PH125.9x_MovieLens_Capstone

Ashley Reinhart

2024-03-21

Introduction

This project is to meet the requirements for the HarvardX PH125.9 MovieLens Captstone.

The aim of this project is to create a movie recommendation system via machine learning utilizing the MovieLens data set. The entire MovieLens data set currently contains 33,000,000 ratings applied to 86,000 movies and 330,975 users. For this project, the smaller 10M version MovieLens data set will be utilized to make computation smoother.

The 10M version will be partitioned into an EDX data set, which will be utilized to build and train the models, and a final_holdout_test data set that will be utilized to evaluate final accuracy of the chosen model.

Accuracy will be evaluated utilizing the Root Mean Square Error (RMSE). The RMSE measures the the difference between the predicted values and actual values. The lower the RMSE, the better. The goal of this project is to identify a model that returns a RMSE on the final_hold_out data set of less than 0.86499, representing a reasonably accurate model.

Loading the Data

This script was provided by the course to load the original data sets.

```
# Note: this process could take a couple of minutes
# STEP 1:
if (!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if (!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(dplyr)
library(caret)
library(tidyr)
library(stringr)
library(tinytex)
library(kableExtra)

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

options(timeout = 120)

dl <- "ml-10M100K.zip"
```

```r
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
)
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))

movielens <- left_join(ratings, movies, by = "movieId")

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index, ]
temp <- movielens[test_index, ]

# Make sure userId and movieId in 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)
edx <- rbind(edx, removed)

# To save memory, which my computer has little of
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Methods and Analysis

Examining the EDX Data Set

```
dim(edx)
```

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

```
head(edx,8)
```

```
##   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)
## 8      1     356      5 838983653           Forrest Gump (1994)
## 9      1     362      5 838984885       Jungle Book, 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
## 8      Comedy|Drama|Romance|War
## 9    Adventure|Children|Romance
```

We can see that the original EDX data set has 9000055 observations and and 6 variables. Additional cleaning will be performed for further analysis and to prep the data set for modeling. But first, let's dig into the basics:

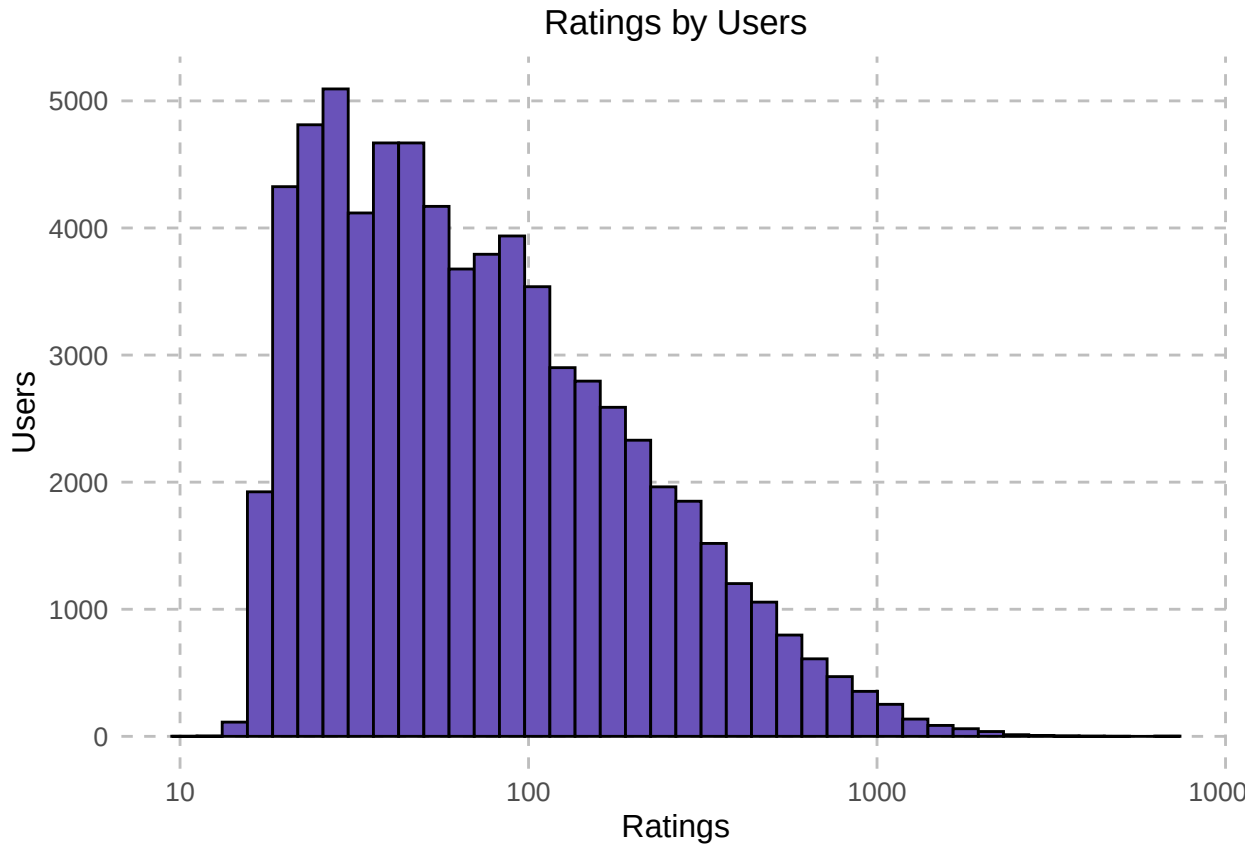
```
distinct_counts <- edx %>%
  summarise(
    distinct_userIds = n_distinct(userId),
    distinct_movieIds = n_distinct(movieId),
    distinct_combo_genre = n_distinct(genres)
  )
# Print the distinct counts
print(distinct_counts)
```

```
##   distinct_userIds distinct_movieIds distinct_combo_genre
## 1            69878             10677                 797
```

We learn that there are 69878 distinct users that have rated 10677 distinct movies. In total there are 797 distinct combinations of genres.

User Data Let's visualize the user data.

