# edX Data Science Capstone: MovieLens Project

Komalkumar Tagdiwala

September 2020

## Contents

# Introduction

Machine Learning has several applications in different industries. One of the most successful and widely used system that employs machine learning is a Recommender system. Some of the examples of recommender systems include the likes of Netflix making a movie recommendation based on what you have watched earlier in your queue or Amazon making a product recommendation to purchase based on prior purchase history or popular items. Recommendation can be made using different approaches, such as predicting an outcome based on a combination of multiple parameters/attributes associated with the problem at hand or by using a ranking system to make recommendations.

# Executive Summary

## Dataset

The MovieLens Capstone project involves application of machine learning techniques on a dataset of movie rankings provided by GroupLens, a research lab in the Department of Computer Science and Engineering at the University of Minnesota, that specializes in recommender systems. GroupLens Research has collected and made available rating data sets from the MovieLens web site (http://movielens.org) that will be the focus of this project. We will use the **10M version of the MovieLens dataset**.

## Goal

The goal of this Capstone is to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set. Reference code has been provided by the University to generate the training and validation sets that should be leveraged for this analysis. RMSE, known as the Root Mean Square Error, will be used to evaluate how close the predictions made by our model are to the actual/true values contained in the validation set. The desired RMSE should be a value less than 0.86490.

## Overview

Before we can review what is in the dataset, we need to load the dataset using the code provided by the University. The code to generate the training and validation dataset has been executed and stored in a workspace file - Univ_Provided_Dataset_Workspace.Rdata to simplify this report. Actual code can be found in the accompanying R file submitted with the assignment.

```
############################################################
# BEGIN: University provided code to generate the datasets for training and validation
# Create edx set, validation set (final hold-out test set)
############################################################

# Load the required libraries

#For Data Load/Processing/Generation of Training and Validation Data
library(tidyverse)
library(caret)
library(data.table)

library(ggplot2) #For Data Visualization (ggplot, qplot, etc.)
library(rafalib) # For mypar() to optimize graphical parameters for the RStudio plot window
```

```
library(dplyr) # For utility functions (%.%, %>%, etc.)
library(lubridate) # For datetime conversion (timestamp field in the data set)

library(caTools) # Utility functions for splitting dataset
library(recosystem) # Recommender System using Matrix Factorization

# The code to generate training and validation set has been executed
# from the R file and stored in a workspace file. We will load this file
# to review the dataset composition.
load("Univ_Provided_Dataset_Workspace.Rdata")

#############################################################
# END: University provided code to generate the datasets for training and validation
#############################################################
```

We begin by first examining what edx and validation are.

```
class(edx)
```

```
## [1] "data.table" "data.frame"
```

```
class(validation)
```

```
## [1] "data.table" "data.frame"
```

Let us now review the structure of the dataset to understand its composition.

```
# Training Set = edx
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 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

We see that edx (*our training set*) dataset has **9000055** observations of 6 variables/features.

```
# Validation Set = validation
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
##  $ 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|Roman
##  - attr(*, ".internal.selfref")=<externalptr>
```

Our *validation set* has **999999** observations of the same 6 features.

In summary, we note that of the **10000054 ratings** in the original 10M Movielens dataset,

- **Training set** = 90%
- **Validation set** = 10%

We are interested in the prediction of a movie rating (y). Consequently, we can make the following determination:

- rating = Outcome/Dependent Variable "y"
- Remaining 5 variables are the predictors/independent variables that will be used for our analysis.

Here is another view of the dataset using the **glimpse()** function from the *dplyr* package.

```
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 42...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)",...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|...
```

Let us see a few of the records to examine type different values in some of the rows

```
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
```

A quick look at the **top 5 rows** in the dataset reveals that each row represents the rating provided by a given user for a single movie. The movie may belong to multiple genres that are pipe-delimited in the genres column.

We will now document our findings for each of the 5 features (predictors) and our **outcome**/dependent variable.

1. **userId**: Contains unique ID for each user who provided a rating
2. **movieId**: Contains unique ID for the movie being rated
3. **timestamp**: Contains the date and time a user provided a rating for the movie under consideration
4. **title**: Holds the movie title
5. **genre**: Contains *pipe-delimited* genres associated with a movie. A movie can be associated with more than 1 genre and this is something that we should keep in mind for any analysis because this single field contains multiple values.

**Outcome (y)**

6. **rating**: Contains the rating provided by a user for a given movie. This is what we want to predict using machine learning.

## Key Steps

We will be undertaking the following steps to achieve our goal of predicting the movie rating:

- Data Wrangling/Cleaning/Pre-processing
- Data Exploration
- Data Visualization
- Insights Gained from the prior steps
- Build one or more Machine Learning Models
- Make Predictions using our model(s)
- Confirm RMSE achieved meets our desired goal of $< 0.86490$

For the model building, we will begin with linear regression to identify statistically significant features followed by gradual inclusion of each feature in our linear regression models. Based on how each iteration yields the RMSE values, this will be followed by Regularization for the most significant features, user and movie effects. Finally, we will employ a Recommender System library, recosystem, that will help deal with matrix sparsity demonstrated in the data exploration and visualization sections by employing Matrix Factorization and build our model to obtain an improved RMSE Score.

# Methods/Analysis

## Process and Techniques

### Data Cleaning/Pre-processing/Wrangling

Data is generated and maintained differently in different systems. Depending on how efficient business constraints exist in a system for capturing data, performing input validations, etc., the data captured by the system may contain a lot of junk/irrelevant kind of data for one or more fields in addition to being plagued by a very common issue, missing data.

We begin our data analysis by first examining the current data structure, looking for obvious signs of missing values, identify the need to either discard the missing values or substitute them with best practices, such as average of the given field across all observations. As such, Data exploration and Data pre-processing go hand in hand and may require multiple iterations before proceeding with actual analysis.

Building upon our knowledge from executive summary, we now dive deep into the constituents of our dataset to observe signs of any missing data as well as get a summary of how the different features of our data can potentially impact our analysis.

We start with the summary() function to obtain the big picture on our dataset for things, such as min, max, median, quantiles, class, etc.

```
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
##
##
##
```

### Missing Data

Our key objective is to predict ratings. So let us first check to see if there are any rows in the dataset that are missing values for rating.

```
# Movies that have a rating of 0/Movies without any rating
zero_ratings<- edx %>%
  filter(rating==0.0)
length(zero_ratings[[1]])
```

```
## [1] 0
```

We find that we do not have any missing ratings. This is a good start.

### Data Pre-processing

To avoid the issue of overfitting by building different models and subjecting them to the same validation/final hold-out set (as confirmed with the TA/edx Staff in edX discussion forum), we will split our current University provided edx training set into a train and test set.

```
library(caTools)
library(dplyr)
library(tidyr)
set.seed(123)

##########################################
# Generate the Training and Test set from original edx training set provided by University
# to avoid overfitting
##########################################

#We will go for 80-20 split of Train and Test data
split = sample.split(edx$rating, SplitRatio = 0.8)

# New Training Set to be used by all models
training_set = subset(edx, split == TRUE) # 7200044 obs. of 6 variables (80% of edx training set)

# New Test Set to be used by all models
test_set = subset(edx, split == FALSE)# 1800011 obs. of 6 variables (20% of edx training set)

#' To make sure we don't include users and movies in the test set
#' that do not appear in the training set, we remove these entries
#' using the semi_join function:
test_set <- test_set %>%
  semi_join(training_set, by = "movieId") %>%
  semi_join(training_set, by = "userId") # 1799975 obs. of 6 variables (36 entries were removed)
```

Effectively, our new test set has 20% of original edx data and new training set has 80% of edx data

For **ALL** our models, we will be using these new train_set and test_set to make predictions and compare RMSE values.

For the **CHOSEN** model, we will additionally subject that to the original validation set to get the RMSE value. This way, we will NOT be overfitting and be in compliance with the requirements confirmed by the TA in the edx discussion forum.

### Data Exploration and Visualization

We begin exploring the different features of the dataset including the outcome "rating" that we want to predict. Understanding what the different dataset features contain helps us gain meaningful insights about how each attribute/feature contributes to our data analysis in addition to determining the appropriate modeling technique.

**Unique Users (Feature: userId)** Now let us see how many unique users rated these different movies.

```
# Number of unique users in the dataset
unique_users_in_edx <- edx %>% count(userId)
length(unique_users_in_edx[[1]])
```

```
## [1] 69878
```

A total of 69,878 users were involved in rating these different movies.

**Unique Movies (Feature: movieId)** Let us see now see how many unique movies exist in our dataset.

```
# Unique movies
unique_movies_in_edx <- edx %>% count(movieId)
length(unique_movies_in_edx[[1]])
```

