MovieLens Capstone Project

Trevor Cummins

18th of April 2019

Abstract

HarvardX PH125.9x Capstone Project: Create a movie recommendation system using the MovieLens dataset.

Contents

1 Executive Summary **2**
1.1 Introduction . . . 2
1.2 Project Goals . . . 2
1.3 Loss Function . . . 2
1.4 Submission requirements . . . 3
1.5 Dataset . . . 3
1.5.1 Movies . . . 3
1.5.2 Rating . . . 3
1.6 Data Science Pipeline . . . 4

2 Packages **5**

3 Data Retrieval **6**

4 Data Cleansing **7**
4.1 Temporal Features . . . 7
4.2 Ignore Unrated Films . . . 7
4.3 Training and Test Set preparation . . . 8
4.4 Dataset Summary . . . 8
4.5 Check for missing values . . . 9
4.6 Split Genre to Category . . . 9

5 Data Exploration and Visualisation **11**
5.1 Movie Ratings . . . 11
5.2 Number of movies released by decade . . . 14
5.3 User Ratings - Temporal Analysis . . . 17
5.4 EDA conclusion . . . 19

6 Modelling **20**
6.1 First Model - Mean . . . 20
6.2 Second Model - Measuring the Movie Effect . . . 21
6.3 Third Model - Measuring the Movie and User Effect . . . 23
6.4 Fourth Model - Measuring the Movie, User and Category Effect . . . 25
6.5 Fifth Model - Measuring the Movie, User and Release Year Effect . . . 27
6.6 Sixth Model - Movie Effect with Regularization . . . 29
6.7 Seventh Model - Regularization of Model 4 . . . 32

7 Results Section **33**

8 Conclusion **34**
8.1 Model selection . . . 34

8.2 Training v Test Split 34
8.3 Dataset Size and performance 34

Appendix A: Remakes - Better the last time? **36**

9 References **41**

1 Executive Summary

1.1 Introduction

The MovieLens Capstone project is the first of two project submissions required to complete the HarvardX PH125.9x Professional Certificate in Data Science. The course was delivered by Professor Rafael A. Irizarry, Professor of Applied Statistics at Harvard and the Dana-Farber Cancer Institute. The course is hosted and delivered via the edX open online course provider founded in May 2012 by scientists from Harvard and MIT.

1.2 Project Goals

The goal of the project is to create a movie recommendation system based upon the MovieLens dataset.

Recommendation systems use ratings that users have given items in order to make specific recommendations (Irizzarry 2019). MovieLens uses a 5 star rating scale where 1 star denotes the film is not good and a five star suggests it is an excellent movie.

Ten million user ratings are downloaded and trained using multiple machine learning algorithms to identify an optimal model to predict movie ratings in a hold out validation test set. 10% of the MovieLens dataset was randomly split to populate the validation test set.

The project submission aims to:

- provide a demonstration and application of Data Science techniques and R programming skills learned throughout the course;
- communicate the project methodology followed; and
- present some insights gained from the analysis of the dataset.

1.3 Loss Function

The Loss Function is a general approach for defining "best" in machine learning (Irizzarry 2019). There are many available Loss Functions that can be applied depending on the problem. The **sum of squares error** Loss Function is selected given the large volume of data in the Movielens dataset and the continuous nature of the expected outcome.

The sum of squares error is the sum of the difference between each predicted value and the actual value. The difference is squared so that we measure the absolute value of the difference. The mean of the sum of squares error is used as the test set contains many observations, N.

$$\text{Root Mean Square Error (RMSE)} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

Where:

N = Number of observations

$\hat{y}_i$ = predictor: predicted rating for movie i

y_i = observed outcome: mean rating for movie i

The R command **RMSE()** is used in this report to compute the RMS error.

Each model for the movie rating is learned by fitting a training set. The goal is to find the best set of weights and bias that minimizes the loss function. Seven different algorithms have been executed to calculate the RMSE and identify the best (minimum) value. The formula and result for each model is detailed in the relevant model section of this report.

1.4 Submission requirements

The following files are submitted for assessment and graded by a scoring rubric that has been defined by course staff:

1. A report in the form of an Rmd (R Markdown) file
2. A report in the form of an Adobe PDF document knit from your Rmd file
3. An R script or Rmd file that generates predicted movie ratings and calculates the RMSE (Root Mean Squared Error).

Each submission (reports and script) will be graded by three peers and the median grade will be awarded.

The movie rating predictions are compared to the true ratings in a test validation set using RMSE.

The report documents the analysis and presents the findings, along with supporting statistics and figures including the RMSE generated for multiple models assessed.

1.5 Dataset

GroupLens Research has collected and made available rating data sets from the MovieLens non-commercial and personalized movie recommendations web site. GroupLens is a research lab in the Department of Computer Science and Engineering at the University of Minnesota.

The project requires that the MovieLens 10M Dataset is used. This contains 10 million ratings applied to 10,000 movies by 72,000 users. The file was released in January 2009. The assignment uses the movies.dat and rating.dat files contained in the compressed file from GroupLens.

1.5.1 Movies

Movie information is contained in the file movies.dat. Each line of this file represents one movie, and has the following format:

MovieID::Title::Genres

1.5.2 Rating

All ratings are contained in the file ratings.dat. Each line of this file represents one rating of one movie by one user, and has the following format:

UserID::MovieID::Rating::Timestamp

The lines within this file are ordered first by UserID, then, within user, by MovieID.

Ratings are made on a 5-star scale, with half-star increments.

1.6 Data Science Pipeline

A data science pipeline is the overall process to prepare, import, tidy, visualise, model, interpret and communicate data (Wickham and Grolemund 2017). As defined by Zacharias Voulgaris, it "is a complex process comprised of a number of inter-dependent steps, each bringing us closer to the end result, be it a set of insights to hand off to our manager or client, or a data product for our end-user." (Voulgaris 2017, ch. 2)

For the purpose of this report, methodologies will refer specifically to machine learning methodologies. I make the distinction here between the definition of **pipeline** and **methodology** due to my background in enterprise applications where methodology in the context of software development lifecycle (SDLC) refers to the waterfall or agile methodologies.

The following pipeline steps where followed for the project and have been documented in separate sections on this report.

Table 1: Data Science Pipeline

Pipeline	Step	Goal
Data Source:		
	Data Extraction	**ETL: Source**
	Data Load	**ETL: Import**
Data Preparation:		
	Data processing and wrangling	**ETL: Clean& Transform**
	Data Exploration	**Initial Discovery**
	Data Visualisation	**Visualise**
Data Modeling:		
	Feature Extraction & Engineering	**Identify Outcome/Features**
	Develop Models	**Capture Pattern**
	Data Learning	**Model Evaluation**
Communication:		
	Summary	**Results**
	Insights	**Intuition**
	Citations and References	**References**

2 Packages

Table 2: Package Installation notes: Additional information for packages required to support this report

Package	Note
bbcplot	British Broadcasting Corporation package for extending ggplot2 themes
devtools	Collection of R development tools
ggalt	Support geom_dumbbell() plots
pacman	Contains tools to more conveniently perform tasks associated with add-on packages.
sjmisc	Collection of miscellaneous utility functions integrated into a tidyverse workflow
snakecase	Collection of miscellaneous utility functions, supporting data transformation
tictoc	Performance Benchmarking
xtable	Added for RMarkdown file functionality
kableExtra	Added for RMarkdown file functionality

```
if(!require(pacman))install.packages("pacman")
if(!require(devtools))install.packages('devtools')
devtools::install_github('bbc/bbplot')                    # Load the BBC plots for ggplot2
pacman::p_load('devtools',                                # Development
               'data.table',                              # Data Importation
               'tidyverse', 'dplyr', 'tidyr', 'stringr',  # Data Manipulation
               'sjmisc', 'snakecase', 'lubridate',        # Data Manipulation
               'ggplot2', 'bbplot', 'ggalt',              # Visualisation
               'caret', 'randomForest',                   # Classification and Regression
               'tictoc',                                  # Performance measuring
               'xtable', 'kableExtra')                    # RMarkdown specific
```

3 Data Retrieval

The file is downloaded and unzipped using the code provided by the course staff team with a modification applied to check if the file has already been downloaded to the working directory to restrict to limit the number of downloads from Grouplens.

Learning Point: The unzip function creates a subfolder to the working directory containing the movies.dat and ratings.dat files. The folder names that are used in variable definition must match exactly to the subfolder in the compressed file otherwise the unzip process will fail.

MovieLens dataset:

http://files.grouplens.org/datasets/movielens/ml-10m.zip

The commands **tic()** and **toc()** are used to record the processing time for some critical blocks of code. This was useful to benchmark and compare different techniques used in the code. The code is not displayed in the generated report (using **echo=FALSE** in the RMarkdown chunk definition)[1].

```
#Data Retrieval
# MovieLens dataset: the file folder that is created must be the same name as
#the folder in the compressed zip file
tic("Prepare Ratings and Movies datasets")
dir_file_ratings <- "ml-10M100K/ratings.dat"
dir_file_movies <- "ml-10M100K/movies.dat"
grouplens_file <- "http://files.grouplens.org/datasets/movielens/ml-10m.zip"

# Check if the file has already been downloaded to the working directory/unzipped folder and only download one time.
if(!file.exists(dir_file_ratings) || !file.exists(dir_file_movies)) {
  tic("Download from GroupLens")
  dl_file <- tempfile()
  download.file(grouplens_file, dl_file)
  toc() #Print the time segment
    # Split the columns of the movies and ratings files using the separator ::. Also rename the columns.
    ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl_file, dir_file_ratings))),
                          col.names = c("userId", "movieId", "rating", "timestamp"))
  movies <- str_split_fixed(readLines(unzip(dl_file, dir_file_movies)), "\\::", 3)
} else {
  print("Files already exist and do not need to be downloaded again")
  ratings <- read.table(text = gsub("::", "\t", readLines(dir_file_ratings)),
                          col.names = c("userId", "movieId", "rating", "timestamp"))
  movies <- str_split_fixed(readLines(dir_file_movies), "\\::", 3)
}
```

```
## [1] "Files already exist and do not need to be downloaded again"
```

```
#Add the column names to the movie dataframe and format the fields
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))
toc()
```

```
## Prepare Ratings and Movies datasets: 88.907 sec elapsed
```

[1]The LaTeX font size can be adjusted in RMarkdown by setting the size to tiny, scriptsize, footnotesize, small, normalisize, large, huge. See the examples provided in this RMD report

4 Data Cleansing

4.1 Temporal Features

The movie title in the the movies.dat file contains both the title of the movie and the year of release, for example, Leaving Las Vegas (1995).

The following function module splits the title "Leaving Las Vegas" and release year "1995" into new columns in the movies dataset so that both features can be used for further data exploration. Removing the release year from the title enables the release year to be used as a prediction feature, the grouping of releases by decade, and movie remakes to be compared.

```
#Create new features for release year and the rating timestamp
#Split the year out from Title in to it's own column.
pattern <- "[(]\\d{4}[)]"    #Builds a pattern (year) i.e. ()
string_format <- function(x){
  #Detect the pattern (year). Build up the pattern using a combination of str_detect() and str_view() on one record only in order to verify
  #that the coding was valid. An extract() function is then used to separate the title into two columns: title and year.
  str_match(x, pattern) %>%
    str_replace("[()]","") %>%
    str_replace("[)]","") %>%
    str_trim()
}
```

The following function module rounds the time of the rating to the nearest hour so some temporal analysis of the data can be later performed.

```
#Format the timestamp
date_format <- function(date){
  #POSIXct formatting
  #Round the time to the nearest hour in order to group by hour later
  new_date <- as_datetime(date, origin="1970-01-01")
  round_date(new_date, unit = "1 hour")
}
```

The movies and ratings dataset are combined into one dataset. The function to split title and release year is called for each observation. The rating timestamp is converted to date/time format and split into seperate columns by year/month/day/hour. The timestamp is rounded to the nearest hour in order to be later used for temporal data exploratory analysis.

```
movies <- movies %>%
  mutate(release_year = string_format(title),
         film = str_replace(title, pattern, ""))

#Combine the ratings and movies datasets
movielens <- left_join(ratings, movies, by = "movieId")
#Create new features for release year and the rating timestamp
movielens <- movielens %>%
  mutate(rating_timestamp = date_format(timestamp),
         rating_year = year(rating_timestamp),
         rating_month = month(rating_timestamp),
         rating_day = day(rating_timestamp),
         rating_hour = hour(rating_timestamp))
```

4.2 Ignore Unrated Films

```
#Identify films that do not have a rating in the ratings dataset
unrated <- anti_join(movies, ratings, by = "movieId")
dim(unrated)
```

```
## [1] 4 5
```

```
unrated_films <- n_distinct(unrated$movieId)
```

There are **4** unrated movies in the dataset. These movies will not be included in the test validation set.

