Movielens

Luke Straughan

23/03/2020

Introduction

This report is one of the capstone assignments in the Harvard edX Data Science course. The goal is to create a machine learning algorithm to accurately predict movie ratings and would, therefore, be able to recommend films to users. The dataset was provided by the edX team. It is the Movielens data set that contains approximately 10 Million movie recommendations. It contains information such as user IDs, movie IDs, the titles of said movies (which include the year in which they were realeased), the genres of said movies, the ratings that each user has given a movie, and the timestamp of the time in which the rating was made. Firstly, data wrangling took place in order to attain the data. This was done entirely with the code provided by the edX team. Secondly, the data is analysed, primarily through data visualisation techniques. Throughout this process, more data cleaning takes place as further columns are added to the data set, such as pulling the year from the title and the date from the timestamp in order to make it readable to humans. Thirdly, the effects model, along with regularisation, is used to create the overall model through which the data can be processed and predictions can be made. Following that section, the results will be tested on the validation set and a final RMSE (root mean square error) can be made. The ultimate goal, and test to see whether the modelling is effective, is to have the RMSE below 0.86490. As aforementioned, however, the process starts with the data wrangling.

Data Wrangling

The following code was provided by the edX team:

```
################################
# 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
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- 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

## Warning: package 'caret' was built under R version 3.6.2

## 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

## Warning: package 'data.table' was built under R version 3.6.2

##
## 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(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")
```

```
## 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$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)
```

The above code downloads and cleans the data in order to make it easier to work with. Then, the dataset is split up into two sets - the training set (edx) and the testing set (validation). 90% of the data remains in the edx set while 10% is assigned to the validation set that will be used to test the final results of the modelling.

Data Analysis

Here the data is checked to see what variables one has to work with.

```
## # A tibble: 9,000,055 x 6
##    userId movieId rating timestamp title              genres
##     <int>   <dbl>  <dbl>     <int> <chr>              <chr>
##  1      1     122      5 838985046 Boomerang (1992)   Comedy|Romance
##  2      1     185      5 838983525 Net, The (1995)    Action|Crime|Thrill~
##  3      1     292      5 838983421 Outbreak (1995)    Action|Drama|Sci-Fi~
##  4      1     316      5 838983392 Stargate (1994)    Action|Adventure|Sc~
##  5      1     329      5 838983392 Star Trek: Generat~ Action|Adventure|Dr~
##  6      1     355      5 838984474 Flintstones, The (~ Children|Comedy|Fan~
##  7      1     356      5 838983653 Forrest Gump (1994) Comedy|Drama|Romanc~
##  8      1     362      5 838984885 Jungle Book, The (~ Adventure|Children|~
##  9      1     364      5 838983707 Lion King, The (19~ Adventure|Animation~
## 10      1     370      5 838984596 Naked Gun 33 1/3: ~ Action|Comedy
## # ... with 9,000,045 more rows
```

In order to make the timestamp analysable for humans, it is converted into date-time, which is then used to create the *week* column. This is intended to avoid overfitting the data by making the date broader. Then, the *year* column is created by extracting it from the title. This will allow the year to be its own variable for the data visualisation.

```r
# Timestamp to date-time
if(!require(lubridate)) install.packages("lubridate")
```

```
## Loading required package: 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 object is masked from 'package:base':
##
##     date
```

```r
edx <- mutate(edx, date = as_datetime(timestamp))
edx %>% as_tibble()
```

```
## # A tibble: 9,000,055 x 7
##    userId movieId rating timestamp title     genres     date
##     <int>   <dbl>  <dbl>     <int> <chr>     <chr>      <dttm>
##  1      1     122      5 838985046 Boomeran~ Comedy|Ro~ 1996-08-02 11:24:06
##  2      1     185      5 838983525 Net, The~ Action|Cr~ 1996-08-02 10:58:45
##  3      1     292      5 838983421 Outbreak~ Action|Dr~ 1996-08-02 10:57:01
##  4      1     316      5 838983392 Stargate~ Action|Ad~ 1996-08-02 10:56:32
##  5      1     329      5 838983392 Star Tre~ Action|Ad~ 1996-08-02 10:56:32
##  6      1     355      5 838984474 Flintsto~ Children|~ 1996-08-02 11:14:34
##  7      1     356      5 838983653 Forrest ~ Comedy|Dr~ 1996-08-02 11:00:53
##  8      1     362      5 838984885 Jungle B~ Adventure~ 1996-08-02 11:21:25
##  9      1     364      5 838983707 Lion Kin~ Adventure~ 1996-08-02 11:01:47
## 10      1     370      5 838984596 Naked Gu~ Action|Co~ 1996-08-02 11:16:36
## # ... with 9,000,045 more rows
```

```r
# Add week column to edx
edx <- edx %>% mutate(week = round_date(as_datetime(timestamp), unit = "week"))

# Add year column
edx <- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
edx %>% as_tibble()
```

```
## # A tibble: 9,000,055 x 9
##    userId movieId rating timestamp title genres date
##     <int>   <dbl>  <dbl>     <int> <chr> <chr>  <dttm>
##  1      1     122      5 838985046 Boom~ Comed~ 1996-08-02 11:24:06
##  2      1     185      5 838983525 Net,~ Actio~ 1996-08-02 10:58:45
##  3      1     292      5 838983421 Outb~ Actio~ 1996-08-02 10:57:01
##  4      1     316      5 838983392 Star~ Actio~ 1996-08-02 10:56:32
##  5      1     329      5 838983392 Star~ Actio~ 1996-08-02 10:56:32
##  6      1     355      5 838984474 Flin~ Child~ 1996-08-02 11:14:34
```

```
##  7      1     356      5 838983653 Forr~ Comed~ 1996-08-02 11:00:53
##  8      1     362      5 838984885 Jung~ Adven~ 1996-08-02 11:21:25
##  9      1     364      5 838983707 Lion~ Adven~ 1996-08-02 11:01:47
## 10      1     370      5 838984596 Nake~ Actio~ 1996-08-02 11:16:36
## # ... with 9,000,045 more rows, and 2 more variables: week <dttm>,
## #   year <dbl>
```

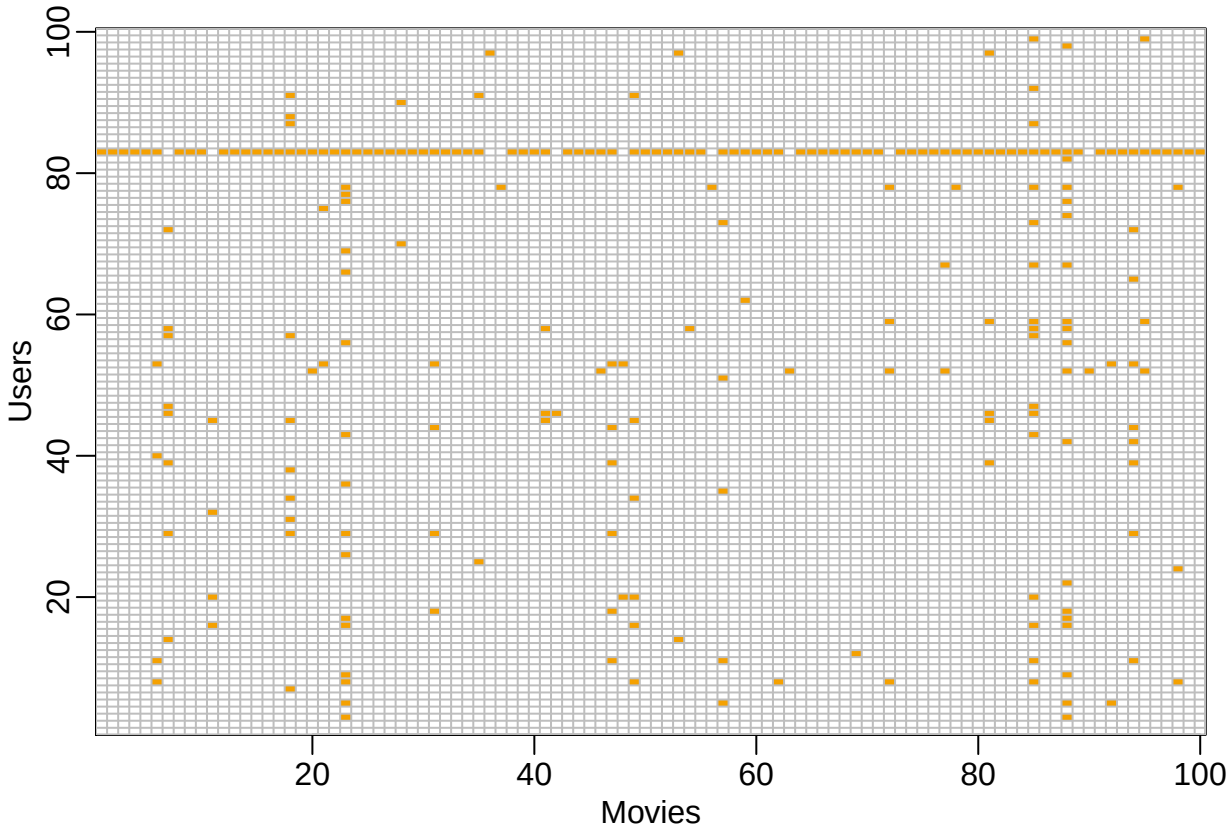
```
# Check for missing values
any(is.na(edx))
```

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

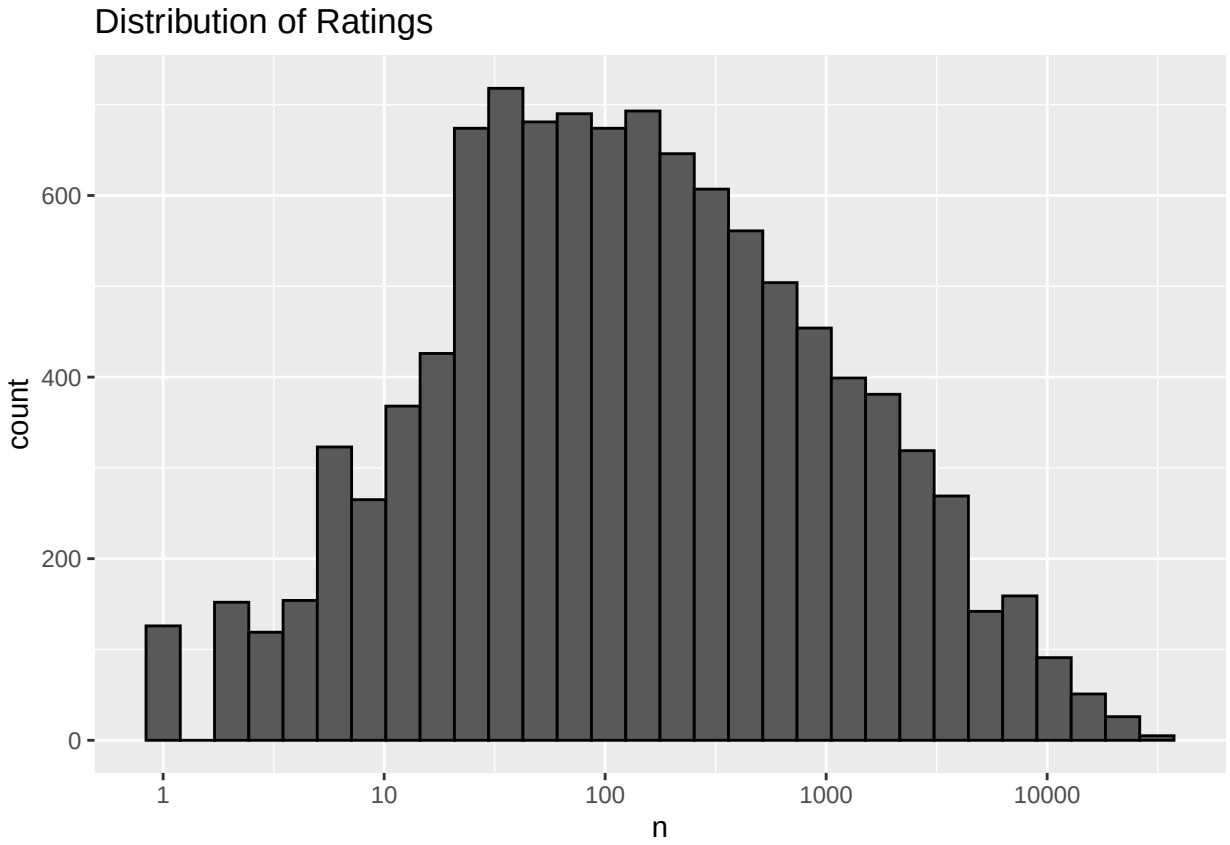
After splitting the original dataset into two sets, the number of unique users and movies will have lessened.

```
##   n_users n_movies
## 1   69878    10677
```

It is important to ensure that the set still varies enough to be useful. A randome sample of users and movies will do this.



Below, one can see that a fairly standard distribution is followed when considering the amount of ratings a film has.



The following films are the films with the most amount of ratings. One can see that they were primarily released between 1991 and 1995 - with the exception of Star Wars in 1997.

```
## # A tibble: 10,677 x 4
## # Groups:   movieId [10,677]
##    movieId title                                     rating_count avg_rating
##      <dbl> <chr>                                            <int>      <dbl>
##  1     296 Pulp Fiction (1994)                              31362       4.15
##  2     356 Forrest Gump (1994)                              31079       4.01
##  3     593 Silence of the Lambs, The (1991)                 30382       4.20
##  4     480 Jurassic Park (1993)                             29360       3.66
##  5     318 Shawshank Redemption, The (1994)                 28015       4.46
##  6     110 Braveheart (1995)                                26212       4.08
##  7     457 Fugitive, The (1993)                             25998       4.01
##  8     589 Terminator 2: Judgment Day (1991)                25984       3.93
##  9     260 Star Wars: Episode IV - A New Hope (a.k~         25672       4.22
## 10     150 Apollo 13 (1995)                                 24284       3.89
## # ... with 10,667 more rows
```

Below depicts the films that have the highest average rating. Films from classical Hollywood seem to be especially well-rated.

```
## # A tibble: 1,902 x 4
## # Groups:   movieId [1,902]
##    movieId title                                     rating_count avg_rating
##      <dbl> <chr>                                            <int>      <dbl>
```

```
##  1     318 Shawshank Redemption, The (1994)                28015       4.46
##  2     858 Godfather, The (1972)                           17747       4.42
##  3      50 Usual Suspects, The (1995)                      21648       4.37
##  4     527 Schindler's List (1993)                         23193       4.36
##  5     912 Casablanca (1942)                               11232       4.32
##  6     904 Rear Window (1954)                               7935       4.32
##  7     922 Sunset Blvd. (a.k.a. Sunset Boulevard) ~         2922       4.32
##  8    1212 Third Man, The (1949)                            2967       4.31
##  9    3435 Double Indemnity (1944)                          2154       4.31
## 10    1178 Paths of Glory (1957)                            1571       4.31
## # ... with 1,892 more rows
```

By looking at which ratings are most and least popular, one can see that the higher ratings are, on average, more popular than the lower ratings. Therefore, it can be inferred that users are more likely to rate the films that they enjoy than the ones that they do not. This is still helpful, however, as the goal is to recommend the users films that they enjoy, not inform them of films that they are likely to dislike. It is also notable that whole numbers, such as 4.0, 3.0 and 5.0, are distinctly more popular than fractions, such as 4.5, 3.5, etc.

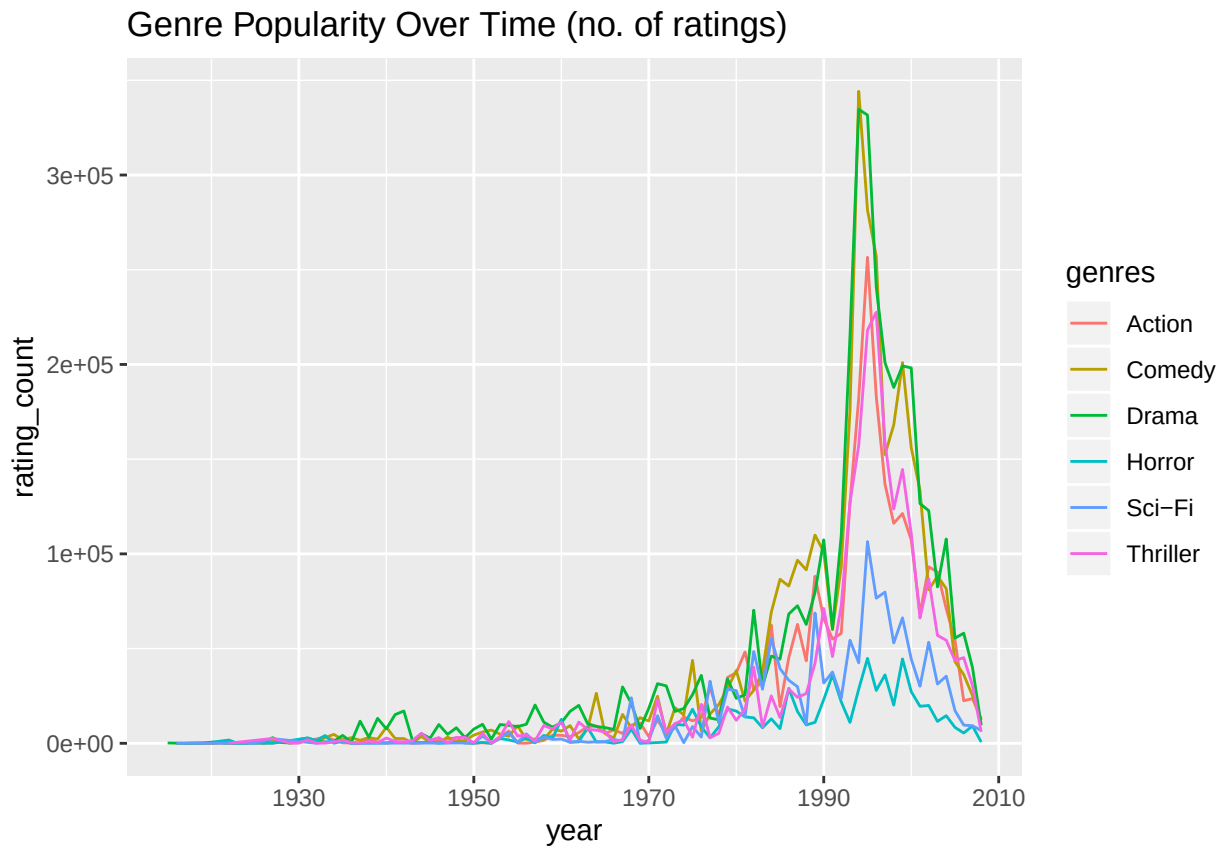
```
## # A tibble: 10 x 2
##    rating rating_count
##     <dbl>        <int>
##  1    4        2588430
##  2    3        2121240
##  3    5        1390114
##  4    3.5       791624
##  5    2         711422
##  6    4.5       526736
##  7    1         345679
##  8    2.5       333010
##  9    1.5       106426
## 10    0.5        85374
```

It is important to note that the two genres with significantly more ratings than the rest, Drama and Comedy, are both relatively broad genres that can be assigned to many films.

```
## # A tibble: 20 x 2
##    genres             rating_count
##    <chr>                     <int>
##  1 Drama                   3910127
##  2 Comedy                  3540930
##  3 Action                  2560545
##  4 Thriller                2325899
##  5 Adventure               1908892
##  6 Romance                 1712100
##  7 Sci-Fi                  1341183
##  8 Crime                   1327715
##  9 Fantasy                  925637
## 10 Children                 737994
## 11 Horror                   691485
## 12 Mystery                  568332
## 13 War                      511147
## 14 Animation                467168
## 15 Musical                  433080
```

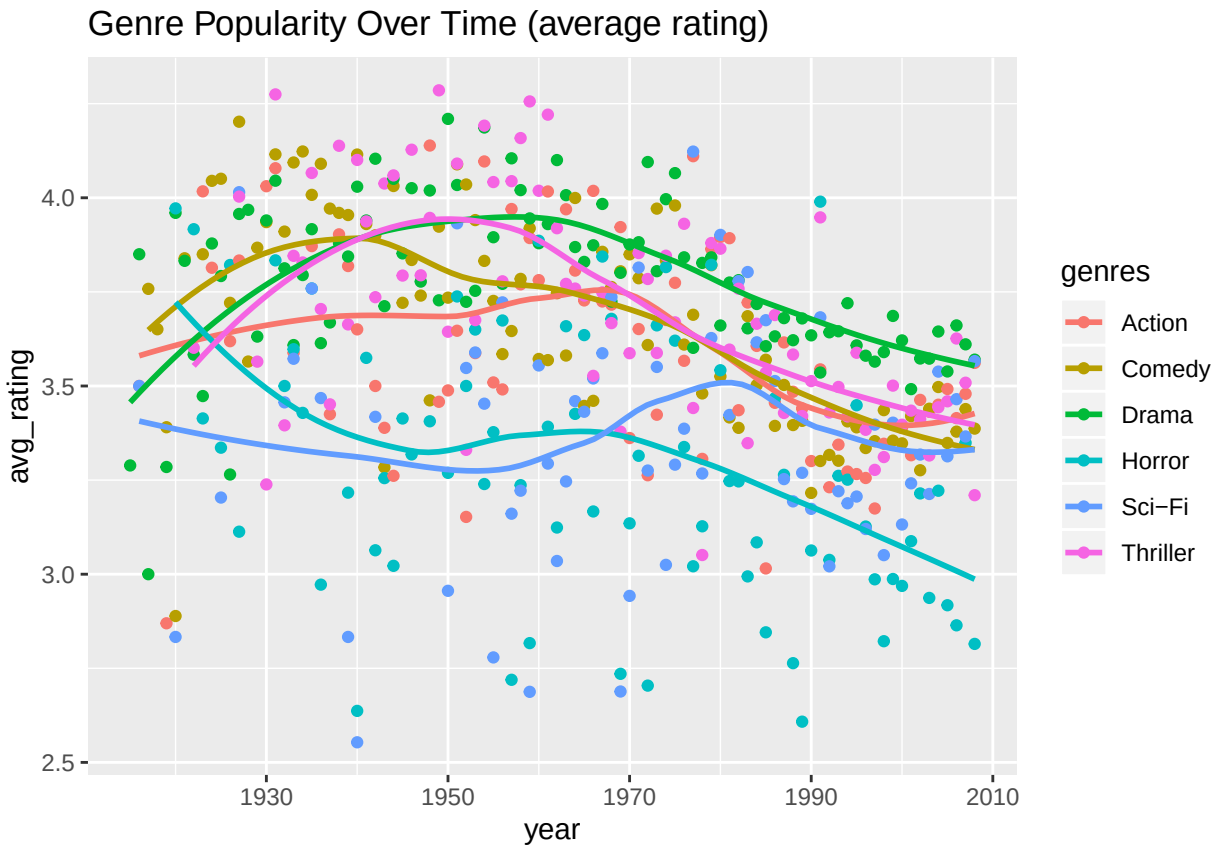
```
## 16 Western                189394
## 17 Film-Noir              118541
## 18 Documentary             93066
## 19 IMAX                     8181
## 20 (no genres listed)          7
```

The following two graphs show that a corellation exists between genre and time. A limited number of genres will be used in order to ensure that the graph is intelligable. The chosen genres are drama, comedy, action, horror, thriller & sci-fi.

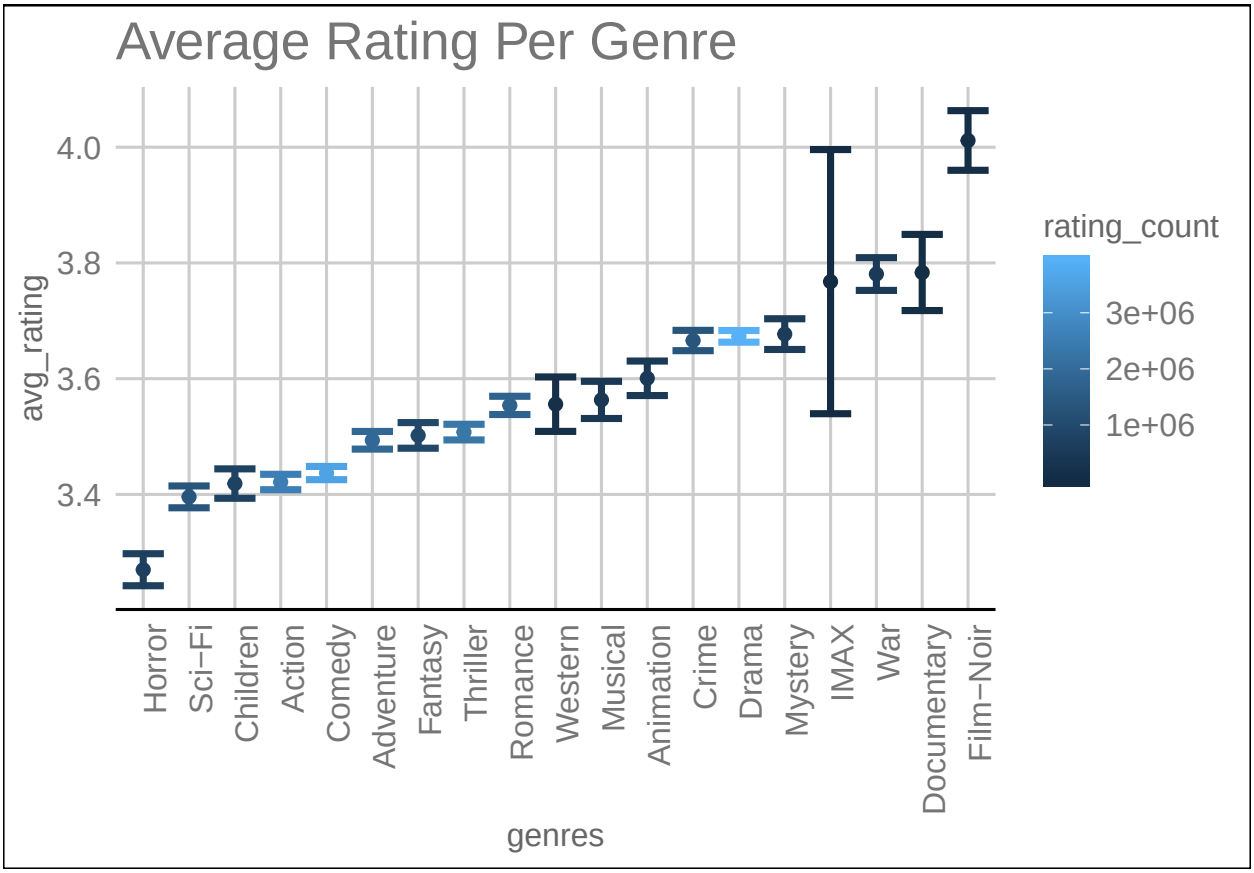


One can see that there are definite trends in genre popularity over time.

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

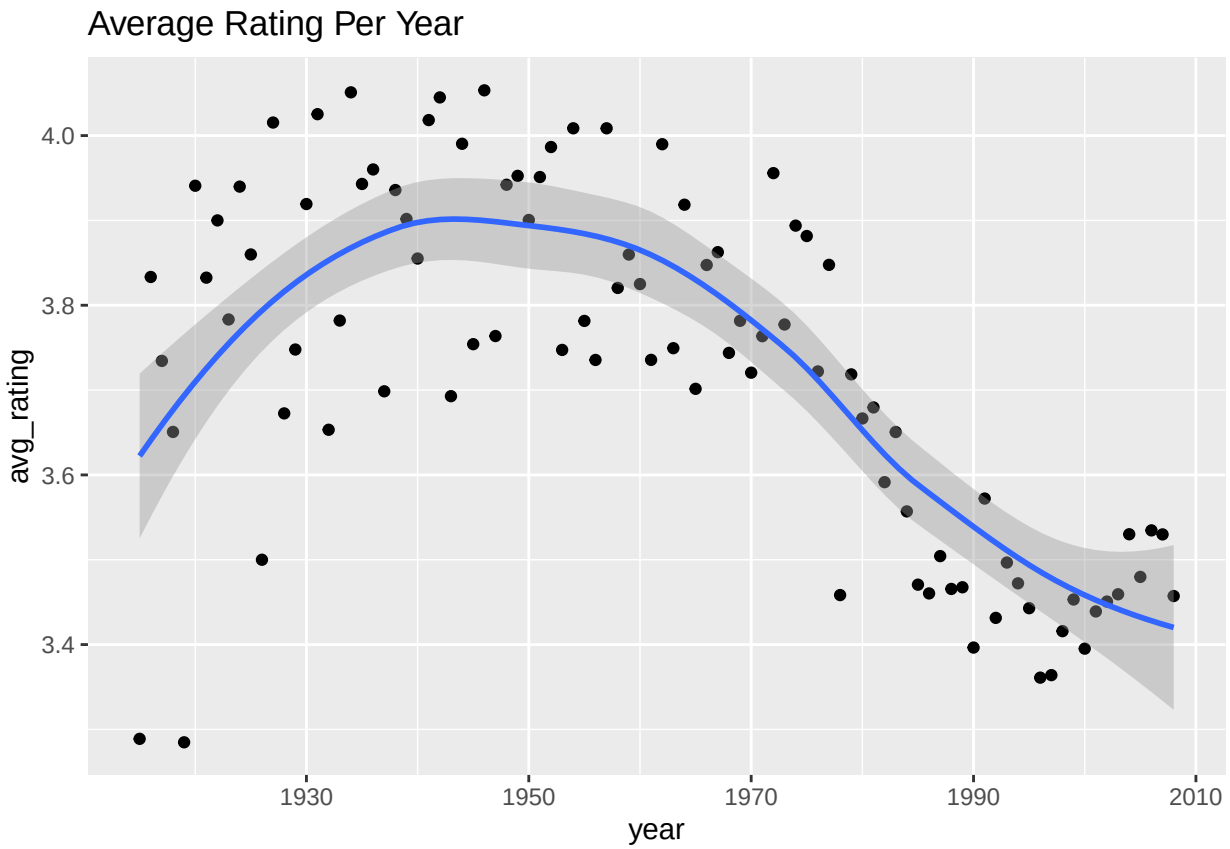


In the following two graphs, one can see the relationship that ratings have with genres and year, respectively.

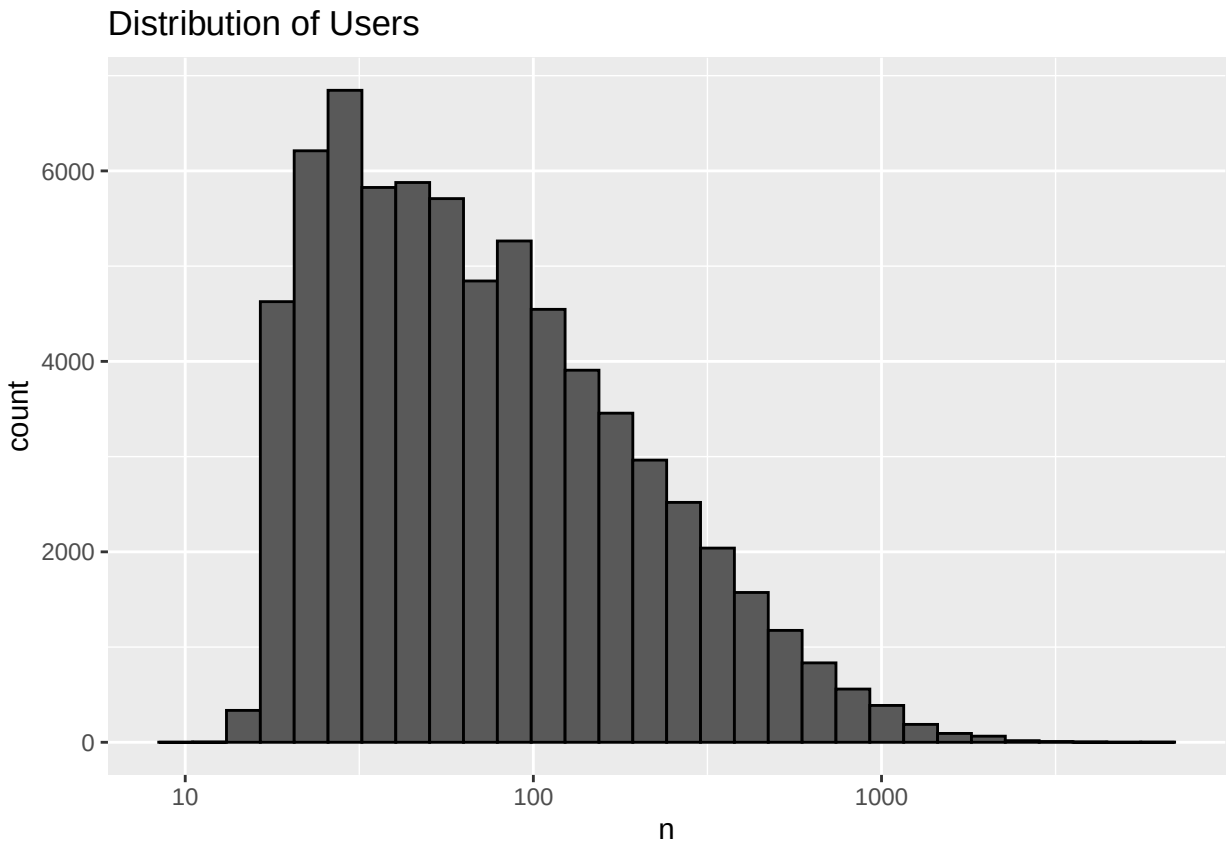


The results seen below reflect our previous findings that the average rating of classical Hollywood films tends to be particularly high. Furthermore, there is a definite correlation between the rating of a film and the time in which it was released.

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



The following graph is similar to the one shown earlier that ploted the disrtibution of movies that have been rated. The graph below depicts the amount of ratings that users tend to make - with most users rating approximately 50 movies. This should still be enough ratings to predict fairly accurately what films each user might enjoy.



The following depicts the users with the most amount of ratings. One can see that the average ratings given by the users that rate the most fits well with the numbers provided for the average ratings of films per year. Naturally, these individuals and their film tastes will have a greater impact on the average ratings per year. Considering each user individually, however, will still ensure that users with similar tastes will be recommended similar films.

```
## # A tibble: 24,115 x 3
##    userId rating_count avg_rating
##     <int>        <int>      <dbl>
##  1  59269         6616       3.26
##  2  67385         6360       3.20
##  3  14463         4648       2.40
##  4  68259         4036       3.58
##  5  27468         4023       3.83
##  6  19635         3771       3.50
##  7   3817         3733       3.11
##  8  63134         3371       3.27
##  9  58357         3361       3.00
## 10  27584         3142       3.00
## # ... with 24,105 more rows
```

Modelling

To begin the modelling phase, the edx data set is split into a training set and a test set. Much like the partitioning of the original edx data into the edx and validation set, 90% is assigned to edx_train and 10%

is assigned to edx_test. This is done so that the validation set is only used for the final testing.

```
# Create a train & test set from edx
edx_test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train <- edx[-edx_test_index,]
edx_test <- edx[edx_test_index,]
```

RMSE function

A function is then created through which the RMSE can be calculated. Additionally, the mu object is created as it is needed for the effects model that will be elaborated upon in the next section.

```
# Create a rmse function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2,na.rm = T))
}

# Calculate mu
mu <- mean(edx_train$rating)
```

The predicted RMSE, without the effects modelling is:

```
## [1] 1.05885
```

This is substantially higher than our objective RMSE: 0.86490.

Effects model

As aforementioned, an effects model is used here to create the recommendation system. This model has been used in the Harvard edX data science course, however extra effects have been incorporated. The model can be represented as

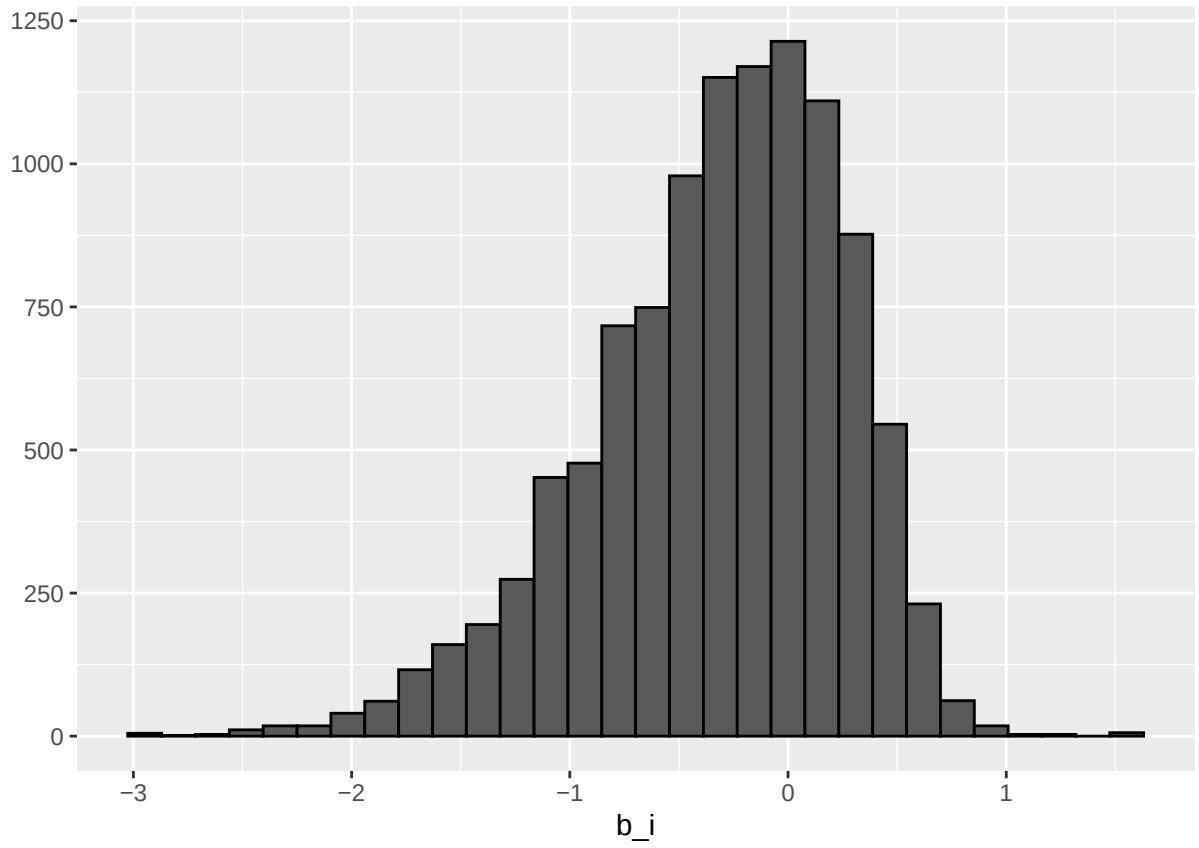
$$y_{i,u,g,t} = \mu + b_i + b_u + b_g + b_t + \epsilon_{i,u,g,t}$$

.

The movie effect - or item effect as it is referred to by contributers to solution that won the Netflix Grand Prize (Koren 2009: 1) - is created in the b_i object. This is the effect that considers how each film is rated.

```
# Movie effect
b_i <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)) # b_i = movie/item effect
qplot(b_i, geom = "histogram", col = I("black"), data = b_i)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The predicted RMSE using only the movie effect is:

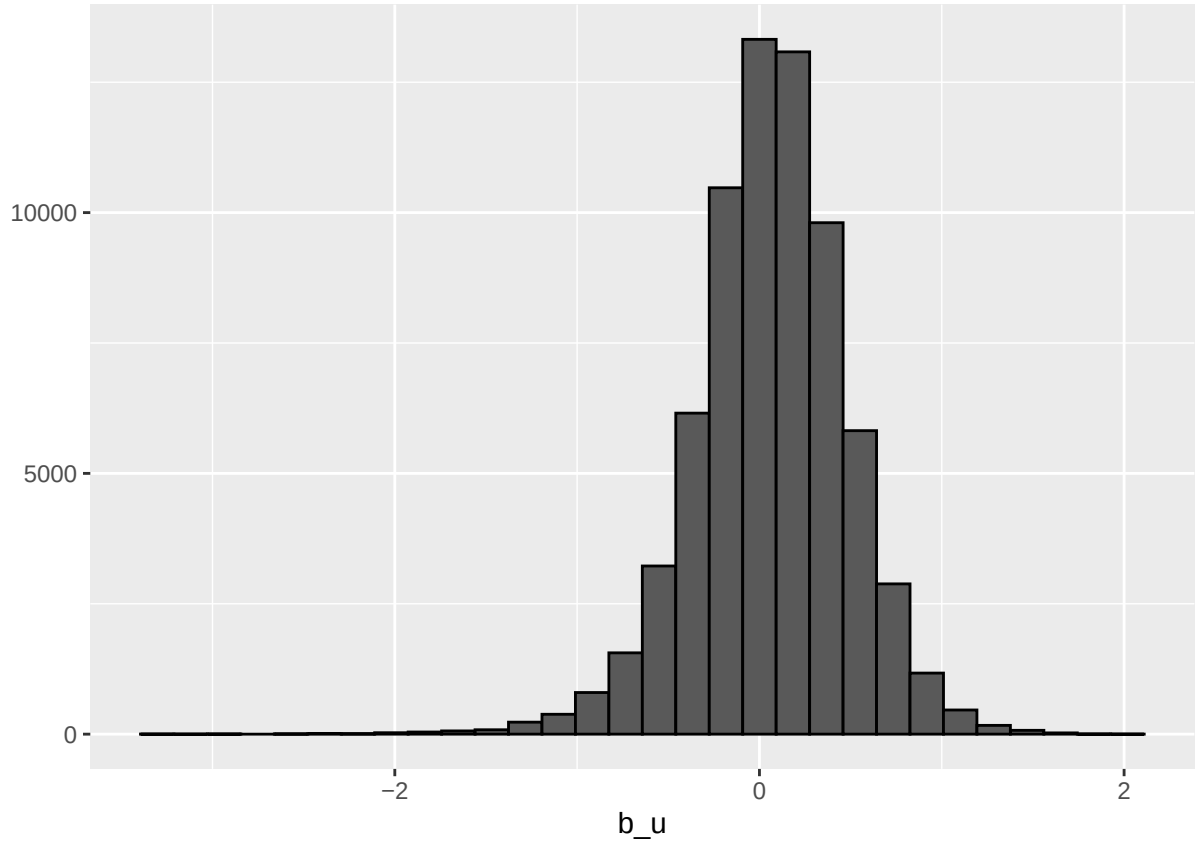
```
## [1] 0.9433419
```

Naturally, this is better than the predicted RMSE without any modelling, but it is still significantly higher than our objective.

The user effect is denoted as b_u. This effect considers how each user rates films and so would be able to pair users with similar tastes.

```
# User effect
b_u <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - b_i - mu)) # b_u = user effect
qplot(b_u, geom = "histogram", col = I("black"), data = b_u)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The predicted RMSE using only the user effect is:

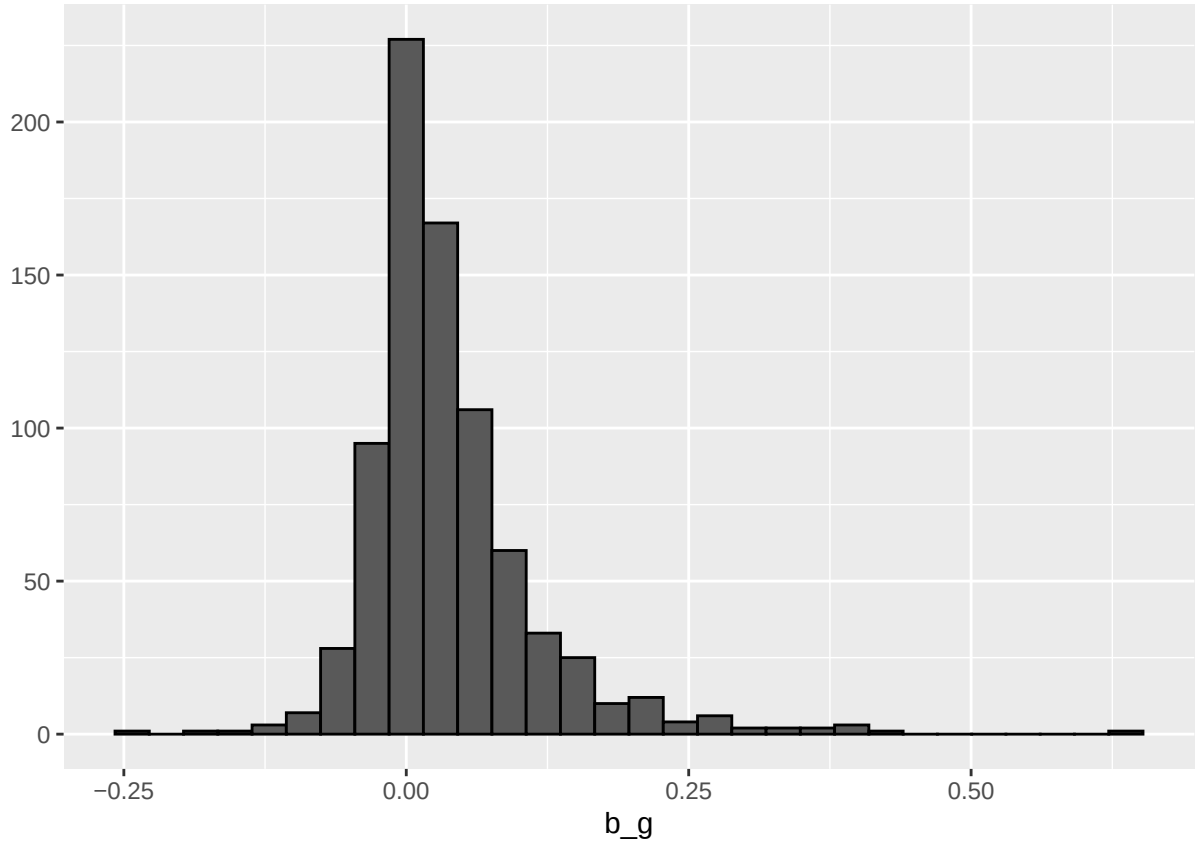
```
## [1] 0.9926056
```

It is clear that a single effect will not be suitable to attain our objective RMSE.

The genre effect is denoted as b_g.

```
# Genre effect
b_g <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarise(b_g = mean(rating - b_i - b_u - mu)) # b_g = genre effect
qplot(b_g, geom = "histogram", col = I("black"), data = b_g)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The predicted RMSE using only the genre effect is:

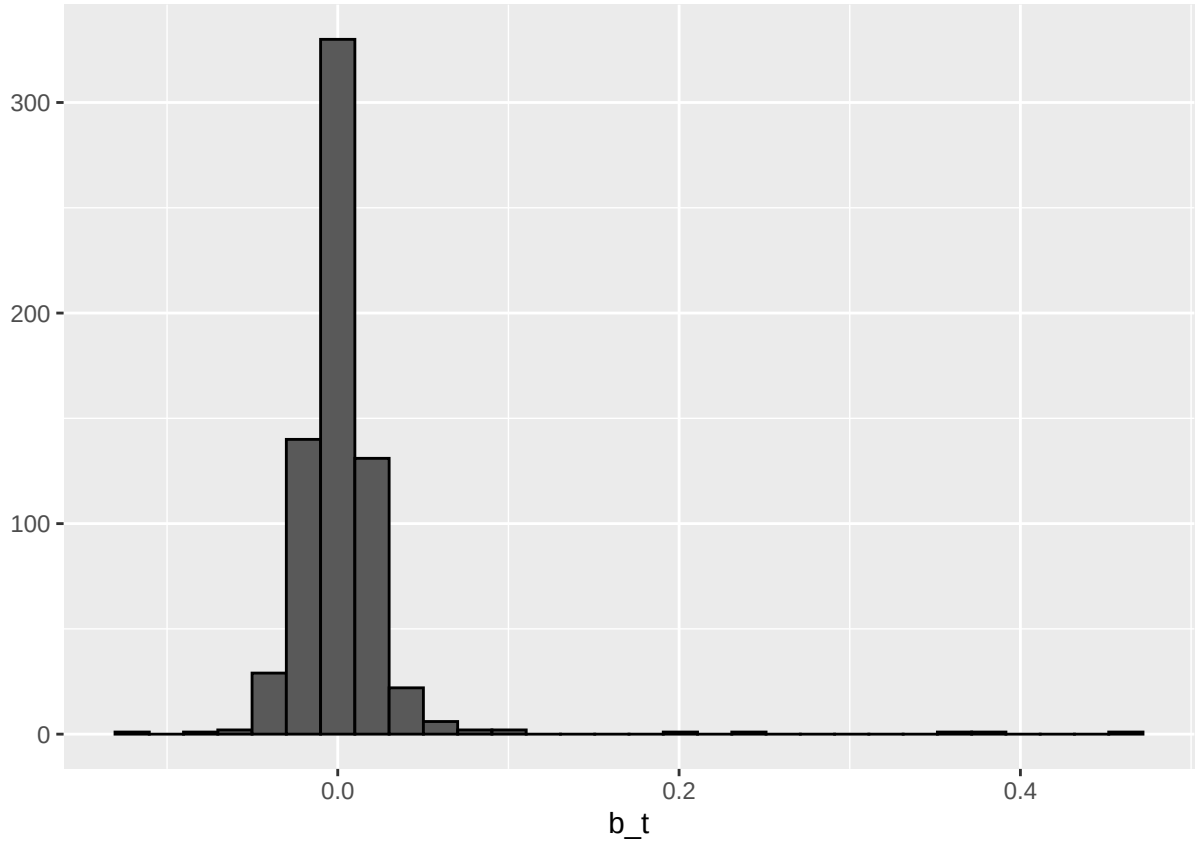
```
## [1] 1.056995
```

Surprisingly, this is barely an imporovement on the predicted RMSE without any modelling. Therefore, the genre effect by itself will have little effect on creating an adequite recommendation system. However, in combination with the other effects, considering genre will assist in optimisng the accuracy of our recommendation system. This is evident by considering the noticeable trends in genre in the data analysis section.

The time effect is denoted as b_t. This is the time in which the rating is made.

