# HarvardX: PH125.9x Data Science: Capstone - MovieLens Rating Prediction Project

Coco Ying

6/19/2020

# 1. Overview

## 1.1 Introduction

This is the MovieLens capstone project for HarvardX: PH125.9x Data Science: Capstone. The purpose of this project is to create a movie rating prediction system using machine learning in order to train algorithms effectively in producing accurate results. This report includes the details of this project from data cleaning, data exploration, data visualization, model building, model performance, and conclusion.

## 1.2 The Goal of this Project

The qualitative goal of this project is to train an effective model to predict movie ratings. The quantitative goal is to lower the RMSE (Root Mean Square Error) of the predictions to RMSE $< 0.86490$ where

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}.$$

## 1.3 Dataset

The dataset used in this project is the MovieLens 10M dataset with links shown below.
https://grouplens.org/datasets/movielens/10m/
http://files.grouplens.org/datasets/movielens/ml-10m.zip

# 2. Methods and Analysis

## 2.1 Generating Edx Dataset

The following is generated using the given code for acquiring MovieLens 10M dataset and generating dataframes.

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

```
##   Attaching packages                                    tidyverse 1.3.0
```

```
##   ggplot2 3.2.1      purrr   0.3.3
##   tibble  3.0.1      dplyr   0.8.5
##   tidyr   1.1.0      stringr 1.4.0
##   readr   1.3.1      forcats 0.4.0
```

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

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
## The following object is masked from 'package:purrr':
##
##     lift
## 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
```

The next part is the generation of edx dataset, splitting the MovieLens data into 90% training set (edx) and 10% validation set (validation) randomly using a particular seed in order for everyone to recreate the same result.

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

## 2.2 Data Cleaning and Exploration

In order to effectively train models, we need to have a better understanding of the dataset. The purpose of this section is to use various techniques to understand the structure of this dataset, eliminate data errors, assemble data in preparation of modeling.

### 2.2.1 Understand Data Structure

The first step is to find out the properties of this dataset, such as the types of data within this dataset and its dimensions. The following results show the 6 types of categories, userId, movieId, ratings, timestamp, movie title, and genres. The dimensions and percentage of both testing and evaluation dataset are also shown below.

Data categories

```
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
```

Dimensions of training and validation set

```
tr<-dim(edx)
print(tr)
```

```
## [1] 9000055       6
```

```
val<-dim(validation)
print(val)
```

```
## [1] 999999      6
```

True Validation Percentage is close to 10%, just as how training and testing sets are generated

```
val_p<-val[1]*100/(tr[1]+val[1])
print(val_p) #True Validation Percentage
```

```
## [1] 9.999936
```

### 2.2.2 Check if there's NA values

In prior of modeling, NA values has to be elimitaed. The NA values for each categories are all 0, meaning that there is no need to data-clean.

```
sapply(edx, function(x) sum(is.na(x)))
```

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

### 2.2.3 Identify distinct users and make sure a movie matches a movieId

It is necessary to make sure that a movie title matches a movieId so that the data acquired have no mistakes. The following shows the number of distinct userId, number of distinct movieId, and number of distinct movie titles. Here we identified a single mismatch between distinct movieId and title. As it is a minor error that requires a long time to examine, it is ignored for now.

```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movieId = n_distinct(movieId),
            n_movies = n_distinct(title))
```

```
##   n_users n_movieId n_movies
## 1   69878     10677    10676
```

### 2.2.4 Extracting useful information from movie title

As we can see from the data structure portion that movie titles are followed by bracket with the year of movie release. This is a set of valuable information that could be used in modeling. Therefore, this information can be extracted from the title and made into a seperate column later on referred to as "year".

```
edx_1 <- mutate(edx, year = as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]$"), regex("\\d{4}

validation_1 <- mutate(validation, year = as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]$"),
```

### 2.2.5 Ratings distribution

The distribution of ratings are shown below, we can see that top three popular ratings 4, 3, and 5.

