MovieLens Project

Tatsiana Sokalava

6/20/2020

Introduction and Overview

To gain competitive edge and preserve their market share, companies such as YouTube, Amazon, and Netflix continue to use critical data insights to stand out among their competitors. That includes developing and implementing the recommender systems to ignite consumer satisfaction and engagement yielding significant profits to the company. This study incorporates data generated by the GroupLens research lab that contains over 20 million ratings for over 27,000 movies by more than 138,000 users. A smaller subset of this data - available on http://files.grouplens.org/datasets/movielens/ml-10m.zip - is used for this analysis and contains 10,000,054 ratings with over 6 variables. The goal of this study is to build a model that predicts movie ratings. The performance of the model will be measured by its value of the root mean square error (RMSE). The key steps involved in this analysis are exploring and wrangling the data, understanding key components and variable importance of the data, preprocessing of the data, testing the models and tuning parameters, and building the final model that yields the lowest RMSE.

The main challenge of the recommender system problem is rather complicated due to high dimensionality of the data. Each outcome in this problem is defined by a different set of predictors. For instance, when a rating for a movie is predicted by a user, all previous ratings for that movie and that user can become predictors. In addition, there is a similarity factor between the movies or users that accounts for outcome variations.

Let's begin by loading the libraries.

```
library(tidyverse)
library(lubridate)
library(dslabs)
library(dplyr)
library(data.table)
library(matrixStats)
library(caret)
library(recosystem)
```

Downloading, wrangling and partitioning the data

The Data Science Capstone course provides the following code to download, wrangle and partition the data into two separate sets: edx and validation. The validation set represents 10% of the downloaded data.

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

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

```r
# Validation set will be 10% of MovieLens data

set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = 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)
```

Since the data was partitioned into the edx and validation sets at random, we begin the exploration of the data on the edx set only as it would yield similar results since proportions and variability of the data is preserved.

Exploration

The edx data frame contains 9,000,055 objects (rows) and 6 variables (columns) with each row representing a rating given by a user to a movie.

```r
str(edx)
```

There is no NA values in the dataset, which otherwise need to be accounted for.

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

```
## [1] 0
```

There are 69,878 records of distinct users and 10,677 of distinct movies.

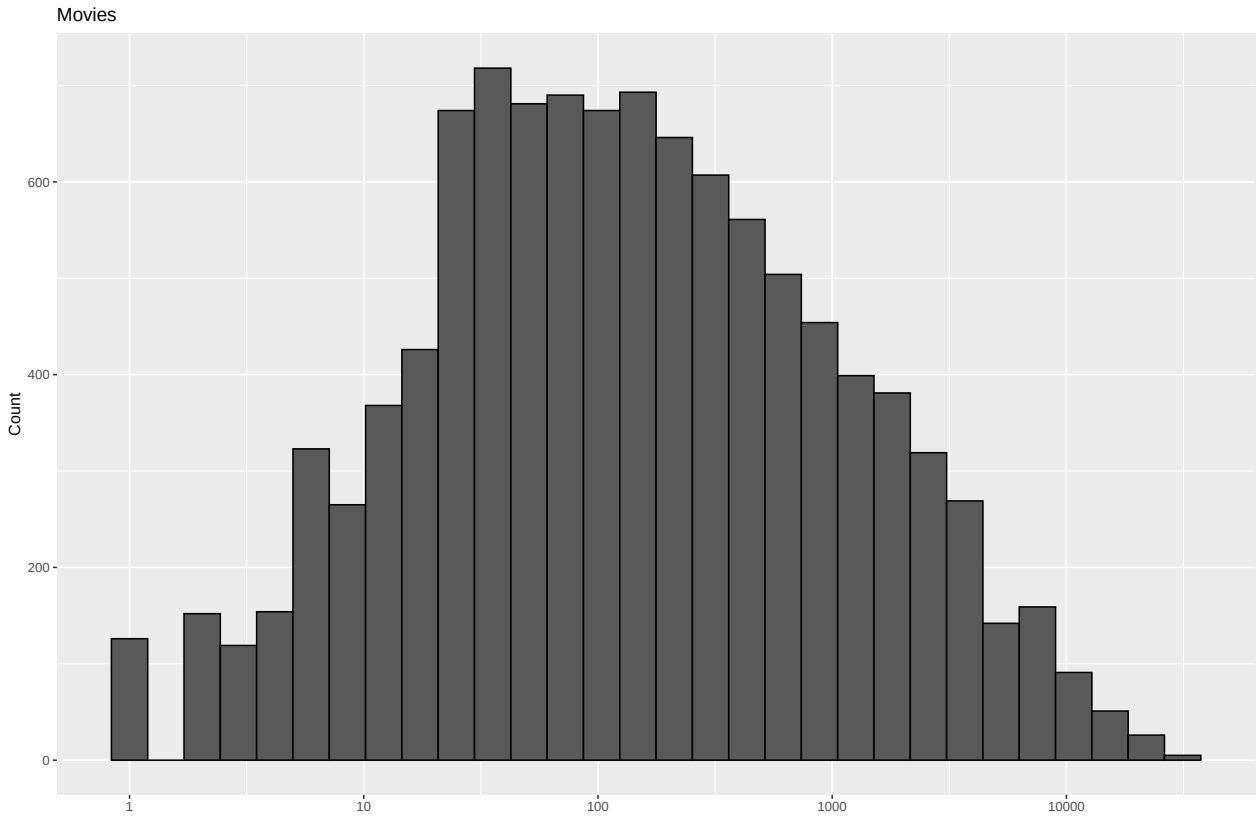
```r
edx%>%summarize(n_users = n_distinct(userId),n_movies = n_distinct(movieId))
```

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

Let's look at some general properties of the data by visualizing it.

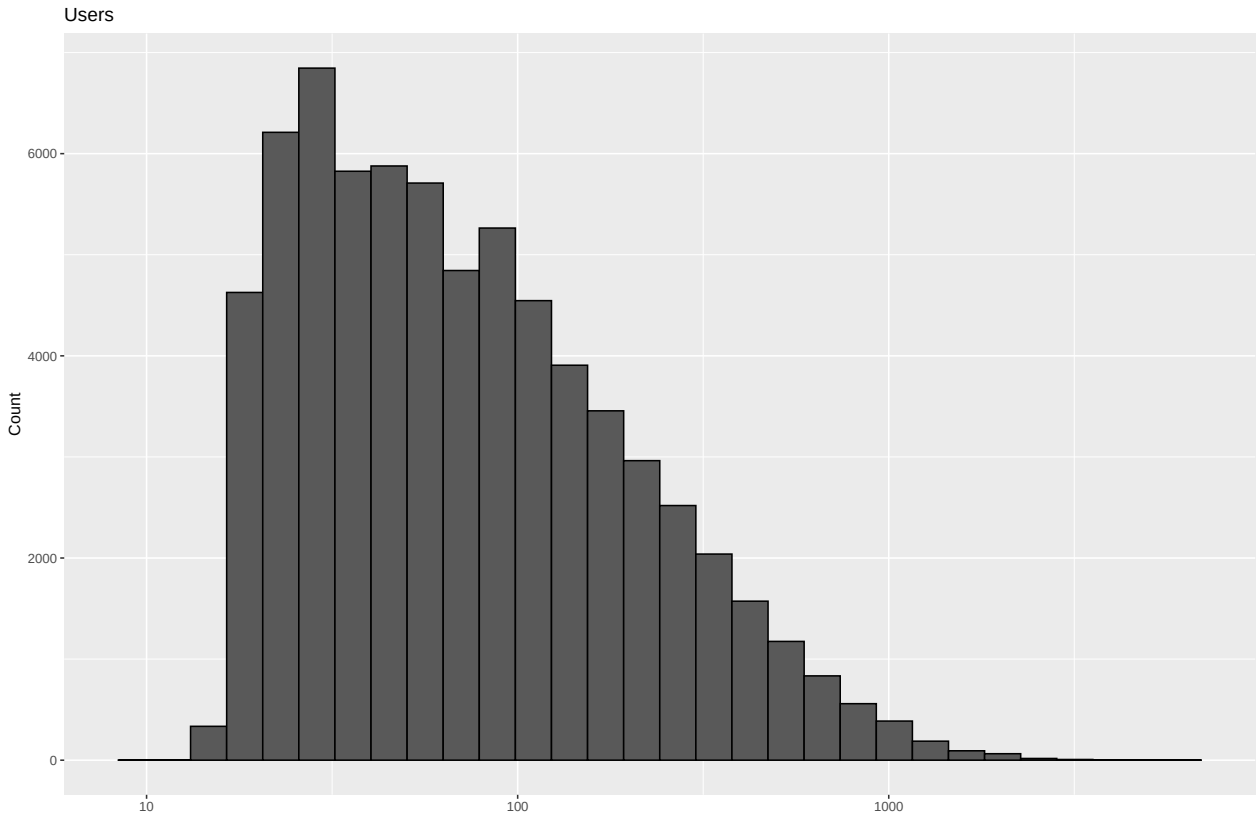
From the following graph, it's evident that there are movies that get rated more frequently then others.

```r
edx %>% dplyr::count(movieId)%>%ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() + labs(title="Movies", x="", y="Count")
```



Likewise, there are some users that rate more often than others.

```
edx %>% dplyr::count(userId)%>%ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() + labs(title="Users", x="", y="Count")
```



From the following table, you can see that there more ratings in some years than others. The data spans from 1995 (with only two data points) to 2009. Year 2000 and 2005 are the highest rated years with over a million ratings each.

