Data Science: Capstone - MovieLens Project

Shanna Jardim

10/17/2020

INTRODUCTION

Due to the wealth of information available, recommendation systems have become more relevant and have developed in recent years as the online world of e-commerce and social media gather more and more data on user's purchase patterns, user profiles, opinions, user ratings, browsing habits etc. Recommendation systems provide suggestions to users based on their likes and dislikes, recommending items they want to buy or services they actually want to subscribe to. Major companies such a LinkedIn, Amazon, Netflix and Takealot utilise recommendation systems.

The Netflix Prize In 2006 , Netflix ran an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films

GOAL OF PROJECT

This MovieLens projects aims to create a movie rating predictor using the **edx** dataset - which is created in the given code. The recommender system should be able to predict a users rating on a new movie. The Root Mean Square Error (RMSE) will be used to evaluate the accuracy of the predictions to the true value in the **validation** set - which 10% of the full data set and is created in the given code

GIVEN CODE

The work on this project needs to build on code that is already provided which I will not include in this Document

EXPLORING THE DATA

In this section I will explore the data to uncover initial patterns, characteristics and points of interest and familiarize myself with information before doing any changes.

```
# Check for any #NA Values
anyNA(edx)
```

```
## [1] FALSE
```

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

- **Observations**
- The release year needs to be separated from the movie title
- Timestamp format needs to be converted , this represents the rating date
- The genres are in the same column need to be separated by the pipe "|"
- The same movie entry might belong to more than one genre.
- Every distinct rating by a user is on a different row.
- UserId, movieId are: quantitative - Discrete unique numbers.
- Title and genres are: qualitative and not unique.

```
unique(edx$rating)
```

```
##  [1] 5.0 3.0 2.0 4.0 4.5 3.5 1.0 1.5 2.5 0.5
```

There are 10 different rate scores. a rate is give by one user for one movie.

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The rate mean 3.512 , Minimum rating is 1, Max is 5

```
edx %>% summarize(n_users = n_distinct(userId) , n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

Distinct number of users = 69878 and distinct number of movies = 10677.

```
head(validation)
```

```
##    userId movieId rating timestamp
## 1:      1     231      5 838983392
## 2:      1     480      5 838983653
## 3:      1     586      5 838984068
## 4:      2     151      3 868246450
## 5:      2     858      2 868245645
## 6:      2    1544      3 868245920
##                                                    title
## 1:                                    Dumb & Dumber (1994)
## 2:                                    Jurassic Park (1993)
## 3:                                       Home Alone (1990)
## 4:                                          Rob Roy (1995)
## 5:                                   Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                     genres
## 1:                                  Comedy
## 2:        Action|Adventure|Sci-Fi|Thriller
## 3:                         Children|Comedy
## 4:                Action|Drama|Romance|War
## 5:                             Crime|Drama
## 6: Action|Adventure|Horror|Sci-Fi|Thriller
```

The validation set is the same format & contains the same columns as our training set and therefore we will perform the same data transformation on both training and test datasets

TRANSFORMATION OF THE DATA

String Extract String Extract to extract the release year from title and store in a separate field

```
edx <- edx %>%
  mutate(releaseyear = as.numeric(str_extract(str_extract(title,
                                             "[/(]\\d{4}[/)]$"), regex("\\d{4}"))))

# Do the same for the Validation Dataset
validation <- validation %>%
  mutate(releaseyear = as.numeric(str_extract(str_extract(title,
                                           "[/(]\\d{4}[/)]$"), regex("\\d{4}"))))
```

Timestamp Change Change the format of the rate timestamp to date and store only the year.

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

```r
#get the rate year
edx$timestamp <- format(edx$timestamp, "%Y")

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

Age of movie Add the Age of the movie (This year minus releasyear).

```r
edx <- edx %>% mutate(movie_age = 2020 - releaseyear)

validation <- validation %>% mutate(movie_age = 2020 - releaseyear)
```

View the changes

```r
head(edx)
```

```
##    userId movieId rating timestamp                          title
## 1:      1     122      5      1996               Boomerang (1992)
## 2:      1     185      5      1996                Net, The (1995)
## 3:      1     292      5      1996                Outbreak (1995)
## 4:      1     316      5      1996                Stargate (1994)
## 5:      1     329      5      1996 Star Trek: Generations (1994)
## 6:      1     355      5      1996        Flintstones, The (1994)
##                           genres releaseyear movie_age
## 1:                Comedy|Romance        1992        28
## 2:         Action|Crime|Thriller        1995        25
## 3:  Action|Drama|Sci-Fi|Thriller        1995        25
## 4:       Action|Adventure|Sci-Fi        1994        26
## 5: Action|Adventure|Drama|Sci-Fi        1994        26
## 6:       Children|Comedy|Fantasy        1994        26
```

```r
head(validation)
```

```
##    userId movieId rating timestamp
## 1:      1     231      5      1996
## 2:      1     480      5      1996
## 3:      1     586      5      1996
## 4:      2     151      3      1997
## 5:      2     858      2      1997
## 6:      2    1544      3      1997
##                                                     title
## 1:                                   Dumb & Dumber (1994)
## 2:                                   Jurassic Park (1993)
## 3:                                      Home Alone (1990)
## 4:                                         Rob Roy (1995)
## 5:                                  Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                 genres releaseyear movie_age
## 1:                              Comedy        1994        26
## 2:        Action|Adventure|Sci-Fi|Thriller        1993        27
```

```
## 3:                          Children|Comedy        1990        30
## 4:                   Action|Drama|Romance|War        1995        25
## 5:                              Crime|Drama        1972        48
## 6: Action|Adventure|Horror|Sci-Fi|Thriller        1997        23
```