```
# Time effect
b_t <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  group_by(week) %>%
  summarize(b_t = mean(rating - b_i - b_u - b_g - mu)) # b_t = time effect
qplot(b_t, geom = "histogram", col = I("black"), data = b_t)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The predicted RMSE using only the time effect is:

```
## [1] 1.058593
```

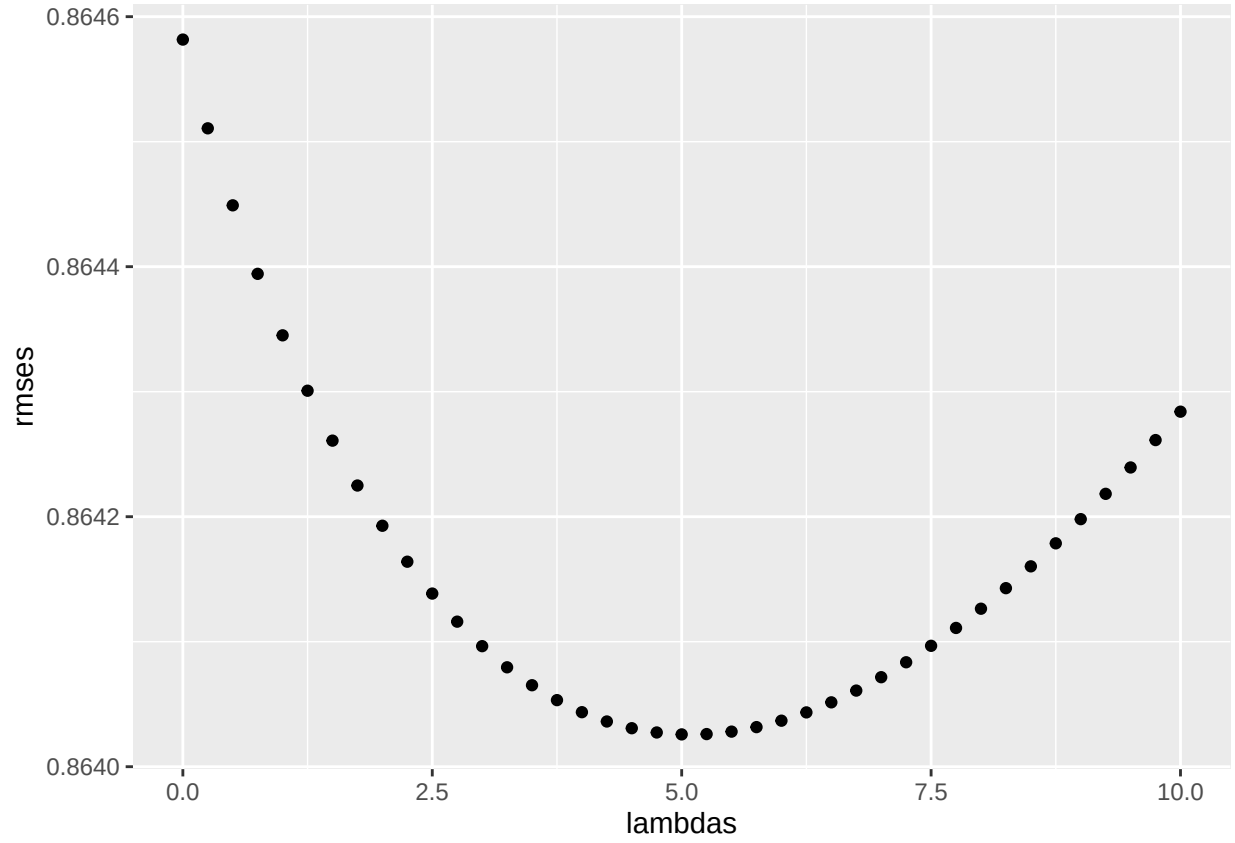
This is even higher than the genre effect. Nonetheless, as with genre, when combined with the other effects, this can only aid in optimising the recommendation system.

The next step is to optimise the tuning parameters for the final models. This is done by considering multiple lambdas - from 0 to 10 in intervals of 0.25 and finding which results in the minimum RMSE.

```
# Optimise tuning parameters
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))
  b_u <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n() + l))
  b_g <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarise(b_g = sum(rating - b_i - b_u - mu)/(n() + l))
  b_t <- edx_train %>%
```

```
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    group_by(week) %>%
    summarize(b_t = sum(rating - b_i - b_u - b_g - mu)/(n() + l))
  predicted_ratings <- edx_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_t, by = "week") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
    pull(pred)
  return(RMSE(predicted_ratings, edx_test$rating))
})
```

After plotting the RMSEs against the lambdas, one can see that the optimum lambda is somewhere around 5.0.



The minimum RMSE is:

```
## [1] 0.8640259
```

This makes the best lambda:

```
## [1] 5
```

In the final step of the modelling, this lambda is used to calculate the final models.

```
# Calculate final models with best_lambda
b_i <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n() + best_lambda))

b_u <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n() + best_lambda))

b_g <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarise(b_g = sum(rating - b_i - b_u - mu)/(n() + best_lambda))

b_t <- edx_train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  group_by(week) %>%
  summarize(b_t = sum(rating - b_i - b_u - b_g - mu)/(n() + best_lambda))
```

Results

Once the validation set has been updated to include the columns added to edx and the effects variables, one can calculate the final RMSE.

```
# Predict ratings for entire effects model
validation_effects_predict <- validation_effects %>%
  mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
  pull(pred)

# Final rmse
validation_effects_rmse <- RMSE(validation_effects_predict, validation$rating)
validation_effects_rmse
```

```
## [1] 0.8646867
```

Final Results

The final RMSE is:

```
## [1] 0.86469
```

This successfully achieves the ultimate goal of this assignment to create a model with an RMSE below 0.86490.

Conclusion

There are multiple limitations of this report and so many ways in which it could be improved upon. Firstly, more effects variables could be considered. This was limited by the amount of data within the original data set. The best way that this could be improved upon is by including information such as cast and directors/writers as many users will have favourite actors, directors, etc. whose careers they enjoy to follow. More specific genres could also be included, such as heist films or spy-thrillers. Furthermore, categories could be created to include more than genre. An example of this may be films that are popular amongst film critics - a category that exists in Netflix's current system. Additionally, a film's runtime might effect how well it is regarded by certain viewers. Moreover, effects modelling was prioritised in this assignment, but many other techniques may be used and even combined to create a more accurate recommendation system. Non theless, the effects model was outlined in the Netflix prize documentation that guided this assignment and succesfully achieved its ultimate goal.

Bibliography

Koren, Y., 2009. The Bellkor Solution to the Netflix Grand Prize. $NetflixPrizeDocumentation$, 81(2009), pp.1-10.