4.3 Training and Test Set preparation

The training and test set are split in a 90% and 10% sample ratio. The training data set is used to train the model. This is an iterative process of selecting the most appropriate features that better fit the model. The algorithm learns the pattern(s) from the training set enabling the model to be fit. The test set is used to measure how well the model performs on a set of data that was not included in the training set to reduce overfitting.

Overfitting is a situation where the model learned the features of the training data so well that it fails to generalize on other unseen data (Chinnamgari 2019). In an overfitting model, the error, noise or random fluctuations in the training data are considered as true signals by the model that can have an impact on the RMSE. In essence the model works well with the training set but not the test set. The regularization technique has been performed for some of the models to provide an illustration of the benefit gained by penalising the coefficients of the model, to constrain the total variability of the effect sizes (Irizzarry 2019) and better perform on the unseen test set.

Underfitting is when the model is too simplistic and performs poorly on both training and test sets. For example this can be caused when an insufficient amount of features have been selected thereby introducing bias by failing to detect important patterns.

```
# Validation set will be 10% of MovieLens data
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)
```

4.4 Dataset Summary

For obtaining summary information the **str()** command was preferred over **glimpse()** due to processing performance. The MovieLens 10 million record datset requires considerable processing of RAM memory. Therefore all efforts are made to reduce the performance requirements when running the RMarkdown report. Runtime memory is also managed using the **rm()** command.

```
#Use str() or glimmpse() to check the fields that are contained in the training set.
#Using string it is clear that there is a small number of dimensions in the training set.
str(edx)
```

```
## 'data.frame':    9000055 obs. of  13 variables:
##  $ userId          : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId         : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating          : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp       : int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 838983707 838984596 ...
##  $ title           : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres          : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi" ...
##  $ release_year    : chr  "1992" "1995" "1995" "1994" ...
##  $ film            : chr  "Boomerang " "Net, The " "Outbreak " "Stargate " ...
##  $ rating_timestamp: POSIXct, format: "1996-08-02 11:00:00" "1996-08-02 11:00:00" ...
##  $ rating_year     : num  1996 1996 1996 1996 1996 ...
##  $ rating_month    : num  8 8 8 8 8 8 8 8 8 8 ...
##  $ rating_day      : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ rating_hour     : int  11 11 11 11 11 11 11 11 11 11 ...
```

Learning Point: desc() provides additional basic descriptive statistics on the dataset. It is however slower than **summary()** when comparing the processing timings. It can be used in conjuction with tidyverse and produces a more readable output.

```
#Use summary() to check the training set.
summary(edx)
```

```
##      userId        movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres          release_year
##  Length:9000055     Length:9000055     Length:9000055
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##      film           rating_timestamp               rating_year
##  Length:9000055     Min.   :1995-01-09 12:00:00   Min.   :1995
##  Class :character   1st Qu.:2000-01-01 23:00:00   1st Qu.:2000
##  Mode  :character   Median :2002-10-24 21:00:00   Median :2002
##                     Mean   :2002-09-21 13:45:11   Mean   :2002
##                     3rd Qu.:2005-09-15 02:00:00   3rd Qu.:2005
##                     Max.   :2009-01-05 05:00:00   Max.   :2009
##   rating_month      rating_day     rating_hour
##  Min.   : 1.000   Min.   : 1.00   Min.   : 0.00
##  1st Qu.: 4.000   1st Qu.: 8.00   1st Qu.: 5.00
##  Median : 7.000   Median :16.00   Median :14.00
##  Mean   : 6.786   Mean   :15.59   Mean   :12.41
##  3rd Qu.:10.000   3rd Qu.:23.00   3rd Qu.:19.00
##  Max.   :12.000   Max.   :31.00   Max.   :23.00
```

4.5 Check for missing values

Based on the results of the dataset summary there is not need to handle NAs for example by using dummy values, mean value, or omitting observations with NAs. This is also verified with the following command:

```
sapply(edx, function(x) sum(length(which(is.na(x)))))
```

```
##            userId           movieId            rating         timestamp
##                 0                 0                 0                 0
##             title            genres      release_year              film
##                 0                 0                 0                 0
## rating_timestamp       rating_year      rating_month       rating_day
##                 0                 0                 0                 0
##       rating_hour
##                 0
```

A check for duplication in the training set was also performed using the **duplicated()** command to create a vector of logical values. This is run in combination with **table()** to verify if there are any duplicate entries. The duplicate check takes place before the genre variable has been split to a specific film category. The duplicated() statement is process intensive so has been removed from the project. The following syntax was used.

table(duplicated(edx))

4.6 Split Genre to Category

Split the genres in to specific categories in order to assess the data by a single genre rather than a combination of genres. So for example, "Musical|Romance" will be split out to "Musical" and "Romance". The category needs to be split in both the training and test sets as they must have identical features when being compared.

```
edx_categories <- edx %>%
  mutate(category = genres) %>%
  separate_rows(category, sep ="[|]")

#Categories need to be factorised to support plotting
edx_categories$category <- as.factor(edx_categories$category)

#Need to split the validation set to category level.
validation_categories <- validation %>%
  mutate(category = genres) %>%
  separate_rows(category, sep ="[|]")
```

Check the validation test set does not contain any missing values.

```
sapply(validation_categories, function(x) sum(length(which(is.na(x)))))
```

```
##           userId          movieId           rating        timestamp
##                0                0                0                0
##            title           genres     release_year             film
##                0                0                0                0
## rating_timestamp      rating_year     rating_month       rating_day
##                0                0                0                0
##      rating_hour         category
##                0                0
```

```
validation_categories$category <- as.factor(validation_categories$category)
```

Splitting the aggregrated genres to an unique category enables the films to be grouped for summation, plotting and feature analysis. The following table provides a total ratings count by category

```
table(edx_categories$category)
```

```
##
## (no genres listed)             Action          Adventure
##                  7            2560545             1908892
##          Animation           Children             Comedy
##             467168             737994            3540930
##              Crime        Documentary              Drama
##            1327715              93066            3910127
##            Fantasy          Film-Noir             Horror
##             925637             118541             691485
##               IMAX            Musical            Mystery
##               8181             433080             568332
##            Romance             Sci-Fi           Thriller
##            1712100            1341183            2325899
##                War            Western
##             511147             189394
```

```
## Data Preparation: 113.281 sec elapsed
```

5 Data Exploration and Visualisation

There are:

- **4** unrated films not included in the models
- **10677** distinct films that have been rated and assessed
- **69878** unique users have provided ratings

The code is not displayed for all outputs in order to reduce the size of the report.

A mixture of styles are used for the graphics to demonstrate different visualisation settings supported by ggplot2. The report requires the installation of the BBC defined R graphics package [2].

5.1 Movie Ratings

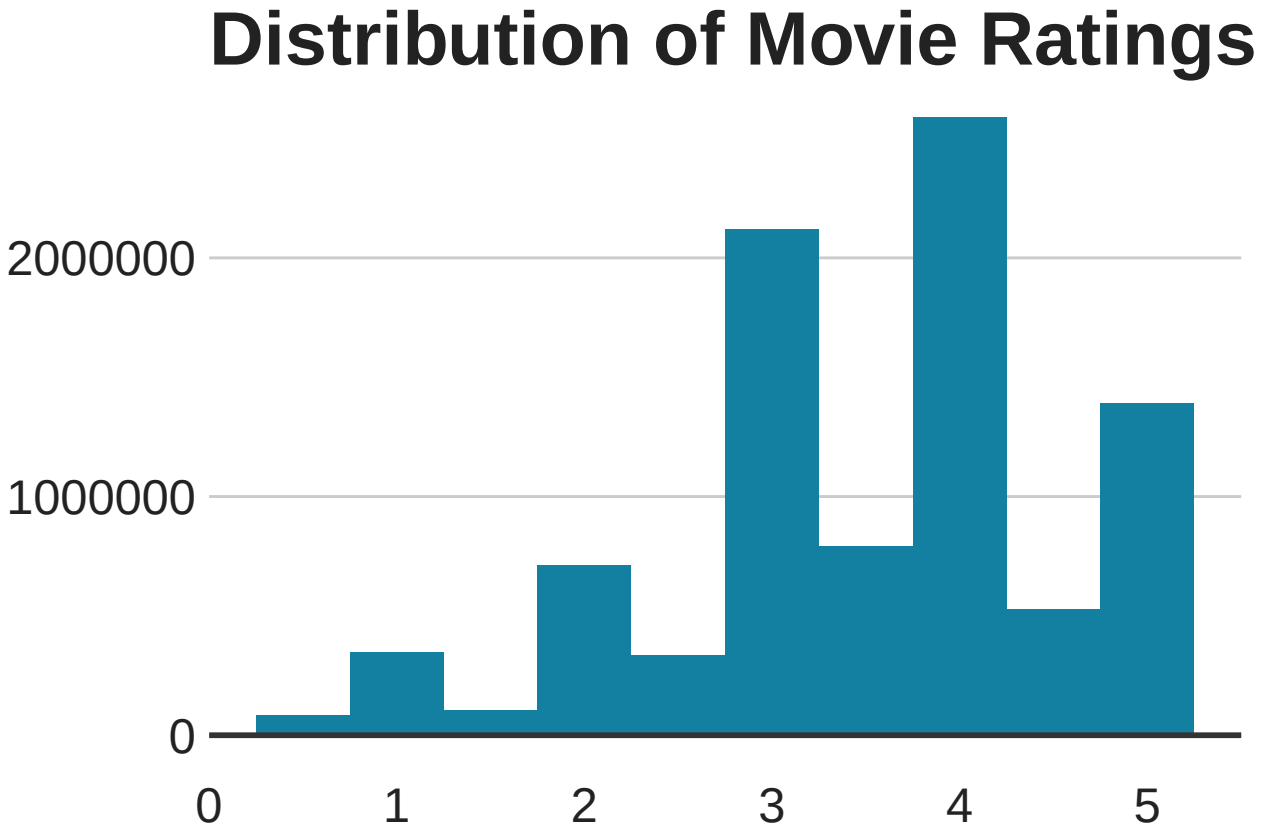
The histogram presents the distribution of the mean movie ratings for all movies rated in the training set.

Learning Point: The histogram calculates the number of occurences and plots on the y-axis. Therefore it is better not to count when the genres have been split to unique categories as this will inflate the count incorrectly. This would not be an issue for a density plot.

