MovieLens Project Report

Mehul Mohta

15/04/2021

Introduction & Executive Summary

The Project is related to the MovieLens Project of the HarvardX:PH125:9x Data science Capstone course. Recommendation systems use ratings that users have given items to make specific recommendations. Items for which a high rating is predicted for a given user are then recommended to that user. Data visualization techniques are used to get a perspective of the data. This also enables us to identify the right approach to arrive at the most appropriate model. **The goal of the project is to develop a machine learning algorithm using the inputs in one subset to predict movie ratings in the other (validation) set.**

This report contains 5 sections - problem definition, data description & loading, exploratory analysis, modeling and data analysis, results and conclusion . The project uses Penalized least squares approach which is centered around the mean movie rating.The mean is adjusted for any biases due to movie, user, year & genre which have larger effect on errors. I have minimized these effects by using the proposed above method to improve the accuracy. **The final optimal RMSE derived out of the below modeling exercise is 0.86429**

#(1/5)Problem Definition

Train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.Movie recommendation system predicts the movie rating by a user based on users past rating of movies. There can be different type of biases present in the movie reviews due to various reasons. One has to build a ML code so as to minimize these biases while predicting movie ratings

#(2/5)Data Description & Loading

For this project, I will be creating a movie recommendation system using the MovieLens dataset. The version of movielens included in the dslabs package is just a small subset of a much larger dataset with millions of ratings. We will use the 10M version of the MovieLens dataset to make the computation a little easier. The same can be found at, MovieLens 10M dataset:

https://grouplens.org/datasets/movielens/10m/
http://files.grouplens.org/datasets/movielens/ml-10m.zip

Create edx set, validation set (final hold-out test set)

Note: this process could take a couple of minutes as we will load packages and libraries

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

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```r
#if(!require(tinytex)) install.packages("tinytex", repos = "http://cran.us.r-project.org")
#Some machines may not have Latex installation and hence need above package

library(tidyverse)
library(caret)
library(data.table)
#library(tinytex)
```

We will now download the MovieLens data from internet and store in variable "d1" and then create two data frames as "ratings" with 4 columns and "movies" with three columns

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

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

# Since I am using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
          title = as.character(title),
          genres = as.character(genres))

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

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

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

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

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)

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

The above chunk of code gives a partition of the dataset for training and validation from our original dataset. It also removes the unnecessary files from the working directory

#(3/5) Exploratory Analysis

Quick preview of the edx dataset shows 6 columns. "userId","movieID","rating","timestamp","title","genres" in the subset. Each row represent a single rating of a user for a single movie.

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 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
```

Two Important Observations
1. Timestamp will need to be converted if used, and release year will need to be split from the title if used for prediction
2. Genres is a single pipe-delimited string containing the various genre categories a movie might be categorized under, and this will need to be split out if it affects rating outcome

A Summary of the subset confirms that there are no missing values.

```
summary(edx)
```

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

```
# Examining the distribution of "rating" in the training "edx" data set.
table(edx$rating)
```

```
##
##     0.5       1     1.5       2     2.5       3     3.5       4     4.5       5
##   85374  345679  106426  711422  333010 2121240  791624 2588430  526736 1390114
```