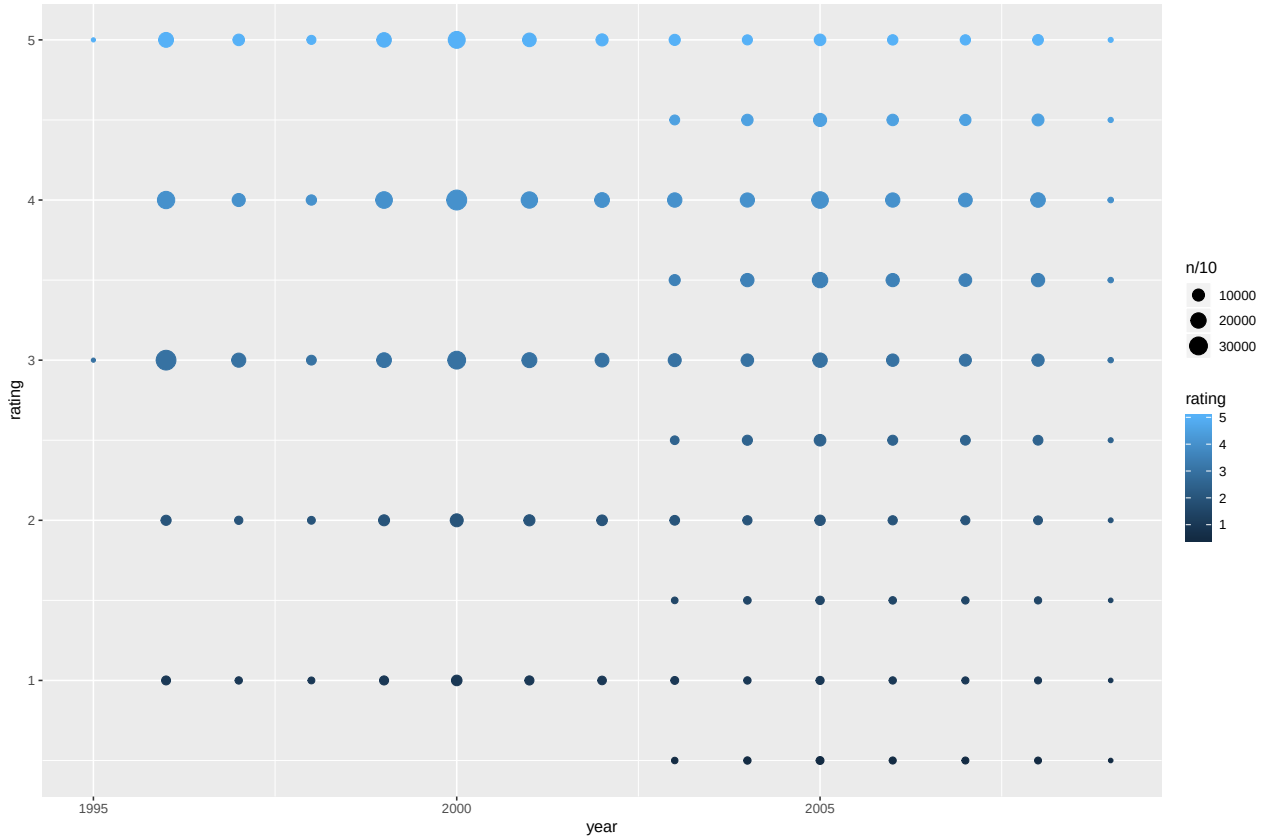
```
edx %>% mutate(timestamp=as_datetime(timestamp)) %>% mutate(Year=year(timestamp)) %>%
  group_by(Year) %>% summarize(Average=round(mean(rating), digits=2), N=n())%>%
  arrange(Year)%>% data.frame() %>%
  cbind(.,Overall_Average=round(mean(edx$rating), digits=2)) %>%
  mutate(Spread=Average-Overall_Average)
```

```
##    Year Average       N Overall_Average Spread
## 1  1995    4.00       2            3.51   0.49
## 2  1996    3.55  942772            3.51   0.04
## 3  1997    3.59  414101            3.51   0.08
## 4  1998    3.51  181634            3.51   0.00
## 5  1999    3.62  709893            3.51   0.11
## 6  2000    3.58 1144349            3.51   0.07
## 7  2001    3.54  683355            3.51   0.03
## 8  2002    3.47  524959            3.51  -0.04
## 9  2003    3.47  619938            3.51  -0.04
## 10 2004    3.43  691429            3.51  -0.08
## 11 2005    3.44 1059277            3.51  -0.07
## 12 2006    3.47  689315            3.51  -0.04
## 13 2007    3.47  629168            3.51  -0.04
## 14 2008    3.54  696740            3.51   0.03
## 15 2009    3.46   13123            3.51  -0.05
```

Next, let's plot each movie rating against year, with dot size representing the number of ratings in each group. Notably, while data spans from 1995 to 2009, half-ratings are only introduced in 2003 giving users

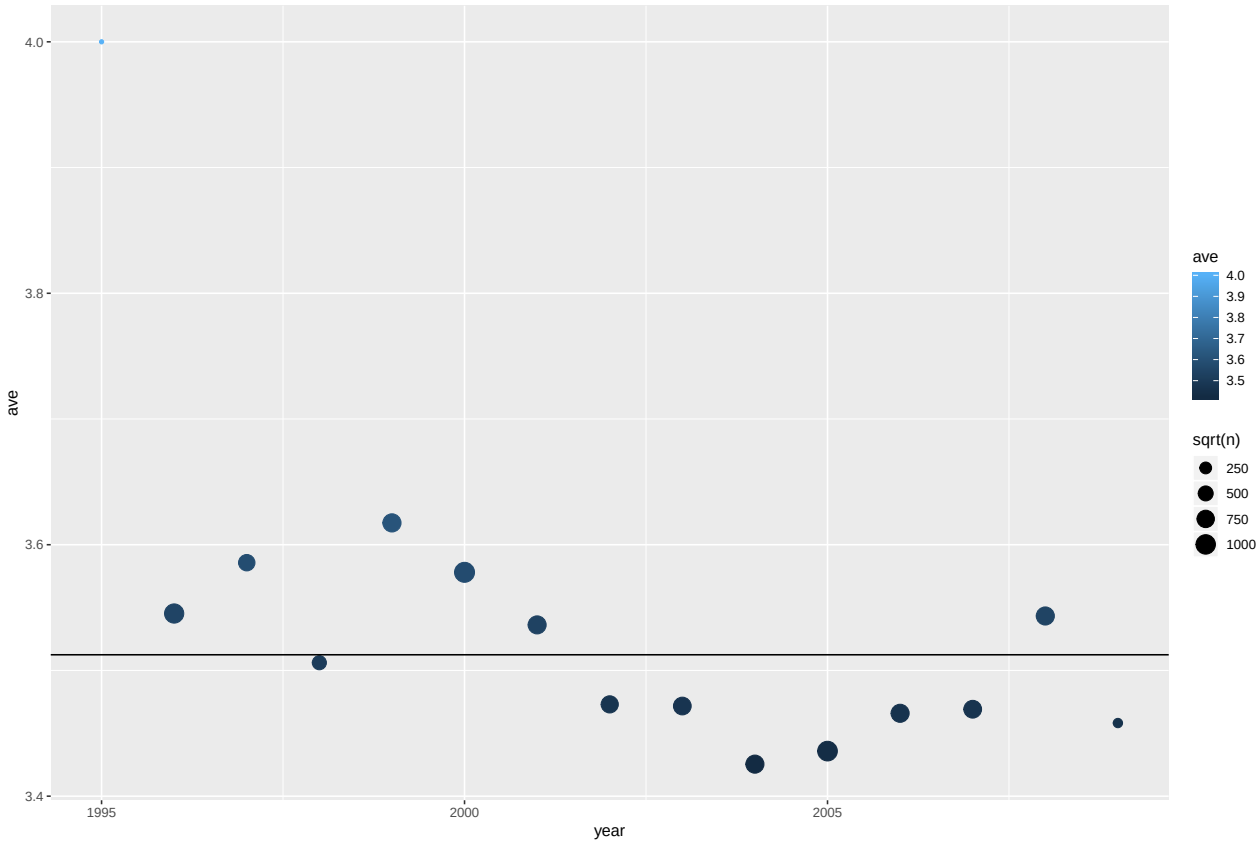
more flexibility to express their opinion about a movie.

```
edx %>% mutate(timestamp=as_datetime(timestamp)) %>% mutate(year=year(timestamp)) %>%
  group_by(year, rating) %>%summarize(n = n()) %>% arrange(year) %>%
  ggplot()+geom_point(aes(year,rating, color=rating, size=n/10))
```



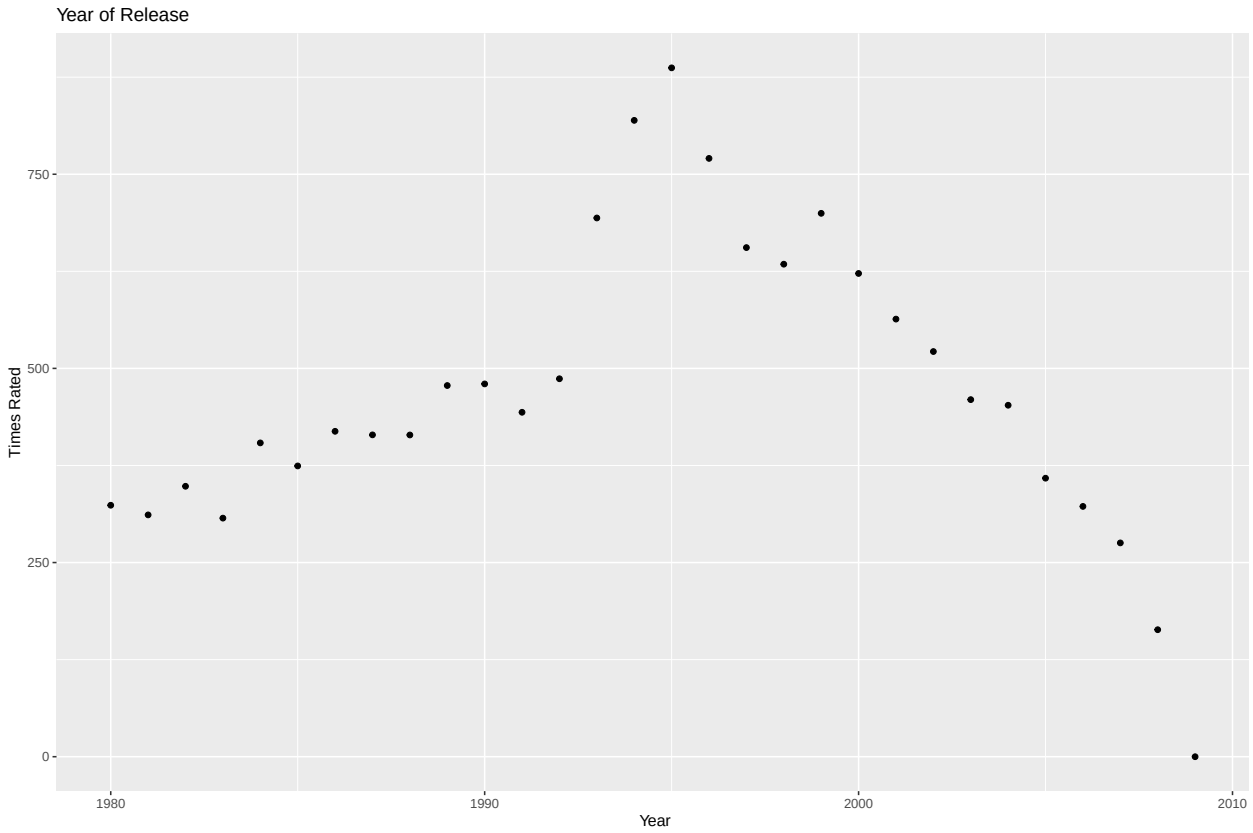
While half movie ratings are introduced in 2003, 2004 has the lowest average rating among all years in the data set, which can be an indication that users were not satisfied with the recommended movie options during that period. All-in-all, the span of 2002 to 2007 had the lowest yearly averages.

```
edx %>% mutate(timestamp=as_datetime(timestamp)) %>% mutate(year=year(timestamp)) %>%
  group_by(year) %>% summarize(ave=mean(rating), n=n()) %>% arrange(year) %>%
  ggplot(aes(year,ave,size=sqrt(n), color=ave ))+geom_point() +
  geom_hline(yintercept = mean(edx$rating))
```



Based on the year of release, we can summarize the number of ratings each movie received. To depict the trend, the chart below visualizes only movies relaesed in 1980 or newer. The most rated movies appear to be released in the mid-90s, with the highest number of ratings for movies released in 1995. This assumption holds when the entire data set is evaluated.

```
years<-seq(1980, 2009)
nrating_year_released<-sapply(as.character(years), function(t){
  sum(str_detect(edx$title, t))
})
data.frame(Year_Released=c(1980:2009),
           N_Ratings=nrating_year_released, row.names = NULL) %>%
  ggplot(aes(Year_Released, sqrt(N_Ratings))) + geom_point() +
  labs(title="Year of Release", x=" Year", y="Times Rated")
```



Let's look closer at the movies released in 1995. There are 362 movies released that year. These movies received a total of 786,762 ratings at an average of 3.4. Not surprisingly, the top rated movies are blockbusters such as "Braveheart (1995)" with 26,212 ratings, "Apollo 13" with 24,284, and "Toy Story (1995)" with 23,790. The plot of 1995 released movies indicates a trend that the higher the number of ratings are, the higher the average rating is. Notably, there is much higher variance in ratings for movies that are not rated that often.

```
ind95<-which(str_detect(edx$title, "1995")==TRUE)
# Summary of 1995 movies
data.frame(NReviews=nrow(edx[ind95,]),Ave=mean(edx[ind95,]$rating))
```

```
##   NReviews     Ave
## 1   786762 3.44288
```

