IMDb_rating_predictor_Full

Callum Thickett

25/01/2022

IMDb_rating_predictor

The goal of this personal project is to try and create a predictive model that can take information from a movie and predict what its IMDb rating would be. When looking for a movie to watch i always find my self consulting the ratings on IMDb, and i tend to find any movie that has a rating of ~7 or higher will generally be a very enjoyable film. I thought it would be interesting to understand further the factors that go into making a succesful movie, based on the criteria of rating.

This project will go through the entire process of gathering, cleaning, manipulating, visualising, and statistically modeling data.

Webscraping

To start, we need to collect data on as many relevent movies as we can. I thought the best way to use this was to create a webscraper that can search thorugh IMDb's developer mode pages to gather information on movies that fit a certain criteria, this criteria includes:

- the movie must be English
- have atleast 2500 IMDb votes
- has to be a feature length movie
- had to be created in 1970 or later.

```
Movies_raw <- data.frame(matrix(ncol = 3,nrow = 0))
cols<- c("name","year","rating")
colnames(Movies_raw) <-cols
```

Create inital dataframe that we will appened with our scraper function

```
scraper <-function(n_movies) {
  page_number <- 1
  for (i in 1:as.integer(n_movies/50)) {
    link <- paste("https://www.imdb.com/search/title/?title_type=feature&release_date=1970-01-01,&num_v
    page <- read_html(link)

    name = page %>% html_nodes(".lister-item-header a") %>% html_text()
```

```r
year = page %>%  html_nodes(".text-muted.unbold") %>%  html_text()

rating = page %>% html_nodes (".ratings-imdb-rating strong") %>%  html_text()

##get individual movie links:
movie_links = page %>% html_nodes(".lister-item-header a") %>%
  html_attr("href") %>% substr(., 1, 16) %>%
  paste("https://www.imdb.com", ., "/reference" , sep = "")


## function to get data from each of the generated movie links.

get_inner <- function(movie_link) {

  movie_page <- read_html(movie_link)
  cast = movie_page %>%
    html_nodes(".itemprop .itemprop") %>%
    html_text() %>%
    .[1:5] %>%
    paste(collapse = ",")

  synopsis <- movie_page %>%
    html_nodes(".titlereference-section-overview div:nth-child(1)") %>%
    html_text()

  genre1 <- movie_page %>%
    html_nodes(".titlereference-header .ipl-inline-list__item:nth-child(3) a:nth-child(1)") %>%
    html_text()

  genre2 <-movie_page %>%
    html_nodes(".titlereference-header a+ a") %>%
    html_text()

  imdb_votes <- movie_page %>%
    html_nodes(".ipl-rating-star__total-votes") %>%
    html_text()

  director <- movie_page %>%
    html_nodes("hr+ .titlereference-overview-section a") %>%
    html_text()

  awards <-  movie_page %>%
    html_nodes(".titlereference-overview-section:nth-child(6) .ipl-inline-list__item:nth-child(1)") %>%
    html_text()

  budget <- movie_page %>%
    html_nodes(".titlereference-section-box-office .ipl-zebra-list__item:nth-child(1) .ipl-zebra-list__
    html_text()

  opening_weekend <- movie_page %>%
    html_nodes(".titlereference-section-box-office .ipl-zebra-list__item:nth-child(2) .ipl-zebra-list__
    html_text()
  gross_world <- movie_page %>%
    html_nodes(".titlereference-section-box-office .ipl-zebra-list__item~ .ipl-zebra-list__item+ .ipl-ze
    html_text()

  run_time <- movie_page %>%
    html_nodes(".titlereference-section-additional-details .ipl-zebra-list__item:nth-child(2) .ipl-inli
    html_text()

  return(c(genre1[1],genre2[1],cast,synopsis[1],imdb_votes[1],director[1],awards[1],
           budget[1],opening_weekend[1],gross_world[1],run_time[1]))


}
movie_inner_mat = sapply(movie_links, get_inner, USE.NAMES = FALSE)

movie_inner_df = as.data.frame(t(movie_inner_mat))

##clean movie_inner_df to account for movies with only one recorded genre.
# without this converting to a data frame wont work (records will be too long
# since genre2 returns a vector)


  movie_inner_df$V2 <-sub("\n", NA, x = movie_inner_df$V2)


   Movies_raw_temp = data.frame(
  name = name,
  year = year,
  rating = rating,
  genre1 = movie_inner_df$V1,
  genre2 = movie_inner_df$V2,
  cast = movie_inner_df$V3,
  synopsis = movie_inner_df$V4,
  imdb_votes = movie_inner_df$V5,
  director = movie_inner_df$V6,
  awards = movie_inner_df$V7,
  budget = movie_inner_df$V8,
  opening_weekend = movie_inner_df$V9,
  gross_world = movie_inner_df$V10,
  run_time = movie_inner_df$V11,
  stringsAsFactors =
    FALSE
)


    Movies_raw <- rbind(Movies_raw,Movies_raw_temp)

    page_number <-page_number+50
     ## percent complete tracker
     pct_complete <-paste(floor((page_number/n_movies)*100),"% complete")
    print(pct_complete)
  }
  return(Movies_raw)
}
#Movies_raw <- scraper(3400)
```

Main scraper function

```


#write.csv(Movies_raw,file="Movies_raw.csv",row.names = FALSE)
```

export Movies_raw to a csv.

Data cleaning

In this section we take the raw data gather from web scraping and clean it up to create useable data that can be analysed.

```
Movies_raw <- read.csv("Movies_raw.csv")
```

first, create a copy to work on.

```
Movies_Clean <- Movies_raw
```

Check for duplicate records

```
tot.records <-nrow(Movies_Clean)
uniq.records <-nrow(unique(Movies_Clean))

paste("There are",tot.records,"total records and", uniq.records, "unique records.",tot.records-uniq.rec
```

```
## [1] "There are 3450 total records and 3400 unique records. 50 duplicates found."
```

remove duplicate records

```
Movies_Clean <- unique(Movies_Clean)
```

clean the year variable

```
## remove the parentheses in year and convert to a date format.
year_cleaner <- function(date) {
  date <- as.integer(gsub("[^0-9]","",date))

}
Movies_Clean$year <-year_cleaner(Movies_Clean$year)
```

clean the rating variable

```
Movies_Clean$rating <- as.numeric(Movies_Clean$rating)
```

```


extra_char_cleaner <- function(x,as.numeric=FALSE) {
  x <- gsub("[(),a-b]","",x)
  if (as.numeric==TRUE){
    as.numeric(x)
  }

  }
```

generic function for removing commas and parentheses cleaned the votes variable

```
Movies_Clean$imdb_votes <- extra_char_cleaner(Movies_Clean$imdb_votes,
                                              as.numeric =TRUE )
```

cleaning budgets variable.

here we have budgets in a variety of currencies, i will convert them all to USD through use of a web scraper. first we need to remove free space from the budget.

```
Movies_Clean$budget <-str_trim(Movies_Clean$budget)
```

scrape all possible currency codes. then we can just see which of these exist in our data frame.

```
currency_codes_link<-"https://www.iban.com/currency-codes"
currency_codes_page <- read_html(currency_codes_link)
currency_codes <- currency_codes_page %>% html_nodes("td:nth-child(3)") %>% html_text()

## clean currency_codes to get rid of the blank values

currency_codes <- currency_codes[nchar(currency_codes)==3]
```

lets take a look at the currency codes we have in our data frame, some of them will need converting to standardized currencies.

```
#codes_dirty includes some extras, just need to get the ones with 3 characters
codes_dirty <-gsub("[^A-Z.-]","",c(Movies_Clean$budget,Movies_Clean$opening_weekend,Movies_Clean$gross_
  .[.!=""] %>%
  .[!is.na(.)] %>%
  unique(.)

codes_clean <- codes_dirty[nchar(codes_dirty)==3]
```

remove none standardized currencies, we can add them back manually after.

```
codes_removed <- codes_clean[!codes_clean %in% currency_codes]

codes_clean <- codes_clean[!codes_clean %in% codes_removed]
```

as we can see there are 17 different currency codes in the data frame split between the three columns; budget, opening_weekend, and gross_world.

currency code conversion rate data frame lets create a dataframe with conversion rates that we can call later.

```
# function for getting conversion rates from the currency codes.
currency_convertor<- function(currency_code) {
  link <- paste("https://www.xe.com/currencyconverter/convert/?Amount=1&From=",
                currency_code,"&To=USD",sep = "")
  page <- read_html(link)
  conversion_rate <- page %>% html_nodes(".iGrAod , .faded-digits") %>% html_text()
  return(conversion_rate)
}
```

```
##apply the currency convertor function
 conversion_rates <- sapply(codes_clean,currency_convertor)
 #transform output into a dataframe, and then convert to longer
 #version
 conversion_rates_df <- as.data.frame(conversion_rates, stringsAsFactors =
                                      FALSE)
 conversion_rates_df <- as.data.frame(t(conversion_rates_df))
```

clean up for the conversion rate dataframe

```
## add index
conversion_rates_df$currency_codes <- row.names(conversion_rates_df)
rownames(conversion_rates_df) <- 1:nrow(conversion_rates_df)
## remove V2 (useless info from scraping )
conversion_rates_df <- conversion_rates_df %>%
  select(currency_codes, Conversion=V1)
##manually add conversion (est)for the 5 depreciated currencies
conversion_rates_df <-
  rbind(conversion_rates_df, data.frame(currency_codes=c("DEM","FRF","ITL","NLG","ESP"),
                                        Conversion=c("1.7293","0.172386","0.000581922", "0.513146","0.33

## clean and change to numeric
conversion_rates_df$Conversion<-
  as.numeric(gsub("[^0-9.-]", "", conversion_rates_df$Conversion))
```

```
## function to get currency codes for other columns
##NOTE: should just change this to accept vectors.

get_conversion_rates <- function(col_name) {
## i should cut this part of the function down.
  #a lot isnt needed and its ugly.
  col_name_new <-"opening_weekend"
  col_name_new <- paste("Movies_Clean$",substr(col_name,1,4),
                    sep = "")
col_name_suffix <- paste(substr(col_name,1,4),"Converted")
col_name_new <- gsub("[:upper:]","",Movies_Clean[,col_name])
col_name_new <- str_trim(col_name_new)
col_name_new <- gsub("[^A-Z]","",col_name_new)
col_name_new[nchar(col_name_new) <3] <-NA
col_name_new <- substr(col_name_new,1,3)

#find cols with a currency code, match them to the conversion rate
## in the conversion_rates data frame.
for (i in 1:length(col_name_new)) {
  if  (col_name_new[i] %in% conversion_rates_df$currency_codes) {
    col_name_new[i] <- conversion_rates_df$Conversion[col_name_new[i]==conversion_rates_df$currency_code
  }}

## clean col to only get the value.
Movies_Clean[,col_name] <- str_trim(Movies_Clean[,col_name])
#remove everything after the currency.
Movies_Clean[,col_name] <-sub(" .*","",Movies_Clean[,col_name])
#remove all none numeric characters.
Movies_Clean[,col_name] <- gsub("[^0-9.-]","",Movies_Clean[,col_name])
#create new col with USD values.
Movies_Clean[,col_name_suffix] <-as.numeric(Movies_Clean[,col_name]) *
  as.numeric(col_name_new)
## need to change NA values in new column for pre-exising
## values in the OG column.
for (i in 1:length(Movies_Clean[,col_name])) {
  if (is.na(Movies_Clean[,col_name_suffix][i])) {
    Movies_Clean[,col_name_suffix][i] <- Movies_Clean[,col_name][i]
  }
}
Movies_Clean$openning_USD
return(Movies_Clean[,col_name_suffix])
}
```

apply the above function to the three money columns.

```
## get cleaned budget in USD.
Movies_Clean$budget_USD <- as.integer(get_conversion_rates("budget"))
```

```
## Warning in get_conversion_rates("budget"): NAs introduced by coercion
```

```
## get the cleaned opening_weekend in USD.
Movies_Clean$openning_USD <- as.integer(get_conversion_rates("opening_weekend"))
```

```
## Warning in get_conversion_rates("opening_weekend"): NAs introduced by coercion
```

```
## get cleaned gross in USD
Movies_Clean$gross_USD <- as.integer(get_conversion_rates("gross_world"))
```

```
## Warning: NAs introduced by coercion to integer range
```

