itle: "MovieLens Project"
uthor: "Pedro Cornejo"
ate: "01/28/2020"
utput:
pdf_document: default
html_document: default
fontsize: 12pt

** Executive Summary**.

The main idea is to build a recommendation system able to predict the users movie preferences. After loading the data set, the first thing to to is to explore it to have an idea of possible predictors and the dependent variable. str() or glimpse() are the fucntions to start the data visualization. In our case, we get the rating of the movies , the type of movies, the year of released movies, the users, and the title of movies. Apperently the dependent variable is the rating column and the rest are the predictors. Therefore the hypothesis is Movie rating is a function of ~ type of movie, the genres, the users, and the movies'Year.–

To ensure that this hypothesis is not a subjective proposition it is crucial to visualize the data distribution and get the final answer with predictors' correlation. The data will be wrangled using PCA as the first step of data exploring, next the visualization to finally obtain the verification of the hypothesis accordingly.–

The final step is to create the model that will predict the depended variable. The data will be normalized to avoid high scales to over influence the results and the model tunning will give an RMSE lower than 0.8649.

Loading the data.

The libraries and database are loaded from Cran.us.r-project and https://grouplens.org/datasets/movielens/10m/, http://files.grouplens.org/datasets/movielens/ml-10m.zip, the MovieLens 10M, respectevily.

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

```
## Loading required package: tidyverse

## -- Attaching packages -------------------------------------------------------------------------------------

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ----------------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

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

```
## Loading required package: lubridate

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following object is masked from 'package:base':
##
##     date
```

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

```
## Loading required package: RColorBrewer
```

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

```
## Loading required package: knitr
```

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

```
## Loading required package: corrplot

## corrplot 0.84 loaded
```

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

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

```r
if(!require(devtools)) install_github("vqv/ggbiplot")
```

```
## Loading required package: devtools

## Loading required package: usethis
```

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

```
## Loading required package: kableExtra

##
## Attaching package: 'kableExtra'

## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```r
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(RColorBrewer)
library(knitr)
library(corrplot)
library(rpart)
library(ggbiplot)
```

```
## Loading required package: plyr

## ------------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##
## Attaching package: 'plyr'

## The following object is masked from 'package:lubridate':
##
##     here

## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following object is masked from 'package:purrr':
##
##     compact

## Loading required package: scales

##
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor

## Loading required package: grid
```

```r
library(kableExtra)
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")

# Validation set will be 10% of MovieLens data

set.seed(1, sample.kind="Rounding")
```

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

```
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)

# Creating the train_set: edx
edx <- movielens[-test_index,]

# Creating the test_set: temp
temp <- movielens[test_index,]

# Unifying data sets.
# The taring and test dataset requires to have the same features
# using the semi_join().

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)
```

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

```
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)

# Checking data
str(edx,4)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

```r
glimpse(validation,4)
```

```
## Observations: 999,999
## Variables: 6
## $ userId    <int> ...
## $ movieId   <dbl> ...
## $ rating    <dbl> ...
## $ timestamp <int> ...
## $ title     <chr> ...
## $ genres    <chr> ...
```

```r
dim(edx)
```

```
## [1] 9000055       6
```

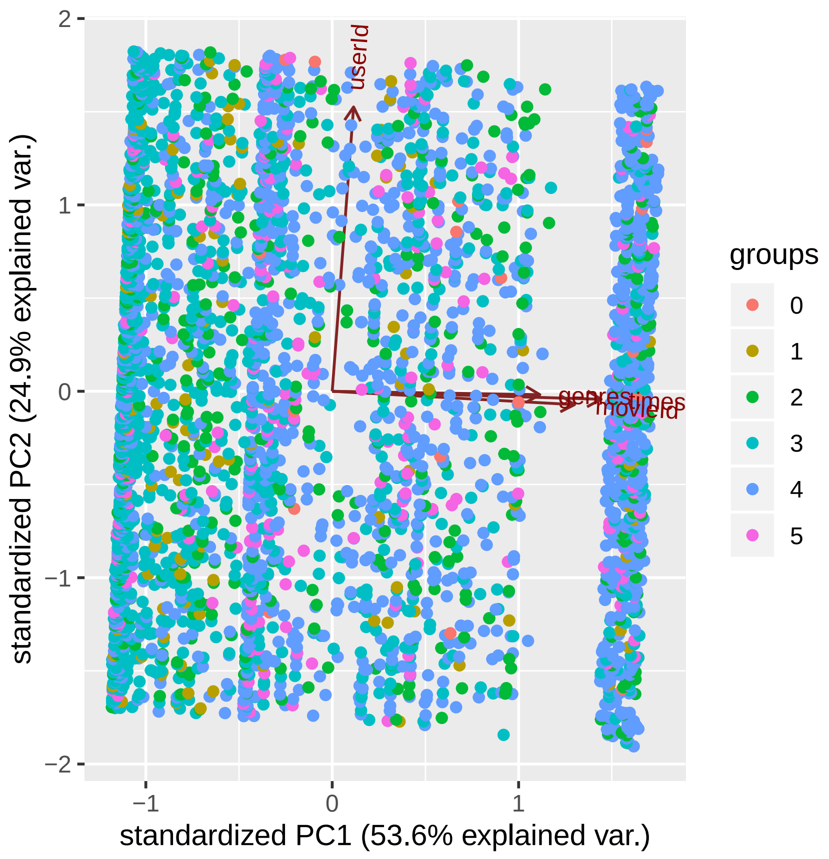
** Exploring the data set

Princiapal Component Analysis (PCA) is used to reduce dimmensions of models. The PCA is able to pack one hundred variables in a small group that will permit to have a better visualization of data. However,its process requires to build the principal components displayed and check the correlation of the varibles. This step is the tool for using PCA as data explorer, it will givce the first impresion in how the data is distributed and the varianbles are correlated.– The dimmension of the data set is big, so it is necessaruy to take a small sample and do the PCA analysis. The prcomp() requires a data matrix to extract the main PCAs, so the numerical data matrix is built. Here a trick is done the genres are changed to numeric to have this data as part of the analysis. A dummy number would be better if a PCA analysis is done, but in this case we only need to visualized the data trend .– partrequired for the function pr The data is plotted and we can see that the variable userId is the principal component and tah data follow its trend. The timestamp and genres vectors show a strong correlation with the userId a,d the whole data is evenly distributed. Now, we can tell that the data set has a real correlation and our hypothesis has a strong objective proposition.– The final step is added to show how PCA reduce from four components to three components witout lossing accuracy, using three or for PCAs will give the same result RMSE 1.29. In cases where the data is numerical and multiple variables this analysis will help to reduce variable and obtain good results. In our case, PCA help us to know the data trend and their correlation, and get ready to start the data visualization knowing that we will get the desire results, which is that the hypothesis is objectived.