VISUALISING THE DATA

Genre Analysis

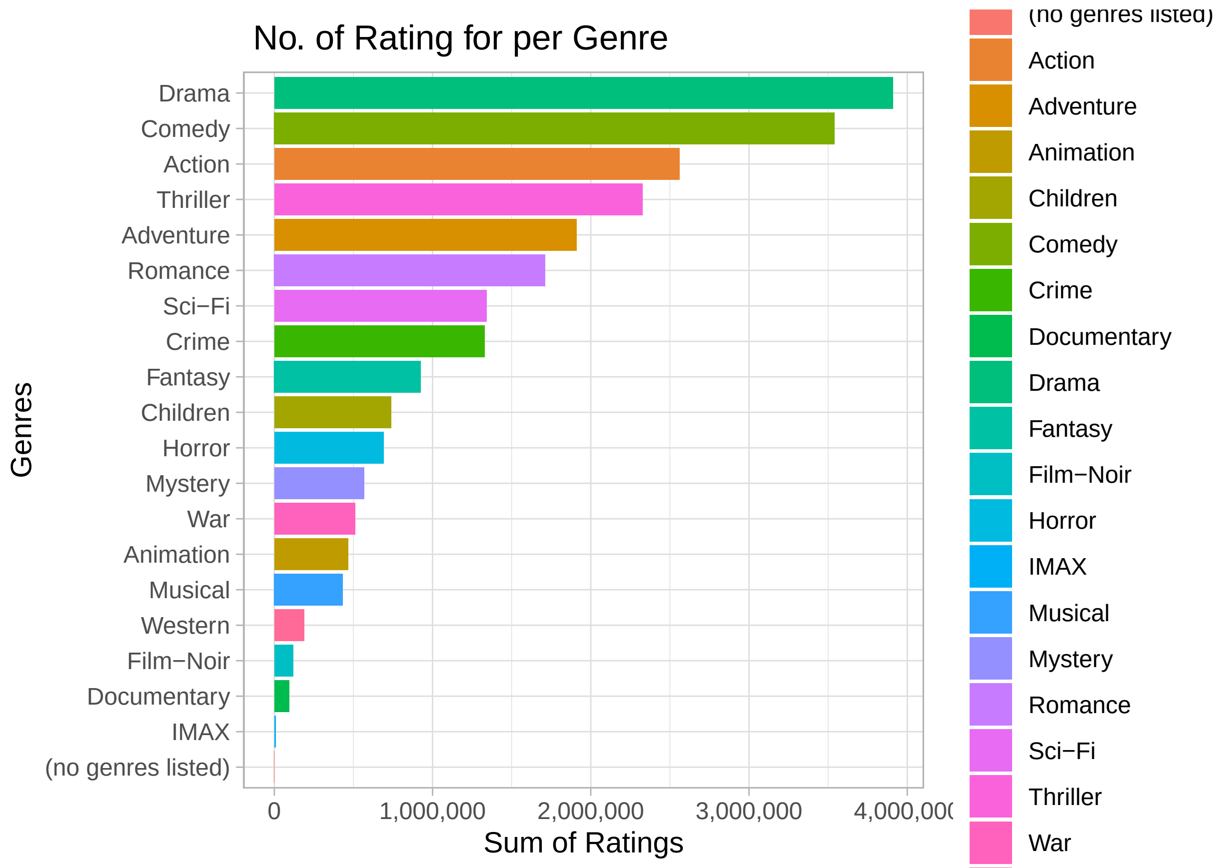
Creation of a new dataframe with useful measures to identify outliers and further analyse the data

```
edx_genre_measures  <- edx %>%
  separate_rows(genres, sep = "\\|") %>% group_by(genres) %>%
  summarize(Ratings_perGenre_Sum = n(),
            Ratings_perGenre_Avg = mean(rating),
            Movies_perGenre_Sum = n_distinct(movieId),
            Users_perGenre_Sum = n_distinct(userId))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
edx_genre_measures[order(-edx_genre_measures$Movies_perGenre_Sum), ]
```

```
## # A tibble: 20 x 5
##    genres   Ratings_perGenre~ Ratings_perGenr~ Movies_perGenre~ Users_perGenre_~
##    <chr>                <int>            <dbl>            <int>            <int>
##  1 Drama              3910127             3.67             5336            69866
##  2 Comedy             3540930             3.44             3703            69864
##  3 Thriller           2325899             3.51             1705            69567
##  4 Romance            1712100             3.55             1685            69530
##  5 Action             2560545             3.42             1473            69607
##  6 Crime              1327715             3.67             1117            68691
##  7 Adventu~           1908892             3.49             1025            69521
##  8 Horror              691485             3.27             1013            60695
##  9 Sci-Fi             1341183             3.40              754            68469
## 10 Fantasy             925637             3.50              543            66833
## 11 Children            737994             3.42              528            64059
## 12 War                 511147             3.78              510            64892
## 13 Mystery             568332             3.68              509            61845
## 14 Documen~             93066             3.78              481            24295
## 15 Musical             433080             3.56              436            58918
## 16 Animati~            467168             3.60              286            59018
## 17 Western             189394             3.56              275            47648
## 18 Film-No~            118541             4.01              148            31270
## 19 IMAX                  8181             3.77               29             6393
## 20 (no gen~                 7             3.64                1                7
```



* **Observations** +19 distinct genres +Drama is the most rated genre = 3 910 127 +Comedy & Action – 2nd & 3rd highest +IMAX least amount of ratings – could be due to the time of this dataset - IMAX was still new +There are also movies that have no genres +Genre may be used as a good predictor but will look at genre over the release year

Outlier - One movie with no genre with only 7 users who rated this is considered an outlier If using Genre as a predictor removing this row may yield better results.

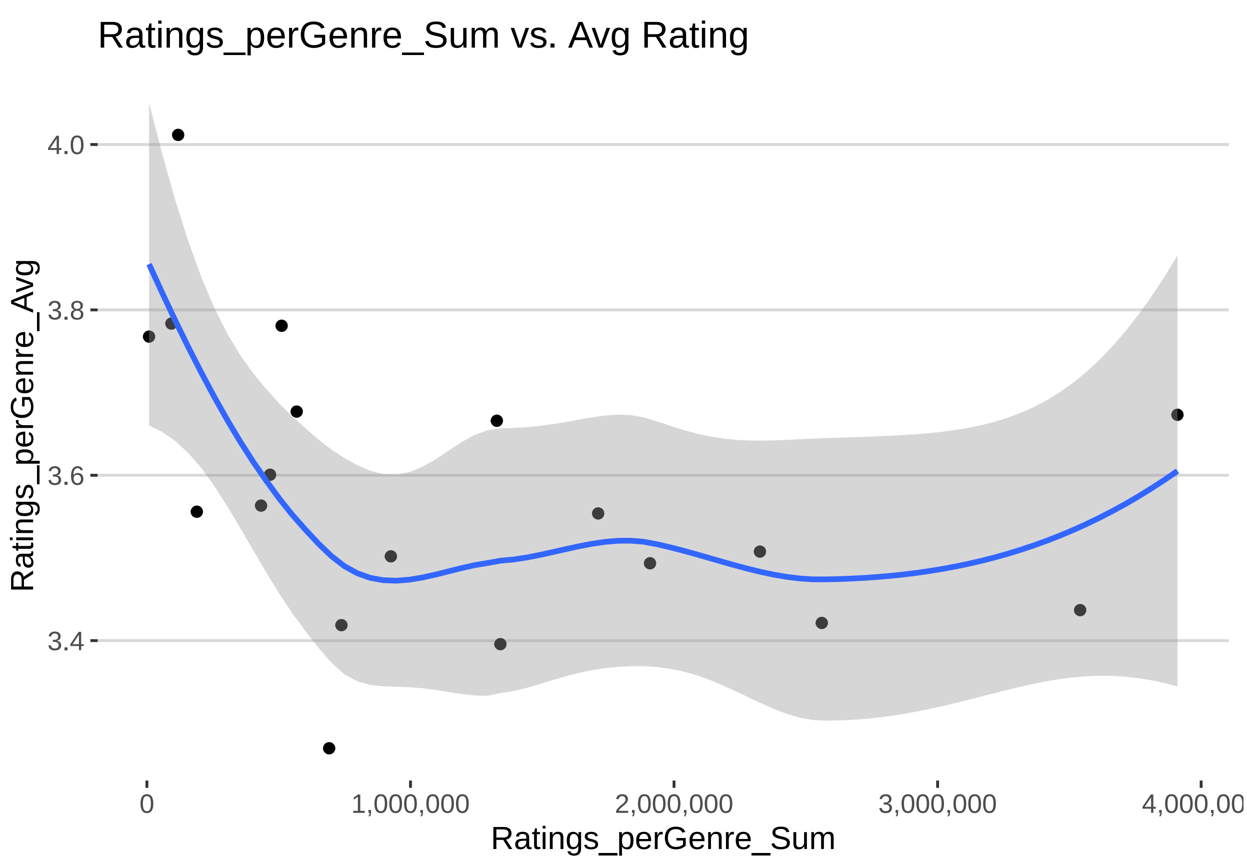
```
edx_genre_measures <- subset(edx_genre_measures, genres != "(no genres listed)")
edx_genre_measures
```

```
## # A tibble: 19 x 5
##    genres   Ratings_perGenre~ Ratings_perGenre~ Movies_perGenre~ Users_perGenre_~
##    <chr>                <int>             <dbl>            <int>            <int>
##  1 Action             2560545              3.42             1473            69607
##  2 Advent~            1908892              3.49             1025            69521
##  3 Animat~             467168              3.60              286            59018
##  4 Childr~             737994              3.42              528            64059
##  5 Comedy             3540930              3.44             3703            69864
##  6 Crime              1327715              3.67             1117            68691
##  7 Docume~              93066              3.78              481            24295
##  8 Drama              3910127              3.67             5336            69866
##  9 Fantasy             925637              3.50              543            66833
## 10 Film-N~             118541              4.01              148            31270
## 11 Horror              691485              3.27             1013            60695
## 12 IMAX                  8181              3.77               29             6393
## 13 Musical             433080              3.56              436            58918
```

```
## 14 Mystery            568332          3.68          509          61845
## 15 Romance           1712100          3.55         1685          69530
## 16 Sci-Fi            1341183          3.40          754          68469
## 17 Thrill~           2325899          3.51         1705          69567
## 18 War               511147           3.78          510          64892
## 19 Western           189394           3.56          275          47648
```