now we can adjust for inflation. note: this isnt a comprehensive adjustment since im adjusting the USD value not the orignal currency. For the sake of time and the scope of this project i dont think this will be a huge deal.

```
## equation for inflation: CPI_today/CPI_year xusd_year =usd_today

## first we need to get CPI values for every year from 1970 to 2021. (2021 CPI are averaged from the fi

CPI_resource <- read_html("https://www.usinflationcalculator.com/inflation/consumer-price-index-and-annu

tables <- CPI_resource %>% html_table(fill=TRUE)
view(tables)
CPI_df <- tables[[1]] %>%
  select(X1,X14)
colnames(CPI_df) <-c("Year","Annual_CPI")

## remove the two top rows, these were the OG names from the website table. dont need them.
CPI_df <- CPI_df[3:nrow(CPI_df),]
```

add a multipier column to the CPI_df i.e the amount the currency in a given year should be mulitplied by to account for inflation.

```
## convert CPI to numeric.
CPI_df$Annual_CPI <-as.numeric(CPI_df$Annual_CPI)
for (i in 1:nrow(CPI_df)) {
  ## 277.948 is the estimated CPI for 2021.
  CPI_df$multiplier[i] <- 277.948/CPI_df$Annual_CPI[i]
}
```

```
## Warning: Unknown or uninitialised column: ‘multiplier‘.
```

```
apply_inflation <- function(x) {
  for (i in 1:nrow(Movies_Clean)) {
    x[i] <- x[i] * CPI_df$multiplier[CPI_df$Year ==Movies_Clean$year[i]]
  }
  return(x)
}
```

function to apply inflation rate to our data

```
Movies_Clean$budget_USD <- apply_inflation(Movies_Clean$budget_USD)

Movies_Clean$gross_USD <- apply_inflation(Movies_Clean$gross_USD)

Movies_Clean$openning_USD <-apply_inflation(Movies_Clean$openning_USD)
```

apply function to the 3 columns. Finally, after taking a further look at the data there seems to be an issue with budget. for some of the movies that have less available information, the budget scraped from the web is actually the opening weekend or gross. To solve this i'm going to make all budgets NA if they don't have an opening weekend value associated with them. This isnt an ideal with to deal with the issue. However, due to the amount of data missing in these columns i doubt it will be an issue. I'm probably going to end up using these as factor variables, and it wont matter if some indiviudal recrods are slightly wrong.

```
Movies_Clean$budget_USD[is.na(Movies_Clean$openning_USD)] <-NA
```

clean awards starting with oscar wins.

```
##clean awards columns, separate into Oscar wins, Oscar nominations
##and other nominations.
Movies_Clean$awards <- str_trim(Movies_Clean$awards)

## function to get just the number related to oscar wins

for (i in 1:nrow(Movies_Clean)) {
  if (grepl("Oscar.*",Movies_Clean$awards[i]) ==TRUE ) {
    Movies_Clean$Oscar_wins[i] <- gsub("Oscar.*","",Movies_Clean$awards[i])
  } else {
    Movies_Clean$Oscar_wins[i] <-0
  }
}
  for (i in 1:nrow(Movies_Clean)) {
    if(grepl("Nomin.*",Movies_Clean$Oscar_wins[i]) ==TRUE) {
      Movies_Clean$Oscar_wins[i] <- (gsub("[0-9^]","",
                                          Movies_Clean$Oscar_wins[i]))
    }
  }
## clean the returned string to just get the number
Movies_Clean$Oscar_wins[is.na(Movies_Clean$Oscar_wins)] <- 0
Movies_Clean$Oscar_wins <-as.integer(gsub("[a-z,A-Z]","",Movies_Clean$Oscar_wins))
```

oscar nominations

```
######Oscar nominations
for (i in 1:nrow(Movies_Clean)) {
  if (grepl("Oscar.*",Movies_Clean$awards[i]) ==TRUE ) {
    Movies_Clean$Oscar_nominations[i] <- gsub("Oscar.*","",Movies_Clean$awards[i])
  } else {
    Movies_Clean$Oscar_nominations[i] <-0
  }
}
for (i in 1:nrow(Movies_Clean)) {
  if(grepl("Won.*",Movies_Clean$Oscar_nominations[i]) ==TRUE) {
    Movies_Clean$Oscar_nominations[i] <- (gsub("[0-9^]","",
                                          Movies_Clean$Oscar_nominations[i]))
  }
}
#extract the number of awards
Movies_Clean$Oscar_nominations <-gsub("[a-z,A-Z]","",Movies_Clean$Oscar_nominations)
```

```r
## make blank records Na and convert to int
Movies_Clean$Oscar_nominations <- str_trim(Movies_Clean$Oscar_nominations)
Movies_Clean$Oscar_nominations <-as.integer(gsub("^$|^ $",0,Movies_Clean$Oscar_nominations))
```

other wins

```r
##other wins
## dont need to put this in  a for loop. do same as for other noms
for (i in 1:nrow(Movies_Clean)) {
  Movies_Clean$other_wins[i] <- gsub(" wins.* | .*Another ","",Movies_Clean$awards)[i]
  Movies_Clean$other_wins[i] <-(gsub("[^0-9]","",Movies_Clean$other_wins[i]))

}

## change blank values and NA to 0
Movies_Clean$other_wins <- gsub("^$|^ $",0,Movies_Clean$other_wins)
## convert to int
Movies_Clean$other_wins <- as.integer(Movies_Clean$other_wins)
```

other nominations

```r
##other nominations
Movies_Clean$other_nominations <- gsub(".*win","",Movies_Clean$awards)
Movies_Clean$other_nominations <- as.integer(gsub("[^0-9]","",
                                                 Movies_Clean$other_nominations))
## change NA values to 0.
Movies_Clean$other_nominations[is.na(Movies_Clean$other_nominations)] <- 0
```

get rid of the NA values and replace with 0s.

```r
## some rows have no data for the awards. for these set all
## rewards columns to 0
Movies_Clean$Oscar_wins[is.na(Movies_Clean$Oscar_wins)] <-0
```

trim synopsis

```r
Movies_Clean$synopsis <- str_trim(Movies_Clean$synopsis)
```

##clean run time

```r
Movies_Clean$run_time <- as.integer(gsub("[^0-9]","",
                                        Movies_Clean$run_time))
```

clean genres, this mainly involes getting rid of the NAs. for this ill just set genre 2 equal to genre 1 when no genre 2 is provided.

```
Movies_Clean$genre2[is.na(Movies_Clean$genre2)] <- Movies_Clean$genre1[is.na(Movies_Clean$genre2)]
```

clean cast, this will involve seperating each of the 5 actors for each movie into their own column, this will make analysis much easier.

```
## the first order of business is to split the current vector (length 1) into a vector of lenngth 5, i.


Movies_Clean$cast <- strsplit(Movies_Clean$cast,",")

## now each of the 5 actors in a movie can be called individually, this means we can simply iterate thr

actor_split <- function(){
  for (actor in 1:5){
    col_name <- paste0("actor",actor)
    Movies_Clean[,col_name] <-NA
    for (i in 1:nrow(Movies_Clean)){

      Movies_Clean[,col_name][i] <- Movies_Clean$cast[[i]][actor]
    }
  }
  return(Movies_Clean)
}

Movies_Clean <-actor_split()
```

Remove old columns.

```
Movies_Clean <- Movies_Clean %>%
  select(-gross_world,-opening_weekend,
         -budget,-awards,-synopsis,-cast)
```

rearrange columns to be in a more intuitive order.

```
Movies_Clean <-
  Movies_Clean %>%
  select(name,year,director,actor1,actor2,actor3,actor4,actor5,imdb_votes,run_time,genre1,genre2,budget
```

```


summary(Movies_Clean)
```

final checks to see if everything looks okay

```
##      name                year        director            actor1
##  Length:3400        Min.   :1970   Length:3400        Length:3400
##  Class :character   1st Qu.:1985   Class :character   Class :character
##  Mode  :character   Median :1995   Mode  :character   Mode  :character
##                      Mean   :1995
##                      3rd Qu.:2005
##                      Max.   :2021
##
##     actor2             actor3             actor4             actor5
##  Length:3400        Length:3400        Length:3400        Length:3400
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##    imdb_votes         run_time        genre1             genre2
##  Min.   :     35   Min.   : 23.0   Length:3400        Length:3400
##  1st Qu.:   9160   1st Qu.: 93.0   Class :character   Class :character
##  Median :  36265   Median :102.0   Mode  :character   Mode  :character
##  Mean   : 114820   Mean   :106.6
##  3rd Qu.: 119046   3rd Qu.:116.0
##  Max.   :2504638   Max.   :271.0
##  NA's   :2         NA's   :2
##    budget_USD         openning_USD         gross_USD           Oscar_wins
##  Min.   :    17399   Min.   :3.200e+01   Min.   :1.185e+05   Min.   : 0.0000
##  1st Qu.: 20592569   1st Qu.:7.060e+06   1st Qu.:1.004e+08   1st Qu.: 0.0000
##  Median : 45541915   Median :1.911e+07   Median :2.117e+08   Median : 0.0000
##  Mean   : 66656390   Mean   :5.138e+07   Mean   :3.384e+08   Mean   : 0.1388
##  3rd Qu.: 91759544   3rd Qu.:4.247e+07   3rd Qu.:4.428e+08   3rd Qu.: 0.0000
##  Max.   :387040011   Max.   :1.658e+09   Max.   :3.556e+09   Max.   :11.0000
##  NA's   :1379        NA's   :1379        NA's   :2288
##  Oscar_nominations   other_wins      other_nominations     rating
##  Min.   : 0.0000   Min.   :  0.00   Min.   :  0.00    Min.   :2.600
##  1st Qu.: 0.0000   1st Qu.:  2.00   1st Qu.:  1.00    1st Qu.:5.700
##  Median : 0.0000   Median :  4.00   Median :  3.00    Median :6.400
##  Mean   : 0.2174   Mean   : 11.52   Mean   : 10.65    Mean   :6.338
##  3rd Qu.: 0.0000   3rd Qu.: 12.00   3rd Qu.: 10.00    3rd Qu.:7.000
##  Max.   :10.0000   Max.   :303.00   Max.   :376.00    Max.   :9.300
##                    NA's   :656
```

potential issues: * min IMDb_votes is 35, i set a filter to only get films with >2000 votes. * min run time is 23 minutes, there should only be feature length movies. *the minimum budget is $35.

lets look at these one at a time.

```
(Movies_Clean[Movies_Clean$imdb_votes==35,])
```

```
##      name year       director             actor1        actor2        actor3
## NA   <NA>   NA           <NA>               <NA>          <NA>          <NA>
## 419  Luca 2021 Kôzô Morishita Taichirô Hirokawa Eiko Masuyama Kei'ichi Noda
## NA.1 <NA>   NA           <NA>               <NA>          <NA>          <NA>
##              actor4         actor5 imdb_votes run_time      genre1    genre2
## NA             <NA>           <NA>         NA       NA        <NA>      <NA>
## 419  Ken'ichi Ogata Kazuhiko Inoue         35       23 16 Jan 1979 Adventure
## NA.1           <NA>           <NA>         NA       NA        <NA>      <NA>
##      budget_USD openning_USD gross_USD Oscar_wins Oscar_nominations other_wins
## NA           NA           NA        NA         NA                NA         NA
## 419          NA           NA        NA          0                 0          0
## NA.1         NA           NA        NA         NA                NA         NA
##      other_nominations rating
## NA                  NA     NA
## 419                  0    7.5
## NA.1                NA     NA
```

```
Movies_Clean <- Movies_Clean[-c(419),]
Movies_Clean[419,]
```

```
##                    name year     director         actor1       actor2
## 470 Prince of the City 1981 Sidney Lumet Treat Williams Jerry Orbach
##              actor3      actor4       actor5 imdb_votes run_time genre1 genre2
## 470 Richard Foronjy Don Billett Kenny Marino       8022      167  Crime  Drama
##     budget_USD openning_USD gross_USD Oscar_wins Oscar_nominations other_wins
## 470   26296510     197875.1        NA          0                 1          4
##     other_nominations rating
## 470                14    7.5
```

the Movie that only has 35 votes is also the movie thats only 23minutes long. its an epsiode of a series, not sure how it ended there but we can just remove it.

