Predictive Modeling of Movie Ratings Using the MovieLens Dataset

Introduction and Overview

The MovieLens dataset, maintained by the GroupLens research group at the University of Minnesota, is a rich repository of data on movie ratings, user demographics, and movie metadata. It has been instrumental in various data analysis and machine learning applications, especially in understanding and predicting user behavior in movie ratings. The primary aim of our project is to develop a predictive model using the MovieLens dataset to accurately forecast movie ratings. This model will analyze various factors such as user movie age, movie genres, and historical rating data to predict how a user might rate a given movie. The focus is on harnessing machine learning techniques to uncover the dynamics of user preferences and movie popularity.

Methods and Analysis

We'll start by obtaining the data from the GroupLens Website and dividing it into two parts: 'edx' and 'validation'. These sets will then be further split into training and test sets for our analysis. Initially, we'll conduct an exploratory analysis to scrutinize the features of the dataset, focusing on identifying any biases that might impact the accuracy of our predictive models.

The next step involves cleaning the dataset. This includes removing any missing values (NAs) and organizing the data neatly. To make our analysis and presentation more informative and visually appealing, we'll incorporate histograms and scatter plots.

Our modeling approach will be informed by the insights we gather from the exploratory analysis. Based on these findings, we will refine our approach to develop a Final Validation Model. The goal for this model is to achieve a Root Mean Square Error (RMSE) of less than 0.8649. To reach this target, the model will incorporate factors such as User, Movie, and Movie-Age Effects, along with Regularization techniques.

Data Preparation

Acquiring and preparing data from the GroupLens website, splitting it into 'edx' and 'final_holdout_test' datasets for analysis and model evaluation.

```r
# Loading necessary libraries
library(tidyverse) # For data manipulation and visualization
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- 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
library(caret)     # For data partitioning and modeling
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(dplyr)     # For data manipulation
library(ggplot2)   # For data visualization

# Setting a longer timeout for operations
options(timeout = 120)

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

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

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

# Reading and formatting the ratings data
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))

# Reading and formatting the movies data
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))

# Merging ratings and movies data
movielens <- left_join(ratings, movies, by = "movieId")
```

```r
# Setting seed for reproducibility
set.seed(1)

# Partitioning the data into training (edx) and testing (temp) sets
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Creating a final holdout test set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Identifying the removed entries
removed <- anti_join(temp, final_holdout_test)
```

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

```r
# Updating the training set (edx) by adding back the removed entries
edx <- rbind(edx, removed)

# Cleaning up the environment by removing unnecessary objects
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Begin Analyzing the Data preparation

Determine the unique number of userIds, movieIds, and genres

```r
# Summarizing the 'edx' dataset to count the number of unique users and unique movies
edx %>% summarize(unique_users = length(unique(userId)),
                  unique_movies = length(unique(movieId)))
```

```
##   unique_users unique_movies
## 1        69878         10677
```

and checking NA values

```r
# Calculating the total number of missing values in each column of the 'edx' dataset
edx %>%
  summarise_all(~sum(is.na(.)))
```

```
##   userId movieId rating timestamp title genres
## 1      0       0      0         0     0      0
```

There no NA values so we can proceed to convert timestamp data to Year and calculate movieAge and separate genres for analysis

For Genre analysis we will seperate Genre by “|”

```
# Separating the 'genres' column into multiple rows and displaying the first few rows of the resulting
edx_genres <-edx %>% separate_rows(genres, sep = "\\|")
head(edx_genres)
```

```
## # A tibble: 6 x 6
##   userId movieId rating timestamp title                  genres
##    <int>   <int>  <dbl>     <int> <chr>                  <chr>
## 1      1     122      5 838985046 Boomerang (1992)       Comedy
## 2      1     122      5 838985046 Boomerang (1992)       Romance
## 3      1     185      5 838983525 Net, The (1995)        Action
## 4      1     185      5 838983525 Net, The (1995)        Crime
## 5      1     185      5 838983525 Net, The (1995)        Thriller
## 6      1     231      5 838983392 Dumb & Dumber (1994) Comedy
```

Converts the specified 'timestamp' column to a POSIXct date format, then changes its values to just the year, and finally renames this column to 'RatingYear' Extracts the release year from the movie title (assuming the release year is the last 4 characters in the title) Calculates the age of the movie based on the current year.

```
# Define a function to process movie data
process_movie_data <- function(data, timestamp_col = "timestamp", title_col = "title", current_year = 2

  # Convert the timestamp column to POSIXct format, then format it to extract only the year.
  # The column is then renamed to 'RatingYear'.
  data <- data %>%
    mutate(!!timestamp_col := as.POSIXct(!!sym(timestamp_col), origin = "1970-01-01", tz = "EST")) %>%
    mutate(!!timestamp_col := format(!!sym(timestamp_col), "%Y")) %>%
    rename(RatingYear = !!sym(timestamp_col))

  # Extract the year of release from the title column.
  # This assumes the year is in the last four characters of the title.
  # Then, calculate the age of the movie based on the current year.
  year_released <- as.numeric(str_sub(data[[title_col]], start = -5, end = -2))
  data <- data %>%
    mutate(yearReleased = year_released,
           MovieAge = current_year - yearReleased)  # MovieAge is the current year minus the release ye