```
## [1] 10677
```

We have 10,677 unique movies that were rated by different users.

**Matrix View and Sparsity** We can imagine this dataset as a matrix of rows and columns where each user would have 1 row and the rating provided by the user would be captured in 1 column for every movie included in the dataset.

If we multiply the number of unique users (69878) by number of unique movies (10677), we get about 747 million rows. However, our data table has only 9,000,055 rows. This may indicate that not every user may have rated every single movie. Visualizing such data as a large matrix, with users on the rows and movies on the columns, we can expect to see a huge amount of empty cells.

Let us validate if that is indeed the case. To do this, we will use the gather function on a sample of records involving 5 movies and 20 users.

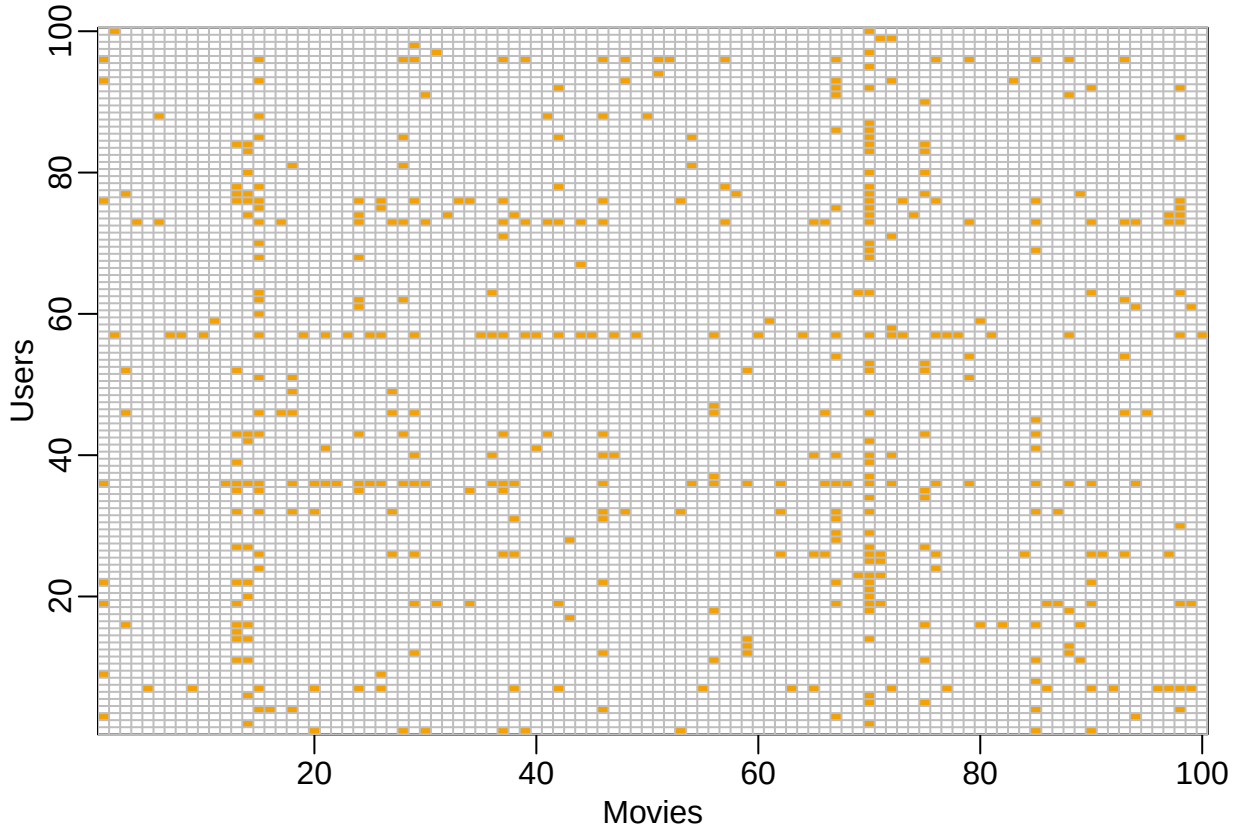
```
# matrix for 5 movies and 20 users
movie_sample <- edx %>%
  count(movieId) %>%
  top_n(5, n) %>%
  .$movieId

tab <- edx %>%
  filter(movieId%in%movie_sample) %>%
  filter(userId %in% c(10:30)) %>%
  select(userId, title, rating) %>%
  spread(title, rating)
tab %>% knitr::kable()
```

| userId | Forrest Gump (1994) | Jurassic Park (1993) | Pulp Fiction (1994) | Shawshank Redemption, The (1994) | Silence of the Lambs, The (1991) |
|---|---|---|---|---|---|
| 10 | 3 | NA | 2 | NA | 3 |
| 11 | NA | 4 | 3 | NA | NA |
| 13 | NA | NA | 4 | NA | NA |
| 16 | NA | 3 | NA | NA | NA |
| 17 | NA | NA | NA | NA | 5 |
| 18 | NA | 3 | 5 | 4.5 | 5 |
| 19 | 4 | 1 | NA | 4.0 | NA |
| 22 | NA | 4 | 5 | NA | 5 |
| 23 | NA | NA | 4 | NA | 5 |
| 30 | 5 | 4 | NA | 5.0 | 5 |

As expected, we see a lot of movies containing NA in their ratings for many users. This indicates that our matrix is sparse. But let us dig further to see how sparse the matrix could possibly be.To do this, we will take a random sample of 100 movies and 100 users and visualize it as follows. The yellow dots indicate a user/movie combination for which we have a rating.

Looking at the picture below, it is clear that we have a very sparse matrix. We will keep this in mind when choosing a model later in our analysis involving Recommendation Systems that help alleviate some of these concerns.

```
# matrix for a random sample of 100 movies and 100 users with yellow
# indicating a user/movie combination for which we have a rating.

sample_users <- sample(unique(edx$userId), 100)
rafalib::mypar() # optimizes graphical parameters for the RStudio plot window
edx %>% filter(userId %in% sample_users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
```




**Movies with Most Ratings/Popular movies (Feature: title)** Next, let us explore which movies received the most ratings to understand their popularity. We do this by grouping the dataset by title and summarizing the rating and sorting the results in descending order of the rating count.

```
# Movies with Most Ratings: Group by Title, Summarize, and Sort in DESC order
edx_by_title <- edx %>% group_by(title) %>%
                        summarize(total_rating=sum(rating)) %>%
                        arrange(desc(total_rating))
```

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

```
head(edx_by_title) # Load the top 5 movies from the results
```

```
## # A tibble: 6 x 2
##   title                                                     total_rating
##   <chr>                                                            <dbl>
## 1 Pulp Fiction (1994)                                            130302.
## 2 Silence of the Lambs, The (1991)                               127729
## 3 Shawshank Redemption, The (1994)                               124810.
## 4 Forrest Gump (1994)                                            124714.
## 5 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)   108370.
## 6 Jurassic Park (1993)                                           107561
```

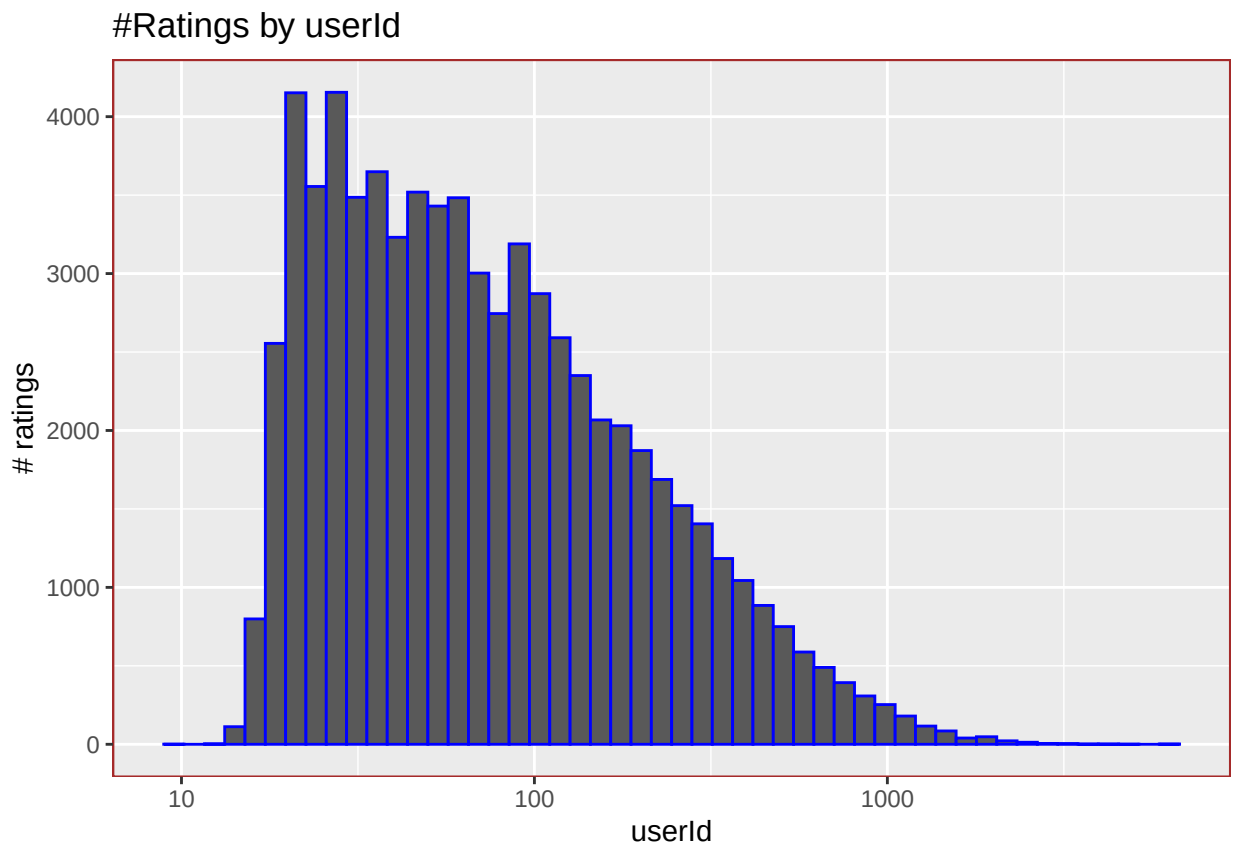
```
#rm(edx_by_title) # Cleanup the temporary variables
```

We see that Pulp Fiction (1994) has the highest total_rating of 130302.

**Visualize Ratings by userId** 10677 unique movies were rated by one or more of the 69878 unique users. But do we know if every user rated every single movie? Let us plot a histogram of Ratings by UserId.

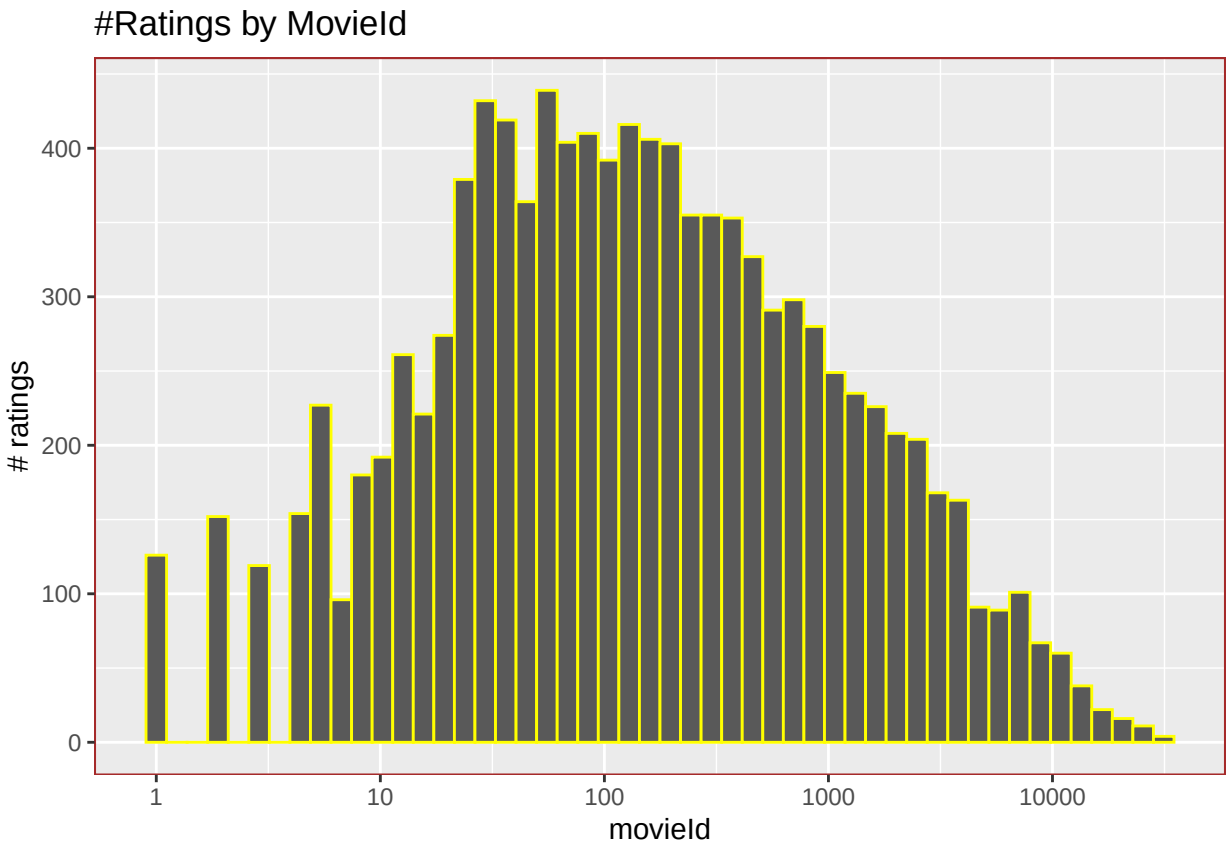
We see that: 1. Some users were very active in rating one or more movies. 2. Some users rated relatively less than others. 3. Not all users rated every single movie.

