HarvardX
PH125.9x Data Science
MovieLens Rating Prediction Project

AHMED EL HAMOUNI
JUNE 06, 2023

Contents

1- Intro

1.1 General Summary. 1

1.2 Introduction 1

1.3 Strategy & Aim 2

2- Methology

2.1 Download And Preparation Of The Data. 3

2.2 Validation Set. 5

2.3 Exploration & Visualization. 5

2.4 Modeling. 9

2.5 Building Models. 10

2.5.1 Baseline Model. 10
2.5.2 Movie Effect Model. 11
2.5.3 User Movie Effect 11
2.5.4 Regularization. 12
2.5.5 Matrix factorization. 14

3- The Final Results 15

4- The Conclusion. 16

1- Intro

1.1 General Summary

This report is focused on the MovieLens Project, which is part of the HervardX: PH125.9x Data Science: Capstone course. The purpose of this report is to provide an overview of the project and its objectives.

Initially, the report discusses the project's general idea and outlines its specific objectives. It then proceeds to describe the preparation and setup of the provided dataset. An exploratory data analysis is conducted to gain insights and inform the development of a machine learning algorithm capable of predicting movie ratings. The report further presents the process of building the final model.

The results obtained from the developed machine learning algorithm are thoroughly explained, highlighting the model's performance and its ability to predict movie ratings accurately. The report concludes by providing concluding remarks, summarizing the key findings and insights derived from the project.

Overall, this report serves as a comprehensive documentation of the MovieLens Project, from data preparation to model development and evaluation. It provides valuable information for understanding the project's objectives and outcomes, showcasing the practical application of data science techniques in predicting movie ratings.

1.2 Introduction

The MovieLens dataset consists of more than 10 million ratings provided by over 72,000 users for over 10,000 movies, according to the GroupLens website. In an ideal scenario where every user rated every movie, we would expect around 720 million ratings. However, the dataset contains only 10 million ratings, indicating that a significant portion of movies has not been rated, resulting in a "Sparse Matrix" representation.

To address this, I plan to build a machine learning algorithm that aims to predict user ratings with high accuracy using the available data. The performance of different models will be evaluated using the Root Mean Square Error (RMSE) metric during the development phase. To facilitate this process, I will utilize subsets of the MovieLens dataset as described below:

The edx dataset, which holds 90% of the MovieLens data, will be used for model development. However, the remaining 10% of the data, treated as the true ratings, cannot be used to assess model performance during development. Therefore, the edx dataset will be divided into two subsets:

train_set: This subset will contain 80% of the edx data and will be used to train different models.

test_set: Comprising 20% of the edx data, this subset will be employed to evaluate the performance of various models.

A validation dataset, encompassing 10% of the MovieLens data, will be exclusively used to calculate the final RMSE for evaluating the best-performing model.

The dataset includes information such as a unique userID assigned to each user, a movieID assigned to each movie, movie ratings, the timestamp of each rating, movie titles, and movie genres.

Given the large size of the dataset, linear models (lm) were not employed for data fitting and prediction. Instead, I will utilize Least Squares Estimates (LSE) to predict the ratings and assess their accuracy using the RMSE metric as mentioned earlier. My objective is to achieve an RMSE value of less than 0.80.

1.3 Strategy & Aim

The techniques employed in this project are focused on creating a movie recommendation system that relies on user ratings. As previously mentioned, Least Squares Estimates (LSE) will be utilized instead of Linear Regression models, considering the substantial data volume. In the following section, I will outline the key steps involved in this process, encompassing data preparation, data exploration, visualization, and the valuable insights obtained. These steps will ultimately lead to the development of the recommendation model.

2 - Methology

You can access information about the MovieLens 10M dataset by visiting this link: https://grouplens.org/datasets/movielens/10m/.

To obtain the dataset, you can download it from this link: http://files.grouplens.org/datasets/movielens/ml-10m.zip.