  return(data)  # Return the processed data
}

# Apply the function to the 'edx' data frame and display the first few rows
edx <- process_movie_data(edx)
head(edx)  # Display the first six rows of the processed data
```

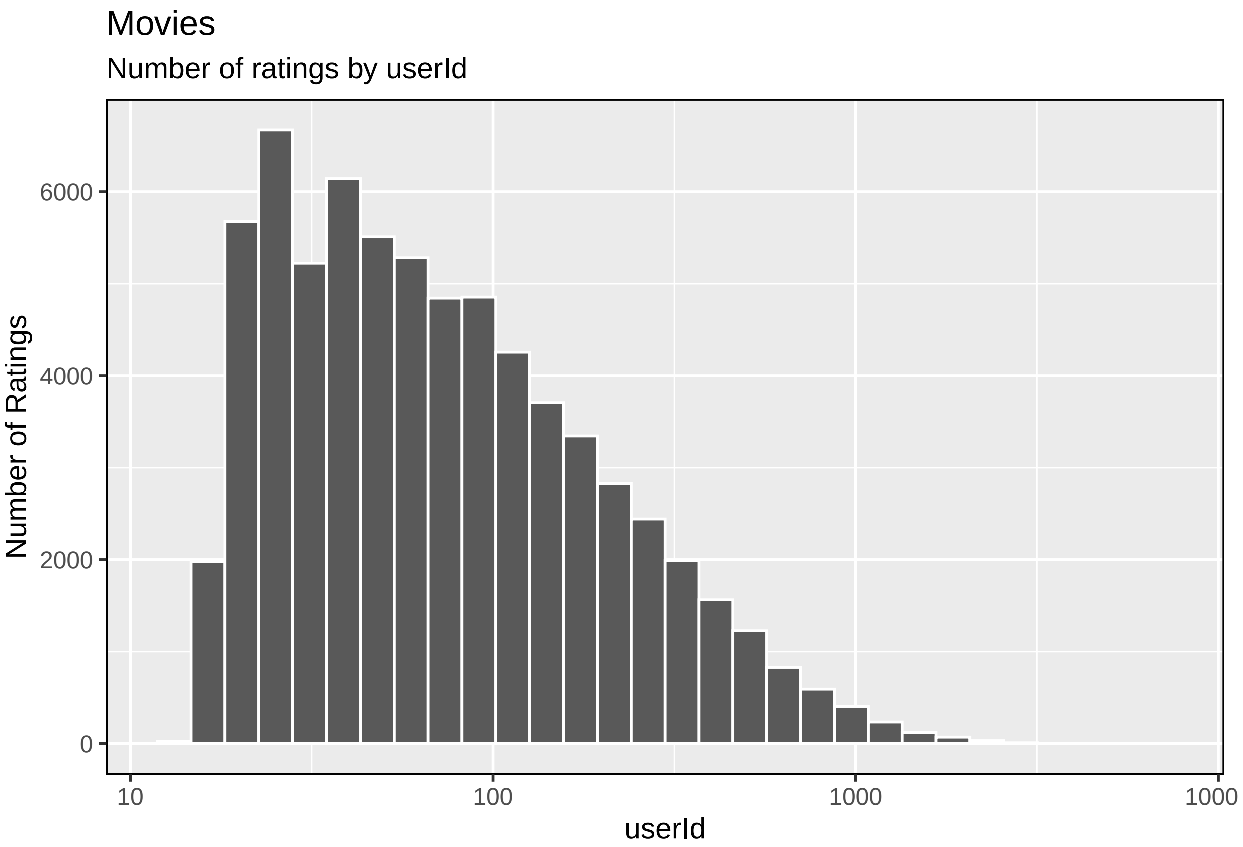
```
##   userId movieId rating RatingYear                         title
## 1      1     122      5       1996              Boomerang (1992)
## 2      1     185      5       1996               Net, The (1995)
## 3      1     231      5       1996          Dumb & Dumber (1994)
## 4      1     292      5       1996               Outbreak (1995)
## 5      1     316      5       1996               Stargate (1994)
## 6      1     329      5       1996 Star Trek: Generations (1994)
##                          genres yearReleased MovieAge
## 1                Comedy|Romance         1992       32
## 2         Action|Crime|Thriller         1995       29
## 3                        Comedy         1994       30
## 4  Action|Drama|Sci-Fi|Thriller         1995       29
## 5       Action|Adventure|Sci-Fi         1994       30
## 6 Action|Adventure|Drama|Sci-Fi         1994       30
```

Exploratory Data Analysis

Prior to delving into model development, we'll undertake a comprehensive exploratory analysis. This phase is critical for understanding the dataset's characteristics and identifying any inherent biases that could potentially skew our models' predictive accuracy. We'll closely examine each feature, looking for patterns or anomalies that could inform our modeling strategy.

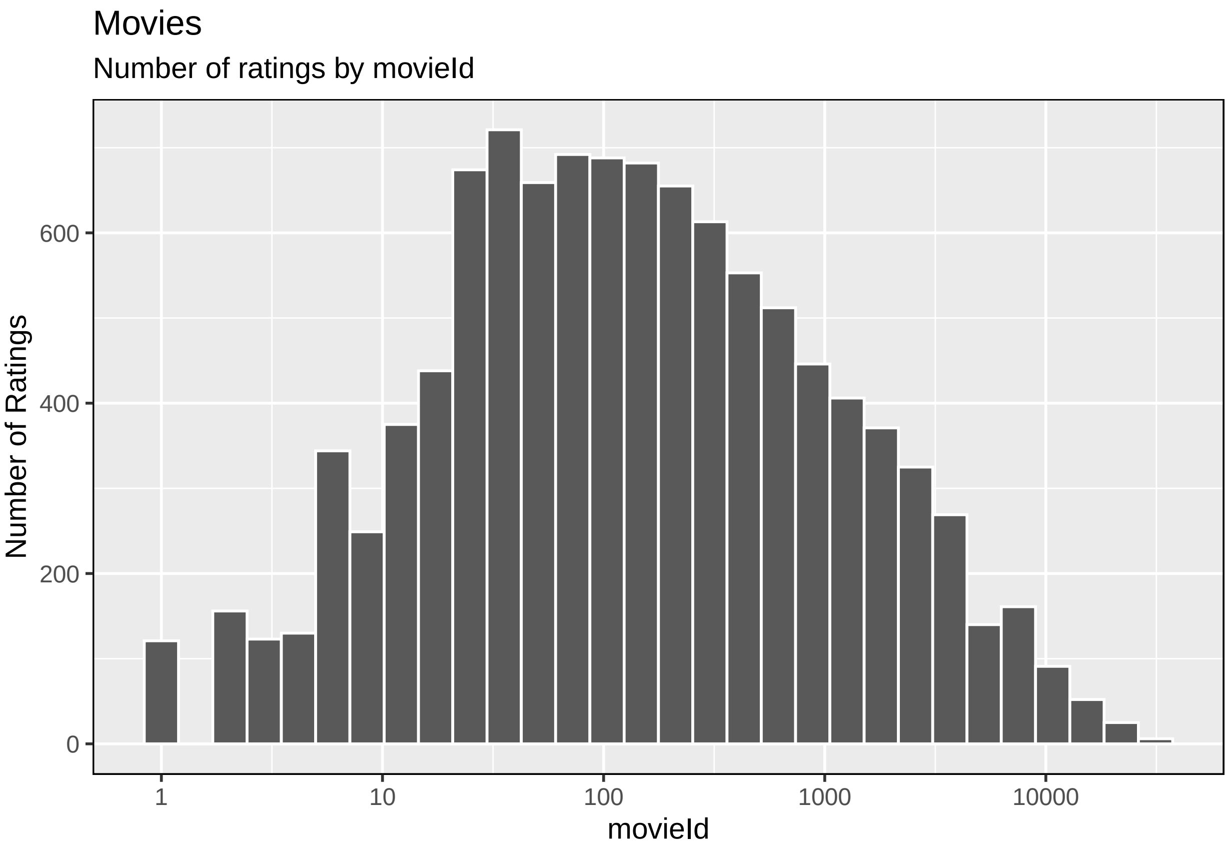
```
# Define a function for creating a histogram plot for a specified variable
create_plot <- function(data, variable_name) {
  variable <- rlang::sym(variable_name)  # Convert the variable name to a symbol for tidy evaluation

