Harvard_Capstone_Movielens_Final

Luke Buaas

2024-05-13

Introduction:

In recent years, the widespread adoption of artificial intelligence has driven a significant push for companies to become more data-driven and technologically advanced. This change has led to an increasing number of organizations leveraging historical data to better inform their strategic decision making processes. Machine learning, which involves the process of developing and using models that learn from existing data to predict unknown variables, is becoming increasingly popular to make theses decisions in the context of business strategy.

Recommendation systems leverage machine learning to predict user preferences and make individualized suggestions that best fit the individual user. In this project, I aim to develop a regression model that accurately predicts movie ratings based on a subset of the MovieLens data set.

First, I will use code provided in class to load the MovieLens data subset and split it into training (edx) and testing (final_holdout_test) data sets. The training data set will be used to build my initial naive model and to refine my regression model in an iterative process. The testing data set will be used to evaluate the model's performance once I create my final model.

For this project I am aiming to achieve an RMSE of 0.86490 or less. Having this goal will help me to create multiple iterations of my regression model, adding variables to better account for the variation in the data. By achieving this RMSE I will have created an accurate movie recommendation multivariate regression model, that I will be applicable to real data.

In the below code received in class, I will load the data then split it into a training and test data set. This is done to overcome any potential over-fitting issues that would potentially occur if my model was solely tested on the training data, as there are data irregularities in real life data.

Create edx and final_holdout_test sets

```
############################################################
# Create edx and final_holdout_test sets - 125.9 Class Code
############################################################

# Note: this process could take a couple of minutes
```

```r
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(caret)

# 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"
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
# set.seed(1) # if using R 3.5 or earlier
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")
```

