Movie Recommendation Model using Movielens Dataset

Data Science: Capstone Project (HarvardX PH125.9x)

Aditya Prasad Dash

Introduction

The realm of data science is to understand the structure and patterns in the data, to develop data structures to effectively store and visualize it, and to use insights from it to predict the target variable of interest from the values other variables. In this project, I am using the movielens dataset from [1] and the Data Science:Capstone course from Harvard at edX[2] to build a movie recommendation system. The dataset consists of userId, movieId, rating,timestamp,title and genres and I have built a model to predict the rating of a movie based on all of the other features in the dataset.

##Methods

###Reading the data

First the libraries and dataset are loaded using the code provided in the course

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

# Note: this process could take a couple of minutes

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")

library(tidyverse)
library(caret)

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

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)
```

```r
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, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
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)
edx <- rbind(edx, removed)

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

I also load some other libraries to be used in this analysis

```r
if(!require(dplyr)) install.packages("dplyr")
library(dplyr)
if(!require(tidyr)) install.packages("tidyr")
library(tidyr)
if(!require(lubridate)) install.packages("lubridate")
library(lubridate)
if(!require(stringr)) install.packages("stringr")
library(stringr)
if(!require(ggcorrplot)) install.packages("ggcorrplot")
library(ggcorrplot)
if(!require(ggplot2)) install.packages("ggplot2")
library(ggplot2)
if(!require(corrplot)) install.packages("corrplot")
library(corrplot)
if(!require(Hmisc)) install.packages("Hmisc")
library(Hmisc)
```

First, we should understand the structure of the edx dataset to prepare for model building.

Visualizing and preprocessing the data

The str(object) function in R shows the structure of the object and can be used to understand the edx dataset.

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  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 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

The edx dataframe contains 9000055 observations(rows) and 6 variables (columns) with userId, movieId , timestamp as integers, rating as number and title and genres as character strings. The head(dataframe,n) function displays the first n rows of the data frame, and we use it to visualize a sample of the edx dataframe.

```
head(edx,7)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
## 8      1     356      5 838983653            Forrest Gump (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
## 8      Comedy|Drama|Romance|War
```

Using sum(is.na(edx)), we see that there are no na values in the dataset. However, to demonstrate the steps needed before analyzing a general dataset, we can impute the missing values with the expected values calculated from other entries in the dataset using the impute method in Hmisc package. For example, to impute the missing entries of ratings with the median of all other ratings we can use

```
#impute(edx$rating, median)
```

As there are no missing values in the dataset, we do not impute the other columns but this is the general procedure one should follow while handling a new dataset. Also, we can filter out only those rows which do not contain any na values before analyzing the dataset using

```
edx<-edx%>%filter(!is.na(userId) & !is.na(movieId) & !is.na(rating) & !is.na(title) & !is.na(genres))
```

We can find the number of unique user ids and movie ids in the dataset using the unique and length functions

```
length(unique(edx$userId))
```

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

```
length(unique(edx$movieId))
```

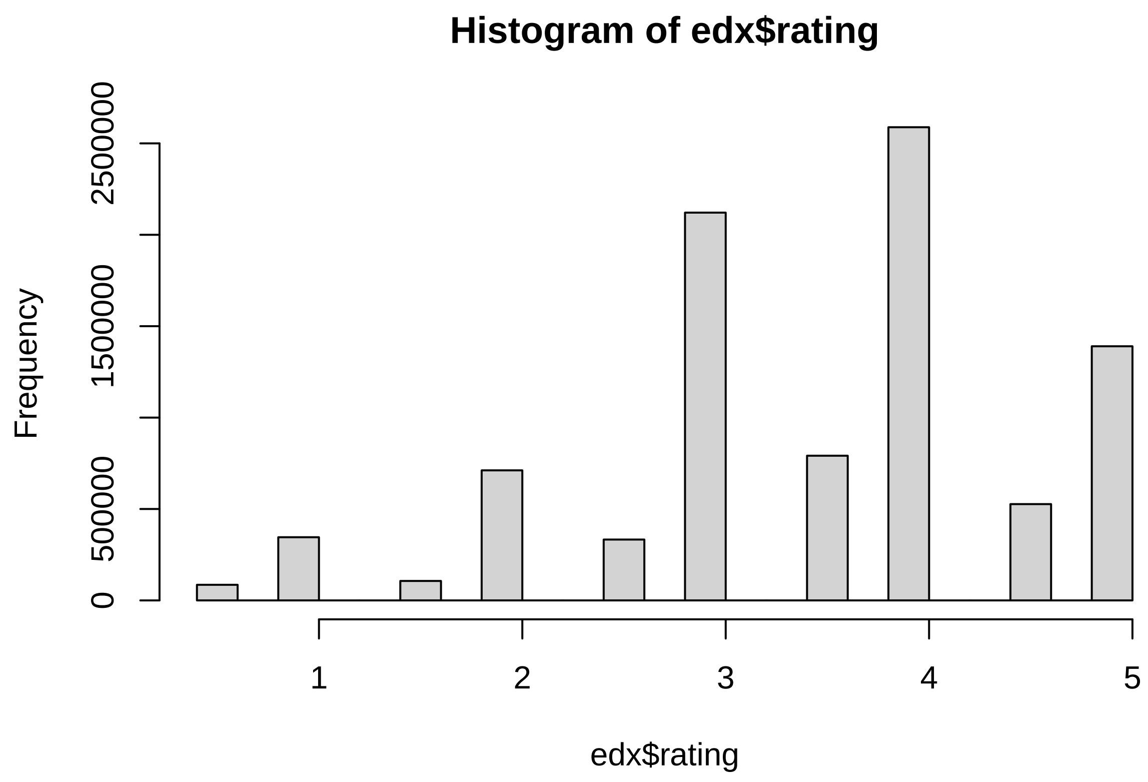
```
## [1] 10677
```

We find the dataset contains 69878 unique user ids and 10677 unique movie ids.We are interested in predicting the ratings from other feature varaibles, therefore we should understand the structure of the ratings column.

```
unique(edx$rating)
```

```
##  [1] 5.0 3.0 2.0 4.0 4.5 3.5 1.0 1.5 2.5 0.5
```

```
hist(edx$rating)
```



We see that there are 10 unique ratings in the dataset ranging from 0.5 to 5 in equal steps. By grouping the dataset by rating, we see that 4 is the most frequent rating with 2588430 occurences while 0.5 is the least frequent rating with 85374 occurences. From the histogram of the ratings, we observe that the integer ratings are usually more frequent than the half integer ratings.