```
ratings_count<-
  edx %>%
  group_by(rating) %>%
  summarise(count=n()) %>%
  arrange(desc(count))
print(ratings_count)
```

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <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
```

### 2.2.6 Statistics of ratings per userId

The mean, median, maximum, and quantiles of unique users are shown below. In this dataset, there is a large gap between the number of average ratings per user and maximum ratings per user. The difference of third quantile and the fourth quantile shows that a small group (less than 25%) of users rates much more than everyone else.

```
user<- edx %>% count(userId)
mean(user$n)
```

```
## [1] 128.7967
```

```
median(user$n)
```

```
## [1] 62
```

```
max(user$n)
```

```
## [1] 6616
```

```
quantile(user$n)
```

```
##   0%  25%  50%  75% 100%
##   10   32   62  141 6616
```

### 2.2.7 Ranked number of ratings for each genre

These are the number of ratings per movie genres ranked from high to low. As we can see, the top three genres with the most ratings are “Drama”, “Comedy”, and “Action”.

```
genre_count<-
  edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
print(genre_count)
```

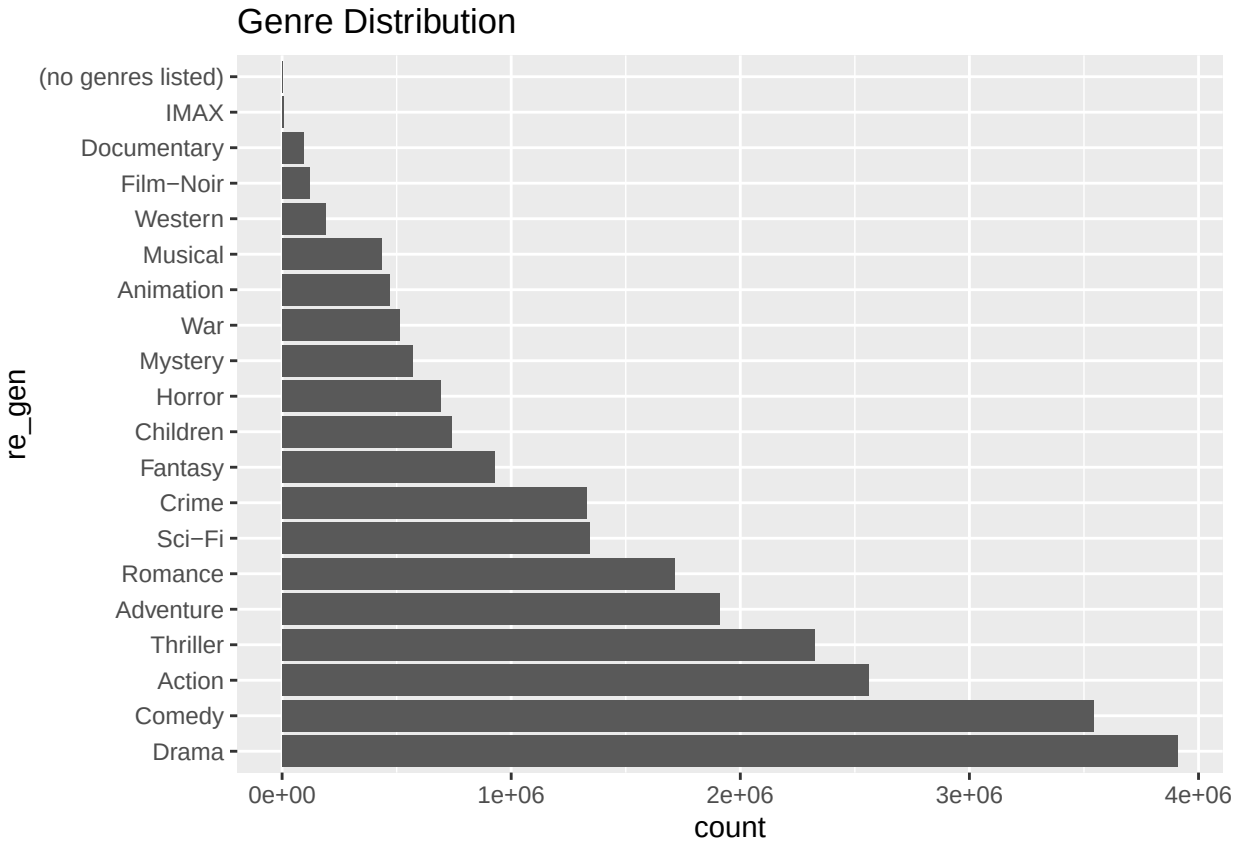
```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama              3910127
##  2 Comedy             3540930
##  3 Action             2560545
##  4 Thriller           2325899
##  5 Adventure          1908892
##  6 Romance            1712100
##  7 Sci-Fi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

## 2.3 Data Visualization

Other than exploring edx data through quantitative statistical methods, data visualization can be used to express data in a simple manner. Therefore, some important visualized data are displayed in this section in order to enhance our understanding of this dataset.

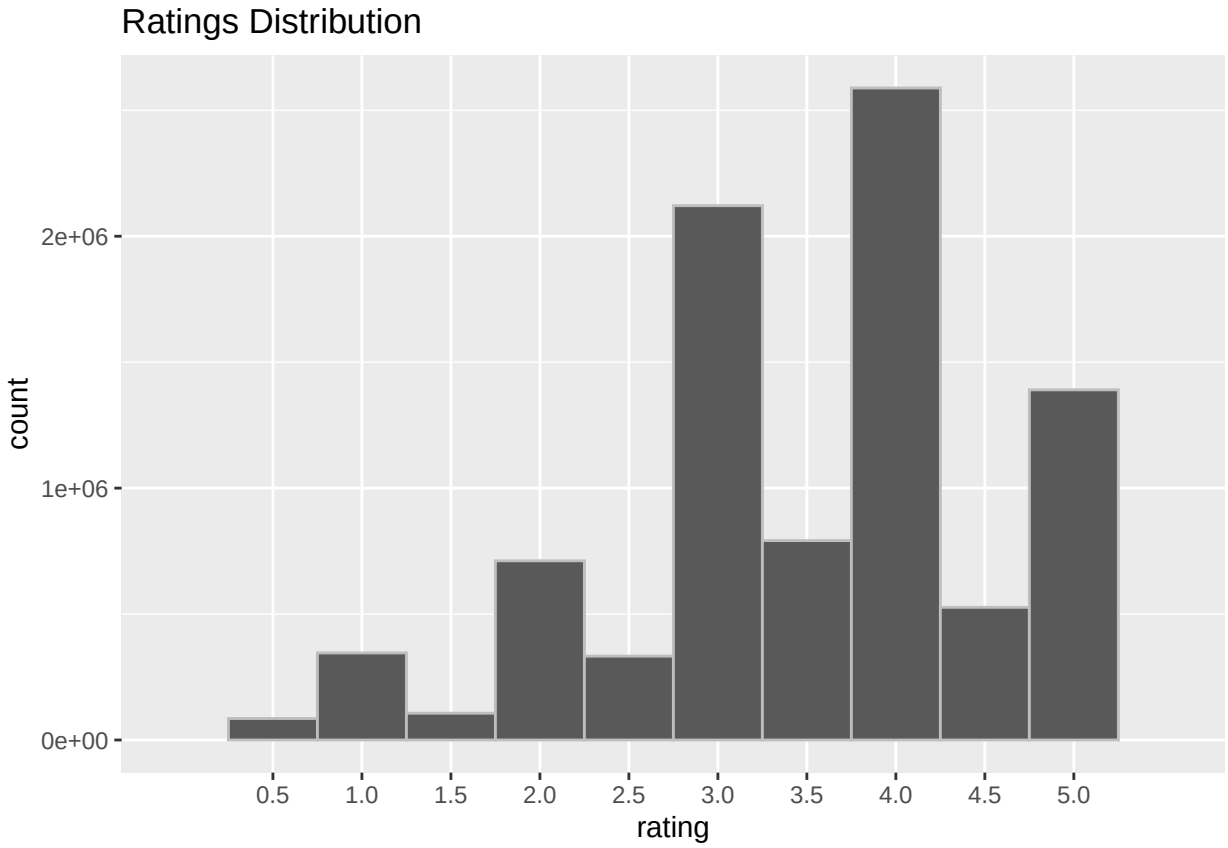
