Code

Predicting the gross of top 1000 movies in IMDB

PSTAT 131 Final Project

Tingting Liang

2024-03-23

Introduction

In today's ever-evolving film industry, understanding the factors that influences a movie's success can be very crucial for film-makers, producers and movie investors. One key metric used to assess a film's success is its gross revenue, which not only reflects its financial success but also how well its been received by the general public. To predict a movie's revenue before its release can offer valuable insight for the decision making process, aiding in resource allocation and strategic planning.

The aim of this project is to predict the gross revenue of films by analyzing the "IMDB Movies Dataset" from Kaggle. Using variables like genre, IMDB rating, runtime, certificate and other relevant factors to predict the gross revenue of movies. By analyzing these factors, we seek to identify patterns and correlations that contribute to a film's financial success. Consequently, providing potentially useful insights for industry professionals involved in film production, distribution, and investment.

Loading and Exploring the Raw Data

Lets load the necessary packages and take a look at the raw IMDB data set. Then use the clean_names() function to clean up the variable names.

Show

```
## ── Attaching core tidyverse packages ─────────────────
──────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.4     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.4.4     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ─────────────────────────────────────────
─ tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-li
b.org/>) to force all conflicts to become errors
```

Show

```
## ── Attaching packages ────────────────────────────────
─────── tidymodels 1.1.1 ──
## ✔ broom        1.0.5     ✔ rsample      1.2.0
## ✔ dials        1.2.0     ✔ tune         1.1.2
## ✔ infer        1.0.5     ✔ workflows    1.1.3
## ✔ modeldata    1.3.0     ✔ workflowsets 1.0.1
## ✔ parsnip      1.1.1     ✔ yardstick    1.3.0
## ✔ recipes      1.0.9
## ── Conflicts ─────────────────────────────────────────
tidymodels_conflicts() ──
## ✖ scales::discard() masks purrr::discard()
## ✖ dplyr::filter()   masks stats::filter()
## ✖ recipes::fixed()  masks stringr::fixed()
## ✖ dplyr::lag()      masks stats::lag()
## ✖ yardstick::spec() masks readr::spec()
## ✖ recipes::step()   masks stats::step()
## • Use suppressPackageStartupMessages() to eliminate pa
ckage startup messages
```

Show

```
##
## Attaching package: 'janitor'
##
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

Show

```
## corrplot 0.92 loaded
```

Show

```
##
## Attaching package: 'psych'
##
## The following objects are masked from 'package:scale
s':
##
##     alpha, rescale
##
## The following objects are masked from 'package:ggplot
2':
##
##     %+%, alpha
```

Show

```
##
## Attaching package: 'discrim'
##
## The following object is masked from 'package:dials':
##
##     smoothness
```

Show

```
##
## Attaching package: 'xgboost'
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

Show

```
##
## Attaching package: 'vip'
##
## The following object is masked from 'package:utils':
##
##     vi
```

Show

```
## Rows: 1000 Columns: 16
## ── Column specification ────────────────────────────────
────────────────────────────
## Delimiter: ","
## chr (12): Poster_Link, Series_Title, Released_Year, Ce
rtificate, Runtime, Ge...
## dbl  (3): IMDB_Rating, Meta_score, No_of_Votes
## num  (1): Gross
##
## ℹ Use `spec()` to retrieve the full column specificati
on for this data.
## ℹ Specify the column types or set `show_col_types = FA
LSE` to quiet this message.
```

Show

```
## # A tibble: 6 × 16
##   Poster_Link   Series_Title Released_Year Certificate
Runtime Genre IMDB_Rating
##   <chr>         <chr>        <chr>         <chr>
<chr>   <chr>       <dbl>
## 1 https://m.me… The Shawsha… 1994          A
142 min Drama         9.3
## 2 https://m.me… The Godfath… 1972          A
175 min Crim…         9.2
## 3 https://m.me… The Dark Kn… 2008          UA
152 min Acti…         9
## 4 https://m.me… The Godfath… 1974          A
202 min Crim…         9
## 5 https://m.me… 12 Angry Men 1957          U
96 min  Crim…         9
## 6 https://m.me… The Lord of… 2003          U
201 min Acti…         8.9
## # ℹ 9 more variables: Overview <chr>, Meta_score <dbl
>, Director <chr>,
## #   Star1 <chr>, Star2 <chr>, Star3 <chr>, Star4 <chr
>, No_of_Votes <dbl>,
## #   Gross <dbl>
```

Show

Data Tidying

Variable selection

Let's first take a look at the dimension of our raw data.

Show

```
## [1] 1000   16
```

This data set contains 1000 rows and 16 columns. Which means we have 1000 movies and 16 variables. One variable `gross` is our response variable so the other 15 are predictor variables. But notice the naming and descriptive variables like `poster_link`, `series_title`, `overview`, `director`, and all the variables with star names aren't really relevant for predicting our outcome gross revenue, so we'll be dropping those columns.

Show

```
## [1] 1000    8
```

This leaves us with a new data set of 1000 rows and 8 columns. This means we have 1000 movies and 8 variables in the data set. One being the outcome variable `gross`, so this leaves us with 7 predictor variables.

Now let's take a look at the variable types we'll be working on and see if we need to make any necessary changes.

Show

```
## released_year   certificate       runtime         genr
e   imdb_rating
##   "character"   "character"   "character"   "characte
r"     "numeric"
##    meta_score   no_of_votes         gross
##     "numeric"     "numeric"     "numeric"
```

We have a data set with both categorical and numeric columns. The categorical columns are `released_year`, `certificate`, `runtime`, and `genre`. The numeric columns are `imdb_rating`, `meta_score`, `no_of_votes`, and `gross`. Notice variables `runtime` and `released_Year` contains numeral values but are inputted as categorical variables, it might be more helpful to use `runtime` and `released_Year` as numeric values to predict gross revenue, so let's covert them into numeric type.

Show

```
## [1] "142 " "175 " "152 " "202 " "96 "  "201 "
```

Show

```
## [1] "numeric"
```

Show

```
## Warning: NAs introduced by coercion
```

Show

```
## [1] "numeric"
```

Tidying the Outcome variable

Notice our outcome variable `gross` is quite large, lets change its unit to millions so it's easier to visualize and analyze.