We can notice that the title column contains the movie title with the movie year in paranthesis. As year can be a significant factor in predicting the ratings, we should seaparate the year into a new coliumn. For this we can use the lubridate library in R and use the str_match function to match the fomat of the year in title to the regex \\(\\\\d{4}\\\\)(\\\\)) which matches any string that has 4 digits, followed by) which is the end of a string. However, as we need only the year we can group the output into the year in \\(\\\\\\d{4}\\\\) and the closing bracket. To extract the year, we select the first component of the return of str_match. To show an example,

```
str_match("Boomerang (1992)", "(\\d{4})(\\))$")[2]
```

```
## [1] "1992"
```

returns 1992. Having the method to extract the year, we now mutate a new column "movie_year" to the edx dataset the year as a column .

```
edx<-edx%>%mutate(movie_year=as.numeric(str_match(title, "(\\d{4})(\\))$")[,2]))
```

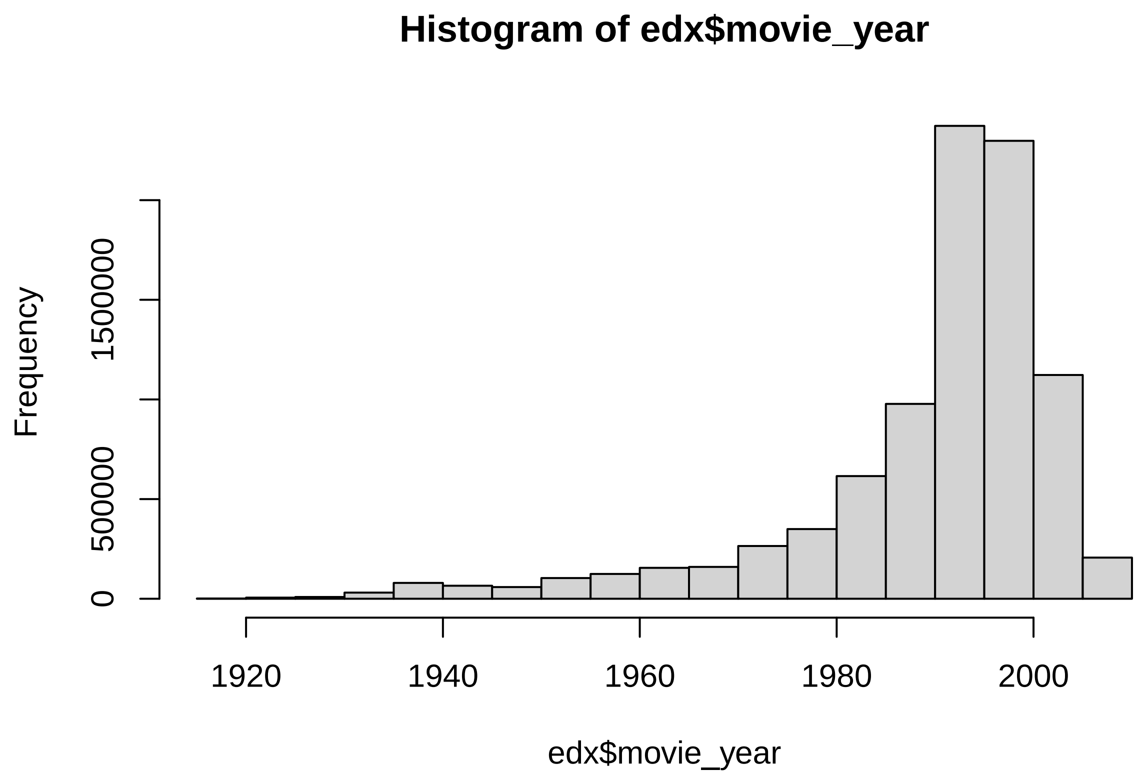
```
min(edx$movie_year)
```

```
## [1] 1915
```

```
max(edx$movie_year)
```

```
## [1] 2008
```

```
hist(edx$movie_year)
```



We see that this dataset contains ratings for movies from years 1915 to 2008 using the min and max functions and group it by movie year to see that the maximum ratings were obtained for movies of 1995 with the number of ratings falling for previous and next years which can be seen from the histogram of the movie years.

```
min(edx$movie_year)
```

```
## [1] 1915
```

