```
#INTRODUCTION/SUMMARY
#This project is to complete the Movielens capstone assignment for EDX Datascience Program.
#I tried to run a variety of algorithms but none would work with my computer. I opted for a very
#simple approach that I found online: Penalized Least Squares Regression. I applied this to a
number of variables, but got the best RMSE with rating/userId. This method assumes that fewer
ratings per user produces more volatility.
#I ran the PLSR on a number of variables but the userId, ratings match returned the lowest

#Download edx dataset

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

ratings <- read.table(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")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[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) # if using R 3.6.0: set.seed(1, sample.kind = "Rounding")
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 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)

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

#Explore data
```

```
edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))

edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
#Number of columns in edx set:
ncol(edx)
#Number of rows in edx set:
nrow(edx)
#How many films have a rating of 0?
sum(edx$rating==0)
#How many films have a rating of 3?
sum(edx$rating == 3)
#How many distinct films are in edx?
edx %>% summarize(n_movies = n_distinct(movieId))
#How many distinct users are there?
edx %>% summarize(n_users = n_distinct(userId))
#How many ratings for each of these genres?
Drama <- edx %>% filter(str_detect(genres,"Drama"))
Comedy <- edx %>% filter(str_detect(genres,"Comedy"))
Thriller <- edx %>% filter(str_detect(genres,"Thriller"))
Romance <- edx %>% filter(str_detect(genres,"Romance"))
#What are the five most given ratings in order from most to least?
edx %>% group_by(title) %>% summarise(number = n()) %>%
  arrange(desc(number))
head(sort(-table(edx$rating)),5)
#Half star ratings are less common than whole ratings:
table(edx$rating)
#Convert timestamp to datetime (for future use)

edx <- edx %>% mutate(timestamp = as.POSIXct(timestamp, origin = "1970-01-01",
                                             tz = "GMT"))
edx$timestamp <- format(edx$timestamp, "%Y")

colnames(edx)

names(edx)[names(edx) == "timestamp"] <- "year_rated"

releaseyear <- stringi::stri_extract(edx$title, regex = "(\\d{4})", comments = TRUE) %>%
  as.numeric()

edx <- edx %>% mutate(release_year = releaseyear)
```

```
validation <- validation %>% mutate(timestamp = as.POSIXct(validation$timestamp,
                                    origin = "1970-01-01", tz = "GMT"))
validation$timestamp <- format(validation$timestamp, "%Y")

colnames(validation)
names(validation)[names(validation) == "timestamp"] <- "year_rated"

releaseyear2 <- stringi::stri_extract(validation$title, regex = "(\\d{4})",
                         comments = TRUE) %>% as.numeric()

validation <- validation %>% mutate(release_year = releaseyear2)
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

Lambdas <- seq(0, 5, 0.25)

Rmses <- sapply(lambdas,function(l){

  #Determine the mean of ratings from the edx training set
  mu <- mean(edx$rating)

  #Adjust with low ratings
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))


  #Use the training set to find the best predicted lambda
  predictedRating<-
    edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(predictedRating,edx$rating))
})

lambda <-Lambdas[which.min(Rmses)]
paste('The best Lambda of',min(Rmses),'is with Lambda',lambda)
```

```
#RESULTS

#Run the same program with best lambda
lambda <- 0.5

predict_a<- sapply(lambda,function(l){

  #Find the mean from the training set
  mu <- mean(edx$rating)

  #Find movie effect with best lambda
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  #Find user effect with best lambda
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  #Predict the ratings on the validation set
  predictedRating<-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred #validation

  return(predictedRating)

})
#CONCLUSION:
#The Penalized Least Squares Regression does indeed produce an acceptable RMSE. Overall,
however, this is a lame machine learning exercise. It isn’t efficient to run various algorithms, and
the assignment doesn’t ask to compare accuracy (which really is the point of machine learning,
isn’t it?).
```