MovieLensCapstone_JHW

John Wilson

8/13/2020

Executive Summary:

This report is focused on predicting movie ratings based upon the variables available in the dataset provided. Being able to predict ratings is of benefit both to streaming services and to the actual viewer so that time is not wasted watching something that would not be of interest. Also it allows the streaming service to provide content that lines up better with the viewers' preferences. The data was cleaned up prior to the beginning of the analysis but includes 10 million ratings for over 10,000 movies utilizing 70,000 users and nearly 800 genres. 10% of this data was reserved for validation of the models that would be formed through out this process. The remaining data was further divided into train and test sets to confirm models prior to validation. 8 models were trained on the data and proved Matrix Factorization to be the most accurate model with a root mean square error of 0.797 on the test set. That model was then applied to the validation set yielding a final RMSE of 0.797 which is below the target of 0.8649.

Data Preparation:

Prior to importing the data, the required packages for processing were checked for and if not present were installed then loaded. Then the data was imported from the grouplens organization and was modified into tidy format with the variables of: movieId, userId, title, timestamp, rating and genres. To prevent errors later in the project once the 10% cut of data was designated for validation, it was confirmed that movieId and userId would exist in both data sets in order for models to be applicable across the sets.

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

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")
if(!require(data.table)) install.packages("data.table", repos =
"http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos =
"http://cran.us.r-project.org")
if(!require(recosystem)) install.packages("recosystem", repos =
"http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-
project.org")
```

```r
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(recosystem)
library(knitr)

# 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 3.6 or earlier
movies <- as.data.frame(movies) %>% mutate(movieId =
as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(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")

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
sampler
## used

# if using R 3.5 or earlier, use `set.seed(1)` instead
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)
```

Data Review:

Below, it can be seen the structure of the edx data set that will be used to form and test models prior to validation. Note the variables, their class and the number of observations.

```
str(edx)

## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392
838984474 838983653 838984885 838983707 838984596 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)"
"Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller"
"Action|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

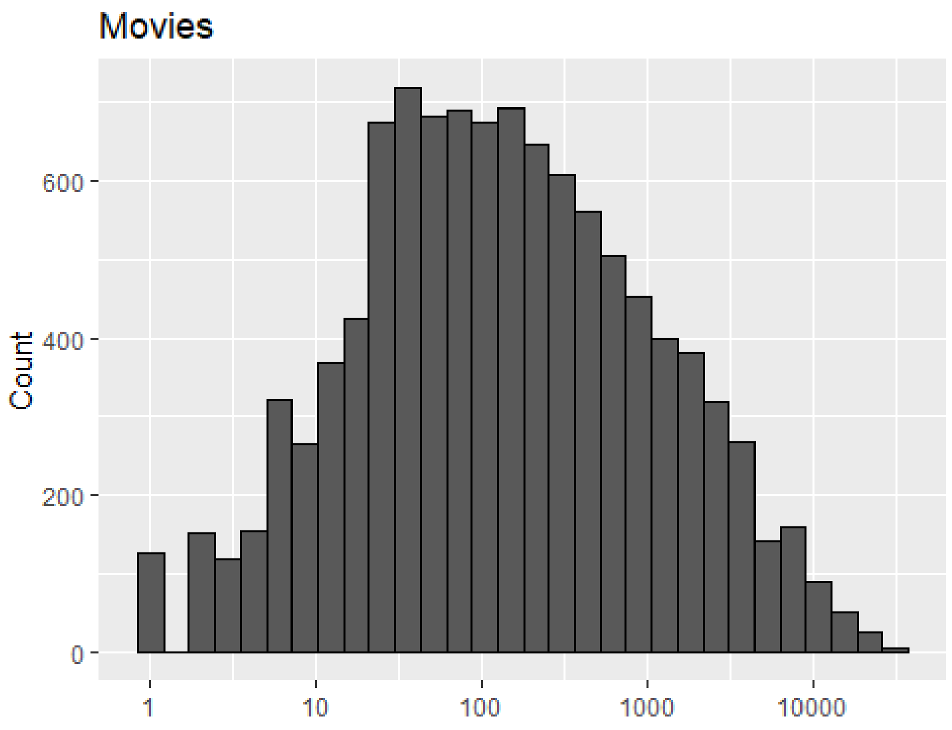
Prior to performing calculations, it is best to know if there are any NA values that cause potential issues. With the code below, it is determined there are none.