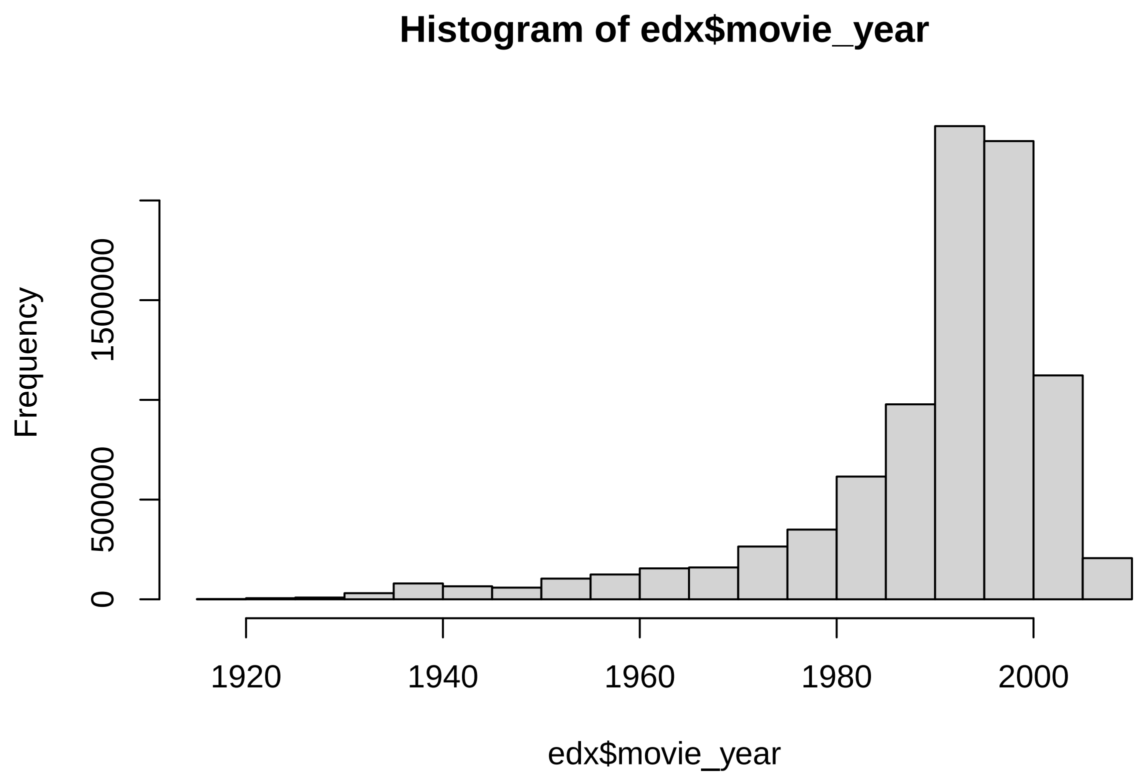
```
max(edx$movie_year)
```

```
## [1] 2008
```

```
edx%>%group_by(movie_year)%>%summarise(n=n())%>%arrange(-n)
```

```
## # A tibble: 94 x 2
##    movie_year      n
##         <dbl>  <int>
##  1       1995 786762
##  2       1994 671376
##  3       1996 593518
##  4       1999 489537
##  5       1993 481184
##  6       1997 429751
##  7       1998 402187
## 8         2000 382763
## 9         2001 305705
## 10        2002 272180
## # i 84 more rows
```

```
hist(edx$movie_year)
```



####Extracting rating month and year from timestamp In this dataset (http://grouplens.org/datasets/movielens/10m/), timestamps represent the seconds after midnight in Coordinated Universal Time (UTC) of January 1, 1970. However, as the ratings could vary depending on the month and season, its useful to extract the month and year from the timestamp into new columns. This can be done by extracting the month and year using the as_datetime function in the lubridate package in r and then mutating them as new columns in the edx dataset.

```
edx<-edx%>%mutate(rating_year=year(as_datetime(timestamp)))
edx<-edx%>%mutate(rating_month=month(as_datetime(timestamp)))
```

From the histogram of the rating year we see that the ratings were given from 1995 to 2009 and there is a spread of number of ratings over the years with a dip around 1998 while the number of ratings in different months is of similar magnitude with slight varaitions eg. a slight dip in 9 corresponding to September and maximum in 11 corresponding to November.

Using `{r warning=FALSE, message=FALSE} head(edx,7)`, we can notice that a movie can be of multiple generes which is mentioned with a separator | in the genres column. As the different genres can influence the ratings and as they do not follow any hirarchy, we should create separate columns for each genre (one hot encoding) which contains 1 if the movie is of that genre and 0 otherwise. To do this we first need to split the genres in the genres column and then unnest it [4]. Then we can use pivot_wider to create the one hot encoding by giving the value_fn as length which takes the value 0 if the genre is present and 0 if it is not present (using values_fill=0).

```
edx<-edx %>% mutate(genreslist = strsplit(genres,"\\|")) %>% unnest(genreslist) %>%
  pivot_wider(names_from = genreslist, values_from = genreslist, values_fn = length,values_fill = 0)
```

After the encoding, we can remove the genres and timestamp columns to avoid duplication of information in the dataset. We can see that the ratings could be dependent on the genre, for example plotting a boxplot for the rating vs Drama show that the rating distributions for drama movies has a lower average rating but is more symmertrical around the median compared to the non-drama movies.

Machine Learning Model

The ratings of a movie are very useful for various reasons. First of all it can be used to suggest new movies to users based on their interest. Secondly it can be used to evaluate different movies based on their overall ratings accross all users. Third it can be used to predict the ratings of a certain type of movie indicating if it would be worthwhile to create the movie. There can be multiple use cases of such a recommendation system which can be generalized to predict the audience liking of the movies based on their features.

We can use Root Mean Squared Error (RMSE) as our metric for evaluating the performance of the model. The RMSE function takes the true values and predicted values arrays and outputs the square root of the mean of the element wise squares of the difference between the arrays. This can be used to compare the ratings output by the model to the actual ratings in the validation and final_holdout_set ratings.

```
RMSE <- function(true_values, predicted_values){
  sqrt(mean((true_values - predicted_values)^2))
}
```

###Building the recommendation system As the dataset is large, it is not feasible to use the models in the caret package for building machine learning models to this dataset. However, we can use a logic similar to that developed in the course to include the effect of each of the variables by a model which sequentially estimates the deviation from the average ratings for different values of a variable and we adding them to the final estimate of the rating. Therefore, we can sequentially build a complex model by staring out with the average of all ratings as the prediction and adding the average deviation from mean produced by the feature variables present in that instance.

