HarvardX: PH125.9x Professional Certificate in Data Science - Capstone MovieLens Project

Stefan Prangenberg

6/8/2019

The submission for the MovieLens project will be three files: a report in the form of an Rmd file, a report in the form of a PDF document knit from your Rmd file, and an R script or Rmd file that generates your predicted movie ratings and calculates RMSE.

Table of Contents

1. Introduction

The goal of this exercise is to leverage movie ratings and machine learning to create movie recommendations.

We create a movie recommendation system using the MovieLens data set. The version of movielens included in the dslabs package is just a small subset of a much larger data set with millions of ratings. We will create a recommendation system using the 10M version of the MovieLens data set.

2. Dataset

The GroupLens research lab (https://grouplens.org (https://grouplens.org)) created the MovieLens data set which contains over 20 million ratings by more than 138.000 users for over 27,000 movies. For this exercise we use a subset of 10 million records to analyse and generate our report. The 'MovieLens' data set with 10 million records can be accessed at:
• [MovieLens 10M data set] https://grouplens.org/datasets/movielens/10m/
(https://grouplens.org/datasets/movielens/10m/)

```r
# Create edx set and validation set
if (file.exists("edxData.Rda")) {
  print("Loading Data from File")
  load("edxData.Rda")
    if (!require(tidyverse))
    install.packages("tidyverse", repos = "http://cran.us.r-project.org")
  if (!require(caret))
    install.packages("caret", repos = "http://cran.us.r-project.org")

} else {
  # Note: this process could take a couple of minutes
  print("Loading Data from Source")
  if (!require(tidyverse))
    install.packages("tidyverse", repos = "http://cran.us.r-project.org")
  if (!require(caret))
    install.packages("caret", repos = "http://cran.us.r-project.org")

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

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

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

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

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

  # Validation set will be 10% of MovieLens data
  set.seed(1) # if using R 3.6.0: set.seed(1, sample.kind = "Rounding")
  test_index <-
    createDataPartition(
      y = movielens$rating,
      times = 1,
      p = 0.1,
      list = FALSE
    )
  edx <- movielens[-test_index, ]
  temp <- movielens[test_index, ]

  # Make sure userId and movieId in validation set are also in edx set
  validation <- temp %>%
    semi_join(edx, by = "movieId") %>%
    semi_join(edx, by = "userId")

  # Add rows removed from validation set back into edx set
  removed <- anti_join(temp, validation)
  edx <- rbind(edx, removed)

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

  save(edx, validation, file = "edxData.Rda")
}
```

```
## [1] "Loading Data from File"
```

3. Analysis

3.1 Data Cleaning

Let’s take a first look at the data:

```
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
```

The data contains the userID, movieID, rating, timestamp, title (incl. year) and a list of genres.

To be able to analyze the year, we need to extract it from the title string:

```
# Extract the Year from the Title in both data sets

edx <-
  edx %>% extract(title,
                  c("title_name", "year"),
                  regex = "^(.*) \\(([0-9 \\-]*)\\)$",
                  remove = T) %>% mutate(year = as.integer(year))
head(edx)
```

```
##   userId movieId rating timestamp             title_name year
## 1      1     122      5 838985046              Boomerang 1992
## 2      1     185      5 838983525               Net, The 1995
## 3      1     292      5 838983421               Outbreak 1995
## 4      1     316      5 838983392               Stargate 1994
## 5      1     329      5 838983392 Star Trek: Generations 1994
## 6      1     355      5 838984474       Flintstones, The 1994
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 3  Action|Drama|Sci-Fi|Thriller
## 4        Action|Adventure|Sci-Fi
## 5 Action|Adventure|Drama|Sci-Fi
## 6        Children|Comedy|Fantasy
```

```
validation <-
  validation %>% extract(title,
                         c("title_name", "year"),
                         regex = "^(.*) \\(([0-9 \\-]*)\\)$",
                         remove = T) %>% mutate(year = as.integer(year))
```

We can now use the year as an integer.

The genre field can contain multiple values. To analyze it, we need to separate the string into multiple values. We create a line for each genre listed. To not falsify the overall stats, we save the new data as a separate variable.

```
# Split the groupings of genres into individual lines
edx_genres_split  <-  edx  %>% separate_rows(genres, sep = "\\|")
head(edx_genres_split)
```

```
##   userId movieId rating timestamp title_name year   genres
## 1      1     122      5 838985046  Boomerang 1992   Comedy
## 2      1     122      5 838985046  Boomerang 1992  Romance
## 3      1     185      5 838983525   Net, The 1995   Action
## 4      1     185      5 838983525   Net, The 1995    Crime
## 5      1     185      5 838983525   Net, The 1995 Thriller
## 6      1     292      5 838983421   Outbreak 1995   Action
```

Now the data is ready to be explored.

3.2 Data Exploration and Visualization

The data contains ratings from 69,878 users for 10,677 movies.

```
#Count number of distinct users and movies
edx %>%   summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

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

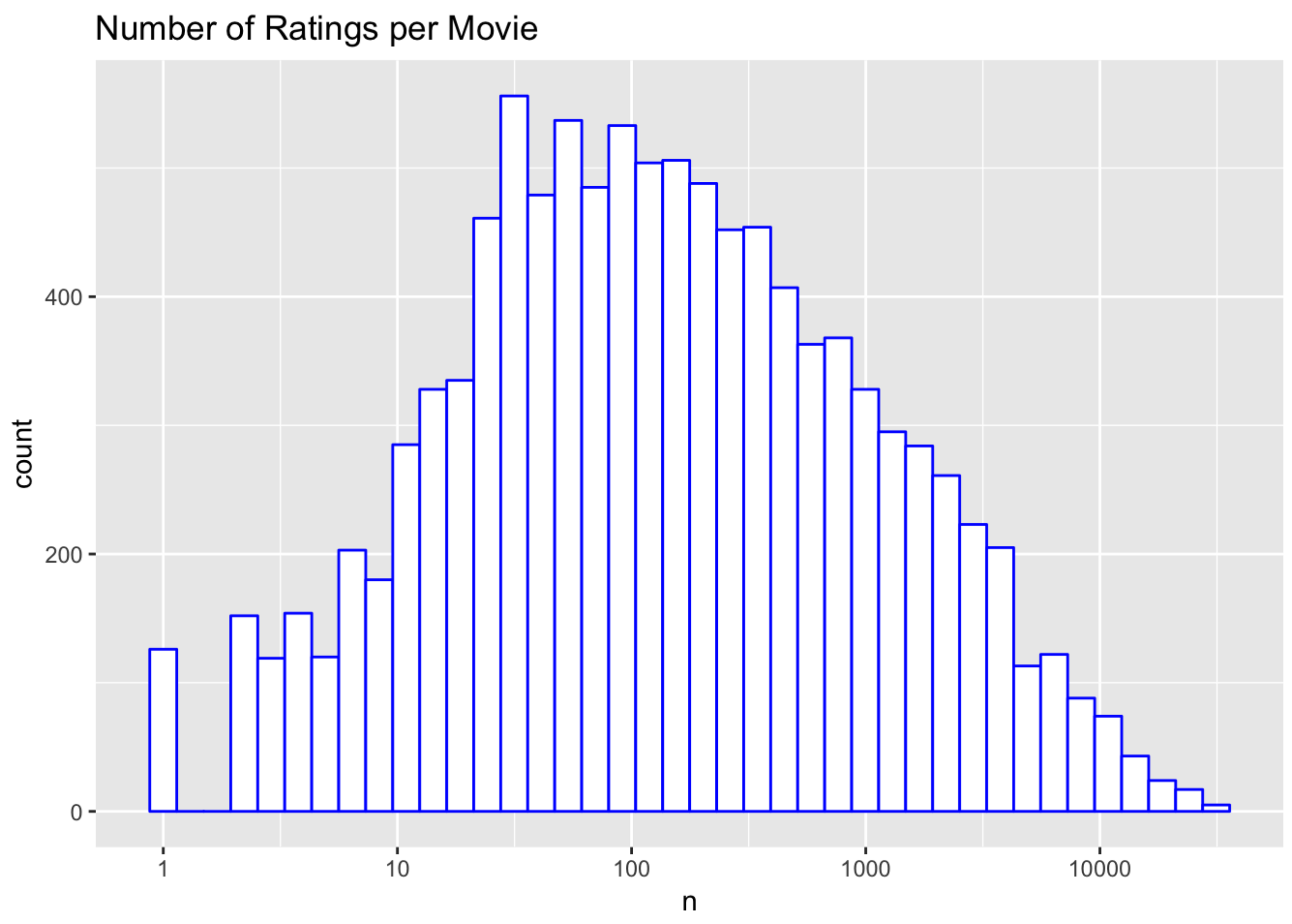
```
summary(edx)
```