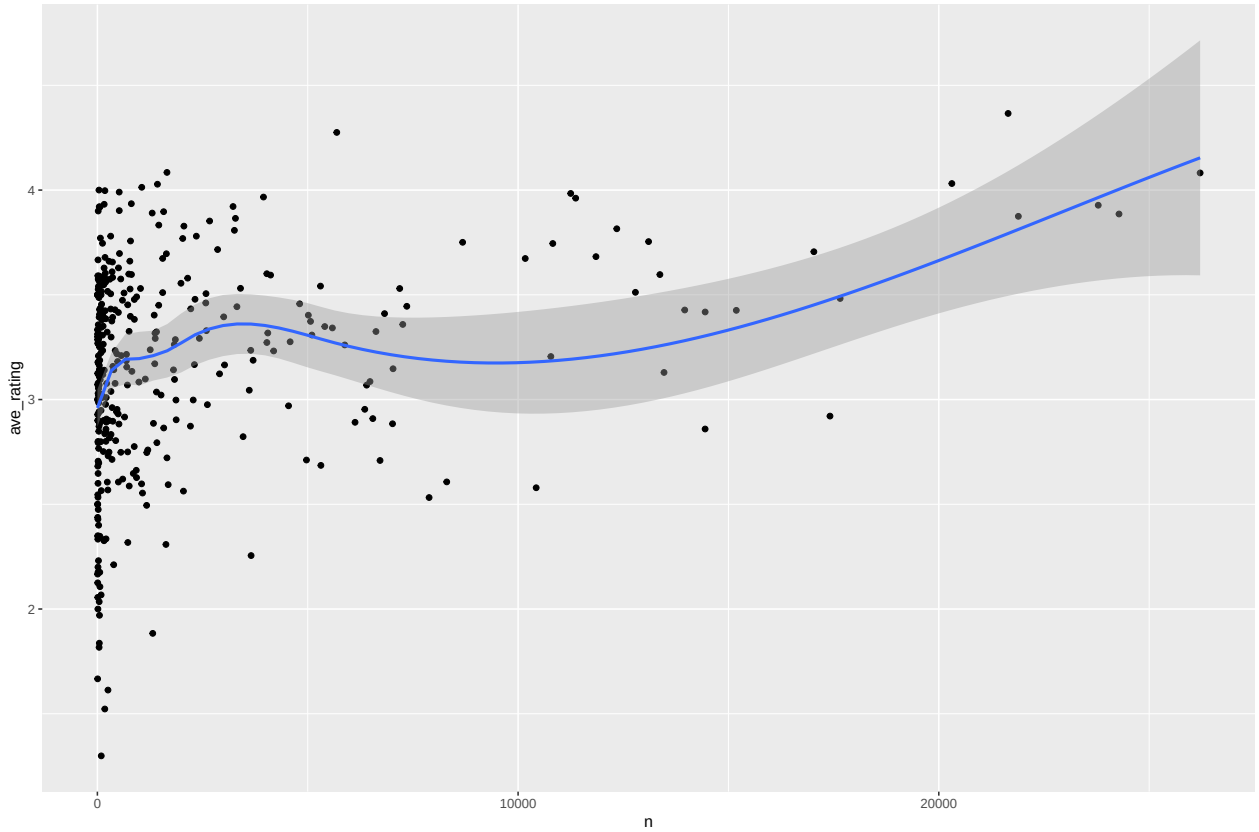
```
edx$title[ind95] %>% n_distinct()
```

```
## [1] 362
```

```
# Top rated in 1995
edx95<-edx[ind95,] %>% group_by(movieId)%>%summarize(n=n(), ave_rating=mean(rating)) %>%
  arrange(desc(n)) %>% data.frame()
# Top 5 rated movies
unique(edx$title[edx$movieId==edx95[1:5,1]])
```

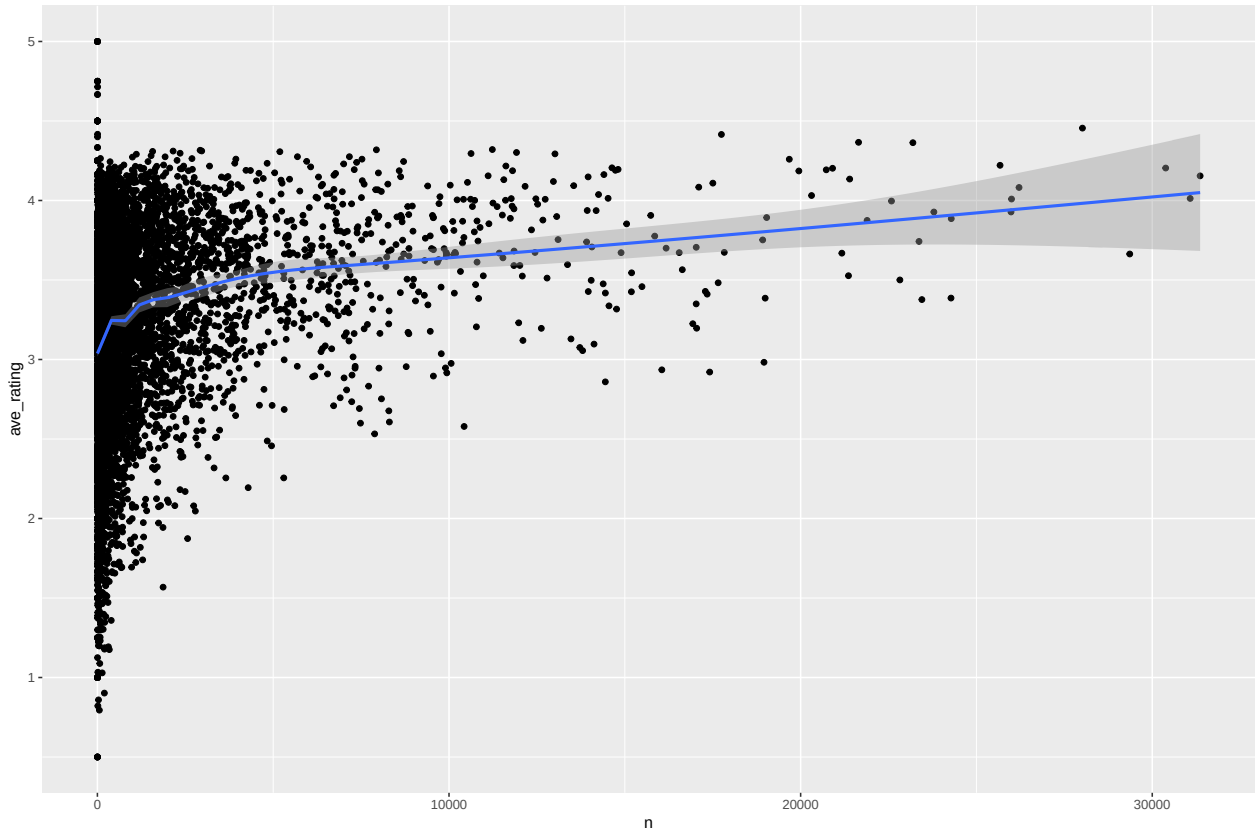
```
## [1] "Toy Story (1995)"
## [2] "12 Monkeys (Twelve Monkeys) (1995)"
## [3] "Usual Suspects, The (1995)"
## [4] "Apollo 13 (1995)"
## [5] "Braveheart (1995)"
```

```
# Plot of movie ratings in 1995
edx95 %>% ggplot(aes(n, ave_rating))+geom_point()+geom_smooth()
```



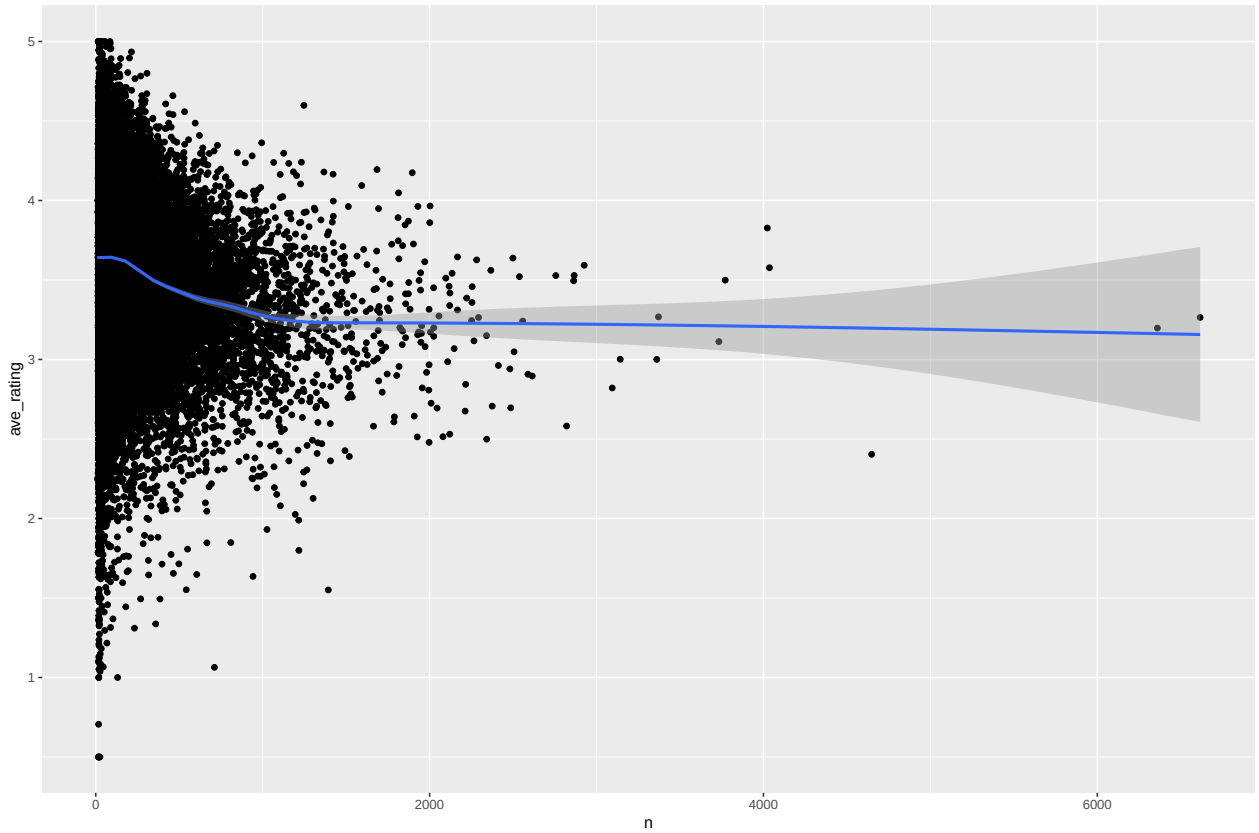
Let's confirm if this assumption holds on the entire data set to ensure it is accounted for in the later analysis. By looking at the distribution by movie id, we confirm that the smaller number of reviews there are, the larger the variance of average ratings. The chart also shows an increasing trend in average ranking for the movies that are frequently rated.

```
edx %>% group_by(movieId) %>% summarize(n=n(), ave_rating=mean(rating)) %>%
  arrange(desc(n)) %>% ggplot(aes(n, ave_rating))+geom_point()+geom_smooth()
```



Similar trend is observed when data is grouped by user id. Users that do not rate frequently, tend to have higher variance of the average rating.