```
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
    geom_histogram(bins=50, color="blue") +
    scale_x_log10() +
        ggtitle ("#Ratings by userId" ) +
        labs(x ="userId" ,
             y ="# ratings") +
      theme (panel.border = element_rect(colour="brown", fill=NA) )
```




**Visualize Ratings by movieId** Similar to ratings by user, we now plot ratings by movieId and find that some movies received much more ratings than others.

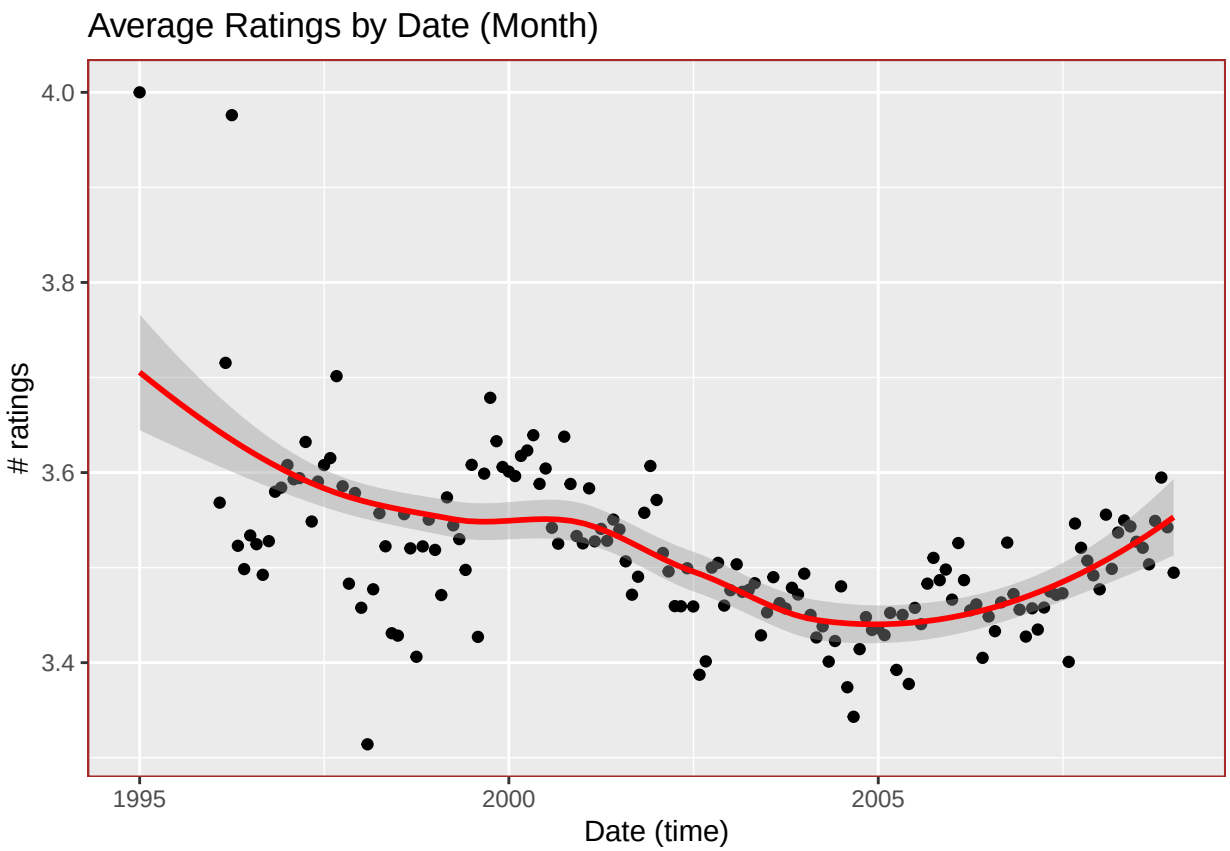
```
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
    geom_histogram(bins=50, color="yellow") +
    scale_x_log10() +
        ggtitle ("#Ratings by MovieId" ) +
        labs(x ="movieId" ,
            y ="# ratings") +
      theme (panel.border = element_rect(colour="brown", fill=NA) )
```




**Visualize Ratings by timestamp** The timestamp field stores date in a format not readily consumable for our analysis. We will first convert it to a date format and attempt to visualize the number of ratings provided by a given month.

```
edx %>%
  mutate(date_rated=round_date(as_datetime(timestamp),unit="month")) %>%
  group_by(date_rated) %>%
  summarize(averaged_rating=mean(rating),.groups = 'drop') %>%
  ggplot(aes(date_rated,averaged_rating)) +
    geom_point() +
    geom_smooth(colour="red", size=1)+
        ggtitle ("Average Ratings by Date (Month)" ) +
        labs(x ="Date (time)" ,
             y ="# ratings") +
      theme (panel.border = element_rect(colour="brown", fill=NA) )
```

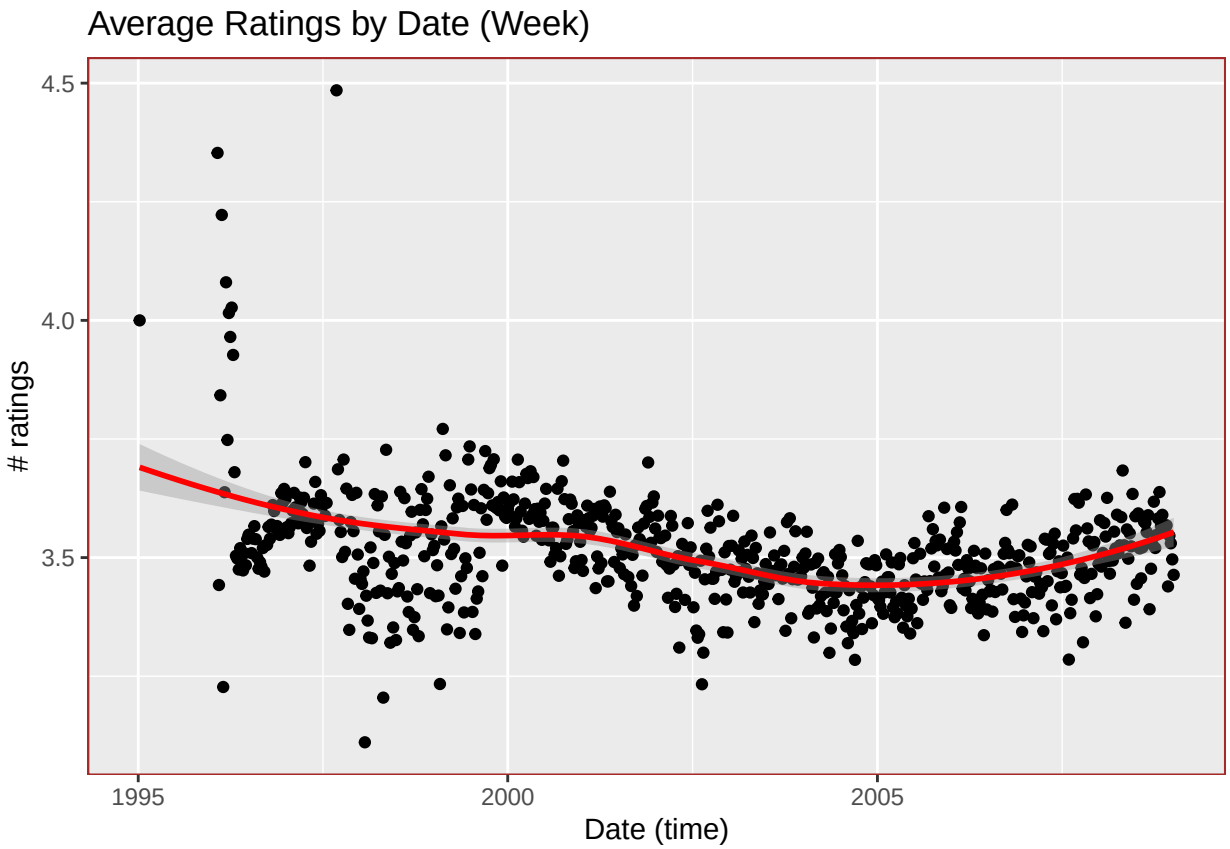
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




Next, let us visualize the number of ratings by a given week to get a more granular view of the data.