```
##      userId         movieId          rating        timestamp        
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08  
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08  
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09  
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09  
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09  
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09  
##   title_name             year         genres         
##  Length:9000055     Min.   :1915   Length:9000055    
##  Class :character   1st Qu.:1987   Class :character  
##  Mode  :character   Median :1994   Mode  :character  
##                     Mean   :1990                     
##                     3rd Qu.:1998                     
##                     Max.   :2008                     
```

The ratings start at 0.5 (there is no rating of 0) and increment in steps of 0.5 up to 5.0. The oldest movie in the data is from 1915 and the most recent from 2008.

Number of Ratings per Movie

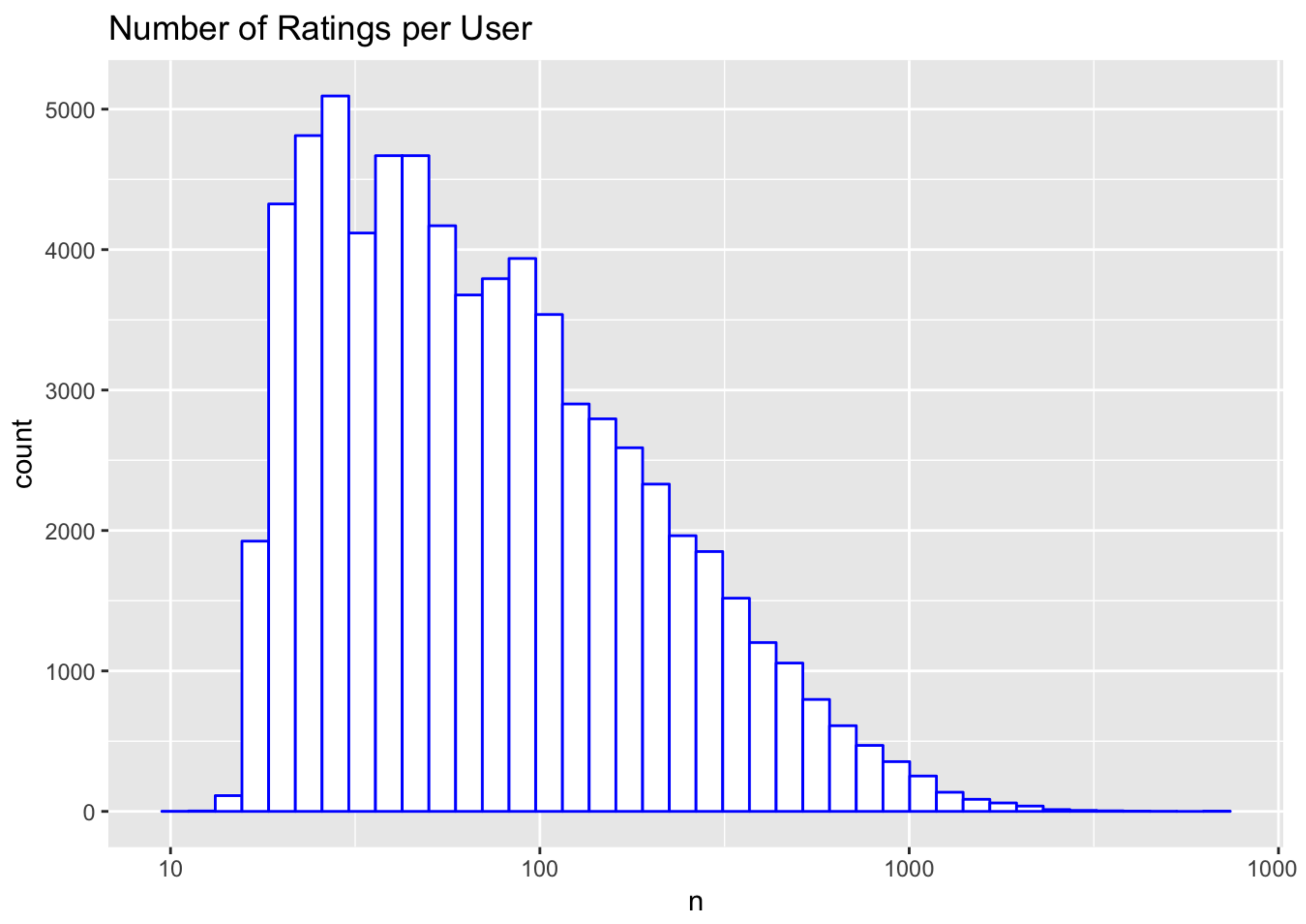
```
#Plot Number of Ratings per Movie
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 40, color = "blue", fill="white") +
  scale_x_log10() +
  ggtitle("Number of Ratings per Movie")
```