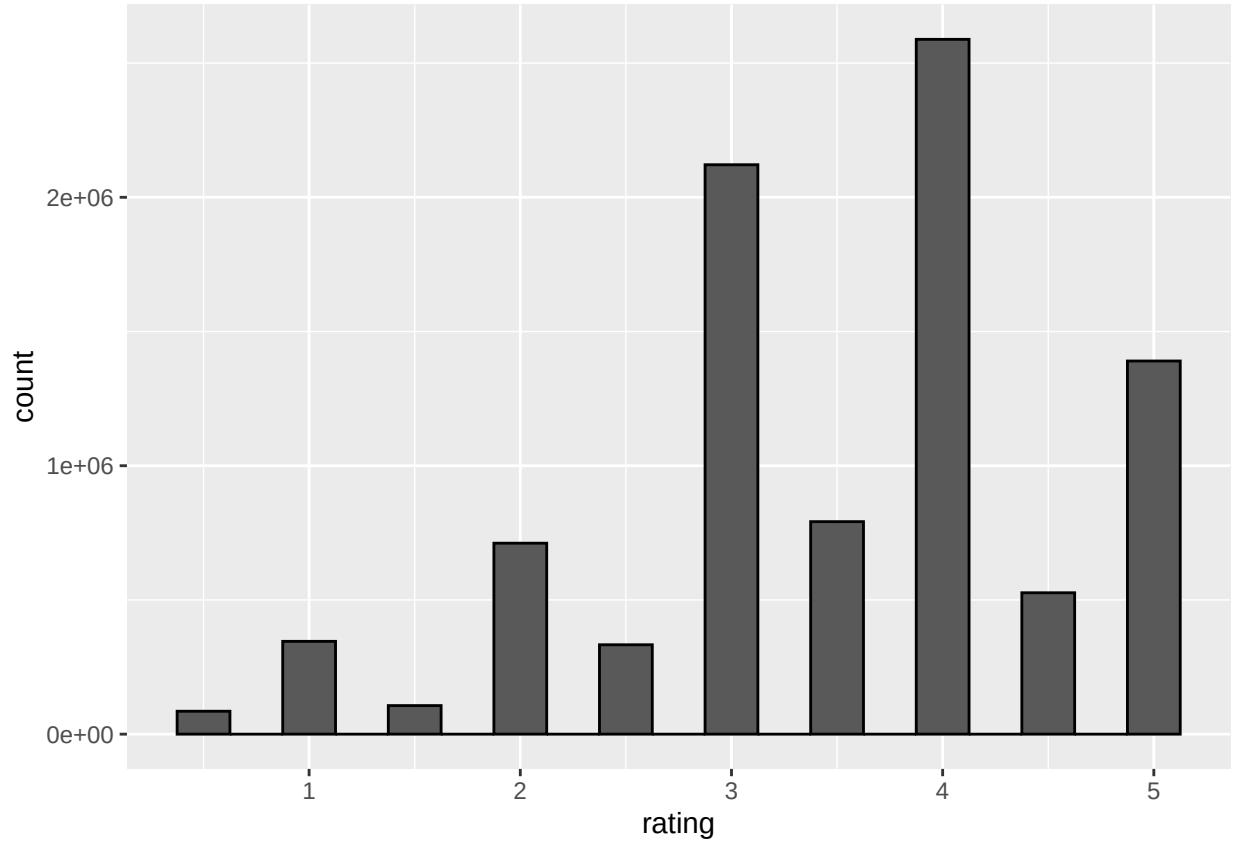
```r
# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

Before beginning to fit models to the data, I will examine the variables present in the edx data set to better understand this data. Below I examine the relationship between the available variables and our target variable, ratings. Below I have decided not to analyze "title" and opt for using the movie ID instead. This was done since there is one unique movie ID to one unique title, meaning that they are the same data in two separate formats.

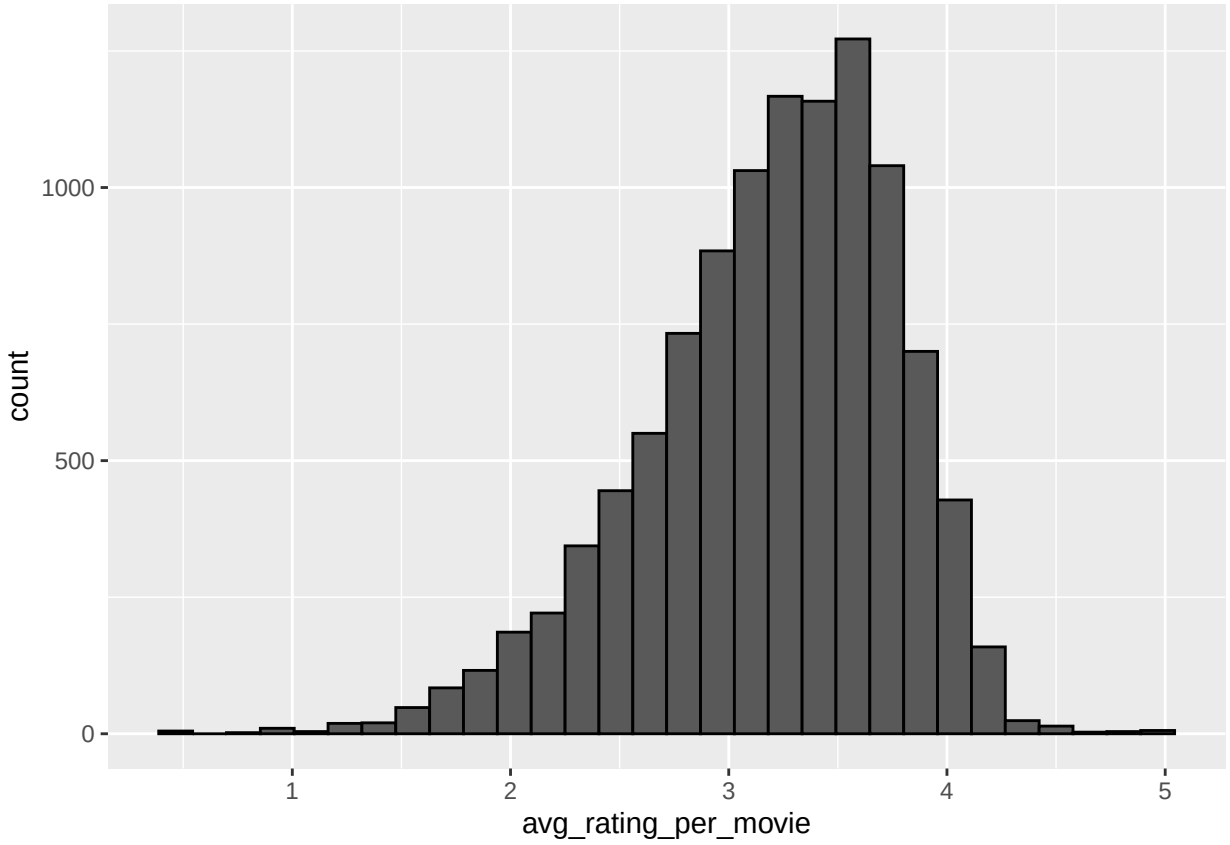
Also, while examining this data I noticed in the "title" variable, that the release year of each movie was included. Since there may be variation in the rating based off of the movie's year of release, I will make a new column in the training (edx) data set for this variable called "movieYear".

Data Exploration & Analysis

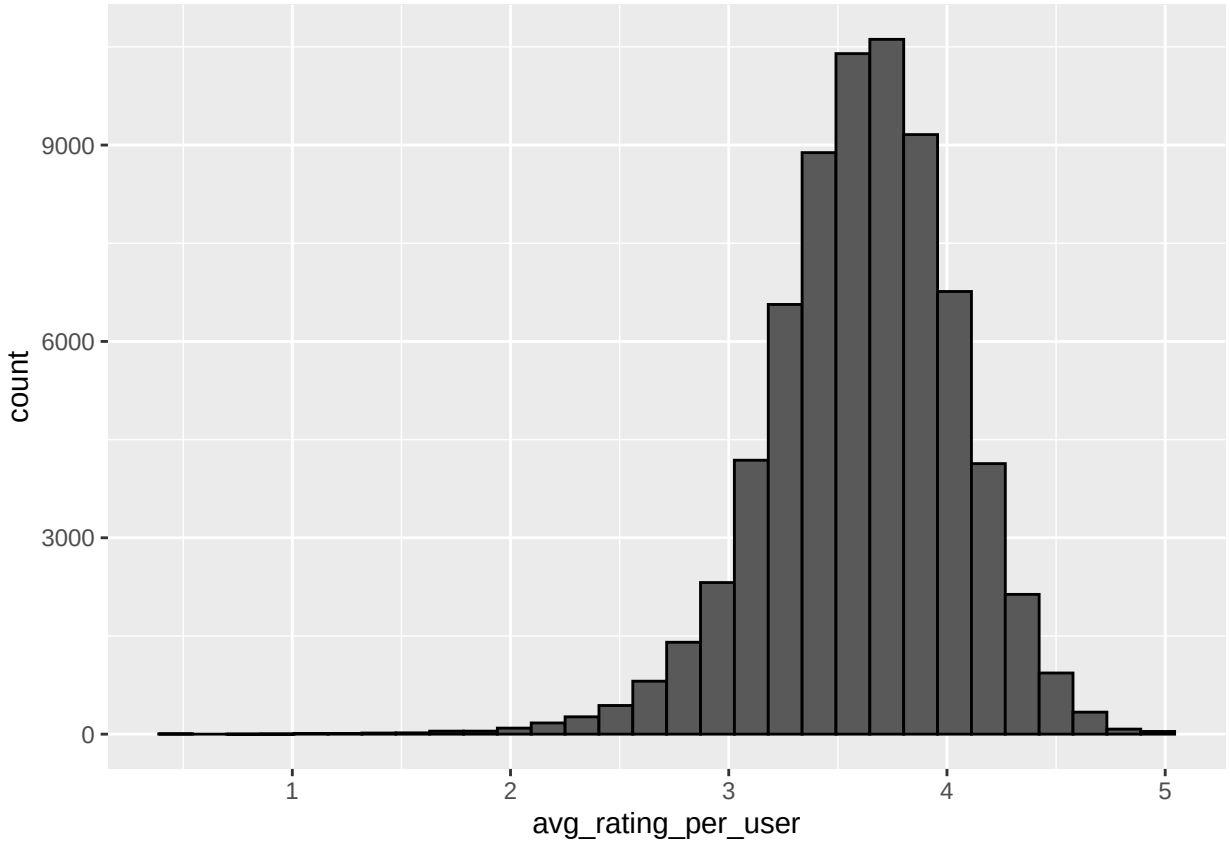
```r
# Plots counts of each rating
edx %>% ggplot(aes(x = rating)) +
  geom_histogram(binwidth = .25, color = "black")
```



