Report Summary

Name	Contribution
Ng Bao Yuan : 6457721	100
Manuel Kevin : 6346935	100
Tong Ming Xuan : 6355481	100
Khor Wen Siang Bryan : 6353885	100
Jin Zixiang : 6739271	100
Chhajlani Chirag : 6354026	100

VARIABLE DESCRIPTIONS :

1. 'movie.data' : Consists of 3 columns [movieId, title, genres]
2. 'ratings.data' : Consists of 4 columns [userId, movieId, rating, timestamp]
3. 'genre_preprocess' : Consists of the genres column which belongs to 'movie_data' data frame.
4. 'genre_preprocess2' : Consists of 10 columns (a random number) and the rows represent the movieId. The column names have been split into different columns by using '|' as a delimiter.
5. 'genreName_list' : Consists of the list of 18 genres.
6. 'genre_matrix1' : forms a matrix of 27279 rows and 18 columns. The first row of the matrix is the 18 genre names.
7. ratings_matrix : Consists of 8277 columns which represent the movieId and 702 rows or the userId.
8. avg_ratings : It consists of 439 rows (UserId) and displays the mean rating provided by each user to the movies they have watched.
9. views_in_movie : The row represents the movie id and the column represents the number of times that movie has been watched. We have also assumed that the number of views equal to the number of ratings.
10. table_views_in_movie : The row represents the movie id and there are three columns they are : views, movie id and the title.
11. movies_user1_recommen : It represents the 10 movie ids recommended to the user.
12. movies_user1_recommen_new : It represents the 10 movie titles recommended to the user.
13. recommendation_matrix : It consists of 10 rows (which represent the 10 movies) and 96 (random) columns (which represent the 96 random users). So for every column (every user), there are 10 movie ids.

CODE EXPLANATIONS :

A.

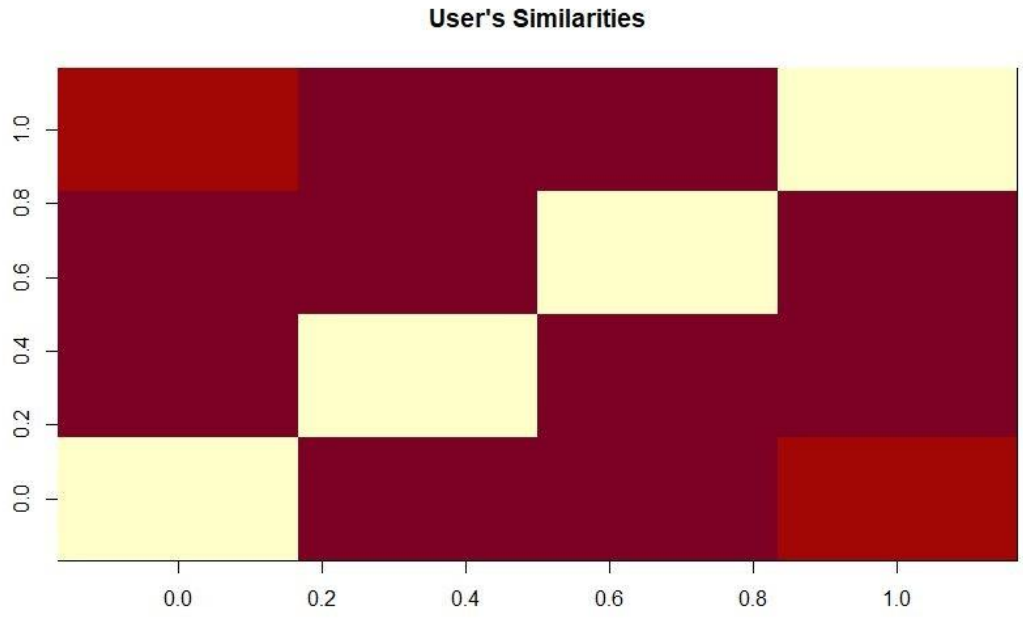
```
> for (row in 1:nrow(genre_preprocess2)) {
+   for (col in 1:ncol(genre_preprocess2)) {
+     new_col = which(genre_matrix1[1,] == genre_preprocess2[row,col])
+     genre_matrix1[row+1,new_col] <- 1
+   }
+ }
```

> In the above code, we used a nested for loop wherein the first for loop iterates over the rows of the 'movie_genre2' and the second for loop iterates over the columns of the 'movie_genre2'. Next, we are finding which column number in the first row of genre_mat1 matches a particular movie's column (or genre). Then as soon as we find a match, we replace that particular movie's genres to 1.

