Harvard Data Science with R - Movielens Capstone Project

Albertus Erwin Susanto

```r
################################################################################

# HARVARD DATA SCIENCE WITH R
# CAPSTONE PROJECT
# Created by: Albertus Erwin Susanto    (IDN)

# A. INTRODUCTION ==============================================================
# (see the pdf report)
# (note: the numbering of this code corresponds with the numbering in the pdf
#        for better cross-checking.)

# B. DATA PREPARATION ==========================================================

# Create edx and final_holdout_test sets
# Note: this process could take a couple of minutes
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
##      tidyverse

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

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

```
##      caret

## Warning:   'caret' R 4.3.3

##      lattice
##
##    'caret'
##
```

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

```r
library(tidyverse)
library(caret)

# The source of MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

# Downloading the dataset
dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)

# Checking the data
head(ratings)
```

```
##   V1  V2 V3        V4
## 1  1 122  5 838985046
## 2  1 185  5 838983525
## 3  1 231  5 838983392
## 4  1 292  5 838983421
## 5  1 316  5 838983392
## 6  1 329  5 838983392
```

```r
# Formatting the data
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))
```

```r
# We combine our data into one
movielens <- left_join(ratings, movies, by = "movieId")

# We take only 10 percent of the whole dataset, cause it's just to big
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE) # set.seed(1)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

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

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


# C. QUIZ ON MOVIELENS DATA SET ================================================
# SERVING AS AN INTRODUCTION TO THE DATASET

head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

```r
# Q1
# How many rows and columns are there in the edx dataset?
# Number of rows:
nrow(edx)
```

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

```r
# Number of columns:
ncol(edx)
```

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

```r
colnames(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```r
sapply(edx, class)
```

```
##      userId     movieId      rating   timestamp       title      genres
##   "integer"   "integer"   "numeric"   "integer" "character" "character"
```

```r
# Q2
# How many zeros were given as ratings in the edx dataset?
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

```r
sum(edx$rating == 0)
```

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

```r
# How many threes were given as ratings in the edx dataset?
sum(edx$rating == 3)
```

```
## [1] 2121240
```

```r
# Q3
# How many different movies are in the edx dataset?
edx %>% distinct(movieId) %>% count()
```

```
##       n
## 1 10677
```

```r
# Q4
# How many different users are in the edx dataset?
edx %>% distinct(userId) %>% count()
```

```
##       n
## 1 69878
```

```r
# Q5
# How many movie ratings are in each of the following genres in the edx dataset?
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

```r
# Drama
edx %>% filter(str_detect(genres, "Drama")) %>% count()
```

```
##         n
## 1 3910127
```

```r
# Comedy
edx %>% filter(str_detect(genres, "Comedy")) %>% count()
```

```
##         n
## 1 3540930
```

```r
# Thriller
edx %>% filter(str_detect(genres, "Thriller")) %>% count()
```

```
##         n
## 1 2325899
```

```r
# Romance
edx %>% filter(str_detect(genres, "Romance")) %>% count()
```

```
##         n
## 1 1712100
```

```r
# Q6
# Which movie has the greatest number of ratings?
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

```r
edx %>%
  group_by(movieId) %>%
  summarize(title = first(title), number_ratings = n()) %>%
  arrange(desc(number_ratings)) %>%
  slice(1)
```

```
## # A tibble: 1 x 3
##   movieId title               number_ratings
##     <int> <chr>                        <int>
## 1     296 Pulp Fiction (1994)          31362
```

```r
# To have a full list:
# install.packages("openxlsx") # Install the openxlsx package if you don't have it
library(openxlsx)
```

```
## Warning:   'openxlsx' R 4.3.3
```

```r
movie_ratings_list <- edx %>%
  group_by(movieId) %>%
  summarize(title = first(title), number_ratings = n()) %>%
  mutate(rank = rank(desc(number_ratings))) %>%
  arrange(desc(number_ratings))
write.xlsx(movie_ratings_list,
           file = "movie_ratings.xlsx",
           sheetName = "Ratings Summary",
           rowNames = FALSE)

# Q7
# What are the five most given ratings in order from most to least?
edx %>%
  group_by(rating) %>%
  summarize(number_of_instances = n()) %>%
  arrange(desc(number_of_instances))
```

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

```r
# Q8
# True or False:
# 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.).
rating_counts <- edx %>%
  mutate(rating_category = ifelse(rating %% 1 == 0, "whole", "half")) %>%
  group_by(rating_category) %>%
  summarize(count = n())
rating_counts
```

```
## # A tibble: 2 x 2
##   rating_category   count
##   <chr>             <int>
## 1 half            1843170
## 2 whole           7156885
```

```r
# D. DATA WRANGLING & DATA EXPLORATION =========================================

# 1. "How many rows does it contain? How many columns and what are their names?"

# --- Number of rows:
nrow(edx)
```

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

```r
# ---Number of columns:
ncol(edx)
```

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

```r
colnames(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```r
sapply(edx, class)
```

```
##      userId     movieId      rating   timestamp       title      genres
##   "integer"   "integer"   "numeric"   "integer" "character" "character"
```

```r
# ---Changing the time format
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

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

# 2. "How many different movies and users are in the edx dataset?"
# --- How many different movies are in the edx dataset?
edx %>% distinct(movieId) %>% count()
```

```
##       n
## 1 10677
```

```r
# --- How many different users are in the edx dataset?
edx %>% distinct(userId) %>% count()
```

```
##       n
## 1 69878
```

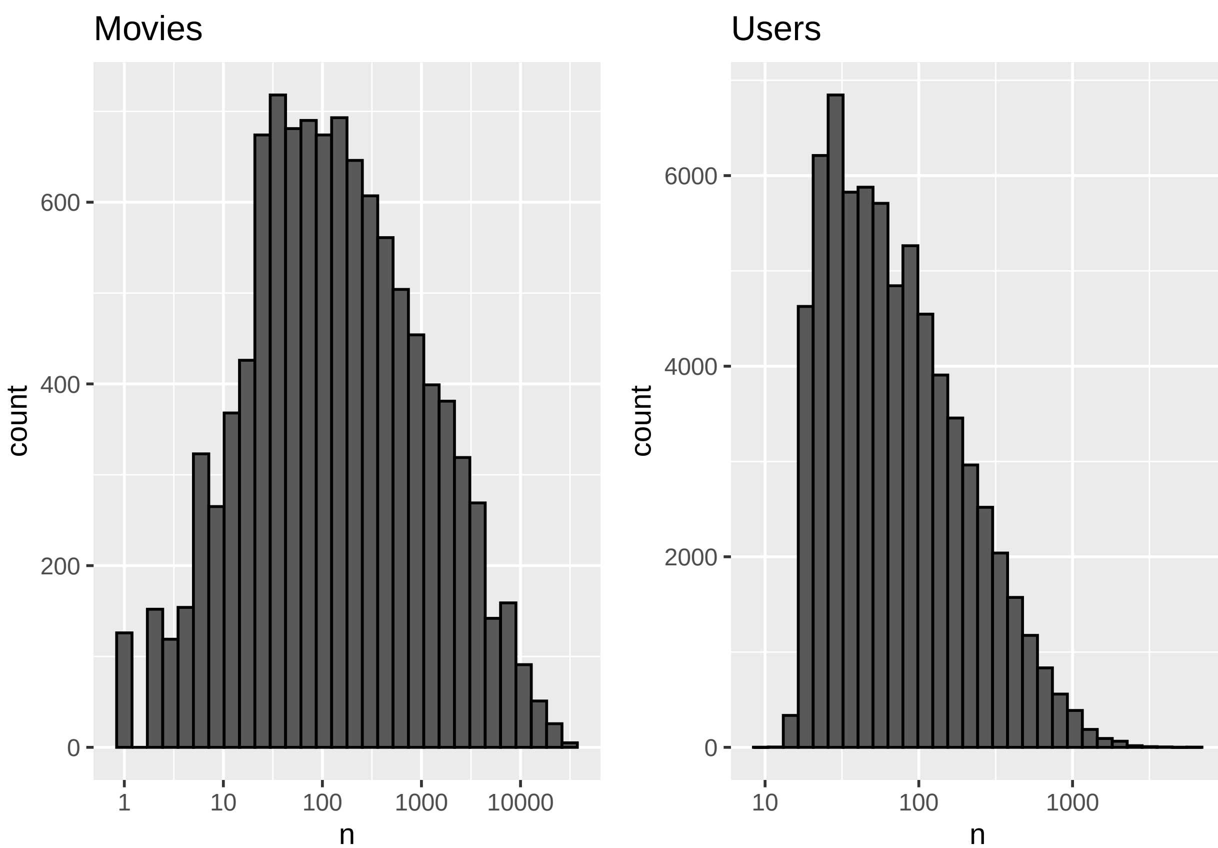
```r
# 3. "How many movies and users are matched?"
# --- We wanna check how many movies are rated by how many users
# Library for Showing Grids of Graphs
library(gridExtra)
```

```
## Warning:   'gridExtra' R 4.3.3
```

```
##
##    'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

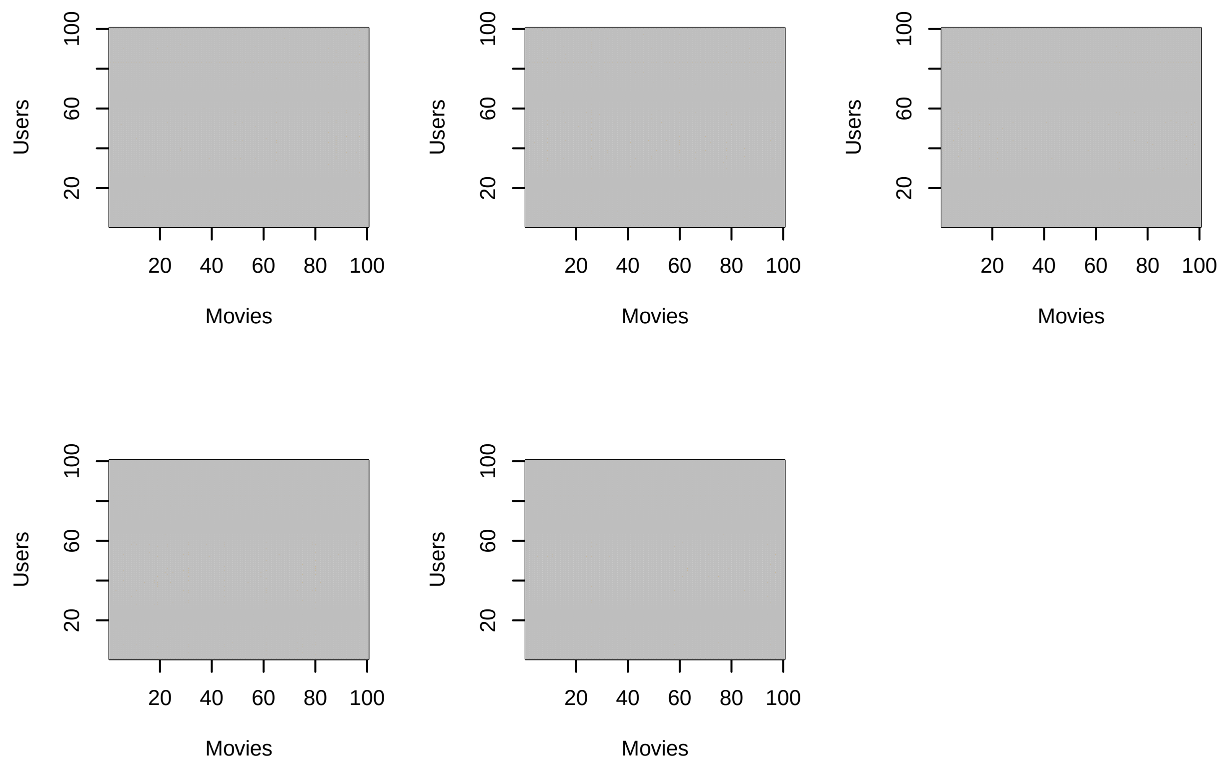
```r
# Movie's Histogram
p1 <- edx %>%
  count(movieId) %>%
```

```r
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")

# User's Histogram
p2 <- edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")

# Show the two Histograms
grid.arrange(p1, p2, ncol = 2)
```




```r
# --- Now we want to know better the data as to how many movies are rated by how many users.
par(mfrow = c(2, 3))
users <- sample(unique(edx$userId), 100)
r1 <- edx %>% filter(userId %in% users) %>%
  dplyr::select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  pivot_wider(names_from = movieId, values_from = rating) %>%
  (\(mat) mat[, sample(ncol(mat), 100)])()%>%
  as.matrix() %>%
```

