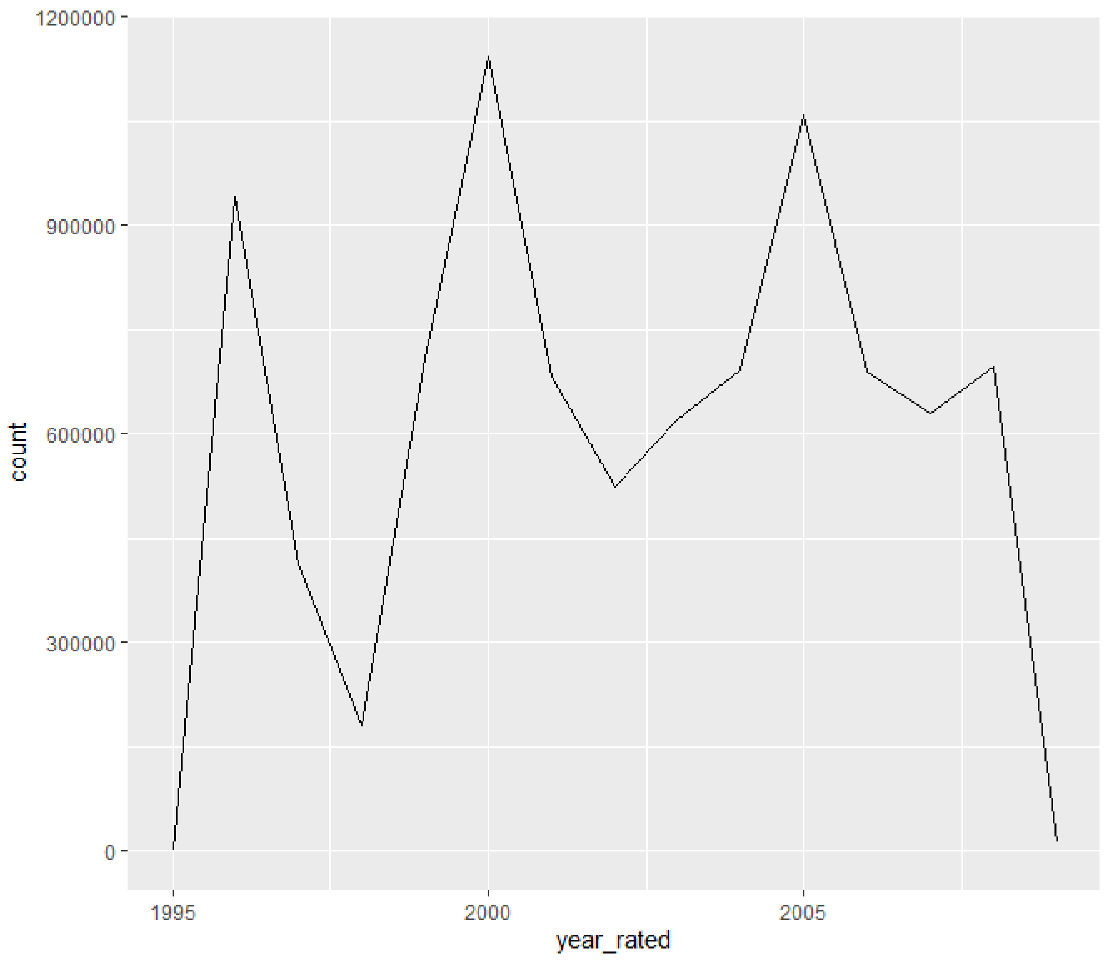
```r
edx %>%
count(movieId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "black") +
scale_x_log10() +
xlab("Number of ratings") +
  ylab("Number of movies") +
ggtitle("Number of ratings per movie")
```



Movies rated per year

A curious variation in the number of ratings per year, which did not grow linearly. The year 2000 had the highest number of ratings

