CAPSTONE PROJECT: MOVIELENS

Saúl Santillán Gutiérrez

September 10, 2022

Contents

General Instructions by HarvardX's Team

You will be creating your own recommendation system using all the tools we have shown you throughout the courses in this series. We will use the 10M version of the MovieLens dataset to make the computation a little easier. The links to download the **10M version of the MovieLens dataset** are:

GroupLens 10M MovieLens Dataset's Home page.
10M MovieLens Dataset's Download page.

Develop your algorithm using the `edx` set. For a final test of your final algorithm, predict movie ratings in the `validation` set (the final hold-out test set) as if they were unknown. **"RMSE"** will be used to evaluate how close your predictions are to the true values in the `validation` set (the final hold-out test set).

IMPORTANT: The `validation` data (the final hold-out test set) should **NOT** be used for training, developing, or selecting your algorithm and **it should ONLY be used for evaluating the RMSE of your final algorithm**. The final hold-out test set should only be used at the end of your project with your final model. **It may not be used to test the RMSE of multiple models during model development**. You should split the `edx` data into separate training and test sets to design and test your algorithm.

IMPORTANT: Please be sure not to use the `validation` set (the final hold-out test set) for training or regularization - you should create an additional partition of training and test sets from the provided `edx` dataset to experiment with multiple parameters or use cross-validation.

Remember your goal is to get a **RMSE < 0.86490**.

Preliminary

The present report belongs to the capstone project **MovieLens of the HarvardX's Data Science Professional Certificate** and it is composed of 4 chapters and by an extra section called Appendix, which are described as follows: Chapter 1 Overview describes the dataset, summarizes the goal of the project and key steps that were performed. In Chapter 2 Methods and Analysis explains the process and techniques used, for example, data science project preparation, data exploration and visualization, preparation and cleaning of the training and testing datasets. Also, the insights gained during the course, and the modeling approach used. Chapter 3 Results presents the modeling results and discusses the model performance. In Chapter 4 Conclusion gives a brief summary of the report, its limitations and future work. And finally, in the extra section **Appendix**, that is divided into two parts, shows the information about the computer equipment used, its hardware specs, `R` session, the Operating Systems (OS) where this code was made and tested, and the loaded packages.

Chapter 1

Overview

In the last four years, during and after the Covid-19 pandemic, people like you and me around the world increase the use of internet, and at the same time, visit online services and e-commerce sites more frequently, either to: buy a product on Amazon, view or rent a movie or series on Netflix, read a good book on Kindle, rent a house or apartment on Airbnb, buy plane tickets on Booking.com, watch tutorial or musical videos on YouTube, listen to our favorite music on Spotify, etc. All those online services that we mentioned and many more have something in common, they all use what we know as **recommendation systems**. But, what is a recommendation system?

A **recommendation system** is *"a subclass of Information filtering Systems that seeks to predict the rating or the preference a user might give to an item"* as defined by the author Agrawal (2021). Another concept is the one that our teacher, Irizarry (2022), provided us in the course, where he mentions that *"recommendation systems use ratings that users have given items to make specific recommendations"*. In other words, it is an algorithm that proposes or recommends notable or outstanding items to the users, according to their preferences (ratings) made in the past, to predict the rating for a new item.

Now, the following question arises, and how they work? The recommendation systems work as follows as described Dilmegani (2017) *"collect customer data and auto analyze this data to generate customized recommendations for your customers"*. Also, these systems are generally based on two types of data: **implicit data**, such as navigation history and purchases, and **explicit data**, such as gradings supplied by an user.

For all the reasons stated above, the recommendation systems play an important role within the e-commerce and online services industries to help us achieve a wonderful user experience when buying a product, renting a movie, reading a book, watching a music video, listening to music, etc. For that, in this report we build a movie recommendation system using the **MovieLens** dataset applying our knowledge acquired in the lessons of the **HarvardX's Data Science Professional Certificate**.

1.1 About Movielens Dataset

GroupLens is a research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities specializing in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems.

GroupLens Research has collected and made available rating data sets from the MovieLens web site. We can obtain different kind of datasets according ours requirements by visiting its MovieLens site. In our case, we go to the section **Recommended for education and development**, and here, we find the full dataset which contains 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users. For the purpose of this project, the dataset we use is the **MovieLens 10M Dataset**, and to get it, we go to the section **older datasets** to choose the MovieLens 10M Dataset. This dataset consists of 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. We can get more information about it in this site https://files.grouplens.org/datasets/movielens/ml-10m-README.html.

1.2 The Goal of the Project

In a linear regression model (linear model) the author Singh (2019) mentions that *"one of the most important tasks is to select an appropriate evaluation metric"*. Definitely, this part of the process of building a linear model must be considered the most essential because there are various types of **evaluation metrics**, also known as **loss functions**, and we have to choose the most appropriate for each project.

The main functions used in linear models are: **MAE (Mean Absolute Error)**, **MSE (Mean Squared Error)** and **RMSE (Root Mean squared Error)**. Some authors also mention RMSLE (Root Mean squared Log Error), R-squared or Coefficient of determination and Adjusted R-squared.

MAE (Mean Absolute Error)
It symbolizes the average of the absolute difference between the predicted and actual (original) values in the dataset. That is to say, it measures the average of the residuals in the dataset and its formula is:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$$

where N is the number of observations, $\hat{y}_i$ is the predicted value and y_i is the actual value.

IMPORTANT: MAE is **NOT sensitive** to **outliers**.

MSE (Mean Squared Error)
It symbolizes the average of the squared difference between the predicted and actual (original) values in the dataset. Namely, it measures the variance of the residuals in the dataset and is given by the formula:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

where N is the number of observations, $\hat{y}_i$ is the predicted value and y_i is the actual value.

IMPORTANT: MSE is **sensitive** to **outliers**, so be careful when the dataset has them because they can distort results.

RMSE (Root Mean Squared Error)
It is the square root of Mean Squared error. Namely, it measures the standard deviation of the residuals in the dataset and is represented by the formula:

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

where N is the number of observations, $\hat{y}_i$ is the predicted value and y_i is the actual value.

IMPORTANT: RMSE is **sensitive** to **outliers**, so be careful when the dataset has them because they can distort results.

For this project, we are going to focus on the **RMSE loss function**, because it is the typical error we make when predicting a movie rating, beside that, it is the typical evaluation metric preferred in the recommendation systems. Therefore, for the specific case of our MovieLens project, the formula to calculate the RMSE value is as follows:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_{u,i} - y_{u,i})^2}$$

where N is the number of ratings (number of observations), $\hat{y}_{u,i}$ is the prediction of movie i by user u (predicted value) and $y_{u,i}$ is the rating of movie i by user u (actual value).

Another important point, *"large errors can increase our RMSE"* as Irizarry (2022) pointed out, so, RMSE penalizes large errors like MSE, and with the help of **regularization** allows us to penalize large estimates that are formed using small sample sizes.

Remember our goal is to get a **RMSE < 0.86490** and if this number is equal to zero means the model is perfect. Otherwise, if this number is larger than 1, it means our typical error is larger than one star, which is not good.

1.3 Key Steps Performed

A lot of data scientist follow a data science process or workflow, *"which is a structured framework used to complete a data science project"* this is how Gupta (2022) defines it and adds that *"there* *are many different frameworks, and some are better for business use cases, while others work best for research use cases"*.

The most popular and widely used data science process frameworks are **CRISP-DM (Cross Industry Standard Process for Data Mining)** and **OSEMN**, is made up of the capital letters of the following words: **O**btain data, **S**crub data, **E**xplore data, **M**odel data and i**N**terpret results, frameworks that is how Gupta (2022) and Nantasenamat (2022) point it out. In addition, **OSEMN** can be used on research, as Gupta (2022) states *"it's ideal for projects focusing on exploratory research, and is often used by research institutions and public health organizations"*. Another important point when selecting a framework is what was cited by Nantasenamat (2022) *"it should be noted that the flow among these processes is not linear and that in practice the flow can be non-linear and can re-iterate until satisfactory condition is met"* and that is fulfilled by **OSEMN**.

For the above, we utilize the **OSEMN framework**, it is composed of 5 steps which are described as follows:

1. **Obtain data**. In this first step we prepare the project, collect and get the datasets from different sources from surveys, databases, internet, files, etc. using different techniques as create new data, query databases, web scraping, downloading files, connecting to APIs, etc. We apply this step in the Section 2 Methods and Analysis, for more details go to Preparing the Data Science Project and the Datasets.

2. **Scrub data**. This step is considered, in data science, the most time-consuming depending on each project, that is why sometimes this step applies before Explore data or after. And it can be apply as many times as necessary throughout the project. It includes data cleaning, data pre-processing and handling missing values (this last process can be applied in the Explore data too, depending on each project). We employ this step in the Section 2 Methods and Analysis, for more details go to Preparing and Cleaning the Training and Testing Datasets.

3. **Explore data**. This procedure implements exploratory data analysis, where here, we must focus to understand and if it is possible, to find the relationship between the predict variable and possible predictor variables, additionally, detect missing values and outliers. Whereby, it involves the use of descriptive statistics and data visualizations. We apply this step in the Section 2 Methods and Analysis, for more details go to Performing Data Exploration and Visualization to `edx` dataset.

4. **Model data**. In this step, we build the model according to the results obtained or produced during the data exploration phase. We must also test and validate it. For more details about this procedure and how to we employ it, go to Section 3 Results.

5. **Interpret results**. This is the last phase and it is considered the most important, because we need to summarize all the results produced, such as the conclusion reached, what were its limitations and find out what the next course of action or future work would be, etc. of the created model in order to transmit them and that they can be understood or interpreted by people who have little or no knowledge of our subject. And to convey the results we must create a final report, a presentation or publish them in some type of media. We made this

final report by creating an Rmarkdown document that generates a PDF document, here is the product.

Chapter 2

Methods and Analysis

2.1 Preparing the Data Science Project and the Datasets

This is the first step that belongs to **Obtain data**, here, we prepare the project and get the datasets from a specific internet site. The next code chunk install packages required if they do not exist, then load them. And assign our **RMSE goal value** to `RMSE_GOAL` variable.

IMPORTAT NOTICE: We advise you to install all the packages required for this project manually and individually before running the code. Because during the installation process they can cause errors due to the lack of libraries either in the operating system, in `R` or in both.

```
###+++++  Install and Load the packages required +++++
#
# Install the libraries if they do not exist and load them
if(!require(this.path)) install.packages("this.path", repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")
if(!require(scales)) install.packages("scales", repos = "http://cran.us.r-project.org")
if(!require(Hmisc)) install.packages("Hmisc", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(huxtable)) install.packages("huxtable", repos = "http://cran.us.r-project.org")


library(this.path)
library(tidyverse)
library(data.table)
library(ggthemes)
library(scales)
library(Hmisc)
library(lubridate)
library(caret)
```

```r
library(huxtable)


### Assign our RMSE goal to RMSE_GOAL variable.
# Remember our goal is RMSE < 0.86490
RMSE_GOAL <- 0.86490
```

Now, we proceed to construct the structure of our project creating the subdirectories, we do this to facilitate and speed up collaborative work, share publications or reproducible research. It is well known that the structure of a project is:

- **our_project_dir**/
  - **rdas**/
  - **raw_data**/
  - **figs**/
- README.md
- final_report.Rmd
- download-data.R
- wrangle-data.R
- analysis.R

For this specific project, the structure of its directories is:

- **our_project_dir**/
  - **rdas**/
    - cp_movielens.rda
    - cp_movielens_train_test.rda
  - **ml-10M100K**/ (raw_data dir)
    - movies.dat
    - ratings.dat
- README.md
- report_movielens_proj.Rmd
- code_movielens_proj.R
- report_movielens_proj.pdf

NOTE: We can download the project from this repository on GitHub.