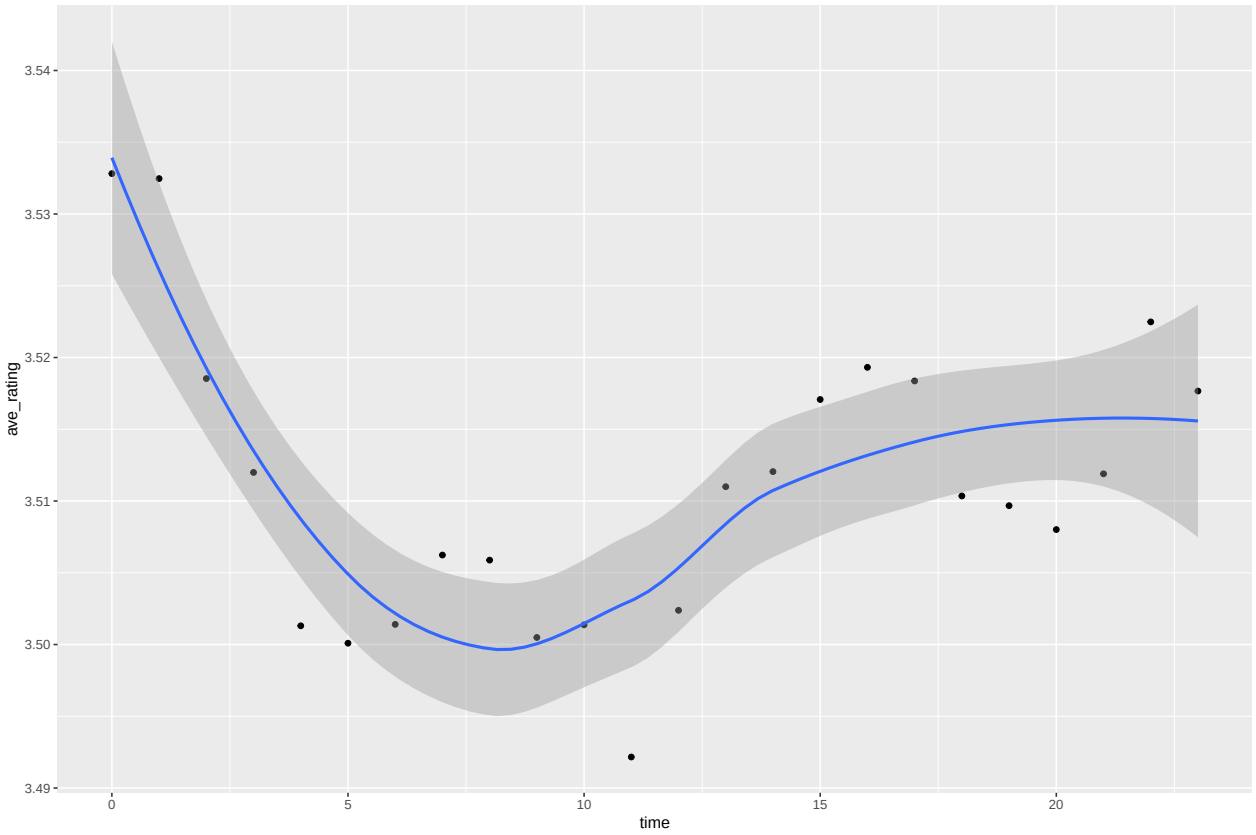
```
edx %>% group_by(userId) %>% summarize(n=n(), ave_rating=mean(rating)) %>%
  arrange(desc(n)) %>% ggplot(aes(n, ave_rating))+geom_point()+geom_smooth()
```



Based on the valuable insights gained from visualizing the data, it is clear that some regularization will be required to account for such variance.

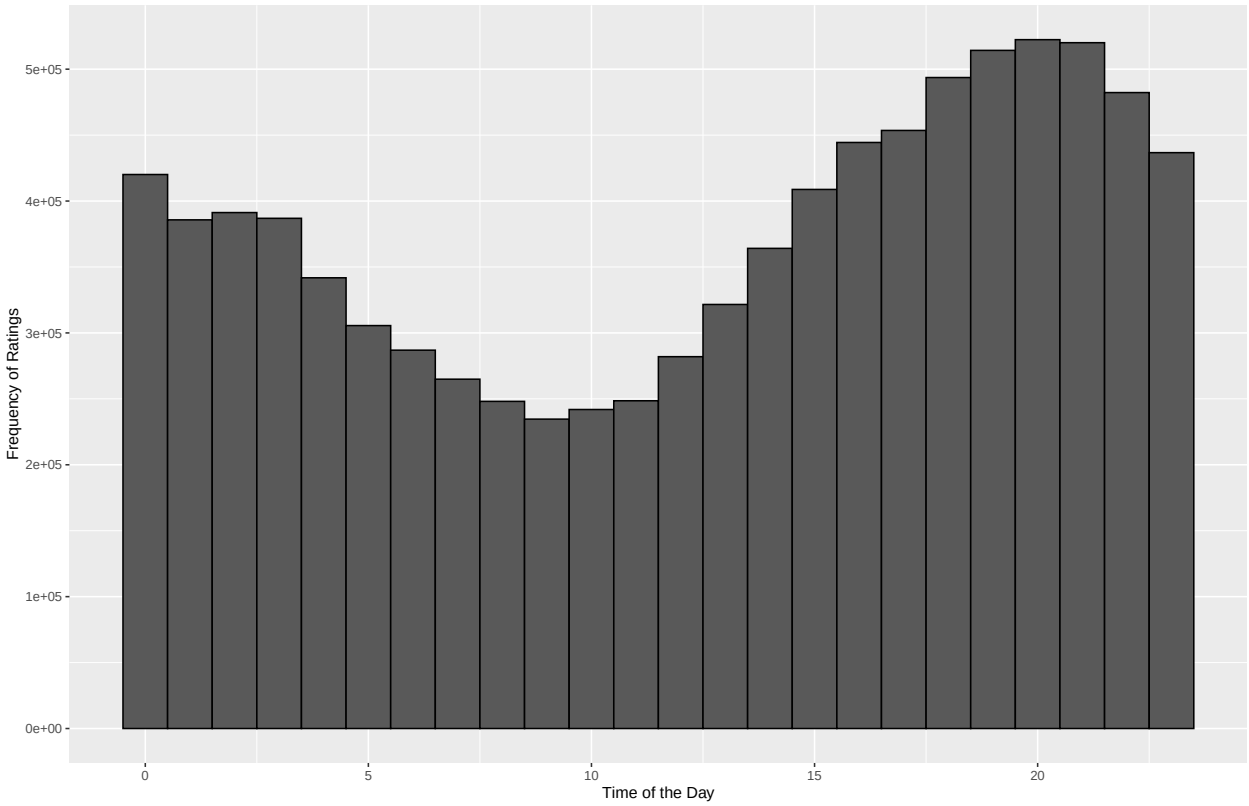
Additionally, since there is a timestamp for each rating, let's visualize how users rate movies during different times of the day. It looks like the movie ratings are higher in the late at night between 10 p.m. and 1 a.m.

```
edx %>% mutate(time=hour(as_datetime(timestamp))) %>% group_by(time) %>%
  summarize( ave_rating=mean(rating)) %>% ggplot(aes(time, ave_rating)) +
  geom_point()+geom_smooth()
```



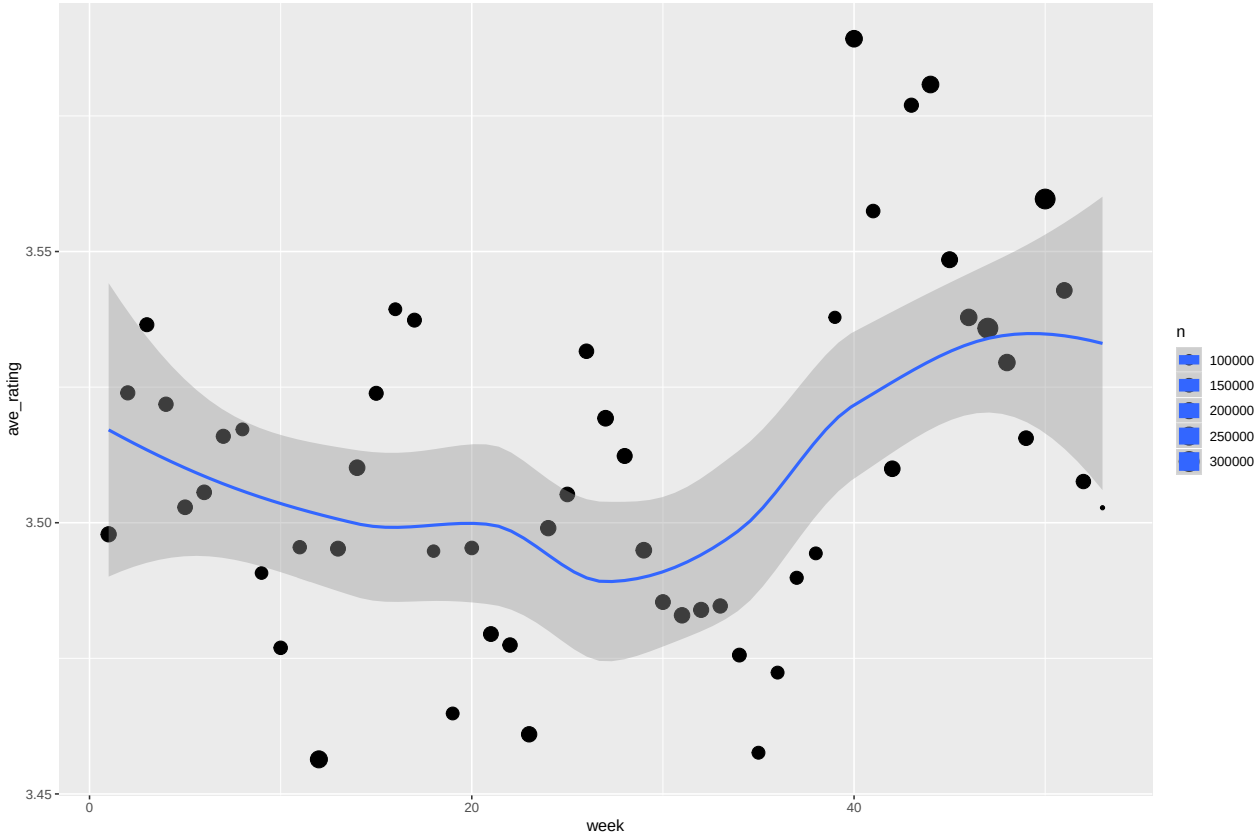
Users rate more often from 5 to 10 p.m. Less users rate after 10 p.m. but - as indicated above - at higher average rating.

```
edx %>% mutate(time=hour(as_datetime(timestamp))) %>% ggplot(aes(time)) +
  geom_histogram(bins=24, color = "black")+labs(title="", x="Time of the Day",
                                                y="Frequency of Ratings")
```



We have seen data variability as it relates to the time and the year of the rating. It is logical to test if the week of the year has any effect on the distribution of ratings. In the graph below, the trendline shows that there are more movies rated around the holidays in November and December. It also shows that the ratings are well above 3.51 - the overall average rating.

```
edx %>% mutate(week=week(as_datetime(timestamp))) %>% group_by(week) %>%
  summarize( ave_rating=mean(rating), n=n()) %>% ggplot(aes(week, ave_rating, size=n)) +
  geom_point()+geom_smooth()
```



So, so far we visualized how user, movie and time impacts our ratings. One feature we did not look at is movie genres which can be helpful when grouping similar movies together. Let's approach this by clustering data.

But first, let's pick a smaller set consisting of top 25 rated movies and filtering our users that don't frequently rate.