### 2.3.1 Genre Distribution

The genre distribution is a histogram showing genres of highest number of ratings to lowest number of ratings. This is a visualized form of 2.2.7 ranked number of ratings for each genre.



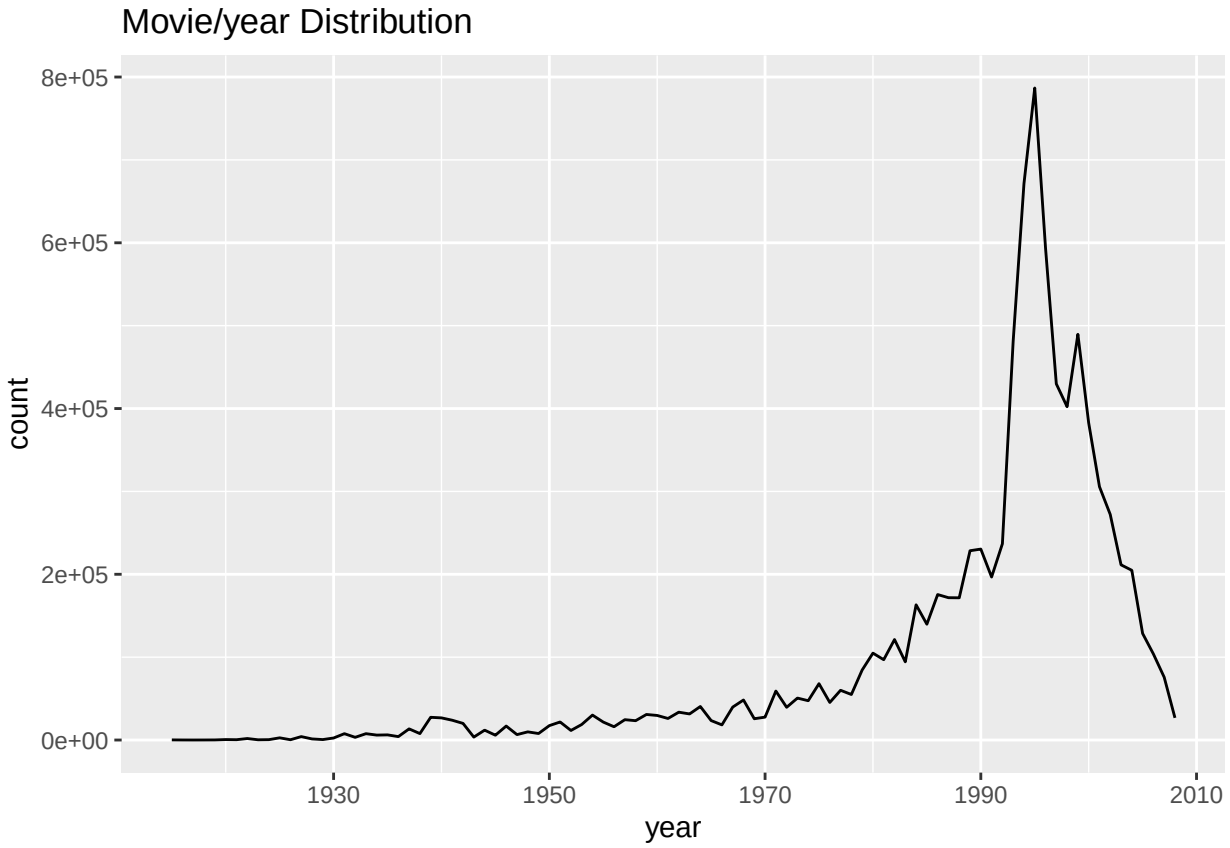

### 2.3.2 Ratings Distribution

This ratings distribution is a historgram showing total number of ratings ranging from 0.5 to 5.0. This is a visualized form of 2.2.5 ratings distribution. It is easier to tell from this visualization that the number of half ranks (ranks with a decimal point) are much lower than whole ranks.



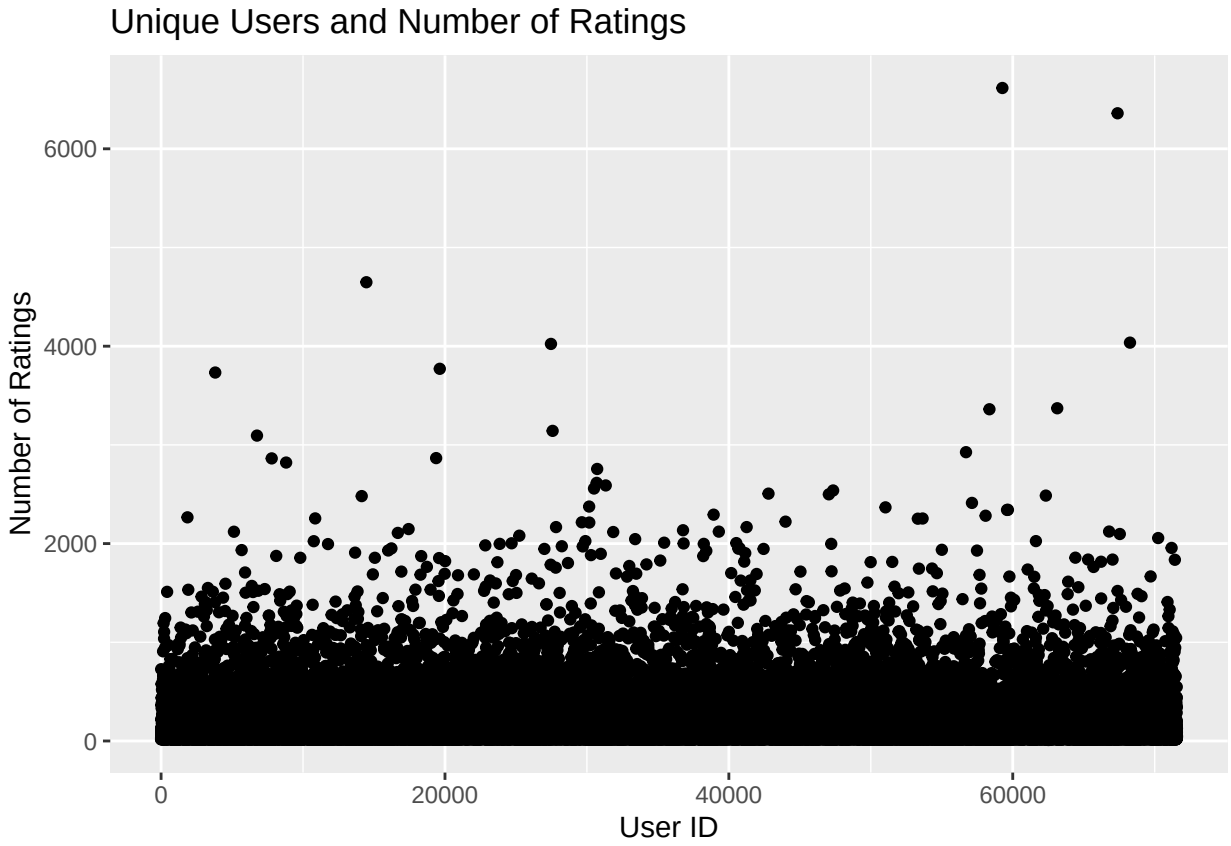

### 2.3.3 Movie/year Distribution

The movie/year distribution shows a line plot of the sum of movies produced each year up to 2008. Here we can tell that there is a peak in movie production around the year 1995. Precisely, in the year 1995, 786762 movies were released according to this dataset.




### 2.3.4 Unique users and number of ratings scatter plot

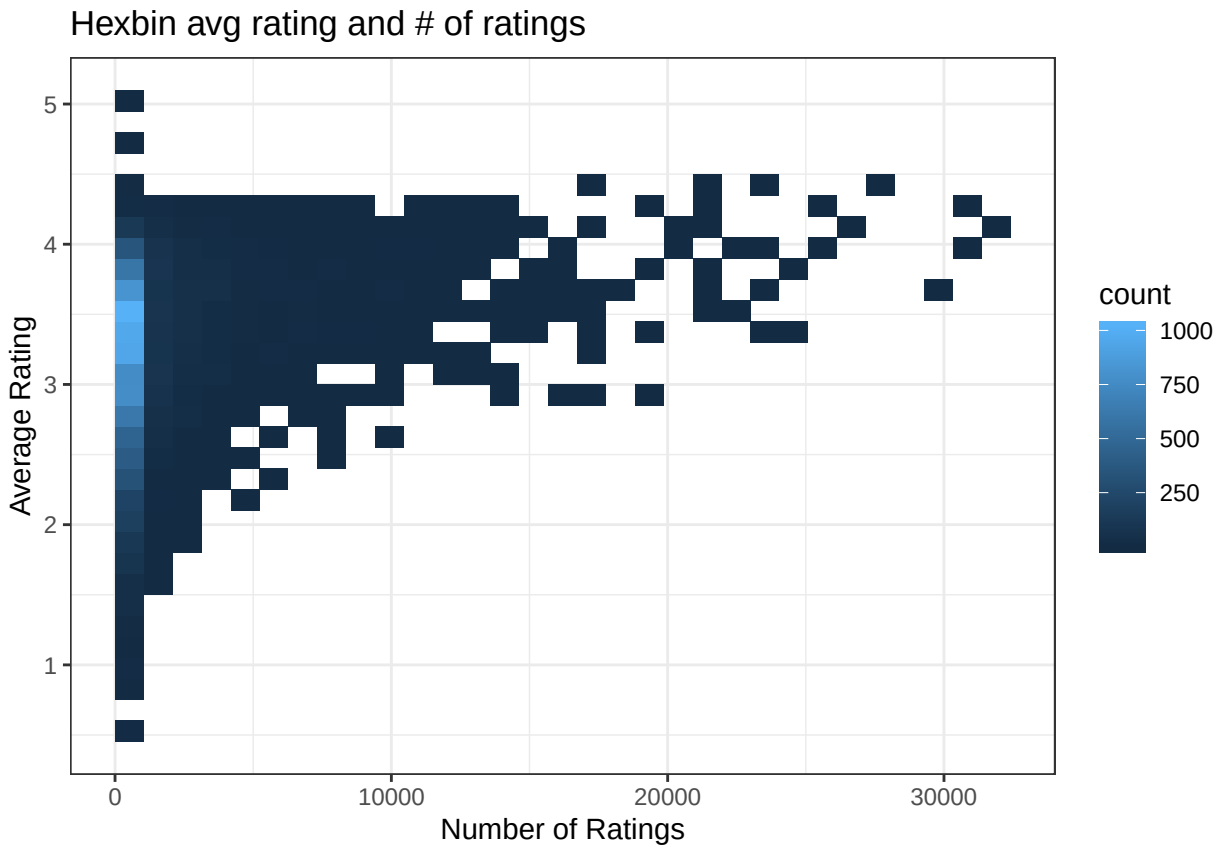
This is the scatter plot for unique users using userIds and their individual number of ratings. This plot is a visualized form of 2.2.6, showing a simpler indicator that only a small portion of users rate a lot of movies. The average number of ratings per user is actually much lower than the maximum number of ratings.




### 2.3.5 Average movie rating and number of ratings hexbin plot

This is a hexbin plot showing the relationship between average ratings for each movie and the number of ratings. As we can see in this plot, when the number of ratings are low, there is a big range in average ratings. Whereas when number of ratings per unique movie increase, the averaged movie ratings are generally high (close to 4). Through this diagram, we can infer that movies that most people have watched are generally better movies, whereas movies that has a small number of ratings range in average ratings probably due to user differences.