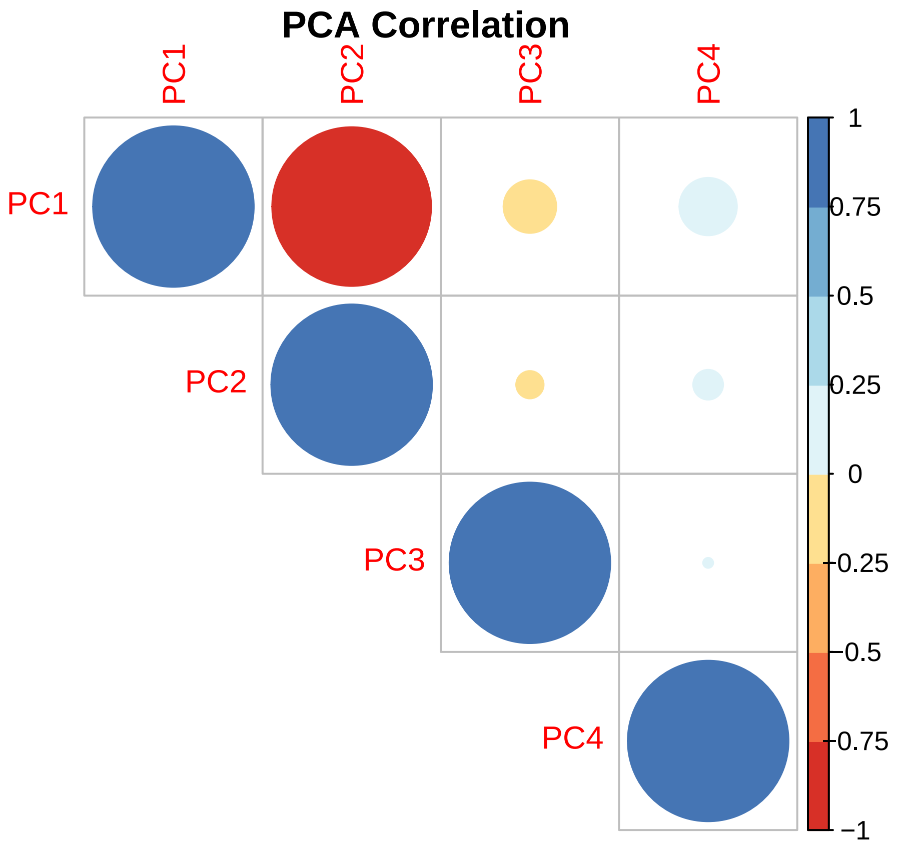
```r
Data_sample_pca <- sample(unique(edx$movieId), 10)
Edx_sample_pca <- subset(edx, edx$movieId %in% Data_sample_pca)
edx.data <- Edx_sample_pca[, c(1:4, 6)]
edx.data$genres <- as.numeric(as.factor(edx.data$genres))
edx.quanti <- as.matrix(edx.data[,c(1,2,4,5)])
row.names(edx.quanti) <- edx.data$rating
prin_comp <- prcomp(edx.quanti, center = TRUE, scale = TRUE)

# Summary gives the principals components variance
# Adding PC1 to PC3 give more tah 85% of dta variance

summary(prin_comp)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4
## Standard deviation     1.4649 0.9986 0.8251 0.41950
## Proportion of Variance 0.5365 0.2493 0.1702 0.04399
## Cumulative Proportion  0.5365 0.7858 0.9560 1.00000
```

```r
rate.pca <- as.factor(round(edx.data$rating,0))
ggbiplot(prin_comp,  groups = rate.pca)
```



```
pca.corr <- cor(as.matrix(prin_comp$rotation))

# Plotting correlation to ensure that PCAs are correlated

corrplot(pca.corr, type="upper", col=brewer.pal(n=8, name="RdYlBu"),
         mar=c(0,0,1,0),title = "PCA Correlation")
```

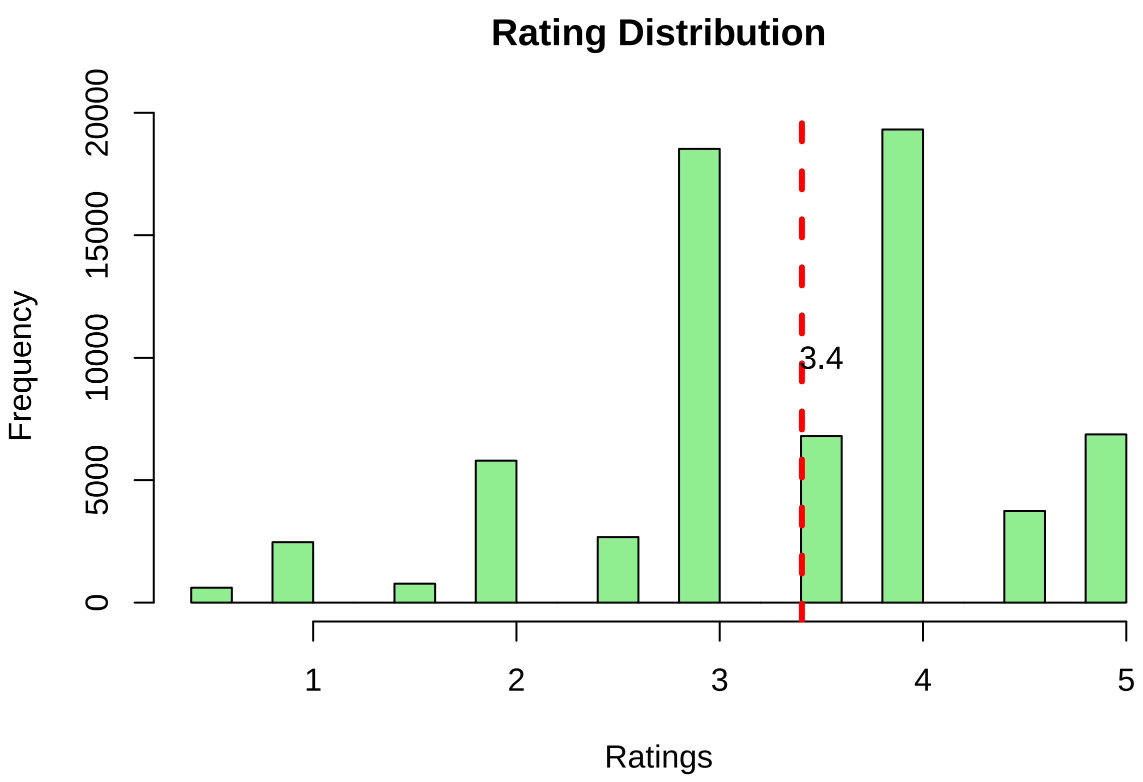


```
# the varibles are correlated and we are ready to start the analysis.
```

Building the Hypothesis

The database has 900005 observations and 6 features or predictors. To have faster data visualization, a random sample is captured from edx, the training set. Using this partial database, the distribution and the correlation of the features are set up. (Irizarry, 2019, para 28) explains that it is helpful to sample a database when it is too big, facilitating to have a faster data management that allows knowing the distribution of the whole set.– **Rating distribution**.– First, it is necessary to know the distribution of the ratings to check if movies have a standard rate or some movies have preferences. The histogram shows that 3 and 4 stars are the more common ratings with a distribution mean = 3.3 that will help to normalize the whole data.

