Investigating the MovieLens Data Set

Rafid Akhyara Agung

10/15/2021

Introduction

In this project for the HarvardX Data Science course, the MovieLens data set was investigated. This data set contains around ten million ratings of movies, and the goal of this project was to create a method to predict these ratings based on the other present variables, being the userId, movieId, timestamp and genre.

In order to do so, the data set was split into a training and test set. Only the training set was used to train the model and the test set was used only for validation purposes. To calculate the effect of each variable, the mean residuals were calculated for each of them and to further improve the predictions, regularization was used to minimize the effect calculated if the sample size was small, and the parameter “lambda” was calculated using cross-validation.

Analysis: Data Exploration

First we obtain the data and create the training and test sets. This code was provided from the course. We also load the required packages.

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

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-proje
ct.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-pro
ject.org")
if(!require(dplyr)) install.packages("dplyr")
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(tidyr)) install.packages("tidyr")

library(tidyverse)
library(caret)
library(data.table)
library(dplyr)
library(ggplot2)
library(tidyr)

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

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

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

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

# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FA
LSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

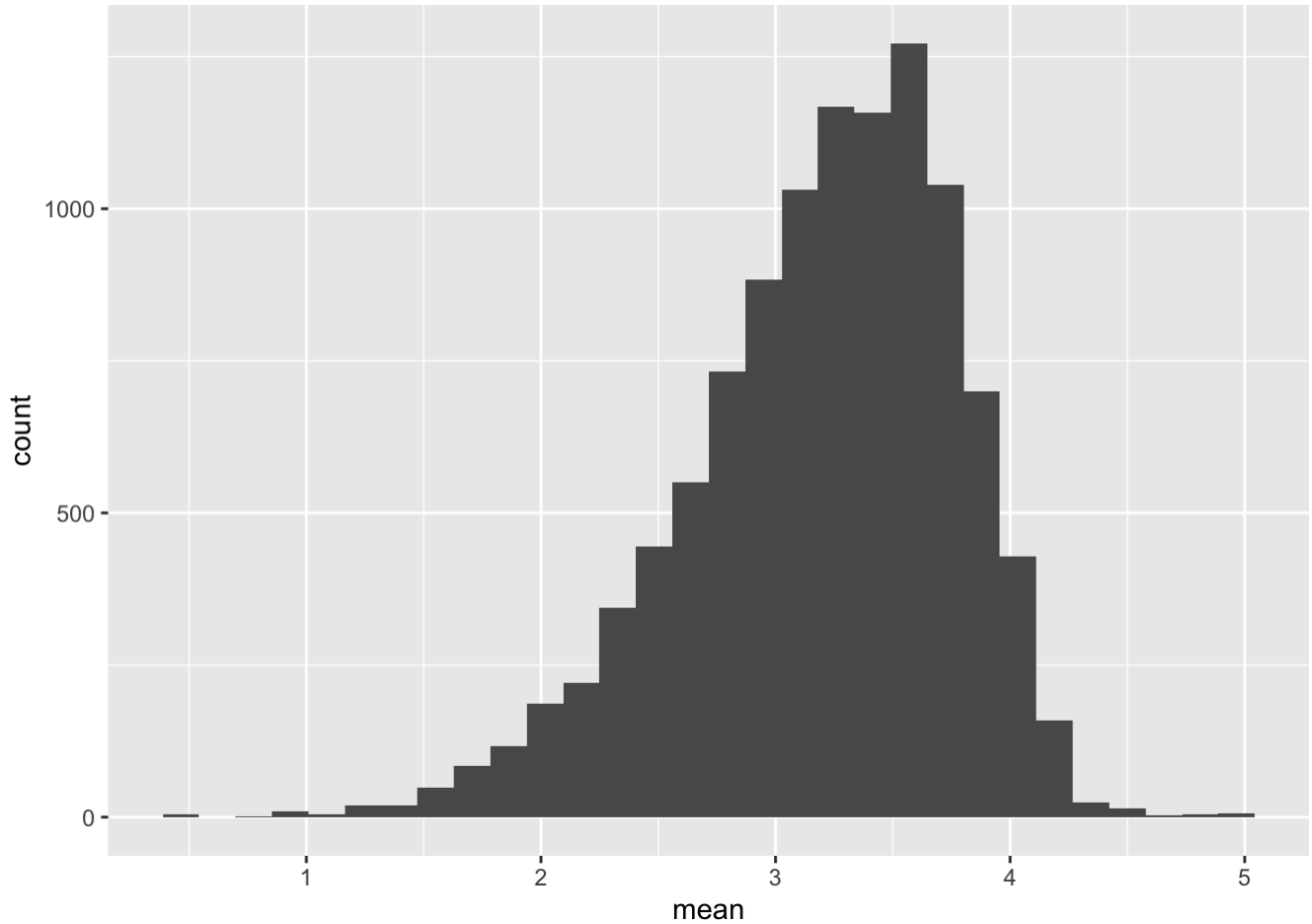
In order to explore the training set and choose which variables to analyze, we will create some visualizations.