```
edx %>%
  group_by(userId) %>% # grouping the data by userId
  summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color = "black", fill = "#6952b9", bins = 40) +
  scale_x_log10() +
  theme_minimal() + # set theme to minimal
  labs(x = "Ratings", y = "Users", title = "Ratings by Users") + # updating
  #label names
  theme(
    plot.title = element_text(hjust = 0.5), # center title
    axis.text = element_text(size = 10), # adjusting text size
    axis.title = element_text(size = 12), # adjusting the title
    panel.grid.major = element_line(color = "gray", linetype = "dashed"), # add
    #dashed lines
    panel.grid.minor = element_blank() # Removing minor grid lines
  )
```



We can see here that the numbers of ratings per user is skewed to the right. Most users provide more than 100 ratings.

Ratings Let's take a closer look at the ratings.

```
# populating rating options
length(unique(edx$rating))
```

```
## [1] 10
```

There are 10 unique rating options users can select.

```
# populating distinct rating options
unique_ratings <- unique(edx$rating)
sort(unique_ratings)
```

```
##  [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

The ratings increase in increments of .5. with the lowest being .5 and highest 5. Now lets look at the distribution of those ratings.

```
# summarizing ratings
edx %>%
  group_by(rating) %>%
  summarize(ratings_sum = n()) %>%
  arrange(desc(ratings_sum))
```

```
## # A tibble: 10 x 2
##    rating ratings_sum
##     <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
```

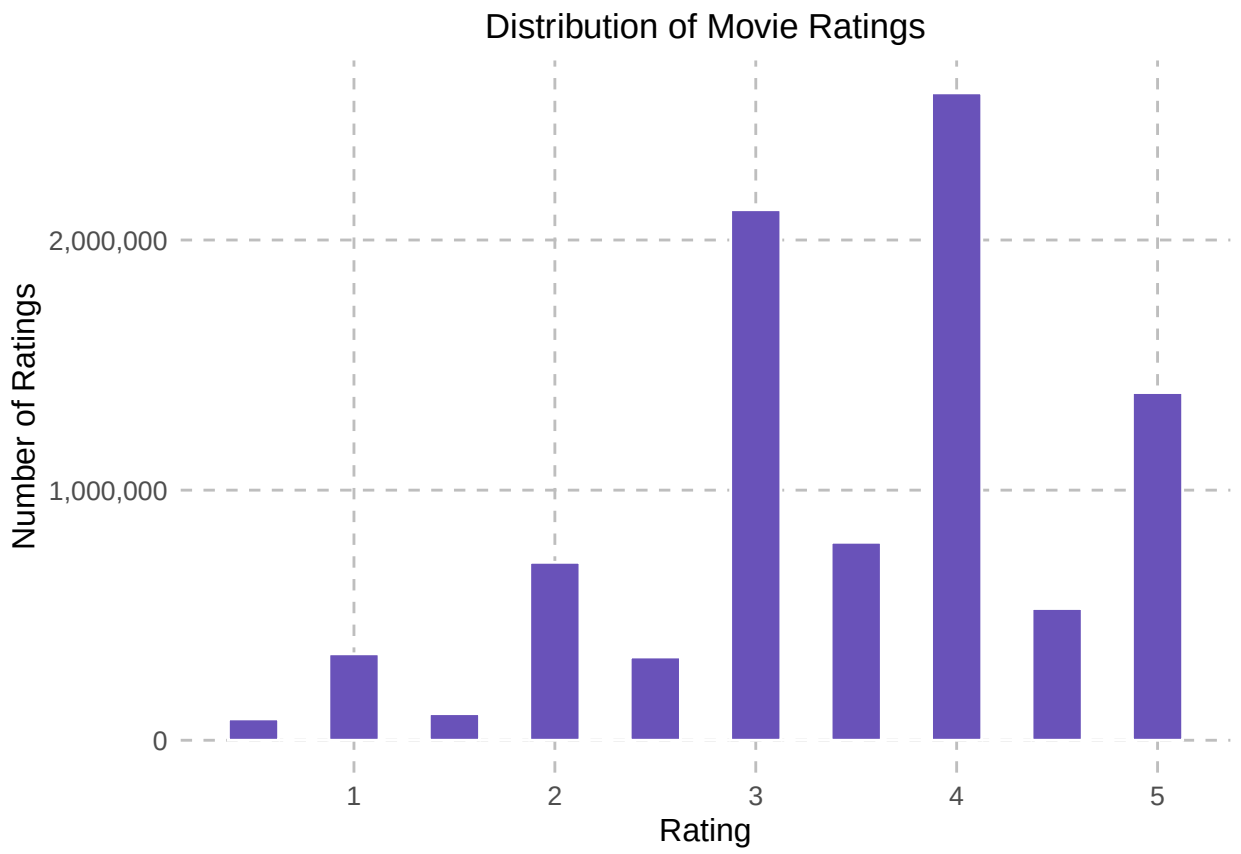
```
# % of ratings that are greater than or equal to a 3 star
ratingpercent <- edx %>% filter(edx$rating >= 3)
nrow(ratingpercent) / length(edx$rating)
```

```
## [1] 0.8242332
```

We can see that 82% of the ratings are >= 3 stars.

```
edx %>%
  ggplot(aes(x = rating)) + # set x axis as the rating
  geom_histogram(binwidth = 0.25, fill = "#6952b9", color = "white") + # add
  #histogram layers
  ggtitle("Distribution of Movie Ratings") + # plot title
  labs(x = "Rating", y = "Number of Ratings") + # set the x and y axis
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5), # align title
    axis.text = element_text(size = 10), # set text size for axis labs
```

```
    axis.title = element_text(size = 12), # set text size for axis titles
    panel.grid.major = element_line(color = "gray", linetype = "dashed"),#major grid lines
    panel.grid.minor = element_blank() # removes minor grid lines
  ) +
  scale_y_continuous(labels = scales::comma) # adjust y-axis scale to
```




```
# use commas for thousands separator
```

We can see that half-star ratings are less popular than whole star ratings.

Data Cleaning and Additional Analysis

The data is cleaned to examine review year, move age, and distinct genres.