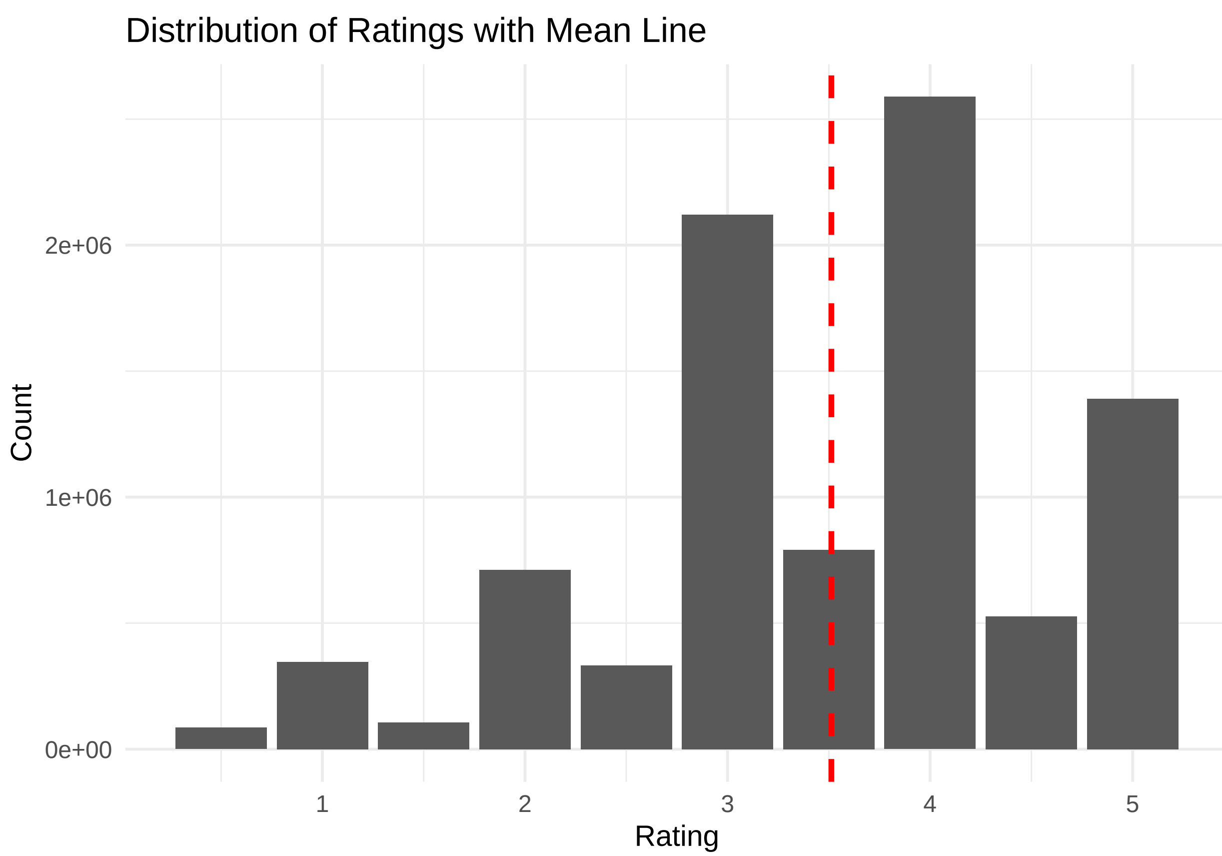
```
  t() %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
r2 <- edx %>% filter(userId %in% users) %>%
  dplyr::select(userId, movieId, rating) %>%
  mutate(rating = 2) %>%
  pivot_wider(names_from = movieId, values_from = rating) %>%
  (\(mat) mat[, sample(ncol(mat), 100)])()%>%
  as.matrix() %>%
  t() %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
r3 <- edx %>% filter(userId %in% users) %>%
  dplyr::select(userId, movieId, rating) %>%
  mutate(rating = 3) %>%
  pivot_wider(names_from = movieId, values_from = rating) %>%
  (\(mat) mat[, sample(ncol(mat), 100)])()%>%
  as.matrix() %>%
  t() %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
r4 <- edx %>% filter(userId %in% users) %>%
  dplyr::select(userId, movieId, rating) %>%
  mutate(rating = 4) %>%
  pivot_wider(names_from = movieId, values_from = rating) %>%
  (\(mat) mat[, sample(ncol(mat), 100)])()%>%
  as.matrix() %>%
  t() %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
r5 <- edx %>% filter(userId %in% users) %>%
  dplyr::select(userId, movieId, rating) %>%
  mutate(rating = 5) %>%
  pivot_wider(names_from = movieId, values_from = rating) %>%
  (\(mat) mat[, sample(ncol(mat), 100)])()%>%
  as.matrix() %>%
  t() %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
par(mfrow = c(1, 1))
```




```
# 4. "What are the top ten most rated movies?"
# Add openxlsx Library to Transfer the Result to Excel
library(openxlsx)

# Creating the List
movie_ratings_list <- edx %>%
  group_by(movieId) %>%
  summarize(title = first(title), number_ratings = n()) %>%
  mutate(rank = rank(desc(number_ratings))) %>%
  arrange(desc(number_ratings))

# Tranfering to Excel
write.xlsx(movie_ratings_list,
           file = "movie_ratings.xlsx",
           sheetName = "Ratings Summary",
           rowNames = FALSE)

# --- Now we want to check the mean and distribution of rating value
# Checking the mean and distribution of rating value
rating_distribution <- edx %>%     # Calculate the distribution of ratings
  group_by(rating) %>%
  count()

mean_rating <- mean(edx$rating) # Calculate the mean of the ratings

rating_distribution %>%  # Plot the distribution using ggplot2 and add a vertical line for the mean
```

```r
  ggplot(aes(x = rating, y = n)) +
  geom_bar(stat = "identity") +      # Use stat = "identity" because counts are pre-calculated
  geom_vline(aes(xintercept = mean_rating), color = "red", linetype = "dashed", size = 1) +  # Add mean
  labs(x = "Rating", y = "Count", title = "Distribution of Ratings with Mean Line") +  # Add labels
  theme_minimal()                    # Apply a clean theme
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```r
# 5. "Is there an obvious difference of rating among different movies,
#     indicating a movie-specific rating mean (movie-effect),
#     as well as user-specific rating mean (user-effect)?"
library(ggthemes)
```

```
## Warning:   'ggthemes' R 4.3.3
```

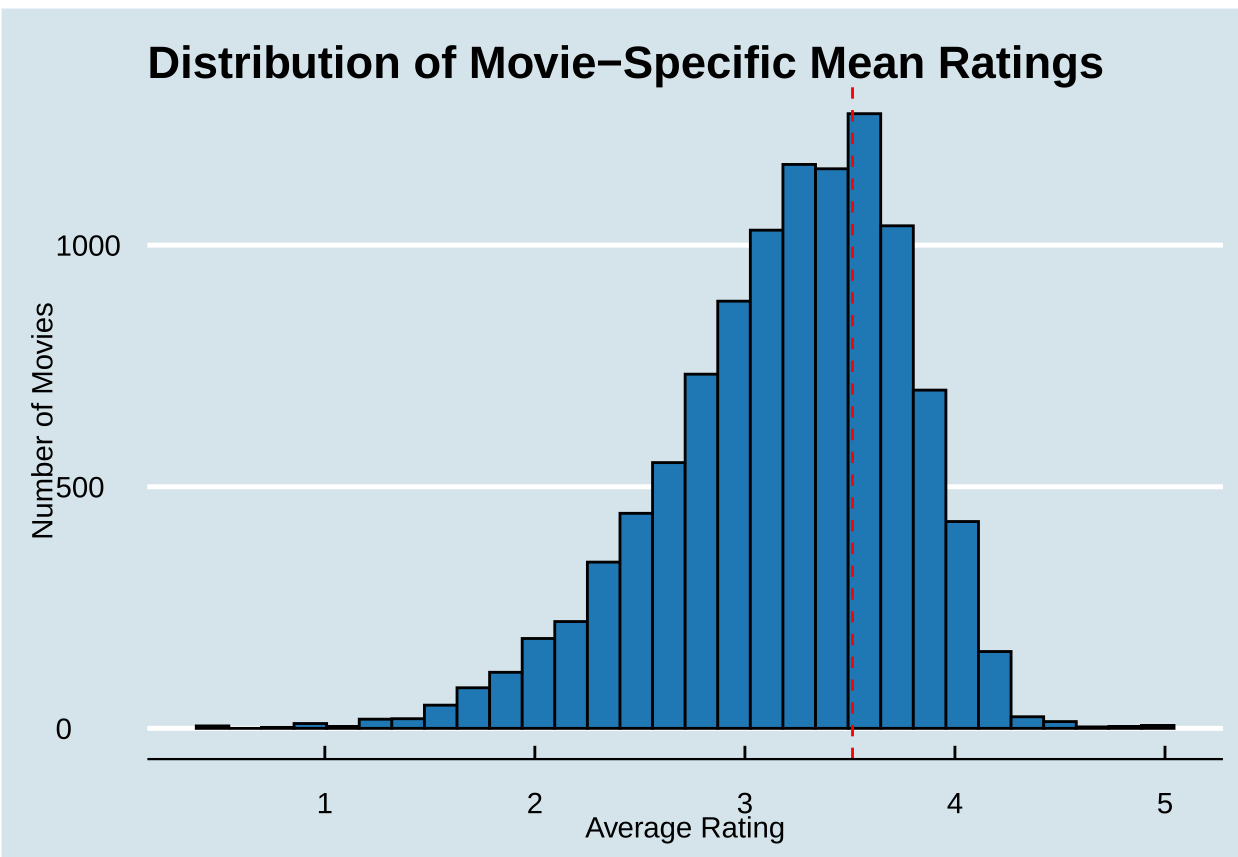
```r
# Overall mean rating
mean_rating
```

```
## [1] 3.512465
```

```r
# Movie-specific mean rating
movie_avg_ratings <- edx %>%
  group_by(movieId) %>%
  summarize(movie_mean = mean(rating))

# Compare with overall mean
head(movie_avg_ratings)
```

```
## # A tibble: 6 x 2
##   movieId movie_mean
##     <int>      <dbl>
## 1       1       3.93
## 2       2       3.21
## 3       3       3.15
## 4       4       2.86
## 5       5       3.07
## 6       6       3.82
```