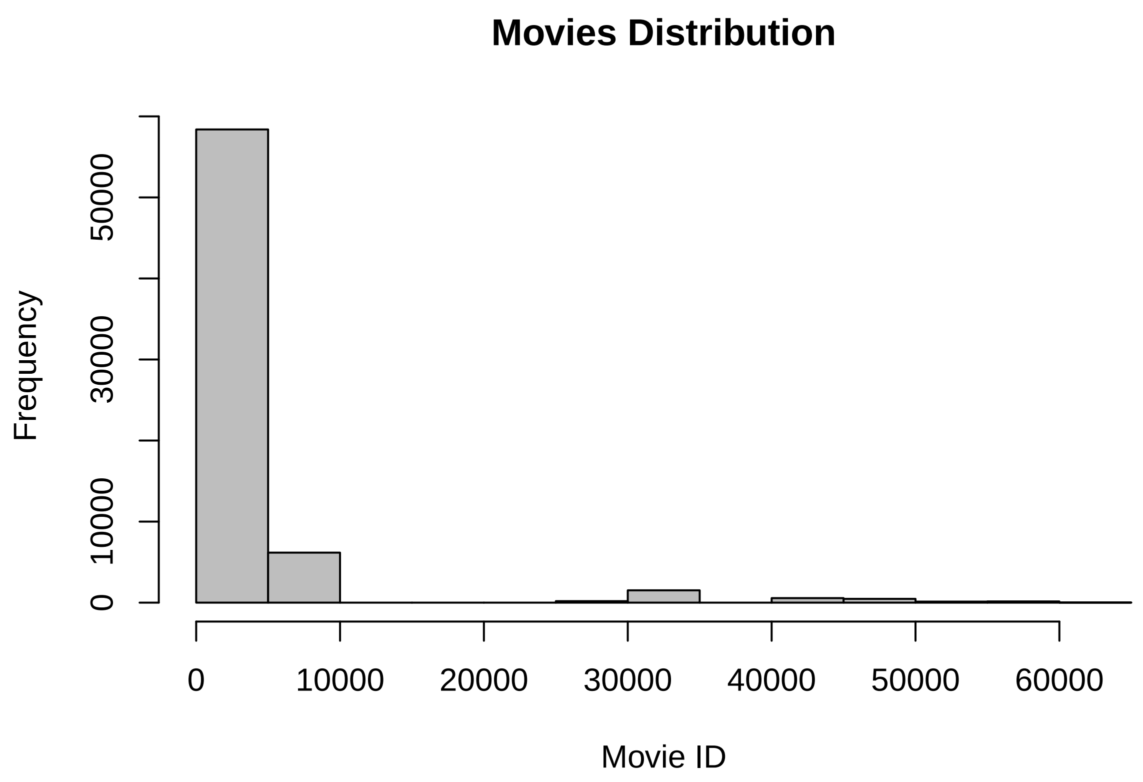
```
 Data_sample <- sample(unique(edx$movieId), 100)
Edx_sample <- subset(edx, edx$movieId %in% Data_sample)

# Checking the distribution of Ratings
RatingMeans <- mean(Edx_sample$rating)
hist(Edx_sample$rating, col= "lightgreen", main ="Rating Distribution", xlab = "Ratings")
abline(v = mean(Edx_sample$rating), col="red", lwd=3, lty=2)
text(x=3.5, y=10000, labels =round(RatingMeans,2) )
```



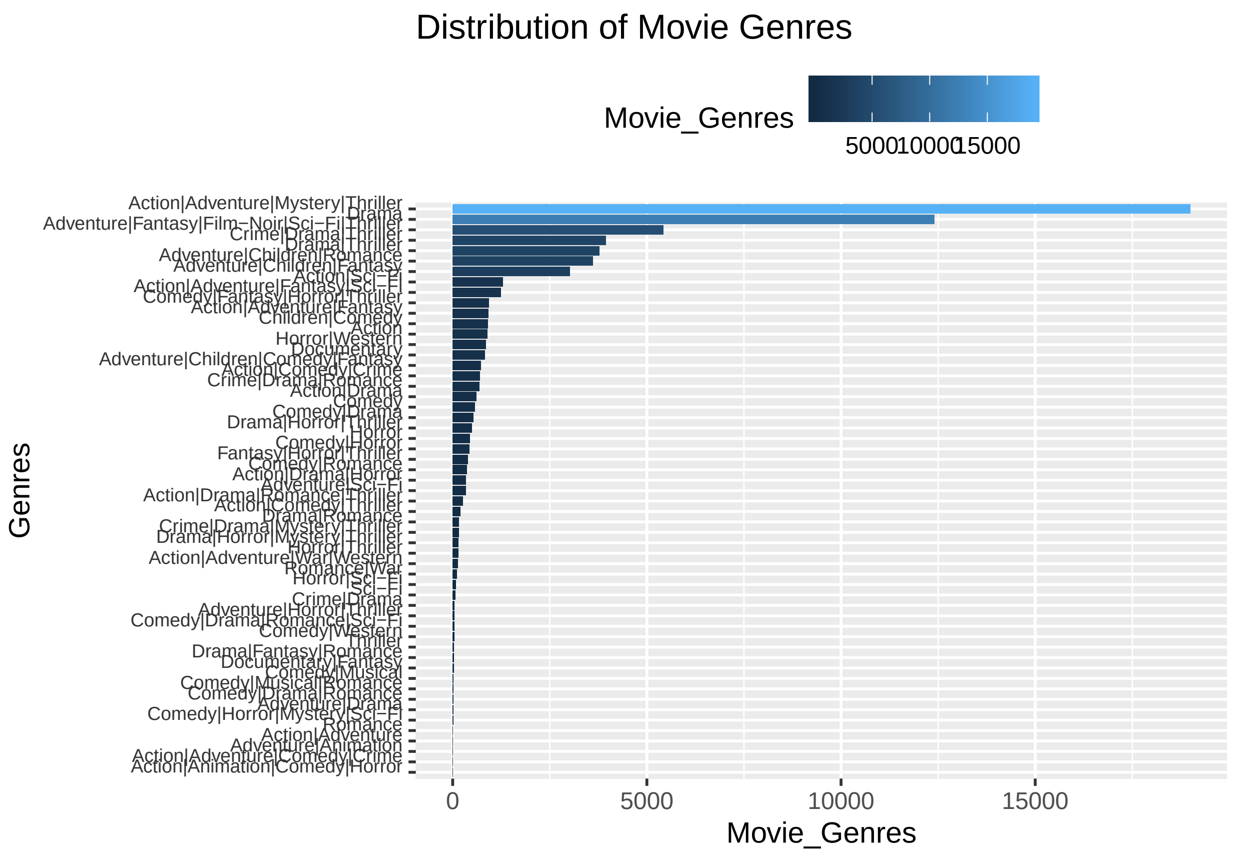
```
# The mean of rating will give an accurate measure because in the graphic we can see 3
# and 4 stars as the more frequent rating
```

Movies and Genres distribution.– The first graphic told us that there are preferences in certain movies. It is important to know which movies are them. "The Movies Distribution"'s bars show the movieIds preferences, which are seen more frequently than others and are more frequently rated. The movie genres distribution will match movieId and genres to know that genres more seen, and later look for the correlation between genres and ratings.–
The **'Distribution of Movie Genres'**'s graphic shows which Genres are the more seen. Therefore, the hypothesis is getting more realistic, the ratings depend on genres and the type of movies.

```
# Checking the distribution of movies shows there are some movies that are more frequently
# seen and rated.
# Now, it is imporant to know which movies genres are they
hist(Edx_sample$movieId, col = "grey",main = "Movies Distribution", xlab = "Movie ID")
```



```
# The are some movies that are more frequently seen and rated. It is important to know which
# movies genres are them.
Genres_dist <- Edx_sample %>% dplyr::group_by(genres)%>% dplyr::summarise(Movie_Genres = n())%>%
  arrange(desc(Movie_Genres))%>% ungroup()

# Using http://r-statistics.co/ggplot2-Tutorial-With-R.html as refrence for plotting:
Genres_dist %>% ggplot(aes(reorder(genres,Movie_Genres), y = Movie_Genres,
                           fill =Movie_Genres)) + geom_col() +
  coord_flip() +theme(legend.position = "top") +
  theme(axis.text.y = element_text(color = "grey20",
                                   size =7, angle = 0, hjust = 1,
                                   vjust = 0, face = "plain"))+
  labs( x = 'Genres', title = 'Distribution of Movie Genres')
```



```r
nrow(Genres_dist)
```