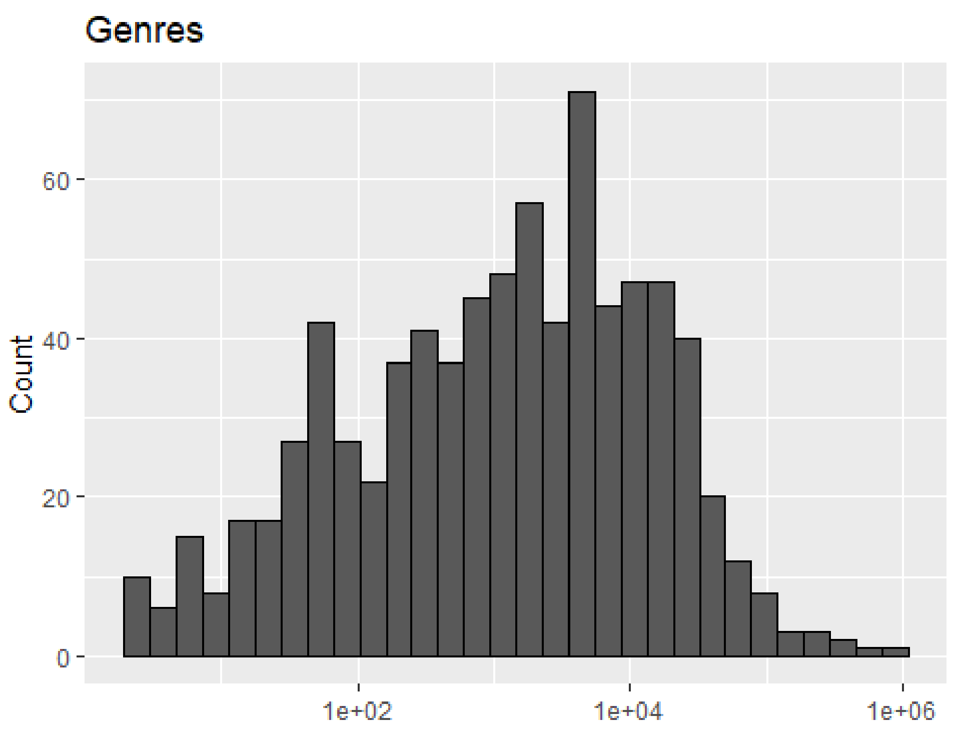
```
sum(is.na(edx))

## [1] 0
```

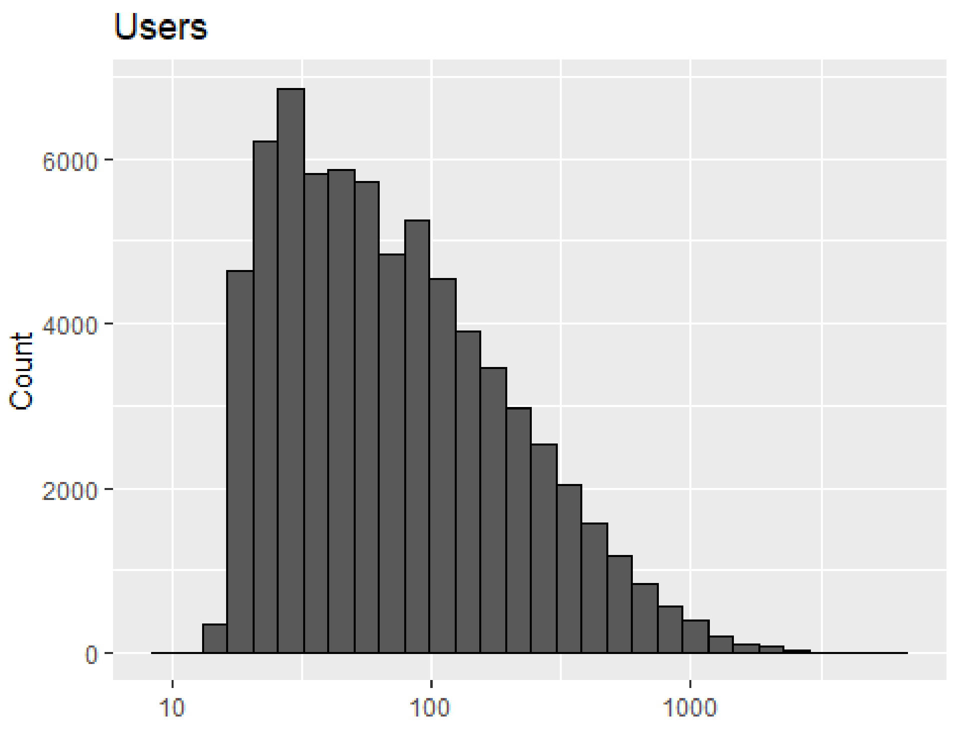
To get an idea of unique items within the variables the following code was ran.

```
unique_variables <- data_frame(Variable=c("Movies","Users","Genres"),
          Count=c(length(unique(edx$movieId)),length(unique(edx$userId)),
                  length(unique(edx$genres))))

unique_variables

## # A tibble: 3 x 2
##   Variable Count
##   <chr>    <int>
## 1 Movies   10677
## 2 Users    69878
## 3 Genres     797
```

