Harvard edX Data Science Capstone

J. Dittenber

December 2022

1. ABSTRACT:

The MovieLens dataset contains over 9,000,000 observations of users' interactions with a movie rating schema over a period of time from 1995 to 2009. The variables include the userId, movieId, movie title, genres and ratings. The objective is to build a model that can predict ratings with a loss of RMSE < 0.86.RMSE is a metric that quantifies how different the predicted values are from known values when the variables are input into a model. After analysis using various models, the model with the lowest loss includes userId and movie id and is a multiple linear regression model.

This is a movie recommendation system created using machine learning for the capstone project of the Harvard edX Data Science Professional Certificate capstone. The R Markdown document will walk through all of the steps, code and reasoning behind each step. The outline is as follows:

1. Abstract
2. **Executive Summary**
3. Importing libraries, data and creating test and train sets
4. Exploratory data analysis and pre-processing
    1. Data cleaning and anomaly detection
    2. Checking that test and train data sets have correct dimensions and variables
    3. Variable Analysis
    4. Correlation Analysis
5. **Methodology and Analysis**
6. Model creation and diagnostics
7. **Results**
8. **Conclusion**

2. EXECUTIVE SUMMARY:

In this project I worked with the MovieLens dataset, a collection of ratings and metadata for a selection of movies. My goal was to build a model to predict ratings for the movies that could achieve an RMSE of less than 0.86.

To start, I sampled the data and generated 1000 models of size n=77 for a multiple linear regression model, decision tree and a random forest model. After evaluating these models RMSE on the train and test set (not the final holdout set), the evidence supported that the multiple linear regression model using userId, movieId and genres was the model with the best (lowest) RMSE.

With this information, I built on the course methodology of utilizing the entire data set and manually calculating the parameters for the multiple linear regression model. This was necessary since the computational time and resources needed to utilize the packages in R (that can generate machine learning models) is not easily available on a laptop or PC.

The conclusion revealed that the best model (calculated using the full 9,000,000 observations) was the model that included only the movieId and userid.

3. Importing libraries, data and creating test and train sets

Import Libraries

```
############################################################
# Import Libraries
############################################################

# Note: this process could take a couple of minutes
options(repos = list(CRAN="http://cran.rstudio.com/"))
install.packages('plyr', repos ="http://cran.us.r-project.org" )
```

```
## Installing package into 'C:/Users/jeffd/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)
```

```
## Warning: cannot remove prior installation of package 'plyr'
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying C:
## \Users\jeffd\AppData\Local\R\win-library\4.2\00LOCK\plyr\libs\x64\plyr.dll to C:
## \Users\jeffd\AppData\Local\R\win-library\4.2\plyr\libs\x64\plyr.dll: Permission
## denied
```

```
## Warning: restored 'plyr'
```

```
library(plyr)
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::arrange()   masks plyr::arrange()
## x purrr::compact()   masks plyr::compact()
## x dplyr::count()     masks plyr::count()
## x dplyr::failwith()  masks plyr::failwith()
## x dplyr::filter()    masks stats::filter()
```

```
## x dplyr::id()        masks plyr::id()
## x dplyr::lag()       masks stats::lag()
## x dplyr::mutate()    masks plyr::mutate()
## x dplyr::rename()    masks plyr::rename()
## x dplyr::summarise() masks plyr::summarise()
## x dplyr::summarize() masks plyr::summarize()
```

```
if(!require(dslabs)) install.packages("dslabs", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: dslabs
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

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

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

```
## Loading required package: data.table
##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
if(!require(psych)) install.packages("psych", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: psych
##
## Attaching package: 'psych'
##
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
if(!require(rafalib)) install.packages("rafalib", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: rafalib
```

```
if(!require(tinytex)) install.packages("tinytex", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tinytex
```

```
if(!require(formatR)) install.packages("formatR", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: formatR
```

```
if(!require(rpart)) install.packages("rpart", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: rpart
```

```
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(nnet)) install.packages("nnet", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: nnet
```

```
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: randomForest
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:psych':
##
##     outlier
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
```

```
library(tidyr)
library(dslabs)
library(tidyverse)
library(caret)
library(data.table)
library(psych)
library(rafalib)
library(tinytex)
library(formatR)
library(rpart)
library(ggplot2)
library(nnet)
library(randomForest)
library(dplyr)
library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
##
## Loaded glmnet 4.1-6
```

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

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

4. Exploratory Data Analysis & Preprocessing

We will examine the variables and their types:

```
vars <- tibble(name = names(edx), class = class(edx))
vars
```

```
## # A tibble: 6 x 2
##   name      class
##   <chr>     <chr>
## 1 userId    data.frame
## 2 movieId   data.frame
## 3 rating    data.frame
## 4 timestamp data.frame
## 5 title     data.frame
## 6 genres    data.frame
```

Change the timestamp to date and year of rating

```
new_edx <- edx %>%  mutate(date_rated = as.POSIXct(timestamp, origin = "1970-01-01", tz="UTC"),
                           year_rated = format(date_rated, "%Y"))
```

Verify that the dates are converted correctly and a new column is formed

```
head(new_edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres          date_rated year_rated
## 1               Comedy|Romance 1996-08-02 11:24:06       1996
## 2         Action|Crime|Thriller 1996-08-02 10:58:45       1996
## 4  Action|Drama|Sci-Fi|Thriller 1996-08-02 10:57:01       1996
## 5       Action|Adventure|Sci-Fi 1996-08-02 10:56:32       1996
## 6 Action|Adventure|Drama|Sci-Fi 1996-08-02 10:56:32       1996
## 7       Children|Comedy|Fantasy 1996-08-02 11:14:34       1996
```

```
#check the data types of the mutated data set

vars <- tibble(name = names(new_edx), class = class(new_edx))
vars
```

```
## # A tibble: 8 x 2
##   name       class
##   <chr>      <chr>
## 1 userId     data.frame
```

```
## 2 movieId    data.frame
## 3 rating     data.frame
## 4 timestamp  data.frame
## 5 title      data.frame
## 6 genres     data.frame
## 7 date_rated data.frame
## 8 year_rated data.frame
```

```
#check the columns for date_rated and year_rated

min(new_edx$date_rate); min(new_edx$year_rate); max(new_edx$date_rate); max(new_edx$year_rate)
```

```
## [1] "1995-01-09 11:46:49 UTC"
```

```
## [1] "1995"
```

```
## [1] "2009-01-05 05:02:16 UTC"
```

```
## [1] "2009"
```

```
#don't need timestamp now

new_edx$timestamp <- NULL
```

Check for NA/Nulls by comparing lengths after applying na.omit()

```
# create new dataframe after applying na.omit -this will remove observations
# with NA
check_df <- na.omit(new_edx)
length(check_df$rating)
```

```
## [1] 9000055
```

```
length(new_edx$rating)
```

```
## [1] 9000055
```

```
rm(check_df)
```

Variable Analysis

I will examine each variable for informative characteristics.

UserId