B

```
> #Exploring similar data of user using cosine method.
> similarity_mat <- similarity(ratings_matrix[1:4, ],
+                              method = "cosine",
+                              which = "users")
> as.matrix(similarity_mat)
          1         2         3         4
1 0.0000000 0.9766944 0.9926923 0.9058216
2 0.9766944 0.0000000 0.9824939 1.0000000
3 0.9926923 0.9824939 0.0000000 0.9394511
4 0.9058216 1.0000000 0.9394511 0.0000000
```

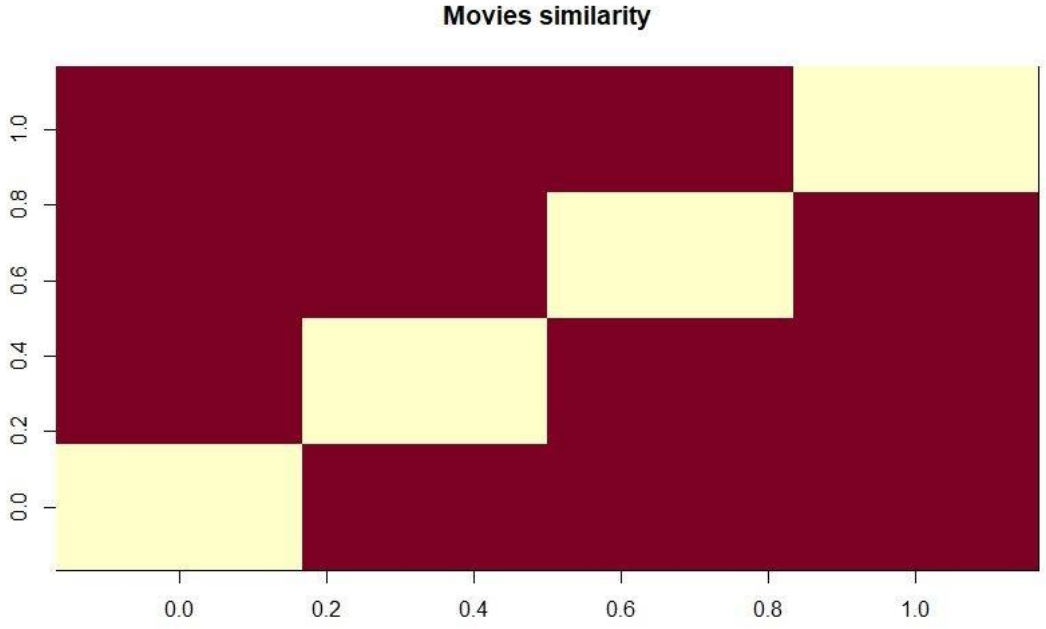
> In the above code, we are using the cosine similarity method to find the similarity between a set of 4 users. On analyzing the output, we can see that for the similarity between the same users has been displayed as 0.0000000. We are assuming the reason behind it, that the moment the cosine similarity comes across users with same id, they skip analyzing it.


User's Similarities
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

C.

```
> #Exploring similar data of movie using cosine method
> movie_similarity <- similarity(ratings_matrix[, 1:4], method =
+                                  "cosine", which = "items")
> as.matrix(movie_similarity)
          1         2         3         4
1 0.0000000 0.9608638 0.9532031 0.9864460
2 0.9608638 0.0000000 0.9279324 0.9660782
3 0.9532031 0.9279324 0.0000000 0.9541739
4 0.9864460 0.9660782 0.9541739 0.0000000
```

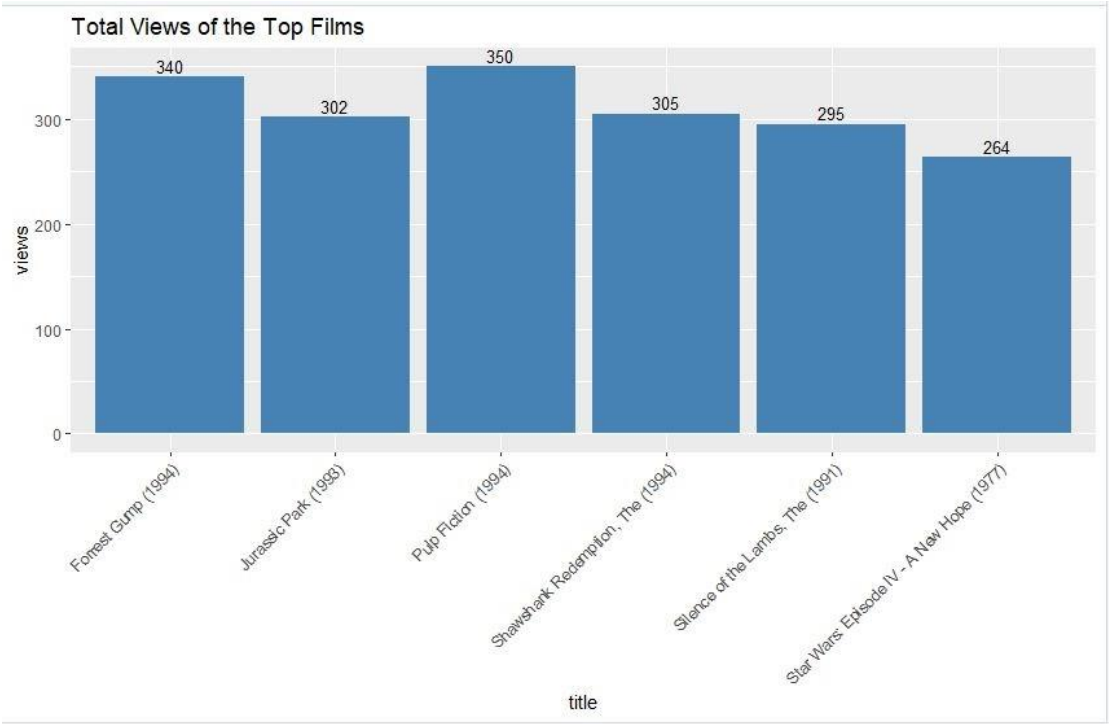
> In the above code, we are using the cosine similarity method to find the similarity between a set of 4 movies. On analyzing the output, we can see that for the similarity between the same movies has been displayed as 0.0000000. We are assuming the reason behind the same to be, that the moment the cosine similarity comes across movies with same id, they skip analyzing it.