The following code snippet demonstrates how to download the dataset and combine the ratings and movie information into a single dataset called "movielens." Additionally, it creates the edx and validation datasets, which will be utilized for algorithm development and testing

2.1 - Download And Preparation Of The Data

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

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

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

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


library(tidyverse)

library(caret)

library(data.table)

library(recommenderlab)

library(ggplot2)

library(recosystem)
```

```
> if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.o
rg")
── Attaching core tidyverse packages ──────────────────────────────────────────
──────────────────────────────── tidyverse 2.0.0 ──
✓ dplyr     1.1.2     ✓ readr     2.1.4
✓ forcats   1.0.0     ✓ stringr   1.5.0
✓ ggplot2   3.4.2     ✓ tibble    3.2.1
✓ lubridate 1.9.2     ✓ tidyr     1.3.0
✓ purrr     1.0.1
── Conflicts ──────────────────────────────────────────────────────────────────
────────────────────────── tidyverse_conflicts() ──
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()    masks stats::lag()
```

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

```
dl <- tempfile()

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


ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),

          col.names = c("userId", "movieId", "rating", "timestamp"))


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

colnames(movies) <- c("movieId", "title", "genres")


# if using R 4.0 or later:

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),

                        title = as.character(title),

                        genres = as.character(genres))




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

2.2 - Validation Set

```
# Validation set will be 10% of MovieLens data

set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `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 validation set are also in edx set

validation <- temp %>%

  semi_join(edx, by = "movieId") %>%

  semi_join(edx, by = "userId")


# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)

edx <- rbind(edx, removed)


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

2.3 - Exploration & Visualization

With the dataset now created, it is essential to perform data exploration to develop the approach for building the algorithm. The following steps will be taken:

- Determination of the number of users and movies in the dataset. This analysis will provide insights into the dataset's size and distribution.

- Plotting the top 20 rated movies. This visualization will showcase the most popular movies based on user ratings, giving an overview of the dataset's preferences.

- Exploring the number of ratings per movie and per user. Separate plots will be generated to examine the distribution of ratings across movies and users, providing a better understanding of the dataset's rating patterns.

-By conducting these data exploration activities, we can gain valuable insights that will guide the development of the algorithm.