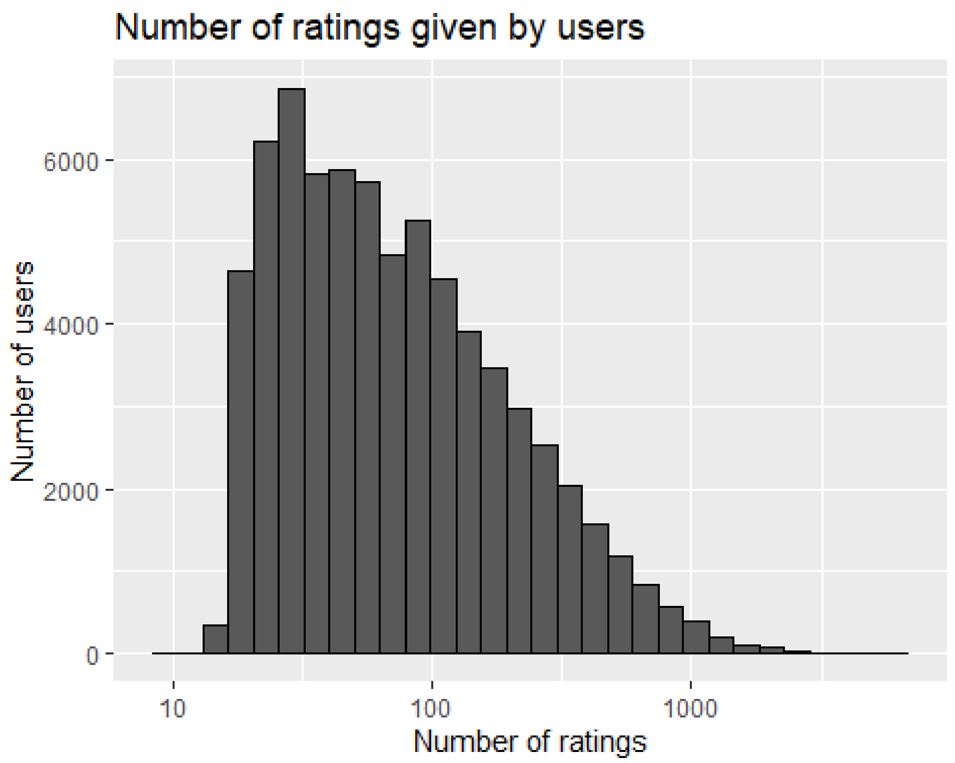
```r
movies_per_year <- edx %>%
  select(movieId, year_rated) %>% # select columns
  group_by(year_rated) %>% # group by year
  summarise(count = n()) %>%  # count movies per year
 arrange(year_rated)

movies_per_year %>%
  ggplot(aes(x = year_rated, y = count)) +
  geom_line()
```



Ratings per users

the ratings per user show the very unbalanced proportion of ratings per user

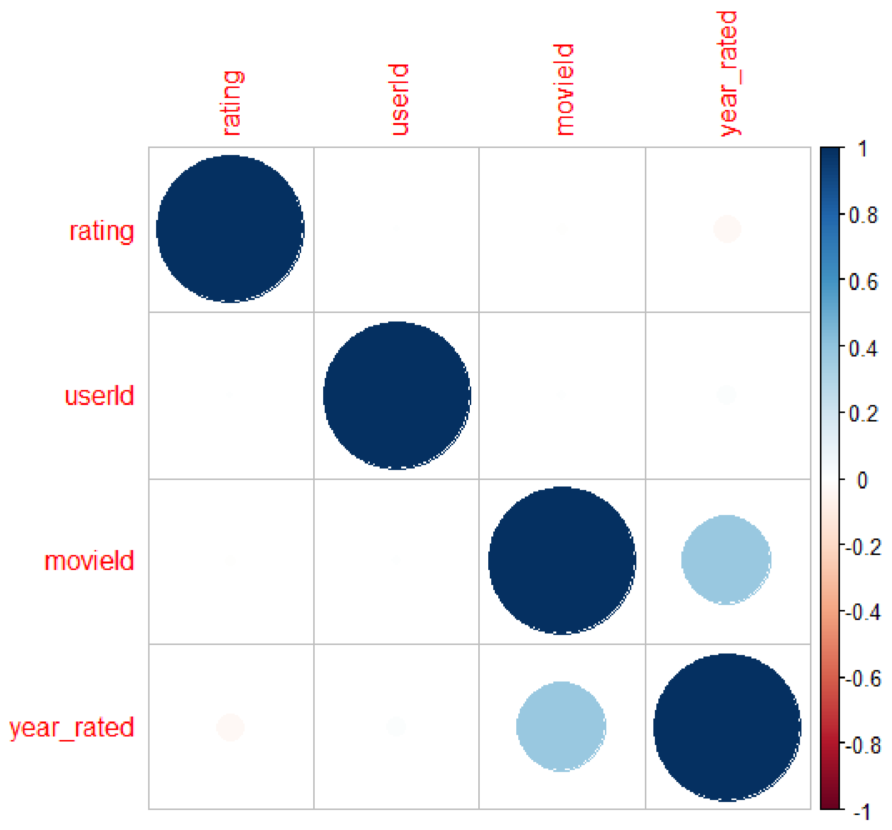
```r
edx %>%
count(userId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "black") +
scale_x_log10() +
xlab("Number of ratings") +
ylab("Number of users") +
ggtitle("Number of ratings given by users")
```



