Movie Recommendation System

S. Roshan Pranao

07-12-2024

Introduction

Leveraging the power of recommendation systems has become essential in enhancing user experiences across e-commerce and online streaming platforms, such as Netflix, YouTube, and Amazon. These systems not only increase user satisfaction but also drive sales and profit growth by providing personalized suggestions that align with individual preferences. As companies compete for customer loyalty, they invest heavily in technologies that analyze user behavior to offer tailored recommendations, significantly impacting their bottom line. For instance, Amazon's success as the largest online retailer is partly attributed to its sophisticated recommendation algorithms, which suggest products based on user interactions.

The significance of recommendation systems was notably highlighted during the 2009 Netflix Prize competition, which offered a million-dollar incentive for anyone who could improve its recommendation algorithm by at least 10%. This challenge underscored the potential of data-driven approaches to enhance user engagement and retention. Typically, these systems utilize a rating scale from 1 to 5, where users can express their satisfaction with movies or products. Additional data points—such as user comments, viewing history, and interaction metrics—serve as valuable predictors for generating accurate recommendations.

In this project, we aim to develop a movie recommendation system utilizing the MovieLens dataset, applying insights gained from the HarvardX Data Science Professional Certificate program. Our primary objective is to achieve an RMSE score below 0.86490 by employing advanced modeling techniques, including Regularized Linear Models and Matrix Factorization methods. This report is structured to provide a comprehensive overview of our methodology: from data ingestion and exploratory analysis to modeling results and performance evaluation.

This Project is a part of HarvardX PH125.9x Data Science: Capstone.

Dataset Summary

The MovieLens dataset is a widely used resource for developing and evaluating recommendation systems, containing user ratings for movies alongside detailed metadata. Various versions are available, with the largest being the 25 million ratings dataset, which includes ratings from over 162,000 users on more than 62,000 movies. For this project, we utilize a specific subset of the dataset to focus our analysis and modeling efforts, ensuring a manageable yet informative foundation for understanding user preferences and enhancing predictive algorithms.

List of Libraries Utilized

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

```
## Loading required package: tidyverse

## Warning: package 'tidyverse' was built under R version 4.4.3

## Warning: package 'ggplot2' was built under R version 4.4.3

## Warning: package 'purrr' was built under R version 4.4.3

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

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

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

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

```
## Loading required package: recosystem

## Warning: package 'recosystem' was built under R version 4.4.2
```

```
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")

# Loading the necessary libraries
library(tidyverse)
library(caret)
library(ggplot2)
library(lubridate)
library(recosystem)
```

1. tidyverse:

A collection of R packages designed for data science that share an underlying design philosophy, grammar, and data structures, making data manipulation and visualization more intuitive.

2. caret:

The "Classification And Regression Training" package in R provides a unified interface for creating predictive models, facilitating tasks such as data splitting, pre-processing, feature selection, and model tuning.

3. ggplot2:

A powerful visualization package in R that implements the grammar of graphics, allowing users to create complex and customizable plots by layering components based on data aesthetics.

4. lubridate:

A package designed to make working with dates and times easier in R by providing functions for parsing, manipulating, and formatting date-time objects.

5. recosystem:

A package specifically designed for building recommendation systems in R, offering tools for matrix factorization and collaborative filtering to predict user preferences.

Steps Performed

Initially, the dataset was downloaded from the specified webpage, followed by scraping and cleaning processes. The dataset was then divided into an edx set and a final_holdout_test set in a 9:1 ratio. The edx set was further split into training and testing subsets in an 8:2 ratio to prepare them for analysis.

Subsequently, exploratory data analysis (EDA) was performed on the EDX and training partitions, utilizing visualizations and descriptive statistics for thorough examination.

Various models were developed, starting from random predictions to regularization techniques and matrix factorization methods. The model with the lowest RMSE loss value was selected for final evaluation. Once the preferred model was tested, it underwent further validation by applying it to the final_holdout_test dataset to ensure consistency with the edx dataset.

Methods/Analysis

Data Ingestion

The below provided code shows how the data wrangling took place from downloading the data from the required URL to partitioning of data for training and testing of Machine Learning Models.

```
############################################################
# Create edx and final_holdout_test sets
############################################################

# Note: this process could take a couple of minutes

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

```r
options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1)
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

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

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

Here the movielens dataset has been partitioned in 90/10 split where edx act as the train set of about 90% and final_holdout_test comprise of 10%.

Data Cleaning

Data cleaning for the MovieLens dataset involves the process of identifying and rectifying issues such as missing values, duplicates, and inconsistencies within the data to ensure its accuracy and usability for analysis. This includes removing irrelevant or corrupted data, correcting formatting errors, and ensuring that each user has rated a sufficient number of movies to maintain the integrity of the dataset. Effective data cleaning is crucial for producing reliable insights and enhancing the performance of recommendation algorithms.

Starting by inspecting the edx and final_holdout_test data sets.

```
# Analysing the structure of the datasets - edx and final_holdout_test
str(edx)
```

```
## 'data.frame':    9000061 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

```
str(final_holdout_test)
```

```
## 'data.frame':    999993 obs. of  6 variables:
##  $ userId   : int  1 2 2 3 3 3 4 4 4 5 ...
##  $ movieId  : int  588 1210 1544 151 1288 5299 380 435 480 477 ...
##  $ rating   : num  5 4 3 4.5 3 3 3 3 5 3 ...
##  $ timestamp: int  838983339 868245644 868245920 1133571026 1133571035 1164885617 844416656 84441707
##  $ title    : chr  "Aladdin (1992)" "Star Wars: Episode VI - Return of the Jedi (1983)" "Lost World:
##  $ genres   : chr  "Adventure|Animation|Children|Comedy|Musical" "Action|Adventure|Sci-Fi" "Action|A
```

The variable timestamp is in the form of int and not in a proper date format which may introduce inaccuracies when training the Machine Learning Model.

Therefore, Conversion of timestamp to datetime format can be seen in the below code snippet

```
#Converting time stamp to date and time
edx <- edx %>% mutate(date=as_datetime(timestamp))
final_holdout_test <- final_holdout_test %>% mutate(date=as_datetime(timestamp))
str(edx)
```

```
## 'data.frame':    9000061 obs. of  7 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
##  $ date     : POSIXct, format: "1996-08-02 11:24:06" "1996-08-02 10:58:45" ...
```

Check for any NA's in the datasets are vital as that may degrade the performance of Machine Learning Models and their ability to learn from the trends would be affected.

```
#check for any missing data in the edx and final_holdout_test data sets
sum(is.na(edx))
```

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

```
sum(is.na(final_holdout_test))
```

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

Since there are no NAs it is safe to proceed with the dataset for further analysis and Model training and testing.

edx dataset comprises of 90% of movielens dataset as mentioned earlier, it is difficult to train models as it would take significant amount of time and may be expensive from hardware point of view. Hence again partitioning it into trainset and test set can be better to training the model and test within the edx model so that it performs well in the final_holdout_test. The partition can be seen in the below output as the division is again made in 90/10 split.

```
#Partitioning edx into train_data and test_data
set.seed(1, sample.kind="Rounding")
```

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

```
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_data <- edx[-test_index,]
temporary <- edx[test_index,]

# Make sure userId and movieId in test_data are also in train_data
test_data <- temporary %>%
  semi_join(train_data, by = "movieId") %>%
  semi_join(train_data, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temporary, test_data)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres,
## date)`
```

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

rm(test_index, temporary, removed)
```

Required Data Wranglings are performed in the data sets for further analysis and model developing.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) for the MovieLens dataset involves analyzing and visualizing the data to understand its underlying structure, identify key patterns, and uncover relationships between user ratings, movie attributes, and user demographics.

```
#edx part

head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     231      5 838983392          Dumb & Dumber (1994)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                            genres                date
## 1                Comedy|Romance 1996-08-02 11:24:06
## 2         Action|Crime|Thriller 1996-08-02 10:58:45
## 3                        Comedy 1996-08-02 10:56:32
## 4  Action|Drama|Sci-Fi|Thriller 1996-08-02 10:57:01
## 5       Action|Adventure|Sci-Fi 1996-08-02 10:56:32
## 6 Action|Adventure|Drama|Sci-Fi 1996-08-02 10:56:32
```

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres              date
##  Length:9000061     Length:9000061     Min.   :1995-01-09 11:46:49.00
##  Class :character   Class :character   1st Qu.:2000-01-01 21:46:51.00
##  Mode  :character   Mode  :character   Median :2002-10-24 02:15:47.00
##                                        Mean   :2002-09-21 05:38:44.78
##                                        3rd Qu.:2005-09-14 03:11:12.00
##                                        Max.   :2009-01-05 04:55:03.00
```

```
dim(edx)
```

```
## [1] 9000061       7
```

There are 7 variables in the edx dataset with 9000061 observations. From the above output we can infer that the 'rating' column is the desired outcome for our prediction. Apparently, the user information is stored in userId and the movie information is stored in both in movieId and title columns. The genres column mentions the class in which the movie belongs to.

```
unique <- c(n_movies = n_distinct(edx$movieId),
    n_genres = n_distinct(edx$genres),
    n_users = n_distinct(edx$userId))
print(unique)
```

```
## n_movies n_genres  n_users
##    10677      797    69878
```

Above is the concise overview of unique users and unique movies based on genres and movieID.