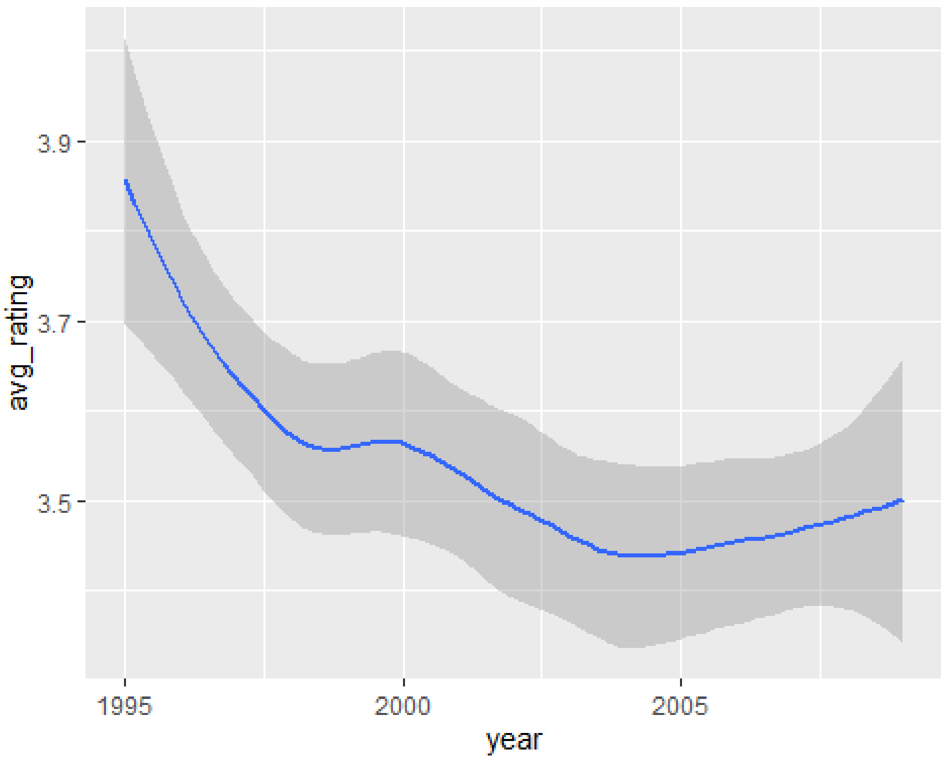
Now with a deeper look into the data, the number of ratings very across movieId, genres, and userId. 3 plots for each of the items were created. To not overload the x axis with labels, histogram bins were used just to display the variation in counts.



