FAMU-FSU College of Engineering

Engineering Data Analysis

ESI 5243

Term Project Report

on

IMDB 5000 Movie Dataset

Instructor

Dr. Arda Vanli

By Group 5

Bhavitha PTK

Lakshmalla David Richard Clinton

Meghana Nagarala

Rakshitha Vundela

Project Report: IMDB 5000 Movie Dataset

1 Introduction

1.1 Background

A commercial success movie not only entertains audience, but also enables film companies to gain tremendous profit. A lot of factors such as good directors, experienced actors are considerable for creating good movies. However, famous directors and actors can always bring an expected box-office income but cannot guarantee a highly rated imdb score.

1.2 Data Description

The dataset is from Kaggle website. It contains 28 variables for 5043 movies, spanning across 100 years in 66 countries. There are 2399 unique director names, and thousands of actors/actresses. "imdb_score" is the response variable while the other 27 variables are possible predictors.

The original dataset has been replaced in Kaggle, here's the link for the original dataset from Dataworld:

https://data.world/data-society/imdb-5000-movie-dataset (https://data.world/data-society/imdb-5000-movie-dataset)

Variable Name	Description
movie_title	Title of the Movie
duration	Duration in minutes
director_name	Name of the Director of the Movie
director_facebook_likes	Number of likes of the Director on his Facebook Page
actor_1_name	Primary actor starring in the movie
actor_1_facebook_likes	Number of likes of the Actor_1 on his/her Facebook Page
actor_2_name	Other actor starring in the movie
actor_2_facebook_likes	Number of likes of the Actor_2 on his/her Facebook Page
actor_3_name	Other actor starring in the movie
actor_3_facebook_likes	Number of likes of the Actor_3 on his/her Facebook Page
num_user_for_reviews	Number of users who gave a review
num_critic_for_reviews	Number of critical reviews on imdb
num_voted_users	Number of people who voted for the movie
cast_total_facebook_likes	Total number of facebook likes of the entire cast of the movie

movie_facebook_likes	Number of Facebook likes in the movie page
plot_keywords	Keywords describing the movie plot
facenumber_in_poster	Number of the actor who featured in the movie poster
color	Film colorization. ‘Black and White’ or ‘Color’
genres	Film categorization like ‘Animation’, ‘Comedy’, ‘Romance’, ‘Horror’, ‘Sci-Fi’, ‘Action’, ‘Family’
title_year	The year in which the movie is released (1916:2016)
language	English, Arabic, Chinese, French, German, Danish, Italian, Japanese etc
country	Country where the movie is produced
content_rating	Content rating of the movie
aspect_ratio	Aspect ratio the movie was made in
movie_imdb_link	IMDB link of the movie
gross	Gross earnings of the movie in Dollars
budget	Budget of the movie in Dollars
imdb_score	IMDB Score of the movie on IMDB

1.3 Problem Statement

Based on the massive movie information, it would be interesting to understand what are the important factors that make a movie more successful than others. So, we would like to analyze what kind of movies are more successful, in other words, get higher IMDB score. We also want to show the results of this analysis in an intuitive way by visualizing outcome using ggplot2 in R.

In this project, we take IMDB scores as response variable and focus on operating predictions by analyzing the rest of variables in the IMDB 5000 movie data. The results can help film companies to understand the secret of generating a commercial success movie.

2 Data Exploration

2.1 Load Data

```
# Load packages
library(ggplot2) # visualization
library(ggrepel)
library(ggthemes) # visualization
library(scales) # visualization
library(dplyr) # data manipulation
library(VIM)
library(data.table)
library(formattable)
library(plotly)
library(corrplot)
library(GGally)
library(caret)
library(car)
```

Now that our packages are loaded, let’s read in and take a peek at the data.

```
IMDB <- read.csv("movie_metadata.csv")
str(IMDB)
```

```
## 'data.frame':    5043 obs. of  28 variables:
##  $ color                    : Factor w/ 3 levels ""," Black and White",..: 3 3
3 3 1 3 3 3 3 3 ...
##  $ director_name            : Factor w/ 2399 levels "","A. Raven Cruz",..: 927
801 2027 377 603 106 2030 1652 1228 551 ...
##  $ num_critic_for_reviews   : int  723 302 602 813 NA 462 392 324 635 375 ...
##  $ duration                 : int  178 169 148 164 NA 132 156 100 141 153 ...
##  $ director_facebook_likes  : int  0 563 0 22000 131 475 0 15 0 282 ...
##  $ actor_3_facebook_likes   : int  855 1000 161 23000 NA 530 4000 284 19000 10
000 ...
##  $ actor_2_name             : Factor w/ 3033 levels "","50 Cent","A. Michael B
aldwin",..: 1407 2218 2488 534 2432 2549 1227 801 2439 653 ...
##  $ actor_1_facebook_likes   : int  1000 40000 11000 27000 131 640 24000 799 26
000 25000 ...
##  $ gross                    : int  760505847 309404152 200074175 448130642 NA
73058679 336530303 200807262 458991599 301956980 ...
##  $ genres                   : Factor w/ 914 levels "Action","Action|Adventure"
,..: 107 101 128 288 754 126 120 308 126 447 ...
##  $ actor_1_name             : Factor w/ 2098 levels "","50 Cent","A.J. Buckley
",..: 302 979 353 1968 526 440 785 221 336 32 ...
##  $ movie_title              : Factor w/ 4917 levels "[Rec] ","[Rec] 2 ",..: 39
```

```
8 2731 3279 3708 3332 1961 3291 3459 399 1631 ...
##  $ num_voted_users          : int  886204 471220 275868 1144337 8 212204 38305
6 294810 462669 321795 ...
##  $ cast_total_facebook_likes: int  4834 48350 11700 106759 143 1873 46055 2036
92000 58753 ...
##  $ actor_3_name             : Factor w/ 3522 levels "","50 Cent","A.J. Buckley
",..: 3442 1392 3134 1769 1 2714 1969 2162 3018 2941 ...
##  $ facenumber_in_poster     : int  0 0 1 0 0 1 0 1 4 3 ...
##  $ plot_keywords            : Factor w/ 4761 levels "","10 year old|dog|florid
a|girl|supermarket",..: 1320 4283 2076 3484 1 651 4745 29 1142 2005 ...
##  $ movie_imdb_link          : Factor w/ 4919 levels "http://www.imdb.com/title
/tt0006864/?ref_=fn_tt_tt_1",..: 2965 2721 4533 3756 4918 2476 2526 2458 4546 255
1 ...
##  $ num_user_for_reviews     : int  3054 1238 994 2701 NA 738 1902 387 1117 973
...
##  $ language                 : Factor w/ 48 levels "","Aboriginal",..: 13 13 13
13 1 13 13 13 13 13 ...
##  $ country                  : Factor w/ 66 levels "","Afghanistan",..: 65 65 6
3 65 1 65 65 65 65 63 ...
##  $ content_rating           : Factor w/ 19 levels "","Approved",..: 10 10 10 1
0 1 10 10 9 10 9 ...
##  $ budget                   : num 2.37e+08 3.00e+08 2.45e+08 2.50e+08 NA ...
##  $ title_year               : int 2009 2007 2015 2012 NA 2012 2007 2010 2015
2009 ...
##  $ actor_2_facebook_likes   : int  936 5000 393 23000 12 632 11000 553 21000 1
1000 ...
##  $ imdb_score               : num  7.9 7.1 6.8 8.5 7.1 6.6 6.2 7.8 7.5 7.5 ...
##  $ aspect_ratio             : num  1.78 2.35 2.35 2.35 NA 2.35 2.35 1.85 2.35
2.35 ...
##  $ movie_facebook_likes     : int  33000 0 85000 164000 0 24000 0 29000 118000
10000 ...
```

We have 5043 observations of 28 variables. The response variable “imdb_score” is numerical, and the predictors are mixed with numerical and categorical variables.

2.2 Remove Duplicates

In the IMDB data, we have some duplicate rows. We want to remove the 45 duplicated rows and keep the unique ones.

```
# duplicate rows
sum(duplicated(IMDB))
```

```
## [1] 45
```

```
# delete duplicate rows
IMDB <- IMDB[!duplicated(IMDB), ]
```

We get 4998 observations left.

2.3 Tidy Up Movie Title

All the movie titles have a special character (Â) at the end and some have whitespaces, they might be generated during the data collection. Let's remove them.

```
library(stringr)
IMDB$movie_title <- gsub("Â", "", as.character(factor(IMDB$movie_title)))
str_trim(IMDB$movie_title, side = "right")
```

2.4 Split Genres

Each record of genres is combined with a few types, which will cause the difficulty of analyzing.

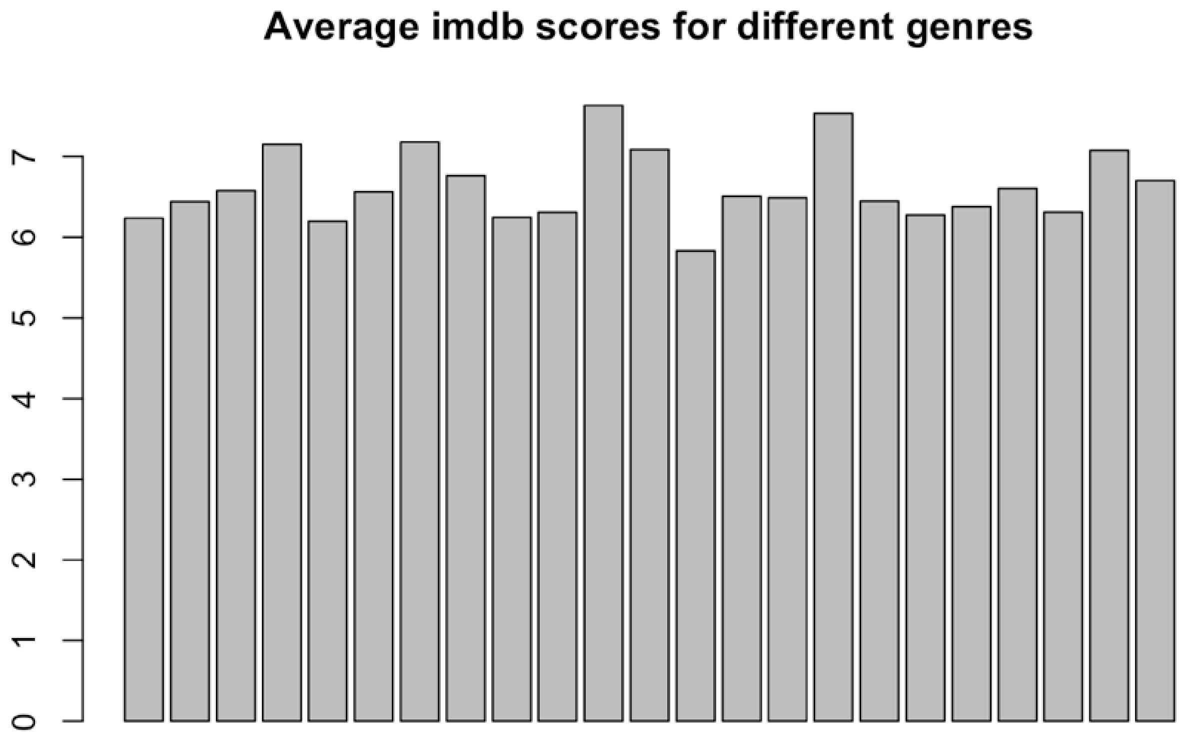
```
head(IMDB$genres)
```

```
## [1] Action|Adventure|Fantasy|Sci-Fi Action|Adventure|Fantasy
## [3] Action|Adventure|Thriller       Action|Thriller
## [5] Documentary                     Action|Adventure|Sci-Fi
## 914 Levels: Action ... Western
```

