Predicting the gross of top 1000 movies in IMDB

PSTAT 131 Final Project

Tingting Liang

2024-01-31

Contents

Introduction

(2 paragraphs, define any necessary vocabulary terms, briefly explain relevant concepts, and discuss the primary goals of the project)

What is imdb?

What is our aim? The aim of this project is to predict the gross of the top 1000 movies and TV shows in the IMDB ranking. I will be working with the IMDB Movies Dataset from Kaggle in this project. The dataset includes top 1000 songs by IMDB rating.

Why?

Loading and Exploring the Data

Lets take a look at the data set. And use clean_names() to clean up the variable names.

```r
# load packages
library(tidyverse)
```

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

```
library(tidymodels)
```

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

```
library(janitor)
```

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

```
library(visdat)
library(kknn)
library(yardstick)
library(ggplot2)
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(psych)
```

```
##
## Attaching package: 'psych'
##
## The following objects are masked from 'package:scales':
##
##     alpha, rescale
##
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```r
library(discrim)
```

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

```r
library(themis)
library(forcats)
library(dplyr)
library(xgboost)
```

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

```r
# load data
imdb_data = read_csv("data/imdb_top_1000.csv")
```

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

```r
set.seed(123)

imdb_data %>%
  head()
```

```
## # A tibble: 6 x 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>
```

```
imdb <- imdb_data %>%
  clean_names()
head(imdb)
```

```
## # A tibble: 6 x 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>
```

Exploratory and Tidying the Raw Data

Variable selection

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

```
dim(imdb)
```

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

The 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 star variables aren't really relevant for predicting our outcome gross revenue, so we'll be dropping those columns.

```
imdb <- imdb %>%
  select(-c("poster_link", "series_title",  "overview", "director", "star1", "star2", "star3", "star4"))

dim(imdb)
```

```
## [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.

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

```r
sapply(imdb, class)
```

```
## released_year   certificate       runtime         genre   imdb_rating
##   "character"   "character"   "character"   "character"     "numeric"
##    meta_score   no_of_votes         gross
##     "numeric"     "numeric"     "numeric"
```

We have a data set with both categorical and numeric columns. The catgorical columns are `released_year`, `certificate`, `runtime`, and `genre`. The numeric columns are `imdb_rating`, `meta_score`, `no_of_votes`, and `gross`. Since `Runtime` and `Released_Year` contains numbers that may be helpful for our gross revenue prediction, lets covert them to numerial type so it's easier to work with.

```r
# Remove 'min' from runtime
imdb$runtime <- gsub("min","",as.character(imdb$runtime))
head(imdb$runtime)
```

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

```r
# convert runtime and released year to numerial type
imdb$runtime <- as.numeric(imdb$runtime)
class(imdb$runtime)
```

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

```r
imdb$released_year <- as.numeric(imdb$released_year)
```

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

```r
class(imdb$released_year)
```

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

Tidying the Outcome variable

Notice the numbers in our outcome variable `gross` is very large, lets change the unit to millions by moving the decimal back 6 times so it's easier to visualize and analyze.

```r
imdb['gross'] = imdb['gross']*(10**-6)
imdb['gross']
```

```
## # A tibble: 1,000 x 1
##     gross
##     <dbl>
##  1  28.3
##  2 135.
##  3 535.
##  4  57.3
##  5   4.36
##  6 378.
##  7 108.
##  8  96.9
##  9 293.
## 10  37.0
## # i 990 more rows
```

Tidying the Number of Votes variable

Notice the numbers in the `no_of_votes` variable is very large, lets also change the unit to millions by moving the decimal back 6 times so it's easier to visualize and analyze.

```
imdb['no_of_votes'] = imdb['no_of_votes']*(10**-6)
imdb['no_of_votes']
```

```
## # A tibble: 1,000 x 1
##    no_of_votes
##          <dbl>
##  1       2.34
##  2       1.62
##  3       2.30
##  4       1.13
##  5       0.690
##  6       1.64
##  7       1.83
##  8       1.21
##  9       2.07
## 10       1.85
## # i 990 more rows
```

Tidying the certificate variable

Upon further inspection, notice there are many types of `certificate` in the data set but some types only have little values. There's also some inconsistency in how UA is written. Let's change U/A to UA and handle these rarer classes by lumping them together into an 'other' category. There are 4 popular certificate types that has contains over 100 movies so let's lump all the other types together except for the top 4 most frequent.

```
# look at counts of each certificate type
table(imdb$certificate)
```

```
##
##       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
```

```
# make U/A = UA
imdb$certificate[imdb$certificate == 'U/A'] <- 'UA'

# Grouping popular classes and rarer classes as other
imdb <- imdb %>%
```

```r
  filter(!is.na(certificate)) %>%
  mutate(certificate = fct_lump(certificate, n = 4))

table(imdb$certificate)
```

```
##
##     A     R     U    UA Other
##   197   146   234   176   146
```

Tidying the genre variable

Some movies contain more than one genre so the genre variable is quite messy to work with. Let's create a new column with just the primary genre and drop the genre column.

```r
imdb['primary_genre'] = str_split_i(imdb$genre, ', ', 1)
imdb <- imdb %>%
  select(-c("genre"))
```

Let's take a look at our tidied dataset.

```r
imdb %>%
  head()
```

```
## # A tibble: 6 x 8
##   released_year certificate runtime imdb_rating meta_score no_of_votes  gross
##           <dbl> <fct>         <dbl>       <dbl>      <dbl>       <dbl>  <dbl>
## 1          1994 A               142         9.3         80       2.34   28.3
## 2          1972 A               175         9.2        100       1.62  135.
## 3          2008 UA              152         9           84       2.30  535.
## 4          1974 A               202         9           90       1.13   57.3
## 5          1957 U                96         9           96       0.690   4.36
## 6          2003 U               201         8.9         94       1.64  378.
## # i 1 more variable: primary_genre <chr>
```

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

Exploratory Data Analysis

(3-5 plots or tables are included in the EDA. They can be the same type or different types. For example, a correlation matrix, a histogram, a scatterplot, box plot, etc.)

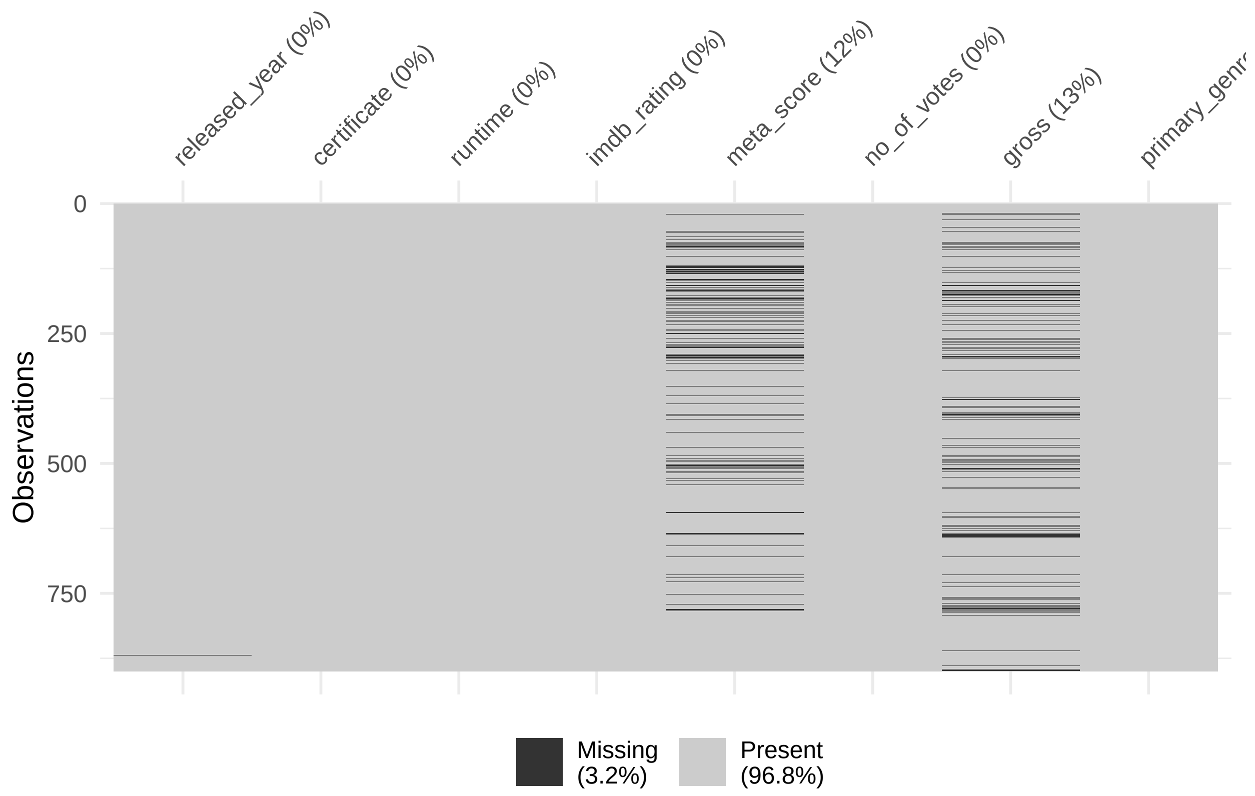
First, we need to convert all the categorical variables to factors.

