MovieLens Project - RMSE-Analysis

Mruthyum J Thatipamala

2021-02-25

Project Title: Harvard Online - edX - Data Science - Capstone - MovieLens Project

1. Introduction/Overview/Executive summary:

The goal of movielens project is to simulate and develop a movie recommendation system to give ratings to movies on test data (as if prior ratings are not available) based on ratings available in train data. It is a sort of machine learning challenge project.

GroupLens research labs generated data with over 20 million ratings for over 27,000 movies by more than 138,000 users. A section of this data, about 9 million records, is provided by edx team in the form of training R data set and validation R data set (.rds). These datasets are downloaded and stored into temporary files. From saved files, corresponding dataframes are created using readRDS function.

For data purposes, two .csv files, namely "covid_de.csv", and "demographics_de.csv" are downloaded from Kaggle website.

The same can be found at - https://www.kaggle.com/headsortails/covid19-tracking-germany

The dataset "covid_de.csv" provides daily updated number of reported cases & deaths in Germany at state and county level. COVID-19 cases and deaths are provided on daily basis for a period of 225 days starting from March 23, 2019. The data is further fine tuned are at gender and agegroup levels also.

The dataset "demographics_de.csv" provides a general demographic data of Germany at state level.

An algorithm is developed using the train set to calculate the Residual Mean Square Error (RMSE), similar to standard deviation, is calculated. As a final test to the final algorithm, predict movie ratings in the set (the final hold-out test set) as if they were unknown. RMSE will be used to evaluate how close your predictions are to the true values in the validation set (the final hold-out test set).

Statical analyses conducted, regression models used and predictions algorithms implemented consists starting from linear models to progressing to non-linear models using the train sets and test sets. Results are visualized with appropriate graphs for better understanding of the data trends and outcomes of the analyses. Accuracy is measured for every model in terms of Residual Mean Square Error (RMSE) of actual cases and predicted numbers.

A comparision table is provided towards the end tabulating the accuracy number of each model. Also, some observations made during the analysis and recommendations for future work also documented.

Step 1: Clean up heap memory and plots - Optimizing memory of environment

```r
# Clear environment
rm(list = ls())
```

```r
# Clear console
cat("\014")
```

```r
# Clear plots
if(!is.null(dev.list())) dev.off()
```

```
## null device
##           1
```

```r
#Supressing unharmful warnings
warning = FALSE
#Avoiding sceintific notation of scales in graphs and plots
options(scipen=10000)
```

Step 2: Installing packages and loading libraries

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

library(caret)
library(tidyverse)
library(dplyr)
library(ggplot2)
library(lubridate)
library(rmarkdown)
library(summarytools)
```

Step 3: Reading the data from links provided by edX staff and saving them to dataframes

```r
td = tempdir()
edx = tempfile(tmpdir=td, fileext=".rds")
download.file("https://www.dropbox.com/s/nspymeso8rmmak1/edx.rds?dl=1", edx)
edx = readRDS(edx)

validation = tempfile(tmpdir=td, fileext=".rds")
download.file("https://www.dropbox.com/s/x0s477b0kzxpl6i/validation.rds?dl=1", validation)
validation = readRDS(validation)
unlink(td)
```

Step 4: First Look at data

```r
#Describe data
nrow(edx)
```

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

```r
nrow(validation)
```

```
## [1] 999999
```

```r
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```r
head(edx)
```

```
##   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)
##                           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
```

```r
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 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
```

Column Descriptions:

1. ***Both edx and validation datasets*** There are a total of 138724 observations of 8 variables. This data can be considered as a *Time Series* data as the measurements are made at evenly spaced time, i.e. daily.

- **userId:** Class Integer. Unique to every user. An userID can rate one or more movies (movieId)
- **movieId:** Class Number. Unique to a movie. One movie (movieId) can be rated by one or more users (userId)
- **rating:** Class Number. Can assume a number between 0 and 5 (x.5 is also allowed).
- **timestamp:** Class Number. Timestamps are stored as the number of seconds from reference date.
- **title:** Class Character. Represents movie title.
- **genres:** Class Character. Will be later converted to a categorical variable for efficient handling.

Predictive and Responsive variables:

Predictive variable is **rating** while all other variables (except 'title') are responsive variabls

Step 5: Data Cleaning

Below code finds out the presence of NA values in any of the columns and deletes the corresponding rows.