First we calculate the mean.

```
mu <- mean(edx$rating)
```

Then we visualize the ratings of movies to determine if there is a movie effect.

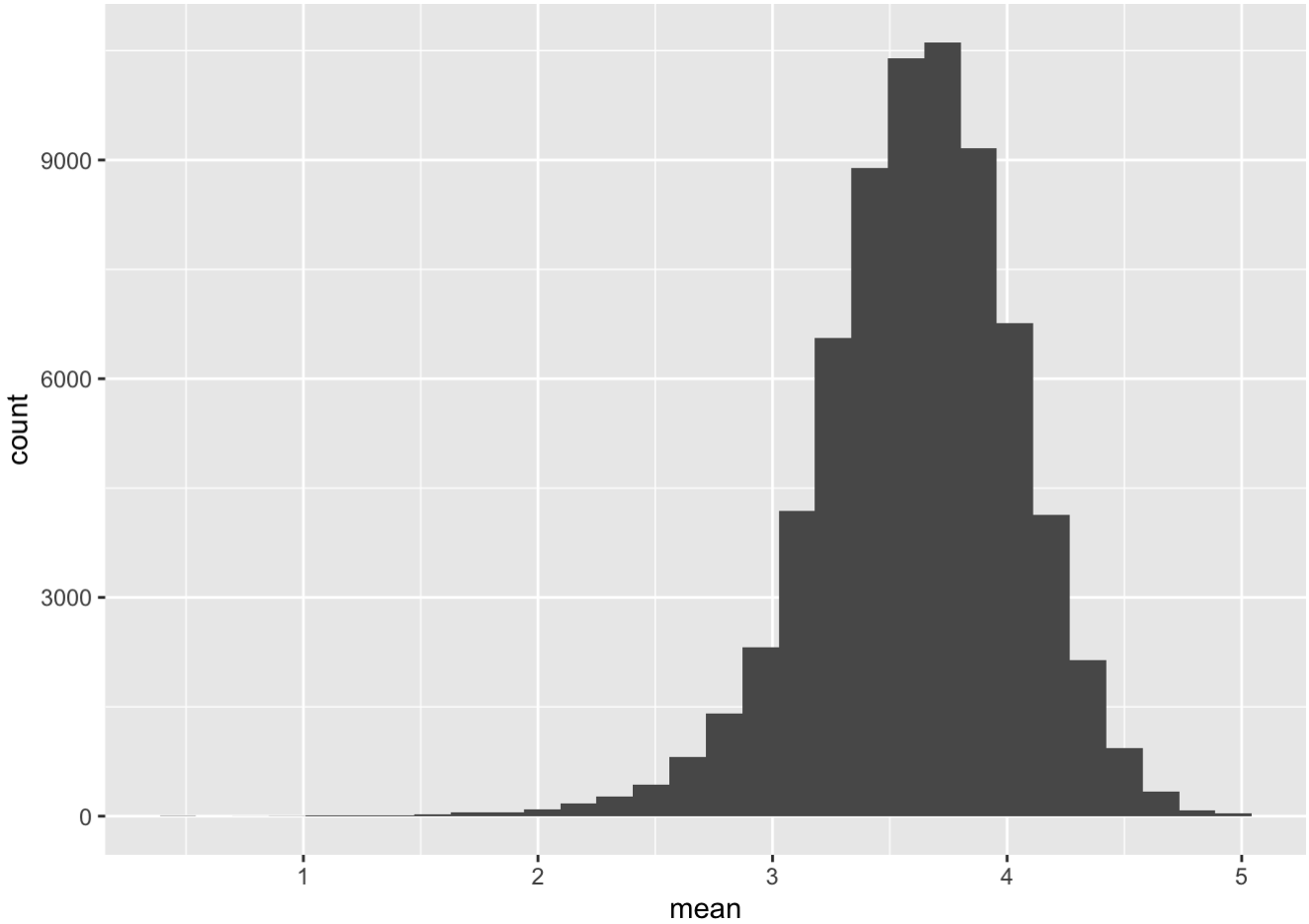
```
#Visualizing movie effect
edx %>%
  group_by(movieId) %>%
  summarize(mean = mean(rating)) %>%
  ggplot(aes(mean)) +
  geom_histogram(bins = 30)
```



As we can see, there is variance around the mean, therefore illustrating a movie effect that we will consider.