```r
# Changing categorical variables to factors
imdb <- imdb %>%
  mutate(certificate = factor(certificate),
        primary_genre = factor(primary_genre))
```

Missing data

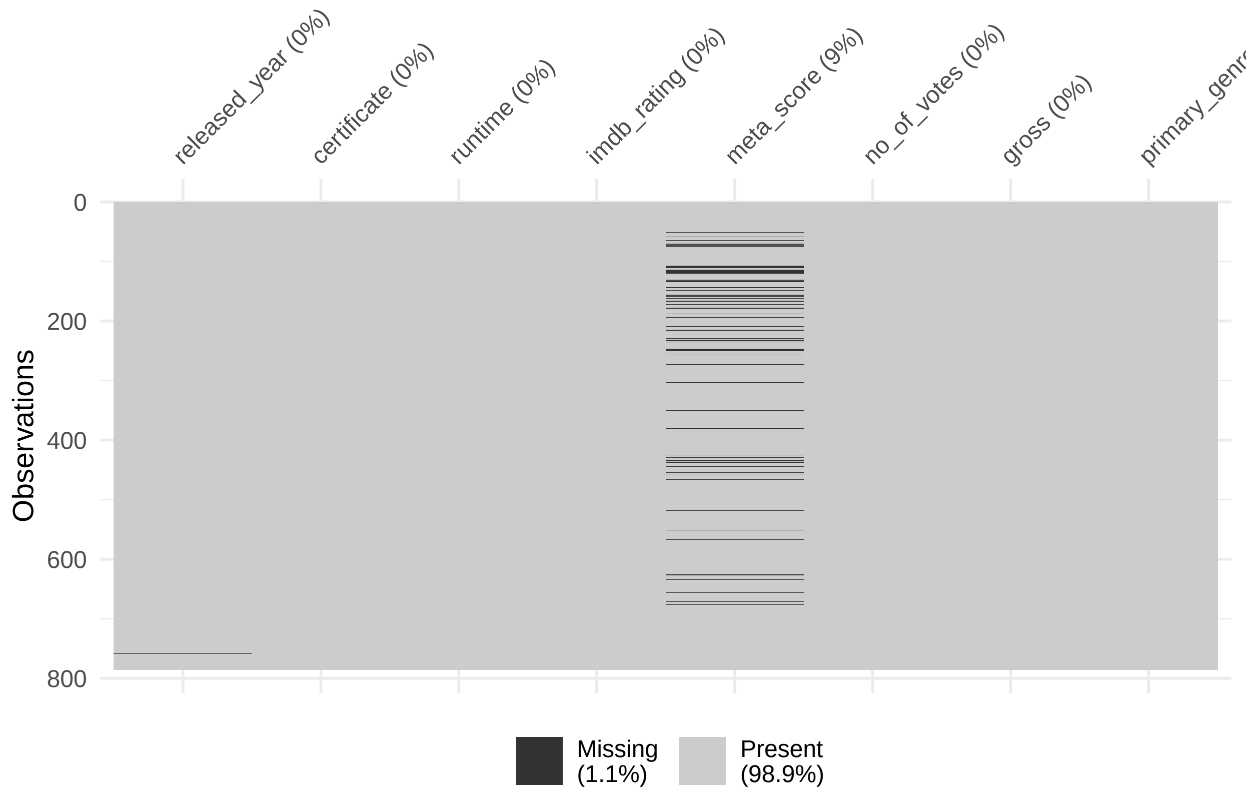
Now let's see if we have any missing values in the data set that can cause issue later on.

```r
imdb %>%
  vis_miss()
```



The plot shows that we have 2.8% of missing values in the entire data set. The missing-ness is from 2 variables `meta_score` and `gross`. Since `gross` is the outcome variable we want to predict, let's remove all the rows with gross missing. It looks like variables `meta_score` are missing on random with relatively small proportions of missing data, we can deal with this by imputing the missing values in our recipe step.

```r
# remove missing gross rows
imdb = imdb[!is.na(imdb$gross),]

# checking if we removed all missing gross
imdb %>%
  vis_miss()
```



After removing the missing gross rows, we only have 1.1% of missing data from the entire imdb data set from the `meta_score` variable. About 9% of meta score is missing, wi one missing value from the `released_year` variable. we will handle this in the recipe step by imputing it with linear regression.

Lets look at the five point summary for the numeric variables.

```r
select_if(imdb, is.numeric) %>% describe()
```

```
##               vars   n    mean     sd  median trimmed   mad     min     max
## released_year    1 784 1994.84  20.22 2001.00 1997.69 16.31 1921.00 2019.00
## runtime          2 785  124.62  27.44  121.00  122.09 25.20   45.00  242.00
## imdb_rating      3 785    7.95   0.29    7.90    7.92  0.30    7.60    9.30
## meta_score       4 714   77.16  12.40   78.00   77.82 11.86   28.00  100.00
## no_of_votes      5 785    0.33   0.35    0.20    0.27  0.22    0.03    2.34
## gross            6 785   71.81 111.75   26.86   47.36 37.81    0.00  936.66
##                range  skew kurtosis   se
## released_year  98.00 -1.23     1.14 0.72
## runtime       197.00  0.98     1.47 0.98
## imdb_rating     1.70  1.07     1.44 0.01
## meta_score     72.00 -0.58     0.47 0.46
## no_of_votes     2.32  2.01     5.33 0.01
## gross         936.66  3.04    13.11 3.99
```

Looks like the average gross revenue is around 71 million with 936 million as the maximum gross for a movie and 0 as minimum.

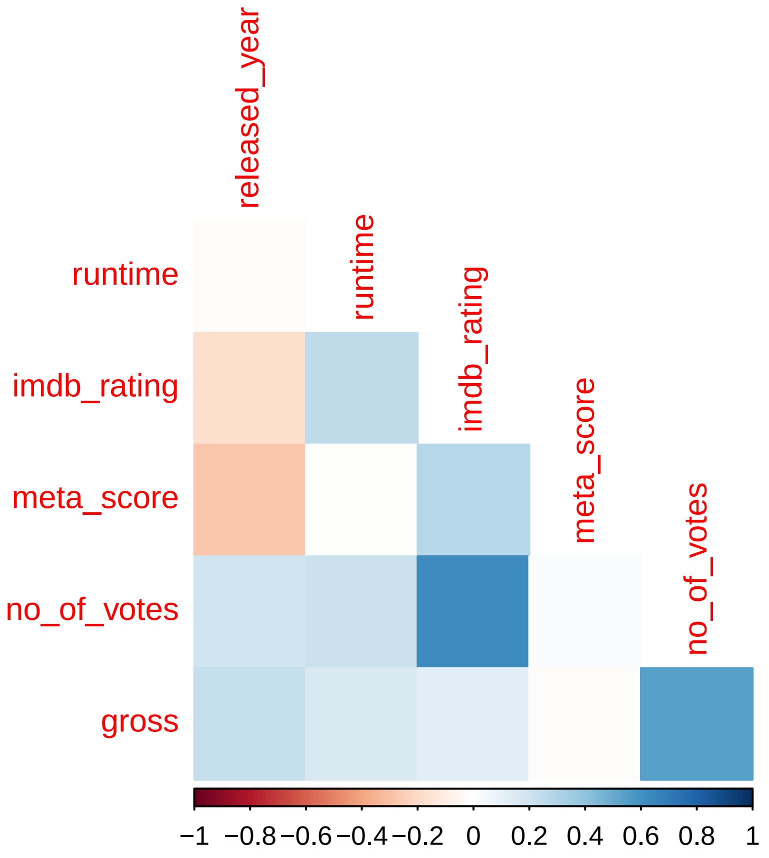
Data visualization

Correlation Plot

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