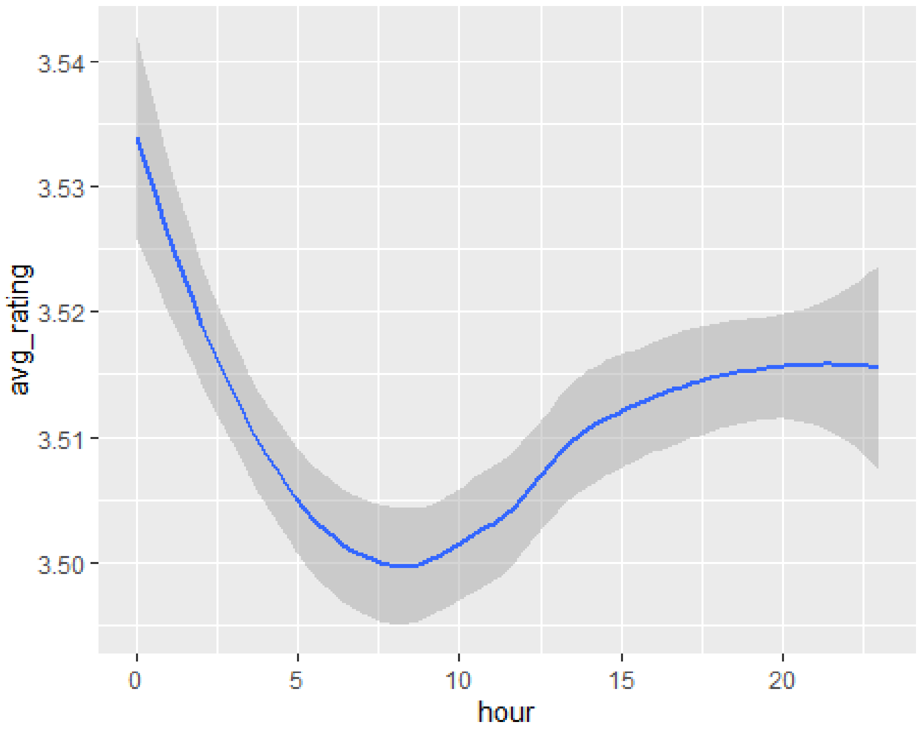




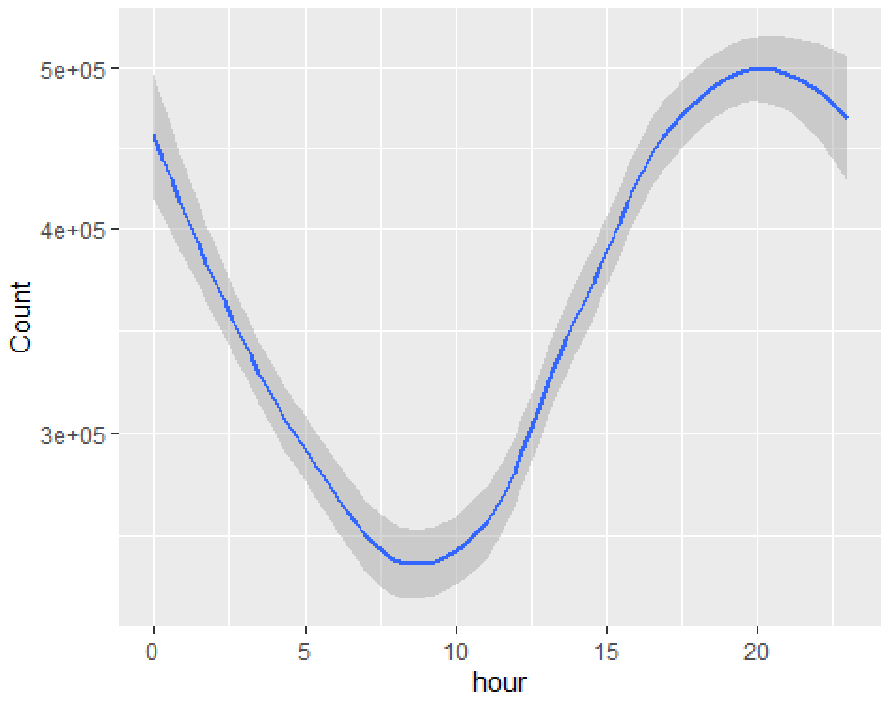
Further investigation shows that harsher ratings have been giving over the years with the earlier years having a higher average rating.



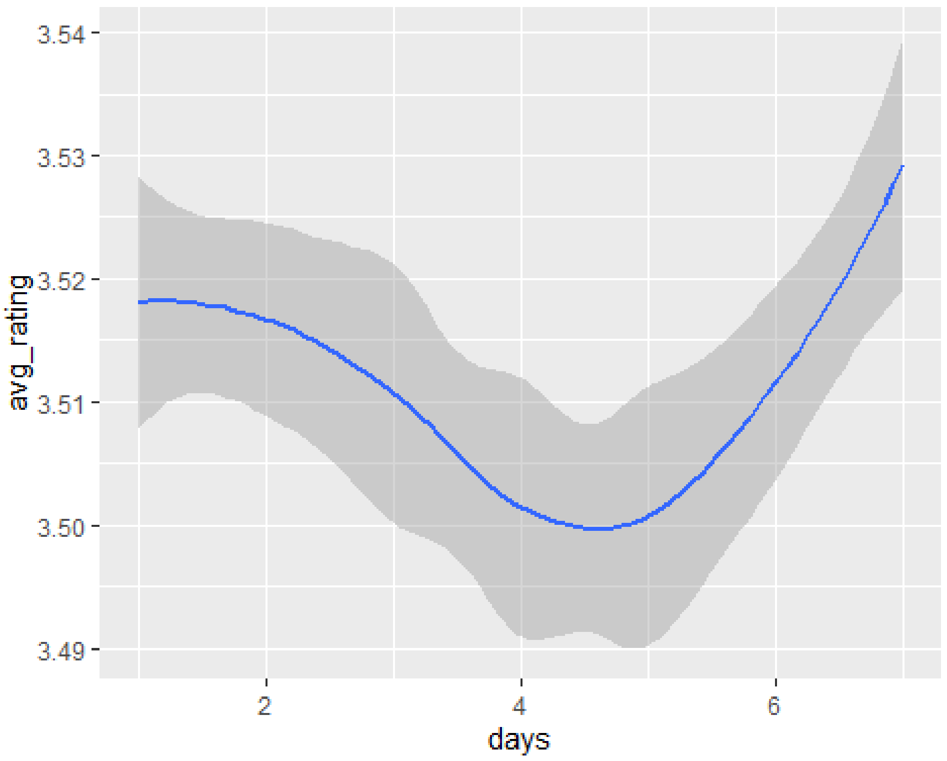
Time of day also appears to play a role in ratings as lower ratings typical are recorded between 5 and 11am. Higher ratings come closer to midnight.

