Contents

MovieLens

Rishabh Singh Dodeja

July 17, 2020

1. Overview

This report is on MovieLens project, a part of Data Science Capstone project of HarvardX's Data Science Professional Certificate program.

The goal of the MovieLens project is to build a movie recommendation system which would predict user ratings for a particular movie with a Root Mean Square Error (RMSE) of < 0.864900

1.1 MovieLens Dataset

The complete MovieLens dataset consists of 27 million ratings of 58,000 movies by 280,000 users. The research presented in this paper is based in a subset of this dataset with 10 million ratings on 10,000 movies by 72,000 users.

1.2 Process and Workflow

The main steps in this project include:

1. **Data Ingestion:** download, parse, import and prepare data for further processing and analysis
2. **Data Exploration:** explore data to understand, analyze and visualize different features and their relationships with movie ratings
3. **Data Cleaning:** eventually remove the unnecessary feature and information from both `edx` and `validation` dataset
4. **Modelling and Analysis:** create the model incorporating features one-by-one using insights gained during data exploration. Also test and validate the model and each step and analyze he RMSE(s)
5. **Communicate:** create report and publish results

2. Data Ingestion

2.1 Data Preparation

In this section we download and prepare the dataset to be used in the analysis. We split the dataset in two parts, the training set called `edx` and the evaluation set called `validation` with 90% and 10% of the original dataset respectively.

While splitting the data we ensure that a user listed in `validation` set is also listed in `edx` set, and same with the movies. This is because, to predict a user rating for a particular movie we would need the data of how that user has rated other movies, as well as how the movie is rated by other users.

The chunk below generated Train and Validation Datasets from MovieLens 10M Data
(source: "Project Overview: MovieLens > Create Train and Validation Sets")

```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

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")

if(!require(data.table)) install.packages("data.table", 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 <- 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")
 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, sample.kind="Rounding")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used

# if using R 3.5 or earlier, use `set.seed(1)` instead
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(ratings, movies, test_index, temp, movielens, removed)
```

2.2 Data Exploration and Anaysis

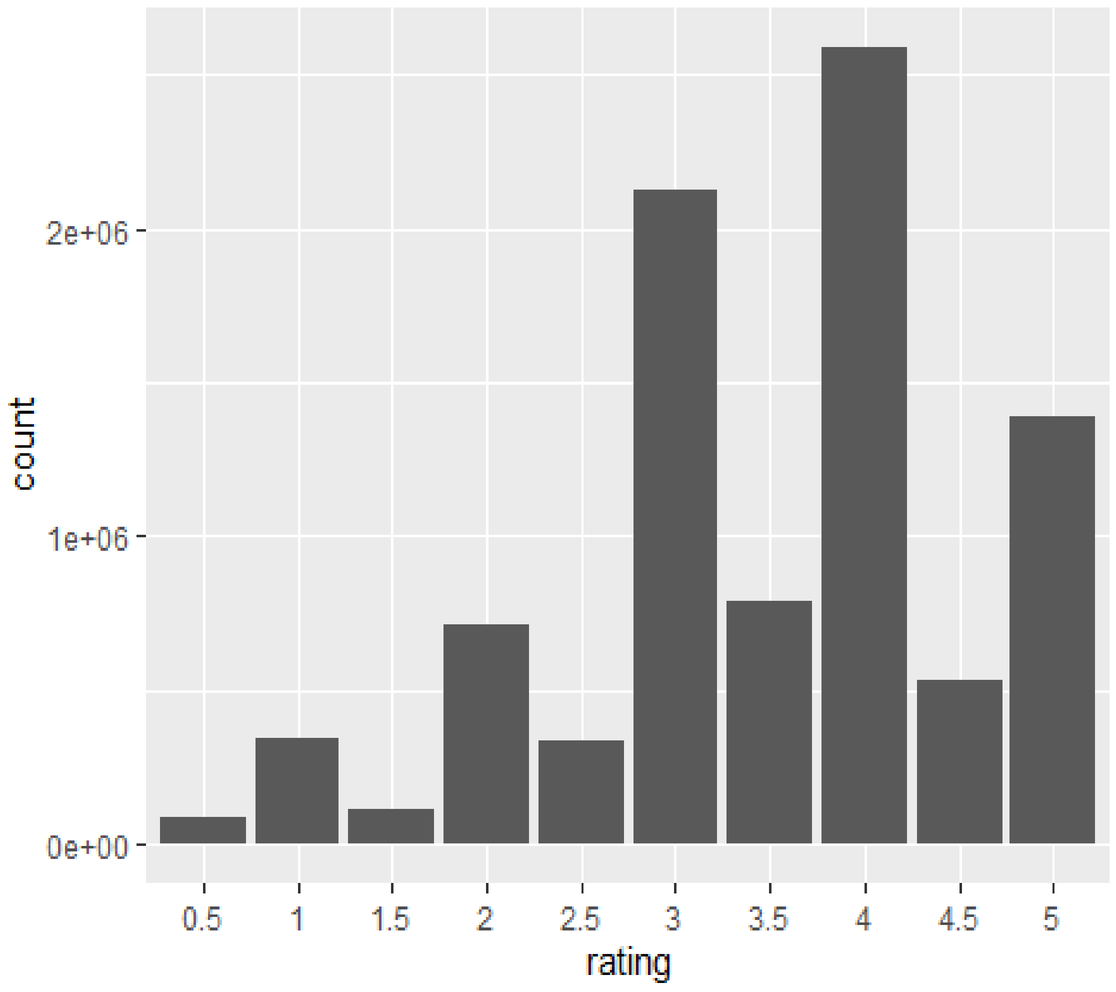
In this section we get to know the structure of our data and try to understanding various features and their relationships with predictors. This helps us building an efficient model

```
str(edx);

## Classes 'data.table' and '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 838983707 838984596 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate
(1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-
Fi|Thriller" "Action|Adventure|Sci-Fi" ...
##  - attr(*, ".internal.selfref")=<externalptr>

str(validation);

## Classes 'data.table' and 'data.frame':   999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494
1133571200 844416936 844417070 ...
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob
Roy (1995)" ...
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy"
"Action|Drama|Romance|War" ...
##  - attr(*, ".internal.selfref")=<externalptr>

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

head(validation,5);

##    userId movieId rating timestamp                title
## 1:      1     231      5 838983392  Dumb & Dumber (1994)
## 2:      1     480      5 838983653  Jurassic Park (1993)
## 3:      1     586      5 838984068     Home Alone (1990)
## 4:      2     151      3 868246450        Rob Roy (1995)
## 5:      2     858      2 868245645 Godfather, The (1972)
##                                     genres
## 1:                                  Comedy
## 2:        Action|Adventure|Sci-Fi|Thriller
## 3:                         Children|Comedy
## 4:                Action|Drama|Romance|War
## 5:                             Crime|Drama
## 6: Action|Adventure|Horror|Sci-Fi|Thriller
```

2.2.1 Ratings

Here we try to visualize in general how users rate movies on a scale of 0.5 to 5. We see, 4 is the most common rating given by users, while 0.5 is the least. We also see that decimal rating are less common than whole number ratings.

```
ratings = edx %>% group_by(rating) %>% summarise(count=n()) %>%
mutate(rating=as.character(rating)) ;

## `summarise()` ungrouping output (override with `.groups` argument)

ratings%>% arrange(desc(count));

## # A tibble: 10 x 2
##    rating   count
##    <chr>    <int>
##  1 4      2588430
##  2 3      2121240
##  3 5      1390114
##  4 3.5     791624
##  5 2       711422
##  6 4.5     526736
##  7 1       345679
##  8 2.5     333010
##  9 1.5     106426
## 10 0.5      85374

ratings %>% ggplot(aes(rating, count)) + geom_col();
```

