BUILDING A MOVIE RECOMMENDATION SYSTEM USING THE MOVIELENS RESEARCH DATA

BUILDING A MOVIE RECOMMENDATION SYSTEM USING THE MOVIELENS RESEARCH DATA

1. INTRODUCTION

In this project, we create a movie recommendation system using the MovieLens dataset. You can obtain the GroupLens Research data from this link. Our main aim is to perform a data analysis and through visualization, find patterns to assist us in building a model that will provide an optimum movie recommendation to users.

The dataset is composed of 10000054 observations with:

- 69878 unique users and
- 10677 movies

The key steps performed include:

- Preparation of the work environment.
- Preparation, exploration and visualizations of the data
- Analysis of the obsertions.
- Calculation of the optimal RMSE based on movieId and userId.

On following the above steps, our data model will reveal that the best predictors used to provide the optimum recommendation system are moveiId and UserId. The RMSE is **0.8431355**.

2. METHODS AND ANALYSIS

2.1 Work Environment Preparation

We are going to use the following libraries:

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

Next we shall download the data

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

2.2 Data Wrangling

We build the desired dataset which we shall use to build our model using the code below:

```
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
```

```
## Error: memory exhausted (limit reached?)
```

```
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
```

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

```
## Error: cannot allocate vector of size 76.3 Mb
```

```
# Validation set will be 10% of MovieLens data
set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
```

```
## Error: cannot allocate vector of size 76.3 Mb
```

```
edx <- movielens[-test_index,]
```

```
## Error: cannot allocate vector of size 38.1 Mb
```

```
temp <- movielens[test_index,]
```

```
validation <- temp %>% semi_join(edx, by = "movieId") %>% semi_join(edx, by = "userId")
```

```
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

```
## Error: cannot allocate vector of size 68.7 Mb
```

```
#Add a year column generated from the timestamp column
dates <- as.Date(as.POSIXct(edx$timestamp, origin="1970-01-01"))
```

```
## Error: cannot allocate vector of size 68.7 Mb
```

```
edx <- edx %>% mutate(year=year(dates), month=month(dates))
```

```
## Error: cannot allocate vector of size 34.3 Mb
```

We shall use the edx dataset we just created onwards to train our model.

2.3 Data Exploration and Visualizations

The edx dataset consists of:
Total number of ratings

```
length(edx$rating)
```

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

Total number of movies

```
n_distinct(edx$movieId)
```

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

Total number of users

```
n_distinct(edx$userId)
```

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

And we can confirm that each user rated a movie using the following code