Show

```
## # A tibble: 6 × 1
##    gross
##    <dbl>
## 1  28.3
## 2 135.
## 3 535.
## 4  57.3
## 5   4.36
## 6 378.
```

Tidying the no_of_votes variable

Notice the numbers in the `no_of_votes` variable is also very large, lets change its unit to thousands so it's easier to visualize and analyze.

Show

```
## # A tibble: 6 × 1
##   no_of_votes
##         <dbl>
## 1       2343.
## 2       1620.
## 3       2303.
## 4       1130.
## 5        690.
## 6       1643.
```

Tidying the certificate variable

Notice there are many classes of `certificate` in the data set and some types only have little values. Upon further inspection, it looks like different certificate systems are used, most film certificates are based on Central Board of Film Certification (CBFC) and fewer on Motion Picture Association film rating system (MPAA). Let's convert all the certificate types to base on CBFC so it's consistent through out the dataset.

All certificates are going to be group into 3 types (U, UA, A). `U` for unrestricted public exhibition, suitable for all ages. `UA` for unrestricted public exhibition but with parental guidance for under 12. And `A` for public exhibition to only adults (18+).

Show

```
## 
##       16        A Approved        G       GP   Passed 
PG    PG-13 
##        1      197       11       12        2       34 
37       43 
##        R    TV-14    TV-MA    TV-PG        U      U/A 
UA  Unrated 
##      146        1        1        3      234        1 
175        1
```

Show

Tidying the genre variable

Many movies contains more than one genre, to make our analysis easier let's extract the first genre listed and create a new column with just the primary genre and drop the genre column.

Show

```
## # A tibble: 6 × 8
##   released_year certificate runtime imdb_rating meta_s
core no_of_votes  gross
##           <dbl> <chr>         <dbl>       <dbl>      <
dbl>       <dbl>  <dbl>
## 1          1994 A               142         9.3
80       2343.  28.3
## 2          1972 A               175         9.2
100       1620. 135.
## 3          2008 UA              152         9
84       2303. 535.
## 4          1974 A               202         9
90       1130.  57.3
## 5          1957 U                96         9
96        690.   4.36
## 6          2003 U               201         8.9
94       1643. 378.
## # ℹ 1 more variable: primary_genre <chr>
```

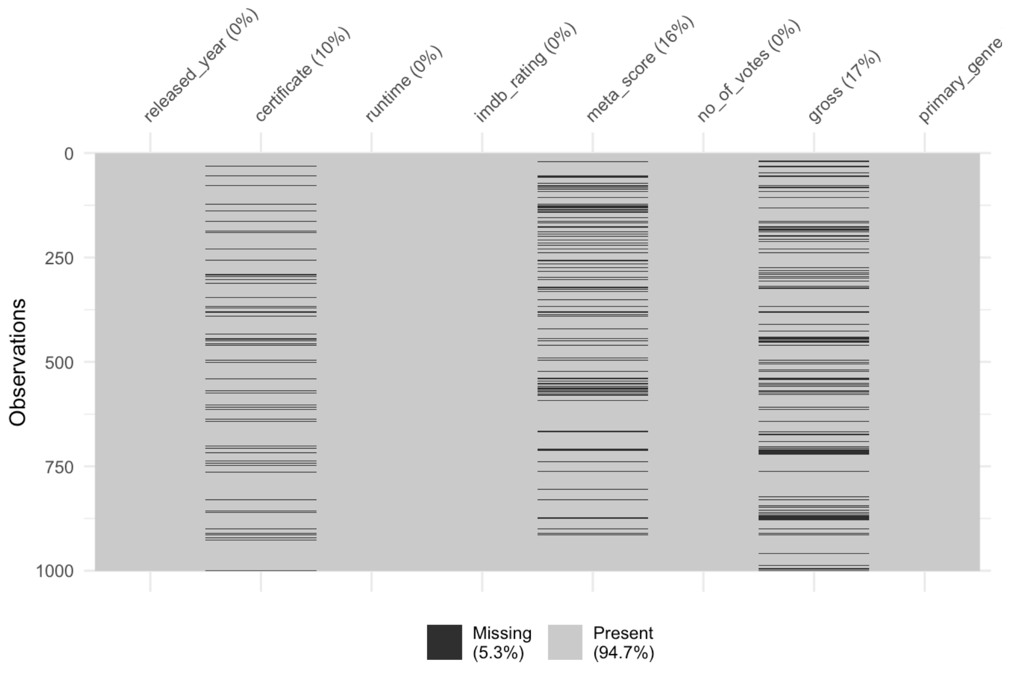
Our data set is now tidied, let's move on to the exploratory data analysis!

Exploratory Data Analysis

Missing data

Let's see if we have any missing values in the data set that could cause issues later on.

Show



The plot shows that we have 5.3% of missing values in the entire data set. The missingness come from 3 variables, `certificate` , `meta_score` and `gross` . Since `gross` is the outcome variable we want to predict, let's remove all the rows with gross missing to reduce bias in our model. Since around 10-20% of data is missing for the other two variables, let's further explore those variables to see how we can fill in those missing values.

Show

Released Year

The released_year seem to have one missing value, let's find the movie title and fill in the released year.

Show

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

Show

```
## [1] 804
```

Show

```
## # A tibble: 1 × 16
##   poster_link   series_title released_year certificate
runtime genre imdb_rating
##   <chr>         <chr>        <chr>         <chr>
<chr>   <chr>        <dbl>
## 1 https://m.me… Rushmore     1998          UA
93 min  Come…          7.7
## # ℹ 9 more variables: overview <chr>, meta_score <dbl
>, director <chr>,
## #   star1 <chr>, star2 <chr>, star3 <chr>, star4 <chr
>, no_of_votes <dbl>,
## #   gross <dbl>
```

Show

Meta score

Meta score is defined by the weighted average of movie reviews of a large group of respected critics, and meta score only exist when a movie had been reviewed by at least 4 critics. So some movies are missing meta score because of the lack of critic reviews. Let's create a new column to indicate if there exist a meta score for the given movie and drop the `meta_score` column.

Show

Certificate

Movies are certified by its suitability for certain audience based on its content, so we can assume that movie certificate are related to the genre of the movie. Let’s create a frequency table of certificate types based on the primary genre.