Some genres have very low sums of ratings – Will check the correlation between sums of rating and the rating mean

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



We can clearly see in this graph that the lower the Sum of Ratings the higher the Average Rate.

Outlier These values can be treated as outliers as there is a slight bias due to not enough people rating the movie, however I will not remove this data as it is needed

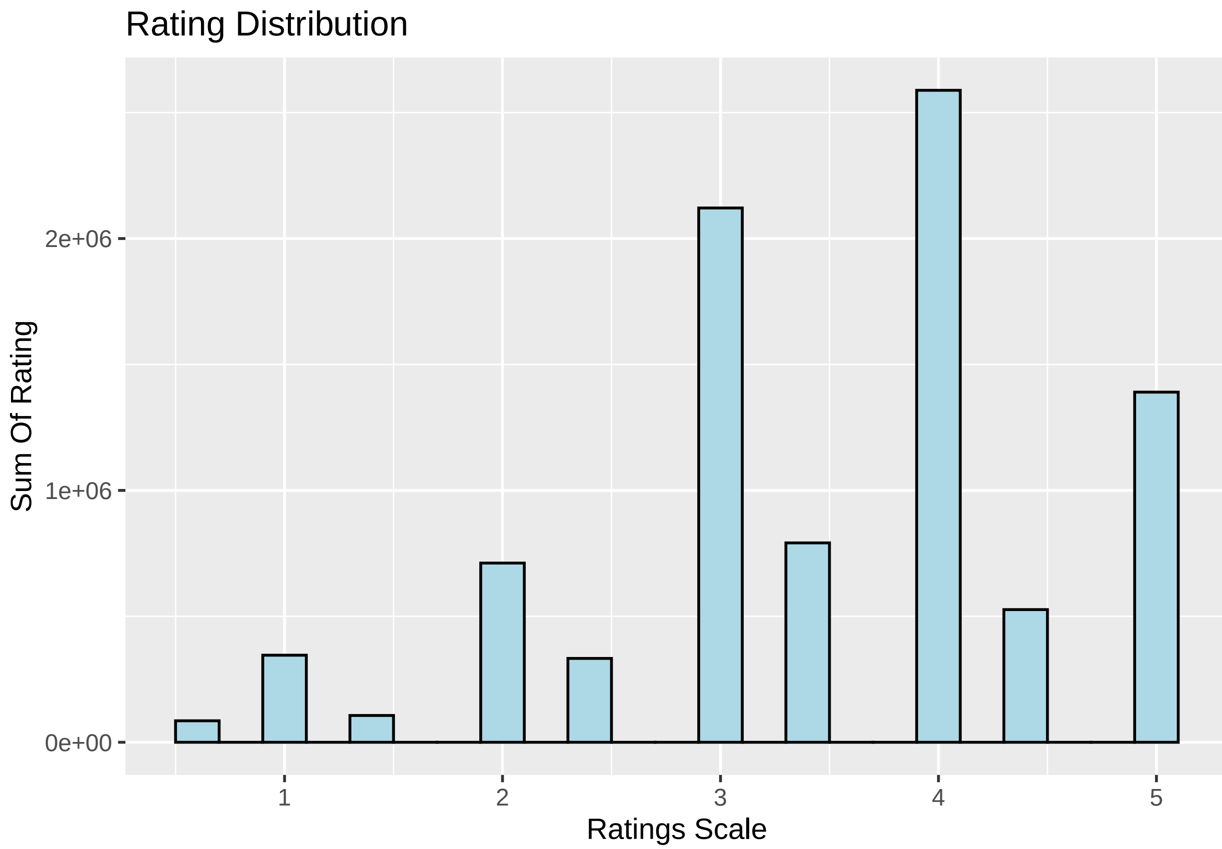
```
#Create same metrix for the validation set.

validation_genre_measures <- validation %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(Ratings_perGenre_Sum = n(),
            Ratings_perGenre_Avg = mean(rating),
            Movies_perGenre_Sum = n_distinct(movieId),
            Users_perGenre_Sum = n_distinct(userId))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
validation_genre_measures[order(-validation_genre_measures$Movies_perGenre_Sum), ]
```

```
## # A tibble: 19 x 5
##    genres  Ratings_perGenre~ Ratings_perGenre~ Movies_perGenre~ Users_perGenre_~
##    <chr>               <int>             <dbl>            <int>            <int>
##  1 Drama              434071              3.67             4835            60897
##  2 Comedy             393138              3.44             3468            59664
##  3 Thrill~            258536              3.50             1615            54653
##  4 Romance            189783              3.55             1586            49043
##  5 Action             284804              3.42             1404            55342
##  6 Crime              147242              3.66             1044            44893
##  7 Advent~            212182              3.49              974            51860
##  8 Horror              76740              3.26              949            28103
##  9 Sci-Fi             149306              3.40              713            43535
## 10 Fantasy            102845              3.50              523            37249
## 11 Childr~             82155              3.42              507            32680
## 12 Mystery             62612              3.68              481            28507
## 13 War                 56916              3.77              453            29050
## 14 Musical             48094              3.56              407            24314
## 15 Docume~             10388              3.78              406             6805
## 16 Animat~             51944              3.59              275            25649
## 17 Western             21065              3.55              244            14262
## 18 Film-N~             13051              4.02              137             9132
## 19 IMAX                  899              3.74               27              871
```