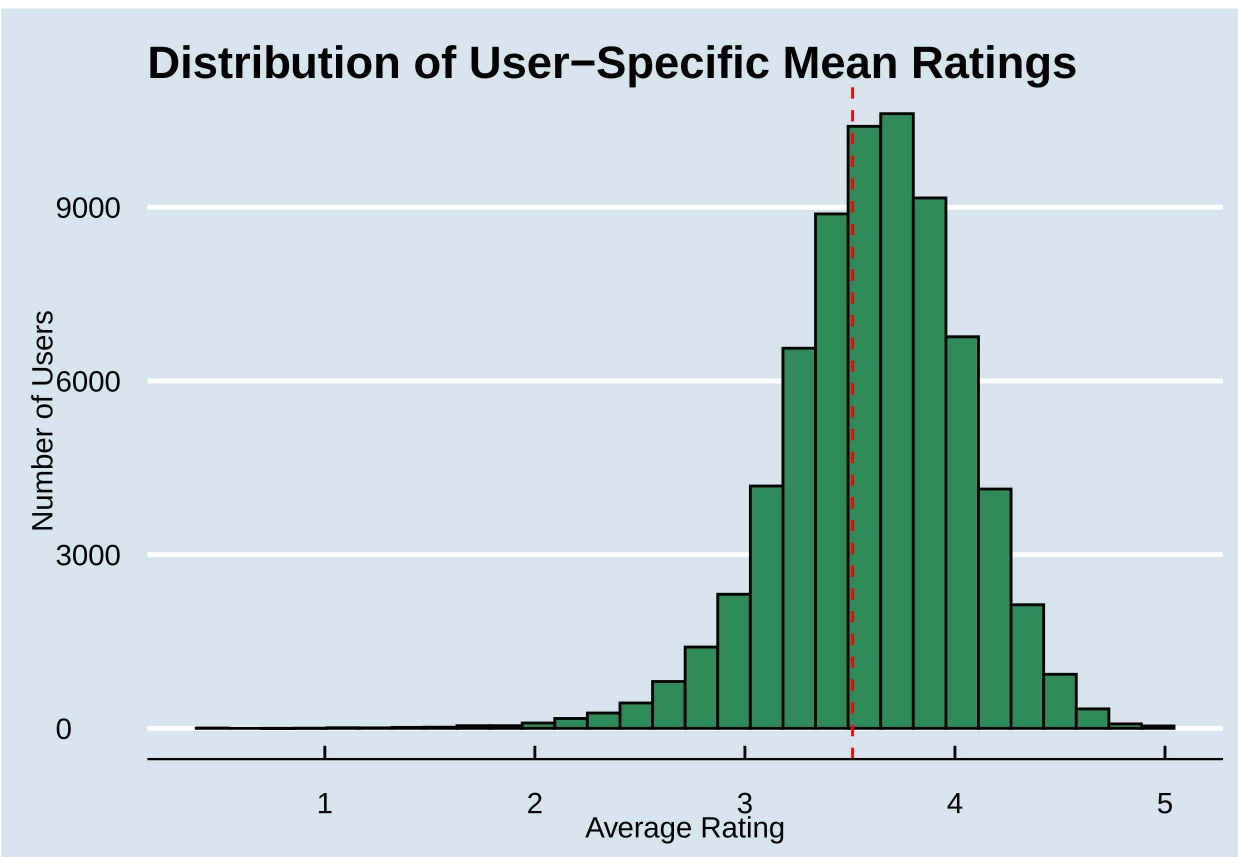
```r
# Plot the distribution of movie-specific means
ggplot(movie_avg_ratings, aes(x = movie_mean)) +
  geom_histogram(bins = 30, color = "black", fill = "#1f77b4") +
  geom_vline(xintercept = mean_rating, linetype = "dashed", color = "red") +
  labs(title = "Distribution of Movie-Specific Mean Ratings", x = "Average Rating", y = "Number of Movi
  theme_economist()
```



```r
# User-specific mean rating
user_avg_ratings <- edx %>%
  group_by(userId) %>%
  summarize(user_mean = mean(rating))

# Compare with overall mean
head(user_avg_ratings)
```

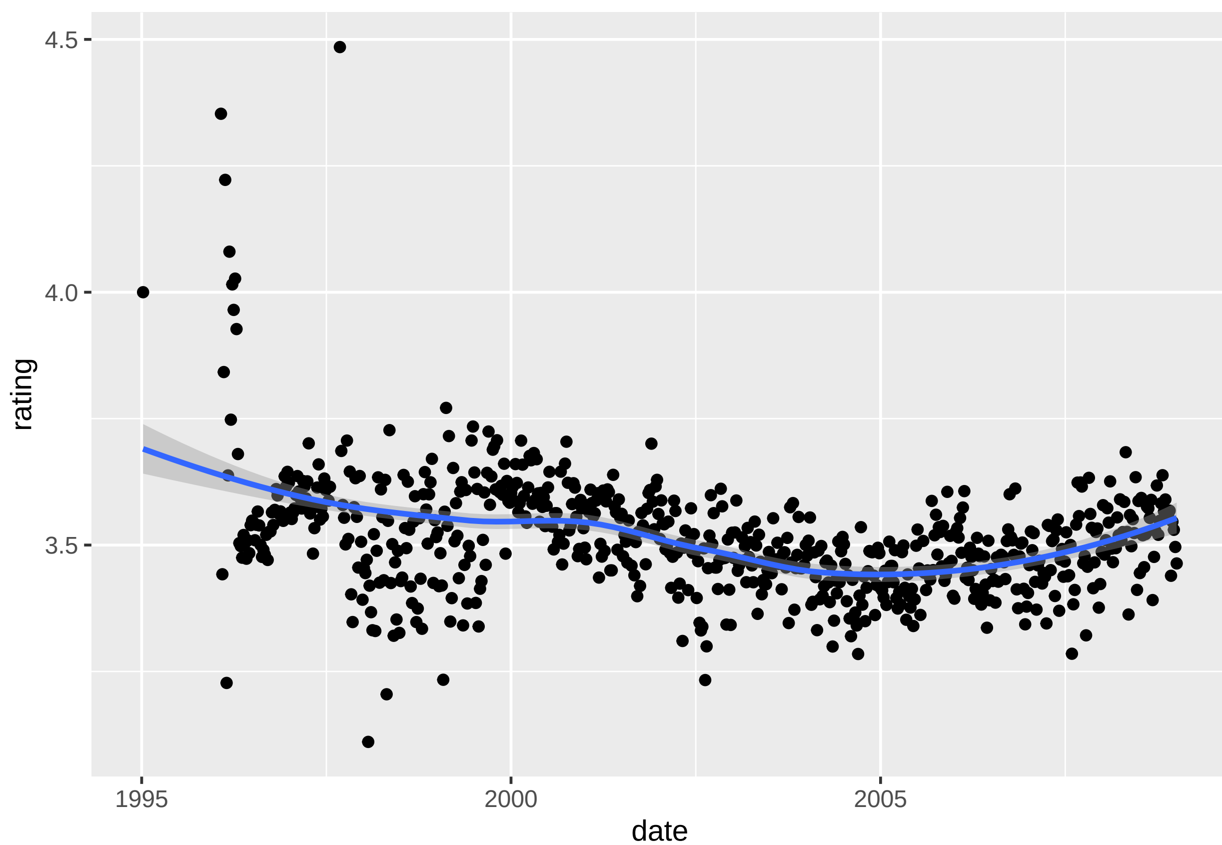
```
## # A tibble: 6 x 2
##   userId user_mean
##    <int>     <dbl>
## 1      1      5   
## 2      2      3.29
## 3      3      3.94
## 4      4      4.06
## 5      5      3.92
## 6      6      3.95
```

```r
# Plot the distribution of user-specific means
ggplot(user_avg_ratings, aes(x = user_mean)) +
  geom_histogram(bins = 30, color = "black", fill = "#2E8B57") +
  geom_vline(xintercept = mean_rating, linetype = "dashed", color = "red") +
  labs(title = "Distribution of User-Specific Mean Ratings", x = "Average Rating", y = "Number of Users
  theme_economist()
```

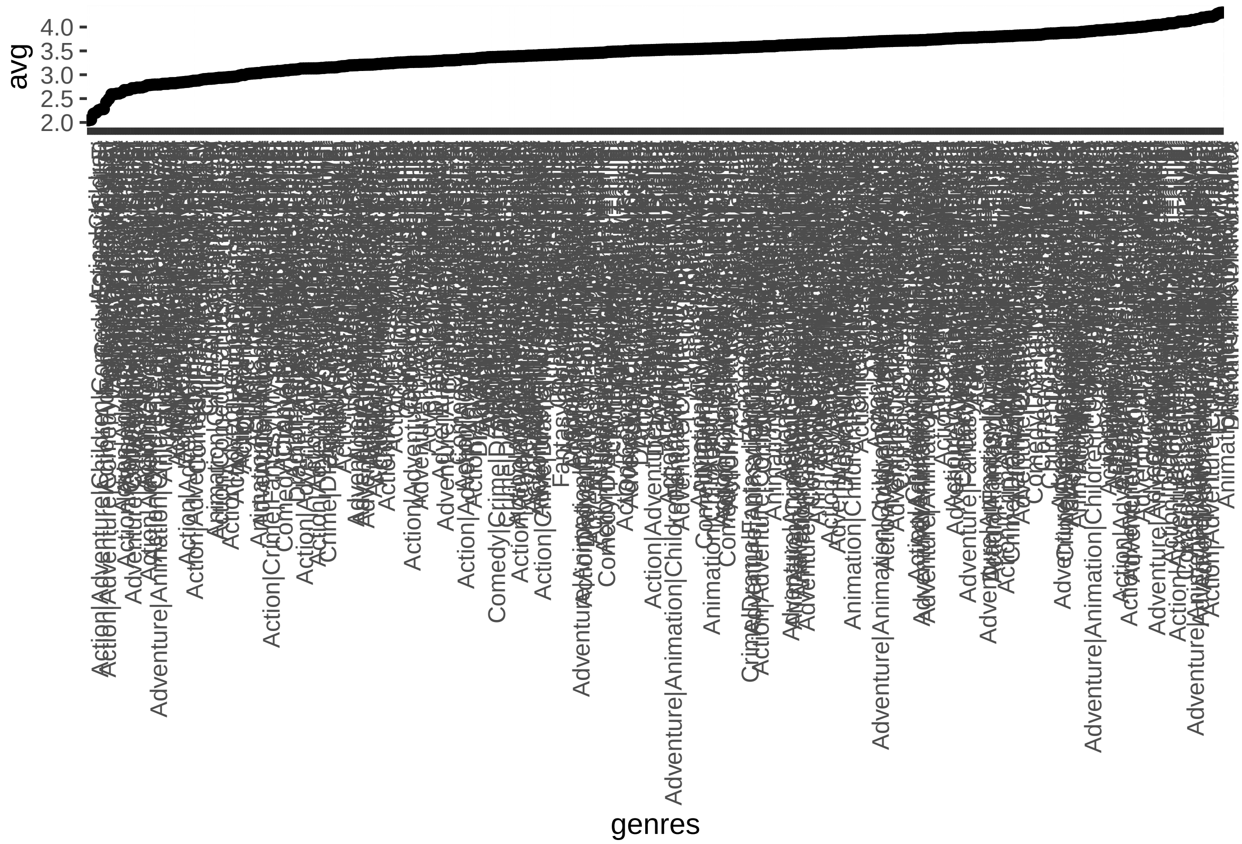


```r
# 6. "Is there a trend/fluctuation in the rating given from time-to-time?"
# Seeing if there's time effect
edx %>% mutate(date = round_date(date, unit = "week")) %>% # we first transform the format of time
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth()
```