```
#length of the variables

length(new_edx$userId)
```

```
## [1] 9000055
```

```
#unique users

n_distinct(new_edx$userId)
```

```
## [1] 69878
```

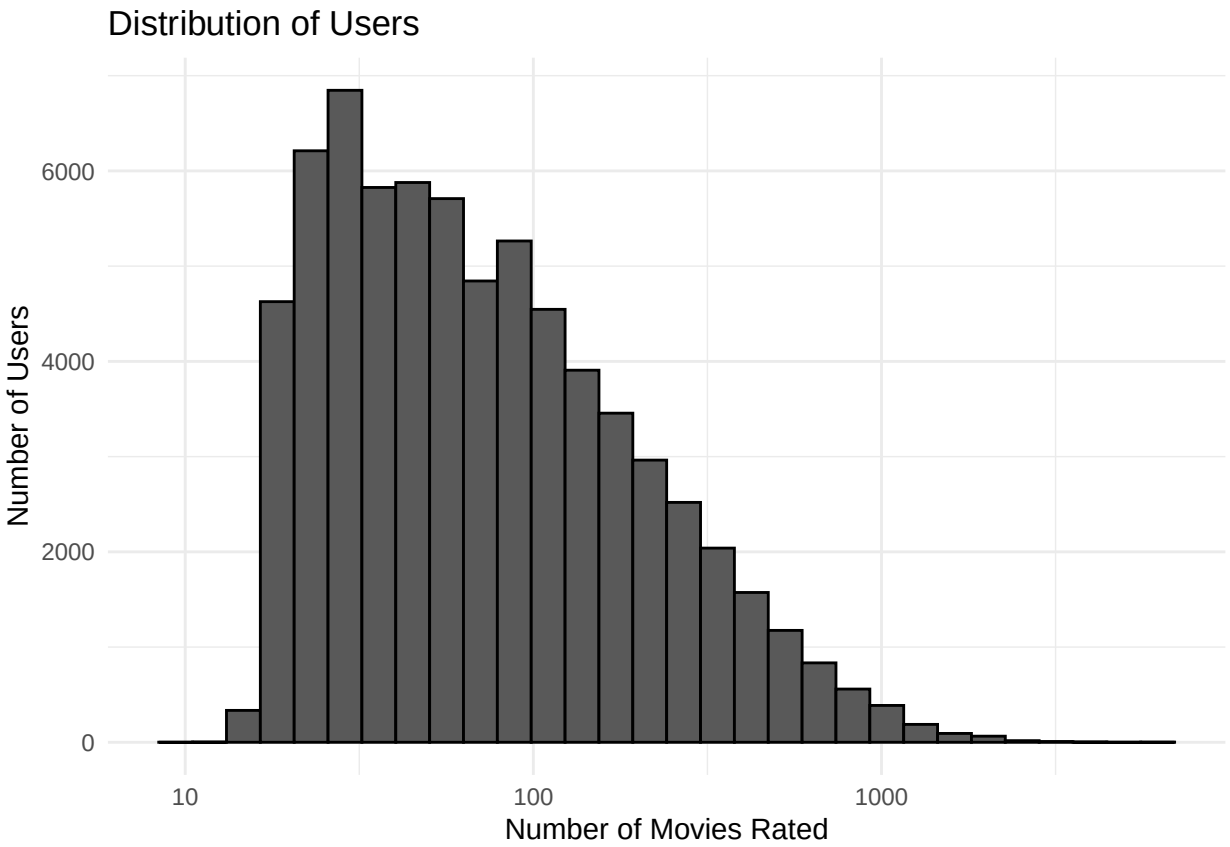
```
#users sorted by rating count

new_edx %>%  group_by(userId) %>% count() %>% arrange(desc(n))
```

```
## # A tibble: 69,878 x 2
## # Groups:   userId [69,878]
##    userId     n
##     <int> <int>
##  1  59269  6616
##  2  67385  6360
##  3  14463  4648
##  4  68259  4036
##  5  27468  4023
##  6  19635  3771
##  7   3817  3733
##  8  63134  3371
##  9  58357  3361
## 10  27584  3142
## # ... with 69,868 more rows
```

```
#check for any ratings of 0

new_edx %>%  group_by(userId) %>% count() %>%  arrange(n)
```

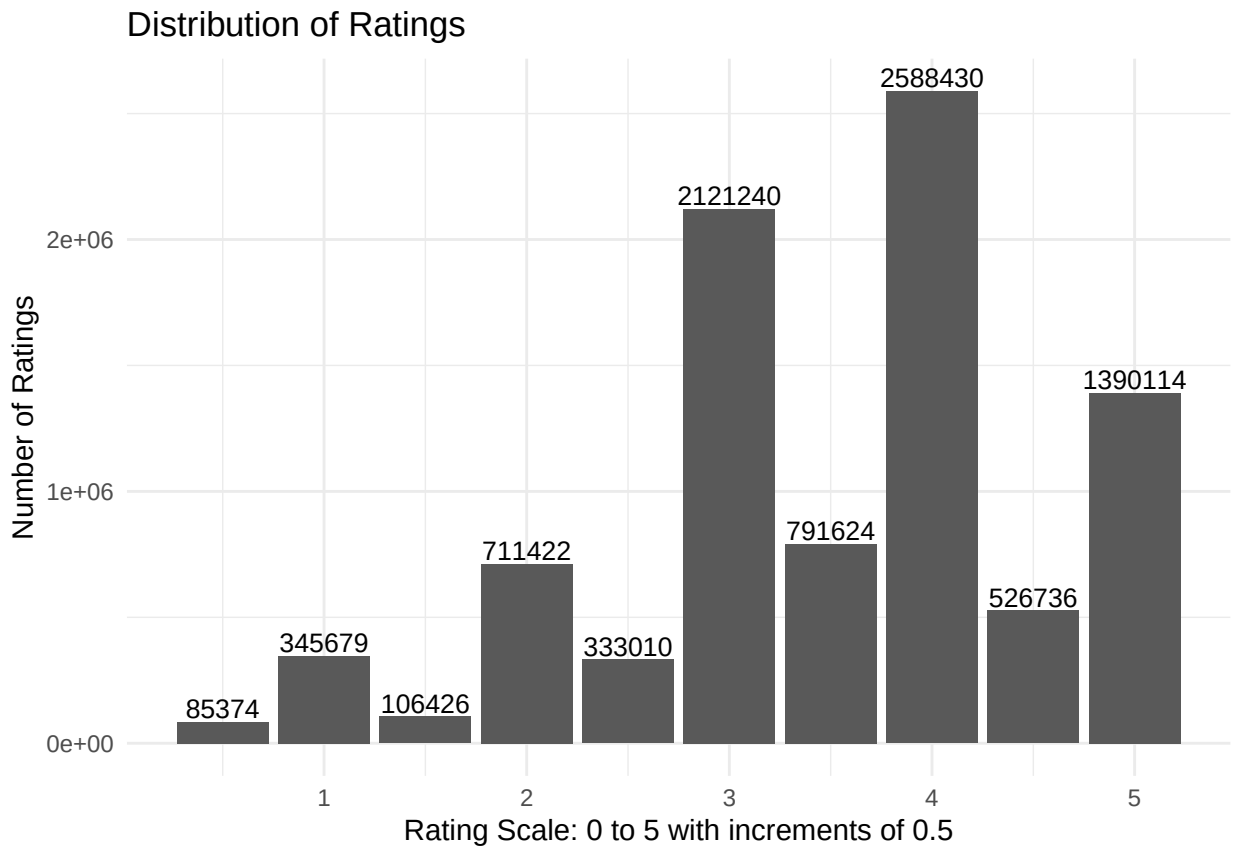
```
## # A tibble: 69,878 x 2
## # Groups:   userId [69,878]
##    userId     n
##     <int> <int>
##  1  62516    10
##  2  22170    12
##  3  15719    13
##  4  50608    13
##  5    901    14
##  6   1833    14
##  7   2476    14
##  8   5214    14
##  9   9689    14
## 10  10364    14
## # ... with 69,868 more rows
```