Knowing about the orientation of final_holdout_test is crucial as it is the final stage where the model has to perform which is a new,unseen data.

```
#final_holdout_test part

head(final_holdout_test)
```

```
##   userId movieId rating  timestamp
## 1      1     588    5.0  838983339
## 2      2    1210    4.0  868245644
## 3      2    1544    3.0  868245920
## 4      3     151    4.5 1133571026
## 5      3    1288    3.0 1133571035
## 6      3    5299    3.0 1164885617
##                                                  title
## 1                                       Aladdin (1992)
## 2       Star Wars: Episode VI - Return of the Jedi (1983)
## 3 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
## 4                                       Rob Roy (1995)
## 5                                This Is Spinal Tap (1984)
## 6                          My Big Fat Greek Wedding (2002)
##                                        genres                date
## 1 Adventure|Animation|Children|Comedy|Musical 1996-08-02 10:55:39
## 2                      Action|Adventure|Sci-Fi 1997-07-07 03:20:44
## 3     Action|Adventure|Horror|Sci-Fi|Thriller 1997-07-07 03:25:20
## 4                     Action|Drama|Romance|War 2005-12-03 00:50:26
## 5                               Comedy|Musical 2005-12-03 00:50:35
## 6                               Comedy|Romance 2006-11-30 11:20:17
```

```
summary(final_holdout_test)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18127   1st Qu.:  653   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35719   Median : 1835   Median :4.000   Median :1.036e+09
##  Mean   :35878   Mean   : 4121   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53649   3rd Qu.: 3633   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres              date
##  Length:999993      Length:999993      Min.   :1995-01-09 11:46:49.0
##  Class :character   Class :character   1st Qu.:2000-01-02 12:57:46.0
##  Mode  :character   Mode  :character   Median :2002-10-28 00:38:30.0
##                                        Mean   :2002-09-23 12:10:26.2
##                                        3rd Qu.:2005-09-16 23:47:40.0
##                                        Max.   :2009-01-05 05:02:16.0
```

Exploring Data based on User ratings

Users have the liberty to provide a rating value from 0.5 to 5.0 based on their satisfaction level. This can lead to total value of 10 . But most of the time users provide a rounded value. Only in rare scenarios the floating points can be encountered.

The rating Distribution can be viewed in the following section:

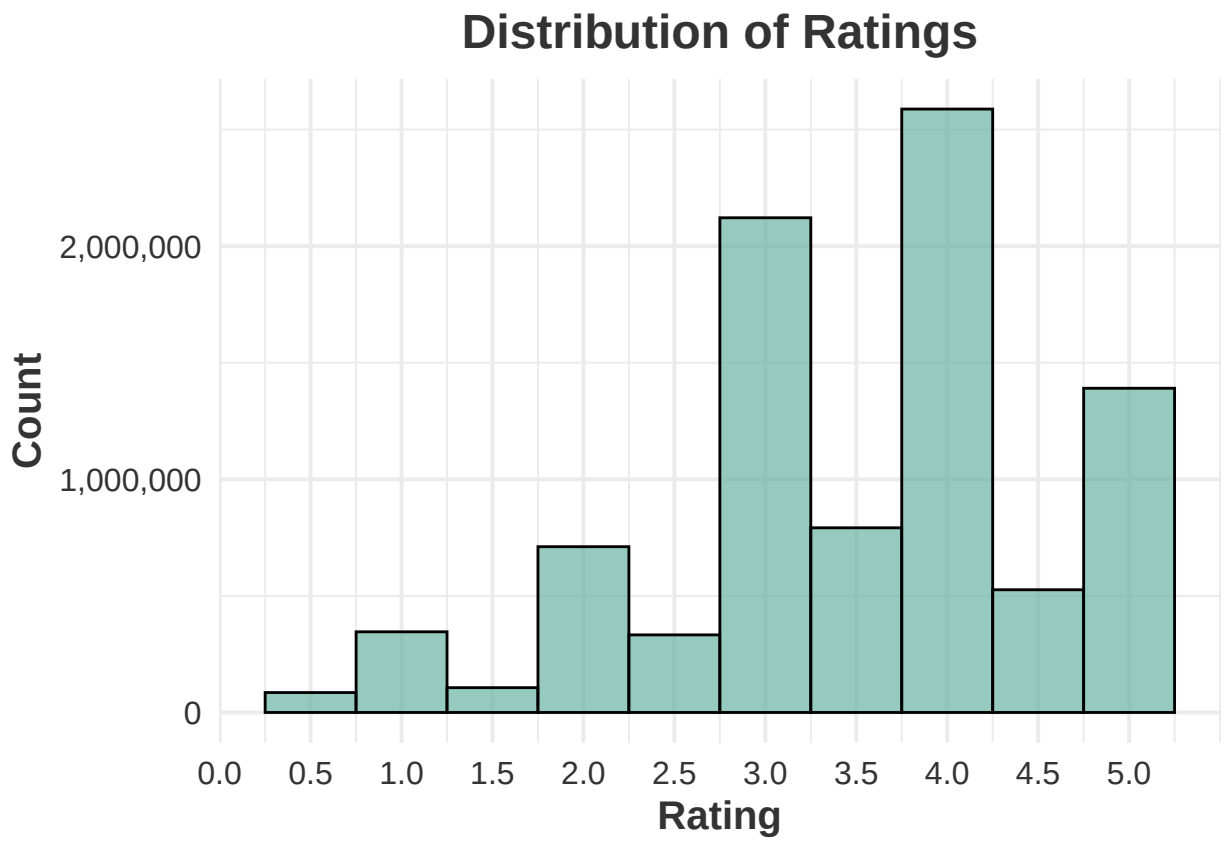
```r
#ratings count
r_count <- edx %>% group_by(rating) %>%
  summarise(ratings_count=n()) %>%
  arrange(desc(ratings_count))
print(r_count)
```

```
## # A tibble: 10 x 2
##    rating ratings_count
##     <dbl>         <int>
##  1    4         2588021
##  2    3         2121638
##  3    5         1390541
##  4    3.5        792037
##  5    2          710998
##  6    4.5        526309
##  7    1          345935
##  8    2.5        332783
##  9    1.5        106379
## 10    0.5         85420
```

Since the rating count is already sorted in descending order it can be clearly inferred that the ratings_count is maximum for rating point 4 of 2588021. It is followed by rating point 3, then 5. Subsequently, rating point 3.5 takes the next place. The rest of the rating takes its respective positions based on the rating count.

The below histogram provides a clear insights about the rating distribution.

```r
#rating distribution
edx %>%
  ggplot(aes(x = rating)) +
  geom_histogram(binwidth = 0.5, fill = "#69b3a2", color = "black", alpha = 0.7) +
  labs(title = "Distribution of Ratings", x = "Rating", y = "Count") +
  theme_minimal(base_size = 15) +
  theme( plot.title = element_text(hjust = 0.5, face = "bold", color = "#333333"), axis.title = element_
  scale_x_continuous(breaks = seq(0, 5, by = 0.5)) +
  scale_y_continuous(labels = scales::comma)
```



Exploring Data based on Genres

From the table of edx dataset it was clearly reflecting that a movie has more than one genres and at rare cases it was a single genre form.

Below exploration would give a clear idea of how many genres are present in the dataset with its count.