D.

```
> ggplot(table_views_in_movie[1:6, ], aes(x = title, y = views)) +
+   geom_bar(stat="identity", fill = 'steelblue') +
+   geom_text(aes(label=views), vjust=-0.3, size=3.5) +
+   theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
+   ggtitle("Total Views of the Top Films")
>
```

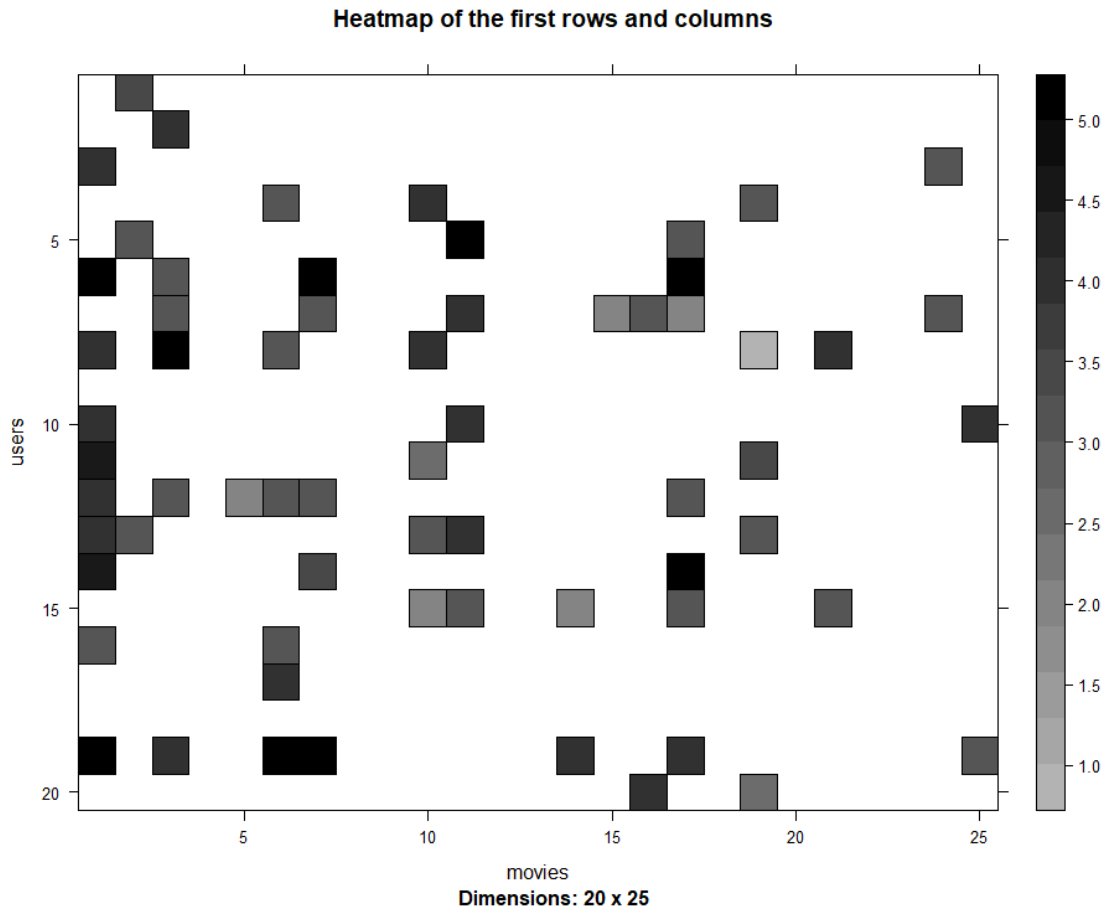
> The above bar plot displays the title of the Top 6 movie names on the X axis and the number of views on the Y axis.



E.

```
> # Overview (Heatmap of movie ratings 1:20, 1:25)
> image(ratings_matrix[1:20, 1:25], axes = FALSE, main = "Heatmap of the first rows and columns",
+       xlab = 'movie', ylab = 'user')
```

> As our initial heatmap shows our first rows and columns of our dataset, information from the heatmap is not helpful and thus, we perform data preparation to identify useful information.



F.