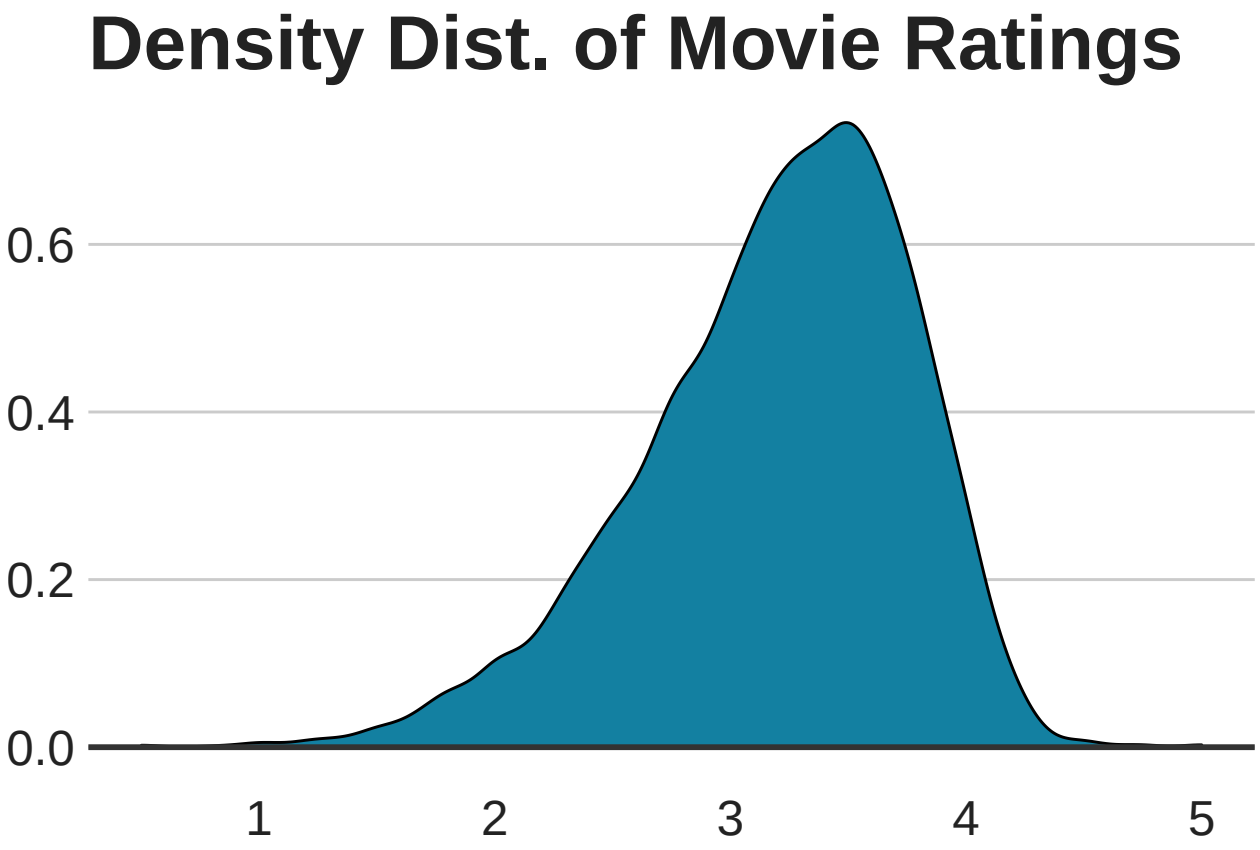
The density plot identifies the right leaning (left skewed) distribution indicating that the mean (**3.5124652**) is typically less than the median (**4**). The values in bold are dynamically calculated in this RMarkdown report. The distributions of both edx_categories and edx datasets were compared to verify that the distribution was similar when the genres where split out to specific categories. The code is not included in this report.

```
#Distribution of Ratings
options(scipen=10000)     #Stop ggplot2 using abbreviations for labels
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(fill="#1380A1", binwidth = 0.5) +
  geom_hline(yintercept = 0, size = 1, colour="#333333") +
  xlab("Movie Rating") +
  labs(title="Distribution of Movie Ratings") +
  bbc_style()
```

[2]The BBC Visual and Data Journalism cookbook for R graphics provides an excellent R package and an R cookbook to make the process of creating publication-ready graphics in the BBC in-house style using R's ggplot2 library.




```
#Distribution of Average Movie ratings
edx_categories %>% group_by(movieId) %>%
  summarise(mean_rating = mean(rating)) %>%
  ggplot(aes(mean_rating)) +
  geom_density(fill="#1380A1") +
  geom_hline(yintercept = 0, size = 1, colour="#333333") +
  xlab("Movie Rating") +
  labs(title="Density Dist. of Movie Ratings") +
  bbc_style()
```

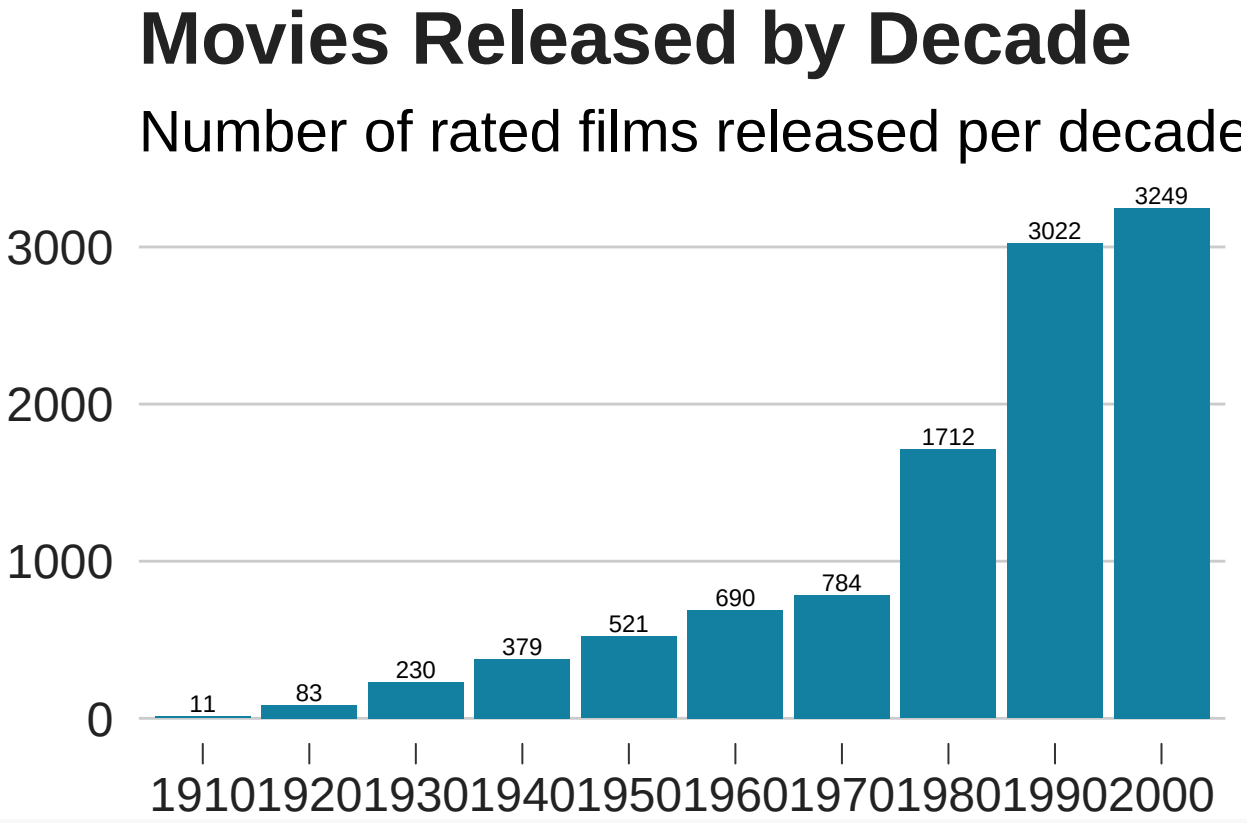

Density Dist. of Movie Ratings
0.6
0.4
0.2
0.0
1
2
3
4
5

5.2 Number of movies released by decade

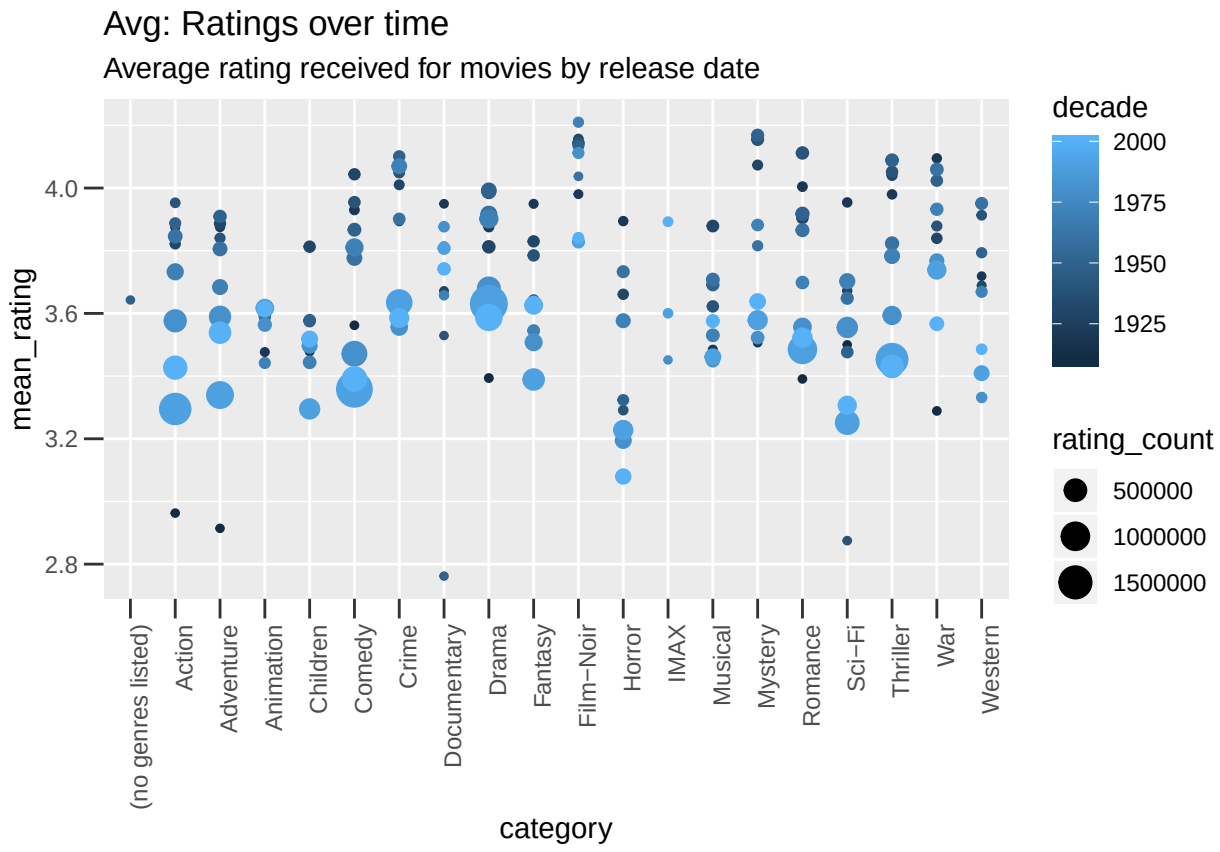
The following Group years by decade using **floor_date()** to improve the readability of the resulting plot. The plot demonstrates that users have rated more recently released movies (past three decades).

However it seems thats the mean rating for those recently released movies tends to be less than earlier released movies. The darker the colour scale indicates the earlier release of the movie.