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




```r
# 7. "Is there obvious difference of rating among different genres,
#     indicating the effects of genre on movie rating?"
# --- Considering genre effect
genre_effect <- edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 1000) %>%
  mutate(genres = reorder(genres, avg))

genre_effect %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```




```r
best_rated_genre <- genre_effect %>% arrange(desc(avg)) %>% slice(1)  # Top-rated genre
best_rated_genre
```

```
## # A tibble: 1 x 4
##   genres                          n   avg     se
##   <fct>                       <int> <dbl>  <dbl>
## 1 Drama|Film-Noir|Romance  2989  4.30 0.0145
```

```r
least_rated_genre <- genre_effect %>% arrange(avg) %>% slice(1)       # Least-rated genre
least_rated_genre
```

```
## # A tibble: 1 x 4
##   genres               n   avg     se
##   <fct>            <int> <dbl>  <dbl>
## 1 Action|Children  3922  2.04 0.0174
```

```r
# E. BUILDING OUR MODEL ========================================================

# Partitioning the Data into Train and Test Set
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
test_set <- test_set %>%
```

```r
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# RMSE Function as the Evaluation Measure of Models
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

# 1. Model 1 - Predicting Simply Using the Mean ----------------------------
# We count the mean:
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512439
```

```r
# We see the RMSE of using mean in the test set:
model1_rmse <- RMSE(test_set$rating, mu_hat)
model1_rmse
```

```
## [1] 1.060023
```

```r
# We compare the use of mean in predicting the result
# with using the median of the scale (not the median of the ratings given)
# which is 2.5
median_scale_prediction <- rep(2.5, nrow(test_set))
model0_rmse <- RMSE(test_set$rating, median_scale_prediction)

# We input it into our collection of results of models RMSEs
rmse_results <- data_frame(Method = "Constant Value 2.5", RMSE = model0_rmse)
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## i Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Using the Data Set Mean", RMSE = model1_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   Method                   RMSE
##   <chr>                   <dbl>
## 1 Constant Value 2.5       1.47
## 2 Using the Data Set Mean  1.06
```

```r
# We export the result to an excel sheet:
# install.packages("writexl") --- in case haven't been installed
library(writexl)
```

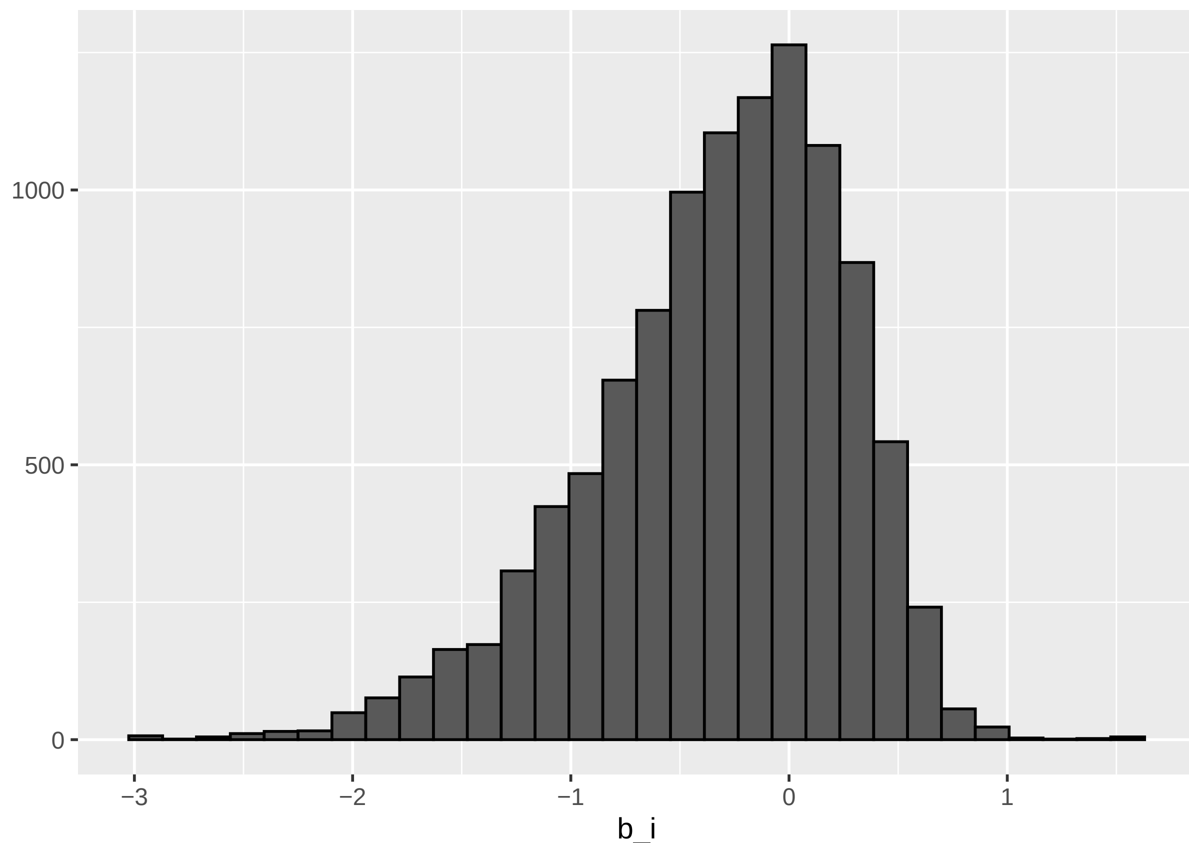
```
## Warning:   'writexl' R 4.3.3
```

```r
write_xlsx(rmse_results, "rmse_results.xlsx")

# 2. Model 2 - Predicting Using the Movie Effect---------------------------

# The course in Machine Learning introduced us to this code below,
# with a caveat that it will take longer time. So we'll just leave it.
# fit <- lm(rating ~ as.factor(userId), data = movielens)

# We use the simpler way of predicting using movie effect
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)) # here we count the mean of the difference of rating mean of movie

# Let's check the value of b_i
movie_avgs
```

```
## # A tibble: 10,635 x 2
##    movieId     b_i
##      <int>   <dbl>
##  1       1  0.413
##  2       2 -0.310
##  3       3 -0.361
##  4       4 -0.624
##  5       5 -0.442
##  6       6  0.310
##  7       7 -0.158
##  8       8 -0.354
##  9       9 -0.507
## 10      10 -0.0836
## # i 10,625 more rows
```

```r
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 30, data = ., color = I("black"))
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```r
# Input the b_i into the test_set
test_set_with_bi <- test_set %>%
  left_join(movie_avgs, by = 'movieId')

# We predict the rating in the test_set
predicted_ratings <- mu + test_set_with_bi$b_i

# Evaluate the efficacy of this model
model2_rmse <- RMSE(predicted_ratings, test_set$rating)

# We input it into our collection of results of models RMSEs
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Movie Effect Model",
                                     RMSE = model2_rmse ))
rmse_results
```

```
## # A tibble: 3 x 2
##   Method                    RMSE
##   <chr>                    <dbl>
## 1 Constant Value 2.5       1.47
## 2 Using the Data Set Mean 1.06
## 3 Movie Effect Model      0.943
```

```r
# We export the result to an excel sheet:
write_xlsx(rmse_results, "rmse_results.xlsx")
```

```r
# 3. Model 3 - Predicting Using the User Effect + Movie Effect -------------

# Again, the course in Machine Learning introduced us to this code below,
# with a caveat that it will take longer time. So we'll just leave it.
# lm(rating ~ as.factor(movieId) + as.factor(userId))

# We use the simpler way of predicting using user effect (on top of the movie effect)
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i)) # note that we take away the movie effect to have a more pre

# Input the b_u into the test_set
# note that in our third model, we also take into our prediction the movie effect, so we also
test_set_with_bi_bu <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId')

# We predict the rating in the test_set
predicted_ratings <- test_set_with_bi_bu %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# We evaluate the model
model3_rmse <- RMSE(predicted_ratings, test_set$rating)

# We input into our rmse collection
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Movie + User Effects Model",
                                     RMSE = model3_rmse ))
rmse_results
```

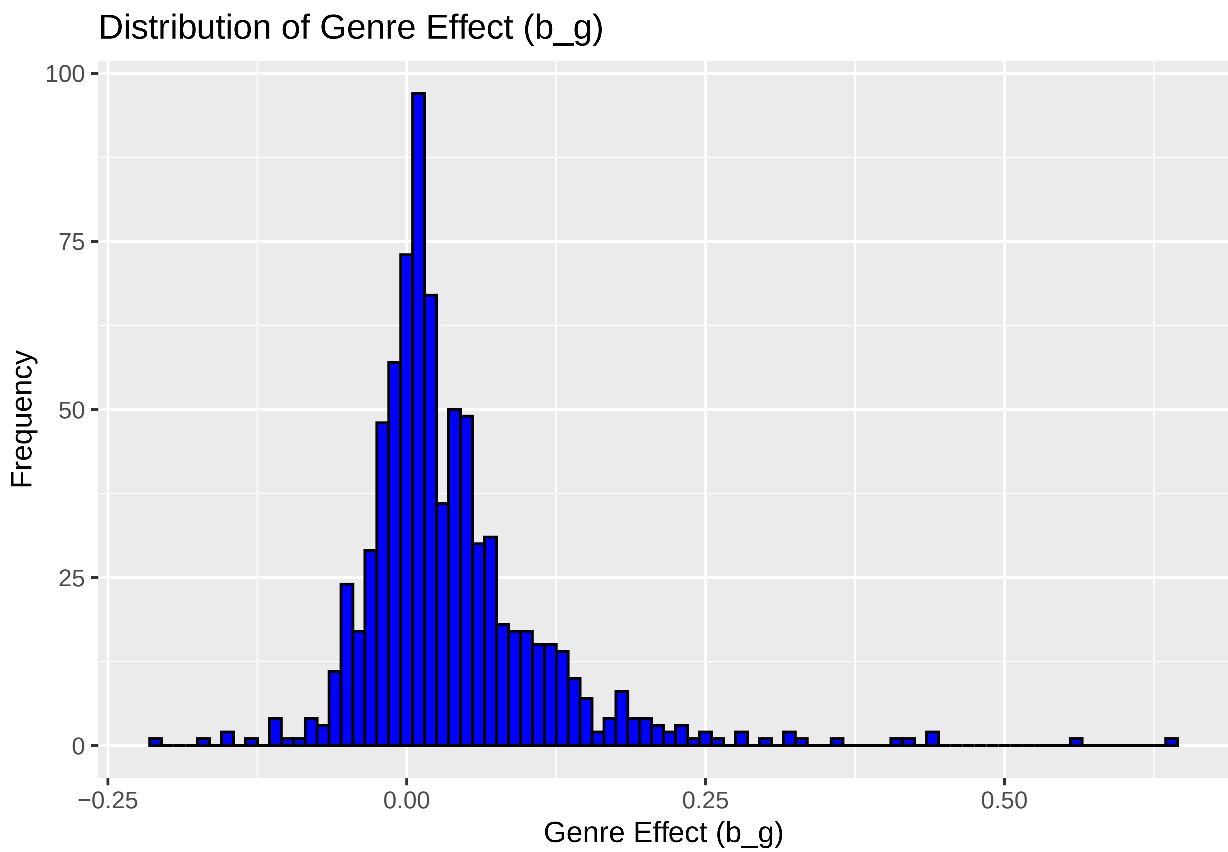
```
## # A tibble: 4 x 2
##   Method                      RMSE
##   <chr>                      <dbl>
## 1 Constant Value 2.5         1.47
## 2 Using the Data Set Mean    1.06
## 3 Movie Effect Model         0.943
## 4 Movie + User Effects Model 0.866
```

