Predicting Movie Revenue Using Machine Learning

Group Members: Keyi Jiang, Freya Wang, Rita Xu

I. Introduction

The film industry has been one of the most profitable and dynamic industries globally, and it continues to grow exponentially. According to Grand View Research, the global movies and entertainment market size was valued at USD 90.92 billion in 2021 and is expected to expand at a compound annual growth rate (CAGR) of 7.2% from 2022 to 2030. However, predicting how the market will respond to the movie and how will that popularity be embedded in the box office is always the largest challenge for the director, movie company, and also investor. We have heard tons of stories that a grand cast leads to a rotten reputation and a small-budget movie rippled huge waves in the market.

One example of a movie with a large budget but small revenue is the 2019 film, 'Cats.' The film had a budget of approximately $95 million but grossed only $73.7 million at the box office, resulting in a loss of around $21 million. The film received negative reviews from both critics and audiences and failed to attract a large audience despite having a star-studded cast and being based on a popular musical.

In contrast, a black horse story in the movie industry is when a low-budget film unexpectedly becomes a box office success. One such example is the 1999 film, 'The Blair Witch Project.' The film had a budget of only $60,000 but grossed over $248 million worldwide, making it one of the most successful independent films of all time. The film's success was due to its unique marketing strategy, which included a viral campaign that suggested the film was based on real events.

Therefore, we are curious about what would be the determining factors for a movie to succeed, in terms of the box office and that might be uplifting for the movie companies to make decisions on how they would carry out a new movie production. In this project, we aim to develop a model that predicts the revenue of a movie with all the information available before its release.

II. Data and data processing

Getting a clean dataset that is accurate by all means is the hardest part of our project. The data about the box office are extremely messy in terms of units because of the multiple channels. Therefore, we chose to merge datasets from multiple resources after validation and tidying. The main data we are using is from IMBD and TMBD, two mainstream movie rating and review websites. These two datasets contain multiple facets of information about the top 5000 movies such as budget, category data including issue company, crew information, genre and keyword tag, text data including plot overview and reviews, etc. However, a lot of data are outdated and contradict each other and we spend most of our time standardizing these data.

Data Selection

we are using four datasets in this project, with two main ones from IMDB and TMDB and two from The Numbers database and Kaggle as supplements for the `*revenue*` variable.

Standardize title

First, we standardized the title as unique IDs to join different datasets because they all use different IDs to distinguish different movies. By removing white space and punctuation, lowercasing, and transferring all characters into the same encoding, we merge four datasets together and reduce missing values by about 50%. Specifically, we found about 100 movies with the same title but actually different ones, so we added `*title_year*` as a new standard to join the dataset. For `*Budget*`, we use the mean when encountering different data from two sources.

Categorize `*content_rating*`

`*content_rating*` is a variable representing the audience restriction of the content. However, `*content_rating*` in the data set is using more than two principles including Hays Code and MPAA due to different time periods. For example, movies released and produced during 1968-1971 had to be censored before release and those that passed the censor are tagged as 'Approved' or 'Passed' and there were no more detailed categories like 'PG-13' used in later MPAA principles. Therefore, we chose to convert those with similar connotations into same categories and combined those vague standards into PG-13 which is the median of all the content ratings. In detail, the observations ranging from 'M', 'GP', and 'TV-PG' are categorized into 'PG'. The observations varying across 'TV-14, 'Approved', 'Passed', 'Unrated', 'Not Rated, and 'PG-13' are given the value 'PG-13'. Those with an original value equal to 'X' are set to be 'NC-17', which stands for not suitable for people under 18. So after standardization of `*content_rating*`, we have an ordered final `*content_rating*` category, which is G-PG(M/GP/TV-PG)-PG13(TV-14)-R-NC17(X).

Separate month of release

In TMDB dataset, we have the variable named *'release_date'* written in the 'year-month-date' format. So we separate it into *'title_year'*, *'month'* and *'day'* three variables for matching different datasets during the merging process.

Fix NA problems

For our interested denpedent variable 'revenue', we use extra datasets to impute the missing revenue when the two main datasets contain some missing 'revenue'.

Compare similar variables in two main datasets

For the variables *'duration'* and *'runtime'* in two main datasets, we first assign 0 to those NA observations and then take the average of time as the duration if both observations are not equal to 0, and take the maximum of time if at least one of the two observations are equal to 0.

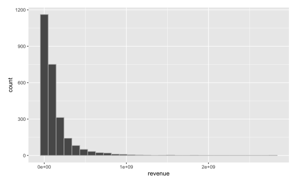
For the variables *'genres*.x' and *'genres*.y', we choose to reserve *'genres*.x' because *'genres*.y' has too many unique values and other useless information such as id.
For the variables *'budget*.x' and *'budget*.y', we choose to reserve *'budget*.x' because *'budget*.y' has too many 0 values.
For the variables *'production_countries'* and *'country*', we choose to drop '*production_countries*' because it has messy formats. And we decide to keep only USA movies since we find the monetary unit of budget and revenue is not uniform.
For the variables 'language' and *'original_language'*, we choose to drop *'original_language'* because it only conveys vague meaning of language. After observing the distribution of *'language'*, we decide it should not be included.

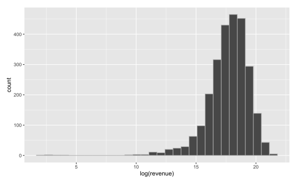
```
 Aramaic  Bosnian     Dari  English Filipino Japanese     Maya  Spanish
       1        1        1     2604        1        1        1        4
```

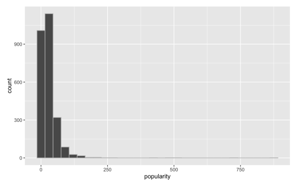
For the variables *'plot_keywords'* and '*keywords*', we choose to keep '*keywords*' for the text analysis because both of them contain similar keywords so we use the one with clear format.

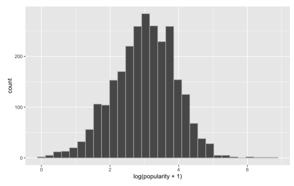
Split text variables
For '*production_companies*', we first create an example data frame with a '*json_string*' column, and then apply a function to extract the *'name'* field from a JSON string. Because some observations only have one production company, so we decide to only keep the main production company. We split the variable *'genre*.x' and 'keywords' on the basis of the same logic.

Regular data processing
Apart from the actions mentioned above, we also do some regular data processing. First we create a dummy variable for whether the movie has a homepage. Second, mutate the length of title, overview and tagline and count the number of keywords. Then, transforming highly right-skewed variables *'revenue'*, *'budget'*, *'popularity'* with a logarithmic transformation. Finally, delete NA values and transform character variables into factor variables.







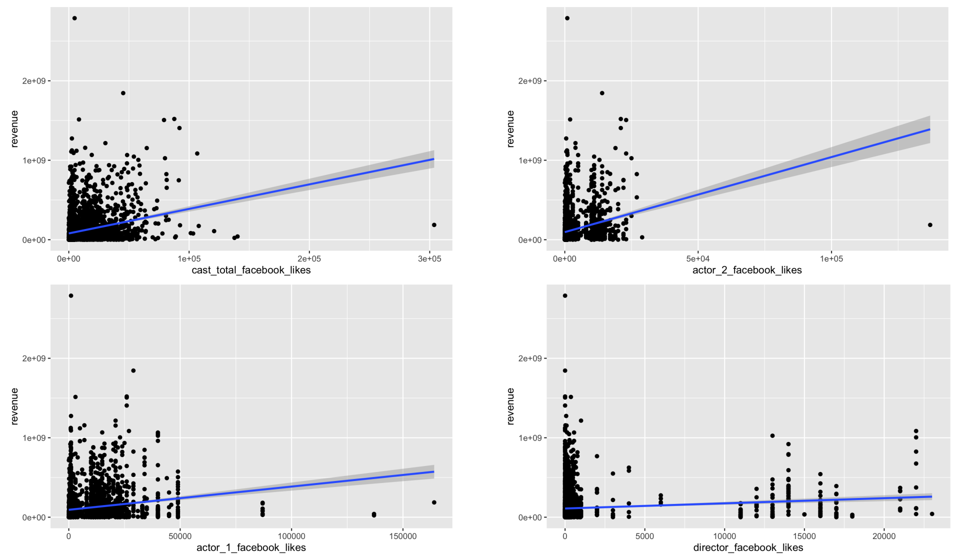


Data explore

Before diving into data processing and engineering, we can get some intuitive interpretations by playing around with data.