```
edx %>%
  mutate(date_rated=round_date(as_datetime(timestamp),unit="week")) %>%
  group_by(date_rated) %>%
  summarize(averaged_rating=mean(rating),.groups = 'drop') %>%
  ggplot(aes(date_rated,averaged_rating)) +
    geom_point() +
    geom_smooth(colour="red", size=1)+
        ggtitle ("Average Ratings by Date (Week)" ) +
        labs(x ="Date (time)" ,
            y ="# ratings") +
      theme (panel.border = element_rect(colour="brown", fill=NA) )
```

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




We find that while the timestamp feature has some effect on the ratings, it is relatively less given the flat nature of the curve. In the modeling section below, we will be examining this more closely with linear regression.

**Genres with Most Ratings/Popular genres (Feature: genres)** A movie may fall under multiple genres, a fact evident by the pipe-delimited values contained in the genres column in the data set. Let us now determine how many genres exist in the dataset and which ones are the most popular; i.e., which ones received the most rating.

Because the genres column contains multiple values, we will use the separate_rows function from the tidyverse package to split out the rows such that each row contains a single genre. This will help us in simplifying our analysis by genre.

**Note:** This process of separating the rows can take longer given the sheer volume of data contained in our dataset.

```
# Genres with Most Ratings/Popular genres: Separate rows for individual Genre,
# Group by Genre, Summarize, and sort in DESC order
edx_by_genre <- edx %>% separate_rows(genres,sep="\\|") %>%
                        group_by(genres) %>%
                        summarize(genre_count=n()) %>%
                        arrange(desc(genre_count))
```

### Number of Unique Genres

Let us now review how many unique Genres exist in the data set. We find that there are 20 records in our summarized results. Upon closer inspection by listing all the results, we see the last row contains "(no genres

listed)", indicating that there were 7 ratings provided for one or more movies that did not have any genre associated with them.

In summary, we see 19 unique Generes.

```
# Number of unique Genres. This includes the last row where no Genre was specified
length(edx_by_genre[[1]])
```

```
## [1] 20
```

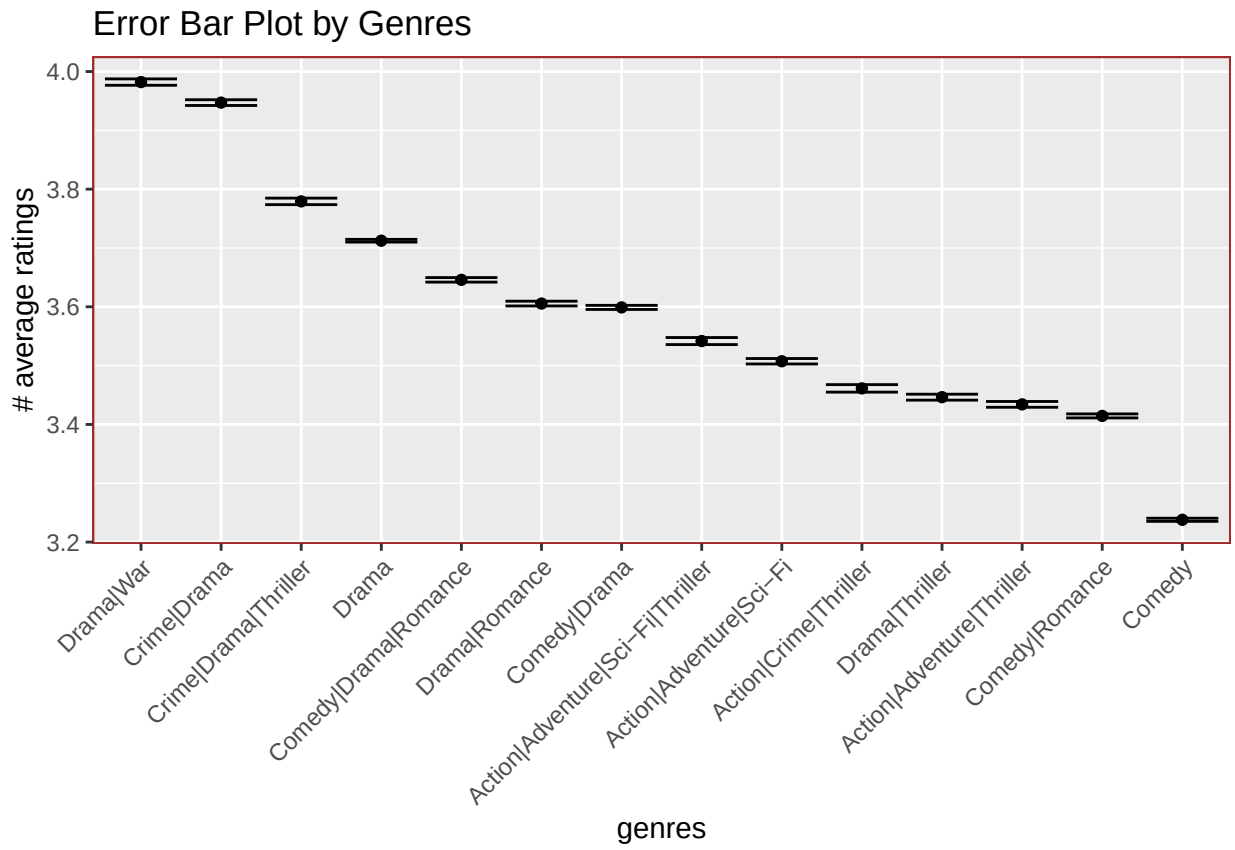
```
edx_by_genre # Display all Genres with their rating count
```

```
## # A tibble: 20 x 2
##    genres             genre_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
```

**Visualize Genre Effects** Given the one-to-many relationship between a movie and genres as stored in the genres column, for a simple visualization of any genre effect on the ratings, we will focus our examination for only those genres that received over 100K ratings. We will not worry about separating genres as individual rows for now.

```
edx %>%
  group_by(genres) %>%
      summarize(n=n(),
                average_rating=mean(rating),
                standard_error= sd(rating)/sqrt(n()),
                .groups = 'drop') %>%
      filter(n>100000) %>%
      mutate(genres = reorder(genres,desc(average_rating))) %>%
      ggplot(aes(x=genres,
                 y = average_rating,
                 ymin=average_rating-2*standard_error,
```

```
            ymax = average_rating+2*standard_error)) +
  geom_point() +
  geom_errorbar()+
      ggtitle ("#Ratings by Genres" ) +
      labs(title="Error Bar Plot by Genres",
          x ="genres" ,
          y ="# average ratings") +
    theme(panel.border = element_rect(colour="brown", fill=NA))+
    theme(axis.text.x = element_text(angle=45,hjust=1))
```




### Popular Genres

In reviewing the summarized Genres above, we see that Drama movies were the most popular followed by those associated with Comedy, Action, and Thriller. Below code shows the top 5 Genres using the head() function.

```
head(edx_by_genre) # Display the top 5 Genres
```

```
## # A tibble: 6 x 2
##   genres    genre_count
##   <chr>           <int>
## 1 Drama         3910127
## 2 Comedy        3540930
## 3 Action        2560545
## 4 Thriller      2325899
## 5 Adventure     1908892
```

```
## 6 Romance       1712100
```

```
#rm(edx_by_genre) # Cleanup the temporary variables
```

**Insight: Popularity, Movie, and Genre Relationship** So far, we identified the most popular movies and genres by summarizing the edx dataset by respective features and sorting them in descending order of their rating count. But is there anything additional we can deduce by putting the 2 results together? Let us find out.

We will now attempt to summarize the dataset by grouping it with both the features - title and genre, followed by a summary of the rating count and sorting the results in descending order of the number of the rating count.

```
edx_by_title_and_genre <- edx %>%
                          group_by(title,genres) %>%
                          summarize(rating_count=n(),.groups = 'drop')%>%
                          top_n(10,rating_count)%>%
                          arrange(desc(rating_count))
```

Let us now compare the results. Reviewing the 3 pieces of information together, we can see that the top rated movies also belong to the top rated genres. For example, Pulp Fiction (1994) has the most rating of 130,302, it is associated with the "Drama" genre, which also happens to possess the most rating of 3,910,127. Similar finding can be made for the remaining top 4 movies displayed in our results.

```
head(edx_by_title_and_genre)
```

```
## # A tibble: 6 x 3
##   title                              genres                          rating_count
##   <chr>                              <chr>                                  <int>
## 1 Pulp Fiction (1994)                Comedy|Crime|Drama                     31362
## 2 Forrest Gump (1994)                Comedy|Drama|Romance|War               31079
## 3 Silence of the Lambs, The (1991)   Crime|Horror|Thriller                  30382
## 4 Jurassic Park (1993)               Action|Adventure|Sci-Fi|Thriller       29360
## 5 Shawshank Redemption, The (1994)   Drama                                  28015
## 6 Braveheart (1995)                  Action|Drama|War                       26212
```

```
head(edx_by_genre)
```

```
## # A tibble: 6 x 2
##   genres    genre_count
##   <chr>           <int>
## 1 Drama         3910127
## 2 Comedy        3540930
## 3 Action        2560545
## 4 Thriller      2325899
## 5 Adventure     1908892
## 6 Romance       1712100
```

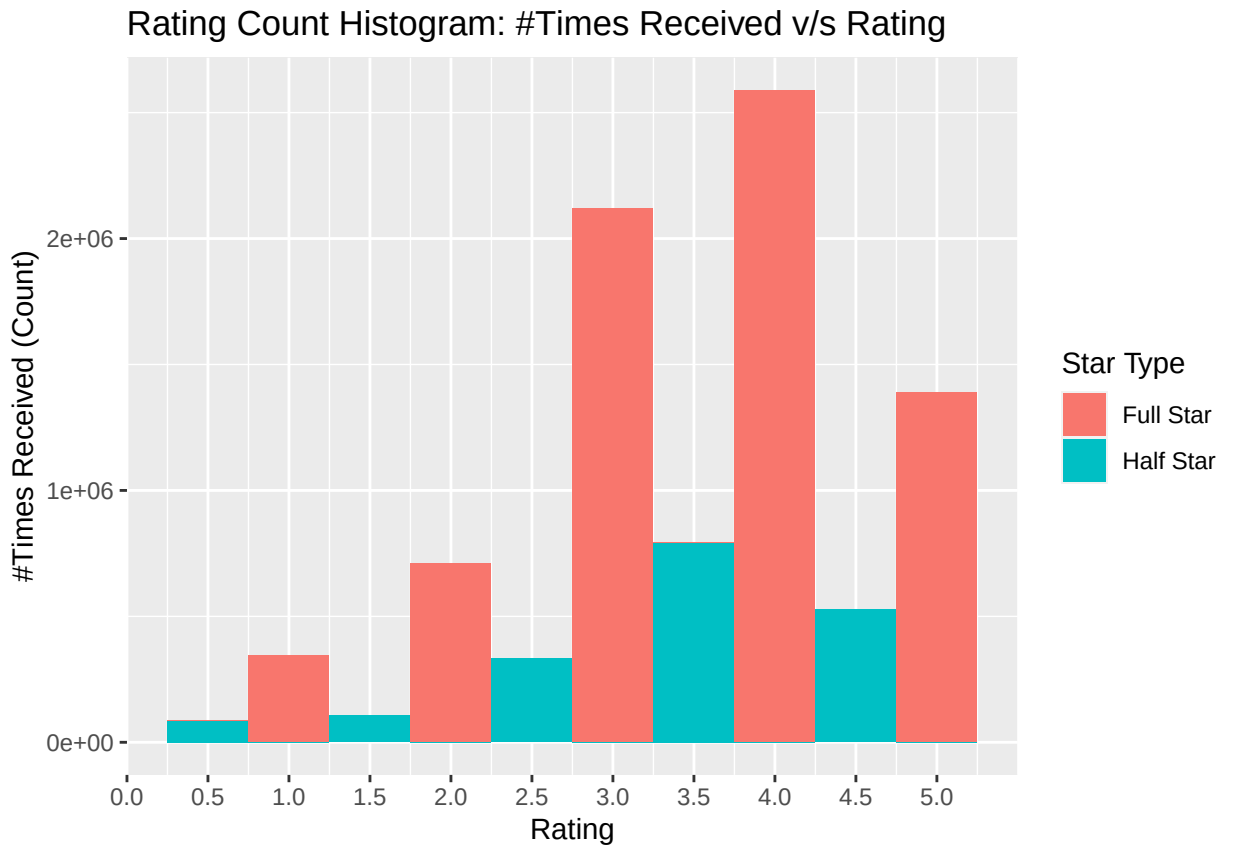
```
head(edx_by_title)
```