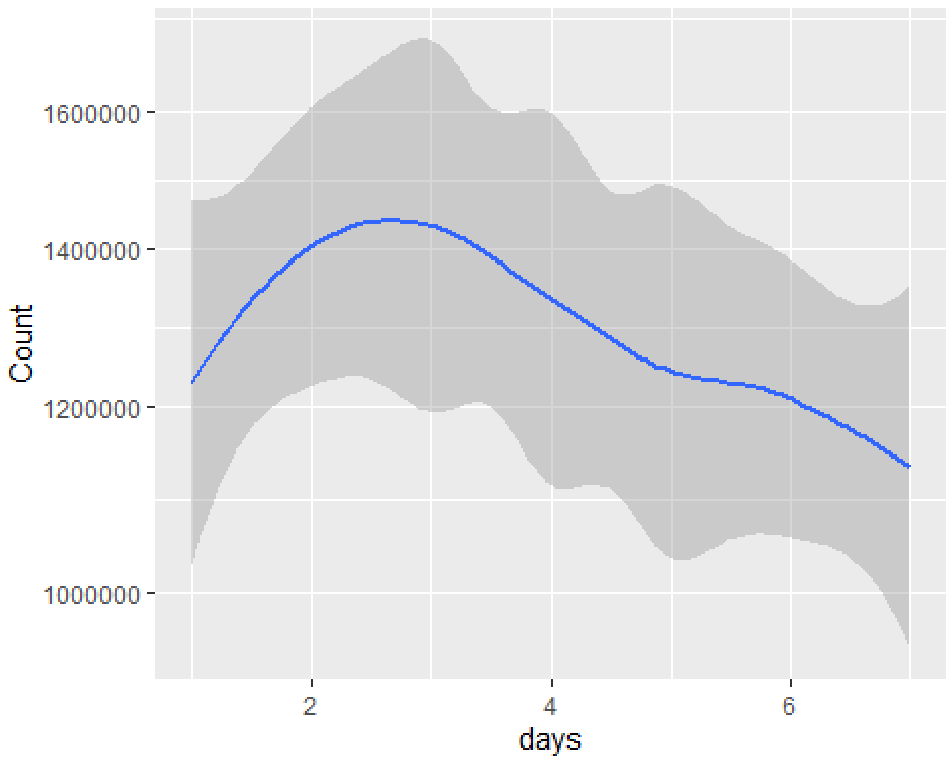


Coincidentally perhaps, those same hours are when the least amount of ratiings are being recorded.



In most households, it is probably expected that the most movies would be watched over the weekend. These charts display how average ratings are impacted by day of the week and it is discovered the most active day for ratings, is actually Wednesday. The middle of the week also seems to yield lower ratiings.





To get an idea of movies included and how they were rated, the 5 best and worse movies were pulled that had more than 100 ratings.

```
edx %>% group_by(title) %>% summarize(n=n(),avg_rating=mean(rating))%>%
  filter(n>100) %>% arrange(desc(avg_rating)) %>% head(5)
```

```
## # A tibble: 5 x 3
##   title                                 n avg_rating
##   <chr>                             <int>      <dbl>
## 1 Shawshank Redemption, The (1994) 28015       4.46
## 2 Godfather, The (1972)            17747       4.42
## 3 Usual Suspects, The (1995)       21648       4.37
## 4 Schindler's List (1993)          23193       4.36
## 5 Casablanca (1942)                11232       4.32
```

```
edx %>% group_by(title) %>% summarize(n=n(),avg_rating=mean(rating))%>%
  filter(n>100) %>% arrange(desc(avg_rating)) %>% tail(5)
```

```
## # A tibble: 5 x 3
##   title                                                     n avg_rating
##   <chr>                                                 <int>      <dbl>
## 1 Barney's Great Adventure (1998)                         208      1.19
## 2 Pokemon 4 Ever (a.k.a. PokÃ©mon 4: The Movie) (2002)    202      1.18
## 3 Glitter (2001)                                          339      1.18
## 4 PokÃ©mon Heroes (2003)                                  137      1.03
## 5 From Justin to Kelly (2003)                             199      0.902
```