```
imdb %>%
  select(is.numeric) %>%
  cor(use = "complete.obs") %>%
  corrplot(type = 'lower', diag = FALSE,
           method = 'color')
```

```
## Warning: Use of bare predicate functions was deprecated 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 where this warning was
## generated.
```



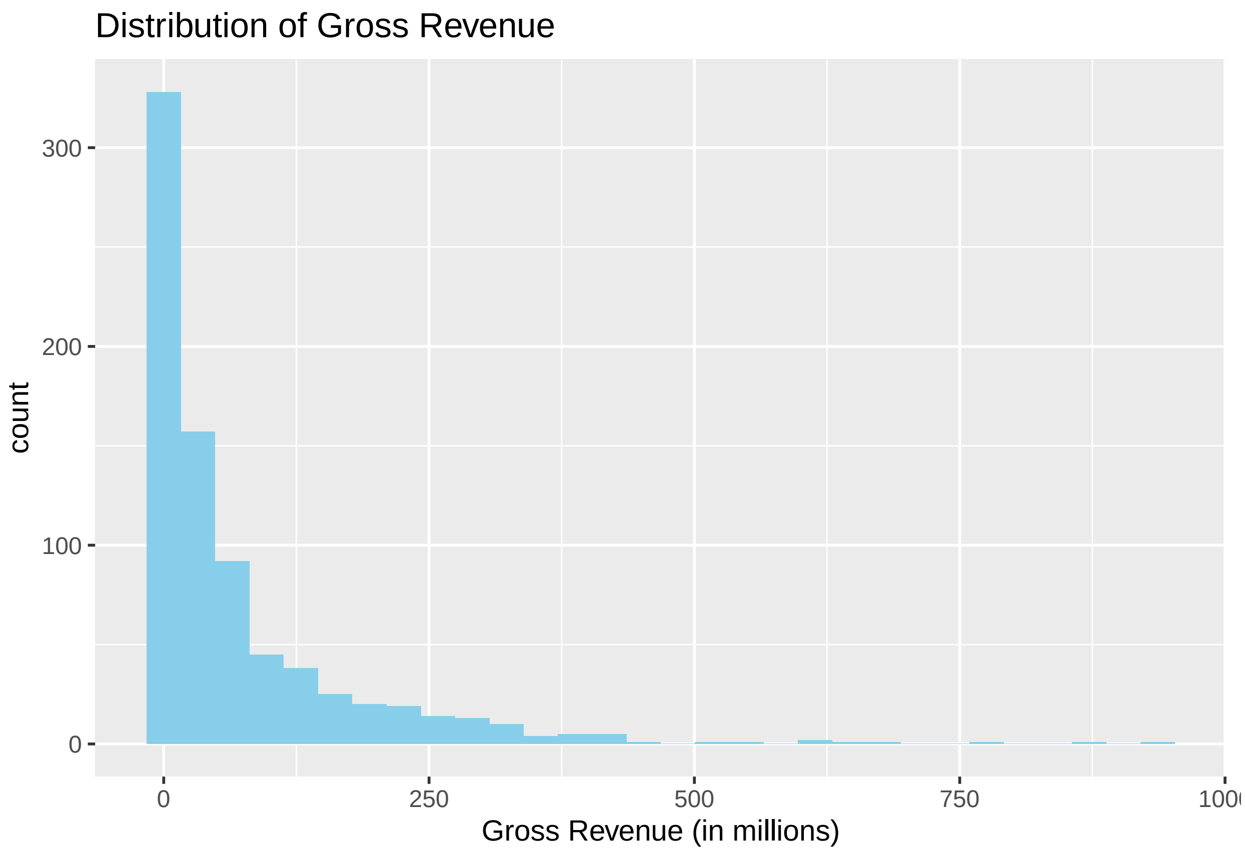
Looks like there's string positive correlation between variables `no_of_votes` and `imdb_rating`. It makes sense the series that get higher number of votes are higher in `imdb_rating`. There's also strong correlation between `gross` and `no_of_votes`. Looks like there's negative correlation between `meta_score` and `released_year` so older series seem to be higher meta scores.

Gross revenue

Let's explore the distribution of the outcome variable `gross`.