A function was created so that it could be easily applied to the final_holdout_test data later on. This function extracts the release year from the title, changes the timestamp to the year reviewed, calculates the movie age, and separates the genres.

```
cleaning_data <- function(data, timestamp_column, title_column) {
  data <- data %>%
    # extracting release year from title
    mutate(release_year = str_extract({{ title_column }}, "\\((\\d{4})\\)")) %>%
    # remove parentheses
    #from release year
```

```
    mutate(release_year = as.integer(str_replace(release_year, "\\((\\d{4})\\)", "\\1"))) %>%
    # convert timestamp column to
    #year and rename it to "review_year"
    mutate(review_year = as.POSIXct({{ timestamp_column }}, origin = "1970-01-01", tz = "EST")) %>%
    mutate(review_year = as.integer(format(review_year, "%Y"))) %>%
    # remove the original timestamp column
    #and add a movie_age column
    select(-{{ timestamp_column }}, review_year) %>%
    mutate(movie_age = as.integer(2024 - release_year)) %>%
    separate_rows(genres, sep = "\\|")

  return(data)
}

edx <- cleaning_data(edx, timestamp, title) # applying function to edx data
```

Validating Function Executed Properly

Before we move on, let's check to ensure our function executed properly. First, we'll ensure release years are between 1900 and 2024:

```
# Filter and print records where release_year is greater than
#2024 or less than 1900 to validate function exceuted properly
edx %>%
  group_by(movieId, title, release_year) %>%
  filter(release_year > 2024 | release_year < 1900) %>%
  distinct(release_year)
```

```
## # A tibble: 0 x 3
## # Groups:   movieId, title, release_year [0]
## # i 3 variables: movieId <int>, title <chr>, release_year <int>
```

The release years populated correctly, as we do not have any outside the 1900-2024 range.

Next we'll examine a movie known to have a 4 digit integer in the title to confirm the release year column split correctly:

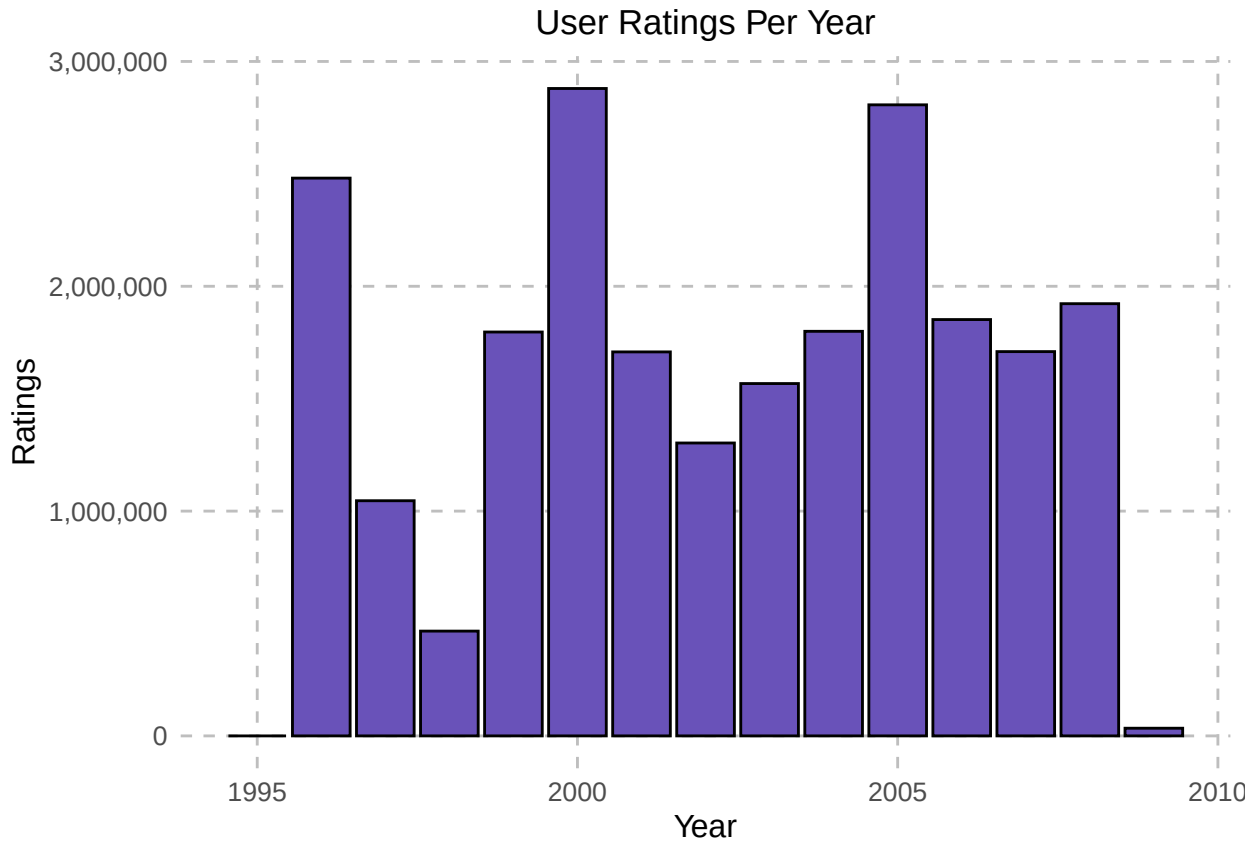
```
# validating that the function correctly extracted the
#release year from movies with 4 digit integers in the title
desired_movie_id <- 53953

edx %>% filter(movieId == desired_movie_id) %>%
 slice(1)
```

```
## # A tibble: 1 x 8
##   userId movieId rating title       genres release_year review_year movie_age
##    <int>   <int>  <dbl> <chr>       <chr>         <int>       <int>     <int>
## 1    435   53953    3.5 1408 (2007) Drama          2007        2007        17
```

The function executed properly! We can see here that the release year of 2007 was extracted instead of 1408. We can now move forward.

Movie Rating Year First, let's evaluate when movies were typically rated.

```
edx %>%
  ggplot(aes(x = review_year)) + # sett x axis to review year
  geom_bar(fill = "#6952b9", color = "black") + # setting to barplot
  ggtitle("User Ratings Per Year") + # plot title
  labs(x = "Year", y = "Ratings") + # set x and y axis
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5), # align title
    axis.text = element_text(size = 10), # sett text size
    axis.title = element_text(size = 12), # setti axis title size
    panel.grid.major = element_line(color = "gray", linetype = "dashed"),#major grid lines
    panel.grid.minor = element_blank() # Remove minor grid lines
  ) +
   # adjust y-axis scale to use commas for thousands separator
  scale_y_continuous(labels = scales::comma)
```