Show

```
##             A  U UA Mode
## Action     50 16 71   UA
## Adventure 14 31 16    U
## Animation  3 55  8    U
## Biography 31 28 20    A
## Comedy    46 44 32    A
## Crime     62  7 16    A
## Drama     96 50 64    A
## Family     0  2  0    U
## Film-Noir  0  0  1   UA
## Horror     7  0  2    A
## Mystery    5  2  2    A
## Thriller   0  0  0    A
## Western    2  2  0    A
```

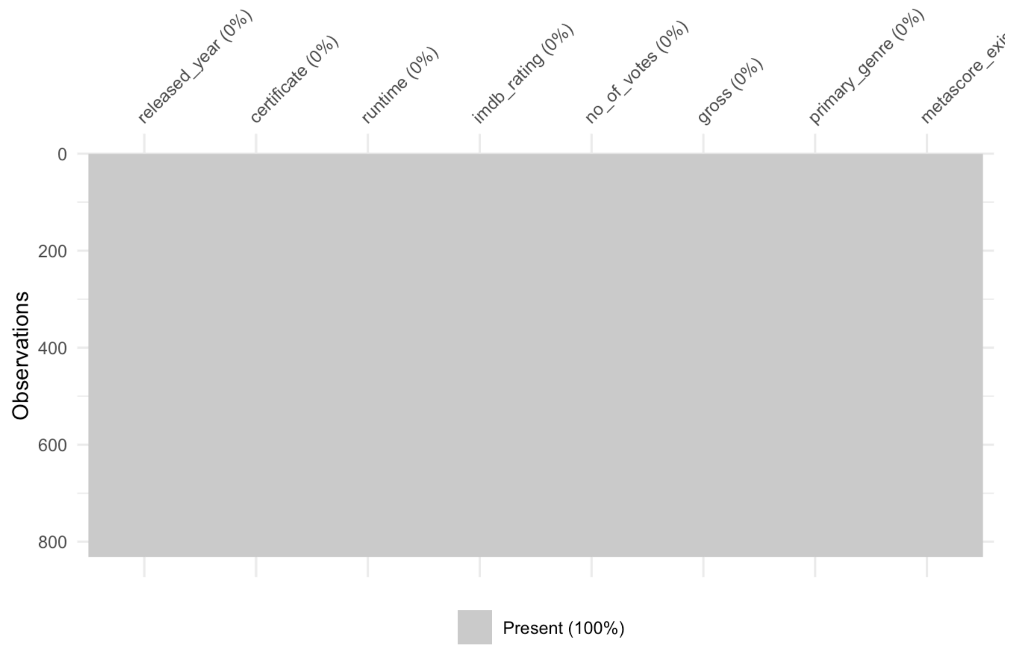
Lets fill in the missing certificate with the mode based on its primary genre.

Show

```
## [1] 0
```

Now let’s check if we have handled all the missing values in the data set.

Show



Now our dataset has no missing values. Let’s look at the five point summary for the numeric variables.

Show

```
##                vars   n    mean     sd  median trimmed
mad     min     max
## released_year    1 831 1993.92  21.14 2000.00 1996.89
17.79 1921.00 2019.00
## runtime          2 831  124.08  27.50  120.00  121.60
25.20   45.00  242.00
## imdb_rating      3 831    7.95   0.28    7.90    7.92
0.30    7.60    9.30
## no_of_votes      4 831  315.25 343.64  186.73  250.56
202.64   25.09 2343.11
## gross            5 831   68.03 109.75   23.53   43.49
33.75    0.00  936.66
##                  range  skew kurtosis    se
## released_year    98.00 -1.18     0.82  0.73
## runtime         197.00  0.96     1.41  0.95
## imdb_rating       1.70  1.08     1.51  0.01
## no_of_votes    2318.02  2.07     5.62 11.92
## gross           936.66  3.12    13.78  3.81
```

After cleaning the data, we ended up with 831 movies and 8 variables. Our dataset contains 5 numeral variables and 3 categorical variables. Looks like the average gross revenue is around 68 million with 936 million as the maximum gross for a movie and 0 as minimum.

Data visualization

First, we need to convert all the categorical variables to factors.

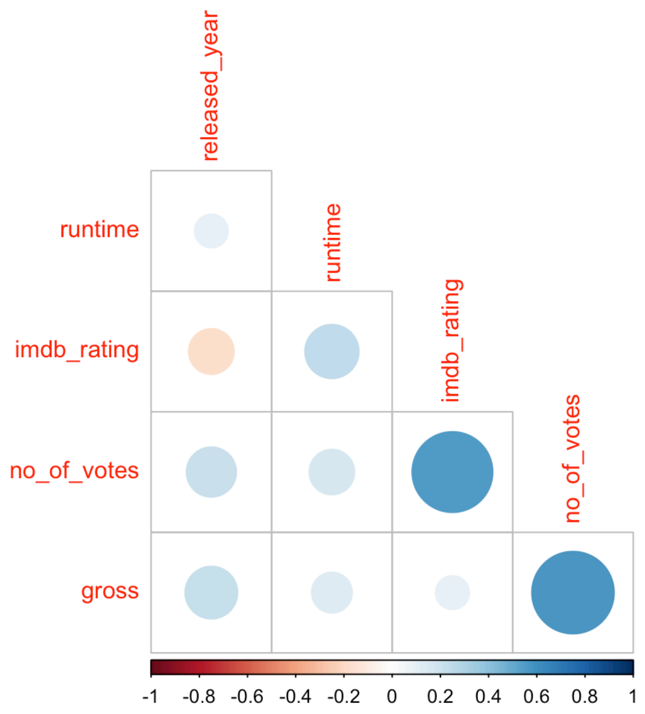
Show

Correlation Plot

Let’s explore the overall correlation between the continuous variables.