Furthermore, users differ vastly in how critical they are with their ratings. Some users tend to give much lower star ratings and some users tend to give higher star ratings than average. The visualization below includes only users that have rated at least 1000 movies.

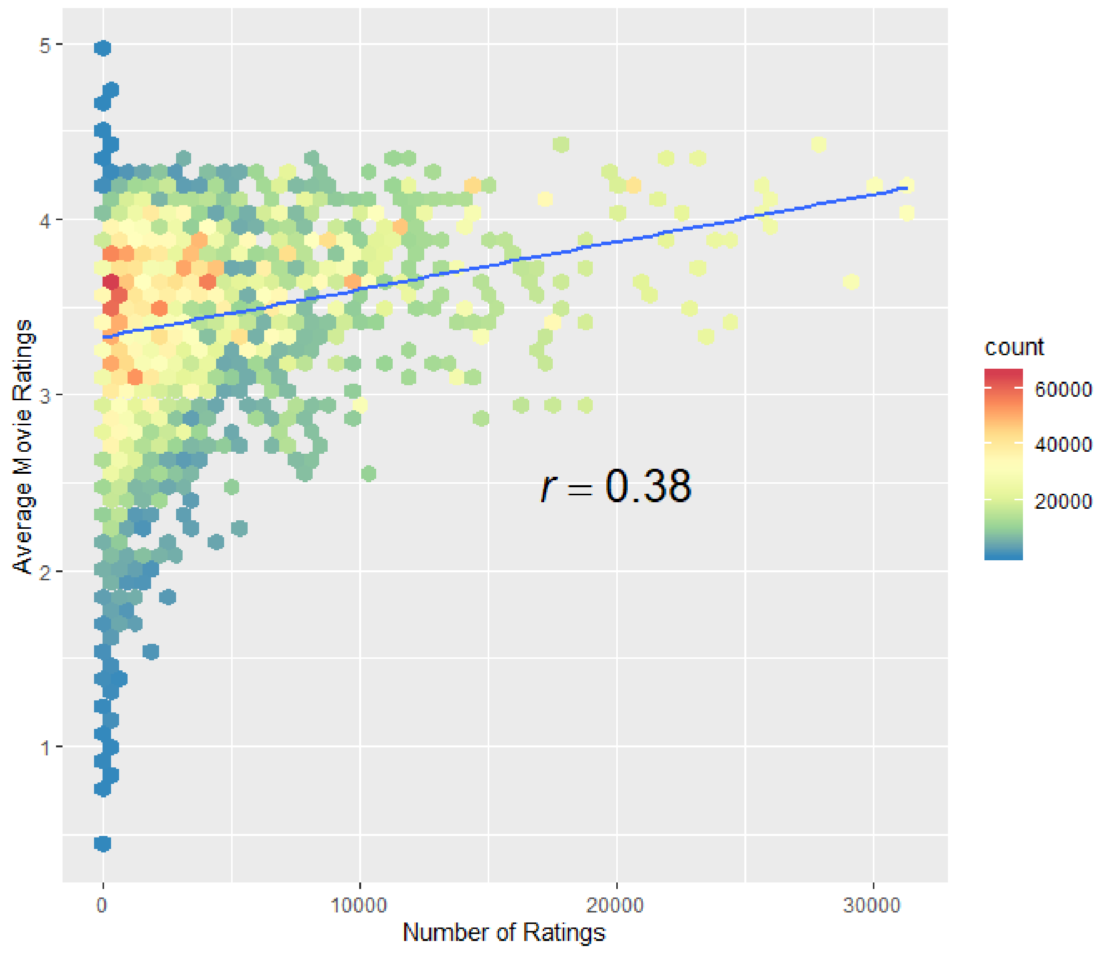
Correlations

```r
#Is there a correlation

#Number of movie ratings per movie
n_movies_ratings <- edx %>% group_by(movieId) %>% summarize(n = n())

#Average Movie Rating for each movie
avg_movie_rat <- edx %>% group_by(movieId) %>% summarize(avg_m_r = mean(ratin
g))

#Create correlation data
cor_dat <- edx %>% select(rating, movieId, userId, year_rated)  %>%
  left_join(n_movies_ratings, by = "movieId") %>%
  left_join(avg_movie_rat, by = 'movieId')
temp <- cor_dat %>% select(one_of("rating", "movieId", "userId", "year_rated"
                                               )) %>% as.matrix()
M <- cor(temp, use = "pairwise.complete.obs")

corrplot(M, method = "circle", order = "hclust")
```