First we can split the edx dataset into a training set (90%) and a val (validation) set (10%) so that we can test our models on the validation set and improve it during the training process. We will test our model on the final_holdout_set at the last after training over the full edx set once it performs well on the validation set.

```
### Recommendation System
val_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.1, list = FALSE)
training_set <- edx[-val_index,]
```

We should not include users and movies in the testing set that do not appear in the training set, we can remove it using the semi join function as

```
  val_set <- edx[val_index,] %>%
  semi_join(training_set, by = "movieId") %>%
  semi_join(training_set, by = "userId")
  removed <- anti_join(edx[val_index,], val_set)
  training_set <- rbind(training_set, removed)
  rm(removed)
```

Model 1: Mean The first model is to estimate the ratings as the average of the ratings accross all movies. This is a reasonable estimate as the ratings of the movies should be distributed around the mean and we are not using any other features of the movies for the prediction at this stage. The data frame rmse_results is created to store the RMSE values from the different models.

```
  # Model=mean
mu_hat <- mean(training_set$rating)
mu_hat
```

```
## [1] 3.512509
```

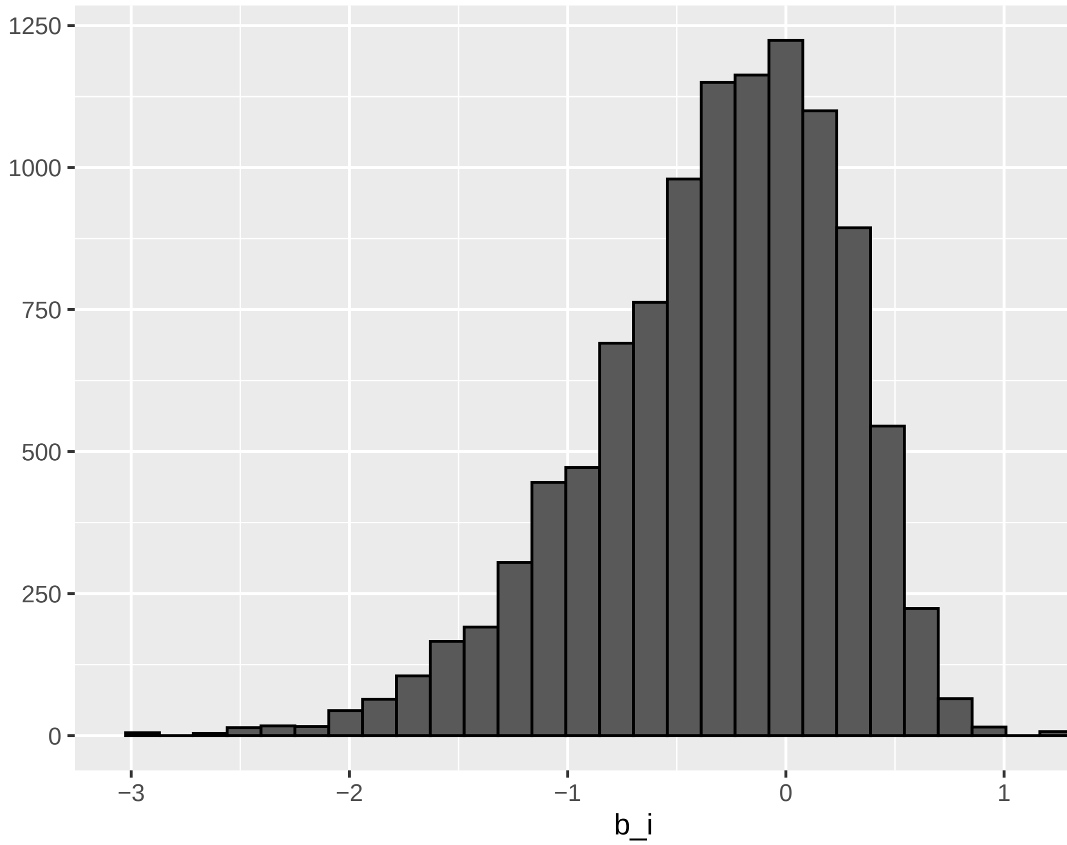
```
naive_rmse <- RMSE(val_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.061135
```

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

We observe that the mean of the ratings is 3.512426 and the RMSE of this model is 1.059963. Next, we can try to include the movie effects by appending the predicted rating of the movie from mu_hat with the average rating of this movie accross all users. The group_by function can categorize the datasets by different movieId's and summarise the mean of the devaitions of their ratings from mu_hat into a column (b_i).

```


#Movie effects
movie_avs <- training_set %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu_hat))
  movie_avs %>% qplot(b_i, geom ="histogram", bins = 30, data = ., color = I("black"))
```



Model 2: Movie Effects

The devaitions (b_i) with the movieIds are stored in a tibble called movie_avs. From its histogram, we see that the the movie ratings show significant deviations from mu_hat, with the peak being at 0, but is negatively skewed. As its width is of the order of 1, it can have a significant influence on the movie rating. The predicted ratings can then by calculated by the sum of mu_hat with the correspoinding movie averages. To do this we can left_join the val_set by movie_avs and then add mu_hat to obtain the predicted rating.

```
predicted_ratings <- mu_hat + val_set %>%
  left_join(movie_avs, by='movieId') %>%
  pull(b_i)
```

We can evaluate the performance of this model with the RMSE between the actual and predicted ratings.

```
model_1_rmse <- RMSE(predicted_ratings, val_set$rating)
model_1_rmse
```

```
## [1] 0.9441568
```