```
#examine the distribution of movies rated per user

new_edx %>%
    dplyr::count(userId) %>%
    ggplot(aes(n)) +
    geom_histogram(bins = 30, color = 'black') +
    scale_x_log10() +
    ggtitle("Distribution of Users") +
    xlab("Number of Movies Rated") +
    ylab("Number of Users")  +
    theme_minimal()
```


Distribution of Users
Number of Users
6000
4000
2000
0
10
100
1000
Number of Movies Rated

Ratings

```
#count  number of ratings for each rating
rating_counts <- new_edx %>%
  group_by(rating) %>%
  summarise(count = n())

#bar chart
ggplot(rating_counts, aes(x = rating, y = count)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = count), vjust = -0.2, size = 3.5) +
  ggtitle("Distribution of Ratings") +
  xlab("Rating Scale: 0 to 5 with increments of 0.5") +
  ylab("Number of Ratings") +
  theme_minimal()
```




```
#count of each rating level
rating_counts %>% knitr::kable()
```

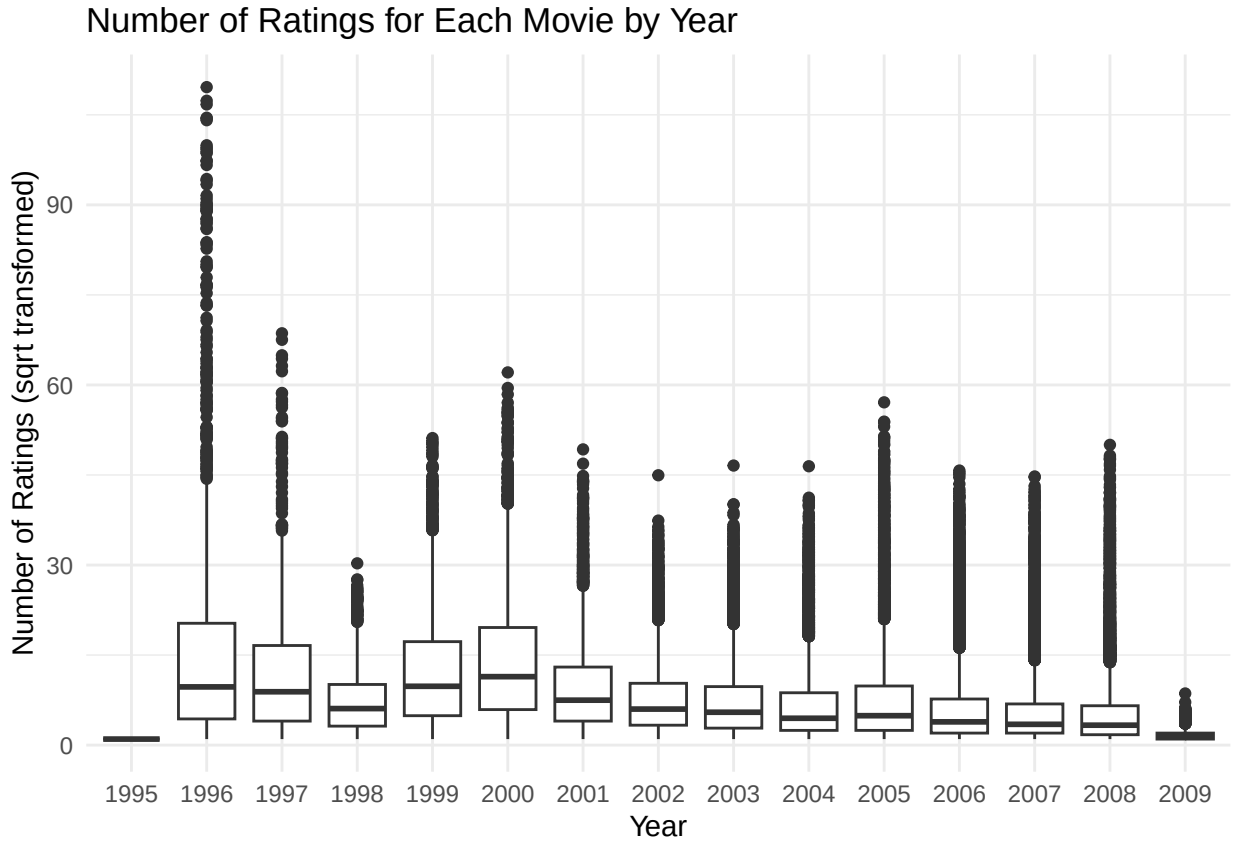
rating	count
0.5	85374
1.0	345679
1.5	106426
2.0	711422
2.5	333010
3.0	2121240

rating	count
3.5	791624
4.0	2588430
4.5	526736
5.0	1390114