Show

```
## Warning: Use of bare predicate functions was deprecate
d in tidyselect 1.1.0.
## ℹ Please use wrap predicates in `where()` instead.
##   # Was:
##   data %>% select(is.numeric)
##
##   # Now:
##   data %>% select(where(is.numeric))
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see whe
re this warning was
## generated.
```



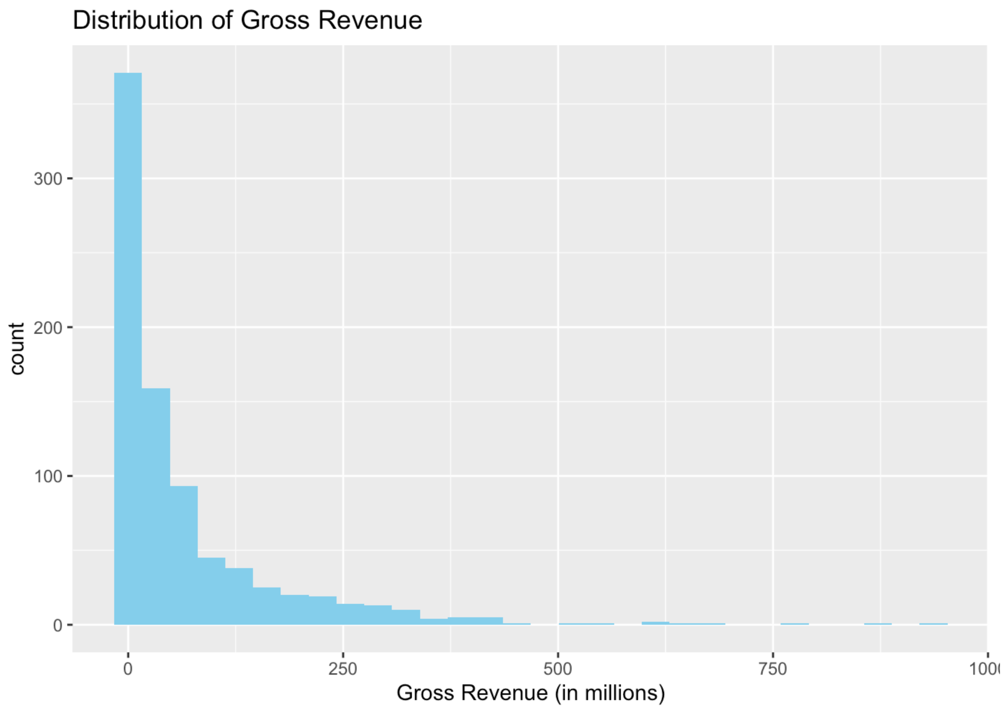
There’s strong positive correlation between variables `no_of_votes` and `imdb_rating`. It makes sense the movies that get more votes gets higher score in `imdb_rating`. There’s also strong correlation between `gross` and `no_of_votes`. So movies that gets higher votes tend to do better in revenue. Which makes sense since higher votes means higher popularity, and we can assume that higher popularity in movies will do better in gross revenue. There’s also positive correlation between movie `released_ year` and `gross`, so newer movies tend to get higher gross. This also makes sense since cinema prices increases overtime.

Gross revenue

Let’s explore the distribution of our outcome variable `gross`.

Show

```
## `stat_bin()` using `bins = 30`. Pick better value with
`binwidth`.
```

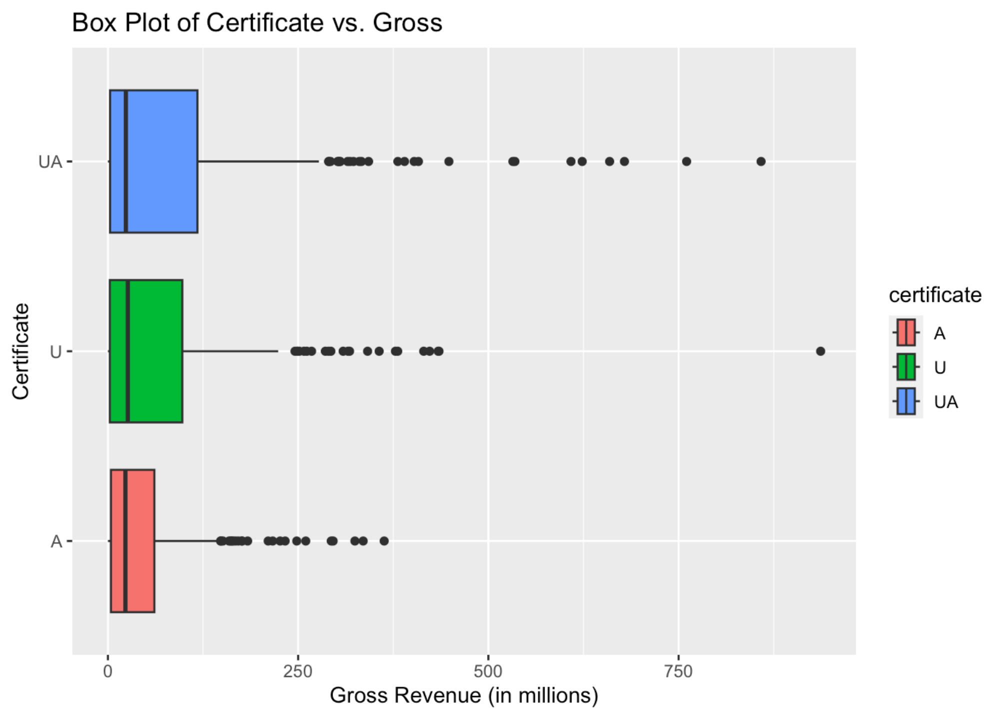


The possible values of `gross` ranges 0 to 1000 million. The distribution of gross revenue is left-skewed, meaning most movie gross revenue lies in the lower range below 100 million and fewer movies gets over 250 million in revenue.

Certificate

We grouped the certificate types into 3 classes, “U”, “UA”, and “A”. Let’s take a look at it’s relationship with gross revenue and see if certain certificate types have higher gross revenue than others.