```r
# Plots average rating per movie
edx %>% group_by(movieId) %>%
  summarise(avg_rating_per_movie = mean(rating)) %>%
  ggplot(aes(avg_rating_per_movie)) +
  geom_histogram(bins=30, color = "black")
```




```r
# Plots average rating per user
edx %>% group_by(userId) %>%
  summarise(avg_rating_per_user = mean(rating)) %>%
  ggplot(aes(avg_rating_per_user)) +
  geom_histogram(bins=30, color = "black")
```



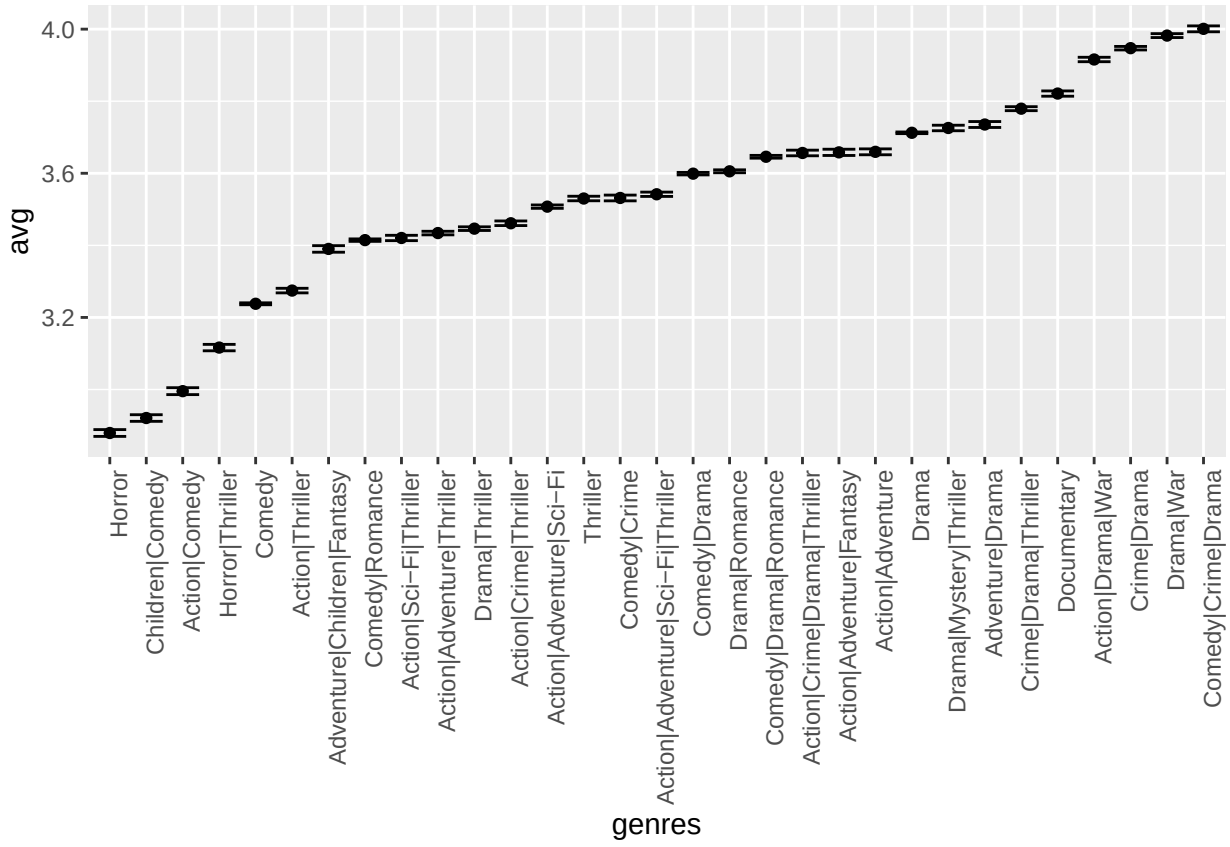