```r
# Distribution of gross revenue
imdb %>% ggplot(aes(gross)) +
  geom_histogram(fill='skyblue') +
  labs(title = 'Distribution of Gross Revenue',
       x = 'Gross Revenue (in millions)'
  )
```

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

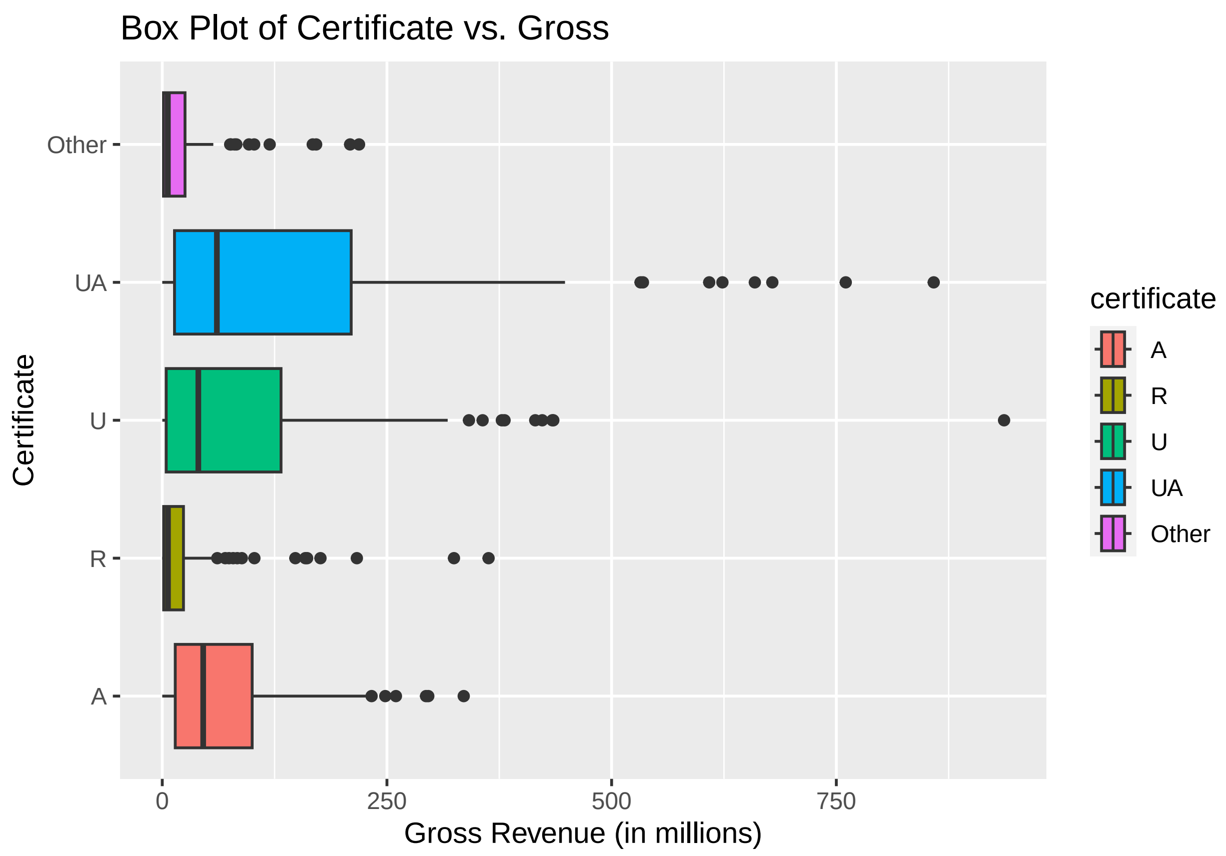


The possible values of `gross` ranges 0 to 1000, since the unit is in millions meaning the possible gross revenue ranges from 0 to 1 billion. Looks like the most common gross revenue is between 0 and 100 million.

Certificate

We grouped the certificate types into 4 most popular certificate classes, "U", "UA", "A" and "R" and others. Let's take a look at it's relationship with gross revenue and see if certain certificate classes have higher gross revenue than others.

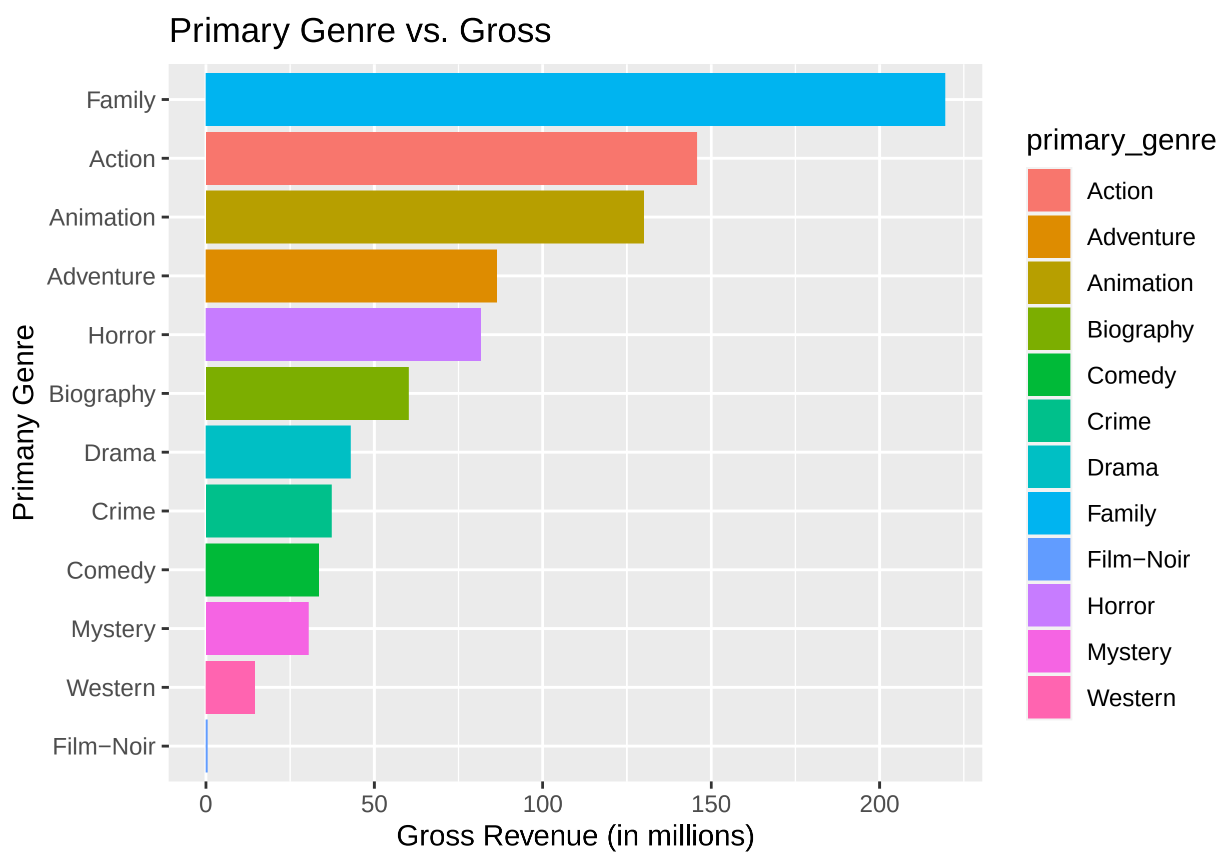
```r
# Boxplot of Gross and Certificate
imdb %>%
  ggplot(aes(x=gross, y=certificate, fill=certificate)) +
  geom_boxplot() + labs(title="Box Plot of Certificate vs. Gross",
                        y='Certificate',
                        x='Gross Revenue (in millions)')
```



The plot shows the maxima, minima and medians of gross revenues of different certificate classes. It shows that the gross revenue of UA certificate movies ranges the most and have a higher maximum on average compare to the other certificate classes. This makes sense since UA rating are quite universal and suitable for general public.

Primary Genre and Gross

```r
# calculate gross revenue average for each primary genre
genre_mean <- aggregate(gross ~ primary_genre, imdb, mean)

genre_mean %>%
  ggplot(aes(x=reorder(primary_genre, gross), y=gross, fill= primary_genre)) +
  geom_bar(stat="identity") + coord_flip( ) +
  labs(title="Primary Genre vs. Gross ",
       x = 'Primany Genre',
       y = 'Gross Revenue (in millions)')
```



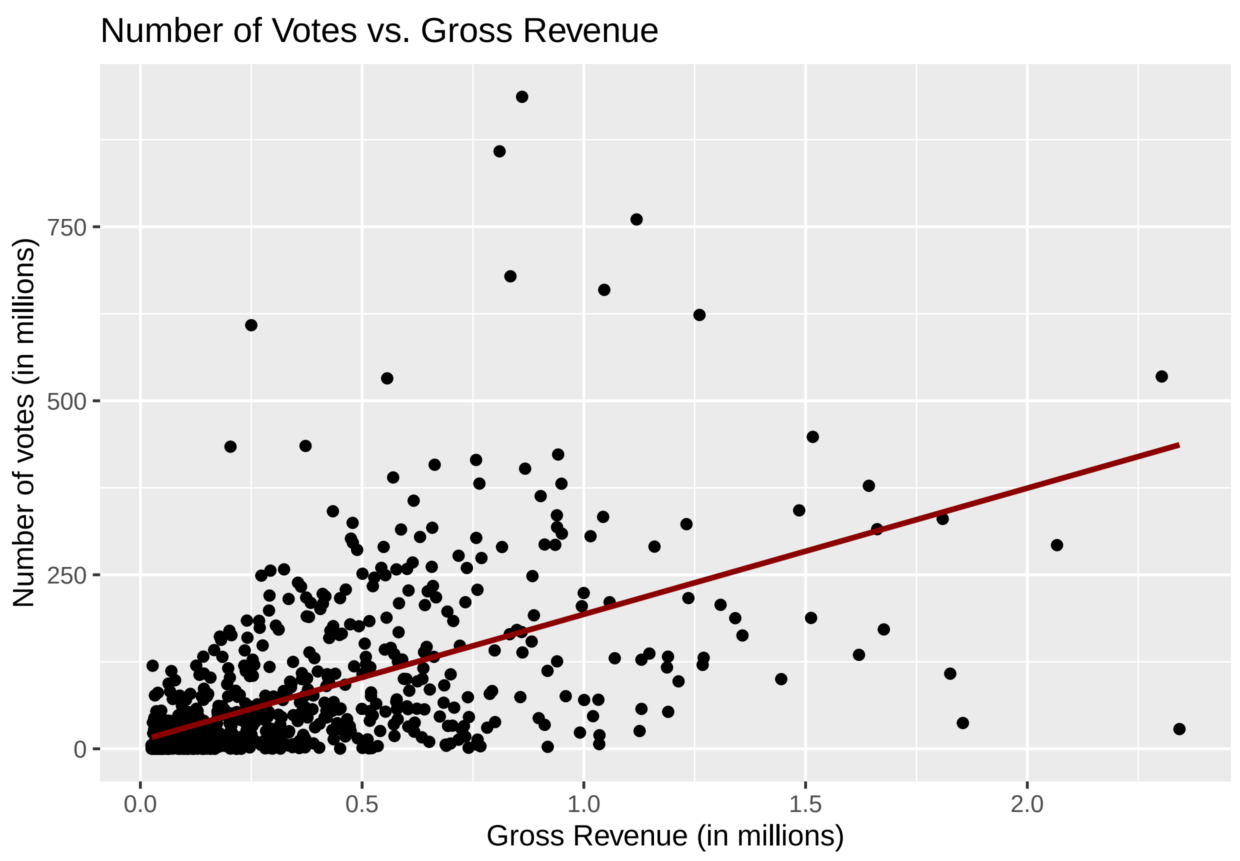
The genre that gets the highest gross renenve is family genre, and the genre that gets the least gross renueve is film noir.

No of Votes

Let's take a look at the relationship between `no_of_votes` and `gross`.