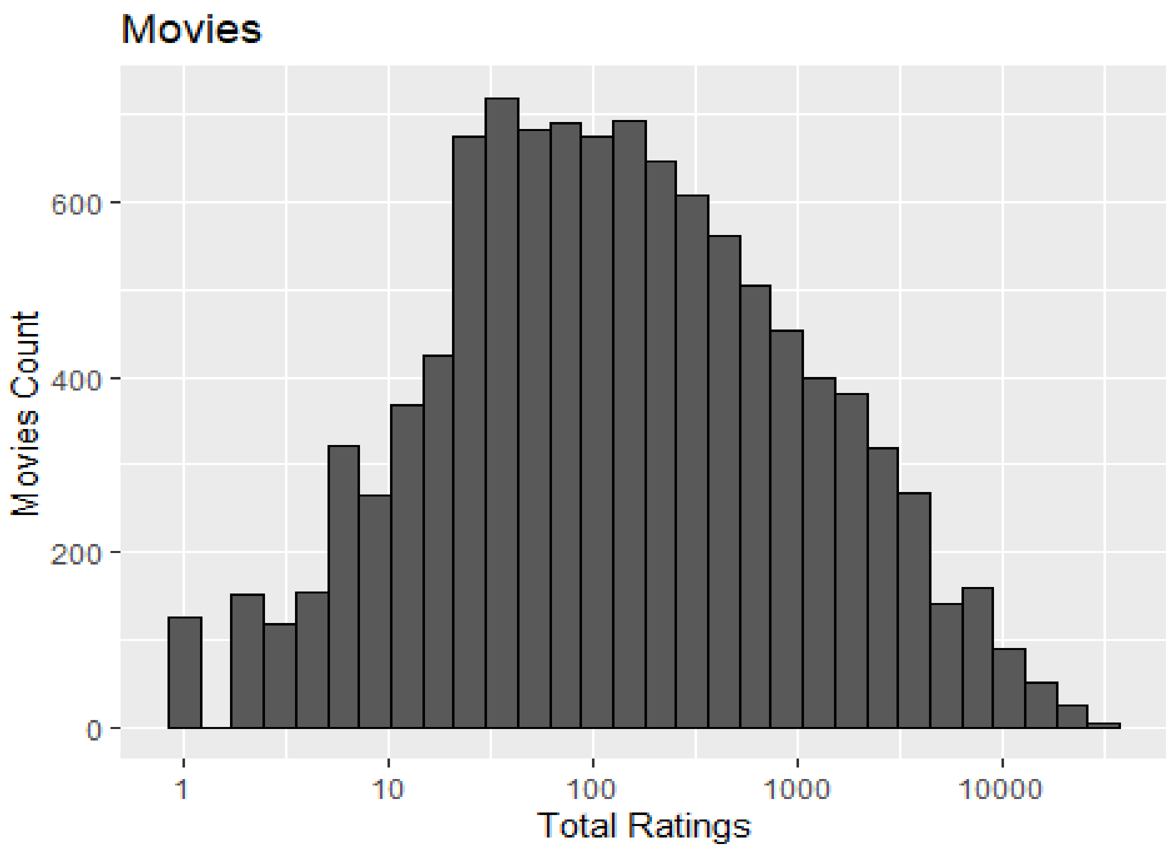


2.2.2 Movies

In this section we explore different movies in the dataset, their average ratings, and their total ratings. This gives us an idea how movies are rated; we see that rating count of movies closely follow a normal distribution and is almost symmetric. There are many movies watched by very few users while there are also many blockbuster movies that very high ratings count.

There are in total 10677 different movies in edx dataset

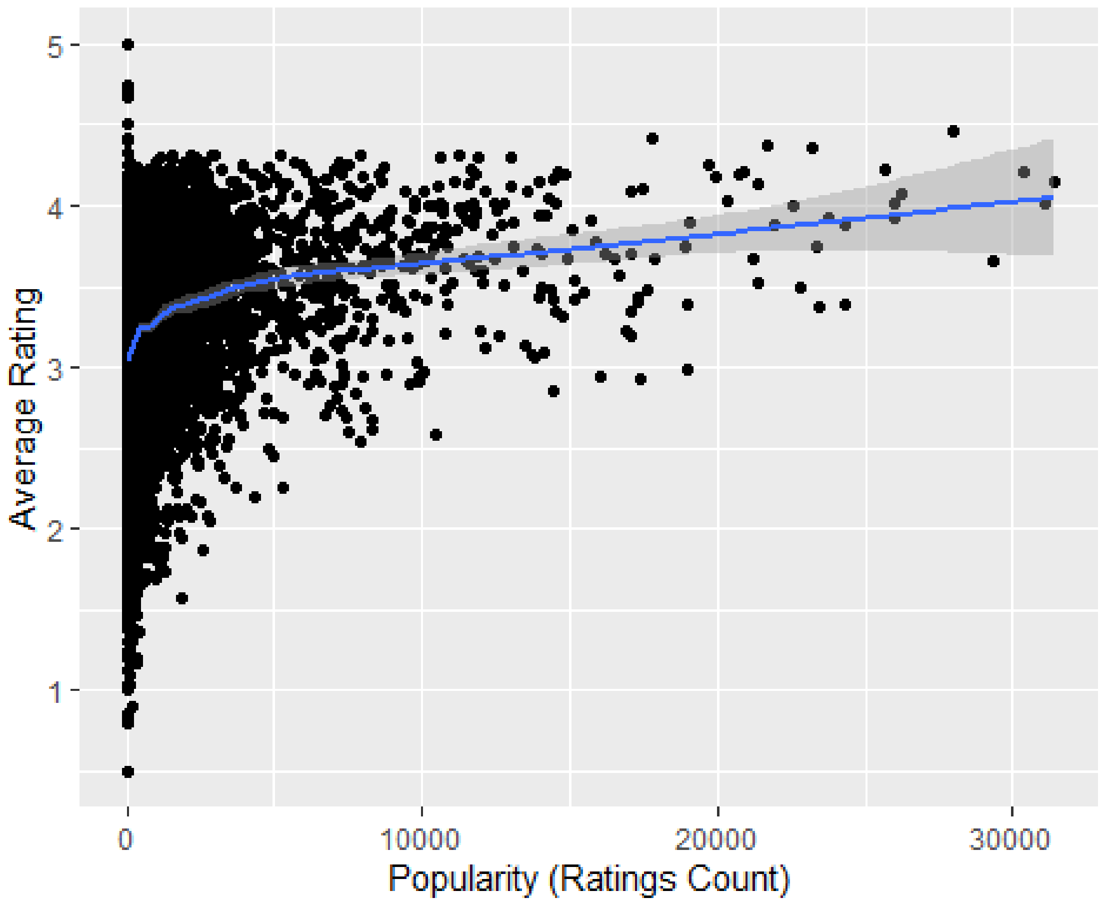
```
movies = edx %>% group_by(movieId, title)%>%
summarise(totalRatings=n(),AverageRating=mean(rating));

## `summarise()` regrouping output by 'movieId' (override with `.groups` argument)

dim(movies);

## [1] 10677     4

movies = movies  %>% arrange(desc(totalRatings));
head(movies);

## # A tibble: 6 x 4
## # Groups:   movieId [6]
##   movieId title                            totalRatings AverageRating
##     <dbl> <chr>                                   <int>         <dbl>
## 1     296 Pulp Fiction (1994)                     31362          4.15
## 2     356 Forrest Gump (1994)                     31079          4.01
## 3     593 Silence of the Lambs, The (1991)        30382          4.20
## 4     480 Jurassic Park (1993)                    29360          3.66
## 5     318 Shawshank Redemption, The (1994)        28015          4.46
## 6     110 Braveheart (1995)                       26212          4.08

movies %>% ggplot(aes(totalRatings)) + geom_histogram(bins = 30, color = "black") +
scale_x_log10() + ggtitle("Movies") + labs(x= "Total Ratings", y = "Movies Count");
```