This code chunk get the current path, where this `.Rmd` file is running with the `this.dir` function from the `this.path` package, to set it like working directory. Check if the folder `rdas` exists in the actual directory, if not creates the `rdas` directory. Get the version of `R`, assign it to the variable `v` that will serve us to set the seed.

```r
### set the working directory and create a subdirectory
#
# Get the current path with the `this.dir` function
wd <- this.dir(default = getwd())
# Set the working directory
setwd(wd)
# check if the folder "rdas" exists in the current
# directory, if not creates a "rdas" directory
ifelse(!dir.exists("rdas"), dir.create("rdas"), "Folder rdas exists already")
```

```
## [1] "Folder rdas exists already"
```

```r
### +++++ Get the version of R +++++
v <- R.Version() # It is a List
```

The next task is to download the datasets from the 10M MovieLens Dataset's Download page, but before that it checks if the `ratings.dat` and `movies.dat` datasets exist in the `ml-10M100K` directory, if they exist it prints a message that they already exist and this task is finished. If they do not exist, then, a message is printed that they do not exist, they are downloaded, the version of `R` is detected to run the correct code to modify (`mutate`) the `movieId`, `title` and `genres` columns. Next, the seed is set depending on the version of `R` to create the `edx` and `validation` datasets. And finally, the `edx` and `validation` objects (datasets) are saved in the file `cp_movielens.rda` in the `rdas` directory, this last process has the objective of being able to help carry out and facilitate collaborative work, share publications or reproducible research.

NOTE: This process could take a couple of minutes. We do not be discouraged.

IMPORTAT NOTICE: If we have trouble with the following code, please skip it and go to the next code chunk "RUN this code chunk in CASE OF EMERGENCY".

```r
### Downloading the MovieLens 10M dataset,
### create the edx and validation datasets
#
# IMPORTANT: We can choose if we run this part of the code only once.
#
# NOTE: If we have trouble with the next code,
# please skip it and go to the section:
# ++++++++ Run in case of emergency ++++++++
#
# Check if the "ratings.dat" and "movies.dat" files
# exist in the "ml-10M100K" directory
if(file.exists(paste0(wd, "/ml-10M100K/ratings.dat"))==TRUE & file.exists(paste0(wd, "/ml-10M100K/movies.dat"))==TRUE){
  print("Files ratings.dat and movies.dat exist already")
}else{
  print("Files ratings.dat and movies.dat do NOT exist...downloading and creating the datasets")
  #
  #
  # NOTE: This process could take a couple of minutes.
  # We do not be discouraged.
  #
  #
  ### Downloading the MovieLens 10M dataset, assigning
  ### to `ratings` and `movies` variables
```

```r
#
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip
#
# Download the dataset
dl <- tempfile()
download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

# Assign to ratings
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

# Assign to movies
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")


### Make a left_join to `ratings` and `movies` assigning to
### `movielens` variable and set the seed according to the R version
#
# Detect the version of R to run the correct code
if (paste(v$major, v$minor, sep = ".") < "4.0.0"){
  print("version of R is MINOR to 4.0.0, mutate movieId with levels")
  # 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))
}else{
  print("version of R is MAJOR or equal to 4.0.0, mutate movieId without levels")
  # 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))
}

### make a left_join to `ratings` and `movies` assigning to `movielens` variable
movielens <- left_join(ratings, movies, by = "movieId")

# Set the seed according to the R version
if (paste(v$major, v$minor, sep = ".") < "3.6.0"){
  print("version of R is MINOR to 3.6.0, use set.seed(1)")
  # if using R 3.5 or earlier, use `set.seed(1)`:
  set.seed(1)
}else{
  print("version of R is MAJOR or equal to 3.6.0, use set.seed(1, sample.kind=Rounding)")
  # if using R 3.6 or later:
  set.seed(1, sample.kind="Rounding")
}

### Create `edx` set, `validation` set (final hold-out test set)
### and save them to `/rdas/cp_movielens.rda`
#
# Validation set will be 10% of MovieLens data
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

###++++++++++ Create `validation` set ++++++++++
# 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
```

```r
  removed <- anti_join(temp, validation)


  ###+++++++++ Create `edx` set +++++++++++
  edx <- rbind(edx, removed)

  # Remove these objects from the Global Environment
  rm(dl, ratings, movies, test_index, temp, movielens, removed)

  ### save objects "edx", "validation" in the file
  ### path: rdas/cp_movielens.rda
  # NOTE: Take aprox. 2 to 3 minutes
  save(edx, validation, file = "./rdas/cp_movielens.rda")
}
```

```
## [1] "Files ratings.dat and movies.dat do NOT exist...downloading and creating the datasets"
## [1] "version of R is MAJOR or equal to 4.0.0, mutate movieId without levels"
## [1] "version of R is MAJOR or equal to 3.6.0, use set.seed(1, sample.kind=Rounding)"
```

RUN this code chunk in CASE OF EMERGENCY

```r
###+++++++++++ Run in case of emergency ++++++++++++
#
# If we had any trouble with above code, at the moment
# to downloading or others problems, because I saw in
# the discussion forum some people wrote about it.
# I provided a .rda file; with the objects "edx" and
# "validation", from my gitlab.
#
# Download the file from gitlab to "rdas" directory
# Run the next line, if we had a trouble with
# the code above. This process is more faster!!
download.file("https://gitlab.com/saulcol/rdas/-/raw/main/cp_movielens.rda",
              paste0(wd, "/rdas/cp_movielens.rda"))
```

The following code verify if the `cp_movielens.rda` file exists in the `rdas` directory, if it exists it prints a message that it already exists and this process is finished. If it does not exist, then, a message is printed that it does not exist, it is downloaded from gitlab to `rdas` directory. And remove the `edx` and `validation` objects from the Global Environment if they exist.

```r
# Verify if the "cp_movielens.rda" file exists in the "rdas" directory
if(file.exists(paste0(wd, "/rdas/cp_movielens.rda"))==TRUE){
  print("File cp_movielens.rda exists already")
}else{
  print("File cp_movielens.rda does NOT exist...downloading")
  # Download the file from gitlab to "rdas" directory
  # NOTE: Take aprox. 1 to 2 minutes
  download.file("https://gitlab.com/saulcol/rdas/-/raw/main/cp_movielens.rda",
                paste0(wd, "/rdas/cp_movielens.rda"))
```

```r
  # Remove these objects from the Global Environment if they exist
  if(exists("edx")) rm("edx", envir = globalenv())
  if(exists("validation")) rm("validation", envir = globalenv())
}
```

```
## [1] "File cp_movielens.rda exists already"
```

2.2 Performing Data Exploration and Visualization to `edx` dataset

It is part of **Explore data**, where, we use descriptive statistics and data visualizations to understand and if it is possible, to find the relationship between the predict variable and possible predictor variables, additionally, detect missing values and outliers.

2.2.1 Overall Exploration in the Dataset

Let us start by performing a general exploration on the `edx` dataset by running the following functions to see its structure and how it is composed. But before, we need to load the `edx` and `validation` objects from the file path: `rdas/cp_movielens.rda`.

```r
### ++++++++++ Overall Exploration in the Dataset ++++++++++++++
#
### load objects "edx", "validation" from the file path: rdas/cp_movielens.rda.
# Take a few seconds
load("./rdas/cp_movielens.rda")


# Knowing its structure and how it is composed the "edx" dataset
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

```r
#userId    <int>
#movieId   <dbl>
#rating    <dbl>
#timestamp <int>
#title     <chr>
#genres    <chr>

str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 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 83898
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ..
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

```r
#userId   : int
#movieId  : num
#rating   : num
#timestamp: int
#title    : chr
#genres   : chr


# How many rows and columns are there in the "edx" dataset?
# Number of rows:
dim(edx)[1]
```

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

```r
# Number of columns:
dim(edx)[2]
```

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

By executing the above code, we know that the `edx` dataset is composed by **9,000,055 observations** (`rows`) and **6 variables** (`columns`). As well, its six `columns` have these characteristics:

```
userId   : integer
movieId  : numeric (double)
rating   : numeric (double)
timestamp: integer
title    : character
genres   : character
```

We continue now observing its statistical summary with the `summary()` function.

```r
# Get a summary statistics
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         
##  Length:9000055     Length:9000055    
##  Class :character   Class :character  
##  Mode  :character   Mode  :character  
##                                       
##                                       
##                                       
```

Other way, to obtain a better summary statistics is with the function `describe()` of the package `Hmisc`. **NOTE:** Take aprox. 2 to 3 minutes.

```r
# Other way, to obtain a better summary statistics
describe(edx)
```

```
## edx 
## 
##  6  Variables      9000055  Observations
## --------------------------------------------------------------------------------
## userId 
##        n  missing distinct     Info     Mean      Gmd      .05      .10 
##  9000055        0    69878        1    35870    23769     3810     7521 
##      .25      .50      .75      .90      .95 
##    18124    35738    53607    64479    68093 
## 
## lowest :     1     2     3     4     5, highest: 71563 71564 71565 71566 71567
## --------------------------------------------------------------------------------
## movieId 
##        n  missing distinct     Info     Mean      Gmd      .05      .10 
##  9000055        0    10677        1     4122     5535      110      260 
##      .25      .50      .75      .90      .95 
##      648     1834     3626     6502     8917 
## 
## lowest :     1     2     3     4     5, highest: 65088 65091 65126 65130 65133
## --------------------------------------------------------------------------------
```

```
## rating
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##  9000055        0       10    0.958    3.512    1.166      1.5      2.0
##      .25      .50      .75      .90      .95
##      3.0      4.0      4.0      5.0      5.0
##
## lowest : 0.5 1.0 1.5 2.0 2.5, highest: 3.0 3.5 4.0 4.5 5.0
##
## Value          0.5     1.0     1.5     2.0     2.5     3.0     3.5     4.0
## Frequency    85374  345679  106426  711422  333010 2121240  791624 2588430
## Proportion   0.009   0.038   0.012   0.079   0.037   0.236   0.088   0.288
##
## Value          4.5     5.0
## Frequency   526736 1390114
## Proportion   0.059   0.154
## ---------------------------------------------------------------------------
## timestamp
##         n   missing  distinct      Info      Mean       Gmd       .05       .10
##   9000055         0   6519590         1 1.033e+09 133321774 8.395e+08 8.506e+08
##       .25       .50       .75       .90       .95
## 9.468e+08 1.035e+09 1.127e+09 1.188e+09 1.213e+09
##
## lowest :  789652009  822873600  823185196  823185197  823185198
## highest: 1231131132 1231131137 1231131142 1231131303 1231131736
## ---------------------------------------------------------------------------
## title
##        n  missing distinct
##  9000055        0    10676
##
## lowest : 'burbs, The (1989)                    'night Mother (1986)                 'Round
## highest: Zoot Suit (1981)                      Zorba the Greek (Alexis Zorbas) (1964) Zorro,
## ---------------------------------------------------------------------------
## genres
##        n  missing distinct
##  9000055        0      797
##
## lowest : (no genres listed)                                      Action
## highest: Thriller|War                                            Thriller|Western
## ---------------------------------------------------------------------------
```

We see the content of the first six rows of the `edx` dataset using this code with the `as_hux()` function from the `huxtable` package to get a more stylish table for this report instead of use just `head(edx)`.

```
edx %>%
  head() %>%
  as_hux() %>%
```

```
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .1, .1, .2, .5, .6)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

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

In general, it can be observed the mentioned dataset **does not have missing values**. The `userId` column has a range from **1** to **71567** and has **69878** different users. The `movieId` column has a range from **1** to **65133** and has **10677** different users. The `rating` column has **10** different ratings that are in a range from **0.5** to **5.0** with increments by **0.5**. The `timestamp` variable has a range from **789652009** to **1231131736** (we need to treat it later). The `title` column has **10676** different titles. And the last one column `genres` has **797** different genres and is labeled with one or more genre. Recall that the `rating` column is **the outcome**.

2.2.2 Are there missing values?

Earlier when we used the `describe` function we saw that the six columns do not have missing values. We can corroborate it with these codes.