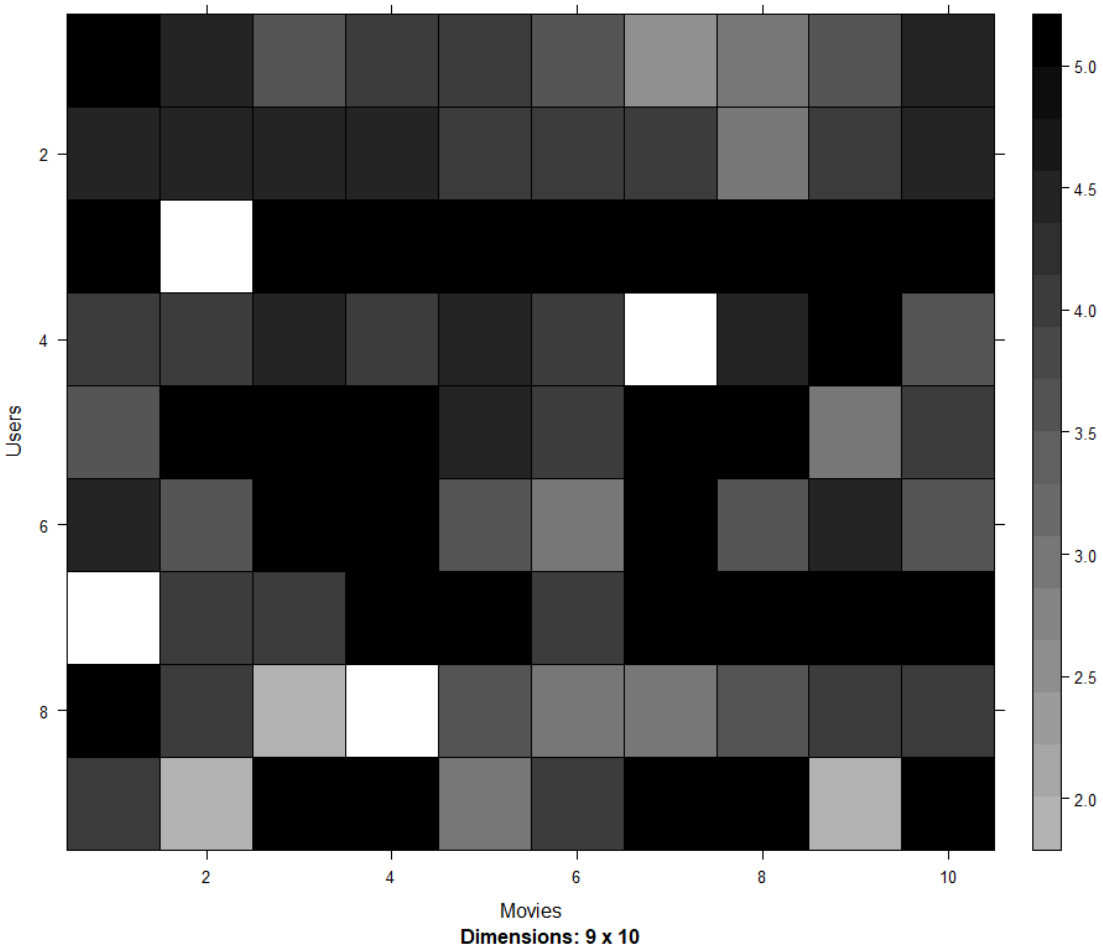
```
> # DATA PREPARATION select useful data/normalizing data/ binarizing data !!!!
> #select only movies with at least 50 ratings and users who've watch at least
50 movies
> selected_movie_ratings <- ratings_matrix[rowCounts(ratings_matrix) > 50,
+                                colCounts(ratings_matrix) > 50]
> selected_movie_ratings
439 x 459 rating matrix of class ‘realRatingMatrix’ with 39905 ratings.
```

> In the above code, we set a threshold value of 50. So here the row of the rating matrix represents userId and by > 50, we are only going to take those users who have watched at least 50 movies. Next the column of the rating matrix represents the movieId and by > 50, we are only going to take those movies which have greater than 50 ratings.

G.

```
> # Visualizing the top 2% of users and movies.
> top_2pct_movies<- quantile(rowCounts(selected_movie_ratings), 0.98)
> top_2pct_users <- quantile(colCounts(selected_movie_ratings), 0.98)
>       image(selected_movie_ratings[rowCounts(selected_movie_ratings)       >
top_2pct_movies,
+                      colCounts(selected_movie_ratings) > top_2pct_users],
+ main = "Heatmap of the top users and movies")
```