First, we want to know if genre is related to imdb score. We divide the string into several substrings by the separator '|', and save each substring along with its correspongding imdb score in the other data frame **genres.df**. Then we plot a histogram for the score and genres to see if they are relative or not.

```
# create a new data frame
genres.df <- as.data.frame(IMDB[,c("genres", "imdb_score")])
# separate different genres into new columns
genres.df$Action <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Action") 1 else 0)
genres.df$Adventure <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Adventure") 1 else 0)
genres.df$Animation <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Animation") 1 else 0)
genres.df$Biography <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Biography") 1 else 0)
genres.df$Comedy <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Comedy") 1 else 0)
genres.df$Crime <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Crime") 1 else 0)
```

```r
genres.df$Documentary <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Documentary") 1 else 0)
genres.df$Drama <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Drama") 1 else 0)
genres.df$Family <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Family") 1 else 0)
genres.df$Fantasy <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Fantasy") 1 else 0)
genres.df$`Film-Noir` <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Film-Noir") 1 else 0)
genres.df$History <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "History") 1 else 0)
genres.df$Horror <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Horror") 1 else 0)
genres.df$Musical <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Musical") 1 else 0)
genres.df$Mystery <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Mystery") 1 else 0)
genres.df$News <- sapply(1:length(genres.df$genres), function(x) if (genres.df[x,
1] %like% "News") 1 else 0)
genres.df$Romance <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Romance") 1 else 0)
genres.df$`Sci-Fi` <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Sci-Fi") 1 else 0)
genres.df$Short <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Short") 1 else 0)
genres.df$Sport <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Sport") 1 else 0)
genres.df$Thriller <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Thriller") 1 else 0)
genres.df$War <- sapply(1:length(genres.df$genres), function(x) if (genres.df[x,1
] %like% "War") 1 else 0)
genres.df$Western <- sapply(1:length(genres.df$genres), function(x) if
(genres.df[x,1] %like% "Western") 1 else 0)
# get the mean of imdb score for different genres
means <- rep(0,23)
for (i in 1:23) {
  means[i] <- mean(genres.df$imdb_score[genres.df[i+2]==1])
}
# plot the means
barplot(means, main = "Average imdb scores for different genres")
```



There isn’t much difference in the averages of imdb score related to different genres, almost all the averages are in the same range of 6~8. So we think the predictor “genres” can be removed because it’s not really related to the score.

```
IMDB <- subset(IMDB, select = -c(genres))
```

3 Data Cleaning

3.1 Missing Values

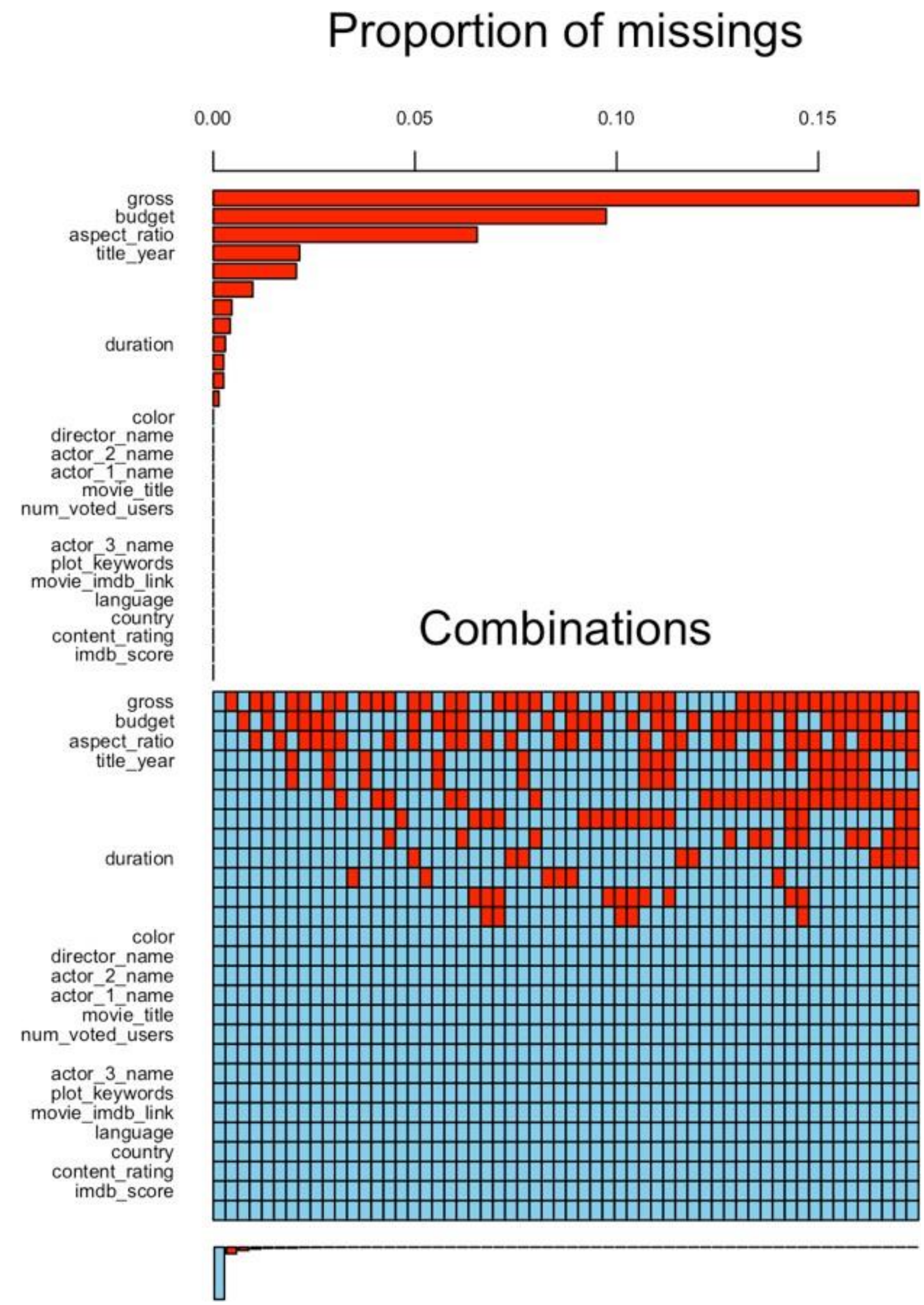
To find missing values in each column, we use *colSums()* function to aggregate NA in each column.

```
colSums(sapply(IMDB, is.na))
```

```
##                     color             director_name
##                         0                         0
##    num_critic_for_reviews                  duration
##                        49                        15
##   director_facebook_likes    actor_3_facebook_likes
##                       103                        23
##              actor_2_name    actor_1_facebook_likes
##                         0                         7
##                     gross              actor_1_name
##                       874                         0
##               movie_title           num_voted_users
##                         0                         0
## cast_total_facebook_likes              actor_3_name
##                         0                         0
##      facenumber_in_poster             plot_keywords
##                        13                         0
##           movie_imdb_link      num_user_for_reviews
##                         0                        21
##                  language                   country
##                         0                         0
##            content_rating                    budget
##                         0                       487
##                title_year    actor_2_facebook_likes
##                       107                        13
##                imdb_score              aspect_ratio
##                         0                       327
##      movie_facebook_likes
##                         0
```

Let’s use heatmap to visualize missing values.

```
missing.values <- aggr(IMDB, sortVars = T, prop = T, sortCombs = T, cex.lab = 1.5
, cex.axis = .6, cex.numbers = 5, combined = F, gap = -.2)
```



Proportion of missings
0.00
0.05
0.10
0.15
gross
budget
aspect_ratio
title_year
duration
color
director_name
actor_2_name
actor_1_name
movie_title
num_voted_users
actor_3_name
plot_keywords
movie_imdb_link
language
country
content_rating
imdb_score
Combinations
gross
budget
aspect_ratio
title_year
duration
color
director_name
actor_2_name
actor_1_name
movie_title
num_voted_users
actor_3_name
plot_keywords
movie_imdb_link
language
country
content_rating
imdb_score

```
##
## Variables sorted by number of missings:
##                  Variable       Count
##                     gross 0.174869948
##                    budget 0.097438976
##              aspect_ratio 0.065426170
##                title_year 0.021408563
##   director_facebook_likes 0.020608243
##    num_critic_for_reviews 0.009803922
##    actor_3_facebook_likes 0.004601841
##      num_user_for_reviews 0.004201681
##                  duration 0.003001200
##      facenumber_in_poster 0.002601040
##    actor_2_facebook_likes 0.002601040
##    actor_1_facebook_likes 0.001400560
##                     color 0.000000000
##             director_name 0.000000000
##              actor_2_name 0.000000000
##              actor_1_name 0.000000000
##               movie_title 0.000000000
##           num_voted_users 0.000000000
## cast_total_facebook_likes 0.000000000
##              actor_3_name 0.000000000
##             plot_keywords 0.000000000
##           movie_imdb_link 0.000000000
##                  language 0.000000000
##                   country 0.000000000
##            content_rating 0.000000000
##                imdb_score 0.000000000
##      movie_facebook_likes 0.000000000
```

3.1.1 Delete some rows

Since gross and budget have too many missing values, and we want to keep these two variables for the following analysis, we can only delete rows with null values for gross and budget because imputation will not do a good job here.

```
IMDB <- IMDB[!is.na(IMDB$gross), ]
IMDB <- IMDB[!is.na(IMDB$budget), ]
dim(IMDB)
```

```
## [1] 3857   27
```

Not too bad, we only omitted 23% of the observations. Now our data has 3857 observations.

Let’s see how many complete cases we have.

```
sum(complete.cases(IMDB))
```

```
## [1] 3768
```

So, there are still 3857 - 3768 = 89 rows with NAs.

3.1.2 Analyze aspect ratio

let's take a look at rest columns with missing values.

```
colSums(sapply(IMDB, is.na))
```

```
##                     color             director_name
##                         0                         0
##    num_critic_for_reviews                  duration
##                         1                         1
##   director_facebook_likes    actor_3_facebook_likes
##                         0                        10
##              actor_2_name    actor_1_facebook_likes
##                         0                         3
##                     gross              actor_1_name
##                         0                         0
##               movie_title           num_voted_users
##                         0                         0
## cast_total_facebook_likes              actor_3_name
##                         0                         0
##      facenumber_in_poster             plot_keywords
##                         6                         0
##           movie_imdb_link      num_user_for_reviews
##                         0                         0
##                  language                   country
##                         0                         0
##            content_rating                    budget
##                         0                         0
##                title_year    actor_2_facebook_likes
##                         0                         5
##                imdb_score              aspect_ratio
##                         0                        74
##      movie_facebook_likes
##                         0
```

Now aspect_ratio has the highest number of missing values. Before trying to impute the missing values, we want to check how important is this variable.

```
table(IMDB$aspect_ratio)
```

```
##
## 1.18 1.33 1.37  1.5 1.66 1.75 1.77 1.78 1.85    2  2.2 2.24 2.35 2.39  2.4
##    1   19   50    1   40    2    1   41 1600    3   10    1 1995   11    3
## 2.55 2.76   16
##    1    3    1
```

The most common aspect ratios are 1.85 and 2.35. For analyzing purpose, we group other ratios together.

In order to compute the mean of imdb score for different aspect_ratio, we need to replace NA with 0 first.

```
IMDB$aspect_ratio[is.na(IMDB$aspect_ratio)] <- 0
mean(IMDB$imdb_score[IMDB$aspect_ratio == 1.85])
```

```
## [1] 6.373938
```

```
mean(IMDB$imdb_score[IMDB$aspect_ratio == 2.35])
```

```
## [1] 6.508471
```

```
mean(IMDB$imdb_score[IMDB$aspect_ratio != 1.85 & IMDB$aspect_ratio != 2.35])
```

```
## [1] 6.672519
```

From the means of imdb score for different aspect ratios, we can see there is no significant difference, all the means fall in the range of 6.3~6.8. So, removing this variable won't affect our following analysis.

```
IMDB <- subset(IMDB, select = -c(aspect_ratio))
```

3.1.3 Deal with 0s

We notice that there are some 0 values which should also be regarded as missing value except for predictor facenumber_in_poster.