```
###+++++++++++++ Are there missing values? ++++++++++++++++++
#
# Count missing values
sum(is.na(edx))
```

```
## [1] 0
```

```
any(is.na(edx))
```

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

```r
# Count total missing values in each column of data frame
colSums(is.na(edx))
```

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

In effect, the `edx` dataset **does not have missing values**. Now, we proceed to perform a data analysis on each of the variables of this dataset.

2.2.3 Data Analysis in the UserId column

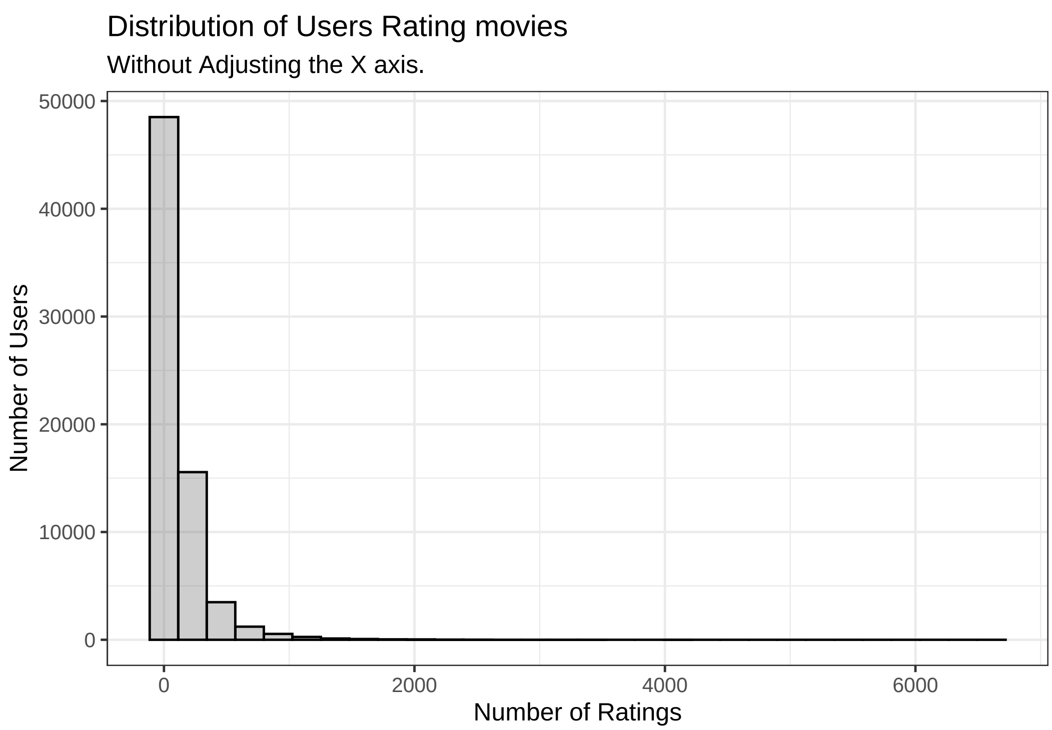
How many different users are in the `edx` dataset?

```r
###++++++++++ Data Analysis in the UserId column +++++++++++++
#
length(unique(edx$userId))
```

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

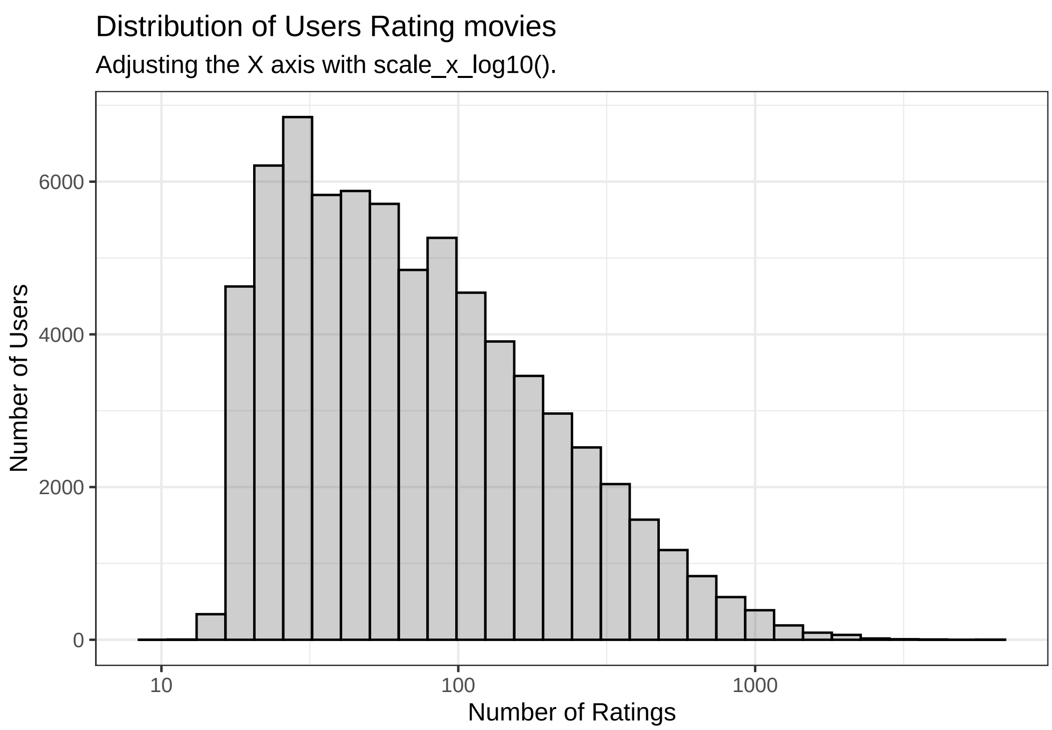
The following code chunk produce a histogram of the Distribution of users rating movies.

```r
# Plot a histogram of the Distribution of users rating movies (incorrect)
# Without adjusting the X axis
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "black", alpha = 0.3) +
  ggtitle("Distribution of Users Rating movies",
          subtitle = "Without Adjusting the X axis.") +
  xlab("Number of Ratings") +
  ylab("Number of Users") +
  theme_bw()
```



As we could see, the plot was not displayed correctly, because the X-axis was not adjusted. For that reason, we plot the same histogram again, but adjusting the X-axis with `scale_x_log10()`.

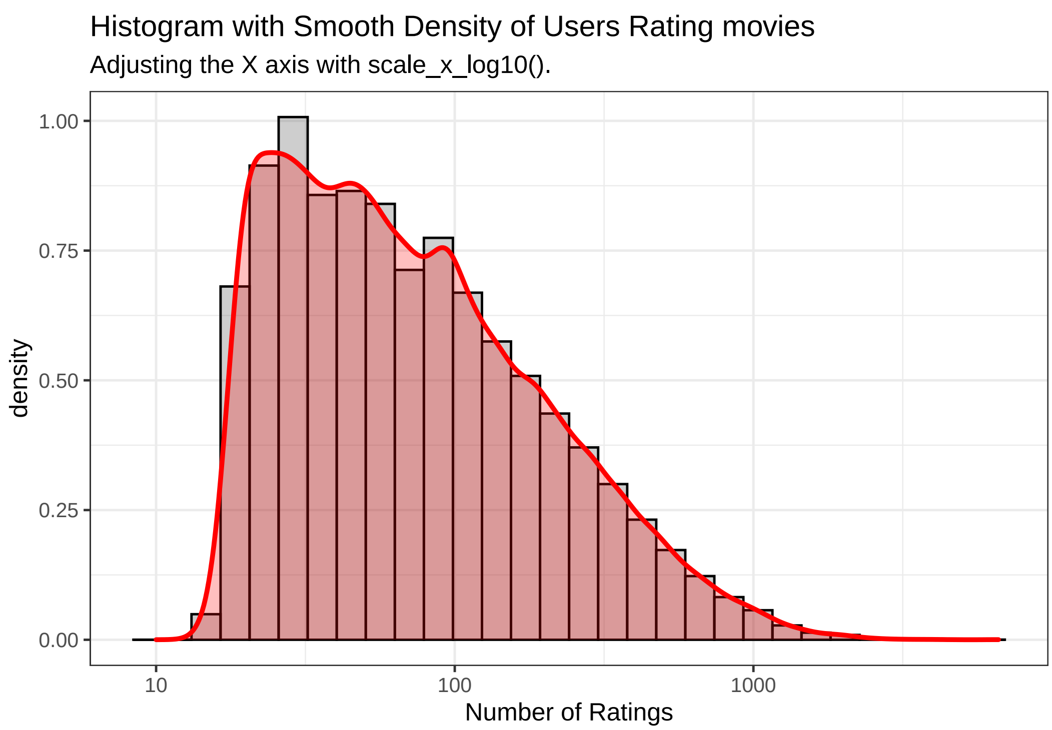
```
# Plot a histogram of the Distribution of users rating movies (correct)
# Adjusting the X axis with scale_x_log10()
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "black", alpha = 0.3) +
  scale_x_log10() +
  ggtitle("Distribution of Users Rating movies",
          subtitle = "Adjusting the X axis with scale_x_log10().") +
  xlab("Number of Ratings") +
  ylab("Number of Users") +
  theme_bw()
```



With that change made, we can appreciate that the major part of users rate less movies, while a few users rate more than a thousand movies. Some users are more active than others at rating movies, and few users rate very few movies. We notice that the distribution is **right skewed** or **positively skewed**.

Here is the histogram with a smooth density curve of the users.