lets see if that has fixed the issue. now the minimum votes and minimum runtime make a lot more sense.

```
summary(Movies_Clean)
```

```
##      name                year        director            actor1
##  Length:3399        Min.   :1970   Length:3399        Length:3399
##  Class :character   1st Qu.:1985   Class :character   Class :character
##  Mode  :character   Median :1995   Mode  :character   Mode  :character
##                     Mean   :1995
##                     3rd Qu.:2005
##                     Max.   :2021
##
##     actor2             actor3             actor4             actor5
##  Length:3399        Length:3399        Length:3399        Length:3399
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##    imdb_votes         run_time        genre1             genre2
##  Min.   :     35   Min.   : 23.0   Length:3399        Length:3399
##  1st Qu.:   9147   1st Qu.: 93.0   Class :character   Class :character
##  Median :  36257   Median :102.0   Mode  :character   Mode  :character
##  Mean   : 114796   Mean   :106.6
##  3rd Qu.: 119005   3rd Qu.:116.0
##  Max.   :2504638   Max.   :271.0
##  NA's   :2         NA's   :2
##    budget_USD        openning_USD         gross_USD           Oscar_wins
##  Min.   :    17399   Min.   :3.200e+01   Min.   :1.185e+05   Min.   : 0.0000
##  1st Qu.: 20597872   1st Qu.:7.047e+06   1st Qu.:1.004e+08   1st Qu.: 0.0000
##  Median : 45553580   Median :1.911e+07   Median :2.117e+08   Median : 0.0000
##  Mean   : 66688859   Mean   :5.136e+07   Mean   :3.384e+08   Mean   : 0.1389
##  3rd Qu.: 91820581   3rd Qu.:4.246e+07   3rd Qu.:4.428e+08   3rd Qu.: 0.0000
##  Max.   :387040011   Max.   :1.658e+09   Max.   :3.556e+09   Max.   :11.0000
##  NA's   :1379        NA's   :1379        NA's   :2287
##  Oscar_nominations   other_wins     other_nominations     rating
##  Min.   : 0.0000   Min.   :  0.00   Min.   :  0.00    Min.   :2.600
##  1st Qu.: 0.0000   1st Qu.:  2.00   1st Qu.:  1.00    1st Qu.:5.700
##  Median : 0.0000   Median :  4.00   Median :  3.00    Median :6.400
##  Mean   : 0.2174   Mean   : 11.52   Mean   : 10.65    Mean   :6.337
##  3rd Qu.: 0.0000   3rd Qu.: 12.00   3rd Qu.: 10.00    3rd Qu.:7.000
##  Max.   :10.0000   Max.   :303.00   Max.   :376.00    Max.   :9.300
##                    NA's   :656
```

the dataframe is good to go, with 3006 movies.

```
write.csv(Movies_Clean,"Movies_Clean.csv",row.names = FALSE)
```

EDA, Feature engineering, and modelling.

```
Movies <- read.csv("Movies_Clean.csv")

## one of the Movies seems to have imported incorrectly. for now we're going to remove it.
summary(Movies)
```

```
##      name                year        director            actor1
##  Length:3399        Min.   :1970   Length:3399        Length:3399
##  Class :character   1st Qu.:1985   Class :character   Class :character
##  Mode  :character   Median :1995   Mode  :character   Mode  :character
##                     Mean   :1995
##                     3rd Qu.:2005
##                     Max.   :2021
##
##     actor2             actor3             actor4             actor5
##  Length:3399        Length:3399        Length:3399        Length:3399
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##    imdb_votes         run_time        genre1             genre2
##  Min.   :     35   Min.   : 23.0   Length:3399        Length:3399
##  1st Qu.:   9147   1st Qu.: 93.0   Class :character   Class :character
##  Median :  36257   Median :102.0   Mode  :character   Mode  :character
##  Mean   : 114796   Mean   :106.6
##  3rd Qu.: 119005   3rd Qu.:116.0
##  Max.   :2504638   Max.   :271.0
##  NA's   :2         NA's   :2
##    budget_USD         openning_USD         gross_USD           Oscar_wins
##  Min.   :    17399   Min.   :3.200e+01   Min.   :1.185e+05   Min.   : 0.0000
##  1st Qu.: 20597872   1st Qu.:7.047e+06   1st Qu.:1.004e+08   1st Qu.: 0.0000
##  Median : 45553580   Median :1.911e+07   Median :2.117e+08   Median : 0.0000
##  Mean   : 66688859   Mean   :5.136e+07   Mean   :3.384e+08   Mean   : 0.1389
##  3rd Qu.: 91820581   3rd Qu.:4.246e+07   3rd Qu.:4.428e+08   3rd Qu.: 0.0000
##  Max.   :387040011   Max.   :1.658e+09   Max.   :3.556e+09   Max.   :11.0000
##  NA's   :1379        NA's   :1379        NA's   :2287
##  Oscar_nominations   other_wins     other_nominations     rating
##  Min.   : 0.0000   Min.   :  0.00   Min.   :  0.00    Min.   :2.600
##  1st Qu.: 0.0000   1st Qu.:  2.00   1st Qu.:  1.00    1st Qu.:5.700
##  Median : 0.0000   Median :  4.00   Median :  3.00    Median :6.400
##  Mean   : 0.2174   Mean   : 11.52   Mean   : 10.65    Mean   :6.337
##  3rd Qu.: 0.0000   3rd Qu.: 12.00   3rd Qu.: 10.00    3rd Qu.:7.000
##  Max.   :10.0000   Max.   :303.00   Max.   :376.00    Max.   :9.300
##                    NA's   :656
```

first we need to do a little clean up, it seems like there are some rows that havnt been properly formated. For now, since there are only 3 im just going to remove them.

```
## remove rows that have incorrect formating
Movies <- Movies[c(-3366,-82,-339,-369),]


## set other wins NAs to 0.

Movies$other_wins[is.na(Movies$other_wins)] <-0
```

```
set.seed(123)
pct <-0.8
df <- sample(nrow(Movies),nrow(Movies)*pct,replace=FALSE)

train <-Movies[df,]
test <-Movies[-df,]

paste("There are",nrow(train),"samples in the training data set, and",nrow(test),"in the test data set.
      "split.")
```

first lets split our data into a training and test set.

```
## [1] "There are 2716 samples in the training data set, and 679 in the test data set.   This is a  80
```

for feature engineering purposes, ill combine the two back together. But it is important to note that the data in the test data frame *will not* be used in any of the analysis, or in creating new features or predictive models.

```
## First, we will store the rating scores for the test data to use at the end. we can then remove it, a
test_ratings <- test$rating
```

```
test$rating <-NA

all <- rbind(train,test)
glimpse(all)
```

```
## Rows: 3,395
## Columns: 20
## $ name              <chr> "Bulletproof", "Afternoon Delight", "The Best Man", ~
## $ year              <int> 1996, 2013, 2005, 2002, 2006, 1980, 1997, 2003, 2018~
## $ director          <chr> "Ernest R. Dickerson", "Joey Soloway", "Stefan Schwa~
## $ actor1            <chr> "Damon Wayans", "Kathryn Hahn", "Stuart Townsend", "~
## $ actor2            <chr> "Adam Sandler", "Link Ruiz", "Raphael Schwartz", "Ma~
## $ actor3            <chr> "James Caan", "Cesar Garcia", "Jacob Collier", "Pete~
## $ actor4            <chr> "Jeep Swenson", "Jane Lynch", "Callum Williams", "Da~
## $ actor5            <chr> "James Farentino", "Michaela Watkins", "Seth Green",~
## $ imdb_votes        <int> 37761, 10372, 3757, 206070, 459056, 9142, 8712, 4760~
## $ run_time          <int> 84, 98, 96, 101, 101, 88, 129, 129, 120, 104, 119, 7~
## $ genre1            <chr> "Action", "Comedy", "Comedy", "Drama", "Comedy", "Ad~
## $ genre2            <chr> "Comedy", "Drama", "Romance", "Romance", "Drama", "H~
## $ budget_USD        <dbl> 44287444, NA, NA, 16995153, 11029683, NA, NA, 226588~
## $ openning_USD      <dbl> 10654496, NA, NA, 18814388, 511201, NA, NA, 73225935~
## $ gross_USD         <dbl> NA, NA, NA, NA, 138592347, NA, NA, 645539212, NA, 63~
## $ Oscar_wins        <int> 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Oscar_nominations <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0~
## $ other_wins        <dbl> 2, 13, 0, 4, 70, 5, 6, 5, 0, 7, 0, 1, 8, 0, 0, 0, 5,~
## $ other_nominations <int> 2, 3, 0, 2, 112, 5, 5, 36, 0, 40, 0, 1, 22, 0, 0, 0,~
## $ rating            <dbl> 5.8, 5.7, 6.0, 7.3, 7.8, 5.2, 6.3, 6.8, 3.2, 6.7, 6.~
```

lets do a quick check of na values

```
nulcols <-all %>%
  select(-rating) %>%
  sapply(.,function(x) sum(is.na(x))) %>%
  data.frame() %>%
    rownames_to_column(var="Categories")

colnames(nulcols) <- c( "Categories","Count")

nulcols %>%
  filter(Count > 0)
```

```
##     Categories Count
## 1   budget_USD  1375
## 2 openning_USD  1375
## 3    gross_USD  2283
```

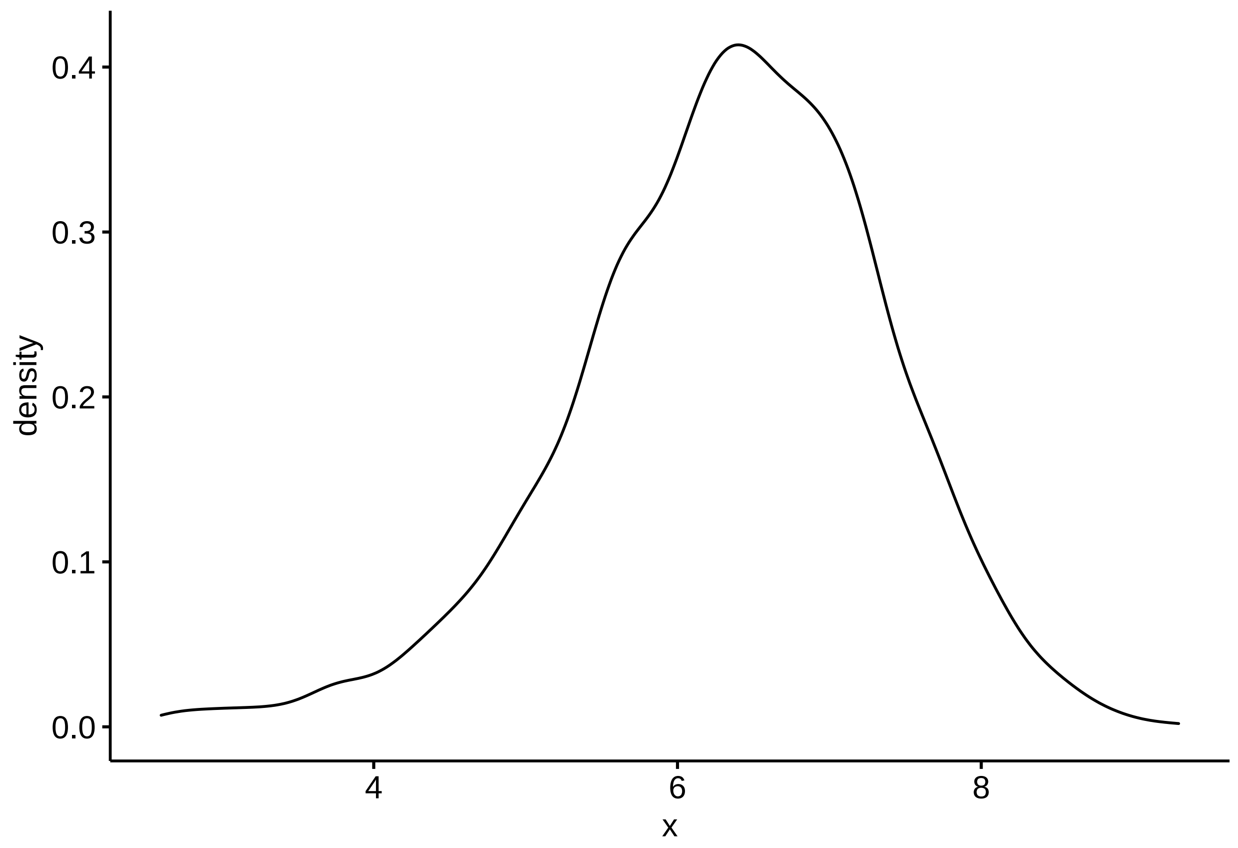
There are only three columns with na, and its the three we expect.(excluding rating)

Exploratory data analysis

In this section we will explore the data a bit more, see how the predictor and outcome variables are related, and find areas that we can work on and improve in the feature engineering section.