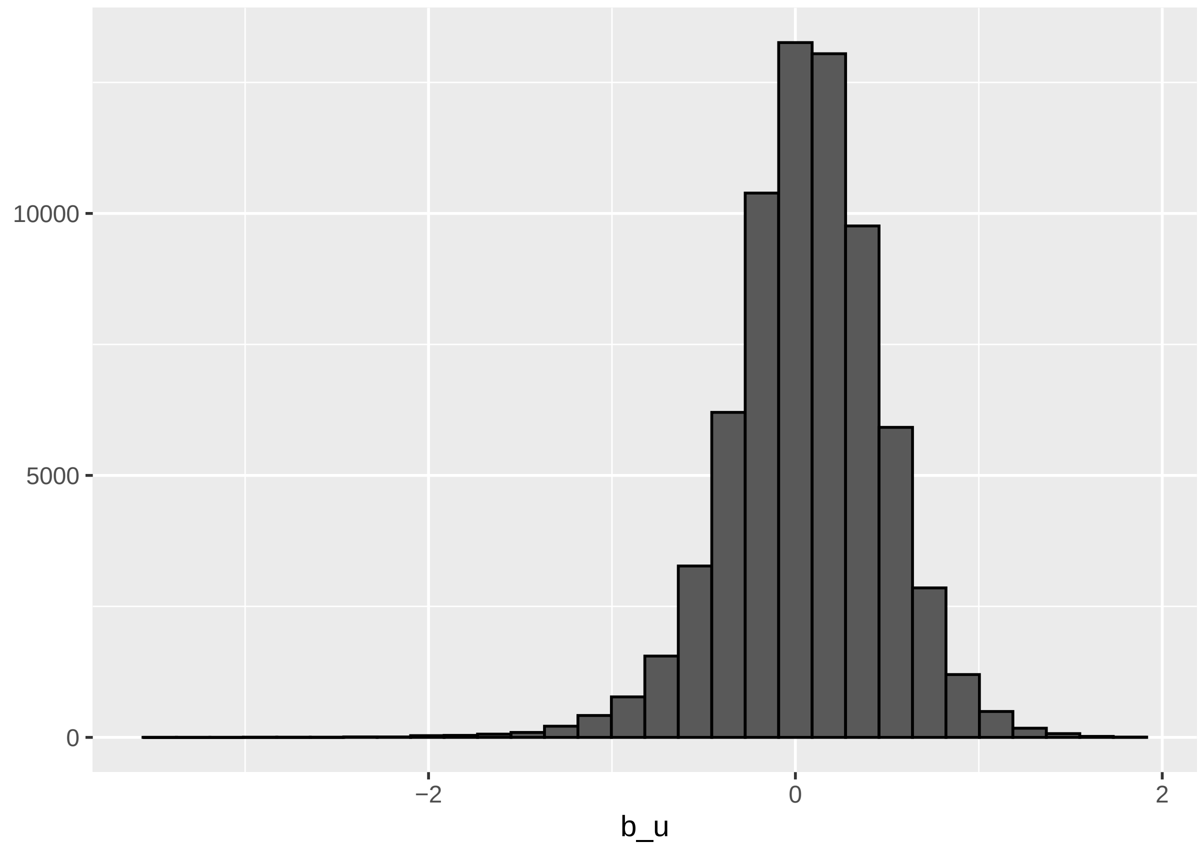
We notice that the RMSE of this model is 0.9434539 which is lower than the intial mean model considering just the average ratings. Next we can consider the user effects. We follow the same steps to obtain the devaitions of the ratings of the movies corresponding to a particular user (b_u) and store it in the user_avs tibble.

```r
#User Effects
user_avs <- training_set %>%
  left_join(movie_avs, by='movieId') %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu_hat - b_i))
str(user_avs)
```

Model 3: User Effects

```
## tibble [69,878 x 2] (S3: tbl_df/tbl/data.frame)
##  $ userId: int [1:69878] 1 2 3 4 5 6 7 8 9 10 ...
##  $ b_u   : num [1:69878] 1.6312 -0.2522 0.345 0.5199 0.0888 ...
```

```r
user_avs %>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black"))
```



The plot of b_u shows that the the devaitions of the ratings of the users from the mean is similar to a Gaussian distribution centered around zero and shows little skewness. Then we predict the ratings from this improved model, obtain the RMSE from the actual ratings and store it in the rmse_results dataframe.

```r
predicted_ratings <- val_set %>%
  left_join(movie_avs, by='movieId') %>%
  left_join(user_avs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)
model_2_rmse <- RMSE(predicted_ratings, val_set$rating)
model_2_rmse
```

```
## [1] 0.8659736
```

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

We see that the RMSE of this model is 0.9434539 which is an improvement from the previous model. Next we notice that as the style and content of the movies can change over the years and different generations of users tend to like movies from a particular range of years, the ratings could depend on the year the movie was released (movie_year). So we improve the model by adding the devaition of the rating from the mean (b_my), after taking into account the movieId and userId effects. We store these deviations into the movie_year_avs tibble and its histogram shows that b_my has a peak near 0.1 with a positive skew.

```


#Movie Year Effects
movie_year_avs <- training_set %>%
  left_join(movie_avs, by='movieId') %>%
  left_join(user_avs, by='userId') %>%
  group_by(movie_year) %>%
  summarise(b_my = mean(rating -mu_hat-b_i-b_u))

predicted_ratings <- val_set %>%
  left_join(movie_avs, by='movieId') %>%
  left_join(user_avs, by='userId') %>%
  left_join(movie_year_avs, by='movie_year') %>%
  mutate(pred = mu_hat+b_i+b_u+b_my) %>%
  pull(pred)
```

Model 4: Movie Year Effects We then add the movie_year_avs information to get an improved version of the predicted ratings. Using the RMSE function to calulate the error of this model over the validation set, we find that the RMSE of this model is 0.864728 which is a significant improvement from the previous version of the model.

```
model_3_rmse <- RMSE(predicted_ratings, val_set$rating)
model_3_rmse
```

```
## [1] 0.8656023
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie+User+Movie Year Effects Model",
                                     RMSE = model_3_rmse ))
rmse_results
```

```
## # A tibble: 3 x 2
##   method                                RMSE
##   <chr>                               <dbl>
## 1 Just the average                    1.06 
## 2 Movie + User Effects Model          0.866
## 3 Movie+User+Movie Year Effects Model 0.866
```

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

method	RMSE
Just the average	1.0611350
Movie + User Effects Model	0.8659736
Movie+User+Movie Year Effects Model	0.8656023

Similar to movie_year, the year at which the movie was rated could have an influence on the rating of the movie. To include this we store the deviations of the rating of a movie from the average (after considering movieId, userId and movie_year effects) and store it into a tibble called rating_year_avs.

```
rating_year_avs <- training_set %>%
  left_join(movie_avs, by='movieId') %>%
  left_join(user_avs, by='userId') %>%
  left_join(movie_year_avs, by='movie_year') %>%
  group_by(rating_year) %>%
  summarise(b_ry = mean(rating -mu_hat-b_i-b_u-b_my))
```

Model 5: Rating Year Effects We notice that the distribution of b_ry is peaked at negative values and has a positive skew. Adding it to the model and calulating the RMSE returns a RMSE of 0.8646441 which is slightly better than the previous model.

```
  predicted_ratings <- val_set %>%
  left_join(movie_avs, by='movieId') %>%
  left_join(user_avs, by='userId') %>%
  left_join(movie_year_avs, by='movie_year') %>%
  left_join(rating_year_avs, by='rating_year') %>%
  mutate(pred = mu_hat+b_i+b_u+b_my+b_ry) %>%
  pull(pred)
model_4_rmse <- RMSE(predicted_ratings, val_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie+User+Movie Year+Rating Year Effects Model",
                                     RMSE = model_4_rmse ))
```

Finally, we can add the information about the genres of the movie to utilize the information to build a better model. As we have many genres, we first calulate the correlations between each of the columns of the training_set with the ratings to understand their influence on the ratings.

```
cor(training_set$rating,training_set%>%select_if(is.numeric))
```

Correlation between the features and ratings

```
##            userId     movieId rating   timestamp movie_year rating_year
## [1,] 0.002272835 -0.00660672      1 -0.03471284 -0.1207008 -0.03533824
##      rating_month     Comedy    Romance      Action      Crime     Thriller
## [1,]   0.01197663 -0.0574247 0.01879349 -0.05418483 0.06008598 -0.002798851
##          Drama      Sci-Fi     Adventure    Children      Fantasy        War
## [1,] 0.132782 -0.04588859 -0.009182749 -0.02639883 -0.003309963 0.06201457
##        Animation    Musical     Western  Mystery  Film-Noir      Horror
## [1,] 0.01941377 0.01080843 0.005913284 0.040212 0.05435465 -0.06595624
##      Documentary        IMAX (no genres listed)
## [1,]  0.02605049 0.007297949        -0.00025684
```

We find that the genre Drama has the highest correlation, so we can first include its effect in the model.We extract the mean deviation of the rating (b_d, after including movieId, userId, movie year and rating year effects) for a movie with and without having drama as genre into a tibble called drama_avs.

```r
#Drama Effects
drama_avs <- training_set %>%
  left_join(movie_avs, by='movieId') %>%
  left_join(user_avs, by='userId') %>%
  left_join(movie_year_avs, by='movie_year') %>%
  left_join(rating_year_avs, by='rating_year') %>%
  group_by(Drama) %>%
  summarise(b_d = mean(rating - mu_hat - b_i - b_u -b_my-b_ry))
drama_avs
```

Model 6: Genres (Drama) Effects

```
## # A tibble: 2 x 2
##   Drama      b_d
##   <int>    <dbl>
## 1     0 -0.00826
## 2     1  0.0108
```

We notice that movies without drama as a genre have a rating -0.00818 lower than the average and movies with Drama have a rating 0.0106 higher than average. Note that they do not need to sum to 0 as the number of movies in each of these catergories is different. We can now include this effect in the model and obtain the RMSE between the predicted and actual ratings in the testing dataset.

```r
predicted_ratings <- val_set %>%
  left_join(movie_avs, by='movieId') %>%
  left_join(user_avs, by='userId') %>%
  left_join(movie_year_avs, by='movie_year') %>%
  left_join(rating_year_avs, by='rating_year') %>%
  left_join(drama_avs, by='Drama') %>%
  mutate(pred = mu_hat+b_i+b_u+b_my+b_ry+b_d) %>%
  pull(pred)
#print(predicted_ratings)
model_5_rmse <- RMSE(predicted_ratings, val_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + MovieYear+RatingYear+Drama Effects Model",
                                     RMSE = model_5_rmse ))
rmse_results
```

```
## # A tibble: 5 x 2
##   method                                              RMSE
##   <chr>                                              <dbl>
## 1 Just the average                                   1.06
## 2 Movie + User Effects Model                         0.866
## 3 Movie+User+Movie Year Effects Model                0.866
## 4 Movie+User+Movie Year+Rating Year Effects Model    0.866
## 5 Movie + User + MovieYear+RatingYear+Drama Effects Model 0.865
```

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

method	RMSE
Just the average	1.0611350
Movie + User Effects Model	0.8659736
Movie+User+Movie Year Effects Model	0.8656023
Movie+User+Movie Year+Rating Year Effects Model	0.8655016
Movie + User + MovieYear+RatingYear+Drama Effects Model	0.8654476

We notice that the RMSE of this model is 0.8646058 which is very slighly lesser than the previous model. However, including the other genres should not decrease the RMSE significantly as they have lesser correlation with the ratings.