Next we visualize the ratings from users to determine if there is a user effect.

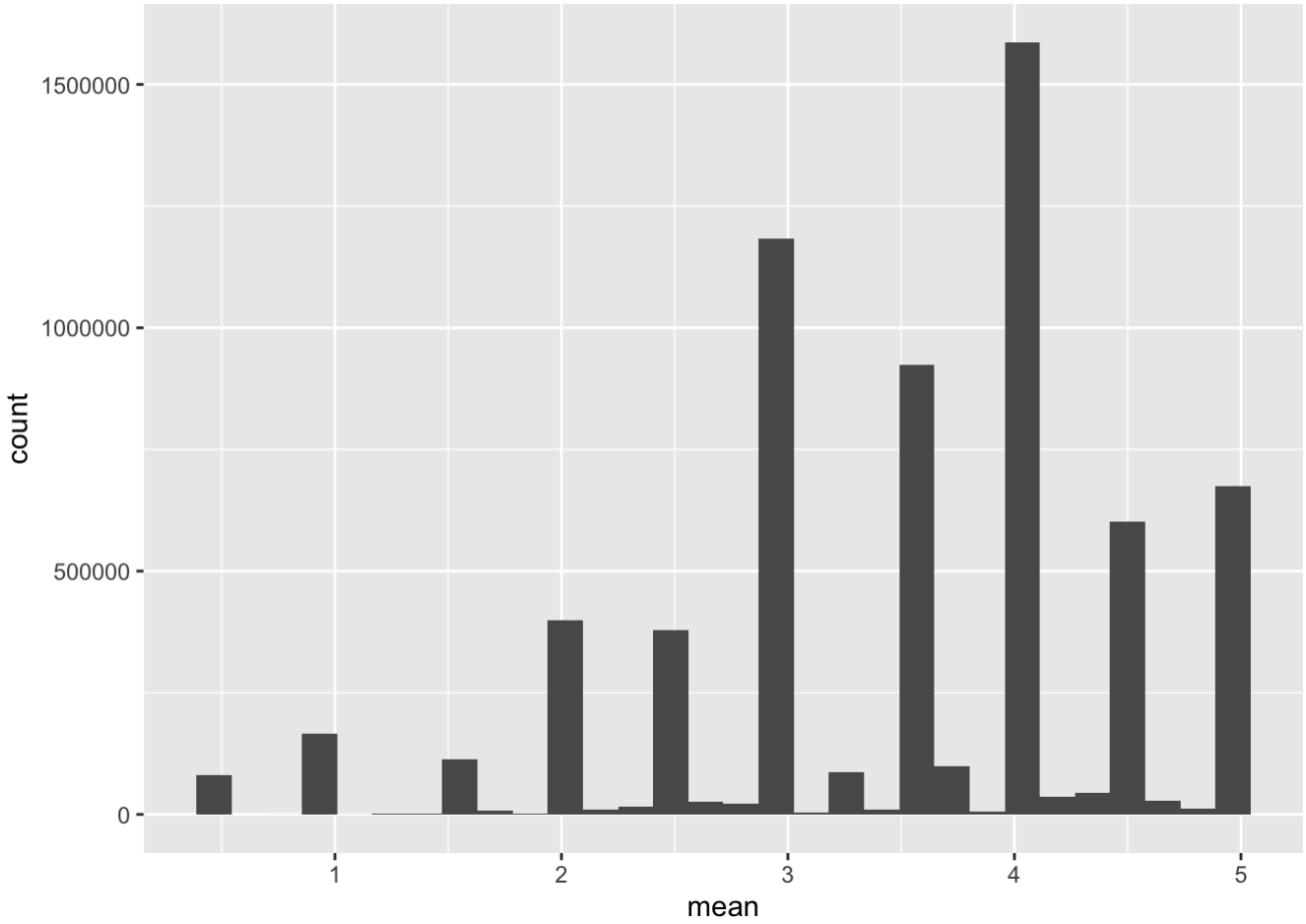
```
#Visualizing user effect
edx %>%
  group_by(userId) %>%
  summarize(mean = mean(rating)) %>%
  ggplot(aes(mean)) +
  geom_histogram(bins = 30)
```



As we can see, there is variance around the mean, therefore illustrating a user effect that we will consider.