```
# See how many different genres there are, and the average rating for this genre
library(knitr)
edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n(), avg_rating = mean(rating)) %>%
  kable(caption = "Genre Check")
```

Table 1: Genre Check

genres	count	avg_rating
(no genres listed)	7	3.642857
Action	2560545	3.421405
Adventure	1908892	3.493544
Animation	467168	3.600644
Children	737994	3.418715
Comedy	3540930	3.436908
Crime	1327715	3.665925
Documentary	93066	3.783487
Drama	3910127	3.673131
Fantasy	925637	3.501946
Film-Noir	118541	4.011625
Horror	691485	3.269815
IMAX	8181	3.767693
Musical	433080	3.563305

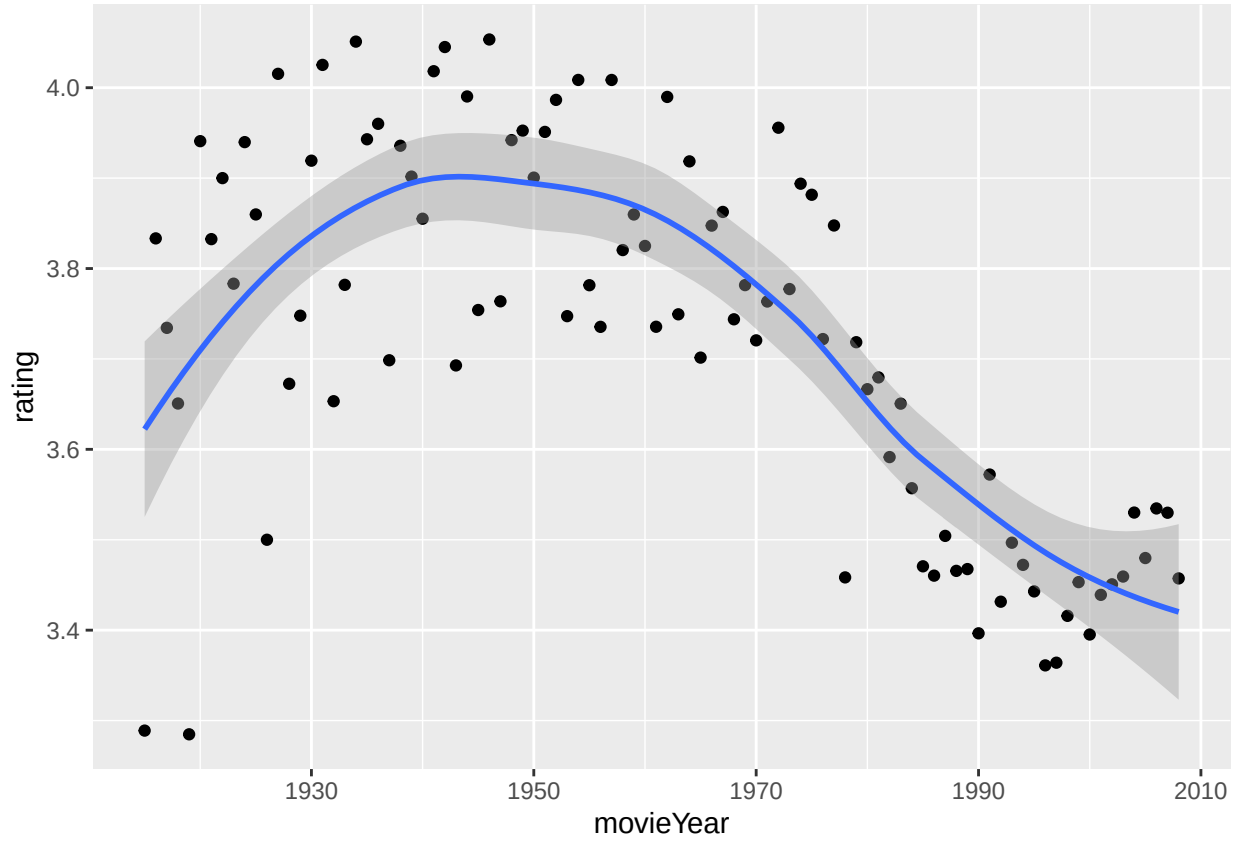
genres	count	avg_rating
Mystery	568332	3.677001
Romance	1712100	3.553813
Sci-Fi	1341183	3.395743
Thriller	2325899	3.507676
War	511147	3.780813
Western	189394	3.555918