```
# Checking the number of unique users that provided ratings and how many unique movies were rated
edx %>% summarize(n_users = n_distinct(userId),n_movies = n_distinct(movieId))
```

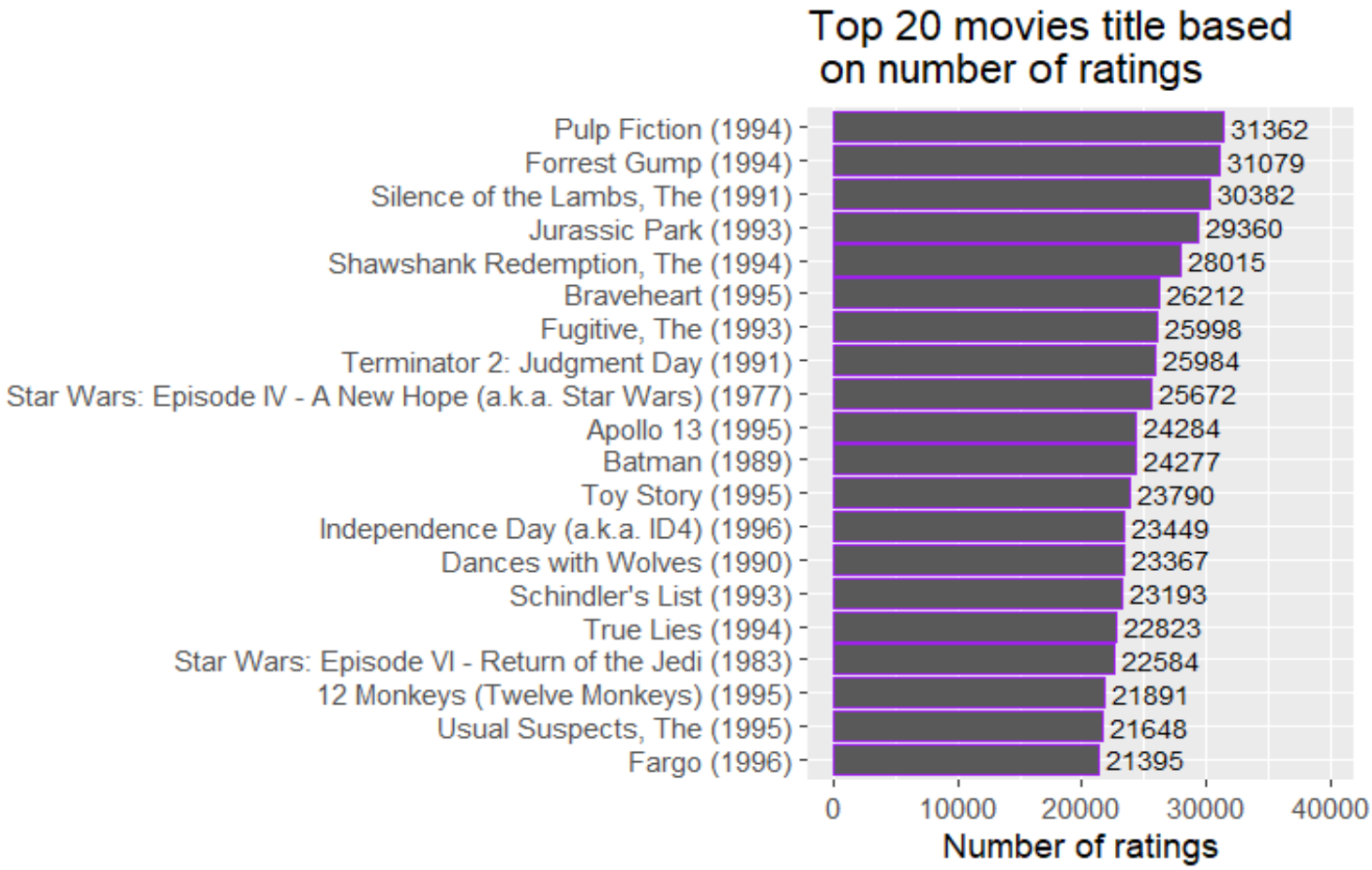
Exploration & Visualization

Now that the dataset is created, some data exploration will be needed to start building the approach with which the algorithm will be developed. Below I First explore how many users & movies are in the dataset, then I plot the top 20 rated movies, then I explore the number of ratings by movie & by user in separate plots.

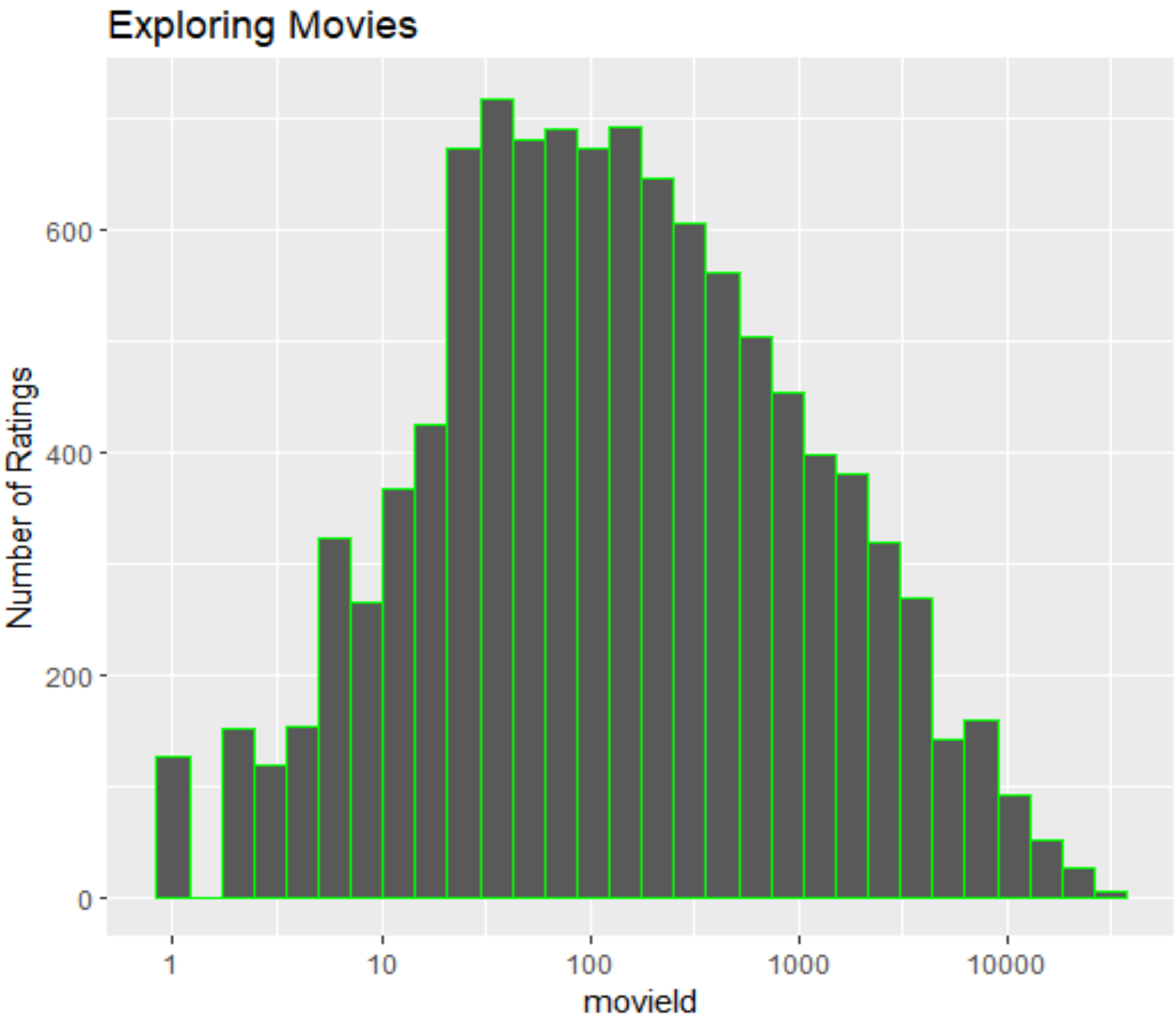
```
# Checking the number of unique users that provided ratings and how many unique movies were rated
edx %>% summarize(n_users = n_distinct(userId),n_movies = n_distinct(movieId))
```