Technical Analysis:

Now that the data has been reviewed and a sense of what is included, the analysis can begin. Per instructions the validation data can only utilized on the final step of testing the best model. Therefore, the edx data set is split into a train and test set to form the models necessary for predictions.

```
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(edx$rating,times=1,p=0.2,list = FALSE)
test_set <- edx[test_index,]
train_set <- edx[-test_index,]
```

To confirm the smooth operation of the models, a quick check to confirm movieId and userId are in both sets.

```
test_set <- test_set %>% semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

The first model formed was just calculating the average of the movie ratings and applying it across the board. This approach is often called the naive average approach and is a good starting point. In order to be organized throughout this analysis a table called RMSE_result was formed.

```
mu <- mean(train_set$rating)
avg_pred <- rep(mu,nrow(test_set))
avg_rmse <- RMSE(test_set$rating,avg_pred)
RMSE_result0 <- data_frame(Model = "Naive-Avg",
                           RMSE = avg_rmse) #create table for results
```

Individuals, are just that individuals. So one viewer could easily consider a movie good while another may not. Also there could be bias based upon the movie itself and certain genres will fair better than others. All of these bias were taken into consideration and models were formed to access these bias by beginning with movieId then progressing with userId and finally genres.

```
# Movie Effect Model
movie_avg <- train_set %>% group_by(movieId) %>%
  summarize(b_m = mean(rating-mu))
movie_pred <- test_set %>% left_join(movie_avg, by='movieId') %>%
  mutate(pred=mu+b_m) %>% pull(pred)
movie_rmse <- RMSE(movie_pred,test_set$rating)
RMSE_result1 <- bind_rows(RMSE_result0,
                          data_frame(Model="Movie Effect Model", RMSE =
movie_rmse ))

# Movie+User Effect Model
user_avg <- train_set %>% left_join(movie_avg, by='movieId') %>%
  group_by(userId) %>% summarize(b_u = mean(rating-mu-b_m))
user_pred <- test_set %>% left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  mutate(pred=mu+b_m+b_u) %>% pull(pred)
user_rmse <- RMSE(user_pred,test_set$rating)
RMSE_result1 <- bind_rows(RMSE_result1,
                          data_frame(Model="Movie+User Effect Model", RMSE =
user_rmse ))

# Movie+User+Genre Effect Model
genre_avg <- train_set %>% left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>% group_by(genres) %>%
  summarize(b_g = mean(rating-mu-b_m-b_u))
genre_pred <- test_set %>% left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>% left_join(genre_avg, by='genres') %>%
  mutate(pred=mu+b_m+b_u+b_g) %>% pull(pred)
genre_rmse <- RMSE(genre_pred,test_set$rating)
RMSE_result1 <- bind_rows(RMSE_result1,
                          data_frame(Model="Movie+User+Genre Model", RMSE =
genre_rmse ))
```

To see if there was enough bias in time impact two models were created to consider time of day and day of the week.

```
# Movie+User+Genre+hr Effect Model
hr_avg <- train_set %>% left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>% left_join(genre_avg, by='genres') %>%
  mutate(hr = hour(as_datetime(timestamp))) %>% group_by(hr) %>%
  summarize(b_hr = mean(rating-mu-b_m-b_u-b_g))
hr_pred <- test_set %>% left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>% left_join(genre_avg, by='genres') %>%
  mutate(hr = hour(as_datetime(timestamp))) %>% left_join(hr_avg, by='hr')
%>%
  mutate(pred=mu+b_m+b_u+b_g) %>% pull(pred)
hr_rmse <- RMSE(hr_pred,test_set$rating)
RMSE_result2 <- bind_rows(RMSE_result1,
                          data_frame(Model="Movie+User+Genre+hr Model", RMSE
= hr_rmse ))

# Movie+User+Genre+day Effect Model
day_avg <- train_set %>% left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>% left_join(genre_avg, by='genres') %>%
  mutate(day = wday(as_datetime(timestamp))) %>% group_by(day) %>%
  summarize(b_dy = mean(rating-mu-b_m-b_u-b_g))
day_pred <- test_set %>% left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>% left_join(genre_avg, by='genres') %>%
  mutate(day = wday(as_datetime(timestamp))) %>%
  left_join(day_avg, by='day') %>% mutate(pred=mu+b_m+b_u+b_dy) %>%
pull(pred)
day_rmse <- RMSE(day_pred,test_set$rating)
RMSE_result2 <- bind_rows(RMSE_result2,
                          data_frame(Model="Movie+User+Genre+day Model", RMSE
= hr_rmse ))
```

In addition to bias, there is the risk of overfitting and hidden impacts to be overlooked. A regularized effect model was created with a range of numbers to be used to find the best fit.