```
# boxplot for ratings per year

ratings_by_year <- new_edx %>%
  group_by(year_rated, movieId) %>%
  summarise(num_ratings = n()) %>%
  mutate(sqrt_num_ratings = sqrt(num_ratings))
```

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

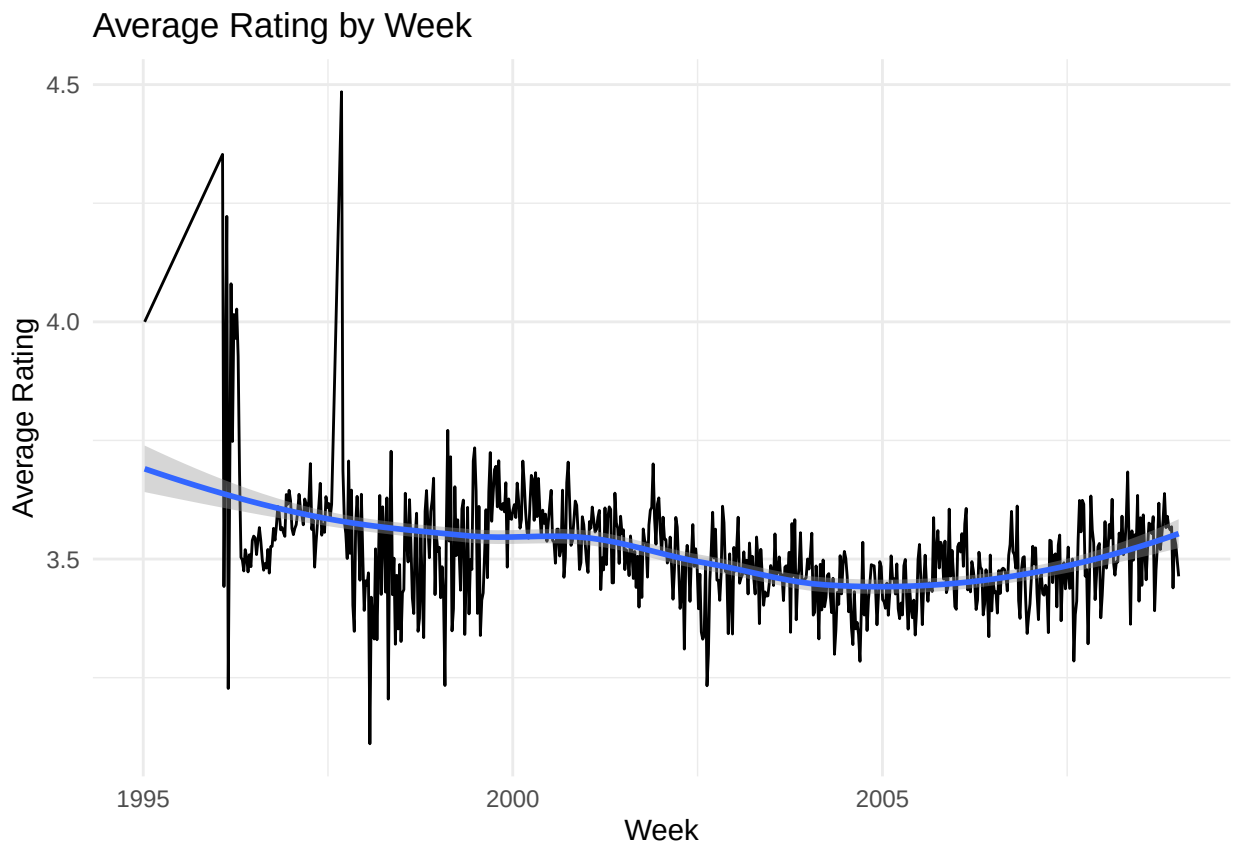
```
# create the plot
ggplot(ratings_by_year, aes(x = year_rated, y = sqrt_num_ratings)) +
  geom_boxplot() +
  ggtitle("Number of Ratings for Each Movie by Year") +
  xlab("Year") +
  ylab("Number of Ratings (sqrt transformed)") +
  theme_minimal()
```




```
#over time per week
ratings_by_week <- new_edx %>%
```

```
  mutate(week = lubridate::round_date(date_rated, "week")) %>%
  group_by(week) %>%
  summarise(mean_rating = mean(rating))

#create the plot
ggplot(ratings_by_week, aes(x = week, y = mean_rating)) +
  geom_line() +
  geom_smooth(method = 'loess')+
  ggtitle("Average Rating by Week") +
  xlab("Week") +
  ylab("Average Rating") +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Genres

```
# examine genres as grouped

# total length
length(new_edx$genres)
```

```
## [1] 9000055
```

```
# distinct genres combinations
n_distinct(new_edx$genres)
```

```
## [1] 797
```

```
# sort by rating count
new_edx %>%
    group_by(genres) %>%
    count() %>%
    arrange(desc(n))
```

```
## # A tibble: 797 x 2
## # Groups:   genres [797]
##    genres                          n
##    <chr>                       <int>
##  1 Drama                      733296
##  2 Comedy                     700889
##  3 Comedy|Romance             365468
##  4 Comedy|Drama               323637
##  5 Comedy|Drama|Romance       261425
##  6 Drama|Romance              259355
##  7 Action|Adventure|Sci-Fi    219938
##  8 Action|Adventure|Thriller 149091
##  9 Drama|Thriller             145373
## 10 Crime|Drama                137387
## # ... with 787 more rows
```

```
# sort by highest average rating (note that rating is categorical, yet we can
# still apply this)
new_edx %>%
    group_by(genres) %>%
    summarize(mean_rating = mean(rating), sd_rating = sd(rating)) %>%
    arrange(desc(mean_rating))
```

```
## # A tibble: 797 x 3
##    genres                                    mean_rating sd_rating
##    <chr>                                           <dbl>     <dbl>
##  1 Animation|IMAX|Sci-Fi                            4.71     0.567
##  2 Drama|Film-Noir|Romance                          4.30     0.791
##  3 Action|Crime|Drama|IMAX                          4.30     0.739
##  4 Animation|Children|Comedy|Crime                  4.28     0.815
##  5 Film-Noir|Mystery                               4.24     0.788
##  6 Crime|Film-Noir|Mystery                         4.22     0.762
##  7 Film-Noir|Romance|Thriller                      4.22     0.738
##  8 Crime|Film-Noir|Thriller                        4.21     0.830
##  9 Crime|Mystery|Thriller                          4.20     0.844
## 10 Action|Adventure|Comedy|Fantasy|Romance         4.20     0.862
## # ... with 787 more rows
```

```
# sort by lowest average rating (note that rating is categorical, yet we can
# still apply this)
new_edx %>%
    group_by(genres) %>%
    summarize(mean_rating = mean(rating), sd_rating = sd(rating)) %>%
    arrange(mean_rating)
```

```
## # A tibble: 797 x 3
##    genres                                    mean_rating sd_rating
##    <chr>                                           <dbl>     <dbl>
##  1 Documentary|Horror                               1.45     1.20
```

```
##  2 Action|Animation|Comedy|Horror                     1.5      0.707
##  3 Action|Horror|Mystery|Thriller                    1.61     1.10
##  4 Comedy|Film-Noir|Thriller                         1.64     0.868
##  5 Action|Drama|Horror|Sci-Fi                        1.75     1.26
##  6 Adventure|Drama|Horror|Sci-Fi|Thriller            1.75     1.09
##  7 Action|Adventure|Drama|Fantasy|Sci-Fi             1.90     1.08
##  8 Action|Children|Comedy                            1.91     1.18
##  9 Action|Adventure|Children                         1.92     1.25
## 10 Adventure|Animation|Children|Fantasy|Sci-Fi       1.92     1.20
## # ... with 787 more rows
```

```
# separate the combinations of genres
genres_edx <- new_edx %>%
    separate_rows(genres, sep = "\\|")
head(genres_edx)
```

```
## # A tibble: 6 x 7
##   userId movieId rating title            genres   date_rated          year_rated
##    <int>   <int>  <dbl> <chr>            <chr>    <dttm>              <chr>
## 1      1     122      5 Boomerang (1992) Comedy   1996-08-02 11:24:06 1996
## 2      1     122      5 Boomerang (1992) Romance  1996-08-02 11:24:06 1996
## 3      1     185      5 Net, The (1995)  Action   1996-08-02 10:58:45 1996
## 4      1     185      5 Net, The (1995)  Crime    1996-08-02 10:58:45 1996
## 5      1     185      5 Net, The (1995)  Thriller 1996-08-02 10:58:45 1996
## 6      1     292      5 Outbreak (1995)  Action   1996-08-02 10:57:01 1996
```

```
str(genres_edx)
```

```
## tibble [23,371,423 x 7] (S3: tbl_df/tbl/data.frame)
##  $ userId    : int [1:23371423] 1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId   : int [1:23371423] 122 122 185 185 185 292 292 292 292 316 ...
##  $ rating    : num [1:23371423] 5 5 5 5 5 5 5 5 5 5 ...
##  $ title     : chr [1:23371423] "Boomerang (1992)" "Boomerang (1992)" "Net, The (1995)" "Net, The (19
##  $ genres    : chr [1:23371423] "Comedy" "Romance" "Action" "Crime" ...
##  $ date_rated: POSIXct[1:23371423], format: "1996-08-02 11:24:06" "1996-08-02 11:24:06" ...
##  $ year_rated: chr [1:23371423] "1996" "1996" "1996" "1996" ...
```

```
# determine how many observations
length(genres_edx$genres)
```

```
## [1] 23371423
```

```
# distinct genres when split
n_distinct(genres_edx$genres)
```

```
## [1] 20
```

```
# sort by highest average rating (note that rating is categorical, yet we can
# still apply this)
genres_edx %>%
    group_by(genres) %>%
    summarize(mean_rating = mean(rating), sd_rating = sd(rating), count = n()) %>%
    arrange(desc(mean_rating))
```

```
## # A tibble: 20 x 4
##    genres             mean_rating sd_rating   count
##    <chr>                    <dbl>     <dbl>   <int>
##  1 Film-Noir                 4.01     0.887  118541
```

```
##  2 Documentary               3.78     1.00    93066
##  3 War                       3.78     1.01   511147
##  4 IMAX                      3.77     1.03     8181
##  5 Mystery                   3.68     1.00   568332
##  6 Drama                     3.67     0.995 3910127
##  7 Crime                     3.67     1.01  1327715
##  8 (no genres listed)        3.64     1.11        7
##  9 Animation                 3.60     1.02   467168
## 10 Musical                   3.56     1.06   433080
## 11 Western                   3.56     1.02   189394
## 12 Romance                   3.55     1.03  1712100
## 13 Thriller                  3.51     1.03  2325899
## 14 Fantasy                   3.50     1.07   925637
## 15 Adventure                 3.49     1.05  1908892
## 16 Comedy                    3.44     1.07  3540930
## 17 Action                    3.42     1.07  2560545
## 18 Children                  3.42     1.09   737994
## 19 Sci-Fi                    3.40     1.09  1341183
## 20 Horror                    3.27     1.15   691485
```

```
# sort by highest count
genres_edx %>%
    group_by(genres) %>%
    summarize(mean_rating = mean(rating), sd_rating = sd(rating), count = n()) %>%
    arrange(desc(count))
```