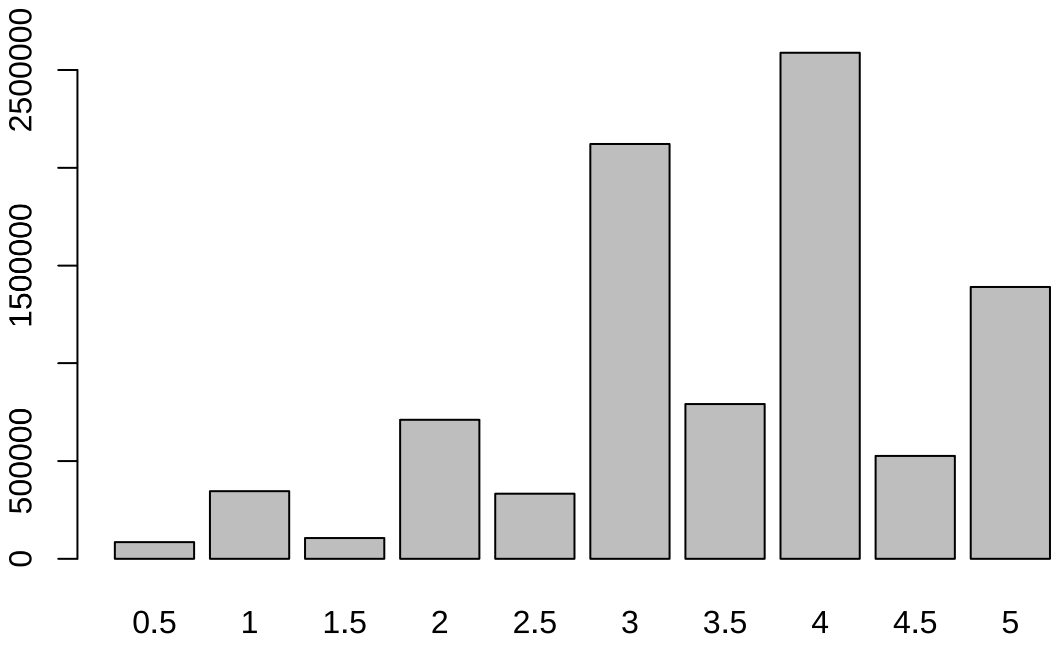
From above output, we can find the rating range as: 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5

```
# We can also see the proportion of each ratings
prop.table(table(edx$rating))
```

```
##
##         0.5           1         1.5           2         2.5           3
## 0.009485942 0.038408543 0.011825039 0.079046406 0.037000885 0.235691893
##         3.5           4         4.5           5
## 0.087957685 0.287601576 0.058525865 0.154456167
```

From above output, we can see that maximum ratings are "3" & "4" and same is also visible from bar plot

```
barplot(table(edx$rating))
```



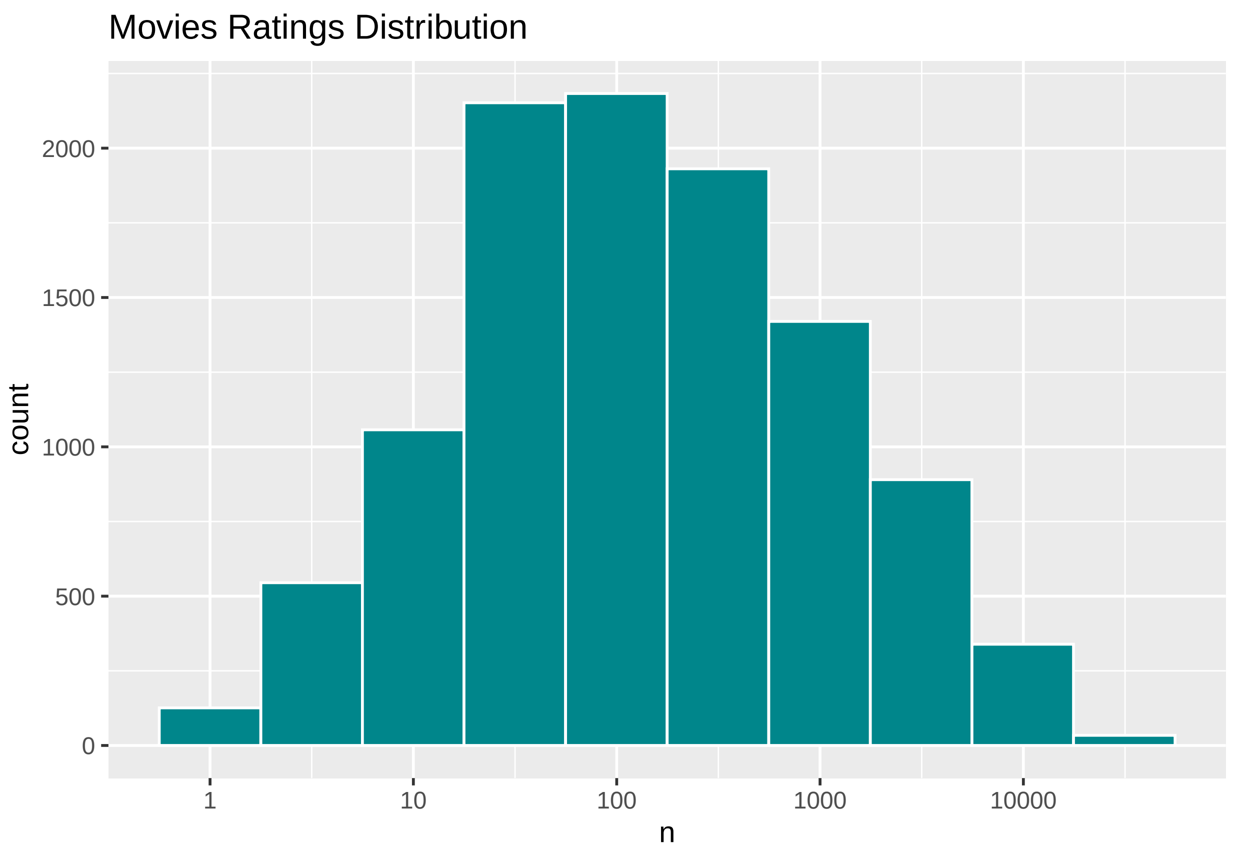
Finding frequencies of users, movies & genres in the edx dataset

```
edx %>%
  summarize(users = n_distinct(userId), movies = n_distinct(movieId), genres = n_distinct(genres))
```

```
##   users movies genres
## 1 69878  10677    797
```