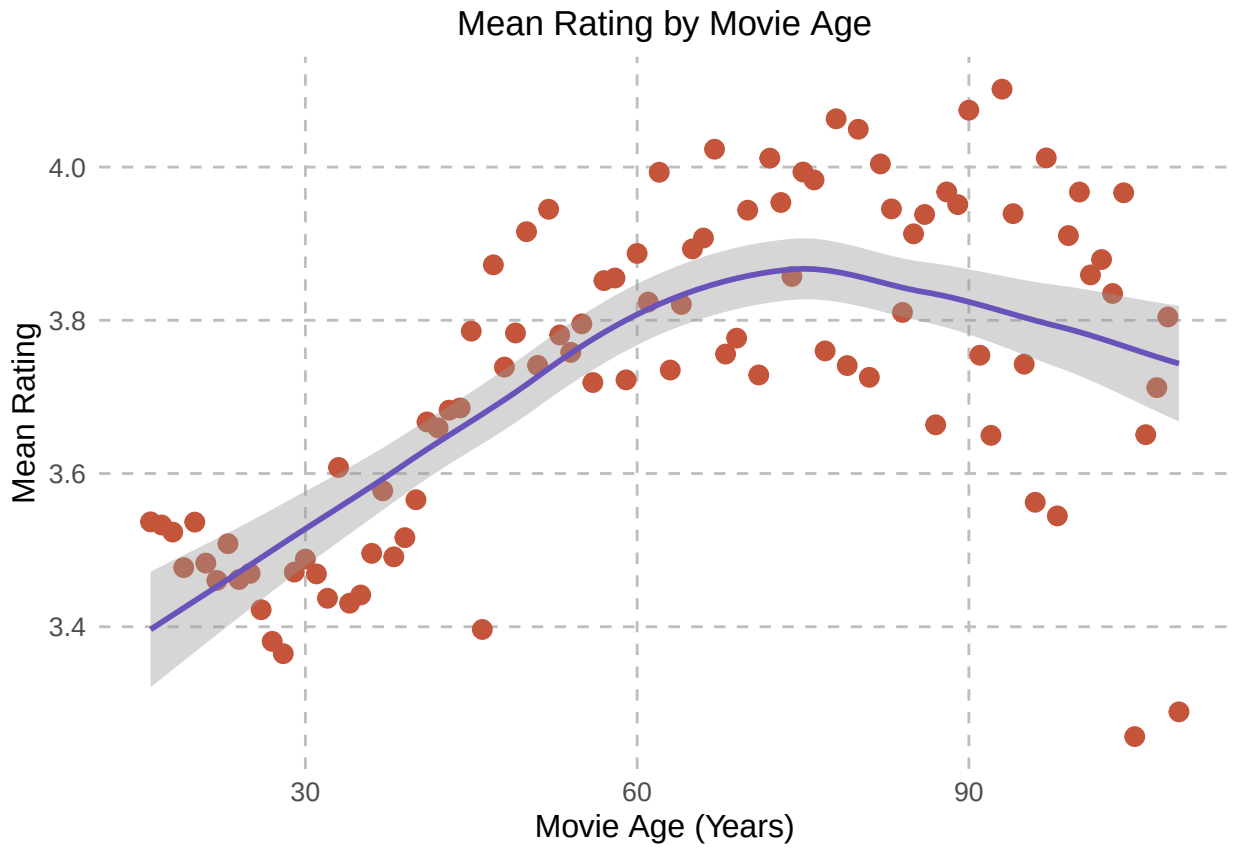


More movies were rated around 1996, 2000, and 2005.

Movie Age Now let's look at movie age vs rating.

```
# Calculate the mean rating for each movie_age group, ignoring NA values.
mean_rating_age <- edx %>%
  group_by(movie_age) %>%
  summarise(mean_rating = mean(rating, na.rm = TRUE))
```

```r
# plotting mean_rating_age
ggplot(mean_rating_age, aes(x = movie_age, y = mean_rating)) +
  geom_point(color = "#c4553a", size = 3) + # setting color and size of points
  geom_smooth(method = "loess", color = "#6952b9", method.args = list(span = 0.15, degree = 1)) +
  labs(x = "Movie Age (Years)", y = "Mean Rating", title = "Mean Rating by Movie Age") +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5), # settitle
    axis.text = element_text(size = 10), # set axis text size
    axis.title = element_text(size = 12), # set axis title size
    panel.grid.major = element_line(color = "gray", linetype = "dashed"), #add dashed grid lines
    panel.grid.minor = element_blank() # removing minor grid lines
  )
```



This plot looks at the relationship between the movie ages and their mean(average) rating.

Each dot represents a movie and the the line was created utilizing the LOESS (Locally Estimated Scatterplot Smoothing) method. This provides an approximate relationship between a movie's age and the mean rating.

Ultimately, this tells us that movie age may have an effect on the rating, as the older the movie becomes, the higher the rating tends to be.

Genre Prior to cleaning the data, we noted 797 distinct combinations of genres, but how many distinct genre categories are there and what is the genre with the highest number of movies?

```r
n_distinct(edx$genres) # counting distinct genres
```

```
## [1] 20
```

We observe that there are 20 distinct genres.

```r
# grouping movies by genre and arranging them in descending order
edx %>%
  group_by(genres) %>%
  summarise(number_movies_genre = n()) %>%
  arrange(desc(number_movies_genre))
```

```
## # A tibble: 20 x 2
##    genres             number_movies_genre
##    <chr>                            <int>
##  1 Drama                          3910127
##  2 Comedy                         3540930
##  3 Action                         2560545
##  4 Thriller                       2325899
##  5 Adventure                      1908892
##  6 Romance                        1712100
##  7 Sci-Fi                         1341183
##  8 Crime                          1327715
##  9 Fantasy                         925637
## 10 Children                        737994
## 11 Horror                          691485
## 12 Mystery                         568332
## 13 War                             511147
## 14 Animation                       467168
## 15 Musical                         433080
## 16 Western                         189394
## 17 Film-Noir                       118541
## 18 Documentary                      93066
## 19 IMAX                              8181
## 20 (no genres listed)                   7
```