```
edx %>% filter(is.na(.$ratings))
```

The top 20 most viewed genres are

```
##                                genres  count
## 1                               Drama 733296
## 2                              Comedy 700889
## 3                      Comedy|Romance 365468
## 4                        Comedy|Drama 323637
## 5                Comedy|Drama|Romance 261425
## 6                        Drama|Romance 259355
## 7             Action|Adventure|Sci-Fi 219938
## 8          Action|Adventure|Thriller 149091
## 9                      Drama|Thriller 145373
## 10                        Crime|Drama 137387
## 11                          Drama|War 111029
## 12               Crime|Drama|Thriller 106101
## 13 Action|Adventure|Sci-Fi|Thriller 105144
## 14              Action|Crime|Thriller 102259
## 15                   Action|Drama|War  99183
## 16                    Action|Thriller  96535
## 17             Action|Sci-Fi|Thriller  95280
## 18                             Thriller  94662
## 19                    Horror|Thriller  75000
## 20                       Comedy|Crime  73286
```

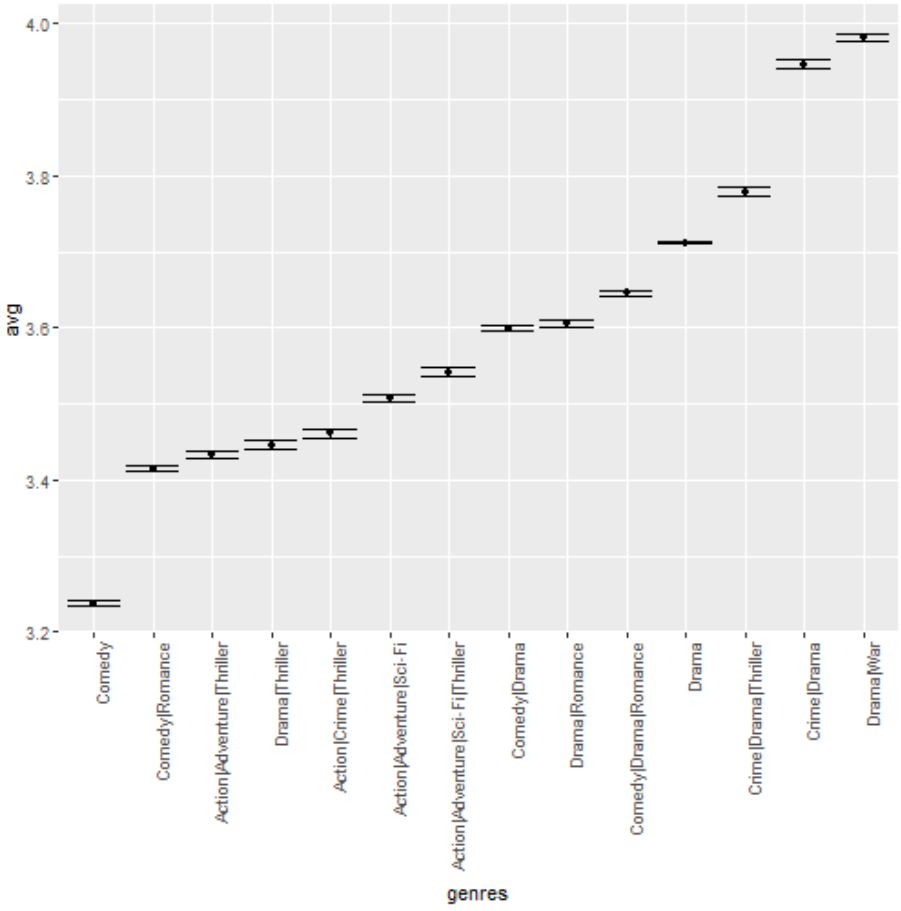
While the top 20 best rated movies are

```
## Error: <text>:1:140: unexpected symbol
## 1: edx %>% group_by(movieId) %>% summarize(title = title[1], count = n()) %>% top_n(20, c
##
```

From the figure below, we can conclude that **1997** had the highest number of ratings