Popularity v/s Average Rating

Here, we try to visualize understand the relationship between Popularity of a movie, i.e., total no. of ratings and Average Rating of a movie. We see the data is highly scattered at low popularity regions and slightly follow a linear relation at mid-to-high popularity regions. Thus, we can't really suggest a strong relationship between the two at this time.

```
movies %>% ggplot(aes(totalRatings,AverageRating))+ geom_point() +geom_smooth()+ labs(x=
"Popularity (Ratings Count)", y = "Average Rating") ;

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

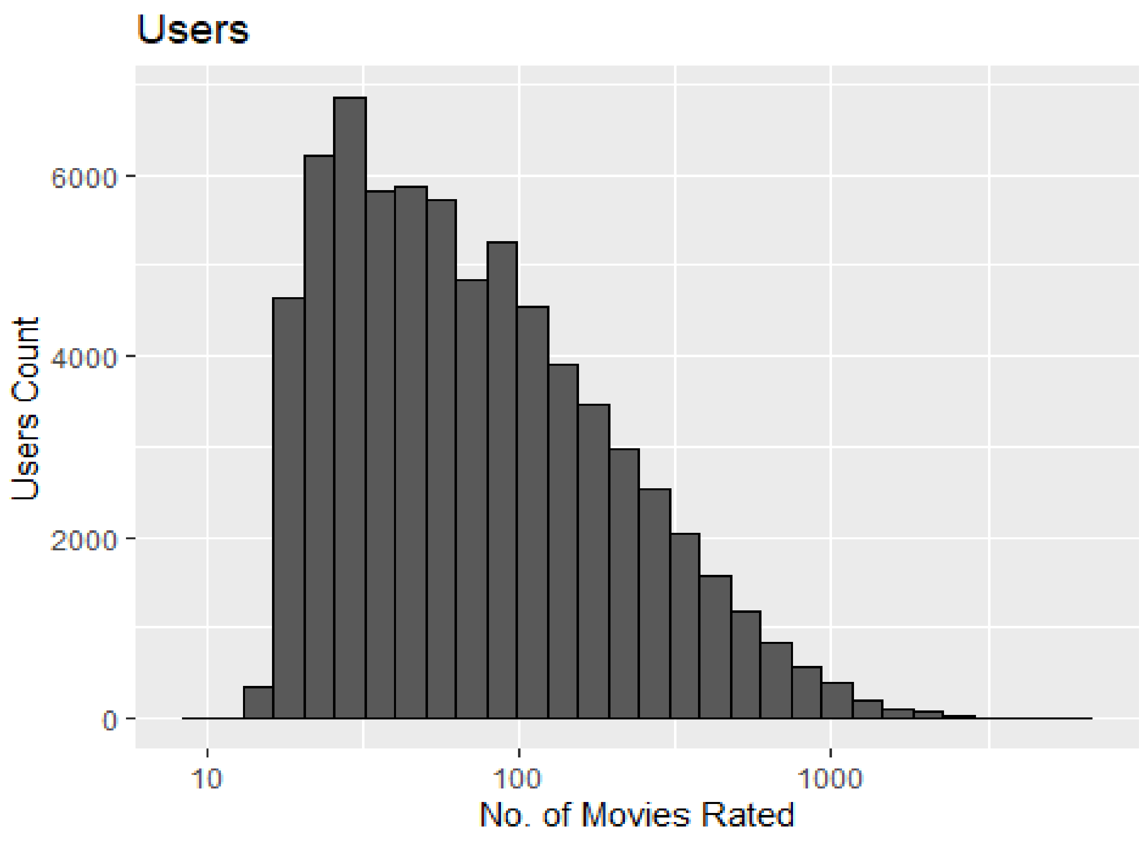


2.2.3 Users

In this section we visualize users' pattern of rating films. The distribution is skewed to right which suggest significant users that rate very high no. or movies, while there are very users that rate <20 movies.

We see out of 69878 users in `edx` dataset each user has at least rated 10 movies and maximum users have rated 50-60 movies

```
users = edx %>% group_by(userId) %>% summarise(Ratedmovies=n());

## `summarise()` ungrouping output (override with `.groups` argument)

dim(users);

## [1] 69878     2

users %>% ggplot(aes(Ratedmovies)) + geom_histogram(bins = 30, color = "black") +
scale_x_log10() + ggtitle("Users") + labs(x= "No. of Movies Rated", y = "Users Count");
```



2.2.4 Genres

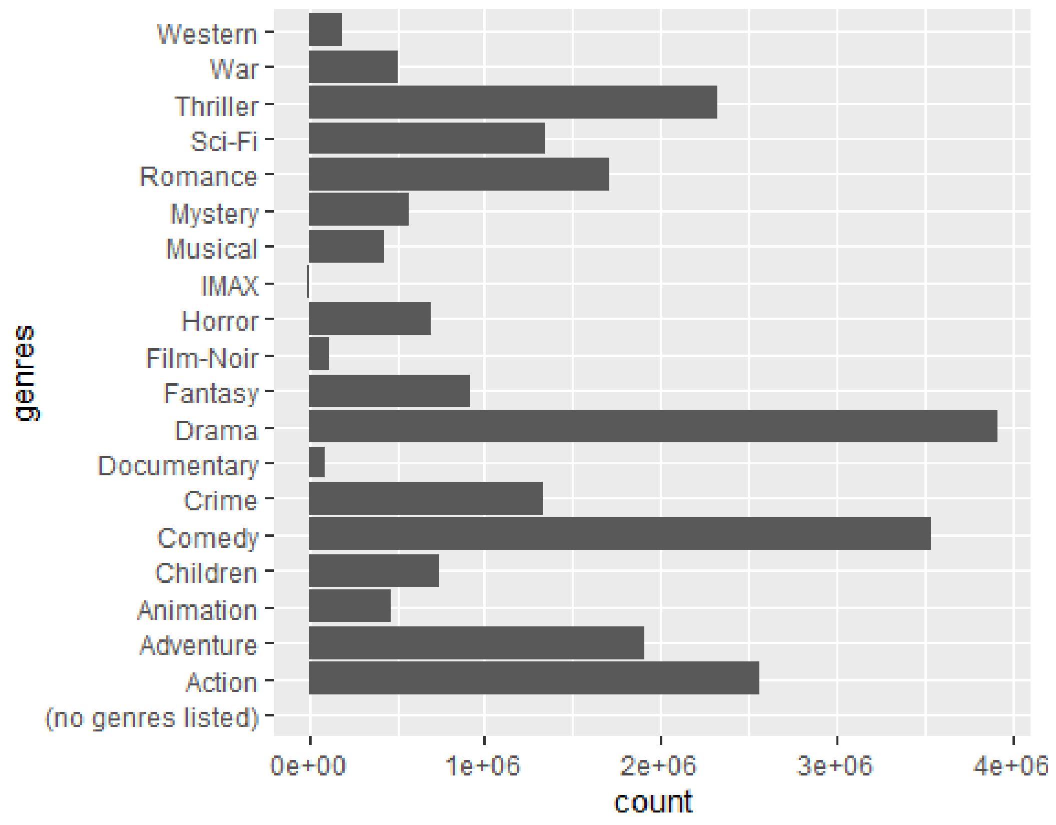
In this section we visualize how average ratings over different genres vary and what are the famous genres. Also, there are 7 movies with no listed genres.

```
genreList = edx %>% separate_rows(genres,sep="\\|") %>% group_by(genres) %>% summarise(count
=n(),

Rating = mean(rating));

## `summarise()` ungrouping output (override with `.groups` argument)

genreList=genreList%>%arrange(count);
genreList;