```
imdb %>%
  ggplot(aes(x=no_of_votes, y=gross)) +
  geom_point() +
  geom_smooth(method = "lm", se =F, col="darkred") +
  labs(title="Number of Votes vs. Gross Revenue",
       x = "Gross Revenue (in millions)",
       y = 'Number of votes (in millions)')
```

```
## 'geom_smooth()' using formula = 'y ~ x'
```



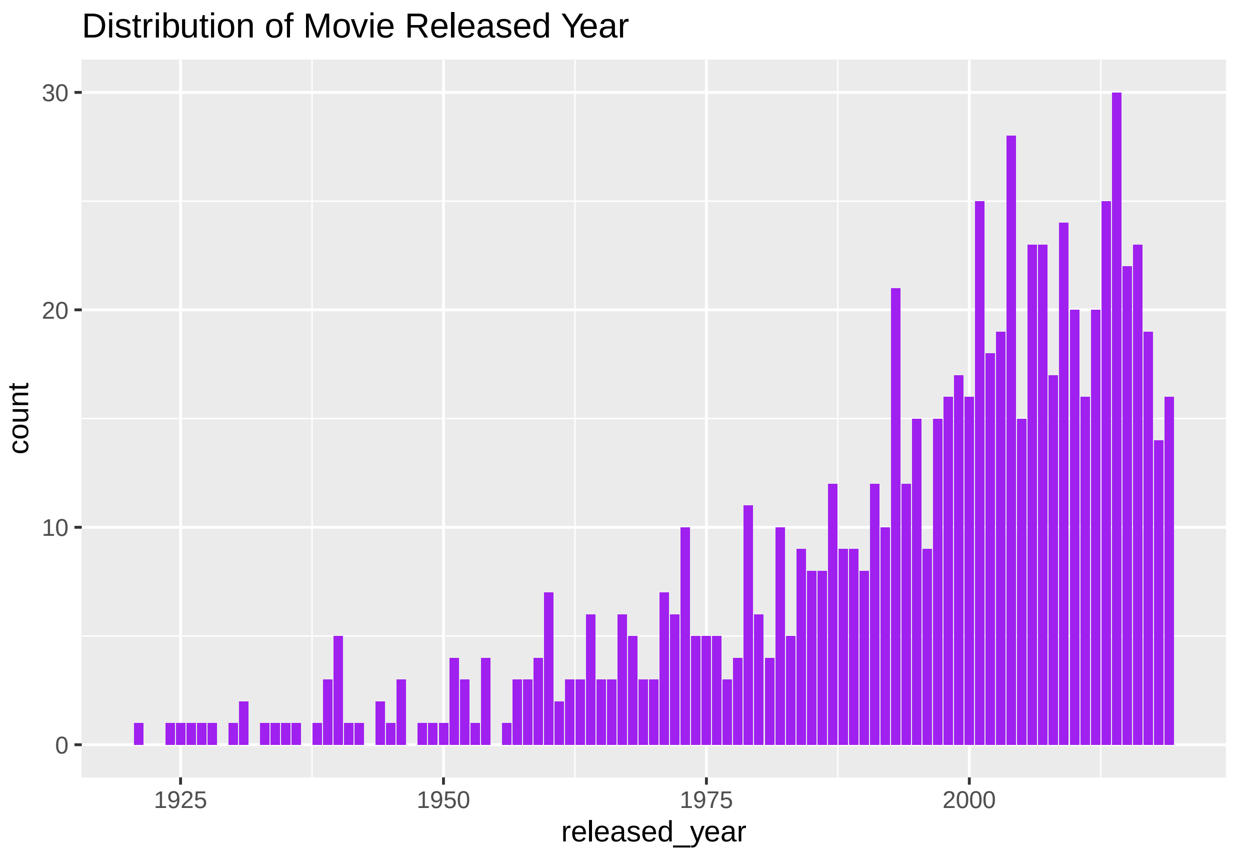
It's very clear that there's a positive linear relationship between the number of votes recieved and the gross revenue of movies.

Released Years

```r
imdb %>%
  ggplot(aes(x=released_year)) +
  geom_bar(bin= 20, fill="purple")  +
  labs(title="Distribution of Movie Released Year")
```