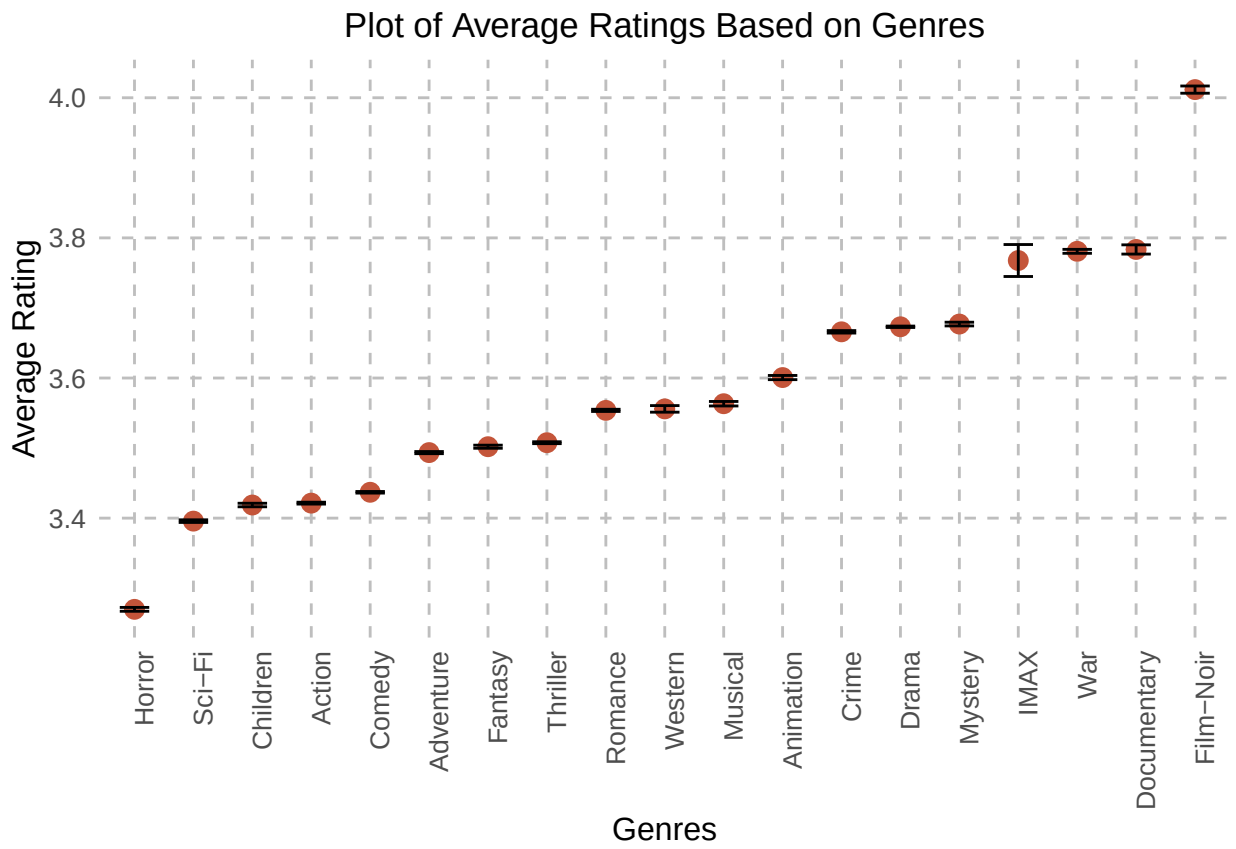
The Drama category has the highest number of movies. Does that mean that it has higher ratings on average? Let's look.

```r
# calculating mean rating per genre and populating by dscending order
edx %>%
  group_by(genres) %>%
  summarize(mean_rating_by_genre = mean(rating)) %>%
  arrange(desc(mean_rating_by_genre))
```

```
## # A tibble: 20 x 2
##    genres             mean_rating_by_genre
##    <chr>                             <dbl>
##  1 Film-Noir                          4.01
##  2 Documentary                        3.78
##  3 War                                3.78
```

```
##  4 IMAX                          3.77
##  5 Mystery                       3.68
##  6 Drama                         3.67
##  7 Crime                         3.67
##  8 (no genres listed)            3.64
##  9 Animation                     3.60
## 10 Musical                       3.56
## 11 Western                       3.56
## 12 Romance                       3.55
## 13 Thriller                      3.51
## 14 Fantasy                       3.50
## 15 Adventure                     3.49
## 16 Comedy                        3.44
## 17 Action                        3.42
## 18 Children                      3.42
## 19 Sci-Fi                        3.40
## 20 Horror                        3.27
```

```r
# Plotting mean rating by genre with error bars
edx %>%
  group_by(genres) %>%
  summarize(
    n = n(),
    avgerge_rating = mean(rating),
    se = sd(rating) / sqrt(n())
  ) %>%
  filter(n >= 1000) %>%
  mutate(genres = reorder(genres, avgerge_rating)) %>%
  ggplot(aes(
    x = genres, y = avgerge_rating,
    ymin = avgerge_rating - 2 * se, ymax = avgerge_rating + 2 * se
  )) +
  geom_point(color = "#c4553a", size = 3) + #setting color and size of points
  geom_errorbar(color = "black", width = .5) + #setting error bar appearance
  ggtitle("Plot of Average Ratings Based on Genres") +
  labs(x = "Genres", y = "Average Rating") + #adding axis labels
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 90, hjust = 1, size = 10), #rotate x axis
    axis.text.y = element_text(size = 10), # set y axis
    axis.title = element_text(size = 12), # set axis title
    plot.title = element_text(hjust = 0.5), # set plot title
    panel.grid.major = element_line(color = "gray", linetype = "dashed"), #add dashed grid lines
    panel.grid.minor = element_blank() # removes minor grid lines
  )
```



Here we can see that that Film Noir genre has on average, higher ratings than the other genres, even though it has fewer movies in the data set as we saw earlier.

Modeling

First, we must further partition the cleaned EDX data set into train_set and test_set. This is in attempt to prevent overfitting of the models. Overfitting is when the model can't generalize the data well, and results in inaccurate predictions.

```
# create train set and test sets from edx
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index, ]
temp <- edx[test_index, ]

# make sure userId and movieId in test set are also in train set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# add rows removed from test set back into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

# remove temporary files to tidy environment
rm(test_index, temp, removed)
```

RMSE

As noted earlier, the RMSE will be utilized to validate the accuracy of the model. The goal is to be $<$ 0.86499.

```
RMSE <- function(true_ratings, predicted_ratings) {
  # calculate the squared differences between true and predicted ratings
  squared_errors <- (true_ratings - predicted_ratings)^2

  # calculate the mean of squared errors, taking care to handle NA values if present
  mean_squared_error <- mean(squared_errors, na.rm = TRUE)

  # return the square root of the mean squared error, which is the RMSE
  return(sqrt(mean_squared_error))
}
```

We calculate $\hat{\mu}$ to utilize in our models.

```
mu_hat <- mean(train_set$rating) # Calculate the mean rating of the training set
mu_hat
```

```
## [1] 3.526972
```

Model 1: Naive Mean

This model utilizes a simple formula of:

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

This model assumes that the rating for all movies and users is the same with a difference that can be explained by random variable.