Number of ratings vs avg movie ratings

```r
get_cor <- function(df){
  m <- cor(df$x, df$y);
  eq <- substitute(italic(r) == cor, list(cor = format(m, digits = 2)))
  as.character(as.expression(eq));
}

#Number of ratings vs average movie ratings
cor_dat %>%
  ggplot(aes(n, avg_m_r)) + stat_bin_hex(bins = 50) + scale_fill_distiller(pa
lette = "Spectral") +
  stat_smooth(method = "glm",  size = 1) +
  annotate("text", x = 20000, y = 2.5, label = get_cor(data.frame(x = cor_dat
$n, y = cor_dat$avg_m_r)),
           parse = TRUE, size = 7) + ylab("Average Movie Ratings") + xlab("Nu
mber of Ratings")
```



Predictive Model

The strategy is to walk from the simplest model and gradually seek improvement. Start applying several methods to build the predictive model to achieve the lowest RMSE. Then keep track of the RMSE for each method applied on a testing dataset (EDX) and report the overall RMSE into validation dataset at the end.

The approach is inspired by the 2006 Netflix challenge (https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf), based on blend the techniques of user and movie effects, regularization, and matrix factorization / Principal Components Analysis. Finally, it is to use the best Lambda and apply to the Validation Set.

I. Average movie rating model

The first basic model predicts the same rating for all movies, so we compute the dataset's mean rating. The expected rating of the underlying data set is between 3 and 4. We start by building the simplest possible recommender system by predicting the same rating for all movies regardless of user who give it. A model based approach assumes the same rating for all movie with all differences explained by random variation :

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

with $\epsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the "true" rating for all movies. This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone. We know that the estimate that minimize the RMSE is the least square estimate of $Y_{u,i}$, in this case, is the average of all ratings: The expected rating of the underlying data set is between 3 and 4.

```r
mu <- mean(edx$rating)
mu_result <- tibble(Method = "Average movie rating ",  mu)
mu_result %>% knitr::kable()
```

Method	**Average Movie Rating**
mu	**3.512465**

First, naive RMSE:

```r
model_1_rmse <- RMSE(validation$rating, mu)
```

Here, we represent results table with the first RMSE: This will be give a baseline to compare to other models

```r
rmse_results <- tibble(method = "Model I - Naive Model", RMSE = model_1_rmse)
kable(rmse_results) %>%
  kable_styling(bootstrap_options = c("striped",  full_width = F))
```

Method	RMSE
	Testing Set
Model I - Naive	1.061202

This give us our baseline RMSE to compare with next modelling approaches.