Next we visualize the ratings for time stamps to determine if length of a movie effects rating.

```
#Visualizing timestamp effect

edx %>%
  group_by(timestamp) %>%
  summarize(mean = mean(rating)) %>%
  ggplot(aes(mean)) +
  geom_histogram(bins = 30)
```



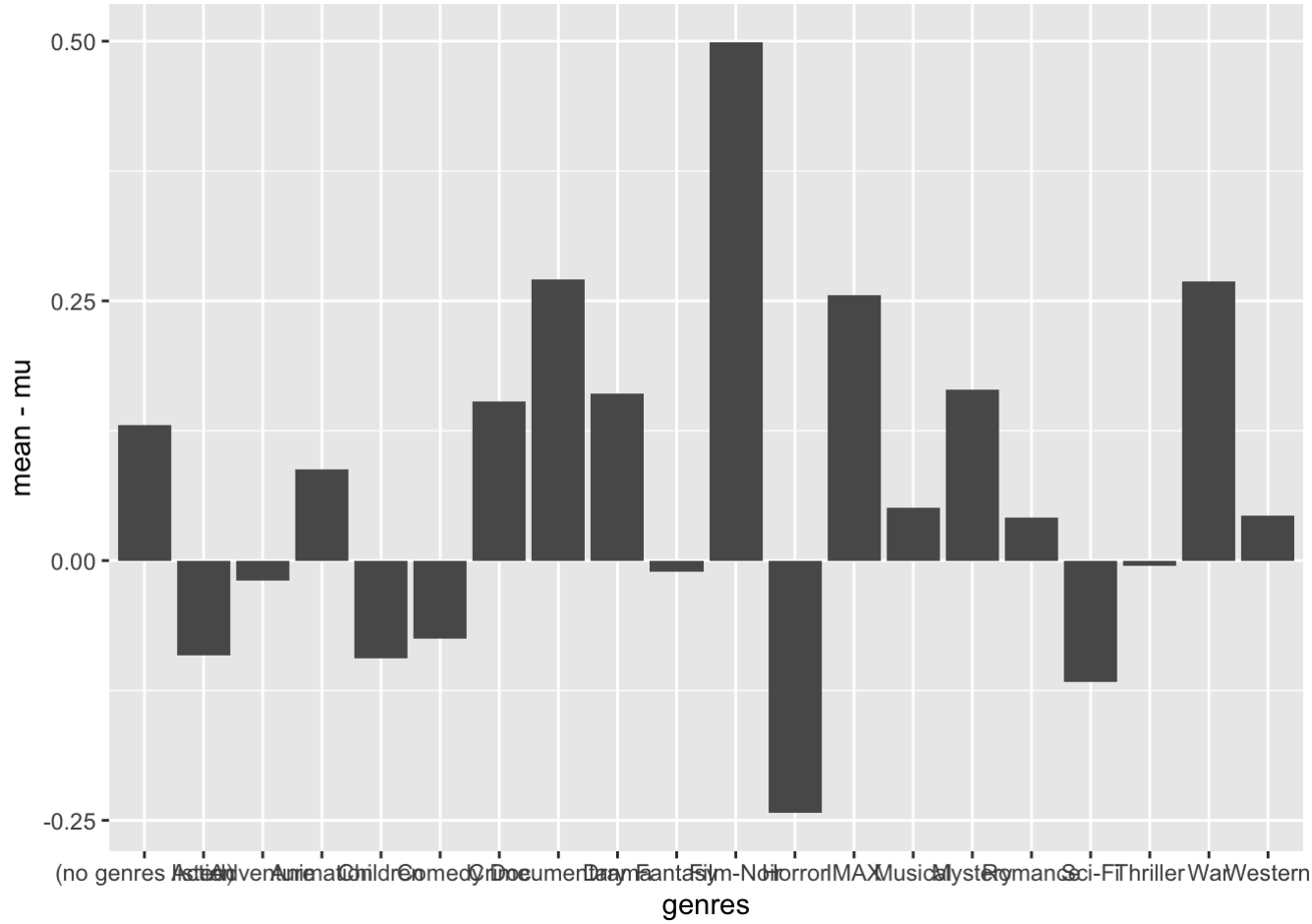

```
tscheck <- edx %>%
  group_by(timestamp) %>%
  summarize(mean = mean(rating), n = n())
mean(tscheck$n)
```

```
## [1] 1.380463
```

Here, though we see variance, we observe that the distribution is not smooth and concentrates at the whole-numbered ratings. This is because, if we observe the mean sample size for each timestamp, it is very low. Because of this, it will be omitted from the model.