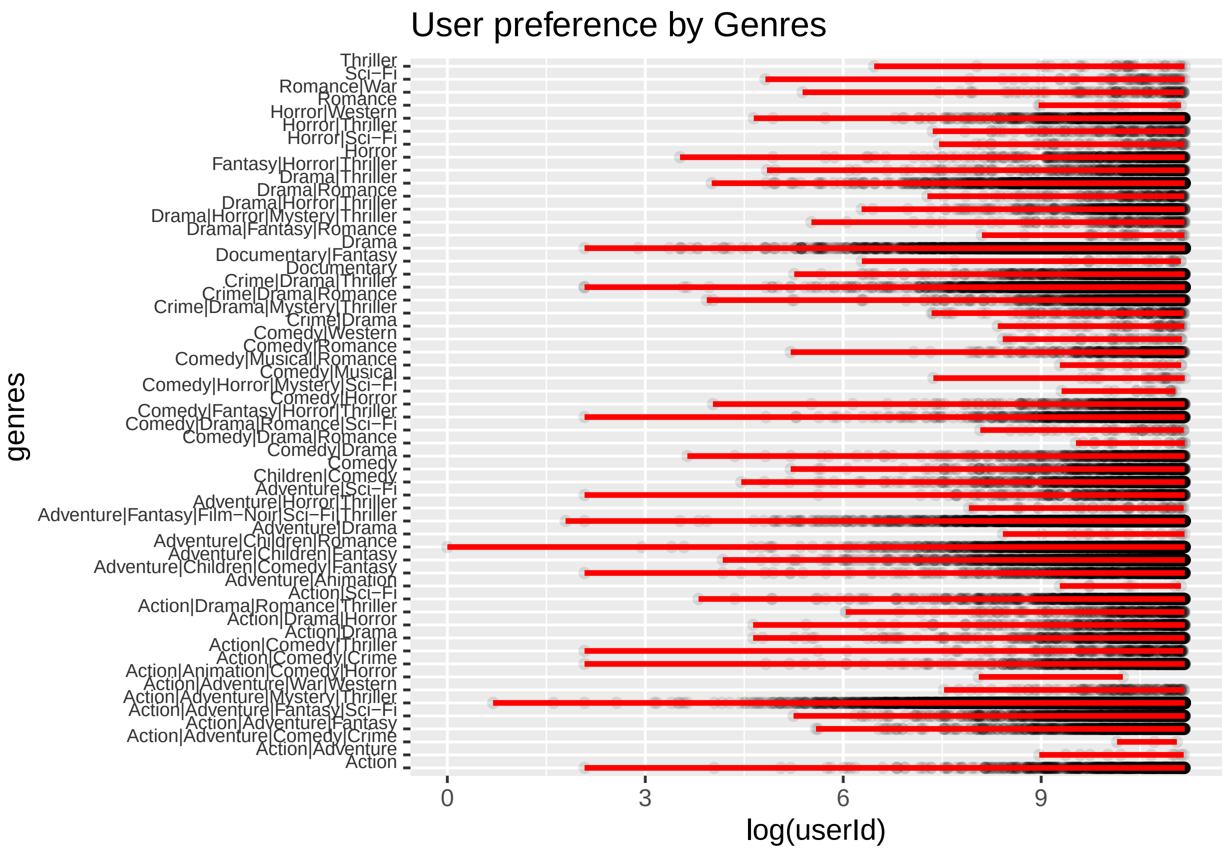
```
## [1] 55
```

```r
# There are 55 Genres and some of them have high preference and ratings. Therefore,
# we can hypotize that ratings depend on genres and movies.
```

"User preference by Genres".– Up to now, the hypothesis is that ratings depend on genres and the type of movies (movieIds). The users and the year's influence are not seen yet. The **User preference by Genres**' graphic shows is prepared to obtain the relationship between ratings and users' preferences. Visualizing he graphics allows finding the relation between users and genre preferences. Action, adventure,and thrillers are prevalant. Now, the hypothesis is: Ratings depend on the type of movie, the genres, and the users. Next, is to ensure that time is related to ratings.–

```r
######### User and movies's year ################



# Now, the distrubution of Users and genres will show if genres and users have dependency.
# using Phillips, YaRrr! The Pirate's Guide to R  2018https://bookdown.org/ndphillips/YaRrr/,
# the plot is created and http://r-statistics.co/ggplot2-Tutorial-With-R.html

p <- Edx_sample%>% ggplot(aes(x = log(userId), y = genres))+
  geom_point(alpha=0.1) +
  geom_smooth(color = "red", method = "lm") +
  theme(axis.text.y = element_text(color = "grey20", size =7, angle = 0,
                                   hjust = 1, vjust = 0, face = "plain")) +
  ggtitle("User preference by Genres")
p
```

```
## Warning in qt((1 - level)/2, df): NaNs produced
```



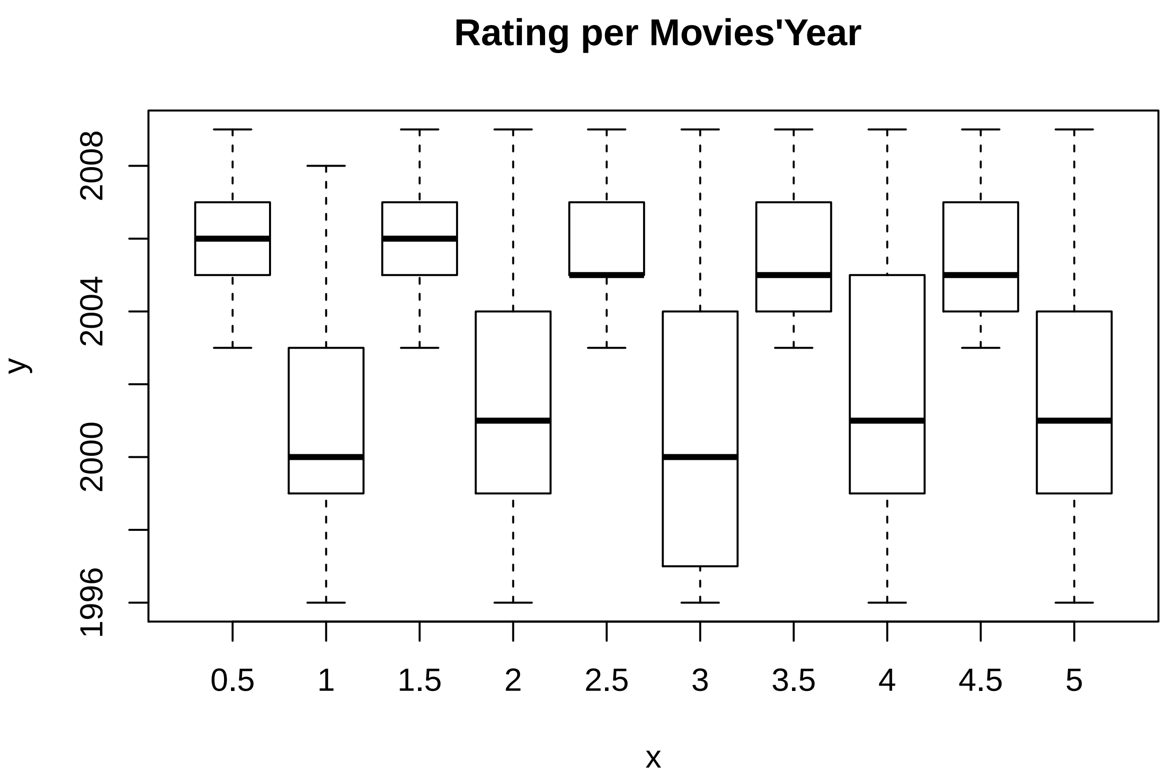
"

