MovieLens Project

Setup data

```
################################
# Create edx set, validation set
################################

# 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
```

```
## Error: package or namespace load failed for 'tidyverse' in loadNamespace(i, c(lib.loc, .libPaths()),
##  namespace 'vctrs' 0.2.4 is already loaded, but >= 0.3.0 is required
```

```
## Installing package into 'C:/Users/stevefuckyou/Documents/R/win-library/3.6'
## (as 'lib' is unspecified)
```

```
## package 'tidyverse' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\stevefuckyou\AppData\Local\Temp\RtmpGQMjsr\downloaded_packages
```

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

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

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

```
## The following object is masked from 'package:naivebayes':
##
##     tables
```

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

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

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

```
# 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"))

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

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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

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

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

Executive Summary

In this project, I have used regression techniques to build a movie recomendation system using the 'edx' 'MovieLens' data. The goal of the project will be to predict how users will rate movies, using the dataset mentioned, as if we know a user is likely to rate a movie highly, we can recommend this movie to the users.

The approaches used to build the recommendation system are based on linear regression. For each category in the dataset we will average how that observations in these categories differ from the mean and apply this

to our future predictions. We know, for example, a movie with a high rating is more likely to be rated highly in the future. I have also used regularization as there are some unique and low number of observations in the dataset so we need will get a better model by punishing extreme results with low or unique observations.

I've used Root Mean Squared Error (RMSE) to evaluate the performance of the model. The final RMSE was 0.8648006.

##Used libraries

The following libraries were used for this project:

Dataset

The Dataset used is from MovieLens. It is split into 90% 'edx' which we will use to train our model and 10% 'validation' which will be used only in the final part of the report to test the accuracy of our model.

I will split the 'edx' dataset into a train and test set in order to evaluate the performance of my model throughout the project as we analyse the data and make further improvements to the final model.

The 'edx' data is a data frame with 9,000,055 rows and is split into 6 columns. It is in tidy format with each row representing one observation. The Columns are: 'userId', 'movieId','rating','timestamp','title' and 'genres'.

##Data summary

A summary of the data is given below along with the head of the data:

```
# edx class
class(edx)
```

```
## [1] "data.frame"
```

```
# Display head of edx
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
```

```
# Number of rows in data
nrow(edx)
```

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

```r
# Number of columns in data
ncol(edx)
```

```
## [1] 6
```

```r
# Number of unique users
length(unique(edx$userId))
```

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

```r
# Number of films in dataset
length(unique(edx$movieId))
```

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

```r
# Average rating
mu <- mean(edx$rating)
```

Method

Pre-processing data

The data is in tidy format and the only pre-processing needed is to turn the timestamp into a readable date:

```r
edx <- edx %>% mutate(year_rated = year(as_datetime(timestamp)))
validation <- validation %>% mutate(year_rated = year(as_datetime(timestamp)))
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     292      5 838983421               Outbreak (1995)
## 4      1     316      5 838983392               Stargate (1994)
## 5      1     329      5 838983392 Star Trek: Generations (1994)
## 6      1     355      5 838984474       Flintstones, The (1994)
##                          genres year_rated
## 1                Comedy|Romance       1996
## 2         Action|Crime|Thriller       1996
## 3  Action|Drama|Sci-Fi|Thriller       1996
## 4       Action|Adventure|Sci-Fi       1996
## 5 Action|Adventure|Drama|Sci-Fi       1996
## 6       Children|Comedy|Fantasy       1996
```

The first thing we need to do is to split the data into a training set (80%) and a test set (20%):

```r
# Split data into train and test set
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
```

```
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# Ensure test set and train set have same movies
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Data exploration and visualisations

A quick look at the data reveals that the most rated films are what you would expect.

```
train_set %>%
  group_by(title) %>%
  summarize(n_ratings = n()) %>%
  arrange(desc(n_ratings))
```

```
## # A tibble: 10,640 x 2
##    title                                                     n_ratings
##    <chr>                                                         <int>
##  1 Pulp Fiction (1994)                                           25137
##  2 Forrest Gump (1994)                                           24943
##  3 Silence of the Lambs, The (1991)                              24432
##  4 Jurassic Park (1993)                                          23492
##  5 Shawshank Redemption, The (1994)                              22406
##  6 Braveheart (1995)                                             20881
##  7 Fugitive, The (1993)                                          20859
##  8 Terminator 2: Judgment Day (1991)                             20752
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)  20591
## 10 Apollo 13 (1995)                                              19494
## # ... with 10,630 more rows
```

However, looking at the highest and lowest rated films gives us a list of quite obscure movies suggesting which we can see are skewed by relatively low number of ratings. This suggests there are movies in the dataset with low number of observations and extreme results:

```
# Highest rated films
train_set %>%
  group_by(title) %>%
  summarize(avg_rating = mean(rating),n_ratings = n()) %>%
  arrange(desc(avg_rating))
```

```
## # A tibble: 10,640 x 3
##    title                                          avg_rating n_ratings
##    <chr>                                               <dbl>     <int>
##  1 Blue Light, The (Das Blaue Licht) (1932)                5         1
##  2 Bullfighter and the Lady (1951)                         5         1
##  3 Fighting Elegy (Kenka erejii) (1966)                    5         1
##  4 Hellhounds on My Trail (1999)                           5         1
##  5 Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)                     5         2
##  6 Shanghai Express (1932)                                 5         1
```

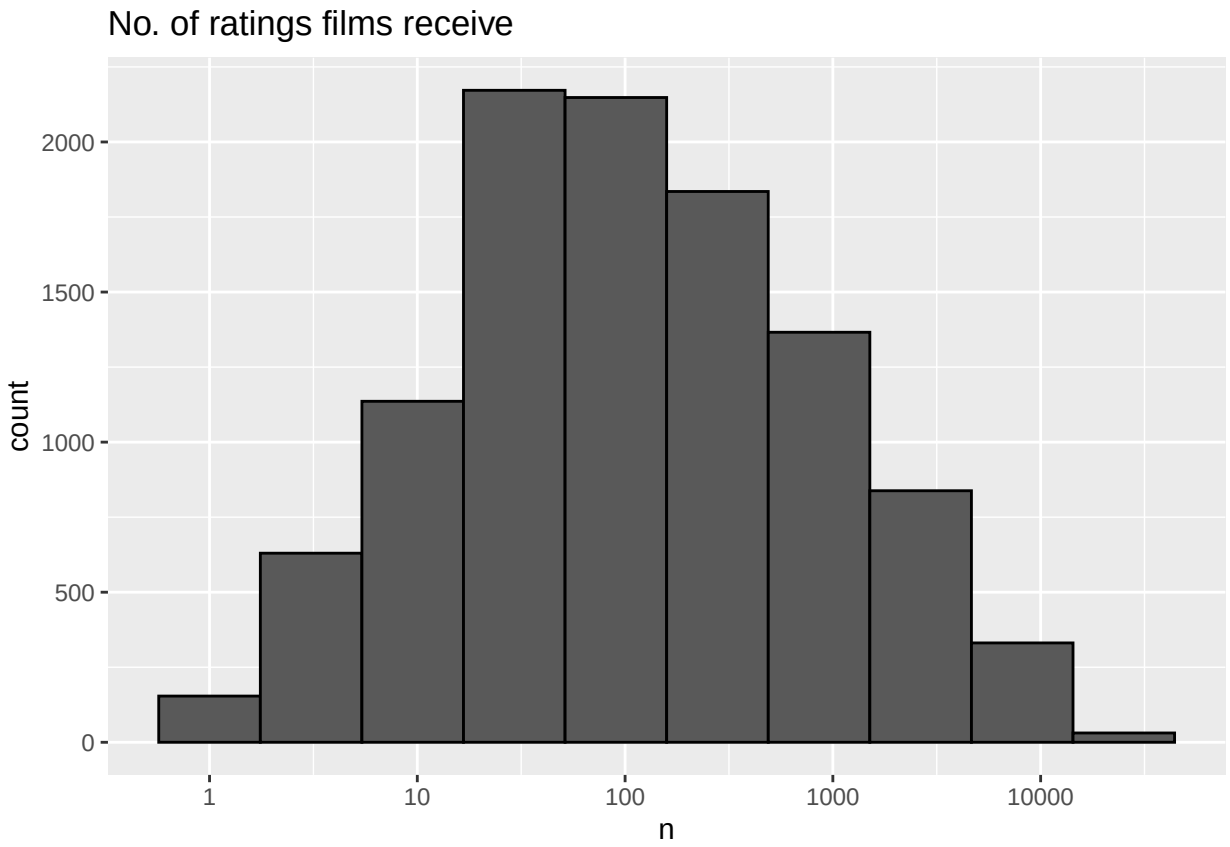
```
##  7 Sun Alley (Sonnenallee) (1999)                              5           1
##  8 Human Condition II, The (Ningen no joken II) (1959)       4.83          3
##  9 Constantine's Sword (2007)                                4.75          2
## 10 Life of Oharu, The (Saikaku ichidai onna) (1952)          4.75          2
## # ... with 10,630 more rows
```

```
# Lowest rates films
train_set %>%
  group_by(title) %>%
  summarize(avg_rating = mean(rating),n_ratings = n()) %>%
  arrange(avg_rating)
```

```
## # A tibble: 10,640 x 3
##    title                                      avg_rating n_ratings
##    <chr>                                           <dbl>     <int>
##  1 Besotted (2001)                                 0.5           2
##  2 Confessions of a Superhero (2007)               0.5           1
##  3 Grief (1993)                                    0.5           1
##  4 War of the Worlds 2: The Next Wave (2008)       0.5           1
##  5 Disaster Movie (2008)                           0.783        30
##  6 SuperBabies: Baby Geniuses 2 (2004)             0.8          40
##  7 From Justin to Kelly (2003)                     0.860       161
##  8 Hip Hop Witch, Da (2000)                        0.9          10
##  9 Camera Obscura (2000)                           1             2
## 10 Criminals (1996)                                1             2
## # ... with 10,630 more rows
```

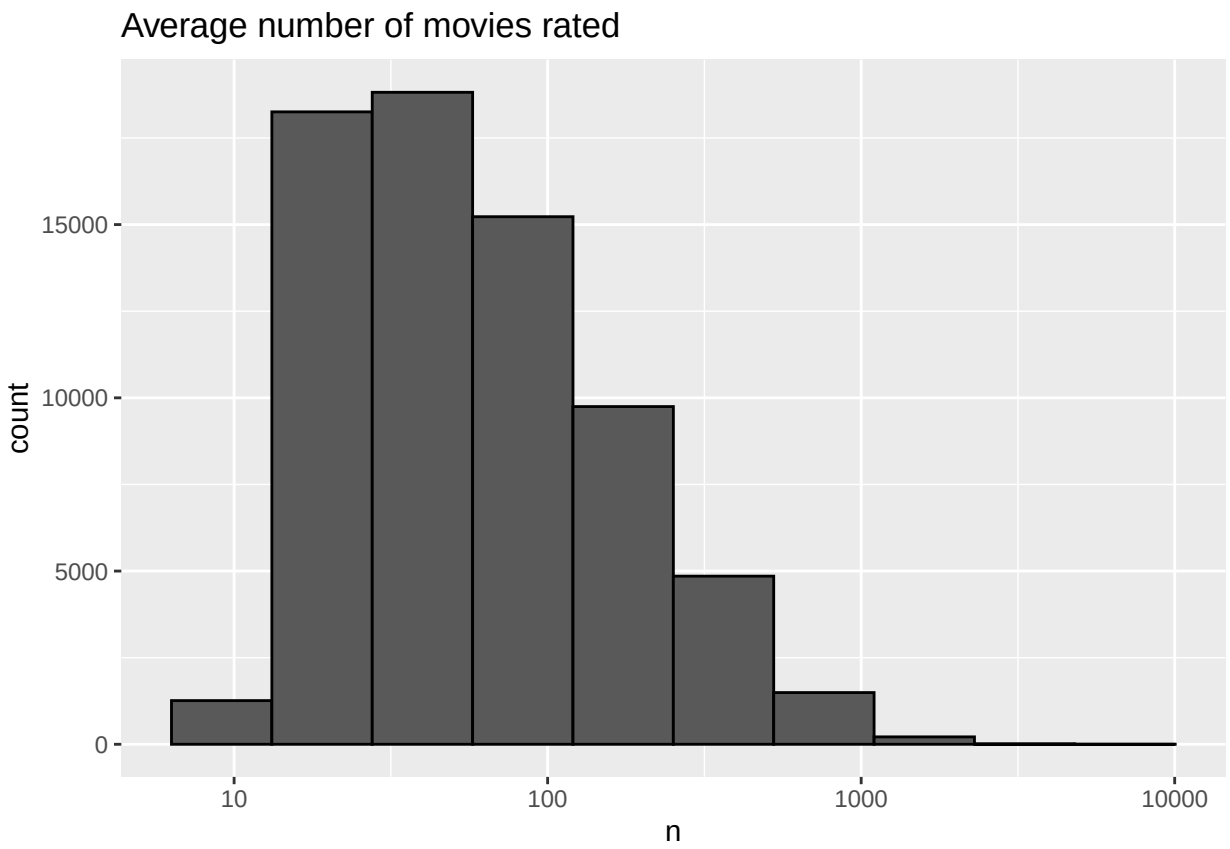
We can visualise the number of ratings each film receives:

```
# Visualise how many ratings the movies in our dataset have received
train_set %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 10, color = "black") +
  scale_x_log10() +
  ggtitle("No. of ratings films receive")
```



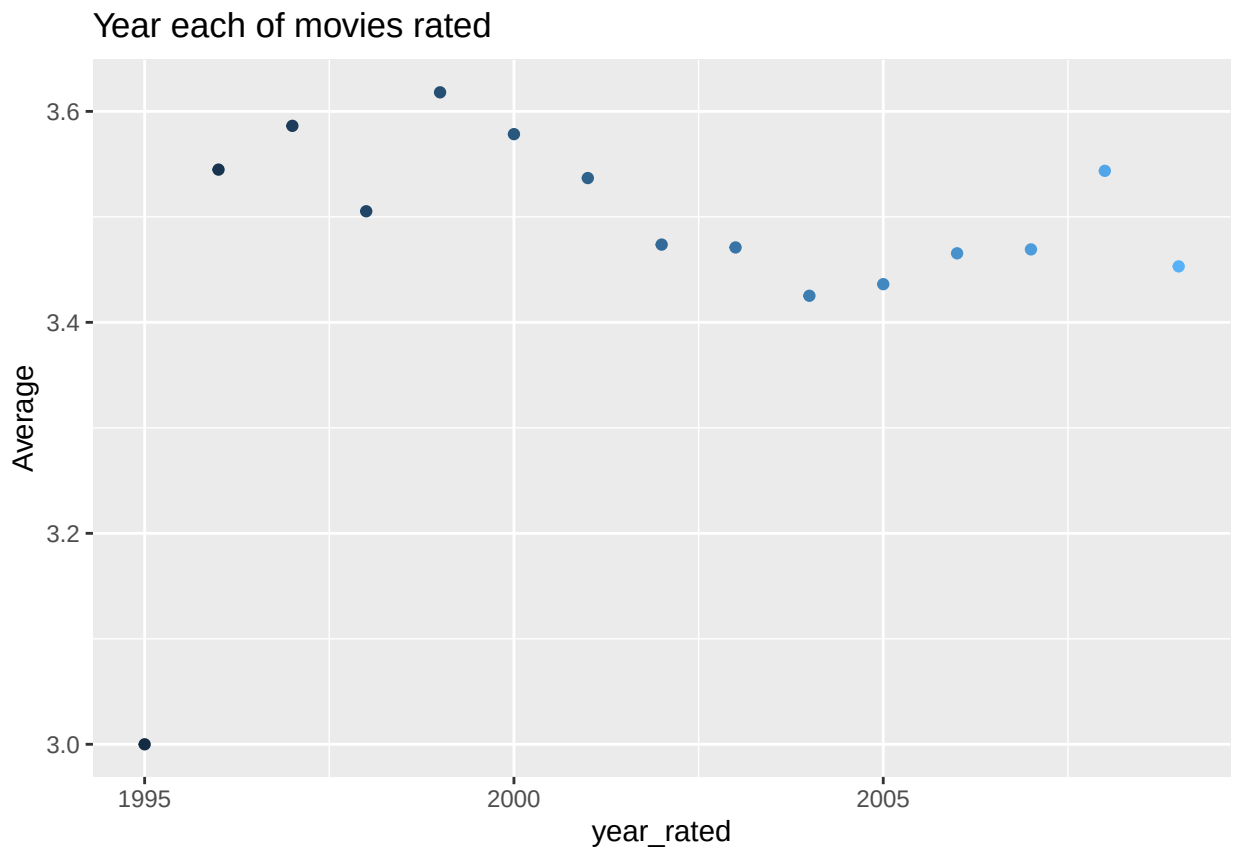
We can also check how many movies each person in the dataset has rated. Whilst the average number of reviews is high there are lots with lower number of observations. Like with the low number of observations for some movie reviews this suggests we should use regularization on the number of user reviews as well as the number of film reviews. :

```
# Visualised how many movies our users rate on average
train_set %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 10, color = "black") +
  scale_x_log10() +
    ggtitle("Average number of movies rated")
```



When we look at the ratings for each year we can see variation throughout. It appears the year the rating was submitted will have a slight impact on expected ratings.

```
train_set %>%
  group_by(year_rated) %>%
  summarize(Average = mean(rating)) %>%
  ggplot(aes(x = year_rated, y = Average, color = year_rated)) +
  geom_point() +
  theme(legend.position="none")+
  ggtitle("Year each of movies rated")
```



Insight gained

From the data exploration and visualisations we can see that when forming our final model it will be prudent to consider the User effect as each user has a positive or negative depending on how they typically rate movies.

We will need to take into account how the movies are typically rated. The fact the top and bottom rated movies are quite obscure indicates we will get a better accuracy if we run reguarization techniques on these factors.

The modelling approach I have used is a linear regression models. We assume the films rating is a linear combination of the average ratings plus random noise (user effect, movie effect, year effect etc.).

Results

The way we will judge the accuracy of the predictions will be through the 'root mean squared error' (RMSE). This is the standard deviation of the prediction errors.

We will define RMSE in r using:

```
# Defining RMSE function

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

The first approach I tried as a benchmark is to use the mean average rating. First we defined the average rating 'mu' then we applied this to our RMSE function and stored the results in a data frame:

```
# Train set average
mu <- mean(train_set$rating)

# Checking RMSE against average
Pred_avg <- RMSE(test_set$rating, mu)

# Creating RMSE results table
rmse_results_table <- data.frame(Methods = "Naive Bayes Average", RMSE = Pred_avg)
rmse_results_table
```

```
##               Methods     RMSE
## 1 Naive Bayes Average 1.060704
```

We can see we get an RMSE of 1.0607045 which shows our predictions are off by more than 1 star on average.

Next we saw from our data exploration that movies had very different ratings. We will add in a movie effect. We do this by first giving the film an average on how far away from the mean ratings it is. We then add this figure to the our test set and finally we calculate the RMSE and add this to our table:

```
# Movie effect

movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarize(b_m = (mean(rating - mu)))

# Creating an object including Movie effect to apply RMSE too Validation set
movie_pred <- left_join(test_set,movie_effect, by = 'movieId') %>%
    mutate(b_m = b_m + mu) %>% .$b_m

# Checking RMSE including Movie effect
Pred_mov <- RMSE(test_set$rating,movie_pred)

# New RMSE
rmse_results_table <- bind_rows(rmse_results_table, data.frame(Methods = "Adding Movie effect", RMSE = I
```

```
## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector, coercing
## into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector, coercing
## into character vector
```

```
rmse_results_table
```

```
##               Methods      RMSE
## 1 Naive Bayes Average 1.0607045
## 2 Adding Movie effect 0.9437144
```

We can see we have managed to improve our RMSE as it is now 0.9437144.

Next we will use the same approach but this time take into account the user effect, year effect and genre effect:

```
# Movie and user effect
movie_user_effect <- train_set %>%
  left_join(movie_effect,by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = (mean(rating - mu - b_m)))

# Creating an object including Movie effect to apply RMSE too Validation set
movie_user_pred <- test_set %>%
  left_join(movie_effect, by = 'movieId') %>%
  left_join(movie_user_effect, by = 'userId') %>%
  mutate(b_u = b_u + b_m + mu) %>%
  .$b_u

# Checking RMSE including Movie effect
Pred_mov_user <- RMSE(test_set$rating,movie_user_pred)

# New RMSE
rmse_results_table <- bind_rows(rmse_results_table, data.frame(Methods = "Adding Movie + user effect", I
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector, coercing
## into character vector
```

```
rmse_results_table
```

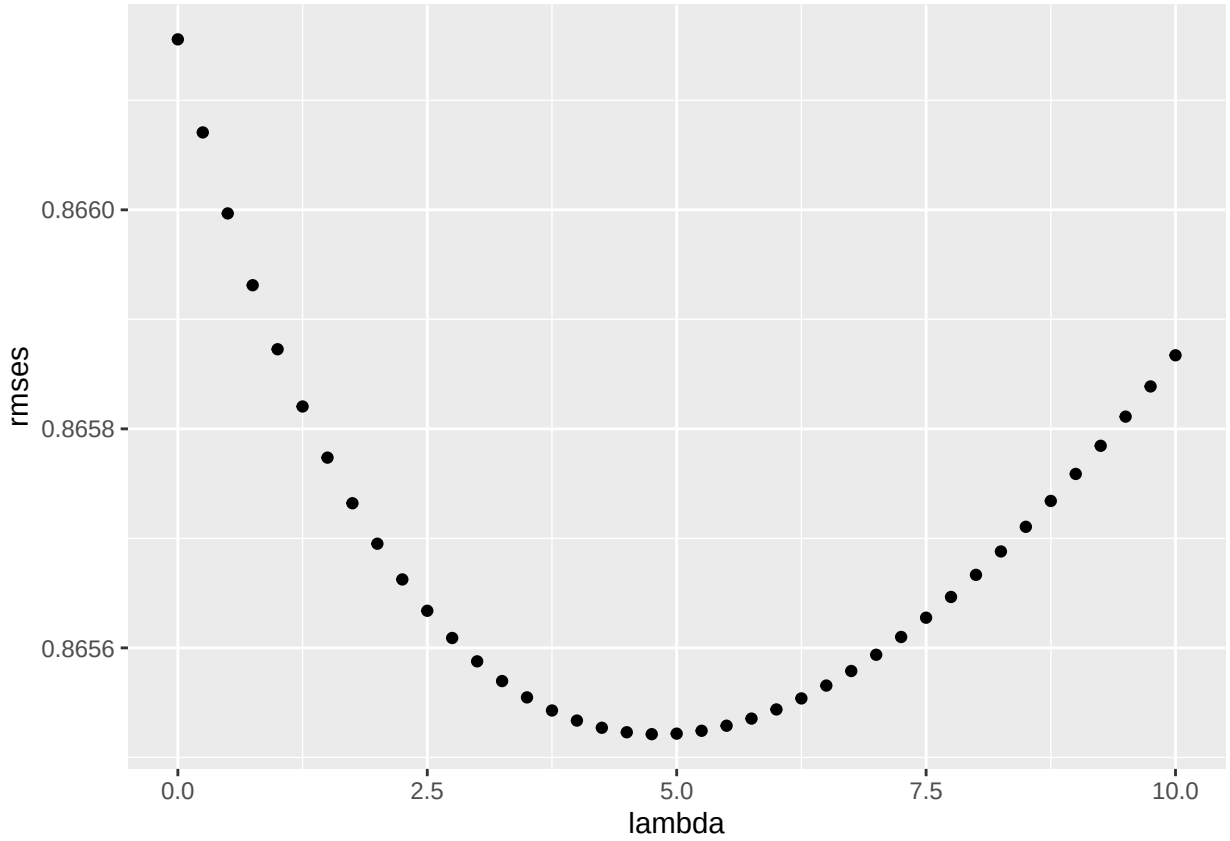
```
##                       Methods      RMSE
## 1         Naive Bayes Average 1.0607045
## 2         Adding Movie effect 0.9437144
## 3 Adding Movie + user effect 0.8661625
```

```
# Year effect

movie_user_year_effect <- train_set %>%
  left_join(movie_effect, by= 'movieId') %>%
  left_join(movie_user_effect, by = 'userId') %>%
  group_by(year_rated) %>%
  summarize(b_y = mean(rating - mu - b_m -b_u))

# Predicting ratings
movie_user_year_pred <- test_set %>%
  left_join(movie_effect, by = "movieId") %>%
  left_join(movie_user_effect, by = "userId") %>%
  left_join(movie_user_year_effect, by = 'year_rated') %>%
  mutate(pred = mu + b_m + b_u + b_y) %>%
  .$pred

# Calculate RMSE
Pred_mov_user_year <- RMSE(test_set$rating,movie_user_year_pred)
```

Finally we will use regularization to take into account users who have rated few movies and movies which have been rated few times. Regularisation uses regression techniques to punish these movies more than ones which have been rated more often, regressing our prediction towards the mean.

In the code we will test lambda from 1-10 to see by which lambda punishing lower numbers improves our RMSE most:

```
lambda <- seq(0, 10, 0.25)
rmses <- sapply(lambda, function(l){
  b_m <- train_set %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_m - mu)/(n()+l))
  b_y <- train_set %>%
    left_join(b_m, by= 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    group_by(year_rated) %>%
    summarize(b_y = mean(rating - mu - b_m -b_u))
  predicted_ratings <-
    test_set %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = 'year_rated') %>%
    mutate(pred = mu + b_m + b_u + b_y) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambda, rmses)
```




```
low_lambda <- lambda[which.min(rmses)]
low_lambda
```

```
## [1] 4.75
```

```
rmses[19]
```

```
## [1] 0.865523
```

Final result

We can we the best performing lambda is 4.75. We will now plug this into our final model which we will now use on the validation set.

Just a note on the method. We didn't use the lm() function in R to calculate these as this would have been very slow to process given the huge number of calculations needed from the sizable dataset.

Our model has managed to achieve a final RMSE of 0.864800 against the validation dataset.

```
# Apply rgularized movie effect
b_m <- edx %>%
  group_by(movieId) %>%
  summarize(b_m = sum(rating - mu)/(n()+4.75))

# Apply regularized user effect
```

```
b_u <- edx %>%
  left_join(b_m, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_m - mu)/(n()+4.75))

# Apply year effect
b_y <- edx %>%
  left_join(b_m, by= 'movieId') %>%
  left_join(b_u, by = 'userId') %>%
  group_by(year_rated) %>%
  summarize(b_y = mean(rating - mu - b_m -b_u))


# Calculating predictions on validation set
pred_final_ratings <-
  validation %>%
  left_join(b_m, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = 'year_rated') %>%
  mutate(pred = mu + b_m + b_u + b_y) %>%
  .$pred

# Calculate RMSE
final_rmse <- RMSE(validation$rating, pred_final_ratings)

# Update table
rmse_results_table <- bind_rows(rmse_results_table, data.frame(Methods = "Final RMSE", RMSE = final_rms
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector, coercing
## into character vector
```

```
rmse_results_table
```

```
##                       Methods      RMSE
## 1         Naive Bayes Average 1.0607045
## 2         Adding Movie effect 0.9437144
## 3 Adding Movie + user effect 0.8661625
## 4                 Final RMSE 0.8648006
```

Conclusion

We can see from the report that we've managed to drastically improve our RMSE from 1.0607045 to 0.8648006 using simple regression techniques just on the factors included in the initial table. We optimised these factors using regularization.

In terms of future work to improve the RMSE even further I would look into see how different genres were rated differently and apply a regularized 'genre effect' which I believe would further improve the model.

I would also look at applying matrix factorisation techniques which would look for trends in the data e.g. some people might prefer blockbuster films and this should be added into the final model.