```r
# We export the result to an excel sheet:
write_xlsx(rmse_results, "rmse_results.xlsx")


# 4. Model 4: Genre Effect ----------------------------------------------------
# Step 1: Calculate the genre effect (b_g) for combined genres
genre_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))
```

```r
# Ensure the genre effect was calculated
head(genre_avgs)
```

```
## # A tibble: 6 x 2
##   genres                                                 b_g
##   <chr>                                                <dbl>
## 1 (no genres listed)                                  0.226
## 2 Action                                             -0.0367
## 3 Action|Adventure                                   -0.0123
## 4 Action|Adventure|Animation|Children|Comedy          0.0118
## 5 Action|Adventure|Animation|Children|Comedy|Fantasy -0.0109
## 6 Action|Adventure|Animation|Children|Comedy|IMAX     0.0141
```

```r
# Checking the genre effect using histogram
genre_avgs %>% ggplot(aes(b_g)) +
  geom_histogram(binwidth = 0.01, fill = "blue", color = "black") +
  labs(title = "Distribution of Genre Effect (b_g)", x = "Genre Effect (b_g)", y = "Frequency")
```

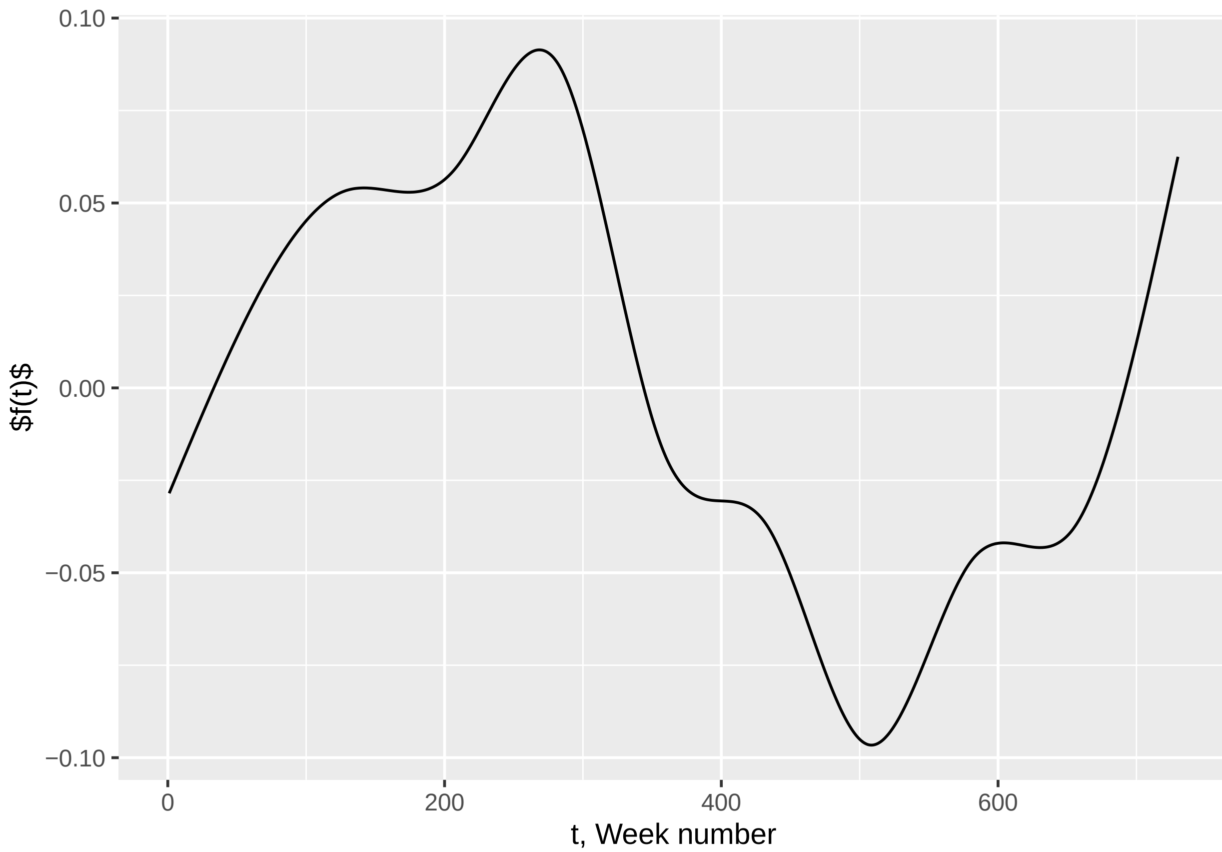



```r
# Step 2: Merge genre effect into test set
# Note: We need to separate multiple genres in the test set too
test_set_with_bi_bu_bg <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres")
```

```r
# Step 3: Predict ratings in the test set using movie, user, time, and genre effects
predicted_ratings <- test_set_with_bi_bu_bg %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

# Step 4: Evaluate the RMSE for Model 5 (Movie + User + Genre Effects Model)
model4_rmse <- RMSE(predicted_ratings, test_set$rating)

# Step 5: Append the Model 5 RMSE result to our collection
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Movie + User + Genre Effects",
                                     RMSE = model4_rmse))

rmse_results
```

```
## # A tibble: 5 x 2
##   Method                        RMSE
##   <chr>                        <dbl>
## 1 Constant Value 2.5           1.47
## 2 Using the Data Set Mean      1.06
## 3 Movie Effect Model           0.943
## 4 Movie + User Effects Model   0.866
## 5 Movie + User + Genre Effects 0.865
```

```r
# Step 6: Export the result to an Excel file
write_xlsx(rmse_results, "rmse_results.xlsx")


# 5. Model 5 - Predicting Using the User + Movie + Genre + Time Effect ----------

# 5. Model 5.1: Using GAM -----------------------------------------------------
# Step 1: Add a week number to each rating in the train and test sets
train_set <- train_set %>%
  mutate(weekNum = (timestamp - min(timestamp)) %/% (7 * 24 * 60 * 60) + 1)

test_set <- test_set %>%
  mutate(weekNum = (timestamp - min(timestamp)) %/% (7 * 24 * 60 * 60) + 1)

# Step 2: Fit a smooth curve to the ratings as a function of time (weekNum)
fit <- mgcv::gam(rating ~ s(weekNum, bs = "cs"),
                 family = gaussian(), data = train_set) # apply smoothing

# Step 3: Evaluate the fitted curve for each week number
r <- seq(1, max(train_set$weekNum))
f_t <- mgcv::predict.gam(fit, data.frame(weekNum = r)) - mu
rm(fit)

# Step 4: Plot the fitted curve
ggplot(data.frame(weekNum = r, f_t), aes(weekNum, f_t)) +
  geom_line() +
  xlab('t, Week number') +
  ylab((r'[$f(t)$]'))
```

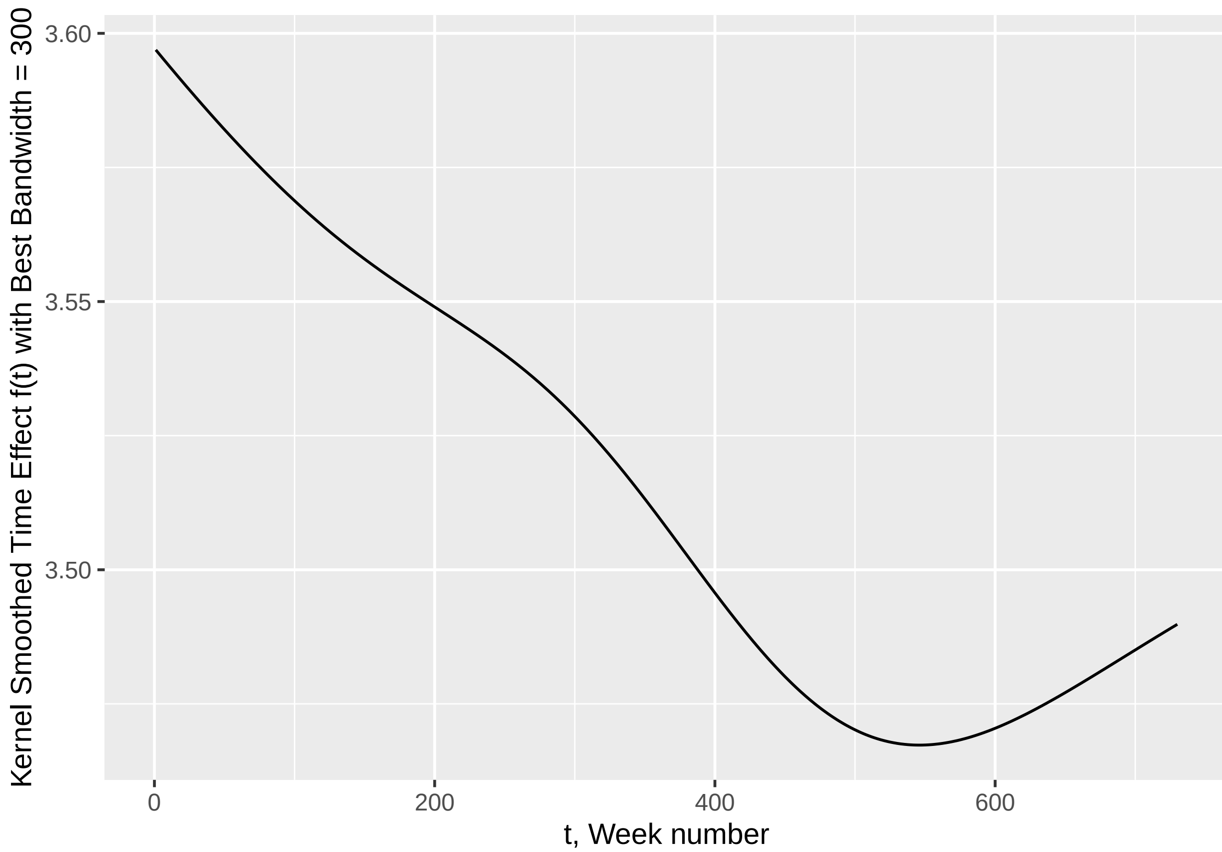



```r
# Step 5: Recalculating the effects
# 5.1: Movie Effect
movie_effect_t <- train_set %>%
  mutate(f_t = f_t[weekNum]) %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu - f_t))