```
#Checking for null values, if any, and removing the corresponding rows
any(is.na(edx))
```

```
## [1] FALSE
```

```
sum(is.na(edx))
```

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

```
edx <- edx %>% na.omit()

any(is.na(validation))
```

```
## [1] FALSE
```

```
sum(is.na(validation))
```

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

```
validation <- validation %>% na.omit()

#Removing the Title column as it is not used in the analysis to make light weight date
edx<-edx[,-5]
edx <- validation[,-5]
```

Step 6: Data Wrangling and reshaping of the data - aka Data Munging

```
#Converting genres to categorical variable
edx$genres <- factor(edx$genres)
validation$genres <- factor(validation$genres)

#Converting timestamp to weekday and adding a new column named day_of_week for time bias analysis#
edx <- edx %>% mutate(day_of_week = wday(as_datetime(timestamp), label = TRUE))
validation <- validation %>% mutate(day_of_week = wday(as_datetime(timestamp), label = TRUE))
```

Step 7: Data Partitioning

```
# Creating Test data sets for cross validation during training phase for different bias evaluation#
#so that Validation set is used only at the end.#

set.seed(1)
movie_index <- createDataPartition(y = edx$rating, times = 1, p = 0.25, list = FALSE)
movieset <- edx[movie_index,]

# Make sure userId and movieId in Trial set are also in edx set#
movieset <- movieset %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

user_index <- createDataPartition(y = edx$rating, times = 1, p = 0.25, list = FALSE)
userset <- edx[user_index,]

# Make sure userId and movieId in Trial set are also in edx set#
```

```r
userset <- userset %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

genres_index <- createDataPartition(y = edx$rating, times = 1, p = 0.25, list = FALSE)
genresset <- edx[genres_index,]

# Make sure userId, movieId, and genres in Trial set are also in edx set#
genresset <- genresset %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId") %>%
  semi_join(edx, by = "genres")

time_index <- createDataPartition(y = edx$rating, times = 1, p = 0.25, list = FALSE)
timeset <- edx[time_index,]

# Make sure userId, movieId, and genres in Trial set are also in edx set#
timeset <- timeset %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId") %>%
  semi_join(edx, by = "day_of_week")
```

2. Methods/Analysis: Data cleaning: Movie title column is not used anywhere at any level of the algorithm. Hence, it is deleted from both train and test sets. Several partitions of train data are created using caret package functions to test the biases within the training phase. It is observed that some of the ratings are blank, those are filled with average value rather than deleting the rows.

Modelling: First model, the simplest model, is developed based on the assumption that same rating is given by all users to all movies, which is the average of the ratings (mu). Then 'naive rmse' is calculated using the mu and the actual rating given for each movie.

It is an interesting observation that users give ratings differently to different movies. Hence, the basic model is augmented by adding movie bias (b_i). The rmse calculated at this level is less than naive rmse.

From the train data, we can make an observation that ratings given to a movie by different users vary substantially. Hence, a new bais, b_u, is added to the model.

Also, movies belonging to a particular set of genres are given higher ratings in general compared to other genres that attract lower ratings. Also, rating will also vary (though slightly) on the day of the week. Hence, the RMSE model is augmented by adding genre bias (b_g) and time bias (b_t).

It can be easily observed in the data that some movies belonging to a particular get 100+ ratings whereas there are certain movies which get only one rating. Hence, there will be a huge variation in calculating RMSE. The movie that gets more ratings is estimated correctly. To constrain the effect of sizes (the general idea behind regularization), a range of (3.5, 5.5, 0.25) lambda values are introduced as a part of last modelling.

Visualization: To support key concepts and data observations used in analysis, bar plots, scatter plots, and box plots are used as part of data visualization.

Insights: As the model is improved by incorporating more biases, computed RMSE value has reduced considerably. Also, the bigger the size of train and validation data, the more accurate the rating predictions are. Let us look at the code of Model and RSME Algorithm.

Step 8: Defining RMSE function, a function that computes the RMSE for vectors of ratings and their corresponding predictors**

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

Step 9: Calculating the mean (average) of rating of edx data