Each movie have anywhere between 1 and 10,000+ ratings. A few hundred movies only have 10 or less ratings, which will make predictions for them challenging. We can be more confident with predictions for movies that have hundreds or even thousands of ratings.

Number of Ratings per User

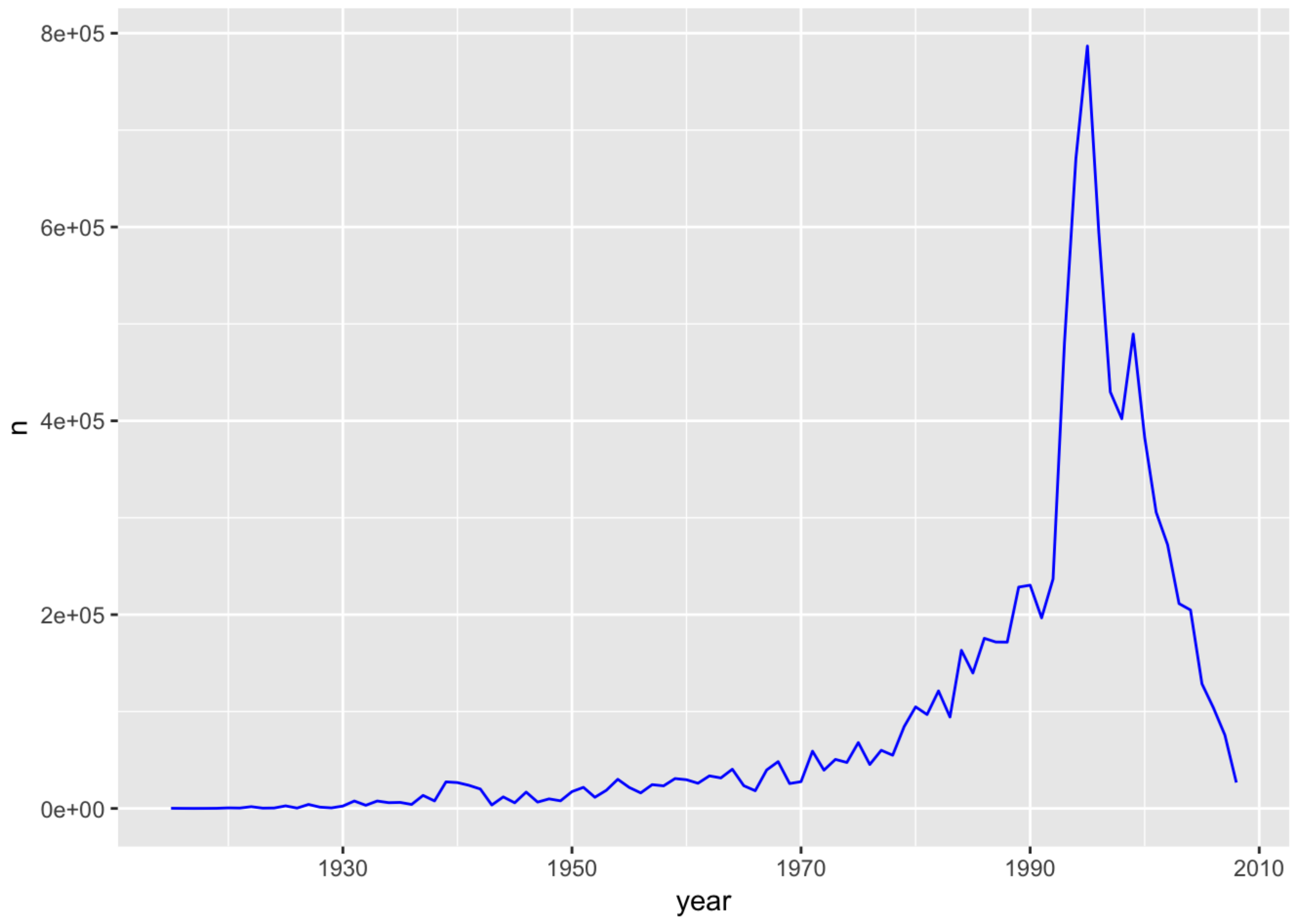
```r
#Plot Number of Ratings per user
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 40, color = "blue",fill="white") +
  scale_x_log10() +
  ggtitle("Number of Ratings per User")
```



A few users have rated more than 1,000 movies. Similar to the movies, there is a large amount of users that have only rated a few movies. Predictions for those users will be challenging as well.

Number of Movies per Year

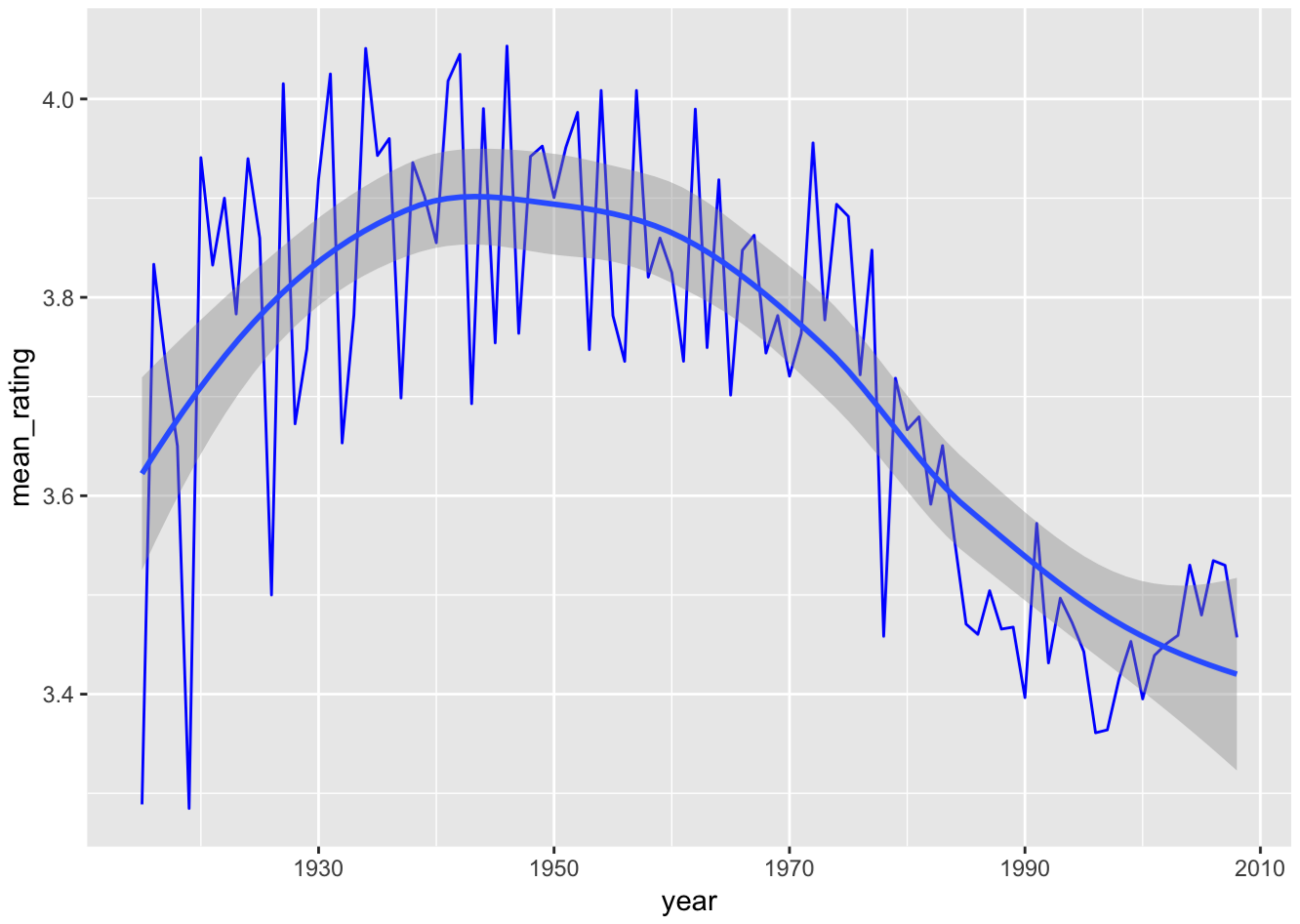
```
#Movies per year
edx %>%   count(year) %>%  ggplot (aes(x = year, y = n)) + geom_line(color ="blue")
```



The vast majority of movies were created between 1990 and 1995. Only very few movies date back earlier than the 1950s.

Average rating per year

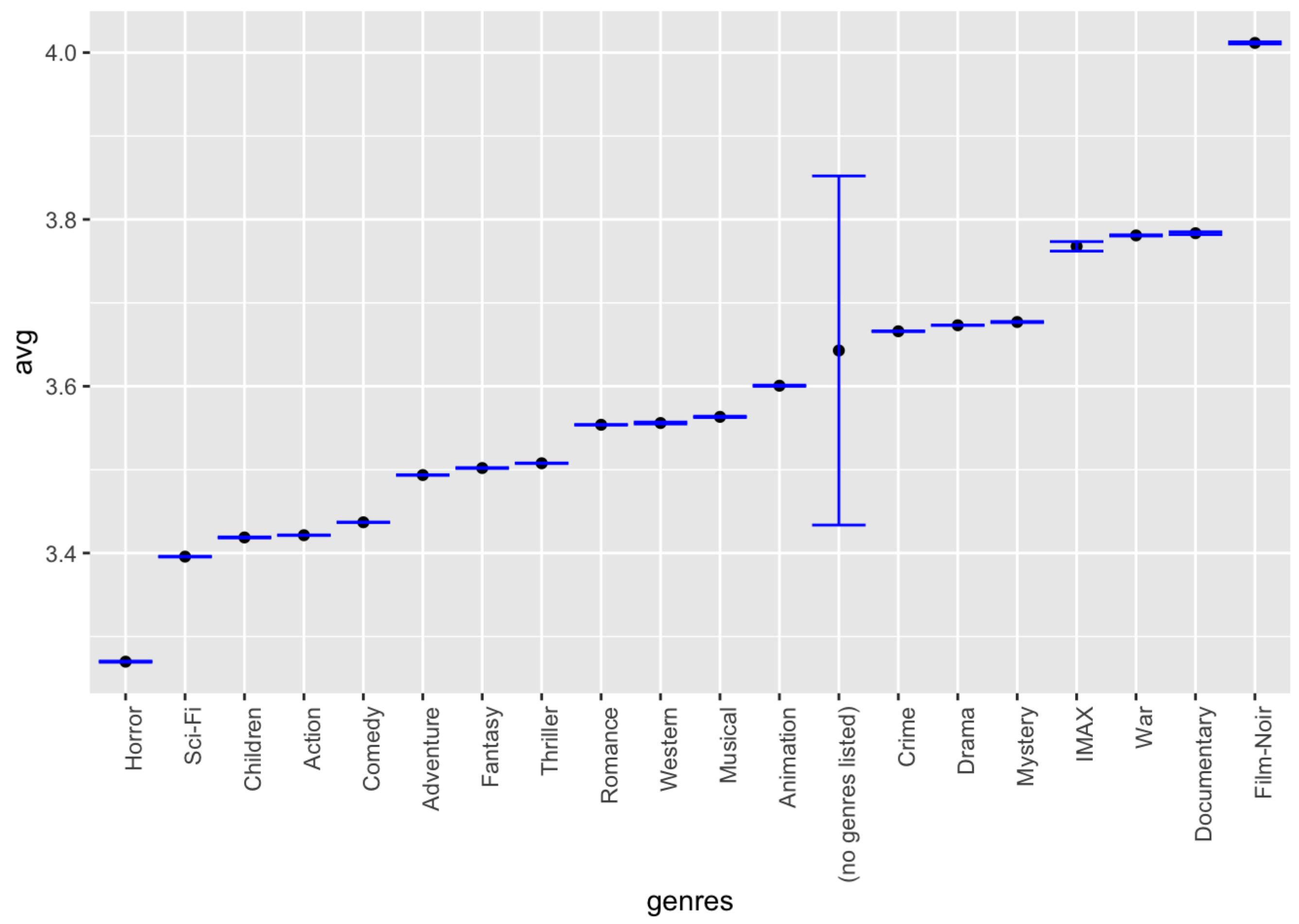
```
#Average Rating per year
edx %>% group_by(year) %>% summarize(mean_rating = mean(rating)) %>% ggplot(aes(x =year,
y = mean_rating)) + geom_line(color = "blue") +   geom_smooth()
```



Older movies seem to be rated higher, with a peak around 1950. Newer movies created after 1990 (which is the majority as we saw earlier) tend to have lower ratings.

Average rating per genre