In order to do better than simply predicting the average rating, we incorporate some of insights we gained during the exploratory data analysis.

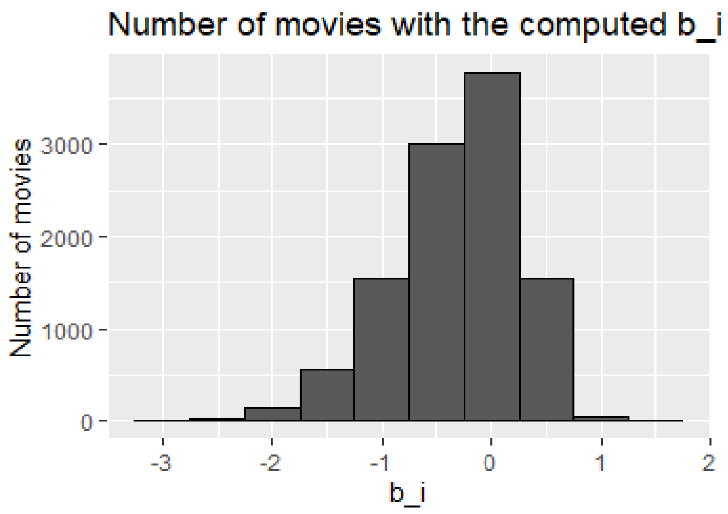
II. Movie effect model

We know from previous EDA that some movies are just generally rated higher than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of each movies' mean rating from the total mean of all movies μ. The resulting variable is called "b" (as bias) for each movie "i" b_i, that represents average ranking for movie i:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

The histogram is left skewed, implying that more movies have negative effects

```r
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("
black"),
ylab = "Number of movies", main = "Number of movies with the computed b_i")
```



Prediction improves once use this model.

```r
predicted_ratings <- mu +  validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                         tibble(method="Movie II - Effect model",
                                     RMSE = model_2_rmse ))

kable(rmse_results) %>%
  kable_styling(bootstrap_options = c("striped",  full_width = F))
```