$Y_{u,i}$ - is the predicted rating for users u on item i

μ - is the overall mean rating in the data set

$\epsilon_{u,i}$ - is the deviation specific to the users u and items i and captures the deviation of predicted rating $Y_{u,i}$ from the μ

We utilize the previously calculated $\hat{\mu}$ to predict all unknown ratings.

```
M1_NRMSE <- RMSE(test_set$rating, mu_hat) # calculate NRMSE for Model 1,
#where predictions are based solely on the mean rating.

# create tibble to store results
results_table <- tibble(
  Model_Type = c("Model 1:NRMSE"),
  RMSE = c(M1_NRMSE)
) %>%
  mutate(RMSE = sprintf("%0.4f", RMSE)) # format decimal to 4 places

results_table
```

```
## # A tibble: 1 x 2
##   Model_Type    RMSE
##   <chr>         <chr>
## 1 Model 1:NRMSE 1.0520
```

Model 2: Movie Effect Bias

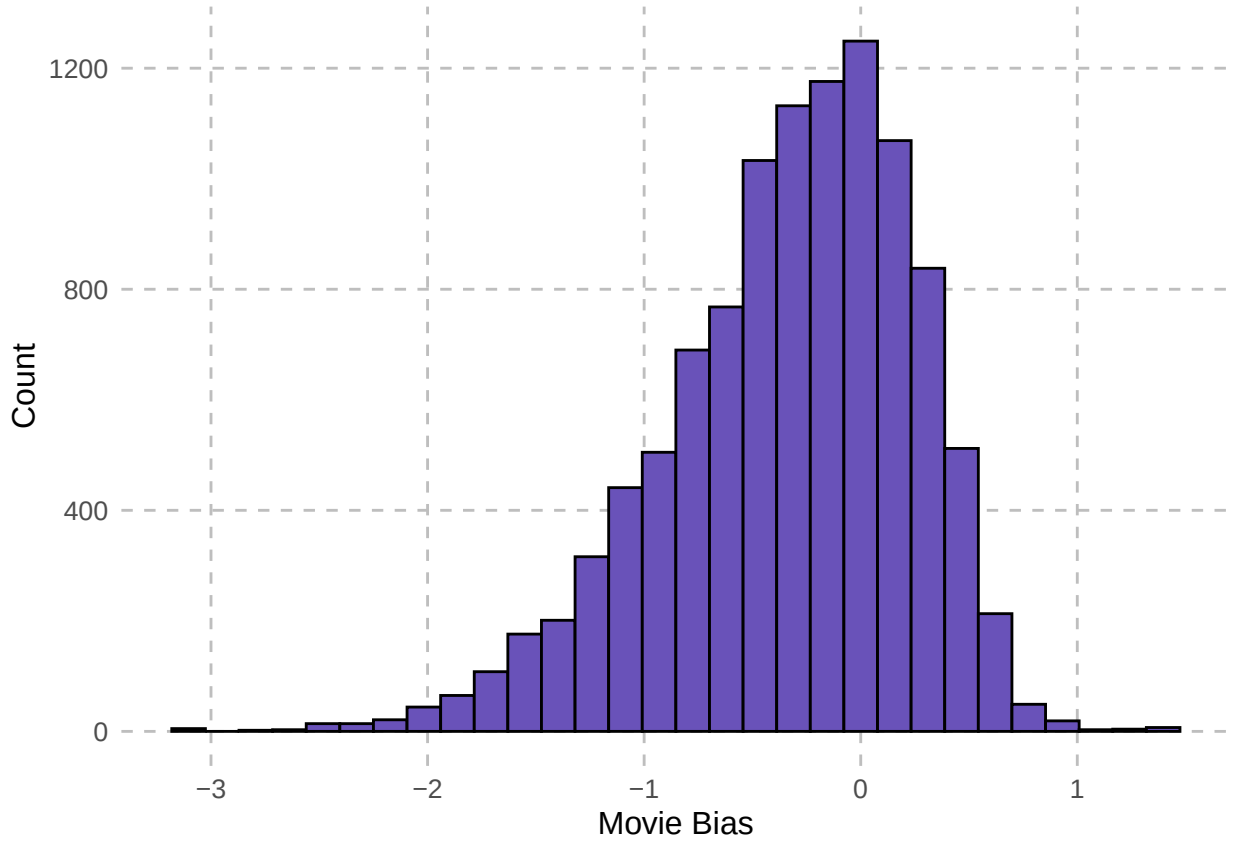
We can improve the first model by introducing movie bias. Based on prior review of the data, we know that some movies are just rated higher than others and that full star ratings are more common.

We can add this effect to the model by adding b_i, which is the average ranking for the movie. The new formula is as follows:

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

We can see this bias through this distribution. The mean is at 0 so a *bi* of 1.5 reflects a 5 star rating.

```
bi <- train_set %>%
  group_by(movieId) %>% # group the data by movieId
  summarize(b_i = mean(rating - mu_hat)) # calculate the mean of (rating - mu_hat) for each movieId

bi %>%
  ggplot(aes(b_i)) + # plotting by b_i
  geom_histogram(color = "black", fill = "#6952b9") +
  xlab("Movie Bias") +
  ylab("Count") +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5), # setting the title
    axis.text = element_text(size = 10), # setting text size
    axis.title = element_text(size = 12), # setting axis title size
    panel.grid.major = element_line(color = "gray", linetype = "dashed"), # adding dashed grid lines
    panel.grid.minor = element_blank() # removing minor grid lines
  )
```




```
prediction_bi <- mu_hat + test_set %>% # start with mu_hat and the test set data
  left_join(bi, by = "movieId") %>% # join with movie bias data
  .$b_i # extract the bias term 'b_i' as a vector of predicted ratings

# calculate the RMSE for Model 2
M2 <- RMSE(test_set$rating, prediction_bi)

# add results to table
results_table <- results_table %>%
  add_row(Model_Type = "Model 2: Mean & Movie Effects", RMSE = sprintf("%0.4f", M2))

results_table
```

```
## # A tibble: 2 x 2
##   Model_Type                    RMSE
##   <chr>                         <chr>
## 1 Model 1:NRMSE                 1.0520
## 2 Model 2: Mean & Movie Effects 0.9410
```

After adding in the movie bias we see an improvement in RMSE. This RMSE does not meet our goal, though, so let's continue adding to the model.