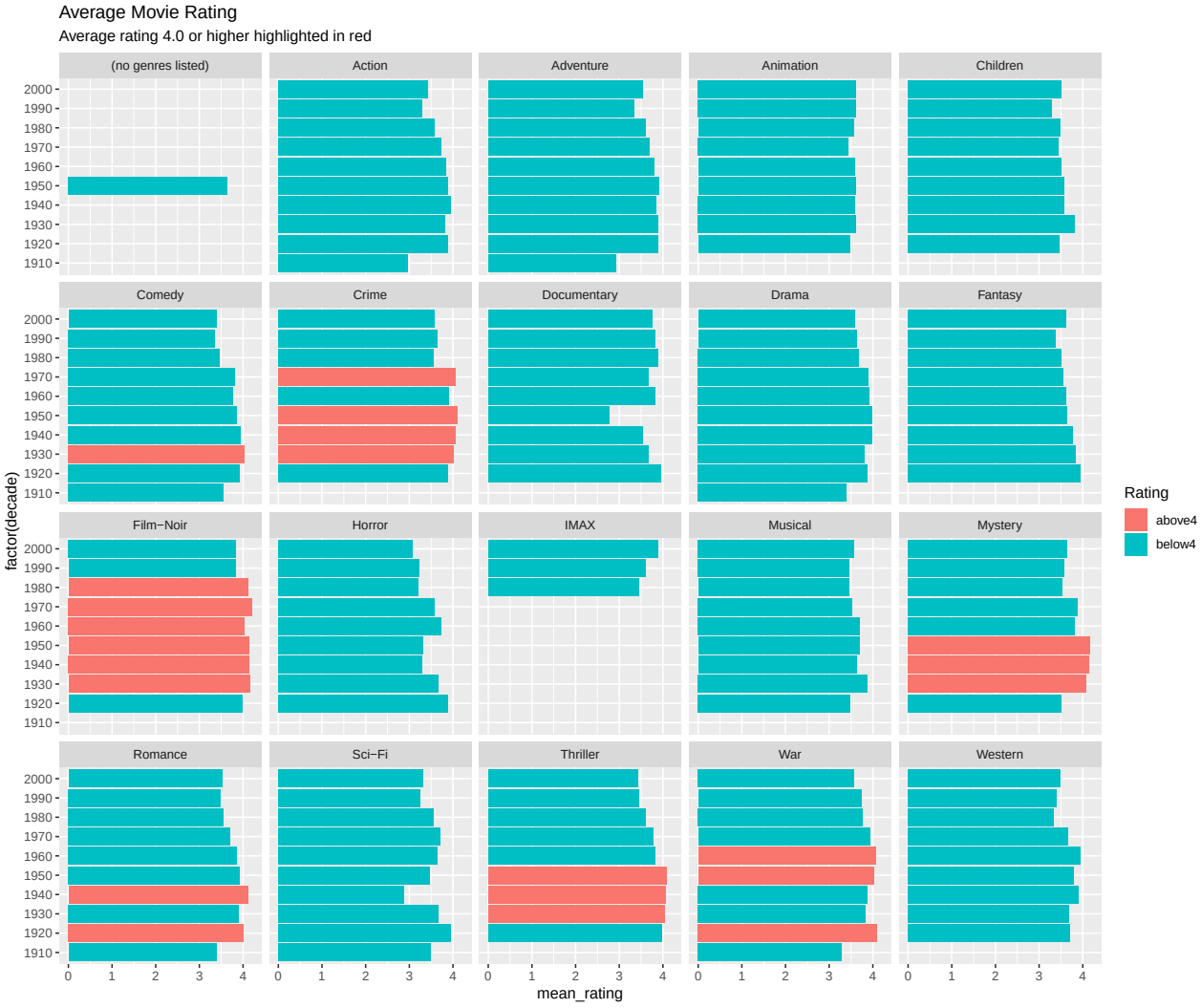
```
#Convert the decades to a discrete (categorical) variable with factor() in order to get the correct labels for geom_bar().
movies %>% mutate(decade = year(floor_date(ISOdate(release_year, 1, 1), unit = "10 years"))) %>%
  group_by(decade) %>%
  summarise(releases = n()) %>%
  ggplot(aes(x=factor(decade), y=releases, label=releases)) +
  geom_col(fill="#1380A1") +
  geom_text(size = 3, position = position_dodge(width = 1), vjust = -0.25) +
  bbc_style() +
  labs(title="Movies Released by Decade",
       subtitle = "Number of rated films released per decade") +
  theme(axis.ticks.x = element_line(colour = "#333333"),
        axis.ticks.length =  unit(0.26, "cm"))
```




```
edx_categories %>% mutate(decade = year(floor_date(ISOdate(release_year, 1, 1), unit = "10 years"))) %>%
  group_by(decade, category) %>%
  summarise(rating_count = n(),
            mean_rating = mean(rating)) %>%
  ggplot(aes(x=category, y=mean_rating, size=rating_count, color=decade)) +
  geom_point() +
  labs(title="Avg: Ratings over time",
       subtitle = "Average rating received for movies by release date") +
  theme(axis.ticks.x = element_line(colour = "#333333"),
        axis.ticks.length =  unit(0.26, "cm"),
        axis.text.x =element_text(angle = 90, hjust = 1))
```


Avg: Ratings over time
Average rating received for movies by release date
mean_rating
4.0
3.6
3.2
2.8
(no genres listed)
Action
Adventure
Animation
Children
Comedy
Crime
Documentary
Drama
Fantasy
Film−Noir
Horror
IMAX
Musical
Mystery
Romance
Sci−Fi
Thriller
War
Western
category
decade
2000
1975
1950
1925
rating_count
500000
1000000
1500000

By introducing faceting of the average movie ratings by category it is easier to identify the temporal pattern. Decades with an average movie rating by category above 4 are highlighted in red. This is a default ggplot2 formatted output. The decades have been factored to support placement on the y-axis. The plot reinforces the general observation that recent movies are not rated as highly as earlier movies.

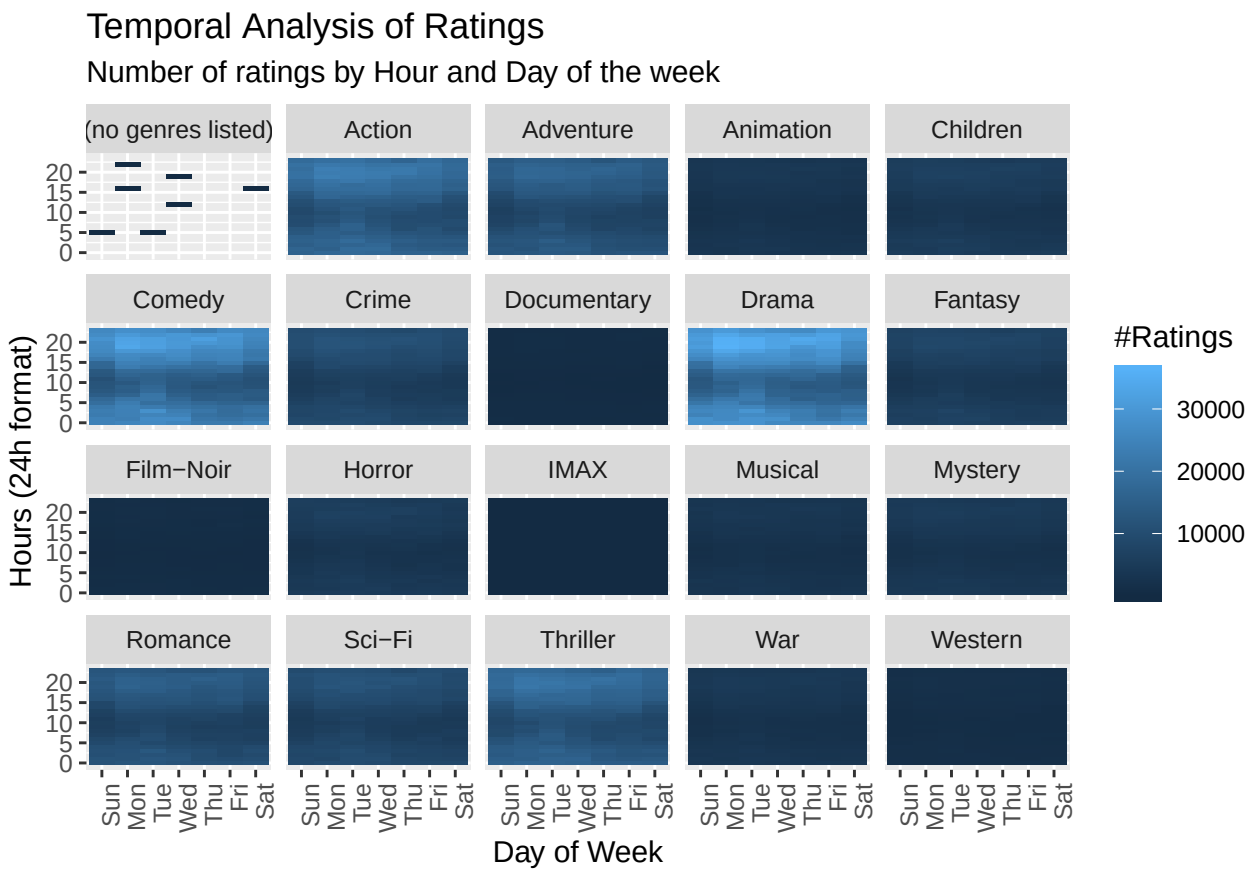


5.3 User Ratings - Temporal Analysis

The Day of week and hour of the user rating is plotted in an heat map and faceted by category in order to:

- Identify potential pattern(s) for the periods of the day and week when users are most likely to record a rating
- Identify from the heat map any temporal pattern by category

```
edx_categories %>% mutate(day_of_week = wday(rating_timestamp, label = TRUE)) %>%
  group_by(day_of_week, rating_hour, category) %>%
  summarise(mean_rating = mean(rating),
            num_ratings = n()) %>%
  ggplot(aes(x=day_of_week, y=rating_hour, fill=num_ratings)) +
  geom_tile() +
  facet_wrap(~category) +
  scale_fill_continuous(name="#Ratings") +
  xlab("Day of Week")+
  ylab("Hours (24h format)")+
  labs(title="Temporal Analysis of Ratings",
       subtitle = "Number of ratings by Hour and Day of the week")+
  theme(axis.text.x =element_text(angle = 90, hjust = 1))
```