Method	RMSE
	Testing Set
Model II- Effect model	0.9439087

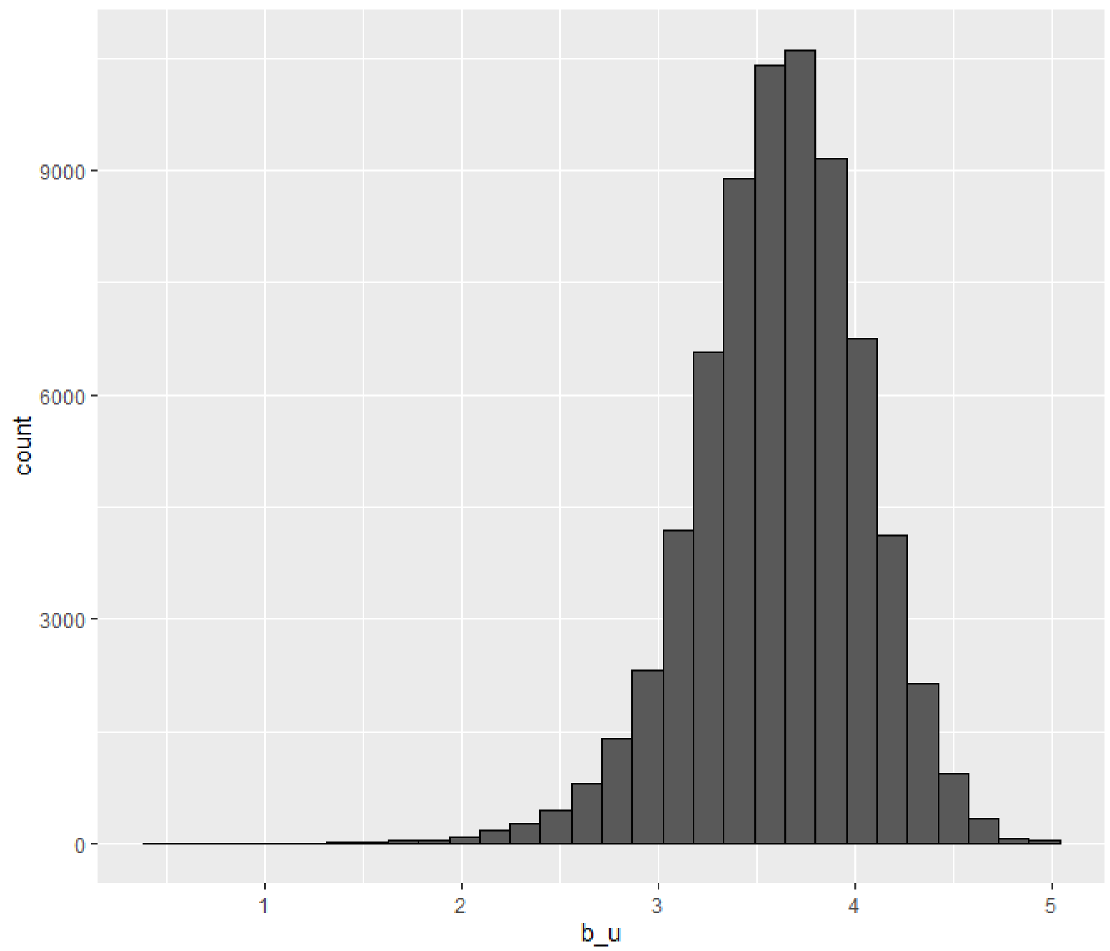
Movies are rated differently by adding the computed b_i to μ. If an individual movie is on average rated worse that the average rating of all movies μ , means that it will rated lower that μ by b_i, the difference of the individual movie average from the total average.

We can see an improvement but this model does not consider the individual user rating effect.

III. Movie and user effect model

We compute the average rating for user μ, for those that have rated over 100 movies, said penalty term user effect. In fact users affect the ratings positively or negatively.

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



There is substantial variability across users as well: some users are very cranky and other love every movie. This implies that further improvement to our model my be:

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

where b_u is a user-specific effect. If a cranky user (negative b_u rates a great movie (positive b_i), the effects counter each other and we may be able to correctly predict that this user gave this great movie a 3 rather than a 5.

We compute an approximation by computing μ and b_i, and estimating b_u, as the average of

$$Y_{u,i} - \mu - b_i$$

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

We can now construct predictors and see RMSE improves:

```
predicted_ratings <- validation%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_3_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model III - Movie and user effec
t model",
                                     RMSE = model_3_rmse))

## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.

kable(rmse_results) %>%
  kable_styling(bootstrap_options = c("striped",  full_width = F))
```

Method	RMSE
	Testing Set
Model III- Movie and user effect model	0.8653488

Our rating predictions further reduced the RMSE. But we made stil mistakes on our first model (using only movies). The supposes "best" and "worst"movie were rated by few users, in most cases just one user. These movies were mostly obscure ones. This is because with a few users, we have more uncertainty. Therefore larger estimates of b_i, negative or positive, are more likely. Large errors can increase our RMSE.

Until now, we computed standard error and constructed confidence intervals to account for different levels of uncertainty. However, when making predictions, we need one number, one prediction, not an interval. For this we introduce the concept of regularization, that permits to penalize large estimates that come from small sample sizes. The general idea is to add a penalty for large values of b_i to the sum of squares equation that we minimize. So having many large b_i, make it harder to minimize. Regularization is a method used to reduce the effect of overfitting.

IV. Regularized movie and user effect model

So estimates of b_i and b_u are caused by movies with very few ratings and in some users that only rated a very small number of movies. Hence this can strongly influence the prediction. The use of the regularization permits to penalize these aspects. We should find the value of lambda (that is a tuning parameter) that will minimize the RMSE. This shrinks the b_i and b_u in case of small number of ratings.