Show

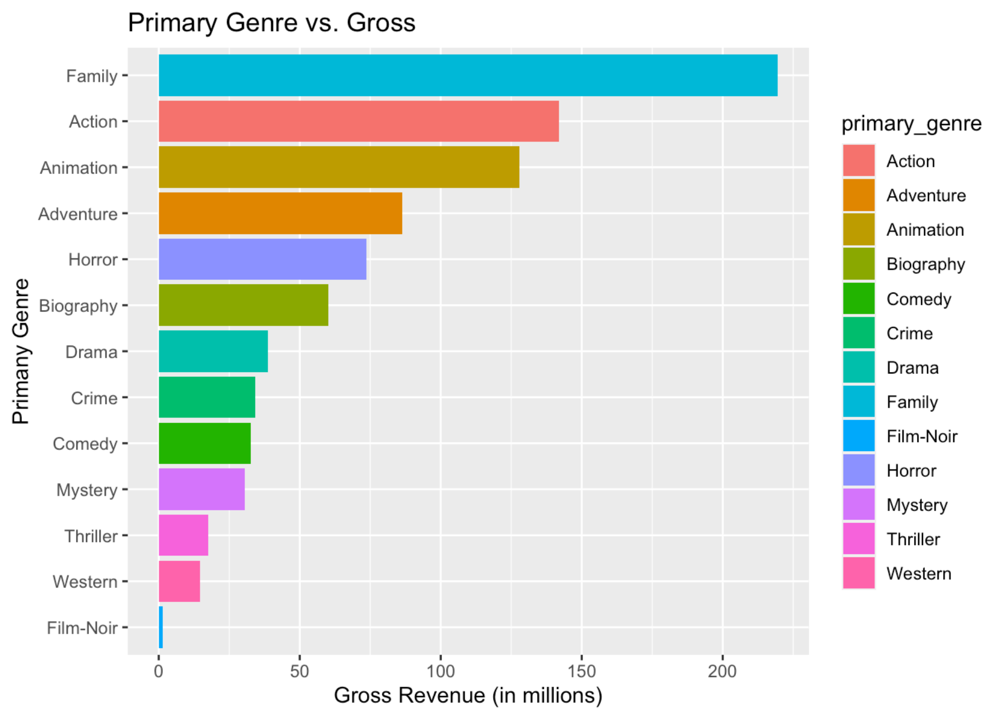


The plot shows the quantiles and medians of gross revenues of different certificate types. It shows that the gross revenue of movies with UA certificate ranges the most and have a higher third quantile on average compare to the other certificate types. Most outliers also come from the UA rated movies. This makes sense since UA rating are suitable for a wider audience.

Primary Genre and Gross

Let's visualize the relationship between genre and gross revenue.

Show

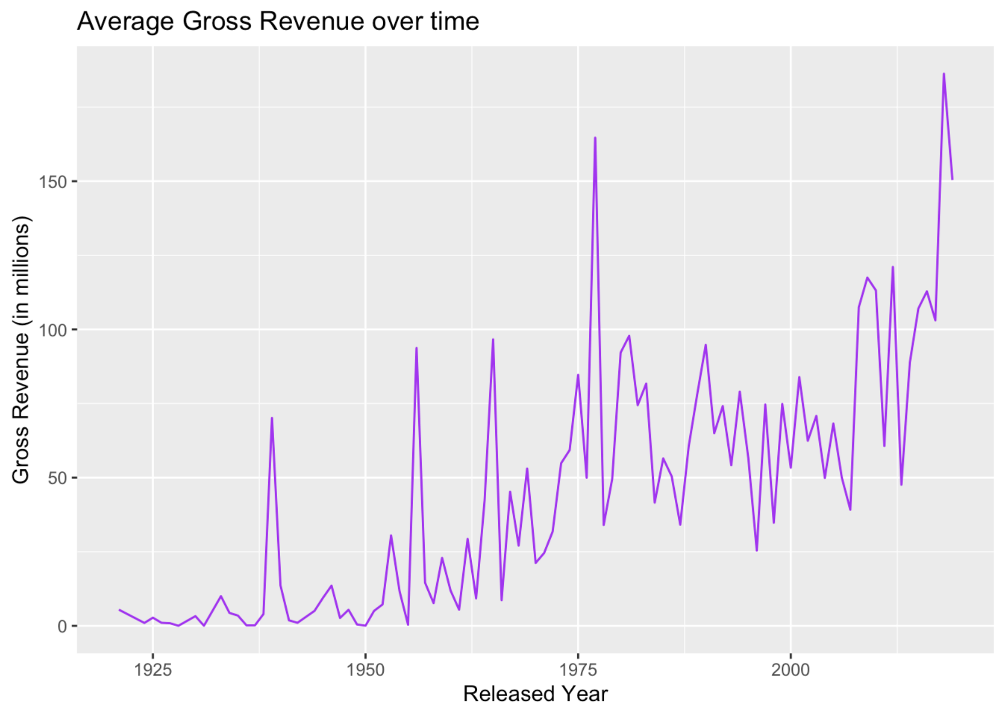


There is a total of 13 primary genres. On average the family genre gets the highest gross revenue, and the film-noir genre gets the lowest gross revenue. The top 3 genres are Family, Action and animation. The genres that tend to do worse in revenue is thriller, western and film-noir.

Released Years

Let's visualize the relationship of `gross` and `released_year`.

Show



Gross revenue has fluctuated over the years but there's a clear upward trend. Which makes sense since cinema prices increases overtime and the filming industry/market is growing overtime. Notably, there's a significant spike around 1975 where revenue surpasses 150 million. However this peak is not replicated until recent years.

Model Building

Split data

First, let's split the data into training and testing set. The training set is used for training our models and the testing set is used in the end to test how well our models predict the outcome variable on unseen data. I split the data into 70/30, 70% are used in training and 30% of the data are used for testing.

Show

```
## <Training/Testing/Total>
## <579/252/831>
```

Write the recipe

Let’s create a universal recipe to use for all of our models. We will be using 7 predictors, `released_year` , `certificate` , `runtime` , `genre` , `imdb_rating` , `meta_score` , and `no_of_votes` to predict `gross` . We will make the categorical variables `certificate` , `genre` and `meta_score` into dummy variables. Since some variables contain only one value, I use the zero variance filter to filter out the variables that contain only a single value. Finally, we’ll normalize all variables by centering and scaling.

Show

```
## # A tibble: 579 × 19
##    released_year runtime imdb_rating no_of_votes  gros
s certificate_U
##            <dbl>   <dbl>       <dbl>       <dbl>  <dbl
>         <dbl>
##  1        -1.92   2.97          2.32    -0.00742 0.269
1.57
##  2        -3.01  -1.35          1.97    -0.430   0.019
2         1.57
##  3         1.14   0.0514        1.61    -0.731   1.66
-0.636
##  4         0.613  1.46          1.61    -0.427   1.22
1.57
##  5         0.422 -0.850         1.61     0.564   0.707
-0.636
##  6         0.136  0.340         1.61     0.0710  2.38
1.57
##  7        -1.44  -1.07          1.61     0.378   0.276
-0.636
##  8         0.136 -1.28          1.25    -0.723   0.934
1.57
##  9        -0.103 -0.922         1.25     1.72    2.83
-0.636
## 10        -2.01   0.664         1.25    -0.714   0.055
2        -0.636
## # ℹ 569 more rows
## # ℹ 13 more variables: certificate_UA <dbl>, primary_g
enre_Adventure <dbl>,
## #   primary_genre_Animation <dbl>, primary_genre_Biogr
aphy <dbl>,
## #   primary_genre_Comedy <dbl>, primary_genre_Crime <d
bl>,
## #   primary_genre_Drama <dbl>, primary_genre_Film.Noir
<dbl>,
## #   primary_genre_Horror <dbl>, primary_genre_Mystery
<dbl>,
## #   primary_genre_Thriller <dbl>, primary_genre_Wester
n <dbl>, …
```