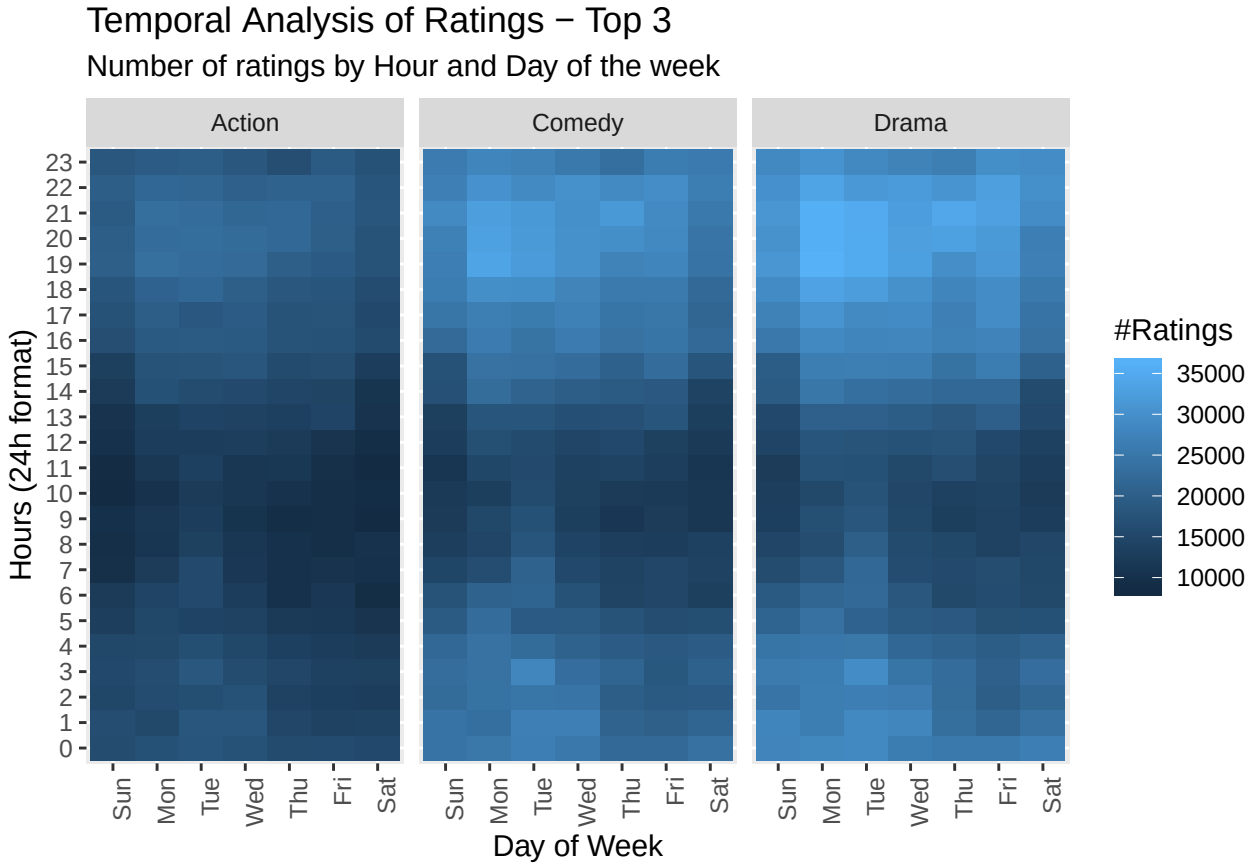
The aim is to look for the lighter coloured heat zones. This is an indication of the larger frequency of rates given by users. From the heat map it is clear that several of the categories cannot be assessed by hour or day of the week. For example Mystery or Westerns. As a result of this the timestamp, rating hour/day/month will not be considered in the models. It is also clear that several categories receive more ratings than others, namely Drama, Action, Comedy, Adventure and Thrillers.

```
#The following tibble supports the findings of the heat map, showing that Comedy,
#Drama and Action receive the largest amount of ratings.
edx_categories %>% group_by(category) %>%
  summarise(mean_rating = mean(rating),
            num_ratings = n()) %>%
  arrange(desc(num_ratings))
```

```
## # A tibble: 20 x 3
##    category           mean_rating num_ratings
##    <fct>                    <dbl>       <int>
##  1 Drama                     3.67     3910127
##  2 Comedy                    3.44     3540930
##  3 Action                    3.42     2560545
##  4 Thriller                  3.51     2325899
##  5 Adventure                 3.49     1908892
##  6 Romance                   3.55     1712100
##  7 Sci-Fi                    3.40     1341183
##  8 Crime                     3.67     1327715
##  9 Fantasy                   3.50      925637
## 10 Children                  3.42      737994
## 11 Horror                    3.27      691485
## 12 Mystery                   3.68      568332
## 13 War                       3.78      511147
## 14 Animation                 3.60      467168
## 15 Musical                   3.56      433080
## 16 Western                   3.56      189394
## 17 Film-Noir                 4.01      118541
## 18 Documentary               3.78       93066
## 19 IMAX                      3.77        8181
## 20 (no genres listed)        3.64           7
```

The top categories (by number of ratings) are displayed by adding **top_n()** to the ggplot. This further reinforces the observation that the majority of ratings are for a small number of categories[3]. The heat map shows a darkened region between the hours of 8:00 and 15:00 indicating that users do not tend to provide ratings during this time period.

[3]A heat map was also generated with the rating month plotted on the y-axis. This plot was not included in the report as it did not add any significant value to the data exploration. The heat map appears to demonstrate an increase in ratings around holiday periods, however this has not been quantified and would need further study



5.4 EDA conclusion

Based on the EDA the following features are assessed and modelled:

- The average rating per movie - movie effect
- The average rating per movie and user effect
- The average rating per movie, user and category effect
- The user, movie and category effect also with regularization

The temporal features (timestamp, rating year/month/day) where not considered important for this report. However the release year will be modelled following the results of running a randomForest model on 10,000 records of the training set which identified the release year with a greater importance than category. Therefore both will be assessed and compared. The randomForest was not included in the report to reduce the performance requirements.

Learning point: Near zero variance was also performed on the features using the **nzv()** command however no benefit was gained over the general performance cost of running the function. Apart from rating and rating year the majority of features are close to unique.

```
## Exploratory Data Analysis: 405.596 sec elapsed
```

6 Modelling

6.1 First Model - Mean

The following model assumes the same rating for all movies and users with all the differences explained by random variation would look like this :

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

Where:

i = Indexing letter for Movies

u = Indexing letter for Users

The overall average rating is denoted by μ with $\epsilon_{u,i}$ independent errors sampled from the same distribution centred at 0 and μ.

```
mu_hat <- mean(edx$rating)
naive_rmse <- RMSE(validation$rating, mu_hat)
naive_rmse
```

```
## [1] 1.061202
```

```
rmse_results <- tibble(method = "Average Rating", RMSE = naive_rmse)
```

$\hat{\mu}$ = 3.5124652 and the calculated RMSE is 1.0612018.

```
## Model 1: Predict the same rating for all movies regardless of user: 0.137 sec elapsed
```

6.2 Second Model - Measuring the Movie Effect

The following function creates entries in a tibble that contains all the results of each model assessed. It will be printed in the results section of this report.

```
#Calculate the average rating for each movie and calculate the bias (difference) for each
add_results <- function(n_rmse, model_method){
  bind_rows(rmse_results,
            tibble(method=model_method,
                       RMSE = n_rmse))
}
```

This model is extended to consider the average ranking for each movie b_i referred to as an effect or bias.

$$\mathrm{Y}_{u,i} = \mu + b_i + \epsilon_{u,i}$$

The bias $\hat{b_i}$ is calculated as the mean difference between the movie ratings in the training set y_i and the mean average for all movies, $\hat{\mu}$, for all observations n_i of movie$i$ in the training set.

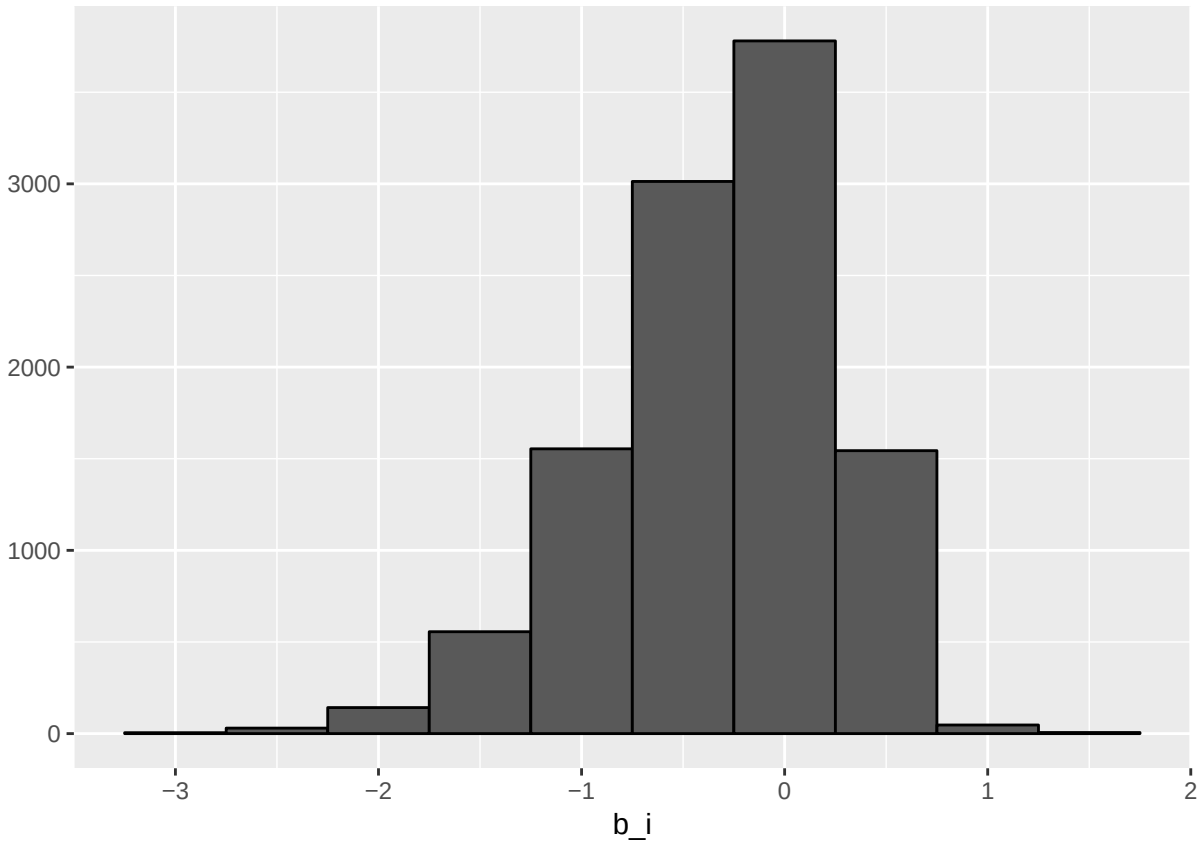
$$\hat{b}_i = \frac{1}{n_i}\sum_{i=1}^{n_i}(y_i - \hat{\mu})$$

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

```
## # A tibble: 6 x 2
##   movieId    b_i
##     <dbl>  <dbl>
## 1       1  0.415
## 2       2 -0.307
## 3       3 -0.365
## 4       4 -0.648
## 5       5 -0.444
## 6       6  0.303
```

The following plot illustrates the difference distribution for the movie effect.

```
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



A predicted value can now be determined using $\hat{y}_i = \hat{\mu} + \hat{b}_i$ for each observation in the test set and compared against the actual value y_i to evaluate the RMSE.

```
predicted_ratings <- mu_hat + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
```

The calculated RMSE is 0.9439087.

```
## Model 2: Predict using the movie effect: 1.708 sec elapsed
```

6.3 Third Model - Measuring the Movie and User Effect

This model is extended to consider the average ranking for each user b_u referred to as an effect or bias.

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

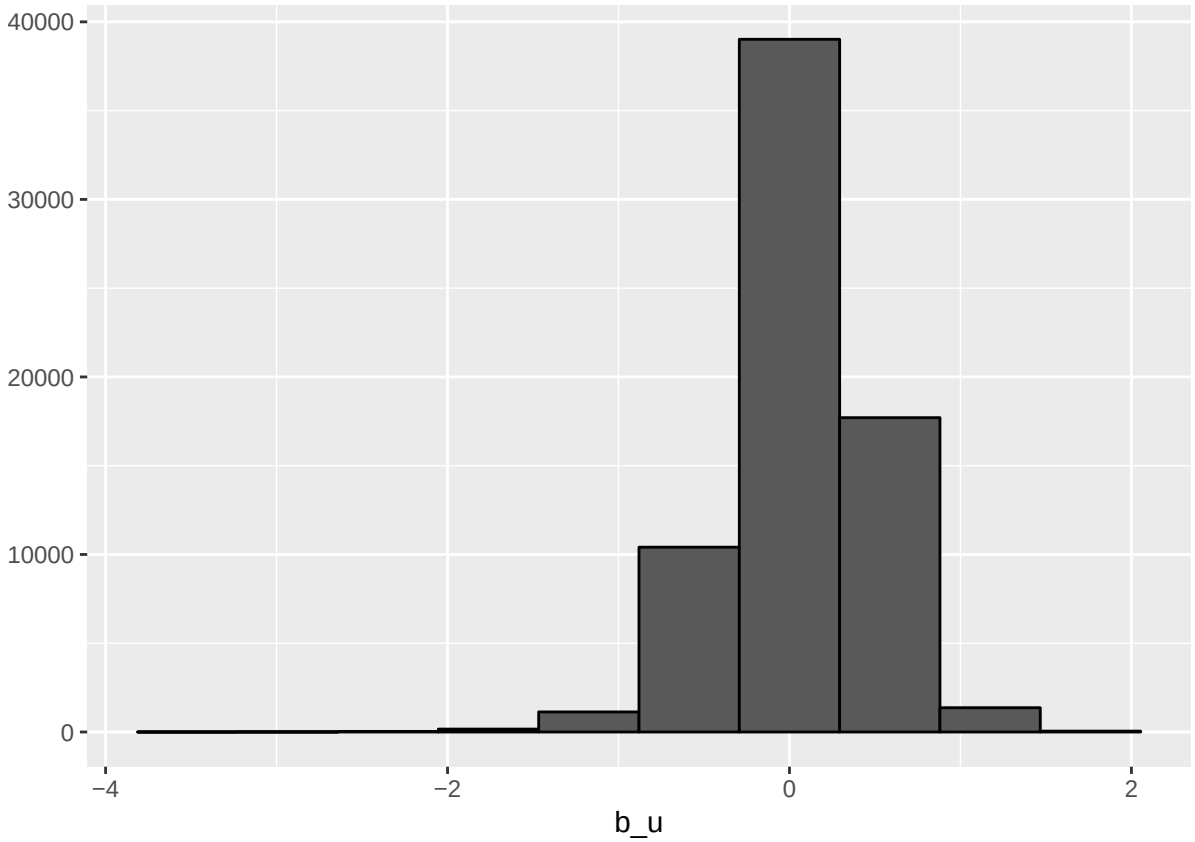
The bias $\hat{b}_u$ is calculated as the mean difference between the movie ratings in the training set y_i and the mean average for all movies $\hat{\mu}$ also taking in to consideration the calculated movie bias $\hat{b}_i$, for all observations n_i. For example, if a movie tends to be rated above the average then the prediction would reflect this by adding a positive bias $\hat{b}_i$ to the $\hat{y}_i$ predicted. However if the user providing the rating tends to score results harshly then $\hat{y}_i$ would be adjusted by the user effect/bias $\hat{b}_u$.

$$\hat{b}_u = \frac{1}{n_i}\sum_{i=1}^{n_i}(y_i - \hat{\mu} - \hat{b}_i)$$

```
#Calculate the user effect using the training set
user_movie_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))
```

The following plot illustrates the difference distribution for the movie effect.