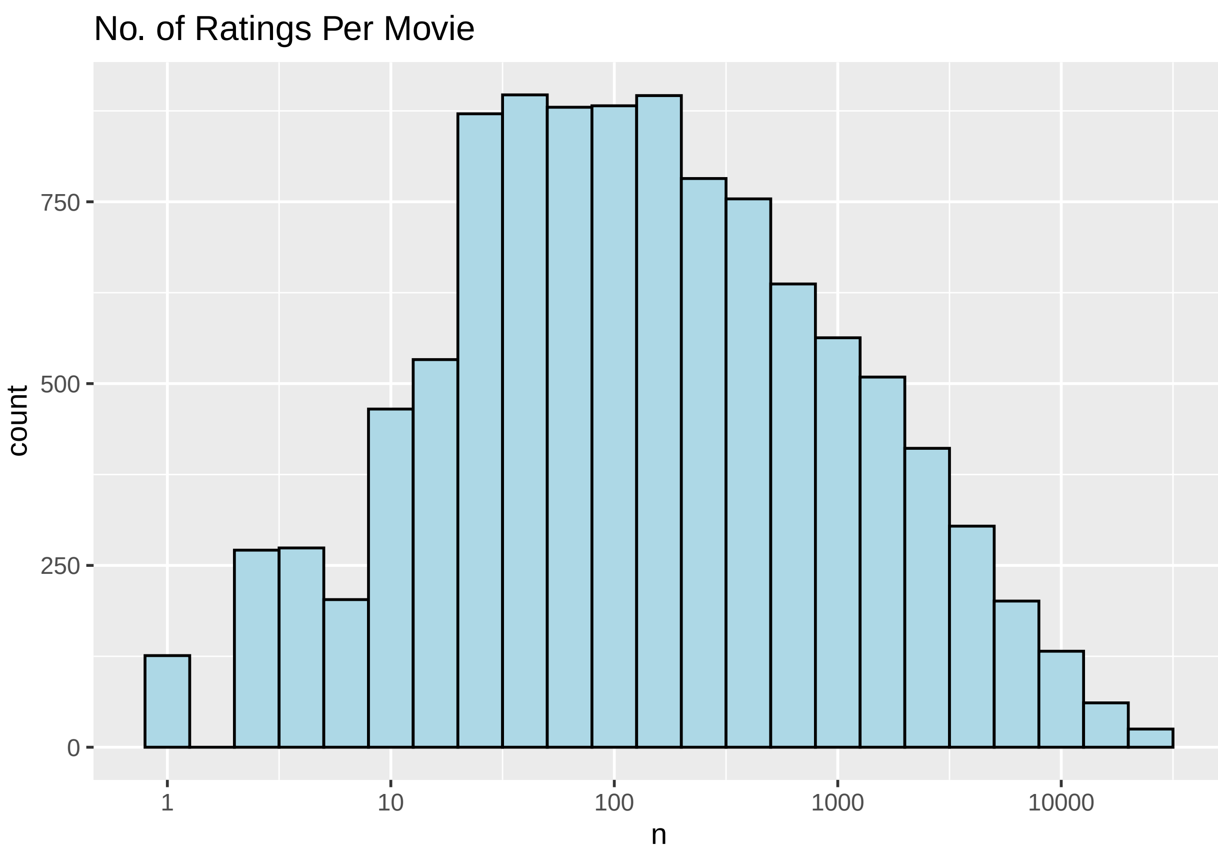
Rating Distribution



Movie Dimension

Number of rating per movies

```
edx %>% count(movieId) %>% ggplot(aes(n))+
  geom_histogram(binwidth=0.2 ,color = "black" , fill= "light blue")+
  scale_x_log10()+
  ggtitle("No. of Ratings Per Movie")+
  theme_gray()
```



Some movies get rated more than others, indicating some movies may be more popular than others. I will add this to the training set and test set – Number of ratings per movie

```
edx <- edx %>% group_by(movieId) %>% mutate(Users_perMovie  = n())
validation <- validation %>% group_by(movieId) %>% mutate(Users_perMovie = n())
```

I will add the average Rating per Movie to the Dataset.

```
# Add the average rating per movie for each row
edx <- edx %>% group_by(movieId) %>% mutate(Avg_rating_per_movie = mean(rating))
validation <- validation %>% group_by(movieId) %>% mutate(Avg_rating_per_movie = mean(rating))
```

User Dimension

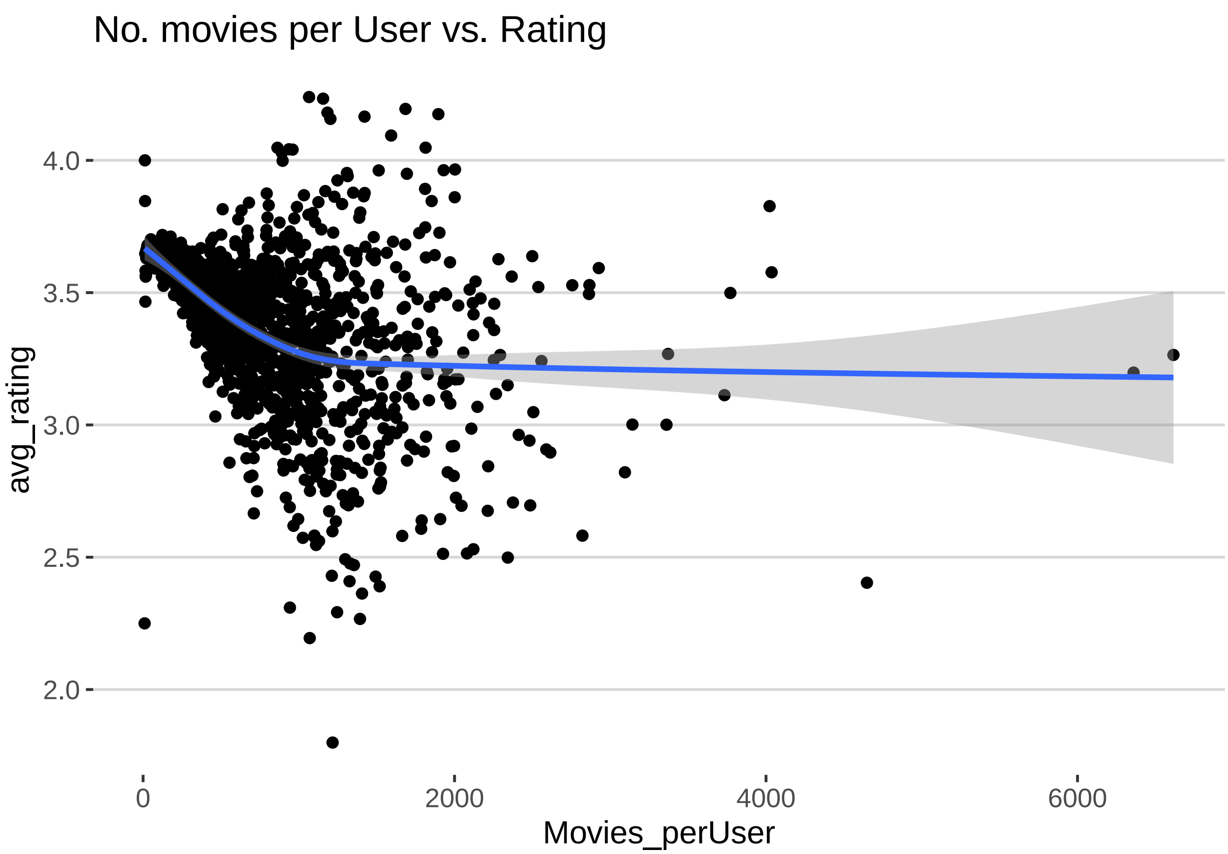
We could penalize users with low number of reviews. I will add this measure to the training set and test set - Number of Ratings per user

```
edx <- edx %>% group_by(userId) %>% mutate(Movies_perUser = n())
validation <- validation %>% group_by(userId) %>% mutate(Movies_perUser = n())
```