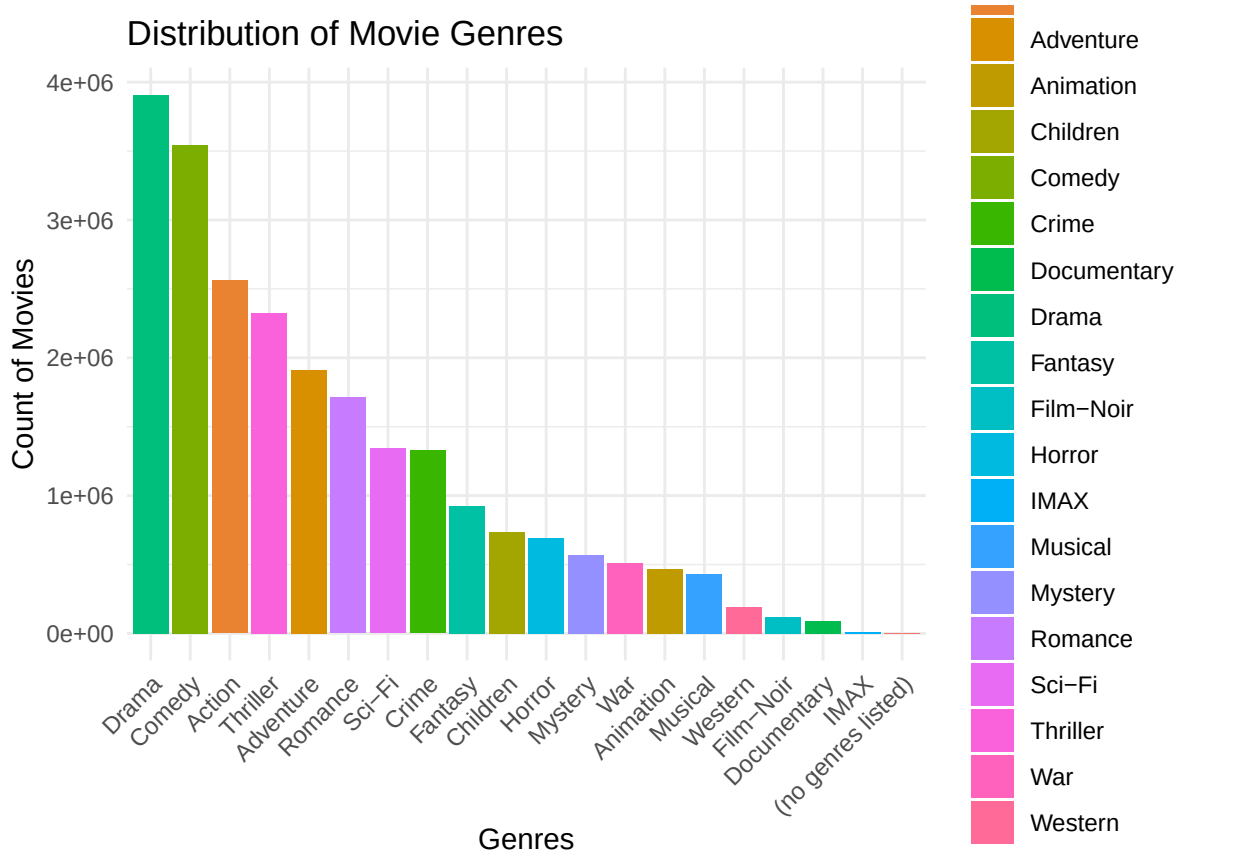
```
#Genre count
g_count <- edx %>% separate_rows(genres,sep="\\|") %>%
  group_by(genres) %>%
  summarise(rating_count=n()) %>%
  arrange(desc(rating_count))

print(g_count)
```

```
## # A tibble: 20 x 2
##    genres             rating_count
##    <chr>                     <int>
##  1 Drama                   3909401
##  2 Comedy                  3541284
##  3 Action                  2560649
##  4 Thriller                2325349
##  5 Adventure               1908692
##  6 Romance                 1712232
##  7 Sci-Fi                  1341750
##  8 Crime                   1326917
##  9 Fantasy                  925624
```

```
## 10 Children                737851
## 11 Horror                  691407
## 12 Mystery                 567865
## 13 War                     511330
## 14 Animation               467220
## 15 Musical                 432960
## 16 Western                 189234
## 17 Film-Noir               118394
## 18 Documentary              93252
## 19 IMAX                      8190
## 20 (no genres listed)           6
```

The below graph would give a detailed insights:

```
# Create the bar plot for genre distribution
ggplot(g_count, aes(x = reorder(genres, -rating_count), y = rating_count, fill = genres)) +
  geom_bar(stat = "identity") +
  labs(title = "Distribution of Movie Genres", x = "Genres", y = "Count of Movies") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

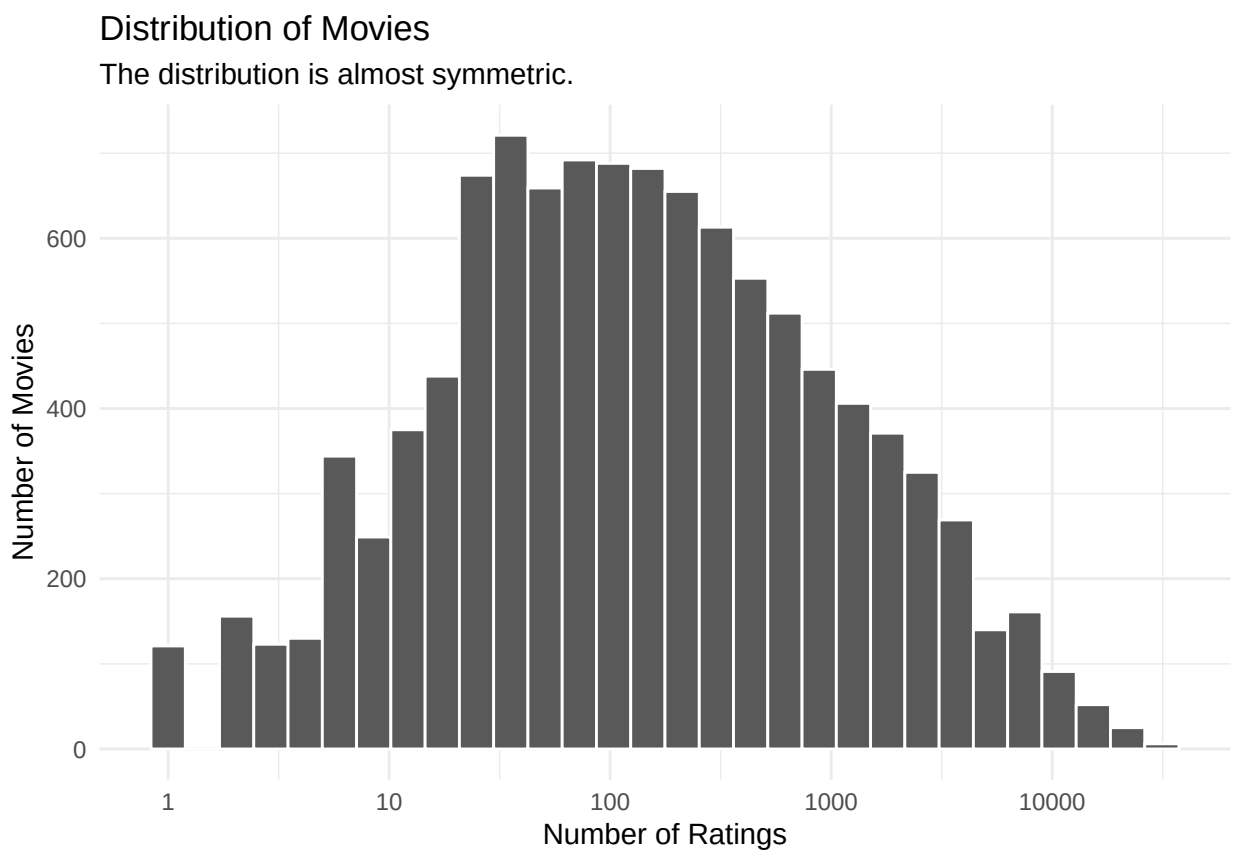


Exploring Data based on Movies

There are 10677 movies present in the edx dataset. It is intuitive to recognize that certain movies receive more ratings than others, as many films are viewed by only a limited number of users, while popular blockbusters typically garner a higher number of ratings.