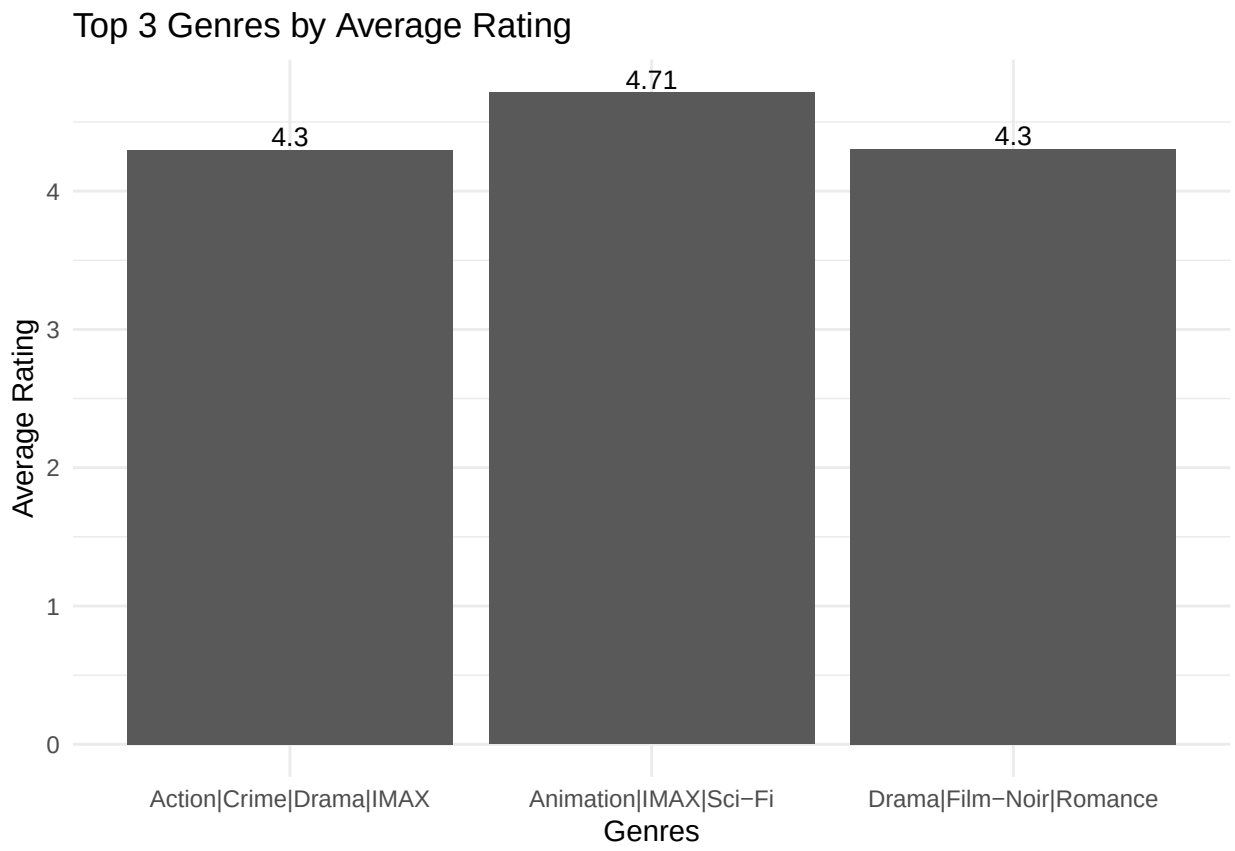
```
## # A tibble: 20 x 4
##    genres              mean_rating sd_rating   count
##    <chr>                     <dbl>     <dbl>   <int>
##  1 Drama                      3.67     0.995 3910127
##  2 Comedy                     3.44     1.07  3540930
##  3 Action                     3.42     1.07  2560545
##  4 Thriller                   3.51     1.03  2325899
##  5 Adventure                  3.49     1.05  1908892
##  6 Romance                    3.55     1.03  1712100
##  7 Sci-Fi                     3.40     1.09  1341183
##  8 Crime                      3.67     1.01  1327715
##  9 Fantasy                    3.50     1.07   925637
## 10 Children                   3.42     1.09   737994
## 11 Horror                     3.27     1.15   691485
## 12 Mystery                    3.68     1.00   568332
## 13 War                        3.78     1.01   511147
## 14 Animation                  3.60     1.02   467168
## 15 Musical                    3.56     1.06   433080
## 16 Western                    3.56     1.02   189394
## 17 Film-Noir                  4.01     0.887  118541
## 18 Documentary                3.78     1.00    93066
## 19 IMAX                       3.77     1.03     8181
## 20 (no genres listed)         3.64     1.11        7
```

```
# sort by lowest average rating (note that rating is categorical, yet we can
# still apply this)
genres_edx %>%
    group_by(genres) %>%
    summarize(mean_rating = mean(rating), sd_rating = sd(rating)) %>%
```