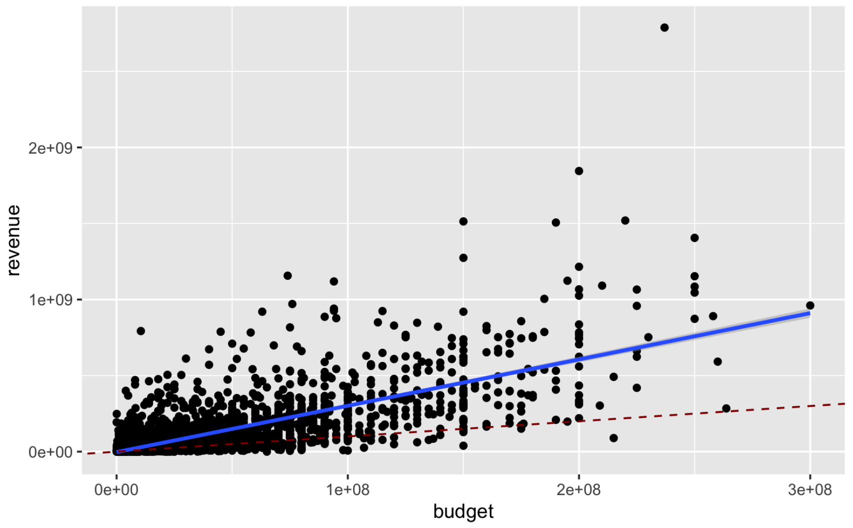
Fan like and revenue

We may think the cast determines how welcome a movie would be and the plots do support this view. Using Facebook likes as the proxy of how popular an actor is, we find a positive relation between Facebook like and revenue. However, it no longer holds true when it comes to the director. That is probably because directors are more like a behind the scene role and thus Facebook fan page, as a product of fan culture, does not apply well to them. Also, the relation may be vague when the actor does not have a fan page because they would have 0 like. We simply delete zero values but still have many tiny values ensemble near 0.



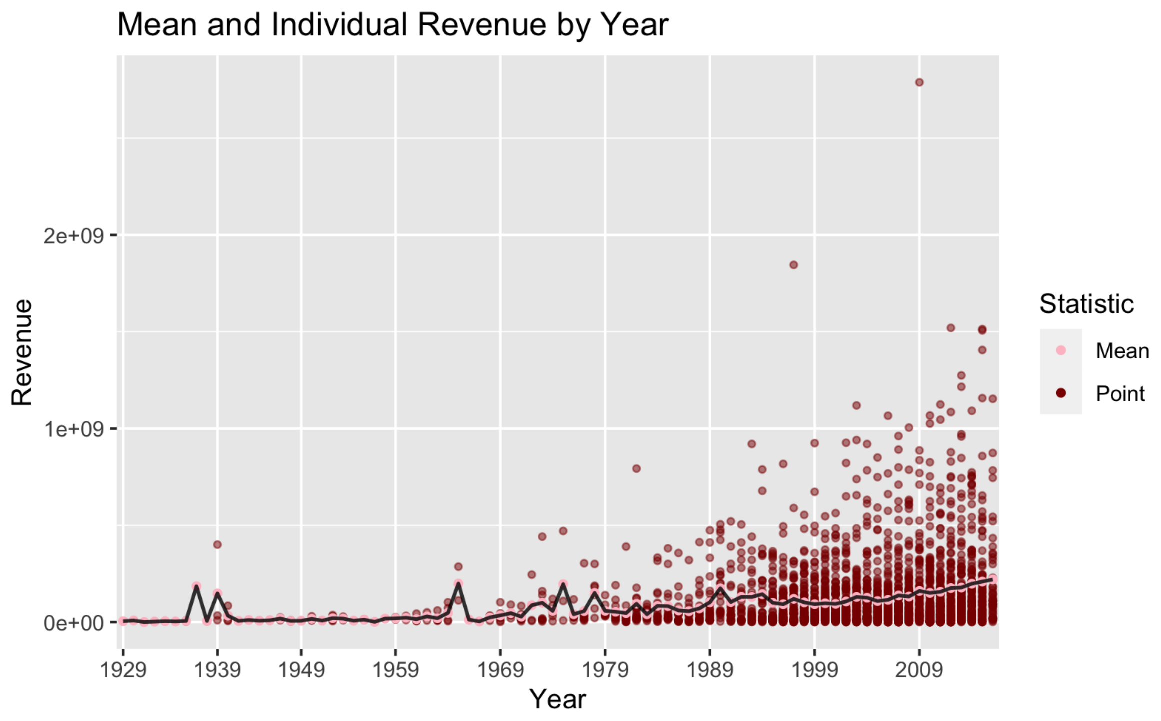
Budget and revenue

Investors and directors are both concerned a lot with the budget. We find a clear positive relation between budget and revenue that the higher the budget is, the higher the revenue of the movies would probably be. The red dashed line represents the budget where the companies can make ends meet. However, we can see lots of movies fall below this line, with a widely known example of 'The Lone Ranger', which costs Walt Disney Pictures two hundred million budget but only owns an 89 million box office.



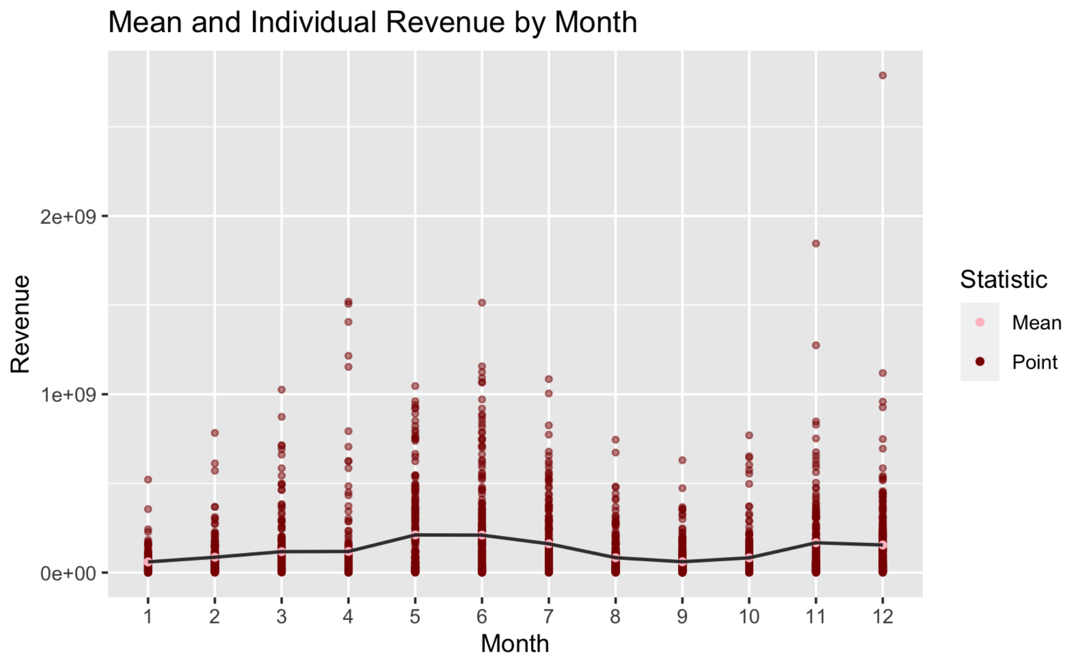
Release date and revenue

From the plot, we can see a slightly increasing average revenue. Several small peaks we observed before 1980 can be explained by multiple re-releases and remakes. For example, Bambi, a Disney movie first released in 1942 and was re-released to theatres in the United States in 1947, 1957, 1966, 1975, 1982, and 1988 and also further reproduction in the forms of DVD and blue ray.