Rating per Movies'Year".– The predictor for time is in the timestamp format, so this format must be changed to POSIXct format to extract the year. The distribution " Rating per Movies'Year" shows some movies' year preferences. For instance 1997 to 2004, and years from 2004-2007 have 0.5 stars and 4.5 stars. Therefore, the rating of the movies depends on the year released and user preferences.– Then, the final hypothesis is: **Ratings depend on the predictors ~ type of movie, the genres, the users, and the movies'Year**. A hypothesis that must be corroborated using the correlation analysis of these predictors.

```r
# Now , we need to work with movies's year
Rating_time <- Edx_sample %>% dplyr::group_by(timestamp,rating)%>%
  dplyr::summarise(RatingTime = n())%>%ungroup()

Rating_time$Stars  <- as.factor(Rating_time$rating)
Rating_time$Movie_Year <- year(as.Date(as.POSIXct(Rating_time$timestamp,
                                                  origin="1970-01-01")))
levels(Rating_time$Stars)
```

```
##  [1] "0.5" "1"   "1.5" "2"   "2.5" "3"   "3.5" "4"   "4.5" "5"
```

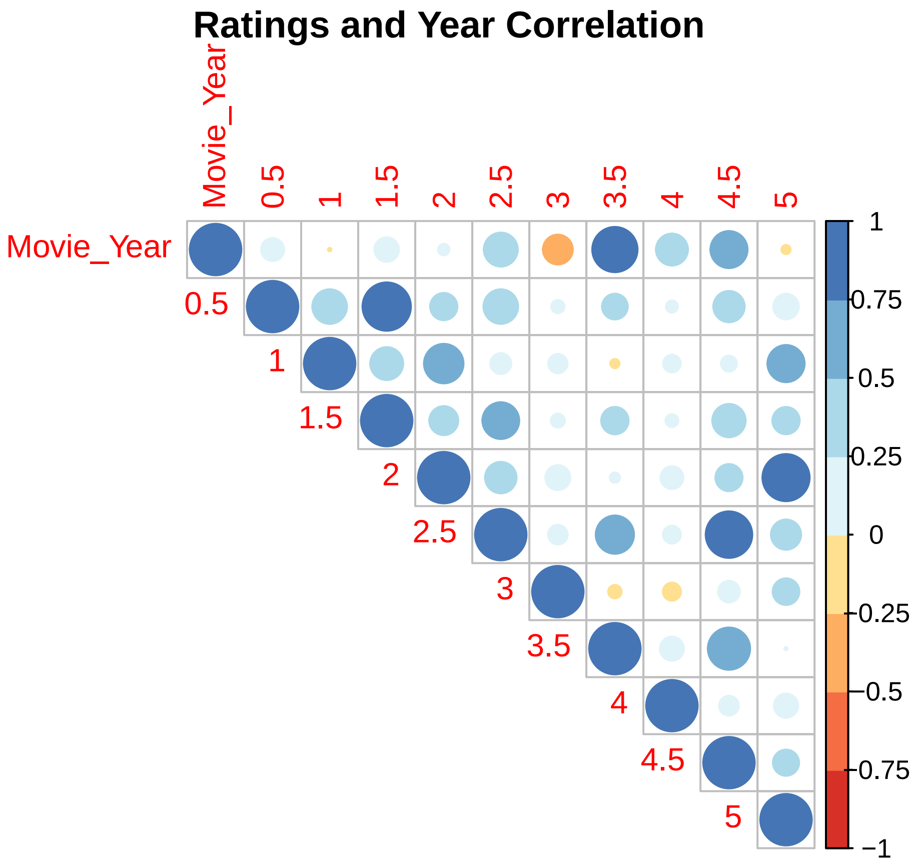
```r
# Checking years distrubtion, it is noticeable that certian years are more frequently seen
plot(Rating_time$Stars, Rating_time$Movie_Year, main= " Rating per Movies'Year")
```



Correlation Analysis.– Using Corrplot package allows controlling the colors and creating specific heatmaps according to users' needs. However, a matrix is required, so the database is changed to a matrix set. Using Cran-R_project.2017, the correlation is calculated and displayed with the coorrplot function.
Ratings and Year Correlation Map.– The highlighted blue rows shows that some years have a high correlation with ratings. Meaning, there is a dependency between ratings and time **Ratings and User Correlation Map**.– Similar to the latter one, the highlighted blue rows shows this time an even correlation between ratings and users. Ratings depend on users' preferences.–

The correlation between predictors has been stablished Ratings correlate with the movies'year and users. Genres and the type of movies are implied in the movies' year. They were not illustrated in this paper to save time and space, but both predictors show also the correlation with ratings.–

```r
### Correlation ####

# It is necessary to see correlation between users and ratings and time with ratings
# to create a credible hypothesis that ratings depends on genres, users and time

Rating_time <- Rating_time[,-c(1:2)]
Rtime <- Rating_time %>% dplyr::group_by_at(vars(-RatingTime)) %>%
  dplyr::mutate(row_id=1:n()) %>% ungroup() %>%
  spread(key=Stars, value=RatingTime) %>%
  select(-row_id)
Rtime[is.na(Rtime)] <- 0
Matrix_RatingTime <- data.matrix(Rtime)
# Some years has a high correlation with ratings. Meaning, there is a
# dependency between ratings and time
Correlation_RatingTime <- cor(Matrix_RatingTime)
corrplot(Correlation_RatingTime, type="upper", col=brewer.pal(n=8, name="RdYlBu"),
         mar=c(0,0,1,0),title = "Ratings and Year Correlation")
```



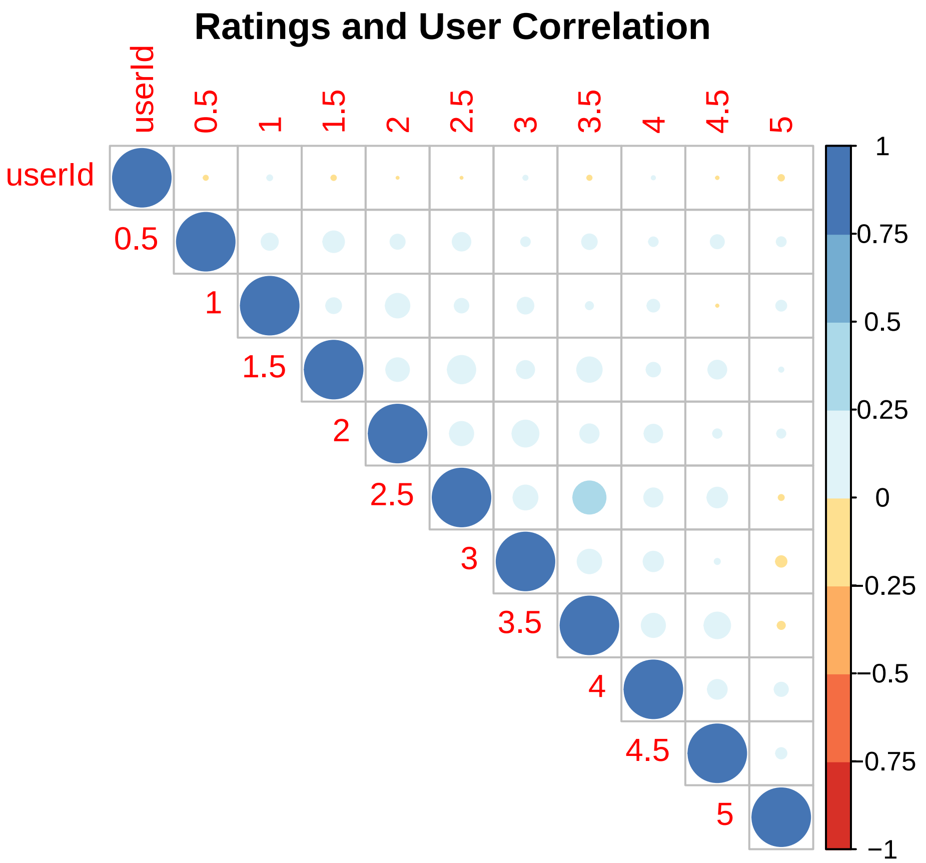
```
# Now, checking the correlation between ratings and users

Rating_user<- Edx_sample %>% dplyr::group_by(userId,rating)%>%
  dplyr::summarise(RatingTime = n())%>%ungroup()

Rating_user$Stars  <- as.factor(Rating_user$rating)
levels(Rating_user$Stars)
```

```
##  [1] "0.5" "1"   "1.5" "2"   "2.5" "3"   "3.5" "4"   "4.5" "5"
```

```
Rating_user <- Rating_user[,-2]

Ruser <- Rating_user %>% dplyr::group_by_at(vars(-RatingTime)) %>%
  dplyr::mutate(row_id=1:n()) %>% ungroup() %>%
  spread(key=Stars, value=RatingTime) %>%
  select(-row_id)
Ruser[is.na(Ruser)] <- 0
row.names(Ruser) <- Ruser$userId
```

```
## Warning: Setting row names on a tibble is deprecated.
```

```
Matrix_RatingUser <- data.matrix(Ruser)

# Users has an even correlation with ratings.
Correlation_RatingUser <- cor(Matrix_RatingUser)
corrplot(Correlation_RatingUser, type="upper",
         col=brewer.pal(n=8, name="RdYlBu"),
         mar=c(0,0,1,0),
         title = "Ratings and User Correlation")
```



```
# All the correlation maps show the dependecy on ratings and users
# Now, we need to normalized the predictors and ratings to make
# them ready to be used in predict()
```