```
## Warning in geom_bar(bin = 20, fill = "purple"): Ignoring unknown parameters:
## ‘bin‘
```

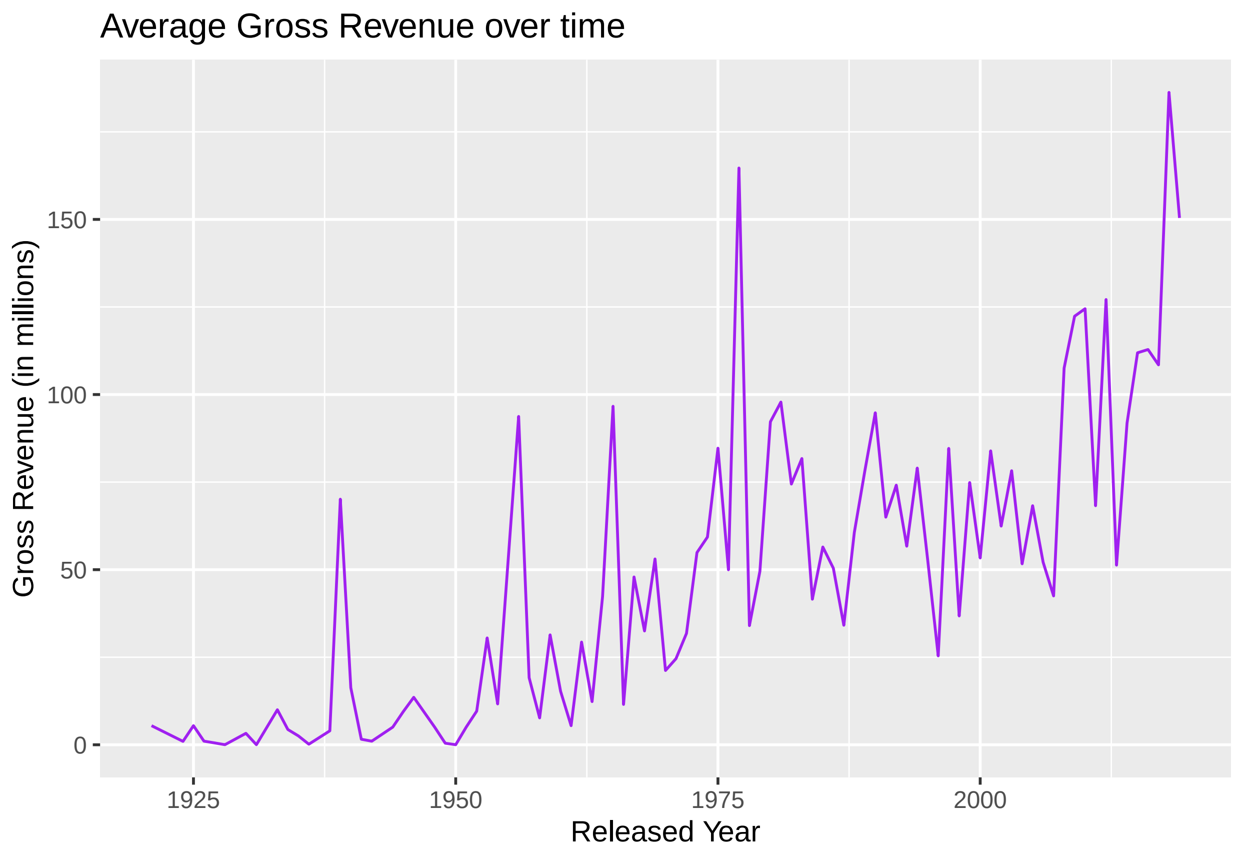
```
## Warning: Removed 1 rows containing non-finite values (‘stat_count()‘).
```



Looks like most movies are released after 1990.

Checking the relationship of `gross` and `released_year`.

```r
# calculate gross revenue average over time
year_mean <- aggregate(gross ~ released_year, imdb, mean)

year_mean %>%
  ggplot(aes(x=released_year, y=gross)) +
  geom_line(col='purple') +
  labs(title="Average Gross Revenue over time",
       x="Released Year",
       y="Gross Revenue (in millions)")
```



The graph shows that Gross revenue average increases overtime. Which makes sense since economy increases overtime.

Setting Up Models

(at least 4 models) ## Spliting data Because the amount of missing values are relatively small, we can impute the data in the recipe.

```r
set.seed(123)

# split the data by 70% stratify on gross
imdb_split <- initial_split(imdb, prop = 0.70,
                                   strata = gross)
imdb_train <- training(imdb_split)
imdb_test <- testing(imdb_split)

imdb_split
```

```
## <Training/Testing/Total>
## <548/237/785>
```

Write the recipe.

```r
# creating
imdb_recipe <- recipe(gross ~ ., data = imdb_train) %>%
  # impute the missing released year with median
  step_impute_median(released_year) %>%
  # impute missing data in meta score
  step_impute_knn(meta_score) %>%
  # dummy coding nominal variables
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  # scaling
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

imdb_recipe %>%
  prep() %>%
  bake(new_data = imdb_train)
```

```
## # A tibble: 548 x 21
##    released_year runtime imdb_rating meta_score no_of_votes  gross certificate_R
##            <dbl>   <dbl>       <dbl>      <dbl>       <dbl>  <dbl>         <dbl>
##  1       -1.86   -1.04          3.78     1.52        1.09   4.36          -0.446
##  2       -2.88   -1.38          1.99     1.52       -0.328  0.163         -0.446
##  3       -3.12   -1.38          1.99     1.77       -0.479  0.0192        -0.446
##  4        1.10    0.0738        1.64    -0.194      -0.795  1.66          -0.446
##  5        0.569   1.53          1.64     0.0348     -0.476  1.22          -0.446
##  6        0.375  -0.857         1.64    -0.0305      0.565  0.707         -0.446
##  7        0.0831  0.372         1.64    -0.112       0.0479 2.38          -0.446
##  8        1.10    0.558         1.28    -0.749      -0.767  1.37          -0.446
##  9        0.812  -0.336         1.28    -0.0305     -0.137  0.687          2.24
## 10        0.229  -0.820         1.28    -0.765       1.32   3.64          -0.446
## # i 538 more rows
## # i 14 more variables: certificate_U <dbl>, certificate_UA <dbl>,
## #   certificate_Other <dbl>, primary_genre_Adventure <dbl>,
## #   primary_genre_Animation <dbl>, primary_genre_Biography <dbl>,
## #   primary_genre_Comedy <dbl>, primary_genre_Crime <dbl>,
## #   primary_genre_Drama <dbl>, primary_genre_Family <dbl>,
## #   primary_genre_Film.Noir <dbl>, primary_genre_Horror <dbl>, ...
```

K-fold Cross Validation

```r
imdb_folds <- vfold_cv(imdb_train, v=10, strata = gross)
```

Model Building

Setting up models

```r
# Linear Regression
lm_model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# K-Nearest Neighbors Model
knn_model <- nearest_neighbor(neighbors = tune()) %>%
  set_engine("kknn") %>%
  set_mode("regression")

# Elastic Net
# Tuning penalty and mixture
en_spec <- linear_reg(penalty = tune(),
                           mixture = tune()) %>%
  set_engine("glmnet") %>%
  set_mode("regression")

# Random Forest
# Tuning mtry (number of predictors), trees, and min_n (number of minimum values in each node)
rf_spec <- rand_forest(mtry = tune(),
                       trees = tune(),
                       min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")
```

Setting up workflow

Set up the workflow for the models and add the model and the recipe.

```r
# Linear Regression
lm_wflow <- workflow() %>%
  add_model(lm_model) %>%
  add_recipe(imdb_recipe)

# K-Nearest Neighbors Model
knn_wkflow <- workflow() %>%
  add_model(knn_model) %>%
  add_recipe(imdb_recipe)

# Elastic net
en_workflow <- workflow() %>%
  add_recipe(imdb_recipe) %>%
  add_model(en_spec)

# Random Forest
rf_workflow <- workflow() %>%
  add_recipe(imdb_recipe) %>%
  add_model(rf_spec)
```

Tuning grid

```r
# K-nearest neighbors
knn_grid <- grid_regular(neighbors(range = c(1,10)),
                         levels = 10)

# Elastic Net
en_grid <- grid_regular(penalty(),
                        mixture(range = c(0,1)),
                        levels = 10)

# Random Forest
rf_grid <- grid_regular(mtry(range = c(1, 10)),
                        trees(range = c(100,600)),
                        min_n(range = c(5,20)),
                        levels = 10)
```

Fit all models to the folded data

```r
# Linear Regression
lm_fit <- lm_wflow %>%
  fit_resamples(imdb_folds)
```

```r
# KNN
knn_tune_res <- tune_grid(
  object = knn_wkflow,
  resamples = imdb_folds,
  grid = knn_grid
)

# Elastic Net
en_tune_res <- tune_grid(
  en_workflow,
  resamples = imdb_folds,
  grid = en_grid
)

# Random Forest
rf_tune_res <- tune_grid(
  rf_workflow,
  resamples = imdb_folds,
  grid = rf_grid
)
```

Save the models