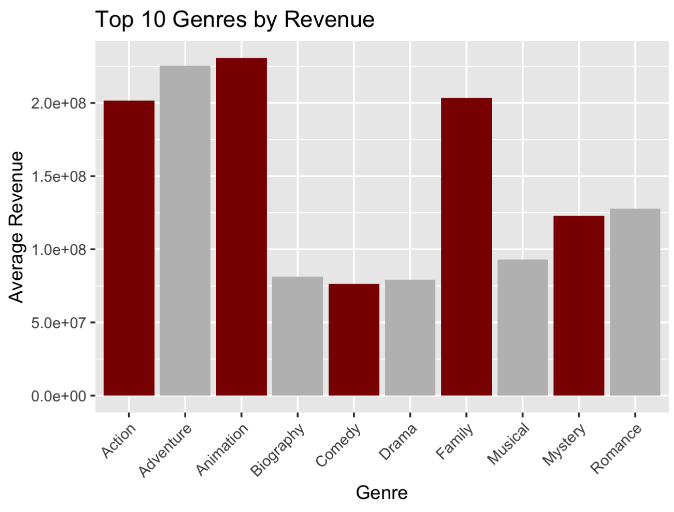
Month

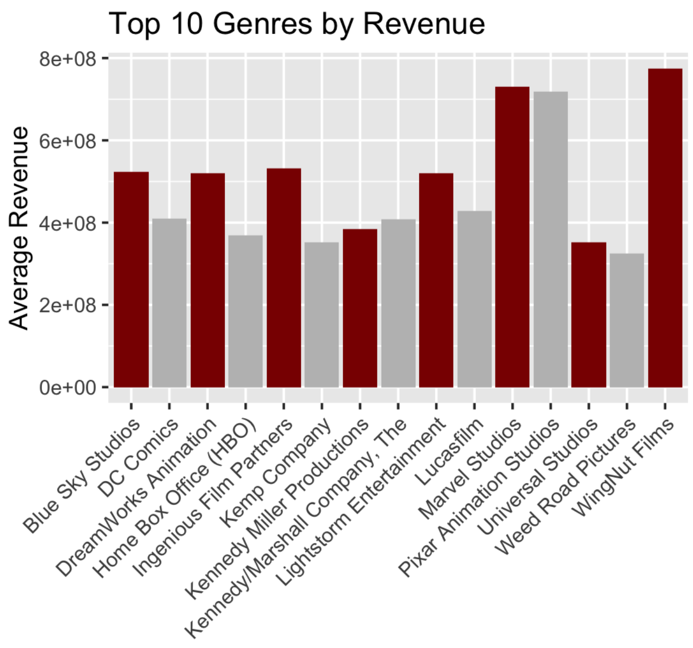
It is reasonable to think of some months to be peak seasons for cinemas such as months near the summer holiday and Christmas. We created the plot describing the relation between mean revenue and month and found some patterns that May and June seem to be profitable months, and November and December have high box office as well.



Revenue by genre and Company

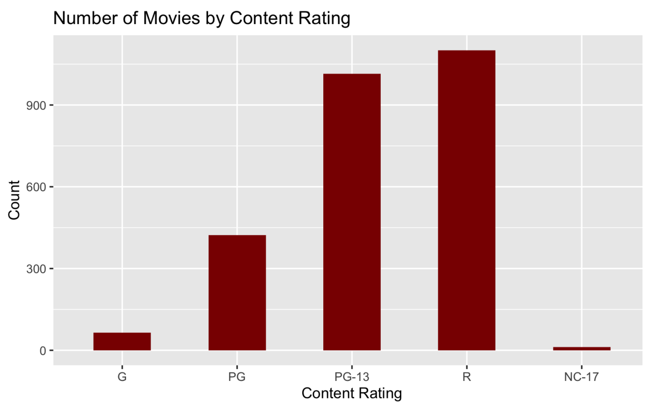
Assigning a main genre and the largest company for each movie and calculating the average avenue, we plotted the top 10 genres and companies that generated the most revenue. Animation, Adventure, Family, and Action ranked top of the list, followed by Mystery and Romance. This can partially reflect people's reference to content when deciding to get into a cinema.
WingNut Films, which produced *The Hobbit* and *The Lord of the Ring* topped the ranking and was followed by Marvel Studios and Pixar Studios.

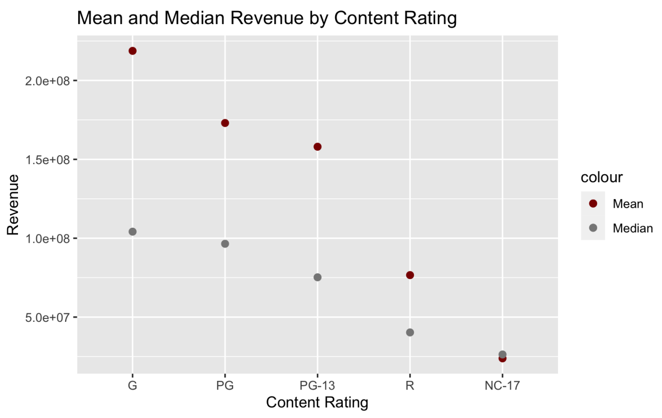




Content rating

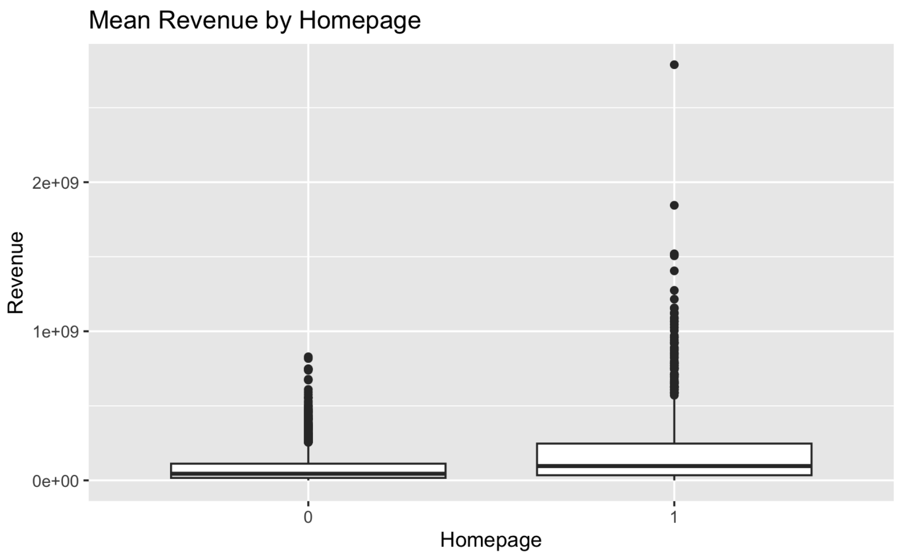
After tiding multiple rating principles into the latest MPAA one, we plotted the number of movies by rating and the mean and median revenue generated by each category of rating. We found 'R' level counts the largest proportion of movies and 'PG-13' ranked second. For the box office, 'G' which means all age audience level generates the highest mean and median revenue which makes sense that people prefer to see 'normal' movies in the cinema.





Homepage

Owing a homepage or not can also be an indicator of a movie's popularity and how the company value this movie before release. Therefore, we transformed the link into a dummy variable indicating whether this movie has a homepage. We found that movies owning a homepage generated way more revenue than movies without one. It can be explained by the correlation between owning a homepage and the company and its budget.