## # A tibble: 20 x 3
##    genres               count Rating
##    <chr>                <int>  <dbl>
##  1 (no genres listed)       7   3.64
##  2 IMAX                  8181   3.77
##  3 Documentary          93066   3.78
##  4 Film-Noir           118541   4.01
##  5 Western             189394   3.56
##  6 Musical             433080   3.56
##  7 Animation           467168   3.60
##  8 War                 511147   3.78
##  9 Mystery             568332   3.68
## 10 Horror              691485   3.27
## 11 Children            737994   3.42
## 12 Fantasy             925637   3.50
## 13 Crime              1327715   3.67
## 14 Sci-Fi             1341183   3.40
## 15 Romance            1712100   3.55
## 16 Adventure          1908892   3.49
## 17 Thriller           2325899   3.51
## 18 Action             2560545   3.42
## 19 Comedy             3540930   3.44
## 20 Drama              3910127   3.67
```

Genre Popularity

Drama comes out to be most famous genre with count of 3910127 followed by Comedy and Action.

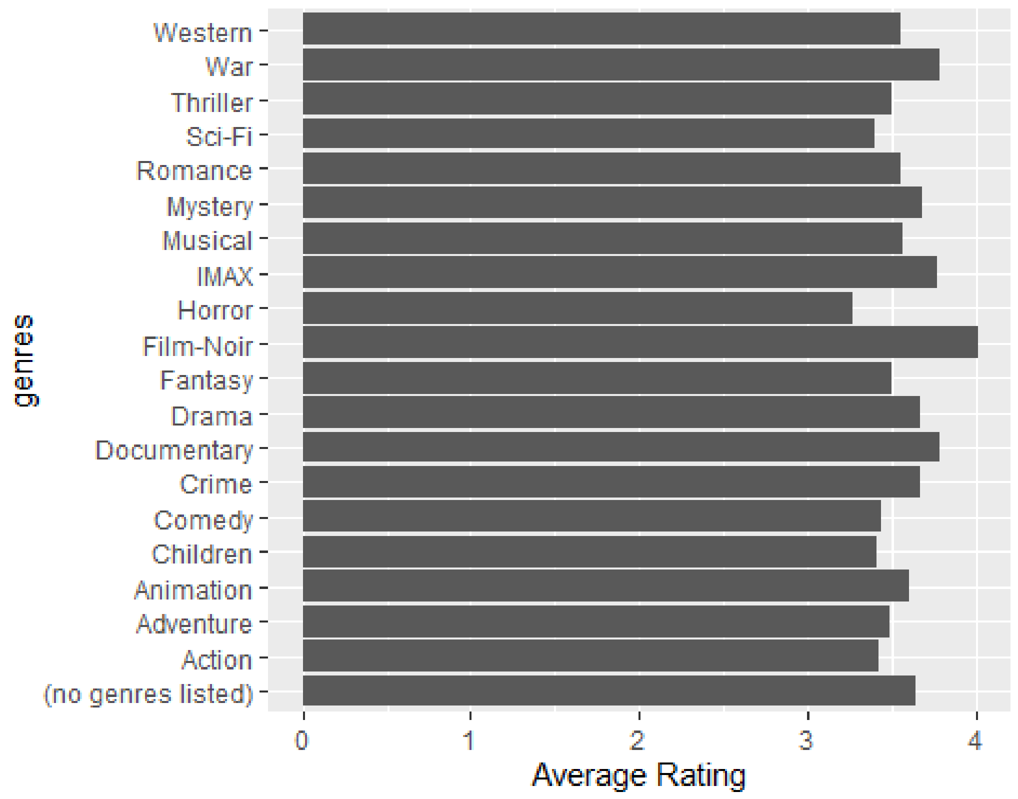
```
genreList %>% ggplot(aes(count,genres)) + geom_col();
```



Genres Average Ratings

Film-Noir is the genre with highest average rating. All genres have average rating between 3.0-4.0

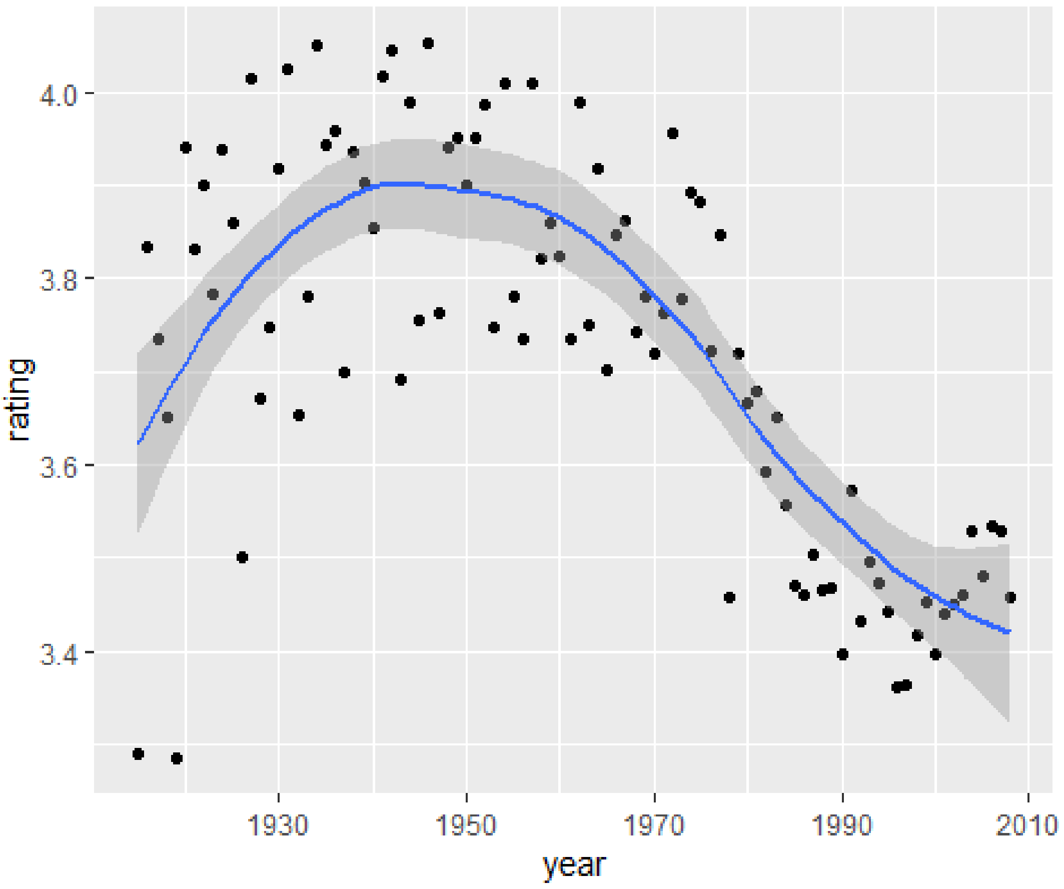
```
genreList %>% ggplot(aes(Rating,genres)) + geom_col() + labs(x="Average Rating");
```



2.2.5 Release Year

In this section we analyze relationship between a movie's release year and average ratings. We see that newer movies has comparatively less average rating, while movies released in mid-90's are at higher side.

```
edx_year = edx %>% mutate(year= as.numeric(str_sub(title,-5,-2)));
edx_year %>% group_by(year) %>% summarize(rating = mean(rating)) %>% ggplot(aes(year,
rating)) +geom_point() +geom_smooth();

## `summarise()` ungrouping output (override with `.groups` argument)

## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



2.3 Data Filtering and Cleaning

In this section final test_set and train_set are created for validation and edx datasets respectively. In these we have dropped features' columns that will be not be used by our model. Using the insights gained from 2.2 Data Exploration, we choose only those features where we could se significant relationship with ratings of a movie.

This step is highly recommended because using too many predictors/features increase te complexity of the model and requires more computational resources.

In this case, models will use only movie and user information.

```
train_set <- edx %>% select(userId, movieId, rating, title,genres) %>% mutate(year =
as.numeric(str_sub(title,-5,-2)));
test_set <- validation %>% select(userId, movieId, rating, title,genres) %>% mutate(year =
as.numeric(str_sub(title,-5,-2)));


# remove other objects to free up memory
rm(edx_year,genreList,movies,ratings,users)
```

3. Methods and Analysis

3.1 Model Evaluation

The Evaluation of machine learning algorithms is based upon comparing predicted values for the `test_set` with actual ones. The loss function is defined that measures difference between both values. The loss functions used in this projects' evaluation are:

3.1.1 Mean Absolute Error - MAE

This is the mean of absolute values of differences between predicted values and corresponding true values. The formula for MAE is given by:

$$MAE = \frac{1}{N}\sum_i |\hat{y}_i - y_i|$$

where N is the number of observations, $\hat{y}_i$ is the predicted value and y_i is the true value.