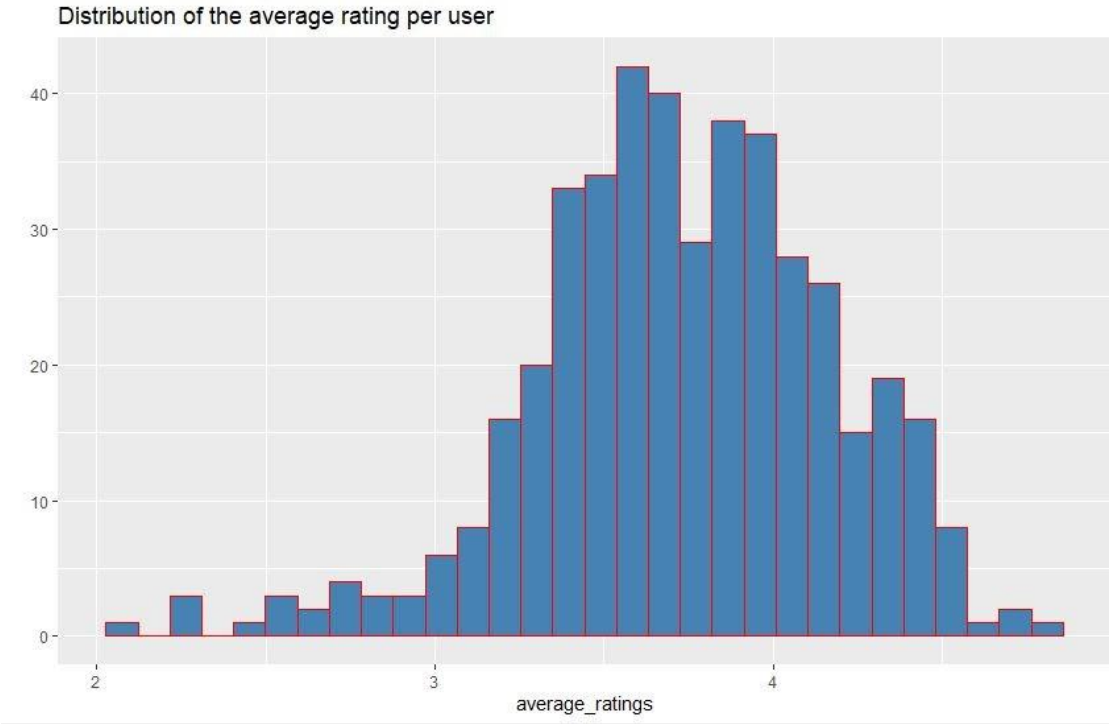
> The X axis represents top 2% of the movieIds and the Y axis represents the top 2% of the userIds. Now for example, it shows that Top User 1 has given a rating of 5 for the Top Movie 1.


Heatmap of the top users and movies
Users
2
4
6
8
2
4
6
8
10
Movies
5.0
4.5
4.0
3.5
3.0
2.5
2.0
Dimensions: 9 x 10

H.

```
> # Visualizing the distribuytion average rating / user.
> avg_ratings <- rowMeans(selected_movie_ratings)
> qplot(avg_ratings, fill=I("steelblue"), col=I("red")) +
+   ggtitle("Distribution of the average rating per user")
```

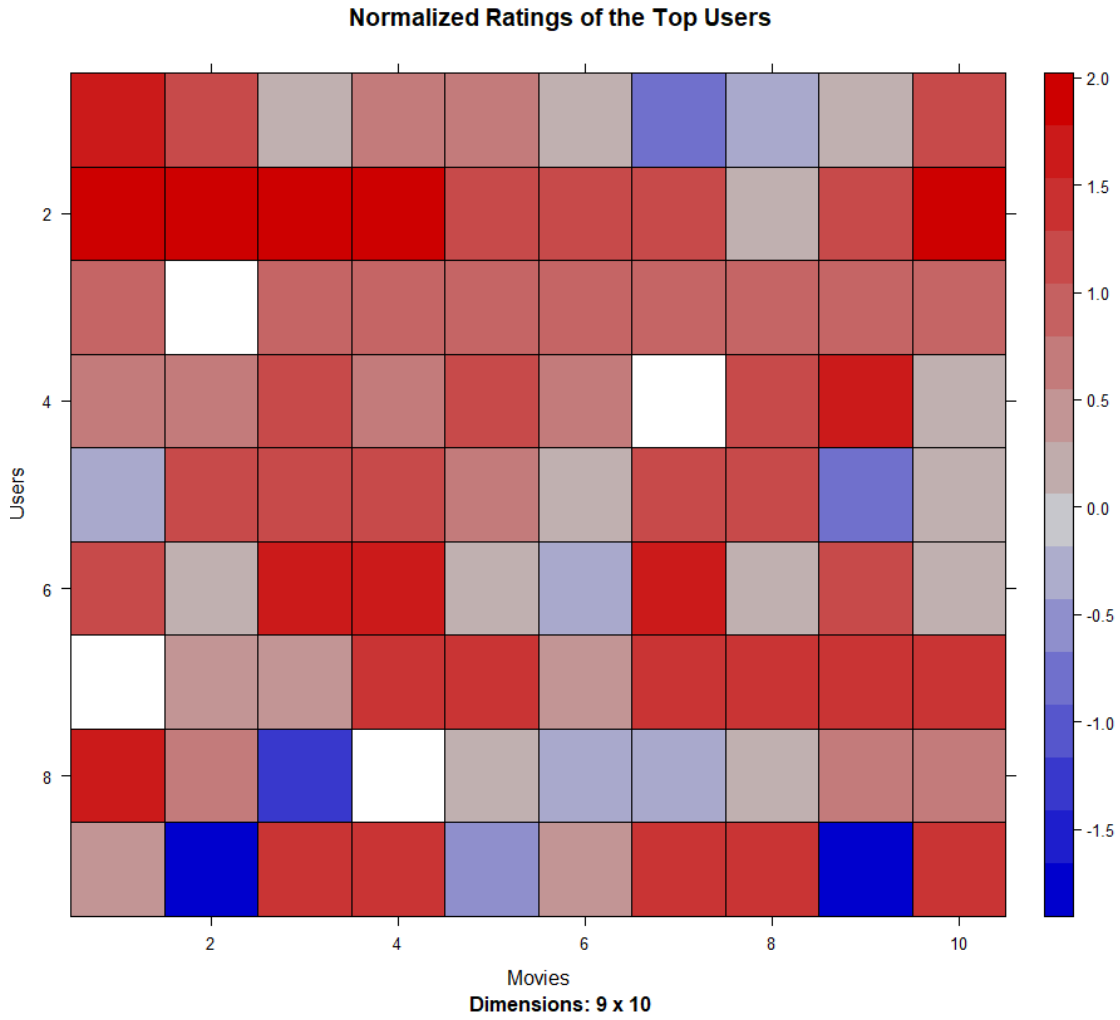
> In the above histogram plot, the X-axis displays the different ratings given by the users with minimum equal to 2 and maximum equal to 5. The Y-axis displays the number of users. So for example, from the above plot we can determine that at an average, rating 3.6 is given by the maximum number of users around 42.



I.

```
> # Normalize ratings to minimize biases.
> normalized_ratings <- normalize(selected_movie_ratings)
> sum(rowMeans(normalized_ratings) > 0.00001)
[1] 0
> image(normalized_ratings[rowCounts(normalized_ratings) > top_2pct_movies,
+                          colCounts(normalized_ratings) > top_2pct_users],
+ main = "Normalized Ratings of the Top Users")
>
```

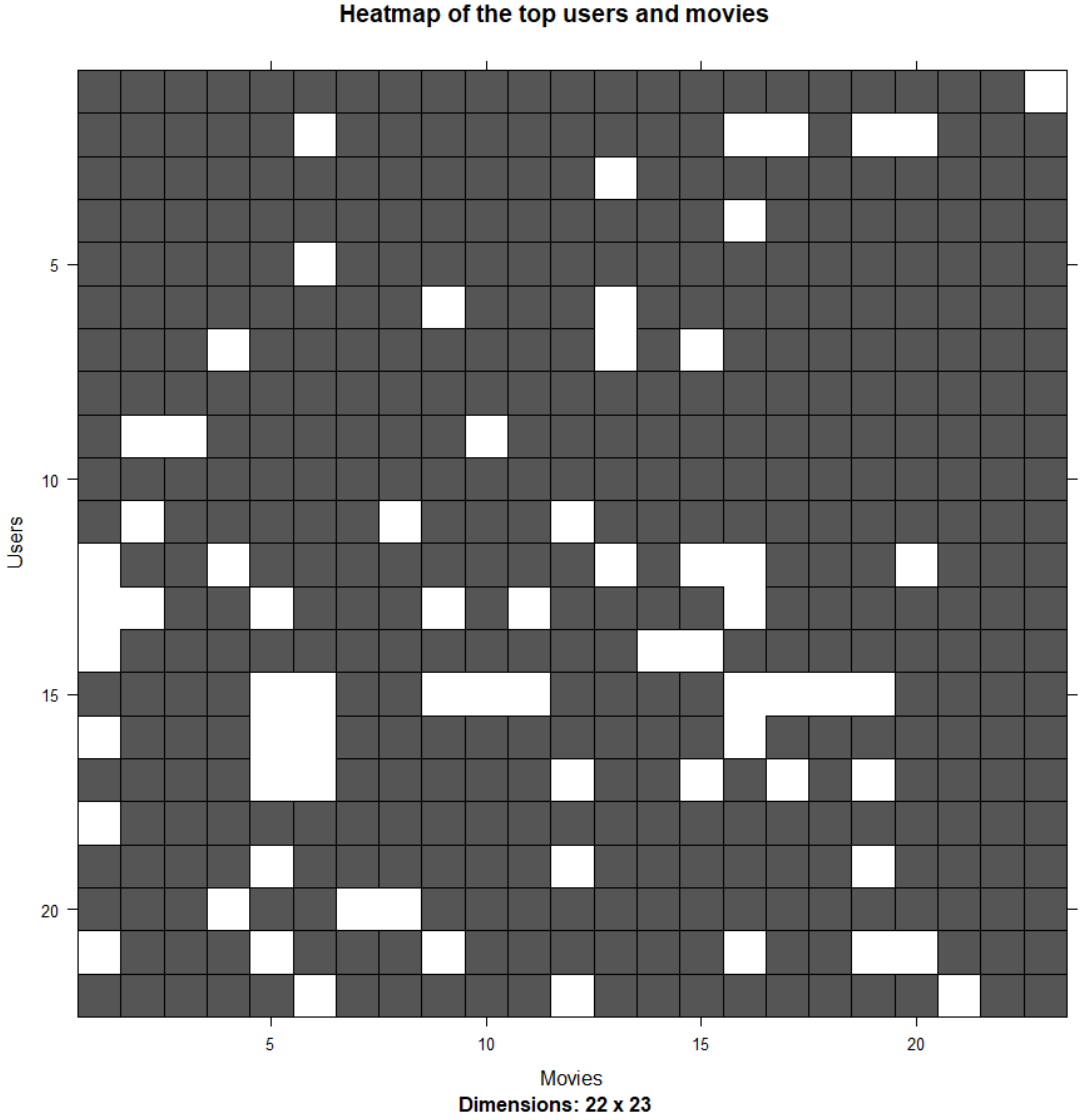
> In the above code, the default normalization technique has been used which is 'Standardize'. After implementing the above normalization technique, the resulting distribution has a mean of 0 and a standard deviation of 1. So our heat map, is predicting the normalized data wherein for example the data in the negative range represents that it is lesser than the mean rating and on the other hand the data in the positive range depicts that its value is greater than the mean rating. From our understanding, in the heat map, from range 0 to 2 represent the rating 3 to 5 whereas the range -2 to 0 represents 1 to 3. Thus, since in our last plot, the average rating was 3.6 we are assuming our mean to be around there. So, in the heat map grid, the one with the extreme blue color represents that the records(or movies) in those grids have received a very poor rating of around 1. On the other hand, the one with extreme red color represents the records(or movies) with a very high rating of around 5. Lastly, the ones that have been labelled as grey represent mediocrity or a rating of around 3. And lastly, the white one represents that there are no ratings for that movie.