```
#Movie Distribution

edx %>% group_by(movieId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "white") +
  scale_x_log10() +
  ggtitle("Distribution of Movies",
          subtitle = "The distribution is almost symmetric.") +
  xlab("Number of Ratings") +
  ylab("Number of Movies") +
  theme_minimal()
```

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



Exploring Data based on Users

The edx dataset has a total of 69878 users. It is not always everyone rates and everyone rate all the movies. Only a few of the people are interested in both experiencing the movies and rating them.

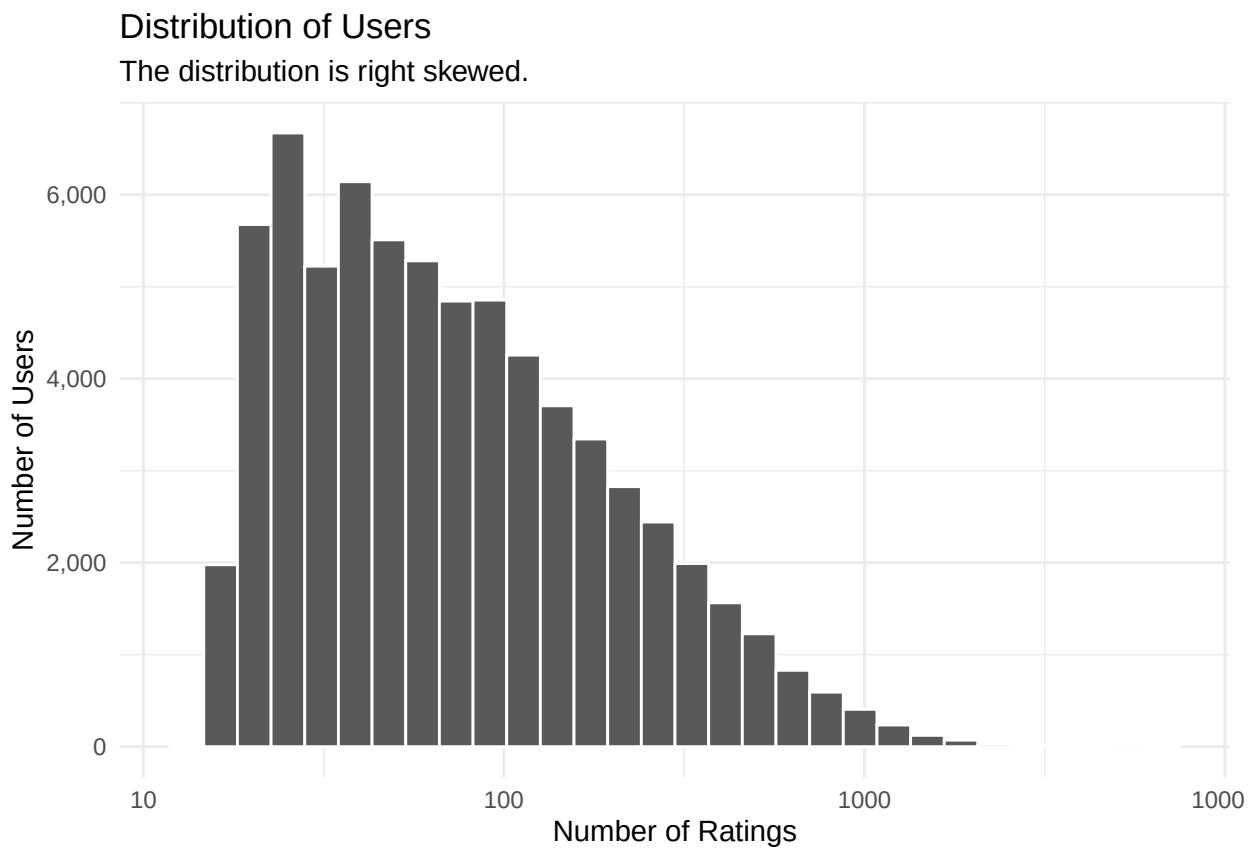
```
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  arrange(n) %>%
  head()
```

```
## # A tibble: 6 x 2
```

```
##   userId     n
##    <int> <int>
## 1  22325    13
## 2  60448    13
## 3  64070    13
## 4   3943    14
## 5  10560    14
## 6  12081    14
```

```
# User Distribution Graph
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "white") +
  scale_x_log10() +
  ggtitle("Distribution of Users",
          subtitle="The distribution is right skewed.") +
  xlab("Number of Ratings") +
  ylab("Number of Users") +
  scale_y_continuous(labels = scales::comma) +
  theme_minimal()
```

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



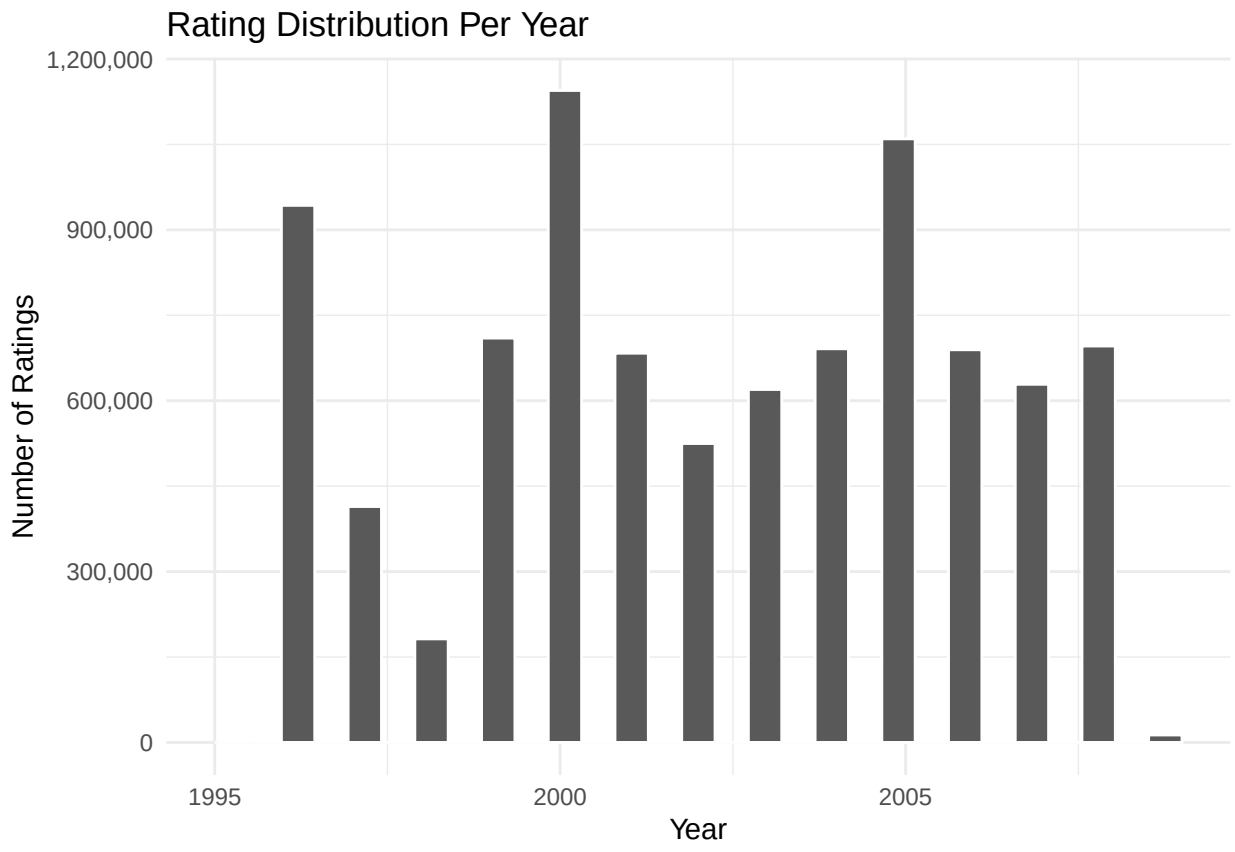
From the above insight it is clear that the histogram is right skewed.

Exploring Data based on Year

The rate distribution per year can be viewed in the below graph

```
# Rating Distribution per year
edx %>% mutate(year = year(as_datetime(timestamp, origin="1970-01-01"))) %>%
  ggplot(aes(x=year)) +
  geom_histogram(color = "white") +
  ggtitle("Rating Distribution Per Year") +
  xlab("Year") +
  ylab("Number of Ratings") +
  scale_y_continuous(labels = scales::comma) +
  theme_minimal()
```

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



Movie with maximum number of ratings can be viewed in the below code snippet:

```
#Movie with maximum number of ratings
edx %>% group_by(movieId,title) %>%
  summarise(ratings_count=n()) %>%
  arrange(desc(ratings_count))
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                ratings_count
##      <int> <chr>                                                        <int>
##  1     296 Pulp Fiction (1994)                                          31336
##  2     356 Forrest Gump (1994)                                          31076
##  3     593 Silence of the Lambs, The (1991)                             30280
##  4     480 Jurassic Park (1993)                                         29291
##  5     318 Shawshank Redemption, The (1994)                             27988
##  6     110 Braveheart (1995)                                            26258
##  7     589 Terminator 2: Judgment Day (1991)                            26115
##  8     457 Fugitive, The (1993)                                         26050
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) ~       25809
## 10     592 Batman (1989)                                                24343
## # i 10,667 more rows
```

Method Approach

This section provides the details regarding the Loss functions - utilized as metrics for evaluation of model performance and different Machine Learning Models employed for the recommendation system.

Loss Functions

1.Root Mean Squared Error (RMSE) Root Mean Squared Error (RMSE) is derived from MSE by taking the square root of the average squared differences. This metric retains the original units of measurement, enhancing interpretability while still penalizing larger errors significantly. RMSE combines the benefits of both MAE and MSE, offering a balance between sensitivity to outliers and ease of understanding. It is widely used in regression analysis to assess model performance effectively.

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

2.Mean Squared Error(MSE) Mean Squared Error (MSE) calculates the average of the squared differences between predicted and actual values. By squaring the errors, MSE emphasizes larger deviations, making it sensitive to outliers and particularly useful when larger errors are more consequential. However, its unit is squared, which can complicate interpretation compared to other metrics. MSE is commonly used as a loss function during model training due to its mathematical properties.

Formula:

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

3. Mean Absolute Error(MAE)

Mean Absolute Error (MAE) measures the average absolute differences between predicted values and actual values. It is calculated by taking the average of the absolute errors, providing a straightforward metric in the same units as the data. MAE is robust to outliers, treating all errors equally, which makes it useful in scenarios where outlier influence should be minimized. Its interpretability allows for easy understanding of average prediction errors.

$MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$

where: The actual value is :y_i The predicted value is : $\hat{y}_i$ the number of observations is : n

Machine Learning models

1. Random Prediction

A fundamental approach to predicting movie ratings is to randomly assign ratings based on observed probabilities from the dataset. For instance, if data analysis reveals that there is a 10% chance of users rating a movie with a score of 3, then it can be assumed that approximately 10% of the ratings will indeed be 3. This method serves as a baseline for model performance; any more sophisticated model should ideally outperform this random prediction.

2. Linear Model

i) Mean of the feature(rating) The simplest predictive model assumes that all users will assign the same rating to every movie, treating variations in ratings as random errors. According to statistical theory, the optimal initial prediction is the average of all observed ratings, expressed mathematically as:

$\hat{Y}_{u,i} = \mu + \epsilon_{i,u}$

In this equation, $\hat{Y}$ represents the predicted rating, and $\epsilon_{i,u}$denotes the error term. By using the mean, RMSE is minimized, providing a solid starting point for predictions.

ii) Movie Effect Model However, it's important to recognize that different movies exhibit distinct rating distributions due to varying popularity and audience preferences. This phenomenon is known as movie bias or movie effect, represented by b_i:

$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{i,u}$

The movie effect can be calculated as the average difference between actual ratings and the overall mean: $\hat{b}_i = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mu)$

iii) User Effect Model Similarly, user-specific preferences also influence ratings; some users tend to rate movies highly while others are more critical. This user bias is represented by b_u:

$\hat{b}_u = \frac{1}{N}\sum_{i=1}^{N}(y_{u,i} - \hat{b}_i - \mu)$

Incorporating user bias into the prediction model results in: $\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$

While this model does not account for genre effects, it lays a foundation for more complex predictions.

3. Regularization

Although linear models provide reasonable predictions, they often overlook cases where certain movies receive very few ratings or where users have limited interactions with movies. Such scenarios can lead to inflated error estimates due to small sample sizes. To address this, regularization techniques can be employed to penalize these small sample sizes without significantly affecting larger datasets. The adjusted calculations for movie and user effects become:

$\hat{b}_i = \frac{1}{n_i+\lambda}\sum_{u=1}^{n_i}(y_{u,i} - \mu)$

$\hat{b}_u = \frac{1}{n_u+\lambda}\sum_{i=1}^{n_u}(y_{u,i} - \hat{b}_i - \mu)$

Here, N refers to the number of ratings for each movie or user, and λ is a regularization parameter that helps control the influence of small sample sizes on predictions. By experimenting with various values of λ one can identify an optimal setting that minimizes RMSE.

4. Matrix Factorization

Matrix factorization uncovers latent factors in user-item rating matrices, mapping users and movies to these underlying dimensions. This technique can be likened to approximating a prime number, such as 61, through multiplication, illustrating how complex relationships within data can be simplified. It is essential for efficient calculations in linear algebra and is related to methods like singular value decomposition (SVD) and principal component analysis (PCA). The predictive model can be expressed as: $\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$

However, to capture variations from similar rating patterns among groups of users and movies, we analyze residuals defined by: $r_{u,i} = y_{u,i} - \hat{b}_i - \hat{b}_u$

This structured approach enhances the effectiveness of recommendation systems.

Results

Loss Functions

Here, the Loss functions - RMSE,MSE and MAE are defined:

```
#Defining the loss function calculation-RMSE,MAE,MSE

#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))
}
```

Machine Learning Models Fitting and Evaluation

Machine Learning Model 1 : Random Prediction

The initial model employs a random prediction approach by utilizing the observed probabilities from the training set to predict ratings.

To implement this, determination of the probability of each rating within the training data is mandatory. Subsequently, these probabilities are used to predict ratings for the test set, which are then compared against the actual ratings. Ideally, any predictive model developed should outperform this baseline random model. Given that the training set represents only a sample of the broader population and we lack knowledge of the true distribution of ratings, employing Monte Carlo simulations with replacement offers a robust method for approximating this distribution. This technique allows us to generate multiple samples from our training data, enabling us to better understand the variability and potential outcomes of movie ratings.

Additionally, using Monte Carlo simulations can help in assessing the uncertainty associated with predictions. By running numerous iterations, we can create a distribution of predicted ratings for each movie, which

provides insights into expected performance and confidence intervals. This approach not only enhances our understanding of user preferences but also lays the groundwork for more sophisticated models that can incorporate these probabilistic insights into their predictions. Ultimately, this methodology serves as a valuable stepping stone towards a development of more accurate and reliable recommendation systems.

```
## Predicting on a random basis

set.seed(1990)

# Create the probability of each rating
p <- function(x, y) mean(y == x)
rating <- seq(0.5 , 5 , 0.5)

# Using Monte Carlo simulation estimating the rating individually
B <- 10000
M <- replicate(B, {
  s <- sample(train_data$rating, 100, replace = TRUE)
  sapply(rating, p, y= s)
})
prob <- sapply(1:nrow(M), function(x) mean(M[x,]))

# Predict random ratings
y_hat <- sample(rating, size = nrow(test_data),
                        replace = TRUE, prob = prob)


# Create a tibble for  the error results
result <- tibble(Method = "Project Goal", RMSE = 0.86490, MSE = NA, MAE = NA)

result <- bind_rows(result,
                    tibble(Method = "Random prediction",
                           RMSE = RMSE(test_data$rating, y_hat),
                           MSE  = MSE(test_data$rating, y_hat),
                           MAE  = MAE(test_data$rating, y_hat)))

print(result)
```

```
## # A tibble: 2 x 4
##   Method             RMSE   MSE   MAE
##   <chr>             <dbl> <dbl> <dbl>
## 1 Project Goal      0.865 NA    NA
## 2 Random prediction 1.50   2.25  1.17
```

Random Prediction Model is providing a higher value than the goal value in terms of the RMSE. The random Prediction model yields RMSE of about 1.500901 which is lesser than 0.86490

Machine Learning Model 2 : Mean of the rating

Using the mean of the rating model may yield better results than the random prediction model but it is not sure that it may yield a lower RMSE than the quoted RMSE of 0.86490.

The Model is developed based on this formula:

$\hat{Y}_{u,i} = \mu + \epsilon_{i,u}$

```r
## Using the average/mean of the rating
mu <- mean(train_data$rating)

result <- bind_rows(result,
                    tibble(Method = "Average/Mean Rating",
                           RMSE = RMSE(test_data$rating, mu),
                           MSE  = MSE(test_data$rating, mu),
                           MAE  = MAE(test_data$rating, mu)))
print(result)
```