Next we visualize the ratings of genres to determine if there is a genre effect.

```
#Splitting the genres
edx_genres_split <- edx %>%
  separate_rows(genres,
                sep = "\\|",
                convert = TRUE)

#Visualizing genre effect
edx_genres_split %>%
  group_by(genres)%>%
  summarize(mean = mean(rating))%>%
  ggplot()+
  geom_col(aes(genres, mean-mu))
```

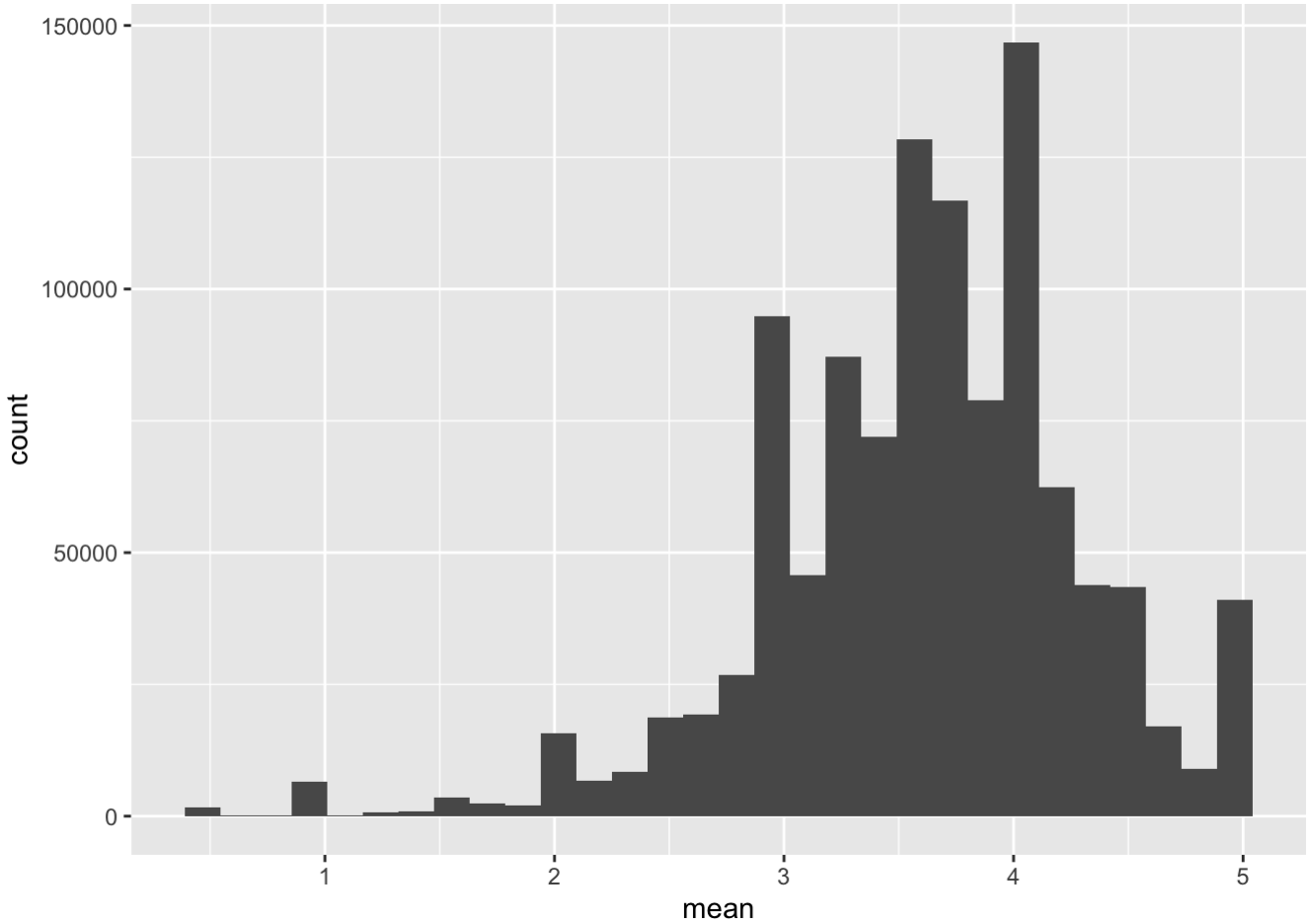


As we can see, some genres do more or worse than average, illustrating a genre effect.

Next we visualize the ratings of users for certain genres to determine if there is an effect from user's genre preference.

```
#Visualizing user's genre preference effect
edx_genres_split %>%
  group_by(genres, userId)%>%
  summarize(mean = mean(rating))%>%
  ggplot(aes(mean)) +
  geom_histogram(bins = 30)
```

```
## `summarise()` has grouped output by 'genres'. You can override using the `.groups`
argument.
```



As we can see, some users have preference/dislike for a genre due to the variance around the mean.