```
user_movie_avgs %>% qplot(b_u, geom ="histogram", bins = 10, data = ., color = I("black"))
```



A predicted value can now be determined using $\hat{y}_i = \hat{\mu} + \hat{b}_i + \hat{b}_u$ for each observation in the test set and compared against the actual value y_i to evaluate the RMSE.

```
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_movie_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)
```

The calculated RMSE is 0.8653488.

```
## Model 3: Predict using the movie + user  effect: 7.935 sec elapsed
```

6.4 Fourth Model - Measuring the Movie, User and Category Effect

This model is extended to consider the effect of the movie category b_c. The category is the unique genre that was split out from the aggregated genres field in the movies.dat file.

$$Y_{u,i,c} = \mu + b_i + b_u + b_c + \epsilon_{c,u,i}$$

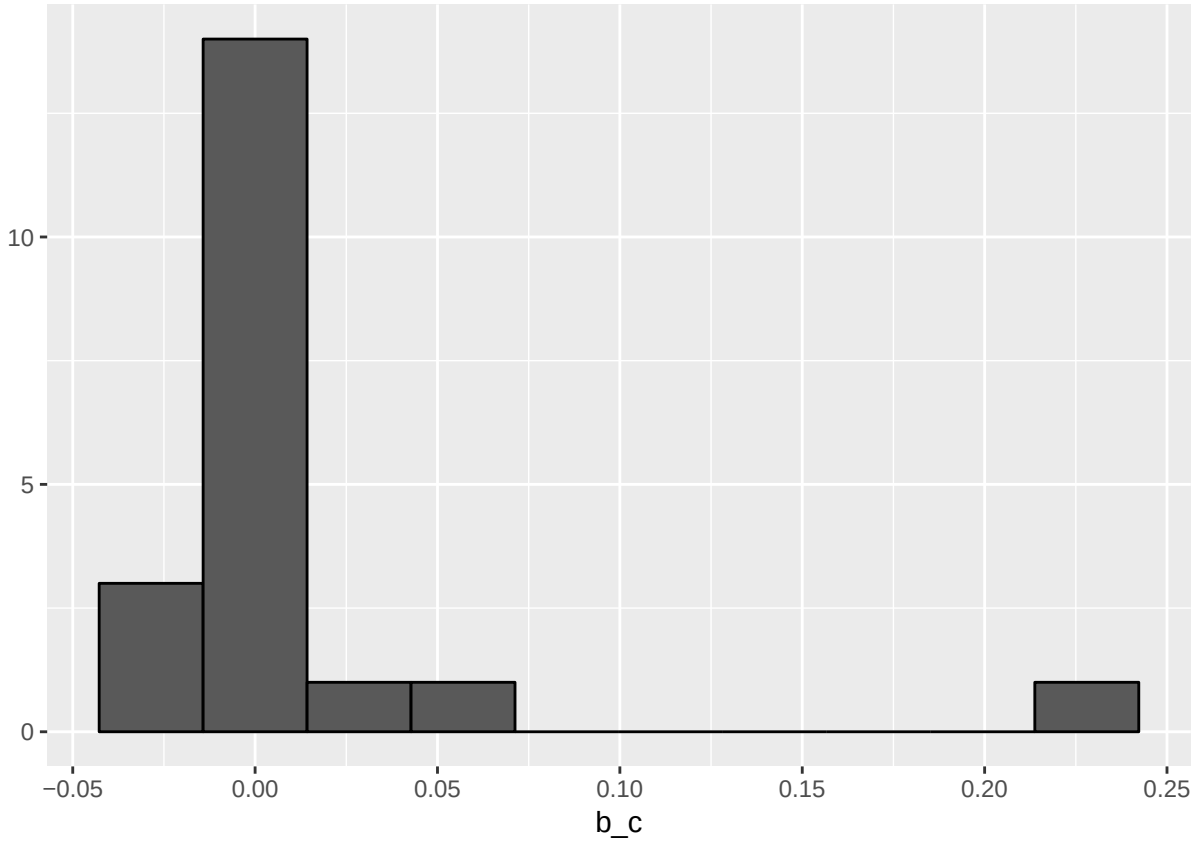
As in the previous models the bias is calculated with consideration of the mean average for all movies $\hat{\mu}$ and the bias introduced by the movie and user effects.

$$\hat{b}_c = \frac{1}{n_i} \sum_{i=1}^{n_i} (y_i - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

```
category_avgs <- edx_categories %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_movie_avgs, by='userId') %>%
  group_by(category) %>%
  summarize(b_c = mean(rating - mu_hat - b_i - b_u))
```

The following plot illustrates the difference distribution for the category effect and indicates that adding the category feature will not alter the predicted rating by any significant amount.

```
category_avgs %>% qplot(b_c, geom ="histogram", bins = 10, data = ., color = I("black"))
```



The plot identifies an outlier category that will impact the prediction. It is a quite small amount though should be checked to identify the category in question. The temporal analysis highlighted a genre (no genres listed).

```r
# Identify the outlier, count the number of movies in this outlier
category_avgs %>% filter(b_c > 0.1) %>%
  mutate(category = as.character(category)) %>%
  left_join(edx, by = c("category" = "genres")) %>%
  group_by(title, category) %>%
  summarise(num_ratings = n(), avg_rating = mean(rating))
```

```
## # A tibble: 1 x 4
## # Groups:   title [1]
##   title                category           num_ratings avg_rating
##   <chr>                <chr>                    <int>      <dbl>
## 1 Pull My Daisy (1958) (no genres listed)           7       3.64
```

Remove the category (no genres listed) from the prediction. **Learning point:** remove this type of exception during data cleanse in future

```r
category_avgs <- category_avgs %>% filter(category != "(no genres listed)")
```

The predicted value is determined using $\hat{y}_i = \hat{\mu} + \hat{b}_i + \hat{b}_u + \hat{b}_c$ for each observation in the test set and compared against the actual value y_i to evaluate the RMSE.

```r
predicted_ratings <- validation_categories %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_movie_avgs, by='userId') %>%
  left_join(category_avgs, by='category') %>%
  mutate(pred = mu_hat + b_i + b_u + b_c) %>%
  pull(pred)
```

The calculated RMSE is 0.8631334.

```
## Model 4: Predict using the movie + user + category effect: 40.204 sec elapsed
```

6.5 Fifth Model - Measuring the Movie, User and Release Year Effect

The release year is assessed in order to contrast with the Category effect to determine the best feature to use in the final model.

This model is extended to consider the effect of the movie category b_r. The category is the unique genre that was split out from the aggregated genres field in the movies.dat file.

$$Y_{u,i,r} = \mu + b_i + b_u + b_r + \epsilon_{r,u,i}$$

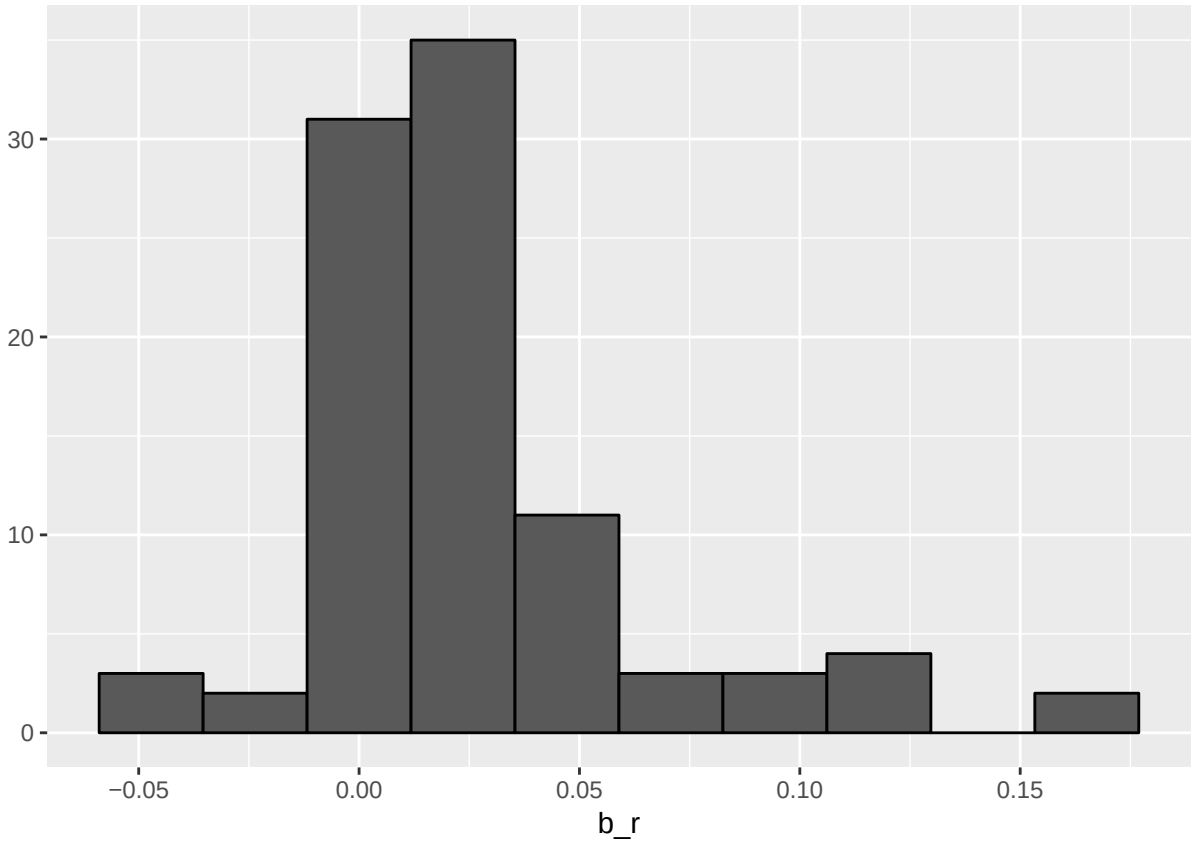
As in the previous models the bias is calculated with consideration of the mean average for all movies $\hat{\mu}$ and the bias introduced by the movie and user effects.

$$\hat{b}_r = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

```
#Calculate the bias introduced by release year
release_yr_avgs <- edx_categories %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_movie_avgs, by='userId') %>%
  group_by(release_year) %>%
  summarize(b_r = mean(rating - mu_hat - b_i - b_u))
```

The following plot illustrates that the distribution of the difference for the Release year is close to the mean, and the effect will be mostly positive on the predicted rating, although by a small margin.