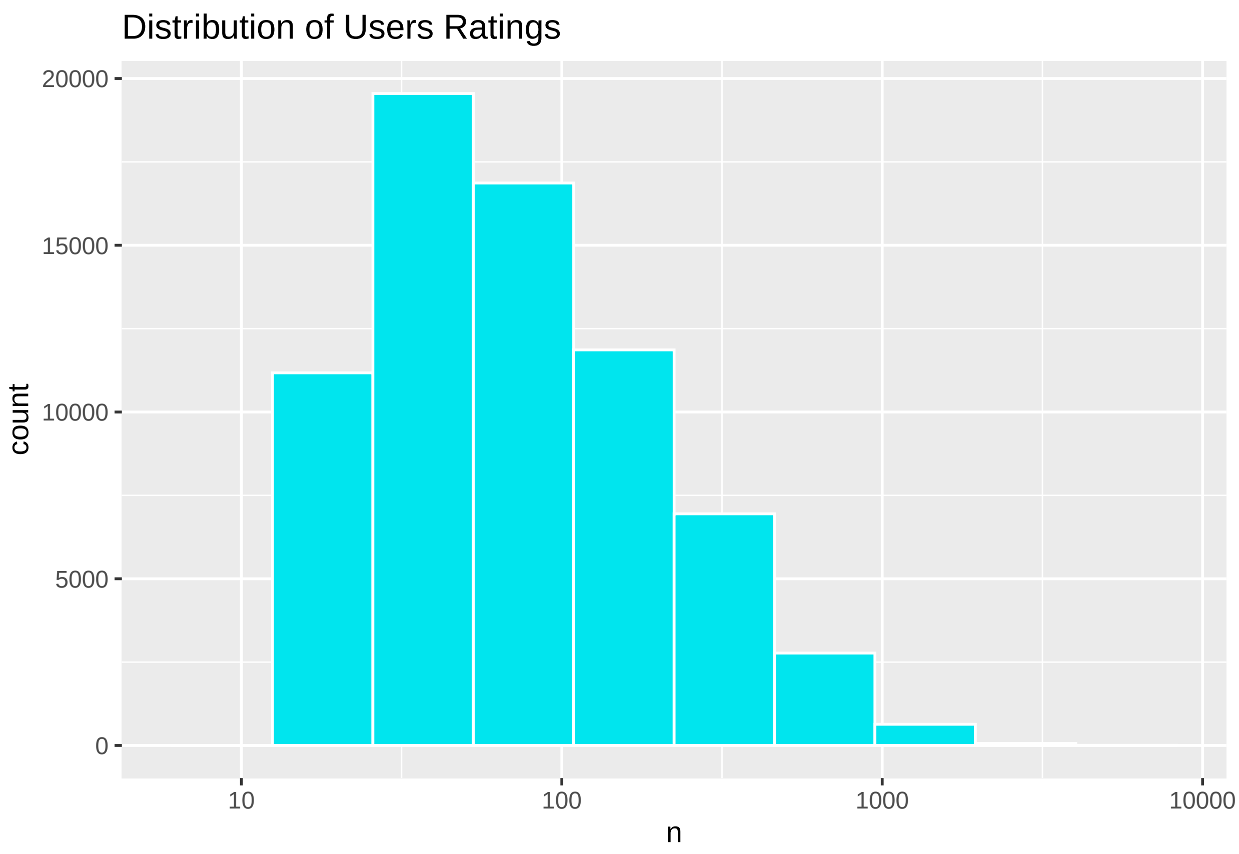
Distribution of Movie Ratings

```
edx %>%
  group_by(movieId) %>%
  summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(fill = "turquoise4", color = "white", bins = 10) +
  scale_x_log10() +
  ggtitle("Movies Ratings Distribution")
```