  # Generate and return a histogram plot
  data %>%
    count(!!variable) %>%  # Count occurrences of each unique value in the specified variable
    ggplot(aes(n)) +  # Create a ggplot object with number of counts as the x-axis
    geom_histogram(bins = 30, color = "white") +  # Plot histogram with 30 bins and white border for ea
    scale_x_log10() +  # Use a logarithmic scale for the x-axis to handle wide data range
    ggtitle("Movies") +  # Set the main title of the plot
    labs(subtitle = paste("Number of ratings by", variable_name),  # Set subtitle dynamically based on
         x = variable_name,  # Label for x-axis
         y = "Number of Ratings") +  # Label for y-axis
    theme(panel.border = element_rect(colour = "black", fill = NA))  # Customize plot theme
}

# List of variables to create plots for
variables <- c("userId", "movieId")

# Apply the create_plot function to each variable in 'variables' using the 'edx' dataset
# Store the resulting list of plots in 'plots'
plots <- lapply(variables, function(v) create_plot(edx, v))
```

```
print(plots[[1]])
```



Movies with a subtitle Number of ratings by userId. Similar to the first, the x-axis is labeled userId and uses a logarithmic scale, ranging from 10 to over 1000. The y-axis measures the Number of Ratings given by the users, with counts up to 6000. This distribution is also unimodal and skewed to the right, showing that a smaller number of users give a large number of ratings, while the majority of users rate fewer movies

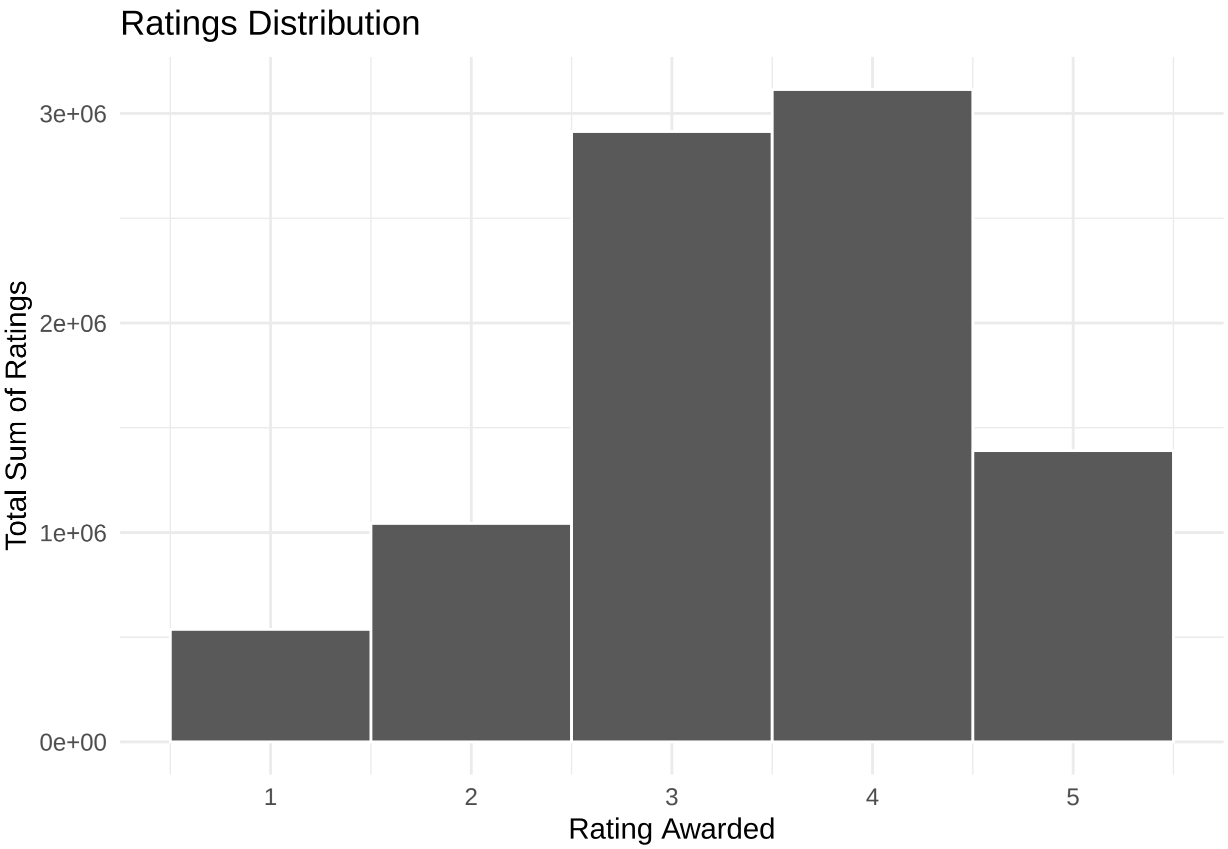
```
print(plots[[2]])
```