```
release_yr_avgs %>% qplot(b_r, geom ="histogram", bins = 10, data = ., color = I("black"))
```



The predicted value is determined using $\hat{y}_i = \hat{\mu} + \hat{b}_i + \hat{b}_u + \hat{b}_r$ for each observation in the test set and compared against the actual value y_i to evaluate the RMSE.

```
predicted_ratings <- validation_categories %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_movie_avgs, by='userId') %>%
  left_join(release_yr_avgs, by='release_year') %>%
  mutate(pred = mu_hat + b_i + b_u + b_r) %>%
  pull(pred)
```

The calculated RMSE is 0.8628589.

```
## Model 5: Predict using the movie + user + release year effect: 40.14 sec elapsed
```

6.6 Sixth Model - Movie Effect with Regularization

Regularization is applied to the movie effect model to generalise the overall patterns (Hodnett 2018). The general idea behind regularization is to constrain the total variability of the effect sizes (Irizzarry 2019) and improve the precision of the model and reduce overfitting.

The following tables highlight the issue. Movies that have only one, or a very small number of ratings, exist in the dataset. The estimate/prediction will be based on only a small number of observations and therefore will not be of much use.

```
#The following are the best 10 films before regularisation
validation %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movies, by="movieId") %>%
  arrange(desc(b_i), n) %>%
  select(title, b_i, n) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 3
##    title                                                     b_i     n
##    <chr>                                                   <dbl> <int>
##  1 Hellhounds on My Trail (1999)                           1.49      1
##  2 More (1998)                                             1.20      1
##  3 Valerie and Her Week of Wonders (Valerie a týden divu) (197~ 0.988     1
##  4 Kansas City Confidential (1952)                         0.988     1
##  5 Shawshank Redemption, The (1994)                        0.943  3111
##  6 Red Desert, The (Deserto rosso, Il) (1964)              0.904     1
##  7 Godfather, The (1972)                                   0.903  2067
##  8 Man Who Planted Trees, The (Homme qui plantait des arbres, ~ 0.888     2
##  9 Usual Suspects, The (1995)                              0.853  2389
## 10 Schindler's List (1993)                                 0.851  2584
```

```
#The following are the worst 10 films before regularisation
validation %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movies, by="movieId") %>%
  arrange(b_i, n) %>%
  select(title, b_i, n) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 3
##    title                                                     b_i     n
##    <chr>                                                   <dbl> <int>
##  1 Confessions of a Superhero (2007)                       -3.01     1
##  2 War of the Worlds 2: The Next Wave (2008)               -3.01     1
##  3 SuperBabies: Baby Geniuses 2 (2004)                     -2.72     5
##  4 Disaster Movie (2008)                                   -2.65     8
##  5 From Justin to Kelly (2003)                             -2.61    17
##  6 Criminals (1996)                                        -2.51     2
##  7 Mountain Eagle, The (1926)                              -2.51     2
##  8 When Time Ran Out... (a.k.a. The Day the World Ended) (1980) -2.51     2
##  9 Pokémon Heroes (2003)                                   -2.48    19
## 10 Roller Boogie (1979)                                    -2.48     2
```

A weighting is applied to give a stable estimate when n_i observations are a large number, and shrink the estimate $\hat{b}_i$ towards 0 given a small number of n_i observations. The strength of the penalty is controlled by the parameter λ which is identified by repeating multiple computations of the estimate to identify the

optimal λ setting.

The model being fitted is

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

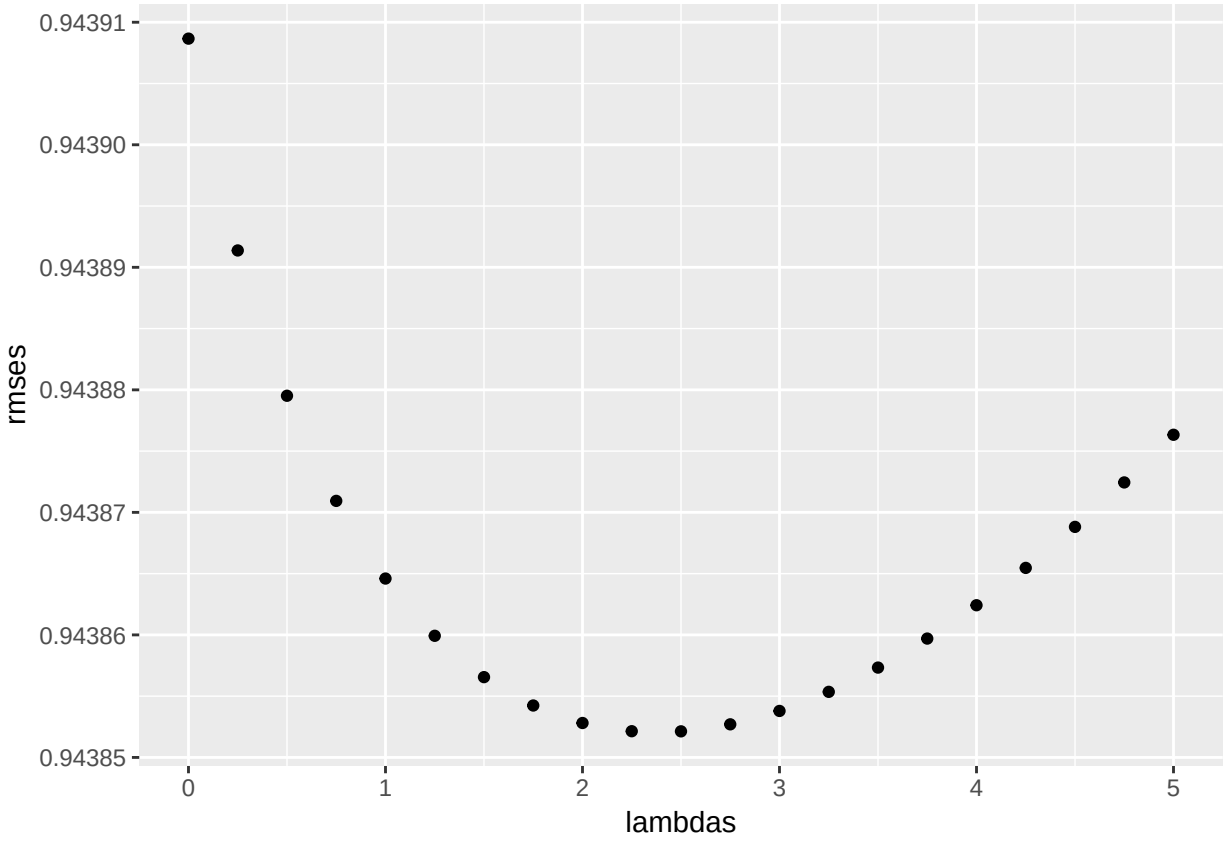
The equation to calculate the bias b_i considering the weighting is:

$$\hat{b}_i(\lambda) = \frac{1}{n_i + \lambda} \sum_{i=1}^{n_i} (y_i - \hat{\mu})$$

sapply() is used to run several iterations of an RMSE computation in order to identify the best weighting factor to use.

```
lambdas <- seq(0, 5, 0.25)
movie_reg <- edx %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu_hat), n_i = n())

rmses <- sapply(lambdas, function(l){
  predicted_ratings <- validation %>%
    left_join(movie_reg, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu_hat + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))
})
```



The optimal weighting factor to use is: **2.5**.

The calculated RMSE is 0.9438521.

```
## Model 6: Movie Effect regularization: 17.661 sec elapsed
```

6.7 Seventh Model - Regularization of Model 4

The model being fitted is

$$Y_{u,i,c} = \mu + b_i + b_u + b_c + \epsilon_{c,u,i}$$

The equation to calculate the bias b_c considering the weighting is:

$$\hat{b}_c(\lambda) = \frac{1}{n_i + \lambda} \sum_{i=1}^{n_i} (y_i - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

Lambdas is set to 15 after having run severals iterations from 0 to 20 on the 10 million MovieLens dataset. This is not included in this report to improve runtime performance. λ is set as a fixed constant while the original code to determine the optional weighting is retained.

```
lambdas <- 15
rm(rmses)
rmses <- sapply(lambdas, function(l){
  b_i <- edx_categories %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat)/(n()+l))
  b_u <- edx_categories %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_hat)/(n()+l))
  b_c <- edx_categories %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(category) %>%
    summarize(b_c = sum(rating - b_i - b_u - mu_hat)/(n()+l))

  predicted_ratings <- validation_categories  %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_c, by = "category") %>%
    mutate(pred = mu_hat + b_i + b_u + b_c) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation_categories$rating))
})
lambdas[which.min(rmses)]
```

```
## [1] 15
```

```
rmse_results <- add_results(rmses[which.min(rmses)], "Movie + User + Category Effect Model Regularized")
```

The calculated RMSE is 0.8626674.

```
## Model 7: Movie, User and Category Effect regularization: 65.108 sec elapsed
```

7 Results Section

The results of the seven models assessed are listed as follows:

method	RMSE
Average Rating	1.0612018
Movie Effect Model	0.9439087
Movie Effect Model Regularized	0.9438521
User + Movie Effect Model	0.8653488
User + Movie + Category Effect Model	0.8631334
User + Movie + Release Effect Model	0.8628589
Movie + User + Category Effect Model Regularized	**0.8626674**

8 Conclusion

This section contains the summary of key main findings from the MovieLens Capstone project. The course was an excellent introduction to Data Science and R and this project in particular was an interesting topic to study.

8.1 Model selection

The best RMSE computed was for **Model 7, the prediction based on the regularized movie, user and category effects**. However it was noticable that the regularization did not greatly alter the RMSE and category could easily be substituted with the release year and the result would be extremely close (the RMSE are both < 0.864).

The effect of using category (unique genres) or release year was assessed and it can be concluded that there was a marginal improvement when using release year in place of category when modelling for the larger dataset. However the difference in RMSE was not significant enough to use release year in the final model. Category was a better option when running the project using the 1 million MovieLens dataset, however also not by any notable RMSE difference.

8.2 Training v Test Split

The RMSE calculated for the larger 10 million dataset was much better than the 1 million dataset. However the difference between the use of 90% or 80% training set resulted in a different conclusion. For the 1 million dataset 80% was better. For the larger dataset 90% gave a better RMSE.

Table 3: Comparison of RMSE computed for different test set and MovieLens file sizes

MovieLens File	Validation Test	RMSE
1 Million	10%	**0.9079764**
	20%	0.9087210
10 Million	10%	0.8639040
	20%	**0.8626674**

8.3 Dataset Size and performance

Comparing the MovieLens 1 million and 10 million record datasets provided me some valuable learning lessons. Undoubtably the main issue is performance optimisation. Running the project using a 1 million record dataset was relatively fast using an 8GB RAM iMac, MacOS Mojave,3.4 GHz Intel Core i5 processor. However once the 10 million dataset was loaded the performance became a critical issue. As a result some of the initial report modules had to be removed. For example running randomForest analysis, analysing and plotting remake versions of films, or regularisation iteration for model 7.

The regularization weighting, λ, varied significantly between the 1 million and 10 million record datasets. Therefore it is recommended to alter the range of the sequence that is used to determine λ when assessing different dataset sizes. For model 7 it has been set to 15 after running several iterations to identify the optimal value to be used. When running on 1 million records the λ is approximately 5.0. Quite a difference from 15.

The 1 million dataset does not contain any half ratings, e.g. no film has a 3.5 rating. Also there was a great number of unrated movies in the 1 million dataset than the larger 10 million dataset.

Thank you for reading and assessing this submission for the HarvardX PH125.9x Professional Certificate in Data Science capstone project. I course was an excellent introduction to Data Science and R Programming and I would like to greatly thank Professor Rafael A. Irizarry, the course staff team, edx platform teams and the other course students who have provided great solutions and advice in the course discussion forums.

Appendix A: Remakes - Better the last time?

The following section was included only for information purposes and is not considered in any of the models assessed. It only included in the RMarkdown report and has been removed from the R code. It was included only to experience and demonstrate the following R programming skills:

- Filtering and grouping with R
- Create a dumbbell ggplot to identify variance over time

First identify any movies that have the same title. This clearly introduces some bias as it is entirely possible that the movies can have the same title but are not remakes. Only the first and last versions of the same movie title are assessed. A further study could extend to include all versions of the same movie.