```
  n_users    n_movies
1   69878    10677
```

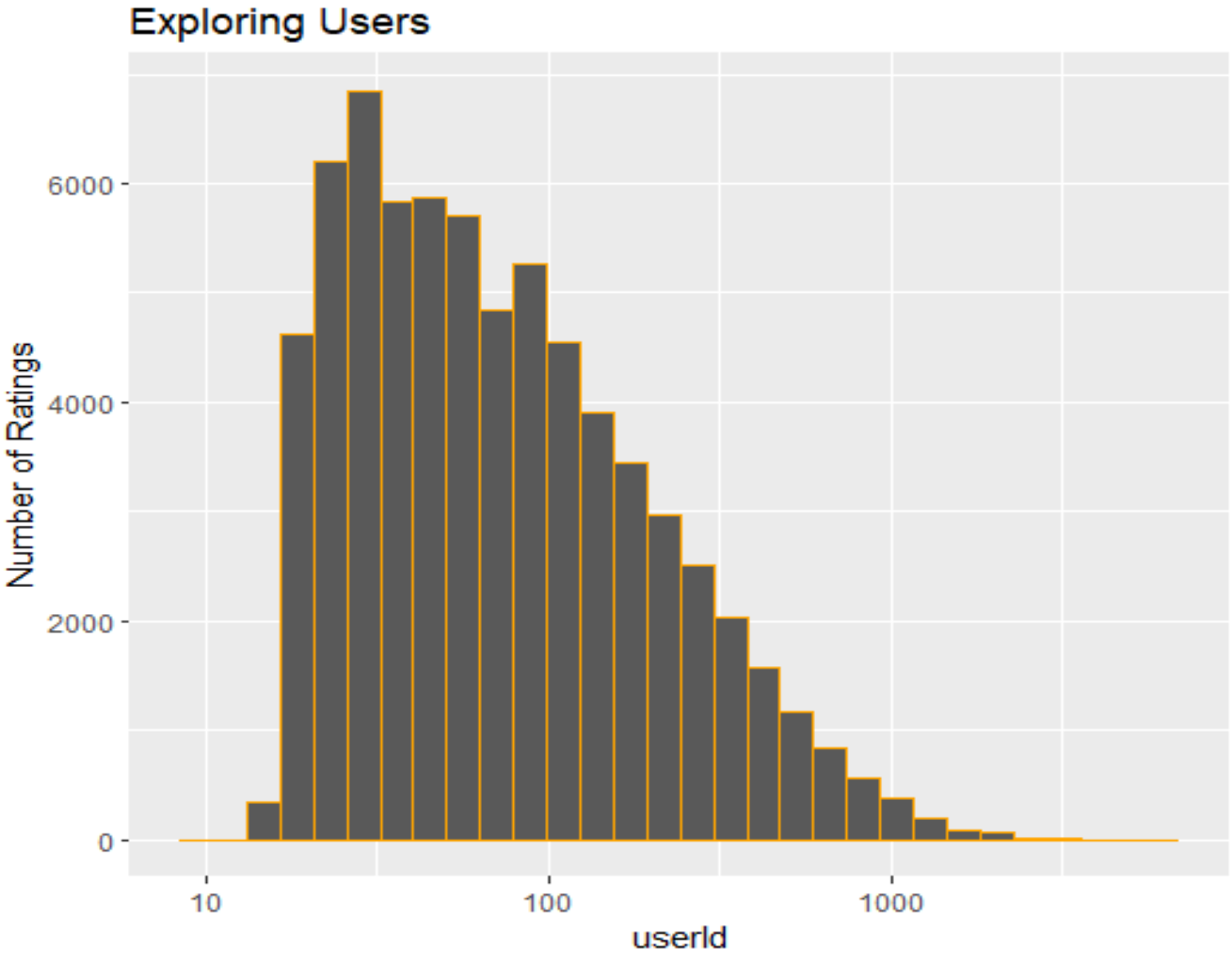
```
# Plotting top 20 movies
top_20 <- edx %>% group_by(title) %>% summarize(count=n()) %>% top_n(20,count) %>%
arrange(desc(count))
top_20 %>%
ggplot(aes(x=reorder(title, count), y=count)) +
geom_bar(stat="identity",color= "purple") + coord_flip(y=c(0, 40000)) +
labs(x="", y="Number of ratings") +