```
#Plot average Ratings per genre - sorted from lowest to highest
edx_genres_split %>% group_by(genres) %>%
  summarize(n = n(),
            avg = mean(rating),
            se = sd(rating) / sqrt(n())) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(
    x = genres,
    y = avg,
    ymin = avg - 0.5 * se,
    ymax = avg + 0.5 * se
  )) +
  geom_point() +
  geom_errorbar(color="blue") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Different genres seem to enjoy different popularity with the users. Horror movies average the lowest rating (3.3). The highest ratings go to IMAX, War, Documentary and Film-Noir (3.8-4). There is a lot of variability in movies that are not assigned to a genre.

3.3 Modeling Approach

Our goal is to reach an Residual Mean Squared Error (RMSE) of <= 0.87750.

This function will be used to determine the RMSE for each model:

```
### Funtion to Calculate RMSE for true ratings + predicted ratings
RMSE <- function(true_ratings, predicted_ratings) {
  sqrt(mean((true_ratings - predicted_ratings) ^ 2))
}
```

Model 1 - Average rating for all movies

In this first (very basic) model, we assume that each movie that we are predicting will have the average rating of all the movies in the edx set.

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

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

The average rating across all movies is around 3.5. Using this to predict all movies in the validation set, we get this RMSE:

```
model_1_rmse <- RMSE(validation$rating, avg)
model_1_rmse
```