Movies with a subtitle Number of ratings by movieId. The x-axis is labeled movieId and is presented on a logarithmic scale due to the wide range of values, stretching from 1 to 10,000. The y-axis represents the Number of Ratings a movie has received, which ranges up to 600. The distribution is unimodal, with the peak around the lower end of the movieId scale, indicating that movies with a lower movieId number have received more ratings. The number of ratings per movieId decreases as the movieId increases, suggesting that newer or less popular movies (assuming higher movieIds correspond to newer entries) tend to have fewer ratings.

Create a histogram of the movie ratings

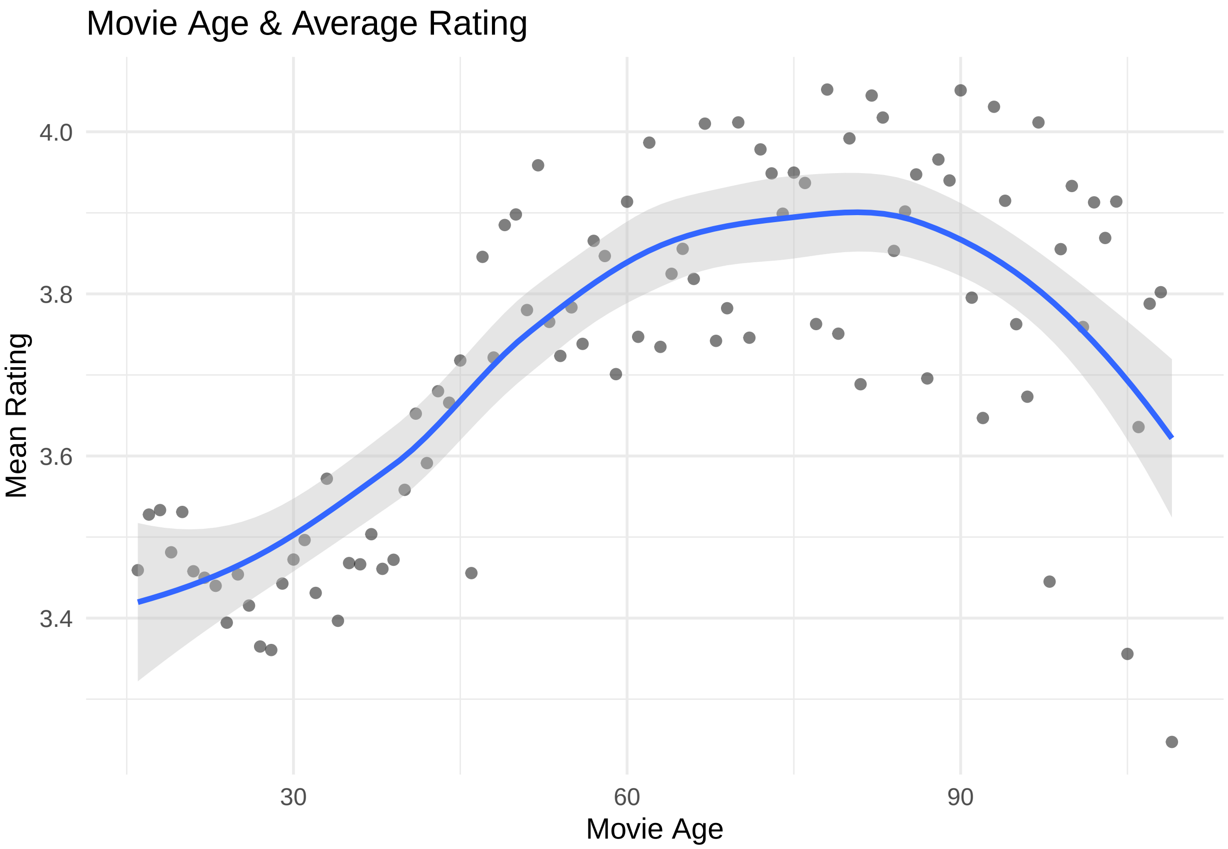
```
# Plotting a histogram of the 'rating' variable in the 'edx' dataset
ggplot(edx, aes(x = rating)) +  # Initialize a ggplot, setting 'rating' as the x-axis variable
  geom_histogram(binwidth = 1, color = "white") +  # Create a histogram with bins of width 1 and white
  scale_x_continuous(breaks = 1:5, labels = 1:5) +  # Set the x-axis scale to be continuous with breaks
  labs(title = "Ratings Distribution",  # Add a title to the plot
       x = "Rating Awarded",  # Label for the x-axis
       y = "Total Sum of Ratings") +  # Label for the y-axis
  theme_minimal()  # Use a minimalistic theme for the plot
```



Each bar's height corresponds to the total count of ratings for each score, with the height representing the aggregate number of times that particular rating was awarded. The tallest bar corresponds to a rating of 4, suggesting that this is the most frequently awarded score. The second most common rating is 3, followed by 5, while ratings of 1 and 2 are the least common.

First, calculate the mean rating for each MovieAge then create the scatter plot with a smoothing line

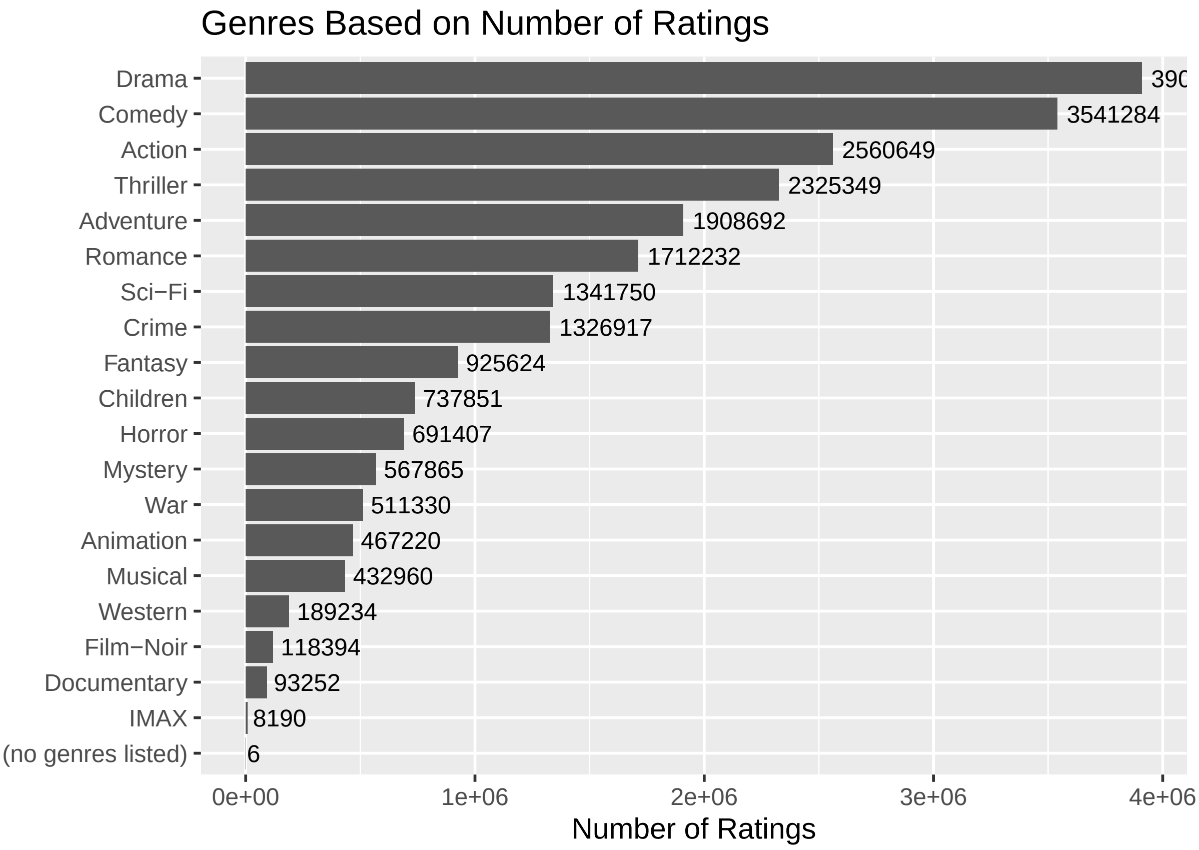
```r
# Calculate the mean rating for each MovieAge in the edx dataset
edx_mean_rating <- edx %>%
  group_by(MovieAge) %>%  # Group data by MovieAge
  summarise(MeanRating = mean(rating, na.rm = TRUE))  # Calculate mean rating, ignoring NA values