Number of rating per user

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

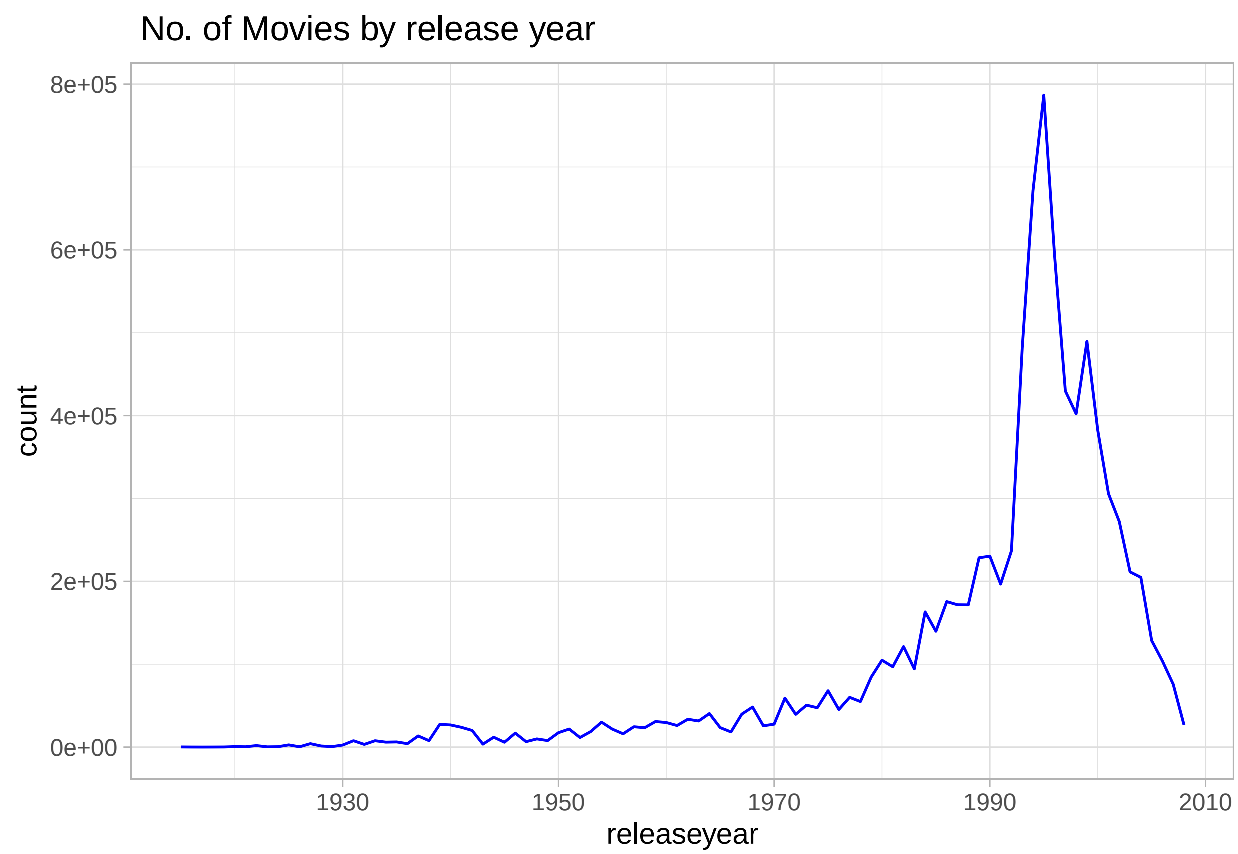


As mentioned earlier - We can see the lower the number of users per movie the higher the rating . There is a slight bias due to not enough people rating the movie

Release Year

No. of movies per year

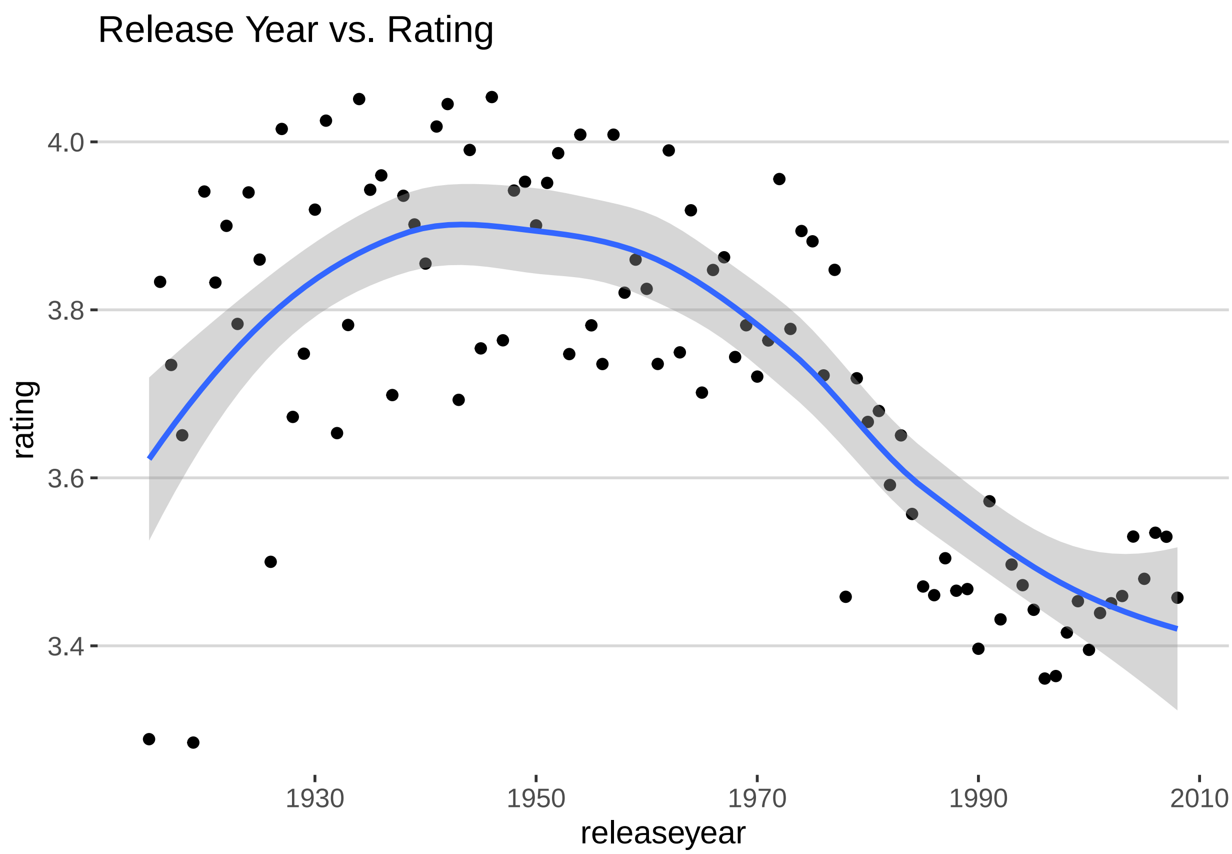
```
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
```



View release year vs rating

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

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



Older "classics" get higher ratings. This could allow us to penalize a movie based on release year by a calculated weight.