Analysis: Forming the Model

Before we start forming our model, we need to create a function to evaluate the RMSE. And also modify the test set so that its genres are split as well.

```
#Defining the RMSE function
rmse <- function(true_ratings, predicted_ratings) {
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

#Splitting genres in the test set
temp_genres_split <- temp %>%
  separate_rows(genres,
                sep = "\\|",
                convert = TRUE)
```

From our visualization, we are going to include four effects in our model, the movie effect, user effect, genre effect and genre-user effect. The type of model used will be a linear model.

Since the data set is too large to perform the “lm()” function without crashing on most computers, we will count the mean residuals instead for each effect.

After the addition of each effect, we will count the RMSE to evaluate the effectiveness of the algorithm.

```
#Effect of user
u_avgs <- edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu))

#Effect of movies
m_avgs <- edx %>%
  left_join(u_avgs, by = "userId")%>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu - b_u))

#Effect of Genres
gs_avgs <- edx_genres_split %>%
  left_join(m_avgs, by = "movieId") %>%
  left_join(u_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_gs = mean(rating - mu - b_i - b_u))

#Effect of users' genre preferences
gu_avgs <- edx_genres_split %>%
  left_join(m_avgs, by = "movieId") %>%
  left_join(u_avgs, by = "userId") %>%
  left_join(gs_avgs, by = "genres") %>%
  group_by(genres, userId) %>%
  summarize(b_gu = mean(rating - mu - b_i - b_u - b_gs))
```

```
## `summarise()` has grouped output by 'genres'. You can override using the `.groups`
argument.
```

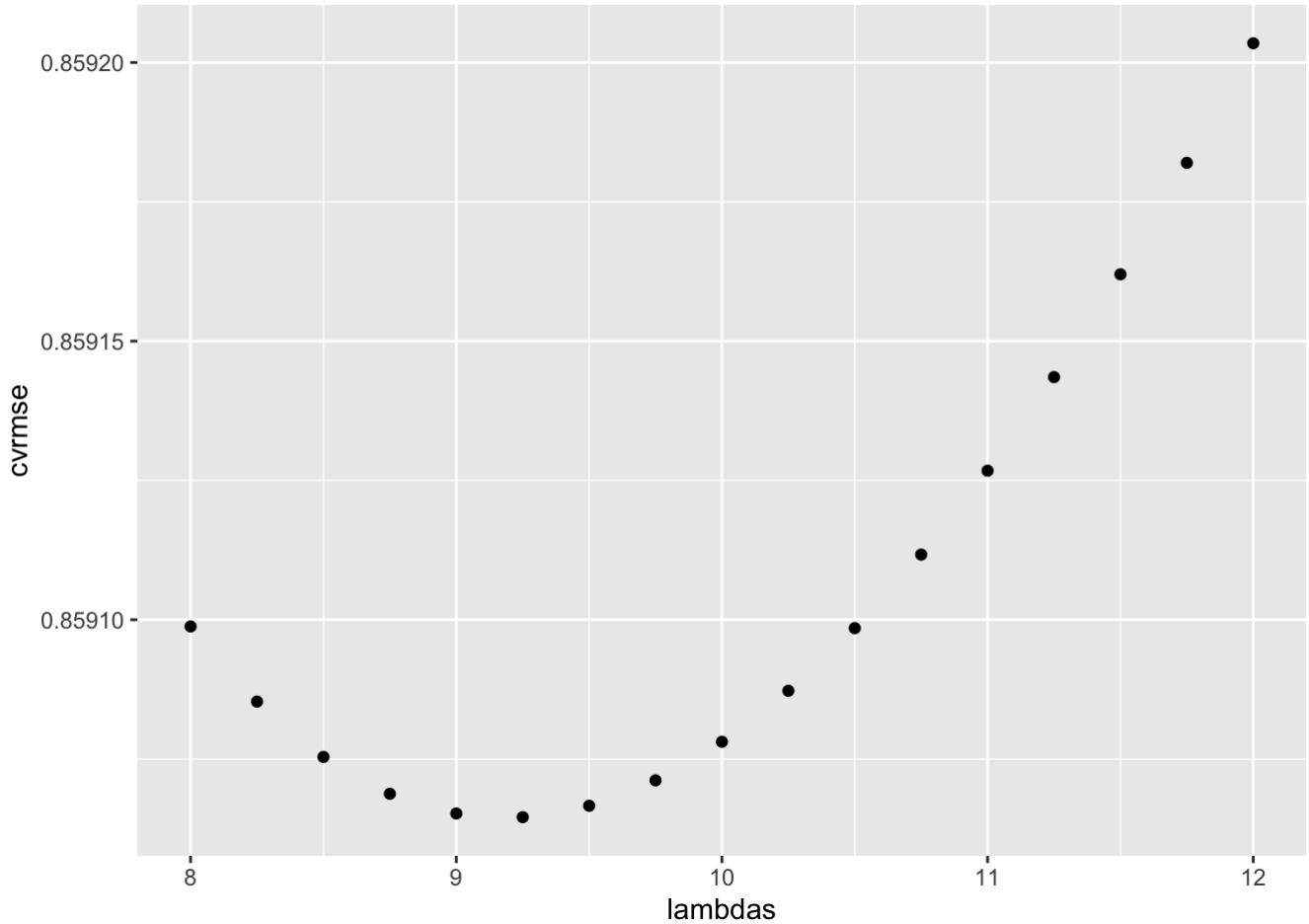
```
predicted_ratings1 <- temp %>%
  left_join(u_avgs, by='userId') %>%