Distribution of Users

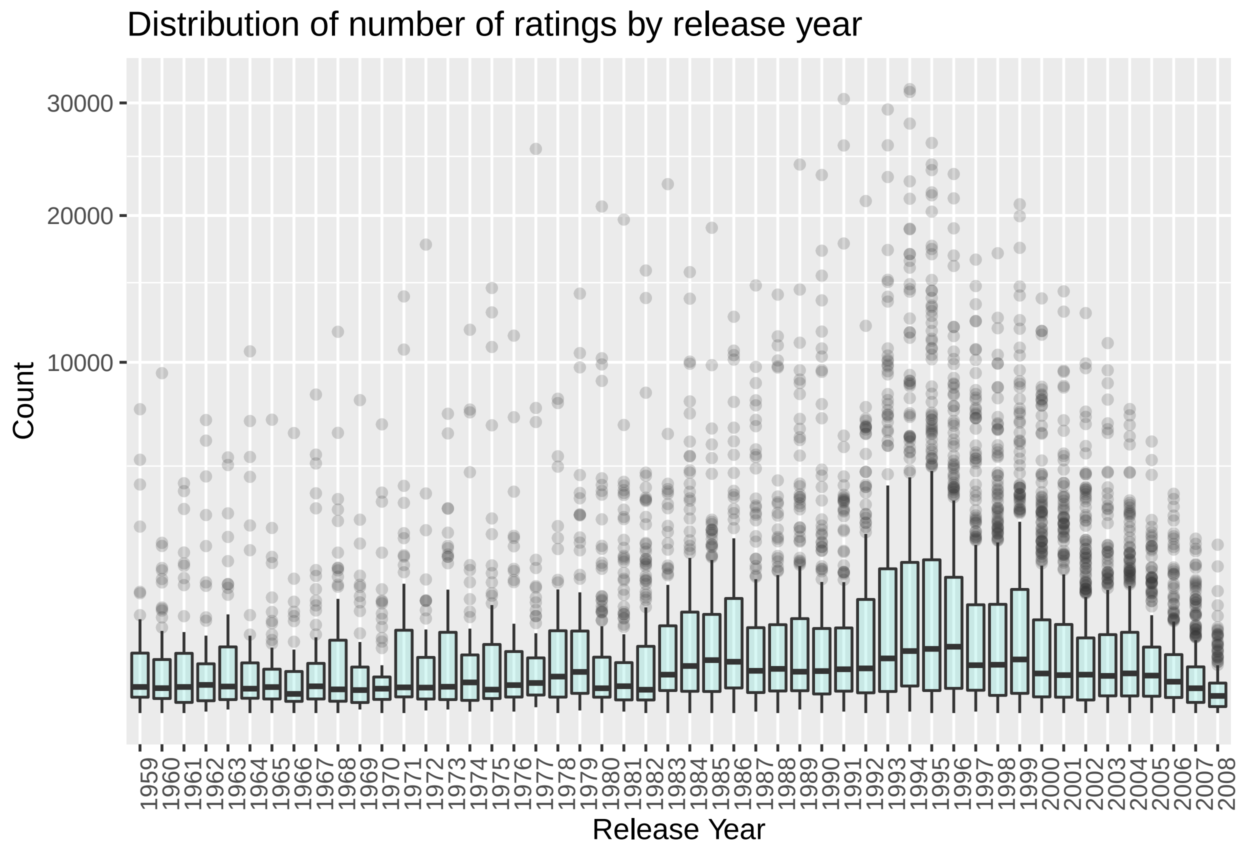
```
edx %>%
  group_by(userId) %>%
  summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(fill = "turquoise2", color = "white", bins = 10) +
  scale_x_log10() +
  ggtitle("Distribution of Users Ratings")
```



From above output, we can see that movie rating distribution follows a almost normal distribution

Distribution of ratings by year

```
edx %>%
  mutate(release = str_sub(title, start = -5, end = -2)) %>%
  filter(as.numeric(release) > 1958) %>%
  group_by(movieId) %>%
  summarize(n = n(), release = first(release)) %>%
  ggplot(aes(x = release, y = n)) +
  geom_boxplot(fill = "turquoise", alpha = 0.2) +
  coord_trans(y = "sqrt") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("Release Year") + ylab("Count") +
  ggtitle("Distribution of number of ratings by release year")
```