```r
# knn model
save(knn_tune_res, file = "knn_tune_res.rda")
```

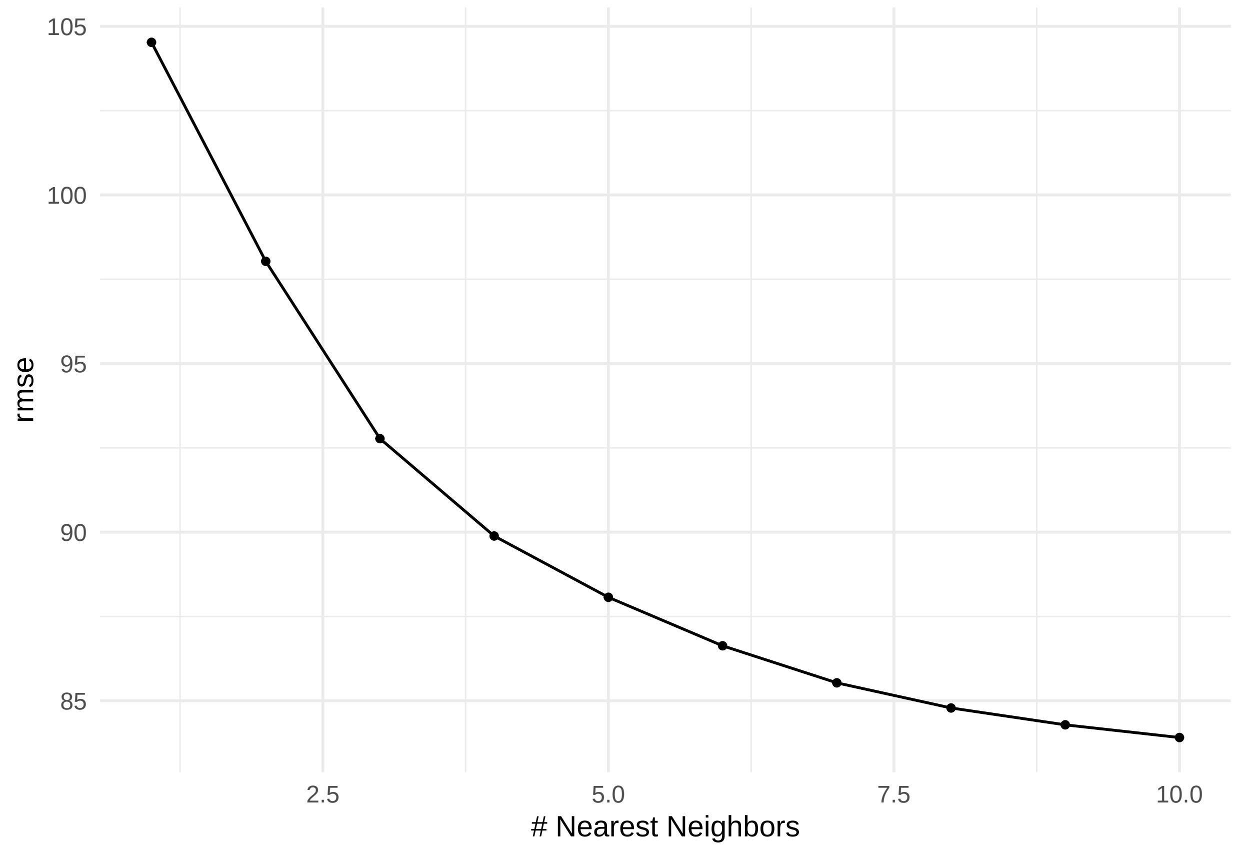
```r
# elastic net model
save(en_tune_res, file = "en_tune_res.rda")

# Random Forest model
save(rf_tune_res, file = "rf_tune_res.rda")
```

Load the models

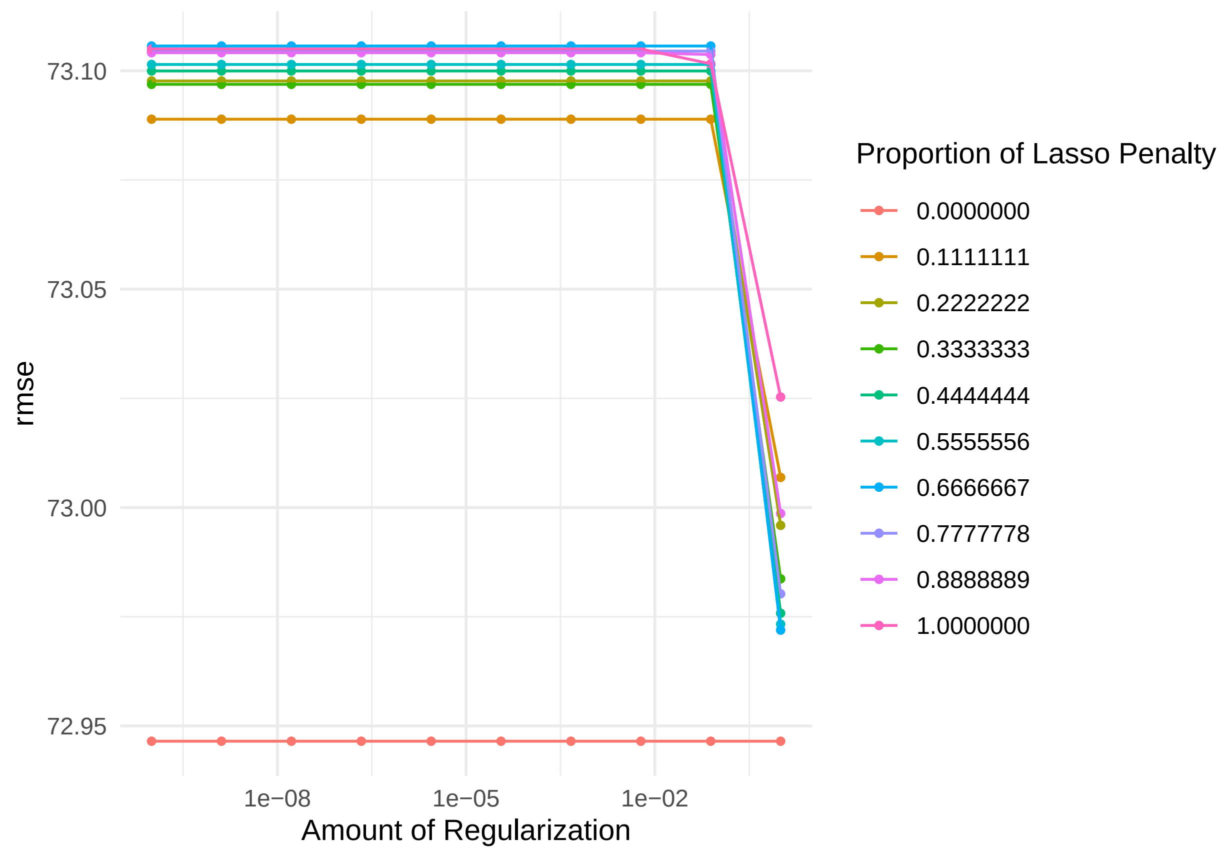
```r
# knn
load("knn_tune_res.rda")