geom_text(aes(label= count), hjust=-0.1, size=3) +
labs(title="Top 20 movies title based \n on number of ratings")
```



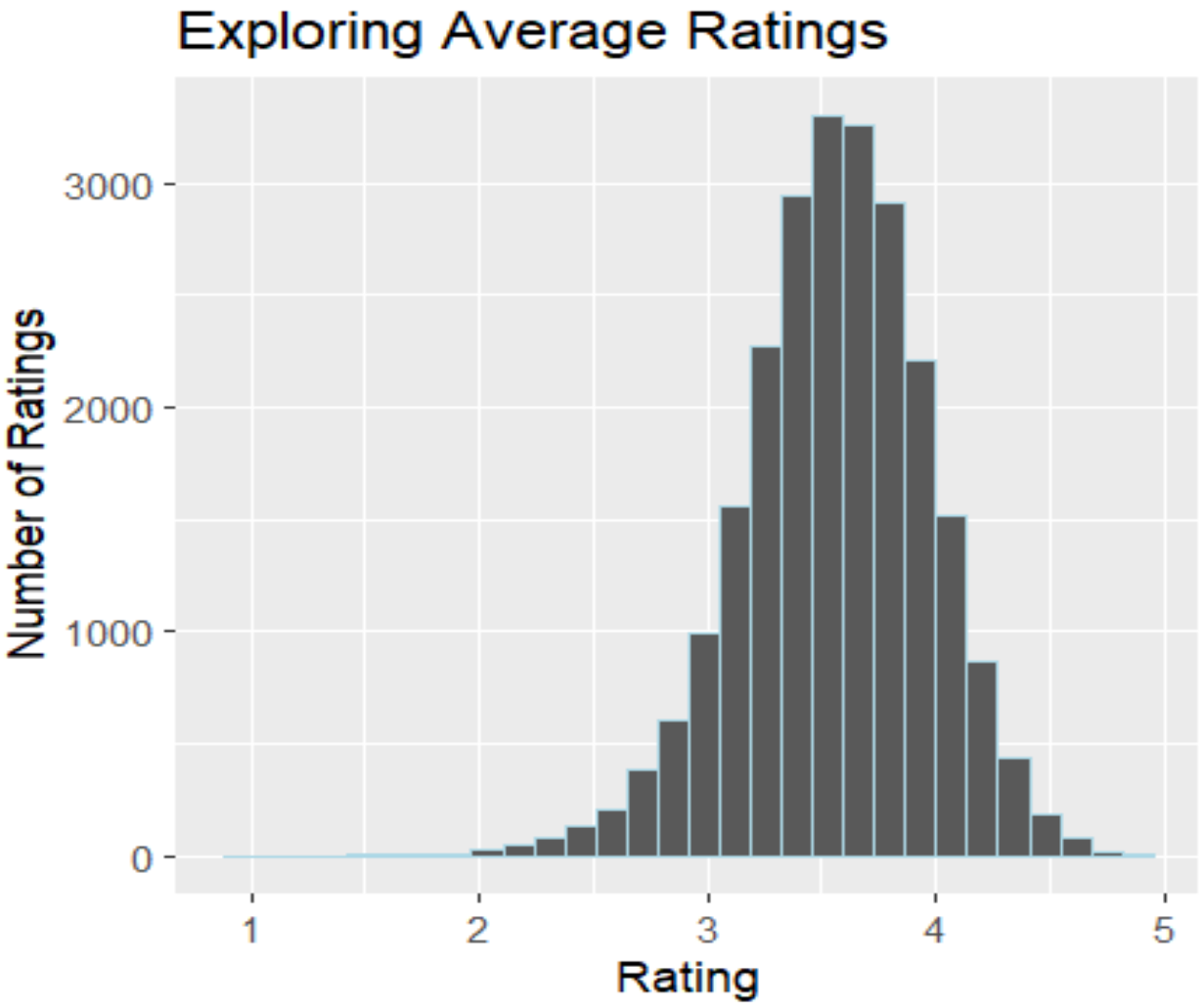
```
# Exploring the number of ratings by movieId
edx %>%
dplyr::count(movieId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "green") +
scale_x_log10() +
ggtitle("Exploring Movies") +
labs(x= "movieId", y= "Number of Ratings")
```