To start lets see if our dependent variable is normally distributed, this is an essential part that will dictate whether or not our predictive models will work.

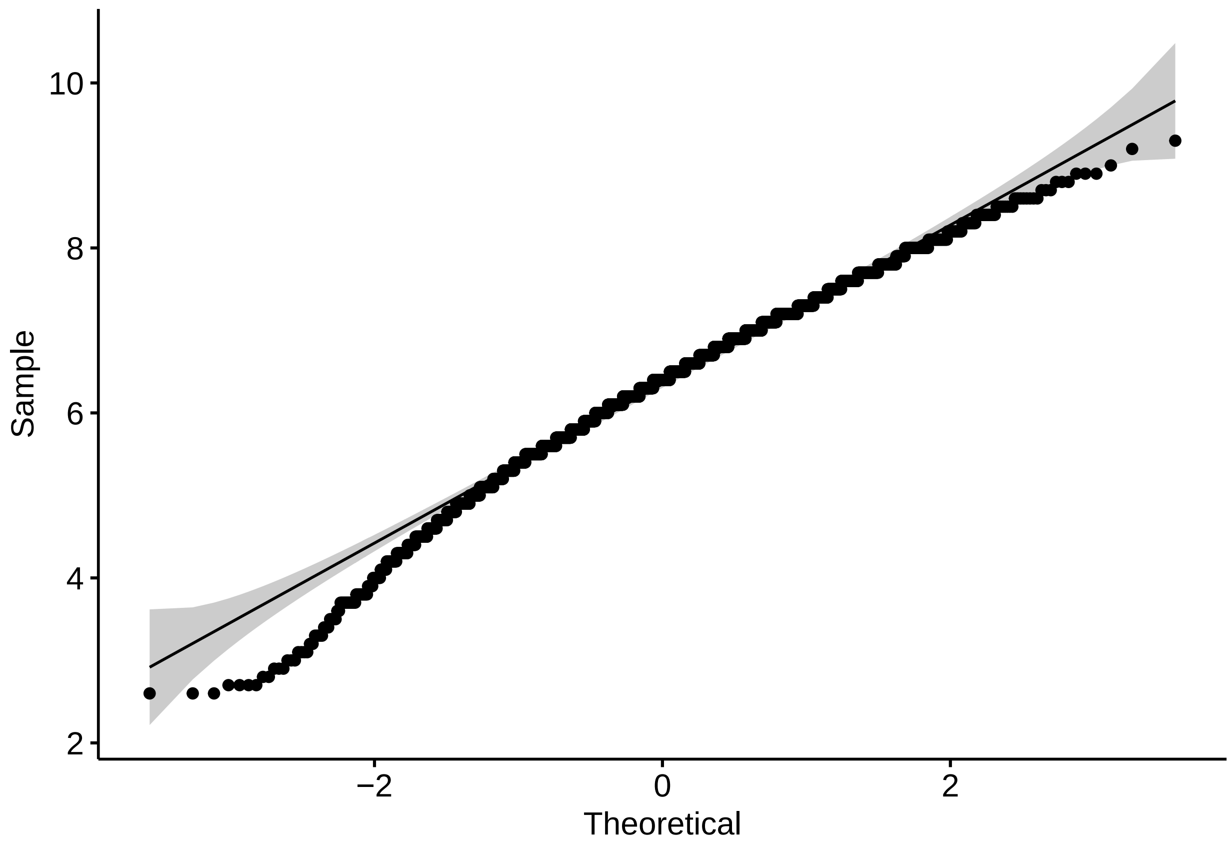
```
ggdensity(all$rating,na.rm=TRUE)
```



As can been in the density plot, the data is fairly normal by it seems to be slightly skewed with a longer tail on the left. This can be seen more easily by reviewing a qqplot.

```
ggqqplot(all$rating,)
```

```
## Warning: Removed 679 rows containing non-finite values (stat_qq).

## Warning: Removed 679 rows containing non-finite values (stat_qq_line).

## Warning: Removed 679 rows containing non-finite values (stat_qq_line).
```



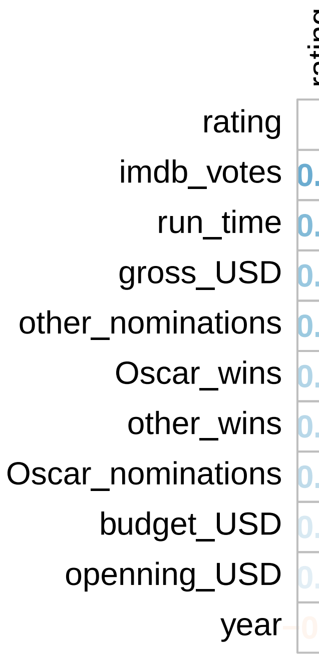
```
skew(all$rating)
```

```
## [1] -0.4795309
```

As could be seen, the data is slightly skewed to the left, this will of course have some effect on metrics that estimate location (i.e median and mean), although the skew is small enough for us to assume normal distribution and continue without transformation of the outcome variable.

```
## lets start of by getting just the numeric columns.
numeric_var_names <- which(sapply(all, is.numeric))

numeric_vars <- all[,numeric_var_names]

## find the correlation between all variables, which allows us to create a correlation matrix
cor_numeric_vars <-cor(numeric_vars,use="pairwise.complete.obs") ##pairwise.complete.obs used to take c


## now we can sort by rating, simply to give us an order so our correlation matrix is easier to read.
cor_sorted <- as.matrix(sort(cor_numeric_vars[,"rating"],decreasing = TRUE))

Cornames <- names(which(apply(cor_sorted, 1, function(x) abs(x)>0)))
```

```
cor_numeric <-cor_numeric_vars[Cornames,Cornames]

corrplot.mixed(cor_numeric, tl.col="black", tl.pos = "lt")
```

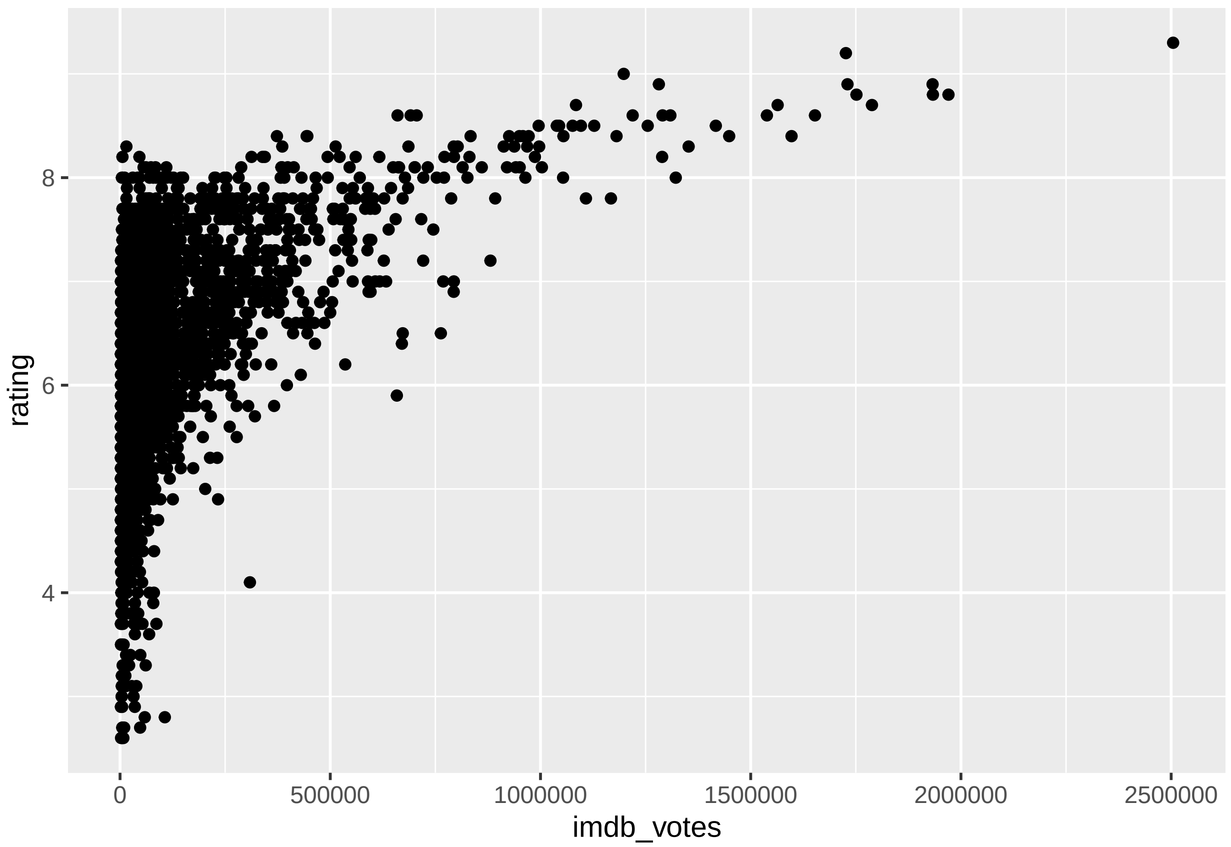


Lets take an initial look at how our numeric variables correlate to rating.
It's clear from the above plot that none of the numeric predictors as they currently stand are particularly amazing. many of these will improve dramatically after some feature engineering. This correlation matrix also gives us a good initial indication as to whether or not we would run into multicolinearity issues, which can cause problems from some algorithms (e.g ones that are related to linear regression)

No two predictors correlate that well with each other, meaning none are redundant and therefore we shouldn't have issues with multiolinearity.

```
all %>%
  filter(!is.na(rating)) %>%
  ggplot(aes(imdb_votes,rating)) +
  geom_point()
```

lets take a look at our outcome variable a little more, aswell as our current best numeric predic-



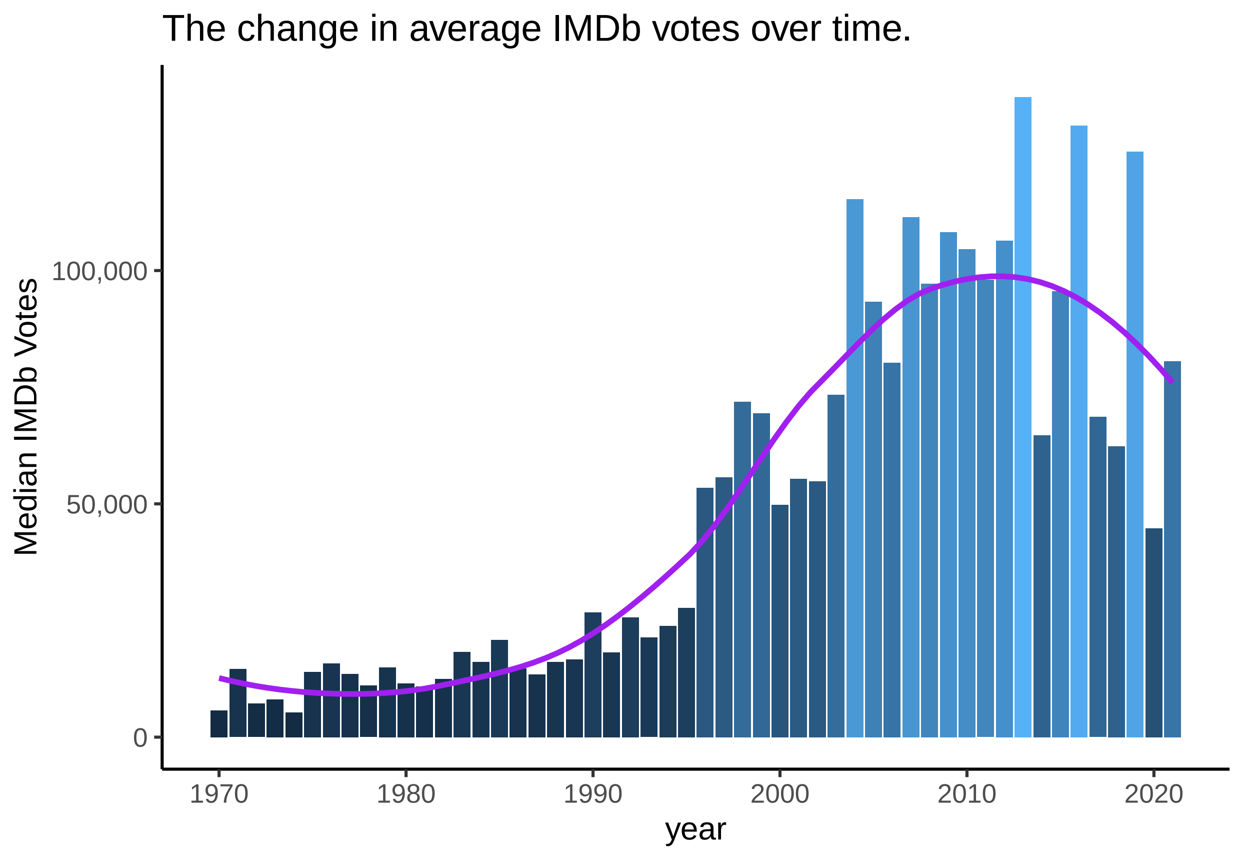
tors.

here we can see there is a hugely different magnitude between the smallest and largest values, taking the log of this predictor should increase correlation, but this is something that will be dealt with later.