```
## # A tibble: 6 x 2
##   title                                                     total_rating
##   <chr>                                                            <dbl>
## 1 Pulp Fiction (1994)                                            130302.
## 2 Silence of the Lambs, The (1991)                               127729
## 3 Shawshank Redemption, The (1994)                               124810.
## 4 Forrest Gump (1994)                                            124714.
## 5 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)   108370.
## 6 Jurassic Park (1993)                                           107561
```

**Visualize Rating Count Histogram (Outcome: rating)** We previously confirmed that no movie has received a rating of 0. A visual on how many times a certain rating was received helps us quickly determine if there were any rating values that were never provided by a user.

There are different values for ratings ranging from 0 through 5 stars with the possibility of half star ratings too. We will now build a Histogram that shows how often a given rating was received.

```
# Add a new column that represents a half star or full star rating.
# We will use this information to visualize how often each type of rating was received
edx_01 <- edx %>% mutate(Star_Type= ifelse((rating==1|
                                            rating==2|
                                            rating==3|
                                            rating==4|
                                            rating==5),"Full Star", "Half Star"))

df_ratings<- data.frame(edx_01$rating,edx_01$Star_Type)

# Plot the ratings using the ggplot() function.
# Provide the appropriate labels for the axes and Chart Title
ggplot(df_ratings,aes(x=edx_01$rating, fill=edx_01$Star_Type))+
  geom_histogram(binwidth = 0.5)+
  scale_x_continuous(breaks=seq(0, 5, by= 0.5)) +
  xlab("Rating")+
  ylab("#Times Received (Count)")+
  ggtitle("Rating Count Histogram: #Times Received v/s Rating")+
  scale_fill_discrete(name="Star Type")
```




We can also get a comprehensive count of how often each rating was received by executing the following code that groups the dataset by rating and sorts it in a descending order.

```
# Count for each rating
edx_01 <- edx_01 %>% group_by(rating) %>% tally() # Group by rating to get count per rating
edx_01 <- edx_01[order(-edx_01$n), ] # Sort the data in descending order of rating count

edx_01 # Display the results to help visualize how often each rating was received
```

```
## # A tibble: 10 x 2
##    rating       n
##     <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
```

```
rm(edx_01,df_ratings) # Clean up the temporary variables
```

**Insight: Ratings**

- The Rating Count Histogram shows that users gave more full star ratings than half star ratings.
- No movie received a rating of 0. This reinforces our previous finding about no missing data for ratings. While we do not know for sure why this is the case, it is possible that the survey system that requested user ratings must be enforcing a business constraint that every movie should receive a non-zero rating to collect meaningful data for analysis.
- The tibble produced by the last snippet of code shows that ratings of 4, 3, 3.5 and 5 were the most provided by users as evident by the corresponding rating count for each of the ratings that are displayed in a descending order of their count.

**Modeling Approach**

For each of the models we build below, we will be measuring their performance using a metric, Root Mean Squared Error (RMSE). RMSE is square root of the average of the residuals squared:

We write a function called RMSE() that takes two numeric vectors as input:

1. True movie ratings
2. Predicted movie ratings as input

The RMSE function returns the RMSE value for the above inputs.

```
# Function to compute RMSE
# This will be used by all models to determine RMSE for comparing each model
# against our goal of desired RMSE < 0.86490
RMSE <- function(actual_ratings, predicted_ratings){
  sqrt(mean((actual_ratings - predicted_ratings)^2))
}
```

**Linear Regression-based Models Feature Scaling**: Feature scaling is a method used to normalize the range of independent variables or features of data. In our current analysis, we are going to use the fit() function which takes care of the needs of any Feature scaling.

**Fitting Multiple Linear Regression to the Training set**

We will use the lm() function to build our model on the training_set. **rating** is going to be a linear combination of multiple independent variables so our formula would include rating ~ column 1 + column 2 + etc. The model will be built on the training_set.

```
regressor_1 = lm(formula = rating ~ userId+movieId+timestamp, data = training_set)

# Let us review the contents of regressor using summary()
summary(regressor_1)
```

```
##
## Call:
## lm(formula = rating ~ userId + movieId + timestamp, data = training_set)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0158 -0.5432  0.4184  0.5286  1.5636
```

```
##
## Coefficients:
##               Estimate Std. Error  t value Pr(>|t|)
## (Intercept)  3.857e+00  3.797e-03 1015.873  < 2e-16 ***
## userId       1.410e-07  1.919e-08    7.346 2.04e-13 ***
## movieId      8.814e-07  4.761e-08   18.513  < 2e-16 ***
## timestamp   -3.420e-10  3.672e-12  -93.146  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.06 on 7200040 degrees of freedom
## Multiple R-squared:  0.001251,   Adjusted R-squared:  0.001251
## F-statistic:  3007 on 3 and 7200040 DF,  p-value: < 2.2e-16
```

We find that all the 3 features, **userId, movieId, and timestamp** have 3 asterisks, indicating a strong statistical significance on our outcome, rating.

Let us make our initial prediction against the test_set.

```
y_pred1 = predict(regressor_1, newdata = test_set)
```

Compute the RMSE on this initial model to get a sense on our model's ability to make good predictions.

```
rmse_model1<- RMSE(test_set$rating,y_pred1)
rmse_model1 # Print the RMSE value: It is 1.05963
```

```
## [1] 1.059632
```

At this point, we have not accounted for the biases resulting from several of the following scenarios:

1. Not all users might have rated all movies.
2. Some users are more active than others
3. The number of movies rated are far less than the number of users rating them
4. Some movies get rated more than others

Above biases are explained in the accompanying report under Data Exploration and Visualization. Since we have already determined from our initial model above that user, movie and time are statistically significant in predicting our outcome, ratings, we will now proceed with the next part of our analysis.

Let us proceed with introducing the biases associated with each of the three effects below:

1. user effects
2. movie effects
3. time effects

Given that using the *lm()* function will result in large computations for this high volume data set, we will handle the regression problem differently using a combination of group_by, summarize, and joins.

### Model #A: Using SIMPLE MEAN

```
#' Step 1: Build the model with simple average/mean of our training_set
mu_hat <- mean(training_set$rating)
mu_hat #  We have a mean of 3.512465
```

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

```
#' Step 2: Make predictions using mu_hat
y_pred_simple_mean<- mu_hat # Since we are simply using the mean at this point

#' Step 3: Evaluate RMSE
rmse_model_a<- RMSE(test_set$rating,y_pred_simple_mean)
rmse_model_a # Print the results
```

```
## [1] 1.060327
```

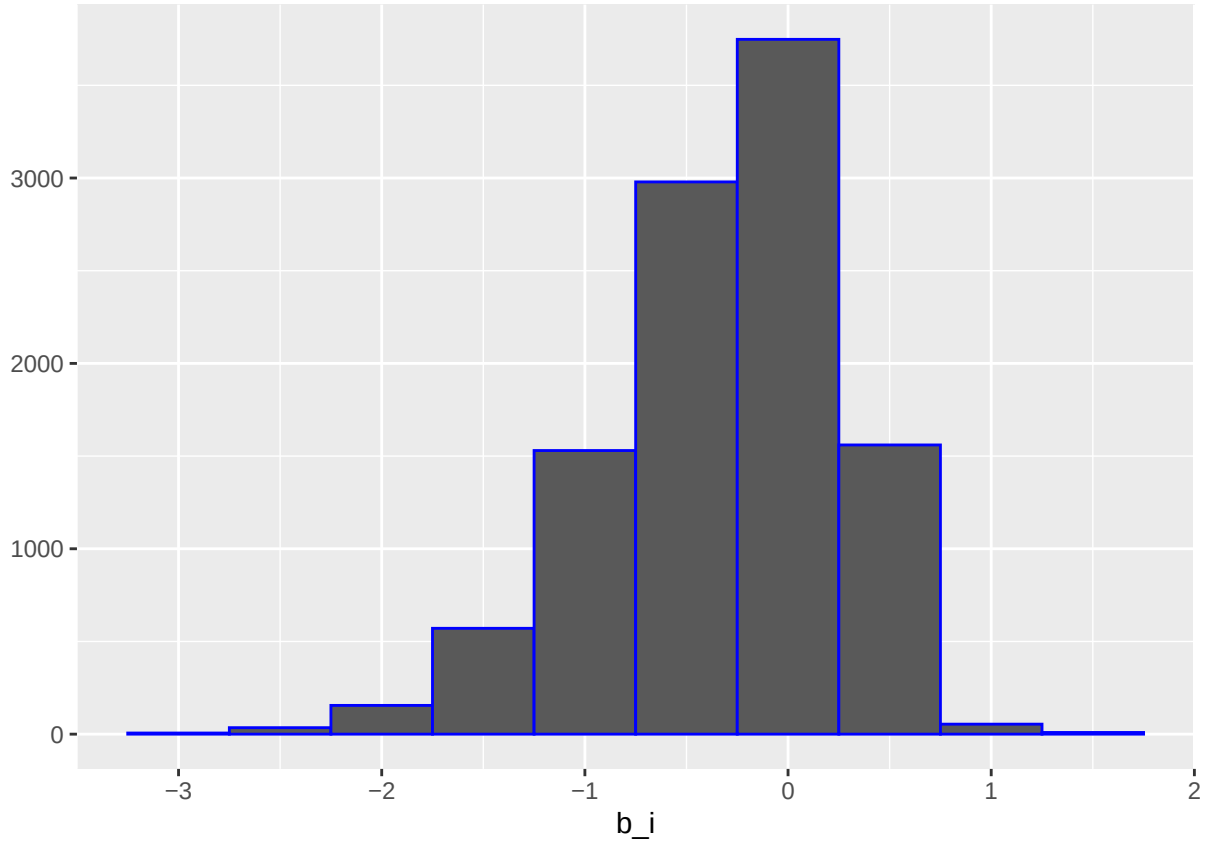
```
#' Note: Even though we know that all the 3 features, movie, user, and timestamp
#' are statistically significant, we will systematically build our model by
#' evaluating each of them gradually to see how they influence our predictions
#' and build a better model
```

### Model #B: Consider Movie Effects