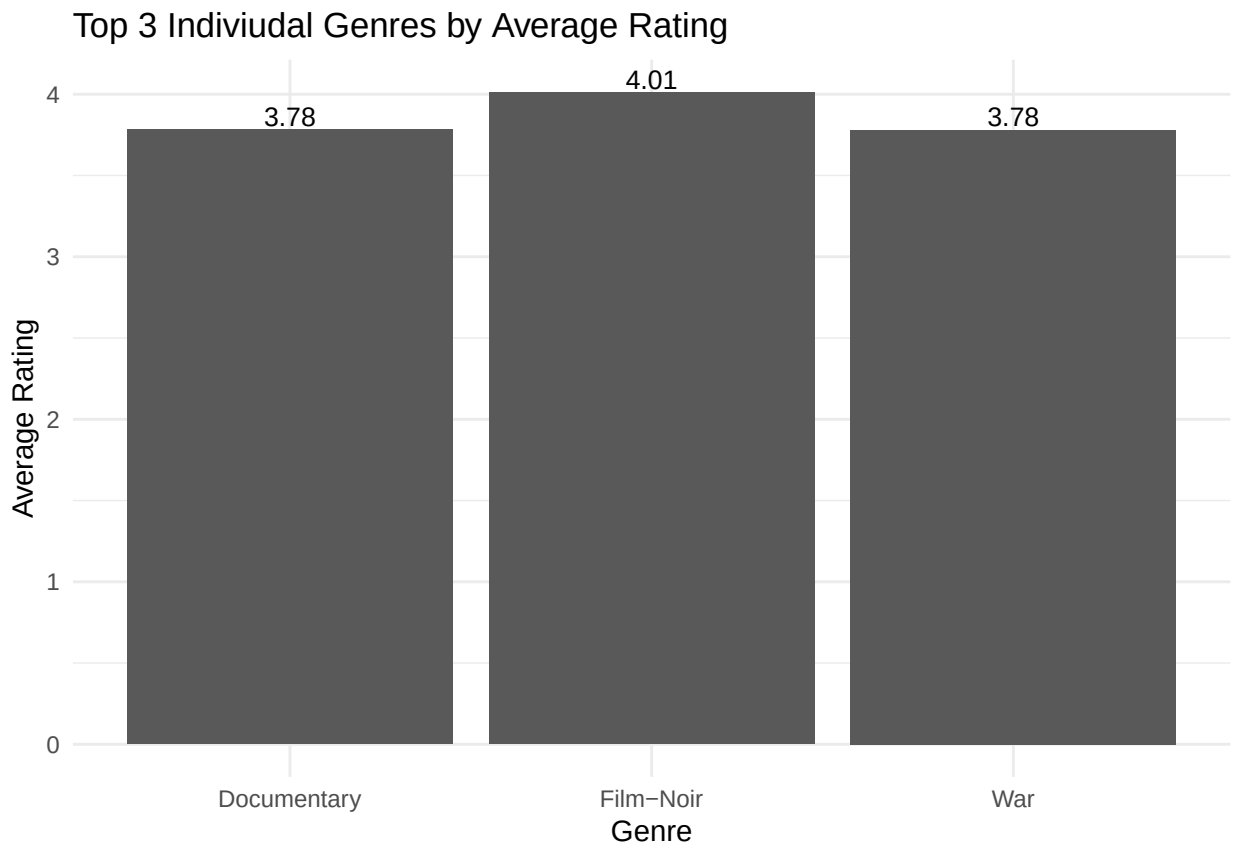
```
    arrange(mean_rating)
```

```
## # A tibble: 20 x 3
##    genres             mean_rating sd_rating
##    <chr>                    <dbl>     <dbl>
##  1 Horror                    3.27     1.15
##  2 Sci-Fi                    3.40     1.09
##  3 Children                  3.42     1.09
##  4 Action                    3.42     1.07
##  5 Comedy                    3.44     1.07
##  6 Adventure                 3.49     1.05
##  7 Fantasy                   3.50     1.07
##  8 Thriller                  3.51     1.03
##  9 Romance                   3.55     1.03
## 10 Western                   3.56     1.02
## 11 Musical                   3.56     1.06
## 12 Animation                 3.60     1.02
## 13 (no genres listed)        3.64     1.11
## 14 Crime                     3.67     1.01
## 15 Drama                     3.67     0.995
## 16 Mystery                   3.68     1.00
## 17 IMAX                      3.77     1.03
## 18 War                       3.78     1.01
## 19 Documentary               3.78     1.00
## 20 Film-Noir                 4.01     0.887
```

```
# filter the movies with at least 50 ratings
movies_50 <- new_edx %>%
    group_by(movieId) %>%
    filter(n() >= 50) %>%
    ungroup()


# count the number of ratings for each genre
genre_counts <- movies_50 %>%
    group_by(genres) %>%
    summarise(count = n()) %>%
    arrange(desc(count))


# calculate the average rating for each genre
genre_ratings <- new_edx %>%
    group_by(genres) %>%
    summarise(average_rating = mean(rating)) %>%
    arrange(desc(average_rating)) %>%
    head(3)

# bar chart
ggplot(genre_ratings, aes(x = genres, y = average_rating)) + geom_col() + geom_text(aes(label = round(av
    2)), vjust = -0.2, size = 3.5) + ggtitle("Top 3 Genres by Average Rating") +
    xlab("Genres") + ylab("Average Rating") + theme_minimal()
```




```
# Separate the Genres

# calculate the average rating for each genre
genre_ratings <- genres_edx %>%
    group_by(genres) %>%
    summarise(mean_rating = mean(rating), sd_rating = sd(rating), count = n()) %>%
    arrange(desc(mean_rating)) %>%
    head(3)

# create bar chart
ggplot(genre_ratings, aes(x = genres, y = mean_rating)) + geom_col() + geom_text(aes(label = round(mean_
    2)), vjust = -0.2, size = 3.5) + ggtitle("Top 3 Indiviudal Genres by Average Rating") +
    xlab("Genre") + ylab("Average Rating") + theme_minimal()
```



5. METHODS AND ANALYSIS

Methodology for the Initial Analysis

Given that the data contains over 9 million observations it seems prudent to sample from these 9 million observations and get an idea of a baseline or even to create a final model.

Models that use the entire data set can have problems in deployment and implementation. Overly complicated models, while reaping low performance metrics have the disadvantage of being impractical to use and taking too much time and computational power.

Therefore, the first attempt at generating a model will be done by sampling the the data and making inferences about the population (of 9 million observations). Afterward, the methodologies utilized in the course will be implemented and expanded.

An a priori power analysis was conducted and it was determined that a sample size of n=77 was sufficient for an alpha = 0.05 and a beta = 0.8. Note that in general, prior research indicates that a rule of them is that there must be 10 observations per variable to reach sufficient statistical power.

The steps:

1. Take 1000 samples of size n = 77
2. For each of these samples build a linear regression model
3. For each of the linear regression models calculate the RMSE
4. Store all of the RMSE in a list
5. Report the minimum RMSE
6. Repeat and add the additional predictors

Furthermore, the approach will be implemented on

1. Linear Regression
2. Decision Tree
3. Random Forest

Summary: In each case, the model with the lowest RMSE is the model that uses all of the predictors, with the exception of the time related predictors (which are excluded). It is worth noting that both the movieId and Title (of the movie) are likely contributing the same amount of variation, and thus, according to the analysis, the model with the lowest RMSE will contain the user Id, movie Id and genre.