```
## Joining, by = "movieId"
```




# 3. Results

The results section includes evaluation metrics, modeling process, and model results. The modeling techinque used in this section consist of naive approach, the layering of result influencing terms (movie, users, and year of release), and penalty using regularization. The results show improvement through each layer of model. The reference of the modeling approach is shown below.
Reference: https://rafalab.github.io/dsbook/

## 3.1 Evaluation Metric/Loss Function

The evaluation metric or loss function used in this project is RMSE, a popular indicator of differences between predicted values and true values. RMSE is commonly used in the evaluation of models in machine learning. The equation of RMSE is stated below, it is obvious that the smaller RMSE, the better, but there are no precise value that indicates a "good" RMSE. This evaluation metric depends on the dataset used. In this section, our goal is to acheive a RMSE less than 0.86490.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

```
RMSE <- function(tru_ratings, pred_ratings){
  sqrt(mean((tru_ratings - pred_ratings)^2))
}
```

## 3.2 Modeling

To create a good movie rating prediction model, we need prior understanding of the dataset, which has been discussed previously. There are several important factors identified previously that effects ratings. Three factors, movie, user, and year of release are all important factors as they are related to the ratings and the number of ratings.

### 3.2.1 Model 1: Naive RMSE using average rating

In order to find a baseline RMSE, we use the most naive method to generate RMSE. The mean of ratings in the training set are identified and used to predict RMSE of the evaluation set.