```
# Repeat Steps 1, 2, and 3 for Movie Effects now

#' Step 1: Build the model against the training_set considering movie effects

#' We already know the mean of our ratings in the training_set above but
#' for the sake of comprehensiveness, we will recompute and call it "mu"
#' for our current analysis involving movie effects

mu <- mean(training_set$rating)

# Now plug this mu in the computation of bias resulting from movie effects below

# Movie Averages
movie_avgs <- training_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)) # where # b_i = Bias = Average Ranking for movie i
```

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

```
library(ggplot2) # For data visualization
#' Since mu = 3.5, a value for b_i=1.5 indicates rating = 5 (5-3.5=1.5),
#' i.e., 5-star rating for the movie
movie_avgs %>% qplot(b_i, geom ="histogram",
                         bins = 10,
                         data = ., color = I("blue"))
```




```
#' Step 2: Make predictions using mu + b_i against our test_set

y_pred_movie_effects <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

#' Step 3: Evaluate RMSE
rmse_model_b<- RMSE(test_set$rating,y_pred_movie_effects)
rmse_model_b # Print the results
```

```
## [1] 0.9434394
```

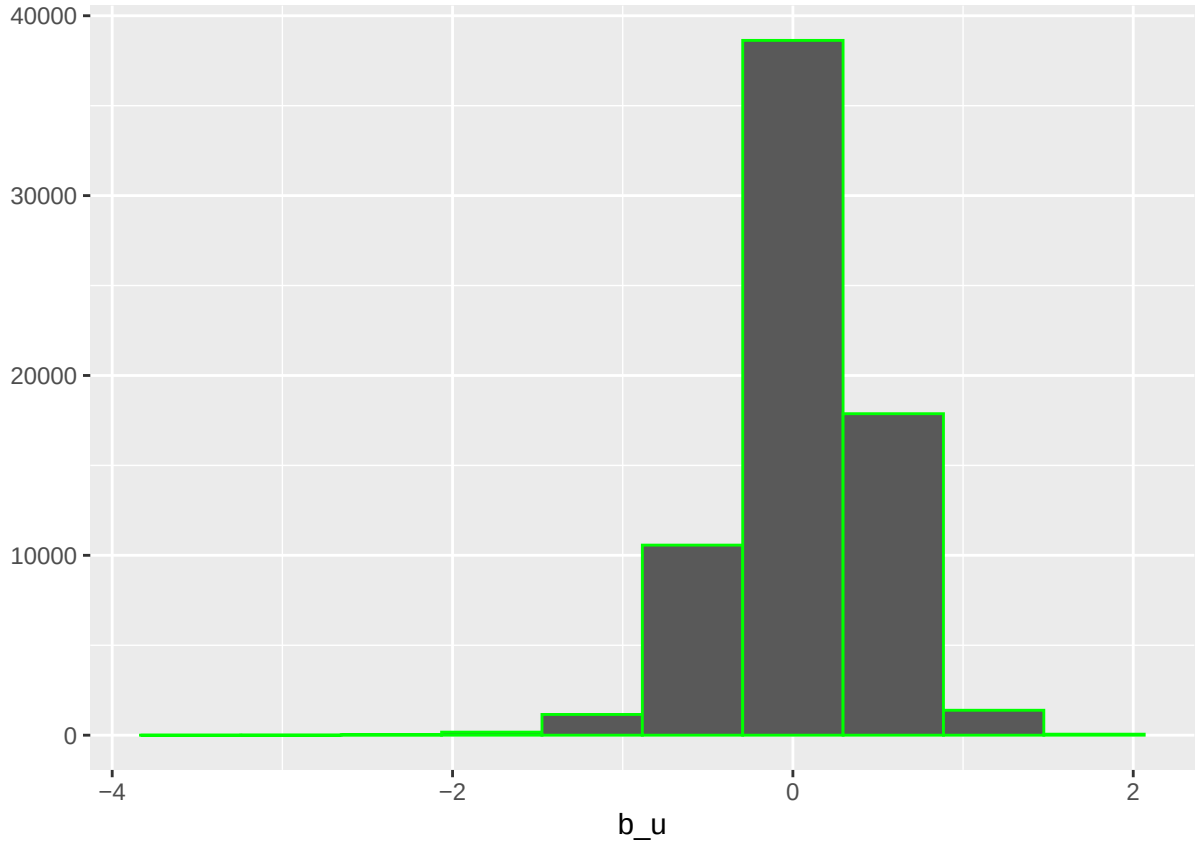
We note that our RMSE improved from 1.06 to 0.9434. This is a step in the right direction.

### Model #C: INCREMENTALLY consider User Effects

```
# Repeat Steps 1, 2, and 3 for User + Movie Effects now

#' Step 1: Build the model against the training_set considering user+movie effects

# User Averages
user_avgs <- training_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i)) # where b_u = bias resulting from user effects
```

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

```
user_avgs %>% qplot(b_u, geom ="histogram",
                    bins = 10,
                    data = ., color = I("green"))
```




```
#' Step 2: Make predictions using mu + b_i + b_u against our test_set

y_pred_movie_and_user_effects <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

```r
#' Step 3: Evaluate RMSE
rmse_model_c<- RMSE(test_set$rating,y_pred_movie_and_user_effects)
rmse_model_c # Print the results
```

```
## [1] 0.866162
```

The RMSE improved further to 0.86616.

### Model #D: INCREMENTALLY consider Time Effects

Repeat Steps 1, 2, and 3 for User + Movie + timestamp Effects now.

*Pre-processing for considering timestamp*

**Note:** Because timestamp is not in a form we can consume directly for our analysis, we will employ the lubridate() function to convert the time in to a proper Date value and round it to let's say a "week" so that it is easy to see how many movies got rated in a "week's" time. We can also choose a month but we will stick with "week" for now.

```
# Step 0: Pre-processing for extracting DATE value from timestamp field
#' We will need to add an extra column to both our training and test sets
#' for this newly extracted DATE value.
#' Because we will be using our training and test set for other models as well,
#' we will first make a copy of the two and only use those 2 copies for the
#' current analysis involving timestamp + user + movie effects

library(lubridate) # For datetime conversions

l_training_set<- training_set # Make a local copy of the training set
l_training_set<- l_training_set %>%
                   mutate(date_rated=round_date(as_datetime(timestamp),unit="week"))

head(l_training_set) # Let us review that the newly extracted date shows correctly
```

```
##    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     355      5 838984474 Flintstones, The (1994)
## 5:      1     356      5 838983653     Forrest Gump (1994)
## 6:      1     364      5 838983707   Lion King, The (1994)
##                                       genres date_rated
## 1:                           Comedy|Romance 1996-08-04
## 2:                     Action|Crime|Thriller 1996-08-04
## 3:               Action|Drama|Sci-Fi|Thriller 1996-08-04
## 4:                    Children|Comedy|Fantasy 1996-08-04
## 5:                   Comedy|Drama|Romance|War 1996-08-04
## 6: Adventure|Animation|Children|Drama|Musical 1996-08-04
```

```
# Repeat the same extraction for a copy of the test_set
l_test_set <- test_set # Make a local copy of the test set
l_test_set<- l_test_set %>%
  mutate(date_rated=round_date(as_datetime(timestamp),unit="week"))

head(l_test_set) # Let us review that the newly extracted date shows correctly
```

```
##    userId movieId rating timestamp                        title
## 1:      1     316      5 838983392               Stargate (1994)
## 2:      1     329      5 838983392 Star Trek: Generations (1994)
## 3:      1     362      5 838984885       Jungle Book, The (1994)
## 4:      1     377      5 838983834                   Speed (1994)
## 5:      1     588      5 838983339                Aladdin (1992)
```