First we need to replace NA with column average for facenumber_in_poster, then replace 0s in other predictors with NA, and lastly replace all NAs with their respective column mean.

```
# replace NA with column average for facenumber_in_poster
IMDB$facenumber_in_poster[is.na(IMDB$facenumber_in_poster)] <- round(mean(IMDB$fa
cenumber_in_poster, na.rm = TRUE))
# convert 0s into NAs for other predictors
IMDB[,c(5,6,8,13,24,26)][IMDB[,c(5,6,8,13,24,26)] == 0] <- NA
# impute missing value with column mean
IMDB$num_critic_for_reviews[is.na(IMDB$num_critic_for_reviews)] <- round(mean(IMD
B$num_critic_for_reviews, na.rm = TRUE))
IMDB$duration[is.na(IMDB$duration)] <- round(mean(IMDB$duration, na.rm = TRUE))
IMDB$director_facebook_likes[is.na(IMDB$director_facebook_likes)] <- round(mean(I
MDB$director_facebook_likes, na.rm = TRUE))
IMDB$actor_3_facebook_likes[is.na(IMDB$actor_3_facebook_likes)] <- round(mean(IMD
B$actor_3_facebook_likes, na.rm = TRUE))
IMDB$actor_1_facebook_likes[is.na(IMDB$actor_1_facebook_likes)] <- round(mean(IMD
B$actor_1_facebook_likes, na.rm = TRUE))
IMDB$cast_total_facebook_likes[is.na(IMDB$cast_total_facebook_likes)] <- round(me
an(IMDB$cast_total_facebook_likes, na.rm = TRUE))
IMDB$actor_2_facebook_likes[is.na(IMDB$actor_2_facebook_likes)] <- round(mean(IMD
B$actor_2_facebook_likes, na.rm = TRUE))
IMDB$movie_facebook_likes[is.na(IMDB$movie_facebook_likes)] <- round(mean(IMDB$mo
vie_facebook_likes, na.rm = TRUE))
```

Now we finished imputing the numeric missing values. There are still some categorical missing values, let's take a look.

3.1.4 Sort out content ratings

We find there are still some missing values in content_rating, which are marked as “”.

```
table(IMDB$content_rating)
```

```
## 
##              Approved         G        GP         M     NC-17 Not Rated 
##        51        17        91         1         2         6        42 
##    Passed        PG     PG-13         R     TV-14      TV-G     TV-MA 
##         3       573      1314      1723         0         0         0 
##     TV-PG      TV-Y     TV-Y7   Unrated         X 
##         0         0         0        24        10
```

Blanks should be taken as missing value. Since these missing values cannot be replaced with reasonable data, we delete these rows.

```
IMDB <- IMDB[!(IMDB$content_rating %in% ""),]
```

According to the history of naming these different content ratings, we find M = GP = PG, X = NC-17. We want to replace M and GP with PG, replace X with NC-17, because these two are what we use nowadays.

```
IMDB$content_rating[IMDB$content_rating == 'M']    <- 'PG'
IMDB$content_rating[IMDB$content_rating == 'GP'] <- 'PG'
IMDB$content_rating[IMDB$content_rating == 'X']    <- 'NC-17'
```

We want to replace “Approved”, “Not Rated”, “Passed”, “Unrated” with the most common rating “R”.

```
IMDB$content_rating[IMDB$content_rating == 'Approved'] <- 'R'
IMDB$content_rating[IMDB$content_rating == 'Not Rated'] <- 'R'
IMDB$content_rating[IMDB$content_rating == 'Passed']     <- 'R'
IMDB$content_rating[IMDB$content_rating == 'Unrated']    <- 'R'
IMDB$content_rating <- factor(IMDB$content_rating)
table(IMDB$content_rating)
```

```
##
##     G NC-17    PG PG-13     R
##    91    16   576  1314  1809
```

Now we only have 5 different content ratings.

3.2 Add Columns

We have gross and budget information. So let’s add two colums: profit and percentage return on investment for further analysis.

```
IMDB <- IMDB %>%
  mutate(profit = gross - budget,
         return_on_investment_perc =  (profit/budget)*100)
```

3.3 Remove Columns

3.3.1 Is the color of a movie influential?

```
table(IMDB$color)
```

```
##
##                    Black and White             Color
##                  2             124              3680
```

More than 96% movies are colored, which indicates that this predictor is nearly constant. Let's remove this predictor.

```
# delete predictor color
IMDB <- subset(IMDB, select = -c(color))
```

3.3.2 Is language an important factor for imdb score? What about country?

```
table(IMDB$language)
```

```
##
##             Aboriginal     Arabic    Aramaic    Bosnian  Cantonese
##          2           2          1          1          1          7
##    Chinese       Czech     Danish       Dari      Dutch   Dzongkha
##          0           1          3          2          3          0
##    English    Filipino     French     German      Greek     Hebrew
##       3644           1         34         11          0          2
##      Hindi   Hungarian  Icelandic Indonesian    Italian   Japanese
##          5           1          0          2          7         10
##    Kannada      Kazakh     Korean   Mandarin       Maya  Mongolian
##          0           1          5         14          1          1
##       None   Norwegian    Panjabi    Persian     Polish Portuguese
##          1           4          0          3          0          5
##   Romanian     Russian  Slovenian    Spanish    Swahili    Swedish
##          1           1          0         24          0          0
##      Tamil      Telugu       Thai       Urdu Vietnamese       Zulu
##          0           0          3          0          1          1
```

Over 95% movies are in English, which means this variable is nearly constant. Let's remove it.

```
IMDB <- subset(IMDB, select = -c(language))
```

Let's take a look at predictor country.

```
table(IMDB$country)
```

```
## 
##                                     Afghanistan            Argentina 
##                     0                         1                    3 
##                 Aruba                 Australia              Bahamas 
##                     1                        40                    0 
##               Belgium                    Brazil             Bulgaria 
##                     1                         5                    0 
##              Cambodia                  Cameroon               Canada 
##                     0                         0                   63 
##                 Chile                     China             Colombia 
##                     1                        13                    1 
##        Czech Republic                   Denmark   Dominican Republic 
##                     3                         9                    0 
##                 Egypt                   Finland               France 
##                     0                         1                  103 
##               Georgia                   Germany               Greece 
##                     1                        79                    1 
##             Hong Kong                   Hungary              Iceland 
##                    13                         2                    1 
##                 India                 Indonesia                 Iran 
##                     5                         1                    4 
##               Ireland                    Israel                Italy 
##                     7                         2                   11 
##                 Japan                     Kenya           Kyrgyzstan 
##                    15                         0                    0 
##                 Libya                    Mexico          Netherlands 
##                     0                        10                    3 
##              New Line               New Zealand              Nigeria 
##                     1                        11                    0 
##                Norway             Official site             Pakistan 
##                     4                         1                    0 
##                Panama                      Peru          Philippines 
##                     0                         1                    1 
##                Poland                   Romania               Russia 
##                     1                         2                    3 
##              Slovakia                  Slovenia         South Africa 
##                     0                         0                    3 
##           South Korea             Soviet Union                Spain 
##                     8                         0                   22 
##                Sweden               Switzerland               Taiwan 
##                     0                         0                    2 
##              Thailand                    Turkey                   UK 
##                     4                         0                  316 
## United Arab Emirates                       USA         West Germany 
##                     0                      3025                    1
```

Around 79% movies are from USA, 8% from UK, 13% from other countries. So we group other countries together to make this categorical variable with less levels: USA, UK, Others.

```
levels(IMDB$country) <- c(levels(IMDB$country), "Others")
IMDB$country[(IMDB$country != 'USA')&(IMDB$country != 'UK')] <- 'Others'
IMDB$country <- factor(IMDB$country)
table(IMDB$country)
```

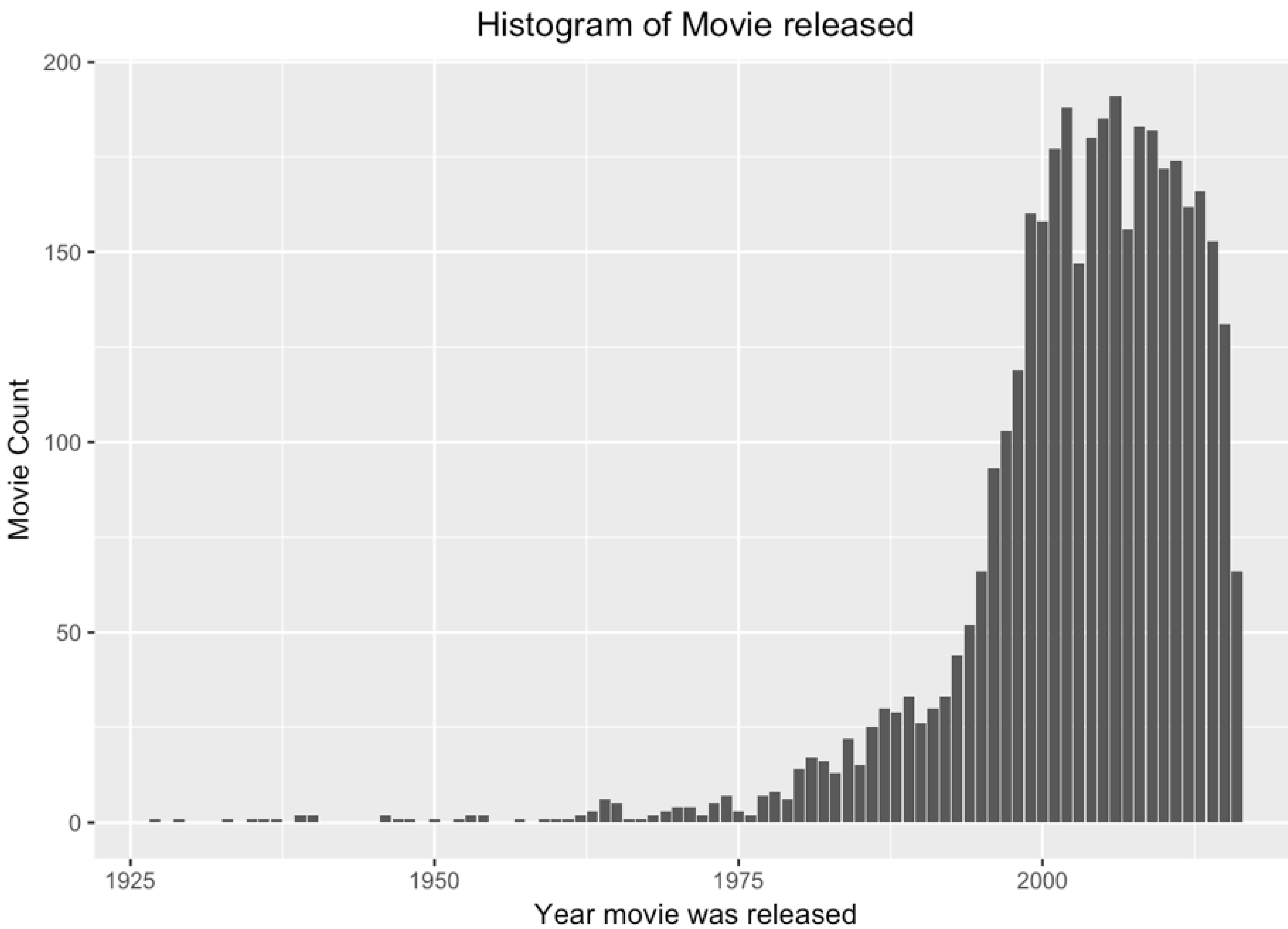
```
##
##     UK    USA Others
##    316   3025    465
```

4 Data Visualization

4.1 Histogram of Movie Released

Movie production just exploded after year 1990. It could be due to advancement in technology and commercialisation of internet.

```
ggplot(IMDB, aes(title_year)) +
  geom_bar() +
  labs(x = "Year movie was released", y = "Movie Count", title = "Histogram of Mo
vie released") +
  theme(plot.title = element_text(hjust = 0.5))
```



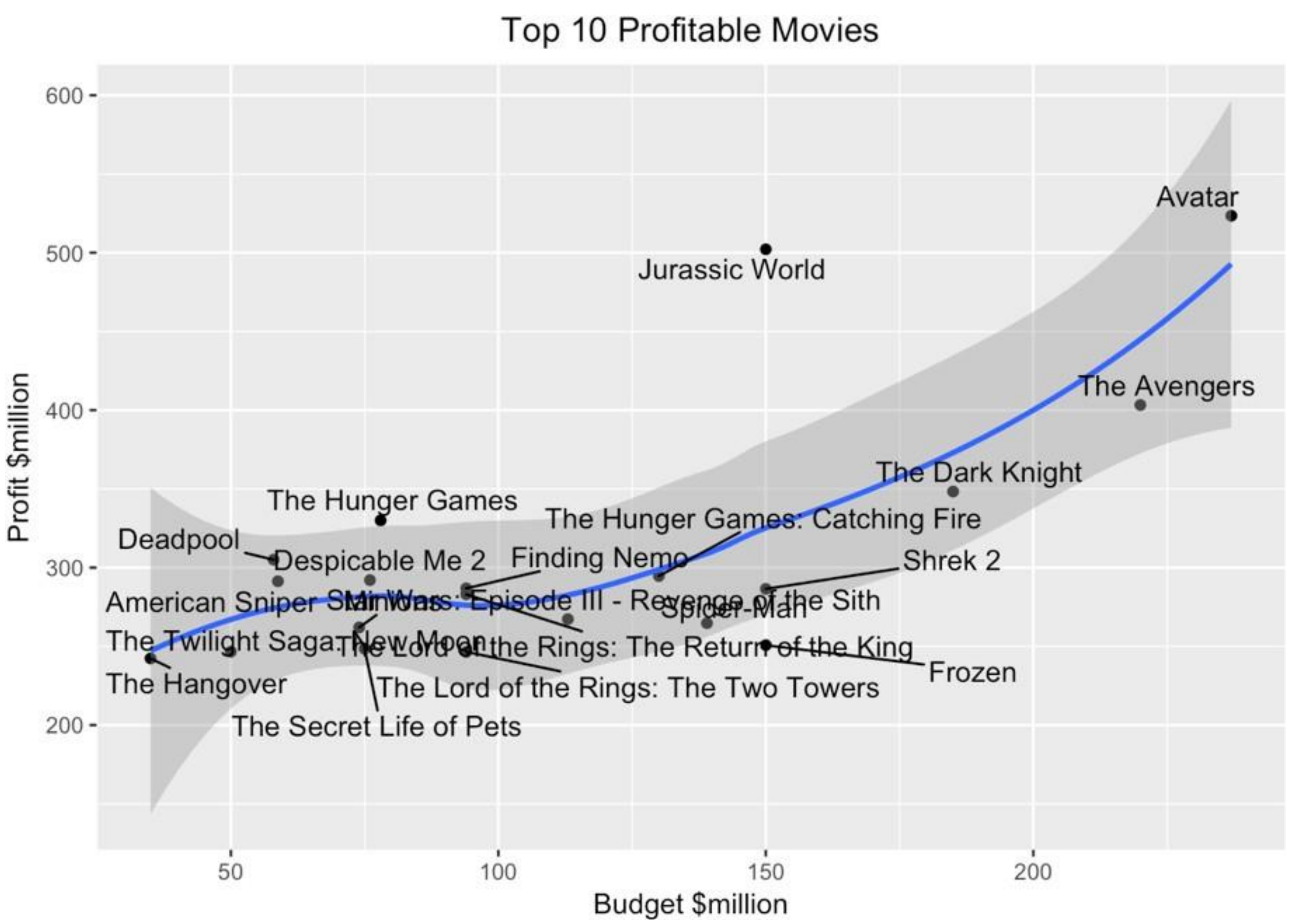
From the graph, we see there aren’t many records of movies released before 1980. It’s better to remove those records because they might not be representative.

```
IMDB <- IMDB[IMDB$title_year >= 1980,]
```

4.2 Top 20 movies based on its Profit

```
IMDB %>%
  filter(title_year %in% c(2000:2016)) %>%
  arrange(desc(profit)) %>%
  top_n(20,  profit)  %>%
  ggplot(aes(x=budget/1000000, y=profit/1000000)) +
  geom_point() +
  geom_smooth() +
  geom_text_repel(aes(label=movie_title)) +
  labs(x = "Budget $million", y = "Profit $million", title = "Top 10 Profitable M
ovies") +
  theme(plot.title = element_text(hjust = 0.5))
```

```
## `geom_smooth()` using method = 'loess'
```

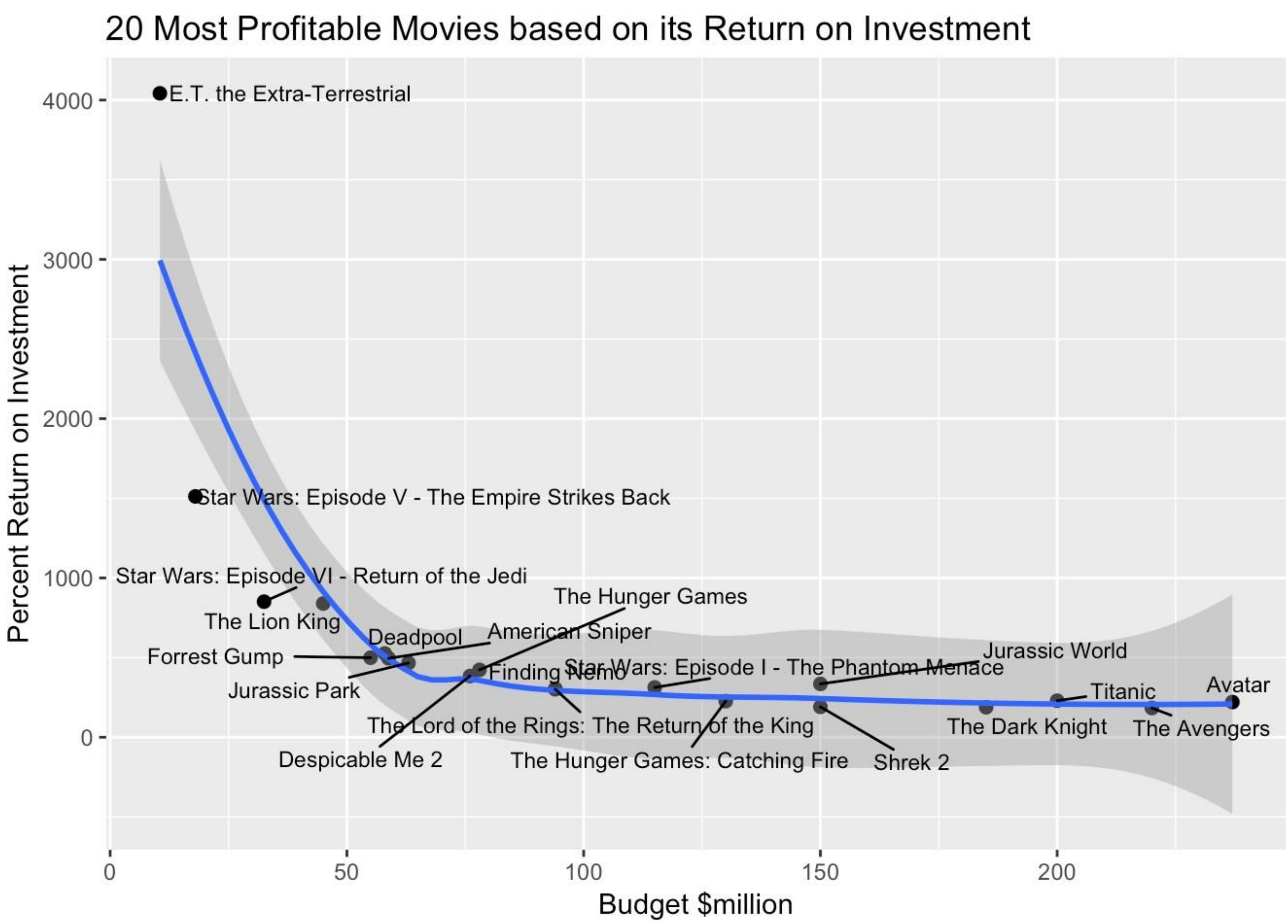


These are the top 20 movies based on the Profit earned (Gross - Budget). It can be inferred from this plot that high budget movies tend to earn more profit. The trend is almost linear, with profit increasing with the increase in budget.

4.3 Top 20 movies based on its Return on Investment

```
IMDB %>%
  filter(budget > 100000) %>%
  mutate(profit = gross - budget,
         return_on_investment_perc = (profit/budget)*100) %>%
  arrange(desc(profit)) %>%
  top_n(20, profit) %>%
  ggplot(aes(x=budget/1000000, y = return_on_investment_perc)) +
  geom_point(size = 2) +
  geom_smooth(size = 1) +
  geom_text_repel(aes(label = movie_title), size = 3) +
  xlab("Budget $million") +
  ylab("Percent Return on Investment") +
  ggtitle("20 Most Profitable Movies based on its Return on Investment")
```

```
## `geom_smooth()` using method = 'loess'
```



These are the top 20 movies based on its Percentage Return on Investment ((profit/budget)*100).

Since profit earned by a movie does not give a clear picture about its monetary success over the years, this analysis, over the absolute value of the Return on Investment(ROI) across its Budget, would provide better results.

As hypothesized, the ROI is high for Low Budget Films and decreases as the budget of the movie increases.

4.4 Top 20 directors with highest average IMDB score

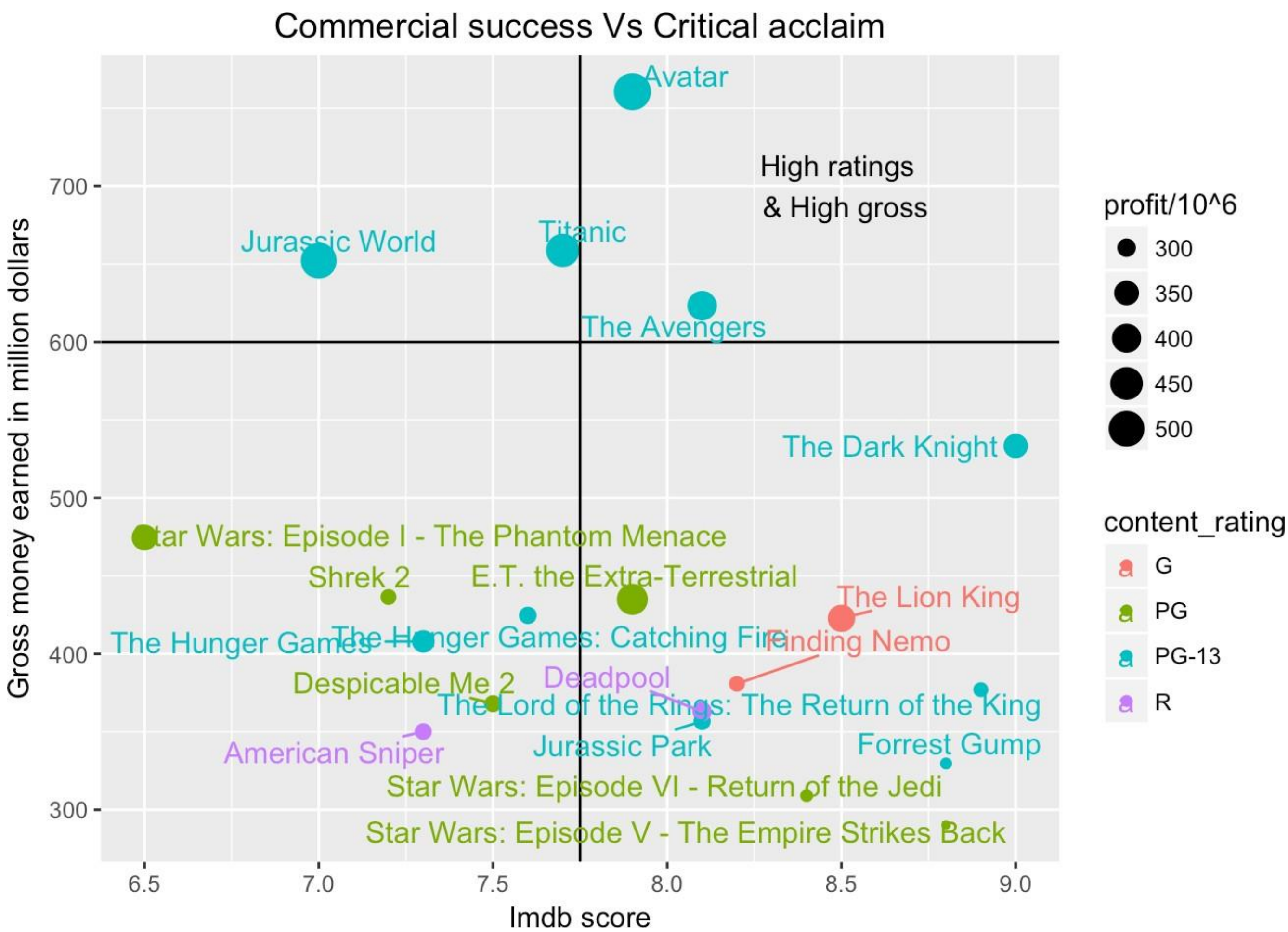
```
IMDB %>%
  group_by(director_name) %>%
  summarise(avg_imdb = mean(imdb_score)) %>%
  arrange(desc(avg_imdb)) %>%
  top_n(20, avg_imdb) %>%
  formattable(list(avg_imdb = color_bar("orange")), align = 'l')
```