```
## # A tibble: 3 x 4
##   Method               RMSE   MSE    MAE
##   <chr>               <dbl> <dbl>  <dbl>
## 1 Project Goal        0.865 NA    NA
## 2 Random prediction   1.50   2.25  1.17
## 3 Average/Mean Rating 1.06   1.13  0.856
```

While the mean prediction model yields a lower RMSE of 1.060054 compared to the random prediction model's RMSE of 1.500901, it still exhibits signs of overfitting, which undermines its effectiveness as a reliable predictive tool.

To enhance the accuracy of predictions, transitioning from a mean-based approach to a linear model is advisable. The mean prediction method does not account for the influences of specific movie IDs or user IDs, which can significantly affect ratings.

By incorporating both movie and user effects into the linear model, we can capture these critical variations, leading to a more accurate estimation of ratings. This adjustment is expected to improve the RMSE further, as it allows for a nuanced understanding of how individual users rate different movies based on their unique preferences and the inherent characteristics of each film. Overall, refining the model in this way should enhance its predictive performance and reduce the risk of overfitting.

Machine Learning Model 3 : Movie Effect

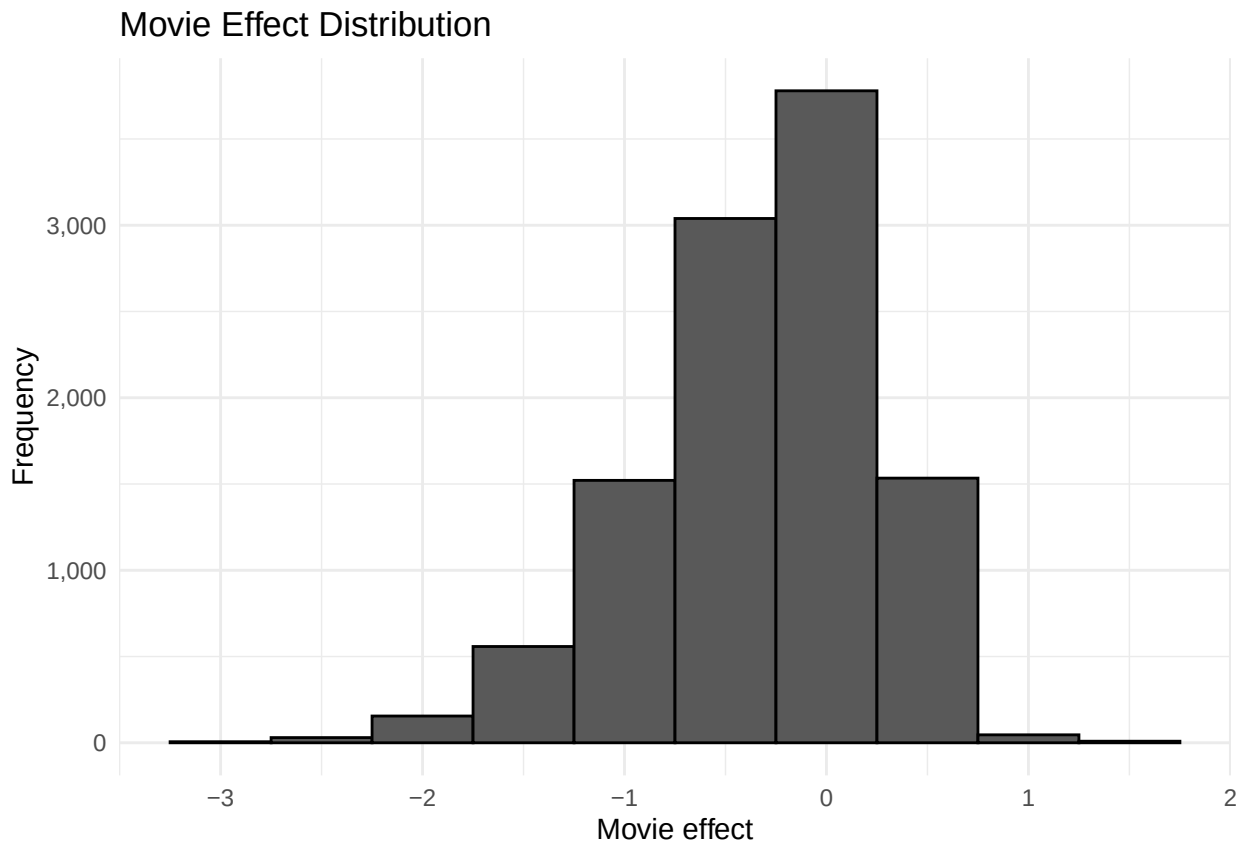
Calculate the b_i and utilize it to the previous formula and develop the new model Movie effect model/Movie bias model. The formula becomes:

$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{i,u}$

```r
# Movie Effect

b_i <- train_data %>%
  group_by(movieId) %>%
  summarize(bi = mean(rating - mu))
head(b_i)
```

```
## # A tibble: 6 x 2
##   movieId     bi
##     <int>  <dbl>
## 1       1  0.411
## 2       2 -0.311
## 3       3 -0.361
## 4       4 -0.638
## 5       5 -0.430
## 6       6  0.299
```

```
#### Movie Effect is distributed in a left skewed manner
# Visualization of Movie Effect Distribution
b_i %>% ggplot(aes(x = bi)) +
  geom_histogram(bins = 10, col = I("black")) +
  ggtitle("Movie Effect Distribution") +
  xlab("Movie effect") +
  ylab("Frequency") +
  scale_y_continuous(labels = scales :: comma) +
  theme_minimal()
```




```
# Predict the rating
y_hat_bi <- mu + test_data %>%
  left_join(b_i, by = "movieId") %>%
  pull(bi)

#Calculate the Loss Functions
result <- bind_rows(result,
                    tibble(Method = "Movie Effect Model",
                           RMSE = RMSE(test_data$rating, y_hat_bi),
                           MSE  = MSE(test_data$rating, y_hat_bi),
                           MAE  = MAE(test_data$rating, y_hat_bi)))
print(result)
```

```
## # A tibble: 4 x 4
##   Method                RMSE    MSE    MAE
```

```
##   <chr>               <dbl>  <dbl>  <dbl>
## 1 Project Goal        0.865 NA     NA
## 2 Random prediction   1.50   2.25   1.17
## 3 Average/Mean Rating 1.06   1.13   0.856
## 4 Movie Effect Model  0.943  0.889  0.738
```

The movie effect model yields a better RMSE value than the rest of the Machine Learning Models of about 0.9429615 but still lags behind the project goal. Improvising this model can lead to further improvement in the RMSE value.

Machine Learning Model 4 : User Effect

Calculating b_u which is the user bias and adding it to the previous model develops a new model - User effect model/User bias model

The formula becomes:

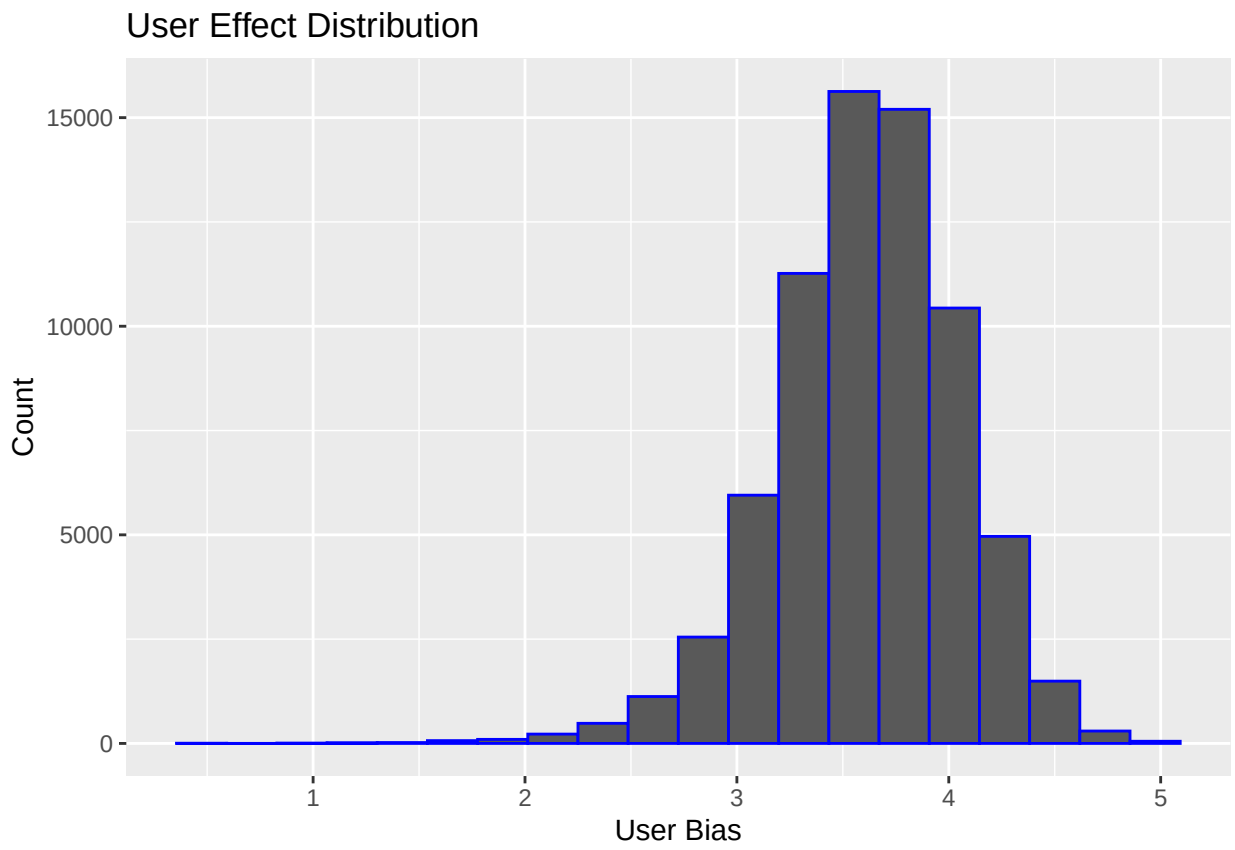
$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$