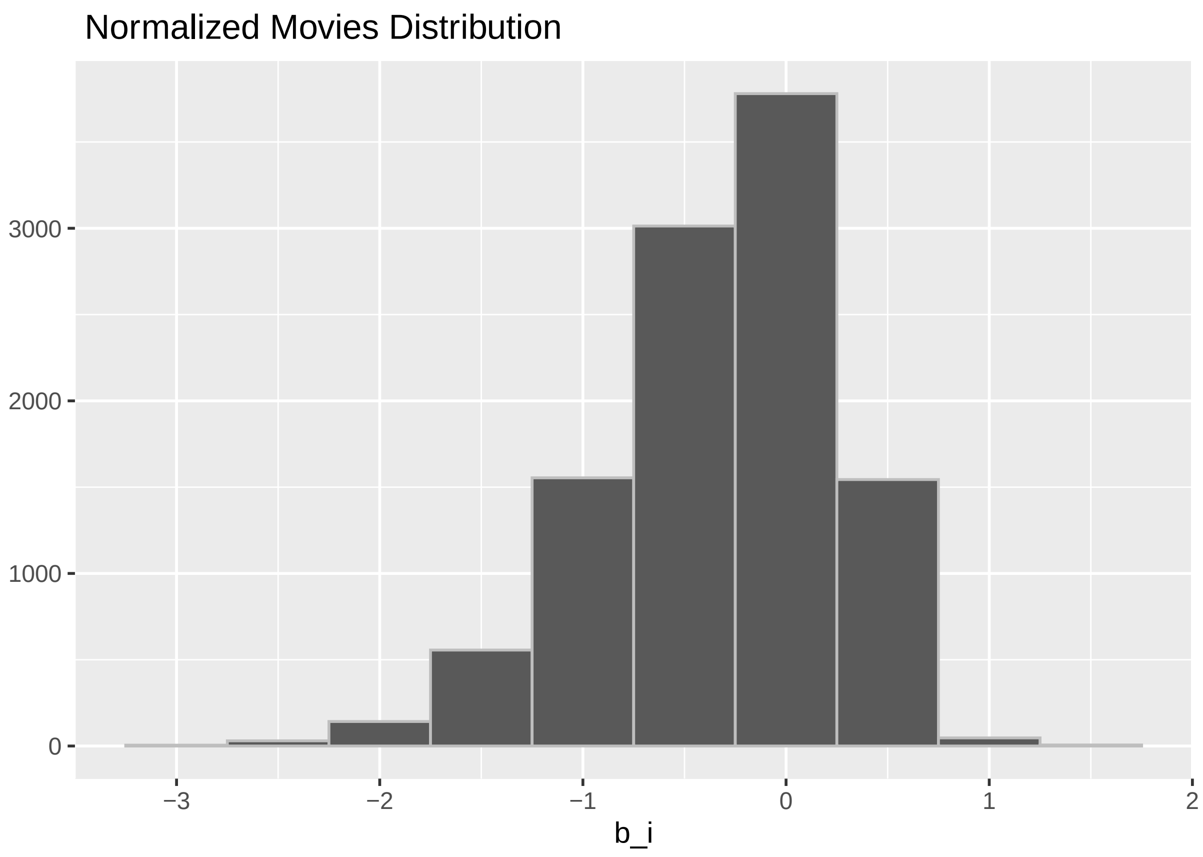
Methodology

Having the hypothesis verified by data and the correlation analysis, the model Ratings~ type movies + users preference+ movies' year + Genres can be applied. Using the database's predictors the model: **rating ~ movieID + userId + year(timestamp) + genres**, which is Yu,i= mu_hat + bi + bu +bt + eu,i, as Irizarry R. 2019 expressed.–

