Movilense_Final

Sourav Dutta

04/06/2020

Executive Summery

Basis on users' recommendation on movies, a Movie recomendation system can be build to predict what rating user can give to a particular movie or marketers can recommend similar kind of movie to a specific user.

Different users rate, different movies and a different number of movies.In principle, all other ratings related to movie i and by user u, may be used as predictors. Similarly we can consider similar type of movie like movie i and similar users like movie u for prediction.

Project Objectives: Using Maching Learning, predict user rating for a new movie or user preference for a movie

Approaches: 1) Loading the data and necessary packages 2) Data exploration: Study Data set and variables. Checking if any missing value present 2) Data Wrangling : Added additional variables (Year of rating and age of movie (2020-release date)), Data checking, replaces wrong values 3) Model buliding considering different predictors(checking correlation) 4) Final model selection 5) penealized least squares approach on test_set 6) Using Optimum Lamda for calculating RMSE on Validation data set

```
#################################
# Create Movielense Dataset
#################################

# Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse

## -- Attaching packages ----------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.1     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts -------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```r
# 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")
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")
set.seed(1, sample.kind="Rounding")
```

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

To add two cloumns "year of movie being rated" and "age of movies"

Data preparation done by adding additional columns

```r
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
#Convert Timestamp to year
movielens_timestamp <- mutate(movielens, year_rated = year(as_datetime(timestamp)))


head(movielens_timestamp)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                           genres year_rated
## 1:                Comedy|Romance       1996
## 2:         Action|Crime|Thriller       1996
## 3:                        Comedy       1996
## 4:  Action|Drama|Sci-Fi|Thriller       1996
## 5:       Action|Adventure|Sci-Fi       1996
## 6: Action|Adventure|Drama|Sci-Fi       1996
```

```r
#extracting the First Movie date
premier <- stringi::stri_extract(movielens_timestamp$title, regex = "(\\d{4})", comments = TRUE ) %>% a

#Add the First Movie Date
movielens_with_title_dates <- movielens_timestamp %>% mutate(premier_date = premier)
head(movielens_with_title_dates)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
```

```
## 4:      1     292      5 838983421             Outbreak (1995)
## 5:      1     316      5 838983392             Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                              genres year_rated premier_date
## 1:                   Comedy|Romance       1996         1992
## 2:            Action|Crime|Thriller       1996         1995
## 3:                           Comedy       1996         1994
## 4:  Action|Drama|Sci-Fi|Thriller       1996         1995
## 5:       Action|Adventure|Sci-Fi       1996         1994
## 6: Action|Adventure|Drama|Sci-Fi       1996         1994
```

```
#drop the timestamp
movielens_with_title_dates <- movielens_with_title_dates %>% select(-timestamp)



#looking at the dates - are they correct? Year between 1997 to 2018

movielens_with_title_dates %>% filter(premier_date > 2018) %>% group_by(movieId, title, premier_date) %>
```

```
## `summarise()` regrouping output by 'movieId', 'title' (override with `.groups` argument)
```

```
## # A tibble: 6 x 4
## # Groups:   movieId, title [6]
##   movieId title                                          premier_date     n
##     <dbl> <chr>                                                 <dbl> <int>
## 1     671 Mystery Science Theater 3000: The Movie (1996)         3000  3620
## 2    2308 Detroit 9000 (1973)                                    9000    24
## 3    4159 3000 Miles to Graceland (2001)                         3000   788
## 4    5310 Transylvania 6-5000 (1985)                             5000   218
## 5    8864 Mr. 3000 (2004)                                        3000   163
## 6   27266 2046 (2004)                                            2046   472
```

```
movielens_with_title_dates %>% filter(premier_date < 1900) %>% group_by(movieId, title, premier_date) %>
```

```
## `summarise()` regrouping output by 'movieId', 'title' (override with `.groups` argument)
```

```
## # A tibble: 8 x 4
## # Groups:   movieId, title [8]
##   movieId title                                               premier_date     n
##     <dbl> <chr>                                                      <dbl> <int>
## 1    1422 Murder at 1600 (1997)                                       1600  1742
## 2    4311 Bloody Angels (1732 Høtten: Marerittet Har et Post~         1732     9
## 3    5472 1776 (1972)                                                 1776   205
## 4    6290 House of 1000 Corpses (2003)                                1000   406
## 5    6645 THX 1138 (1971)                                             1138   525
## 6    8198 1000 Eyes of Dr. Mabuse, The (Tausend Augen des Dr~         1000    30
## 7    8905 1492: Conquest of Paradise (1992)                           1492   152
## 8   53953 1408 (2007)                                                 1408   520
```

```r
#Fix the incorrect dates
movielens_with_title_dates[movielens_with_title_dates$movieId == "27266", "premier_date"] <- 2004
movielens_with_title_dates[movielens_with_title_dates$movieId == "671", "premier_date"] <- 1996
movielens_with_title_dates[movielens_with_title_dates$movieId == "2308", "premier_date"] <- 1973
movielens_with_title_dates[movielens_with_title_dates$movieId == "4159", "premier_date"] <- 2001
movielens_with_title_dates[movielens_with_title_dates$movieId == "5310", "premier_date"] <- 1985
movielens_with_title_dates[movielens_with_title_dates$movieId == "8864", "premier_date"] <- 2004
movielens_with_title_dates[movielens_with_title_dates$movieId == "1422", "premier_date"] <- 1997
movielens_with_title_dates[movielens_with_title_dates$movieId == "4311", "premier_date"] <- 1998
movielens_with_title_dates[movielens_with_title_dates$movieId == "5472", "premier_date"] <- 1972
movielens_with_title_dates[movielens_with_title_dates$movieId == "6290", "premier_date"] <- 2003
movielens_with_title_dates[movielens_with_title_dates$movieId == "6645", "premier_date"] <- 1971
movielens_with_title_dates[movielens_with_title_dates$movieId == "8198", "premier_date"] <- 1960
movielens_with_title_dates[movielens_with_title_dates$movieId == "8905", "premier_date"] <- 1992
movielens_with_title_dates[movielens_with_title_dates$movieId == "53953", "premier_date"] <- 2007

# Cross Checking

movielens_with_title_dates %>% filter(premier_date > 2018) %>% group_by(movieId, title, premier_date) %
```

```
## `summarise()` regrouping output by 'movieId', 'title' (override with `.groups` argument)

## # A tibble: 0 x 4
## # Groups:   movieId, title [0]
## # ... with 4 variables: movieId <dbl>, title <chr>, premier_date <dbl>, n <int>
```

```r
movielens_with_title_dates %>% filter(premier_date < 1900) %>% group_by(movieId, title, premier_date) %
```

```
## `summarise()` regrouping output by 'movieId', 'title' (override with `.groups` argument)

## # A tibble: 0 x 4
## # Groups:   movieId, title [0]
## # ... with 4 variables: movieId <dbl>, title <chr>, premier_date <dbl>, n <int>
```

```r
#Calculate the age of the movie

#Calculate the age of a movie
movielens_with_title_dates <- movielens_with_title_dates %>% mutate(age_of_movie = 2020 - premier_date,
head(movielens_with_title_dates)
```

```
##    userId movieId rating                         title
## 1:      1     122      5              Boomerang (1992)
## 2:      1     185      5               Net, The (1995)
## 3:      1     231      5          Dumb & Dumber (1994)
## 4:      1     292      5               Outbreak (1995)
## 5:      1     316      5               Stargate (1994)
## 6:      1     329      5 Star Trek: Generations (1994)
##                           genres year_rated premier_date age_of_movie
## 1:                Comedy|Romance       1996         1992           28
## 2:         Action|Crime|Thriller       1996         1995           25
## 3:                        Comedy       1996         1994           26
```

```
## 4:  Action|Drama|Sci-Fi|Thriller       1996         1995            25
## 5:        Action|Adventure|Sci-Fi       1996         1994            26
## 6: Action|Adventure|Drama|Sci-Fi       1996         1994            26
##    rating_date_range
## 1:                 4
## 2:                 1
## 3:                 2
## 4:                 1
## 5:                 2
## 6:                 2
```

```
#year the movie was rated
year_avgs <- movielens_with_title_dates%>% group_by(year_rated) %>% summarize(avg_rating_by_year = mean
```

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

```
#age of movie
age_avgs <- movielens_with_title_dates %>% group_by(age_of_movie) %>% summarize(avg_rating_by_age = mean
```

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

"Validation Dataset" preparation (10% of data)

```
set.seed(1, sample.kind="Rounding")
```

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

```
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens_with_title_dates$rating, times = 1, p = 0.1, list = FALS
edx <- movielens_with_title_dates[-test_index,]
temp <- movielens_with_title_dates[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)
```

```
## Joining, by = c("userId", "movieId", "rating", "title", "genres", "year_rated", "premier_date", "age_
```

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

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

Method/Analysis section

1) Data Validation

Now we will exam edx and validation data set

```
# number of rows and columns
dim(edx)
```

```
## [1] 9000055       9
```

```
dim(validation)
```

```
## [1] 999999      9
```

```
# checking if there is any 0 in Ratings
edx %>% filter(rating == 0) %>% tally()
```

```
##   n
## 1 0
```

```
# How many differnt movies in edx
n_distinct(edx$movieId)
```

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

```
# How many differnt users in edx
n_distinct(edx$userId)
```

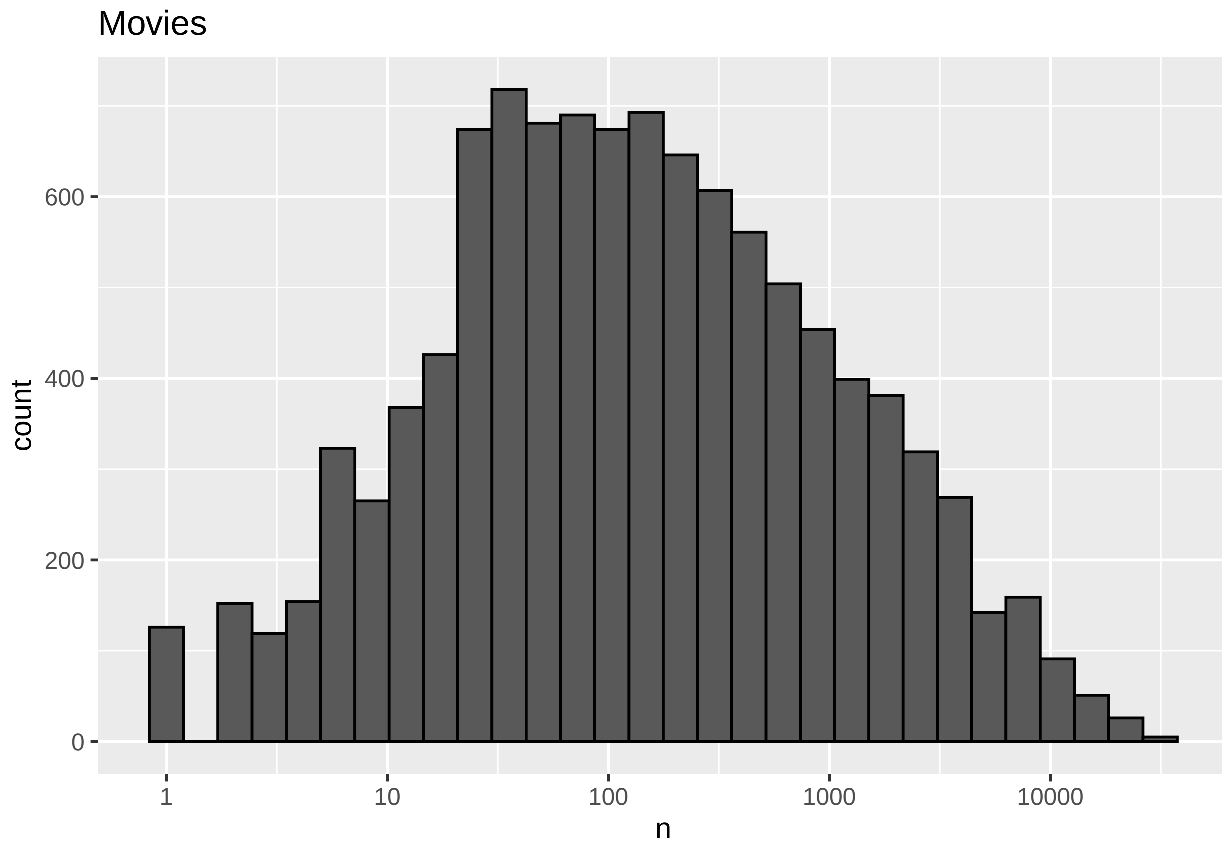
```
## [1] 69878
```

```
#any missing value in edx
edx[!complete.cases(edx),]
```

```
## Empty data.table (0 rows and 9 cols): userId,movieId,rating,title,genres,year_rated...
```

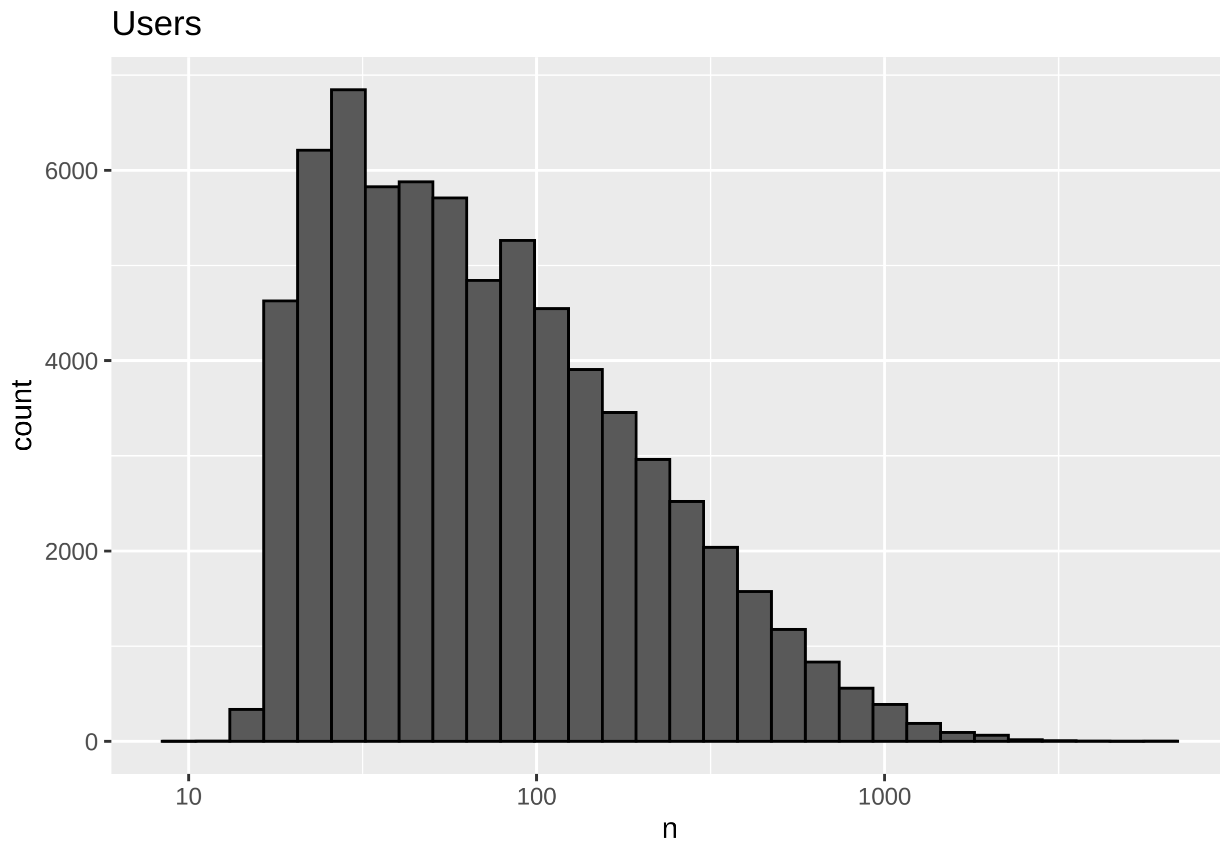
Study "Movie rating" & "User rating" behaviour

```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```



Observations: Some movies rated higher than others

```
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```



Observation: Some users are more active

Test Data Creation

```
# Test set creation
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,
                                  list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

Modeling approaches

First Model: Same rating for all movies

```
# First Model(movie rating)
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512574
```

```r
## predict unknown ratings with mu_hat
rmse_movie_rating<- RMSE(test_set$rating, mu_hat)

# Creating a result table of RMSE
rmse_results <- tibble(method = "Same rating for all movies", RMSE = rmse_movie_rating)
rmse_results
```

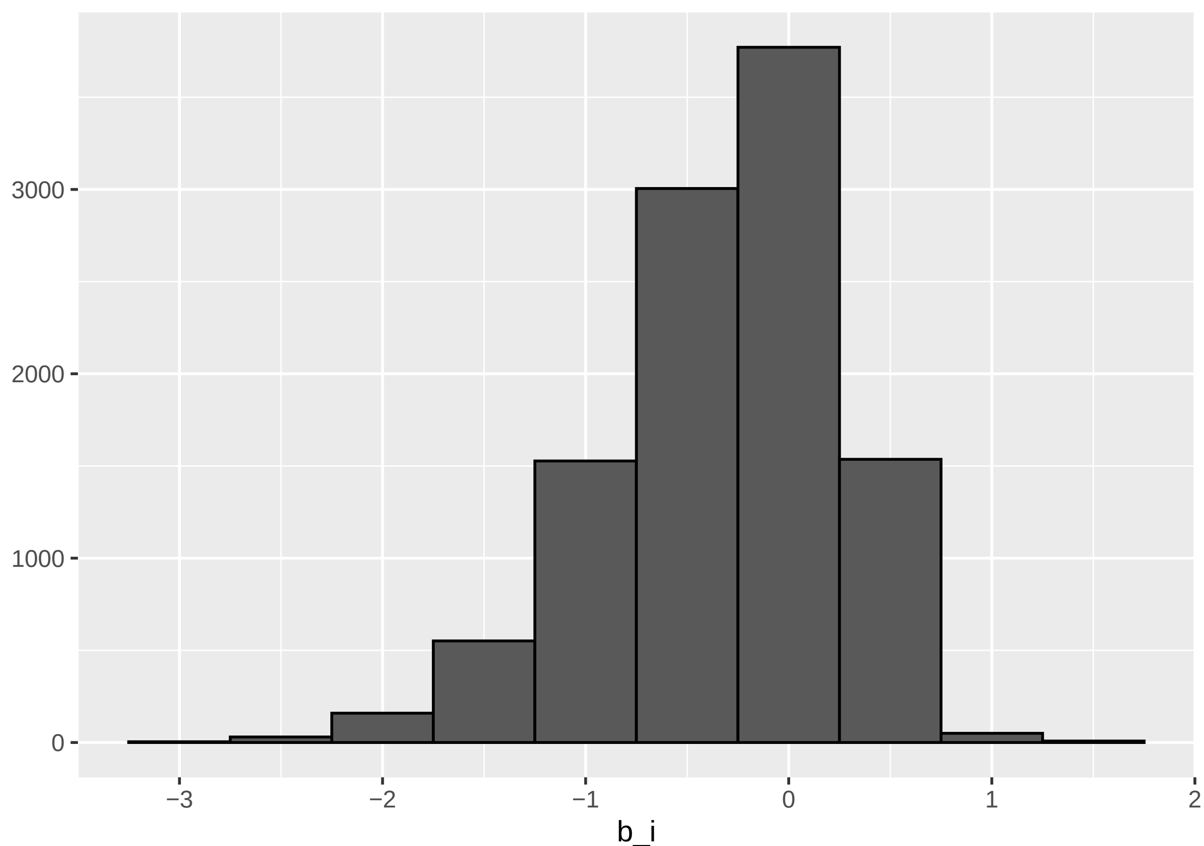
```
## # A tibble: 1 x 2
##   method                      RMSE
##   <chr>                      <dbl>
## 1 Same rating for all movies  1.06
```

2nd Model: Different movies have rated differently

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

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

```r
# variance of rating by movie
qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



```r
## predict unknown ratings with b_i
#Root Mean Square Error Loss Function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2,na.rm=TRUE))
}

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

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

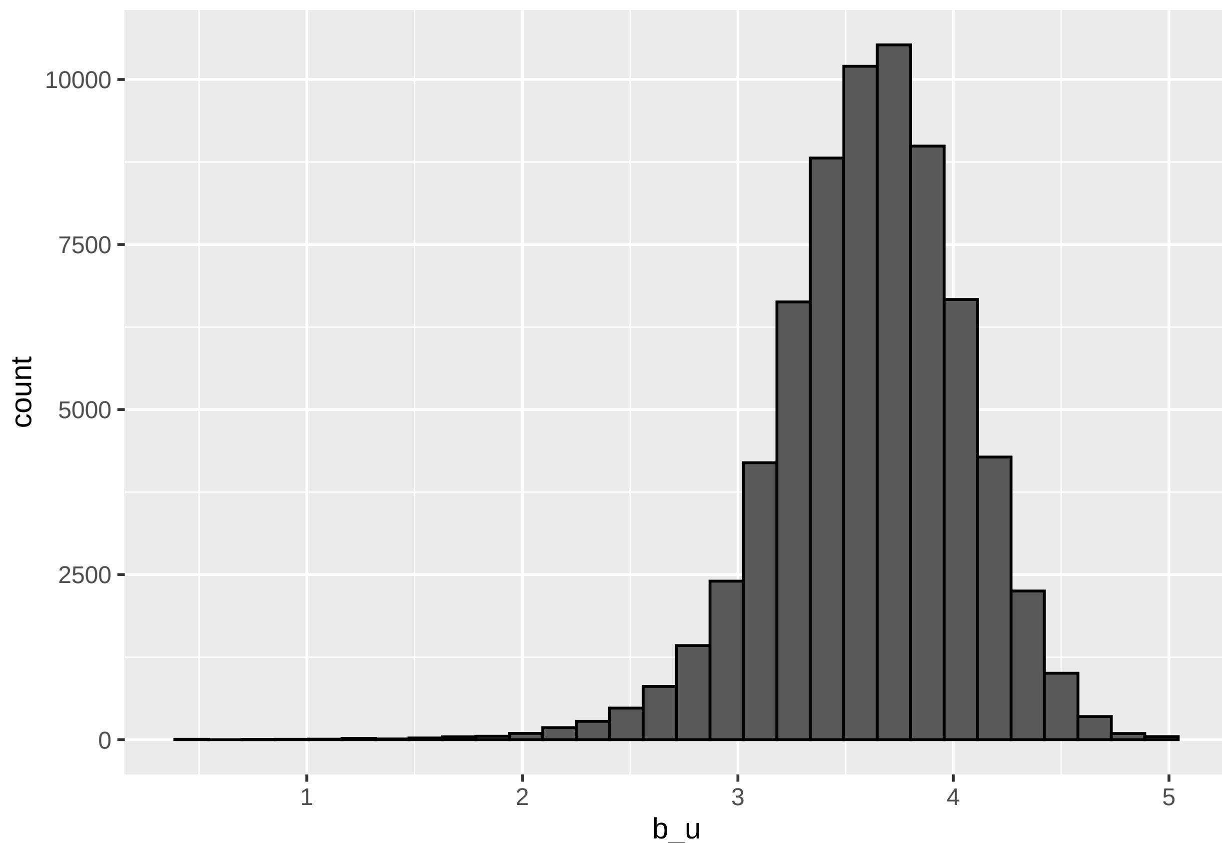
```r
#predict ratings in the test set
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  .$pred
rmse_movie_effect<-RMSE(predicted_ratings, test_set$rating)

# updating result table of RMSE
rmse_results <- bind_rows (rmse_results,tibble(method = "Adjusted mean by Movie effect", RMSE = rmse_mo
rmse_results
```

```
## # A tibble: 2 x 2
##   method                         RMSE
##   <chr>                         <dbl>
## 1 Same rating for all movies    1.06
## 2 Adjusted mean by Movie effect 0.944
```

3rd Model: Different users rated movies differently

```r
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```

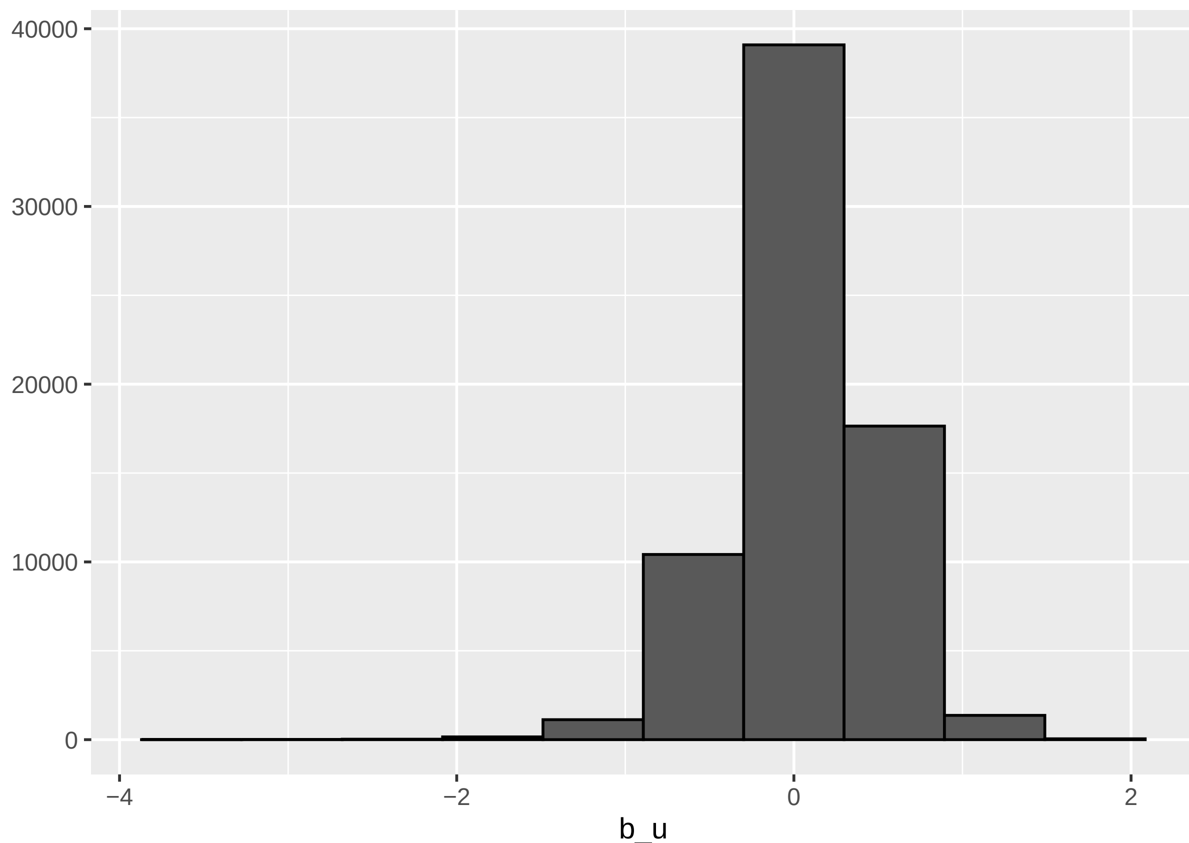
```
## `summarise()` ungrouping output (override with `.groups` argument)
```




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

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

```
qplot(b_u, data = user_avgs, bins = 10, color = I("black"))
```




```
#ajdust mean by user and movie effect
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - b_i - mu))
```

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

```
#predict ratings in the test set
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
rmse_user_effect<-RMSE(predicted_ratings, test_set$rating)

# updating result table of RMSE
rmse_results <- bind_rows (rmse_results,tibble(method = "Adjusted mean by Movie  & User effect", RMSE =
rmse_results
```

```
## # A tibble: 3 x 2
##   method                                RMSE
##   <chr>                                <dbl>
## 1 Same rating for all movies           1.06
```

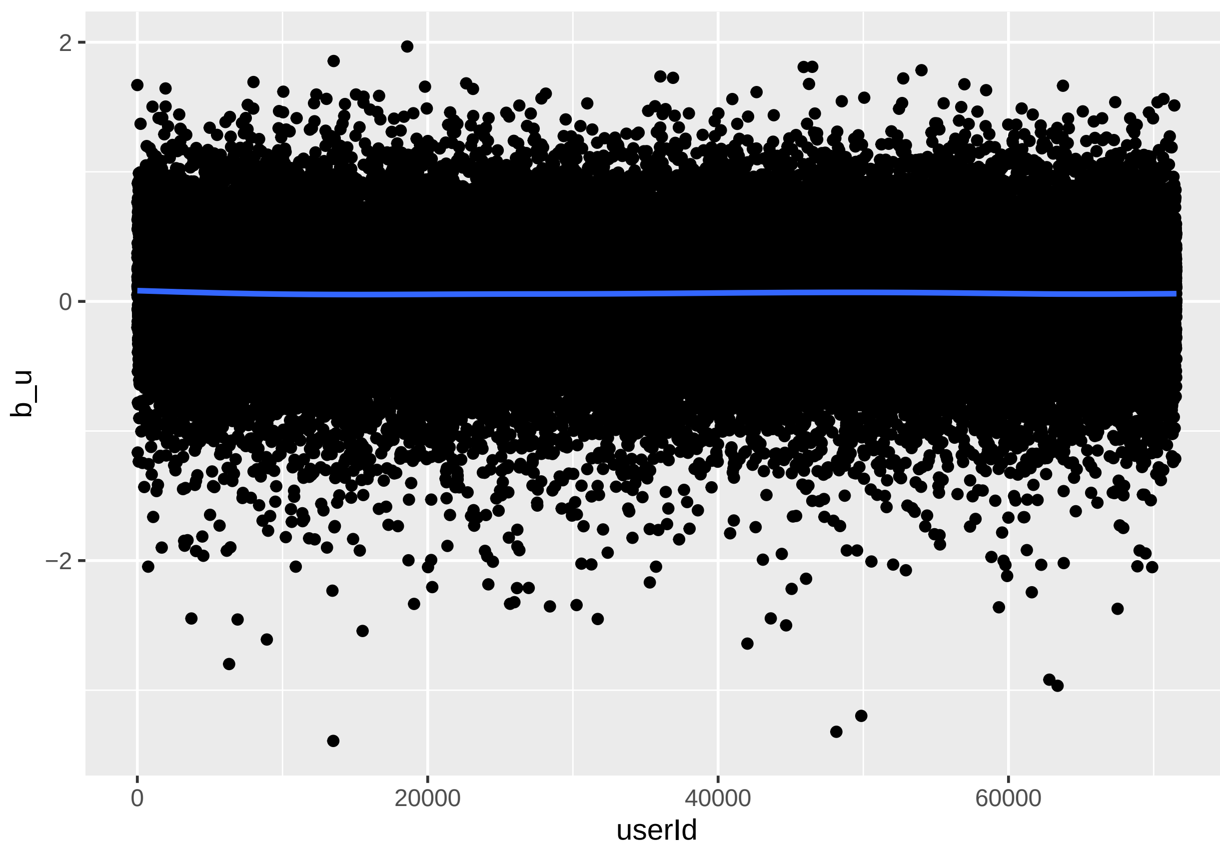
```
## 2 Adjusted mean by Movie effect         0.944
## 3 Adjusted mean by Movie  & User effect 0.866
```

Ploting

```
train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i)) %>%
  ggplot(aes(userId, b_u)) +
  geom_point() +
  geom_smooth()
```

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

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



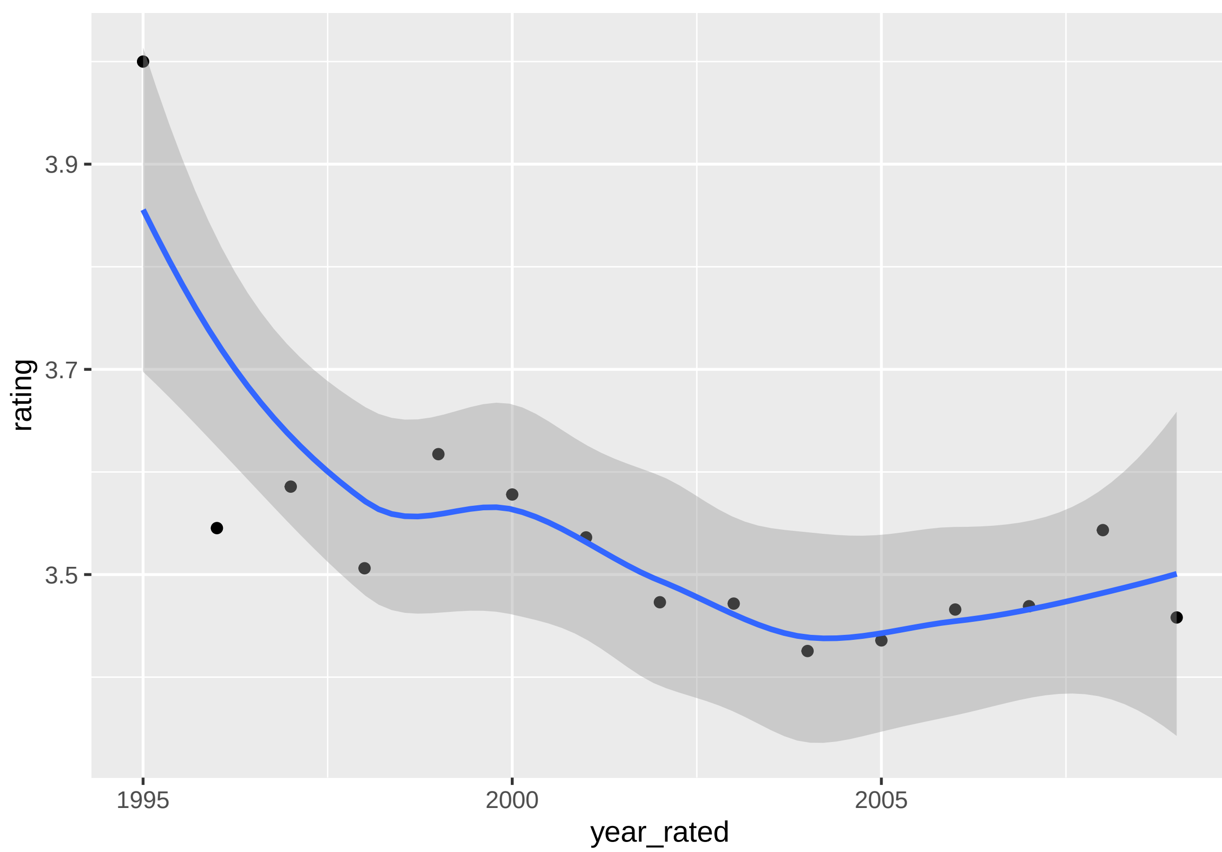
Desiding on other predictors

is "year of movie rating" significant for model improvement

```
edx %>%
  group_by(year_rated) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(year_rated, rating)) +
  geom_point() +
  geom_smooth()
```

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

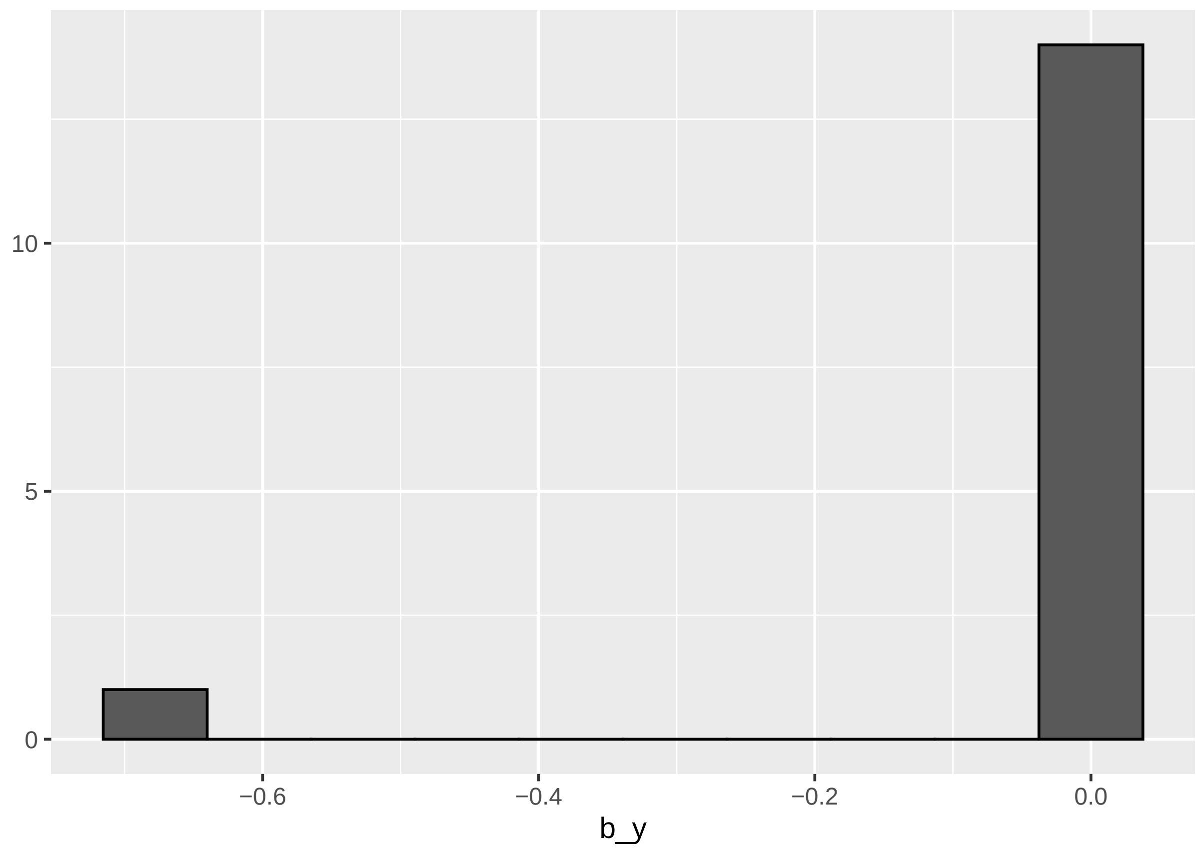
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




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

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

```
qplot(b_y, data = year_rating_avgs, bins = 10, color = I("black"))
```



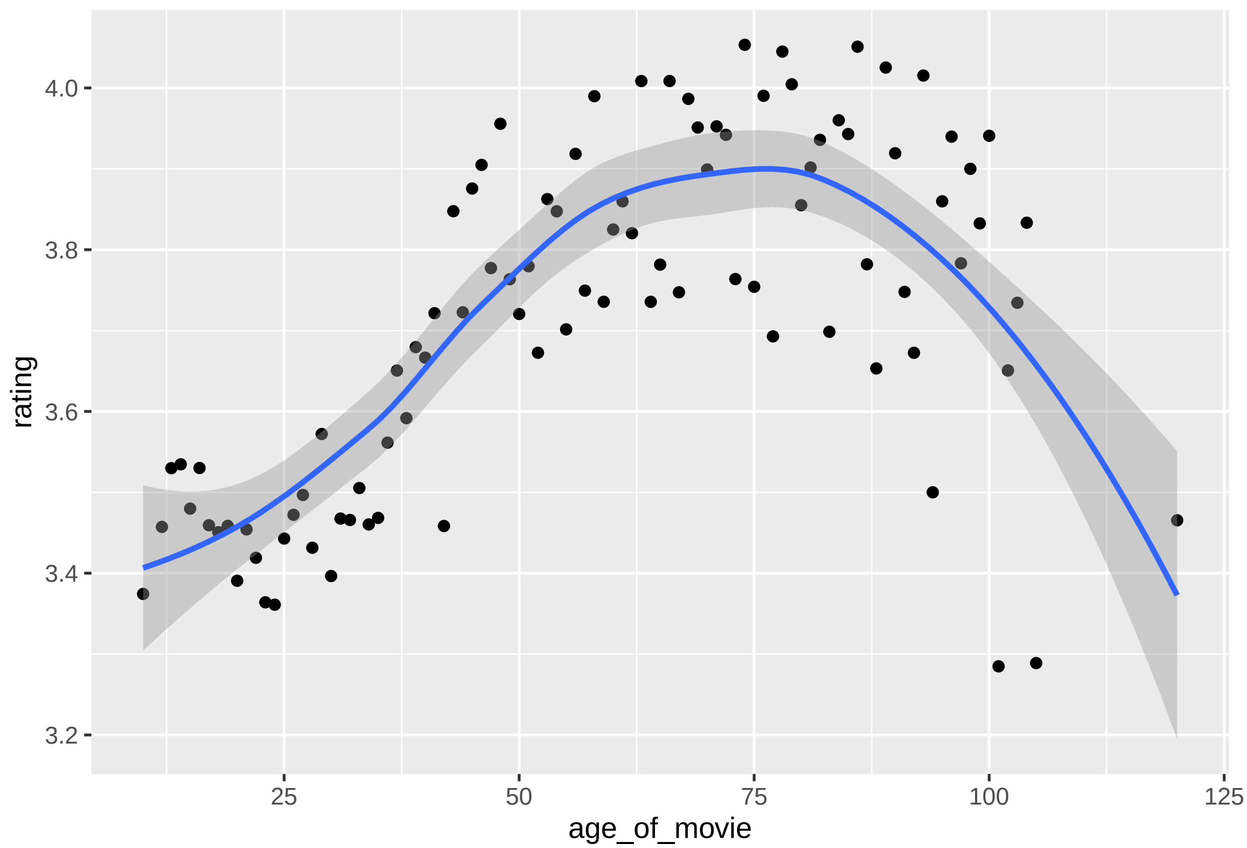
Observations: From above graph, we can see there is a very less evidence of time effect. We are not going to consider this variable for our model improvements

Do we need to consider age of movie

```
edx %>%
  group_by(age_of_movie) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(age_of_movie, rating)) +
  geom_point() +
  geom_smooth()
```

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

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Observations:Graph looks more linear between age between 25 and 75 years

```
age_avgs <- edx %>% group_by(age_of_movie) %>% summarize(avg_rating_by_age = mean(rating))
```

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

```
age_between25_and_75 <- age_avgs %>% filter((age_of_movie > 25) & (age_of_movie < 75))
#plot the graph
age_between25_and_75 %>%
  ggplot(aes(age_of_movie, avg_rating_by_age)) +
  geom_point() + ggtitle("Movies between 25 and 75 years old vs average movie rating")+
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

Observation: We can find out a linear trend. We can check the R-Squared

```r
# R-square
summary(lm(avg_rating_by_age ~ age_of_movie, data = age_between25_and_75))
```

```
##
## Call:
## lm(formula = avg_rating_by_age ~ age_of_movie, data = age_between25_and_75)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.215866 -0.078000 -0.006203  0.063304  0.237322
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.216953   0.055956  57.491  < 2e-16 ***
## age_of_movie 0.010447   0.001077   9.701 8.44e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1066 on 47 degrees of freedom
```

```
## Multiple R-squared:  0.6669, Adjusted R-squared:  0.6599
## F-statistic: 94.12 on 1 and 47 DF,  p-value: 8.443e-13
```

Observation: Here the R-Squared is 0.6599

Now we should check the correlation with Age of movies and others

```r
# Correlation

#Number of movie ratings per movie
n_movies_ratings <- edx %>% group_by(movieId) %>% summarize(n = n())
```

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

```r
#Average Movie Rating for each movie
avg_movie_rat <- edx %>% group_by(movieId) %>% summarize(avg_m_r = mean(rating))
```

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

```r
#Create correlation data
cor_dat <- edx %>% select(rating, movieId, userId, year_rated, age_of_movie, rating_date_range, premier_
  left_join(n_movies_ratings, by = "movieId") %>%
  left_join(avg_movie_rat, by = 'movieId')
head(cor_dat)
```

```
##    rating movieId userId year_rated age_of_movie rating_date_range premier_date
## 1:      5     122      1       1996           28                 4         1992
## 2:      5     185      1       1996           25                 1         1995
## 3:      5     292      1       1996           25                 1         1995
## 4:      5     316      1       1996           26                 2         1994
## 5:      5     329      1       1996           26                 2         1994
## 6:      5     355      1       1996           26                 2         1994
##        n  avg_m_r
## 1:  2178 2.858586
## 2: 13469 3.129334
## 3: 14447 3.418011
## 4: 17030 3.349677
## 5: 14550 3.337457
## 6:  4831 2.487787
```

```r
corr_by_age_of_movie <- cor_dat %>% filter((age_of_movie >25) & (age_of_movie < 70))
head(corr_by_age_of_movie)
```

```
##    rating movieId userId year_rated age_of_movie rating_date_range premier_date
## 1:      5     122      1       1996           28                 4         1992
## 2:      5     316      1       1996           26                 2         1994
## 3:      5     329      1       1996           26                 2         1994
## 4:      5     355      1       1996           26                 2         1994
```

```
## 5:       5     356      1       1996           26                     2          1994
## 6:       5     362      1       1996           26                     2          1994
##         n  avg_m_r
## 1:  2178 2.858586
## 2: 17030 3.349677
## 3: 14550 3.337457
## 4:  4831 2.487787
## 5: 31079 4.012822
## 6:  3612 3.455011
```

```
cor(data.frame(x = corr_by_age_of_movie$age_of_movie, y = corr_by_age_of_movie$avg_m_r))
```

```
##           x         y
## x 1.0000000 0.2772166
## y 0.2772166 1.0000000
```

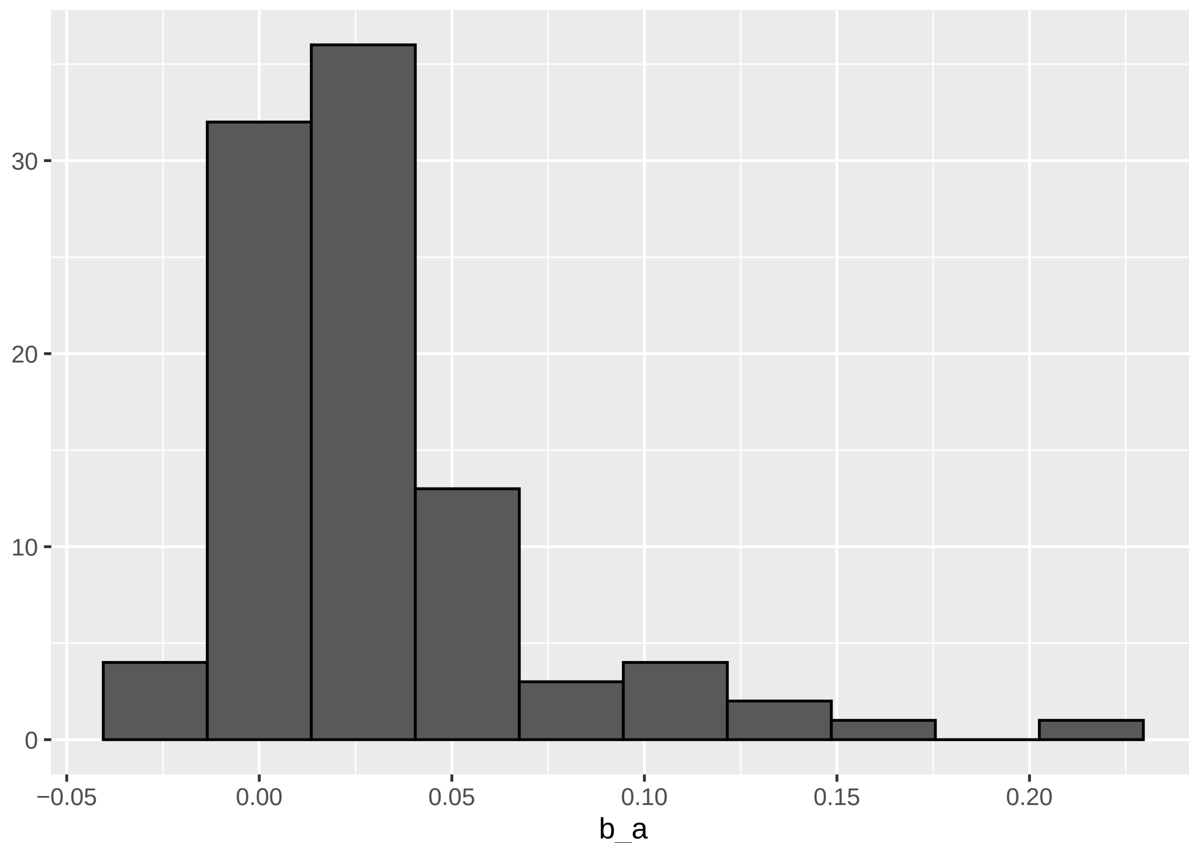
HERE r =0.28, hence not a strong correlation with age of movies (25 to 75 years) and average movie rating.But,there is a positive correlation.

Ploting

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

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

```
qplot(b_a, data = age_avgs, bins = 10, color = I("black"))
```



Observations: We may consider age_of_movie as a predictor

Model 4: Adjusted mean by Movie,User effect & age of movie

```
#ajdust mean by user, movie effect and age of movie effect
b_a <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(age_of_movie) %>%
  summarize(b_a = mean(rating - b_i - mu - b_u))
```

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

```
#predict ratings in the test set
predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_a, by = "age_of_movie") %>%
  mutate(pred = mu + b_i + b_u + b_a) %>%
  .$pred
rmse_age_movie_effect<-RMSE(predicted_ratings, test_set$rating)
```

```r
# updating result table of RMSE
rmse_results <- bind_rows (rmse_results,tibble(method = "Adjusted mean by Movie  & User effect & age of
rmse_results
```

```
## # A tibble: 4 x 2
##   method                                                RMSE
##   <chr>                                                <dbl>
## 1 Same rating for all movies                           1.06
## 2 Adjusted mean by Movie effect                        0.944
## 3 Adjusted mean by Movie  & User effect                0.866
## 4 Adjusted mean by Movie  & User effect & age of movie 0.866
```

Results: Final Model Selection

I have started with average_ranking for movie i, user rating variablity (different users rated differently), year of rating, age of movie (2020-release date) to establish the suitable model. Basis on above analysis, I am not considering year of rating for my fnal model building as those are not contributing much on my model's overall RMSE. I also tried to explore differnt models for the best fit (not adding all trials here) with test set , but while adopting, basis on the matching on predicted rating with validation sets rating, I have chosen penealized least squares approach.

Choose Optimal Penalty Rate lamda

```r
#Root Mean Square Error Loss Function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2,na.rm=TRUE))
}
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas,function(l){

  #Calculate the mean of ratings from the edx training set
  mu <- mean(train_set$rating)

  #Adjust mean by movie effect and penalize low number on ratings
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  #ajdust mean by user and movie effect and penalize low number of ratings
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  #ajdust mean by user and movie effect and age of movie penalize low number of ratings
  b_a <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(age_of_movie) %>%
    summarize(b_a = sum(rating - b_i - mu - b_u)/(n()+l))
```

```r
  #predict ratings in the test set to derive optimal penalty value 'lambda'
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_a, by = "age_of_movie") %>%
    mutate(pred = mu + b_i + b_u + b_a) %>%
    .$pred

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

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

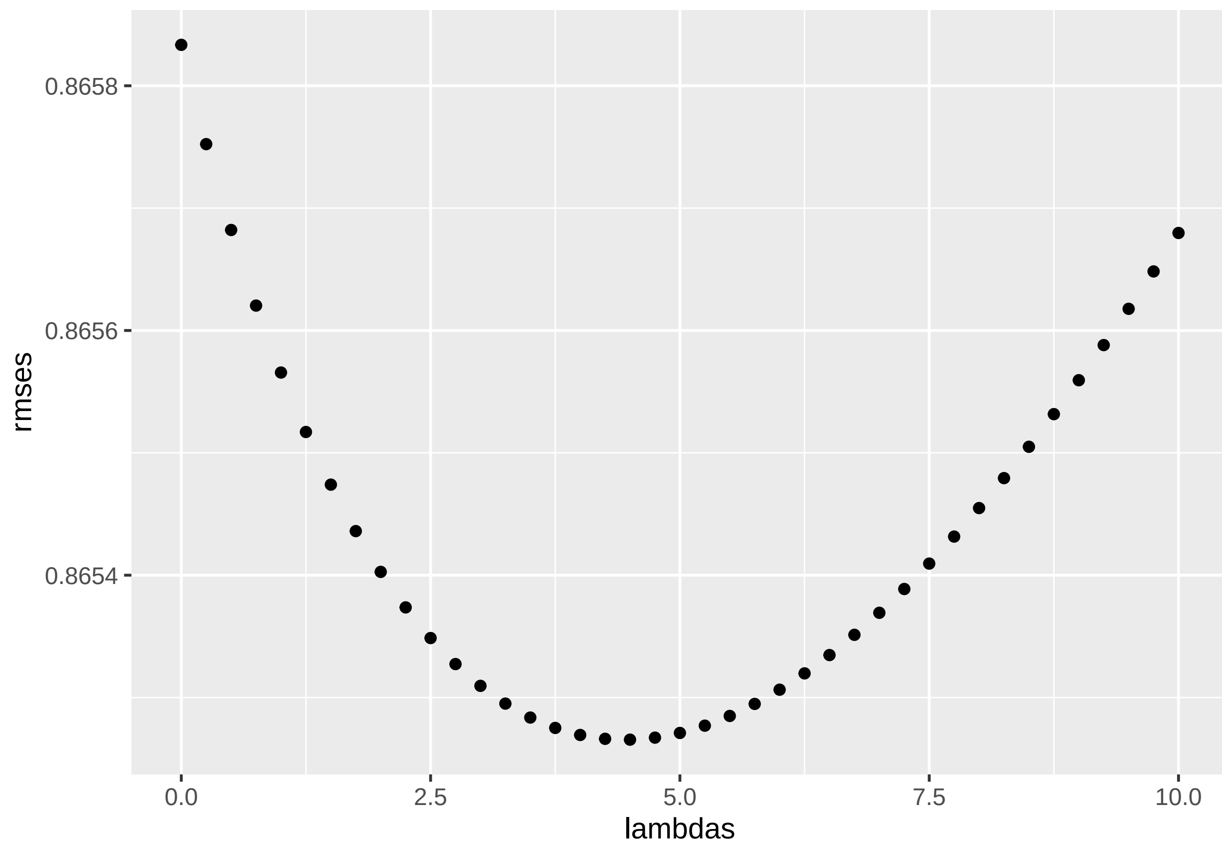
```
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘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

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

```r
lambda <- lambdas[which.min(rmses)]
paste('Optimal RMSE of',min(rmses),'is achieved with Lambda',lambda)
```

```
## [1] "Optimal RMSE of 0.865265580979917 is achieved with Lambda 4.5"
```

```r
qplot(lambdas, rmses)
```



#Using optimal lamda to validate validation data set

```
# validation data
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2,na.rm=TRUE))
}
lambdas <- 4.5

RMSE_validation <- sapply(lambdas,function(l){


  #Calculate the mean of ratings
  mu <- mean(train_set$rating)

  #Adjust mean by movie effect and penalize low number on ratings
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  #ajdust mean by user and movie effect and penalize low number of ratings
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  #ajdust mean by user and movie effect and age of movie penalize low number of ratings
```

```r
  b_a <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(age_of_movie) %>%
    summarize(b_a = sum(rating - b_i - mu - b_u)/(n()+l))

  #predict ratings in the validation set to derive optimal penalty value 'lambda'
    predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_a, by = "age_of_movie") %>%
    mutate(pred = mu + b_i + b_u + b_a) %>%
    .$pred

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

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

```r
RMSE_validation
```

```
## [1] 0.8653378
```

##Conclusion The accuracy is measured as absolute difference between the predicted value and the acutal value. In final model, "Movie effect", "User effect" and "age of Movie" have been considered as predictors. In the validation data set the RMSE is 0.8653378. .I faced lot of challenges while managing this huge dataset considering my Laptop configuration constraints. I tried my level best to make the model building in logical manner and tried to include as much evidence (code), I can. Considering the Loptop configuration, I only included the codes which are logical for this analysis. Also, I really did lot of exploration on different models building but I am not including them here.