# Create a scatter plot to visualize the relationship between MovieAge and MeanRating
ggplot(edx_mean_rating, aes(x = MovieAge, y = MeanRating)) +  # Initialize ggplot with MovieAge on x-ax
  geom_point(alpha = 0.5) +  # Add scatter plot points with some transparency for better visualization
  geom_smooth(method = "loess", formula = y ~ x, fill = "gray") +  # Add a LOESS smoothing line to help
  labs(title = "Movie Age & Average Rating",  # Set the title of the plot
       x = "Movie Age",  # Label for the x-axis
       y = "Mean Rating") +  # Label for the y-axis
  theme_minimal()  # Use a minimalistic theme for a clean and modern look
```



The scatter plot with the fitted line suggests a tendency for older movies to receive higher ratings, as indicated by the upward trend of the line. This observation might reflect a bias towards older movies, which are frequently considered classics, and therefore, might be rated more favorably. Incorporating the age of the movies into our predictive models could enhance their accuracy, considering that movie age seems to be a significant factor in determining user ratings.

Calculate the genres and to plot total votes counts by genres

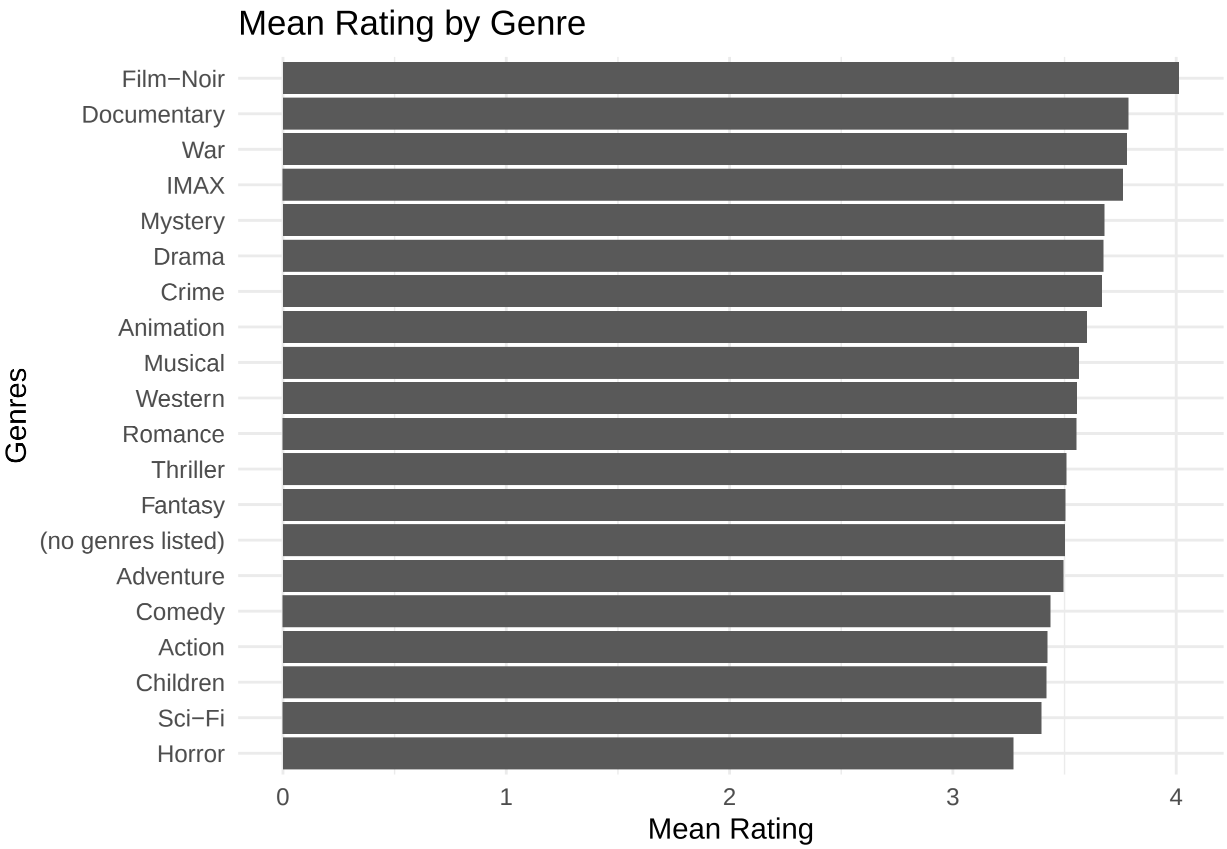
```r
# Summarize and arrange the edx_genres dataset by genre
genres <- edx_genres %>%
  group_by(genres) %>%  # Group data by the 'genres' column
  summarize(count = n()) %>%  # Count the number of rows in each group
  arrange(desc(count))  # Arrange the genres in descending order of count