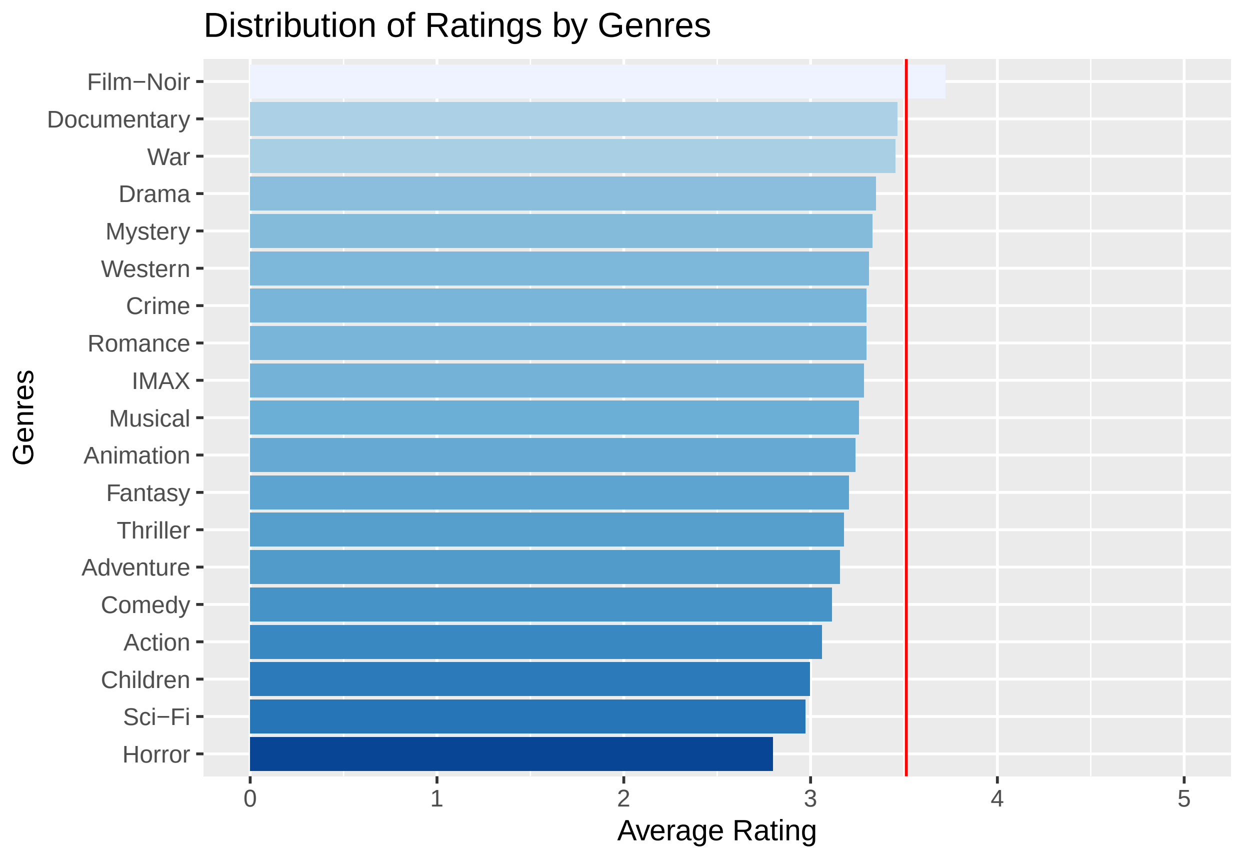
From above output, we can see that between 1993 to 1995 has seen a much higher number of ratings than any other year. It also seems that movies released in the 1990-2000 periods have more ratings on average. However, the rate of ratings tend to decrease for newer movies. For these reasons we will split year out as it may affect our model

Modify the year as a column in the both datasets

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

In the following code, we extract the different genres values to examine its effect on the average rating.

```
genres <- edx %>%
  group_by(movieId) %>%
  summarize(r = mean(rating), title = title[1], genre = genres[1]) %>%
  separate_rows(genre, sep = "\\|") %>%
  group_by(genre) %>%
  summarize(r = mean(r)) %>%
  filter(!genre == "(no genres listed)")

genres %>%
  ggplot(aes(x=r, y=reorder(genre, r), fill=r)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  coord_cartesian(xlim = c(0, 5)) +
  scale_fill_distiller(palette = "Blues") +
  labs(x="Average Rating", y="Genres") +
  geom_vline(xintercept = mean(edx$rating), color = "red") +
  ggtitle("Distribution of Ratings by Genres")
```



From above output, it seems that movies tagged with the genre "Film-Noir" tend to have higher ratings, while movies in the "Horror" and "Sci-Fi" genres tend to have lower ratings. The average rating is also plotted as a red line as a reference.