Model 3: User Effect Bias

Bias can be found in users as well. Some tend to rate more positively and others negatively and some rate more movies than others. We can add this effect to the model as b_u. The new formula is as follows:

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

```
bu <- train_set %>%
  left_join(bi, by = "movieId") %>% # join with movie bias data
  group_by(userId) %>% # group the data by userId
  summarize(b_u = mean(rating - mu_hat - b_i)) # calculate the mean of
#(rating - mu_hat - b_i) for each userId

predicted_ratings <- test_set %>%
  left_join(bi, by = "movieId") %>% # join with movie bias data
  left_join(bu, by = "userId") %>% # Join with user bias data
  mutate(pred = mu_hat + b_i + b_u) %>% # calculate the predicted ratings
  #incorporating both movie and user effects
  pull(pred) # extract the predicted ratings as a vector

# Calculate the RMSE for Model 3
M3 <- RMSE(predicted_ratings, test_set$rating)

# Add results to table
results_table <- results_table %>%
  add_row(Model_Type = "Model 3: Mean, Movie, & User Effects", RMSE = sprintf("%0.4f", M3))

results_table
```

```
## # A tibble: 3 x 2
##   Model_Type                          RMSE
##   <chr>                               <chr>
## 1 Model 1:NRMSE                       1.0520
## 2 Model 2: Mean & Movie Effects       0.9410
## 3 Model 3: Mean, Movie, & User Effects 0.8575
```

With the third model we've hit below our target RMSE of 0.86499. Let's see if we can further improve upon this model by adding in an additional bias of Movie Age.

Model 4: Movie Age Bias

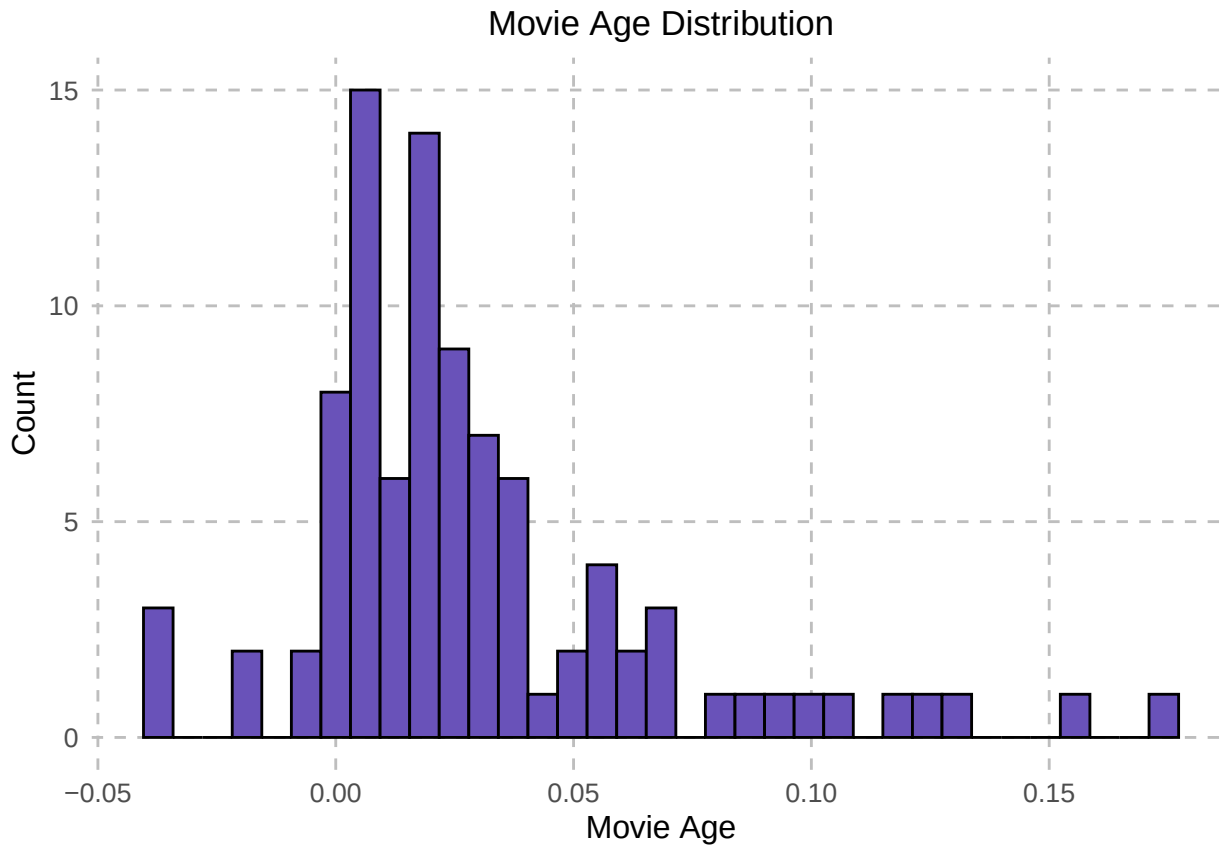
Previously, we noted that on average, the older the movie becomes the more likely it is to obtain a higher rating. This model incorporates that variation among the rating distribution for movies of different ages.

This model will add in b_a for movie age bias resulting in the following formula:

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

```
ba <- train_set %>%
  left_join(bi, by = "movieId") %>% # join with movie bias data
  left_join(bu, by = "userId") %>% # join with user bias data
  group_by(movie_age) %>% # group the data by movie_age
  summarize(b_a = mean(rating - b_i - b_u - mu_hat)) # calculate the mean of
#(rating - b_i - b_u - mu_hat) for each movie age group
```

```r
# plotting movie_age distribution based on b_a
ba %>%
  ggplot(aes(x = b_a)) + # setting x axid to b_a
  geom_histogram(fill = "#6952b9", color = "black", bins = 35) + # setting fill color
  ggtitle("Movie Age Distribution") +
  labs(x = "Movie Age", y = "Count") + # add axis labels
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5), # set title
    axis.text = element_text(size = 10), # set text size
    axis.title = element_text(size = 12), # set axis title size
    panel.grid.major = element_line(color = "gray", linetype = "dashed"), # adding dashed grid lines
    panel.grid.minor = element_blank() # removing minor grid lines
  )
```



This plot shows the distribution of movies ages based on their mean residual ratings (b_a). This is showing a right skew, with some outliers near .20.