# elastic net
load("en_tune_res.rda")

# random forest
load("rf_tune_res.rda")
```

cv curve, need to decrease significantly and go back up ## Model results

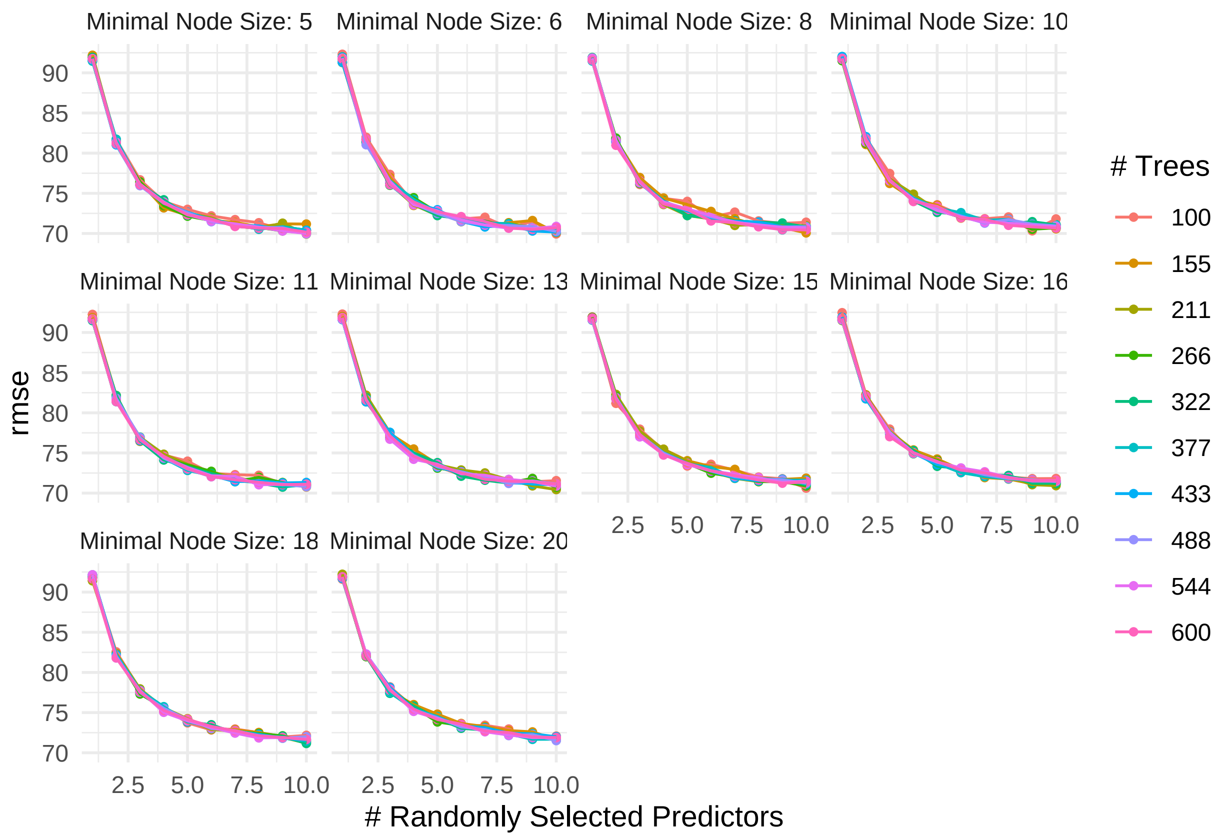
```r
autoplot(knn_tune_res, metric = 'rmse') + theme_minimal()
```



```r
autoplot(en_tune_res, metric = 'rmse') + theme_minimal()
```



```r
autoplot(rf_tune_res, metric = 'rmse') + theme_minimal()
```



```
# best linear regression model
best_lm_imdb <- select_by_one_std_err(lm_fit,
                                      metric = "rmse",
                                      n)

# best knn model
best_knn_imdb <- select_by_one_std_err(knn_tune_res,
                                      metric = "rmse",
                                      neighbors
                                      )
# best elastic model

best_en_imdb <- select_by_one_std_err(en_tune_res,
                                      metric = "rmse",
                                      penalty,
                                      mixture
                                      )

# best random forest model
best_rf_imdb <- select_by_one_std_err(rf_tune_res,
                                      metric = "rmse",
                                      mtry,
                                      trees,
                                      min_n
                                      )
```

RMSE Table to compare best models by RMSE

```
rmse_tibble <- tibble(Model = c("Linear Regression", "K Nearest Neighbors", "Elastic Net", "Random Fores

rmse_tibble = rmse_tibble %>% arrange(RMSE)
rmse_tibble
```

```
## # A tibble: 4 x 2
##   Model                RMSE
##   <chr>               <dbl>
## 1 Elastic Net          72.9
## 2 Linear Regression    74.7
## 3 Random Forest        76.7
## 4 K Nearest Neighbors  89.9
```

Results From the Best Model

```
best_en_imdb
```

```
## # A tibble: 1 x 10
##      penalty mixture .metric .estimator  mean     n std_err .config .best .bound
##        <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>   <dbl>  <dbl>
## 1      1e-10       0 rmse    standard    72.9    10    6.81 Prepro~  72.9   79.8
```

(fit the best or 2 to testing set) ## Fitting to training Data

```
final_en_model <- finalize_workflow(en_workflow, best_en_imdb)
final_en_model <- fit(final_en_model,
                        data = imdb_train)
```

Fitting to testing Data

```
# evaluate performance on testing
final_en_model_test <- augment(final_en_model, imdb_test) %>%
  rmse(truth = gross, estimate = .pred)

final_en_model_test
```

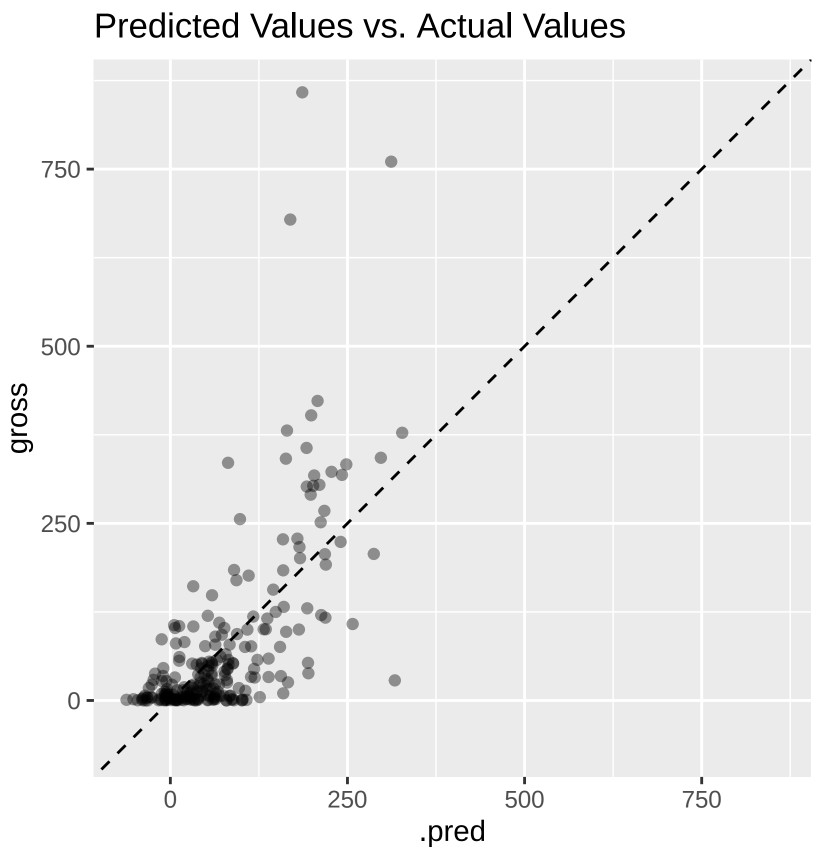
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        90.9
```

```
imdb_tibble <- predict(final_en_model, new_data = imdb_test %>% select(-gross))
imdb_tibble <- bind_cols(imdb_tibble, imdb_test %>% select(gross))

imdb_tibble
```

```
## # A tibble: 237 x 2
##    .pred gross
##    <dbl> <dbl>
##  1 317.   28.3
##  2 123.   57.3
##  3 327.  378.
##  4 257.  108.
##  5 163.   96.9
##  6  43.1   6.1
##  7 297.  343.
##  8 198.  290.
##  9 159.   10.1
## 10 182.  217.
## # i 227 more rows
```

Visualizing Model Performance

```
imdb_tibble %>%
  ggplot(aes(x = .pred, y = gross)) +
  geom_point(alpha = 0.4) +
  geom_abline(lty = 2) +
  theme_grey() +
  coord_obs_pred() +
  labs(title = "Predicted Values vs. Actual Values")
```



Conclusion

(2 paragraphs, summarize the main results of the paper, discuss which models did best, and briefly mention possibilities for future models/research)

Sources

https://www.kaggle.com/datasets/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows