MovieLens Capstone Project

Intro

We will train and evaluate a movie rating algorithm based on the **10M version of the MovieLens dataset**. The algorithm will be developed using the `edx` data set. For a final test of the final algorithm, we will predict movie ratings for the `validation` data set (the final hold-out test set) as if they were unknown. We will use RMSE as the measure of our prediction accuracy.

We will go through the following steps when building and evaluating our model:

- Downloading and splitting the data to `edx` and `validation` data sets
  - Sampling the `edx` data into a smaller data set to ease computational burden
- Modifying the data:
  - Extracting new variables from existing columns
- Exploratory data analysis:
  - Splitting the smalller `edx` data set further to test and train data sets
  - Exploring the structure of the data with the train set with the aim of finding variables which can be used in prediction
- Building different models:
  - Building models with the training data and testing model performance with the test data set
  - Choosing the final model
- Re-training the final model with combined training and test data from `edx`
- Final model assessment:
  - Evaluating the performance of the final model with the `validation` data set.

Downloading data and creating training and validation datasets

We can download and split the data as shown on the course website. However, as downloading the data over and over again during model development is quite inconvenient, a local copy of the data has been made in the form of an Rds-file.

```
###### Original script for getting the data is shown below --------
# Please uncomment if you need to run this file and don't have a local copy of the data

# ############################################################
# # Create edx set, validation set (final hold-out test set)
# ############################################################
```

```
#
# # Note: this process could take a couple of minutes
#
# if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
# if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
# if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
#
# library(tidyverse)
# library(caret)
# library(data.table)
#
# # 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")
#
# # if using R 3.6 or earlier:
# # movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
# #                                            title = as.character(title),
# #                                            genres = as.character(genres))
# # if using R 4.0 or later:
# 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")
#
# # 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)`
# 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)
# edx <- rbind(edx, removed)
#
# rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

```
#### Saving the edx and validation objects to the local project working directory -----------------

### Save edx and validation as Rdata structures ####
#
# saveRDS(object = edx, file = "MovielensData/edx.Rds")
# saveRDS(object = validation, file = "MovielensData/validation.Rds")
```

Getting the training data from local project directory

Data which was saved as R data structure (Rds) files, can be downloaded as follows:

```
## Import training data set from local folder
dataset <- readRDS(file = paste0("MovielensData/", params$data))

#Dimensions
dim(dataset)
```

```
## [1] 9000055       6
```

```
#Head of the data set
dataset %>% head()
```

```
##    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
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

```
#How many users and movies are there?
dataset %>%
  summarise(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

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

The data consists of 6 columns with the following names: userId, movieId, rating, timestamp, title, genres. The `genres` column seems to contain multiple genres divided by a vertical line symbol: `|`. The `title` column seems to also contain the year in which the movie was released.

Taking a subset of the edx data

The edx data set is simply too large for the analysis to run on my standard laptop. Let's therefore only use a subset of the data for training the model.

```
#Setting the seed
set.seed(32)

# Amount of observation to be sampled
N <- 2000000
#Index of sampled observations
sample_index <- sample(x = 1:length(dataset$rating), size = N, replace = FALSE)
#Subset data
dataset <- dataset[sample_index,]

dim(dataset)
```

```
## [1] 2000000       6
```

We will use 2×10^6 observations from the edx data for exploring different models.

Modifying the data before analysis

Let's modify the data before diving in deeper. We will try to extract the individual movie genres as their own columns, as well as the year of movie release from the title.

Separating genres

Let's start by separating the individual genres into different columns. We will do this by replacing the genres column with columns named as genre1...genre8. That is to say, we will list up to eight separate genres for any given movie.

```
## a function which takes a data.frame and separates genres-column based on vertical bar symbol "|"
separate_genres <- function(df){
  df <- df %>%
    separate(col = genres,
             into = paste0("genre", 1:8),
             sep = "\\|",
             extra = "warn",
             fill = "right") %>%
    mutate(across(matches("genre"),
                  function(x){if_else(is.na(x), "(no genres listed)", x)})
           ) # Turn NA into "(no genres listed)"
  return(df)
}

# separate the genres for the part of edx we use for data exploration
dataset <- separate_genres(dataset)

#Let's see what the data looks like
head(dataset)
```

```
##    userId movieId rating  timestamp
## 1:  66124    4896    4.5 1212945704
## 2:   5917    1189    4.0 1017031514
## 3:  53310    1562    3.0  866903215
## 4:  59741      34    5.0  843219144
## 5:  20370   40278    3.0 1222393777
## 6:  37999     432    3.0  836767477
##                                                                                                 title
## 1: Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001)
## 2:                                                                                Thin Blue Line, The (1988)
## 3:                                                                                     Batman & Robin (1997)
## 4:                                                                                               Babe (1995)
## 5:                                                                                            Jarhead (2005)
## 6:                                                                 City Slickers II: The Legend of Curly's Gold (1994)
##         genre1             genre2             genre3             genre4
## 1:   Adventure           Children            Fantasy (no genres listed)
## 2: Documentary (no genres listed) (no genres listed) (no genres listed)
## 3:      Action          Adventure            Fantasy           Thriller
## 4:    Children             Comedy              Drama            Fantasy
## 5:      Action              Drama                War (no genres listed)
## 6:   Adventure             Comedy            Western (no genres listed)
##                 genre5             genre6             genre7             genre8
## 1: (no genres listed) (no genres listed) (no genres listed) (no genres listed)
## 2: (no genres listed) (no genres listed) (no genres listed) (no genres listed)
## 3: (no genres listed) (no genres listed) (no genres listed) (no genres listed)
## 4: (no genres listed) (no genres listed) (no genres listed) (no genres listed)
## 5: (no genres listed) (no genres listed) (no genres listed) (no genres listed)
## 6: (no genres listed) (no genres listed) (no genres listed) (no genres listed)
```

Getting the individual genre names

Most of the columns in `genre1` to `genre8` seem to have the value `(no genres listed)`. Let's try to extract all unique values from these columns.

```
### Extracting individual genres -----------------------

genre_df <- dataset %>%
  select(-userId, -movieId, -rating, -timestamp, -title) %>% ## Leave individual genre columns
  pivot_longer(cols = everything(),
               names_to = "genre_col",
               values_to = "genre") # names "genre1", "genre2", values "Adventure" etc.

#get unique values for individual genres
indiv_genres <- unique(genre_df$genre)

#Let's remove the "genre_df" data.frame as do not need it anymore
rm("genre_df")
```

Individual genres have the following values: Adventure, Children, Fantasy, (no genres listed), Documentary, Action, Thriller, Comedy, Drama, War, Western, Romance, Sci-Fi, Horror, Mystery, Crime, Film-Noir, Musical, Animation, IMAX.

Genres as individual columns

Let's adjust the data further, so that each individual genre get's a column. We will set the value for an individual genre as `1` if a given observation falls into that particular genre and `0` otherwise.

```
# A function which determines in which genres a given movie belongs to
count_genres <- function(df){
  ### Let's create columns for all individual genres --------------------------
  ## Setting value as 1 if genre present, 0 otherwise

  #Ugly as hell but works
  #Could not think of a more elegant way to achieve this
  df <- df %>% # set values as 1 if genre name found from genre1-genre8-columns
    mutate(
      #Adventure
      Adventure = if_else((genre1 == "Adventure"|genre2 == "Adventure"|genre3 == "Adventure"|
                             genre4 == "Adventure"|genre5 == "Adventure"|genre6 == "Adventure"|
                             genre7 == "Adventure"|genre8 == "Adventure"),
                          true = 1, false = 0),
      #Children
      Children =  if_else((genre1 == "Children"|genre2 == "Children"|genre3 == "Children"|
                             genre4 == "Children"|genre5 == "Children"|genre6 == "Children"|
                             genre7 == "Children"|genre8 == "Children"),
                          true = 1, false = 0),
      #Fantasy
      Fantasy =  if_else((genre1 == "Fantasy"|genre2 == "Fantasy"|genre3 == "Fantasy"|
                            genre4 == "Fantasy"|genre5 == "Fantasy"|genre6 == "Fantasy"|
                            genre7 == "Fantasy"|genre8 == "Fantasy"),
                         true = 1, false = 0),
      #(no genres listed)
      `(no genres listed)` =  if_else((genre1 == "(no genres listed)"|genre2 == "(no genres listed)"|ge
                                         genre4 == "(no genres listed)"|genre5 == "(no genres listed)"|
                                         genre7 == "(no genres listed)"|genre8 == "(no genres listed)")
                                      true = 1, false = 0),
      #Documentary
      Documentary =  if_else((genre1 == "Documentary"|genre2 == "Documentary"|genre3 == "Documentary"|
                                genre4 == "Documentary"|genre5 == "Documentary"|genre6 == "Documentary"
                                genre7 == "Documentary"|genre8 == "Documentary"),
                             true = 1, false = 0),
      #Action
      Action =  if_else((genre1 == "Action"|genre2 == "Action"|genre3 == "Action"|
                           genre4 == "Action"|genre5 == "Action"|genre6 == "Action"|
                           genre7 == "Action"|genre8 == "Action"),
                        true = 1, false = 0),
      #Thriller
      Thriller =  if_else((genre1 == "Thriller"|genre2 == "Thriller"|genre3 == "Thriller"|
                             genre4 == "Thriller"|genre5 == "Thriller"|genre6 == "Thriller"|
                             genre7 == "Thriller"|genre8 == "Thriller"),
                          true = 1, false = 0),
      #Comedy
      Comedy =  if_else((genre1 == "Comedy"|genre2 == "Comedy"|genre3 == "Comedy"|
                           genre4 == "Comedy"|genre5 == "Comedy"|genre6 == "Comedy"|
                           genre7 == "Comedy"|genre8 == "Comedy"),
                        true = 1, false = 0),
```

```
#Drama
Drama =  if_else((genre1 == "Drama"|genre2 == "Drama"|genre3 == "Drama"|
                   genre4 == "Drama"|genre5 == "Drama"|genre6 == "Drama"|
                   genre7 == "Drama"|genre8 == "Drama"),
                 true = 1, false = 0),
#War
War =  if_else((genre1 == "War"|genre2 == "War"|genre3 == "War"|
                  genre4 == "War"|genre5 == "War"|genre6 == "War"|
                  genre7 == "War"|genre8 == "War"),
                true = 1, false = 0),
#Western
Western =  if_else((genre1 == "Western"|genre2 == "Western"|genre3 == "Western"|
                     genre4 == "Western"|genre5 == "Western"|genre6 == "Western"|
                     genre7 == "Western"|genre8 == "Western"),
                   true = 1, false = 0),
#Romance
Romance =  if_else((genre1 == "Romance"|genre2 == "Romance"|genre3 == "Romance"|
                     genre4 == "Romance"|genre5 == "Romance"|genre6 == "Romance"|
                     genre7 == "Romance"|genre8 == "Romance"),
                   true = 1, false = 0),
#Sci-Fi
`Sci-Fi` =  if_else((genre1 == "Sci-Fi"|genre2 == "Sci-Fi"|genre3 == "Sci-Fi"|
                      genre4 == "Sci-Fi"|genre5 == "Sci-Fi"|genre6 == "Sci-Fi"|
                      genre7 == "Sci-Fi"|genre8 == "Sci-Fi"),
                    true = 1, false = 0),
#Horror
Horror =  if_else((genre1 == "Horror"|genre2 == "Horror"|genre3 == "Horror"|
                    genre4 == "Horror"|genre5 == "Horror"|genre6 == "Horror"|
                    genre7 == "Horror"|genre8 == "Horror"),
                  true = 1, false = 0),
#Mystery
Mystery =  if_else((genre1 == "Mystery"|genre2 == "Mystery"|genre3 == "Mystery"|
                     genre4 == "Mystery"|genre5 == "Mystery"|genre6 == "Mystery"|
                     genre7 == "Mystery"|genre8 == "Mystery"),
                   true = 1, false = 0),
#Crime
Crime =  if_else((genre1 == "Crime"|genre2 == "Crime"|genre3 == "Crime"|
                   genre4 == "Crime"|genre5 == "Crime"|genre6 == "Crime"|
                   genre7 == "Crime"|genre8 == "Crime"),
                 true = 1, false = 0),
#Film-Noir
`Film-Noir` =  if_else((genre1 == "Film-Noir"|genre2 == "Film-Noir"|genre3 == "Film-Noir"|
                         genre4 == "Film-Noir"|genre5 == "Film-Noir"|genre6 == "Film-Noir"|
                         genre7 == "Film-Noir"|genre8 == "Film-Noir"),
                       true = 1, false = 0),
#Musical
Musical =  if_else((genre1 == "Musical"|genre2 == "Musical"|genre3 == "Musical"|
                     genre4 == "Musical"|genre5 == "Musical"|genre6 == "Musical"|
                     genre7 == "Musical"|genre8 == "Musical"),
                   true = 1, false = 0),
#Animation
Animation =  if_else((genre1 == "Animation"|genre2 == "Animation"|genre3 == "Animation"|
                       genre4 == "Animation"|genre5 == "Animation"|genre6 == "Animation"|
```

```
                              genre7 == "Animation"|genre8 == "Animation"),
                            true = 1, false = 0),
      #IMAX
      IMAX =  if_else((genre1 == "IMAX"|genre2 == "IMAX"|genre3 == "IMAX"|
                         genre4 == "IMAX"|genre5 == "IMAX"|genre6 == "IMAX"|
                         genre7 == "IMAX"|genre8 == "IMAX"),
                      true = 1, false = 0)
    ) %>%
    select(-contains(as.character(1:8))) %>% # Lets get rid of genre1...genre8-columns
    janitor::clean_names()  #Let's clean column names

  return(df)
}


### Modified dataframe ------------------------

dataset <- count_genres(dataset)
#Inspecting the dataset
head(dataset)
```

```
##    user_id movie_id rating  timestamp
## 1:   66124     4896    4.5 1212945704
## 2:    5917     1189    4.0 1017031514
## 3:   53310     1562    3.0  866903215
## 4:   59741       34    5.0  843219144
## 5:   20370    40278    3.0 1222393777
## 6:   37999      432    3.0  836767477
##                                                                                title
## 1: Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001)
## 2:                                                                  Thin Blue Line, The (1988)
## 3:                                                                       Batman & Robin (1997)
## 4:                                                                                Babe (1995)
## 5:                                                                             Jarhead (2005)
## 6:                                                  City Slickers II: The Legend of Curly's Gold (1994)
##    adventure children fantasy no_genres_listed documentary action thriller
## 1:         1        1       1                1           0      0        0
## 2:         0        0       0                1           1      0        0
## 3:         1        0       1                1           0      1        1
## 4:         0        1       1                1           0      0        0
## 5:         0        0       0                1           0      1        0
## 6:         1        0       0                1           0      0        0
##    comedy drama war western romance sci_fi horror mystery crime film_noir
## 1:      0     0   0       0       0      0      0       0     0         0
## 2:      0     0   0       0       0      0      0       0     0         0
## 3:      0     0   0       0       0      0      0       0     0         0
## 4:      1     1   0       0       0      0      0       0     0         0
## 5:      0     1   1       0       0      0      0       0     0         0
## 6:      1     0   0       1       0      0      0       0     0         0
##    musical animation imax
## 1:       0         0    0
## 2:       0         0    0
## 3:       0         0    0
```

```
## 4:       0         0    0
## 5:       0         0    0
## 6:       0         0    0
```

Extracting year from the title

Let's extract the year of release from the title.

```
## A function for extracting the movie release year from the title column -----------------
extract_year <- function(df){

  df <- df %>%
    mutate(year = str_extract(string = title, pattern = "\\(\\d{4}\\)")) %>% #extract year in brackets
    mutate(year = gsub(pattern = "\\(", replacement = "", x = year)) %>% # remove brackets
    mutate(year = gsub(pattern = "\\)", replacement = "", x = year)) %>% # remove brackets
    mutate(year = as.numeric(year)) #make numeric

  return(df)
}

dataset <- extract_year(dataset)
#Let's see if the function works
dataset %>% select(title, year) %>% head()
```

```
##                                                                                 title
## 1: Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001)
## 2:                                                                  Thin Blue Line, The (1988)
## 3:                                                                       Batman & Robin (1997)
## 4:                                                                                 Babe (1995)
## 5:                                                                              Jarhead (2005)
## 6:                                                  City Slickers II: The Legend of Curly's Gold (1994)
##    year
## 1: 2001
## 2: 1988
## 3: 1997
## 4: 1995
## 5: 2005
## 6: 1994
```

```
#Range of movie release years
dataset %>% pull(year) %>% range()
```

```
## [1] 1915 2008
```

Exploratory data analysis

Before diving into exploratory data analysis, let's split the subset of `edx` data further into training and test sets. We will explore and train our models with the training data and use the test set for testing model performance. The main purpose of exploratory data analysis is to gain ideas for model building.

Splitting the data

We split the data in 80:20 proportion into train and test sets.

```
# Index for data partitioning
set.seed(32)
trainIndex <- createDataPartition(dataset$rating,
                                  p = 0.8,
                                  list = FALSE,
                                  times = 1)

#Train test split
dataset_train <- dataset[trainIndex,]
dataset_test <- dataset[-trainIndex,]


#Making sure test set does not include users and movies not available in the train set
dataset_test <- dataset_test %>%
  semi_join(dataset_train, by = "movie_id") %>%
  semi_join(dataset_train, by = "user_id")

# How large is our train data
dim(dataset_train)
```

```
## [1] 1600001      26
```

```
# How large is our test data
dim(dataset_test)
```

```
## [1] 399637     26
```

Let's start exploring the data using the train set.

Effect of genre

Let's explore the effect of genre on movie rating.

```
indiv_genres <- names(dataset)[-(1:5)] #The names for individual genres
# Let's get rid of the newly created "year" column
(indiv_genres <- indiv_genres[indiv_genres != "year"])
```

```
##  [1] "adventure"        "children"         "fantasy"          "no_genres_listed"
##  [5] "documentary"      "action"           "thriller"         "comedy"
##  [9] "drama"            "war"              "western"          "romance"
## [13] "sci_fi"           "horror"           "mystery"          "crime"
## [17] "film_noir"        "musical"          "animation"        "imax"
```

```
# Let's summarise mean and standard deviation of ratings
# as well as number of observations for all genre columns
dataset_train %>%
  pivot_longer(cols = all_of(indiv_genres),
```

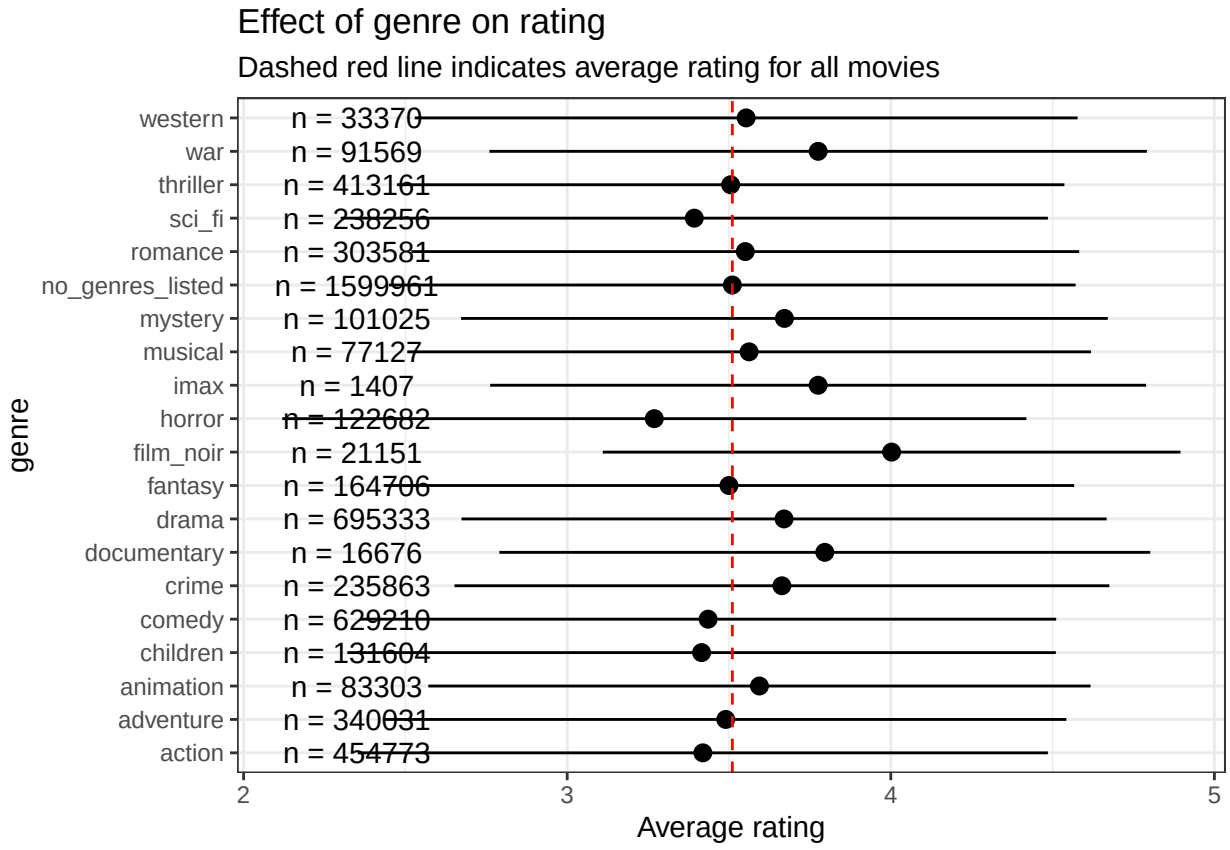
```
              names_to = "genre",
              values_to = "value") %>%
  group_by(genre, value) %>%
  summarise(avg = mean(rating), sd = sd(rating), n = n())
```

```
## `summarise()` has grouped output by 'genre'. You can override using the `.groups` argument.
```

```
## # A tibble: 40 x 5
## # Groups:   genre [20]
##    genre     value   avg    sd       n
##    <chr>     <dbl> <dbl> <dbl>   <int>
##  1 action        0  3.55  1.06 1145228
##  2 action        1  3.42  1.07  454773
##  3 adventure     0  3.52  1.06 1259970
##  4 adventure     1  3.49  1.05  340031
##  5 animation     0  3.51  1.06 1516698
##  6 animation     1  3.59  1.02   83303
##  7 children      0  3.52  1.06 1468397
##  8 children      1  3.42  1.09  131604
##  9 comedy        0  3.56  1.05  970791
## 10 comedy        1  3.44  1.08  629210
## # ... with 30 more rows
```

```
#Average rating for all movies
average_rating <- mean(dataset_train$rating)

# Plot average rating for individual genres
dataset_train %>%
  pivot_longer(cols = all_of(indiv_genres),
              names_to = "genre",
              values_to = "value") %>%
  group_by(genre, value) %>%
  summarise(avg = mean(rating), sd = sd(rating), n = n()) %>%
  filter(value == 1) %>%
  ggplot(aes(x = genre, y = avg)) +
  geom_pointrange(aes(ymin = (avg - sd), ymax = (avg+sd))) +
  geom_hline(yintercept = average_rating, lty = 2, color = "red") +
  geom_text(aes( y = 2.35, label = paste("n =", n))) +
  coord_flip() +
  ggtitle("Effect of genre on rating",
          subtitle = "Dashed red line indicates average rating for all movies") +
  labs(y = "Average rating")
```

```
## `summarise()` has grouped output by 'genre'. You can override using the `.groups` argument.
```



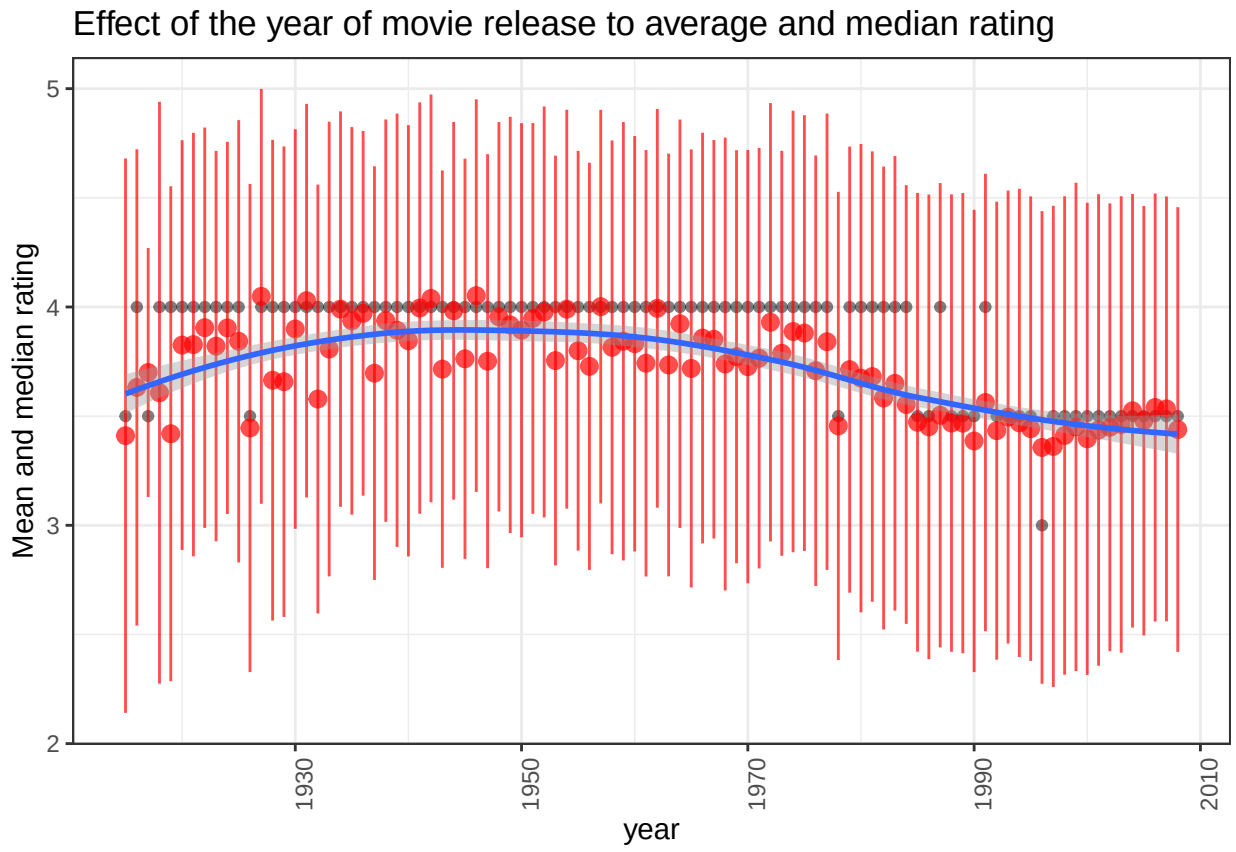
The genre of the movie seems to have an effect and should be taken into consideration.

Effect of year of release

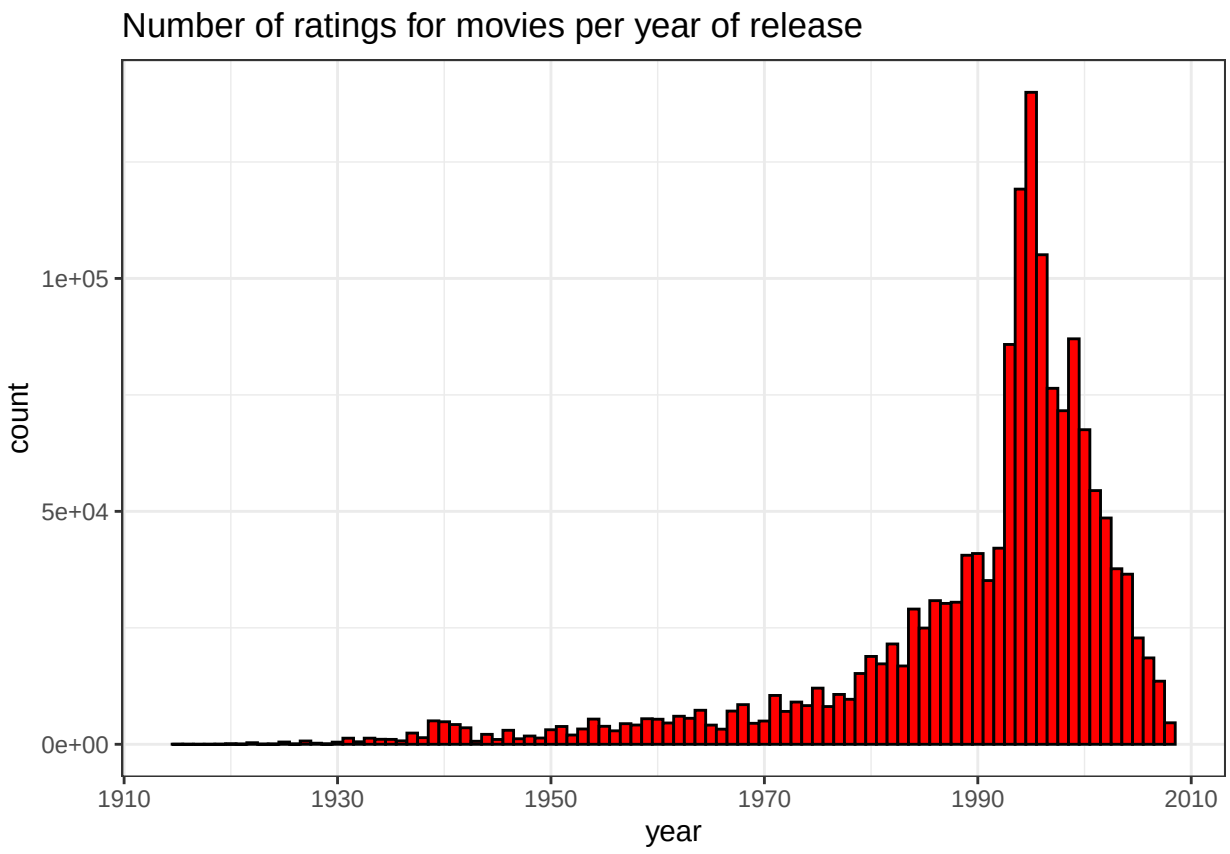
Let's examine if there is a pattern between average movie ratings and the year the movie was released.

```
#Does the release year have an effect on the rating distribution
dataset_train %>%
  group_by(year) %>%
  summarise(median = median(rating),
            mean = mean(rating),
            sd = sd(rating)) %>%
  ggplot(aes(x = year, y = mean)) +
  geom_point(aes(y = median), alpha = 0.5) +
  geom_pointrange(aes(ymin = (mean - sd), ymax = (mean + sd)), color = "red", alpha = 0.7) +
  geom_smooth(method = "loess") +
  theme(axis.text.x = element_text(angle = 90)) +
  labs(y = "Mean and median rating") +
  ggtitle("Effect of the year of movie release to average and median rating")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
#ratings per year of release
dataset_train %>% ggplot(aes(x= year)) +
  geom_histogram(color = "black", fill = "red", binwidth = 1) +
  ggtitle("Number of ratings for movies per year of release")
```



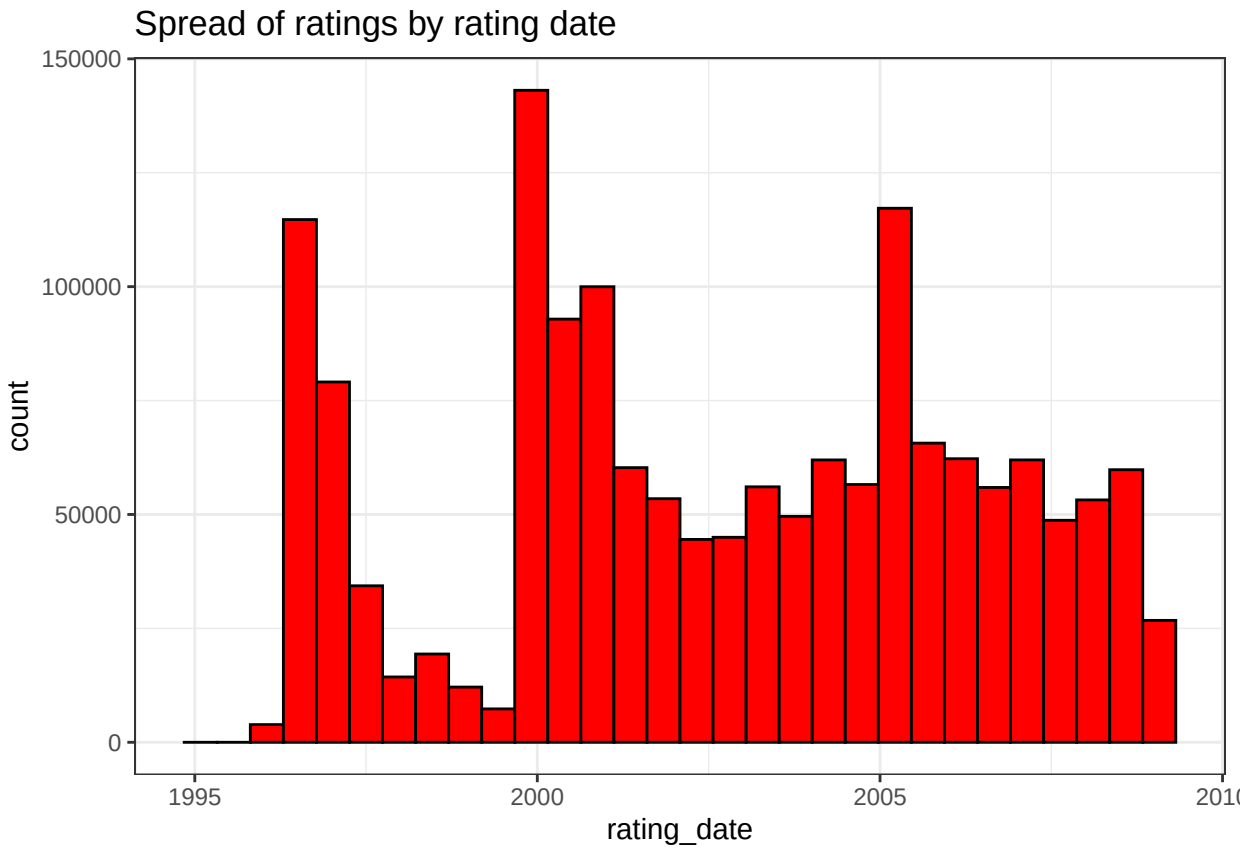
There seems to be an effect. Also the amount of movies varies greatly per year, which suggests that regularization might be in order to avoid placing too much weight to items with only few observations.

Effect of rating time

The data also contains a `timestamp` column. We can use this column to evaluate if the time when a movie was rated has an effect.

```
# How do rating dates spread out
dataset_train %>%  mutate(rating_date = as_datetime(timestamp)) %>%
   mutate(rating_date = round_date(rating_date, unit ="day")) %>%
  ggplot(aes(x = rating_date)) +
  geom_histogram(color = "black", fill = "red") +
  ggtitle("Spread of ratings by rating date")
```

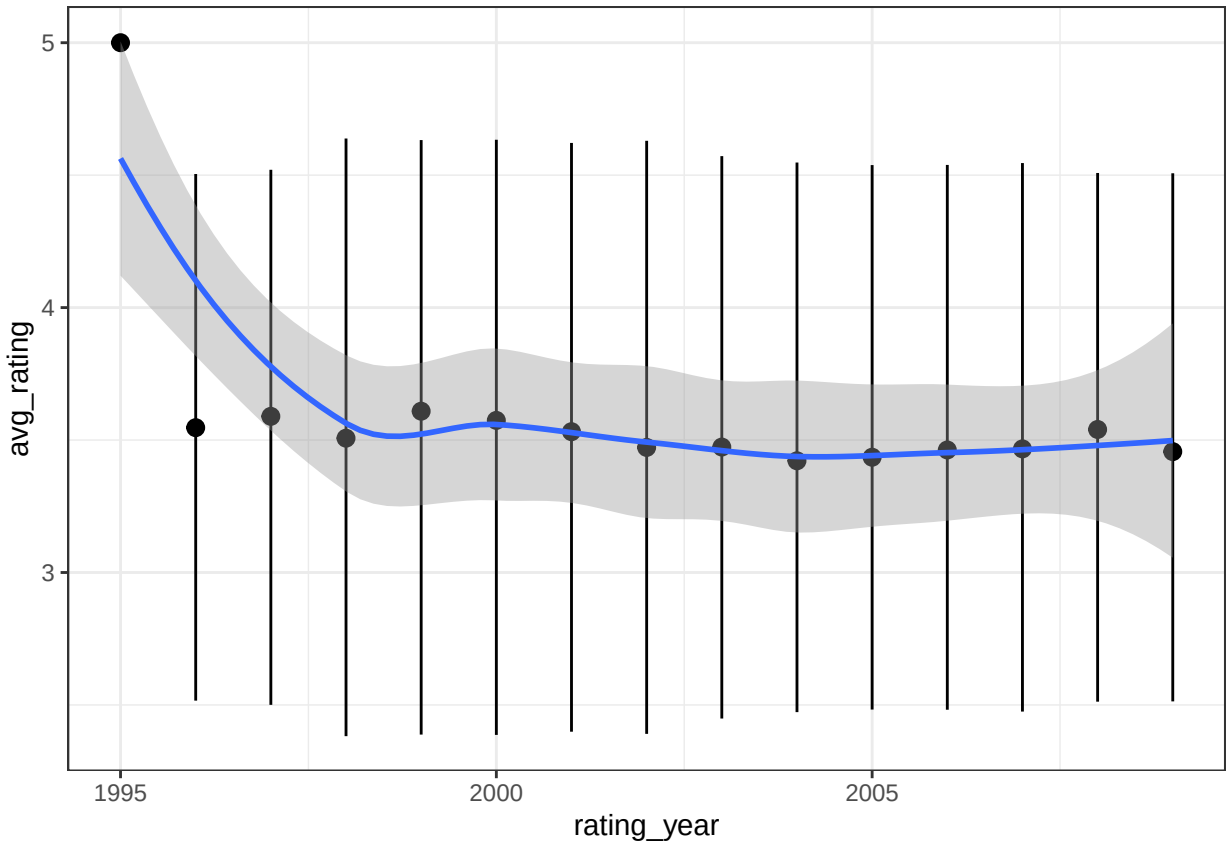
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Function for extracting rating year and month from timestamp
calculate_rating_time <- function(df){
  df_mod <- df %>%
    mutate(rating_date = as_datetime(timestamp)) %>%
    mutate(rating_date = round_date(rating_date, unit ="day")) %>%
    mutate(rating_year = year(rating_date),
           rating_month = month(rating_date))
  return(df_mod)
}

# Do the average ratings vary by rating year?
# Plot rating_year vs rating
dataset_train %>% calculate_rating_time() %>%
  group_by(rating_year) %>%
  summarise(avg_rating = mean(rating), sd_rating = sd(rating)) %>%
  ggplot(aes(x = rating_year, y = avg_rating)) +
  geom_pointrange(aes(ymin = average_rating - sd_rating,
                      ymax = average_rating + sd_rating)) +
  geom_smooth()
```

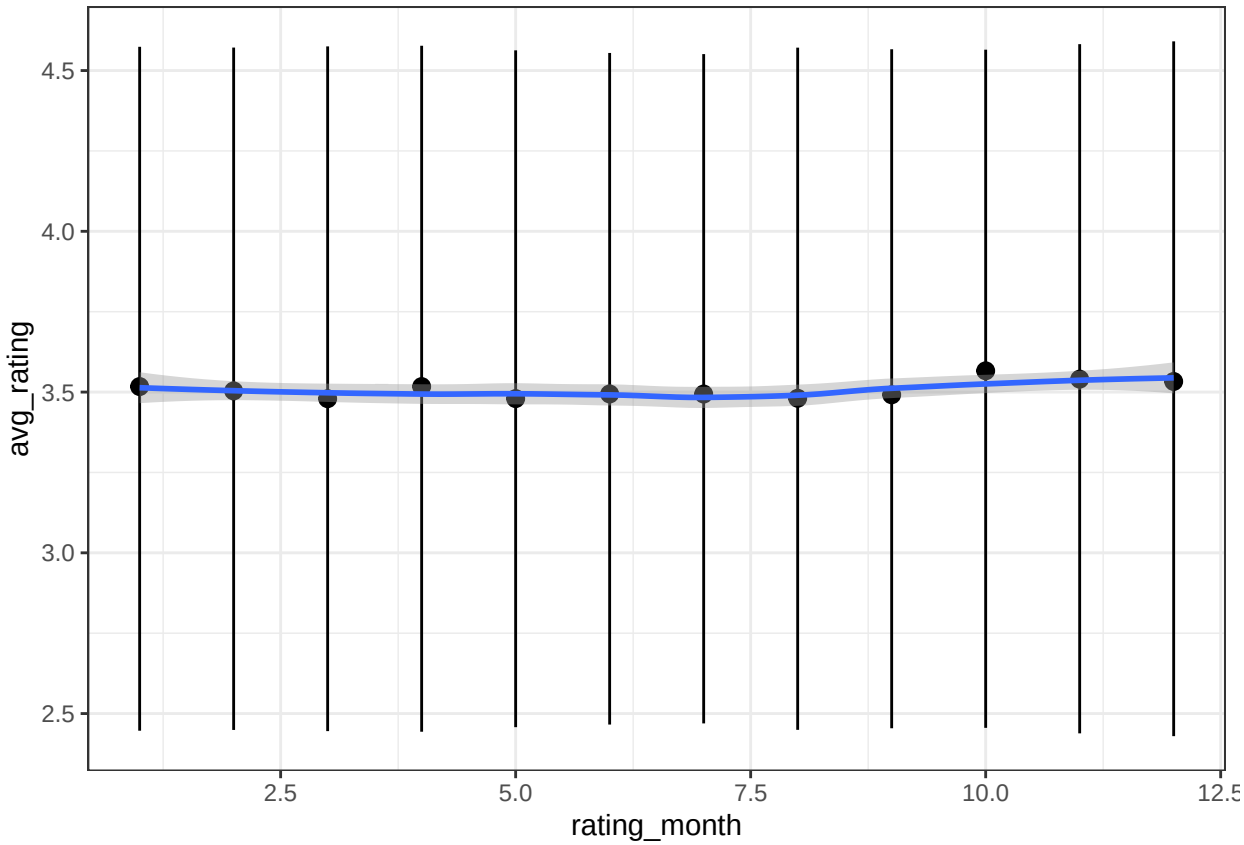
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

```
## Warning: Removed 1 rows containing missing values (geom_segment).
```




```
# Do the average ratings vary by month
dataset_train %>% calculate_rating_time() %>%
  group_by(rating_month) %>%
  summarise(avg_rating = mean(rating), sd_rating = sd(rating)) %>%
  ggplot(aes(x = rating_month, y = avg_rating)) +
  geom_pointrange(aes(ymin = average_rating - sd_rating,
                      ymax = average_rating + sd_rating)) +
  geom_smooth()
```

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



Rating time also seems effect the average rating. A fancy model would fit a temporal function $f(t)$ to the data, but just taking into account a simple yearly/monthly average should also improve the predictions.

Let's use all the observations we made here to help us build a rating algorithm.

Model training

Let's start building models based on the training data. From the exploratory data analysis we saw that we should try to incorporate the following effects into the final model:

- Release year effect
- Rating time effect (year and month of rating)
- Genre effect.

We will build these effects on top of the model which was presented in the earlier machine learning course. This model included:

- Average rating of the training data
- Movie effect
- User effect.

Additionally, we will incorporate an effect which tries to take into account the similarity of movies, i.e.:

- Movie neighbor effect (do similar movies get similar ratings?)

Model evaluation

Trained models will be evaluated by calculating the (Root Mean Square Error) RMSE for the test data set, which was created by splitting the subset of the `edx` data set. The final model with the best RMSE value, will be trained with the subset of `edx` data set including train and test data *(Note that we would use the entire* ***edx*** *data for this if our computer would have sufficient memory)*. The final model will be evaluated based on prediction performance on the `validation` data set.

```
# A function for calculating RMSE
calculate_RMSE <- function(y, y_hat){
  sqrt(mean((y_hat - y)^2))
}
```

Simple model with average rating

We start of with a very simple model, which only includes the average rating of the training data, and is as follows:

$$Y_i = \mu + \epsilon_i,$$

where Y_i is the rating for the ith observation, μ is the average rating for training data, and ϵ_i is the residual error for the ith observation.

Let's calculate the average of training data ratings, and use that for prediction.

```
#mean for training dataset
mu <- mean(dataset_train$rating)
```

Evaluating the baseline for model performance

This is the baseline for our model performance. Other models should be able to improve on this number.

```
#Evaluating the performance of the model with test data
model1 <- calculate_RMSE(y = dataset_test$rating, y_hat = mu)

# Let's collect the performance of different models in this tibble
rmse_results <- tibble(method = "Just the average", RMSE = model1)

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

method	RMSE
Just the average	1.059127

Movie effect

Some movies are generally liked more than others. Let's incorporate a movie effect to the model in the following way:

$$Y_i = \mu + b_m + \epsilon_i,$$

where b_m is the bias, or effect, caused by an individual movie. If we set the error term $\epsilon_i = 0$, we get:

$$b_m = Y_i - \mu$$

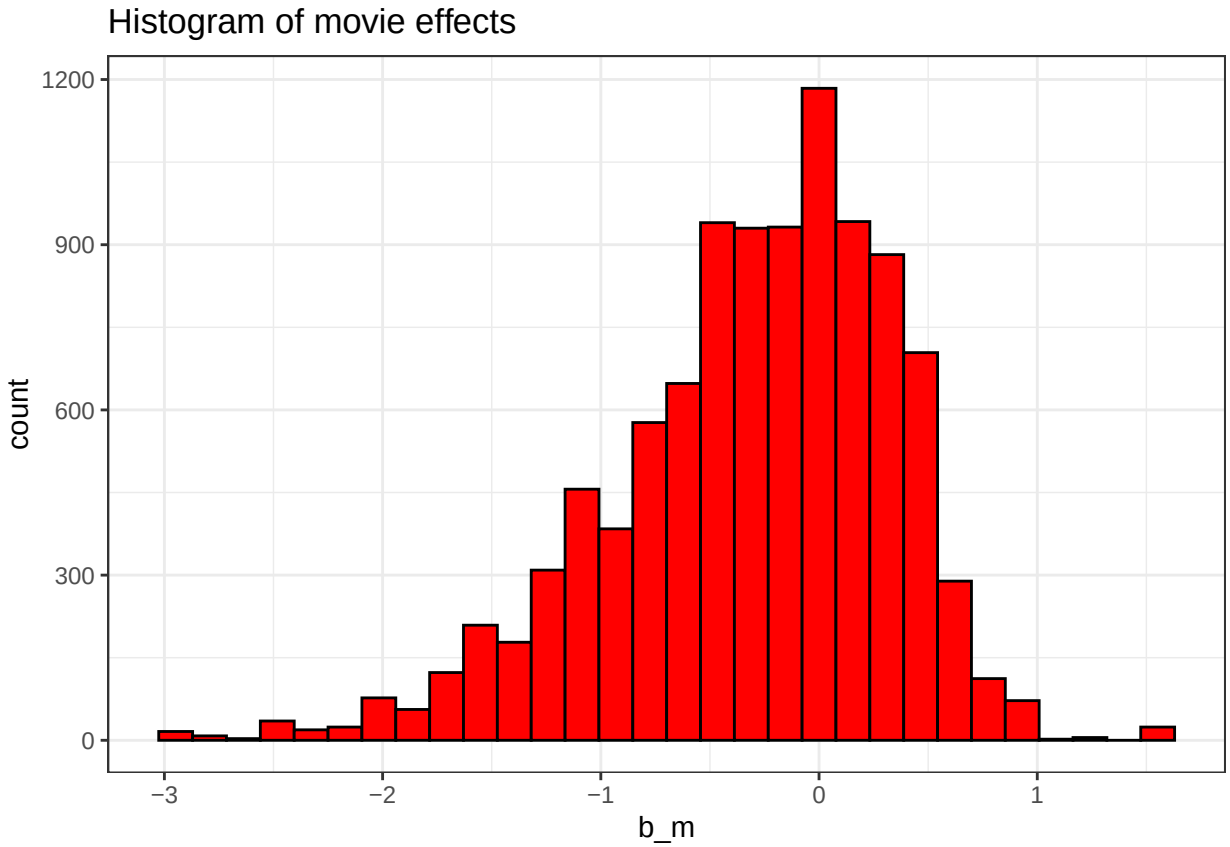
We will use the average residual per movie to account for the movie effect.

```
# Effect of movie on the rating
movie_effect <- dataset_train %>%
  group_by(movie_id) %>%
  summarise(b_m = mean(rating - mu))

#Let's inspect the movie effects
head(movie_effect)
```

```
## # A tibble: 6 x 2
##   movie_id    b_m
##      <dbl>  <dbl>
## 1        1  0.414
## 2        2 -0.303
## 3        3 -0.371
## 4        4 -0.532
## 5        5 -0.436
## 6        6  0.305
```

```
#How do the values spread out
movie_effect %>%
  ggplot(aes(x = b_m)) +
  geom_histogram(fill = "red", color = "black") +
  ggtitle("Histogram of movie effects")
```

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




```
# predictions
predicted_ratings <- mu + dataset_test %>%
  left_join(movie_effect, by='movie_id') %>%
  pull(b_m)

#calculate RMSE for model 2
model2 <- calculate_RMSE(dataset_test$rating, predicted_ratings)

#Compare performance
rmse_results <- rmse_results %>% bind_rows(tibble(method = "Movie effect", RMSE = model2))

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

method	RMSE
Just the average	1.0591270
Movie effect	0.9443401

User effect

We saw earlier that the movie has an effect on average rating. In a similar way, some users are harsher critics than others. We take this is to account in the following way:

$$Y_i = \mu + b_m + b_u + \epsilon_i,$$

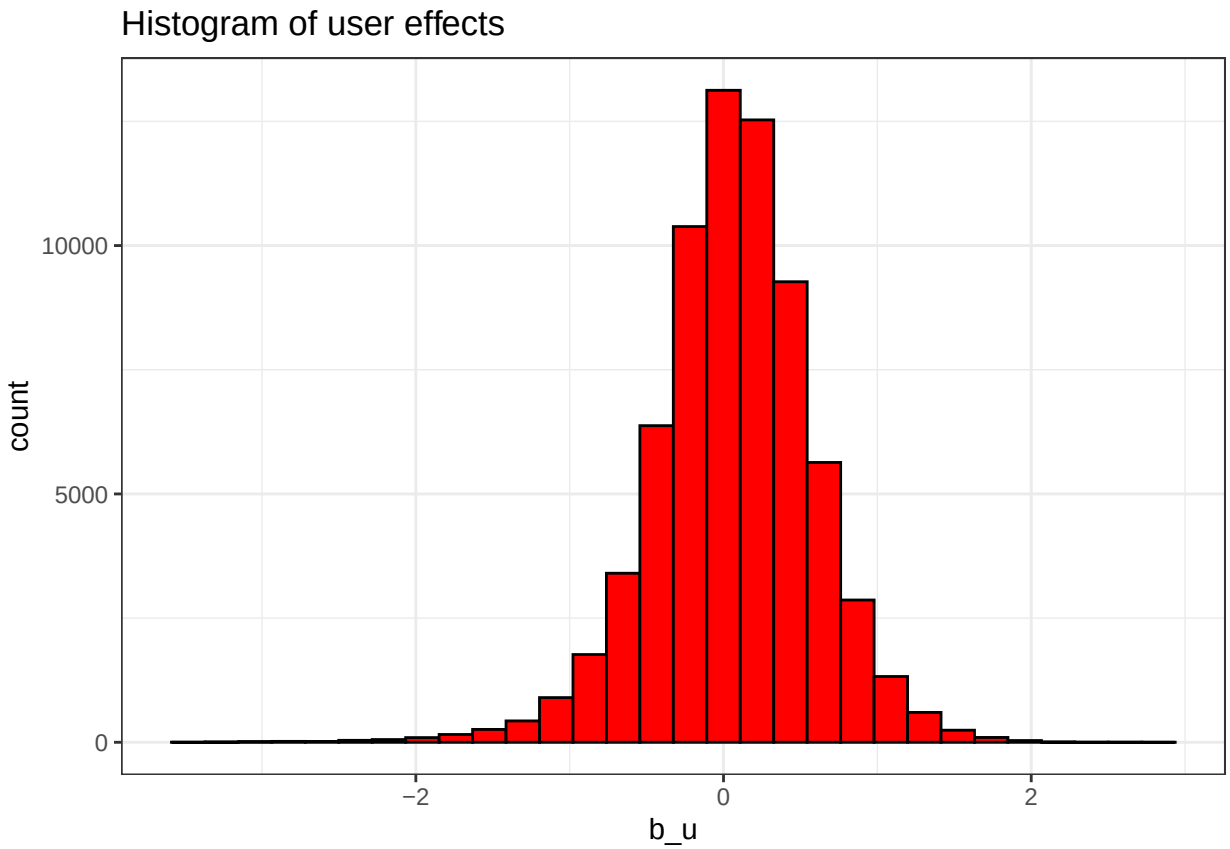
where b_u is the average residual effect of a user.

```
#Let's estimate the user effect
user_effect <- dataset_train %>%
  left_join(movie_effect, by = "movie_id") %>%
  group_by(user_id) %>%
  summarise(b_u = mean(rating - mu - b_m))

#Inspecting the element
head(user_effect)
```

```
## # A tibble: 6 x 2
##   user_id    b_u
##     <int>  <dbl>
## 1       1  2.09
## 2       2 -0.568
## 3       3  0.135
## 4       4  0.157
## 5       5  0.361
## 6       6  0.319
```

```
#How do the values spread out
user_effect %>%
  ggplot(aes(x = b_u)) +
  geom_histogram(fill = "red", color = "black") +
  ggtitle("Histogram of user effects")
```

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




```
#Predictions with user effect
predicted_ratings <- dataset_test %>%
  left_join(movie_effect, by='movie_id') %>%
  left_join(user_effect, by='user_id') %>%
  mutate(pred = mu + b_m + b_u) %>%
  pull(pred)

#Calculate RMSE for model 3
model3 <- calculate_RMSE(dataset_test$rating, predicted_ratings)

#Compare performance
rmse_results <- rmse_results %>% bind_rows(tibble(method = "Movie+user effect", RMSE = model3))

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

method	RMSE
Just the average	1.0591270
Movie effect	0.9443401
Movie+user effect	0.8865647

This is the model we saw in an earlier course. Let's start building on this to further improve our model.

Year effect

Exploratory data analysis revealed that the average ratings vary as a function of the release year of the movie. We will add a term for this as well:

$$Y_i = \mu + b_m + b_u + b_y + \epsilon_i,$$

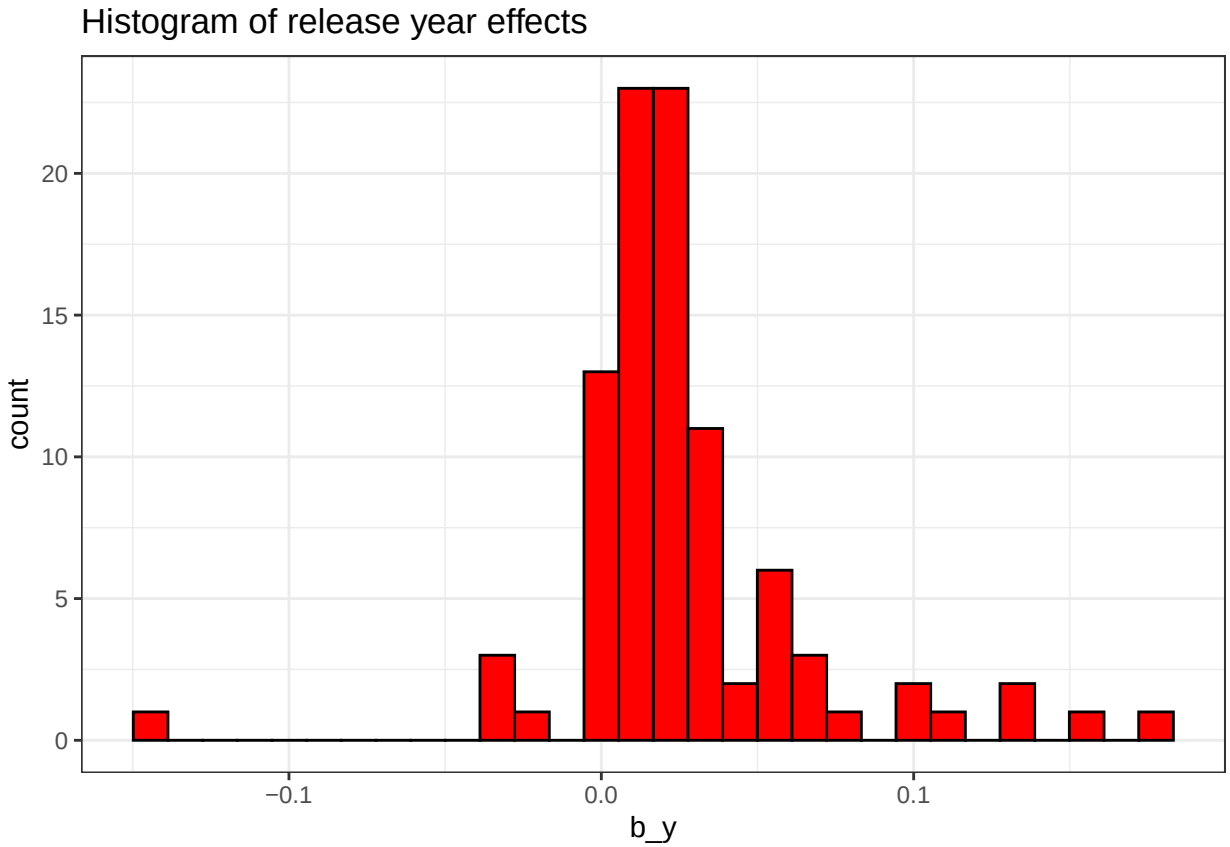
where b_y is the effect of the movie release year.

```
# Calculate effect of release year
year_effect <- dataset_train %>%
  left_join(movie_effect, by = "movie_id") %>%
  left_join(user_effect, by = "user_id") %>%
  group_by(year) %>%
  summarise(b_y = mean(rating - mu - b_m - b_u))

#Inspecting the element
head(year_effect)
```

```
## # A tibble: 6 x 2
##    year      b_y
##   <dbl>    <dbl>
## 1  1915  0.161
## 2  1916  0.113
## 3  1917  0.0346
## 4  1918 -0.148
## 5  1919  0.00721
## 6  1920  0.128
```

```
#How do the values spread out
year_effect %>%
  ggplot(aes(x = b_y)) +
  geom_histogram(fill = "red", color = "black") +
  ggtitle("Histogram of release year effects")
```

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




```
#Predictions with movie + user and release year effect
predicted_ratings <- dataset_test %>%
  left_join(movie_effect, by='movie_id') %>%
  left_join(user_effect, by='user_id') %>%
  left_join(year_effect, by='year') %>%
  mutate(pred = mu + b_m  + b_u + b_y) %>%
  pull(pred)

#Calculate RMSE for model 4
(model4 <- calculate_RMSE(dataset_test$rating, predicted_ratings))
```

```
## [1] 0.8862471
```

```
#Compare performance
rmse_results <- rmse_results %>% bind_rows(tibble(method = "Movie+user+release year effect", RMSE = mod

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

method	RMSE
Just the average	1.0591270
Movie effect	0.9443401
Movie+user effect	0.8865647
Movie+user+release year effect	0.8862471

Rating time effect

In addition to release year, the time when the movie was rated also has an effect, as we saw earlier. We can incorporate a term by grouping the data based on year and month of rating. Our model is as follows:

$$Y_i = \mu + b_m + b_u + b_y + b_{rym} + \epsilon_i,$$

where b_{rym} is the effect of the year and month when a movie was rated.

```
# Calculate effect of rating year and month
rating_year_effect <- dataset_train %>%
  left_join(movie_effect, by = "movie_id") %>%
  left_join(user_effect, by = "user_id") %>%
  left_join(year_effect, by = "year") %>%
  calculate_rating_time() %>%
  group_by(rating_year, rating_month) %>%
  summarise(b_rym = mean(rating - mu - b_m - b_u - b_y))
```

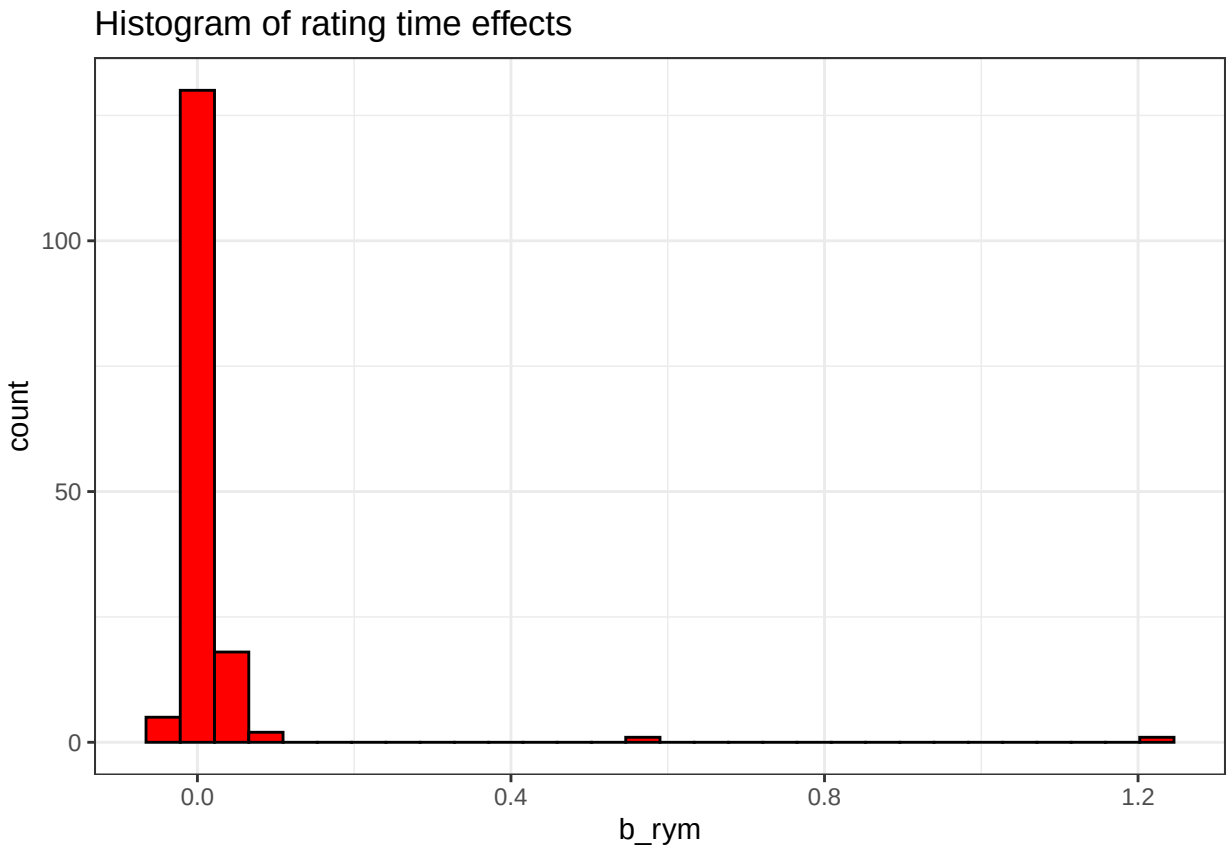
```
## `summarise()` has grouped output by 'rating_year'. You can override using the `.groups` argument.
```

```
#Inspecting the element
head(rating_year_effect)
```

```
## # A tibble: 6 x 3
## # Groups:   rating_year [2]
##   rating_year rating_month  b_rym
##         <dbl>        <dbl>  <dbl>
## 1        1995            1 1.24
## 2        1996            1 0.568
## 3        1996            2 0.0913
## 4        1996            3 0.0735
## 5        1996            4 0.0539
## 6        1996            5 0.0245
```

```
#How do the values spread out
rating_year_effect %>%
  ggplot(aes(x = b_rym)) +
  geom_histogram(fill = "red", color = "black") +
  ggtitle("Histogram of rating time effects")
```

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




```
#Predictions with movie + user and release year effect
predicted_ratings <- dataset_test %>%
  left_join(movie_effect, by='movie_id') %>%
  left_join(user_effect, by='user_id') %>%
  left_join(year_effect, by='year') %>%
  calculate_rating_time() %>%
  left_join(rating_year_effect, by = c('rating_year', 'rating_month')) %>%
  mutate(pred = mu + b_m  + b_u + b_y + b_rym) %>%
  pull(pred)

#Calculate RMSE for model 5
model5 <- calculate_RMSE(dataset_test$rating, predicted_ratings)

#Compare performance
rmse_results <- rmse_results %>% bind_rows(tibble(method = "Movie+user+release year+rating time effect"

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

method	RMSE
Just the average	1.0591270
Movie effect	0.9443401
Movie+user effect	0.8865647
Movie+user+release year effect	0.8862471
Movie+user+release year+rating time effect	0.8861815

Genre effect

Genre also seems to have an effect on the average rating. However, accounting for the genre is slightly trickier, as a single movie may be categorized into several different genres. One way of doing this is by finding the effects for all individual movie genres and adding them together. Furthermore, the individual genre effects will be weighed by multiplying them with either `1` or `0` based on a certain movie's genre classification.

We incorporate a term based on the genre classification as follows:

$$Y_i = \mu + b_m + b_u + b_y + b_{rym} + \sum_i w_i b_{g,i} + \epsilon_i,$$

where $b_{g,i}$ is the effect of an individual genre, e.g. "adventure", and w_i is the weight term for genre i, which is set as $w_i = 1$ if movie belongs to genre i, and $w_i = 0$ otherwise.

```
#The effect of individual genres
genre_effect <- dataset_train  %>%
  left_join(movie_effect, by = "movie_id") %>%
  left_join(user_effect, by = "user_id") %>%
  left_join(year_effect, by = "year") %>%
  calculate_rating_time() %>%
  left_join(rating_year_effect, by = c('rating_year', 'rating_month')) %>%
  pivot_longer(cols = all_of(indiv_genres), names_to = "genre", values_to = "value") %>%
  filter(value == 1) %>%
  group_by(genre) %>%
  summarise(b_g = mean(rating - mu - b_m - b_u - b_y - b_rym))

#Effect of individual genres
genre_effect
```

```
## # A tibble: 20 x 2
##    genre                  b_g
##    <chr>                <dbl>
##  1 action           -0.0122
##  2 adventure        -0.0142
##  3 animation        -0.0133
##  4 children         -0.0236
##  5 comedy           -0.000992
##  6 crime             0.00918
##  7 documentary       0.0579
##  8 drama             0.0104
##  9 fantasy          -0.00726
## 10 film_noir         0.0220
## 11 horror            0.00299
## 12 imax             -0.00962
## 13 musical          -0.0149
## 14 mystery           0.0114
## 15 no_genres_listed -0.00000239
## 16 romance          -0.00259
## 17 sci_fi           -0.0127
## 18 thriller         -0.00232
## 19 war               0.00157
## 20 western          -0.00810
```

```r
# A function for calculating the summed total genre combination effect
#Individual genre weights will be set as 0 if genre not present
calculate_genre_effect <- function(df){
  df_with_genre_effect <- df %>%
    mutate(b_g =
             adventure * genre_effect$b_g[genre_effect$genre == "adventure"] +
             children * genre_effect$b_g[genre_effect$genre == "children"] +
             fantasy * genre_effect$b_g[genre_effect$genre == "fantasy"] +
             documentary * genre_effect$b_g[genre_effect$genre == "documentary"] +
             action * genre_effect$b_g[genre_effect$genre == "action"] +
             thriller * genre_effect$b_g[genre_effect$genre == "thriller"] +
             comedy * genre_effect$b_g[genre_effect$genre == "comedy"] +
             drama * genre_effect$b_g[genre_effect$genre == "drama"] +
             war * genre_effect$b_g[genre_effect$genre == "war"] +
             western * genre_effect$b_g[genre_effect$genre == "western"] +
             romance * genre_effect$b_g[genre_effect$genre == "romance"] +
             sci_fi * genre_effect$b_g[genre_effect$genre == "sci_fi"] +
             horror * genre_effect$b_g[genre_effect$genre == "horror"] +
             mystery * genre_effect$b_g[genre_effect$genre == "mystery"] +
             crime* genre_effect$b_g[genre_effect$genre == "crime"] +
             film_noir * genre_effect$b_g[genre_effect$genre == "film_noir"] +
             musical * genre_effect$b_g[genre_effect$genre == "musical"] +
             animation * genre_effect$b_g[genre_effect$genre == "animation"] +
             imax * genre_effect$b_g[genre_effect$genre == "imax"]
           )

  return(df_with_genre_effect)
}



#Predictions with movie + user + release year and genre effect
predicted_ratings <- dataset_test %>%
  left_join(movie_effect, by='movie_id') %>%
  left_join(user_effect, by='user_id') %>%
  left_join(year_effect, by='year') %>%
  calculate_rating_time() %>%
  left_join(rating_year_effect, by = c('rating_year', 'rating_month')) %>%
  calculate_genre_effect() %>%
  mutate(pred = mu + b_m + b_u + b_y + b_rym + b_g) %>%
  pull(pred)


#Calculate RMSE for model 6
model6 <- calculate_RMSE(dataset_test$rating, predicted_ratings)

#Compare performance
rmse_results <- rmse_results %>%
    bind_rows(tibble(method = "Movie+user+release year+rating time+genre effect", RMSE = model6))

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

method	RMSE
Just the average	1.0591270
Movie effect	0.9443401
Movie+user effect	0.8865647
Movie+user+release year effect	0.8862471
Movie+user+release year+rating time effect	0.8861815
Movie+user+release year+rating time+genre effect	0.8861260

Movie neighbors model

Finally, we will try to find similar movies based on the genre classification. The movies will be grouped into different buckets based on the genre effect term ($\sum_i w_i b_{g,i}$), and we will adjust for the average residual error for movies with the same genre effect term value. Here we don't take into account the somewhat unlikely probability of different genre combinations having the same genre term value.