K-fold Cross Validation

We will create 10 folds for k-fold stratified cross validation. This means that the dataset is divided into 10 equally folds into validation sets with each fold being a testing set with the other k-1 folds being the training set for that fold. After each training-validation iteration, the average accuracy is taken from the testing set of each of the folds to evaluate the performance of the model. By using k-fold cross validation, we can get a more reliable and stable estimate of the model’s performance to new unseen data.

We stratify on the outcome variable `gross` to ensure we fold the data proportionally.

Show

Setting up models

Now let's start setting up our models! We will be building five models, linear regression, k-nearest neighbors, elastic net, random forest and boosted tree. Since some models will take very long time to run, I will save each model result to avoid needing to rerun the models every time. After fitting the models I will evaluate their performances with Root Mean Squared Error (RMSE). RMSE measures how far the model's predictions are from the true values using Euclidian distance, it is one of the most commonly used metric for evaluating the performance of regression models.

Setting up models by specifying the model, setting up appropriate mode and engine, as well as the parameters we want to tune.

Show

Setting up workflow

Let's set up the workflow for each model and add the model and the recipe.

Show

Tuning grid

Let's create a tuning grid with the ranges for the parameters we wish to tune and the number of levels.

Show

Fit all models to the folded data

Let's fit all the models and tune the models with our tuning grid.

Show

Show

Save the models

Save the model results so we don't need to rerun every time.

Show

Load the models

Load the models back in.

Show

Collect Metrics

Let’s select the best model for each model types we fitted using the RMSE metric.

Show

Model results

Now that we have fitted all our models, let’s create a RMSE table to compare the model performances!

Show

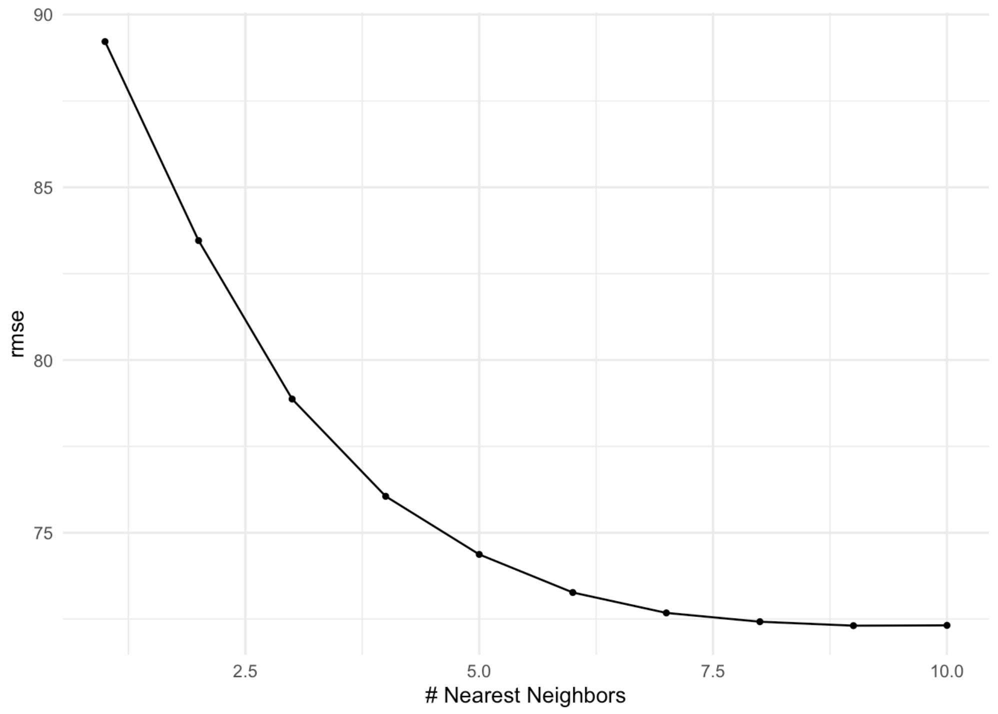
```
## # A tibble: 5 × 2
##   Model                RMSE
##   <chr>               <dbl>
## 1 Boosted Tree         62.3
## 2 Random Forest        66.2
## 3 Elastic Net          69.5
## 4 Linear Regression    69.7
## 5 K Nearest Neighbors  72.3
```

Overall the boosted model has the lowest RMSE on the cross-validation data, so we can conclude that the boosted tree model model had the best performance! It looks like the simpler models like linear regression and K nearest neighbors performed the worse, probably indicating our data is not linear.