Additionally, there is not evidence of overfitting between the test and training sets as they have roughly the same RMSE.

The best model was the multiple linear regression model that used user id, movie id and genres as predictors. Note that these predictors were converted to factors, yet we can still achieve a prediction with an RMSE within the boundaries set forth.

Train and Test Split

```
set.seed(42)

test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

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

6. MODEL CREATION AND DIAGNOSTICS

Sampling Approach - Mutli-Linear Regression Model

```
# sampling for baseline RMSE
rmse_calc <- function(df, formula) {
    model <- lm(formula, data = df)
    sqrt(mean((predict(model) - df$rating)^2))
}

# sample data 1000 times and calculate RMSE for each model
rmse_samples <- data.frame(method = c("User Id Only", "User Id and Movie Id ", "User Id, Movie Id, Genr
    "Userd Id, Movie Id, Genres, Title"), RMSE = sapply(1:4, function(i) {
    formulas <- c("rating ~ userId", "rating ~ userId + movieId", "rating ~ userId + movieId + genres",
        "rating ~ userId + movieId + genres + title")
    mean(replicate(1000, rmse_calc(sample_n(train_set, 77, replace = TRUE), formulas[i])))
}))

# print RMSE values
rmse_samples %>%
    knitr::kable()
```

method	RMSE
User Id Only	1.0421067
User Id and Movie Id	1.0333230
User Id, Movie Id, Genres	0.5684285
Userd Id, Movie Id, Genres, Title	0.0781390

Run on the test set

```
# sample data 1000 times and calculate RMSE for each model
rmse_samples <- data.frame(method = c("User Id Only", "User Id and Movie Id ", "User Id, Movie Id, Genr
    "Userd Id, Movie Id, Genres, Title"), RMSE = sapply(1:4, function(i) {
    formulas <- c("rating ~ userId", "rating ~ userId + movieId", "rating ~ userId + movieId + genres",
        "rating ~ userId + movieId + genres + title")
    # Sample the data 1000 times, calculate the RMSE for each sample, and
    # return the mean RMSE value
    mean(replicate(1000, rmse_calc(sample_n(test_set, 77, replace = TRUE), formulas[i])))
}))

# print RMSE values
rmse_samples %>%
    knitr::kable()
```

method	RMSE
User Id Only	1.0442199
User Id and Movie Id	1.0401380
User Id, Movie Id, Genres	0.5651554
Userd Id, Movie Id, Genres, Title	0.0767749

Sampling Approach - Decision Tree

```
# sample data 1000 times and calculate RMSE for each model
rmse_samples <- data.frame(method = c("User Id Only", "User Id and Movie Id ", "User Id, Movie Id, Genr
    "Userd Id, Movie Id, Genres, Title"), RMSE = sapply(1:4, function(i) {
    formulas <- c("rating ~ userId", "rating ~ userId + movieId", "rating ~ userId + movieId + genres",
        "rating ~ userId + movieId + genres + title")
    mean(replicate(1000, rmse_calc(sample_n(train_set, 77, replace = TRUE), formulas[i])))
}))

# print RMSE values
rmse_samples %>%
    knitr::kable()
```

method	RMSE
User Id Only	1.0498812
User Id and Movie Id	1.0387218
User Id, Movie Id, Genres	0.5690309
Userd Id, Movie Id, Genres, Title	0.0780796

Run on test set

```
# sample data 1000 times and calculate RMSE for each model
rmse_samples <- data.frame(method = c("User Id Only", "User Id and Movie Id ", "User Id, Movie Id, Genre
    "Userd Id, Movie Id, Genres, Title"), RMSE = sapply(1:4, function(i) {
    formulas <- c("rating ~ userId", "rating ~ userId + movieId", "rating ~ userId + movieId + genres",
        "rating ~ userId + movieId + genres + title")
    mean(replicate(1000, rmse_calc(sample_n(test_set, 77, replace = TRUE), formulas[i])))
}))

# Print RMSE values
rmse_samples %>%
    knitr::kable()
```

method	RMSE
User Id Only	1.0457221
User Id and Movie Id	1.0383951
User Id, Movie Id, Genres	0.5733153
Userd Id, Movie Id, Genres, Title	0.0830023

Sampling Approach - Random Forest

```
# sample data 1000 times and calculate RMSE for each model
rmse_samples <- data.frame(method = c("User Id Only", "User Id and Movie Id ", "User Id, Movie Id, Genre
    "Userd Id, Movie Id, Genres, Title"), RMSE = sapply(1:4, function(i) {
    formulas <- c("rating ~ userId", "rating ~ userId + movieId", "rating ~ userId + movieId + genres",
        "rating ~ userId + movieId + genres + title")
    mean(replicate(1000, rmse_calc(sample_n(train_set, 77, replace = TRUE), as.formula(formulas[i]))))
}))

# print RMSE values
rmse_samples %>%
    knitr::kable()
```

method	RMSE
User Id Only	1.0466735
User Id and Movie Id	1.0334554
User Id, Movie Id, Genres	0.5711593
Userd Id, Movie Id, Genres, Title	0.0767302

Run on test set

```
# sample data 1000 times and calculate RMSE for each model
rmse_samples <- data.frame(method = c("User Id Only", "User Id and Movie Id ", "User Id, Movie Id, Genre
    "Userd Id, Movie Id, Genres, Title"), RMSE = sapply(1:4, function(i) {
    formulas <- c("rating ~ userId", "rating ~ userId + movieId", "rating ~ userId + movieId + genres",
        "rating ~ userId + movieId + genres + title")
    mean(replicate(1000, rmse_calc(sample_n(test_set, 77, replace = TRUE), as.formula(formulas[i]))))
}))
```

```
# print RMSE values
rmse_samples %>%
    knitr::kable()
```

method	RMSE
User Id Only	1.0440493
User Id and Movie Id	1.0365935
User Id, Movie Id, Genres	0.5661853
Userd Id, Movie Id, Genres, Title	0.0792026

Expanding on the Model from the Course:

The model from the course used the following method. Note that here, we also have that the predictors are not treated as factors. We will expand on this idea and incorporate the results of the previous analysis.