```
# Plot a histogram with Smooth density of users rating movies (correct)
# Adjusting the X axis with scale_x_log10()
edx %>% count(userId) %>%
  ggplot(aes(x=n, y=..density..)) +
  geom_histogram(color = "black", alpha = 0.3) +
  geom_density(lwd = 1, color="red", fill="red", alpha = 0.25) +
  scale_x_log10() +
  ggtitle("Histogram with Smooth Density of Users Rating movies",
          subtitle = "Adjusting the X axis with scale_x_log10().") +
  xlab("Number of Ratings") +
  theme_bw()
```



2.2.4 Data Analysis in the MovieId column

How many different movies are in the `edx` dataset?

```
###+++++++++++++++++++ Data Analysis in the MovieId column +++++++++++++++++++++++
#
n_distinct(edx$movieId)
```

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

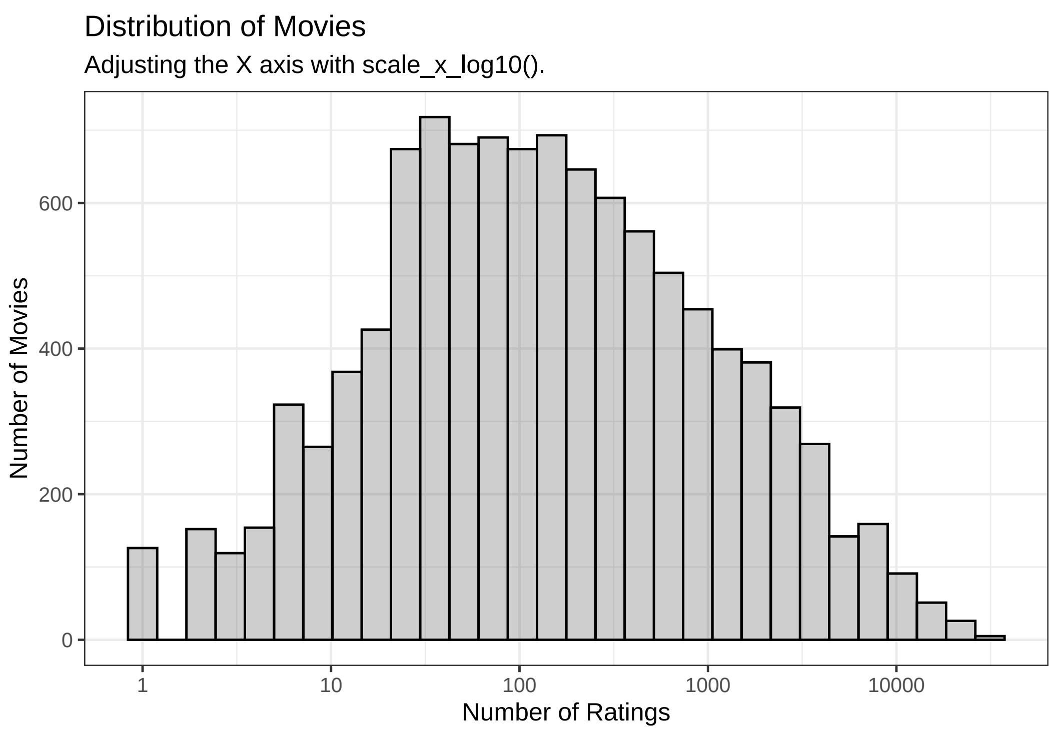
Which movie has the greatest number of ratings?

```
edx %>%
  group_by(movieId, title) %>%
  summarise(total = n()) %>%
  arrange(desc(total)) %>%
  head(20) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, 1.1, .1)) %>%
  set_number_format(everywhere, 1, 0) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

movieId	title	total
296	Pulp Fiction (1994)	31362
356	Forrest Gump (1994)	31079
593	Silence of the Lambs, The (1991)	30382
480	Jurassic Park (1993)	29360
318	Shawshank Redemption, The (1994)	28015
110	Braveheart (1995)	26212
457	Fugitive, The (1993)	25998
589	Terminator 2: Judgment Day (1991)	25984
260	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672
150	Apollo 13 (1995)	24284
592	Batman (1989)	24277
1	Toy Story (1995)	23790
780	Independence Day (a.k.a. ID4) (1996)	23449
590	Dances with Wolves (1990)	23367
527	Schindler's List (1993)	23193
380	True Lies (1994)	22823
1210	Star Wars: Episode VI - Return of the Jedi (1983)	22584
32	12 Monkeys (Twelve Monkeys) (1995)	21891
50	Usual Suspects, The (1995)	21648
608	Fargo (1996)	21395

The plot of the distribution of movies adjusting the X-axis with `scale_x_log10()`.

```r
### Plot a histogram of the Distribution of movies (correct)
# Adjusting the X axis with scale_x_log10()
edx %>% count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(color = "black", alpha = 0.3) +
  scale_x_log10() +
  ggtitle("Distribution of Movies",
          subtitle = "Adjusting the X axis with scale_x_log10().") +
  xlab("Number of Ratings") +
  ylab("Number of Movies") +
  theme_bw()
```



We can observe that many movies were rated by very few users, and some movies get rated more than others. The distribution is **closely normal**.

2.2.5 Data Analysis in the Ranting column

As we know, the `rating` column is **the outcome** and it has **10** different ratings that are in a range from **0.5** to **5.0** with **increments** by **0.5**.

How many zeros were given as ratings in the `edx` dataset?

```
###+++++++++ Data Analysis in the Ranting column +++++++++
#
edx %>%
  filter(rating == 0) %>%
  count()
```

n
0

How many threes were given as ratings in the `edx` dataset?

```
edx %>%
  filter(rating == 3) %>%
  count()
```

n
2121240

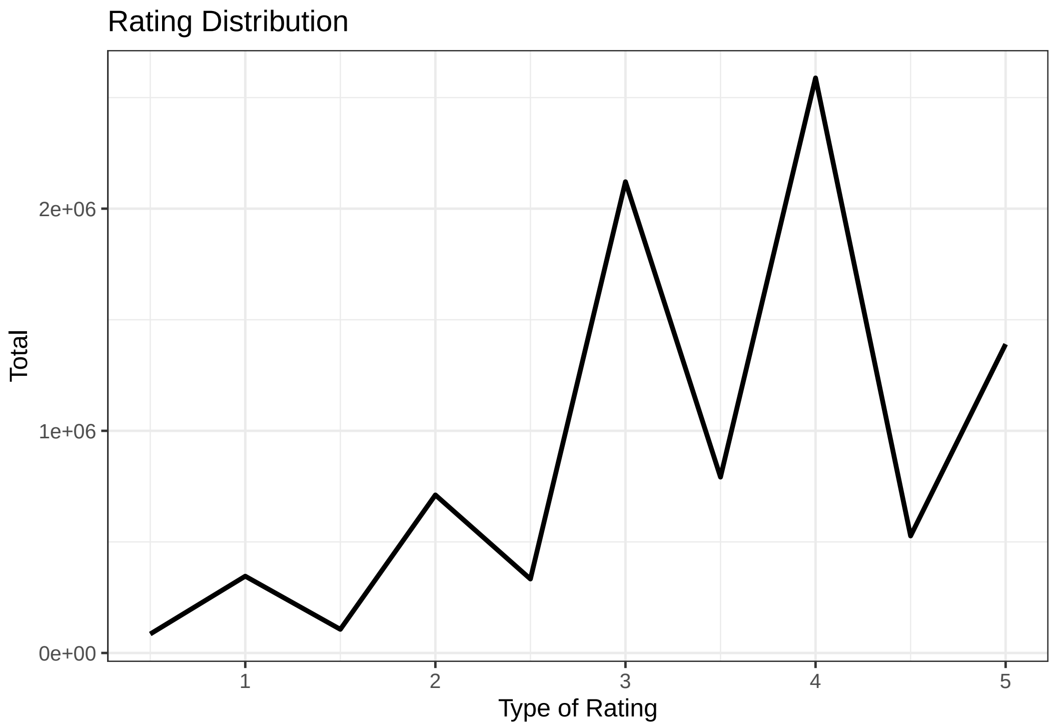
What are the five most given ratings in order from most to least?

```
edx %>%
  group_by(rating) %>%
  summarise(total = n()) %>%
  arrange(desc(total)) %>%
  head(5) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .4)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

rating	total
4	2588430
3	2121240
5	1390114
3.5	791624
2	711422

In general, half star ratings are less common than whole star ratings (e.g., there are fewer ratings of 3.5 than there are ratings of 3 or 4, etc.). Graphically they can be obtained with this code:

```
edx %>%
  group_by(rating) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  geom_line(lwd = 1) +
  ggtitle("Rating Distribution") +
  xlab("Type of Rating") +
  ylab("Total") +
  theme_bw()
```



2.2.6 Data Analysis in the Timestamp column

If we remember the `timestamp` variable has a range from **789652009** to **1231131736**. And its values *"represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970"*, as Harper and Konstan (2016) affirm. Let us to use the `as.Date` and `as.POSIXct` functions of the package `base` with the minimum value to know what is **the starting date**. We could also use the `as_datetime` function from the `lubridate` package.

```
###++++++++ Data Analysis in the Timestamp column +++++++
#
as.Date(as.POSIXct(min(edx$timestamp), origin="1970-01-01", tz = "GMT"))
```

```
## [1] "1995-01-09"
```

Now the same process but with the maximum value to know what is **the ending date**.

```
as.Date(as.POSIXct(max(edx$timestamp), origin="1970-01-01", tz = "GMT"))
```

```
## [1] "2009-01-05"
```

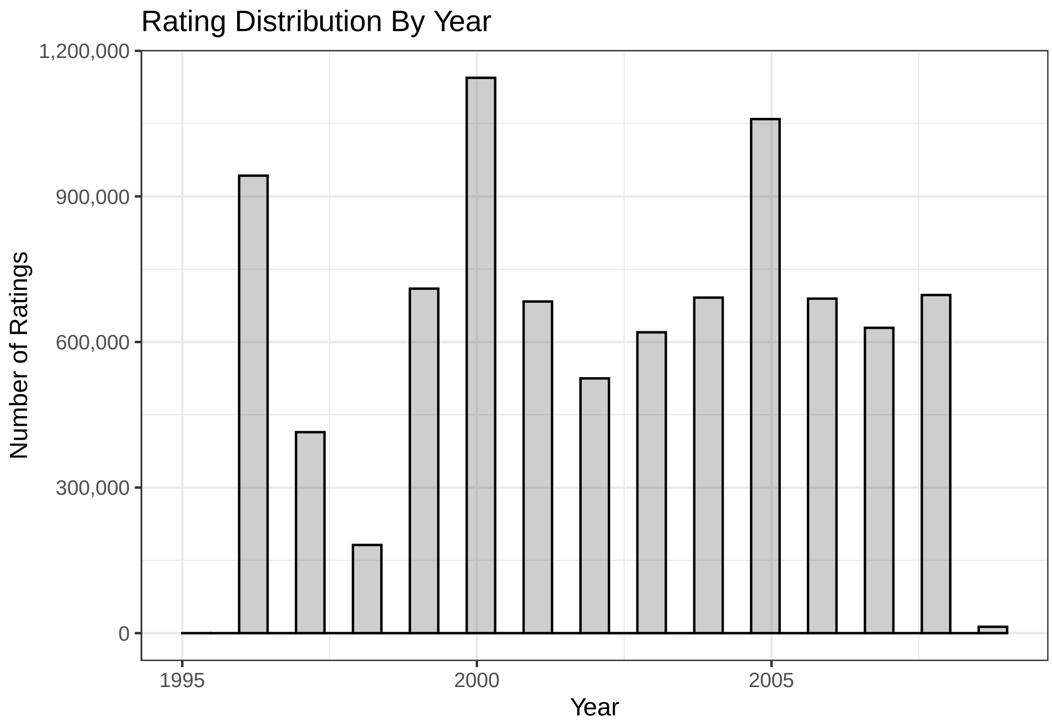
Next, we can verify which is the range period of the time of ratings using the `duration` function from the `lubridate` package.