The general idea of penalized regression is to control the total variability of the movie effects. Specifically, instead of minimizing the least square equation, it minimize an equation that adds a penalty:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i\right)^2+\lambda\sum_i b_i^2$$

The first term is just least squares and the second is a penalty that gets larger when many b_i are large. Using calculus it it is possible to show that the values of b_i that minimize this equation are:

$$\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}\left(Y_{u,i}-\hat{\mu}\right)$$

Use Regularization for the estimate user effects and use cross-validation to choose λ as a tuning parameter.

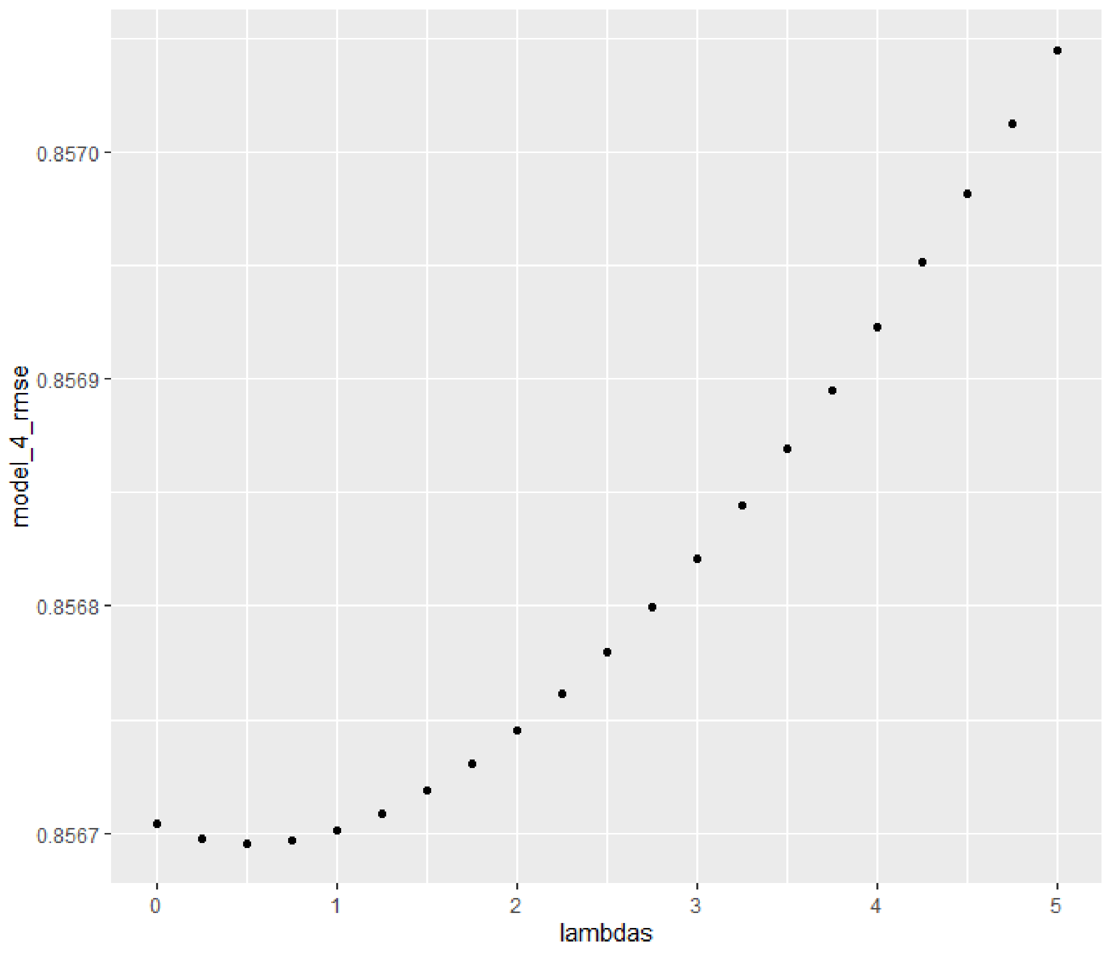
$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

The Respective r code:

```
lambdas <- seq(0, 5, 0.25)
model_4_rmse <- sapply(lambdas, function(l){

  mu <- mean(edx$rating)

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

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

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

We plot RMSE vs lambdas to select the optimal lambda Note that λ is a tuning parameter that apply to the training set (EDX).