  mutate(
    b_u = ifelse(is.na(b_u), 0, b_u),
    pred = mu + b_u) %>%
  .$pred
rmse_results <- data.frame(Method = "User Effect", RMSE = rmse(predicted_ratings1, te
mp$rating))

predicted_ratings2 <- temp %>%
  left_join(m_avgs, by='movieId') %>%
  left_join(u_avgs, by='userId') %>%
  mutate(
    b_u = ifelse(is.na(b_u), 0, b_u),
    b_i = ifelse(is.na(b_i), 0, b_i),
    pred = mu + b_i + b_u) %>%
  .$pred

rmse_results <- bind_rows(rmse_results,
                          data.frame(Method = "User + Movie Effect",
                                     RMSE = rmse(predicted_ratings2, temp$rating)))

predicted_ratings3 <- temp_genres_split %>%
  left_join(m_avgs, by='movieId') %>%
  left_join(u_avgs, by='userId') %>%
  left_join(gs_avgs, by = "genres") %>%
  mutate(
    b_u = ifelse(is.na(b_u), 0, b_u),
    b_i = ifelse(is.na(b_i), 0, b_i),
    b_gs = ifelse(is.na(b_gs), 0, b_gs),
    pred = mu + b_i + b_u + b_gs) %>%
  .$pred

rmse_results <- bind_rows(rmse_results,
                          data.frame(Method = "User + Movie + Genre Effect",
                                     RMSE = rmse(predicted_ratings3, temp_genres_spli
t$rating)))

predicted_ratings4 <- temp_genres_split %>%
  left_join(m_avgs, by='movieId') %>%
  left_join(u_avgs, by='userId') %>%
  left_join(gs_avgs, by = "genres") %>%
  left_join(gu_avgs, by = c("genres", "userId")) %>%
  mutate(
    b_u = ifelse(is.na(b_u), 0, b_u),
    b_i = ifelse(is.na(b_i), 0, b_i),
    b_gs = ifelse(is.na(b_gs), 0, b_gs),
    b_gu = ifelse(is.na(b_gu), 0, b_gu),
    pred = mu + b_i + b_u + b_gs + b_gu) %>%
  .$pred

rmse_results <- bind_rows(rmse_results,
                          data.frame(Method = "User + Movie + Genre + Genre/User Effe
ct",
                                     RMSE = rmse(predicted_ratings4, temp_genres_spli
t$rating)))
rmse_results %>% knitr::kable()
```