```
remakes <- edx_categories %>%
  filter(category == "Action") %>%
  group_by(film, release_year) %>%
  summarise(ratings_count = n(),
            mean_rating = mean(rating)) %>%
  group_by(film) %>%
  filter(n()>1)
```

There are **25** Action movies in the training set that have been remade. The following ggplot lists all remakes and release years. It is possible to use all categories for a small dataset (MovieLens 1 million ratings) however becomes unreadable when using a larger dataset (Movielens 10 million ratings). Therefore the remakes are plotted only for one category.

The first and last releases are determined using **min()** and **max()** commands. The first release is referred to as the "original" and the last release as the "latest". All other releases are skipped.

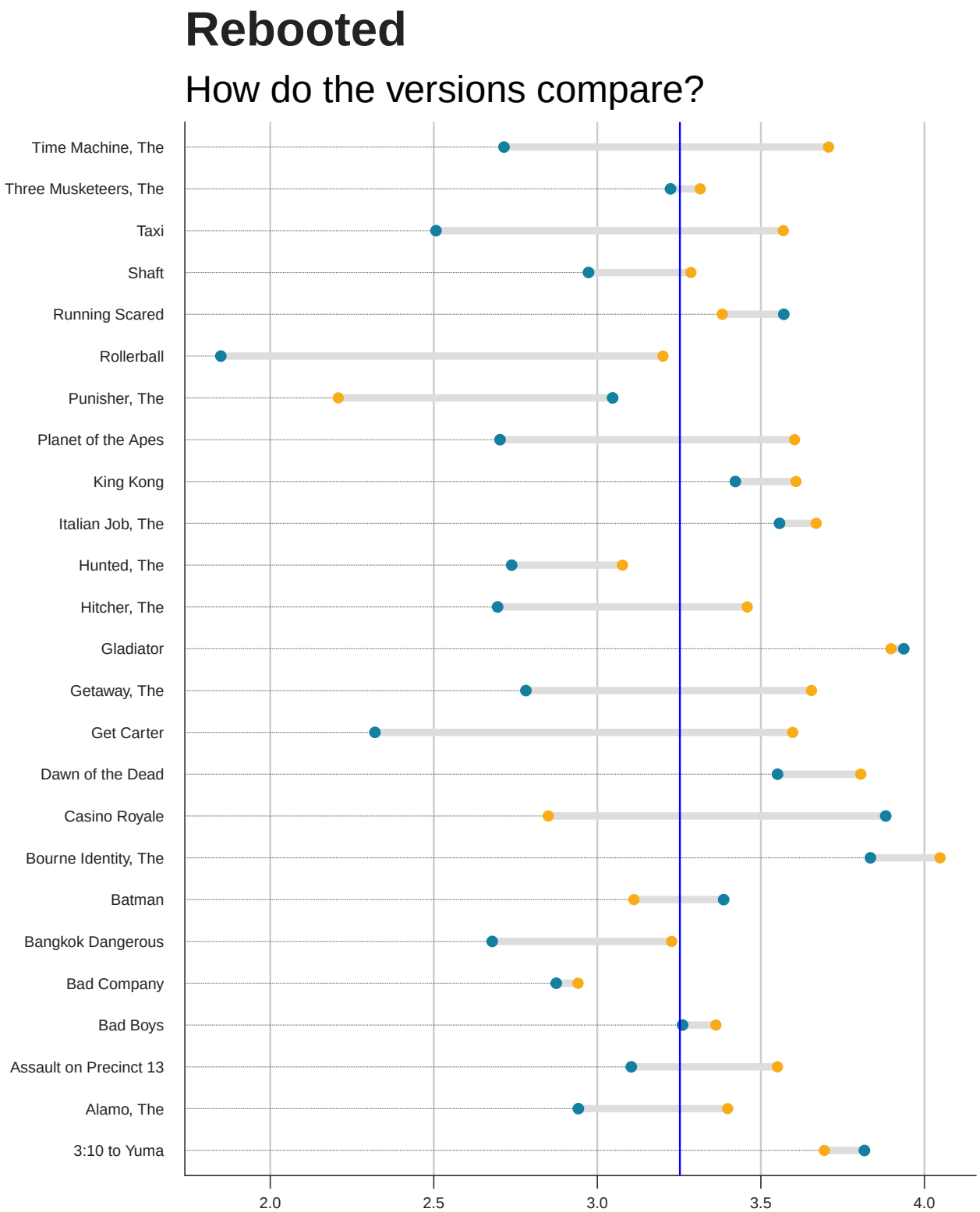
```
#Find the year with the minimum mean_rating and the year with the maximum
#Check which is better, the earlier or later version
```

```
#Flag the result and highlight in a dumbell.
dubbell_edx <- remakes %>%
  group_by(film) %>%
  mutate(version = ifelse(release_year == min(release_year), "original", ifelse(release_year == max(release_year), "latest", "skip"))) %>%
  filter(version %in% c("original","latest")) %>%
  select(film, mean_rating, version) %>%
  spread(version, mean_rating) %>%
  mutate(difference = latest-original)
```

The dubbell_edx dataframe is passed to ggplot command with the dumbbell geometry added. The original and latest versions are added to the x-axis in the aesthetic. The films are added to the y-axis. The average movie rating *for the remakes* is the blue intercept line to provide a context for the distribution of the original and remake,

The original movie is identified by an orange dot while the last remake is identified in blue.

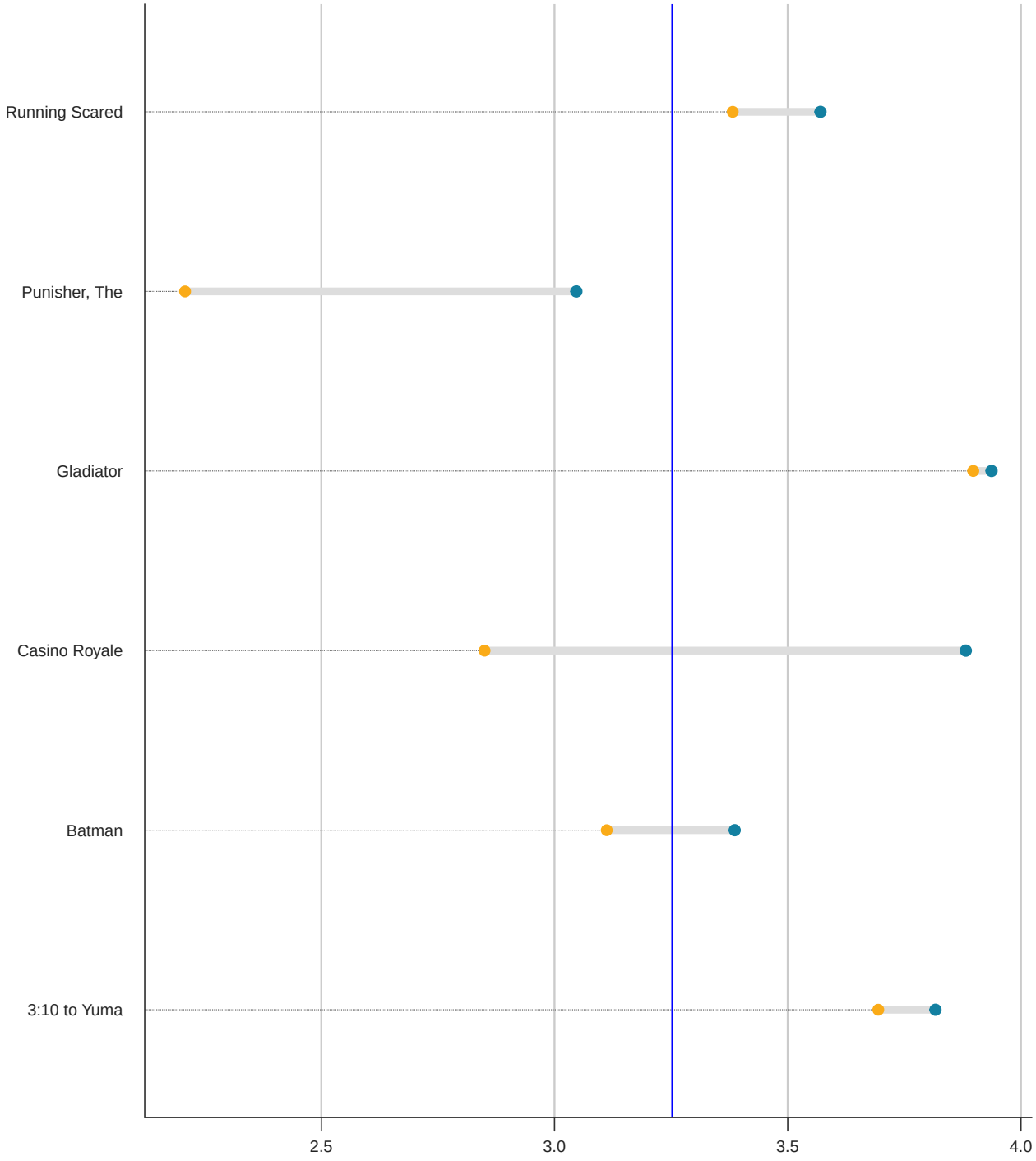
```
#Use a dumbbell_geom() to compare the ratings of original versions and the latest remake
dubbell_edx %>%
  ggplot(aes(x = original, xend = latest, y = film)) +
  geom_dumbbell(colour = "#dddddd",
                size = 2,
                colour_x = "#FAAB18",
                colour_xend = "#1380A1",
                dot_guide=TRUE, dot_guide_size=0.25,
                show.legend = TRUE) +
  geom_vline(xintercept = mean(remakes$mean_rating), color="blue") +
  xlab("Average Movie Rating")+
  bbc_style() +
  theme(panel.grid.major.x = element_line(color="#cbcbcb"),
        panel.grid.major.y = element_line(FALSE),
        axis.line.x = element_line(colour = "#333333"),
        axis.line.y = element_line(colour = "#333333"),
        axis.text = element_text(size = 9),
        axis.ticks.x = element_line(colour = "#333333"),
        axis.ticks.length =  unit(0.26, "cm")) +
  labs(title="Rebooted",
       subtitle="How do the versions compare?")
```



When not filtering by category the plot becomes cluttered and difficult to read. Therefore to ensure that the plots are meaningful and accurate it is better to ask a specific question: **which remakes were better rated than their original version?**. This is achieved by using **filter()** on the dubbell_edx object where the mean rating of the latest film is greater than the mean rating of the original. The original movie is identified by an orange dot while the last remake is identified in blue.

Rebooted Successfully!

The remake has better ratings



9 References

Please note that the references had to be added manually as it was not an option to include a .bib file. It is also possible to add a reference inline in the YAML header though this was not preferred.

Chinnamgari, Sunil. 2019. *R Machine Learning Projects Implement Supervised, Unsupervised, and Reinforcement Learning Techniques Using R 3. 5.* Birmingham: Packt Publishing Ltd.

Hodnett, Mark. 2018. *R Deep Learning Essentials : A Step-by-Step Guide to Building Deep Learning Models Using Tensorflow, Keras, and Mxnet.* Birmingham, UK: Packt Publishing.

Irizzarry, Rafael A. Prof. 2019. *Introduction to Data Science.* https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems.

Voulgaris, Zacharias. 2017. *Data Science Mindset, Methodologies, and Misconceptions.* City: Technics Pubns Llc.

Wickham, Hadley, and Garrett Grolemund. 2017. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data.* 1st ed. O'Reilly Media, Inc.