IMDb votes exploration lets see if year effects imdb_votes effect on rating (it could be possible that old movies i.e from the 70s would have less votes)

```
VotesVTime <-all %>%
  filter(!is.na(rating),) %>%
  group_by(year) %>%
  summarise(avg.imdb_votes = median(imdb_votes),mean.imdb_votes = mean(imdb_votes))

VotesVTime %>%
  ggplot(aes(year,avg.imdb_votes,fill=avg.imdb_votes)) +
  geom_col() +
  theme_classic2() +
  theme(legend.position = 0) +
  scale_y_continuous(labels = comma) +
  labs(y="Median IMDb Votes", title="The change in average IMDb votes over time.") +
  geom_smooth(se=FALSE,colour="purple")+
    scale_fill_gradient()
```

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



Clearly, older movies inherently have fewer votes irrelevent of their rating, giving older movies a disadvantage. This is something that should be addressed in the feature engineering section.

Non numeric variables Before we start feature engineering lets see if we can learn anything for the non-numeric variables

director first ill create a dataframe that shows the median rating of all directors Movies, aswell as the number of Movies they have directed in the training data set.

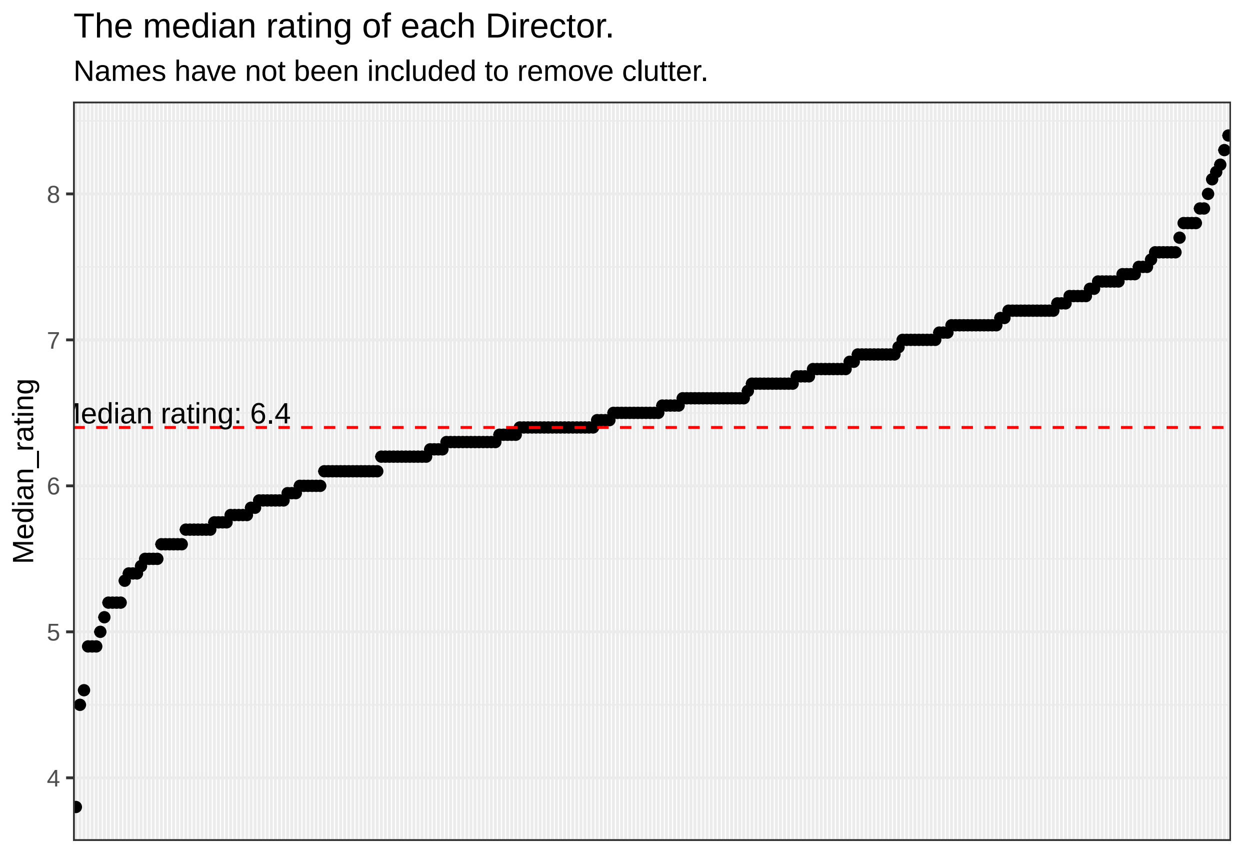
```
director.df <-all %>%
  filter(!is.na(rating)) %>%
  group_by(director) %>%
  summarise("Median_rating"=median(rating),count=n()) %>%
  arrange(desc(Median_rating)) %>%
  filter(count >2)




head(director.df)
```

```
## # A tibble: 6 x 3
##   director          Median_rating count
##   <chr>                     <dbl> <int>
## 1 Christopher Nolan           8.4     5
## 2 Stanley Kubrick            8.3      3
## 3 Martin Scorsese            8.2     10
## 4 Quentin Tarantino          8.15     8
## 5 Pete Docter                8.1      4
## 6 Lars von Trier             8        3
```

An issue arises, a bunch of directors only have 1 data point in the training data set.We will have to take this into account when creating our features.

Despite this, the director very clearly, and rather intuitively effects a movies rating substantially

```
## median rating variable
median.rating <- all %>%
  filter(!is.na(rating)) %>%
  summarise(median(rating)) %>%
  .[1,1]

director.df %>%
  ggplot(aes(reorder(director,Median_rating),Median_rating)) +
  geom_point() +
  geom_hline(yintercept = median.rating, colour="red",linetype="dashed") +
  theme_bw()+
  geom_text(aes(x=25,y=6.5),label=paste("Median rating:",median.rating),colour="black",check_overlap = 
  theme(axis.ticks.x = element_blank(),
        axis.text.x = element_blank(),
        axis.title.x = element_blank()) +
  labs(title = "The median rating of each Director.",
       subtitle = "Names have not been included to remove clutter.")
```



Feature Engineering

In this section we will go through our predictor variables one by one and see how they can be manipulated, combined, or used to create new variables to improve predictability of our final model.

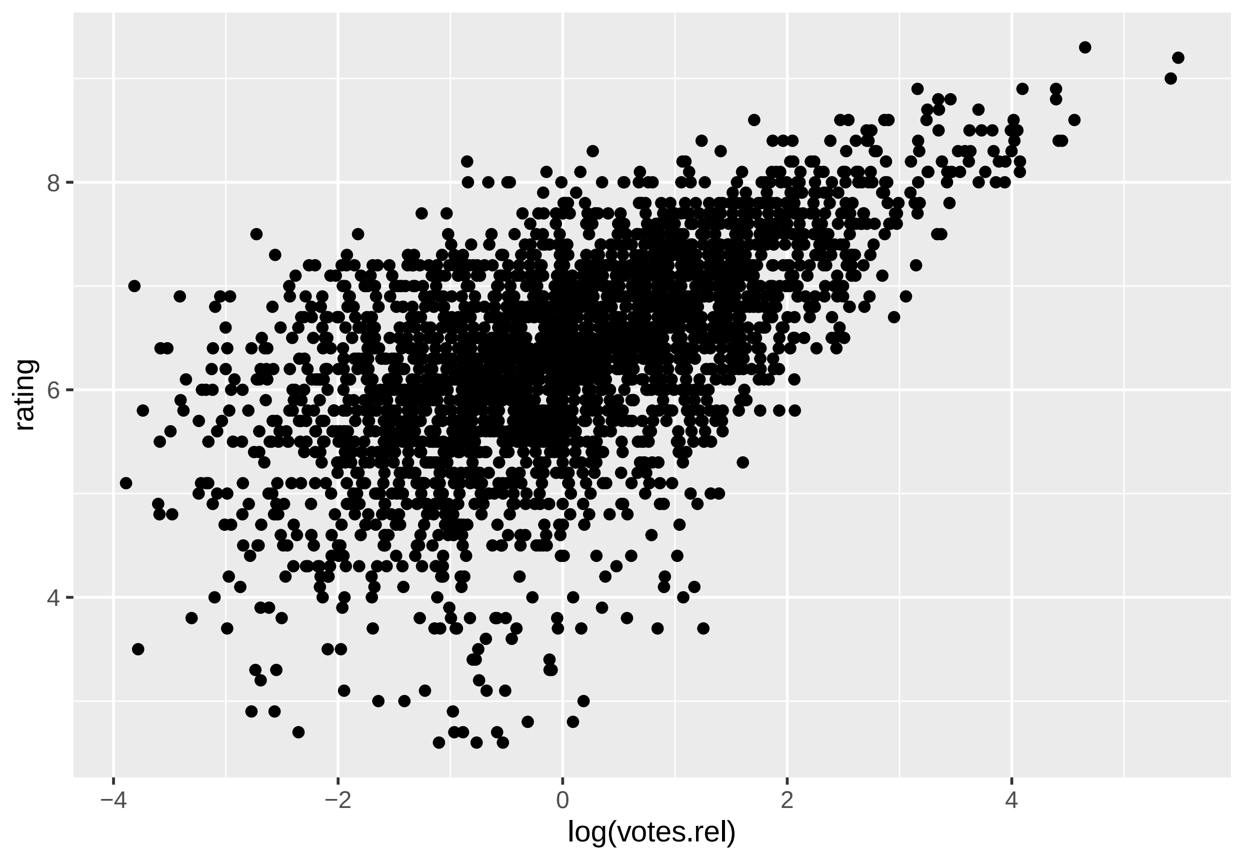
imdb_votes As was seen in the previous section year greatly influences IMDb votes, resulting in votes not being a very viable predictor, intuitively this seems odd, you would think a movie with more votes would see higher ratings. We have to account for the fact that IMDb didnt exist for a large portion of these movies, and newer, younger audiences are probably more likely to be involved with a website like this.