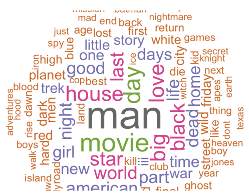


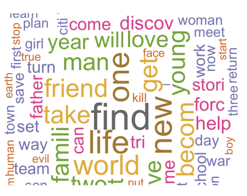
Text Analysis

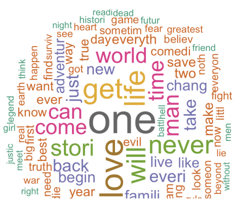
We applied sentiment analysis on the variables 'overview', 'tagline' and 'keywords', trying to transform text information into categories and scores that we can then use in the predictive model. We first gain an overview of the word by using the WordCloud tool and then used two methods to realize this envision.

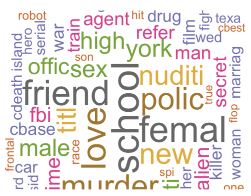
Word Cloud

We found the most frequent words and drew the word cloud plots for 'title', 'overview', 'tagline', 'keywords'.









Title Frequency Overview Frequency Tagline Frequency Keywords Frequency

The underlying theory for drawing a word cloud plot is based on the Bag-of-Words model in Natural Language Processing (NLP). In this model, a text **document** is represented as a bag of words, with each word being treated as a separate unit of analysis, and the frequency of occurrence of each word being recorded.

To create a word cloud plot, the most frequently occurring words in a text document are extracted and their frequencies are used to determine the size and position of each word in the plot. Typically, the larger the font size of a word in the word cloud, the more frequently it occurs in the text. The words are usually arranged in a way that maximizes the visibility of the more important words, while minimizing the overlap of less important words.

Sentiment Analysis

Sentiments can be represented on a numeric scale. We used the Syuzhet package for generating sentiment scores, which has four sentiment dictionaries and offers a method for accessing the sentiment extraction tool developed in the NLP group at Stanford. For example, the table below shows the summary statistics for the sentiment scores of *'title'* variable. We generated sentiment score variables for *'title'*, *'overview'*, *'tagline'* and *'keywords'* in the data frame.

However, we found the sentiment analysis inefficient at some points. For example, it cannot identify the overall sentimental direction only by adding the score of each word together that it identified *Titanic* as a movie with positive emotion. We believe that it can be improved by tools that can better capture the main connotation of the text.

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
-1.75000  0.00000  0.00000 -0.02573  0.00000  2.30000 
```

Potential operations

TF-IDF Analysis
TF-IDF stands for term frequency-inverse document frequency and is used to identify important words in a paragraph by **comparing** weight in different scenarios. And we believe it can help improve our performance on text and sentiment analysis.

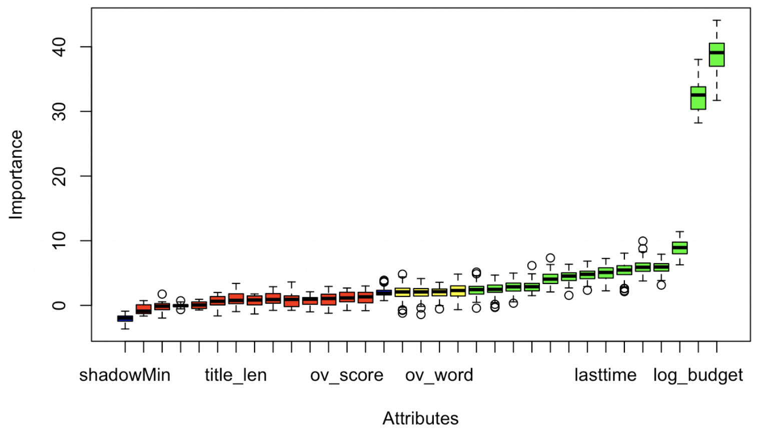
We can try to use the weights gained from TF-IDF to identify the most important and connotative words in the overview and then combine them with the sentiment score we captured in the previous section. Therefore, the words that are more closely related to the main idea of the overview will be weighed higher and the sentimental score linked to it will be higher as well. By doing so, we can avoid giving every word the same weight that some common words may noise the sentiment direction of the overview.

III. Feature Engineering and Modeling

Feature Selection

We use three methods for variable selection: Boruta, Forward Stepwise, and Random Forest variable importance.

Variables Selected by Boruta