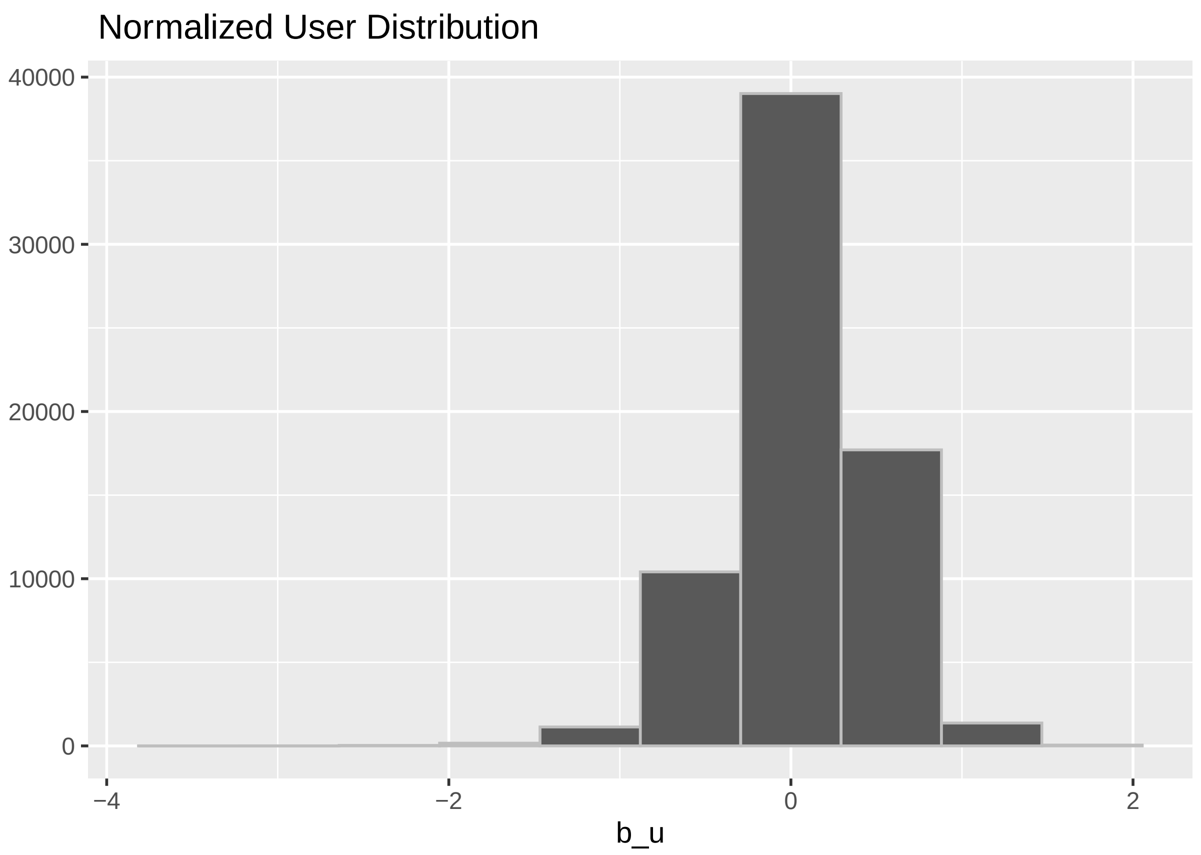
The RMSE prediction model is used so the predictors must be normalized to predict the factor rating. The theory is that y_hat = sum(of normalized predictors). y_hat is the mean of thevariable to predict, and the predictors will be changed to Z score and calculate the total mean using RMSE formula.– Therefore, the normalization of predictors is the first step. The Movies'year normalization is more complex because POSIXct is needed to get the year of the movie released, and the edx and validation datasets should have a new column Year as a predictor to be used instead oof timestamp.

```
####### Methodology #########

# Irizarry R. 2019, code RMSE. explais that RMSE > 1 is not a good
# prediction and its formula:
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))}
```

```
# Normalization of predictors

# Irizarry R. 2019 code, as a reference to build normalization.
# Movies normalization
mu_movies <- mean(edx$rating)
movie_avgs <- edx %>% dplyr::group_by(movieId)
movie_normalized <-  movie_avgs%>% dplyr::summarize(b_i = mean(rating - mu_movies))
#plotting distribution to verified movies normalization
movie_normalized%>% qplot(b_i, geom ="histogram", bins = 10, data = .,
                          color = I("grey"),
                          main= " Normalized Movies Distribution")
```



```
# Movies normalization
user_normalized <- edx %>%
  dplyr::left_join(movie_normalized, by='movieId') %>%
  dplyr::group_by(userId) %>%
  dplyr::summarize(b_u = mean(rating - mu_movies - b_i))
#plotting distribution to verified movies normalization
user_normalized%>% qplot(b_u, geom ="histogram", bins = 10,
                         data = ., color = I("grey"),
                         main= " Normalized User Distribution")
```



```r
# Movie's Year  normalization
# Timestamp format to POSIXct to get the year of released
time_normalized<-  edx %>% dplyr::left_join(movie_normalized, by ="movieId") %>%
  dplyr::left_join(user_normalized, by= "userId") %>%
  dplyr::mutate(date = year(as.Date(as.POSIXct(timestamp,
                                       origin="1970-01-01")))) %>%
  dplyr::group_by(date) %>%
  dplyr::summarize(b_t = mean(rating - mu_movies - b_i - b_u))

# adding year to validation and edx sets
edx_time <- edx %>%
  mutate(date = year(as.Date(as.POSIXct(timestamp, origin="1970-01-01"))))
validation_time <- validation %>%
  mutate(date = year(as.Date(as.POSIXct(timestamp, origin="1970-01-01"))))
```

First Prediction. All the database is normalized and they are used as the training. A data frame that includes all the predictors are built, and the prediction is using the model established: rating ~ movieID + userId + year(timestamp) + genres. ** which is Yu,i= mu_hat + bi + bu +bt + eu,i**. Then, the RMSE function is called to get the first value predicted.

```r
# First prediction
predicted_ratings_hypothesis <- validation_time %>%
  dplyr::left_join(movie_normalized, by="movieId")%>%
  dplyr::left_join(user_normalized, by='userId') %>%
  dplyr::left_join(time_normalized, by='date') %>%
  dplyr::mutate(pred = mu_movies + b_i + b_u + b_t) %>% .$pred
Hypothesis_rmse <- RMSE(validation_time$rating, predicted_ratings_hypothesis)
```

```r
# The table that will store all the RMSE results. Using Irizarry R. 2019 code as aguide to
# build our code for rmse results dataframe
rmse_results <- data_frame(method = "Movie Effect Model + User Effect + Time_Released Effect",
                           RMSE = Hypothesis_rmse)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

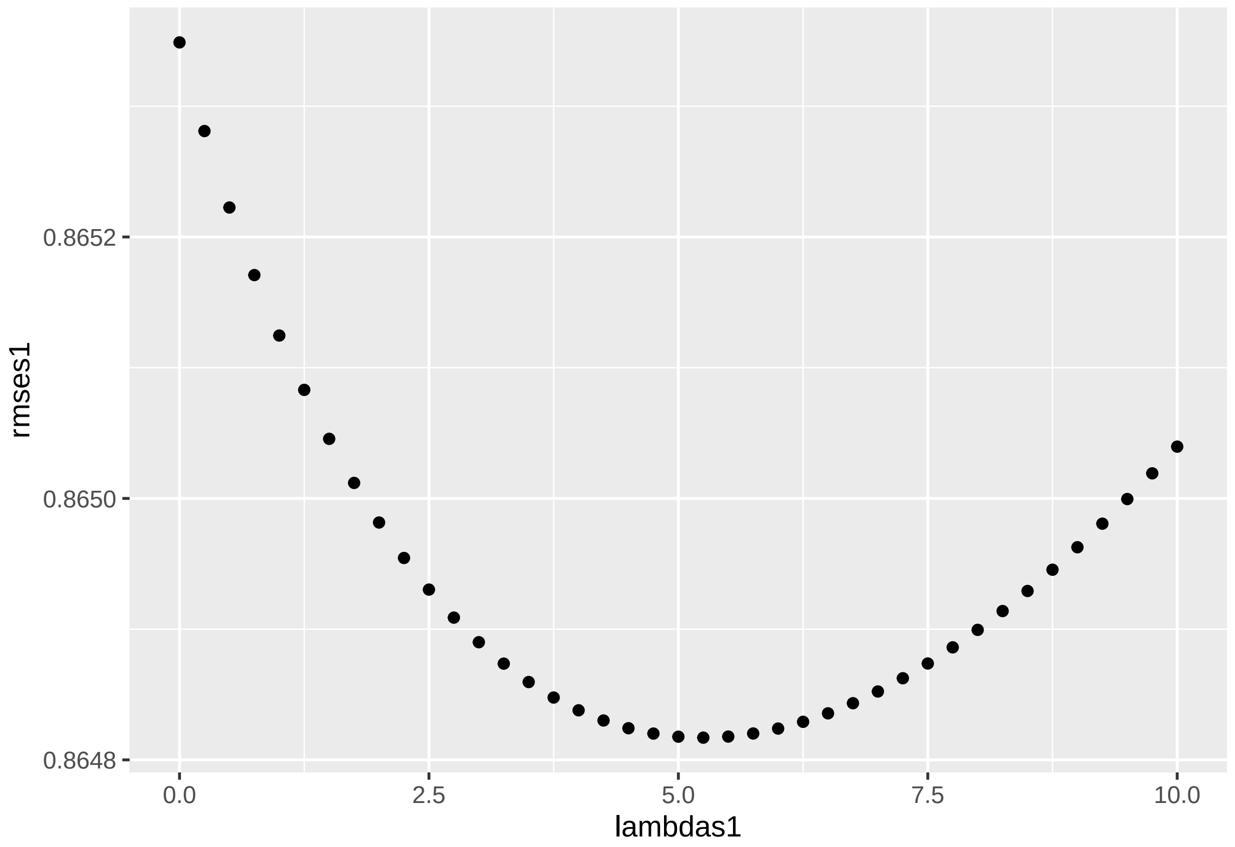
```r
rmse_results %>% knitr::kable()
```

method	RMSE
Movie Effect Model + User Effect + Time_Released Effect	0.8653369

Regularization of the model.Irizarry.2019, lambdas exemplified how to find the best lambda to make the model more effecient. The lambda is found over the movieId and userID, the more correlatedpredictors, and then, it is applied to include the movie's year, and genres.