Normalized Ratings of the Top Users
Users
2
4
6
8
Movies
2
4
6
8
10
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
Dimensions: 9 x 10

J.

```
> # ratings of above 3 are mapped to 1,
> # else will be mapped to 0
> binary_top_movies <- quantile(rowCounts(selected_movie_ratings), 0.95)
> binary_top_users <- quantile(colCounts(selected_movie_ratings), 0.95)
> top_rated_films <- binarize(selected_movie_ratings, minRating = 3)
> image(top_rated_films[rowCounts(selected_movie_ratings) > binary_top_movies,
+ colCounts(selected_movie_ratings) > binary_top_users],
+ main = "Heatmap of the top users and movies")
```

> In the above code, we are binarizing are data because it would help our Movie Recommendation System to work more efficiently. So all the movie ratings that are above 3 would be replaced by 1 whereas the movie rating less than 3 would be replaced by 0. For the heatmap, the X-axis represents the movie ids and the Y-axis represents the row ids. The heat map shows the binary data, wherein the black grids represent movie with ratings higher than 3 which was replaced with 1. And on the other hand, the white grid represents movie with ratings lower than 3 which was replaced with 0.



K.

```
> #collaborative filtering system
> data_sample<- sample(x = c(TRUE, FALSE),
+                      size = nrow(selected_movie_ratings),
+                      replace = TRUE,
+                      prob = c(0.8, 0.2))
> X <- selected_movie_ratings[data_sample, ]
> y <- selected_movie_ratings[!data_sample, ]
```

> In the above code, we created two datasets namely training_data and testing_data. The training_data dataset is 80% of the original dataset. The testing_data dataset is 20% of the original dataset.

L.

```
recommendation_system <- recommenderRegistry$get_entries(dataType
="realRatingMatrix")
recommendation_system$IBCF_realRatingMatrix$parameters
```

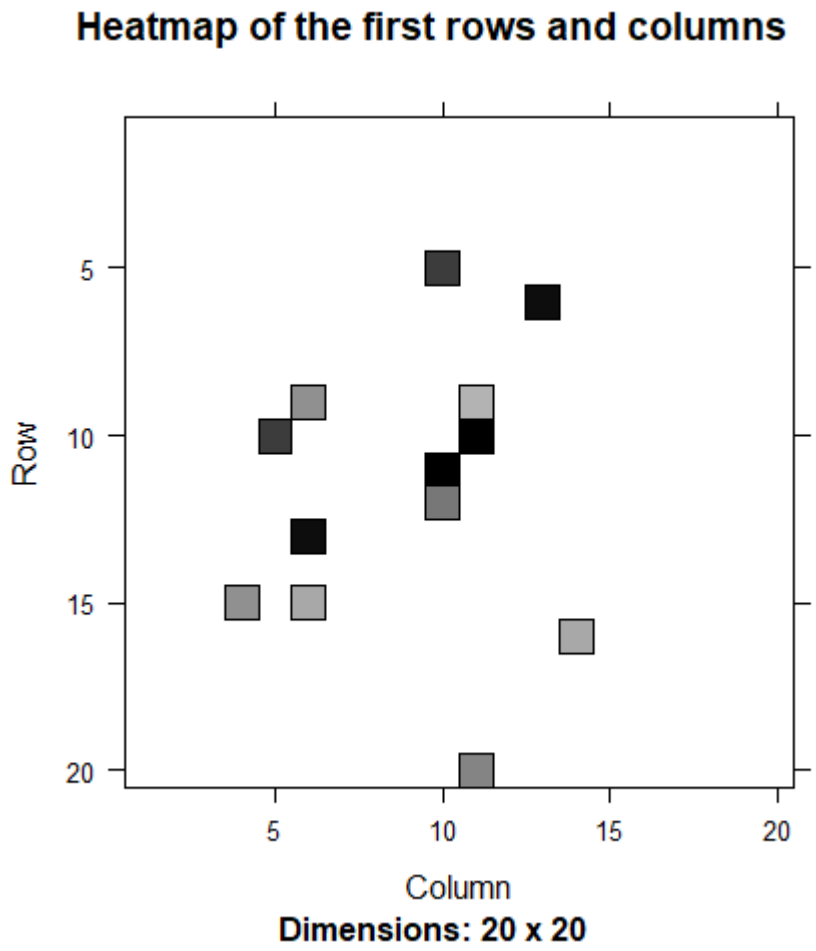
```
recommender_model <- Recommender(data = X,
                          method = "IBCF",
                          parameter = list(k = 30))
recommender_model
class(recommender_model)
```

> In the above code we are creating a Item Based Collaborative Filtering (IBCF) wherein we find the similarity in items based on the people's ratings of them. Then k denotes the number of items for computing their similarities. After which the algorithm would find k most similar items and store their ids.