# 5.2: User Effect
user_effect_t <- train_set %>%
  left_join(movie_effect_t, by = "movieId") %>%
  mutate(f_t = f_t[weekNum]) %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i - f_t))

head(user_effect_t)
```

```
## # A tibble: 6 x 2
##   userId     b_u
##    <int>   <dbl>
## 1      1  1.65
## 2      2 -0.237
## 3      3  0.398
## 4      4  0.739
## 5      5  0.0194
## 6      6  0.342
```

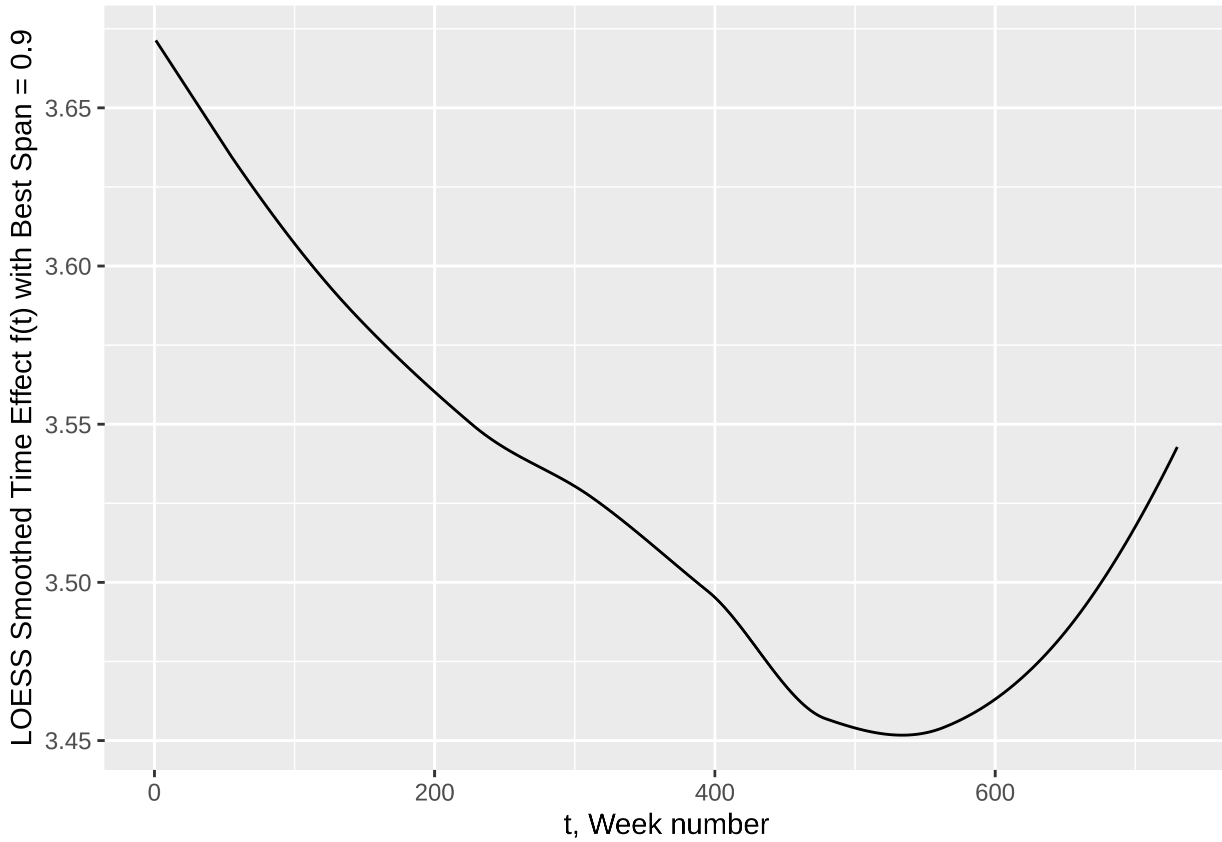
```r
# 5.3: Genre Effect
genre_effect_t <- train_set %>%
  left_join(movie_effect_t, by = "movieId") %>%
  left_join(user_effect_t, by = "userId") %>%
  mutate(f_t = f_t[weekNum]) %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - f_t))

# Step 6: Obtain predictions for the test set using the new model
predicted_ratings <- test_set %>%
  mutate(f_t = f_t[weekNum]) %>%
  left_join(movie_effect_t, by = "movieId") %>%
  left_join(user_effect_t, by = "userId") %>%
  left_join(genre_effect_t, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g + f_t) %>%
  pull(pred)

# Step 7: Compute RMSE for the new model
model_rmse <- RMSE(predicted_ratings, test_set$rating)

# Step 8: Append the RMSE result to the collection of RMSEs
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Movie + User + Genre + Time Effects (GAM)",
                                     RMSE = model_rmse))

# Step 9: Export the updated RMSE results to an Excel file
write_xlsx(rmse_results, "rmse_results.xlsx")


# 5. Model 5.2: Using KSmooth Bandwidth ----------------------------------------
# Step 1: Add a week number to each rating in the train and test sets
train_set_byweek <- train_set %>%
  mutate(weekNum = (timestamp - min(timestamp)) %/% (7 * 24 * 60 * 60) + 1) %>%
  group_by(weekNum) %>%
  summarize(y = mean(rating)) %>%
  arrange(weekNum)

test_set <- test_set %>%
  mutate(weekNum = (timestamp - min(timestamp)) %/% (7 * 24 * 60 * 60) + 1)

# Step 2: Iterate over possible bandwidth values for kernel smoothing
bandwidth_values <- seq(50, 500, by = 50)  # Adjust range and increment as necessary
best_rmse <- Inf
best_bandwidth <- NA

for (bandwidth in bandwidth_values) {
  # Apply kernel smoothing with the current bandwidth
  smoothed_train <- ksmooth(x = train_set_byweek$weekNum, y = train_set_byweek$y, kernel = "normal", bar

  # Generate smoothed time effect for the train and test sets
  smoothed_train_fitted <- approx(smoothed_train$x, smoothed_train$y, xout = train_set$weekNum)$y
  smoothed_test_fitted <- approx(smoothed_train$x, smoothed_train$y, xout = test_set$weekNum)$y
```

```r
  # Calculate movie, user, and genre effects using the current time effect
  movie_effect_t <- train_set %>%
    mutate(f_t = smoothed_train_fitted) %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu - f_t))

  user_effect_t <- train_set %>%
    left_join(movie_effect_t, by = "movieId") %>%
    mutate(f_t = smoothed_train_fitted) %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu - b_i - f_t))

  genre_effect_t <- train_set %>%
    left_join(movie_effect_t, by = "movieId") %>%
    left_join(user_effect_t, by = "userId") %>%
    mutate(f_t = smoothed_train_fitted) %>%
    group_by(genres) %>%
    summarize(b_g = mean(rating - mu - b_i - b_u - f_t))

  # Obtain predictions for the test set
  predicted_ratings <- test_set %>%
    mutate(f_t = smoothed_test_fitted) %>%
    left_join(movie_effect_t, by = "movieId") %>%
    left_join(user_effect_t, by = "userId") %>%
    left_join(genre_effect_t, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g + f_t) %>%
    pull(pred)

  # Calculate RMSE for the current bandwidth
  model_rmse <- RMSE(predicted_ratings, test_set$rating)

  # Check if this bandwidth yields a lower RMSE
  if (model_rmse < best_rmse) {
    best_rmse <- model_rmse
    best_bandwidth <- bandwidth
  }
}

# Step 3: Store the best result
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = paste("Movie + User + Genre + KSmooth Time Effects (box ba
                                     RMSE = best_rmse))

# Step 4: Export the updated RMSE results to an Excel file
write_xlsx(rmse_results, "rmse_results.xlsx")

# Step 5: Plot the best kernel-smoothed curve using the optimal bandwidth
smoothed_train_best <- ksmooth(x = train_set_byweek$weekNum, y = train_set_byweek$y, kernel = "normal",
ggplot(data.frame(weekNum = smoothed_train_best$x, f_t = smoothed_train_best$y),
       aes(weekNum, f_t)) +
  geom_line() +
  xlab('t, Week number') +
  ylab(paste('Kernel Smoothed Time Effect f(t) with Best Bandwidth =', best_bandwidth))
```




```r
# 5. Model 5.3: Time Effect Using LOESS ------------------------------------------
# Step 1: Add a week number to each rating in the train and test sets
train_set_byweek <- train_set %>%
  mutate(weekNum = (timestamp - min(timestamp)) %/% (7 * 24 * 60 * 60) + 1) %>%
  group_by(weekNum) %>%
  summarize(y = mean(rating)) %>%
  arrange(weekNum)

# Step 2: Iterate over possible span values for LOESS
span_values <- seq(0.1, 0.9, by = 0.1)  # Adjust range and increment if necessary
best_rmse <- Inf
best_span <- NA

for (span in span_values) {
  # Apply LOESS with the current span value
  loess_fit <- loess(y ~ weekNum, data = train_set_byweek, span = span)

  # Generate smoothed time effect for train and test sets
  smoothed_train_fitted <- predict(loess_fit, newdata = data.frame(weekNum = train_set$weekNum))
  smoothed_test_fitted <- predict(loess_fit, newdata = data.frame(weekNum = test_set$weekNum))

  # Calculate movie, user, and genre effects using the current time effect
  movie_effect_t <- train_set %>%
    mutate(f_t = smoothed_train_fitted) %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu - f_t))
```

```r
  user_effect_t <- train_set %>%
    left_join(movie_effect_t, by = "movieId") %>%
    mutate(f_t = smoothed_train_fitted) %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu - b_i - f_t))

  genre_effect_t <- train_set %>%
    left_join(movie_effect_t, by = "movieId") %>%
    left_join(user_effect_t, by = "userId") %>%
    mutate(f_t = smoothed_train_fitted) %>%
    group_by(genres) %>%
    summarize(b_g = mean(rating - mu - b_i - b_u - f_t))

  # Obtain predictions for the test set
  predicted_ratings <- test_set %>%
    mutate(f_t = smoothed_test_fitted) %>%
    left_join(movie_effect_t, by = "movieId") %>%
    left_join(user_effect_t, by = "userId") %>%
    left_join(genre_effect_t, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g + f_t) %>%
    pull(pred)

  # Calculate RMSE for the current span
  model_rmse <- RMSE(predicted_ratings, test_set$rating)

  # Check if this span yields a lower RMSE
  if (model_rmse < best_rmse) {
    best_rmse <- model_rmse
    best_span <- span
  }
}

# Step 3: Store the best result
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = paste("Movie + User + Genre + LOESS Time Effects (span ="
                                     RMSE = best_rmse))