Autoplots

K-nearest Neighbor

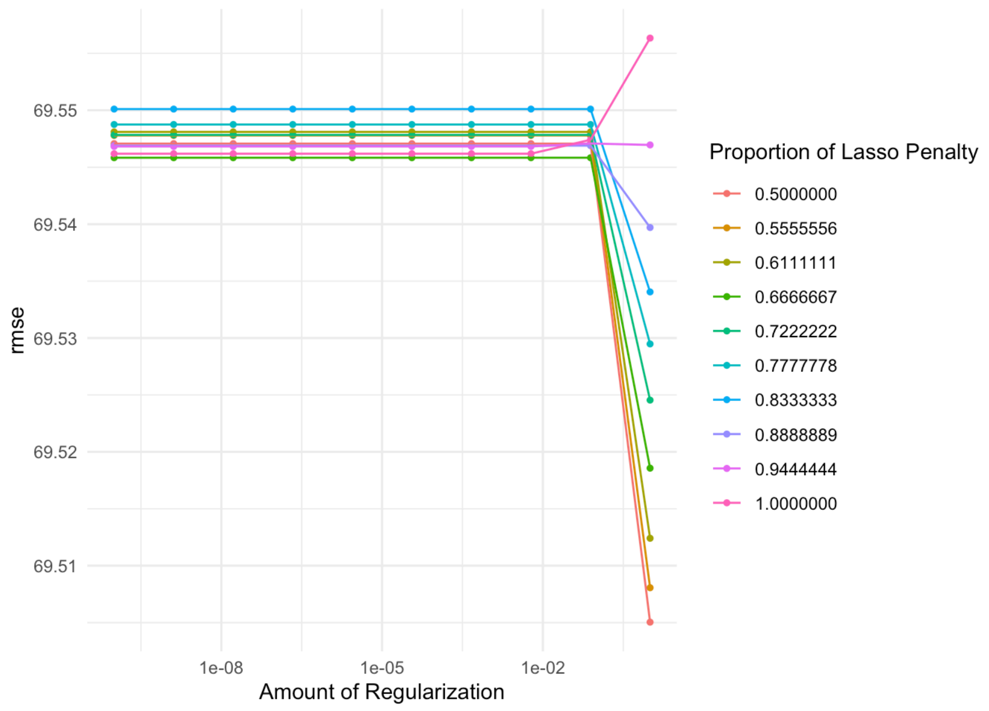
Show



From the plot we can see that as the number of neighbors increases, our model performance increases and the RMSE decreases. But even with 10 neighbors, the RMSE is still around 70.

Elastic Net

Show



We tuned the elastic net model with penalty and mixture at 10 levels. From the plot, it looks like from higher range of penalty value tend to do better. For lower penalty, mid to lower-range of mixture tend to get lower RMSE and as penalty increases, lower mixture values tend to get decreasing

Introduction

Exploratory Data Analysis

Model Building

Model results

Results From the Best Model

Visualizing Final Model Performance

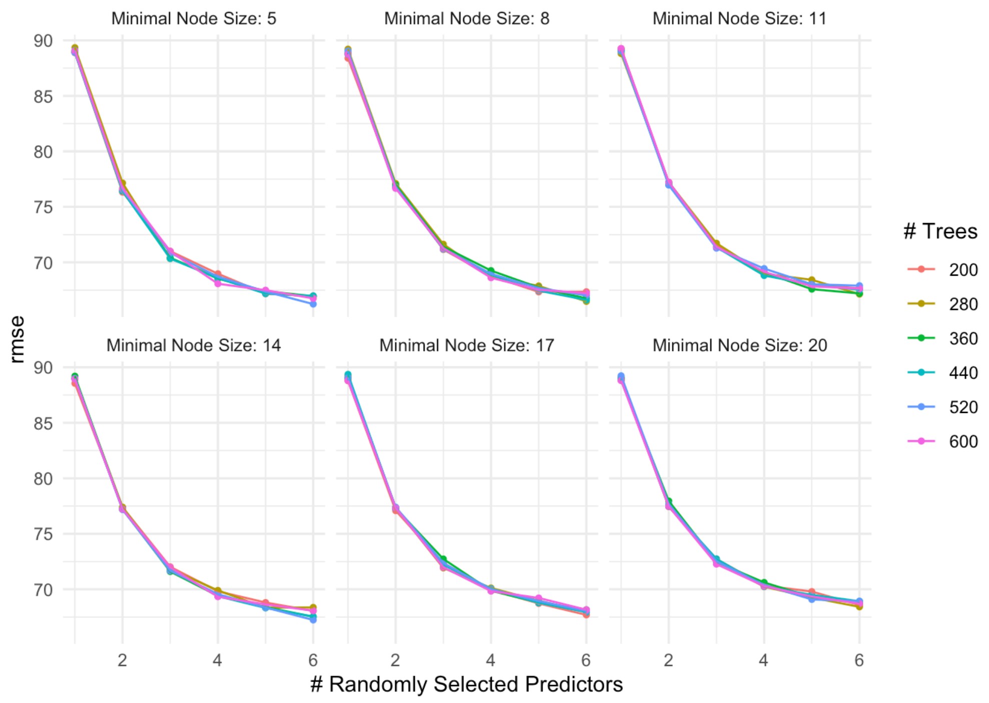
Conclusion

Sources

RMSE and higher mixture value that's closer to 1 gets increasing RMSE. So it seems that lower values of mixture only perform better than higher values of mixture when the penalty term is very high.

Random Forest

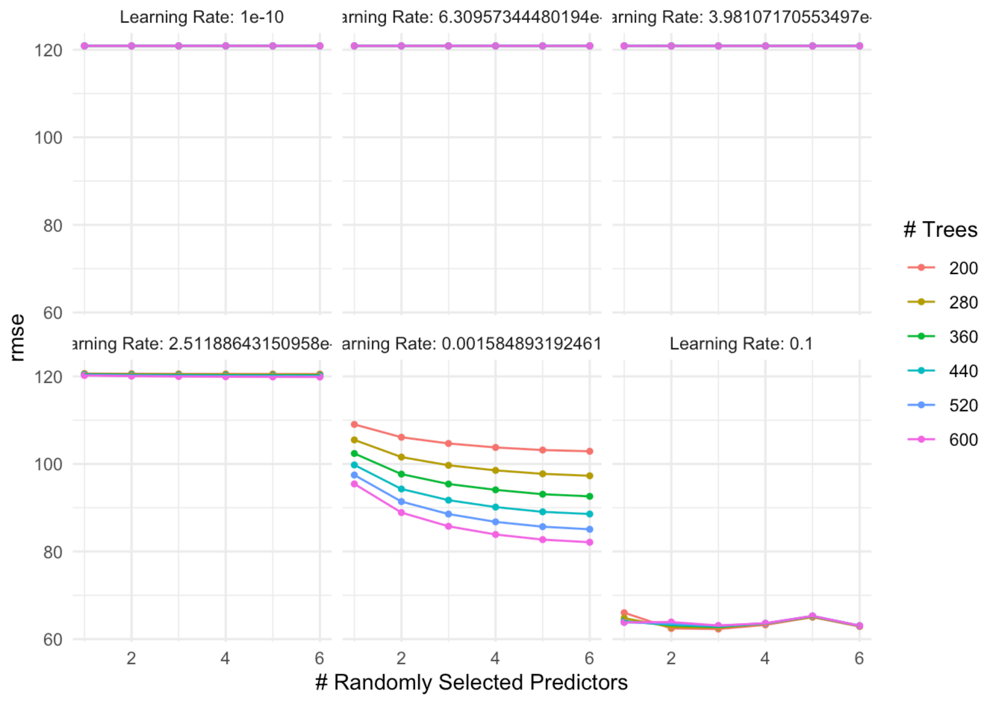
Show



For random forest, we tuned the minimal node size, number trees and number of randomly selected predictors. Since we have 7 predictors in total, I choose the range of mtry to be between 1 and 6 so not all predictors are used in the first split. This is to prevent all trees to have the same first split and keep the trees to be independent from one another. From the plot, it looks like the number of trees doesn't make much of a difference in the model performance. The minimal node size doesn't appear to have a huge effect on the performance but lower values of `min_n` appear to have slightly lower RMSE. The number of mtry appear to have a huge affect on model performance, higher number of predictor did significantly better.