```r
tibble(`Starting Date` = as.Date(as.POSIXct(min(edx$timestamp),
                                            origin="1970-01-01", tz = "GMT")),
       `Ending Date` = as.Date(as.POSIXct(max(edx$timestamp),
                                          origin="1970-01-01", tz = "GMT"))) %>%
  mutate(`Range Period` = as.character(duration(max(edx$timestamp)-min(edx$timestamp)))) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.2, .2, .8)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Starting Date	Ending Date	Range Period
1995-01-09	2009-01-05	441479727s (~13.99 years)

The histogram of rating distribution by years is:

```r
### Plot the histogram of rating distribution by years
edx %>% mutate(year = year(as.Date(as.POSIXct(timestamp,
                                              origin="1970-01-01", tz = "GMT")))) %>%
  ggplot(aes(x=year)) +
  geom_histogram(color = "black", alpha = 0.3) +
  ggtitle("Rating Distribution By Year") +
  xlab("Year") +
  ylab("Number of Ratings") +
  scale_y_continuous(labels = comma) +
  theme_bw()
```



We can see the dates with more ratings.

```
edx %>% mutate(date = as.Date(as.POSIXct(timestamp,
                                          origin="1970-01-01", tz = "GMT"))) %>%
  group_by(date, title) %>%
  summarise(total = n()) %>%
  arrange(desc(total)) %>%
  head(10) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, 1.1, .1)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

date	title	total
1998-05-22	Chasing Amy (1997)	322
2000-11-20	American Beauty (1999)	277
1999-12-11	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	254
1999-12-11	Star Wars: Episode V - The Empire Strikes Back (1980)	251
1999-12-11	Star Wars: Episode VI - Return of the Jedi (1983)	241
2005-03-22	Lord of the Rings: The Two Towers, The (2002)	239
2005-03-22	Lord of the Rings: The Fellowship of the Ring, The (2001)	227
2000-11-20	Terminator 2: Judgment Day (1991)	221
1999-12-11	Matrix, The (1999)	210
2000-11-20	Jurassic Park (1993)	201

2.2.7 Data Analysis in the Genres column

How many different genres are in the `edx` dataset?

```
###++++++++ Data Analysis in the Genres column +++++++
#
length(unique(edx$genres))
```

```
## [1] 797
```

We display the ten most viewed genres.

```
edx %>% group_by(genres) %>%
  summarise(total=n()) %>%
  arrange(desc(total)) %>%
  head(10) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

genres	total
Drama	733296
Comedy	700889
Comedy|Romance	365468
Comedy|Drama	323637
Comedy|Drama|Romance	261425
Drama|Romance	259355
Action|Adventure|Sci-Fi	219938
Action|Adventure|Thriller	149091
Drama|Thriller	145373
Crime|Drama	137387

Various movies are organized or cataloged by more than one genre. To confirm that, we can view the ten genres that have most number of different genres for each movie with this code.

```
tibble(num_genres = str_count(edx$genres, fixed("|")),
       genres = edx$genres) %>%
  group_by(num_genres, genres) %>%
  summarise(n = n()) %>%
  arrange(desc(num_genres)) %>%
  head(10) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, 1.2, .1)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

num_genres	genres	n
7	Action|Adventure|Comedy|Drama|Fantasy|Horror|Sci-Fi|Thriller	256
6	Adventure|Animation|Children|Comedy|Crime|Fantasy|Mystery	10975
6	Adventure|Animation|Children|Comedy|Drama|Fantasy|Mystery	355
6	Adventure|Animation|Children|Comedy|Fantasy|Musical|Romance	515
5	Action|Adventure|Animation|Children|Comedy|Fantasy	187
5	Action|Adventure|Animation|Children|Comedy|IMAX	66
5	Action|Adventure|Animation|Children|Comedy|Sci-Fi	600
5	Action|Adventure|Animation|Drama|Fantasy|Sci-Fi	239
5	Action|Adventure|Children|Comedy|Fantasy|Sci-Fi	2832
5	Action|Adventure|Children|Crime|Mystery|Thriller	62

By the last, we see the top ten most genres by movie.

```
edx %>% group_by(genres, movieId) %>%
  summarise(total=n()) %>%
  arrange(desc(total)) %>%
  head(10) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1, .1)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

genres	movieId	total
Comedy|Crime|Drama	296	31362
Comedy|Drama|Romance|War	356	31079
Crime|Horror|Thriller	593	30382
Action|Adventure|Sci-Fi|Thriller	480	29360
Drama	318	28015
Action|Drama|War	110	26212
Thriller	457	25998
Action|Sci-Fi	589	25984
Action|Adventure|Sci-Fi	260	25672
Adventure|Drama	150	24284

2.3 Preparing and Cleaning the Training and Testing Datasets

This phase is **Scrub data**, but, before we start building our model, we need to divide the `edx` dataset in two new datasets: One for **training** and the other for **testing** as instructed by **Harvardx's team**. Here are the instructions again:

IMPORTANT: The `validation` data (the final hold-out test set) should **NOT** be used for training, developing, or selecting your algorithm and **it should ONLY be used for evaluating the RMSE of your final algorithm**. The final hold-out test set should only be used at the end of your project with your final model. **It may not be used to test the RMSE of multiple models during model development**. You should split the `edx` data into separate training and test sets to design and test your algorithm.

IMPORTANT: Please be sure not to use the `validation` set (the final hold-out test set) for training or regularization - you should create an additional partition of training and test sets from the provided `edx` dataset to experiment with multiple parameters or use cross-validation.

Remember our goal is to get a **RMSE < 0.86490**.

As we mentioned, first we must split the `edx` dataset in two new datasets: `train` and `test` datasets, we use the same process employed to create `edx` and `validation` datasets. We do that in the following way: Set the seed depending on the version of `R` to create the `train` and `test` datasets, where, `test` set will be 10% and `train` set 90% of `edx` dataset.

```
###+++++++++ Create the train set and test set from the edx set ++++++++++
#
#
# Set the seed according to the R version
if (paste(v$major, v$minor, sep = ".") < "3.6.0"){
  print("version of R is MINOR to 3.6.0, use set.seed(1)")
  # if using R 3.5 or earlier, use `set.seed(1)`:
  set.seed(1)
}else{
  print("version of R is MAJOR or equal to 3.6.0, use set.seed(1, sample.kind=Rounding)")
  # if using R 3.6 or later:
  set.seed(1, sample.kind="Rounding")
}
```

```
## [1] "version of R is MAJOR or equal to 3.6.0, use set.seed(1, sample.kind=Rounding)"
```

```
###++++++++ Create `train` set, `test` set from the `edx` set +++++++++
###+++++++++++ and save them to `/rdas/cp_movielens.rda` ++++++++++++++
#
# We must split the `edx` set in 2 parts: the training and test sets.
# We use the same process employed to create `edx` and 'validation' sets.
#
```

```r
# `test` set will be 10% and `train` set 90% of `edx` data
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train <- edx[-test_index,]
temp <- edx[test_index,]

###++++++++++++++++++++++ Create `test` set ++++++++++++++++++
# Make sure userId and movieId in `test` set are also in `train` set
test <- temp %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")

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

###+++++++++++++++++++++ Create `train` set +++++++++++++++++++++++++
train <- rbind(train, removed)


# Remove these objects from the Global Environment
rm(test_index, temp, removed)
```

As prior to this talked about, various or many characteristics can be used to predict the rating for a given user. But nevertheless, a large number of predictors increment the model complexity and need more computer resources. For this project, the estimated rating applies to movie and user information only. So, we proceed to clean the `train` and `test` datasets keeping only the `userId`, `movieId`, `rating`, `title` columns.

And finally, the `train` and `test` objects (datasets) are saved in the file `cp_movielens_train_test.rda` in the `rdas` directory, this last process has the objective of being able to help carry out and facilitate collaborative work, share publications or reproducible research.

```r
###++++++++ Cleaning the `train` and `test` set ++++++++++
#
train <- train %>% select(userId, movieId, rating, title)
test  <- test  %>% select(userId, movieId, rating, title)


###+++++++ save objects "train", "test" in the file +++++++
###+++++++ path: rdas/cp_movielens_train_test.rda +++++++++
# NOTE: Take aprox. 2 to 3 minutes
save(train, test, file = "./rdas/cp_movielens_train_test.rda")
```

If we have some trouble, run the next code chunk to download `cp_movielens_train_test.rda` file and save it in the file path: `rdas/cp_movielens_train_test.rda`.

```
###+++ If we have some trouble download "cp_movielens_train_test.rda" file and ++++
###++++++++++ save it in the file path: rdas/cp_movielens_train_test.rda +++++++++++
#
# Download the file from gitlab to "rdas" directory
# NOTE: Take aprox. 1 to 2 minutes
download.file("https://gitlab.com/saulcol/rdas/-/raw/main/cp_movielens_train_test.rda",
              paste0(wd, "/rdas/cp_movielens_train_test.rda"))
```

2.4 Modeling approach

2.4.1 Linear Model

We start by creating the simplest model (recommendation system) that predicts the same rating, for all movies, given by all users. This model assumes the same rating for all movies and users with all the differences explained by independent errors sampled from the same distribution (random errors distribution or random variation). Also, we know that the average of all ratings is the least squares estimate of mu and that is the estimate that minimizes the RMSE. Therefore, our initial model is the average of all ratings and its formula is:

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

where mu is the "true" rating for all movies and $\varepsilon_{u,i}$ are the independent errors sampled from the same distribution centered at 0 (random variation). We must consider that if we place or put any other number, we obtain a higher RMSE. This formula and the formulas of the movie and user effect models are those proposed by Irizarry (2022).

Our next model is the movie effect (movie variability), it is named **movie bias** and is represented by b_i, which this effect is due to some movies get rated more than others or distinct movies being rated differently. Since *"[...] there are blockbuster movies watched by millions and artsy, independent movies watched by just a few"*, that is how Irizarry (2022) expresses it. The formula to compute the movie bias is:

$$\hat{b_i} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\mu})$$

The formula that we are going to use for this model is:

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

With the user effect model, it is called **user bias** and is expressed by b_u, its effect is due to that some users are more active than others at rating movies or distinct users have distinct rating movies. And the formula is:

$$\hat{b_u} = \frac{1}{N}\sum_{i=1}^{N}(y_{u,i} - \hat{b}_i - \hat{\mu})$$

The formula of the model for adding the user effect is:

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

In this project we will test those two effects.