We incorporate a term based on the movie neighbors as follows:

$$Y_i = \mu + b_m + b_u + b_y + b_{rym} + \sum_i w_i b_{g,i} + b_{mn} + \epsilon_i,$$

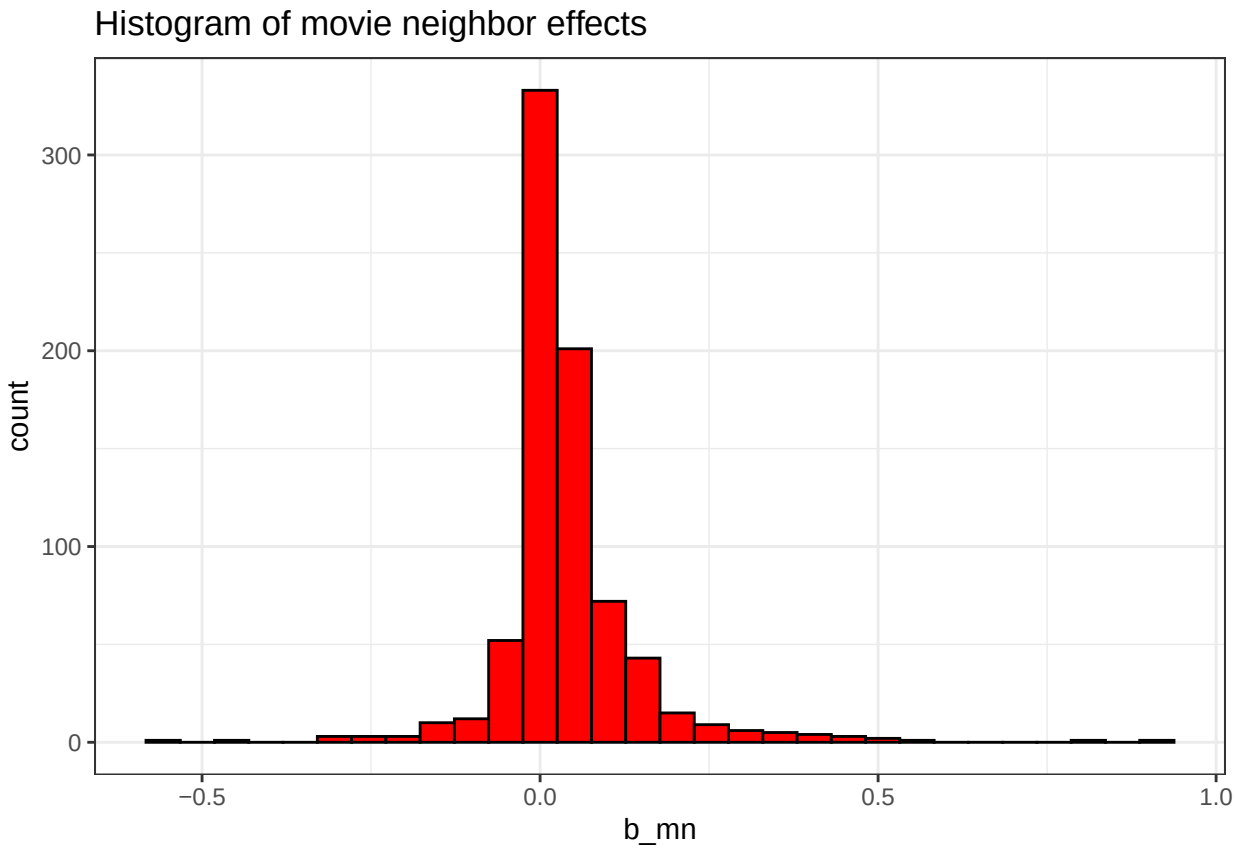
where b_{mn} is the effect of a certain movie genre combination, so-called "movie neighbor effect".

```
movie_neighbor_effects <- dataset_train %>%
  left_join(movie_effect, by='movie_id') %>%
  left_join(user_effect, by='user_id') %>%
  left_join(year_effect, by='year') %>%
  calculate_rating_time() %>%
  left_join(rating_year_effect, by = c('rating_year', 'rating_month')) %>%
  calculate_genre_effect() %>%
  group_by(b_g) %>%
  summarise(b_mn = mean(rating - mu - b_m - b_u - b_y - b_rym - b_g))

#Inspecting the element
head(movie_neighbor_effects)
```

```
## # A tibble: 6 x 2
##       b_g    b_mn
##     <dbl>   <dbl>
## 1 -0.0771  0.125
## 2 -0.0769  0.0924
## 3 -0.0761 -0.0121
## 4 -0.0743  0.0452
## 5 -0.0740  0.154
## 6 -0.0722  0.0711
```

```
#How do the values spread out
movie_neighbor_effects %>%
  ggplot(aes(x = b_mn)) +
  geom_histogram(fill = "red", color = "black") +
  ggtitle("Histogram of movie neighbor effects")
```

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




```
#Predictions with movie + user + release year + genre effect and movie neighbor model
predicted_ratings <- dataset_test %>%
  left_join(movie_effect, by='movie_id') %>%
  left_join(user_effect, by='user_id') %>%
  left_join(year_effect, by='year') %>%
  calculate_rating_time() %>%
  left_join(rating_year_effect, by = c('rating_year', 'rating_month')) %>%
  calculate_genre_effect() %>%
  left_join(movie_neighbor_effects, by='b_g') %>%
  mutate(pred = mu + b_m + b_u + b_y + b_rym + b_g + b_mn) %>%
  pull(pred)

#Calculate RMSE for model 7
model7 <- calculate_RMSE(dataset_test$rating, predicted_ratings)

#Compare performance
rmse_results <- rmse_results %>% bind_rows(tibble(method = "Movie+user+(r&r)year+genre+movie neighbor e

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

method	RMSE
Just the average	1.0591270
Movie effect	0.9443401
Movie+user effect	0.8865647
Movie+user+release year effect	0.8862471

method	RMSE
Movie+user+release year+rating time effect	0.8861815
Movie+user+release year+rating time+genre effect	0.8861260
Movie+user+(r&r)year+genre+movie neighbor effect	0.8859050

Regularization

Some movies and users have several entries in the data set, while many have only a few. There is also wide variation regarding movies in different genres and release years vary quite a bit as well. To prevent the model from being over trained on rare data points, we will apply regularization to place larger weight on more common observations.

We add a penalty term to the least squares estimate, as follows:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i)^2+\lambda\sum_i\beta_i^2$$

where λ is the regularization parameter. The optimal value of λ minizes the above equation. It was shown in the *course material* that the values b_i, which minimize the function are:

$$\hat{b_i}(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu})$$

where n_i is the number of ratings for a given group.

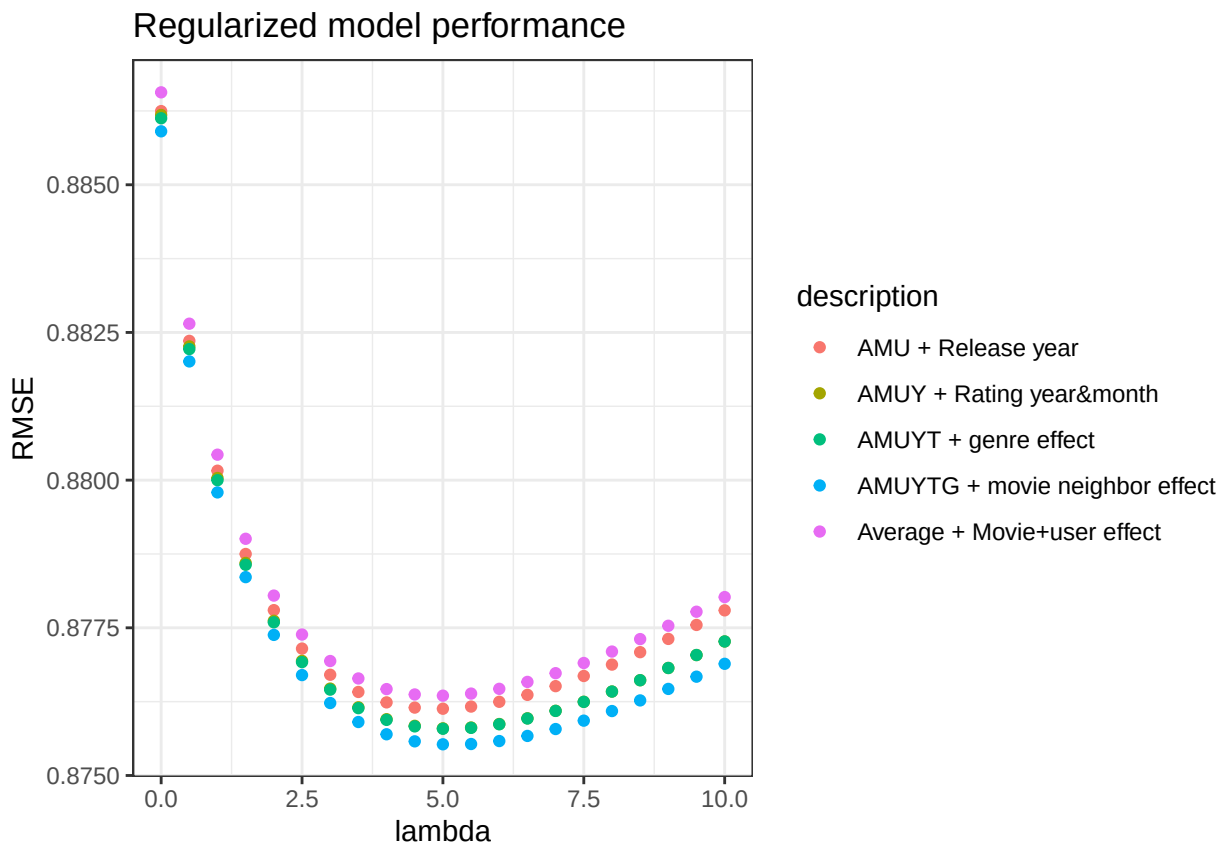
Let's find the optimal value for λ, and compare model performance.

```
# Different regularization parameter values to try
lambdas <- seq(0, 10, 0.5)

# Ridge regression type of regularization (L2)
rmses <- sapply(lambdas, function(l){
  mu <- mean(dataset_train$rating)

  ## Effects --------------------

  #movie effect
  b_m <- dataset_train %>%
    group_by(movie_id) %>%
    summarize(b_m = sum(rating - mu)/(n()+l))

  #user effect
  b_u <- dataset_train %>%
    left_join(b_m, by="movie_id") %>%
    group_by(user_id) %>%
    summarize(b_u = sum(rating - mu - b_m)/(n()+l))

  # Calculate effect of release year
  b_y <- dataset_train %>%
    left_join(b_m, by = "movie_id") %>%
    left_join(b_u, by = "user_id") %>%
    group_by(year) %>%
```

```
  summarise(b_y = sum(rating - mu - b_m - b_u)/(n()+l))

# Calculate effect of rating year and month
b_rym <- dataset_train %>%
  left_join(b_m, by = "movie_id") %>%
  left_join(b_u, by = "user_id") %>%
  left_join(b_y, by = "year") %>%
  calculate_rating_time() %>%
  group_by(rating_year, rating_month) %>%
  summarise(b_rym = sum(rating - mu - b_m - b_u - b_y)/(n() + l))

# Calculate effect of genre
genre_effect <- dataset_train  %>%
  left_join(b_m, by = "movie_id") %>%
  left_join(b_u, by = "user_id") %>%
  left_join(b_y, by = "year") %>%
  calculate_rating_time() %>%
  left_join(b_rym, by = c('rating_year', 'rating_month')) %>%
  pivot_longer(cols = all_of(indiv_genres), names_to = "genre", values_to = "value") %>%
  filter(value == 1) %>%
  group_by(genre) %>%
  summarise(b_g = sum(rating - mu - b_m - b_u - b_y - b_rym)/(n()+l))

# Calculate effect of movie neighbors == similar in terms of genre spread
b_mn <- dataset_train %>%
  left_join(b_m, by='movie_id') %>%
  left_join(b_u, by='user_id') %>%
  left_join(b_y, by='year') %>%
  calculate_rating_time() %>%
  left_join(b_rym, by = c('rating_year', 'rating_month')) %>%
  calculate_genre_effect() %>%
  group_by(b_g) %>%
  summarise(b_mn = sum(rating - mu - b_m - b_u - b_y - b_rym - b_g)/(n()+l))

## Models --------------------

#Movie + user effect
predicted_ratings <- dataset_test %>%
  left_join(b_m, by = "movie_id") %>%
  left_join(b_u, by = "user_id") %>%
  mutate(pred = mu + b_m + b_u) %>%
  pull(pred)

#performance of model
RMSE1 <- calculate_RMSE(predicted_ratings, dataset_test$rating)

#Movie + user + release year effect
predicted_ratings <- dataset_test %>%
  left_join(b_m, by = "movie_id") %>%
  left_join(b_u, by = "user_id") %>%
  left_join(b_y, by = "year") %>%
  mutate(pred = mu + b_m + b_u + b_y) %>%
  pull(pred)
```

```
  #performance of model
  RMSE2 <- calculate_RMSE(predicted_ratings, dataset_test$rating)


  #Predictions with movie + user + release year and rating time effect
  predicted_ratings <- dataset_test %>%
    left_join(b_m, by='movie_id') %>%
    left_join(b_u, by='user_id') %>%
    left_join(b_y, by='year') %>%
    calculate_rating_time() %>%
    left_join(b_rym, by = c('rating_year', 'rating_month')) %>%
    mutate(pred = mu + b_m  + b_u + b_y + b_rym) %>%
    pull(pred)

  #performance of model
  RMSE3 <- calculate_RMSE(predicted_ratings, dataset_test$rating)

  #Predictions with movie + user + release year and genre effect
  predicted_ratings <- dataset_test %>%
    left_join(b_m, by='movie_id') %>%
    left_join(b_u, by='user_id') %>%
    left_join(b_y, by='year') %>%
    calculate_rating_time() %>%
    left_join(b_rym, by = c('rating_year', 'rating_month')) %>%
    calculate_genre_effect() %>%
    mutate(pred = mu + b_m + b_u + b_y + b_rym + b_g) %>%
    pull(pred)

  #performance of model
  RMSE4 <- calculate_RMSE(predicted_ratings, dataset_test$rating)

  #Predictions with movie + user + releas year + genre effect and movie neighbor model
  predicted_ratings <- dataset_test %>%
    left_join(b_m, by='movie_id') %>%
    left_join(b_u, by='user_id') %>%
    left_join(b_y, by='year') %>%
    calculate_rating_time() %>%
    left_join(b_rym, by = c('rating_year', 'rating_month')) %>%
    calculate_genre_effect() %>%
    left_join(b_mn, by='b_g') %>%
    mutate(pred = mu + b_m + b_u + b_y + b_rym + b_g + b_mn) %>%
    pull(pred)

  #performance of model
  RMSE5 <- calculate_RMSE(predicted_ratings, dataset_test$rating)

  ## Results for regularized models -----------------------

  #Combine models
  RMSEs <- cbind(RMSE1, RMSE2, RMSE3, RMSE4, RMSE5)

  return(RMSEs)
})
```

```
# Regularized results for models
results <- tibble(lambda = lambdas,
                  model1 = rmses[1,], #Average + Movie + user effect
                  model2 = rmses[2,], # + Release year
                  model3 = rmses[3,], # + Rating year and month
                  model4 = rmses[4,], # + genre effect
                  model5 = rmses[5,]) # + movie neighbor effect

#Different models used in regularization
model_desc <- tribble(
  ~model, ~description,
  "model1", "Average + Movie+user effect",
  "model2", "AMU + Release year", #AMU = Average, Movie, User
  "model3", "AMUY + Rating year&month", #Y = year
  "model4", "AMUYT + genre effect", # T = time (of rating)
  "model5", "AMUYTG + movie neighbor effect" # G = Genre
)

#Visualize regularization effect on different models
results %>% pivot_longer(cols = -lambda, names_to = "model", values_to = "RMSE") %>%
  left_join(model_desc, by = "model") %>%
  ggplot(aes(x = lambda, y = RMSE, color = description)) +
  geom_point() +
  ggtitle("Regularized model performance")
```



```
#Find minimum rmse
min_rmse <- results %>%
  pivot_longer(cols = -lambda, names_to = "model", values_to = "RMSE") %>%
  pull(RMSE) %>%
  which.min()

#BEST RESULT
results %>%
  pivot_longer(cols = -lambda, names_to = "model", values_to = "RMSE") %>%
  left_join(model_desc, by = "model") %>%
  slice(min_rmse)
```

```
## # A tibble: 1 x 4
##   lambda model   RMSE description
##    <dbl> <chr>  <dbl> <chr>
## 1      5 model5 0.876 AMUYTG + movie neighbor effect
```

```
# Saving the lambda for final model training
tuning_parameter <- results %>%
  pivot_longer(cols = -lambda, names_to = "model", values_to = "RMSE") %>%
  left_join(model_desc, by = "model") %>%
  slice(min_rmse) %>% pull(lambda)
```

The most complicated model gives the best RMSE with a tuning parameter λ value of 5. We will train our final model based on this information.

Training the final model with the entire training data

Before we evaluate model performance with the `validation` data, we will use the combined test and train data set to train the model parameters for the best performing model.

```
# training the final model with training and test data

#Let's free up memory first
rm("dataset_test", "dataset_train") #remove train and test
gc() #Collect garbage
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2617333 139.8    7113248  379.9  21707904 1159.4
## Vcells 74846268 571.1  603248474 4602.5 975609542 7443.4
```

```
## Parameter setting -----------
# determining model parameters with entire train and test data
  mu <- mean(dataset$rating)

  ## Effects --------------------

  #movie effect
  b_m <- dataset %>%
    group_by(movie_id) %>%
```

```
    summarize(b_m = sum(rating - mu)/(n()+tuning_parameter))

  #user effect
  b_u <- dataset %>%
    left_join(b_m, by="movie_id") %>%
    group_by(user_id) %>%
    summarize(b_u = sum(rating - mu - b_m)/(n()+tuning_parameter))

  # Calculate effect of release year
  b_y <- dataset %>%
    left_join(b_m, by = "movie_id") %>%
    left_join(b_u, by = "user_id") %>%
    group_by(year) %>%
    summarise(b_y = sum(rating - mu - b_m - b_u)/(n()+tuning_parameter))

  # Calculate effect of rating year and month
  b_rym <- dataset %>%
    left_join(b_m, by = "movie_id") %>%
    left_join(b_u, by = "user_id") %>%
    left_join(b_y, by = "year") %>%
    calculate_rating_time() %>%
    group_by(rating_year, rating_month) %>%
    summarise(b_rym = sum(rating - mu - b_m - b_u - b_y)/(n() + tuning_parameter))
```

```
## `summarise()` has grouped output by 'rating_year'. You can override using the `.groups` argument.
```

```
  # Calculate effect of genre
  genre_effect <- dataset  %>%
    left_join(b_m, by = "movie_id") %>%
    left_join(b_u, by = "user_id") %>%
    left_join(b_y, by = "year") %>%
    calculate_rating_time() %>%
    left_join(b_rym, by = c('rating_year', 'rating_month')) %>%
    pivot_longer(cols = all_of(indiv_genres), names_to = "genre", values_to = "value") %>%
    filter(value == 1) %>%
    group_by(genre) %>%
    summarise(b_g = sum(rating - mu - b_m - b_u - b_y - b_rym)/(n()+tuning_parameter))

  # Calculate effect of movie neighbors == similar in terms of genre spread
  b_mn <- dataset %>%
    left_join(b_m, by='movie_id') %>%
    left_join(b_u, by='user_id') %>%
    left_join(b_y, by='year') %>%
    calculate_rating_time() %>%
    left_join(b_rym, by = c('rating_year', 'rating_month')) %>%
    calculate_genre_effect() %>%
    group_by(b_g) %>%
    summarise(b_mn = sum(rating - mu - b_m - b_u - b_y - b_rym - b_g)/(n()+tuning_parameter))

## Model predictions -----------
# A function for making predictions with the final model
predict_ratings <- function(df){
  ##Predictions with movie + user + release year + genre effect and movie neighbor model
```

```
  df <- df %>%
    left_join(b_m, by='movie_id') %>%
    left_join(b_u, by='user_id') %>%
    left_join(b_y, by='year') %>%
    calculate_rating_time() %>%
    left_join(b_rym, by = c('rating_year', 'rating_month')) %>%
    calculate_genre_effect() %>%
    left_join(b_mn, by='b_g') %>%
    mutate(pred = mu + b_m + b_u + b_y + b_rym + b_g + b_mn)

  ## Let's impute the average for predictions that turn out NA
  ## This may happen since we did not use the entire edx data for training
  ## so some user_id & movie_id values might not be included
  predicted_ratings <- df %>%
    mutate(pred = if_else(is.na(pred), # IF prediction is NA
                          true = mu, # RETURN the train average
                          false = pred)) %>%  # ELSE return the prediction
    pull(pred)

  return(predicted_ratings)
}
```

Calculating training error (RMSE) for the final model

Let's see how well the final model performs on the training data by computing the training RMSE.

```
calculate_RMSE(dataset$rating, predict_ratings(dataset))
```

```
## [1] 0.8475195
```

Let's also visualize the predictions vs. true ratings for a small group of observations (n = 1000).

```
dataset %>% head(n = 1000) %>%
  mutate(pred = predict_ratings(head(dataset, n = 1000))) %>%
  ggplot(aes(x = pred, y = rating)) +
  geom_point() +
  geom_abline(slope = 1, color = "red") +
  labs(x = "Predicted rating", y = "True rating") +
  ggtitle("Comparison of 1000 predicted and true ratings",
          subtitle = "Predictions on the training data")
```

Final model performance with validation dataset

Here we evaluate the performance of the final model using the `validation` data set as a yardstick. We will predict ratings and calculate the RMSE using the true ratings for the `validation` data set.

```
# Loading the validation data set from local folder
validation <- readRDS(file = "MovielensData/validation.Rds")
dim(validation)
```

```
## [1] 999999      6
```

```
# We need to do the same modifications for the validation data we did for the training data set
validation <- validation %>%
  separate_genres() %>%
  count_genres() %>%
  extract_year()

dim(validation)
```

```
## [1] 999999     26
```

```r
# RMSE for predictions made for validation dataset
(Final_model_RMSE <- calculate_RMSE(validation$rating, predict_ratings(validation)))
```

```
## [1] 0.8742688
```

The final model achieved the following RMSE when tested on the `validation` dataset: 0.8742688.

Improvement suggestions for further model development

The model we trained performed quite well on the validation dataset, achieving a RMSE below `0.90`. If we compare this to the winning score of the Netflix prize (`RMSE=0.8712`), we are not that far off. The model already captures many important aspects, but the following tweaks would probably improve it further:

- Adding a smooth function for approximating the effect of rating time. Now we just used the average residuals for rating year and month in our model
- The movie neighbors term could be extended to account for individual movie preferences of different users.
- Training the model with the entire `edx` data (unfortunately not feasible with my computer)

Adding new features to the model might get us closer to the winning score of the Netflix prize. However, this comes with a cost of more heavy computation. The capacity and performance of my laptop was already quite stretched with the current model, so more complicated ones might need additional computational resources.