# Step 4: Export the updated RMSE results to an Excel file
write_xlsx(rmse_results, "rmse_results.xlsx")

# Step 5: Plot the best LOESS-smoothed curve using the optimal span
loess_fit_best <- loess(y ~ weekNum, data = train_set_byweek, span = best_span)
ggplot(data.frame(weekNum = train_set_byweek$weekNum, f_t = predict(loess_fit_best, train_set_byweek)),
       aes(weekNum, f_t)) +
  geom_line() +
  xlab('t, Week number') +
  ylab(paste('LOESS Smoothed Time Effect f(t) with Best Span =', best_span))
```

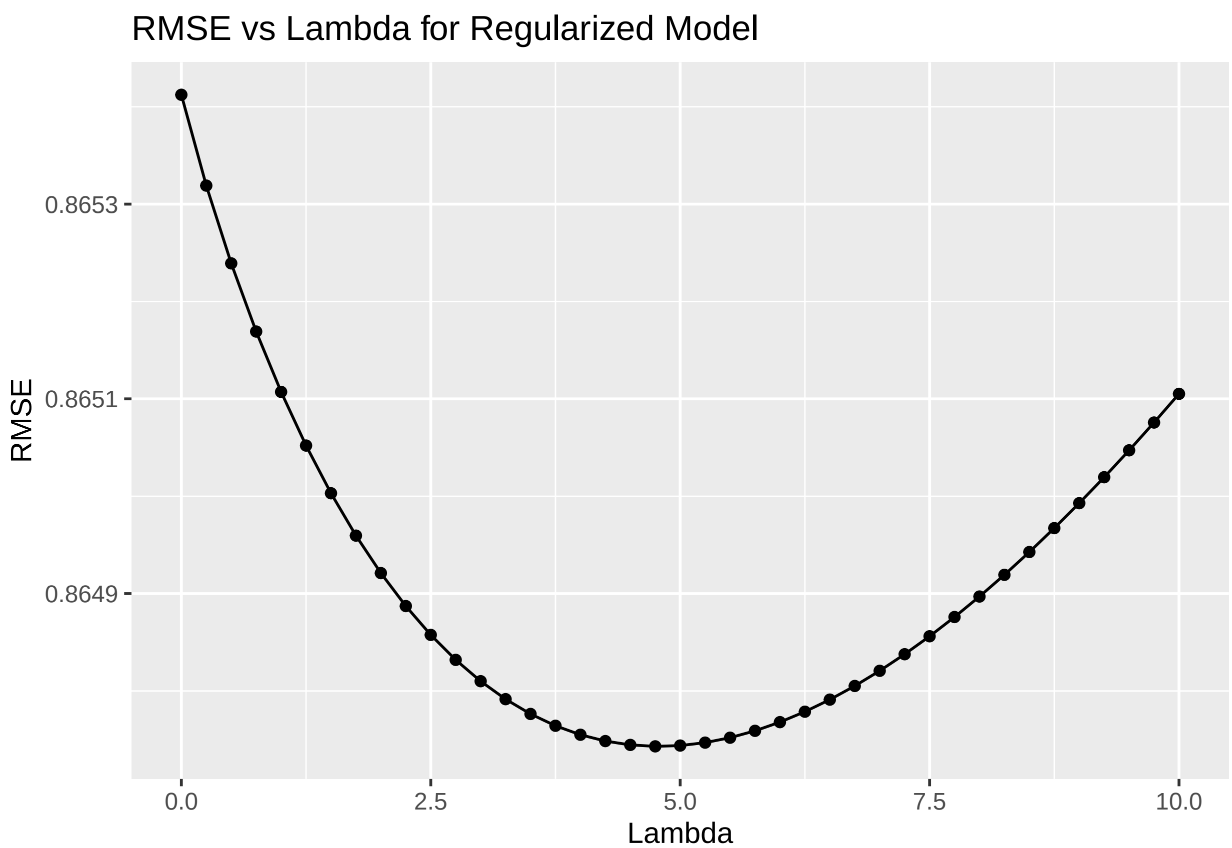



```r
# 6. Model 6: Regularized Model ---------------------------------------------------

# 6. Model 6.1: Regularized Model Without Time Effect -----------------------------
# Step 1: Define lambda values for regularization
lambdas <- seq(0, 10, 0.25)

# Step 2: Apply regularization to the movie, user, and genre effects
regularization_rmse_results <- data.frame()

for (lambda in lambdas) {

  # Movie Effect with Regularization
  movie_avgs_reg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + lambda)) # Regularized

  # User Effect with Regularization
  user_avgs_reg <- train_set %>%
    left_join(movie_avgs_reg, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i) / (n() + lambda)) # Regularized

  # Genre Effect with Regularization
  genre_avgs_reg <- train_set %>%
    left_join(movie_avgs_reg, by = "movieId") %>%
    left_join(user_avgs_reg, by = "userId") %>%
```

```r
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + lambda)) # Regularized

  # Step 3: Predict ratings on the test set using the regularized movie, user, and genre effects
  predicted_ratings <- test_set %>%
    left_join(movie_avgs_reg, by = "movieId") %>%
    left_join(user_avgs_reg, by = "userId") %>%
    left_join(genre_avgs_reg, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)

  # Step 4: Calculate RMSE for each lambda
  model_rmse <- RMSE(predicted_ratings, test_set$rating)

  # Step 5: Store the RMSE results for each lambda in a separate data frame for regularization
  regularization_rmse_results <- bind_rows(regularization_rmse_results,
                                           data_frame(Method = paste("Regularized Model (lambda =", lam
                                                      RMSE = model_rmse))
}

# Step 6: Find the best lambda based on the lowest RMSE in regularization_rmse_results
best_lambda <- lambdas[which.min(regularization_rmse_results$RMSE)]
best_rmse <- min(regularization_rmse_results$RMSE)

# Step 7: Append the best result of the regularized model to the overall rmse_results (that contains pr
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = paste("Regularized Model with Movie, User, Genre Effects
                                     RMSE = best_rmse))

# Step 8: Export the updated RMSE results to an Excel file
write_xlsx(rmse_results, "rmse_results.xlsx")

# Step 9: Print the best lambda and the corresponding RMSE
print(paste("Optimal lambda:", best_lambda, "with RMSE:", best_rmse))
```

```
## [1] "Optimal lambda: 4.75 with RMSE: 0.864743125233994"
```

```r
# Optional: Plot RMSE vs Lambda for Regularized Model
ggplot(regularization_rmse_results, aes(x = lambdas, y = RMSE)) +
  geom_point() +
  geom_line() +
  xlab('Lambda') +
  ylab('RMSE') +
  ggtitle('RMSE vs Lambda for Regularized Model')
```

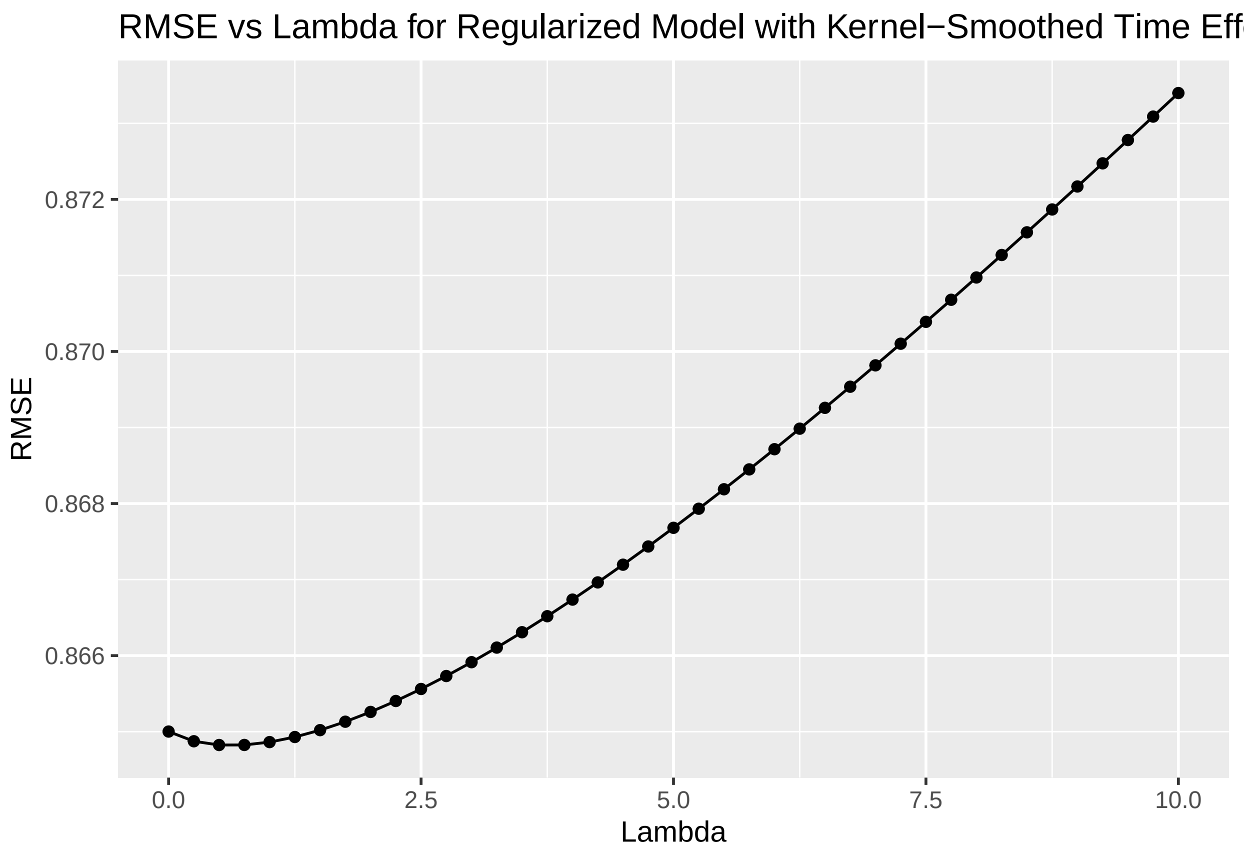



```
# 6. Model 6.2: Regularized Model with Kernel-Smoothed Time Effect ----------------
# Step 1: Define lambda values for regularization
lambdas <- seq(0, 10, 0.25)

# Step 2: Calculate the time effect using kernel smoothing (ksmooth)
train_set_byweek <- train_set %>%
  mutate(weekNum = (timestamp - min(timestamp)) %/% (7 * 24 * 60 * 60) + 1) %>%
  group_by(weekNum) %>%
  summarize(y = mean(rating)) %>%
  arrange(weekNum)

# Apply kernel smoothing on the train set
smoothed_train <- ksmooth(x = train_set_byweek$weekNum, y = train_set_byweek$y, kernel = "normal", bandw
# we use 300 as bandwidth since it was found as the best bandwidth for our data set at the previous mod

# Extract fitted values for each week in the train set
smoothed_train_fitted <- approx(smoothed_train$x, smoothed_train$y, xout = train_set$weekNum)$y

# Step 3: Calculate the movie, user, and genre effects with regularization, incorporating the time effe
regularization_rmse_results <- data.frame()

for (lambda in lambdas) {

  # Movie Effect with Regularization and Time Effect
  movie_effect_t <- train_set %>%
    mutate(f_t = smoothed_train_fitted) %>%
```

```
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu - f_t) / (n() + lambda)) # Regularized

  # User Effect with Regularization and Time Effect
  user_effect_t <- train_set %>%
    left_join(movie_effect_t, by = "movieId") %>%
    mutate(f_t = smoothed_train_fitted) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i - f_t) / (n() + lambda)) # Regularized

  # Genre Effect with Regularization and Time Effect
  genre_effect_t <- train_set %>%
    left_join(movie_effect_t, by = "movieId") %>%
    left_join(user_effect_t, by = "userId") %>%
    mutate(f_t = smoothed_train_fitted) %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u - f_t) / (n() + lambda)) # Regularized

  # Step 4: Use kernel smoothing to extract the smoothed time effect for the test set
  smoothed_test_fitted <- approx(smoothed_train$x, smoothed_train$y, xout = test_set$weekNum)$y

  # Step 5: Obtain predictions for the test set using regularized effects and time effect
  predicted_ratings <- test_set %>%
    mutate(f_t = smoothed_test_fitted) %>%
    left_join(movie_effect_t, by = "movieId") %>%
    left_join(user_effect_t, by = "userId") %>%
    left_join(genre_effect_t, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g + f_t) %>%
    pull(pred)

  # Step 6: Compute RMSE for each lambda
  model_rmse <- RMSE(predicted_ratings, test_set$rating)

  # Step 7: Store RMSE for each lambda
  regularization_rmse_results <- bind_rows(regularization_rmse_results,
                                          data_frame(Method = paste("Regularized Model (lambda =", lam
                                                     RMSE = model_rmse))
}

# Step 8: Find the best lambda based on the lowest RMSE
best_lambda <- lambdas[which.min(regularization_rmse_results$RMSE)]
best_rmse <- min(regularization_rmse_results$RMSE)

# Step 9: Append the best regularized model result with time effect to the overall RMSE results
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = paste("Regularized Model with Kernel-Smoothed Time Effect
                                     RMSE = best_rmse))