Method	RMSE
User Effect	0.9783384
User + Movie Effect	0.8816060
User + Movie + Genre Effect	0.8802738
User + Movie + Genre + Genre/User Effect	0.8676553

As we can see, the RMSE decreases with the addition of each variable. But to further improve accuracy, we can use regularization, to minimize the effect from small sample sizes. In the regularization process, we use cross validation to calculate the optimal “lambda” value to use

```
#Creating a vector of lambdas
lambdas <- seq(8, 12, 0.25)

#Partitioning the training set for cross-validation
set.seed(1, sample.kind="Rounding")
test_indexcv <- createDataPartition(y = edx$rating,
                                    times = 1,
                                    p = 0.2,
                                    list = FALSE)

edxcv <- edx[-test_indexcv,]
tempcv <- edx[test_indexcv,]

#Splitting genres for these sets
edx_genres_splitcv <- edxcv %>%
  separate_rows(genres,
                sep = "\\|",
                convert = TRUE)

temp_genres_splitcv <- tempcv %>%
  separate_rows(genres,
                sep = "\\|",
                convert = TRUE)

#Running the cross-validation to determine the optimum lambda
cvrmse <- sapply(lambdas, function(l){

  #Average of the ratings
  mucv <- mean(edxcv$rating)

  #Effect of user
  u_avgscv <- edxcv %>%
    group_by(userId) %>%
    summarize(b_u = (sum(rating - mucv)/ (n() + l)))

  #Effect of movies
  m_avgscv <- edxcv %>%
    left_join(u_avgscv, by = "userId")%>%
    group_by(movieId) %>%
    summarize(b_i = (sum(rating - mucv - b_u)/ (n() + l)))

  #Effect of Genres
  gs_avgscv <- edx_genres_splitcv %>%
    left_join(m_avgscv, by = "movieId") %>%
    left_join(u_avgscv, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_gs = (sum(rating - mucv - b_i - b_u)/ (n() + l)))

  #Effect of users' genre preferences
  gu_avgscv <- edx_genres_splitcv %>%
    left_join(m_avgscv, by = "movieId") %>%
    left_join(u_avgscv, by = "userId") %>%
    left_join(gs_avgscv, by = "genres") %>%
    group_by(genres, userId) %>%
    summarize(b_gu = (sum(rating - mucv - b_i - b_u - b_gs)/ (n() + l)))

  predicted_ratingscv <- temp_genres_splitcv %>%
```

```
    left_join(m_avgscv, by='movieId') %>%
    left_join(u_avgscv, by='userId') %>%
    left_join(gs_avgscv, by = "genres") %>%
    left_join(gu_avgscv, by = c("genres", "userId"))%>%
    mutate(
      b_u = ifelse(is.na(b_u), 0, b_u),
      b_i = ifelse(is.na(b_i), 0, b_i),
      b_gs = ifelse(is.na(b_gs), 0, b_gs),
      b_gu = ifelse(is.na(b_gu), 0, b_gu),
      pred = mucv + b_i + b_u + b_gs + b_gu) %>%
    .$pred

  return(rmse(predicted_ratingscv, temp_genres_splitcv$rating))
})
qplot(lambdas,cvrmse)
```




```
lambda <- lambdas[which.min(cvrmse)]
lambda
```

```
## [1] 9.25
```

Results

Our final RMSE will be the RMSE from the regularized model.

```
#Average of the ratings
mu <- mean(edx$rating)

#Effect of user
u_avgs <- edx %>%
  group_by(userId) %>%
  summarize(b_u = (sum(rating - mu)/ (n() + lambda)))

#Effect of movies
m_avgs <- edx %>%
  left_join(u_avgs, by = "userId")%>%
  group_by(movieId) %>%
  summarize(b_i = (sum(rating - mu - b_u)/ (n() + lambda)))

#Effect of Genres
gs_avgs <- edx_genres_split %>%
  left_join(m_avgs, by = "movieId") %>%
  left_join(u_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_gs = (sum(rating - mu - b_i - b_u)/ (n() + lambda)))


#Effect of users' genre preferences
gu_avgs <- edx_genres_split %>%
  left_join(m_avgs, by = "movieId") %>%
  left_join(u_avgs, by = "userId") %>%
  left_join(gs_avgs, by = "genres") %>%
  group_by(genres, userId) %>%
  summarize(b_gu = (sum(rating - mu - b_i - b_u - b_gs)/ (n() + lambda)))
```

```
## `summarise()` has grouped output by 'genres'. You can override using the `.groups`
argument.
```

```
predicted_ratings <- temp_genres_split %>%
  left_join(m_avgs, by='movieId') %>%
  left_join(u_avgs, by='userId') %>%
  left_join(gs_avgs, by = "genres") %>%
  left_join(gu_avgs, by = c("genres", "userId"))%>%
  mutate(
      b_u = ifelse(is.na(b_u), 0, b_u),
      b_i = ifelse(is.na(b_i), 0, b_i),
      b_gs = ifelse(is.na(b_gs), 0, b_gs),
      b_gu = ifelse(is.na(b_gu), 0, b_gu),
      pred = mu + b_i + b_u + b_gs + b_gu) %>%
  .$pred

rmse_results <- bind_rows(rmse_results,
                              data_frame(Method = "Regularized User + Movie + Genre
 + Genre/User                                  Effect",
                               RMSE = rmse(predicted_ratings, temp_genres_split$rati
ng)))
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
```

```
rmse_results %>% knitr::kable()
```

Method	RMSE
User Effect	0.9783384
User + Movie Effect	0.8816060
User + Movie + Genre Effect	0.8802738
User + Movie + Genre + Genre/User Effect	0.8676553
Regularized User + Movie + Genre + Genre/User Effect	0.8567081

We see that regularization has done well to reduce the RMSE further and our model can predict the ratings to a very sufficient level of accuracy.

Conclusion

In this project, a model was designed for the MovieLens data set. Through consideration of the user, movie, genre and genre/user effect and regularization of these factors, the model was able to perform sufficiently in predicting ratings.

In order to improve the performance, more factors could be included such as the demographics of the users or the people involved in the movie (e.g. demographics of actors, directors, etc).