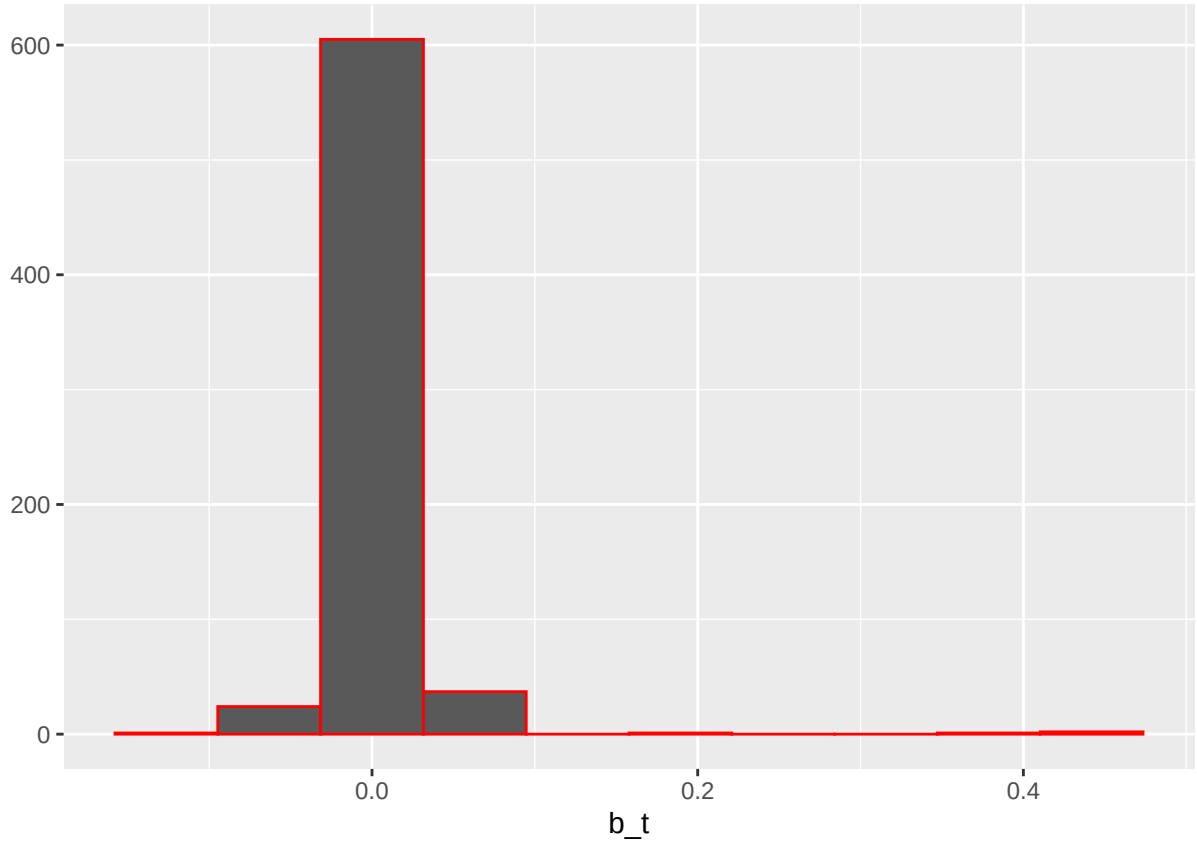
```
## 6:      2     110      5 868245777             Braveheart (1995)
##                                        genres date_rated
## 1:                     Action|Adventure|Sci-Fi 1996-08-04
## 2:                Action|Adventure|Drama|Sci-Fi 1996-08-04
## 3:                   Adventure|Children|Romance 1996-08-04
## 4:                      Action|Romance|Thriller 1996-08-04
## 5: Adventure|Animation|Children|Comedy|Musical 1996-08-04
## 6:                              Action|Drama|War 1997-07-06
```

We are now ready to proceed with our analysis involving timestamp. We will undertake this analysis against the l_training_set and l_test_set both of which contain a new column, date_rated, which is a date representation of the timestamp field present in the original data set.

```
#' Step 1: Build the model against the training_set considering user+movie effects

time_avgs <- l_training_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(date_rated) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u)) # where b_t = bias resulting from timestamp effects
```

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

```
time_avgs %>% qplot(b_t, geom ="histogram",
                    bins = 10,
                    data = ., color = I("red"))
```



```
#' Step 2: Make predictions using mu + b_i + b_u +b_t against our l_test_set

y_pred_movie_user_time_effects <- l_test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(time_avgs, by="date_rated") %>%
  mutate(pred = mu + b_i + b_u+b_t) %>%
  .$pred

#' Step 3: Evaluate RMSE
rmse_model_d<- RMSE(test_set$rating,y_pred_movie_user_time_effects)
rmse_model_d # Print the results
```

```
## [1] 0.8660979
```

```
#' The RMSE decreased marginally to 0.86609

#' Let us get a quick summary on our RMSEs gathered thus far
#' into rmse_results
temp1<- c("Without accounting for any biases (test set)", "Simple Mean","Movie Effects (test set)", "Mo
         "Movie+User+Time Effects (test set)")
temp2<- c(rmse_model1,rmse_model_a,rmse_model_b,rmse_model_c,rmse_model_d)
rmse_results <- data.frame(Model_Method=temp1,RMSE_Values=temp2)
rmse_results %>% knitr::kable()
```

| Model_Method | RMSE_Values |
|---|---|
| Without accounting for any biases (test set) | 1.0596317 |
| Simple Mean | 1.0603269 |
| Movie Effects (test set) | 0.9434394 |
| Movie+User Effects (test set) | 0.8661620 |
| Movie+User+Time Effects (test set) | 0.8660979 |

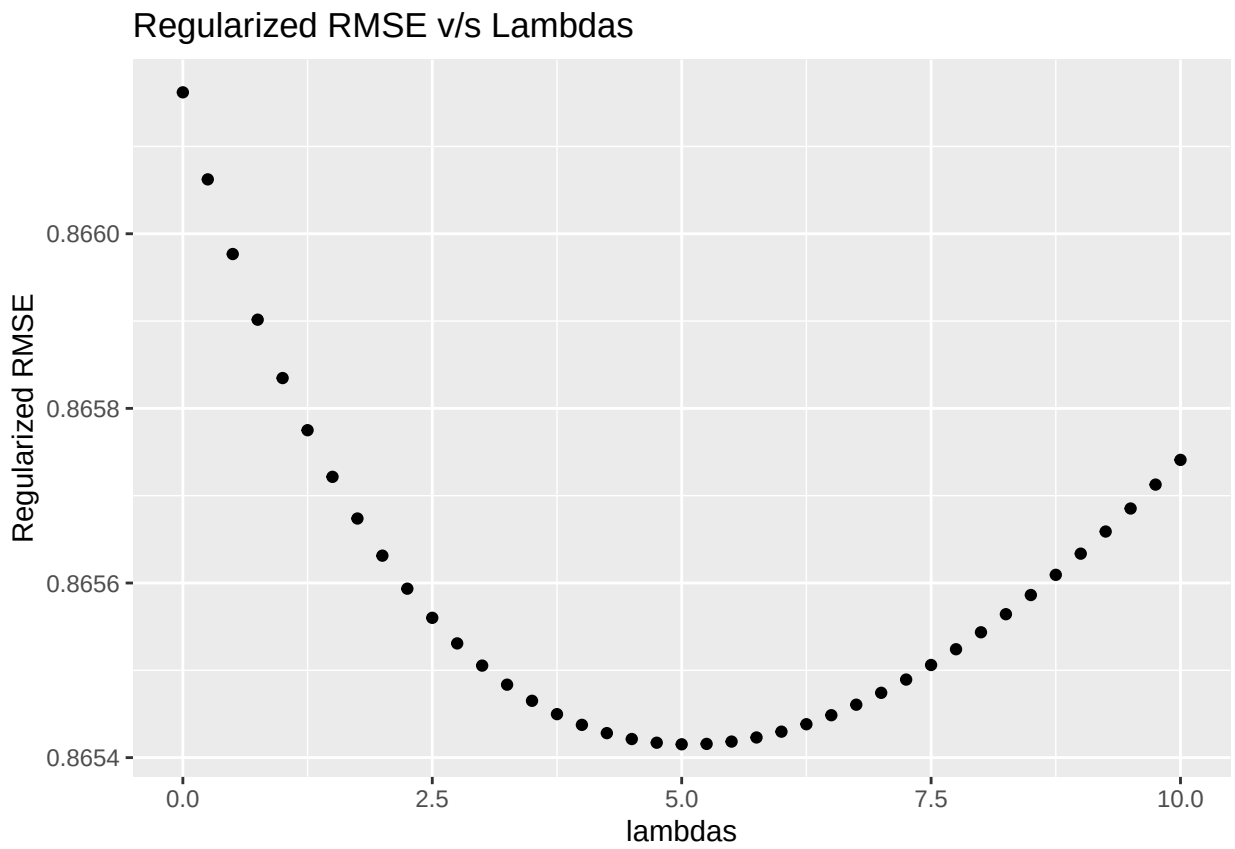
**Insights Gained so Far** We make the following findings:

1. Without accounting for any biases, we at least found that user, movie, and timestamp were statistically significant in predicting our outcome "rating"
2. The simple mean based model did not yield a good RMSE value because it does not account for the biases present in the data set
3. Introduction of the first bias for "Movie effects" showed a noticeable improvement in RMSE because we are sharpening our model with additional information about the influence of Movie related biases.
4. An incremental addition of User effects to the movie effects-based model led to a further improvement of our RMSE from 0.9434 to 0.86616
5. A further incremental addition of Time effects had a very marginal impact on our RMSE value.

```
rm(l_training_set,l_test_set,regressor_1,temp1,temp2,
   y_pred1,y_pred_movie_effects,
   y_pred_movie_and_user_effects,
   y_pred_movie_user_time_effects) # Clean up temporary variables
```

**Model: Regularize Movie + User Effects** Given that we have already exhausted our options so far and the fact that adding timestamp had very little impact on our RMSE, we will now focus on **Regularizing our data** and consider only the other 2 statistically significant features - user and movie for our further analysis

```
# Perform cross-validation to choose lambda, our tuning parameter.

lambdas <- seq(0,10,0.25)

reg_rmse<- sapply(lambdas,function(m){
  reg_mu<- mean(training_set$rating)

  reg_b_i <- training_set %>%
    group_by(movieId) %>%
    summarize(reg_b_i= sum(rating-reg_mu)/(n()+m),.groups = 'drop')

  reg_b_u <-  training_set %>%
    left_join(reg_b_i, by="movieId") %>%
      group_by(userId) %>%
    summarize(reg_b_u= sum(rating - reg_b_i - reg_mu)/(n()+m),.groups = 'drop')

  reg_predictions <- test_set %>%
    left_join(reg_b_i, by="movieId") %>%
    left_join(reg_b_u, by="userId") %>%
    mutate(pred = reg_mu + reg_b_i + reg_b_u)%>%
    .$pred

  return(RMSE(test_set$rating,reg_predictions))
})

qplot(lambdas,reg_rmse,
      main="Regularized RMSE v/s Lambdas",
      xlab="lambdas",
      ylab="Regularized RMSE")
```




Looking at the plot above, we see that the optimum lambda is around 5.0.