2.4.2 Regularization

The linear model doesn't contemplate that many movies were rated by very few users, and few users rate very some movies. Irizarry (2022) affirms *"these are noisy estimates that we should not trust, especially when it comes to prediction"* and adds that *"large errors can increase our RMSE, so we would rather be conservative when unsure"*. For those reasons, with the help of **regularization** allows us to penalize large estimates that are formed using small sample sizes.

For that, we need to regularize the movie and user effects aggregating a **penalty term** or factor which is known as **lambda** (λ) and which is also a **tuning parameter**. To compute regularized estimates of movie effects ($\hat{b_i}$) using λ, the formula is:

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

And to calculate regularized estimates of user effects ($\hat{b_u}$) using λ, the formula is:

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

Finally, we have to pick the best value of λ that **minimizes the RMSE**. We can do it by setting a set of values for lambda, and use **cross-validation**.

Chapter 3

Results

It is time to describe the procedure **Model data**, here, we build the model according to the results obtained or produced during the data exploration phase. We must also test and validate it.

3.1 Defining the RMSE Function to Evaluate the Model

First, we define the loss function, in this case, the **Root Mean Squared Error (RMSE)** function.

```
# Define the Root Mean Squared Error (RMSE) function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

3.2 Linear Model

It is important to remember that before creating the model we need to know that: The model must be **built on the training dataset**, and the test dataset must be **employed to test the model**. After that, when the model is finished and ready, we will use the `validation` dataset to compute **the final RMSE**.

To carry out this model we are going to follow the procedures stipulated and learned in this lesson.

We are going to create our linear model established by this formula:

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

Before to continue, we Verify if the `cp_movielens_train_test.rda` file exists in the "rdas" directory. If the file does not exists, it is downloaded. Immediately it is loaded to be able to have the `train` and `test` objects.

We can load the objects `train`, `test` to simplify the construction of our model if we want, just by running this code. In case it is necessary.

```
###++++++++++ If we have some trouble download "cp_movielens_train_test.rda" file and ++++++++++
###+++++++++++++++++ save it in the file path: rdas/cp_movielens_train_test.rda ++++++++++++++++
#
# Verify if the `cp_movielens_train_test.rda` file exists in the "rdas" directory
if(file.exists(paste0(wd, "/rdas/cp_movielens_train_test.rda"))==TRUE){
  print("File cp_movielens_train_test.rda exists already")
}else{
  print("File cp_movielens_train_test.rda does NOT exist...downloading")
  # Download the file from gitlab to "rdas" directory
  # NOTE: Take aprox. 1 to 2 minutes
  download.file("https://gitlab.com/saulcol/rdas/-/raw/main/cp_movielens_train_test.rda",
                paste0(wd, "/rdas/cp_movielens_train_test.rda"))

  # Remove these objects from the Global Environment if they exist
  if(exists("train")) rm("train", envir = globalenv())
  if(exists("test")) rm("test", envir = globalenv())
}
```

```
## [1] "File cp_movielens_train_test.rda exists already"
```

```
# load objects `train`, `test` from the file path:
# rdas/cp_movielens_train_test.rda. Take a few seconds
load("./rdas/cp_movielens_train_test.rda")
```

3.2.1 Predict the Mean of the Rantings

We start by predicting the mean of the ratings (mu) and its formula is:

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

```
# Mean of observed values
mu <- mean(train$rating)

# Create a table to store the error scores
scores <- tibble(Method = "RMSE Project Goal", RMSE = RMSE_GOAL)

# Update the scores table
scores <- bind_rows(scores,
                    tibble(Method = "Mean",
                           RMSE = RMSE(test$rating, mu)))
```

```r
# Display the RMSE improvement
scores %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Method	RMSE
RMSE Project Goal	0.8649000
Mean	1.0600537

3.2.2 Adding the Movie Effect (bi)

bi is the movie effect (bias) for movie i. The formula is:

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

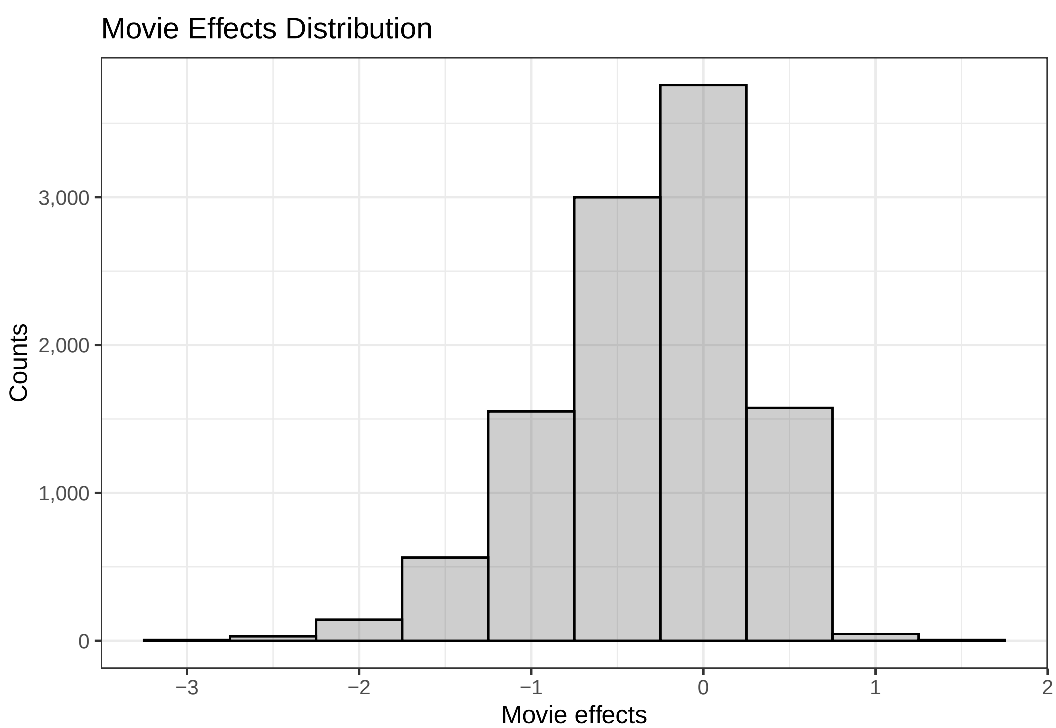
We create the movie effects (`bi`) subset and view its first six rows.

```r
# Create the Movie effects (bi) subset
bi <- train %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))

# Show the first six rows of this subset
bi %>%
  head() %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.04, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

movieId	b_i
1	0.4150040
2	-0.3064057
3	-0.3613952
4	-0.6372808
5	-0.4416058
6	0.3018943

We plot the distribution of movie effects (bi).

```
# Plot the distribution of movie effects
bi %>% ggplot(aes(x = b_i)) +
  geom_histogram(bins=10, col = I("black"), alpha = 0.3) +
  ggtitle("Movie Effects Distribution") +
  xlab("Movie effects") +
  ylab("Counts") +
  scale_y_continuous(labels = comma) +
  theme_bw()
```



The distribution is a **little left skewed** as we could see. Now, we can go ahead to predict it and get its scores.

```r
# Predict the rating with mean + bi
y_hat_bi <- mu + test %>%
  left_join(bi, by = "movieId") %>%
  .$b_i

# Compute the RMSE and update the scores table
scores <- bind_rows(scores,
                    tibble(Method = "Mean + bi",
                           RMSE = RMSE(test$rating, y_hat_bi)))

# Display the RMSE improvement
scores %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Method	RMSE
RMSE Project Goal	0.8649000
Mean	1.0600537
Mean + bi	0.9429615

3.2.3 Adding the User Effect (bu)

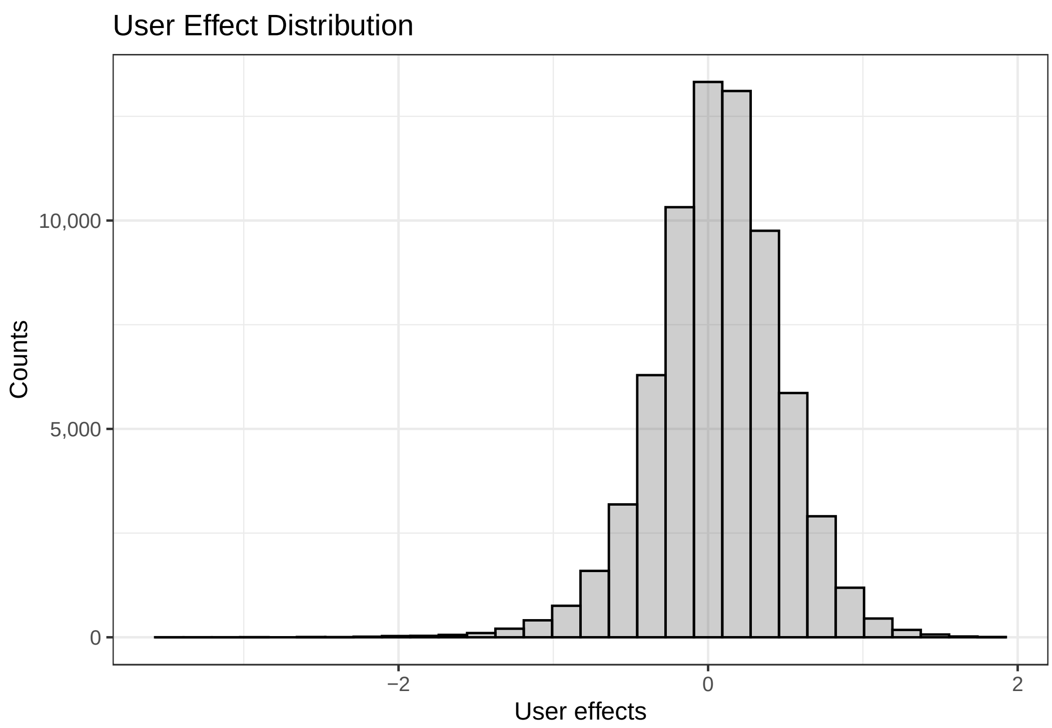
bu is the user effect (bias) for user u. The formula is:

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

We create the user effects (`bu`) subset and plot its distribution.

```r
# Create the User effects (bu) subset
bu <- train %>%
  left_join(bi, by = 'movieId') %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))

# Plot the distribution of user effects
```

```r
bu %>% ggplot(aes(x = b_u)) +
  geom_histogram(col = I("black"), alpha = 0.3) +
  ggtitle("User Effect Distribution") +
  xlab("User effects") +
  ylab("Counts") +
  scale_y_continuous(labels = comma) +
  theme_bw()
```



The user effects distribution looks **normal**. So, we predict it.

```r
# Predict the rating with mean + bi + bu
y_hat_bi_bu <- test %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

# Compute the RMSE and update the scores table
scores <- bind_rows(scores,
                    tibble(Method = "Mean + bi + bu",
                           RMSE = RMSE(test$rating, y_hat_bi_bu)))

# Display the RMSE improvement
```

```
scores %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Method	RMSE
RMSE Project Goal	0.8649000
Mean	1.0600537
Mean + bi	0.9429615
Mean + bi + bu	0.8646843

3.2.4 Verifying the model

We are going to check if the model Movie Effect (bi) makes good ratings predictions. For that, we need to create this new dataset using the `train` set that connects `movieId` to movie title.