#(4/5) Modeling and Data Analysis

The most suitable approach to this project to get a RMSE of less than 0.86490 would be the Root Mean Square Method. The essential part in this exercise is to arrive at the optimal tuning factor 'Lambda' at which the RMSE is minimal and also below 0.86490.

The following code is used to determine the optimal tuning factor: (it will take few minutes to run)

#As instructed we will split the edx data set also into two sets - training and test sets, to experiment with multiple parameters

```
edx_index <- createDataPartition(y = edx$rating, times = 1, p = 0.15, list = FALSE)
train_edx <- edx[-edx_index,]
test_edx <- edx[edx_index,]

test_edx <- test_edx %>%
  semi_join(train_edx, by = "movieId") %>%
  semi_join(train_edx, by = "userId")

rm(edx_index)

#  Root Mean Square Error Loss Function
  RMSE <- function(true_ratings, predicted_ratings)
  {
       sqrt(mean((true_ratings - predicted_ratings)^2))
  }
```

Model 1: Rating Average

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

  rat_avg_rmses <- RMSE(mu,test_edx$rating)

  rmse_results <- tibble(method = "Average rating", RMSE = min(rat_avg_rmses))
  rmse_results %>% knitr::kable()
```

method	RMSE
Average rating	1.061102

Since the Rating Average RMSE is 1.0611, which is very high compared to our target, for all other models we will try obatianing RMSE using regularization approach

Model 2: Movie Effect with Regularization

```
  lambdas <- seq(0, 5, 0.25)

  movie_rat_rmses <- sapply(lambdas,function(l)
  {
      pen_m <- train_edx %>%
      group_by(movieId) %>%
      summarize(pen_m = sum(rating - mu)/(n()+l))

  #Predict ratings in the TEST set
```

```
    predicted_ratings <- test_edx %>%
    left_join(pen_m, by = "movieId") %>%
    mutate(pred = mu + pen_m ) %>%
    .$pred

return(RMSE(predicted_ratings,test_edx$rating))
})
rmse_results <- bind_rows(rmse_results,
                tibble(method="Movie Effects Model",RMSE = min(movie_rat_rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Average rating	1.0611021
Movie Effects Model	0.9438993

Model 3: User + Movie Effect with Regularization

```
user_movie_rat_rmses <- sapply(lambdas,function(l)
{
  #Adjust mean by movie effect
    pen_m <- train_edx %>%
    group_by(movieId) %>%
    summarize(pen_m = sum(rating - mu)/(n()+l))

  #Adjust mean by user and movie effect
     pen_m_u <- train_edx %>%
     left_join(pen_m, by="movieId") %>%
     group_by(userId) %>%
     summarize(pen_m_u = sum(rating - mu - pen_m)/(n()+l))

  #Predict ratings in the TEST set
    predicted_ratings <- test_edx %>%
    left_join(pen_m, by = "movieId") %>%
    left_join(pen_m_u, by = "userId") %>%
    mutate(pred = mu + pen_m + pen_m_u) %>%
    .$pred

    return (RMSE(predicted_ratings,test_edx$rating))
})
    rmse_results <- bind_rows(rmse_results,
                    tibble(method="User + Movie Effects Model",RMSE = min(user_movie_rat_rmses)))
    rmse_results %>% knitr::kable()
```

method	RMSE
Average rating	1.0611021
Movie Effects Model	0.9438993
User + Movie Effects Model	0.8653022