M.

```
> model_info <- getModel(recommender_model)
> class(model_info$sim)
[1] "dgCMatrix"
attr(,"package")
[1] "Matrix"
> dim(model_info$sim)
[1] 459 459
> top_items <- 20
> image(model_info$sim[1:top_items, 1:top_items],
+    main = "Heatmap of the first rows and columns")
>
```

> In the above code, we are using the getModel() function to find the recommend_model. After which we will find the class and dimensions of the similarity matrix. lastly, we will generate a heat map that will contain the top 20 items (movies) and visualize the similarity between them.



N

```
> # give recommendation to users
> top_recommendations <- 10 # the number of movies to recommend to each user
> predicted_recommendations <- predict(object = recommender_model,
+                          newdata = y,
+                          n = top_recommendations)
> predicted_recommendations
```

Recommendations as 'topNList' with n = 10 for 86 users.

> We will create a top_recommendations variable which will be initialized to 10, specifying the number of films to each user. We will then use the predict() function that will identify similar items and will rank them appropriately. Here, each rating is used as a weight. Each weight is multiplied with related similarities. Finally, everything is added in the end.

0.

```
> number_of_recommendation <- factor(table(recommendation_matrix))
```

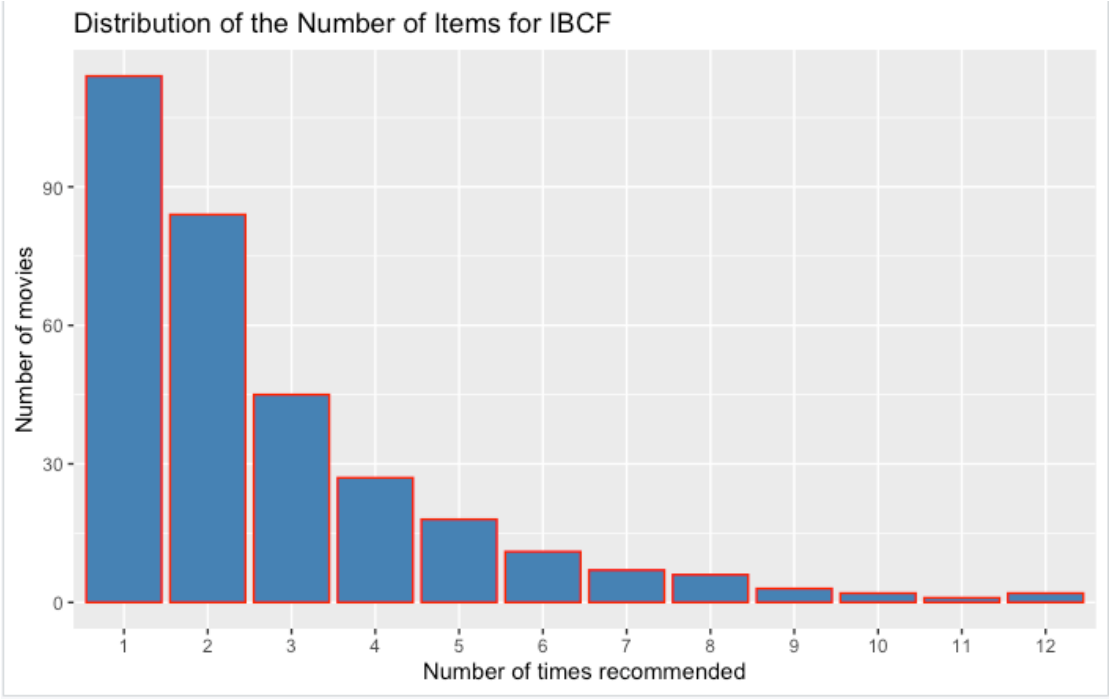
```
> qplot(number_of_recommendation, fill=I("steelblue"), col=I("red"),
+       ylab = "Number of movies", xlab = "Number of times recommended")
```

> In the above code, we are factoring the recommendation matrix to partition the movies into 12 categories which are, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11 and 12. In this plot, the X-axis represents the 'Number of times a movie was recommended' Y-axis represents the 'Number of movies'.

If we take X-axis = 11

If we take Y-axis = 1

it shows that only one particular movie was recommended 11 times over a range of 86 users



<u>Summary:</u>

1. We found about the different filtering that are used in real-life situations by the OTT platforms like Netflix, Amazon Prime and all. It helped us understand the different criterion that are used by these platforms to suggest movies to us.

2. We also learnt about how these OTT platforms might be binarizing the data to make efficient and accurate movie suggestions to us.

3. Thirdly, we also learnt about how the Cosine Similarity function works. And how they use the angle between two vectors to show the similarity between the two. Besides, we are also assuming that the reason for the ratingMatrix diagonal to have 0.00000 is because the Cosine function does not evaluate the vectors if they have the same labels.

4. Using Ming Xuan's laptop which has 16GB RAM, we tried to run and process the data. However the most intriguing part, was that although the R Studio did not use the entire RAM, his laptop crashed and just shut down.