Age of Movie Analysis

Let's check if there is a correlation between average rating per movie and age of movie.

```
avg_rating_per_age <- edx %>%
    group_by(movie_age) %>% summarize(avg_rating_by_age = mean(rating))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

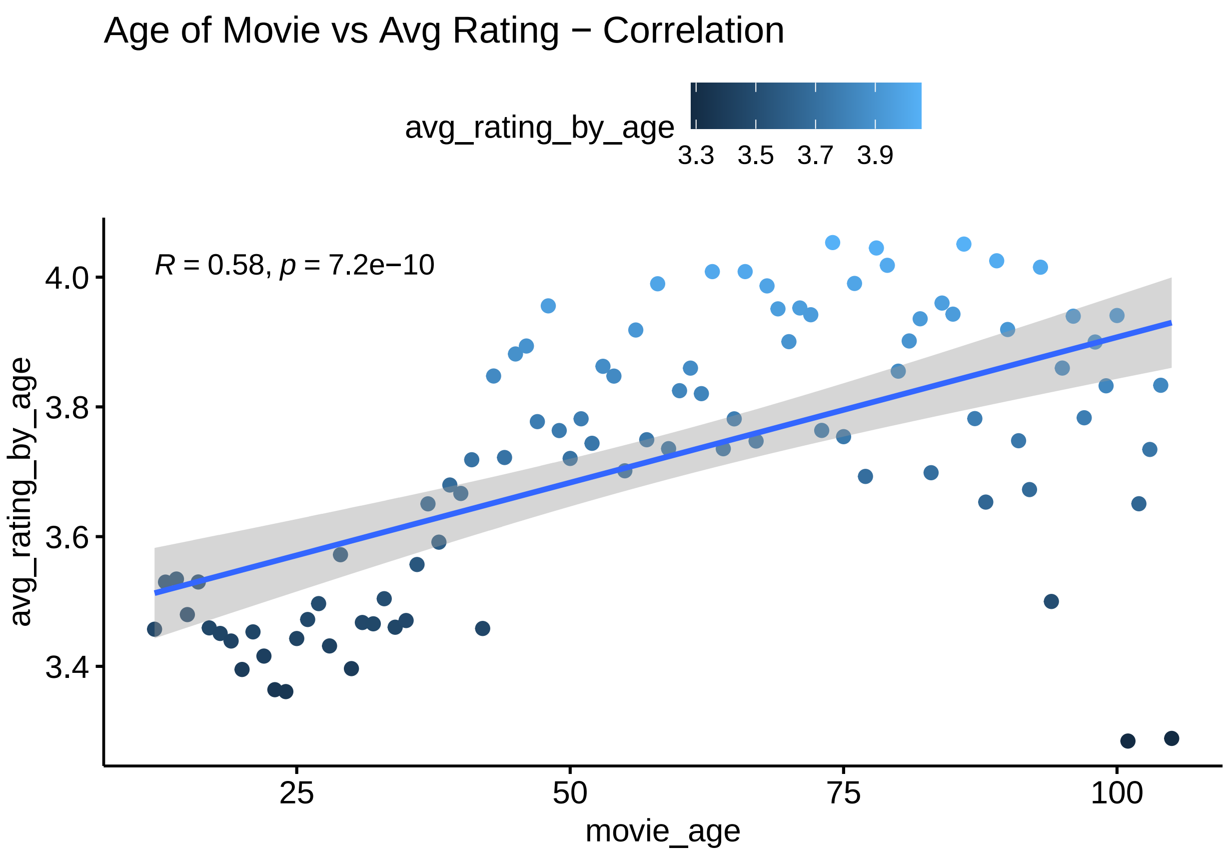
```
avg_rating_per_age
```

```
## # A tibble: 94 x 2
##    movie_age avg_rating_by_age
##        <dbl>             <dbl>
##  1        12              3.46
##  2        13              3.53
##  3        14              3.53
##  4        15              3.48
##  5        16              3.53
##  6        17              3.46
##  7        18              3.45
##  8        19              3.44
```

```
## 9          20                  3.40
## 10         21                  3.45
## # ... with 84 more rows
```

```
## Warning: package 'ggpubr' was built under R version 3.6.3
```

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



We can clearly notice that there is a positive trend. The older the movie the higher the ratings it receives. The age dimension will definitely have affect on predicting the rating. We can remove the release year as it is no longer needed.

```
edx <- subset(edx, select = -c(releaseyear) )
validation <- subset(validation, select = -c(releaseyear) )
```

Year Rated Dimension

Let's check if there is a correlation between The year the movie was rated and the average rating.

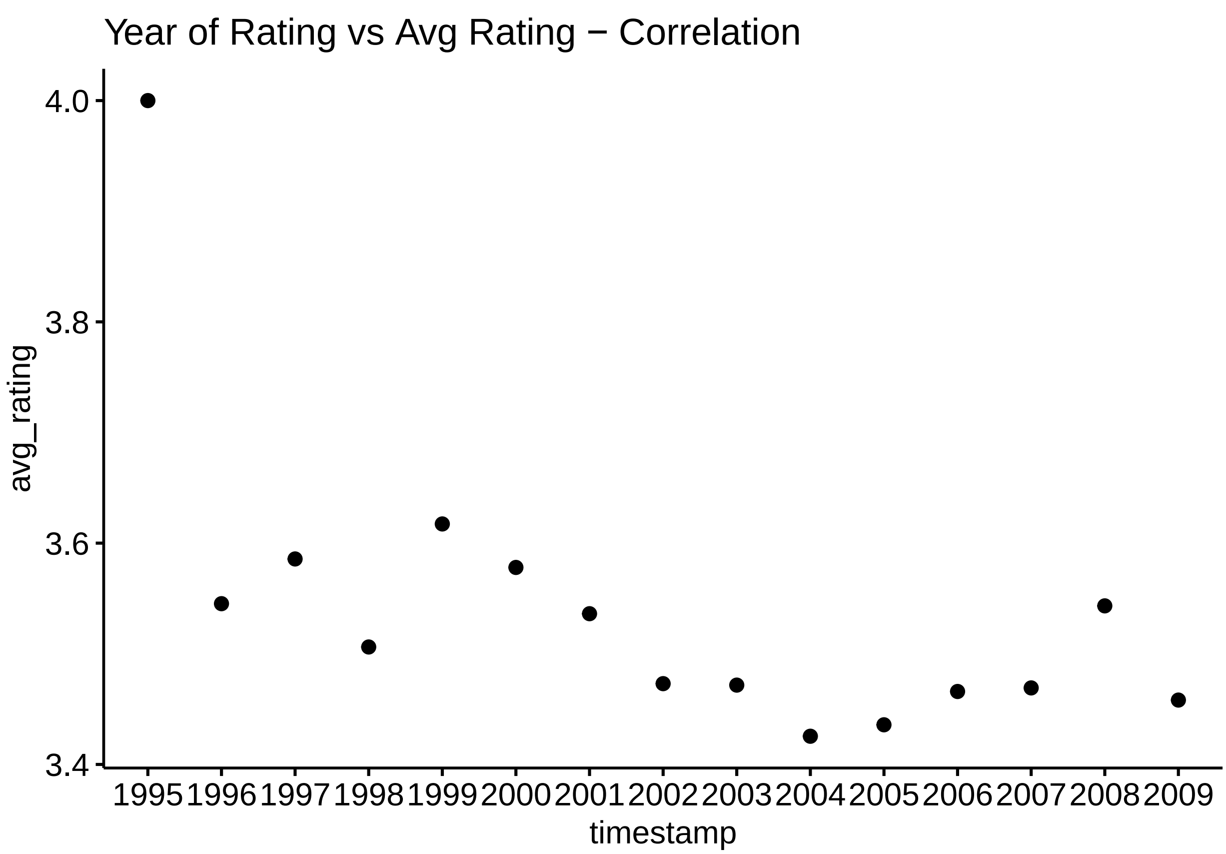
```
avg_rating_per_timestamp_year <- edx %>%
    group_by(timestamp) %>% summarize(avg_rating = mean(rating))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
avg_rating_per_timestamp_year
```

```
## # A tibble: 15 x 2
##    timestamp avg_rating
##    <chr>          <dbl>
##  1 1995            4
##  2 1996            3.55
##  3 1997            3.59
##  4 1998            3.51
##  5 1999            3.62
##  6 2000            3.58
##  7 2001            3.54
##  8 2002            3.47
##  9 2003            3.47
## 10 2004            3.43
## 11 2005            3.44
## 12 2006            3.47
## 13 2007            3.47
## 14 2008            3.54
## 15 2009            3.46
```

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



We observe that the oldest rating was in 1995 given the highest rating – this is also an outlier. There is a slight downward trend with the remaining of the movies , showing that the older the rating the higher the avg rating.

DATA CLEANING

Remove entries with no genres

```
#remove entries with no genres
edx_clean <- subset(edx, genres != "(no genres listed)")
```

Remove entries with timestamp year =1995

```
edx_clean <- subset(edx_clean, timestamp != "1995")
```

MODEL BUILDING & TRAINING

Creating the training and testing datasets The course instructors provided a segment of code in order to download and clean the MovieLens 10M dataset. Originally, the given code separated the dataset into two subsets: the edx dataset, for training, and the validation dataset, for testing the final algorithm.

It is important to note the instructions given: *IMPORTANT: Make sure you do NOT use the validation set (the final hold-out test set) to train your algorithm. The validation set (the final hold-out test set) should ONLY be used to test your final algorithm. You should split the edx data into a training and test set or use cross-validation.*

From the exploration analysis, it is also evident that the MovieLens dataset has a great deal of data, therefore the approach I take needs to ensure that my machine does not run out memory when running the algorithms and regression models to predict movie ratings.

```
#Partition the edx data
library(caret)
set.seed(1)
test_index <- createDataPartition(y = edx_clean$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx_clean[-test_index,]
test_set <- edx_clean[test_index,]
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
#to make sure we don't include users and movies in the test set that do not appear in
#the training set, we remove these entries using the semi_join function:
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

RMSE Before we proceed with the model building,training and validation we define the RMSE function

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

MODEL 1 - Simple Linear Regression *"Simple linear regression is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables"* ~ STATS ONLINE

The first model is remarkably simple. Let us assume a linear equation across all movies and apply an Average rating regardless of the movie, user , genre or release year. No bias are considered.

```r
## Get Ave Rate across all movies
Pred_1 <- mean(train_set$rating)
Pred_1
```

```
## [1] 3.512381
```

```r
## Average Rate across all movies (3.512) is used as a baseline
Rmse_1 <- RMSE(test_set$rating,Pred_1)
Rmse_1
```

```
## [1] 1.059797
```

The result RMSE for this model will not return a very accurate prediction, however it will be used as a starting point in which to better all other models.

MODEL 2 - Multilinear Regression *"Multiple linear regression (MLR) is a statistical technique that uses several explanatory variables to predict the outcome of a response variable".* ~ INVESTOPEDIA

The Multilinear Regression model will use all elements to predict the rating.

```r
MLRregressor = lm(formula = rating ~ movieId + userId + movie_age + Users_perMovie + Avg_rating_per_movi
                  data = train_set)
summary(MLRregressor)
```

```
##
## Call:
## lm(formula = rating ~ movieId + userId + movie_age + Users_perMovie +
##     Avg_rating_per_movie + Movies_perUser, data = train_set)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9594 -0.5534  0.0810  0.6621  4.0264
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           6.873e-02  2.752e-03   24.97   <2e-16 ***
## movieId               5.858e-07  4.312e-08   13.59   <2e-16 ***
## userId                1.960e-07  1.703e-08   11.51   <2e-16 ***
## movie_age             2.957e-04  2.869e-05   10.30   <2e-16 ***
## Users_perMovie       -1.670e-06  6.061e-08  -27.55   <2e-16 ***
## Avg_rating_per_movie  9.919e-01  8.384e-04 1183.16   <2e-16 ***
## Movies_perUser       -1.116e-04  6.863e-07 -162.62   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9408 on 7200028 degrees of freedom
## Multiple R-squared:  0.213,  Adjusted R-squared:  0.213
## F-statistic: 3.248e+05 on 6 and 7200028 DF,  p-value: < 2.2e-16
```

From the P value, we can see that all dimension used for the Regressor are highly statistically significant.

"Most authors refer to statistically significant as P < 0.05 and statistically highly significant as P < 0.001 (less than one in a thousand chance of being wrong)."- *REFERENCE https://www.statsdirect.com/help/basics/p_values.htm*

```
## Average Rate across all movies (3.512) is used as a baseline
Pred_2 = predict(MLRregressor, newdata = test_set)
Rmse_2 <- RMSE( test_set$rating,Pred_2)
Rmse_2
```

```
## [1] 0.9399841
```

The RMSE is a lot lower that the first model, however I am going to still try other methods to get a lower RMSE.

MODEL 3 - K-Nearest Neighbors (K-NN) *"An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small)"* ~ WIKIPEDIA

In this example we are going to use only 2 dimensions to predict the ratings

```
## Average Rate across all movies (3.512) is used as a baseline
KNN_dataset = edx_clean %>% select(1,2,7,8,9,10,3)
```

```
# Splitting the dataset into the Training set and Test set
KNN_test_index <- createDataPartition(y = KNN_dataset$rating, times = 1, p = 0.2, list = FALSE)
KNN_train_set <- KNN_dataset[-test_index,]
KNN_test_set <- KNN_dataset[test_index,]
#to make sure we don't include users and movies in the test set that do not appear in
#the training set, we remove these entries using the semi_join function:
KNN_test_set <- KNN_test_set %>%
  semi_join(KNN_train_set, by = "movieId") %>%
  semi_join(KNN_train_set, by = "userId")

# Feature Scaling

KNN_train_set[-7] = scale(KNN_train_set[-7])
KNN_test_set[-7] = scale(KNN_test_set[-7])

# Fitting K-NN to the Training set and Predicting the Test set results

#train = KNN_train_set[, -7]
#cl=KNN_train_set[, 7]
#length(cl)
#length(train)
#library(class)
#Pred_3 = knn(train = KNN_train_set[, -7, drop = FALSE],
#             test = KNN_test_set[, -7, drop = FALSE],
#             cl = KNN_train_set$rating,
#              k = 5,
##             prob = TRUE)
# Rmse_3 <- RMSE( test_set$rating,Pred_3)
```

Unfortunately my machine did not have enough memory to run this model :-(as well as Polynomial Regression model & Random Forest.