Let us obtain that optimum value for lambda which yields the best (lowest) RMSE.

```
optimum_lambda <- lambdas[which.min(reg_rmse)]
optimum_lambda # Print the optimum lambda
```

```
## [1] 5
```

```
# The lowest RMSE we can get can be computed below
rmse_model_reg<- min(reg_rmse)
rmse_model_reg # 0.8654153
```

```
## [1] 0.8654153
```

```
rmse_results <- bind_rows(rmse_results,
                          data.frame(Model_Method="Regularized Movie+User Effects (test set)",
                                     RMSE_Values = rmse_model_reg ))
rmse_results %>% knitr::kable()
```

| Model_Method | RMSE_Values |
|---|---|
| Without accounting for any biases (test set) | 1.0596317 |
| Simple Mean | 1.0603269 |
| Movie Effects (test set) | 0.9434394 |

| Model_Method | RMSE_Values |
|---|---|
| Movie+User Effects (test set) | 0.8661620 |
| Movie+User+Time Effects (test set) | 0.8660979 |
| Regularized Movie+User Effects (test set) | 0.8654153 |

The regularized model has yielded the lowest RMSE thus far. However, this model is only accounting for the following key things:

1. Variations from movie to movie ratings (the movie related biases)
2. Variations from user to user ratings (user related biases)

We are now going to account for another key point; groups of movies have similar rating patterns and groups of users may have similar rating patterns.

**Matrix Factorization using Recommender System** As demonstrated in the data exploration earlier in this report, we can think of this data as a very large matrix, with users on the rows and movies on the columns, with many empty cells. The amount of computation required to build the entire matrix of user by movie is very high.

To deal with this challenge, we will employ a recommendation system using the **recosystem** library. The **recosystem** is typically used to approximate an incomplete matrix using the product of two matrices in a latent space.

```
#' Step 1: Build the model

# Movie effects on training set
movie_avgs_reco <- training_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu),.groups = 'drop')

# User effect on training_set
user_avgs_reco <- training_set %>%
  left_join(movie_avgs_reco, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - b_i - mu),.groups = 'drop')

# Compute residuals on training_set
training_set <- training_set %>%
  left_join(movie_avgs_reco, by = "movieId") %>%
  left_join(user_avgs_reco, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)

# Compute residuals on test set
test_set <- test_set %>%
  left_join(movie_avgs_reco, by = "movieId") %>%
  left_join(user_avgs_reco, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)

# Load data from memory focusing on only 3 columns for userId, movieId, and rating
#' We use the data_memory() function to pull data from memory
#' as R objects from our training_set and test_set
train_data <- data_memory(user_index = training_set$userId,
                          item_index = training_set$movieId,
                          rating = training_set$res, index1 = T)

test_data <- data_memory(user_index = test_set$userId,
                         item_index = test_set$movieId,
                         index1 = T)

# create a model object using the Reco() function
recommender <- Reco()

# This is a randomized algorithm
set.seed(1)

# NOTE: Resource intensive: Takes 25 minutes to run against our training set.
# call the '$tune()' method to select best tuning parameters
res = recommender$tune(
  train_data,
```

```
  opts = list(dim = c(10, 20, 30),
              costp_l1 = 0, costq_l1 = 0,
              lrate = c(0.05, 0.1, 0.2), nthread = 2)
)

# Print the tuning parameters that are optimum for our analysis
print(res$min)
```

```
## $dim
## [1] 30
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.01
##
## $costq_l1
## [1] 0
##
## $costq_l2
## [1] 0.1
##
## $lrate
## [1] 0.05
##
## $loss_fun
## [1] 0.8022528
```

```
# Train the model by calling the `$train()` method
# some parameters coming from the result of `$tune()`
# This is a randomized algorithm
set.seed(1)
suppressWarnings(recommender$train(train_data,
                                   opts = c(dim = 30,
                                            costp_l1 = 0,
                                            costp_l2 = 0.01,
                                            costq_l1 = 0,
                                            costq_l2 = 0.1,
                                            lrate = 0.05,
                                            verbose = FALSE)))

# use the `$predict()` method to compute predicted values
# return predicted values in memory
y_pred_matrix_fact_reco <- recommender$predict(test_data, out_memory()) +
                                  mu + test_set$b_i + test_set$b_u

# ceiling rating at 5
ind <- which(y_pred_matrix_fact_reco > 5)
y_pred_matrix_fact_reco[ind] <- 5

# floor rating at 0.50
ind <- which(y_pred_matrix_fact_reco < 0.5)
```

```
#' Step 2: Make predictions
y_pred_matrix_fact_reco[ind] <- 0.5

#' Step 3: Evaluate RMSE
rmse_model_matrix_fact <- RMSE(test_set$rating, y_pred_matrix_fact_reco)

# Add results to our RMSE Results table for comparison with prior models
rmse_results <- bind_rows(rmse_results,
                          tibble(Model_Method="Movie and User effects and Matrix Fact. (Test Set)",
                                 RMSE_Values = rmse_model_matrix_fact))
rmse_results %>% knitr::kable()
```

| Model_Method | RMSE_Values |
|---|---|
| Without accounting for any biases (test set) | 1.0596317 |
| Simple Mean | 1.0603269 |
| Movie Effects (test set) | 0.9434394 |
| Movie+User Effects (test set) | 0.8661620 |
| Movie+User+Time Effects (test set) | 0.8660979 |
| Regularized Movie+User Effects (test set) | 0.8654153 |
| Movie and User effects and Matrix Fact. (Test Set) | 0.8003622 |

The RMSE has improved further to **0.8003622**. This is the **lowest** of all the prior models we built above. Consequently, we will now subject this model to the final hold-out validation set below to see the results.

# Results

## RMSE Returned by Algorithm on Validation Set

```
#' ##############################################################################
#' FINAL TEST against ORIGINAL edx train set and hold-out validation set
#' ##############################################################################
#' As a final test before choosing this model,
#' we will run this model against our original edx train set and
#' our final holdout validation set to obtain the FINAL RMSE

# Movie effects on ORIGINAL training set - edx
movie_avgs_reco <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu),.groups = 'drop')

# User effects on ORIGINAL training set - edx
user_avgs_reco <- edx %>%
  left_join(movie_avgs_reco, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - b_i - mu),.groups = 'drop')

# Compute residuals on ORIGINAL training set - edx
edx <- edx %>%
  left_join(movie_avgs_reco, by = "movieId") %>%
  left_join(user_avgs_reco, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)

# Compute residuals on our final hold-out validation set
validation <- validation %>%
  left_join(movie_avgs_reco, by = "movieId") %>%
  left_join(user_avgs_reco, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)

# Using recosystem
# create data saved on disk in 3 columns with no headers
edx_data <- data_memory(user_index = edx$userId,
                        item_index = edx$movieId,
                        rating = edx$res, index1 = T)

validation_data <- data_memory(user_index = validation$userId,
                               item_index = validation$movieId,
                               index1 = T)

# create a model object
recommender <- Reco()

# Train the model by calling the `$train()` method
# some parameters coming from the result of `$tune()`
# This is a randomized algorithm
set.seed(1)

suppressWarnings(recommender$train(edx_data, opts = c(dim = 30, costp_l1 = 0,
```

```
                                                    costp_l2 = 0.01, costq_l1 = 0,
                                                    costq_l2 = 0.1, lrate = 0.05,
                                                    verbose = FALSE)))

# use the `$predict()` method to compute predicted values
# return predicted values in memory

y_pred_matrix_fact_reco_final <- recommender$predict(validation_data, out_memory()) + mu +
                                                    validation$b_i + validation$b_u

# ceiling rating at 5
ind <- which(y_pred_matrix_fact_reco_final > 5)
y_pred_matrix_fact_reco_final[ind] <- 5

# floor rating at 5
ind <- which(y_pred_matrix_fact_reco_final < 0.5)
y_pred_matrix_fact_reco_final[ind] <- 0.5

# create a results table with this and prior approaches
rmse_reg_final <- RMSE(validation$rating, y_pred_matrix_fact_reco_final)

rmse_results <- bind_rows(rmse_results,
                          tibble(Model_Method="Movie and User effects and Matrix Fact. (FINAL validatio
rmse_results %>% knitr::kable()
```

| Model_Method | RMSE_Values |
|---|---|
| Without accounting for any biases (test set) | 1.0596317 |
| Simple Mean | 1.0603269 |
| Movie Effects (test set) | 0.9434394 |
| Movie+User Effects (test set) | 0.8661620 |
| Movie+User+Time Effects (test set) | 0.8660979 |
| Regularized Movie+User Effects (test set) | 0.8654153 |
| Movie and User effects and Matrix Fact. (Test Set) | 0.8003622 |
| Movie and User effects and Matrix Fact. (FINAL validation set) | 0.7939559 |

## Model Performance

The results documented in the above table demonstrate how we improved performance of our models by gradually adding the biases followed by regularization for user and movie effects. The **recommender system employing Matrix Factorization** yielded the best performance as we confirmed it against the final hold-out validation set to get the **lowest RMSE score of 0.7939**.

# Conclusion

We started our analysis by data exploration and visualization, something that should always be undertaken to get a sense of what it is that needs to be analyzed. The findings of this exploration helped us evaluate if the data required some pre-processing or data wrangling work, as we call it.

After building a decent understanding of our data set, we started our modeling with simple linear regression to determine the statistically significant factors. We gradually added these factors one by one to make predictions and generate the RMSE score, the metric used for our decision-making purposes. As we progressed, we found that of the various features, userId and movieId had the most impact on our RMSE. We enhanced our model by undertaking Regularization to achieve an RMSE value that was very close to our desired RMSE of $< 0.86490$. However, that model missed out on the challenge with sparsity of the matrix; something that could be addressed using a recommender system.

The recommender system using the recosystem library helped achieve a better performance as measured using the RMSE score of 0.7939 (much lower than our desired RMSE of 0.86490) against our final hold-out validation set.

**Chosen Model**: We choose the recommender system using Matrix Factorization for this project as it yielded the lowest RMSE score of 0.7939 (much $< 0.86490$).

## Limitations

While we noticed from the Error bar plot for Generes effects, we didn't really focus much on that feature for our current analysis. One of the challenges with doing the Genres related analysis was the fact that the column contained pipe-delimited multiple values for Genres. Using the resources just on the local machine it would be very difficult to undertake comprehensive analysis for this feature because it will require even more computation resources.

## Future Work

We can possibly continue with our future analysis focused on Genres effect. One of the first things that would need to be done is further data wrangling just for the Genres column such that we have all genres for a given row arranged in an alphabetically sorted fashion. Doing so would allow us to prepare our data for one-hot encoding of the Genres column that could then be later included for further impact on our dependent variable, ratings. We can also consider running this in a cloud environment; albeit, I would admit that I am still getting to know data science. I know Python has some really good libraries that are much more optimized than R so that is another option to consider - build the model in Python and run it in Google Cloud lab to get more computation power.