```
rm(list = setdiff(ls(), c("edx", "final_holdout_test")))

set.seed(42)

test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index, ]
test_set <- edx[test_index, ]

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




mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512451
```

```
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.059841
```

```
predictions <- rep(2.5, nrow(test_set))
RMSE(test_set$rating, predictions)
```

```
## [1] 1.465766
```

```
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
```

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

```
# fit <- lm(rating ~ as.factor(userId), data = movielens)

mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
```

```
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu))


predicted_ratings <- mu + test_set %>%
    left_join(movie_avgs, by = "movieId") %>%
    .$b_i



model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results, data_frame(method = "Movie Effect Model",
    RMSE = model_1_rmse))

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

method	RMSE
Just the average	1.0598412
Movie Effect Model	0.9427265

```
# lm(rating ~ as.factor(movieId) + as.factor(userId))

user_avgs <- test_set %>%
    left_join(movie_avgs, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu - b_i))


predicted_ratings <- test_set %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results, data_frame(method = "Movie + User Effects Model",
    RMSE = model_2_rmse))
rmse_results %>%
    knitr::kable()
```

method	RMSE
Just the average	1.0598412
Movie Effect Model	0.9427265
Movie + User Effects Model	0.8425420

```
# lm(rating ~ as.factor(movieId) + as.factor(userId)) + genres



genre_avgs <- test_set %>%
```

```
    left_join(user_avgs, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = mean(rating - mu - b_u))


predicted_ratings <- test_set %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(genre_avgs, by = "genres") %>%
    mutate(pred = mu + b_u + b_g) %>%
    .$pred


model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results, data_frame(method = "Movie + User Effects Model + Genres",
    RMSE = model_3_rmse))


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

method	RMSE
Just the average	1.0598412
Movie Effect Model	0.9427265
Movie + User Effects Model	0.8425420
Movie + User Effects Model + Genres	0.9258154

7. RESULTS

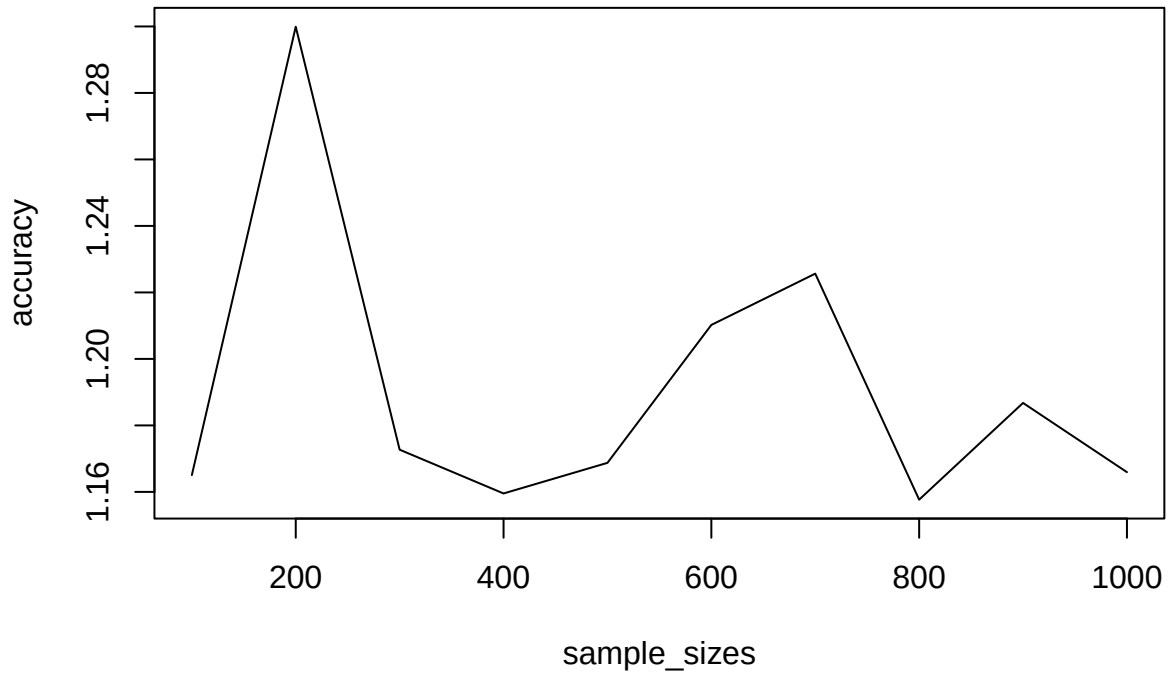
The final model for the purposes of this analysis is the model that includes the userId and movie effects and that adding genres increases the RMSE. Next, we run the model on the final holdout set. According the RMSE, the model generalizes well to new data and there is no evidence of overfitting for this particular set of predictors.

Sample Size v. Accuracy

Upon applying the model to the final hold out set, and in accordance with this study and constrained by the limitations herein, the best model is the multiple linear regression model using the aforementioned predictors.

The MovieLens dataset is extremely large. In fact, as the sample size increases, the accuracy decreases. As shown below:

Demonstrate Relationship Between Sample Size and Accuracy



As demonstrated, the accuracy appears to converge as the sample size increases. This seems to indicate that having 9,000,000 plus observations does not make the predictive power greater than a model with fewer samples from which it is built. However, it is difficult to test this using conventional methods and conventional computing power.

9. CONCLUSION

The results of the analysis indicate that of the models tested in this analysis, a multiple linear regression utilizing as predictors, user id and movie id provide the best RMSE. Below, we will demonstrate by running the modeling procedure on the final hold out data set. The sampling method seems to be contradicted by the results obtained by utilizing the full data set. That is, that utilizing the genres predictor leads to lower and more stable RMSE for both test and the train set. This contradiction arises in the the fact that the RMSE increases when genres is included in the final model using the method that utilizes the entire data set. Ideas for future studies might include the ability to use more computational power to be able to run case-wise diagnostics and utilize modeling packages that can generate reports about the statistical significance of the variables and the ability to deal with extremely large categorical variables with large numbers of levels. Ideally, the study would seek to identify the ideal (minimal and optimal) number of observations and predictors.

```
#note - I used less white space so that the results would be on page
rm(list = setdiff(ls(), c('final_holdout_test')))
set.seed(42)
mu_hat <- mean(final_holdout_test$rating)
mu_hat
```

```
## [1] 3.512033
```

```
naive_rmse <- RMSE(final_holdout_test$rating, mu_hat)
naive_rmse
```

```
## [1] 1.061202
```

```
predictions <- rep(2.5, nrow(final_holdout_test))
RMSE(final_holdout_test$rating, predictions)
```

```
## [1] 1.46641
```

```
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
# fit <- lm(rating ~ as.factor(userId), data = movielens)

mu <- mean(final_holdout_test$rating)
movie_avgs <- final_holdout_test %>%
     group_by(movieId) %>%
     summarize(b_i = mean(rating - mu))
predicted_ratings <- mu + final_holdout_test %>%
     left_join(movie_avgs, by='movieId') %>%
     .$b_i
model_1_rmse <- RMSE(predicted_ratings, final_holdout_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))
# lm(rating ~ as.factor(movieId) + as.factor(userId))
user_avgs <- final_holdout_test %>%
     left_join(movie_avgs, by='movieId') %>%
     group_by(userId) %>%
     summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- final_holdout_test %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     mutate(pred = mu + b_i + b_u) %>%
     .$pred
```

```r
model_2_rmse <- RMSE(predicted_ratings, final_holdout_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_2_rmse ))
# lm(rating ~ as.factor(movieId) + as.factor(userId)) + genres
genre_avgs <- final_holdout_test %>%
     left_join(user_avgs, by='userId') %>%
     group_by(genres) %>%
     summarize(b_g = mean(rating - mu - b_u))


predicted_ratings <- final_holdout_test %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     left_join(genre_avgs, by = 'genres') %>%
     mutate(pred = mu + b_u + b_g) %>%
     .$pred


model_3_rmse <- RMSE(predicted_ratings, final_holdout_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model + Genres",
                                     RMSE = model_3_rmse ))


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

method	RMSE
Just the average	1.0612017
Movie Effect Model	0.9383091
Movie + User Effects Model	0.8251770
Movie + User Effects Model + Genres	0.9139315