```
# Plot average genre combination rating for genre combinations with more than 50,000 ratings
edx %>% group_by(genres) %>%
    summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
    filter(n >= 50000) %>%
    mutate(genres = reorder(genres, avg)) %>%
    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))
```

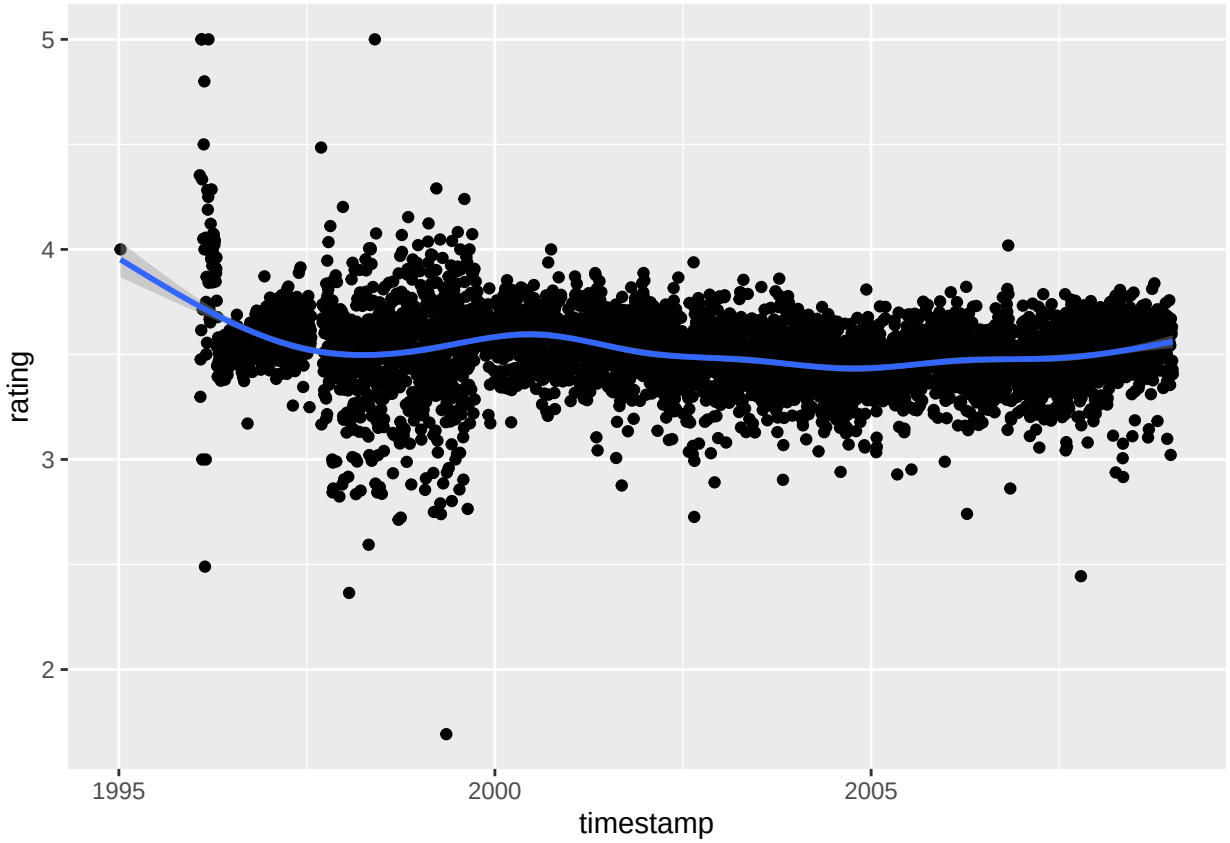



```
# Creates a new column in edx dataset for the movie release year
library(stringr)
edx$movieYear <- str_sub(edx$title, start = -6, end = -1)
edx$movieYear <- as.numeric(gsub("\\(|\\)", "", edx$movieYear))

# Plots the average movie rating per movie release year
```

```r
edx %>% group_by(movieYear) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(x = movieYear, y = rating)) +
  geom_point() +
  geom_smooth()
```




```r
# Changes timestamp into a date object and plots the average rating per timestamp date
edx$timestamp <- as_date(as_datetime(edx$timestamp))

edx %>% group_by(timestamp) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(x = timestamp, y = rating)) +
  geom_point() +
  geom_smooth()
```



From the graphs above it would seem that there are 4 main factors affecting the rating: The user effect, movie effect, genre effect, and release year effect. I also have examined the effect of the "timestamp" variable, but it seems that there is very little variation in the ratings when plotted against this variable over time. So, for this iteration of the regression model the "timestamp" variable will not be used.

After examining the above graphs I believe the best approach to create this model will be to use an iterative approach. I will start by creating the naive model below, and check it's RMSE:

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

This model is called a naive model as it assumes that all variation between the ratings is due to random chance. Since this is the most basic model possible, I will start here and add one variable per iteration and check the RMSE. Finally after adding in the 4 variables I will regularize my model to help further improve the accuracy and lessen any extreme ratings. The final model that I will aim to produce will be the below model:

$$Y_{u,m} = \mu + b_m + b_u + b_g + b_y + \varepsilon_{u,m}$$

```
set.seed(1)
test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)
```

```
train_edx <- edx[-test_index,]
test_edx_1 <- edx[test_index,]

# Ensure that the test data only contains data present in the train

test_edx <- test_edx_1 %>%
  semi_join(train_edx, by="userId") %>%
  semi_join(train_edx, by="movieId")

# Adds removed test rows to training data

removed_data <- anti_join(test_edx_1,test_edx)
train_edx <- rbind(train_edx,removed_data)

rm(removed_data, test_edx_1, test_index)

# Defines an RMSE & MAE function
RMSE <- function(ratings, pred_ratings){
  sqrt(mean((ratings-pred_ratings)^2))
}

MAE <- function(ratings, pred_ratings){
  mean(abs(ratings - pred_ratings))
}

goal_rmse <- 0.86490
```

In the below code, I will begin this process by splitting my data into training and test sets. I will also define the RMSE function used to judge this model along with the goal RMSE of 0.86490.

As mentioned previously I will now begin building my model. For the below regression model, I will be computing approximations of the effects to more efficiently create my model. The below model is the naive model which is the base for my model. To improve calculation times I will be calculating approximations of the 4 variable effects as opposed to using the lm() function.

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

```
# Finds training set average mu_hat

mu <- mean(train_edx$rating)
rmse_basic_model <- RMSE(test_edx$rating,mu)
mae_basic_model <- MAE(test_edx$rating,mu)
print(rmse_basic_model)
```