```
# Exploring the number of ratings by userId
edx %>%
dplyr::count(userId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "orange") +
scale_x_log10() +
ggtitle("Exploring Users") +
labs(x= "userId", y= "Number of Ratings")
```



```
# Exploring the average ratings
edx %>%
group_by(userId) %>%
filter(n()>=100) %>%
summarize(b_u = mean(rating)) %>%
ggplot(aes(b_u)) +
geom_histogram(bins = 30, color = "lightblue") +
ggtitle("Exploring Average Ratings") +
labs(x= "Rating", y= "Number of Ratings")
```



2.4- Modeling

In order to judge which algorithm performs better we need to specify a way to quantify what does better or worse mean. In this project I will use RMSE (Residual Mean Squared Error). We can interpret the RMSE similar to a standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1,it means our typical error is larger than one star, which is not good. Below is the formula used for RMSE.

```
RMSE <- function(true_ratings, predicted_ratings){
sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

As the validation dataset cannot be utilized until the models have been constructed, it becomes imperative to employ a test set to assess the performance of the models during development and compare various approaches. To accomplish this, I will divide the edx dataset into a train set and a test set, as demonstrated in the provided code. It is important to note that I have utilized the semi_join() function to ensure that the movies and users in the test set correspond with those in the train set.

```
# Test & Training Sets

test_index <- createDataPartition(y = edx$rating, times = 1,

                                  p = 0.2, list = FALSE)

train_set <- edx[-test_index,]

test_set <- edx[test_index,]


test_set <- test_set %>%

 semi_join(train_set, by = "movieId") %>%

 semi_join(train_set, by = "userId")
```

2.5 - Building Models

Initially, I will begin with a simple "Baseline" approach for rating prediction. Subsequently, I will enhance this basic model by incorporating movie effects, user effects, regularization, and matrix factorization techniques. The iterative process will continue until achieving an RMSE value below 0.80.

2.5.1 Baseline Model

To establish a starting point and serve as our baseline, we can utilize the simplest model of predicting the same rating for all movies and users. In this case, the first algorithm will involve using the average rating of all movies across all users. Subsequently, I will calculate the RMSE for this model to evaluate its performance.

```
# Baseline Model

mu_baseline<- mean(train_set$rating)                                    # Average Rating

baseline_RMSE<- RMSE(test_set$rating, mu_baseline)                              # Baseline RMSE

RMSE_results <- tibble(Method = "Baseline Model", RMSE = baseline_RMSE)                  # Create a table with RMSE's

RMSE_results
```

```
# A tibble: 1 × 2
  Method          RMSE
  <chr>          <dbl>
1 Baseline Model  1.06
```

2.5.2 - Movie Effect Model

Upon investigating the distribution of ratings per movie, as depicted in the Data Exploration section, it becomes evident that there is a significant level of variability in the number of ratings received by different movies. This variation is to be expected since blockbuster movies generally garner more views compared to smaller budget films. In the following model, I will strive to address this effect and incorporate it into the analysis.

```
# Movie Effect Model

mu_least_squares <- mean(train_set$rating)


LSE_rating_movies<- train_set %>% group_by(movieId) %>%

  summarize(b_i = mean(rating - mu_least_squares))


LSE_prediction_movies<- mu_least_squares + test_set %>%

  left_join(LSE_rating_movies, by='movieId') %>% .$b_i


LSE_RMSE_movies <- RMSE(LSE_prediction_movies, test_set$rating)


RMSE_results <- bind_rows(RMSE_results, tibble(Method="Movie Effect Model", RMSE = LSE_RMSE_movies ))

RMSE_results %>% knitr::kable()
```