```
 [1] "title_year"                 "director_facebook_likes"   "actor_3_facebook_likes"
 [4] "actor_1_facebook_likes"     "main_genre"                "cast_total_facebook_likes"
 [7] "content_rating"             "actor_2_facebook_likes"    "homepage"
[10] "lasttime"                   "log_budget"                "log_popu"
[13] "tag_len"                    "tag_word"
```

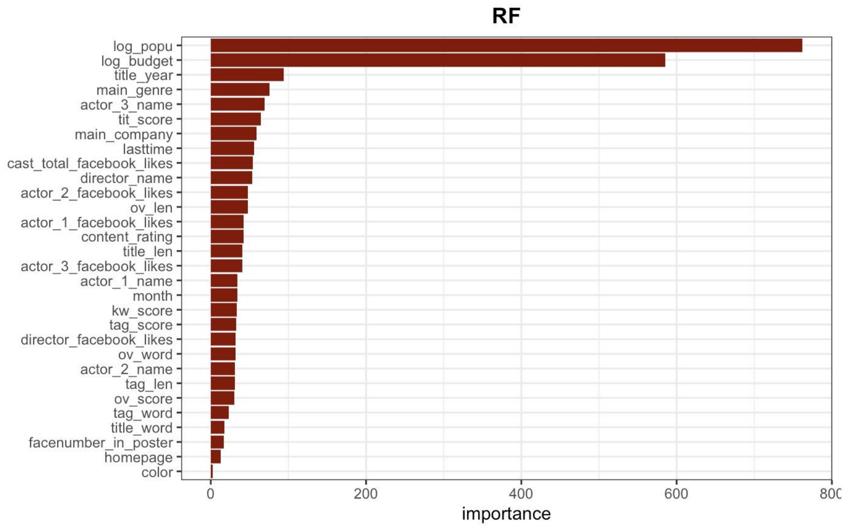
Boruta selected 14 variables.

Variables Selected by Forward Stepwise

	log_revenue		
Predictors	*Estimates*	*CI*	*p*
(Intercept)	33.85	23.55 – 44.14	**<0.001**
log popu	0.95	0.88 – 1.02	**<0.001**
log budget	0.52	0.46 – 0.57	**<0.001**
content_ratingNC-17	-0.84	-1.78 – 0.11	0.082
content rating [PG]	-0.04	-0.43 – 0.35	0.856
content_ratingPG-13	-0.31	-0.68 – 0.07	0.110
content rating [R]	-0.60	-0.98 – -0.22	**0.002**
title year	-0.01	-0.02 – -0.01	**<0.001**
Observations	1568		
R^2	0.561		

Forward stepwise only selected four variables: '*log_popu*', '*log_budget*', '*content_rating*' and '*title_year*'. We may not use forward stepwise because the linear variable selection method does not capture many relationships we explored in section II. Besides, there are many categorical predictors with multiple levels, it can be challenging to represent the relationship between the predictors with response using a linear model.

Variables Selected by Random Forest Variable Importance



According to the RF importance plot, we found quite same result as Boruta that '*log_popu*' and '*log_budget'* has dominantly importance over other variables. And the release year and main genre are also important to the prediction. In the next step, we will test these results by using them in different predictive model to see how they perform on the out-of-sample data.

Models

Random Forest

First, we built three random forest models use all variables, variables selected by Boruta, and variables selected by variable importance. We used the out of sample RMSE to find the best tuning parameters, and predicted on the testing set to get the RMSE.

The table below compares the RMSE on the training and testing data of the three random forest models and the linear model. We can found that the Random Forest model with Boruta feature selection performs best on the testing data among all the models. The linear model with forward stepwise method performs the worst, validating that linear model is not suitable for our problem.

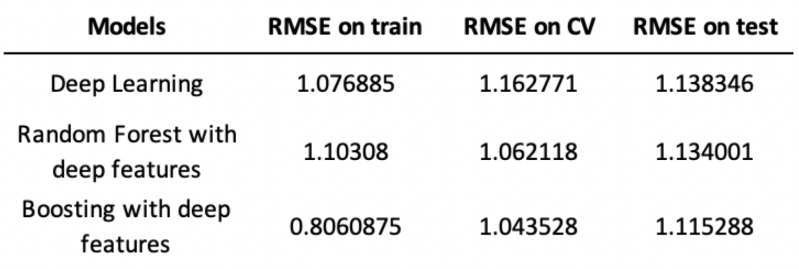
Models	Variable Selection Method	RMSE on train	RMSE on test
Random Forest	None	1.1684	1.1568
Random Forest	Boruta	1.1807	1.1333
Random Forest	Variable Importance	1.1785	1.1560
Linear Model	Forward Stepwise+BIC		1.1647

Neural Networks

We used Neural Networks to check if it can improve the prediction performance. We tried Deep Learning models with 2, 3, and 4 layers and found 2-layer with 50 features per layer achieved the best performance, with 5 cross-validation folds, 0.05 input dropout ratio, 10 epochs.

Next, we extract deep features using the best deep learning model selected and use them to further build a random forest model and a boosting model. The reason we do so is that using deep features can help to capture complex and non-linear relationships and interactions between the features and the response variable that may be difficult to capture with traditional feature engineering. Furthermore, it can help reduce the dimensionality of the input space and reduce the risk of overfitting.

The table below shows the RMSEs on training data, cross-validation data and testing data. It shows that the boosting model using deep features performs best on testing data, with a RMSE 1.12 on test data, with 5 cross-validation folds, 500 trees, max depth 5, sample rate 0.5 and learning rate 0.01. Compare the neural networks models with traditional random forest models above, we can conclude that neural networks performs better in general. That’s because neural networks are capable of learning highly complex and non-linear relationships between input and output data. It can handle large amounts of data and are able to generalize well to new data since they are able to learn from the data and identify patterns that traditional models may miss. Furthermore, can automatically extract relevant features from the data, which can be time-consuming and difficult to do manually in traditional models.



Models	RMSE on train	RMSE on CV	RMSE on test
Deep Learning	1.076885	1.162771	1.138346
Random Forest with deep features	1.10308	1.062118	1.134001
Boosting with deep features	0.8060875	1.043528	1.115288

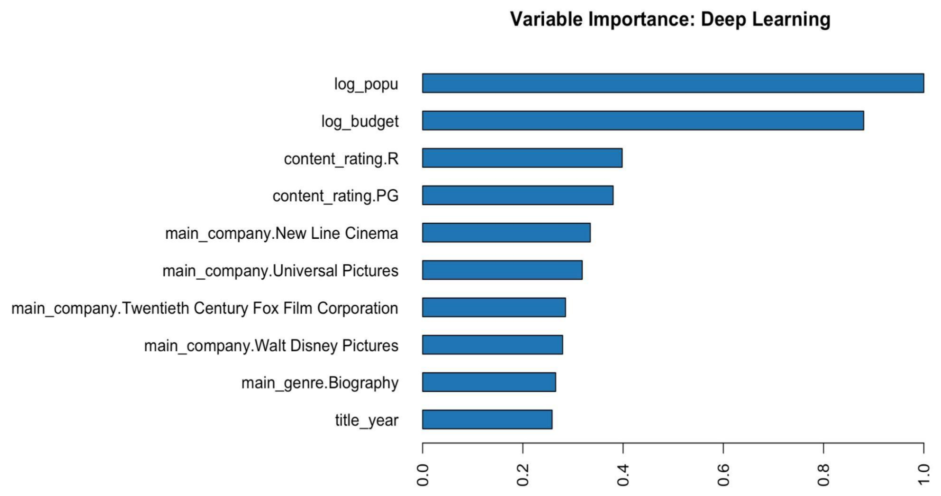
V. Conclusion

Business Insights

From the best deep learning model, we can see that there are several important factors which can help us predict the revenue of the movie.

Among all explanatory variables, popularity is the most important factor and budget ranks the second. It matches our common sense because the more popular a movie is, there will be more people willing to buy the tickets to watch it, thus brings more revenue. As for budget, high budget usually indicates that the cast and director of the movie is well-known and the movie content is intriguing. And movies equipped with such strengths tend to have higher revenue.

Content rating also plays an important role here. One potential reason is that restrictions such as “R” and “PG” are strict requirements, which can have a significant impact on movie revenue.
Some major movie production companies also appear in the imporant variable list. Some famous companies may have better resources. They can secure more movie schedule, which leads to more revenue.



Appendix

Final Project

Keyi Jiang, Freya Wang, Rita Xu

1 Load relevant libraries

```
library(dplyr)
library(caret)
#library(glmnet)
#library(rpart)
#library(rpart.plot)
library(ranger)
#library(dbarts)
library(gbm)
library(glmnet)
#library(ROCR)
library(readr)
library(tidyverse)
library(jsonlite)
library(h2o)
library(ggplot2)
library(tidyr)
library(tm)
library(SnowballC)
library(wordcloud)
library(RColorBrewer)
library(syuzhet)
library(Boruta)
#library(sjPlot)
#library(sjmisc)
#library(sjlabelled)
library(data.table)
```

2 Load the data

```
df_meta<-read.csv("~/Desktop/ml/Final project/movie_metadata.csv",header = TRUE)
df_tmdb<-read.csv("~/Desktop/ml/Final project/tmdb_5000_movies.csv",header = TRUE)
df_extra<-read.csv("~/Desktop/ml/Final project/extra_data.csv")
df_extra2<-read.csv("~/Desktop/ml/Final project/movies.csv")

# change column name
colnames(df_meta)[12] = "title"
colnames(df_extra)[2]="title"
colnames(df_extra)[1]="title_year"
colnames(df_extra)[3]="revenue"
colnames(df_extra2)[1]="title"
colnames(df_extra2)[4]="title_year"
colnames(df_extra2)[13]="revenue"

#leave only year, title and revenue
df_extra2<- df_extra2 %>% select(c("title","title_year","revenue"))

#tranform "revenue" in df_extra to numeric
df_extra <- df_extra %>%
  mutate(revenue = as.numeric(gsub("\\$|,", "", revenue)))
```

3 Data Processing

```
# remove useless variables & variables that we don't know before the movie is open for consumers
df_meta = select(df_meta,-c("movie_imdb_link","num_critic_for_reviews","num_voted_users","num_user_for_
df_tmdb = select(df_tmdb,-c("id","original_title","status","vote_average","vote_count","spoken_language

# Load the stringi package
library(stringi)

# Convert the merging column to lowercase or uppercase in both data frames
df_meta$title <- tolower(df_meta$title)
df_tmdb$title <- tolower(df_tmdb$title)
df_extra$title <- stri_trans_tolower(df_extra$title, locale = "en_US.UTF-8")
df_extra2$title <- tolower(df_extra2$title)

standartize_title <- function(dt){
# Remove whitespace and other special characters from the merging column in both data frames
dt$title<- gsub("[[:punct:]]", "", dt$title)

# Remove rows with missing values in the merging column
dt <- dt[complete.cases(dt$title), ]

# Convert the encoding of the merging column in both data frames to a common encoding
dt$title <- iconv(dt$title, from = "UTF-8", to = "ASCII//TRANSLIT")

# Detect the encoding of the merging column in both data frames
encoding <- guess_encoding(dt$title, n_max = 100)$encoding

# Convert the merging column to a common encoding
dt$title <- iconv(dt$title, from = encoding, to = "UTF-8")

# Remove leading and trailing whitespace from the merging column
dt$title <- trimws(dt$title)

# Remove rows with missing values in the merging column
dt <- dt[complete.cases(dt$title), ]

# Check for and remove any duplicate values in the merging column
```

```r
dt <- unique(dt)
}

for (df_name in c("df_extra", "df_extra2", "df_meta","df_tmdb")) {
  df <- get(df_name)
  # apply function
  df_processed <- standartize_title(df)
  assign(df_name, df_processed)
}

df_tmdb <- df_tmdb %>%
  separate(release_date, into = c("title_year","month","day"), convert=T)
df_tmdb = select(df_tmdb,-c("day"))

# Merge the data tables by "title" and "title_year"
df <- merge(df_meta, df_tmdb, by = c("title","title_year"))

# Remove any duplicate values by "title" and "title_year"
df <- df[!duplicated(df$title,df$title_year),]

#use extra datasets to impute the missing revenue
df$revenue <- ifelse(df$revenue == 0,
                     df_extra$revenue[match(paste(df$title,df$title_year),
                                  paste(df_extra$title,df_extra$title_year))],
                     df$revenue)
df$revenue <- ifelse(is.na(df$revenue) == TRUE,
                     df_extra2$revenue[match(paste(df$title,df$title_year),
                                paste(df_extra2$title,df_extra2$title_year))],
                     df$revenue)

#check na
sum(is.na(df$revenue))
```

```r
# Split production_companies
## Create an example data frame with a "json_string" column
df_split <- data.frame(json_string = df$production_companies)

# Define a function to extract the "name" field from a JSON string
extract_name <- function(json_string) {
  df_split <- fromJSON(json_string)
  return(df_split$name)
}

# Apply the function to the "json_string" column using lapply()
df$production_companies <- lapply(df_split$json_string, extract_name)

# Remove blank strings
df[df == ""] <- NA
df[df == 'NULL'] <- NA

#Extraxt the main production company
for(i in 1:nrow(df)){
  df$main_company[i] <- df$production_companies[[i]][1]
}
df = select(df,-c("production_companies"))


# Replace NA to 0 in duration and runtime
df$duration <- ifelse(is.na(df$duration), 0, df$duration)
df$runtime <- ifelse(is.na(df$runtime), 0, df$runtime)
# Compare duration & runtime
sum(df$duration == df$runtime, na.rm = TRUE)

for(i in 1:nrow(df)){
  df$max[i] = max(c(df$duration[i], df$runtime[i]))
}

# take the average of time as the duration if not 0 value, the maximum of time if 0 value
df$lasttime = ifelse(df$duration == 0|df$runtime==0, df$max,
                     rowMeans(df[, c("duration", "runtime")], na.rm = TRUE))

df = select(df,-c("duration", "runtime","max"))
# delete 0 values in lasttime
df <- df[df$lasttime != 0, ]


# Compare and Split genres.x & genres.y
# unique(df$genres.x)
# unique(df$genres.y)
df = select(df,-c("genres.y")) # genres.y has too many unique values and other useless information such
colnames(df)[9] = "genre"
# keep the first genre as the main genre
df <- df %>%
  separate(genre,into=c("main_genre"))

# Compare budget.x & budget.y
sum(df$budget.x == df$budget.y, na.rm = TRUE)
df = select(df,-c("budget.y")) # budget.y has too many 0 values
colnames(df)[18] = "budget"
# delete 0 values in budget
df <- df[df$budget != 0, ]


# Compare language & original_language
# unique(df$original_language)
# unique(df$language)
df = select(df,-c("original_language")) # choose to drop the column which contains vague meaning of lan


# Compare production_countries & country
df = select(df,-c("production_countries")) # drop the one with messy format
# keep only USA movies since we find the monetary unit of budget and revenue is not uniform
df <- df[df$country == "USA",]
df = select(df,-c("country"))
```

```
# Compare and Split plot_keywords & keywords
# unique(df$plot_keywords)
# unique(df$keywords)
# Both contain similar keywords so we use the one with clear format
df$keyword = strsplit(df$plot_keywords, split = "\\|")
df = select(df,-c("plot_keywords","keywords"))


#content rating is using two principle: Hays Code and MPAA
#passed and approved to be the same-in PG-13 because it's mode
#reference: https://movies.stackexchange.com/questions/65430/what-are-the-meanings-of-the-terms-passed-
#TV-G=G;TV-PG=PG; X=NC-17: not suitable for under 18
#G--PG--PG13--R(under16-parent)--NC17(no under 16)
#G--M(parental discretion)--R(under16-parent)--X(no under 16)
#M=PG:GP=PG(1969-1972)
#final category: G-PG(M/GP/TV-PG)-PG13(TV-14)-R-NC17(X)
df <- df %>% mutate(
    content_rating=case_when(
    content_rating=="TV-G"|content_rating=="G"           ~"G",
    content_rating=="M"|content_rating=="GP"|
    content_rating=="TV-PG"|content_rating=="PG"        ~"PG",
    content_rating=="TV-14"|content_rating=="Approved"|
    content_rating=="Passed"|content_rating=="Unrated"|content_rating=="Not Rated"|
    content_rating=="PG-13"                          ~"PG-13",
    content_rating=="X"|content_rating=="NC-17"      ~"NC-17",
    content_rating=="R"                                  ~"R"))


# Create dummy variables
df$homepage <- ifelse(is.na(df$homepage), 0, 1)

# Delete NA values
df<- na.omit(df)

# Transform character into factor
df <- df %>%
  mutate_if(is.character, as.factor)

summary(df$language) # should not include language variable
df = select(df,-c("language"))

df$homepage <- as.factor(df$homepage)

#str(df)
```

```
# log revenue, budget and popularity
ggplot(df)+
  geom_histogram(aes(x=revenue), color = "grey")
ggplot(df)+
  geom_histogram(aes(x=log(revenue)), color = "grey")

ggplot(df)+
  geom_histogram(aes(x=budget), color = "grey")
```

```r
ggplot(df)+
  geom_histogram(aes(x=log(budget)), color = "grey")

ggplot(df)+
  geom_histogram(aes(x=popularity), color = "grey")
ggplot(df)+
  geom_histogram(aes(x=log(popularity+1)), color = "grey")

df <- df %>%
  mutate(log_revenue=log(revenue)) %>%
  mutate(log_budget=log(budget)) %>%
  mutate(log_popu=log(popularity+1))
df = select(df,-c("revenue","budget","popularity"))
```

```r
# mutate the length of title, overview and tagline
df$title_len <- nchar(gsub("[[:punct:][:space:]]", "", df$title))
df$title_word <- str_count(df$title,"\\S+")

df$ov_len <- nchar(gsub("[[:punct:][:space:]]", "", df$overview))
df$ov_word <- str_count(df$overview,"\\S+")

df$tag_len <- nchar(gsub("[[:punct:][:space:]]", "", df$tagline))
df$tag_word <- str_count(df$tagline,"\\S+")

# count keywords
df$num_keyword <- lengths(df$keyword)
ggplot(df)+
  geom_histogram(aes(x=num_keyword)) # the number of keywords may not be useful
df <- select(df,-c("num_keyword"))
```

4 Sentiment Analysis-Score

4.1 For title

```r
tit <- as.character(df$title)
titDoc <- Corpus(VectorSource(tit))

# Cleaning up text data
#Replacing "/", "@" and "|" with space
toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
titDoc <- tm_map(titDoc, toSpace, "/")
titDoc <- tm_map(titDoc, toSpace, "@")
titDoc <- tm_map(titDoc, toSpace, "\\|")
# Convert the text to lower case
titDoc <- tm_map(titDoc, content_transformer(tolower))
# Remove numbers
titDoc <- tm_map(titDoc, removeNumbers)
# Remove english common stopwords
titDoc <- tm_map(titDoc, removeWords, stopwords("english"))
# Remove punctuations
```

```r
titDoc <- tm_map(titDoc, removePunctuation)
# Eliminate extra white spaces
titDoc <- tm_map(titDoc, stripWhitespace)
# Text stemming - which reduces words to their root form
#titDoc <- tm_map(titDoc, stemDocument)

# Build a term-document matrix
titDoc_dtm <- TermDocumentMatrix(titDoc)
dtm_m <- as.matrix(titDoc_dtm)
# Sort by descearing value of frequency
dtm_v <- sort(rowSums(dtm_m),decreasing=TRUE)
dtm_d <- data.frame(word = names(dtm_v),freq=dtm_v)
# Display the top 5 most frequent words
head(dtm_d, 5)


# Plot the most frequent words
barplot(dtm_d[1:5,]$freq, las = 2, names.arg = dtm_d[1:5,]$word,
        col ="lightgreen", main ="Top 5 most frequent words",
        ylab = "Word frequencies")