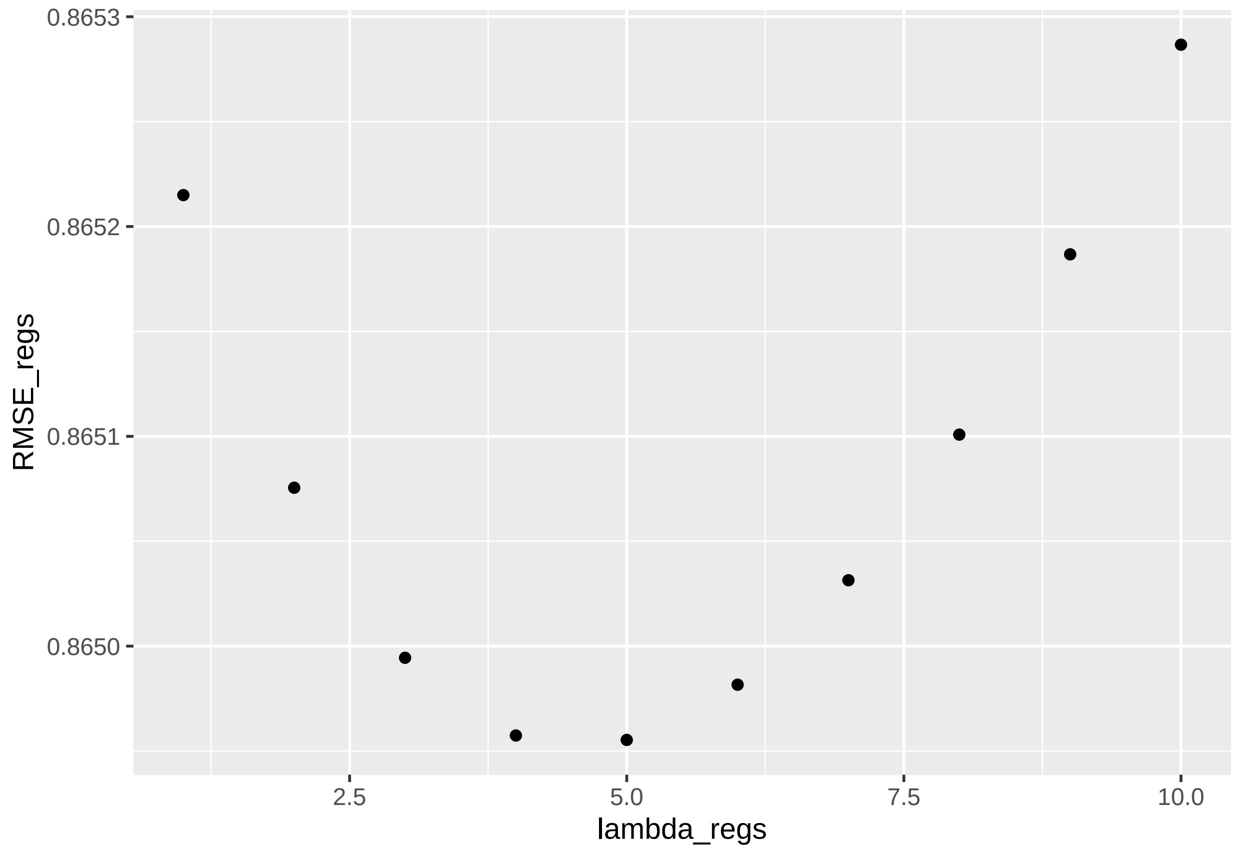
Regularization

In our model, we calculate the predicted ratings based on the mean deviation of rating for a particular category (eg. movieID, userID, rating year) from the overall mean of the dataset. However, when we have small number of observations in a category, it is hard to estimate the effect of this category on the rating as there can be statistical fluctuations. Therefore, regularization is a method where we penalize a large variablity in the effects [3,5]. This amounts to adding a penalty term to the cost function to be minimized during the estimate and is equivalent to a different version of the parameter (b_i) estimation where we divide the sum of the deviations from the average ratings by the (number of obseravtions+ a regularization parameter). As we cannot estimate the best value of the regularization parameter, lambda_reg which minimizes the RMSE, we can try different values of the regularization parameter and choose the one which minimizes the RMSE.

```r
lambda_regs<-seq(1,10,1)

RMSE_regs<-sapply(lambda_regs,function(lambda_reg){
  # Model=mean
  mu_hat <- mean(training_set$rating)

  #Movie effects
  movie_avs_reg <- training_set %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu_hat)/(n()+lambda_reg))

  predicted_ratings <- mu_hat + val_set %>%
  left_join(movie_avs_reg, by='movieId') %>%
  pull(b_i)

  #User Effects
  user_avs_reg <- training_set %>%
  left_join(movie_avs_reg, by='movieId') %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - mu_hat - b_i)/(n()+lambda_reg))

  predicted_ratings <- val_set %>%
  left_join(movie_avs_reg, by='movieId') %>%
  left_join(user_avs_reg, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)

  #Movie Year Effects
  movie_year_avs_reg <- training_set %>%
  left_join(movie_avs_reg, by='movieId') %>%
  left_join(user_avs_reg, by='userId') %>%
  group_by(movie_year) %>%
  summarise(b_my = sum(rating -mu_hat-b_i-b_u)/(n()+lambda_reg))

  predicted_ratings <- val_set %>%
  left_join(movie_avs_reg, by='movieId') %>%
  left_join(user_avs_reg, by='userId') %>%
  left_join(movie_year_avs_reg, by='movie_year') %>%
  mutate(pred = mu_hat+b_i+b_u+b_my) %>%
  pull(pred)

  #Rating Year Effects
  rating_year_avs_reg <- training_set %>%
  left_join(movie_avs_reg, by='movieId') %>%
  left_join(user_avs_reg, by='userId') %>%
  left_join(movie_year_avs_reg, by='movie_year') %>%
  group_by(rating_year) %>%
  summarise(b_ry = sum(rating -mu_hat-b_i-b_u-b_my)/(n()+lambda_reg))

  predicted_ratings <- val_set %>%
  left_join(movie_avs_reg, by='movieId') %>%
  left_join(user_avs_reg, by='userId') %>%
  left_join(movie_year_avs_reg, by='movie_year') %>%
  left_join(rating_year_avs_reg, by='rating_year') %>%
  mutate(pred = mu_hat+b_i+b_u+b_my+b_ry) %>%
  pull(pred)

  #Drama Effects
  drama_avs_reg <- training_set %>%
  left_join(movie_avs_reg, by='movieId') %>%
  left_join(user_avs_reg, by='userId') %>%
  left_join(movie_year_avs_reg, by='movie_year') %>%
  left_join(rating_year_avs_reg, by='rating_year') %>%
  group_by(Drama) %>%
  summarise(b_d = sum(rating - mu_hat - b_i - b_u -b_my-b_ry)/(n()+lambda_reg))

  predicted_ratings <- val_set %>%
  left_join(movie_avs_reg, by='movieId') %>%
  left_join(user_avs_reg, by='userId') %>%
  left_join(movie_year_avs_reg, by='movie_year') %>%
  left_join(rating_year_avs_reg, by='rating_year') %>%
  left_join(drama_avs_reg, by='Drama') %>%
  mutate(pred = mu_hat+b_i+b_u+b_my+b_ry+b_d) %>%
  pull(pred)

  model_reg_rmse <- RMSE(predicted_ratings, val_set$rating)

  return(model_reg_rmse)

})
```

Here, 10 models similar to the one developed before are generated with different regularization parameters (integers from 1 to 10), and the RMSE is calculated for each of these models over the val_set, and stored into a varible called RMSE_regs. Next we can plot the rmses as a function of the regularization parameters

```
qplot(lambda_regs,RMSE_regs)
```