```
## Error: Column `year` must be length 1 (a summary value), not 0
```

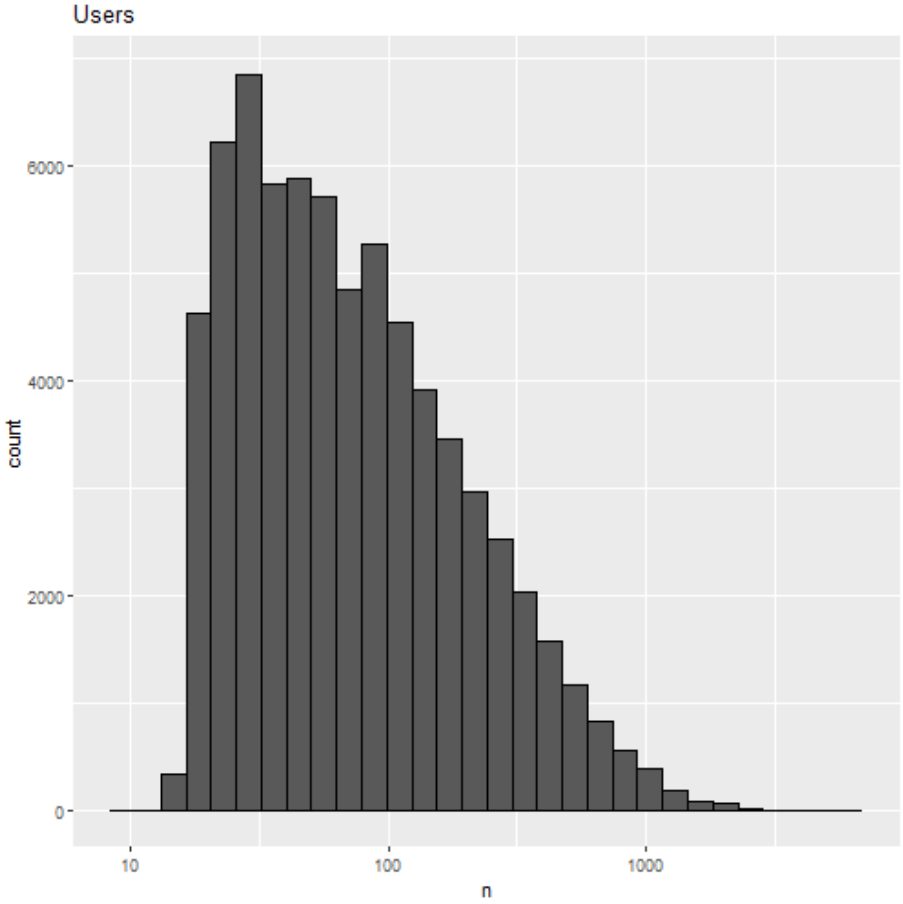
And the genre **Drama/War** with the highest average ratings.



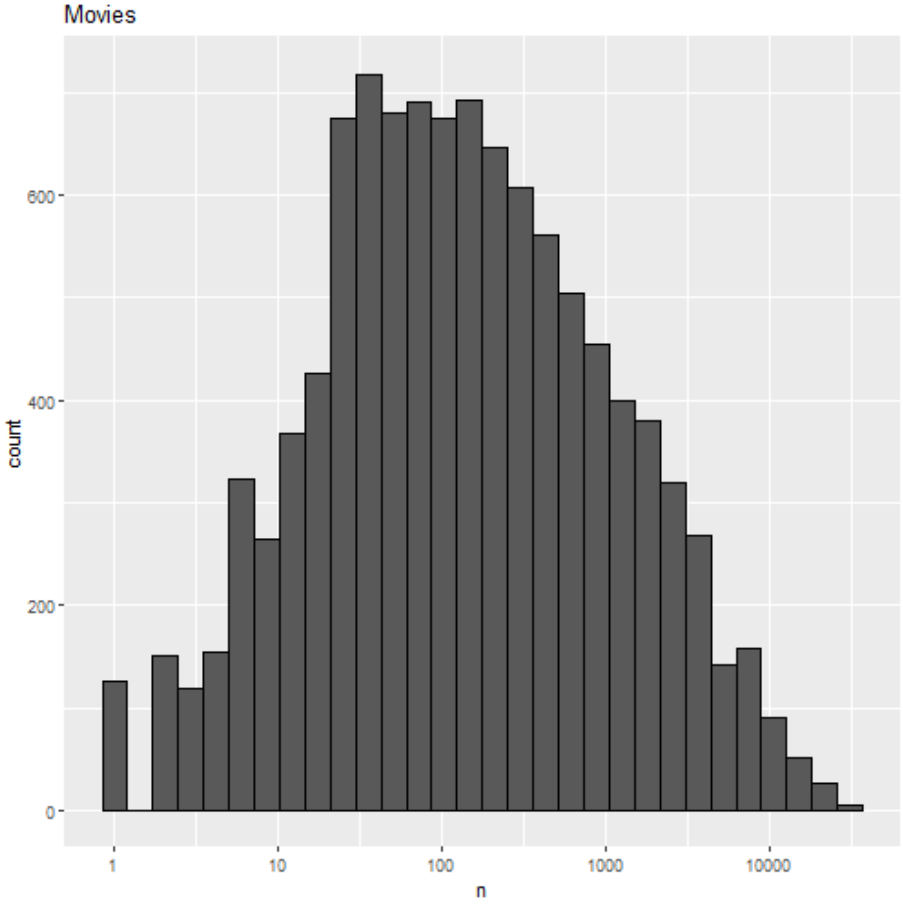
2.4 Data Analysis and Modelling

Let's look at some of the general properties of the data to better understand the challenge.

The first observation is that some users are more active than others at rating movies. Notice that some users have rated over 1,000 movies while others have only rated a handful.



The second thing we notice is that some movies get rated more than others. Here is the distribution.



From the two observations above, we can then prove that there's indeed a movie variability and a user variability. We will use these two predictors to model the data.

To compare different models or to see how well we're doing compared to some baseline, we need to quantify what it means to do well. We need a loss function, in this case the residual mean squared error since we can interpret it as similar to standard deviation. It is the typical error we make when predicting a movie rating. This will therefore be our modelling approach.

We're going to predict the same rating for all movies, regardless of the user and movie. In this case, that's just the average of all the ratings. Using that, we'll get the first RMSE.

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