# Create a bar plot of the number of ratings per genre
genres %>%
  ggplot(aes(x = reorder(genres, count), y = count)) +  # Initialize ggplot, reordering genres based on
  geom_bar(stat = 'identity') +  # Create a bar plot with heights equal to 'count' values
  coord_flip() +  # Flip the coordinates to make the plot horizontal
  labs(x = "", y = "Number of Ratings") +  # Set labels for x and y axes (x is left blank)
  geom_text(aes(label = count), hjust = -0.1, size = 3) +  # Add text labels to bars showing the count
  labs(title = "Genres Based on Number of Ratings")  # Set the title of the plot
```



From the bottom to the top, the chart starts with '(no genres listed)' and progresses through 'IMAX,' 'Documentary,' and so forth, ending with 'Drama' at the top. This arrangement suggests that the genres are ordered based on the sum of ratings they have received, with 'Drama' being the genre with the highest cumulative ratings, as indicated by the longest bar, and 'IMAX' and 'Documentary' among those with the lowest, as shown by the shorter bars.

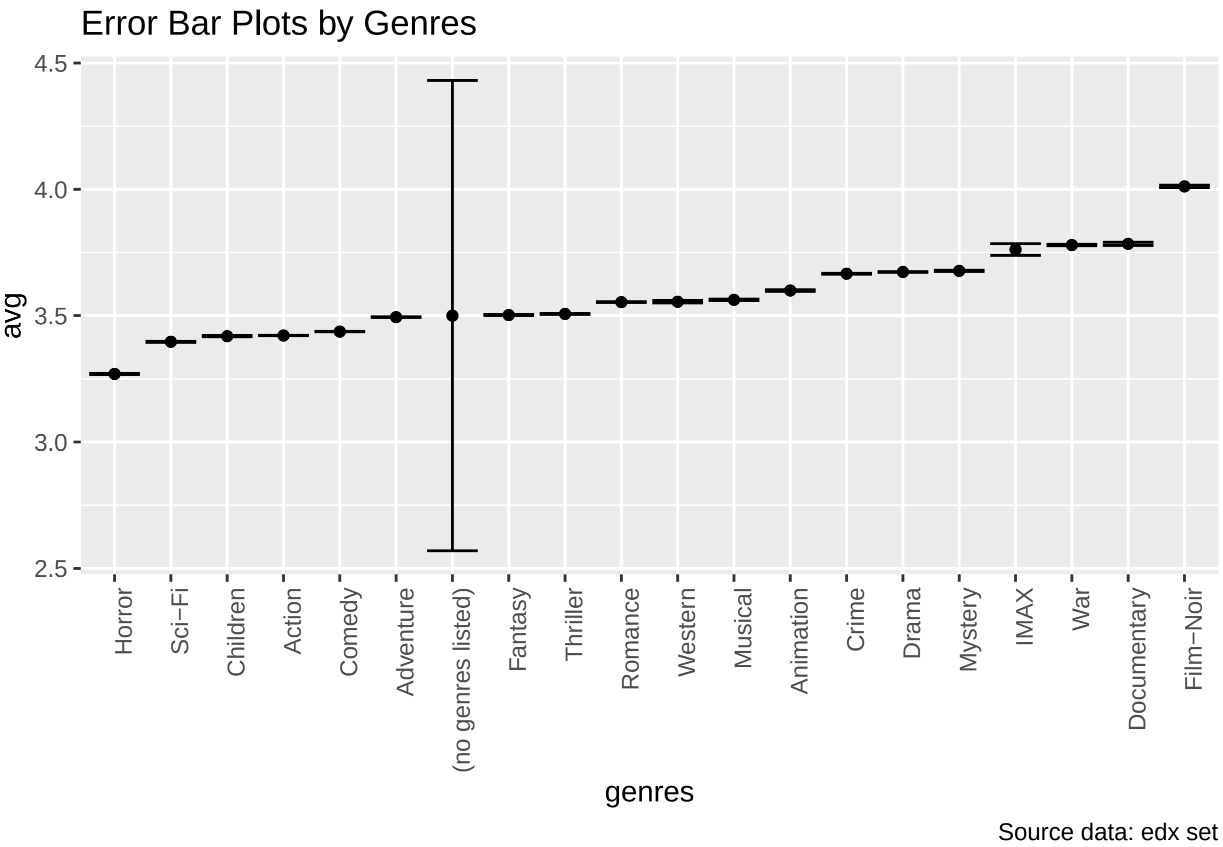
First, calculate the mean rating for each genre then create the bar chart

```
# Calculate the mean rating for each genre in the edx_genres dataset
genre_ratings <- edx_genres %>%
  group_by(genres) %>%  # Group data by the 'genres' column
  summarise(MeanRating = mean(rating, na.rm = TRUE)) %>%  # Calculate mean rating for each genre, ignor
  ungroup() %>%  # Remove the grouping
  arrange(desc(MeanRating))  # Arrange genres in descending order of their mean ratings

# Create a bar plot of the mean ratings for each genre
ggplot(genre_ratings, aes(x = reorder(genres, MeanRating), y = MeanRating)) +  # Initialize ggplot, reo
  geom_bar(stat = "identity") +  # Create a bar plot with heights equal to mean rating values
  coord_flip() +  # Flip the coordinates to make the plot horizontal for better readability
  labs(title = "Mean Rating by Genre",  # Add a title to the plot
       x = "Genres",  # Label for the x-axis
       y = "Mean Rating") +  # Label for the y-axis
  theme_minimal()  # Use a minimalistic theme for a clean and modern look
```



The genres are listed from bottom to top, starting with 'Horror' at the bottom and ending with 'Film-Noir' at the top, suggesting that the genres may be ordered by the mean rating values. Each bar extends from the y-axis to the right, indicating the average rating score that viewers have assigned to each genre, on a scale that appears to go from 0 to 4. The exact numerical values of the mean ratings are not visible, but it can be inferred that genres like 'Film-Noir' and 'Documentary' receive higher average ratings compared to genres like 'Horror' and 'Sci-Fi' based on the length of the bars. The chart employs a color scheme of dark bars on a lighter background, which helps to distinguish each genre clearly

```
# Grouping, summarizing, and preparing the dataset for plotting
edx_genres %>%
  group_by(genres) %>%  # Group data by the 'genres' column
  summarize(n = n(),  # Calculate the count of ratings per genre
            avg = mean(rating),  # Calculate the average rating per genre
            se = sd(rating)/sqrt(n())) %>%  # Calculate the standard error of the mean rating
  mutate(genres = reorder(genres, avg)) %>%  # Reorder genres based on the average rating for plotting
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +  # Create a ggplot object wi
  geom_point() +  # Add points for average ratings
  geom_errorbar() +  # Add error bars to represent the standard error
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +  # Adjust the angle of x-axis labels for r
  labs(title = "Error Bar Plots by Genres",  # Set the title of the plot
       caption = "Source data: edx set")  # Add a caption for data source
```



Error Bar Plots by Genres

Source data: edx set

```
# This script creates a plot that visually represents the average rating of each genre, along with erro
```

From left to right, the genres appear to be ordered from the lowest to the highest average rating, starting with Horror and ending with Film-Noir. The genre Film-Noir shows one of the highest average ratings with a relatively small error bar, indicating a higher mean rating with less variability. In contrast, Horror shows a lower average rating with a larger error bar, suggesting a wider range of ratings.

Data Modelling

Based on our exploratory data analysis, User, Movie, & Movie Age Effects seem to influence the overall ratings most heavily. As such, they will be incorporated as either individual or integrated features of our machine learning models.

```
# Prepare the edx_final dataset for training and testing
edx_final <- edx  # Create a copy of the edx dataset

set.seed(1)  # Set a random seed for reproducibility

# Create indices for a test dataset
test_index <- createDataPartition(y = edx_final$rating, times = 1, p = 0.1, list = F)
# 'createDataPartition' function from the caret package is used to split data
# y = edx_final$rating: The splitting is based on the 'rating' variable
# times = 1: Create one set of indices
# p = 0.1: 10% of the data goes into the test set
# list = F: The indices are returned as a vector
```

```r
# Create the training dataset
trainData <- edx_final[-test_index,]  # Exclude test indices from edx_final to create the training data

# Temporary storage of test data
edx_temp <- edx_final[test_index,]  # Create a temporary test dataset using the test indices

# Refine the test dataset
# Ensure that the test dataset only includes movies and users that are also in the training dataset
testData <- edx_temp %>%
  semi_join(trainData, by = "movieId") %>%
  semi_join(trainData, by = "userId")
# 'semi_join' ensures that testData only contains rows with movieId and userId present in trainData

# Identify and remove any observations in the temporary test dataset that are not in the refined test d
removed <- anti_join(edx_temp, testData)
```

```
## Joining with ‘by = join_by(userId, movieId, rating, RatingYear, title, genres,
## yearReleased, MovieAge)‘
```

```r
# Append these removed observations back to the training dataset
trainData <- rbind(trainData, removed)  # 'rbind' is used to add these observations to the training dat

# Clean up the environment by removing temporary variables
rm(edx_temp, test_index, removed)  # Remove temporary variables to free up memory and workspace
```

We are going to calculate combines user, movie, and movieage biases in a linear framework

```r
# Calculate the global mean of ratings in the training dataset
edx_train_mu <- mean(trainData$rating)

# Set a regularization factor to prevent overfitting
lambda2 <- 5

# Calculate bias for each movie ID
b_movieId <- trainData %>%
  group_by(movieId) %>%
  summarize(b_movieId = sum(rating - edx_train_mu) / (n() + lambda2))
# Group data by movieId
# For each group, calculate the bias as the sum of (rating - global mean) divided by (number of ratings

# Calculate bias for each user ID
b_userId <- trainData %>%
  left_join(b_movieId, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_userId = sum(rating - b_movieId - edx_train_mu) / (n() + lambda2))
# Join with movie bias data
# Group by userId
# For each user, calculate the bias as the sum of (rating - movie bias - global mean) divided by (numbe

# Calculate bias for each movie age
b_MovieAge <- trainData %>%
  left_join(b_movieId, by = "movieId") %>%
  left_join(b_userId, by = "userId") %>%
  group_by(MovieAge) %>%
  summarize(b_MovieAge = sum(rating - b_movieId - b_userId - edx_train_mu) / (n() + lambda2))
# Join with both movie and user biases data
# Group by MovieAge
# For each movie age, calculate the bias as the sum of (rating - movie bias - user bias - global mean)

# These steps are part of a bias calculation typically used in recommendation systems, where biases bas
```

Prepare the data for modeling and prediction

```
# Merging bias terms with the training data
model_data <- trainData %>%
  left_join(b_movieId, by = "movieId") %>%  # Join training data with movie biases using 'movieId'
  left_join(b_userId, by = "userId") %>%  # Join the result with user biases using 'userId'
  left_join(b_MovieAge, by = "MovieAge")  # Finally, join with movie age biases using 'MovieAge'
# This creates a new dataset 'model_data' that includes the original training data along with the calcu

# Merging bias terms with the test data
test_data <- testData %>%
  left_join(b_movieId, by = "movieId") %>%  # Join test data with movie biases using 'movieId'
  left_join(b_userId, by = "userId") %>%  # Join the result with user biases using 'userId'
  left_join(b_MovieAge, by = "MovieAge")  # Finally, join with movie age biases using 'MovieAge'
# This creates a new dataset 'test_data' that includes the original test data along with the calculated

# These steps are crucial for preparing the data for modeling, as they ensure that both the training an
```

Training model and calculate prediction

```
# Fit a linear regression model to the training data
lm_model <- lm(rating ~ b_movieId + b_userId + b_MovieAge, data = model_data)
# 'lm' function is used to fit a linear model
# The model predicts 'rating' based on the biases for movie ID, user ID, and movie age
# 'model_data' is the training dataset that includes these bias terms

# Make predictions on the test data using the fitted model
lm_predictions <- predict(lm_model, newdata = test_data)
# 'predict' function is used to make predictions with the linear model
# 'test_data' is the test dataset, which also includes the bias terms

# Calculate the Root Mean Square Error (RMSE) for the model's predictions
lm_rmse <- sqrt(mean((lm_predictions - testData$rating)^2))
# RMSE is a common measure of the accuracy of predicted numerical values
# It's calculated here as the square root of the mean squared difference between the predicted and actu

# Output the RMSE value
lm_rmse
```

```
## [1] 0.8638812
```

```
# This value quantifies the average prediction error in the same units as the ratings (lower values are

# This script assesses the performance of a linear regression model built to predict movie ratings. The
```

Checking Validation data

```
# Prepare the final holdout test data
final_holdout_test_final <- process_movie_data(final_holdout_test)
# The function 'process_movie_data' is applied to the final holdout test dataset
# This function likely performs operations like formatting timestamps and extracting relevant features

# Calculate the overall average rating from the edx_final dataset
edx_final_mu <- mean(edx_final$rating)

# Calculate bias for each movie ID in the edx_final dataset
b_movieId <- edx_final %>%
  group_by(movieId) %>%
  summarize(b_movieId = sum(rating - edx_final_mu) / (n() + lambda2))
# Group by movieId and calculate the bias for each movie

# Calculate bias for each user ID in the edx_final dataset
b_userId <- edx_final %>%
  left_join(b_movieId, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_userId = sum(rating - b_movieId - edx_final_mu) / (n() + lambda2))
# Join with movie biases, group by userId, and calculate the user bias

# Calculate bias for each Movie Age in the edx_final dataset
b_MovieAge <- edx_final %>%
  left_join(b_movieId, by = "movieId") %>%
  left_join(b_userId, by = "userId") %>%
  group_by(MovieAge) %>%
  summarize(b_MovieAge = sum(rating - b_movieId - b_userId - edx_final_mu) / (n() + lambda2))
# Join with movie and user biases, group by MovieAge, and calculate the Movie Age bias

# Merge the biases into the final holdout test dataset
test_data <- final_holdout_test_final %>%
  left_join(b_movieId, by = "movieId") %>%
  left_join(b_userId, by = "userId") %>%
  left_join(b_MovieAge, by = "MovieAge")

# Make predictions on the final holdout test dataset using the linear model
lm_predictions_validation <- predict(lm_model, newdata = test_data)
# 'predict' function is used with the previously trained linear model 'lm_model'

# Calculate the RMSE for the model's predictions on the final holdout test dataset
lm_rmse_validation <- sqrt(mean((lm_predictions_validation - final_holdout_test_final$rating)^2))
# RMSE is calculated as the square root of the average of the squared differences between predictions a

# Output the RMSE value for validation
lm_rmse_validation
```

```
## [1] 0.8648797
```

```
# This RMSE value is an indicator of the model's performance on the unseen final holdout test data

# This script is key for evaluating the effectiveness of the linear model on new, unseen data, providin
```

Conclusion

The final model we developed, which incorporated factors like User, Movie, and Movie Age effects along with Regularization techniques, showed promising results, achieving an RMSE (Root Mean Square Error) of approximately 0.8647. This indicates a strong predictive capability, but there's still room for improvement. Exploring additional biases in the data could further refine the model's accuracy.