#generate word cloud
set.seed(1)
wordcloud(words = dtm_d$word, freq = dtm_d$freq, min.freq = 5,
          max.words=100, random.order=FALSE, rot.per=0.40,
          colors=brewer.pal(8, "Dark2"))

# Sentiment Scores
df$tit_score <- get_sentiment(tit, method="syuzhet")
head(df$tit_score)
summary(df$tit_score)
```

4.2 For overview

```r
ov <- as.character(df$overview)
ovDoc <- Corpus(VectorSource(ov))

# Cleaning up text data
#Replacing "/", "@" and "|" with space
toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
ovDoc <- tm_map(ovDoc, toSpace, "/")
ovDoc <- tm_map(ovDoc, toSpace, "@")
ovDoc <- tm_map(ovDoc, toSpace, "\\|")
# Convert the text to lower case
ovDoc <- tm_map(ovDoc, content_transformer(tolower))
# Remove numbers
ovDoc <- tm_map(ovDoc, removeNumbers)
# Remove english common stopwords
ovDoc <- tm_map(ovDoc, removeWords, stopwords("english"))
# Remove punctuations
ovDoc <- tm_map(ovDoc, removePunctuation)
```

```r
# Eliminate extra white spaces
ovDoc <- tm_map(ovDoc, stripWhitespace)
# Text stemming - which reduces words to their root form
#ovDoc <- tm_map(ovDoc, stemDocument)

# Build a term-document matrix
ovDoc_dtm <- TermDocumentMatrix(ovDoc)
dtm_m <- as.matrix(ovDoc_dtm)
# Sort by descearing value of frequency
dtm_v <- sort(rowSums(dtm_m),decreasing=TRUE)
dtm_d <- data.frame(word = names(dtm_v),freq=dtm_v)
# Display the top 5 most frequent words
head(dtm_d, 5)


# Plot the most frequent words
barplot(dtm_d[1:5,]$freq, las = 2, names.arg = dtm_d[1:5,]$word,
        col ="lightgreen", main ="Top 5 most frequent words",
        ylab = "Word frequencies")

#generate word cloud
set.seed(1)
wordcloud(words = dtm_d$word, freq = dtm_d$freq, min.freq = 5,
          max.words=100, random.order=FALSE, rot.per=0.40,
          colors=brewer.pal(8, "Dark2"))

# Sentiment Scores
df$ov_score <- get_sentiment(ov, method="syuzhet")
head(df$ov_score)
summary(df$ov_score)
```

4.3 For tagline

```r
tag <- as.character(df$tagline)
tagDoc <- Corpus(VectorSource(tag))

# Cleaning up text data
#Replacing "/", "@" and "|" with space
toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
tagDoc <- tm_map(tagDoc, toSpace, "/")
tagDoc <- tm_map(tagDoc, toSpace, "@")
tagDoc <- tm_map(tagDoc, toSpace, "\\|")
# Convert the text to lower case
tagDoc <- tm_map(tagDoc, content_transformer(tolower))
# Remove numbers
tagDoc <- tm_map(tagDoc, removeNumbers)
# Remove english common stopwords
tagDoc <- tm_map(tagDoc, removeWords, stopwords("english"))
# Remove punctuations
tagDoc <- tm_map(tagDoc, removePunctuation)
# Eliminate extra white spaces
```

```r
tagDoc <- tm_map(tagDoc, stripWhitespace)
# Text stemming - which reduces words to their root form
#tagDoc <- tm_map(tagDoc, stemDocument)

# Build a term-document matrix
tagDoc_dtm <- TermDocumentMatrix(tagDoc)
dtm_m <- as.matrix(tagDoc_dtm)
# Sort by descearing value of frequency
dtm_v <- sort(rowSums(dtm_m),decreasing=TRUE)
dtm_d <- data.frame(word = names(dtm_v),freq=dtm_v)
# Display the top 5 most frequent words
head(dtm_d, 5)


# Plot the most frequent words
barplot(dtm_d[1:5,]$freq, las = 2, names.arg = dtm_d[1:5,]$word,
        col ="lightgreen", main ="Top 5 most frequent words",
        ylab = "Word frequencies")

#generate word cloud
set.seed(1)
wordcloud(words = dtm_d$word, freq = dtm_d$freq, min.freq = 5,
          max.words=100, random.order=FALSE, rot.per=0.40,
          colors=brewer.pal(8, "Dark2"))

# Sentiment Scores
df$tag_score <- get_sentiment(tag, method="syuzhet")
head(df$tag_score)
summary(df$tag_score)
```

4.4 For keywords

```r
kw <- as.character(df$keyword)
kwDoc <- Corpus(VectorSource(kw))

# Cleaning up text data
#Replacing "/", "@" and "|" with space
toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
kwDoc <- tm_map(kwDoc, toSpace, "/")
kwDoc <- tm_map(kwDoc, toSpace, "@")
kwDoc <- tm_map(kwDoc, toSpace, "\\|")
# Convert the text to lower case
kwDoc <- tm_map(kwDoc, content_transformer(tolower))
# Remove numbers
kwDoc <- tm_map(kwDoc, removeNumbers)
# Remove english common stopwords
kwDoc <- tm_map(kwDoc, removeWords, stopwords("english"))
# Remove punctuations
kwDoc <- tm_map(kwDoc, removePunctuation)
# Eliminate extra white spaces
kwDoc <- tm_map(kwDoc, stripWhitespace)
```

```
# Text stemming - which reduces words to their root form
#kwDoc <- tm_map(kwDoc, stemDocument)

# Build a term-document matrix
kwDoc_dtm <- TermDocumentMatrix(kwDoc)
dtm_m <- as.matrix(kwDoc_dtm)
# Sort by descearing value of frequency
dtm_v <- sort(rowSums(dtm_m),decreasing=TRUE)
dtm_d <- data.frame(word = names(dtm_v),freq=dtm_v)
# Display the top 5 most frequent words
head(dtm_d, 5)


# Plot the most frequent words
barplot(dtm_d[1:5,]$freq, las = 2, names.arg = dtm_d[1:5,]$word,
        col ="lightgreen", main ="Top 5 most frequent words",
        ylab = "Word frequencies")

#generate word cloud
set.seed(1)
wordcloud(words = dtm_d$word, freq = dtm_d$freq, min.freq = 5,
          max.words=100, random.order=FALSE, rot.per=0.40,
          colors=brewer.pal(8, "Dark2"))

# Sentiment Scores
df$kw_score <- get_sentiment(kw, method="syuzhet")
head(df$kw_score)
summary(df$kw_score)
```

5 Sentiment Classification

5.1 For overview

```
#ovClass <- get_nrc_sentiment(ov)
#head (ovClass,10)
#df <- cbind(df,ovClass)
```

6 Build Models and Prediction

6.1 Variable Selection

6.1.1 Boruta

```
df.all <- df
df <- select(df, -c("overview","tagline","keyword","title"))

# split data into training and testing set
```

```r
set.seed(327)
N = dim(df)[1]
train_ratio = 0.6
train_index = sample(N,train_ratio*N)
train = df[train_index,]
test = df[-train_index,]

train.X = select(train, -c("log_revenue"))
train.y = train$log_revenue
test.X = select(test, -c("log_revenue"))
test.y = test$log_revenue

# use boruta to select variables
boruta <- Boruta(train.X, train.y, doTrace=0)
plot(boruta)

# Select important features
#features=names(boruta$finalDecision)[boruta$finalDecision %in% c("Confirmed","Tentative")]
features = getSelectedAttributes(boruta)
features

train.red_dim = train[ , c(features, "log_revenue")]
test.red_dim = test[ , c(features, "log_revenue")]
```

6.1.2 Forward Stepwise+BIC