To start, we will see how creating a new variable that looks at imdb votes *relative* to those in the same year effects its predictive power.

```
for (i in 1:nrow(all)) {
  all$votes.rel[i] <- all$imdb_votes[i]/ VotesVTime$avg.imdb_votes[VotesVTime$year ==all$year[i]]
}
```

lets see how our new feature correlates with rating.

```
all %>%
  filter(!is.na(rating)) %>%
  ggplot(aes(log(votes.rel),rating)) +
  geom_point()
```



```
cor(all$rating,log(all$votes.rel),use="pairwise.complete.obs")
```

```
## [1] 0.5941181
```

Director A reasonable way of improving this particular variable is to convert it to ordinal data, where directors are categorized based on their average Movie rating. If a director isn't in the training data ill just put them in the central group. (note: to improve this, it could be worth looking at the most common genre of movie these directors make and use that to choose what group they go in. i.e horror Movie directors will typically have lower ratings than drama directors.)

```
## the quantiles we want to use to split the groups
director.quantiles <- c(0.05,0.15,0.35,0.65,0.75,0.85,0.95)

## setting the value of the highest ranked group
director.df$director_rating <- length(director.quantiles) +1

## setting the value for all subsequent categories

for (i in 1:length(director.quantiles)) {
  director.df$director_rating[!director.df$director %in% head(director.df$director, floor(nrow(director
}
```

Now to create a column in all that corresponds to these ratings.

```
for (i in 1:nrow(all)) {
  if (all$director[i] %in% director.df$director==TRUE){
    all$director_rating[i] <- director.df$director_rating[all$director[i] == director.df$director]
  } else {
    all$director_rating[i] <-4
  }

}
```

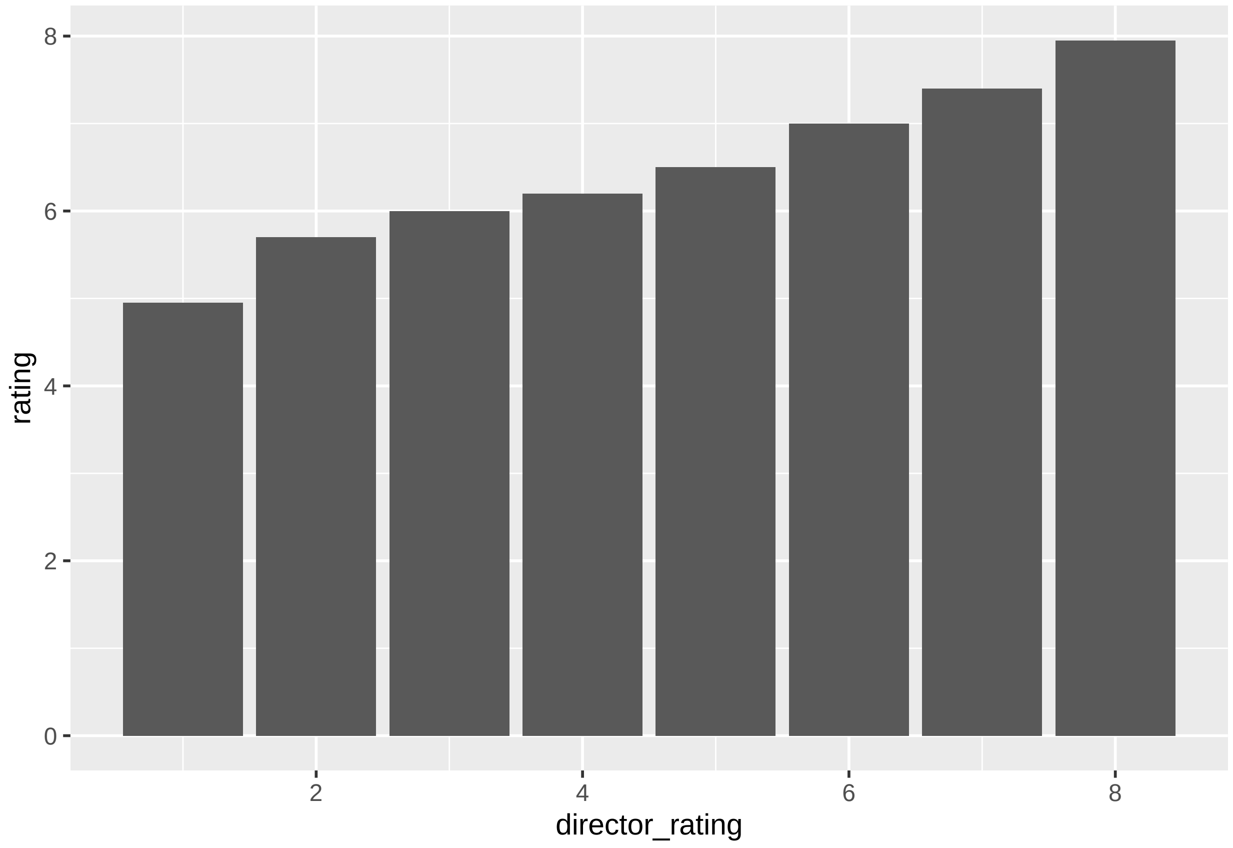
lets see how good it is and see if it needs changing.

```
cor(all$rating,all$director_rating,use="pairwise.complete.obs")
```

```
## [1] 0.470422
```

we have to take into consideration that many of the directors on here only have 1 movie, that isnt enough information to categories them correctly, it may be better to simply put these in the middle group.

```
all %>%
  filter(!is.na(rating)) %>%
  ggplot(aes(director_rating,rating)) +
  stat_summary(fun="median",geom = "bar")
```



```
cor(all$rating,all$director_rating,use="pairwise.complete.obs")
```

```
## [1] 0.470422
```

actors I think it makes sense to treat actors in much the same way as directors, of course here we have 5 actor columns, ill simply do the same as above for each of the 5 and average out the resultant value. It may be worth applying a weight to actors that are listed earlier (i.e the top credited actor), but we can experiment and see if that's needed.

We need to start by assigning each actor to a category

```
##A function to assign a score to each actor in the dataframe (including actors from all 5 columns.)

actor.score <- function() {
  actor.df <-data.frame()
  for (i in 1:5){                  ##iterate through the 5 actor columns
    col_name <- paste0("actor",i)
    med_rating <- all %>%  ##get the medium rating for actors in a column
      filter(!is.na(rating)) %>%
      group_by(all %>% filter(!is.na(rating)) %>% .[,col_name]) %>%
      summarise(med.rating=median(rating))
    actor.df <- rbind(actor.df,med_rating) ## put all the results into a dataframe


  }
  names(actor.df)[1]<-"Actor"

  actor.df <- actor.df %>%   ## get median rating for each unique actor.
    group_by(Actor) %>%
    summarise(rating =median(med.rating),count = n()) %>%
    arrange(desc(rating)) %>%
    filter(count>2) ### we only want actors that are in 3 or more movies. We cant use a single movie to

  return(actor.df)

}

actor.df <- actor.score()
```

Now we need to apply the actor scores to actors in our data frame, and assign and overall score for each movie based on the 5 top credited actors.

```
all$actor_score <-0
for (i in 1:5) {
  col_name <- paste0("actor",i)
  for (j in 1:nrow(all)) {
    if(all[,col_name][j] %in% actor.df$Actor) {
      all$actor_score[j] <- all$actor_score[j] +actor.df$rating[actor.df$Actor ==all[,col_name][j]]
    } else{
      all$actor_score[j] <-all$actor_score[j] +median(actor.df$rating)
    }
  }
}
```

There are a bunch of actors in the test set that arnt present in the training set. This means they dont have an associated value with them. For now, i have assigned a central value to these actors. *I will need to come back and find an appropriate way to assign values to these. for the time being though it will have to do.*

```
cor(all$actor_score,all$rating,use = "pairwise.complete.obs")
```

```
## [1] 0.4500487
```

as we can see the correlation in the training data is very high. in reality this is just because it is over fit to the training data. this is something i will have to comeback to and address. cross validation may be needed.

awards as it stands the awards variables dont offer much in terms of predictive power, i think we can solve that by combining the three variables, applying different weights to each caregory.

```
all$award_score <- all$Oscar_wins +all$Oscar_nominations +all$other_wins*0.05 +
  all$other_nominations*0.05

cor(all$rating,all$award_score,use = "pairwise.complete.obs")
```

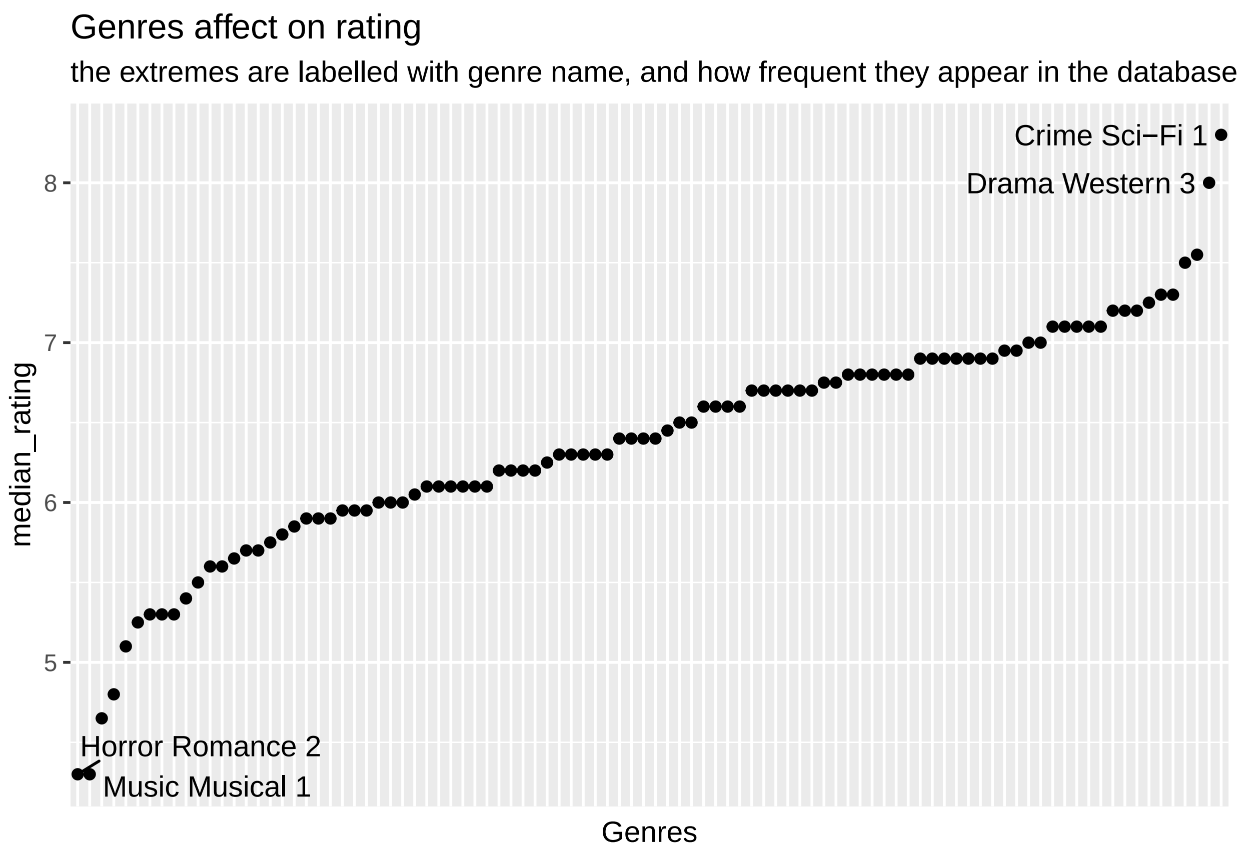
```
## [1] 0.4242807
```

```
median_ratings_genre1 <-all %>%
  filter(!is.na(rating)) %>%
  group_by(genre1) %>%
  summarise(median_rating = median(rating),count=n()) %>%
  arrange(desc(median_rating))
```

```
median_ratings_genre2 <-all %>%
  filter(!is.na(rating)) %>%
  group_by(genre2) %>%
  summarise(median_rating = median(rating),count=n()) %>%
  arrange(desc(median_rating))
```

genre Bother genre1 and genre 2 seem to have an effect on rating, we may however see a better result if we combine the two together

```
all$genres <- paste(all$genre1, all$genre2)

median_ratings_genres <-all %>%
  filter(!is.na(rating)) %>%
  group_by(genres) %>%
  summarise(median_rating = median(rating),count=n()) %>%
  arrange(desc(median_rating))

median_ratings_genres %>%
  ggplot(aes(reorder(genres,median_rating),median_rating)) +
  geom_point() +
  theme(axis.ticks.x = element_blank(),
        axis.text.x = element_blank()) +
  labs(x="Genres",
       title="Genres affect on rating",
       subtitle = "the extremes are labelled with genre name, and how frequent they appear in the databa
  geom_text_repel(aes(label=ifelse(median_rating >7.7 |median_rating <4.5,
                                   paste(genres,
                                         count)
                                   ,"")))
```



```
## set them to be factor variables (for now, i may change this.)

all$genre1 <- as.factor(all$genre1)
all$genre2 <- as.factor(all$genre2)
```

again, there's a clear trend here and it seems that some genres are for sure better in terms of getting a higher rating, however, many of the categoires only have a few data points in them. Because of this, it would be better to more broadly categories them, or remove genres that have few data points when creating dummy variables.