# Step 10: Export the updated RMSE results to an Excel file
write_xlsx(rmse_results, "rmse_results.xlsx")

# Step 11: Print the best lambda and the corresponding RMSE
print(paste("Optimal lambda:", best_lambda, "with RMSE:", best_rmse))
```

```
## [1] "Optimal lambda: 0.5 with RMSE: 0.864824037232209"
```

```r
# Plot RMSE vs Lambda for Regularized Model with Kernel-Smoothed Time Effect
ggplot(regularization_rmse_results, aes(x = lambdas, y = RMSE)) +
  geom_point() +
  geom_line() +
  xlab('Lambda') +
  ylab('RMSE') +
  ggtitle('RMSE vs Lambda for Regularized Model with Kernel-Smoothed Time Effect')
```




```r
# 7. Model 7: Matrix Factorization with Singular Value Decomposition ------------
# Load necessary libraries
library(Rcpp)
```

```
## Warning:   'Rcpp' R 4.3.2
```

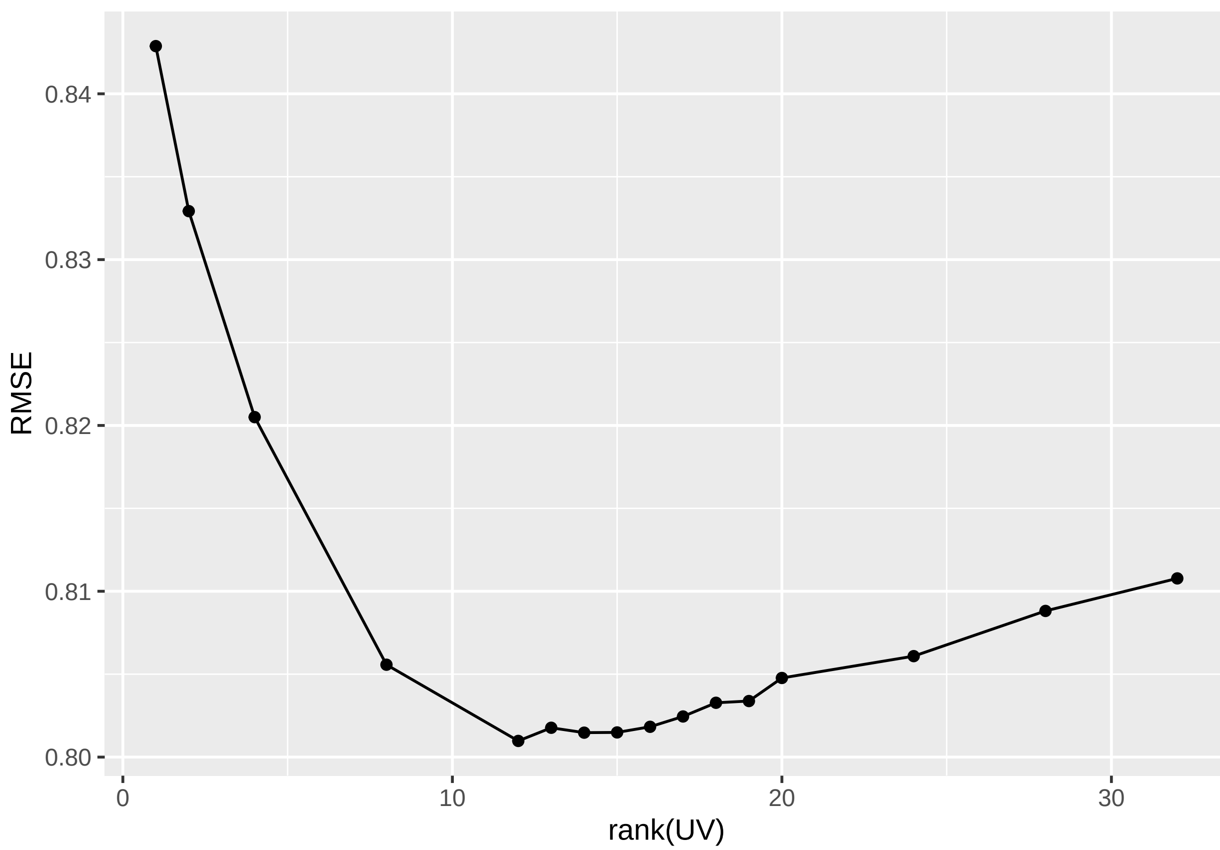
```r
library(RcppArmadillo)
```

```
## Warning:   'RcppArmadillo' R 4.3.3
```

```r
library(dplyr)
library(purrr)
library(ggplot2)
```

```r
# Load the C++ file
Rcpp::sourceCpp("C:/Users/HP/Downloads/Harvard DS/svd.cpp")

# Step 1: Prepare residuals from the previous best model
mu <- mean(train_set$rating)  # Assume `mu` is the global mean rating

previous_train <- train_set %>%
  left_join(movie_avgs_reg, by = "movieId") %>%
  left_join(user_avgs_reg, by = "userId") %>%
  left_join(genre_avgs_reg, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

residuals_train <- as.numeric(train_set$rating - previous_train)

# Test set predictions for the previous model
previous_test <- test_set %>%
  left_join(movie_avgs_reg, by = "movieId") %>%
  left_join(user_avgs_reg, by = "userId") %>%
  left_join(genre_avgs_reg, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

# Create unique indices for userId without gaps
Uidx <- numeric(max(train_set$userId))
Uidx[unique(train_set$userId)] <- seq_along(unique(train_set$userId))

# Create unique indices for movieId without gaps
Vidx <- numeric(max(train_set$movieId))
Vidx[unique(train_set$movieId)] <- seq_along(unique(train_set$movieId))

# Step 2: Define matrix factorization function with parameters
funk <- function(Uidx, Vidx, residuals, nFeatures, steps = 500,
                 regCoef = 0.02, learningRate = 1e-3) {
  funkCpp(Uidx[train_set$userId] - 1,  # Convert to 0-based indexing
          Vidx[train_set$movieId] - 1,
          residuals,
          nFeatures, steps, regCoef, learningRate)
}

# Step 3: Tune latent features
set.seed(1)
if(!file.exists('funk_tuning.Rdata')) {
  nFeatures <- c(1, 2, 4, 8, seq(12, 20), 24, 28, 32)
  rmses <- sapply(nFeatures, function(nF) {
    message(nF, ' features')

    # Run matrix factorization with the current feature count
    set.seed(1)
    funkResult <- funk(Uidx, Vidx, residuals_train, nFeatures = nF, steps = 500)
    U <- funkResult$U
    V <- funkResult$V
```

```r
    # Predict ratings for the test set
    predicted_ratings_funk <- test_set %>%
      mutate(pred = previous_test +
               map2_dbl(userId, movieId, function(u, v) U[Uidx[u], ] %*% V[Vidx[v], ])) %>%
      pull(pred)

    # Calculate RMSE for current number of features
    rmse <- RMSE(predicted_ratings_funk, test_set$rating)
    message(rmse, '\n')
    return(rmse)
  })

  save(nFeatures, rmses, file = 'funk_tuning.Rdata')
}

# Load RMSE results if file exists
if (file.exists('funk_tuning.Rdata')) {
  load('funk_tuning.Rdata')
}

# Plot RMSE against the number of MF features
par(cex = 0.7)
qplot(nFeatures, rmses, xlab = 'rank(UV)', ylab = 'RMSE', geom = c('point','line'))
```



```r
# Find the optimal number of features
optimal_nfeatures <- nFeatures[which.min(rmses)]
optimal_nfeatures
```

```
## [1] 12
```

```r
# Run matrix factorization with the optimal number of features
set.seed(1)
if (!file.exists('funk.Rdata')) {
  funkResult <- funk(Uidx, Vidx, residuals_train, nFeatures = optimal_nfeatures, steps = 500)
  save(optimal_nfeatures, funkResult, file = 'funk.Rdata')
}

# Load matrix factorization data from file
load('funk.Rdata')

U <- funkResult$U
V <- funkResult$V

# Predict ratings for the test set
predicted_ratings_funk <- test_set %>%
  mutate(pred = previous_test +
           map2_dbl(userId, movieId, function(u, v) U[Uidx[u], ] %*% V[Vidx[v], ])) %>%
  pull(pred)

# Compute and print RMSE
mf_rmse <- RMSE(predicted_ratings_funk, test_set$rating)
mf_rmse
```

```
## [1] 0.8009753
```

```r
# Step 9: Append the best regularized model result with time effect to the overall RMSE results
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = paste("Matrix Factorization (features =", optimal_nfeatur
                                     RMSE = mf_rmse))

# Step 10: Export the updated RMSE results to an Excel file
write_xlsx(rmse_results, "rmse_results.xlsx")





# F. FINAL TESTING USING FINAL HOLDOUT TEST
# We will not test the efficacy of our model on the final_holdout_test set.
# The final or the best model we use will be our model 7 - matrix factorization
# which is based on the regularized model that takes into account movie, user, and genre effects (model

# Rechecking the final_holdout_test
head(final_holdout_test)
```

```
##   userId movieId rating timestamp
## 1      1     231      5 838983392
```

```
## 2      1     480      5 838983653
## 3      1     586      5 838984068
## 4      2     151      3 868246450
## 5      2     858      2 868245645
## 6      2    1544      3 868245920
##                                                      title
## 1                                     Dumb & Dumber (1994)
## 2                                     Jurassic Park (1993)
## 3                                        Home Alone (1990)
## 4                                           Rob Roy (1995)
## 5                                    Godfather, The (1972)
## 6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                   genres
## 1                                 Comedy
## 2        Action|Adventure|Sci-Fi|Thriller
## 3                        Children|Comedy
## 4                Action|Drama|Romance|War
## 5                            Crime|Drama
## 6 Action|Adventure|Horror|Sci-Fi|Thriller
```

```r
# Note that some users and movieid in the final_holdout_test would not be present in the our model.
# This is because after creating the final_holdout_test, even though we made sure that it has the same
# we still partitioned the edx data set into train and test set, making some users or movies absent in
# Since we cannot alter the final_holdout_test, we then will just assign 0 to the b_i, b_u, and b_g whe

# Generate final predictions for the FINAL HOLDOUT TEST dataset
predicted_ratings_fht <- final_holdout_test %>%
  left_join(movie_avgs_reg, by = "movieId") %>%
  left_join(user_avgs_reg, by = "userId") %>%
  left_join(genre_avgs_reg, by = "genres") %>%
  mutate(
    # Replace any NA values in b_i, b_u, and b_g with 0
    b_i = ifelse(is.na(b_i), 0, b_i),
    b_u = ifelse(is.na(b_u), 0, b_u),
    b_g = ifelse(is.na(b_g), 0, b_g),
    pred = mu + b_i + b_u + b_g +
      map2_dbl(userId, movieId, function(u, v) {
        # Check if userId and movieId have valid indices in Uidx and Vidx
        if (!is.na(Uidx[u]) && Uidx[u] > 0 && !is.na(Vidx[v]) && Vidx[v] > 0) {
          U[Uidx[u], ] %*% V[Vidx[v], ]
        } else {
          0  # Return 0 if index is missing
        }
      })
  ) %>%
  pull(pred)

# Compute RMSE
fht_rmse <- RMSE(predicted_ratings_fht, final_holdout_test$rating)
fht_rmse
```

```
## [1] 0.8006021
```

```
# Append the final holdout test RMSE to the overall RMSE results
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = paste("Final Holdout Test - Matrix Factorization with", op
                                     RMSE = fht_rmse))

# Export the updated RMSE results to an Excel file
write_xlsx(rmse_results, "rmse_results.xlsx")
```