```r
lgfit.all <- glm(log_revenue~ .,
                 data=train, family="gaussian")

lgfit.null <- glm(log_revenue~ 1,
                  data=train, family="gaussian")

lgfit.selected <- step(lgfit.null,                  # the starting model for our search
                       scope=formula(lgfit.all),    # the largest possible model that we will consider.
                       direction="forward",
                       k=log(nrow(train)),       # by default step() uses AIC, but by
                                                    # multiplying log(n) on the penalty, we get BIC.
                                                    # See ?step -> Arguments -> k

                       trace=1)
summary(lgfit.selected)

pred.lgfit.selected <- predict(lgfit.selected,
                               newdata = test,
                               type = "response")

rmse <- sqrt(mean((test$log_revenue - pred.lgfit.selected)^2))
rmse
rmse.fw <- rmse

tab_model(lgfit.selected)
```

6.1.3 Lasso

```
#X.train <- as.matrix(train.X)
#y.train <- as.matrix(train.y)
#lasso <- cv.glmnet(X.train, y.train, alpha = 1)
```

6.1.4 Random Forest

```
hyper_grid <- expand.grid(
  mtry        = seq(2, 26, by = 2),
  node_size   = c(5,15,25),
  sample_size = c(.55, .632, .80),
  OOB_RMSE    = 0
)

for(i in 1:nrow(hyper_grid)) {

  # train model
  RF <- ranger(
    formula         = log_revenue ~ .,
    data            = train,
    num.trees       = 500,
    mtry            = hyper_grid$mtry[i],
    min.node.size   = hyper_grid$node_size[i],
    sample.fraction = hyper_grid$sample_size[i],
    seed            = 1108
  )

  # add OOB error to grid
  hyper_grid$OOB_RMSE[i] <- sqrt(RF$prediction.error)
}

(oo = hyper_grid %>%
  dplyr::arrange(OOB_RMSE) %>%
  head(10))
```

```
rf.fit.final <- ranger(
    formula         = log_revenue ~ .,
    data            = train,
    num.trees       = 500,
    mtry            = oo[1,]$mtry,
    min.node.size   = oo[1,]$node_size,
    sample.fraction = oo[1,]$sample_size,
    importance      = 'impurity',
    seed = 1108
    )

yhat.rf = predict(rf.fit.final, data = test)
rmse <- sqrt(mean((test$log_revenue - yhat.rf$predictions)^2))
rmse
```

Variable Importance

```
importance_rf = as.data.table(rf.fit.final$variable.importance, keep.rownames = T)
sort(rf.fit.final$variable.importance)

rf.select <- c("log_popu","log_budget","title_year","main_genre","actor_3_name","tit_score","main_compa
train.rf = train[ , c(rf.select, "log_revenue")]
test.rf = test[ , c(rf.select, "log_revenue")]
```

6.2 Model

6.2.1 rf+b selection

```
rf.fit.b <- ranger(
    formula         = log_revenue ~ .,
    data            = train.red_dim,
    num.trees       = 500,
    mtry            = 10,
    min.node.size   = 5,
    sample.fraction = .8,
    importance      = 'impurity',
    seed = 1108
    )
rmse.rfbtrain<-sqrt(rf.fit.b$prediction.error)
rmse.rfbtrain
yhat.rf.b = predict(rf.fit.b, data = test.red_dim)
rmse.rfb <- sqrt(mean((test$log_revenue - yhat.rf.b$predictions)^2))
rmse.rfb
```

6.2.2 rf+rf selection

```
rf.fit.rf <- ranger(
    formula         = log_revenue ~ .,
    data            = train.rf,
    num.trees       = 500,
    mtry            = 4,
    min.node.size   = 4,
    sample.fraction = .8,
    importance      = 'impurity',
    seed = 1108
    )
rmse.rfrftrain<-sqrt(rf.fit.rf$prediction.error)
rmse.rfrftrain
yhat.rf.rf = predict(rf.fit.rf, data = test.rf)
rmse.rfrf <- sqrt(mean((test$log_revenue - yhat.rf.rf$predictions)^2))
rmse.rfrf
```

7 Neural Network

```r
# Initialize H2O cluster
h2o.init()
h2o.removeAll()
h2o.no_progress()

df_h2o <- as.h2o(df, use_datatable = TRUE)
train_h2o <- as.h2o(train, use_datatable = TRUE)
test_h2o <- as.h2o(test, use_datatable = TRUE)


# Set response and predictor variables
response <- "log_revenue"
predictors <- setdiff(names(train_h2o), response)
```

```r
# Tuning with Random Search
hyper_params <- list(
  activation=c("Rectifier","Tanh","RectifierWithDropout","TanhWithDropout"),
  hidden=list(c(20,20),c(50,50),c(30,30,30),c(25,25,25,25),c(64,64,64,64)),
  input_dropout_ratio=c(0,0.05),
  l1=seq(0,1e-4,1e-6),
  l2=seq(0,1e-4,1e-6),
  max_w2=c(5,10,15)
)

search_criteria = list(
  strategy = "RandomDiscrete",
  max_runtime_secs = 600,
  max_models = 100,
  seed=1,
  stopping_rounds=5,
  stopping_tolerance=1e-2 # stop once the top 5 models are within 1% of each other
  )

dl_random_grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id = "dl_grid_random",
  training_frame=train_h2o,
  nfolds = 5,
  x=predictors,
  y=response,
  epochs=10,
  stopping_metric="MSE",
  score_duty_cycle=0.025, # don't score more than 2.5% of the wall time
  hyper_params = hyper_params,
  search_criteria = search_criteria
)
summary(dl_random_grid, show_stack_traces = TRUE)
```

```r
#summary(dl_random_grid, show_stack_traces = TRUE)
grid <- h2o.getGrid("dl_grid_random", sort_by="RMSE", decreasing=FALSE)
```

```r
grid

grid@summary_table[1,]

best_model <- h2o.getModel(grid@model_ids[[1]]) ## model with lowest RMSE
best_model

plot(best_model)
importance <- summary(best_model)[1:10,]
importance
h2o.varimp_plot(best_model)
```

```r
best_perf <- h2o.performance(best_model, newdata=test_h2o) # prediction
best_perf

h2o.rmse(best_perf) # report RMSE on the test data
```

8 Extract deep features using the best model selected above

```r
l <- as.numeric(str_extract_all(grid@summary_table[1,"hidden"], "[0-9]+")[[1]])

trainX.deep.features = h2o::h2o.deepfeatures(best_model, train_h2o[,predictors], layer = length(l))
testX.deep.features = h2o::h2o.deepfeatures(best_model, test_h2o[,predictors], layer = length(l))

dim(trainX.deep.features)
```

9 Random Forest

```r
train_reduced <- h2o.cbind(trainX.deep.features, train_h2o[,response])

drf <- h2o.randomForest(x = 1:dim(trainX.deep.features)[2],
                        y = dim(trainX.deep.features)[2]+1,
                        training_frame = train_reduced,
                        nfolds = 5,
                        ntrees = 500,
                        min_rows = 3,
                        stopping_rounds = 5,
                        stopping_metric="MSE",
                        model_id = "drf_features1",
                        seed = 1)
drf

plot(drf)
```

```r
test_reduced <- h2o.cbind(testX.deep.features, test_h2o[,response])

drf_perf <- h2o.performance(drf, newdata = test_reduced) # prediction
```

```r
drf_perf

h2o.rmse(drf_perf) # report RMSE
```

10 Boosting

```r
boost <- h2o.gbm(x = 1:dim(trainX.deep.features)[2],
                 y = dim(trainX.deep.features)[2]+1,
                 training_frame = train_reduced,
                 nfolds = 5,
                 ntrees = 500,
                 max_depth = 5,
                 min_rows = 5,
                 learn_rate = 0.01,
                 sample_rate = 0.5,
                 col_sample_rate = 0.6,
                 stopping_metric="MSE",
                 stopping_rounds = 5,
                 model_id = "boost_features1",
                 seed = 1)
boost

plot(boost)
```

```r
boost_perf <- h2o.performance(boost, newdata = test_reduced) # prediction
boost_perf

h2o.rmse(boost_perf) # report RMSE
h2o.varimp_plot(boost)
```

```
‘‘‘r
h2o.shutdown()
```