```r
# Regularization of lambda, to improve RMSE

###################
# Using lambdas
####################
# Regularization of lambda, to improve RMSE


#######################
#### Using lambdas ####
#######################



lambdas1 <- seq(0, 10, 0.25)

rmses1 <- sapply(lambdas1, function(l){

  mu_lambda1 <- mean(edx$rating)

  bi_l <- edx %>%
    dplyr::group_by(movieId) %>%
    dplyr::summarize(bi_l = sum(rating - mu_lambda1)/(n()+l))

  bu_l <- edx %>%
    dplyr::left_join(bi_l, by="movieId") %>%
    dplyr::group_by(userId) %>%
    dplyr::summarize(bu_l = sum(rating - bi_l - mu_lambda1)/(n()+l))

  predicted_ratings_lambda <-
    validation %>%
    dplyr::left_join(bi_l, by = "movieId") %>%
    dplyr::left_join(bu_l, by = "userId") %>%
    dplyr::mutate(pred = mu_lambda1 + bi_l + bu_l) %>%
    .$pred

  return(RMSE(validation$rating, predicted_ratings_lambda))
})
```

```
qplot(lambdas1, rmses1)
```



```
lambda1 <- lambdas1[which.min(rmses1)]
lambda1
```

```
## [1] 5.25
```

```
model_Regularized <- min(rmses1)
model_Regularized
```

```
## [1] 0.864817
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model + User Effect  + Regularization",
                                     RMSE = model_Regularized ))

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

method	RMSE
Movie Effect Model + User Effect + Time_Released Effect	0.8653369
Movie Effect Model + User Effect + Regularization	0.8648170

```
# Using lamba1 to include time
movie_Regularized <- edx_time %>%
  dplyr::group_by(movieId) %>%
  dplyr::summarize(movie_reg= sum(rating - mu_movies)/(n()+lambda1))
user_Regularized <- edx_time %>%
  dplyr::left_join(movie_Regularized, by='movieId') %>%
  dplyr::group_by(userId) %>%
  dplyr::summarize(user_reg = sum(rating - mu_movies - movie_reg)/(n()+lambda1))
```

```r
time_Regularized <- edx_time %>%
  dplyr::left_join(movie_Regularized, by='movieId') %>%
  dplyr::left_join(user_Regularized, by='userId') %>%
  dplyr::group_by(date) %>%
  dplyr::summarize(time_reg = sum(rating - mu_movies - movie_reg - user_reg)/(n()+lambda1))
predictionRegularized<- validation_time %>%
  dplyr::left_join(movie_Regularized, by='movieId') %>%
  dplyr::left_join(user_Regularized, by='userId') %>%
  dplyr::left_join(time_Regularized, by = 'date') %>%
  dplyr::mutate(pred = mu_movies + movie_reg + user_reg + time_reg) %>% .$pred
rmse_Regularized <- RMSE(validation_time$rating,predictionRegularized)
rmse_Regularized
```

```
## [1] 0.8647958
```

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model + User Effect + Time_Released Effect + R
                                     RMSE = rmse_Regularized))

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

method	RMSE
Movie Effect Model + User Effect + Time_Released Effect	0.8653369
Movie Effect Model + User Effect + Regularization	0.8648170
Movie Effect Model + User Effect + Time_Released Effect + Regularization	0.8647958

```r
# Adding genres
genre_Regularized <- edx_time %>%
  dplyr::left_join(movie_Regularized, by='movieId') %>%
  dplyr::left_join(user_Regularized, by='userId') %>% left_join(time_Regularized, by="date") %>%
  dplyr::group_by(genres) %>% dplyr::summarize(genre_reg =
                                   sum(rating - mu_movies - movie_reg - user_reg - time_reg)/(n()+lambda
predictionRegGenres <- validation_time %>%
  dplyr::left_join(movie_Regularized, by='movieId') %>%
  dplyr::left_join(user_Regularized, by='userId') %>% dplyr::left_join(time_Regularized, by="date") %>%
  dplyr::left_join(genre_Regularized, by = 'genres') %>%
  dplyr::mutate(pred = mu_movies + movie_reg + user_reg + time_reg+ genre_reg) %>% .$pred
rmse_RegGenres <- RMSE(validation_time$rating,predictionRegGenres)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model + User Effect + Time_Released Effect + G
                                     RMSE = rmse_RegGenres))

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

method	RMSE
Movie Effect Model + User Effect + Time_Released Effect	0.8653369
Movie Effect Model + User Effect + Regularization	0.8648170
Movie Effect Model + User Effect + Time_Released Effect + Regularization	0.8647958
Movie Effect Model + User Effect + Time_Released Effect + Genres+ Regularization	0.8644106

```r
#  Zhu,2019 as a guide to build the final table results.
# Color picker in Google give the correct HEX color number

 # Settinp up all the results

kable(rmse_results) %>%
  kable_styling("striped" , full_width = F) %>%
  column_spec(2,bold = T,border_right = F,border_left = T ) %>%
  row_spec(1:3,bold =T ,color = "black" , background ="white") %>%
  kable_styling(latex_options = c("striped","scale_down")) %>%
  row_spec(4,bold =T , font_size = 19,monospace = T, italic=T ,color = "white"
           , background ="#eb3434")
```

method	RMSE
Movie Effect Model + User Effect + Time_Released Effect	**0.8653369**
Movie Effect Model + User Effect + Regularization	**0.8648170**
Movie Effect Model + User Effect + Time_Released Effect + Regularization	**0.8647958**
Movie Effect Model + User Effect + Time_Released Effect + Genres+ Regularization	***0.8644106***

Results

The model has been successful to achieve a recommendation system with an RMSE 0.8644. The system can be improved using other more complex methods as SVD. In our case, this model can be improved using the titles as predictors and genres will require to be categorized in bucket groups to reduce multiple duplicates as drama/ love and dram/ sex. This categorization will allow the model running faster and more accurately because it will be fewer observations to compare. A model requires to create a data visualization to ensure the data has a distribution and verified that a hypothesis has a solid foundation. –

References

Cran-R_project.2017.Package 'corrplottt'.Retrieved from https://cran.r-project.org/web/packages/corrplot/corrplot.pdf

Irizarry R. 2019 para 28. Recommeddation System.Retrieved from https://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+2T2019/courseware/a49844e4a3574c239f354654f9679888/7e7727ce543b4ed6ae6338626862eada/1?activate_block_id=block-v1%3AHarvardX%2BPH125.8x%2B2T2019%2Btype%40vertical%2Bblock%40df3d8a86b43f4247a4dd42bcabb1a663

Irizarry R. 2019 code RMSE.Recommeddation System. Retrived from https://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+2T2019/courseware/a49844e4a3574c239f354654f9679888/7e7727ce543b4ed6ae6338626862eada/1?activate_block_id=block-v1%3AHarvardX%2BPH125.8x%2B2T2019%2Btype%40vertical%2Bblock%40df3d8a86b43f4247a4dd42bcabb1a663

Irizarry R. 2019 code. Building a Recommeddation System. Retrived fromhttps://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+2T2019/courseware/a49844e4a3574c239f354654f9679888/7e7727ce543b4ed6ae6338626862eada/?child

hIrizarry R. 2019 Lambdas. codettps://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+2T2019/courseware/a49844e4

Zhu,H. 2019.Create Awesome HTML Table with knitr::kable and kableExtra. Retrieved from https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html