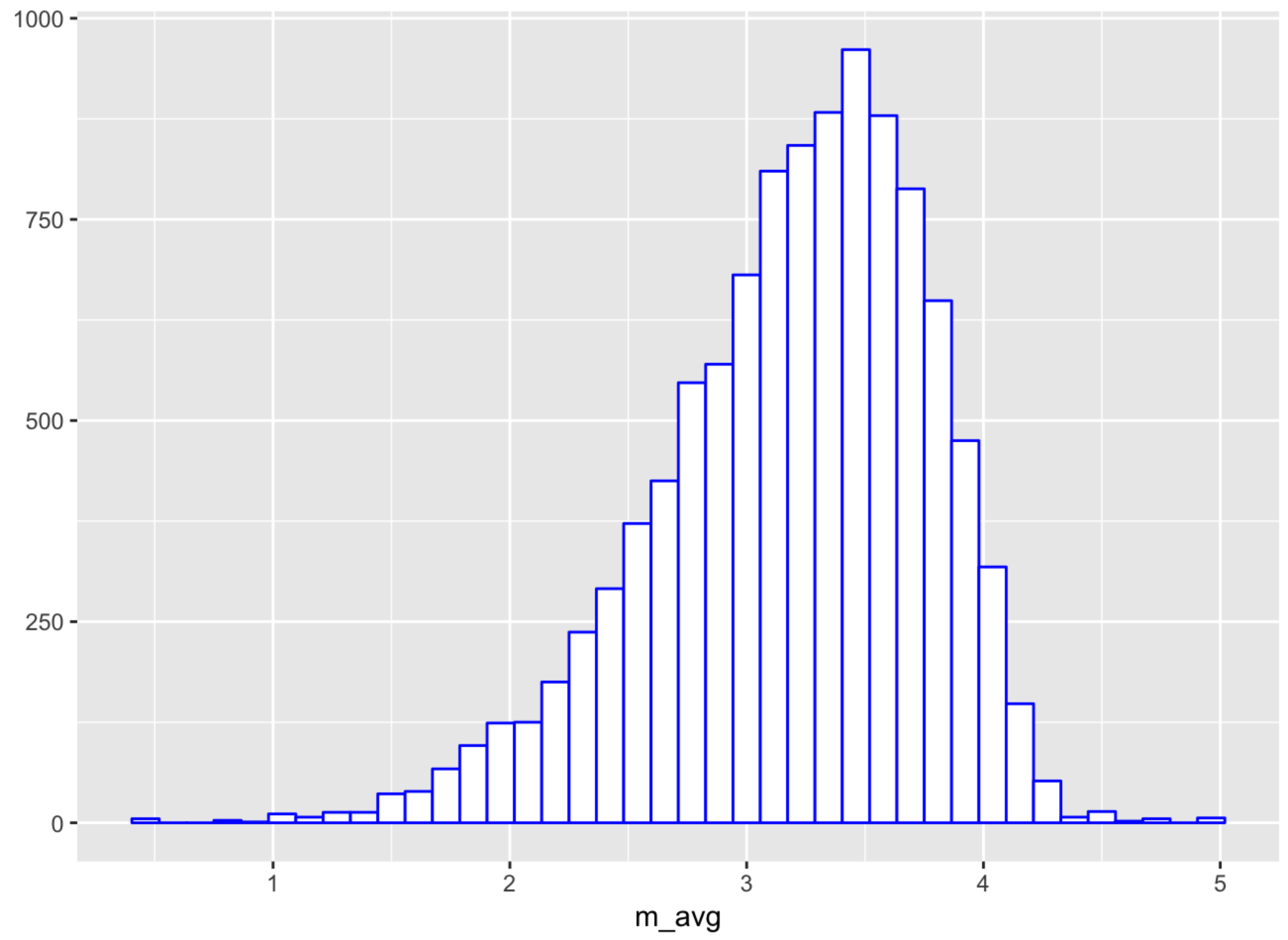
```
## [1] 1.061202
```

This is far from our target, but will be the baseline for our next models.

Model 2 - Movie Effect

This model assumes each movie in the validation set will have the same ratings than the average of all ratings the movie has in the edx set. First, we determine the average rating for each movie.

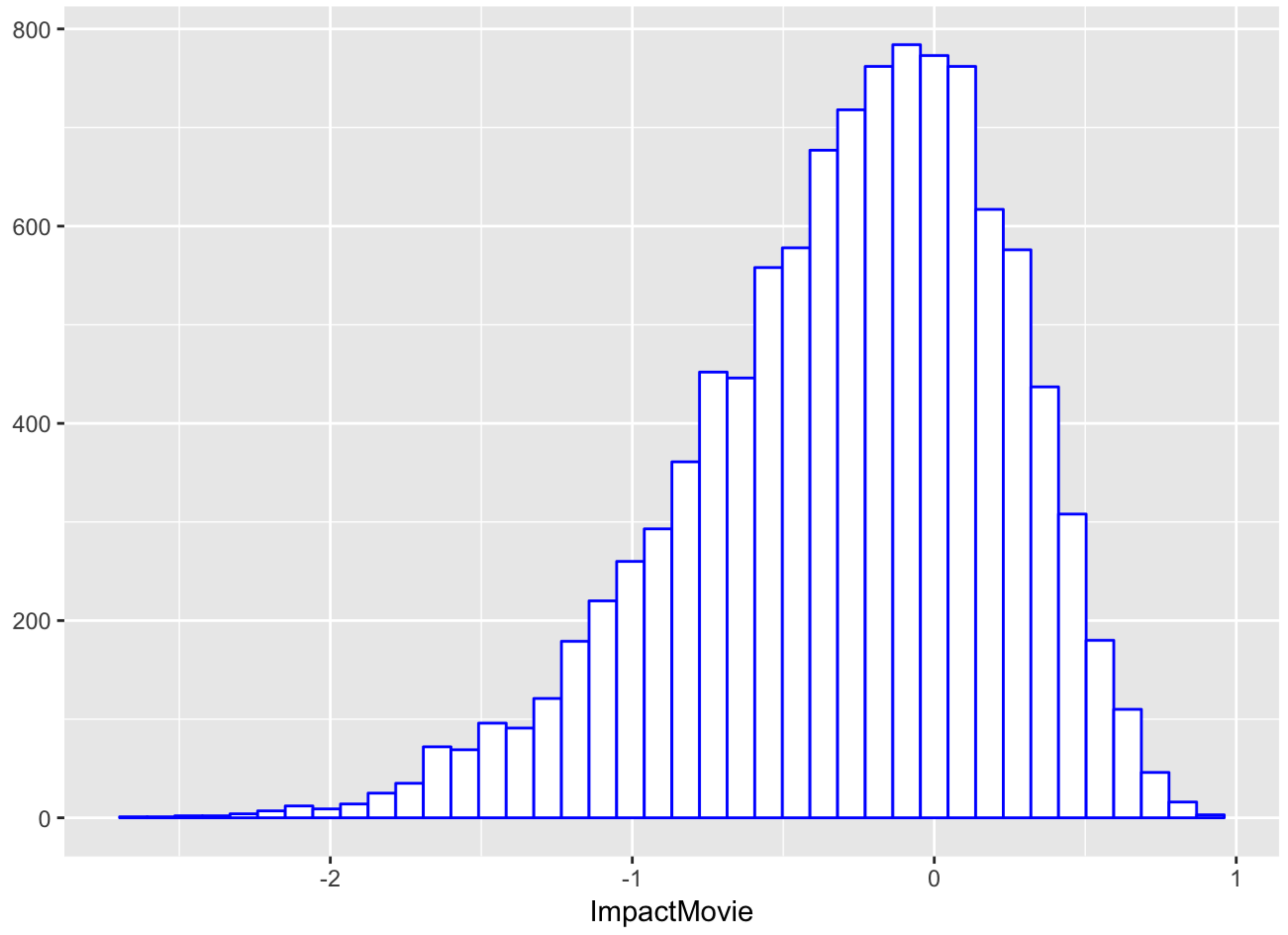
```
# Determine mean rating for each individual movie
movie_avgs <-
  edx %>%   group_by(movieId) %>%   summarize(m_avg = mean(rating))
# Plot distribution of mean ratings for all movies
movie_avgs %>% qplot(
  m_avg,
  geom = "histogram",
  bins = 40,
  data = .,
  color = I("blue"),
  fill =  I("white")
)
```



It’s interesting to note that this is not a regular bell curve, but the left side is wider. This makes sense, since the average (3.51) is higher than the middle of the scale (which would be 2.5).

We can now determine how “different” the average of each movie is from the average of all movies.

```
# What's the impact per MOVIE compared to the overall Mean?
ImpactMovie <-
  edx %>%   group_by(movieId) %>%  summarize(ImpactMovie = sum(rating - avg) /
                                                (n() + 2))
# Plot Impact per Movie (compared to overall mean)
ImpactMovie %>% qplot(
  ImpactMovie,
  geom = "histogram",
  bins = 40,
  data = .,
  color = I("blue"),
  fill =  I("white")
)
```



This plot is consistent with our earlier observation, that there are more movies with a negative impact on the (fairly high) overall average rating.