MODEL 4 - Matrix Factorization (MF) *"Matrix factorization is a class of collaborative filtering algorithms used in recommender systems. Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices."* ~ WIKIPEDIA

With some research, Matrix Factorization is one method most used for Recommendation Systems.
In comparison to the K-nearest-neighbours method, Matrix Factorization is often better in terms of prediction accuracy and time needed to train the model. The recosystem Package in R is specifically used for recommendation systems.

```
# install.packages("recosystem")
library(recosystem)
```

```
## Warning: package 'recosystem' was built under R version 3.6.3
```

```
set.seed(123)

train_set_fac <- train_set %>% select(movieId, userId, rating, movie_age)
test_set_fac <- test_set %>% select(movieId, userId, rating, movie_age)

train_set_fac <- as.matrix(train_set_fac)
test_set_fac <- as.matrix(test_set_fac)

write.table(train_set_fac, file = "edx_train_set.txt", sep = " ",
            row.names = FALSE, col.names = FALSE)

write.table(test_set_fac, file = "edx_test_set.txt", sep = " ",
            row.names = FALSE, col.names = FALSE)

set.seed(1)
MF_train_dataset <- data_file("edx_train_set.txt")
MF_test_dataset <- data_file("edx_test_set.txt")

# Create a model object (a Reference Class object in R) by calling Reco().
recommender <- Reco()

# Call the $tune() method to select best tuning parameters along a set of candidate values.
opts = recommender$tune(MF_train_dataset, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
        costp_l1 = 0, costq_l1 = 0, nthread = 1, niter = 10))
opts
```

```
## $min
## $min$dim
## [1] 30
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.1
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.01
##
## $min$lrate
```

```
## [1] 0.1
##
## $min$loss_fun
## [1] 0.8068518
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.8311176
## 2   20        0     0.01        0     0.01   0.1 0.8215116
## 3   30        0     0.01        0     0.01   0.1 0.8314526
## 4   10        0     0.10        0     0.01   0.1 0.8335750
## 5   20        0     0.10        0     0.01   0.1 0.8105330
## 6   30        0     0.10        0     0.01   0.1 0.8068518
## 7   10        0     0.01        0     0.10   0.1 0.8313237
## 8   20        0     0.01        0     0.10   0.1 0.8137179
## 9   30        0     0.01        0     0.10   0.1 0.8132296
## 10  10        0     0.10        0     0.10   0.1 0.8406085
## 11  20        0     0.10        0     0.10   0.1 0.8336802
## 12  30        0     0.10        0     0.10   0.1 0.8323411
## 13  10        0     0.01        0     0.01   0.2 0.8271885
## 14  20        0     0.01        0     0.01   0.2 0.8825051
## 15  30        0     0.01        0     0.01   0.2 1.0185222
## 16  10        0     0.10        0     0.01   0.2 0.8278344
## 17  20        0     0.10        0     0.01   0.2 0.8092234
## 18  30        0     0.10        0     0.01   0.2 0.8102185
## 19  10        0     0.01        0     0.10   0.2 0.8269285
## 20  20        0     0.01        0     0.10   0.2 0.8696412
## 21  30        0     0.01        0     0.10   0.2 0.9203626
## 22  10        0     0.10        0     0.10   0.2 0.8430007
## 23  20        0     0.10        0     0.10   0.2 0.8298019
## 24  30        0     0.10        0     0.10   0.2 0.8286388
```

```
# Train the model by calling the $train() method.
# A number of parameters can be set inside the function, possibly coming from the result of $tune().

recommender$train(MF_train_dataset, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9948   1.0067e+007
##    1       0.8782   8.0860e+006
##    2       0.8460   7.5029e+006
##    3       0.8243   7.1524e+006
##    4       0.8075   6.9035e+006
##    5       0.7945   6.7205e+006
##    6       0.7837   6.5832e+006
##    7       0.7745   6.4726e+006
##    8       0.7663   6.3768e+006
##    9       0.7593   6.2999e+006
##   10       0.7530   6.2354e+006
##   11       0.7475   6.1765e+006
##   12       0.7425   6.1266e+006
##   13       0.7379   6.0845e+006
##   14       0.7338   6.0453e+006
```

```
##   15       0.7300  6.0113e+006
##   16       0.7266  5.9805e+006
##   17       0.7233  5.9534e+006
##   18       0.7204  5.9276e+006
##   19       0.7177  5.9032e+006
```

```
## Use the $predict() method to compute predicted values.
## Write predictions to file
pred_file <- tempfile()
recommender$predict(MF_test_dataset, out_file(pred_file))
```

```
## prediction output generated at C:\Users\SHANNA~1\AppData\Local\Temp\RtmpiWtc8G\file2624743e7bfa
```

```
print(scan(pred_file, n = 20))
```

```
##  [1] 4.06956 5.14192 5.47930 4.91553 4.60456 4.47660 4.84132 4.53395 3.82264
## [10] 2.57443 2.75795 3.11943 2.90980 4.13798 3.50208 3.73649 4.06693 4.12891
## [19] 3.51474 3.88011
```

```
edx_test_ratings <- read.table("edx_test_set.txt", header = FALSE, sep = " ")$V3

pred_ratings <- scan(pred_file)

# will calculate RMSE
Rmse_4 <- RMSE(edx_test_ratings, pred_ratings)
Rmse_4
```

```
## [1] 0.7896134
```

```
#Final Test on validation Set
validation_fac <- validation %>% select(movieId, userId, rating, movie_age)
validation_fac <- as.matrix(validation_fac)

write.table(validation_fac, file = "validation_set.txt", sep = " ",
            row.names = FALSE, col.names = FALSE)

MF_validation_dataset <- data_file("validation_set.txt")
recommender <- Reco()

recommender$train(MF_train_dataset, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9968  1.0073e+007
##    1       0.8799  8.0909e+006
##    2       0.8473  7.5112e+006
##    3       0.8254  7.1585e+006
##    4       0.8090  6.9101e+006
##    5       0.7958  6.7334e+006
##    6       0.7847  6.5922e+006
##    7       0.7752  6.4755e+006
##    8       0.7670  6.3813e+006
```

```
##    9        0.7598  6.3052e+006
##   10        0.7535  6.2376e+006
##   11        0.7477  6.1794e+006
##   12        0.7427  6.1309e+006
##   13        0.7381  6.0851e+006
##   14        0.7339  6.0464e+006
##   15        0.7302  6.0125e+006
##   16        0.7266  5.9811e+006
##   17        0.7234  5.9529e+006
##   18        0.7205  5.9280e+006
##   19        0.7178  5.9051e+006
```

```
pred_file <- tempfile()
recommender$predict(MF_validation_dataset, out_file(pred_file))
```

```
## prediction output generated at C:\Users\SHANNA~1\AppData\Local\Temp\RtmpiWtc8G\file2624d6b1cc5
```

```
real_ratings <- read.table("validation_set.txt", header = FALSE, sep = " ")$V3
pred_ratings <- scan(pred_file)
Final_Rmse <- RMSE(real_ratings, pred_ratings)
Final_Rmse
```

```
## [1] 0.7900151
```

We observe that the RMSE is much lower than all other models used. Therefore, the Matrix factorization may be the best approach to create a recommendation system.

CONCLUSION

After trying a few different approaches (Polynominal Regression, Random Forest) I have come to the conclusion that, due to the size of the data, some algorithms were very resource heavy and unable to run. I believe a machine used for machine learning would probably be equipped with better resources. However, the Recosystem is a fairly good choice for the MovieLens dataset and yielded the lowest RMSE compared to other algorithims.

Further investigations and appling algorithms with more dimension that could be added to the dataset, may yield better results such as: **Such as** + Genre + Budget of movie + User demographics - age, gender, interests etc. + Director, Actors.

Thank you for taking the time to read my report

REFERENCES

https://www.statsdirect.com/help/basics/p_values.htm

https://towardsdatascience.com/understanding-matrix-factorization-for-recommender-systems-4d3c5e67f2c9

https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html

https://www.csie.ntu.edu.tw/~cjlin/papers/libmf/mf_adaptive_pakdd.pdf