```
titles_bi <- train %>%
  select(movieId, title) %>%
  distinct()
```

Now, we are going to display the 10 best movies based on `bi` (ranked by bi) with this code. Here we are using the `bi` and `titles_bi` datasets.

```
bi %>%
  inner_join(titles_bi, by = "movieId") %>%
  arrange(-b_i) %>%
  slice(1:10) %>%
  select(title) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(1.1)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

title
Hellhounds on My Trail (1999)
Satan’s Tango (Sátántangó) (1994)
Shadows of Forgotten Ancestors (1964)
Fighting Elegy (Kenka erejii) (1966)
Sun Alley (Sonnenallee) (1999)
Blue Light, The (Das Blaue Licht) (1932)
Who’s Singin’ Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)
Life of Oharu, The (Saikaku ichidai onna) (1952)
Human Condition II, The (Ningen no joken II) (1959)
Human Condition III, The (Ningen no joken III) (1961)

And here are the 10 worst movies according to `bi`.

```
bi %>%
  inner_join(titles_bi, by = "movieId") %>%
  arrange(b_i) %>%
  slice(1:10) %>%
  select(title) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(1.1)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

title
Besotted (2001)
Hi-Line, The (1999)
Accused (Anklaget) (2005)
Confessions of a Superhero (2007)
War of the Worlds 2: The Next Wave (2008)
SuperBabies: Baby Geniuses 2 (2004)
Disaster Movie (2008)
From Justin to Kelly (2003)
Hip Hop Witch, Da (2000)
Criminals (1996)

Number of ratings for 10 best movies based on `bi` using the `train` dataset.

```
train %>%
  left_join(bi, by = "movieId") %>%
  arrange(-b_i) %>%
  group_by(title) %>%
  summarise(n = n()) %>%
  slice(1:10) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(1.1, .1)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

title	**n**
'burbs, The (1989)	1201
'night Mother (1986)	178
'Round Midnight (1986)	40
'Til There Was You (1997)	242
"Great Performances" Cats (1998)	4
*batteries not included (1987)	389
...All the Marbles (a.k.a. The California Dolls) (1981)	17
...And God Created Woman (Et Dieu... créa la femme) (1956)	68
...And God Spoke (1993)	19
...And Justice for All (1979)	500

Let us inspect how often they are rated.

```
train %>% count(movieId) %>%
  left_join(bi, by="movieId") %>%
  arrange(-b_i) %>%
  slice(1:10) %>%
  pull(n)
```

```
##  [1] 1 1 1 1 1 1 4 2 4 4
```

```
train %>% count(movieId) %>%
  left_join(bi, by="movieId") %>%
  arrange(b_i) %>%
  slice(1:10) %>%
  pull(n)
```

```
## [1]   1   1   1   1   2  47  30 183  11   1
```

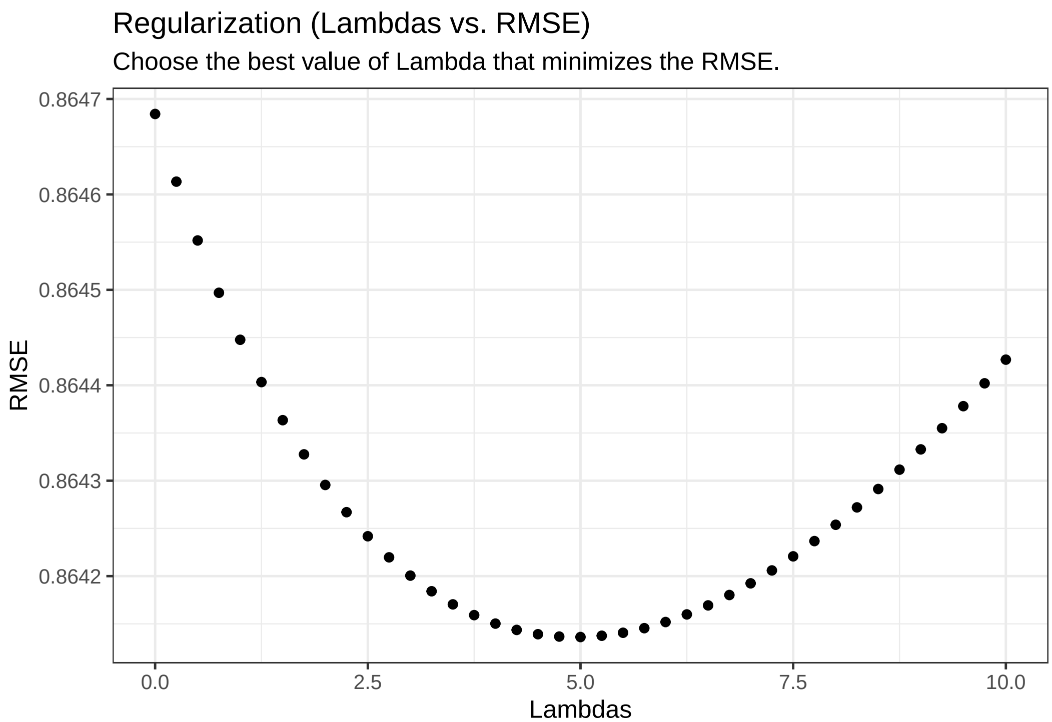
3.3 Regularization

The next process is **Regularization**, for that we need to regularize the movie and user effects aggregating a **penalty term** or factor which is known as **lambda** (λ) and which is also a **tuning parameter**. So, we establish a set of values for lambda and use **cross-validation** to pick the best value that **minimizes the RMSE**.

```
# establish a set of values for lambda
lambdas <- seq(0, 10, 0.25)

# use cross-validation for tuning lambda
# NOTE: Take aprox. 2 to 3 minutes
rmses <- sapply(lambdas, function(lambda){

  # Mean
  mu <- mean(train$rating)

  # Movie effects (bi)
  b_i <- train %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(n()+lambda))

  # User effects (bu)
  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - b_i - mu)/(n()+lambda))

  # Predict mu + bi + bu
  predicted_ratings <- test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    filter(!is.na(b_i), !is.na(b_u)) %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

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

Here is the plot of the Regularization (Lambdas versus RMSE).

```r
# Plot the Lambdas vs RMSE
tibble(Lambdas = lambdas, RMSE = rmses) %>%
  ggplot(aes(x = Lambdas, y = RMSE)) +
  geom_point() +
  ggtitle("Regularization (Lambdas vs. RMSE)",
          subtitle = "Choose the best value of Lambda that minimizes the RMSE.") +
  theme_bw()
```



According to the graph obtained, the best value of lambda is **5**. We can confirm this value with this code.

```r
# To know which is the best value of lambda
lambda <- lambdas[which.min(rmses)]
lambda
```

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

Now, we use the optimal value of lambda on the linear model.

```r
# Mean
mu <- mean(train$rating)

# Movie effects (bi)
```

```r
b_i <- train %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu)/(n()+lambda))

# User effects (bu)
b_u <- train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - b_i - mu)/(n()+lambda))

# Predict mu + bi + bu
y_hat_reg <- test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Compute the RMSE and update the scores table
scores <- bind_rows(scores,
                    tibble(Method = "Regularized bi + bu",
                           RMSE = RMSE(test$rating, y_hat_reg)))

# Display the RMSE improvement
scores %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Method	RMSE
RMSE Project Goal	0.8649000
Mean	1.0600537
Mean + bi	0.9429615
Mean + bi + bu	0.8646843
Regularized bi + bu	0.8641362

3.4 Ending Results in the Validation Set

3.4.1 Linear Model With Regularization

Previously we were able to verify that our Linear model with Regularization reached the goal of RMSE. Now, we will proceed to perform **the final validation** on the `validation` set, using the `edx` set like training set. Here is the code.

```
# Mean
mu_end_edx <- mean(edx$rating)

# Movie effects (bi)
bi_end_edx <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu_end_edx)/(n()+lambda))

# User effects (bu)
bu_end_edx <- edx %>%
  left_join(bi_end_edx, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - b_i - mu_end_edx)/(n()+lambda))

# Predict mu + bi + bu
y_hat_end_edx <- validation %>%
  left_join(bi_end_edx, by = "movieId") %>%
  left_join(bu_end_edx, by = "userId") %>%
  mutate(pred = mu_end_edx + b_i + b_u) %>%
  pull(pred)

# Compute the RMSE and update the scores table
scores <- bind_rows(scores,
                    tibble(Method = "Ending Regularization on edx and validation",
                           RMSE = RMSE(validation$rating, y_hat_end_edx)))

# Display the RMSE improvement
scores %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Method	RMSE
RMSE Project Goal	0.8649000
Mean	1.0600537
Mean + bi	0.9429615
Mean + bi + bu	0.8646843
Regularized bi + bu	0.8641362
Ending Regularization on edx and validation	0.8648177

We check if the **Ending Regularization** on `edx` and `validation` is minor than **RMSE GOAL**.

```
scores[6,] %>%
  as_hux() %>%
  set_font_size(10) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Method	RMSE
Ending Regularization on edx and validation	0.8648177

```
scores[6,2] < RMSE_GOAL
```

```
##      RMSE
## [1,] TRUE
```

As we could see, we achieved our goal.

We verify our **the final validation**, the Top 10 best movies.

```
validation %>%
  left_join(bi_end_edx, by = "movieId") %>%
  left_join(bu_end_edx, by = "userId") %>%
  mutate(pred = mu_end_edx + b_i + b_u) %>%
  arrange(-pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(1.1)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

title
Usual Suspects, The (1995)
Shawshank Redemption, The (1994)
Shawshank Redemption, The (1994)
Shawshank Redemption, The (1994)
Eternal Sunshine of the Spotless Mind (2004)
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
Schindler's List (1993)
Donnie Darko (2001)
Star Wars: Episode VI - Return of the Jedi (1983)
Schindler's List (1993)

And the Top 10 worst movies.

```
validation %>%
  left_join(bi_end_edx, by = "movieId") %>%
  left_join(bu_end_edx, by = "userId") %>%
  mutate(pred = mu_end_edx + b_i + b_u) %>%
  arrange(pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(1.1)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

title
Battlefield Earth (2000)
Police Academy 4: Citizens on Patrol (1987)
Karate Kid Part III, The (1989)
Pokémon Heroes (2003)
Turbo: A Power Rangers Movie (1997)
Kazaam (1996)
Pokémon Heroes (2003)
Free Willy 3: The Rescue (1997)
Shanghai Surprise (1986)
Steel (1997)

Chapter 4

Conclusion

We began by preparing our project, downloading and creating our datasets. Immediately, we perform data exploration and visualization on the `edx` dataset to find the relationship between the predict variable and the possible predictor variables for our model. We proceeded to prepare and clean the `train` and `test` datasets, where we decided to keep 4 variables (`userId`, `movieId`, `rating`, `title`), from the `edx` dataset.