We can now use this information to create our second model, which assumes that movies tend to have the same average rating across all data sets.

```
prediction_Movie <-
  validation %>%   left_join(ImpactMovie, by = 'movieId') %>%  mutate(pred = avg + Impac
tMovie)
model_2_rmse <- RMSE(validation$rating, prediction_Movie$pred)
model_2_rmse
```

```
## [1] 0.9438528
```

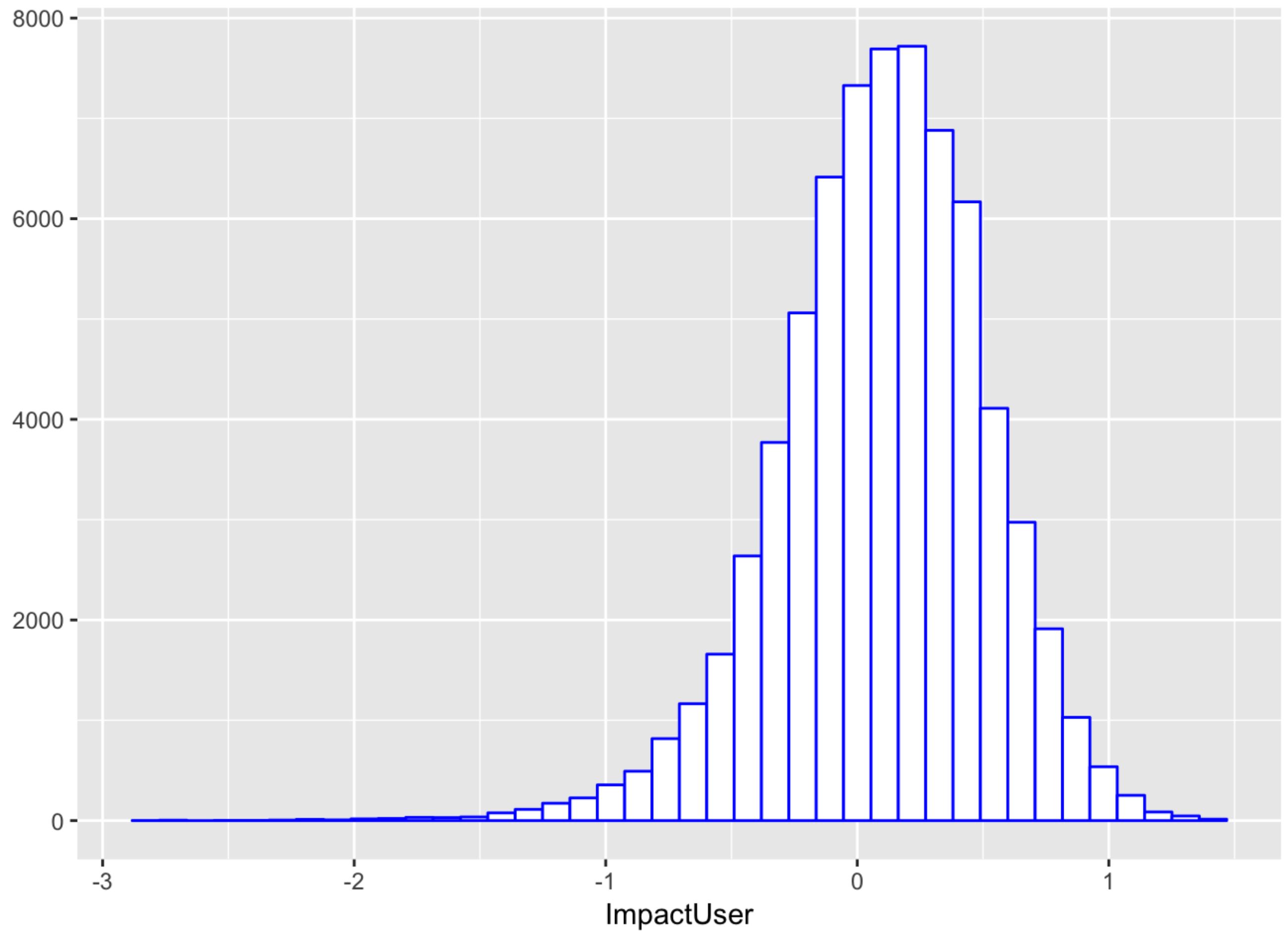
This RMSE is much better than our baseline of 1.06. But still some way to go 0.87.

Next. let’s look at our users.

Model 3 - User Effect

Similar to the movies above, let’s now look at the impact of users.

```
# What's the impact per USER compared to the overall Mean?
ImpactUser <-
  edx %>%   group_by(userId) %>%  summarize(ImpactUser = sum(rating - avg) /
                                              (n() + 2))
# Plot Impact per USER (compared to overall mean)
ImpactUser %>% qplot(
  ImpactUser, geom = "histogram", bins = 40, data = ., color = I("blue"),fill =  I("whit
e")
)
```



This looks more like a regular Gaussian bell curve, except that it stretches further to the left. A few users seem to give a lot of very negative reviews..

Let’s look at the prediction that leverages the user effect:

```
prediction_User <-
  validation %>%   left_join(ImpactUser, by = 'userId') %>%  mutate(pred = avg + ImpactU
ser)
model_3_rmse <- RMSE(validation$rating, prediction_User$pred)
model_3_rmse
```