opening, budget, and gross

```
nulcols <-all %>%
  select(-rating,gross_USD,budget_USD,openning_USD) %>%
  sapply(.,function(x) sum(is.na(x))) %>%
  data.frame() %>%
    rownames_to_column(var="Categories")

colnames(nulcols) <- c( "Categories","NA_Count")

nulcols %>%
  filter(NA_Count > 0)
```

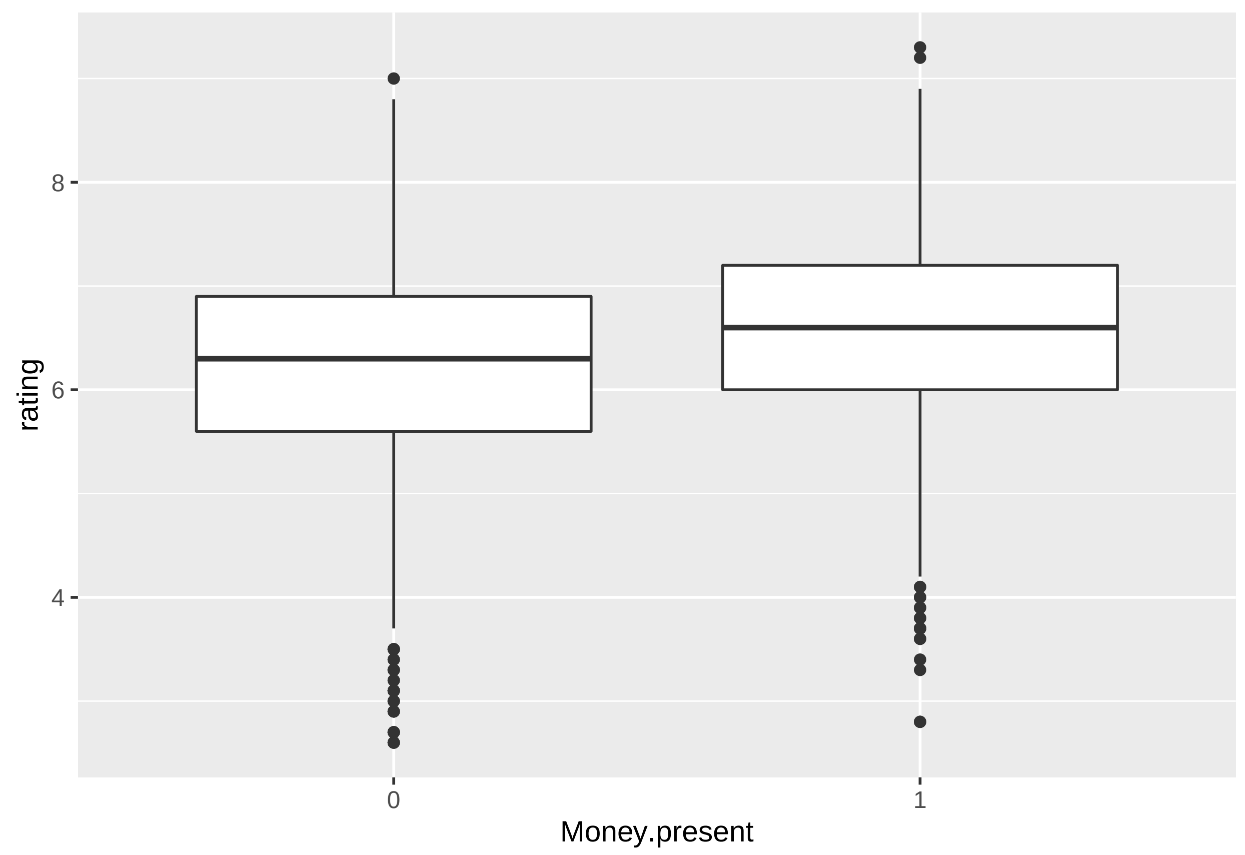
```
##     Categories NA_Count
## 1   budget_USD     1375
## 2 openning_USD     1375
## 3    gross_USD     2283
```

As we can see, over half of all the records have an NA value in atleast one of the three columns, because of this itll be hard to use these variables without going back and filling in the missing data.

For now im going to use them to create a new factor variable, with the idea that Movies that have a recorded budget, oppenning, and gross are more likely to have been succesful movies, and thus higher rated.

```
all$Money.present <- as.factor(ifelse(is.na(all$gross_USD +all$budget_USD +all$openning_USD ), 0,1))
```

```
all %>%
  filter(!is.na(rating)) %>%
  ggplot(aes(Money.present,rating)) +
  geom_boxplot()
```



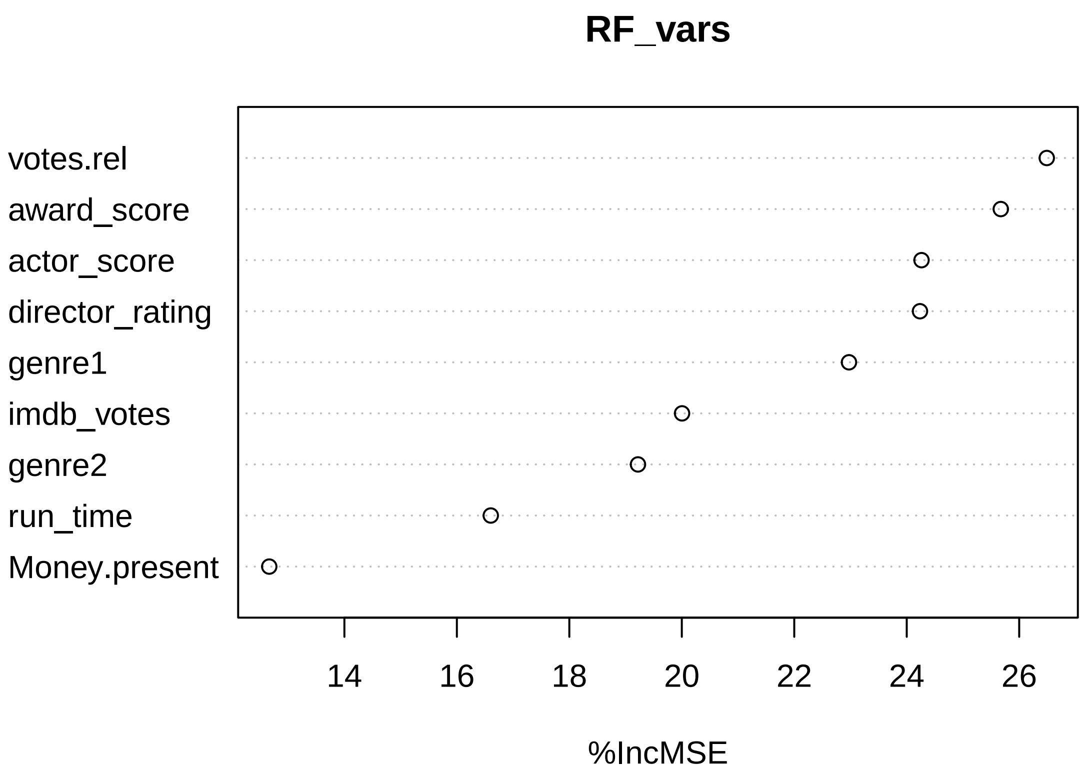
It would appear that movies that have data for all 3 variables do, on average score a higher rating. although the difference seems rather small and there's quite large variance, with several outlier points.

Random forst for identifying variable importance. First we need to clean up the dataframe and remove unwanted/un-used variables.

```
all <- all %>% select(c(-actor1,-actor2,-actor3,-actor4,-actor5,-budget_USD,
                        -openning_USD,-gross_USD,-Oscar_wins,-Oscar_nominations,-other_wins,
                        -other_nominations,-name,-director,-genres,-year))
```

since we're just using this for variable importance we can keep it simple, no need for cross validation or optimization.

```
RF_vars <-randomForest(x=all[!is.na(all$rating),-5],
             y=all$rating[!is.na(all$rating)],
             ntree=200,
             importance=TRUE)

varImpPlot(RF_vars,type = 1)
```



This essentially shows the effect of random permutation of each variable, which removes that variables predictive power, if it results in a higher MSE, then that variable has a large effect on the model and is more important of a variable.

Pre-processing the data. We're going to be utalising KNN and other algorithms that are either distance based methods or gradient descent methods, both of these require standardization and normalization of numeric variables. we also need to create dummy variables from our categorical data.

```
numericvars <- which(sapply(all,is.numeric))

numericvars.df <- all[,numericvars] %>% select(-rating)

factors.df <- all[,!names(all) %in% names(numericvars.df)] %>% select(-rating)
```

Skewness for our models to work we must assume our variables are normally distributed. an easy way to see if this assumption is true is by looking at the skewness of each variable, typically we look for a value between -0.8 and 0.8 for the assumption to be true. for values out of this range we will take the log of them which should result in a more Gaussian distribution

```
for (i in 1:ncol(numericvars.df)) {
  if (abs(skew(numericvars.df[,i]) >0.8)){
    numericvars.df[,i] <- log(numericvars.df[,i] +1)
  }
}
```

```r
PreProc <- preProcess(numericvars.df,method=c("center","scale"))
print(PreProc)
```

Normalizing the data

```
## Created from 3395 samples and 6 variables
##
## Pre-processing:
##   - centered (6)
##   - ignored (0)
##   - scaled (6)
```

```r
norm.DF <- predict(PreProc,numericvars.df)
```

```r
dummies.df <- as.data.frame(model.matrix(~.-1 ,factors.df))
```

One hot encoding of the factor variables. note: we use the -1 in the formula so we dont have an intercept value, or more accurately the intercept term is given the name its based on (since we want one hot encoding we dont need an intercept value, this would only be needed for linear regression where we would run into redundancy issues that lead to multicolinearity .)

cleaning up the dummy variables by removing those that either: are not present in the test data, or have fewer than 10 ones in the train data

```r
emptylevels.test <- which(colSums(dummies.df[(nrow(all[!is.na(all$rating),])+1):nrow(all),])==0)

## remove these levels from dummies.df

dummies.df <-dummies.df[,-emptylevels.test]
```

Now to remove levels that are either not present, or are rarely present in the training data.

```r
emptylevels.train <- which(colSums(dummies.df[1:nrow(all[!is.na(all$rating),]),])<10)

dummies.df <-dummies.df[,-emptylevels.train]
```

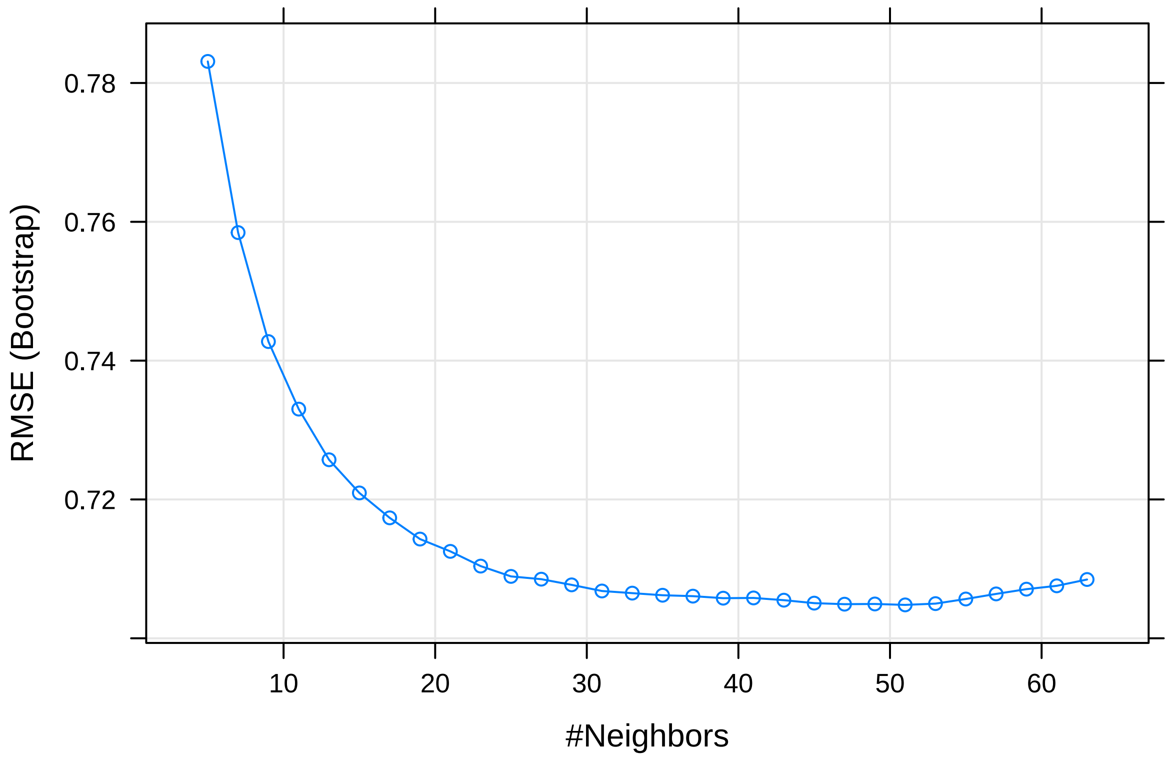
now we need to combine our numerics and dummy dataframes.

```r
all1 <- cbind(norm.DF,dummies.df)
```

knn regression predictor. Finally i'll use a basic knn model to introduce local information to the model and create one last predictor.