```
# Regularized Effects Model
lambdas <- seq(3, 15, 0.5)
RMSEreg <- sapply(lambdas, function(l){
    user_ave <- train_set %>% group_by(userId) %>%
      summarize(u_i = sum(rating-mu)/(n()+l))
    movie_ave <- train_set %>% left_join(user_ave, by='userId') %>%
      group_by(movieId) %>%  summarize(m_i = sum(rating-mu-u_i)/(n()+l))

    pred_rating <- test_set %>% left_join(movie_ave, by= "movieId") %>%
      left_join(user_ave, by='userId') %>%
      mutate(pred = mu + m_i + u_i ) %>% pull(pred)
    RMSE(pred_rating, test_set$rating)
  })
lambda <- lambdas[which.min(RMSEreg)]
RMSEreg_eff <- RMSEreg[[which.min(RMSEreg)]]
RMSE_result3 <- bind_rows(RMSE_result2,
                          data_frame(Model="Regularization Model", RMSE =
RMSEreg_eff ))
```

The final model to be fitted to this data was Matrix Factorization. This process goes through using algorithms to decompose the user-item interface matrix by looking at two smaller matrices. Prior to doing the Matrix Factorization, the data was reduced to a residual data set which removes all the bias noted previously (genres, userId, and movieId). The variables that need determined to use this is a user index, item index and rating. The recosystem instructions were used to step through this calculation, which provided a good start for initial parameters for fitting. Once the model was trained it was ready to be applied to the test set with the parameters optimally tuned.

```
# Matrix Factorization Model
# Calculating residuals by removing movie, user and genres effect
residual_set <- train_set %>% left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  left_join(genre_avg, by='genres') %>%
  mutate(resid = rating - mu - b_m - b_u - b_g)

# Setting data and tuning parameters - used R documentation for parameters
reco <- Reco()
train_reco <- data_memory(user_index=residual_set$userId,
                          item_index=residual_set$movieId,
                              rating=residual_set$resid, index1=TRUE)
test_reco <- data_memory(user_index=test_set$userId,
                         item_index=test_set$movieId, index1=TRUE)

# This step is lengthy and could take 15+ minutes
opts <- reco$tune(train_reco, opts = list(dim = c(10, 20, 30),
                                          lrate = c(0.1, 0.2),
                                              costp_l1=0, costq_l1=0,
                                              nthread = 1, niter = 10))

# Train the model of reco
reco$train(train_reco, opts = c(opts$min, nthread=1, niter=20))

## iter      tr_rmse          obj
##    0       0.8605   5.5669e+006
##    1       0.8385   5.1931e+006
##    2       0.8197   5.0324e+006
##    3       0.8025   4.8985e+006
##    4       0.7872   4.7814e+006
##    5       0.7737   4.6842e+006
##    6       0.7620  4.6024e+006
##    7       0.7519  4.5319e+006
##    8       0.7432  4.4749e+006
##    9       0.7356  4.4262e+006
##   10       0.7288  4.3832e+006
##   11       0.7228  4.3437e+006
##   12       0.7176  4.3125e+006
##   13       0.7128  4.2837e+006
##   14       0.7085  4.2577e+006
##   15       0.7047  4.2353e+006
##   16       0.7012  4.2147e+006
##   17       0.6980  4.1956e+006
##   18       0.6951  4.1795e+006
##   19       0.6924  4.1637e+006
```