```
mu_hat <- mean(edx$rating)
RMSE_1<-RMSE(validation$rating, mu_hat) #1.061
rmse_table<- data.table(Method="Naive Mean", RMSE=RMSE_1)
rmse_table %>%knitr::kable()
```

| Method | RMSE |
|---|---|
| Naive Mean | 1.061202 |

As we can tell, the baseline RMSE is 1.061. If RMSE of models below are higher than this baseline RMSE, it means that the model is ineffective.

### 3.2.2 Model 2: Movie Effects

The most intuitive model we can build first is to find the effects of individual movies. The rating difference between each individual movies are captured and indicated using "b_movie". This is the average difference per individual movie comparing with the average ratings.

```
movie_avg <- edx %>%
  group_by(movieId) %>%
  summarize(b_movie = mean(rating - mu_hat))

predict_2 <- validation %>%
  left_join(movie_avg, by='movieId') %>%
  pull(b_movie) + mu_hat

RMSE_2 <- RMSE(predict_2, validation$rating) #0.9439
rmse_table <- bind_rows(rmse_table,
                        data_frame(Method="Movie Model",
                                   RMSE = RMSE_2))
rmse_table %>%knitr::kable()
```

| Method | RMSE |
|---|---|
| Naive Mean | 1.0612018 |
| Movie Model | 0.9439087 |

The RMSE for this model is 0.9439, which is an improvement from previous RMSE of 1.061. Since the goal has not been reached, other factors should be considered as well.

### 3.2.3 Model 3: Movie and User Effects

The next factor considered in modeling is the user effects. Users have different preferences, some users are more positive while others might be more critical. This indicates that perhaps some users generally rate movies lower or higher than others. This model takes user differences into consideration, adding an user factor to the previous model. The approach of this model is almost exactly the same as above while adding a "b_user" factor to the prediction.

```
user_avg<- edx %>%
  left_join(movie_avg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_user=mean(rating-mu_hat-b_movie))

predict_3<- validation %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  mutate(pred3=mu_hat+b_movie+b_user) %>%
  pull(pred3)

RMSE_3 <- RMSE(predict_3, validation$rating) #0.86535
rmse_table <- bind_rows(rmse_table,
                        data_frame(Method="Movie+User Model",
                                   RMSE = RMSE_3))
rmse_table %>%knitr::kable()
```

| Method | RMSE |
|---|---|
| Naive Mean | 1.0612018 |
| Movie Model | 0.9439087 |
| Movie+User Model | 0.8653488 |

The RMSE for this model is 0.86535, improving from 0.9439 and approaching the goal. As we have identified three factors that may influence ratings, the next factor can be considered and added to the model.

### 3.2.4 Model 4: Movie, User, and Release Year Effects

The last factor considered is the release year effect. This factor is added to the previous model, creating a 3-factor model. The added factor is very similar to the procedures shown previously. The only difference is that we use the altered dataset shown in 2.2.4, "edx_1" as training set and "validation_1" as validation set. This altered dataset extracted release year from the original dataset. Therefore, we can use this information to help the modeling process.