Method	RMSE
Baseline Model	1.060823
Movie Effect Model	0.944771

2.5.3 - User Movie Effect

Continuing the improvement process, I will now focus on the "users" aspect. As demonstrated in the Data Exploration section, users exhibit varying patterns in how they rate movies. To address this, I will introduce the user effect, denoted as "b_u," into the model. By accounting for this effect, the RMSE metric displays an improvement, reflecting the impact of considering individual user preferences on the overall rating predictions.

```
LSE_rating_users <- train_set %>% left_join(LSE_rating_movies, by='movieId') %>%
 group_by(userId) %>% summarize(b_u = mean(rating - mu_least_squares - b_i))


LSE_prediction_users <- test_set %>% left_join(LSE_rating_movies, by='movieId') %>%
 left_join(LSE_rating_users, by='userId') %>% mutate(pred = mu_least_squares + b_i + b_u) %>% .$pred


LSE_RMSE_users <- RMSE(LSE_prediction_users, test_set$rating)
RMSE_results <- bind_rows(RMSE_results, tibble(Method="Movie + User Effect Model", RMSE = LSE_RMSE_users ))


RMSE_results %>% knitr::kable()
```

Method	RMSE
Baseline Model	1.0608235
Movie Effect Model	0.9447710
Movie + User Effect Model	0.8671241

2.5.4 – Regularization

Examining the table below provides additional insights into the variability of ratings for both movies and users. It becomes evident that movies that have received thousands of ratings should not carry the same weight when calculating the average rating as a movie that has only been rated once. The same principle applies to user ratings as well. Taking this into consideration is crucial to ensure a more accurate representation of the true preferences and opinions of both movies and users.

Thus far, the previous models have not considered the impact of sample size when calculating averages. To address this, I will incorporate Regularization into the model presented below. The introduction of the term lambda allows for penalizing large estimates derived from small sample sizes. To determine the optimal value for lambda, cross-validation will be employed. By tuning lambda using cross-validation, we can obtain the best possible value and calculate the resulting RMSE for evaluation.

```
lambdas <- seq(0, 10, 0.25)
best_lambda <- sapply(lambdas, function(l){
 mu <- mean(train_set$rating)
 b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))
 b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l))
```

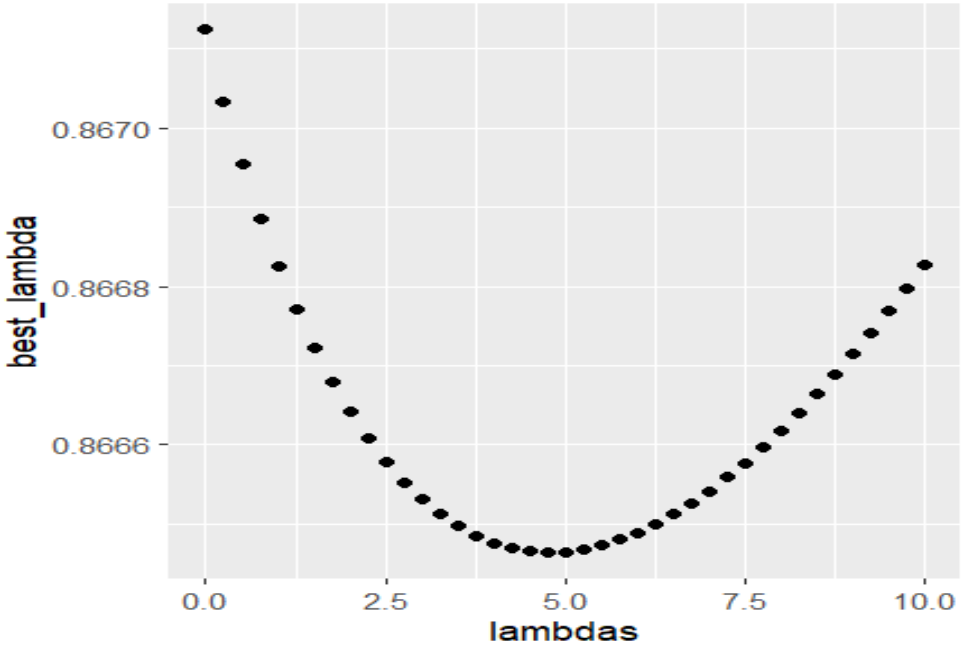
```
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, best_lambda)
lambda <- lambdas[which.min(best_lambda)]
lambda

RMSE_results <- bind_rows(RMSE_results,
                          tibble(Method="Regularized Movie + User Effect Model",
                                 RMSE = min(best_lambda)))

RMSE_results %>% knitr::kable()
```

Method	RMSE
Baseline Model	1.0608235
Movie Effect Model	0.9447710
Movie + User Effect Model	0.8671241
Regularized Movie + User Effect Model	0.8664645