We will apply several approaches to understand how data is clustered: (1) Hierarchical; (2) K-Means; (3) Heatmaps

(1) Hierarchical clustering groups data in several clusters. These clusters consist of iconic movies such as Independence Day and Jurassic Park; three Star Wars movies; Pulp Fiction and Shawshank Redemption to name a few. By setting cutree parameter k to 5, we create five relatively uniform groups featured below. The movies within each cluster share similarities in their genres whether its action, sci-fi or drama.

```
# Clustering data
top <- edx %>%
  group_by(movieId) %>%
  summarize(n=n(), title = first(title)) %>% top_n(25, n) %>%
  pull(movieId)

x <- edx %>%
  filter(movieId %in% top) %>% group_by(userId) %>%
  filter(n() >= 25) %>%
  ungroup() %>%
  select(title, userId, rating) %>% spread(userId, rating)
head(x)
```

```
## # A tibble: 6 x 49
```

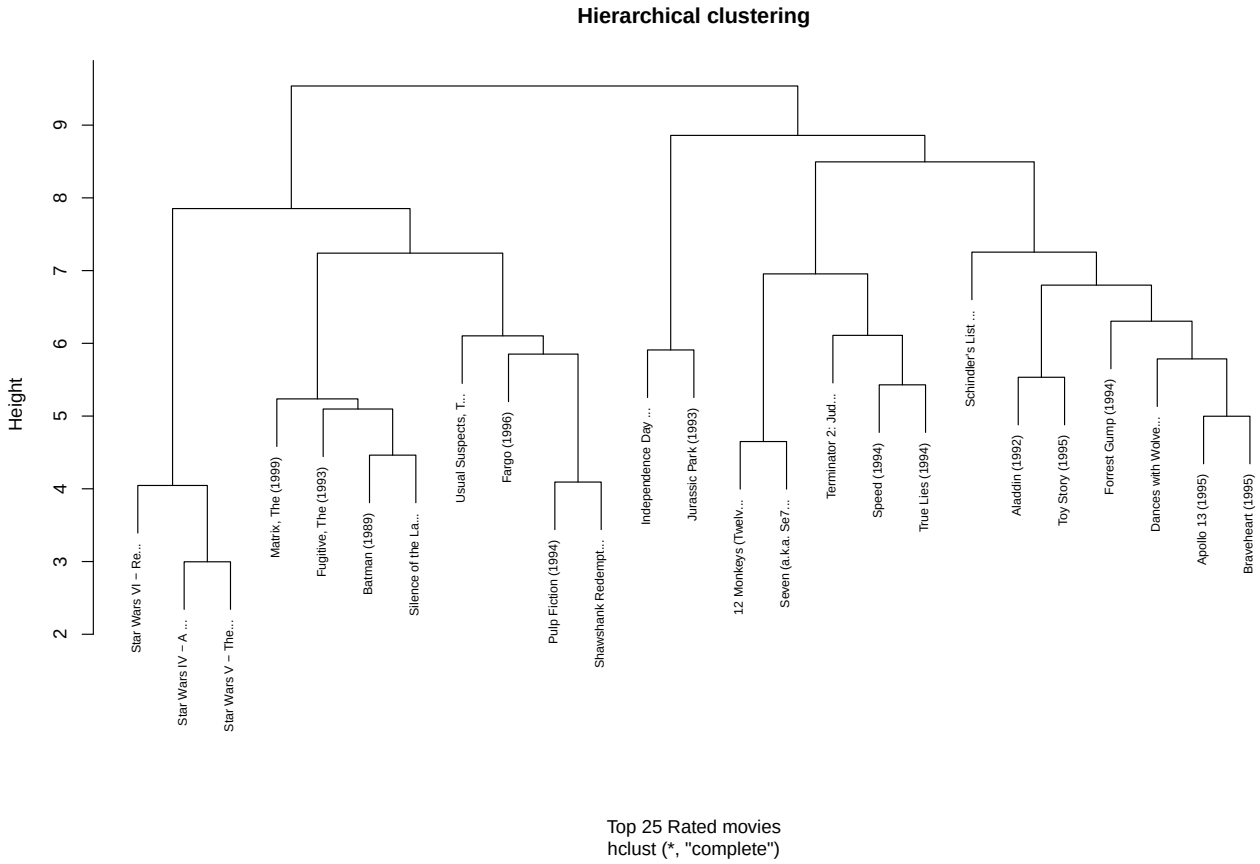
```
##   title `126` `4129` `5191` `7127` `7356` `7744` `7774` `19191` `19635`
##   <chr> <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>   <dbl>
## 1 12 M~   3.5    4.5      5    3.5    4.5    4.5    3.5       4       4
## 2 Alad~   3.5    3.5      4    4.5    4      4      3.5       4       3
## 3 Apol~   3.5    4        4    4      4      3      4         3       4
## 4 Batm~   4      4        4    4      3.5    4.5    5         4       4
## 5 Brav~   5      4        5    4      4.5    4      3         5       5
## 6 Danc~   4      3.5      5    4.5    4      4.5    3.5       5       4
## # ... with 39 more variables: `20410` <dbl>, `20528` <dbl>, `20597` <dbl>,
## #   `21089` <dbl>, `21941` <dbl>, `22047` <dbl>, `27637` <dbl>,
## #   `27663` <dbl>, `27751` <dbl>, `27945` <dbl>, `28788` <dbl>,
## #   `31013` <dbl>, `32689` <dbl>, `33086` <dbl>, `33983` <dbl>,
## #   `33987` <dbl>, `38409` <dbl>, `38616` <dbl>, `41956` <dbl>,
## #   `48670` <dbl>, `48725` <dbl>, `49077` <dbl>, `49447` <dbl>,
## #   `49759` <dbl>, `50056` <dbl>, `55661` <dbl>, `57123` <dbl>,
## #   `57359` <dbl>, `61468` <dbl>, `61548` <dbl>, `63637` <dbl>,
## #   `64240` <dbl>, `64411` <dbl>, `64775` <dbl>, `66123` <dbl>,
## #   `66311` <dbl>, `67976` <dbl>, `68395` <dbl>, `69385` <dbl>
```

```r
row_names <- str_remove(x$title, ": Episode") %>% str_trunc(20)
x <- x[,-1] %>% as.matrix()
head(x)
```

```
##      126 4129 5191 7127 7356 7744 7774 19191 19635 20410 20528 20597 21089
## [1,] 3.5  4.5    5  3.5  4.5  4.5  3.5     4     4     4   5.0   4.0     4
## [2,] 3.5  3.5    4  4.5  4.0  4.0  3.5     4     3     5   2.0   2.5     4
## [3,] 3.5  4.0    4  4.0  4.0  3.0  4.0     3     4     5   2.5   3.5     4
## [4,] 4.0  4.0    4  4.0  3.5  4.5  5.0     4     4     4   2.5   3.5     4
## [5,] 5.0  4.0    5  4.0  4.5  4.0  3.0     5     5     5   1.0   3.0     5
## [6,] 4.0  3.5    5  4.5  4.0  4.5  3.5     5     4     5   2.0   4.0     5
##      21941 22047 27637 27663 27751 27945 28788 31013 32689 33086 33983
## [1,]   3.0   3.5   4.5   4.5     4   4.0   4.0   3.0   4.0   4.0   3.5
## [2,]   3.0   3.5   4.0   3.0     3   3.0   4.0   5.0   2.0   3.5   3.5
## [3,]   4.5   2.0   3.0   4.0     4   1.0   4.0   3.5   3.5   4.0   3.0
## [4,]   3.5   4.0   4.5   4.5     4   3.0   4.0   3.0   3.5   4.0   3.5
## [5,]   3.5   2.5   4.0   4.0     5   1.0   4.5   3.0   4.0   4.0   4.0
## [6,]   5.0   3.5   4.0   5.0     4   2.5   4.0   3.0   3.0   2.5   3.0
##      33987 38409 38616 41956 48670 48725 49077 49447 49759 50056 55661
## [1,]   4.0     4   4.5   2.0   4.5   3.0     5   4.5   5.0   4.5   4.0
## [2,]   4.0     3   3.0   4.5   3.0   1.0     4   4.0   3.0   3.5   3.0
## [3,]   4.0     4   4.0   3.5   3.0   2.5     5   4.0   3.5   3.5   3.5
## [4,]   4.0     4   4.0   3.5   3.5   3.5     3   4.0   4.0   4.0   4.0
## [5,]   4.0     5   5.0   4.0   3.5   2.0     5   5.0   3.5   4.0   4.5
## [6,]   4.5     4   3.5   4.0   4.5   2.0     4   4.5   2.5   3.5   4.5
##      57123 57359 61468 61548 63637 64240 64411 64775 66123 66311 67976
## [1,]   3.5   3.5     5   3.0   2.0   3.5   3.0   3.5     4   3.0   4.5
## [2,]   2.0   3.0     4   4.5   4.0   3.0   4.0   4.0     5   3.0   3.0
## [3,]   3.0   2.5     3   4.5   3.0   3.5   4.0   2.5     4   2.5   4.5
## [4,]   3.0   4.0     3   4.5   3.5   4.0   4.0   5.0     4   1.0   3.5
## [5,]   3.5   4.0     4   5.0   4.0   4.5   4.5   4.0     4   2.0   5.0
## [6,]   3.0   3.0     4   4.0   3.0   4.0   5.0   4.0     5   3.5   5.0
##      68395 69385
## [1,]   3.0     4
## [2,]   4.0     5
## [3,]   3.5     4
```

```
## [4,]   4.0     4
## [5,]   5.0     5
## [6,]   3.5     4
```

```r
x <- sweep(x, 2, colMeans(x, na.rm = TRUE))
x <- sweep(x, 1, rowMeans(x, na.rm = TRUE))
rownames(x) <- row_names

# Calculating the distances
d <- dist(x)
# Hierarchical clustering
h <- hclust(d)
plot(h, cex = 0.65, main = "Hierarchical clustering", xlab = "Top 25 Rated movies")
```




```r
groups <- cutree(h, k = 5)
split(names(groups),groups)
```

```
## $`1`
## [1] "12 Monkeys (Twelv..." "Seven (a.k.a. Se7..." "Speed (1994)"
## [4] "Terminator 2: Jud..." "True Lies (1994)"
##
## $`2`
## [1] "Aladdin (1992)"       "Apollo 13 (1995)"     "Braveheart (1995)"
## [4] "Dances with Wolve..." "Forrest Gump (1994)"  "Schindler's List ..."
## [7] "Toy Story (1995)"
##
## $`3`
```

```
## [1] "Batman (1989)"        "Fargo (1996)"         "Fugitive, The (1993)"
## [4] "Matrix, The (1999)"   "Pulp Fiction (1994)"  "Shawshank Redempt..."
## [7] "Silence of the La..." "Usual Suspects, T..."
##
## $`4`
## [1] "Independence Day ..." "Jurassic Park (1993)"
##
## $`5`
## [1] "Star Wars IV - A ..." "Star Wars V - The..." "Star Wars VI - Re..."
```