Model 4: Movie + User + Genre Effects Model

```
    genres_user_movie_rat_rmses <- sapply(lambdas,function(l)
    {
      #Adjust mean by movie effect
      pen_m <- train_edx %>%
      group_by(movieId) %>%
      summarize(pen_m = sum(rating - mu)/(n()+l))

    #Adjust mean by user and movie effect
      pen_m_u <- train_edx %>%
      left_join(pen_m, by="movieId") %>%
      group_by(userId) %>%
      summarize(pen_m_u = sum(rating - mu - pen_m)/(n()+l))

    #Adjust mean by user, movie,genres effect
      pen_g <- train_edx %>%
      left_join(pen_m, by="movieId") %>%
      left_join(pen_m_u, by="userId") %>%
      group_by(genres) %>%
      summarize(pen_g = sum(rating - mu - pen_m - pen_m_u)/(n()+l))

    #Predict ratings in the TEST set
      predicted_ratings <- test_edx %>%
      left_join(pen_m, by = "movieId") %>%
      left_join(pen_m_u, by = "userId") %>%
      left_join(pen_g, by = "genres")%>%
      mutate(pred = mu + pen_m + pen_m_u + pen_g) %>%
      .$pred

      return(RMSE(predicted_ratings,test_edx$rating))
    })
      rmse_results <- bind_rows(rmse_results,
                     tibble(method="Genres + User + Movie Effects Model",
                     RMSE = min(genres_user_movie_rat_rmses)))
      rmse_results %>% knitr::kable()
```

method	RMSE
Average rating	1.0611021
Movie Effects Model	0.9438993
User + Movie Effects Model	0.8653022
Genres + User + Movie Effects Model	0.8649804

Model 5: Movie + User + Genre + Year Effects Model

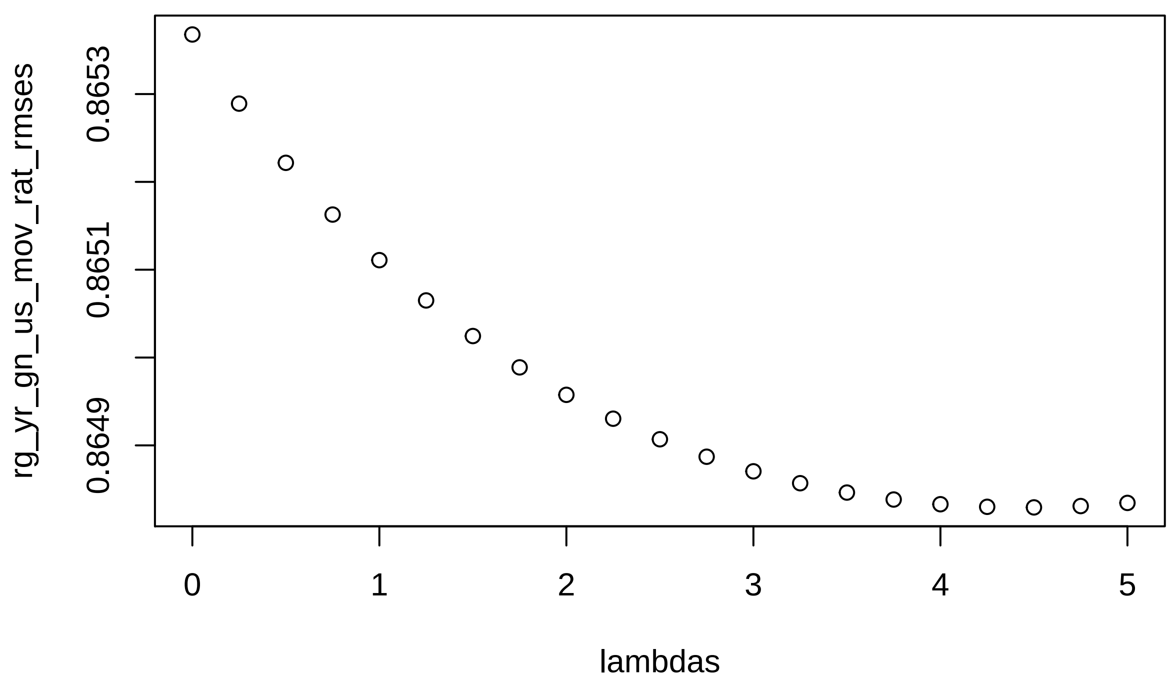
```
   rg_yr_gn_us_mov_rat_rmses <- sapply(lambdas,function(l)
    {
      #Adjust mean by movie effect and penalize low number on ratings
         pen_m <- train_edx %>%
         group_by(movieId) %>%
         summarize(pen_m = sum(rating - mu)/(n()+l))

     #Adjust mean by user and movie effect and penalize low number of ratings
```

```
      pen_m_u <- train_edx %>%
      left_join(pen_m, by="movieId") %>%
      group_by(userId) %>%
      summarize(pen_m_u = sum(rating - mu - pen_m)/(n()+1))

  #Adjust mean by user, movie,genres effect and penalize low number of ratings
      pen_g <- train_edx %>%
      left_join(pen_m, by="movieId") %>%
      left_join(pen_m_u, by="userId") %>%
      group_by(genres) %>%
      summarize(pen_g = sum(rating - mu - pen_m - pen_m_u)/(n()+1))

  #Adjust mean by user, movie,year & genres effect and penalize low number of ratings
      pen_y <- train_edx %>%
      left_join(pen_m, by="movieId") %>%
      left_join(pen_m_u, by="userId") %>%
      left_join(pen_g, by="genres") %>%
      group_by(year) %>%
      summarize(pen_y = sum(rating - mu - pen_m - pen_m_u - pen_g)/(n()+1))


  #Predict ratings in the TEST set to derive optimal penalty value 'lambda'
      predicted_ratings <- test_edx %>%
      left_join(pen_m, by = "movieId") %>%
      left_join(pen_m_u, by = "userId") %>%
      left_join(pen_g, by = "genres")%>%
      left_join(pen_y, by = "year") %>%
      mutate(pred = mu + pen_m + pen_m_u + pen_g + pen_y) %>%
      .$pred

      return(RMSE(predicted_ratings,test_edx$rating))
})

rmse_results <- bind_rows(rmse_results,
                tibble(method="Year + Genres + User + Movie Effect Model",
                RMSE = min(rg_yr_gn_us_mov_rat_rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Average rating	1.0611021
Movie Effects Model	0.9438993
User + Movie Effects Model	0.8653022
Genres + User + Movie Effects Model	0.8649804
Year + Genres + User + Movie Effect Model	0.8648294

From the above table it is visible that we have been able to bring down RMSE below the target of 0.8649 when we use the "Year + Genres + User + Movie" regularized model