3.1.2 Mean Squared Error - MSE

This is the mean of squared values of differences between predicted values and corresponding true values. If difference is more than 1 it scales up, ex 2 is counted as 4, while if the difference is less than one it scales down, ex 0.2 will be counted as 0.04. The formula for MSE is given by:

$$MSE = \frac{1}{N}\sum_i (\hat{y}_i - y_i)^2$$

3.1.3 Mean Absolute Error - RMSE

This is the square root of MSE. This metric is most commonly used for evaluation of recommendation systems. In this project we've a target of RMSE < 0.864900. The formula of RMSE is given by:

$$RMSE = \frac{1}{N}\sqrt{\sum_i (\hat{y}_i - y_i)^2}$$

```r
# Define Mean Absolute Error (MAE)
MAE <- function(true_ratings, predicted_ratings){
  mean(abs(true_ratings - predicted_ratings))
}

# Define Mean Squared Error (MSE)
MSE <- function(true_ratings, predicted_ratings){
  mean((true_ratings - predicted_ratings)^2)
}

# Define Root Mean Squared Error (RMSE)
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

result <- tibble(Method = "Project Goal", RMSE = 0.8649, MSE = NA, MAE = NA)
result;
```

```
## # A tibble: 1 x 4
##   Method        RMSE   MSE    MAE
##   <chr>        <dbl> <lgl>  <lgl>
## 1 Project Goal 0.8649  NA     NA
```

3.2 Modelling

In this project we start with a simplest linear model to predict movie ratings for different users, we add on more feature and complexity as we move ahead and analyze RMSE(s) for different models and later we add regularization to minimize the RMSE

3.2.1 Linear Model

Movie Mean Model

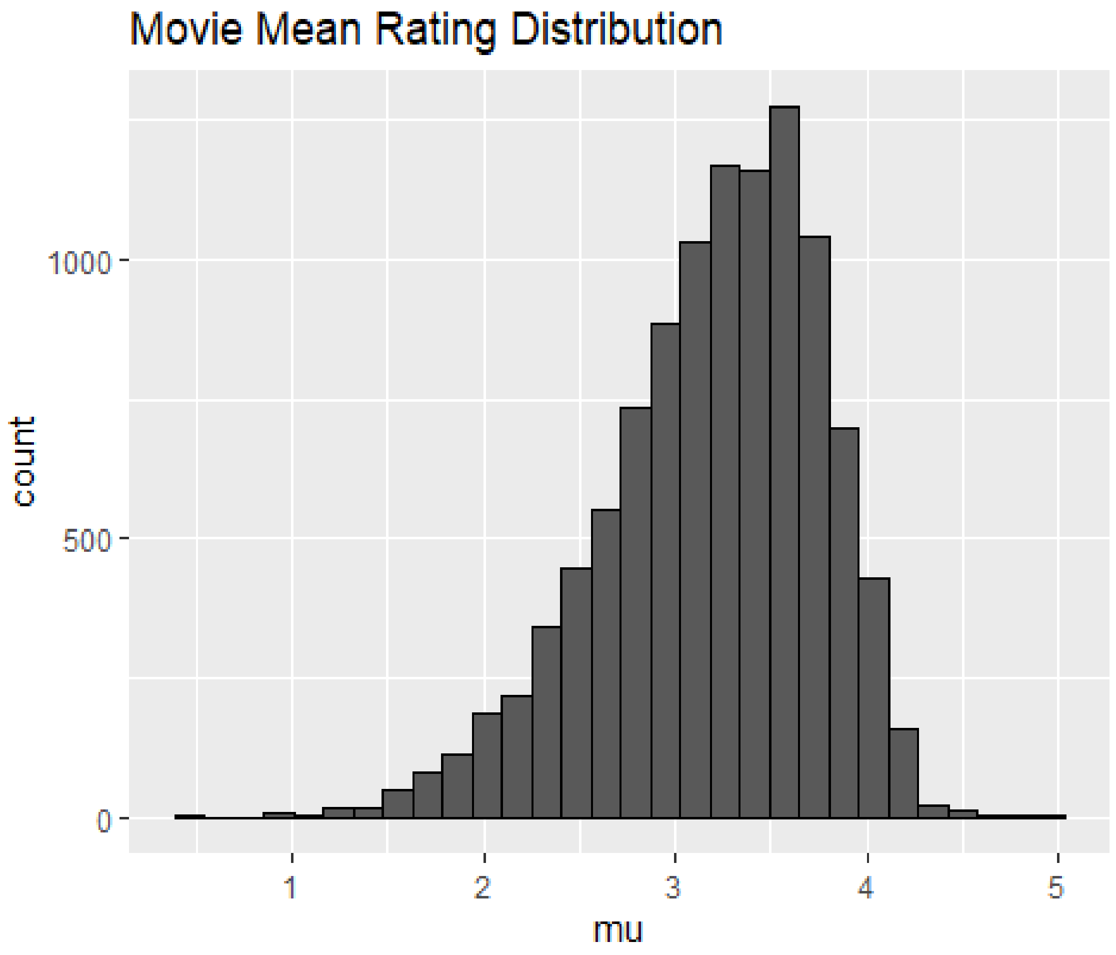
We start with the simplest linear model that would predict same ratings for a particular movie for all users and that would be the average rating of the movie calculated from train_set. So basically, we calculate movies' mean ratings and directly assign them as the predicted ratings.

The formula for predicted value of Movie Mean Model is given as:

$$\widehat{Y_{\iota.u}} = \mu_i + \epsilon_{i,u}$$

where $\widehat{Y_\iota}$ is the predicted value of u^{th} user for i^{th} movie, μ_i is the mean rating for the i^{th} movie, and $\epsilon_{i,u}$ is the error distribution

```
# Mean of observed values
movie_mean = train_set %>% group_by(movieId) %>% summarise(mu=mean(rating))

## `summarise()` ungrouping output (override with `.groups` argument)

movie_mean %>% ggplot(aes(mu)) + geom_histogram(color="black") +ggtitle("Movie Mean Rating
Distribution");

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
y_hat_mu = test_set %>% left_join(movie_mean,by="movieId") %>% .$mu

# Update the error table
result <- bind_rows(result,
                    tibble(Method = "Movie Mean",
                           RMSE = RMSE(test_set$rating, y_hat_mu),
                           MSE  = MSE(test_set$rating, y_hat_mu),
                           MAE  = MAE(test_set$rating, y_hat_mu)))
# Show the RMSE improvement
result;

## # A tibble: 2 x 4
##   Method                      RMSE    MSE    MAE
##   <chr>                      <dbl>  <dbl>  <dbl>
## 1 Project Goal              0.8649    NA      NA
## 2 Movie Mean                0.9439  0.8909  0.7379
```

In movie mean model an RMSE of 0.9439 was achieved, thus we moved on to include User effect

User Effect Model

In this model we include user effect term which accounts for user bias. Different users have different rating pattern for example some users like most movie and rate most movies between 4-5, while some users can be highly critic and rate most movies between 2-3.

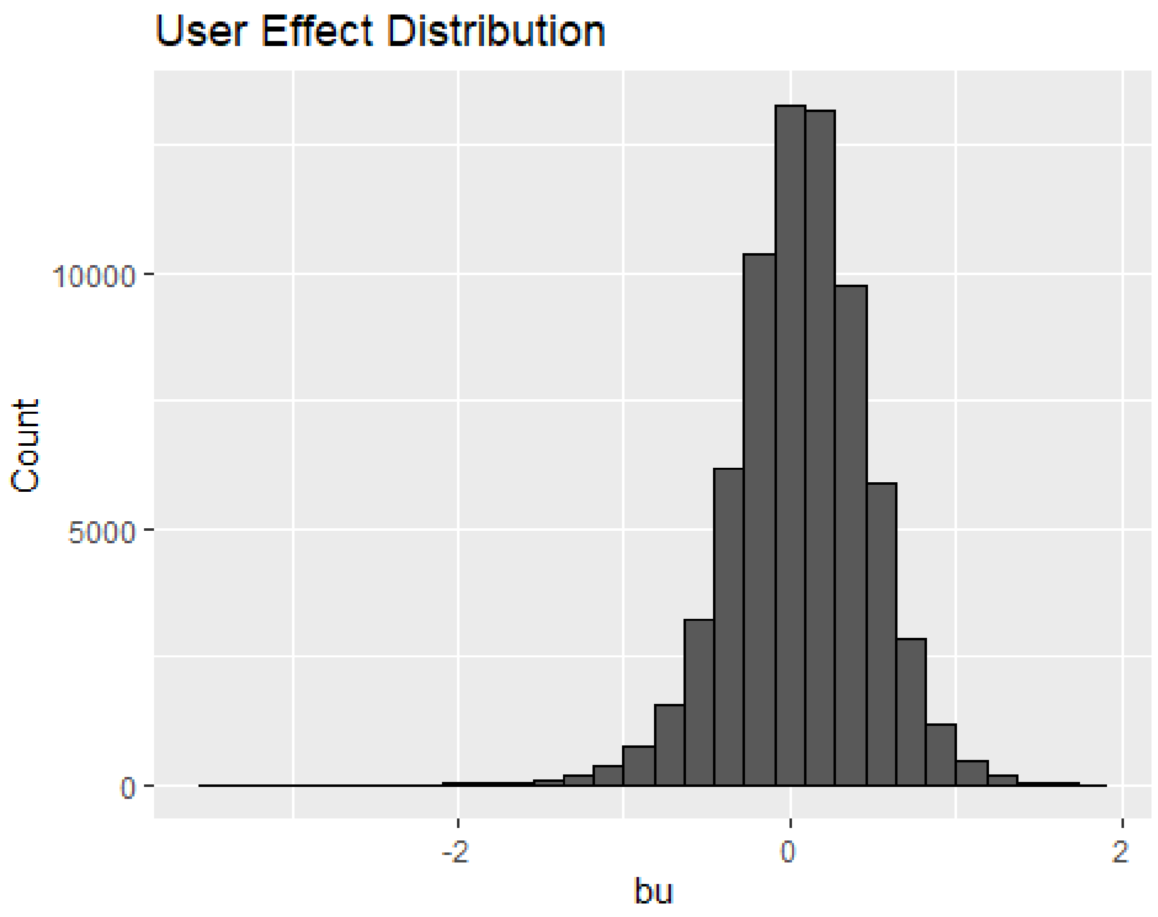
To take into account this effect we calculate bias term for each user as mean of deviation of his/her ratings from movie mean ratings. The formula is given by:

$$b_u = \frac{1}{N}\sum_i (y_{\{u,i\}} - \mu_i)$$

where N is the number of movies rated by u^{th} user, $y_{\{u,i\}}$ is the user's rating for i^{th} movie
So, prediction is given by:

$$\widehat{Y_{i.u}} = \mu_i + b_u + \epsilon_{i,u}$$

```
user_effect = train_set %>% left_join(movie_mean,by="movieId") %>% group_by(userId) %>%
summarise(bu = mean(rating-mu));

## `summarise()` ungrouping output (override with `.groups` argument)

user_effect %>% ggplot(aes(bu)) + geom_histogram(color="black") +ggtitle("User Effect
Distribution") +labs(x="bu",y="Count")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

y_hat_bu = test_set %>% left_join(movie_mean, by="movieId") %>%
left_join(user_effect,by="userId") %>% mutate(pred=mu+bu) %>% .$pred;

result <- bind_rows(result,
                    tibble(Method = "Movie Mean + bu",
                           RMSE = RMSE(test_set$rating, y_hat_bu),
                           MSE  = MSE(test_set$rating, y_hat_bu),
                           MAE  = MAE(test_set$rating, y_hat_bu)))
# Show the RMSE improvement
result;

## # A tibble: 3 x 4
##   Method                            RMSE    MSE    MAE
##   <chr>                            <dbl>  <dbl>  <dbl>
## 1 Project Goal                    0.8649    NA      NA
## 2 Movie Mean                      0.9439  0.8909  0.7379
## 3 Movie Mean + bu                 0.8653  0.7488  0.6691
```



In User effect model an RMSE of 0.8653 was achieved, now it's time to take into account Genre effect

Genre Effect Model

Generally, movies having same genres receive similar user ratings, and also sum genres have higher average ratings than others, this due to some genres are liked by many people while some genres are liked by very few people. Considering these insights, we take into account Genre effect term in our linear model.

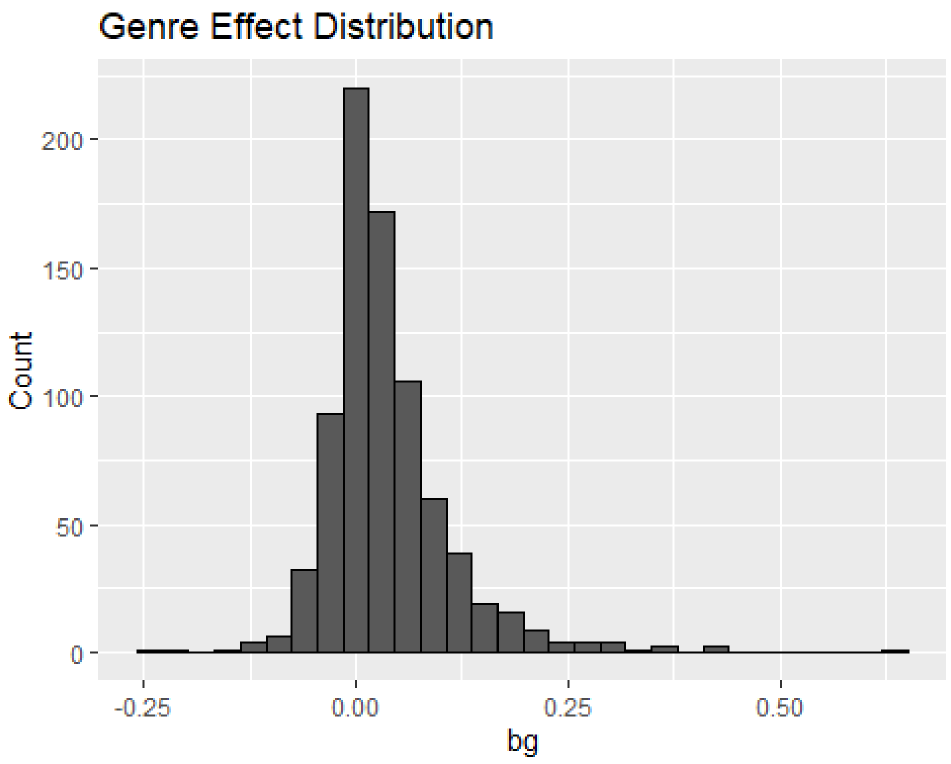
To account for this effect, we calculate bias term for each genre. The formula is given by:

$$b_g = \frac{1}{N}\sum_i \left(y_{\{u,i\}} - \mu_i - b_u\right)$$

where N is the number of movies under g^{th} genre, $y_{\{u,i\}}$ is the u^{th} user's rating for i^{th} movie and b_u is the bias term for u^{th} user. So, prediction is now given by:

$$\widehat{Y_{i,u}} = \mu_i + b_u + b_g + \epsilon_{i,u}$$

```
genre_effect = train_set %>% left_join(movie_mean,by="movieId") %>%
left_join(user_effect,by="userId") %>% group_by(genres) %>% summarise(bg = mean(rating-mu-
bu));

## `summarise()` ungrouping output (override with `.groups` argument)

genre_effect %>% ggplot(aes(bg)) + geom_histogram(color="black") +ggtitle("Genre Effect
Distribution") +labs(x="bg",y="Count")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
y_hat_bg = test_set %>% left_join(movie_mean, by="movieId") %>%
left_join(user_effect,by="userId") %>% left_join(genre_effect,by="genres") %>%
mutate(pred=mu+bu +bg) %>% .$pred;

result <- bind_rows(result,
                    tibble(Method = "Movie Mean + bu + bg",
                           RMSE  = RMSE(test_set$rating, y_hat_bg),
                           MSE   = MSE(test_set$rating, y_hat_bg),
                           MAE   = MAE(test_set$rating, y_hat_bg)))
# Show the RMSE improvement
result

## # A tibble: 4 x 4
##   Method                              RMSE    MSE    MAE
##   <chr>                              <dbl>  <dbl>  <chr>
## 1 Project Goal                      0.8649    NA      NA
## 2 Movie Mean                        0.9439  0.8909  0.7379
## 3 Movie Mean + bu                   0.8653  0.7488  0.6691
## 4 Movie Mean + bu + bg              0.8649  0.7481  0.6685
```

With Genre and User effects in consideration RMSE of 0.8649 was achieved, now we add Year effect

Release Year Effect

Now, the Release Year Effect will be included as final addition to linear model. In data exploration we saw movies with different release year had different average ratings. The movies released in 2000's had lowest average ratings while the ones in mid-90's had highest.