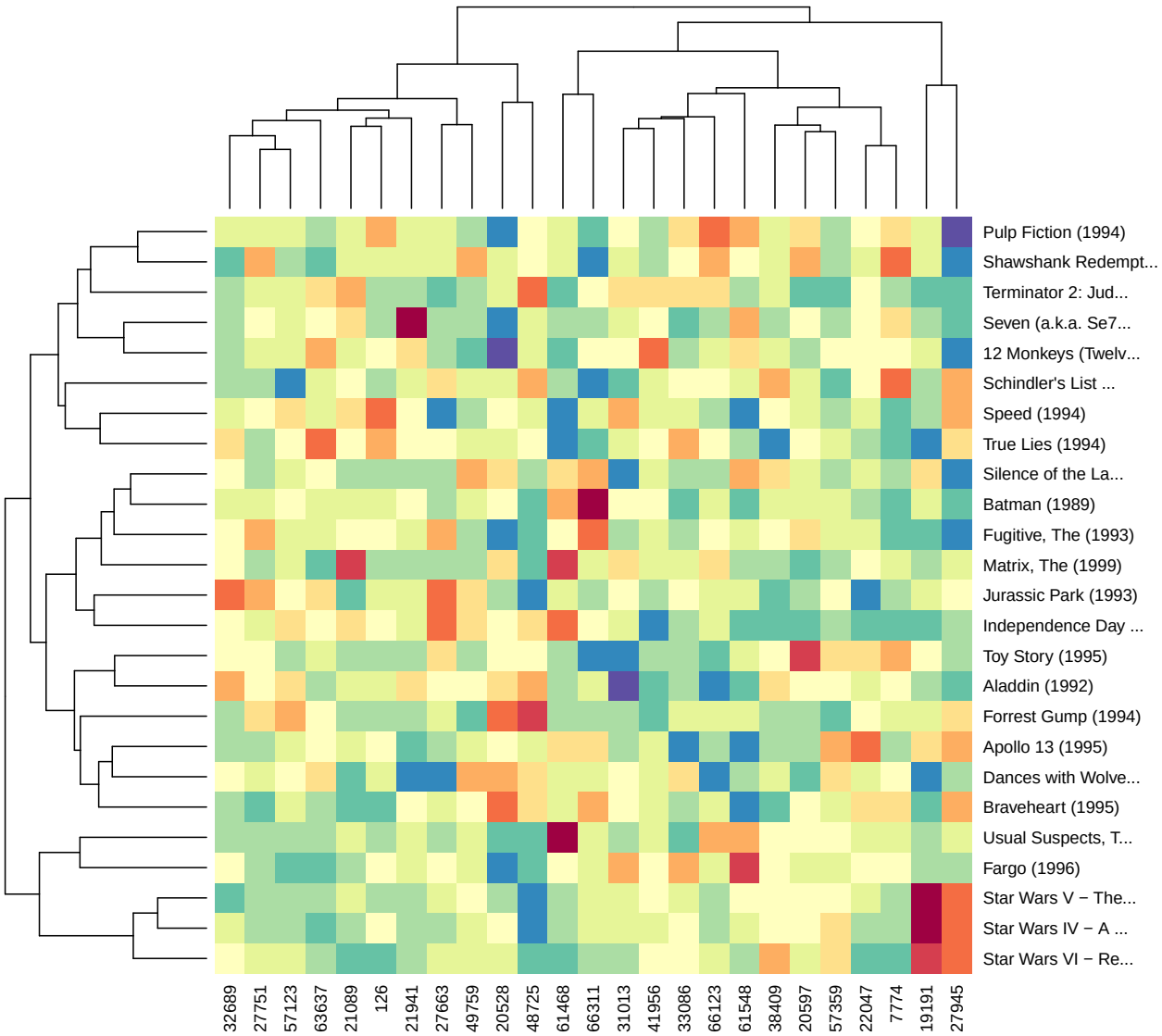
(2) When using iterative K-Means clustering, the first center is chosen at random and the final clusters are random. To stabilize the result, the entire function can be repeated several times and the results averaged. In this example, one random selection is picked for illustrative purposes. The algorithm is set to pick 5 centers and similar to Hierarchical five clusters are formed (set.seed(10)).

```
# Using k-means
set.seed(10, sample.kind="Rounding")
x_0 <- x
x_0[is.na(x_0)] <- 0
k <- kmeans(x_0, centers = 5)
groupsk <- k$cluster
split(names(groupsk),groupsk)
```

```
## $`1`
##  [1] "12 Monkeys (Twelv..." "Batman (1989)"        "Fargo (1996)"
##  [4] "Fugitive, The (1993)" "Matrix, The (1999)"   "Pulp Fiction (1994)"
##  [7] "Seven (a.k.a. Se7..." "Shawshank Redempt..." "Silence of the La..."
## [10] "Usual Suspects, T..."
##
## $`2`
## [1] "Apollo 13 (1995)"     "Braveheart (1995)"    "Dances with Wolve..."
## [4] "Forrest Gump (1994)"  "Speed (1994)"         "Terminator 2: Jud..."
## [7] "True Lies (1994)"
##
## $`3`
## [1] "Independence Day ..." "Jurassic Park (1993)"
##
## $`4`
## [1] "Star Wars IV - A ..." "Star Wars V - The..." "Star Wars VI - Re..."
##
## $`5`
## [1] "Aladdin (1992)"       "Schindler's List ..." "Toy Story (1995)"
```

(3) Heatmaps is another great way to depict clusters. To better visualize, we account for noise from users with low variance in their ratings and sort the list of users by the standard deviation of their ratings. This approach groups clusters with similar kind of movies.

```
# Heatmap sorted by users from the highest standard deviation to low
sds <- colSds(x, na.rm = TRUE)
o <- order(sds, decreasing = TRUE)[1:25]
heatmap(x[,o], col = RColorBrewer::brewer.pal(11, "Spectral"))
```



All-in-all, our data exploratory analysis revealed that there are various effects that need to be captured by our model. There are not only user or movie-specific. There are time and genre-specific effects that should be evaluated. In addition, there are similarities among movies or users that should be factored in the model.

Building a Model

So, let's define our model the following way:

Y = mu + bi +bu + bt + bm + by + bg +eu,i; where: mu - mean of observations; bi - movie-specific effect; bu - user-specific effect; bt, bm, by - time-specific effects related to time, month and year; ei,u - independent errors.

Per course instructions, we will further split the edx data into the training and test sets:

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

We apply semi_join to make sure we don't include users and movies in the test set that do not appear in the training set.

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

Let's define the loss function (RMSE) using the following function..

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

Model 0: Naive

In order to create a benchmark for RMSEs, let's compute a basic model RMSE that predicts the same rating across all movies and users. That will allow us to understand if we are moving in the right direction when adding other effects.

```
mu <- mean(train_set$rating)
RMSE_0 <- RMSE(test_set$rating, mu)
RMSE_0
```

```
## [1] 1.051374
```

```
rmse_results0 <- data_frame(Model = "Naive (just the average)", RMSE = RMSE_0,
                            Improvement = NA, OvImprovement = NA)
rmse_results0
```

```
## # A tibble: 1 x 4
##   Model                      RMSE Improvement OvImprovement
##   <chr>                     <dbl> <lgl>       <lgl>
## 1 Naive (just the average)  1.05 NA          NA
```

Model 1: Movie Specific

Now as we have the RMSE from our first model, let's estimate the movie-specific effect by calculating the mean and movie averages summarized by bi and predicting it using the test set. We are able to get an improvement from 1.051374 to 0.926325.

```
mu
```

```
## [1] 3.512345
```

```
movie_avgs <- train_set %>%group_by(movieId) %>% summarize(b_i = mean(rating - mu))
predicted_ratings <- test_set %>%left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>% pull(pred)
RMSE_1<-RMSE(test_set$rating, predicted_ratings)
RMSE_1
```

```
## [1] 0.9263246
```

```
rmse_results1 <- bind_rows(rmse_results0, data_frame(Model = "Movie effect", RMSE = RMSE_1,
                                                     Improvement = RMSE_0-RMSE_1,
                                                     OvImprovement = RMSE_0-RMSE_1))
rmse_results1
```

```
## # A tibble: 2 x 4
##   Model                      RMSE Improvement OvImprovement
##   <chr>                     <dbl>       <dbl>         <dbl>
## 1 Naive (just the average) 1.05       NA            NA
```

```
## 2 Movie effect             0.926       0.125         0.125
```

Model 2: Movie and User Specific

In the next step, we add the user-specific effect. The RMSE showed another improvement from 0.926325 to 0.847828.

```
user_avgs <- train_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) %>% pull(pred)
RMSE_2<-RMSE(test_set$rating, predicted_ratings)
RMSE_2
```

```
## [1] 0.8478277
```

```
RMSE_1-RMSE_2
```

```
## [1] 0.0784969
```

```
rmse_results2 <- bind_rows(rmse_results1,
                           data_frame(Model = "Movie and User effect",
                                      RMSE = RMSE_2, Improvement = RMSE_1-RMSE_2,
                                      OvImprovement = RMSE_0-RMSE_2))
rmse_results2
```

```
## # A tibble: 3 x 4
##   Model                      RMSE Improvement OvImprovement
##   <chr>                     <dbl>       <dbl>         <dbl>
## 1 Naive (just the average) 1.05      NA             NA
## 2 Movie effect             0.926      0.125          0.125
## 3 Movie and User effect    0.848      0.0785         0.204
```

Model 3: Movie, User, Genre Specific

Next, we add genre-specific effect and get a slightly improved RMSE of 0847299.

```
genres_avgs<-train_set %>% left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId')%>%group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))
predicted_ratings <- test_set %>%left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId')%>%
  left_join(genres_avgs, by='genres')%>%mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
RMSE_3<-RMSE(test_set$rating, predicted_ratings)
RMSE_3
```

```
## [1] 0.847299
```

```
RMSE_2-RMSE_3
```

```
## [1] 0.0005287625
```

```
rmse_results3 <- bind_rows(rmse_results2,
                           data_frame(Model = "Movie, User and Genre effect",
                                      RMSE = RMSE_3, Improvement = RMSE_2-RMSE_3,
                                      OvImprovement = RMSE_0-RMSE_3))
rmse_results3
```

```
## # A tibble: 4 x 4
##   Model                          RMSE Improvement OvImprovement
##   <chr>                         <dbl>       <dbl>         <dbl>
## 1 Naive (just the average)      1.05    NA               NA
## 2 Movie effect                  0.926    0.125            0.125
## 3 Movie and User effect         0.848    0.0785           0.204
## 4 Movie, User and Genre effect 0.847    0.000529         0.204
```

Model 4: Movie, User, Genre, Time Specific

Since genre-specific effect did not drastically improve the model's performance signaling that movie and user-specific effects explained a lot of ratings' variabilities, we will be adding time-effects one at a time testing how it improves the model.