Now let's add this bias, b_a to our model:

```r
predictions_ma <- test_set %>%
  left_join(bi, by = "movieId") %>% # join with the movie bias data
  left_join(bu, by = "userId") %>% # join with user bias data
  left_join(ba, by = "movie_age") %>% # join with age bias data
  mutate(predictions = mu_hat + b_i + b_u + b_a) %>% # calculate the predicted ratings incorporating mo
  .$predictions # extract the predicted ratings as a vector
```

```
# calculate the RMSE
M4 <- RMSE(test_set$rating, predictions_ma)

# add results to table
results_table <- results_table %>%
  add_row(Model_Type = "Model 4: Mean, Movie, User, & Movie Age Effects", RMSE = sprintf("%0.4f", M4))

results_table
```

```
## # A tibble: 4 x 2
##   Model_Type                                        RMSE
##   <chr>                                             <chr>
## 1 Model 1:NRMSE                                     1.0520
## 2 Model 2: Mean & Movie Effects                     0.9410
## 3 Model 3: Mean, Movie, & User Effects              0.8575
## 4 Model 4: Mean, Movie, User, & Movie Age Effects 0.8571
```

A slight improvement was made. Next, let's see if genre has any impact.

Model 5: Genre Bias

Based on the data, we know that certain genres just have more movies in the category. But we also saw that genres with fewer movies had higher ratings on average. Let's add b_g into the model to account for this bias, and see if this impacts our RMSE. This results in the following formula:

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

```
bg <- train_set %>%
  left_join(bi, by = "movieId") %>% # join with movie bias data
  left_join(bu, by = "userId") %>% # join with user bias data
  left_join(ba, by = "movie_age") %>% # join with age bias data
  group_by(genres) %>% # Group the data by genres.
  summarize(b_g = mean(rating - b_i - b_u - b_a - mu_hat)) # calculate the mean of
#(rating - b_i - b_u - b_a - mu_hat) for each genre

predictions_mg <- test_set %>%
  left_join(bi, by = "movieId") %>% #  join with movie bias data
  left_join(bu, by = "userId") %>% # join with user bias data
  left_join(ba, by = "movie_age") %>% # join with age bias data
  left_join(bg, by = "genres") %>% # join with genre bias data
  mutate(predictions = mu_hat + b_i + b_u + b_a + b_g) %>% # calculate the predicted ratings
  #incorporating movie, user, age, and genre effects
  .$predictions # extract the predicted ratings as a vector

# calculate the RMSE
M5 <- RMSE(test_set$rating, predictions_mg)

# add results to table
results_table <- results_table %>%
  add_row(Model_Type = "Model 5: Mean, Movie, User, Movie Age, & Genre Effects", RMSE = sprintf("%0.4f"

results_table
```

```
## # A tibble: 5 x 2
##   Model_Type                                                RMSE
##   <chr>                                                     <chr>
## 1 Model 1:NRMSE                                             1.0520
## 2 Model 2: Mean & Movie Effects                             0.9410
## 3 Model 3: Mean, Movie, & User Effects                      0.8575
## 4 Model 4: Mean, Movie, User, & Movie Age Effects           0.8571
## 5 Model 5: Mean, Movie, User, Movie Age, & Genre Effects 0.8571
```

No substantial gain in RMSE occurred with adding in the genre bias. Thus we will utilize Model 4 to balance performance and complexity.

Final Results

Before we complete the final test to confirm our model meets the desired accuracy, we must first apply the cleaning function to the final_holdout_test data set.

```
final_holdout_test <- cleaning_data(final_holdout_test, timestamp, title) # applying cleaning function
```

The data set is now cleaned and we can apply the final test based on Model 4.

```
ba_final_test <- final_holdout_test %>%
  left_join(bi, by = "movieId") %>% # join with movie bias data
  left_join(bu, by = "userId") %>% # join with the user bias data
  group_by(movie_age) %>% # group the data by movie_age
  summarize(b_a = mean(rating - b_i - b_u - mu_hat)) # calculate the mean of
#(rating - b_i - b_u - mu_hat) for each movie age group.

predictions_final <- final_holdout_test %>%
  left_join(bi, by = "movieId") %>% # join with movie bias data
  left_join(bu, by = "userId") %>% # join with user bias data
  left_join(ba_final_test, by = "movie_age") %>% # join with age bias data
  mutate(predictions = mu_hat + b_i + b_u + b_a) %>% # calculate the predicted ratings
  #incorporating movie, user, and age effects.
  .$predictions # extract the predicted ratings as a vector

# Calculate Final RMSE
final_RMSE <- RMSE(final_holdout_test$rating, predictions_final)

# Print Final RMSE
final_RMSE
```

```
## [1] 0.8633477
```

The final RMSE is *0.8633* which is below the goal. Our model was successful.

Conclusion

While Model 4 was successful in returning a RMSE lower than .8649, there are additional methods that could be deployed to determine if a lower RMSE could be obtained. One such method would be to utilize

a regularization technique, such as penalize least square errors, to introduce additional constraints to the model to penalize large estimates and prevent overfitting. Future should include adding in one of these techniques to improve the RMSE.

Similarly, there are several packages available that can assist in improving the RMSE. One such package, glment, fits generalize linear and similar models via penalize maximum likelihood. This package includes functions for cross-validation, and methods for prediction in plotting. Future work should investigate this package and others to determine if the RMSE value can be further lowered.

References

MovieLens Latest Datasets. (2015, September 23). GroupLens. https://grouplens.org/datasets/movielens/latest/

Irizarry, R. A. (n.d.). Chapter 33 Large datasets | Introduction to Data Science. In rafalab.dfci.harvard.edu. Retrieved March 18, 2024, from https://rafalab.dfci.harvard.edu/dsbook/large-datasets.html

An Introduction to `glmnet`. (n.d.). Glmnet.stanford.edu. https://glmnet.stanford.edu/articles/glmnet.html#:~:text=Glmnet%20is%20a%20package%20that