```
b_u <- train_data %>%
  left_join(b_i, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(bu = mean(rating - mu - bi))

# Prediction
y_hat_bi_bu <- test_data %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(prediction = mu + bi + bu) %>%
  pull(prediction)

# Update the result tibble
result <- bind_rows(result,
                    tibble(Method = "User Effect Model",
                           RMSE = RMSE(test_data$rating, y_hat_bi_bu),
                           MSE  = MSE(test_data$rating, y_hat_bi_bu),
                           MAE  = MAE(test_data$rating, y_hat_bi_bu)))
print(result)
```

```
## # A tibble: 5 x 4
##   Method               RMSE    MSE    MAE
##   <chr>               <dbl>  <dbl>  <dbl>
## 1 Project Goal        0.865 NA     NA
## 2 Random prediction   1.50   2.25   1.17
## 3 Average/Mean Rating 1.06   1.13   0.856
## 4 Movie Effect Model  0.943  0.889  0.738
## 5 User Effect Model   0.865  0.748  0.669
```

```
# visualization of user effect distribution
train_data %>%
  group_by(userId) %>%
  summarize(bu = mean(rating)) %>%
  filter(n()>=100) %>%
```

```
  ggplot(aes(bu)) +
  geom_histogram(bins = 20, color='blue') +
  ggtitle("User Effect Distribution") +
  xlab("User Bias") +
  ylab("Count")
```



Based on the graph it is regularly distributed. But on account of the RMSE value, improvement can be seen relatively as it outperforms the rest of model developed till now. But still lies closer to the Project Goal 0.86490 as the RMSE obtained is 0.8646843. This may sound lesser than the Goal value but if used for the final dataset there are chances of underperformance.

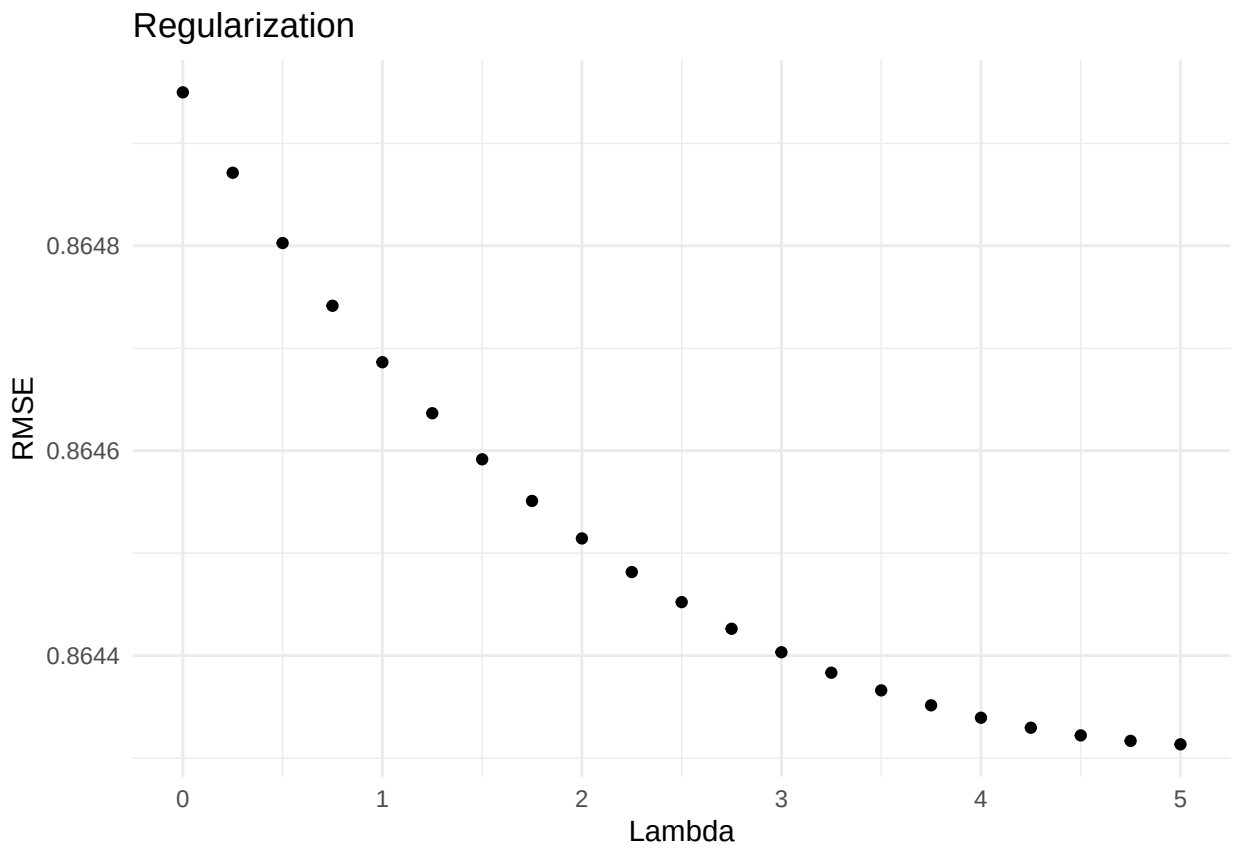
Consideration of the genre effect may develop the model to a more advanced form.

Machine Learning Model 5 : Regularized Movie and User Effect

Now, regularization of the user and movie effects model can be done by adding a penalty factor λ,a tuning parameter. Definition of range of λ values are performed within which the best value yielding a lower RMSE is selected.

```
# Regularization

lambdas <- seq(0, 5 , 0.25)

rmse <- sapply(lambdas,function(l){
  # Mean
```

```r
  mu <- mean(train_data$rating)

  # Movie effect (bi)
  bi <- train_data %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(n()+l))

  # User effect (bu)
  bu <- train_data %>%
    left_join(bi, by="movieId") %>%
    filter(!is.na(bi)) %>%
    group_by(userId) %>%
    summarize(bu = sum(rating - bi - mu)/(n()+l))

  # Prediction: mu + bi + bu
  predicted_ratings <- test_data %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    filter(!is.na(bi), !is.na(bu)) %>%
    mutate(prediction = mu + bi + bu) %>%
    .$prediction

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

#Plot the Lambdas vs RMSE

tibble(Lambda = lambdas, RMSE = rmse) %>%
  ggplot(aes(x = Lambda, y = RMSE)) +
  geom_point() +
  ggtitle("Regularization") +
  theme_minimal()
```



The selected λ is used for the model.

```
# Consider the lambda that returns the lowest RMSE value.
lambda <- lambdas[which.min(rmse)]

# calculate the predicted rating using the fittest parameters
# achieved through regularization.
mu <- mean(train_data$rating)

# Movie effect (bi)
bi <- train_data %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(n()+lambda))

# User effect (bu)
bu <- train_data %>%
  left_join(bi, by="movieId") %>%
  group_by(userId) %>%
  summarize(bu = sum(rating - bi - mu)/(n()+lambda))

# Prediction
predicted_reg <- test_data %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  mutate(prediction = mu + bi + bu) %>%
  pull(prediction)
```

```r
# Update the result table
result <- bind_rows(result,
                    tibble(Method = "Regularized movie and user effect model",
                           RMSE = RMSE(test_data$rating, predicted_reg),
                           MSE  = MSE(test_data$rating, predicted_reg),
                           MAE  = MAE(test_data$rating,predicted_reg)))
print(result)
```

```
## # A tibble: 6 x 4
##   Method                                    RMSE    MSE    MAE
##   <chr>                                    <dbl>  <dbl>  <dbl>
## 1 Project Goal                             0.865 NA     NA
## 2 Random prediction                        1.50   2.25   1.17
## 3 Average/Mean Rating                      1.06   1.13   0.856
## 4 Movie Effect Model                       0.943  0.889  0.738
## 5 User Effect Model                        0.865  0.748  0.669
## 6 Regularized movie and user effect model 0.864  0.747  0.669
```

The regularized model yields 0.8641362 which is lesser than the project goal 0.86490

Machine Learning Model 6: Matrix Factorization

Using the recosystem package the Matrix Factorization model can be implemented by:

```r
# Matrix Factorization

set.seed(1990)

# Convert the train and test data into recosystem input format
train_set <-  with(train_data, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))
test_set  <-  with(test_data,  data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))

# Create the model object
r <-  recosystem::Reco()

# Select the best tuning parameters
opts <- r$tune(train_set, opts = list(dim = c(10, 20, 30),
                                      lrate = c(0.1, 0.2),
                                      costp_l2 = c(0.01, 0.1),
                                      costq_l2 = c(0.01, 0.1),
                                      nthread  = 4, niter = 10))