We begin by adding time-specific effect to the model. The resulting RMSE did not improve.

```
time_avgs<-train_set %>% left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId')%>%left_join(genres_avgs, by='genres')%>%
  mutate(time=hour(as_datetime(timestamp)))%>%group_by(time) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u - b_g))
predicted_ratings <- test_set %>%left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId')%>%
  left_join(genres_avgs, by='genres') %>% mutate(time=hour(as_datetime(timestamp))) %>%
  left_join(time_avgs, by='time')%>% mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
  pull(pred)
RMSE_4<-RMSE(test_set$rating, predicted_ratings)
RMSE_4
```

```
## [1] 0.8473014
```

```
RMSE_3-RMSE_4
```

```
## [1] -2.414061e-06
```

```
rmse_results4 <- bind_rows(rmse_results3,
                           data_frame(Model = "Movie, User, Genre and Time effect",
                                      RMSE = RMSE_4, Improvement = RMSE_3-RMSE_4,
                                      OvImprovement = RMSE_0-RMSE_4))
rmse_results4
```

```
## # A tibble: 5 x 4
##   Model                              RMSE Improvement OvImprovement
##   <chr>                             <dbl>       <dbl>         <dbl>
## 1 Naive (just the average)          1.05  NA               NA
## 2 Movie effect                      0.926  0.125            0.125
## 3 Movie and User effect             0.848  0.0785           0.204
## 4 Movie, User and Genre effect      0.847  0.000529         0.204
## 5 Movie, User, Genre and Time effect 0.847 -0.00000241       0.204
```

Model 5: Movie, User, Genre, Week Specific

Next, we add week-specific factor. Likewise, the model did not yield a better RMSE.

```
week_avgs<-train_set %>% left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId')%>%left_join(genres_avgs, by='genres')%>%
  mutate(week=week(as_datetime(timestamp)))%>%group_by(week) %>%
  summarize(b_w = mean(rating - mu - b_i - b_u - b_g))
predicted_ratings <- test_set %>%left_join(movie_avgs, by='movieId') %>%
```

```
  left_join(user_avgs, by='userId')%>%
  left_join(genres_avgs, by='genres') %>% mutate(week=week(as_datetime(timestamp)))%>%
  left_join(week_avgs, by='week')%>% mutate(pred = mu + b_i + b_u + b_g + b_w) %>% pull(pred)
RMSE_5<-RMSE(test_set$rating, predicted_ratings)
RMSE_5
```

```
## [1] 0.8473409
```

```
RMSE_3-RMSE_5
```

```
## [1] -4.197036e-05
```

```
rmse_results5 <- bind_rows(rmse_results4,
                           data_frame(Model = "Movie, User, Genre and Week effect",
                                      RMSE = RMSE_5, Improvement = RMSE_3-RMSE_5,
                                      OvImprovement = RMSE_0-RMSE_5))
rmse_results5
```

```
## # A tibble: 6 x 4
##   Model                                RMSE Improvement OvImprovement
##   <chr>                               <dbl>       <dbl>         <dbl>
## 1 Naive (just the average)            1.05  NA                 NA
## 2 Movie effect                        0.926  0.125              0.125
## 3 Movie and User effect               0.848  0.0785             0.204
## 4 Movie, User and Genre effect        0.847  0.000529           0.204
## 5 Movie, User, Genre and Time effect 0.847 -0.00000241          0.204
## 6 Movie, User, Genre and Week effect 0.847 -0.0000420           0.204
```

Model 6: Movie, User, Genre, Year Specific

Finally, we add year-specific factor to the movie, user, genre-specific model. There is no improvement in its RMSE.

```
year_avgs<-train_set %>% left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId')%>%left_join(genres_avgs, by='genres')%>%
  mutate(year=year(as_datetime(timestamp)))%>%group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u - b_g))
predicted_ratings <- test_set %>%left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId')%>%
  left_join(genres_avgs, by='genres') %>% mutate(year=year(as_datetime(timestamp)))%>%
  left_join(year_avgs, by='year')%>% mutate(pred = mu + b_i + b_u + b_g + b_y) %>%
  pull(pred)
RMSE_6<-RMSE(test_set$rating, predicted_ratings)
RMSE_6
```

```
## [1] 0.8473163
```

```
RMSE_3-RMSE_6
```

```
## [1] -1.732097e-05
```

```
rmse_results6 <- bind_rows(rmse_results5,
                           data_frame(Model = "Movie, User, Genre and Year effect",
                                      RMSE = RMSE_6, Improvement = RMSE_3-RMSE_6,
                                      OvImprovement = RMSE_0-RMSE_6))
rmse_results6
```

```
## # A tibble: 7 x 4
```

```
##   Model                                RMSE Improvement OvImprovement
##   <chr>                               <dbl>       <dbl>         <dbl>
## 1 Naive (just the average)            1.05  NA                 NA
## 2 Movie effect                        0.926  0.125              0.125
## 3 Movie and User effect               0.848  0.0785             0.204
## 4 Movie, User and Genre effect        0.847  0.000529           0.204
## 5 Movie, User, Genre and Time effect 0.847 -0.00000241         0.204
## 6 Movie, User, Genre and Week effect 0.847 -0.0000420          0.204
## 7 Movie, User, Genre and Year effect 0.847 -0.0000173          0.204
```

While model 4 improved RMSE slightly, models 5 - 7 which consider time, week, and year-specific effects did not improve RMSEs and therefore will be excluded from further analysis.

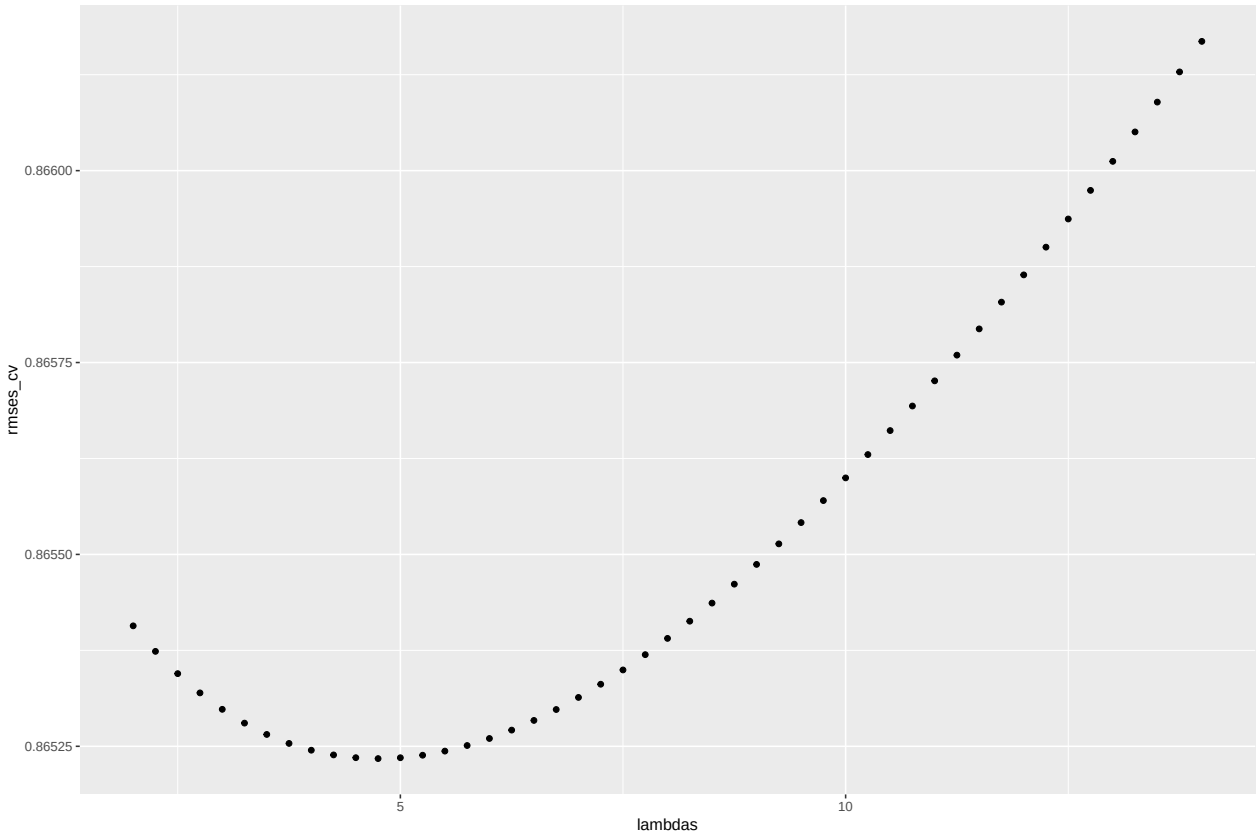
Cross Validation

To further improve RMSE of model 4, we apply regularization technique using 5-fold cross validation and refitting the model with the optimized parameters.

NOTE: the following code will take time to run, instead you can skip and directly assign the best lambda of 4.75 (see below).

```
set.seed(1, sample.kind = "Rounding")
# Make 5 splits
cv_5 <- createFolds(train_set$rating, k=5, returnTrain =TRUE)

# Define a matrix to store the results of cross validation
# With 5 rows for each fold (k) and 49 columns for each lambda (length(seq(2,14,0.25)))
RMSEs <- matrix(nrow=5,ncol=49)
lambdas <- seq(2, 14, 0.25)

# Split the train set
for(k in 1:5) {
  train_set_cv <- train_set[cv_5[[k]],]
  test_set_cv <- train_set[-cv_5[[k]],]

  # Making sure userId and movieId in test set are also in the train set
  test_joined <- test_set_cv %>%
    semi_join(train_set_cv, by = "movieId") %>%
    semi_join(train_set_cv, by = "userId")

  # Add rows removed back into the set
  removed <- anti_join(test_set_cv, test_joined)
  train_joined <- rbind(train_set_cv, removed)

  mu <- mean(train_joined$rating)

  RMSEs[k,] <- sapply(lambdas, function(l){
    b_i <- train_joined %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu)/(n()+l))
    b_u <- train_joined %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    b_g <- train_joined %>%
```

```
      left_join(b_i, by='movieId') %>% left_join(b_u, by='userId') %>%
      group_by(genres) %>% summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
    predicted_ratings <-
      test_joined %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>% left_join(b_g, by="genres") %>%
      mutate(pred = mu + b_i + b_u + b_g) %>%
      pull(pred)
    return(RMSE(test_joined$rating, predicted_ratings))
  })
}
RMSEs
rmses_cv <- colMeans(RMSEs)
qplot(lambdas,rmses_cv)
```




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

Model 7: Movie, User, Genre, Year Specific Regularized

The cross-validation revealed that lambda of 4.75 yields the best RMSE for that model. So, let's apply it.

```
#lambda<-4.75
mu <- mean(train_set$rating)
b_i_reg <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
b_u_reg <- train_set %>%
```

```
  left_join(b_i_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
b_g_reg <- train_set %>%
  left_join(b_i_reg, by="movieId") %>% left_join(b_u_reg, by="userId") %>%
  group_by(genres) %>% summarize(b_g=sum(rating - b_i - b_u - mu)/(n()+lambda))
predicted_ratings7 <- test_set %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  left_join(b_g_reg, by="genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
RMSE_7 <- RMSE(test_set$rating, predicted_ratings7)
RMSE_7
```

```
## [1] 0.8467079
```

```
RMSE_3-RMSE_7
```

```
## [1] 0.0005910691
```

```
rmse_results7 <- bind_rows(rmse_results6,
                           data_frame(Model = "Movie, User, and Genre effect reg",
                                      RMSE = RMSE_7, Improvement = RMSE_3-RMSE_7,
                                      OvImprovement = RMSE_0-RMSE_7))
rmse_results7
```

```
## # A tibble: 8 x 4
##   Model                                RMSE Improvement OvImprovement
##   <chr>                               <dbl>       <dbl>         <dbl>
## 1 Naive (just the average)            1.05  NA                 NA
## 2 Movie effect                        0.926  0.125              0.125
## 3 Movie and User effect               0.848  0.0785             0.204
## 4 Movie, User and Genre effect        0.847  0.000529           0.204
## 5 Movie, User, Genre and Time effect 0.847 -0.00000241         0.204
## 6 Movie, User, Genre and Week effect 0.847 -0.0000420          0.204
## 7 Movie, User, Genre and Year effect 0.847 -0.0000173          0.204
## 8 Movie, User, and Genre effect reg  0.847  0.000591           0.205
```

The regularized model revealed an improved RMSE of 0.846708.

Test Validation Set - Movie, User, Genre, Year Specific Regularized

Let's apply this final model to the validation set and see how it would behave. We get the RMSE of 0.864461. It is a bit highter than on the test set.

```
predicted_ratings_fin <- validation %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  left_join(b_g_reg, by="genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
RMSE_8<-RMSE(validation$rating, predicted_ratings_fin)

# table with results
final_assessment8 <-
```

```
  bind_rows(rmse_results7,
            data_frame(Model = "Test Validation Set: Movie, User, and Genre effect reg",
                       RMSE = RMSE_8, Improvement = NA, OvImprovement = NA))
final_assessment8
```

```
## # A tibble: 9 x 4
##   Model                                      RMSE Improvement OvImprovement
##   <chr>                                     <dbl>       <dbl>         <dbl>
## 1 Naive (just the average)                  1.05  NA                  NA
## 2 Movie effect                              0.926  0.125               0.125
## 3 Movie and User effect                     0.848  0.0785              0.204
## 4 Movie, User and Genre effect              0.847  0.000529            0.204
## 5 Movie, User, Genre and Time effect        0.847 -0.00000241          0.204
## 6 Movie, User, Genre and Week effect        0.847 -0.0000420           0.204
## 7 Movie, User, Genre and Year effect        0.847 -0.0000173           0.204
## 8 Movie, User, and Genre effect reg         0.847  0.000591            0.205
## 9 Test Validation Set: Movie, User, and Ge~ 0.864 NA                  NA
```

Model 8: User, Movie, Genre Specific Regularized and Matrix Factorization

Let's see if we can do even better with the matrix factorization approach. Matrix factorization was used by the winners of the Netflix challenge - a similar problem we are solving here. We will apply factorization to the remaining residuals after movie, user and genre effects are subtracted. The idea is to approximate the whole rating matrix by the product of two matrices of lower dimensions. In this case, our model would look like this:

Y = mu + bi + bu + bg + ru,i + eu,i; where: mu - mean of observations; bi - movie-specific effect; bu - user-specific effect; ru,i=pu*qi - matrix contains the residuals for users u = 1, . . . , for movies i = 1, ...; ei,u - independent errors.

We apply the Recosystem package - an R wrapper of the LIBMF library which is an open source tool for approximating an incomplete matrix using the product of two matrices. More information can be found by reading help documentation. The algorithm used by this package is described in the following paper: http://www.csie.ntu.edu.tw/~cjlin/papers/libmf/libmf_journal.pdf.

We begin by calculating the residuals.

```
# Calculatings the residuals by substracting regularized model predictors
train_set_res <- train_set %>%
  left_join(b_i_reg, by = "movieId") %>%
  left_join(b_u_reg, by = "userId") %>%
  left_join(b_g_reg, by="genres")%>%
  mutate(residual = rating - mu - b_i - b_u - b_g) %>%
  select(userId, movieId, residual)
head(train_set_res)
```

```
##   userId movieId  residual
## 1      1     122 0.8184469
## 2      1     185 0.5583312
## 3      1     292 0.2879155
## 4      1     316 0.3349608
## 5      1     329 0.3690853
## 6      1     355 1.2144777
```

We create train and test set matrices.

```
# Creating two matrices that include train set and test set
train_set_mf <- as.matrix(train_set_res)
test_set_mf <- test_set %>%
  select(userId, movieId, rating)
test_set_mf <- as.matrix(test_set_mf)
```

Per recosystem's package instructions, an object of class DataSource specifies the source of a data set, which can be created by calling it from a hard disk or an R object. In this example, we will write it to a hard disc by using the write.table command and call it using data_file() command.

```
# Writing to the hard disc
write.table(train_set_mf , file = "trainset.txt" , sep = " " ,
            row.names = FALSE, col.names = FALSE)
write.table(test_set_mf, file = "testset.txt" , sep = " " ,
            row.names = FALSE, col.names = FALSE)
# We use data_file() to specify a data set from a file on the hard disk.
set.seed(1, sample.kind = "Rounding")
train_set_mf <- data_file("trainset.txt")
test_set_mf <- data_file("testset.txt")
```

Next, we will follow these steps outlined in package documentation:

(1) Create a model object (a Reference Class object in R) by calling Reco().
(2) Call the $tune() method to select best tuning parameters along a set of candidate values.
(3) 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().
(4) Use the $predict() method to compute predicted values.

Note: This code may take about 20 minutes to execute.

```
# create a model object (step 1)
r <-Reco()

# per recosystem's help documentation (step 2), tuning training set with parameters
opts <- r$tune(train_set_mf, 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] 0.01
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
```

```
## [1] 0.1
##
## $min$loss_fun
## [1] 0.7939907
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.8078914
## 2   20        0     0.01        0     0.01   0.1 0.8117141
## 3   30        0     0.01        0     0.01   0.1 0.8207244
## 4   10        0     0.10        0     0.01   0.1 0.8056768
## 5   20        0     0.10        0     0.01   0.1 0.8016042
## 6   30        0     0.10        0     0.01   0.1 0.8023594
## 7   10        0     0.01        0     0.10   0.1 0.8031174
## 8   20        0     0.01        0     0.10   0.1 0.7957276
## 9   30        0     0.01        0     0.10   0.1 0.7939907
## 10  10        0     0.10        0     0.10   0.1 0.8252626
## 11  20        0     0.10        0     0.10   0.1 0.8244779
## 12  30        0     0.10        0     0.10   0.1 0.8231388
## 13  10        0     0.01        0     0.01   0.2 0.8099991
## 14  20        0     0.01        0     0.01   0.2 0.8239275
## 15  30        0     0.01        0     0.01   0.2 0.8382379
## 16  10        0     0.10        0     0.01   0.2 0.8061519
## 17  20        0     0.10        0     0.01   0.2 0.8063656
## 18  30        0     0.10        0     0.01   0.2 0.8085508
## 19  10        0     0.01        0     0.10   0.2 0.8038811
## 20  20        0     0.01        0     0.10   0.2 0.7984503
## 21  30        0     0.01        0     0.10   0.2 0.7989964
## 22  10        0     0.10        0     0.10   0.2 0.8224580
## 23  20        0     0.10        0     0.10   0.2 0.8230946
## 24  30        0     0.10        0     0.10   0.2 0.8216593
```

```
# training the recommender model (step 3)
r$train(train_set_mf, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.8604   6.9595e+06
##    1       0.8359   6.4545e+06
##    2       0.8166   6.2582e+06
##    3       0.7987   6.0857e+06
##    4       0.7838   5.9417e+06
##    5       0.7720   5.8347e+06
##    6       0.7623   5.7506e+06
##    7       0.7540   5.6803e+06
##    8       0.7468   5.6220e+06
##    9       0.7404   5.5716e+06
##   10       0.7347   5.5278e+06
##   11       0.7297   5.4897e+06
##   12       0.7251   5.4550e+06
##   13       0.7211   5.4266e+06
##   14       0.7175   5.3989e+06
##   15       0.7142   5.3755e+06
##   16       0.7112   5.3542e+06
##   17       0.7085   5.3356e+06
```

```
##   18       0.7060   5.3177e+06
##   19       0.7037   5.3019e+06
```

```
# writing prediction to file (step 4)
pred_file <- tempfile()
r$predict(test_set_mf, out_file(pred_file))
```

```
## prediction output generated at /var/folders/4s/p1qzsym95197zvp_c31ms_s40000gp/T//Rtmp7EeWiG/file16e9
```

```
predicted_res_mf <- scan(pred_file)
```

Now, let's calculate RMSE.

```
# Adding residual predictor ratings to previously predicted ratings
predicted_ratings_mf <- predicted_ratings7 + predicted_res_mf

# Calculating RMSE
RMSE_MF <- RMSE(test_set$rating, predicted_ratings_mf)
RMSE_MF
```

```
## [1] 0.7720752
```

```
rmse_results9 <-
  bind_rows(final_assessment8,data_frame(
    Model="Movie, User, and Genre effect regularized and Matrix Factorization",
                                    RMSE = RMSE_MF, Improvement = RMSE_7-RMSE_MF,
    OvImprovement = RMSE_0-RMSE_MF))
rmse_results9
```

```
## # A tibble: 10 x 4
##    Model                                  RMSE Improvement OvImprovement
##    <chr>                                 <dbl>       <dbl>         <dbl>
##  1 Naive (just the average)              1.05  NA                  NA
##  2 Movie effect                          0.926  0.125               0.125
##  3 Movie and User effect                 0.848  0.0785              0.204
##  4 Movie, User and Genre effect          0.847  0.000529            0.204
##  5 Movie, User, Genre and Time effect    0.847 -0.00000241          0.204
##  6 Movie, User, Genre and Week effect    0.847 -0.0000420           0.204
##  7 Movie, User, Genre and Year effect    0.847 -0.0000173           0.204
##  8 Movie, User, and Genre effect reg     0.847  0.000591            0.205
##  9 Test Validation Set: Movie, User, and G~ 0.864 NA                NA
## 10 Movie, User, and Genre effect regulariz~ 0.772  0.0746            0.279
```

The model yields the lowest result of 0.772075. That is 0.28 points improvement from the Naive model.

Final Validation Set - Movie, User, Genre, Year Specific Regularized plus Matrix Factorization

Finally, we evaluate the model on the Validation set and get the RMSE of 0.772075.

```
# Repeating steps 1 through 4 above
pred_file <- tempfile()
final_validation<-validation %>% select(userId, movieId, rating) %>% as.matrix()
write.table(final_validation, file = "finvalset.txt" , sep = " " , row.names = FALSE,
            col.names = FALSE)
final_validation <- data_file("finvalset.txt")
r$predict(final_validation, out_file(pred_file))
```

```
## prediction output generated at /var/folders/4s/p1qzsym95197zvp_c31ms_s40000gp/T//Rtmp7EeWiG/file16e9
predicted_res_mf <- scan(pred_file)

# Adding predictions
final_predicted_ratings_mf <- predicted_ratings_fin + predicted_res_mf

# Calculating RMSE using VALIDATION DATA
RMSE_MF_FIN<- RMSE(test_set$rating, predicted_ratings_mf)
RMSE_MF_FIN
```

```
## [1] 0.7720752
```

```
final_assessment_fm <- bind_rows(
  rmse_results9, data_frame(
    Model="Final Validation Set: Movie, User, and Genre effect regularized and MF",
                                    RMSE = RMSE_MF_FIN, Improvement = RMSE_8-RMSE_MF_FIN,
    OvImprovement = RMSE_8-RMSE_MF_FIN))
final_assessment_fm
```

```
## # A tibble: 11 x 4
##    Model                                  RMSE Improvement OvImprovement
##    <chr>                                 <dbl>       <dbl>         <dbl>
##  1 Naive (just the average)              1.05  NA                NA
##  2 Movie effect                          0.926  0.125             0.125
##  3 Movie and User effect                 0.848  0.0785            0.204
##  4 Movie, User and Genre effect          0.847  0.000529          0.204
##  5 Movie, User, Genre and Time effect    0.847 -0.00000241        0.204
##  6 Movie, User, Genre and Week effect    0.847 -0.0000420         0.204
##  7 Movie, User, Genre and Year effect    0.847 -0.0000173         0.204
##  8 Movie, User, and Genre effect reg     0.847  0.000591          0.205
##  9 Test Validation Set: Movie, User, and G~ 0.864 NA               NA
## 10 Movie, User, and Genre effect regulariz~ 0.772  0.0746            0.279
## 11 Final Validation Set: Movie, User, and ~ 0.772  0.0924            0.0924
```

Conclusion

During the course of this study, we evaluated several models that account for user, movie, genre, and time-specific affects. We optimized model's parameters using cross-validation and regularization techniques. In addition, we used matrix factorization to model the remaining residuals. The results show that the best model is the factorized model that accounts for user, movie and genre-specific effects. This model features RMSE of 0.772075. The limitations of this model are constraints associated with the local system capacity and time. The second best model that doesn't have these constraints is the regularized model. With RMSE of 0.864461, it accounts for user, movie, and genre-specific effects. The future work would include straight forward matrix factorization, without pre-processing of the data for movie, user and genre-specific effects, and comparing the performance of that model based on its RMSE result.