director_name	avg_imdb
Tony Kaye	8.600000
Damien Chazelle	8.500000
Majid Majidi	8.500000
Ron Fricke	8.500000
Christopher Nolan	8.425000
Asghar Farhadi	8.400000
Marius A. Markevicius	8.400000
Richard Marquand	8.400000
Sergio Leone	8.400000
Lee Unkrich	8.300000
Lenny Abrahamson	8.300000
Pete Docter	8.233333
Hayao Miyazaki	8.225000
Joshua Oppenheimer	8.200000
Juan José Campanella	8.200000
Quentin Tarantino	8.200000
David Sington	8.100000

Je-kyu Kang	8.100000
Terry George	8.100000
Tim Miller	8.100000

4.5 Commercial Success v.s. Critical Acclaim

```
IMDB %>%
  top_n(20, profit) %>%
  ggplot(aes(x = imdb_score, y = gross/10^6, size = profit/10^6, color = content_
rating)) +
  geom_point() +
  geom_hline(aes(yintercept = 600)) +
  geom_vline(aes(xintercept = 7.75)) +
  geom_text_repel(aes(label = movie_title), size = 4) +
  xlab("Imdb score") +
  ylab("Gross money earned in million dollars") +
  ggtitle("Commercial success Vs Critical acclaim") +
  annotate("text", x = 8.5, y = 700, label = "High ratings \n & High gross") +
  theme(plot.title = element_text(hjust = 0.5))
```

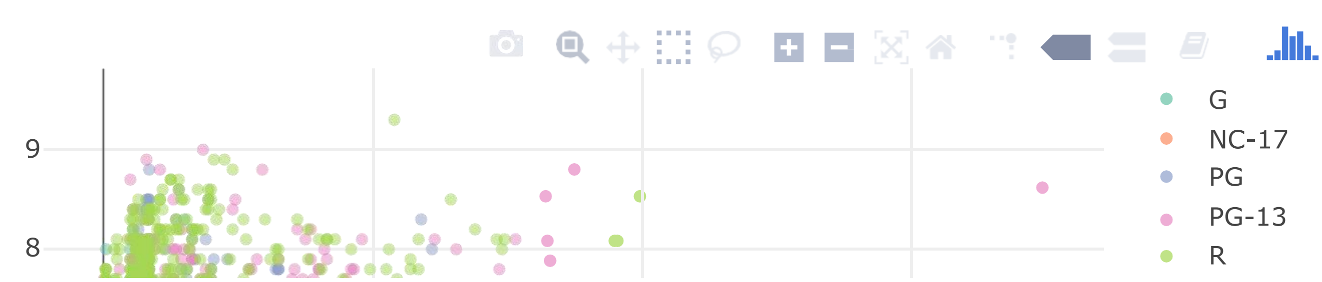


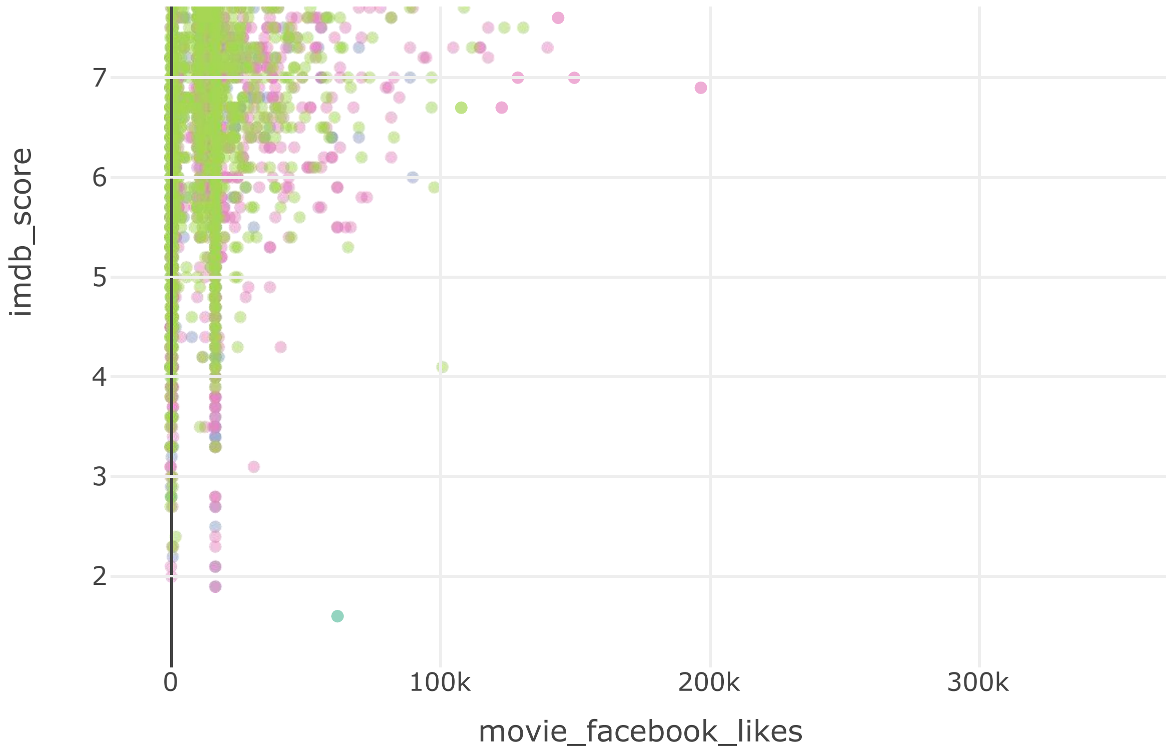
This is an analysis on the Commercial Success acclaimed by the movie (Gross earnings and profit earned) v.s. its IMDB Score.

As expected, there is not much correlation since most critically acclaimed movies do not do much well commercially.

4.6 Relation between number of facebook likes and imdb_score

```
IMDB %>%
  plot_ly(x = ~movie_facebook_likes, y = ~imdb_score, color = ~content_rating , m
ode = "markers", text = ~content_rating, alpha = 0.7, type = "scatter")
```





We divide this scatter plot by content-rating. Movie with extremely high Facebook likes tend to have higher imdb score. But the score for movie with low Facebook likes vary in a very wide range.

5 Data Pre-processing

5.1 Remove Names

We have 1660 directors, and 3621 actors in this data.

```
# number of directors
sum(uniqueN(IMDB$director_name))
```

```
## [1] 1660
```

```
# number of actors
sum(uniqueN(IMDB[, c("actor_1_name", "actor_2_name", "actor_3_name")]))
```

```
## [1] 3621
```

Since all the names are so different for the whole dataset, there is no point to use names to predict score.

Same with plot keywords, they are too diverse to be used in the prediction.

And movie link is also a redundant variable.

```
IMDB <- subset(IMDB, select = -c(director_name, actor_2_name, actor_1_name,
                                 movie_title, actor_3_name, plot_keywords,
                                 movie_imdb_link))
```

5.2 Remove Linear Dependent Variables

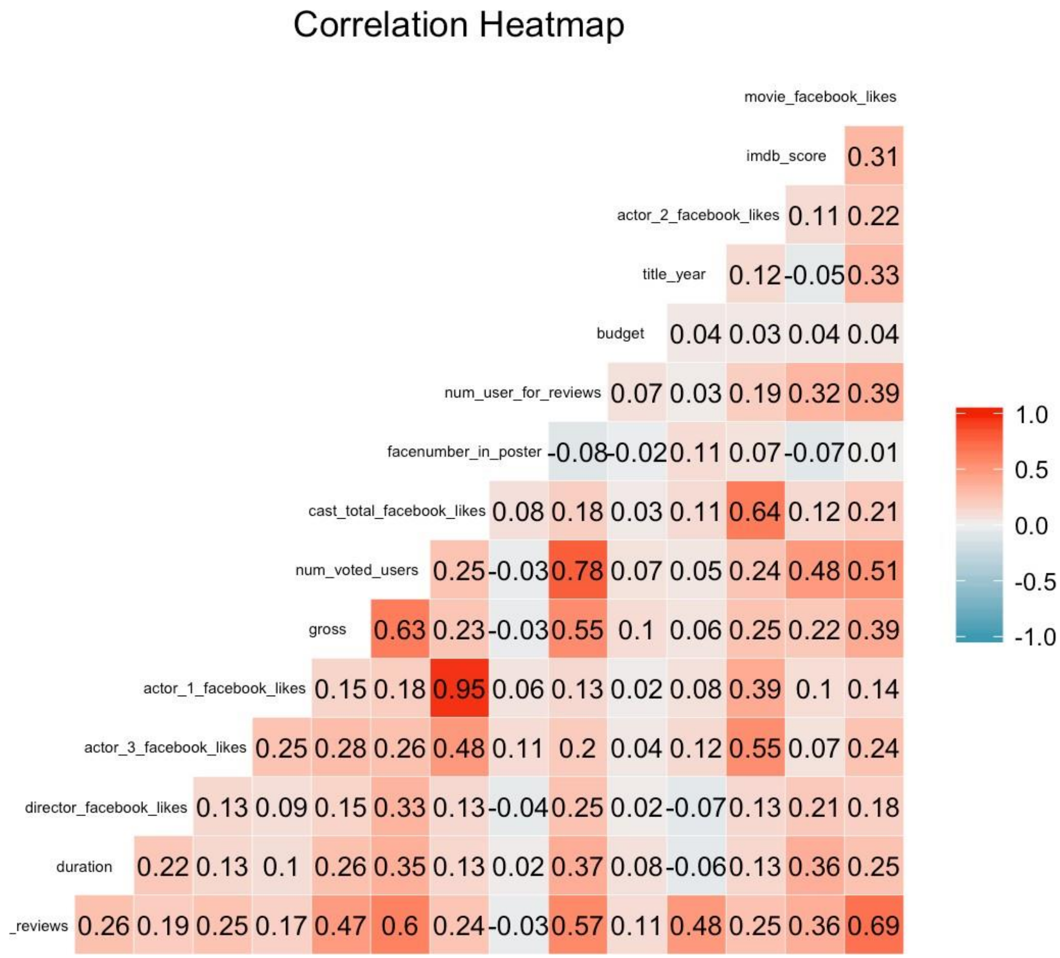
For the purpose of data exploration, we added two variables based on existing variables: profit and return_on_investment_perc. In order to avoid multicollinearity, here we remove these two added variables.

```
IMDB <- subset(IMDB, select = -c(profit, return_on_investment_perc))
```

5.3 Remove Highly Correlated Variables

First we plot the correlation heatmap for our data.

```
ggcorr(IMDB, label = TRUE, label_round = 2, label_size = 3.5, size = 2, hjust = .
85) +
  ggtitle("Correlation Heatmap") +
  theme(plot.title = element_text(hjust = 0.5))
```



Based on the heatmap, we can see some high correlations (greater than 0.7) between predictors.

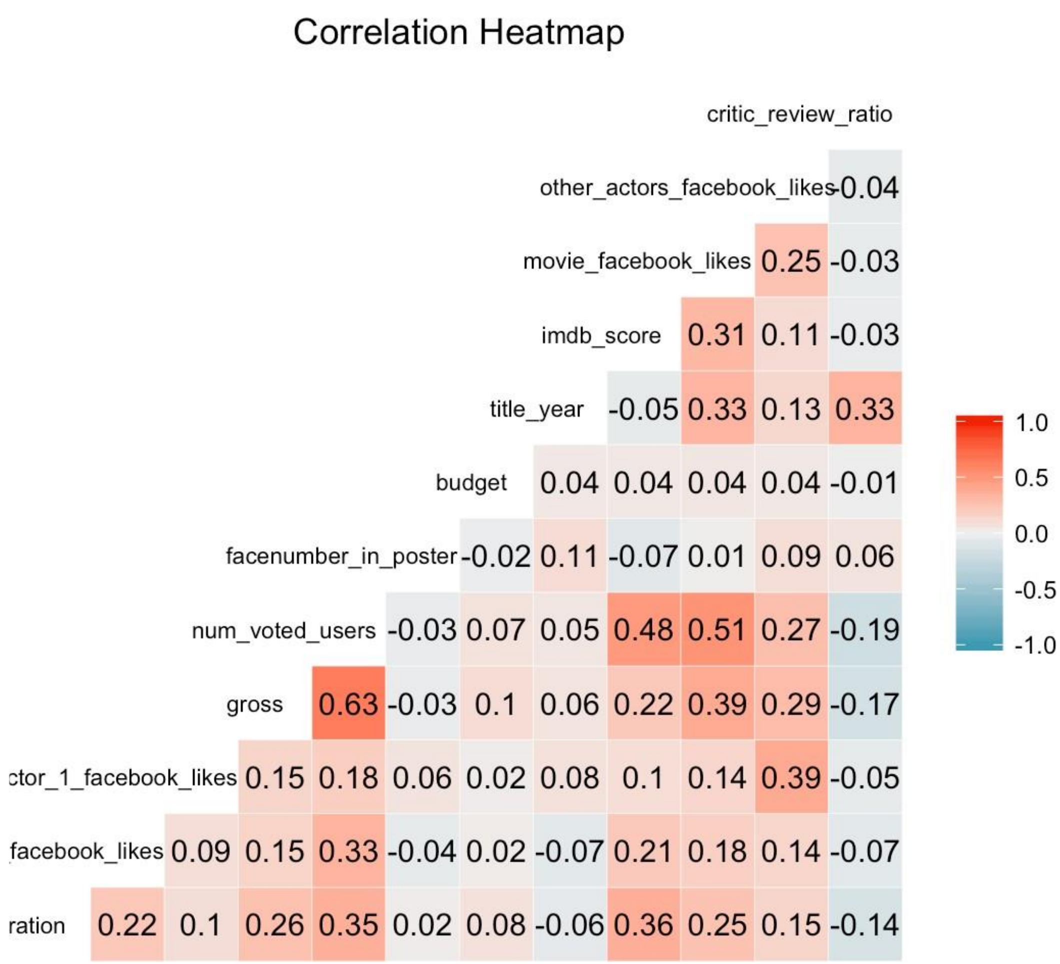
According to the highest correlation value 0.95, we find actor_1_facebook_likes is highly correlated with the cast_total_facebook_likes, and both actor2 and actor3 are also somehow correlated to the total. So we want to modify them into two variables: actor_1_facebook_likes and other_actors_facebook_likes.

There are high correlations among num_voted_users, num_user_for_reviews and num_critic_for_reviews. We want to keep num_voted_users and take the ratio of num_user_for_reviews and num_critic_for_reviews.

```
# add up actor 2 and 3 facebook likes into other actors facebook likes
IMDB$other_actors_facebook_likes <- IMDB$actor_2_facebook_likes + IMDB$actor_3_fa
cebook_likes
# use the ratio of critical reviews amount to total reviews amount
IMDB$critic_review_ratio <- IMDB$num_critic_for_reviews / IMDB$num_user_for_revie
ws
# delete columns
IMDB <- subset(IMDB, select = -c(cast_total_facebook_likes, actor_2_facebook_like
s, actor_3_facebook_likes, num_critic_for_reviews, num_user_for_reviews))
```

Now the correlation heatmap becomes like this.

```
ggcorr(IMDB, label = TRUE, label_round = 2, label_size = 4, size = 3, hjust = .85
) +
  ggtitle("Correlation Heatmap") +
  theme(plot.title = element_text(hjust = 0.5))
```



We don’t see any strong correlation (absolute value greater than 0.7) any more.

5.4 Bin Response Variable

Our goal is to build a model, which can help us predict if a movie is good or bad. So we don't really want an exact score to be predicted, we only want to know how good or how bad is the movie. Therefore, we bin the score into 4 buckets: less than 4, 4~6, 6~8 and 8~10, which represents bad, OK, good and excellent respectively.

```
IMDB$binned_score <- cut(IMDB$imdb_score, breaks = c(0,4,6,8,10))
```

5.5 Organize the dataset

We want to reorder the columns to make the dataset easier to be understood. And we also renamed the columns to make the names shorter.

```
IMDB <- IMDB[,c(9,4,5,14,12,2,3,13,1,6,10,7,8,11,15)]
colnames(IMDB) <- c("budget", "gross", "user_vote", "critic_review_ratio",
                     "movie_fb", "director_fb", "actor1_fb", "other_actors_fb",
                     "duration", "face_number", "year", "country", "content",
                     "imdb_score", "binned_score")
```

5.6 Split Data

Here we split data into training, validation and test sets with the ratio of 6:2:2.

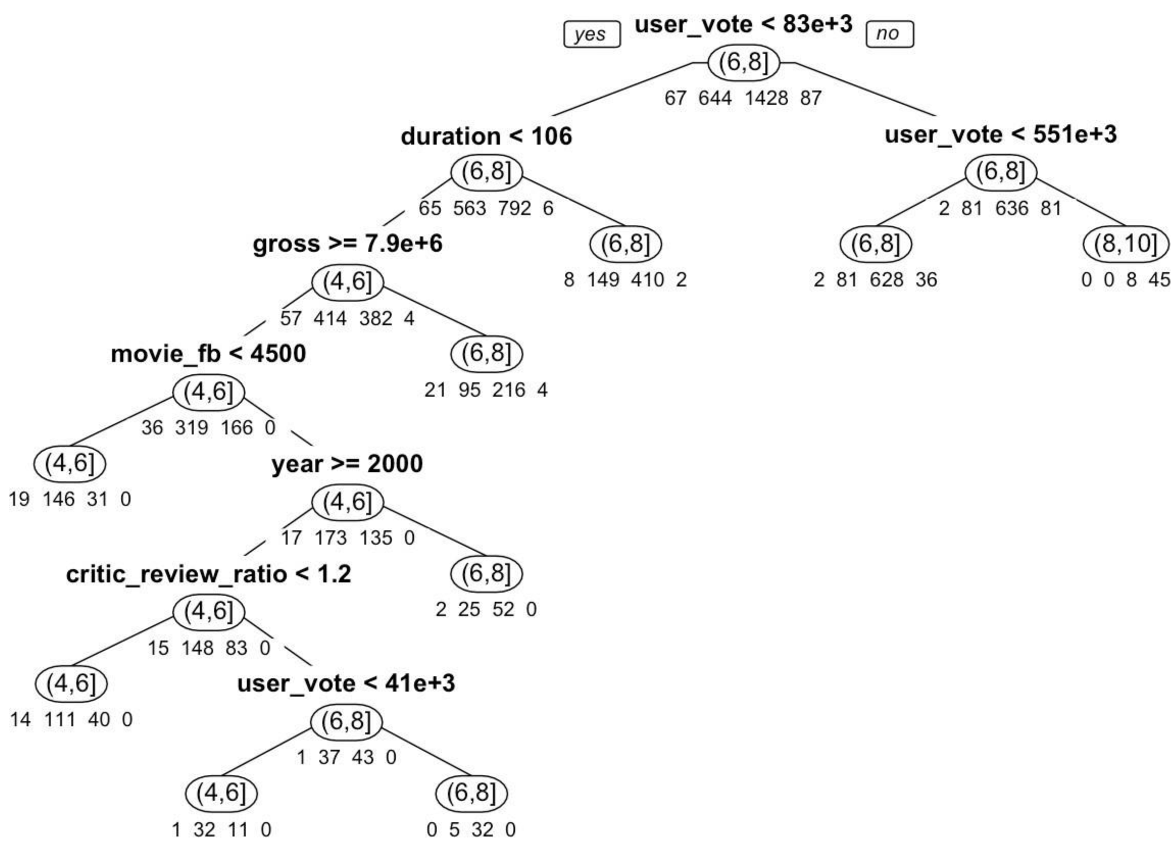
```
set.seed(45)
train.index <- sample(row.names(IMDB), dim(IMDB)[1]*0.6)
valid.index <- sample(setdiff(row.names(IMDB), train.index), dim(IMDB)[1]*0.2)
test.index <- setdiff(row.names(IMDB), union(train.index, valid.index))
train <- IMDB[train.index, ]
valid <- IMDB[valid.index, ]
test <- IMDB[test.index, ]
```

6 Implement Algorithm

6.1 Classification Tree

6.1.1 Full-grown Tree

```
library(rpart)
library(rpart.plot)
# Full grown tree
class.tree <- rpart(binned_score ~ . -imdb_score, data = train, method = "class")
## plot tree
prp(class.tree, type = 1, extra = 1, under = TRUE, split.font = 2, varlen = 0)
```



Classification rules:

- 1. If (user_vote >= 551000) then class = (8,10].
- 2. If (83000 <= user_vote < 551000) then class = (6,8].
- 3. If (user_vote < 83000) and (duration >= 106) then class = (6,8].
- 4. If (user_vote < 83000) and (duration < 106) and (gross < 7900000) then class = (6,8].

- 5. If (user_vote < 83000) and (duration < 106) and (gross >= 7900000) and (movie_fb < 4500) then class = (4,6].
- 6. If (user_vote < 83000) and (duration < 106) and (gross >= 7900000) and (movie_fb >= 4500) and (year < 2000) then class = (6,8].
- 7. If (user_vote < 83000) and (duration < 106) and (gross >= 7900000) and (movie_fb >= 4500) and (year >= 2000) and (critic_review_ratio < 1.2) then class = (4,6].
- 8. If (user_vote < 41000) and (duration < 106) and (gross >= 7900000) and (movie_fb >= 4500) and (year >= 2000) and (critic_review_ratio >= 1.2) then class = (4,6].
- 9. If (41000 <= user_vote < 83000) and (duration < 106) and (gross >= 7900000) and (movie_fb >= 4500) and (year >= 2000) and (critic_review_ratio >= 1.2) then class = (6,8].

From these rules, we can conclude that movies with a lot of votes in imdb website tend to have a higher score, which really makes sense because popular movies will have a lot of fans to vote high scores for them.

On the contrary, if a movie has fewer votes, it can still be a good movie if its duration is longer (rule #3).

It is surprise to see that movies make less profit are good, but ok if they make more profit (rule #4).

6.1.2 Best-pruned Tree

```
# cross-validation procedure
# argument cp sets the smallest value for the complexity parameter.
set.seed(51)
cv.ct <- rpart(binned_score ~ . -imdb_score, data = train, method = "class",
               cp = 0.00001, minsplit = 5, xval = 5)
printcp(cv.ct)
```

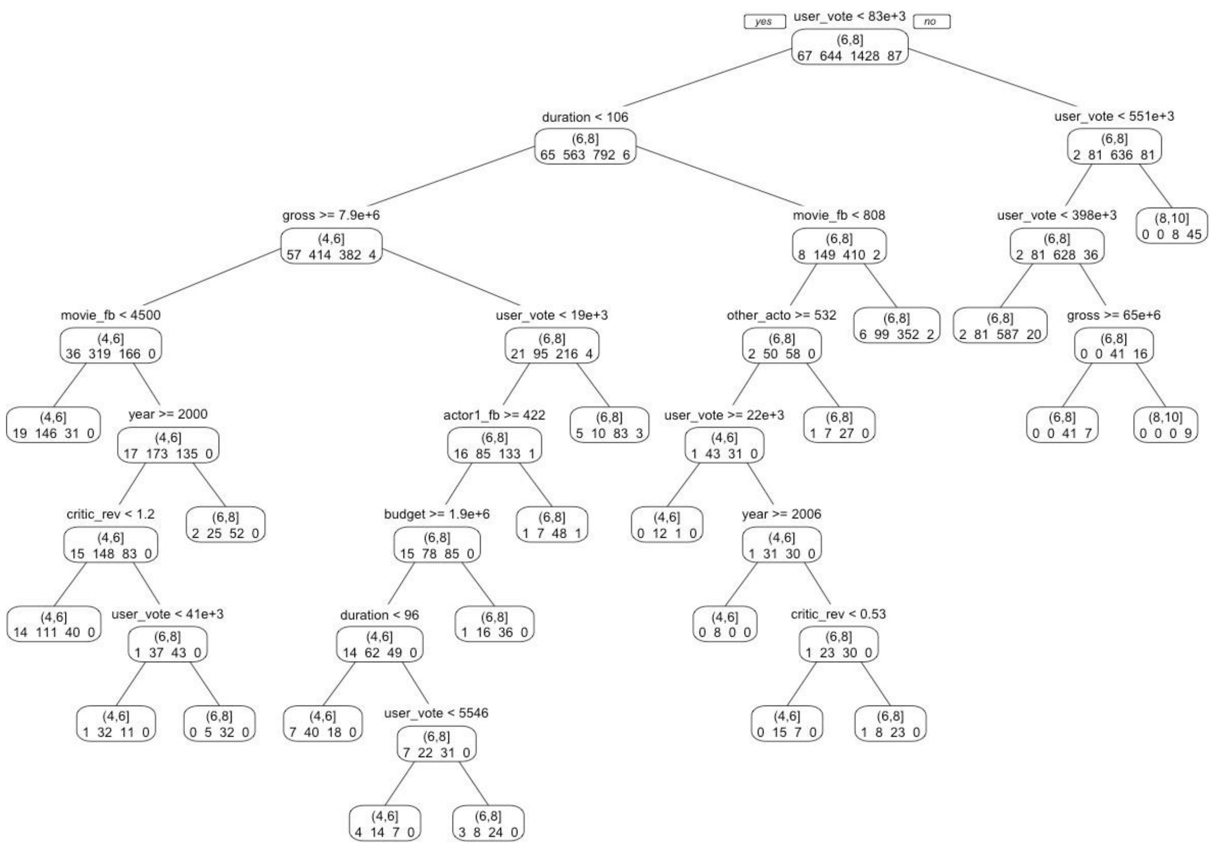
```
##
## Classification tree:
## rpart(formula = binned_score ~ . - imdb_score, data = train,
##     method = "class", cp = 1e-05, minsplit = 5, xval = 5)
##
## Variables actually used in tree construction:
##  [1] actor1_fb           budget              content
##  [4] country             critic_review_ratio director_fb
##  [7] duration            face_number         gross
## [10] movie_fb            other_actors_fb     user_vote
## [13] year
##
## Root node error: 798/2226 = 0.35849
##
## n= 2226
##
##            CP nsplit rel error  xerror     xstd
## 1  0.06390977      0   1.00000 1.00000 0.028353
## 2  0.04636591      3   0.80827 0.86216 0.027322
## 3  0.01691729      4   0.76190 0.79574 0.026697
## 4  0.00751880      8   0.69424 0.77694 0.026503
## 5  0.00626566     10   0.67920 0.76316 0.026357
## 6  0.00563910     13   0.66040 0.75815 0.026303
## 7  0.00543024     15   0.64912 0.75815 0.026303
## 8  0.00501253     20   0.61278 0.75564 0.026276
## 9  0.00407268     25   0.58772 0.76817 0.026411
## 10 0.00375940     29   0.57143 0.77820 0.026517
## 11 0.00325815     45   0.50877 0.78070 0.026543
## 12 0.00322234     50   0.49248 0.78446 0.026582
## 13 0.00313283     57   0.46992 0.78446 0.026582
## 14 0.00292398     63   0.45113 0.79574 0.026697
## 15 0.00250627     66   0.44236 0.79574 0.026697
## 16 0.00219298    112   0.31830 0.80827 0.026821
## 17 0.00187970    120   0.29950 0.80451 0.026784
## 18 0.00167084    133   0.27444 0.80576 0.026797
## 19 0.00156642    142   0.25940 0.80576 0.026797
## 20 0.00125313    151   0.24436 0.83208 0.027049
## 21 0.00093985    200   0.18045 0.84712 0.027188
## 22 0.00083542    204   0.17669 0.84336 0.027154
## 23 0.00062657    207   0.17419 0.85714 0.027278
## 24 0.00050125    213   0.17043 0.85714 0.027278
## 25 0.00027847    218   0.16792 0.86842 0.027376
## 26 0.00025063    227   0.16541 0.86842 0.027376
## 27 0.00001000    237   0.16291 0.86842 0.027376
```

The 8th tree has the lowest cross-validation error (xerror): 0.75564.

```
# prune by lowest cp
pruned.ct <- prune(cv.ct,
                   cp  =  cv.ct$cptable[which.min(cv.ct$cptable[,"xerror"]),"CP"])
length(pruned.ct$frame$var[pruned.ct$frame$var == "<leaf>"])
```

```
## [1] 21
```

```
prp(pruned.ct, type = 1, extra = 1, split.font = 1, varlen = -10)
```



6.1.3 Apply Model

```
# apply model on training set
tree.pred.train <- predict(pruned.ct, train, type = "class")
# generate confusion matrix for training data
confusionMatrix(tree.pred.train, train$binned_score)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]     45   378   115      0
##     (6,8]     22   266  1305     33
##     (8,10]     0     0     8     54
##
## Overall Statistics
##
##                Accuracy : 0.7803
##                  95% CI : (0.7625, 0.7974)
##     No Information Rate : 0.6415
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5228
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity                0.0000       0.5870       0.9139       0.62069
## Specificity                1.0000       0.8989       0.5977       0.99626
## Pos Pred Value                NaN       0.7026       0.8026       0.87097
## Neg Pred Value             0.9699       0.8424       0.7950       0.98475
## Prevalence                 0.0301       0.2893       0.6415       0.03908
## Detection Rate             0.0000       0.1698       0.5863       0.02426
## Detection Prevalence       0.0000       0.2417       0.7305       0.02785
## Balanced Accuracy          0.5000       0.7429       0.7558       0.80847
```

Accuracy is 0.7803 for training set.

```
# apply model on validation set
tree.pred.valid <- predict(pruned.ct, valid, type = "class")
# generate confusion matrix for validation data
confusionMatrix(tree.pred.valid, valid$binned_score)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]      9    88    62      0
##     (6,8]      6   121   424     10
##     (8,10]     0     0     5     17
##
## Overall Statistics
##
##                Accuracy : 0.7129
##                  95% CI : (0.6789, 0.7453)
##     No Information Rate : 0.6617
##     P-Value [Acc > NIR] : 0.001616
##
##                   Kappa : 0.345
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity               0.00000       0.4211       0.8635       0.62963
## Specificity               1.00000       0.8668       0.4542       0.99301
## Pos Pred Value                NaN       0.5535       0.7558       0.77273
## Neg Pred Value            0.97978       0.7925       0.6298       0.98611
## Prevalence                0.02022       0.2817       0.6617       0.03639
## Detection Rate            0.00000       0.1186       0.5714       0.02291
## Detection Prevalence      0.00000       0.2143       0.7561       0.02965
## Balanced Accuracy         0.50000       0.6439       0.6589       0.81132
```

Accuracy is 0.7129 for validation set.

```
# apply model on test set
tree.pred.test <- predict(pruned.ct, test, type = "class")
# generate confusion matrix for test data
confusionMatrix(tree.pred.test, test$binned_score)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]      8   107    76      0
##     (6,8]      5   105   423     10
##     (8,10]     0     0     1      8
##
## Overall Statistics
##
##                Accuracy : 0.7241
##                  95% CI : (0.6904, 0.756)
##     No Information Rate : 0.6729
##     P-Value [Acc > NIR] : 0.001485
##
##                   Kappa : 0.3651
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity                0.0000       0.5047       0.8460       0.44444
## Specificity                1.0000       0.8418       0.5062       0.99862
## Pos Pred Value                NaN       0.5602       0.7790       0.88889
## Neg Pred Value             0.9825       0.8098       0.6150       0.98638
## Prevalence                 0.0175       0.2853       0.6729       0.02423
## Detection Rate             0.0000       0.1440       0.5693       0.01077
## Detection Prevalence       0.0000       0.2571       0.7308       0.01211
## Balanced Accuracy          0.5000       0.6733       0.6761       0.72153
```

Accuracy is 0.7241 for test set.

6.2 K-Nearest Neighbors

6.2.1 Data Pre-processing

First, we need to prepare our data for applying knn purpose. Dummy variables are required for categorical variables. We use a copy of our data, so we can still use our original data in the future.

```r
library(FNN)
# Use model.matrix() to create dummy variables for country and content.
IMDB2 <- IMDB
IMDB2$country <- as.factor(IMDB2$country)
IMDB2$content <- as.factor(IMDB2$content)
IMDB2[,c("country_UK", "country_USA", "country_Others")] <- model.matrix( ~ count
ry - 1, data = IMDB2)
IMDB2[,c("content_G", "content_NC17", "content_PG", "content_PG13", "content_R")]
<- model.matrix( ~ content - 1, data = IMDB2)
# Select useful variables for future prediction.
IMDB2 <- IMDB2[, c(1,2,3,4,5,6,7,8,9,10,11,16,17,18,19,20,21,22,23,15)]
# Partition the data into training and validation sets.
set.seed(52)
train2 <- IMDB2[train.index, ]
valid2 <- IMDB2[valid.index, ]
test2 <- IMDB2[test.index, ]
```

Then we need to normalize our data.

```r
# initialize normalized training, validation, test data, complete data frames to
originals
train2.norm <- train2
valid2.norm <- valid2
test2.norm <- test2
IMDB2.norm <- IMDB2
# use preProcess() from the caret package to normalize predictors.
norm.values <- preProcess(train2[, -20], method=c("center", "scale"))
train2.norm[, -20] <- predict(norm.values, train2[, -20])
valid2.norm[, -20] <- predict(norm.values, valid2[, -20])
test2.norm[, -20] <- predict(norm.values, test2[, -20])
IMDB2.norm[, -20] <- predict(norm.values, IMDB2[, -20])
```

6.2.2 Find the best k

We will set k as 1 to 20, and build 20 different models. We calculate each model's classification accuracy, and find the best k according to the highest accuracy.

```r
# initialize a data frame with two columns: k, and accuracy.
accuracy.df <- data.frame(k = seq(1, 20, 1), accuracy = rep(0, 20))
# compute knn for different k on validation data.
for(i in 1:20) {
  knn.pred <- knn(train2.norm[, -20], valid2.norm[, -20],
                  cl = train2.norm[, 20], k = i)
  accuracy.df[i, 2] <- confusionMatrix(knn.pred, valid2.norm[, 20])$overall[1]
}
accuracy.df
```

```
##      k  accuracy
## 1    1 0.6671159
## 2    2 0.5916442
## 3    3 0.6940701
## 4    4 0.6792453
## 5    5 0.6954178
## 6    6 0.6913747
## 7    7 0.6886792
## 8    8 0.6873315
## 9    9 0.7142857
## 10  10 0.7061995
## 11  11 0.7021563
## 12  12 0.6873315
## 13  13 0.6940701
## 14  14 0.6846361
## 15  15 0.7008086
## 16  16 0.6886792
## 17  17 0.6913747
## 18  18 0.6886792
## 19  19 0.6873315
## 20  20 0.6927224
```

When k = 9, we get the highest accuracy: 0.7142857

6.2.3 Apply model on test set

```
# apply model on test set
knn.pred.test <- knn(train2.norm[, -20], test2.norm[, -20],
                cl = train2.norm[, 20], k = 9)
# generate confusion matrix for test data
accuracy <- confusionMatrix(knn.pred.test, test2.norm[, 20])$overall[1]
accuracy
```

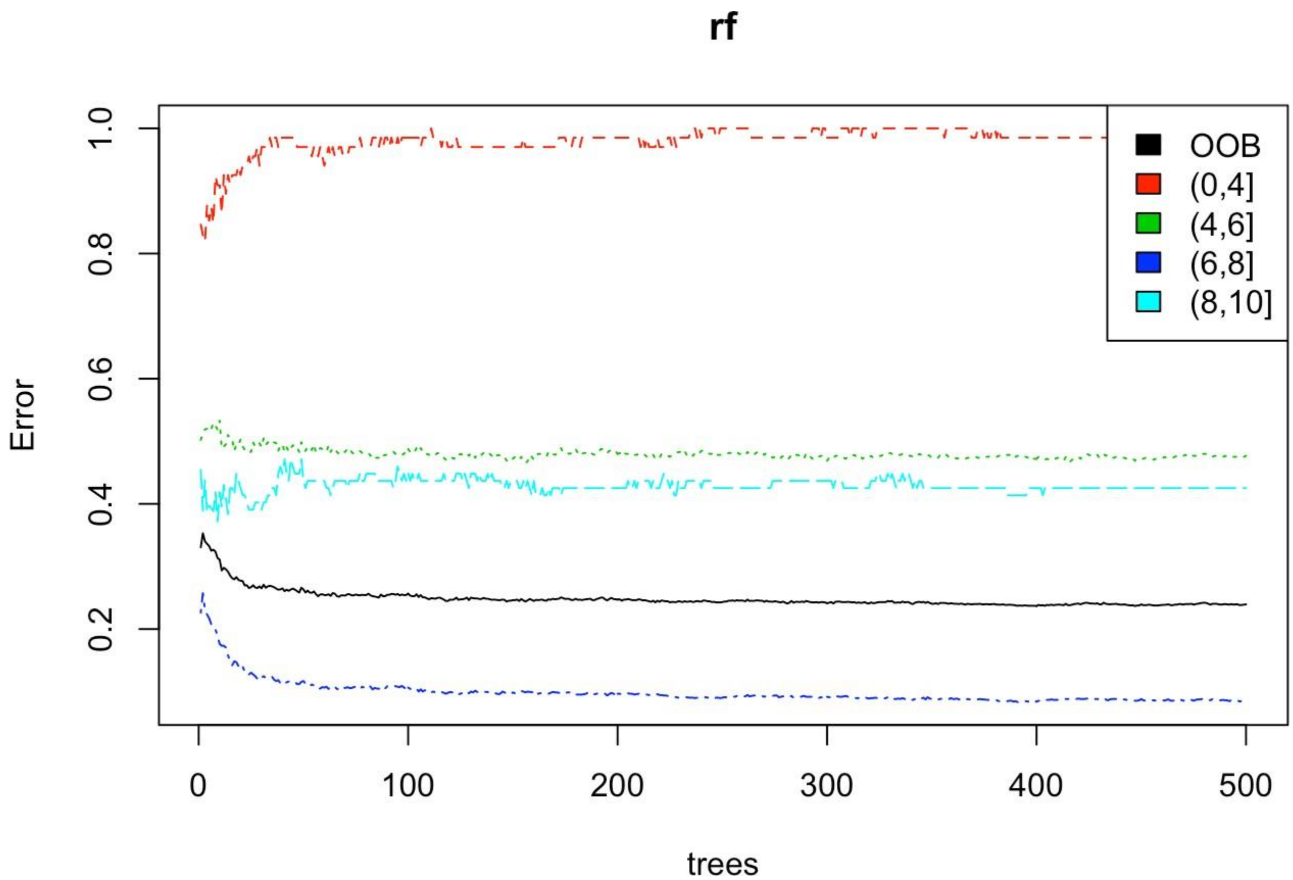
```
## Accuracy
## 0.7456258
```

Test set accuracy: 0.7456258

6.3 Random Forest

6.3.1 Build Model

```
library(randomForest)
set.seed(53)
rf <- randomForest(binned_score ~ . -imdb_score, data = train, mtry = 5)
# Show model error
plot(rf)
legend('topright', colnames(rf$err.rate), col=1:5, fill=1:5)
```



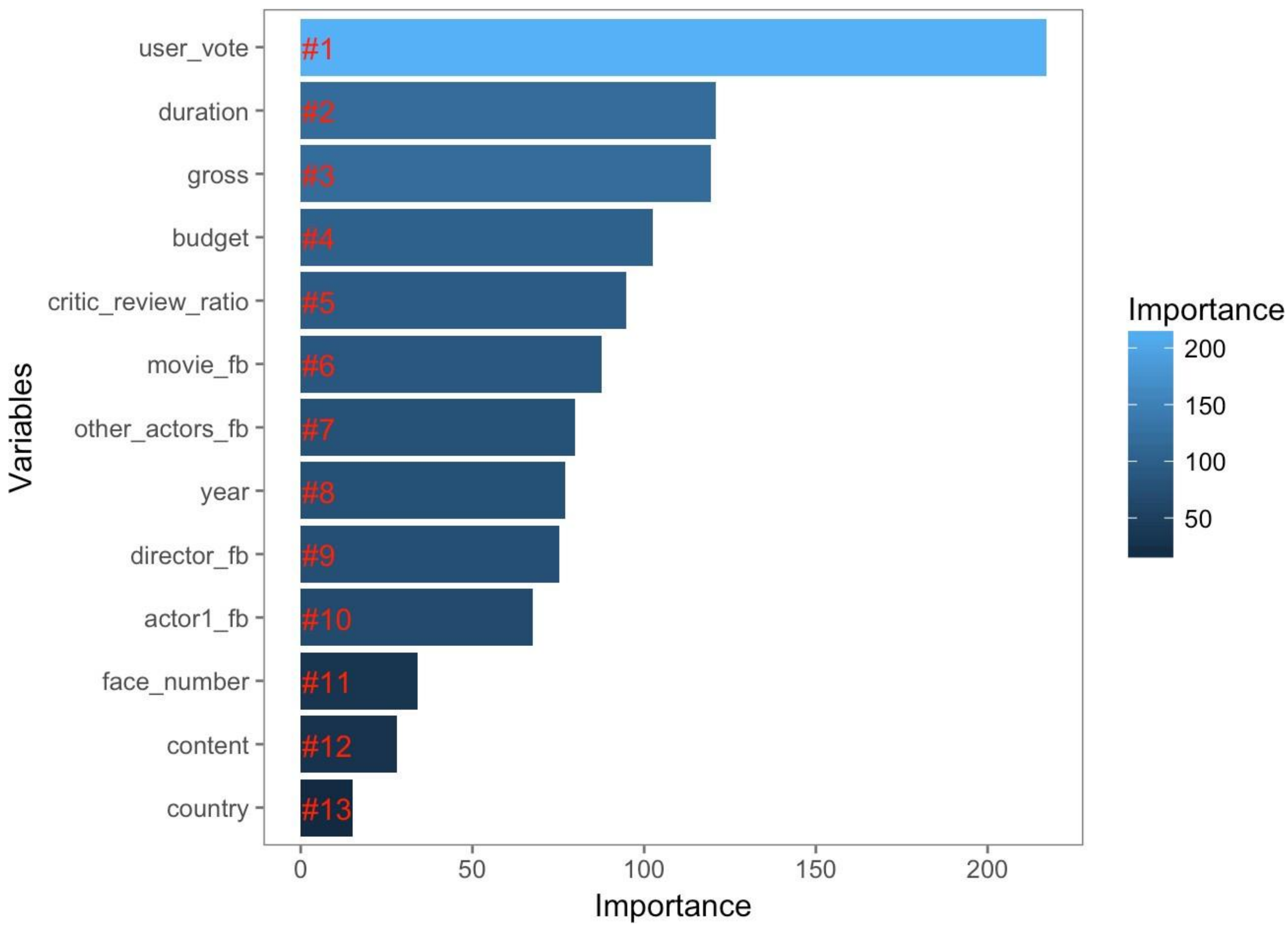
The black line shows the overall error rate which falls below 30%. The red, green, blue and aqua lines show the error rate for bad, ok, good and excellent movies respectively. We can see that right now we're much more successful predicting good movies. We cannot predict bad movies very well.

Let's look at relative variable importance by plotting the mean decrease in Gini calculated across all trees.

```
# Get importance
importance <- importance(rf)
varImportance <- data.frame(Variables = row.names(importance),
                            Importance = round(importance[ ,'MeanDecreaseGini'],2
))

# Create a rank variable based on importance
rankImportance <- varImportance %>%
  mutate(Rank = paste0('#',dense_rank(desc(Importance))))

# Use ggplot2 to visualize the relative importance of variables
ggplot(rankImportance, aes(x = reorder(Variables, Importance),
                           y = Importance, fill = Importance)) +
  geom_bar(stat='identity') +
  geom_text(aes(x = Variables, y = 0.5, label = Rank),
            hjust=0, vjust=0.55, size = 4, colour = 'red') +
  labs(x = 'Variables') +
  coord_flip() +
  theme_few()
```



From the plot, we see **User_vote** is a very important variable, while **face_number**, **content** and **country** are not so important.

6.3.2 Apply Model

```
set.seed(632)
# apply model on validation set
rf.pred.valid <- predict(rf, valid)
# generate confusion matrix for validation data
confusionMatrix(rf.pred.valid,  valid$binned_score)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]      7   106    43      0
##     (6,8]      8   103   446     12
##     (8,10]     0     0     2     15
##
## Overall Statistics
##
##                Accuracy : 0.7642
##                  95% CI : (0.7319, 0.7943)
##     No Information Rate : 0.6617
##     P-Value [Acc > NIR] : 8.023e-10
##
##                   Kappa : 0.4547
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity               0.00000       0.5072       0.9084       0.55556
## Specificity               1.00000       0.9062       0.5100       0.99720
## Pos Pred Value                NaN       0.6795       0.7838       0.88235
## Neg Pred Value            0.97978       0.8242       0.7399       0.98345
## Prevalence                0.02022       0.2817       0.6617       0.03639
## Detection Rate            0.00000       0.1429       0.6011       0.02022
## Detection Prevalence      0.00000       0.2102       0.7668       0.02291
## Balanced Accuracy         0.50000       0.7067       0.7092       0.77638
```

Accuracy is 0.7642 for validation set.

```r
set.seed(633)
# apply model on test set
rf.pred.test <- predict(rf, test)
# generate confusion matrix for test data
confusionMatrix(rf.pred.test, test$binned_score)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     1     0      0
##     (4,6]      8   114    51      0
##     (6,8]      5    97   447     10
##     (8,10]     0     0     2      8
## 
## Overall Statistics
## 
##                Accuracy : 0.7658
##                  95% CI : (0.7337, 0.7958)
##     No Information Rate : 0.6729
##     P-Value [Acc > NIR] : 1.796e-08
## 
##                   Kappa : 0.4515
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity              0.000000       0.5377       0.8940       0.44444
## Specificity              0.998630       0.8889       0.5391       0.99724
## Pos Pred Value           0.000000       0.6590       0.7996       0.80000
## Neg Pred Value           0.982480       0.8281       0.7120       0.98636
## Prevalence               0.017497       0.2853       0.6729       0.02423
## Detection Rate           0.000000       0.1534       0.6016       0.01077
## Detection Prevalence     0.001346       0.2328       0.7524       0.01346
## Balanced Accuracy        0.499315       0.7133       0.7165       0.72084
```

Accuracy is 0.7658 for test set.

7 Conclusion

Accuracy table for different models:

Dataset	Decision Tree	K-NN	Random Forest
Training	0.7803		
Validation	0.7129	0.7143	0.7642
Test	0.7241	0.7456	0.7658

For Decision tree model, we have a higher accuracy for training data because the tree was built based on the training data.

Based on the overall performance, we find the best model is random forest, which gives a high accuracy around 0.76.