```
# Predict results for the test set
reco_pred <- reco$predict(test_reco, out_memory())
preds <- cbind(test_set, reco_pred) %>%
  left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  left_join(genre_avg, by='genres') %>%
  mutate(pred = mu + b_m + b_u + b_g + reco_pred) %>%
  pull(pred)
rmseMF <- RMSE(preds, test_set$rating)
RMSE_result4 <- bind_rows(RMSE_result3,
                          data_frame(Model="Matrix Factorization", RMSE =
rmseMF )) %>%
  arrange(desc(RMSE))
```

Results:

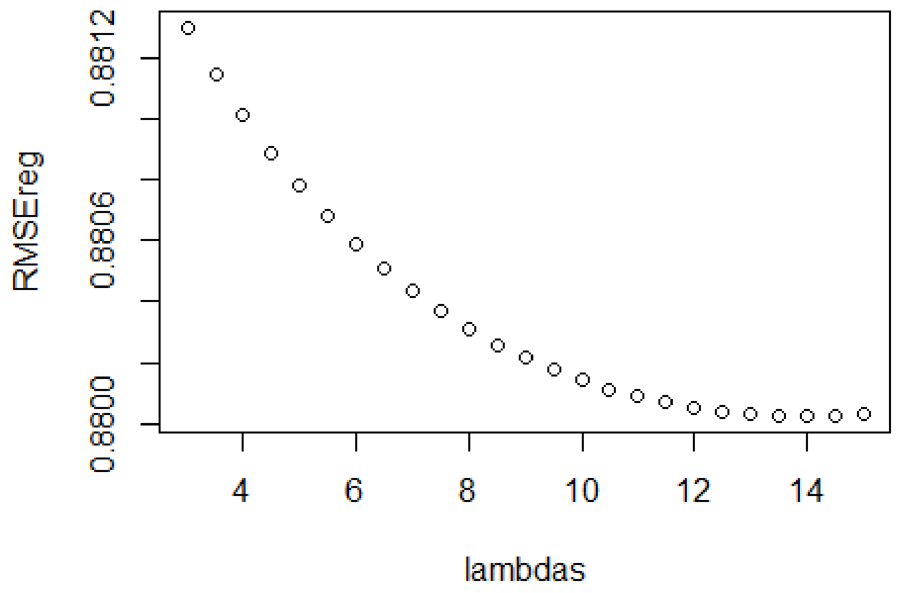
The baseline model was the Naive average model with an RMSE of 1.0599043 and was the starting point. From there bias were built in one at a time from movieId to userId and finally genres. A rather decent improvement was seen from the average to the first bias model and again to the second (movieId+userId). Subsequently, the adding the genres bias models only improved the RMSE marginally.

Model	RMSE
Naive-Avg	1.0599043
Movie Effect Model	0.9437429
Movie+User Effect Model	0.8659320
Movie+User+Genre Model	0.8655941

Different times of the day and days of the week would be expected to play some bias as well but it did not improve the RMSE to an acceptable level (target of 0.8649). Actually, it didn't change the results at all.

Model	RMSE
Naive-Avg	1.0599043
Movie Effect Model	0.9437429
Movie+User Effect Model	0.8659320
Movie+User+Genre Model	0.8655941
Movie+User+Genre+hr Model	0.8655941
Movie+User+Genre+day Model	0.8655941

To ensure the models were not over trained or missing some buried relationship unintentionally, the bias effect model was regularized with a lambda. After tuning, the optimum lambda was determined to be 14. Regularizing the model increased the RMSE unexpectedly.



```
lambdas[which.min(RMSEreg)]

## [1] 14
```

Model	RMSE
Naive-Avg	1.0599043
Movie Effect Model	0.9437429
Movie+User Effect Model	0.8659320
Movie+User+Genre Model	0.8655941
Movie+User+Genre+hr Model	0.8655941
Movie+User+Genre+day Model	0.8655941
Regularization Model	0.8800252

Finally, the Matrix Factorization yielded the desired results of RMSE <0.8649. This process takes quite a bit of computing time but yielded an RMSE of 0.7972506. The table was rearranged by descending RMSE to have the best model being the last entry.

Model	RMSE
Naive-Avg	1.0599043
Movie Effect Model	0.9437429
Regularization Model	0.8800252
Movie+User Effect Model	0.8659320
Movie+User+Genre Model	0.8655941
Movie+User+Genre+hr Model	0.8655941
Movie+User+Genre+day Model	0.8655941
Matrix Factorization	0.7972506

This model proved to be the best model and was utilized against the validation data set. The original trained reco model was applied to the validation data to form new predictions.

```r
# Making sure userId and movieId in validation set are also in the train set
valid <- validation %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
valid_reco <- data_memory(user_index=valid$userId,
                          item_index=valid$movieId, index1=TRUE)
r_pred <- reco$predict(valid_reco, out_memory())
val_preds <- cbind(valid, r_pred) %>%
  left_join(movie_avg, by='movieId') %>%
  left_join(user_avg, by='userId') %>%
  left_join(genre_avg, by='genres') %>%
  mutate(pred = mu + b_m + b_u + b_g + r_pred) %>%
  pull(pred)
rmseFinal <- RMSE(val_preds, valid$rating)
```

Following these steps resulted in a final RMSE of 0.7970038.

Conclusion:

After a review of the data and dissecting of the edx dataset the analysis was performed. Initially, there was a step increase but then the other bias and regularization models are improved marginally. The final Matrix Factorization, was the most time consuming but yielded the best results. Within the feature it has algorithms built in already that proved very beneficial in predictiing rating. Through all the models an improvement from 1.0599043 with Naive Average prediction to 0.7970038 with the final Matrix Factorization with validation data was achieved. If genres were more regulated so that there aren’t 797 variations, this could greatly increase predictions. Other factors that could be documented to assist in modeling are movie length and main character.