```
## [1] 1.059904
```

Adds in a variable for movies to account for movie by movie variability "b_m" :

$$Y_{u,m} = \mu + b_m + \varepsilon_{u,m}$$

```r
# Define b_m
avg_by_movie <- train_edx %>%
     group_by(movieId) %>%
     summarize(b_m = mean(rating - mu))

# Calculates the predicted ratings & RMSE using the Y_{u,m} = mu + b_m + e{u,m} model

predicted_ratings <- mu + test_edx %>%
  left_join(avg_by_movie, by = "movieId") %>%
  pull(b_m)

b_m_model_rmse <- RMSE(test_edx$rating,predicted_ratings)
b_m_model_mae <- MAE(test_edx$rating,predicted_ratings)

b_m_model_rmse
```

```
## [1] 0.9437429
```

```r
rm(predicted_ratings)
```

Adds a variable for users "b_u" to account for user by user variability :

$$Y_{u,m} = \mu + b_m + b_u + \varepsilon_{u,m}$$

```r
# Defines b_u estimate
avg_by_user <- train_edx %>%
  left_join(avg_by_movie, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_m))

# Calculates the predicted ratings & RMSE using the Y_{u,m} = mu + b_m + b_u + e{u,m} model
predicted_ratings <- test_edx %>%
  left_join(avg_by_movie, by = "movieId") %>%
  left_join(avg_by_user, by = "userId") %>%
  mutate(predicted_ratings = mu + b_m + b_u) %>%
  pull(predicted_ratings)

b_u_model_rmse <- RMSE(test_edx$rating,predicted_ratings)
b_u_model_mae <- MAE(test_edx$rating,predicted_ratings)

b_u_model_rmse
```

```
## [1] 0.8659319
```

```r
rm(predicted_ratings)
```

Adds a genre variable "b_g" to account for the genre variability by grouped genres :

$$Y_{u,m} = \mu + b_m + b_u + b_g + \varepsilon_{u,m}$$

```r
# Defines b_g estimate
avg_by_genre <- train_edx %>%
  left_join(avg_by_movie, by = "movieId") %>%
  left_join(avg_by_user, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_m - b_u))

# Calculates the predicted ratings & RMSE using the Y_{u,m} = mu + b_m + b_u + b_g + e{u,m} model
predicted_ratings <- test_edx %>%
  left_join(avg_by_movie, by = "movieId") %>%
  left_join(avg_by_user, by = "userId") %>%
  left_join(avg_by_genre, by = "genres") %>%
  mutate(predicted_ratings = mu + b_m + b_u + b_g) %>%
  pull(predicted_ratings)

b_g_model_rmse <- RMSE(test_edx$rating,predicted_ratings)
b_g_model_mae <- MAE(test_edx$rating,predicted_ratings)

b_g_model_rmse
```

```
## [1] 0.8655941
```

```r
rm(predicted_ratings)
```

Adds a release year variable "b_y" to account for the release year by release year variability :

$$Y_{u,m} = \mu + b_m + b_u + b_g + b_y + \varepsilon_{u,m}$$

```r
# Defines b_y estimate
avg_by_year <- train_edx %>%
  left_join(avg_by_movie, by = "movieId") %>%
  left_join(avg_by_user, by = "userId") %>%
  left_join(avg_by_genre, by = "genres") %>%
  group_by(movieYear) %>%
  summarise(b_y = mean(rating - mu - b_m - b_u - b_g))

## Calculates the predicted ratings & RMSE using the
# Y_{u,m} = mu + b_m + b_u + b_g + b_y + e{u,m} model
predicted_ratings <- test_edx %>%
  left_join(avg_by_movie, by = "movieId") %>%
  left_join(avg_by_user, by = "userId") %>%
  left_join(avg_by_genre, by = "genres") %>%
  left_join(avg_by_year, by = "movieYear") %>%
  mutate(predicted_ratings = mu + b_m + b_u + b_g + b_y) %>%
  pull(predicted_ratings)

b_y_model_rmse <- RMSE(test_edx$rating,predicted_ratings)
b_y_model_mae <- MAE(test_edx$rating,predicted_ratings)