Boosted Tree

Show



For the boosted tree, we tuned the minimal node size, number of trees and learning rate. At lower learning rate, the number of trees and mtry didn't make much of a difference in model performance. As learning rate increases, the RMSE significantly gets lower. Which means the model does better when it's learning faster. Higher number of trees performed better when the learning rate is at 0.0015 but has no effect when learning rate is at 0.1. At the learning rate of 0.0015, as the number of mtry increases, the RMSE decreases. But at learning rate of 0.1, as the mtry increases, the RMSE decreases slightly then increases slightly then goes back down.

Results From the Best Model

So the boosted tree model performed the best out of the 5 models we built.

Show

```
## # A tibble: 1 × 9
##    mtry trees learn_rate .metric .estimator  mean
n std_err .config
##   <int> <int>      <dbl> <chr>   <chr>      <dbl> <int
>   <dbl> <chr>
## 1     3   200        0.1 rmse    standard    62.3    1
0    3.99 Preprocessor1_M…
```

Boosted tree #103 with 3 predictors, 200 trees and 0.1 learning rate performed the best with an RMSE of 62.30871!

Fitting to Training Data

Now, we will take the best performed model and fit it to the training set. This trains our model again on the entire training set.

Show

Fitting to Testing Data

Now let’s take out finalize model and fit it to the testing set to see how well our model performed on unseen data!

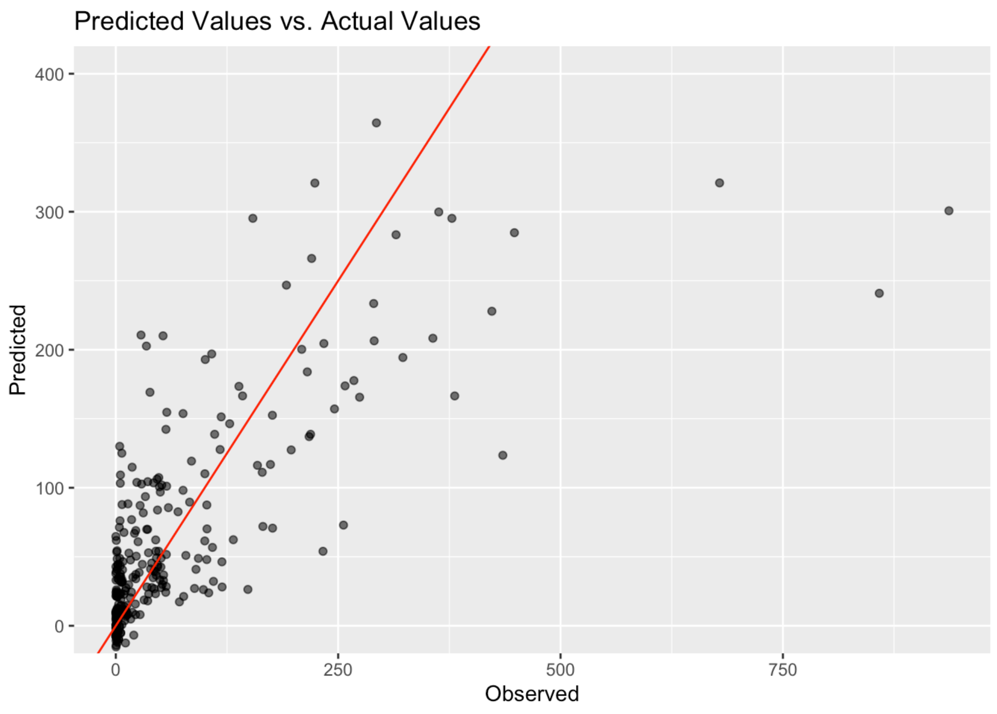
Show

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        84.5
```

Looks like our boosted tree model actually performed worse on the testing set than the cross-validation set with RMSE of 84.5!

Visualizing Final Model Performance

Show



If our model predicted every observation accurately, the points would form a straight line. However, from the plot we can see that very few points fall on the line, especially when gross gets larger. That makes sense since we have very little data for when gross revenue goes over 250 million. It looks like our model didn't really performed well since many points didn't follow the line.

Conclusion

After fitting and analyzing 5 different types of models, the best model to predict gross revenue of movies is the boosted tree model. This is not surprising since boosted tree models tend to do good for a wide range of data types and tend to handle high-dimensional data well. However, this model still didn't perform well in predicting the gross revenue of movies. Overall, our model produces a higher RMSE on the testing set compared to the cross-validation set, this indicates that the model may not generalize as well to unseen data. It also suggests that the model might be over-fitting to the training data, it captures noise in the training data rather than the underlying pattern.

For future research, I would like to explore more predictors such as authors and directors to see how the popularity of these figures featured in film can affect its success. I believe the low correlation between the predictors used and `gross` might contribute to the model performance, since only number of votes have a significant correlation with `gross` . I would also like to explore more models and try out different modeling techniques. Although this model did not perform quite well, I still find this project very interesting. This was a great opportunity for me to build my machine learning modeling skills while gaining valuable insights about movies in general.

Sources

The data set was taken from Kaggle (https://www.kaggle.com/datasets/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows), by user HARSHIT SHANKHDHAR. The data was extracted from the IMDB website.

Information about certificate types were found on wikipedia. (CBFC (https://en.wikipedia.org/wiki/Central_Board_of_Film_Certification), MPAA Film (https://en.wikipedia.org/wiki/Motion_Picture_Association_film_rating_system) and MPAA TV (https://en.wikipedia.org/wiki/TV_Parental_Guidelines))