```
# It is observed that some of the Genres are very popular. Movies belong to these Genres are watched by
# On contrary, some of the Genres are less popular and movies belong to these Genres are watched by few

high_boxplot_genres_rating <- edx %>% filter(genres %in% c("Drama", "Comedy", "Comedy|Romance", "Comedy
str(high_boxplot_genres_rating)
```

```
## 'data.frame':    293935 obs. of  2 variables:
##  $ genres: Factor w/ 773 levels "Action","Action|Adventure",..: 474 695 505 522 640 505 522 640 695
##  $ rating: num  5 3 3 3 3 3 4 4 5 5 ...
```

```
head(high_boxplot_genres_rating)
```

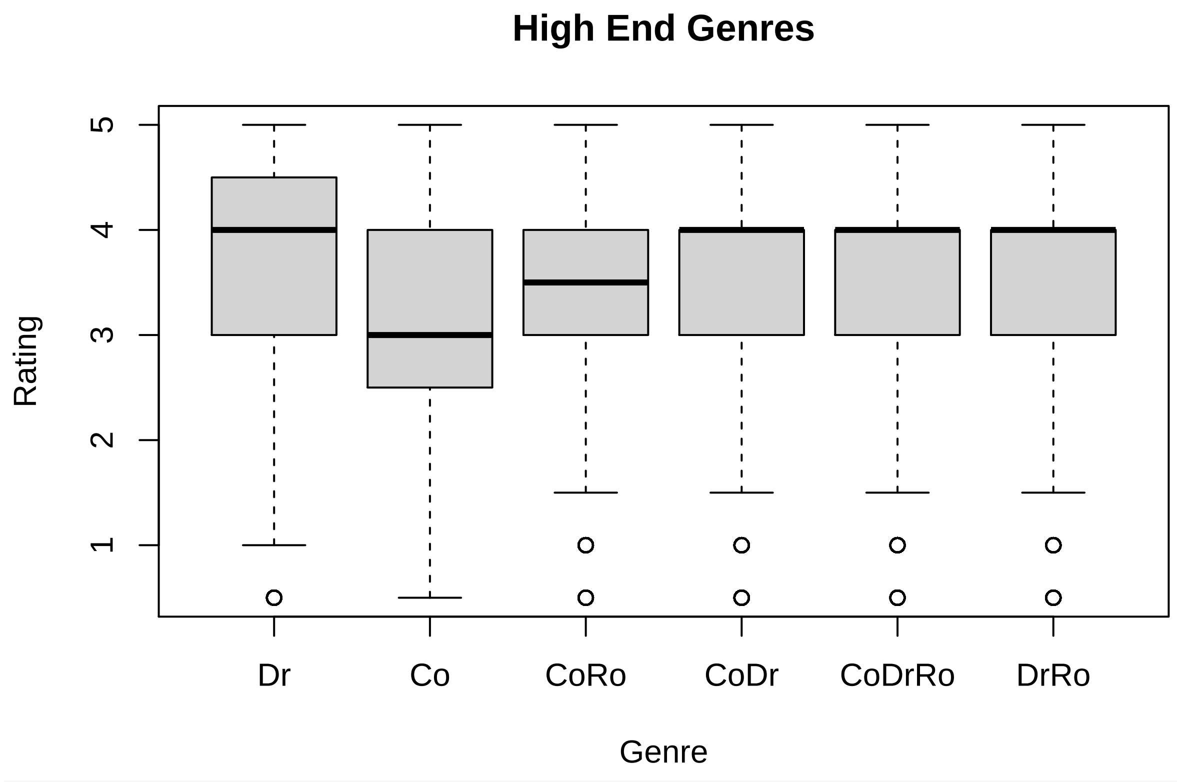
```
##                  genres rating
## 1                Comedy      5
## 2         Drama|Romance      3
## 3          Comedy|Drama      3
## 4 Comedy|Drama|Romance      3
## 5                 Drama      3
## 6          Comedy|Drama      3
```

```
mean(high_boxplot_genres_rating$rating)
```

```
## [1] 3.514129
```

```
high_boxplot_genres_rating$genres <- factor(high_boxplot_genres_rating$genres, levels = c("Drama", "Com

boxplot(rating ~ genres, data = high_boxplot_genres_rating, xlab = "Genre", ylab = "Rating", main = "Hi
```



```r
low_boxplot_genres_rating <- edx %>% filter(genres %in% c("Action|Drama|Horror|Sci-Fi", "Action|Romance
str(low_boxplot_genres_rating)
```

```
## 'data.frame':    7 obs. of  2 variables:
##  $ genres: Factor w/ 773 levels "Action","Action|Adventure",..: 370 313 363 313 193 313 363
##  $ rating: num  3 5 5 4.5 1 4.5 0.5
```

```r
head(low_boxplot_genres_rating)
```

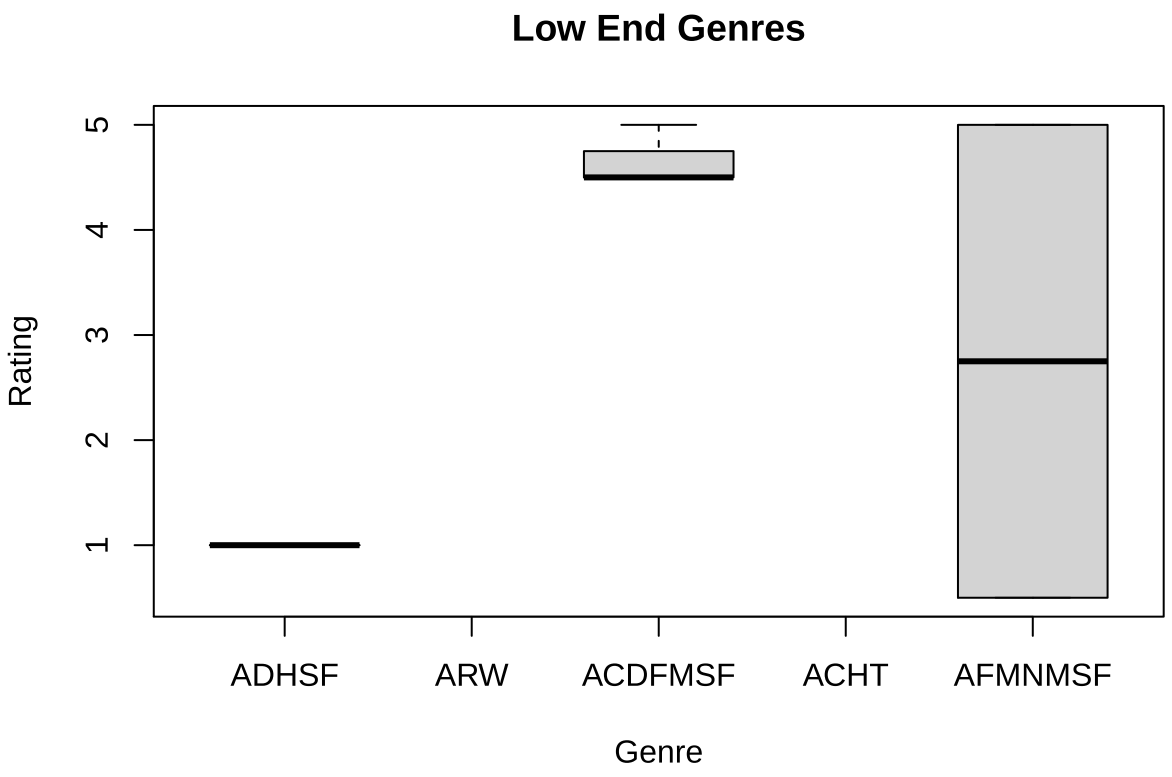
```
##                                           genres rating
## 1               Adventure|Horror|Romance|Sci-Fi    3.0
## 2 Adventure|Comedy|Drama|Fantasy|Mystery|Sci-Fi    5.0
## 3    Adventure|Fantasy|Film-Noir|Mystery|Sci-Fi    5.0
## 4 Adventure|Comedy|Drama|Fantasy|Mystery|Sci-Fi    4.5
## 5                    Action|Drama|Horror|Sci-Fi    1.0
## 6 Adventure|Comedy|Drama|Fantasy|Mystery|Sci-Fi    4.5
```

```r
mean(low_boxplot_genres_rating$rating)
```

```
## [1] 3.357143
```

```r
low_boxplot_genres_rating$genres <- factor(low_boxplot_genres_rating$genres, levels = c("Action|Drama|H

boxplot(rating ~ genres, data = low_boxplot_genres_rating, xlab = "Genre", ylab = "Rating", main = "Low
```



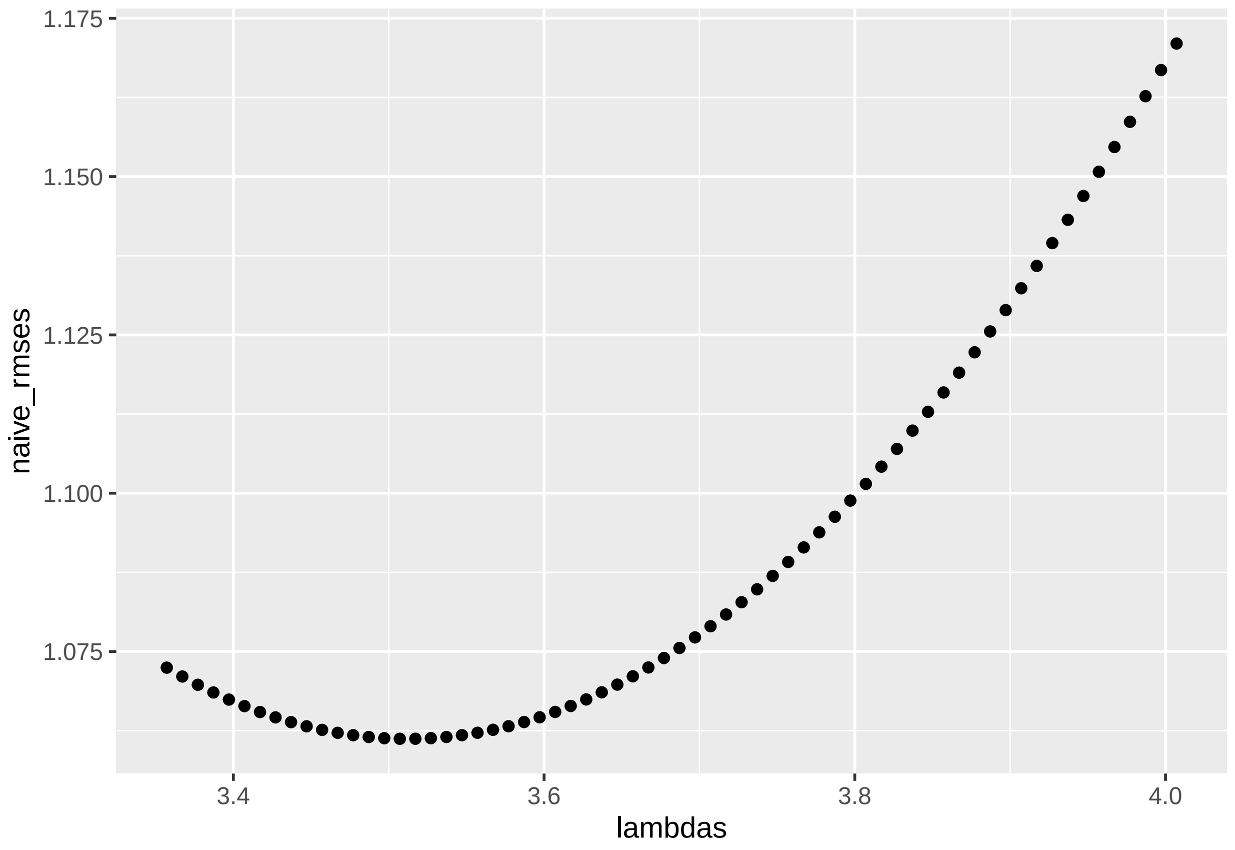
Step 10: Instead of taking the mean of entire dataset, we take the mean of rating of lower end Genres, add increments to reach mean of upper end Genres to calculate the lowest naive rmse and its 'mu'. This a similar to weighted/average distribution analysis

```
lambdas<-seq(mean(low_boxplot_genres_rating$rating),mean(high_boxplot_genres_rating$rating)+0.5, 0.01)

rm(list = c("high_boxplot_genres_rating", "low_boxplot_genres_rating"))

naive_rmses <- sapply(lambdas, function(l){
  return(RMSE(validation$rating,l))
})

qplot(lambdas, naive_rmses)
```



```r
# Clear plots
if(!is.null(dev.list())) dev.off()
```

```
## null device
##           1
```

```r
#(Weighted) average of ratings across the train set
mu <- lambdas[which.min(naive_rmses)]
mu
```

```
## [1] 3.507143
```

```r
#Storing rmse results for various baises in training set
rmse_results <- tibble(method = "naive", RMSE = min(naive_rmses))
rm(list = c("lambdas", "naive_rmses"))
```

Step 11: Movie bias - We observe from the data that different movies are rated differently. Hence, Modeling movie effects by a bias, b_i

```r
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

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

```r
predicted_ratings <- mu + movieset %>% left_join(movie_avgs, by='movieId') %>% pull(b_i)

predicted_ratings <- predicted_ratings %>% replace_na(mu)
```

```
movie_rmse <- RMSE(predicted_ratings, movieset$rating)
movie_rmse
```

```
## [1] 0.9384948
```

```
rmse_results <- rmse_results %>% add_row(method = "movie", RMSE = movie_rmse)
rm(list = c("predicted_ratings", "movieset"))
```

Step 12: User bias - there is substantial variability in rating for a given movie. Different users give different rating for the same movie. Calculating user effects thru a bias, named b__u

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

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

```
predicted_ratings <- userset %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

predicted_ratings <- predicted_ratings %>% replace_na(mu)

user_rmse <- RMSE(predicted_ratings, userset$rating)
user_rmse
```

```
## [1] 0.8254478
```

```
rmse_results <- rmse_results %>% add_row(method = "user", RMSE = user_rmse)
```

Step 13: Genres bias - As calculated and observed in Step 9, movies belonging some genres are watched more and ratings are also higher. On contrary, some genres are watched less and rating are also low. Inclusion of genre bias

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

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

```
predicted_ratings <- genresset %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>% pull(pred)

predicted_ratings <- predicted_ratings %>% replace_na(mu)
```

```
genre_rmse <- RMSE(predicted_ratings, genresset$rating)
genre_rmse
```

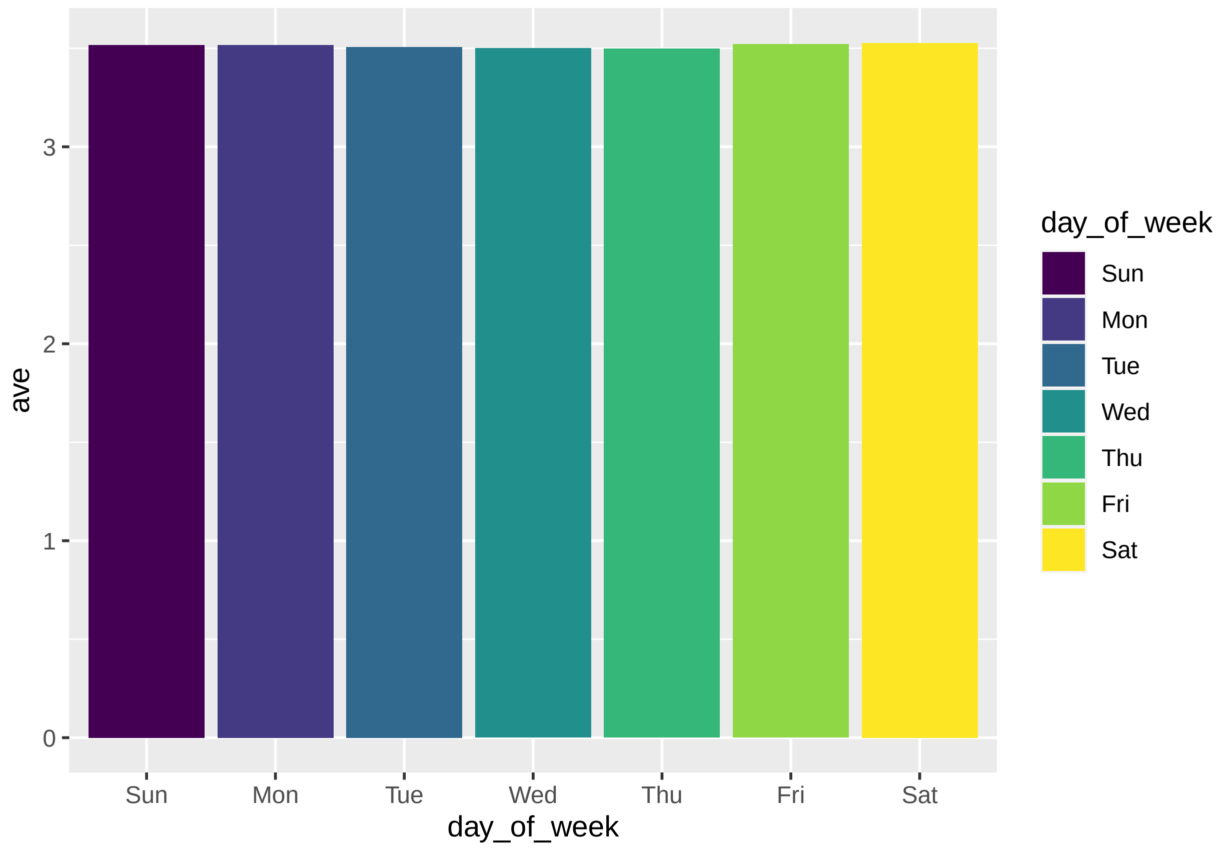
```
## [1] 0.8255379
```

```
rmse_results <- rmse_results %>% add_row(method = "genre", RMSE = genre_rmse)
```

Step 14: Time bias. How the ratings averages are varying on a day of the week can be observed from the bar charts plotted in the below given code. Though small change, the average of ratings given on Tues/Wed/Thursdays are less compared to other days. Accommodating time/day bias.

```
#Barchart
edx %>% group_by(day_of_week) %>% summarize(total = n(), ave = mean(rating)) %>% ggplot(aes(x=day_of_we
```

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



```
time_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  group_by(day_of_week) %>%
  summarize(b_d = mean(rating - mu - b_i - b_u - b_g))
```

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

```r
predicted_ratings <- timeset %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(time_avgs, by='day_of_week') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_d) %>%pull(pred)

predicted_ratings <- predicted_ratings %>% replace_na(mu)

time_rmse <- RMSE(predicted_ratings, timeset$rating)
time_rmse
```

```
## [1] 0.8259151
```

```r
rmse_results <- rmse_results %>% add_row(method = "time", RMSE = time_rmse)
```

FINAL Step: Putting together all biases together and calculating the RMSE against validation set. This analysis also includes a tuning parameter lambda and use it for cross-validation to accommodate regularization for total variability of effect of sizes.

```r
lambdas <- seq(3.5, 5.5, 0.25)
rmses_lambda <- sapply(lambdas, function(l){
  b_i <- edx %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu)/(n()+l), .groups = 'drop')

  b_u <- edx %>% left_join(b_i, by="movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l), .groups = 'drop')

  b_g <- edx %>%
    left_join(b_i, by='movieId') %>% left_join(b_u, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l), .groups = 'drop')

  b_d <- edx %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    group_by(day_of_week) %>%
    summarize(b_d = sum(rating - mu - b_i - b_u - b_g)/(n()+l), .groups = 'drop')

  predicted_ratings <- validation %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    left_join(b_d, by='day_of_week') %>%
    mutate(pred = mu + b_i + b_u + b_g + b_d) %>% pull(pred)

  predicted_ratings <- predicted_ratings %>% replace_na(mu)

  RMSE(predicted_ratings, validation$rating)
  return(RMSE(predicted_ratings, validation$rating))
})
```

______________________________****______________________________

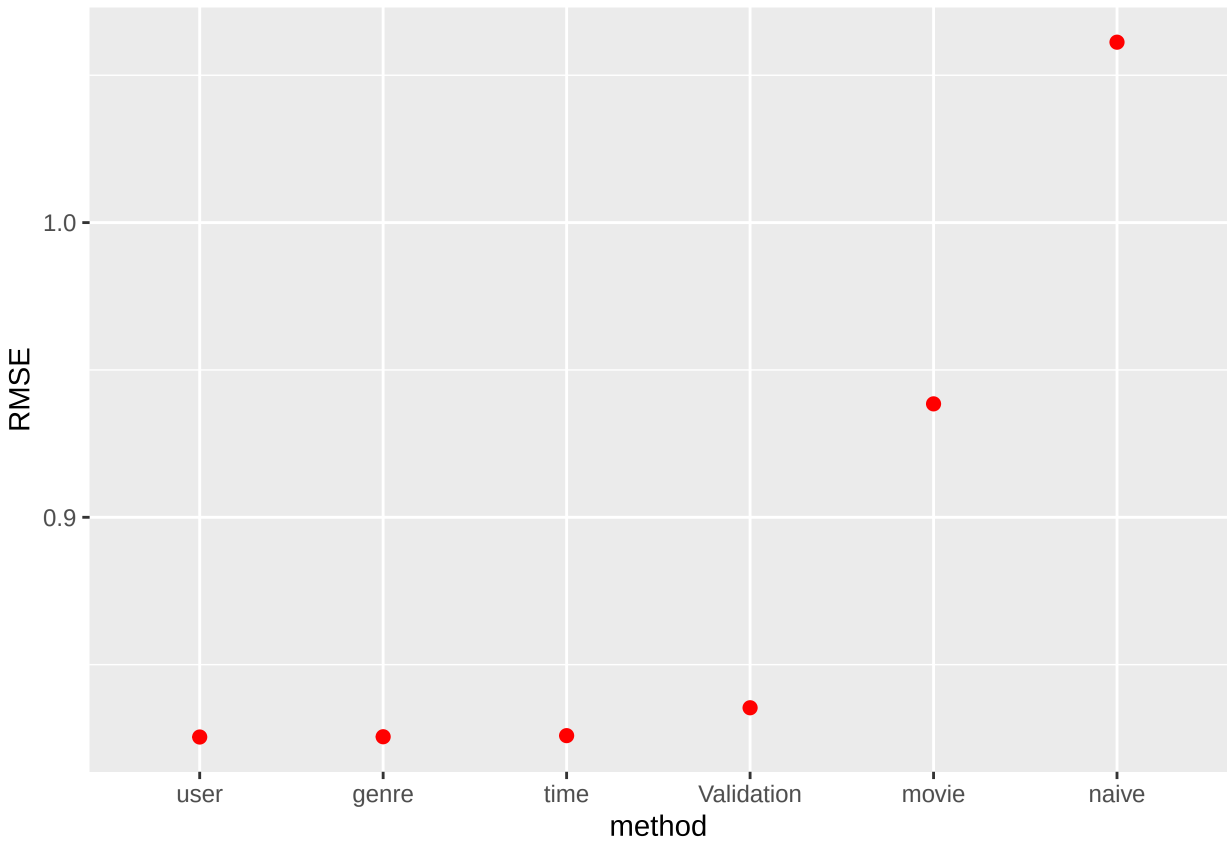
3. Results:

Training Modeling results:

```
#Adding lowest validation RMSE to rmse_results tibble
low_validation_rmse <- min(rmses_lambda)
rmse_results <- rmse_results %>% add_row(method = "Validation", RMSE = low_validation_rmse)

#Plotting the RMSE progression curve
rmse_results$method <- factor(rmse_results$method, levels = rmse_results$method[order(rmse_results$RMSE

rmse_results %>% ggplot(aes(x=method, y=RMSE)) + geom_point(colour = 'red', size = 2)
```



Model Performance:

```
knitr::kable(rmse_results, caption = "RMSE Performance")
```

Table 1: RMSE Performance

method	RMSE
naive	1.0612130
movie	0.9384948
user	0.8254478
genre	0.8255379
time	0.8259151
Validation	0.8353762

——————————————-****———————————————

Conclusion: Scatter plot and table above demonstrates that:

The minimum RMSE value for training data is: **0.825**   The minimum RMSE value for validation data is: **0.835**

Summary and Future work:

Performance of the model improves as more biases are included in the algorithm. It can be improved further by applying advanced techniques such as Matrix factorization, singular value decomposition (SVD), and principal component analysis (PCA).

——————————————-End Report——————————————-