We find that the value of the regularization parameter which minimizes the RMSE is 5 with RMSE 0.8649553. As we had split the initial model into training and validation datasets, we should train the model over the full edx dataset to produce the final trained dataset. This can later be evaluated over the final_holdout_set or over some new unknown data.

```
#Training on full set
lambda_reg=5.0 #value fo which RMSE is least on the validation set adter training on the training set
# Model=mean
```

```r
mu_hat <- mean(edx$rating)

#Movie effects
movie_avs_reg <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu_hat)/(n()+lambda_reg))

#User Effects
user_avs_reg <- edx %>%
  left_join(movie_avs_reg, by='movieId') %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - mu_hat - b_i)/(n()+lambda_reg))


#Movie Year Effects
movie_year_avs_reg <- edx %>%
  left_join(movie_avs_reg, by='movieId') %>%
  left_join(user_avs_reg, by='userId') %>%
  group_by(movie_year) %>%
  summarise(b_my = sum(rating -mu_hat-b_i-b_u)/(n()+lambda_reg))


#Rating Year Effects
rating_year_avs_reg <- edx %>%
  left_join(movie_avs_reg, by='movieId') %>%
  left_join(user_avs_reg, by='userId') %>%
  left_join(movie_year_avs_reg, by='movie_year') %>%
  group_by(rating_year) %>%
  summarise(b_ry = sum(rating -mu_hat-b_i-b_u-b_my)/(n()+lambda_reg))

#Drama Effects
drama_avs_reg <- edx %>%
  left_join(movie_avs_reg, by='movieId') %>%
  left_join(user_avs_reg, by='userId') %>%
  left_join(movie_year_avs_reg, by='movie_year') %>%
  left_join(rating_year_avs_reg, by='rating_year') %>%
  group_by(Drama) %>%
  summarise(b_d = sum(rating - mu_hat - b_i - b_u -b_my-b_ry)/(n()+lambda_reg))
```

##Results Now, our model is fully trained over the edx dataset and is ready to be evaulted over the final holdout set. In this section I use the model trained over the edX set and evaluate the RMSE between the predicted and actual ratings in the final_holdout_set. We do not use the final holdout set for training, however, we should still preprocess the structed of the final_holdout_set similar to the structure of the edx dataset so that the model can be run over it.

```r
#Performing same data preprocessing on final holdout set
final_holdout_test<-final_holdout_test%>%mutate(movie_year=as.numeric(str_match(title, "(\\d{4})(\\))$"
final_holdout_test<-final_holdout_test%>%mutate(rating_year=year(as_datetime(timestamp)))
final_holdout_test<-final_holdout_test%>%mutate(rating_month=month(as_datetime(timestamp)))
final_holdout_test<-final_holdout_test %>% mutate(genreslist = strsplit(genres,"\\|")) %>% unnest(genre
  pivot_wider(names_from = genreslist, values_from = genreslist, values_fn = length,values_fill = 0)
final_holdout_test<-subset(final_holdout_test,select=-c(genres,timestamp))
```

Now, we can run our model on the final_holdout_set to get the predicted ratings

```r
predicted_ratings <- final_holdout_test %>%
  left_join(movie_avs_reg, by='movieId') %>%
  left_join(user_avs_reg, by='userId') %>%
  left_join(movie_year_avs_reg, by='movie_year') %>%
  left_join(rating_year_avs_reg, by='rating_year') %>%
  left_join(drama_avs_reg, by='Drama') %>%
  mutate(pred = mu_hat+b_i+b_u+b_my+b_ry+b_d) %>%
  pull(pred)
```

Now, we calculate the RMSE for the predicted and actual ratings in the final__holdout__set.

```r
holdoutset_rmse <- RMSE(predicted_ratings, final_holdout_test$rating)
holdoutset_rmse
```

```
## [1] 0.8643656
```

We notice that the RMSE between the predicted and actual ratings in the final__holdout__set is 0.8643656 which is below the required tolerance of 0.86490 in the course.

Conclusion

In this project, a movie recommendation model has been built based on the Movielens dataset[1] using the various features of the movies and the users who rated the movies. The first step was to preprocess the data to structure the important information into separate columns and visualze them using histogram and bar plots. As the dataset is very large to efficiently use the machine learning models in caret package in a reasonable timeline, a sequential linear model was built, inspired from the Data Sciene course[2,3] where we predict the ratings as the average rating for all movies with the addition of deviations corresponding to each feature of the movie. To do this 5 features of the movies and users, movieId, userId movie__year, rating__year, and Drama (1 if it is of the Drama genre and 0 otherwise) were included to train the model. The edx dataset was split into a training and validation set and in each step the RMSE was estimated on the validation set. We notice that most of the variablility was captured with these features and the Drama genre, which had the highest correlations with ratings compared to all genres, only slightly reduced the RMSE. Therefore, the concept of regularization, which takes into account the inefficieny of the estimates due to smaller observations and fluctuations, was used to build a regularized model with different regularization parameters. The model was finally trained on the full edx dataset and the RMSE of its predictions with the actual ratings was calculated on the final__holdout__set. This RMSE was found to be 0.8643656, which indicates that this model is able to estimate the ratings to within 0.8643656 from the actual rating. The RMSE could have been further reduced by including the effects of other genres and including polynomial regression by including the features genrerated by including two or more features can can further imporve the model. Having to be able to predict the rating for a movie for any user based on only a handful of features demostrates the use and importance of data to get predictability for a parameter of interest and use them to design the platforms which can save significant resources and improve the user friendliness of the streaming platforms.

References

1. http://grouplens.org/datasets/movielens/10m/
2. Data Science: Capstone, HarvardX PH125.9x provided via the edX platform
3. Introduction to Data Science by Rafael A. Irizarry

4. https://stackoverflow.com/questions/74706268/how-to-create-dummy-variables-in-r-based-on-multiple-values-within-each-cell-in

5. https://rpubs.com/christianakiramckinnon/MovieLens