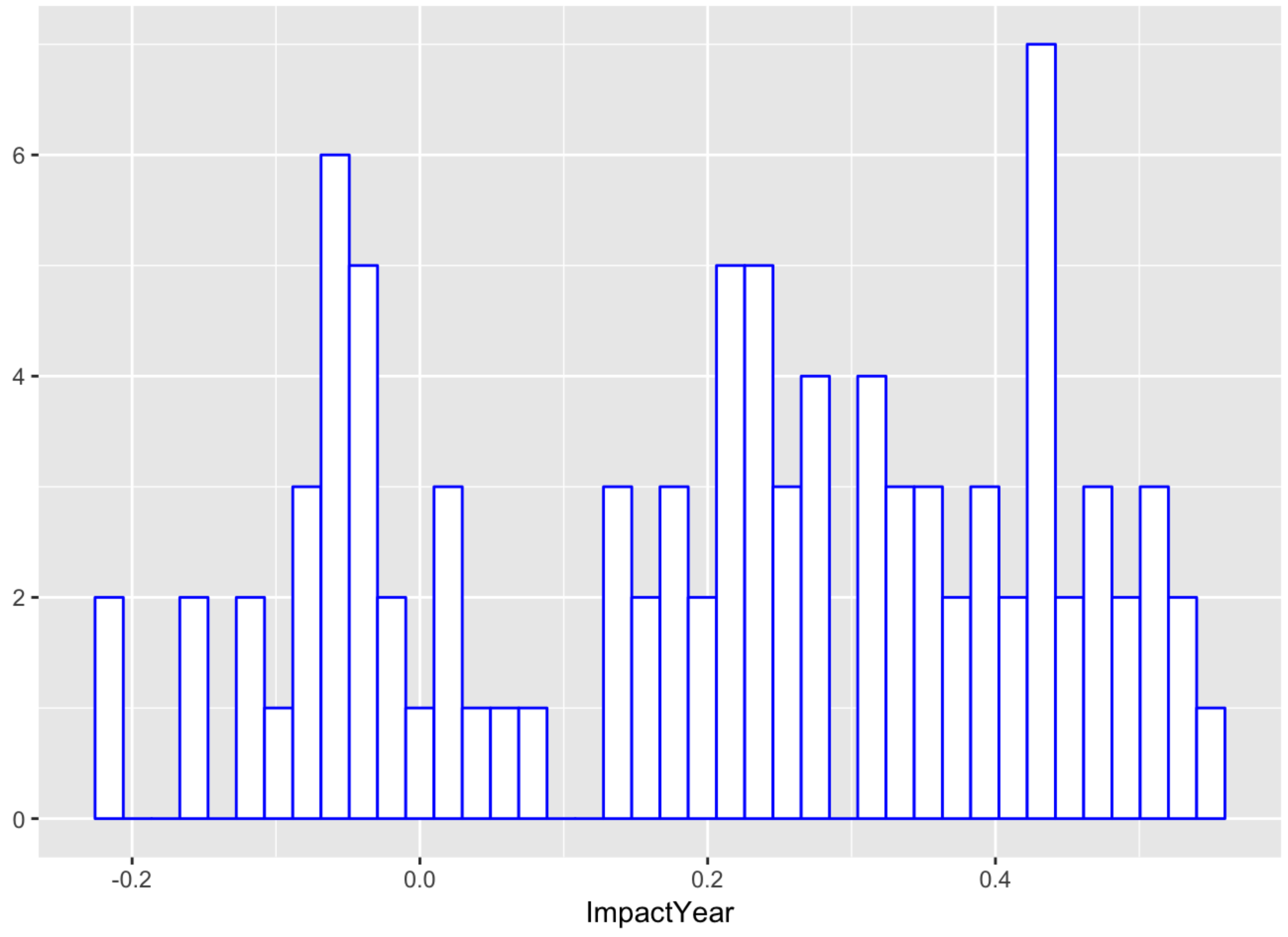
```
## [1] 0.9780767
```

This is better than our baseline (1.06), but not quite as good as Model 2 that used the Movie effect.

Model 4 - Year Effect

Let’s see if the year has a similar effect as movie and user:

```
# What's the impact per YEAR compared to the overall Mean?
ImpactYear <-
  edx %>%   group_by(year) %>%  summarize(ImpactYear = sum(rating - avg) /  (n() + 2))
# Plot Impact per YEAR (compared to overall mean)
ImpactYear %>% qplot(
  ImpactYear,   geom = "histogram",   bins = 40,   data = .,   color = I("blue"),fill =
I("white")
)
```



```
#MODEL 4 - YEAR - Add Mean per Year  model
prediction_Year <-
  validation %>%   left_join(ImpactYear, by = 'year') %>%  mutate(pred = avg + ImpactYea
r)
model_4_rmse <- RMSE(validation$rating, prediction_Year$pred)
model_4_rmse
```

```
## [1] 1.050026
```

This is only slightly better than our baseline of 1.06. Not as helpful as movie or user.

So let’s see what a model levering BOTH movie and user can do.

Model 5 - User and Movie Effects

```
#MODEL 5 - -  User  AND Movie
prediction_UserMovie <-
  validation %>%   left_join(ImpactUser, by = 'userId') %>%
  left_join(ImpactMovie, by = 'movieId') %>%   mutate(pred = avg + ImpactUser + ImpactMo
vie)

model_5_rmse <- RMSE(validation$rating, prediction_UserMovie$pred)
model_5_rmse
```

```
## [1] 0.8840212
```

This is much better than looking at each effect individually. We're much closer to our goal of 0.87. Maybe we can use the year effect to bring us closer?

Model 6 - User, Movie and Year Effects

```
prediction_UserMovieYear <-
  validation %>%   left_join(ImpactUser, by = 'userId') %>%
  left_join(ImpactMovie, by = 'movieId') %>%  left_join(ImpactYear, by =
                                                        'year') %>%
  mutate(pred = avg + ImpactUser + ImpactMovie + ImpactYear)

model_6_rmse <-  RMSE(validation$rating, prediction_UserMovieYear$pred)
model_6_rmse
```

```
## [1] 0.9004851
```

Adding the Year did not improve our result. Another option to improve Model 5 (User and Movie Effect) is to introduce regularization.

Model 7 - Regularisation of Movie and User Effects

Regularization is one of the basic and most important concept in the world of Machine Learning. It helps us to avoid over-fitting. As seen in Data Exploration, there are big differences in the numbers of ratings per user and per movie. We want to be less reliant one those data points, since there low number makes them less reliable for us.

We will use a lambda value to find the best RMSE. If lambda is too low, we risk over-fitting. If it is too high, we risk under-fitting.

So we create a function that returns the RMSE for a given lambda:

```
Movie_User_Lambda <- function(lambda) {

#avg <- mean(edx$rating)

  # calculate movie coefficients for the Movie Effect
  MovieRegEffect <-
    edx %>%   group_by(movieId) %>%  summarize(MovieRegEffect = sum(rating - avg) /
                                                (n() + lambda))
  # calculate user coefficients for the User Effect
  UserRegEffect <-
    edx %>%  left_join(MovieRegEffect, by = "movieId") %>% group_by(userId) %>%  summari
ze(UserRegEffect = sum(rating - avg - MovieRegEffect) / (      n() + lambda))

  # add coefficients to validation to create our Prediction
  validation <-
    validation %>%
    left_join(MovieRegEffect, by = 'movieId') %>%
    left_join(UserRegEffect, by = 'userId')

  # Determine RMSE for current lambda
  lambda_rmse <-
    RMSE(validation$rating,
         (avg + validation$MovieRegEffect + validation$UserRegEffect))

  return(lambda_rmse)
}
```

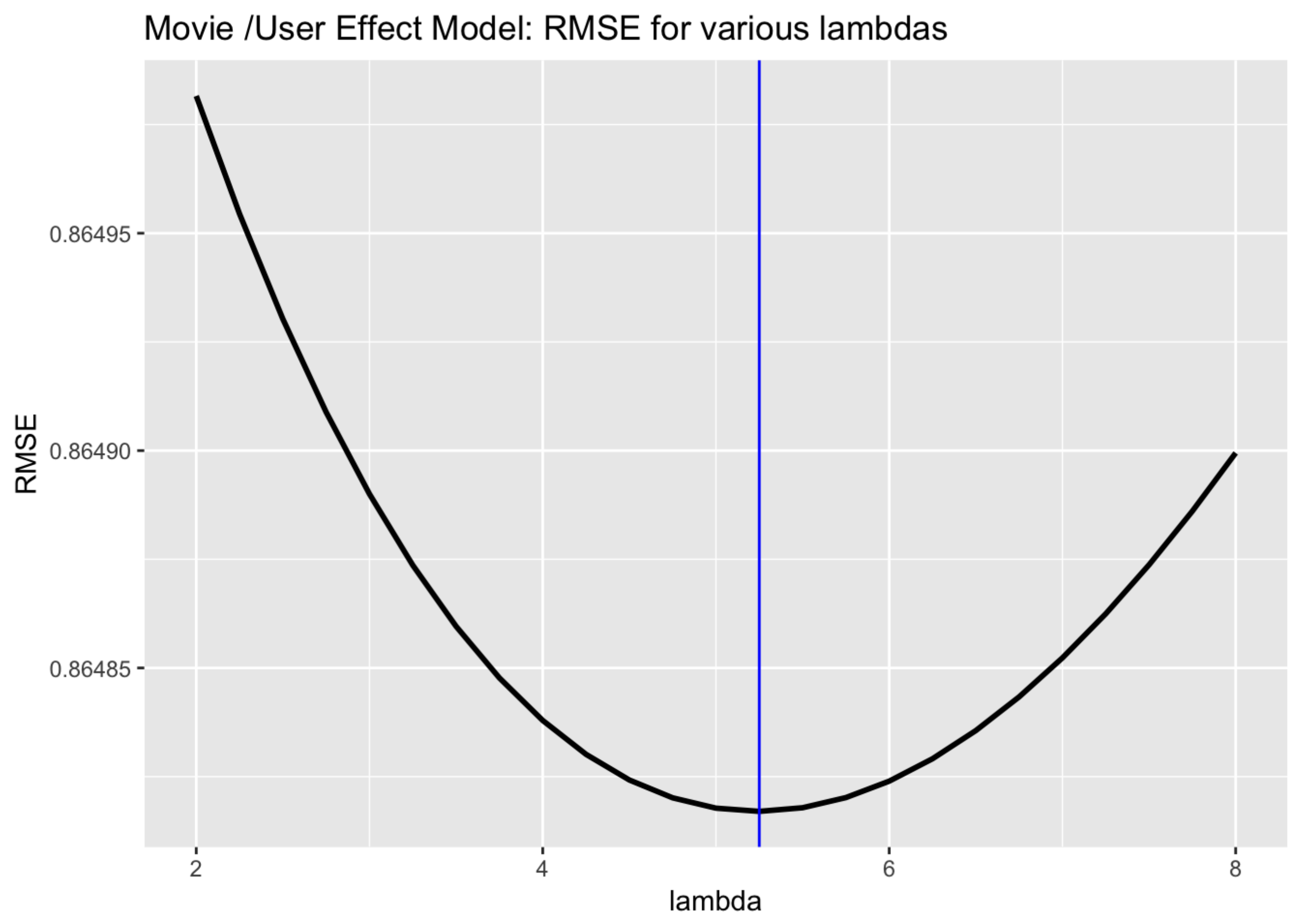
We try a sequence of lambdas from 2 to 8 and use the function above to determine the smallest RMSE.

```
lambdas <- seq(2, 8, 0.25)    #target: 5.25

lambda_rmses <- sapply(lambdas, Movie_User_Lambda)

#Select the best Lambda with the smallest RMSE
lambda_of_smallest_rmse <- lambdas[which.min(lambda_rmses)]
lambda_of_smallest_rmse
```

```
## [1] 5.25
```