Afterwards, we defined our **loss function** (**RMSE**) and created our first model to predict the mean of the ratings. Next, we aggregated the movie and user effects to that model to observe and model the behavior of these two effects.

Finally, we used regularization to normalize the movie and user effects adding a **penalty term** or **factor** which is known as **lambda**. We did a table to store the score for each model and we concluded that the linear model with regularization got the best score with a **RMSE** of 0.8648177 and it exceeded the project goal of 0.8649.

4.1 Limitations

Some limitations that we were able to find are:

- The computer equipment that does not have the necessary characteristics such as the processor and/or the ram memory to be able to run or execute some machine learning models, for example the models that we created, as well as, the internet connection we had to download the required data sets. On the other hand, when rendering the Rmd document to generate the PDF document, both computers, the Linux and Windows 10 operating systems, hung at 96%, the problem was when the tables were created with the `kable` function from the `knitr` package and the `kable_styling` function from the `kableExtra` package. The solution to that problem was to use the `as_hux()` function from the `huxtable` package.

- The model we built used two predictors, we know that other recommendation systems employ more than two predictors, and also, apply some methods like content-based and collaborative filtering. For that, we can use `recommenderlab` package.

4.2 Future work

We can make use of the `recommenderlab` package to get better results, because it applies some methods, like content-based and collaborative filtering, that were not implemented in this document. Also, it offers the necessary tools to create and test these kind of systems.

References

Agrawal, S. K. (2021, July 13). *Recommendation System - Understanding The Basic Concepts*. Analytics Vidhya. Retrieved August 13, 2022, from https://www.analyticsvidhya.com/blog/2021/07/recommendation-system-understanding-the-basic-concepts/.

Dilmegani, C. (2022, February 9). *Recommendation Systems: Applications and Examples in 2022*. AIMultiple. Retrieved August 13, 2022, from https://research.aimultiple.com/recommendation-system/.

Gupta, S. (2022, May 16). *Data Science Process: A Beginner's Guide in Plain English*. Springboard Blog. Retrieved August 17, 2022, from https://www.springboard.com/blog/data-science/data-science-process/.

Harper, F. M., & Konstan, J. A. (2016). The MovieLens Datasets. *ACM Transactions on Interactive Intelligent Systems*, 5(4), 1–19. https://doi.org/10.1145/2827872.

Irizarry, R. A. (2022, July 7). *Introduction to Data Science: Data Analysis and Prediction Algorithms with R*. Github.Io. Retrieved August 14, 2022, from https://rafalab.github.io/dsbook/.

Nantasenamat, C. (2022, January 14). *The Data Science Process - Towards Data Science*. Medium. Retrieved August 17, 2022, from https://towardsdatascience.com/the-data-science-process-a19eb7ebc41b.

Singh, A. (2019, March 20). *Evaluation Metrics for Regression models- MAE Vs MSE Vs RMSE vs RMSLE*. Akhilendra.Com. Retrieved August 19, 2022, from https://akhilendra.com/evaluation-metrics-regression-mae-mse-rmse-rmsle/.

Appendix

.1 Appendix A - Computer equipment with Windows 10 OS

The Operating Systems (OS) where this code was made is Windows 10 and its specs are:

```
System type: 64 bits
Edition:     Windows 10 Home
Version:     21H2
OS build:    19044.1889
Experience:  Windows Feature Experience Pack 120.2212.4180.0
```

Hardware specs:

```
Processor: Intel(R) Core(TM) i5-10500H CPU @ 2.50GHz
    - Cores: 6
    - Logical processors (Threads): 12
RAM Memory:  24 GB.
```

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
## locale:
## [1] LC_COLLATE=Spanish_Latin America.utf8
```

```
## [2] LC_CTYPE=Spanish_Latin America.utf8
## [3] LC_MONETARY=Spanish_Latin America.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=Spanish_Latin America.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] huxtable_5.5.0    caret_6.0-93      lubridate_1.8.0   Hmisc_4.7-1
##  [5] Formula_1.2-4     survival_3.3-1    lattice_0.20-45   scales_1.2.1
##  [9] ggthemes_4.2.4    data.table_1.14.2 forcats_0.5.2     stringr_1.4.1
## [13] dplyr_1.0.9       purrr_0.3.4       readr_2.1.2       tidyr_1.2.0
## [17] tibble_3.1.8      ggplot2_3.3.6     tidyverse_1.3.2   this.path_0.8.0
##
## loaded via a namespace (and not attached):
##  [1] googledrive_2.0.0    colorspace_2.0-3     deldir_1.0-6
##  [4] ellipsis_0.3.2       class_7.3-20         htmlTable_2.4.1
##  [7] base64enc_0.1-3      fs_1.5.2             rstudioapi_0.14
## [10] farver_2.1.1         listenv_0.8.0        prodlim_2019.11.13
## [13] fansi_1.0.3          xml2_1.3.3           codetools_0.2-18
## [16] splines_4.2.1        knitr_1.40           jsonlite_1.8.0
## [19] pROC_1.18.0          broom_1.0.1          cluster_2.1.3
## [22] dbplyr_2.2.1         png_0.1-7            compiler_4.2.1
## [25] httr_1.4.4           backports_1.4.1      assertthat_0.2.1
## [28] Matrix_1.4-1         fastmap_1.1.0        gargle_1.2.0
## [31] cli_3.3.0            htmltools_0.5.3      tools_4.2.1
## [34] gtable_0.3.0         glue_1.6.2           reshape2_1.4.4
## [37] Rcpp_1.0.9           cellranger_1.1.0     vctrs_0.4.1
## [40] nlme_3.1-157         iterators_1.0.14     timeDate_4021.104
## [43] xfun_0.32            gower_1.0.0          globals_0.16.1
## [46] rvest_1.0.3          lifecycle_1.0.1      googlesheets4_1.0.1
## [49] future_1.27.0        MASS_7.3-57          ipred_0.9-13
## [52] hms_1.1.2            parallel_4.2.1       RColorBrewer_1.1-3
## [55] yaml_2.3.5           gridExtra_2.3        rpart_4.1.16
## [58] latticeExtra_0.6-30  stringi_1.7.8        foreach_1.5.2
## [61] checkmate_2.1.0      hardhat_1.2.0        lava_1.6.10
## [64] commonmark_1.8.0     rlang_1.0.4          pkgconfig_2.0.3
## [67] evaluate_0.16        labeling_0.4.2       recipes_1.0.1
## [70] htmlwidgets_1.5.4    tidyselect_1.1.2     parallelly_1.32.1
## [73] plyr_1.8.7           magrittr_2.0.3       R6_2.5.1
## [76] generics_0.1.3       DBI_1.1.3            pillar_1.8.1
## [79] haven_2.5.1          foreign_0.8-82       withr_2.5.0
## [82] nnet_7.3-17          future.apply_1.9.0   modelr_0.1.9
## [85] crayon_1.5.1         interp_1.1-3         utf8_1.2.2
```

```
## [88] tzdb_0.3.0          rmarkdown_2.16      jpeg_0.1-9
## [91] grid_4.2.1          readxl_1.4.1        ModelMetrics_1.2.2.2
## [94] reprex_2.0.2        digest_0.6.29       stats4_4.2.1
## [97] munsell_0.5.0
```

.2 Appendix B - Computer equipment with an Ubuntu Linux Distribution OS

The Operating Systems (OS) where this code was tested is Zorin OS 16.1 and its specs are:

```
System type:      64 bits
Edition:          Zorin OS Education
Description:      Zorin OS 16.1
Ubuntu_codename:  focal
```

Hardware specs:

```
Processor: Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz
    - Cores: 2
    - Logical processors (Threads): 4
RAM Memory:  8 GB.
```

```
sessionInfo()
```

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Zorin OS 16.1

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=es_MX.UTF-8       LC_NUMERIC=C               LC_TIME=es_MX.UTF-8        LC_COLLATE=es_MX.UTF-8
 [5] LC_MONETARY=es_MX.UTF-8    LC_MESSAGES=es_MX.UTF-8    LC_PAPER=es_MX.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C             LC_MEASUREMENT=es_MX.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] huxtable_5.5.0     caret_6.0-93       lubridate_1.8.0    Hmisc_4.7-0        Formula_1.2-4      survival_3.4-0
 [7] lattice_0.20-45    scales_1.2.0       ggthemes_4.2.4     data.table_1.14.2  forcats_0.5.1      stringr_1.4.0
[13] dplyr_1.0.9        purrr_0.3.4        readr_2.1.2        tidyr_1.2.0        tibble_3.1.8       ggplot2_3.3.6
[19] tidyverse_1.3.2    this.path_0.8.0

loaded via a namespace (and not attached):
 [1] googledrive_2.0.0    colorspace_2.0-3     deldir_1.0-6         ellipsis_0.3.2       class_7.3-20
 [6] htmlTable_2.4.1      base64enc_0.1-3      fs_1.5.2             rstudioapi_0.13      listenv_0.8.0
[11] prodlim_2019.11.13   fansi_1.0.3          xml2_1.3.3           codetools_0.2-18     splines_4.2.1
```

```
[16] knitr_1.39           jsonlite_1.8.0       pROC_1.18.0          broom_1.0.0          cluster_2.1.4
[21] dbplyr_2.2.1         png_0.1-7            compiler_4.2.1       httr_1.4.3           backports_1.4.1
[26] assertthat_0.2.1     Matrix_1.4-1         fastmap_1.1.0        gargle_1.2.0         cli_3.3.0
[31] htmltools_0.5.3      tools_4.2.1          gtable_0.3.0         glue_1.6.2           reshape2_1.4.4
[36] Rcpp_1.0.9           cellranger_1.1.0     vctrs_0.4.1          nlme_3.1-159         iterators_1.0.14
[41] timeDate_4021.104    xfun_0.32            gower_1.0.0          globals_0.16.0       rvest_1.0.2
[46] lifecycle_1.0.1      googlesheets4_1.0.0  future_1.27.0        MASS_7.3-58.1        ipred_0.9-13
[51] hms_1.1.1            parallel_4.2.1       RColorBrewer_1.1-3   yaml_2.3.5           gridExtra_2.3
[56] rpart_4.1.16         latticeExtra_0.6-30  stringi_1.7.8        foreach_1.5.2        checkmate_2.1.0
[61] hardhat_1.2.0        lava_1.6.10          rlang_1.0.4          pkgconfig_2.0.3      evaluate_0.16
[66] recipes_1.0.1        htmlwidgets_1.5.4    tidyselect_1.1.2     parallelly_1.32.1    plyr_1.8.7
[71] magrittr_2.0.3       R6_2.5.1             generics_0.1.3       DBI_1.1.3            pillar_1.8.0
[76] haven_2.5.0          foreign_0.8-82       withr_2.5.0          nnet_7.3-17          future.apply_1.9.0
[81] modelr_0.1.8         crayon_1.5.1         interp_1.1-3         utf8_1.2.2           tzdb_0.3.0
[86] rmarkdown_2.14       jpeg_0.1-9           grid_4.2.1           readxl_1.4.0         ModelMetrics_1.2.2.2
[91] reprex_2.0.1         digest_0.6.29        stats4_4.2.1         munsell_0.5.0
```