Here we're going to be using the train function from caret to perform cross validation and find the ideal number for k.

```r
set.seed(123)

train1 <- all1[!is.na(all$rating),]
train1$rating <- train$rating
test1 <- all1[is.na(all$rating),]

## for knn we need to seperate the outcome variable form the predictor variables

train.knn.x <-train1 %>% select(-rating)
train.knn.y <- train$rating
test.knn.x <- test1

control <-trainControl(
  method="cv",
  number =10
)

knn.model <- train(x=train.knn.x,
                   y=train.knn.y,
                   method="knn",
                   trcontrol=control,
                   tuneLength=30)
best.k <- knn.model$results$k[knn.model$results$RMSE==min(knn.model$results$RMSE)] ## select the best k


plot(knn.model)
```



here were using the best k value found in cross validation to create our actual model.

```
knnreg.fit <-knnreg(x=train.knn.x,
                    y=train.knn.y,
                   method="knn",
                   k=best.k)
knn.pred.train <- predict(knnreg.fit,train.knn.x)
mean((knn.pred.train-all$rating[!is.na(all$rating)])^2)
```

```
## [1] 0.4591016
```

```
knn.pred.test <- predict(knnreg.fit,test.knn.x)
knn.pred <- append(knn.pred.train,knn.pred.test)

all1$knn.pred <- knn.pred
```

now that we have a new predictor we need to ensure we preprocess the data once more.

```
preproc <- preProcess(all1[,c(colnames(numericvars.df),"knn.pred")],
                      method = c("center","scale"))

all1[,c(colnames(numericvars.df),"knn.pred")] <- predict(preproc,all1[,c(colnames(numericvars.df),"knn.
##  update our train and test data sets to have the knn predictor variable.

train1 <-all1[!is.na(all$rating),]

train1$rating <- all$rating[!is.na(all$rating)]

test1 <-all1[rownames(test),]
```

modelling

elastic net

```
set.seed(5435)

rownames <- sample(nrow(train1),nrow(train1)*pct,replace=FALSE)


train.net.x <- as.matrix(train1[rownames,] %>% select(-rating))
train.net.y <- train1$rating[rownames]

holdout.net.x <- as.matrix(train1[-rownames,] %>% select(-rating))
holdout.net.y <- train1$rating[-rownames]
```

```
set.seed(53)

alpha.values <- seq(0,1,0.1)
alpha.fits <- data.frame()
for (i in 1:length(alpha.values)) {
  temp.fit <-cv.glmnet(train.net.x,train.net.y,type.measure = "mse",
                      alpha=alpha.values[i],family="gaussian",nfolds = 50)
  temp.pred <-predict(temp.fit,newx=holdout.net.x,s=temp.fit$lambda.1se)
  temp.df <- data.frame(alpha=(i-1)/10,
                      MSE=mean((holdout.net.y-temp.pred)^2))
  alpha.fits <- rbind(alpha.fits,temp.df)

}
best.alpha <-alpha.fits$alpha[alpha.fits$MSE==min(alpha.fits$MSE)]

alpha.fits
```

```
##    alpha       MSE
## 1    0.0 0.4272901
## 2    0.1 0.4210080
## 3    0.2 0.4185838
## 4    0.3 0.4220874
## 5    0.4 0.4136172
## 6    0.5 0.4191295
## 7    0.6 0.4184498
## 8    0.7 0.4209818
## 9    0.8 0.4172774
## 10   0.9 0.4207417
## 11   1.0 0.4217602
```

as we can see from the above tests, an alpha value of 0.5 provides the best results. We will use this for our actual model.

```
## creating test matrices based on our test1 set (this is completely unseen/new data data)
set.seed(8312)

test.x <- as.matrix(test1)
test.y <- test_ratings


knn.fit <- cv.glmnet(train.net.x,train.net.y,type.measure = "mse",
                        alpha=best.alpha,family="gaussian")

elastic.prediction <- predict(knn.fit,s=knn.fit$lambda.min,newx=test.x)


mean((test.y-elastic.prediction)^2) ## output the MSE
```

```
## [1] 0.536939
```

xgboosted tree model

First we need to tune the hyperparamters, of which there are a few. An efficient method to go about this is by utalising expand.grid to generate a bunch of sets of hyperparameters.

```
xgb_grid <- expand.grid(
nrounds =500 ,
eta = c(0.1, 0.05, 0.01),
max_depth = c(2, 3, 4, 5, 6),
gamma = 0,
colsample_bytree=1,
min_child_weight=c(1, 2, 3, 4 ,5),
subsample=1
)
```

now we can simply use our xgb_grid in caret's built in xgb function to perform cross validation and find the optimal set of parameters.

```
set.seed(7123)
xgb_control <- trainControl(method = "cv",number=3)

xgb_fit <- train(x=as.matrix(train1 %>% select(-rating,-knn.pred)) , y=train1$rating, method='xgbTree',
xgb_fit$bestTune
```

```
##    nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 29     500         2 0.05     0                1                4         1
```

Now that we have ideal hyperparameters we can work with the xgboost package to create the acutal model.

```
##xgboost requires the x variables to be in a xgb.DMatrix
xgb.label <- train1$rating
xgb.train <- xgb.DMatrix(data =as.matrix(train1 %>% select(-rating,-knn.pred)) ,label=xgb.label)
xgb.test <- xgb.DMatrix(data=as.matrix(test1 %>% select(-knn.pred)))
```

```
xgb_param<-list(
        objective = "reg:squarederror",
        booster = "gbtree",
        eta=0.05,
        gamma=0,
        max_depth=3,
        min_child_weight=1,
        subsample=1,
        colsample_bytree=1
)
```

Now to perform cross validation to determine how many rounds is best.

```
set.seed(332145)
xgb_cv <- xgb.cv(params =xgb_param,data=xgb.train,nrounds = 2000,nfold = 5,showsd = T,stratified = T,pr
```

```
## [1]  train-rmse:5.636881+0.007563    test-rmse:5.636895+0.031790
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 10 rounds.
##
## [21] train-rmse:2.140352+0.002386    test-rmse:2.147974+0.022696
## [41] train-rmse:1.001846+0.002915    test-rmse:1.025364+0.014975
## [61] train-rmse:0.711441+0.004109    test-rmse:0.749405+0.013988
## [81] train-rmse:0.648103+0.005075    test-rmse:0.696202+0.017365
## [101]    train-rmse:0.627707+0.005399    test-rmse:0.682384+0.019811
## [121]    train-rmse:0.616264+0.006077    test-rmse:0.676410+0.021375
## [141]    train-rmse:0.606991+0.006117    test-rmse:0.673299+0.022148
## [161]    train-rmse:0.599457+0.005726    test-rmse:0.670826+0.023007
## [181]    train-rmse:0.592805+0.005654    test-rmse:0.669341+0.023527
## [201]    train-rmse:0.587191+0.005465    test-rmse:0.668522+0.024251
## Stopping. Best iteration:
## [200]    train-rmse:0.587468+0.005454    test-rmse:0.668396+0.024284
```

ideal number of rounds seems to be 1045.

```
set.seed(433245)
xgb.model <- xgb.train(params=xgb_param,data=xgb.train,nround=236)
```

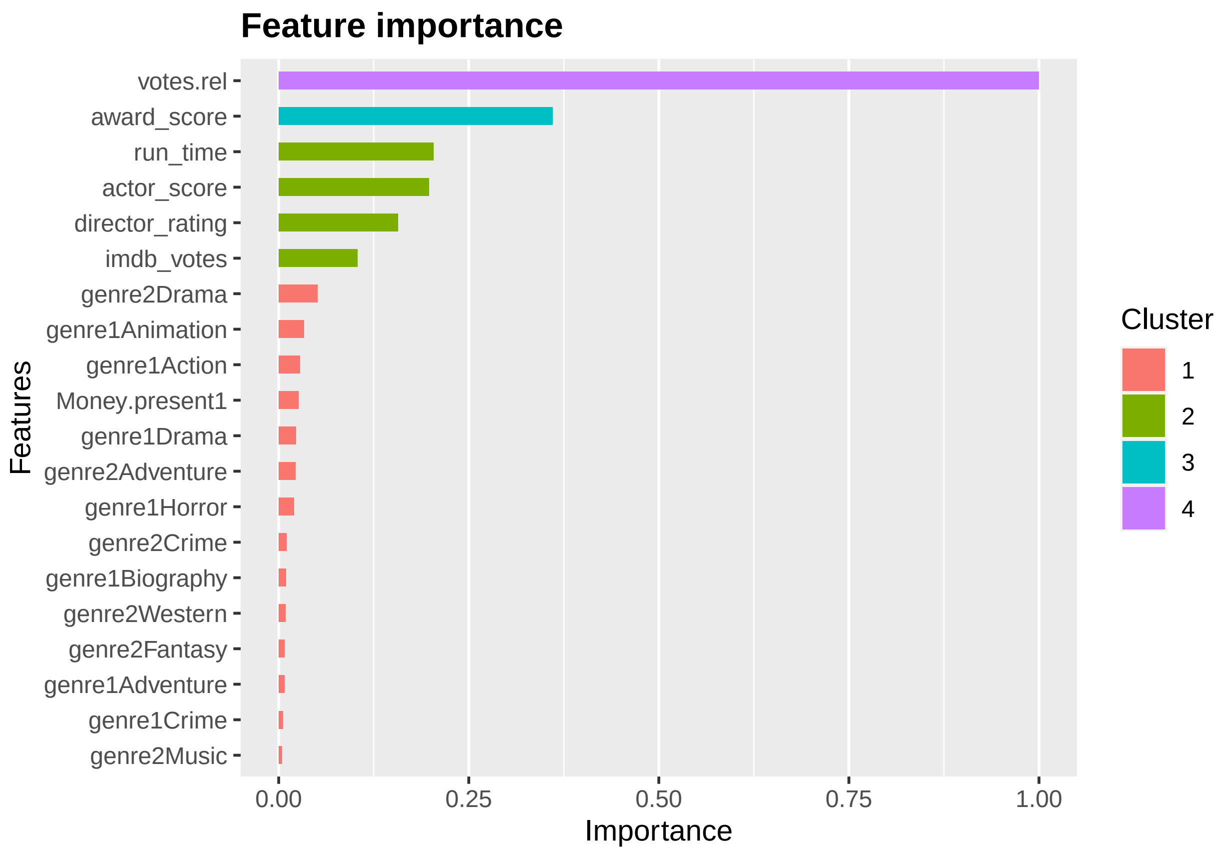
```
xgb.pred <- predict(xgb.model,xgb.test)

mean((test.y-xgb.pred)^2) ## output the MSE
```

```
## [1] 0.5147086
```

```
library(Ckmeans.1d.dp) #required for ggplot clustering
```

```
## Warning: package 'Ckmeans.1d.dp' was built under R version 4.1.2
```

```
mat <- xgb.importance (feature_names = colnames(train1 %>% select(-rating, -knn.pred)),model = xgb.model
xgb.ggplot.importance(importance_matrix = mat[1:20], rel_to_first = T)
```


Feature importance
Features
votes.rel
award_score
run_time
actor_score
director_rating
imdb_votes
genre2Drama
genre1Animation
genre1Action
Money.present1
genre1Drama
genre2Adventure
genre1Horror
genre2Crime
genre1Biography
genre2Western
genre2Fantasy
genre1Adventure
genre1Crime
genre2Music
Importance
Cluster