# Train the algorithm
r$train(train_set, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
```

```
##    0       0.9820   1.1047e+07
##    1       0.8752   8.9836e+06
##    2       0.8432   8.3466e+06
##    3       0.8212   7.9644e+06
##    4       0.8045   7.6930e+06
##    5       0.7921   7.5025e+06
##    6       0.7820   7.3574e+06
##    7       0.7733   7.2421e+06
##    8       0.7660   7.1453e+06
##    9       0.7595   7.0688e+06
##   10       0.7540   7.0001e+06
##   11       0.7490   6.9438e+06
##   12       0.7446   6.8944e+06
##   13       0.7404   6.8506e+06
##   14       0.7368   6.8095e+06
##   15       0.7335   6.7776e+06
##   16       0.7303   6.7465e+06
##   17       0.7274   6.7177e+06
##   18       0.7248   6.6945e+06
##   19       0.7223   6.6709e+06
```

```
#Estimating the predicted values
predict_reco <-  r$predict(test_set, out_memory())
head(predict_reco, 10)
```

```
##  [1] 4.908112 4.586782 3.643171 3.125633 1.991844 3.864915 4.148281 4.146956
##  [9] 4.317591 3.619072
```

```
result <- bind_rows(result,
                    tibble(Method = "Matrix Factorization - recosystem",
                           RMSE = RMSE(test_data$rating, predict_reco),
                           MSE  = MSE(test_data$rating, predict_reco),
                           MAE  = MAE(test_data$rating, predict_reco)))
print(result)
```

```
## # A tibble: 7 x 4
##   Method                                   RMSE    MSE    MAE
##   <chr>                                   <dbl>  <dbl>  <dbl>
## 1 Project Goal                            0.865 NA     NA
## 2 Random prediction                       1.50   2.25   1.17
## 3 Average/Mean Rating                     1.06   1.13   0.856
## 4 Movie Effect Model                      0.943  0.889  0.738
## 5 User Effect Model                       0.865  0.748  0.669
## 6 Regularized movie and user effect model 0.864  0.747  0.669
## 7 Matrix Factorization - recosystem       0.786  0.617  0.606
```

The result of the Matrix Factorization model yields the lowest RMSE value than the earlier trained models. The RMSE value Yielded by this model is 0.7850868 which is lesser than the Project Goal 0.86490

Implementation of Machine Learning Models into final_holdout_test

On comparing all the models trained in the above result table, it is the Regularized movie and user effect model and Matrix Factorization model has a lower RMSE value than the rest of the models. So, far only

the train and test set of the edx dataset has been used for model development and testing. Now, the entire edx dataset is used as a train set and final_holdout_test will be used for the final testing model

Using Regularized movie and user effect model

Applying the regularized movie and user effect model:

```
#Implementing the Regularized movie and user effect model on final_holdout_test

#Mean of rating from edx data set
mu_edx <- mean(edx$rating)

# Movie effect (bi)
b_i_edx <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_edx)/(n()+lambda))

# User effect (bu)
b_u_edx <- edx %>%
  left_join(b_i_edx, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_edx)/(n()+lambda))

# Prediction
predict_edx <- final_holdout_test  %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  mutate(prediction = mu_edx + b_i + b_u) %>%
  pull(prediction)

#  results tibble for final_holdout_test
result_final <- tibble(Method = "Project Goal", RMSE = 0.86490, MSE = NA, MAE = NA)
result_final <- bind_rows(result_final,
                    tibble(Method = "Regularized movie and user effect model",
                           RMSE = RMSE(final_holdout_test $rating, predict_edx),
                           MSE  = MSE(final_holdout_test $rating, predict_edx),
                           MAE  = MAE(final_holdout_test $rating, predict_edx)))

# Show the RMSE improvement
print(result_final )
```

```
## # A tibble: 2 x 4
##   Method                                  RMSE    MSE    MAE
##   <chr>                                  <dbl>  <dbl>  <dbl>
## 1 Project Goal                           0.865 NA     NA
## 2 Regularized movie and user effect model 0.865  0.748  0.670
```

The RMSE of this model is 0.8648177 when applied to the final_holdout_test dataset. This is lesser than the Project Goal 0.86490. On both the training dataset and this final dataset, this model has performed phenomenal keeping RMSE low as far as it can.

Using Matrix Factorization model

Applying the Matrix Factorization model:

```r
#Implementing the Matrix Factorization model on final_holdout_test

set.seed(1990)

# Transforming the 'edx' and 'validation' sets to recosystem datasets
edx.reco <-  with(edx, data_memory(user_index = userId,
                                   item_index = movieId,
                                   rating = rating))
final_holdout_test.reco  <-  with(final_holdout_test, data_memory(user_index = userId,
                                                    item_index = movieId,
                                                    rating = rating))

# Creating the reco model object
r <-  recosystem::Reco()

# Parameter Tuning
opts <-  r$tune(edx.reco, opts = list(dim = c(10, 20, 30),
                                      lrate = c(0.1, 0.2),
                                      costp_l2 = c(0.01, 0.1),
                                      costq_l2 = c(0.01, 0.1),
                                      nthread  = 4, niter = 10))

# Model Training
r$train(edx.reco, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9718   1.2013e+07
##    1       0.8715   9.8757e+06
##    2       0.8377   9.1606e+06
##    3       0.8161   8.7439e+06
##    4       0.8006   8.4653e+06
##    5       0.7889   8.2686e+06
##    6       0.7791   8.1149e+06
##    7       0.7711   7.9987e+06
##    8       0.7643   7.9037e+06
##    9       0.7585   7.8228e+06
##   10       0.7534   7.7577e+06
##   11       0.7490   7.6978e+06
##   12       0.7448   7.6500e+06
##   13       0.7410   7.6027e+06
##   14       0.7376   7.5651e+06
##   15       0.7345   7.5325e+06
##   16       0.7315   7.5003e+06
##   17       0.7288   7.4723e+06
##   18       0.7263   7.4478e+06
##   19       0.7239   7.4231e+06
```

```r
#Prediction
predict_reco_final <-  r$predict(final_holdout_test.reco, out_memory())

# Update the result table
result_final <- bind_rows(result_final,
                    tibble(Method = "Final Matrix Factorization - Validation",
```

```
                            RMSE = RMSE(final_holdout_test$rating, predict_reco_final),
                            MSE  = MSE(final_holdout_test$rating, predict_reco_final),
                            MAE  = MAE(final_holdout_test$rating, predict_reco_final)))
print(result_final)
```

```
## # A tibble: 3 x 4
##   Method                                    RMSE    MSE    MAE
##   <chr>                                    <dbl>  <dbl>  <dbl>
## 1 Project Goal                             0.865 NA     NA
## 2 Regularized movie and user effect model 0.865  0.748  0.670
## 3 Final Matrix Factorization - Validation 0.783  0.613  0.604
```

Matrix Factorization model has got the RMSE of 0.784 when applied to the final_holdout_test dataset. This is lesser than the Project Goal 0.86490. This model has managed in keeping RMSE low as far as it can on both training and test datasets.

Conclusion

From the above inference it is clear that the Matrix Factorization has better performance than the Regularized movie and user effect model, having RMSE of 0.7831877 .

In conclusion, the MovieLens dataset project has successfully demonstrated the effectiveness of Matrix Factorization as the final predictive model for generating movie recommendations. After conducting a comprehensive analysis that included various modeling approaches and thorough validations, matrix factorization emerged as the most robust method, providing superior accuracy and generalization compared to the rest of the models.

Overall, the Matrix Factorization model satisfies the project goal of having RMSE lesser than 0.86490 .

References

https://rpubs.com/vsi/movielens

A Matrix-factorization Library for Recommender Systems - https://www.csie.ntu.edu.tw/~cjlin/libmf/

Rafael A. Irizarry (2019), Introduction to Data Science: Data Analysis and Prediction Algorithms with R

Yixuan Qiu (2017), recosystem: recommendation System Using Parallel Matrix Factorization

Michael Hahsler (2019), recommendationlab: Lab for Developing and Testing recommendation Algorithms. R package version 0.2-5.

Georgios Drakos, How to select the Right Evaluation Metric for Machine Learning Models: Part 1 Regression Metrics

Bickel Peter J and Li Bo (2006), recosystem: recommendation System Using Parallel Matrix Factorization

Jason Brownlee (2019), A Gentle Introduction to Matrix Factorization for Machine Learning

Jason Brownlee (2020), Machine Learning Mastery with R, Get started, Build Accurate Models, and Work through Projects step-by-step

Ong Cheng Soon (2005), Kernels: Regularization and Optimization

Vijay Kotu, Bala Deshpande, (2019), [Recomendation Ssystem: Matrix Factorization] (https://www.sciencedirect.com/topics/computer-science/matrix-factorization)