b_y_model_rmse
```

```
## [1] 0.8654189
```

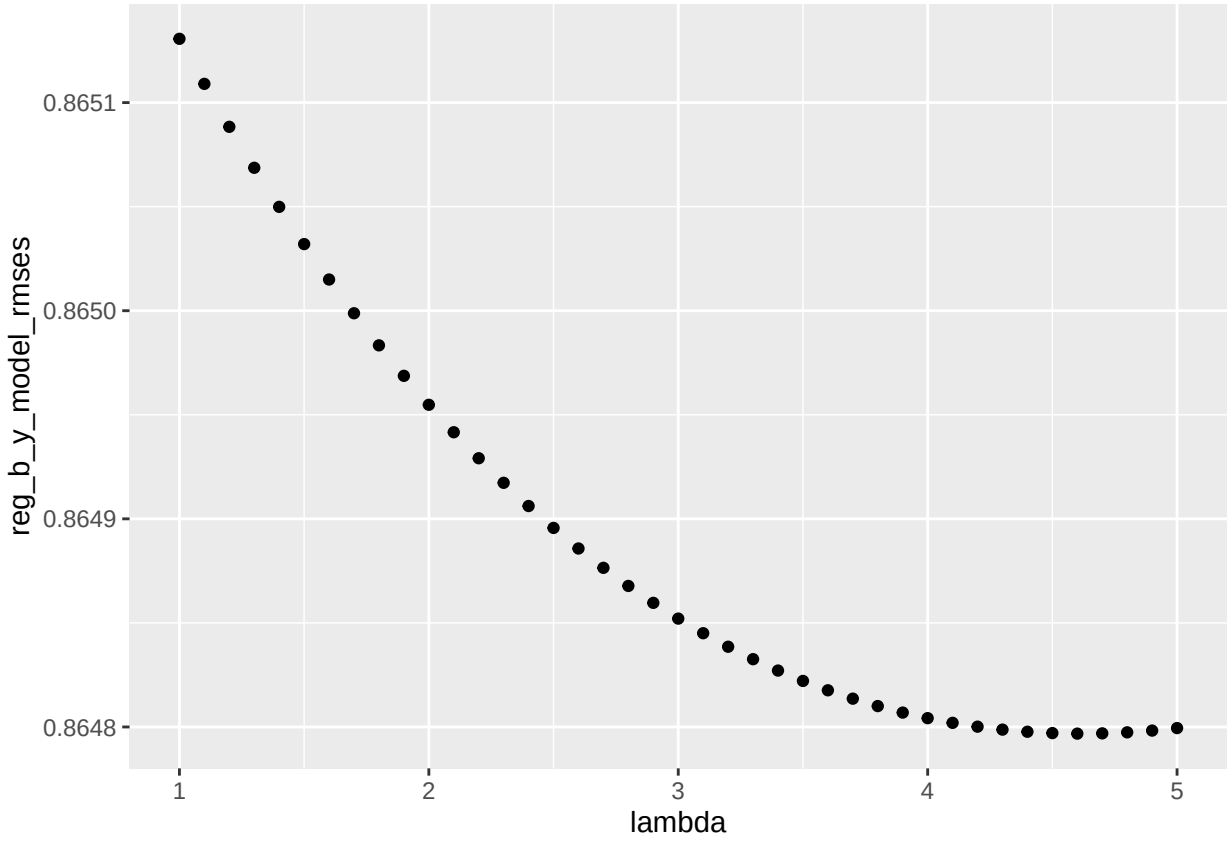
```
rm(predicted_ratings)
```

The model listed above uses all of the 4 features that I have found through my data analysis. This model's RMSE is 0.8654189, which is slightly above my goal of 0.86490.

To further improve the results of this regression model, I will now regularize the model to better account for any small sample sizes within my data set. This approach will allow me to penalize extreme errors that result from a small sample size, and therefore create a more accurate model.

```
# Define lambda, our tuning parameter
lambda <- seq(1,5, by = .1)

## Creates a function that uses all values of lambda in our final model
# Y_{u,m} = mu + b_m + b_u + b_g + b_y + e{u,m} to predict ratings and find RMSE
reg_b_y_model_rmses <- sapply(lambda, function(lambda){

  # redefines b_m
  avg_by_movie <- train_edx %>%
     group_by(movieId) %>%
     summarize(b_m = sum(rating - mu)/(n() + lambda))

  # Redefine b_u
  avg_by_user <- train_edx %>%
    left_join(avg_by_movie, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_m)/(n() + lambda))

  # Redefines b_g
  avg_by_genre <- train_edx %>%
    left_join(avg_by_movie, by = "movieId") %>%
    left_join(avg_by_user, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_m - b_u)/(n() + lambda))

  # Redefines b_y
  avg_by_year <- train_edx %>%
    left_join(avg_by_movie, by = "movieId") %>%
    left_join(avg_by_user, by = "userId") %>%
    left_join(avg_by_genre, by = "genres") %>%
    group_by(movieYear) %>%
    summarise(b_y = sum(rating - mu - b_m - b_u - b_g)/(n() + lambda))


  ## Calculates the predicted ratings & RMSE using the regularized
  # Y_{u,m} = mu + b_m + b_u + b_g + b_y + e{u,m} model
  predicted_ratings <- test_edx %>%
    left_join(avg_by_movie, by = "movieId") %>%
    left_join(avg_by_user, by = "userId") %>%
```

```
    left_join(avg_by_genre, by = "genres") %>%
    left_join(avg_by_year, by = "movieYear") %>%
    mutate(predicted_ratings = mu + b_m + b_u + b_g + b_y) %>%
    pull(predicted_ratings)

  reg_b_y_model_rmses <- RMSE(test_edx$rating,predicted_ratings)

  return(reg_b_y_model_rmses)
})

# Creates a graph of RMSE plotted against Lambda
ggplot() + geom_point(aes(x = lambda, y = reg_b_y_model_rmses))
```




```
# Finds the best RMSE & tuning parameter
best_lambda <- lambda[which.min(reg_b_y_model_rmses)]
best_lambda
```

```
## [1] 4.6
```

```
best_rmse <- min(reg_b_y_model_rmses)
best_rmse
```

```
## [1] 0.8647968
```

The multivariate regularized regression model seen above has achieved an RMSE of 0.8647968, which is under our goal RMSE. To examine all of our models and RMSEs' easily, I have created a table below showing all models along with their RMSEs'. This will allow us to easily see which RMSE is the lowest, and therefore the best fit.