2.5.5 - Matrix factorization

Due to the fact that not all users have rated every movie, the resulting matrix contains numerous missing values, resulting in a sparse matrix. To address this issue, matrix factorization is employed as a solution. The primary objective is to reconstruct the residuals, denoted as "r," utilizing the available data in our dataset. The model presented below utilizes matrix factorization techniques to enhance the RMSE metric by capturing latent factors and improving the accuracy of rating predictions.

```
# Calculating Movie Effect to be used in Matrix Factorization

b_i <- edx %>% group_by(movieId) %>%

  summarize(b_i = sum(rating - mu_least_squares)/(n()+lambda))


# Calculating Movie + User Effect to be used in Matrix Factorization

b_u <- edx %>% left_join(b_i, by="movieId") %>% group_by(userId) %>% summarize(b_u = sum(rating - b_i -
mu_least_squares)/(n()+lambda))


# Adding the Residuals to "edx"

edx_residual <- edx %>%

  left_join(b_i, by = "movieId") %>%

  left_join(b_u, by = "userId") %>%

  mutate(residual = rating - mu_least_squares - b_i - b_u) %>%

  select(userId, movieId, residual)


# Preparing the datasets to be used with recommenderlab packagae

write.table(train_set , file = "trainset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)

write.table(test_set , file = "testset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)


edx_mf <- as.matrix(edx_residual)

validation_mf <- validation %>% select(userId, movieId, rating)

validation_mf<- as.matrix(validation_mf)


write.table(validation_mf , file = "validation.txt" , sep = " " , row.names = FALSE, col.names = FALSE)

train_set_mf<- data_file("trainset.txt")

test_set_mf<- data_file("testset.txt")

validation_mf<- data_file("validation.txt")


# Building Recommender object & tuning it's parameters

r<- Reco()
```

```
tuning_mf <- r$tune(train_set_mf, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                costp_l1 = 0, costq_l1 = 0,
                                nthread = 1, niter = 10))


# Training the Recommender model
r$train(train_set_mf, opts = c(tuning_mf$min, nthread = 1, niter = 20))



# Making prediction on the test_set % calculating RMSE
pred_file <- tempfile()
r$predict(test_set_mf, out_file(pred_file))
predicted_residuals_mf <- scan(pred_file)
mf_RMSE <- RMSE(predicted_residuals_mf, test_set$rating)
RMSE_results <- bind_rows(RMSE_results, tibble(Method="Matrix Factorization Model",  RMSE = mf_RMSE))


RMSE_results %>% knitr::kable()
```

```
|Method                                |      RMSE|
|Matrix Factorization Model            | 0.7914233|
```

3- The Final Results

Throughout the process of constructing the models, the RMSE metric consistently decreased until it eventually reached the target value of less than 0.8. The table provided below summarizes the RMSE results obtained from all the models developed in this project. Notably, the lowest RMSE was achieved through the "Matrix Factorization" approach. However, it is important to note that this RMSE value was calculated based on predictions made using the test_set rather than the validation set.

```
|Method                                |      RMSE|
|:-------------------------------------|---------:|
|Baseline Model                        | 1.0608235|
|Movie Effect Model                    | 0.9447710|
|Movie + User Effect Model             | 0.8671241|
|Regularized Movie + User Effect Model | 0.8664645|
|Matrix Factorization Model            | 0.7914233|
```

```
# Preparing Validation Set

validation_rating <- read.table("validation.txt", header = FALSE, sep = " ")$V3


# Making predictions on the validation set

r$predict(validation_mf, out_file(pred_file))

predicted_residuals_mf_validation <- scan(pred_file)


# Calculating RMSE

mf_RMSE_validation <- RMSE(predicted_residuals_mf_validation, validation_rating)

RMSE_results <- bind_rows(RMSE_results, tibble(Method="Validation using Matrix Factorization",  RMSE = mf_RMSE_validation))


options(digits = 8)

RMSE_results %>% knitr::kable()
```

Method	RMSE
Validation using Matrix Factorization	0.79029979

4-The Conclusion

Method	RMSE
Baseline Model	1.06082353
Movie Effect Model	0.94477104
Movie + User Effect Model	0.86712411
Regularized Movie + User Effect Model	0.86646449
Matrix Factorization Model	0.79142328
Validation using Matrix Factorization	0.79029979

The analysis of the RMSE summary table highlights that the Matrix Factorization model emerged as the top-performing model. It successfully reduced the RMSE from its initial Baseline value of 1.060 to an improved value of 0.790. This represents a significant improvement in prediction accuracy. Future iterations of this project could explore additional aspects, such as exploring correlations between movie genres and users belonging to the same age groups. By incorporating techniques like Principal Component Analysis (PCA) and Singular Value Decomposition (SVD), it is possible to further enhance the prediction of ratings and improve movie recommendations, increasing the likelihood of users enjoying the suggested movies.