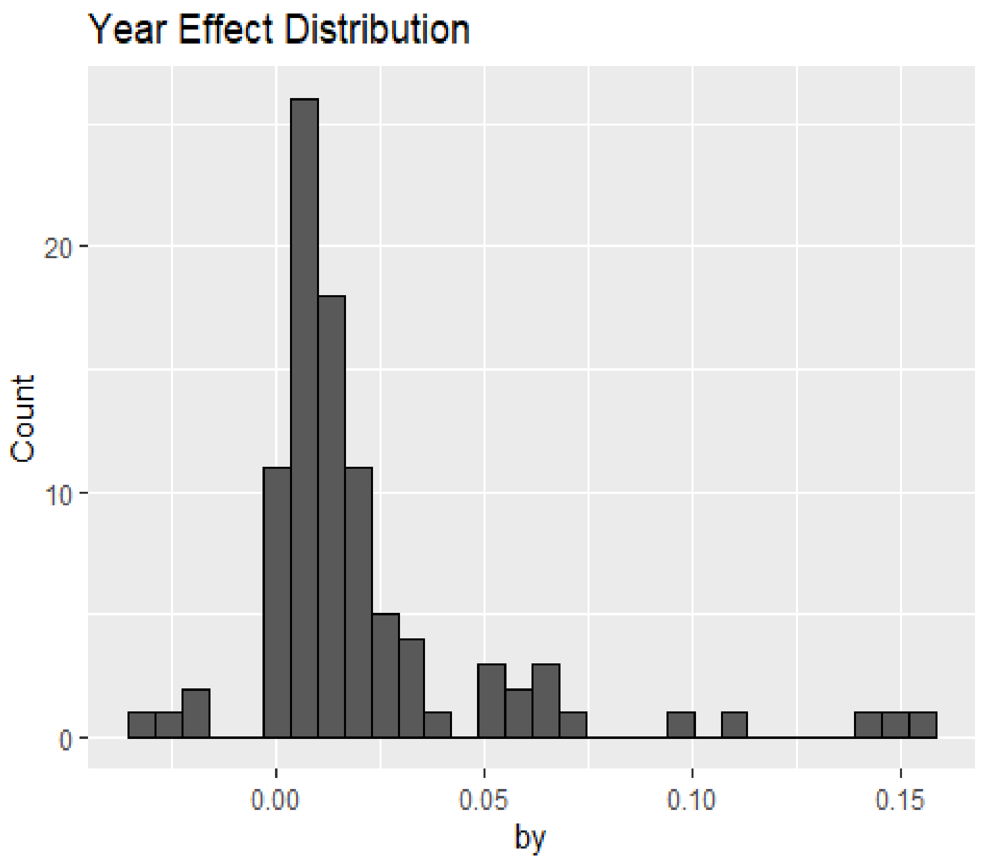
To model this effect, we calculate year bias term given by:

$$b_y = \frac{1}{N}\sum_{i,u}\left(y_{\{u,i\}} - \mu_i - b_u - b_g\right)$$

where N is the number of movies released in y^{th} year, $y_{\{u,i\}}$ is the u^{th} user's rating for i^{th} movie, b_u is the bias term for u^{th} user, and b_g the genre bias term. So, prediction is now given by:

$$\widehat{Y_{\iota,u}} = \mu_i + b_u + b_g + \epsilon_{i,u}$$

```
year_effect = train_set %>% left_join(movie_mean,by="movieId") %>%
left_join(user_effect,by="userId") %>%left_join(genre_effect,by="genres") %>% group_by(year)
%>% summarise(by = mean(rating-mu-bu-bg));

## `summarise()` ungrouping output (override with `.groups` argument)

year_effect %>% ggplot(aes(by)) + geom_histogram(color="black") +ggtitle("Year Effect
Distribution") +labs(x="by",y="Count")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
y_hat_by = test_set %>% left_join(movie_mean, by="movieId") %>%
left_join(user_effect,by="userId") %>% left_join(genre_effect,by="genres") %>%
left_join(year_effect,be="year")%>% mutate(pred=mu+bu +bg+by) %>% .$pred;

## Joining, by = "year"

result = bind_rows(result,
                    tibble(Method = "Movie Mean + bu + bg + by",
                           RMSE  = RMSE(test_set$rating, y_hat_by),
                           MSE   = MSE(test_set$rating, y_hat_by),
                           MAE   = MAE(test_set$rating, y_hat_by)));
# Show the RMSE improvement
result

## # A tibble: 5 x 4
##   Method                      RMSE    MSE    MAE
##   <chr>                      <dbl>  <dbl>  <dbl>
## 1 Project Goal               0.8649  NA     NA
## 2 Movie Mean                 0.9439  0.8909 0.7379
## 3 Movie Mean + bu            0.8653  0.7488 0.6691
## 4 Movie Mean + bu + bg       0.8649  0.7481 0.6685
## 5 Movie Mean + bu + bg + by  0.8647  0.7478 0.6681
```

With the final linear model considering user, genre and year effect, an RMSE of 0.8647 was achieved!

3.2.2 Regularization

The linear model provides a good estimation for the ratings, but doesn't consider that many movies have very few numbers of ratings, and some users rate very few movies. This means that the sample size is very small for these movies and these users. Statistically, this leads to large estimated error.

The estimated value can be improved adding a factor that penalizes small sample sizes and have little or no impact otherwise. Thus, estimated movie and user effects can be calculated with these formulas:

$$\widehat{b_u} = \frac{1}{n_u + \lambda} \sum\nolimits_i (y_{\{u,i\}} - \mu_i)$$

$$\widehat{b_g} = \frac{1}{n_g + \lambda} \sum\nolimits_i (y_{\{u,i\}} - \mu_i - \widehat{b_u})$$

$$\widehat{b_y} = \frac{1}{n_y + \lambda} \sum\nolimits_{i,u} (y_{\{u,i\}} - \mu_i - \widehat{b_u} - \widehat{b_g})$$

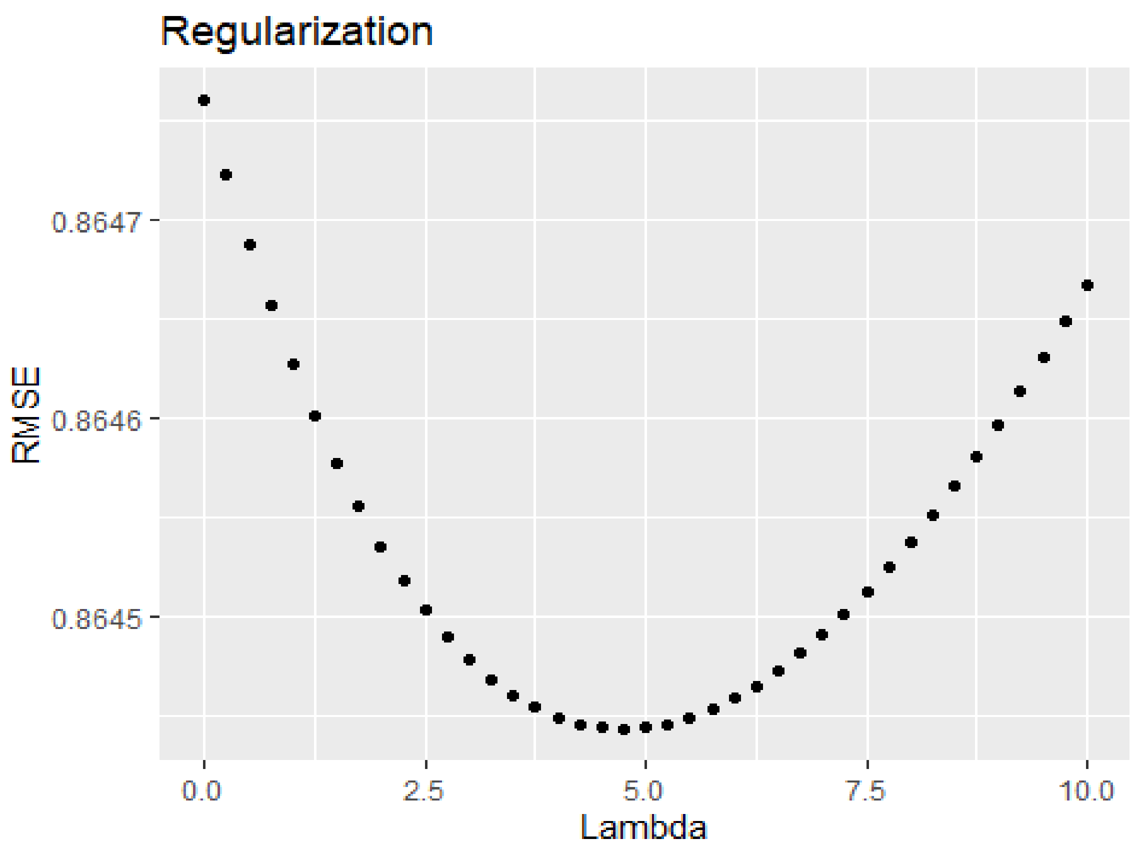
For values N of smaller than or similar to λ, $\hat{b}$ (bias terms) are smaller than the original values, whereas for values of N much larger than λ, $\hat{b}$ change very little.

An effective method to choose λ that minimizes the RMSE is running simulations with several values of λ

```
regularization <- function(lambda, trainset, testset){

  # Movie Mean
  movie_mean = trainset %>% group_by(movieId) %>% summarise(mu=mean(rating));

  # User effect (bu)
  user_effect = trainset %>% left_join(movie_mean,by="movieId") %>% group_by(userId) %>%
summarise(bu =
sum(rating-mu)/(n()+lambda));
  #Genre effect (bg)
  genre_effect = trainset %>% left_join(movie_mean,by="movieId") %>%
left_join(user_effect,by="userId") %>%
    group_by(genres) %>% summarise(bg = sum(rating-mu-bu)/(n()+lambda));

  #Year effect (by)
  year_effect = trainset %>% left_join(movie_mean,by="movieId") %>%
left_join(user_effect,by="userId") %>%
    left_join(genre_effect,by="genres") %>% group_by(year) %>% summarise(by = sum(rating-mu-
bu-bg)/(n()+lambda));

  # Prediction: mu + bu + bg + by
  predicted_ratings = testset %>% left_join(movie_mean, by="movieId") %>%
left_join(user_effect,by="userId") %>%
    left_join(genre_effect,by="genres") %>% left_join(year_effect,be="year")%>%
mutate(pred=mu+bu +bg+by) %>% .$pred;

  return(RMSE(testset$rating,predicted_ratings));
}

## Run Simulations
# Define a set of lambdas to tune
lambdas = seq(0, 10, 0.25)

# Tune lambda
rmses = sapply(lambdas, regularization, trainset = train_set, testset = test_set)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)

## Joining, by = "year" …
```

```
# Plot the lambda vs RMSE
tibble(Lambda = lambdas, RMSE = rmses) %>%
  ggplot(aes(x = Lambda, y = RMSE)) +
    geom_point() +
    ggtitle("Regularization")
```



```
#picking lambda with lowest RMSE
lambda = lambdas[which.min(rmses)];

  # Movie Mean
  movie_mean = train_set %>% group_by(movieId) %>% summarise(mu=mean(rating));

## `summarise()` ungrouping output (override with `.groups` argument)

  # User effect (bu)
  user_effect = train_set %>% left_join(movie_mean,by="movieId") %>% group_by(userId) %>%
summarise(bu =
sum(rating-mu)/(n()+lambda));

## `summarise()` ungrouping output (override with `.groups` argument)

  #Genre effect (bg)
  genre_effect = train_set %>% left_join(movie_mean,by="movieId") %>%
left_join(user_effect,by="userId") %>%
    group_by(genres) %>% summarise(bg = sum(rating-mu-bu)/(n()+lambda));

## `summarise()` ungrouping output (override with `.groups` argument)

  #Year effect (by)
  year_effect = train_set %>% left_join(movie_mean,by="movieId") %>%
left_join(user_effect,by="userId") %>%
    left_join(genre_effect,by="genres") %>% group_by(year) %>% summarise(by = sum(rating-mu-
bu-bg)/(n()+lambda));
```

```
## `summarise()` ungrouping output (override with `.groups` argument)

  # Prediction: mu + bu + bg + by
  y_hat_reg = test_set %>% left_join(movie_mean, by="movieId") %>%
left_join(user_effect,by="userId") %>%
    left_join(genre_effect,by="genres") %>% left_join(year_effect,be="year")%>%
mutate(pred=mu+bu +bg+by) %>% .$pred;

## Joining, by = "year"

  result <- bind_rows(result,
                      tibble(Method = "Regularized (mu + bu + bg + by)",
                             RMSE  = RMSE(test_set$rating, y_hat_reg),
                             MSE   = MSE(test_set$rating, y_hat_reg),
                             MAE   = MAE(test_set$rating, y_hat_reg)));
  result

## # A tibble: 6 x 4
##   Method                           RMSE     MSE     MAE
##   <chr>                           <dbl>   <dbl>   <dbl>
## 1 Project Goal                    0.8649    NA      NA
## 2 Movie Mean                      0.9439   0.8909  0.7379
## 3 Movie Mean + bu                 0.8653   0.7488  0.6691
## 4 Movie Mean + bu + bg            0.8649   0.7481  0.6685
## 5 Movie Mean + bu + bg + by       0.8647   0.7478  0.6681
## 6 Regularized (mu + bu + bg + by) 0.8644   0.7472  0.6682
```

Final Model with Regularization yields an RMSE of 0.8644!

4. Results

After testing on all the models one by one, the final results for all are compiled in the table below. We can see, Linear Regularized model gives us best results with RMSE of 0.8643 closely followed by Linear Model with all user, genre and year effects.

The evaluation table of all the models is given as below:

Method <chr>	**RMSE** <dbl>	**MSE** <dbl>	**MAE** <dbl>
Project Goal	0.8649000	*NA*	*NA*
Movie Mean	0.9439087	0.8909636	0.7379797
Movie Mean + bu	0.8653488	0.7488286	0.6691501
Movie Mean + bu + bg	0.8649469	0.7481331	0.6685198
Movie Mean + bu + bg + by	0.8647606	0.7478110	0.6681335
Regularized (mu + bu + bg + by)	0.8644435	0.7472626	0.6682611

5. Conclusion

We started with data ingestion where we prepared the dataset for training and validation, then we explored the data and analyzed the relationship between features and finally filtered out the required features based on the insights from data exploration into final train and test data sets.

After we had data prepared, we defined evaluation schemes and started modelling with simplest Linear Model, and then proceeded step-by-step adding complexities as user, genre, and release year effect term into our linear model, which improved our RMSE values, or basic Linear model had RMSE of 0.9443 and our final linear model ha RMSE of 0.8647, successfully passing the 0.8649 target.

Finally, we applied Regularization to our final linear model that included all user and movie effects. This brought our RMSE further down to 0.8643.

5.1 Limitations

We used only three predictors user, genre and release year information. To make our model more accurate we can use more features such as time-stamp, individual genres, bookmarks, popularity, etc. But adding these makes your model more complex and computationally expensive.

Apart from precision, our model only works with existing users and movies that already rated by some users, so there is no initial recommendation for a new user. Algorithms that use several other features can solve this issue.

Also, the algorithm must run every time a movie is rated by a user, this might become issue with large datasets.

5.2 Future Work

There are also other machine learning algorithms such as K-Nearest Neighbors, and Collaborative Filtering that perform extremely well for movie rating predictions, but are computationally expensive. I personally tried K-NN approach but it was taking too much time for this big dataset on my laptop even after using parallel processing with GPU.

Check my K-NN approach on MovieLens Project here:
https://github.com/rishabhdodeja/MovieLens_KNN

With small datasets there not mush issue. I've tried both KNN and Collaborative Filtering approach for movie recommendation system with small datasets and they work extremely well.

References

1. https://www.edx.org/professional-certificate/harvardx-data-science↩
2. https://www.udemy.com/course/data-science-and-machine-learning-with-python-hands-on/
3. https://www.netflixprize.com/↩
4. https://grouplens.org/↩
5. https://movielens.org/↩
6. https://grouplens.org/datasets/movielens/latest/↩
7. https://grouplens.org/datasets/movielens/10m/↩

End of the Document