```
RMSE_table <- tibble(Regression_Model =
                       c("rmse_basic_average_model", "b_m_model_rmse",
                         "b_u_model_rmse", "b_g_model_rmse",
                         "b_y_model_rmse", "regularized_b_y_model_rmse"),
       RMSE = c(rmse_basic_model, b_m_model_rmse, b_u_model_rmse,
                b_g_model_rmse, b_y_model_rmse, best_rmse),
       Diff_from_goal = c(goal_rmse - rmse_basic_model, goal_rmse - b_m_model_rmse,
                          goal_rmse - b_u_model_rmse, goal_rmse - b_g_model_rmse,
                          goal_rmse - b_y_model_rmse, goal_rmse - best_rmse))

kable(x = RMSE_table, caption = "RMSE Table")
```

A positive number in the "Difference_from_goal" column indicates that the RMSE is under my goal of 0.86490, meaning it is a passing model.

Table 2: RMSE Table

Regression_Model	RMSE	Diff_from_goal
rmse_basic_average_model	1.0599043	-0.1950043
b_m_model_rmse	0.9437429	-0.0788429
b_u_model_rmse	0.8659319	-0.0010319
b_g_model_rmse	0.8655941	-0.0006941
b_y_model_rmse	0.8654189	-0.0005189
regularized_b_y_model_rmse	0.8647968	0.0001032

From the above we can see that the regularized model has the lowest RMSE, and is under our target of 0.86490. Since I have found the best lambda parameter above, I will construct the previously mentioned regularized model using the entire edx data as the training set. I will then mutate the final_holdout_test data set so it matches the new edx data set (ie. add movieYear), and compare the predictions made to measure the true RMSE of this model.

Final model:

$$Y_{u,m} = \mu + b_m + b_u + b_g + b_y + \varepsilon_{u,m}$$

```
# Mutates final_holdout_test data to match the mutations made to the edx data set
final_holdout_test$movieYear <- str_sub(final_holdout_test$title, start = -6, end = -1)
final_holdout_test$movieYear <- as.numeric(gsub("\\(|\\)", "", final_holdout_test$movieYear))


############################################################
# Trains the regularized regression model on the edx data set
############################################################
```

```
# Y_{u,m} = mu + b_m + b_u + b_g + b_y + e{u,m}

# Redefines b_m
b_m <- edx %>%
   group_by(movieId) %>%
   summarize(b_m = sum(rating - mu)/(n() + best_lambda))

# Redefine b_u
b_u <- edx %>%
  left_join(avg_by_movie, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_m)/(n() + best_lambda))

# Redefines b_g
b_g <- edx %>%
  left_join(avg_by_movie, by = "movieId") %>%
  left_join(avg_by_user, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_m - b_u)/(n() + best_lambda))

# Redefines b_y
b_y <- edx %>%
  left_join(avg_by_movie, by = "movieId") %>%
  left_join(avg_by_user, by = "userId") %>%
  left_join(avg_by_genre, by = "genres") %>%
  group_by(movieYear) %>%
  summarise(b_y = sum(rating - mu - b_m - b_u - b_g)/(n() + best_lambda))


## Calculates the predicted ratings & RMSE using the regularized
# Y_{u,m} = mu + b_m + b_u + b_g + b_y + e{u,m} model
predicted_ratings <- final_holdout_test %>%
  left_join(b_m, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_y, by = "movieYear") %>%
  mutate(predicted_ratings = mu + b_m + b_u + b_g + b_y) %>%
  pull(predicted_ratings)

final_regularized_b_y_model_rmse <- RMSE(final_holdout_test$rating,predicted_ratings)
final_regularized_b_y_model_rmse
```

```
## [1] 0.8643351
```

Adds final RMSE to the RMSE_table

```
RMSE_table <- rbind(RMSE_table, data.frame(Regression_Model = "final_regularized_b_y_model_rmse",
                                           RMSE = final_regularized_b_y_model_rmse,
                                           Diff_from_goal = goal_rmse -
                                             final_regularized_b_y_model_rmse))
```

```
RMSE_table %>% kable(caption = "Final RMSE Table")
```

Table 3: Final RMSE Table

Regression_Model	RMSE	Diff_from_goal
rmse_basic_average_model	1.0599043	-0.1950043
b_m_model_rmse	0.9437429	-0.0788429
b_u_model_rmse	0.8659319	-0.0010319
b_g_model_rmse	0.8655941	-0.0006941
b_y_model_rmse	0.8654189	-0.0005189
regularized_b_y_model_rmse	0.8647968	0.0001032
final_regularized_b_y_model_rmse	0.8643351	0.0005649

Conclusion:

In this analysis, I have explored various regression models and the effectiveness in predicting ratings using RMSE., mainly through the use of 4 main variables: movie effect, user effect, genre effect, and movie release year effect. Through and iterative process of adding one variable at a time, I was able to achieve a final regularized model with the 4 aforementioned variables that yields an RMSE of 0.8643351.

This RMSE value suggests that, generally, my model's predicted ratings vary from users actual ratings by approximately 0.8643351, on a scale of 1-5. For this project while this RMSE is acceptable and reasonable, there is still potential to increase the accuracy of this model. I will list 3 potential ways to improve this model further:

1) To better examine the genre effect it could be beneficial in future iterations to split the genres into their smallest parts, instead of examining them as grouped. This would help to add more data points to our model and further reduce errors.

2) Explore non-linear models as opposed to the strictly linear models seen above. There are other models that work well with regression like the "Random Forest" model and the "Decision Tree" model that may help to catch extra complexities in this data.

3) Explore ensemble methods. This potential improvement pairs very well with improvement point number 2. Using an ensemble instead of only the multivariate regression model could potentially help to further increase the accuracy.

While there is still room for improvement, this regularized multivariate regression model is a strong foundation for predicting movie ratings and can be used for multiple real life situations in its current form. Also, the insights gained from this model can inform future research and models, helping to increase the speed and efficiency of model creation. Overall, this is a very strong recommendation model, and can currently be used to as a recommendation system or to help with user preference analysis.