```
mu_hat <- mean(train_set$rating)
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.059904
```

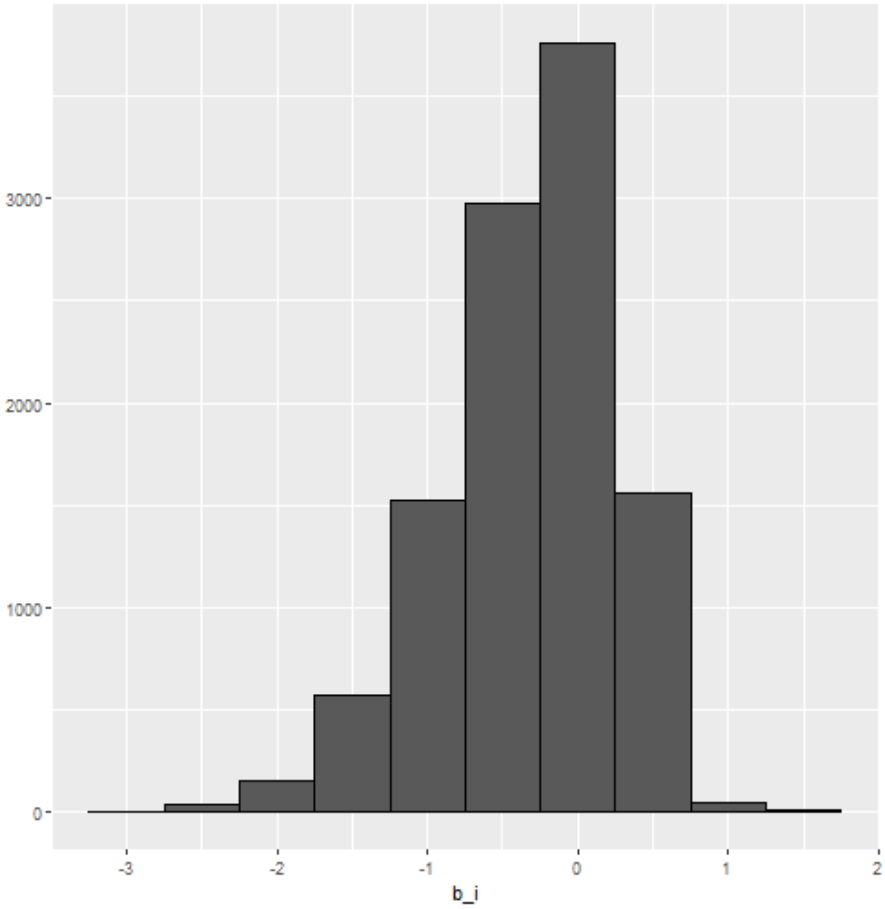
As we go along we will be comparing different approaches, we're going to create a table that's going to store the results that we obtain as we go along.

```
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
```

2.5 LAMBDA and RMSE Calculations

Now let's see how much our prediction improves once we predict using the model that we just fit.