```
year_avg<- edx_1 %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  group_by(year) %>%
  summarize(b_year=mean(rating-mu_hat-b_movie-b_user))

predict_4<- validation_1 %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(year_avg, by="year") %>%
  mutate(pred4=mu_hat+b_movie+b_user+b_year) %>%
  pull(pred4)
```

```
RMSE_4 <- RMSE(predict_4, validation_1$rating) #0.86500
rmse_table <- bind_rows(rmse_table,
                        data_frame(Method="Movie+User+Year Model",
                                   RMSE = RMSE_4))
rmse_table %>%knitr::kable()
```

| Method | RMSE |
|---|---|
| Naive Mean | 1.0612018 |
| Movie Model | 0.9439087 |
| Movie+User Model | 0.8653488 |
| Movie+User+Year Model | 0.8650043 |

The RMSE result for this model is 0.86500, a little bit better than the previous result. It has almost reached our goal.

#### 3.2.5 Model 5: Regularized Movie, User, and Release Year Effects

Since the results of the previous model has not reached the goal, we can take another factor into consideration. Another observation we had in the visualization section 2.3.5 is that the hexbin plot shows a large difference in movie ratings when the number of ratings are small. It can be inferred that movies with less ratings are less accurate in its ratings. In the previous model, the rating of a movie with only a single rating plays a big part in the prediction model. Therefore, the goal of this model is to reduce user effects in movies with less ratings. This can be acheived through regularization. The formula regularization is shown below, where $\lambda$ is an user-defined value that penalize small number of ratings $n$. This penalty term does not really affect large $n$, but it makes a big difference when $n$ is small.

$$b_movie_reg = \frac{1}{\lambda + n} \sum_{i=1}^{n} Y_i - \mu_i$$

After creating the penalized movie term, the procedure is exactly the same as model 4. The penalized movie term "b_movie_reg" replaces the original "b_movie".

```
reg_mov_avg<-edx_1 %>%
  group_by(movieId) %>%
  summarize(b_movie_reg=sum(rating-mu_hat)/(n()+4), n_movie=n()) #lambda=4

predict_5<- validation_1 %>%
  left_join(reg_mov_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(year_avg, by="year") %>%
  mutate(pred_5=mu_hat+b_movie_reg+b_user+b_year) %>%
  pull(pred_5)

RMSE_5 <- RMSE(predict_5, validation_1$rating) #0.86488
rmse_table <- bind_rows(rmse_table,
                        data_frame(Method="Reg_Movie+User+Year Model",
                                   RMSE = RMSE_5))
rmse_table %>%knitr::kable()
```

| Method | RMSE |
|---|---|
| Naive Mean | 1.0612018 |
| Movie Model | 0.9439087 |

| Method | RMSE |
|---|---|
| Movie+User Model | 0.8653488 |
| Movie+User+Year Model | 0.8650043 |
| Reg_Movie+User+Year Model | 0.8648752 |

The RMSE result for this model is 0.86488, which has reached the goal of acheiving a RMSE of less than 0.86490.

# 4. Conclusion

We have produced an effective machine learning model to predict movie ratings and satisfying the goal of RMSE being less than 0.86490. Our RMSE result of the final model (Model 5) has reached 0.86488. In order to produce an effective model, the first step is to clean up the dataset and eliminate error data. Then, understand the given dataset through statistical methods or visualization. Next, make observations of the dataset and use these observations to build an effective model.

The limitations of this work include the amount of categories given in the dataset. This dataset consist of only 6 categories. If reviews in text are given, more analysis can be made to increase model accuracy. Possible future work includes the modeling of the large version of this dataset to improve model accuracy.

# Appendix

## Environment

```
## [1] "Operating System:"
```

```
##                _
## platform       x86_64-apple-darwin15.6.0
## arch           x86_64
## os             darwin15.6.0
## system         x86_64, darwin15.6.0
## status
## major          3
## minor          6.2
## year           2019
## month          12
## day            12
## svn rev        77560
## language       R
## version.string R version 3.6.2 (2019-12-12)
## nickname       Dark and Stormy Night
```