```
plot(lambdas, rg_yr_gn_us_mov_rat_rmses)
```



```
lambda <- lambdas[which.min(rg_yr_gn_us_mov_rat_rmses)]  #This gives optimal lambda

#Now, the prediction on the VALIDATION set would be done based on the Lambda obtained.

#Calculate movie effect with optimal lambda
    movie_effect <- edx %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n()+lambda))

#Calculate user effect with optimal lambda
    user_effect <- edx %>%
    left_join(movie_effect, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_m)/(n()+lambda))

#Calculate genres effect with optimal lambda
    genres_effect <- edx %>%
    left_join(movie_effect, by="movieId") %>%
    left_join(user_effect, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_m - b_u)/(n()+lambda))

#Calculate year effect with optimal lambda
```

```
    year_effect <- edx %>%
    left_join(movie_effect, by="movieId") %>%
    left_join(user_effect, by="userId") %>%
    left_join(genres_effect, by="genres") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - b_m - b_u - b_g)/(n()+lambda))


#Predict ratings on validation set
    predicted_ratings <- validation %>%
    left_join(movie_effect, by="movieId") %>%
    left_join(user_effect, by="userId") %>%
    left_join(genres_effect, by="genres") %>%
    left_join(year_effect, by ="year") %>%
    mutate(pred = mu + b_m + b_u + b_g + b_y) %>%
   .$pred

final_model_rmse <- RMSE(validation$rating, predicted_ratings)
min(final_model_rmse)
```

```
## [1] 0.8642948
```

#When we use "Year + Genres + User + Movie" regularized model even on the validation data set, we see that RMSE is still under the target of *0.8649* and thus have achieved our objective.

#(5/5) Results and Conclusion
The RMSE value of Movie, User, Year Effect Model is given below.

method	RMSE
Average rating	1.0611021
Movie Effects Model	0.9438993
User + Movie Effects Model	0.8653022
Genres + User + Movie Effects Model	0.8649804
Year + Genres + User + Movie Effect Model	0.8648294
Year + Genres + Movie + User Effect Model after validation	0.8642948

The aim of the project was to predict movie ratings from a dataset of existing movie ratings. The optimal least square method turns out to be one of the better methods to the algorithm.We first split the data between edx and validation data sets, and then we split edx further into train and test data. We ran five models and calculated RMSE at every stage and progressively seen it improved to optimized number of **0.86429**. The most effective prediction model was "Movie + User + Genre + Year Effects Model" where biases by movie, user, genre and year were used on the training set and then regularized.

The entire MovieLens project helps understand how sometimes knowingly or unknowingly biases can creep in our most simplest of day to day actions. As a budding data scientist it should be our foremost responsibility to lookout for these biases and keep removing it from larger ML models