```
mu <- mean(train_set$rating)

movie_avgs <- train_set %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))

movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```

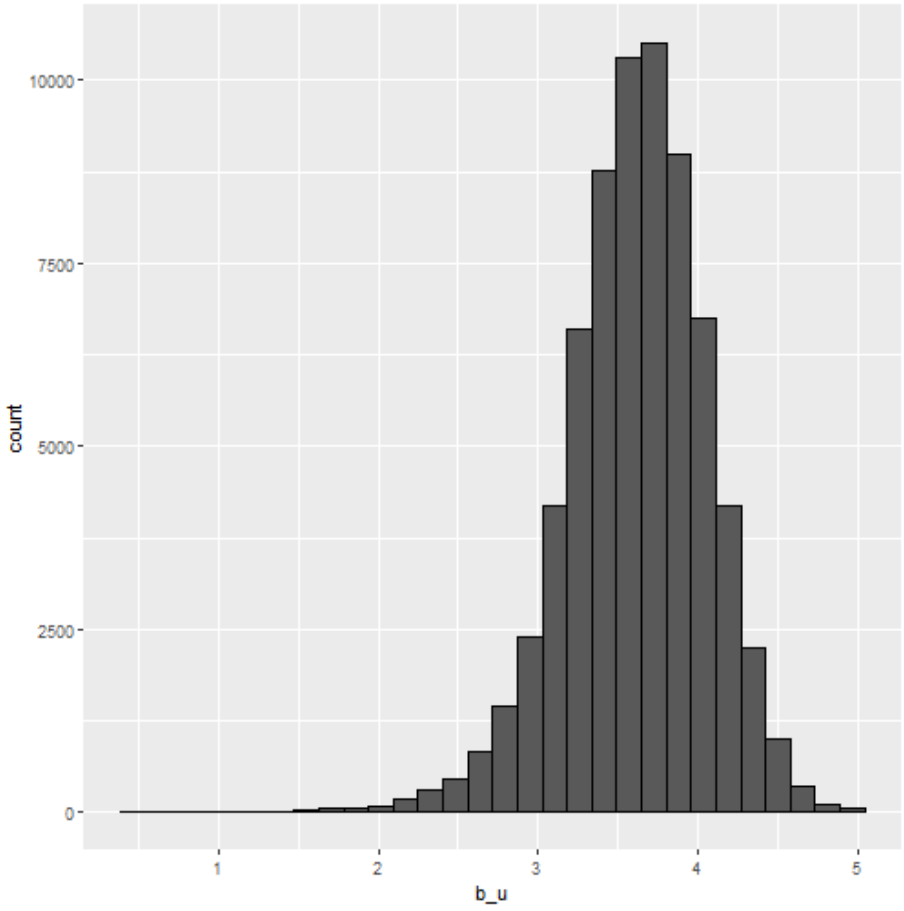



```
predicted_ratings <- mu + test_set %>% left_join(movie_avgs, by='movieId') %>% .$b_i

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results, data_frame(method="Movie Effect Model", RMSE = model
```

Our residual mean squared error did drop a little bit. From **1.0599043** to **0.9437429**

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429

We continue to make it better. This time we factor in user variability. Are different users different in terms of how they rate movies? To explore the data, let's compute the average rating for user, u, for those that have rated over 100 movies.



Note that there is substantial variability across users, as well. Some users are very cranky. And others love every movie they watch, while others are somewhere in the middle. Now we move ahead and implement user variability.

```
user_avgs <- test_set %>% left_join(movie_avgs, by='movieId') %>% group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

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

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
```

```
rmse_results <- bind_rows(rmse_results,data_frame
                          (method="Movie + User Effects Model",  RMSE = model_2_rmse ))
```

3. RESULT

We see that now we obtain a further improvement. Our residual mean squared error dropped down to about **0.84**, hence this is our optimal model.

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

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8431355

4. CONCLUSION

The main objective of the project was to predict movie ratings from a long list of rated movies where we achieved an optimal RMSE of **0.8431355**. Throughout the analysis, some important trends were discovered, some of which were used in modelling the data like user and movie variability. Can we make the model even more better? A challenge worthy of research.