```
qplot(lambdas, model_4_rmse)
```



The optimal λ is:

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

## [1] 0.5
```

The new RMSE results will be:

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model  IV - Regularized movie an
d user effect model",
                                     RMSE = min(model_4_rmse)))
kable(rmse_results) %>%
  kable_styling(bootstrap_options = c("striped",  full_width = F))
```

Method	RMSE
	Testing Set
Model IV- Regularized movie and user effect model	0.8566952

V - Apply Model to Validation Set

Finally, apply the final model to the validation set using the best λ obtained from previous test and see the result

```
mu <- mean(validation$rating)
l <- best_lambda

b_i <- validation %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))

b_u <- validation %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n() +l))

predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i +  b_u) %>% .$pred
RMSE(predicted_ratings, validation$rating)

## [1] 0.8258487
```

Results

The RMSE values of all the represented models from Testing and Validation Set EDX are the following:

Method	RMSE
	Testing Set
Model I - Naive	1.061202
Model II- Effect model	0.9439087
Model III - Movie and user effect model	0.8653488
Model IV - Regularized movie and user effect model	0.8566952
	Validation Set
Model IV - Regularized movie and user effect	0.824567
BEST RMSE OVERALL	0.824567

Note: The Validation Set was only used at the final step (V)

Conclusion

The model with regularization, including the user effect, obtained the lowest value of RMSE and was, therefore, the the best model in the scope of this project.

The results obtained by this project in order to create a predictive model for recommendation systems has fully achieved. The RMSE of 0.8566952 on the test dataset and RMSE 0.8258487 on the validation dataset exceeded the best evaluation criteria established by grader, RMSE <= 0.8649.

The training and validation set RMSE gap suggests that models used in this project still have space to improvements. Other techniques may be doing even better, especially Matrix factorization methods were probably the most important class of techniques for winning the Netflix Prize. In their 2008 Progress Prize paper, Bell and Koren write:

It seems that models based on matrix-factorization were found to be most accurate (and thus popular), as evident by recent publications and discussions on the Netflix Prize forum. We definitely agree to that, and would like to add that those matrix-factorization models also offer the important flexibility needed for modeling temporal effects and the binary view. Nonetheless, neighborhood models, which have been dominating most of the collaborative filtering literature, are still expected to be popular due to their practical characteristics - being able to handle new users/ratings without re-training and offering direct explanations to the recommendations.

References

Hadley W., & GGrolemund G. (2017). R for Data Science. ISBN-10: 1491910399

RA Irizarry. (2019) Introduction to Data Science Data Analysis and Prediction Algorithms with R

R. Bell, Y. Koren and C. Volinsky. The BellKor 2008 Solution to the Netflix Prize. 2008

Y. Koren, “Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model,” Proc. 14th ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining, ACM Press, 2008, pp. 426-434.

Y.F. Hu, Y. Koren, and C. Volinsky, “Collaborative Filtering for Implicit Feedback Datasets,” Proc. IEEE Int’l Conf. Data Mining (ICDM 08), IEEE CS Press, 2008, pp. 263-272.

R. Bell, Y. Koren and C. Volinsky. The BellKor 2008 Solution to the Netflix Prize. 2008

Several websites, blogs, tutorials, and excerpts of codes related to recommendation systems and Movielens were also consulted and used in this work. Especially kaggle.com, towardsdatascience.com, and stackoverflow.com.

Environment

```
print("Operating System:")

## [1] "Operating System:"

version

##                _                           
## platform       x86_64-w64-mingw32          
## arch           x86_64                      
## os             mingw32                     
## system         x86_64, mingw32             
## status                                     
## major          3                           
## minor          6.1                         
## year           2019                        
## month          07                          
## day            05                          
## svn rev        76782                       
## language       R                           
## version.string R version 3.6.1 (2019-07-05)
## nickname       Action of the Toes          
```