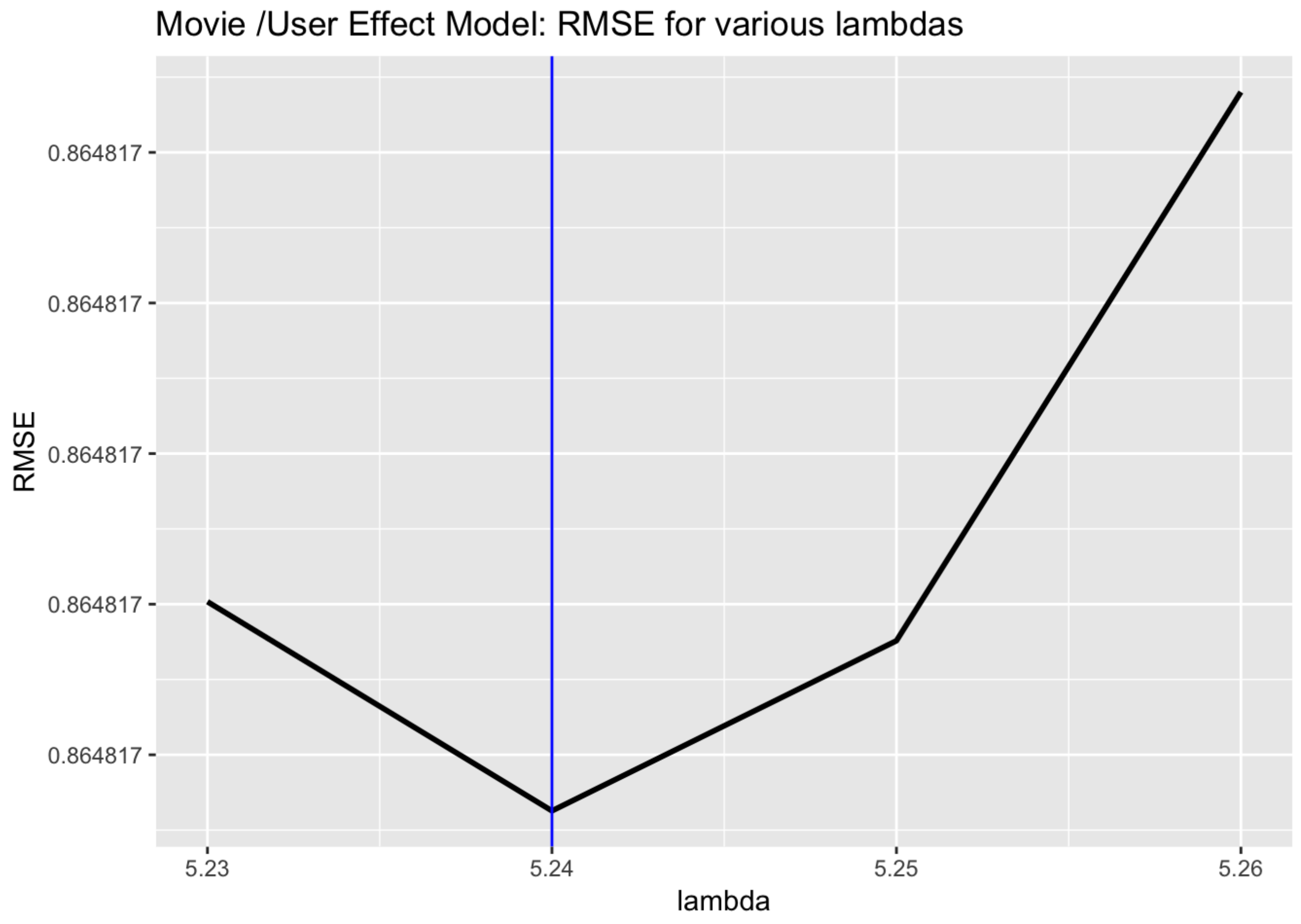
This plot shows us the distribution of all values:

```
# plot Lambdas vs. RMSEs
ggplot() + geom_line(aes(lambdas, lambda_rmses), col = "black", size = 1) +
  geom_vline(xintercept = lambda_of_smallest_rmse, col = "blue") + ggtitle("Movie /User
 Effect Model: RMSE for various lambdas") + ylab("RMSE") + xlab("lambda")
```



5.25 is our best lambda value right now. It’s better than 5 or 5.5. But maybe there is an even better value in between.

```
## [1] 5.24
```



We see that a lambda of 5.24 gives us the best RMSE.

```
 lambdas[which.min(lambda_rmses)]
```

```
## [1] 5.24
```

```
model_7_rmse <- lambda_rmses[which.min(lambda_rmses)]
```

4. Results

These are the RMSEs of the various models we have used:

```
## Model 1 - Average rating for all movies       1.061202
```

```
## Model 2 - Movie Effect                        0.9438528
```

```
## Model 3 - User Effect                         0.9780767
```

```
## Model 4 - Year Effect                         1.050026
## Model 5 - User and Movie Effects              0.8840212
```

```
## Model 6 - User, Movie and Year Effects            0.9004851
```

```
## Model 7 - Regularisation of Movie and User Effects    0.864817
```

5. Conclusion

Movies and Users have a very strong impact on the accuracy of the prediction and the RMSE. A movies year seems to be a bad predictor of a users rating. By using regularization to determine the right balance of over- and under-fitting, we get the lowest RMSE of <0.87.