

# MATH 154 - Final Project Writeup

due on Thursday, December 12, 2024

Jazelle Saligumba, Kellie Au, Kartika Santoso

## Table of contents

Introduction / Motivation (include a visualization maybe) . . . . .	1
Data . . . . .	2
Data Wrangling . . . . .	6
Visualizations . . . . .	7
Random Forest . . . . .	10
Ethical Considerations . . . . .	14
Conclusion . . . . .	14
References . . . . .	15
Our study can be improved: . . . . .	17

## Introduction / Motivation (include a visualization maybe)

The Academy Awards, colloquially known as the Oscars, are an iconic American cultural institution approaching its century-long anniversary. As a defining force in Hollywood, the arts, and the global audience it captivates, the Oscars have evolved into the foremost arbiter in determining which films are immortalized in history. Over time, the ceremony has come to shape not only the landscape of cinema but also public perception of artistic achievement.

For as long as the Oscars has existed, the concept of ‘Oscar Bait’ continuously makes its yearly occurrence - the idea that certain movies are intentionally created with the Academy’s specific tastes to win an Oscar as part of its marketing strategy. There are certain factors such as having a late release date which approaches the end of the Oscar submission deadline, being a drama, historical piece, or a biopic, and a star-studded cast and director which aligns with most Oscar winners. (“Hooked on Oscar Bait – Is There a Hack to Winning an Oscar?” 2021; Hoffer 2010; Franklin 2019)

Considering the Oscars has become the primary authority of American film taste, it is important to critically consider the media which we consume, and the people behind the Academy who decide what becomes gilded for the rest of time. As we explore what makes an Oscar

Award winner versus a nominee, we question the authenticity of such wins, especially when there is a prominent trend of who wins an Oscar award, and the recorded phenomena of directors gaming the system to increase their odds to win one. In this chicken-and-egg situation of which came first, it is important to recognize that the winners may not purely be based on the film's quality, but its features which cater to the Academy.

## Data

We began by using this [dataset of Oscar nominees and winners from 1927-2024](#). This dataset had info on each film, its release year, the category it was nominated for, and whether or not it won.

```
library(tidyverse)
oscar_nominees <- read.csv("the_oscar_award.csv")
```

For each film in this dataset, we used the [TMDB API](#) to info on its budget, popularity, revenue, runtime, release date, vote average, vote count, genres, production countries, and spoken languages.

To set up authorization with the TMDB API, make an account and set the API key as an environment variable by creating a `.Renviron` file and listing the API key as below:

```
TMDB_API_KEY=YOURAPIKEY # No spaces or quotation marks!
```

Then, run `readRenviron(".Renviron")` in the console.

```
library(httr)
library(jsonlite)

url <- "https://api.themoviedb.org/3/authentication"

personal_authorization <- paste0('Bearer ', Sys.getenv("TMDB_API_KEY"))

response <- VERB("GET", url, add_headers('Authorization' = personal_authorization),
               content_type("application/octet-stream"), accept("application/json"))
```

In order to get the movie data, we first had to make an API call to search for the movie in the TMDB database and get its TMDB id. We search using the film name and year from the Oscar nominees dataset, then take the result with the highest popularity. We assume here movies that get nominated are much more popular than non-nominated movies.

The following function does exactly as described above, make an API call to retrieve an identified movie and then returns the movie with the highest popularity.

```

# function to search for a movie by title and year
search_movie <- function(auth, movie_title, year) {
  url <- "https://api.themoviedb.org/3/search/movie"

  # parameters for the query
  queryString <- list(
    query = movie_title,
    year = as.character(year)
  )

  response <- VERB("GET", url, query = queryString,
                  add_headers('Authorization' = auth),
                  content_type("application/octet-stream"),
                  accept("application/json"))

  # check for response status
  if (http_error(response)) {
    stop("Failed to retrieve movie data. HTTP Status: ", status_code(response))
  }

  # parse json
  response_content <- content(response, "text")
  json_data <- fromJSON(response_content)

  # return the top movie hit
  if (length(json_data$results) > 0) {
    return(
      as.data.frame(json_data$results |>
        arrange(desc(popularity)) |> # choose based on popularity
        head(1)
      )
    )
  } else { # modify to search +/- 1
    return(NULL) # No results found
  }
}

```

We now try this on an example movie: Parasite to exhibit the information that is returned:

```

# Example: Search for Parasite
parasite <- search_movie(personal_authorization, "Parasite", 2020)
print(parasite)

```

```

adult          backdrop_path genre_ids      id original_language
1 FALSE /hiKmpZMGZsrkA3cdce8a7Dpos1j.jpg 35, 53, 18 496243      ko
original_title
1

1 All unemployed, Ki-taek's family takes peculiar interest in the wealthy and glamorous Parks
  popularity          poster_path release_date    title video
1    96.484 /7IiTTgloJzvGI1TAYymCfbfl3vT.jpg 2019-05-30 Parasite FALSE
  vote_average vote_count
1         8.505      18352

```

We then applied this function to all observations in the Oscar nominations dataset to get our initial dataframe.

Now that we have the all of the movies' TMDB id, we can make another API call to get more details about the movie including its spoken language, budget, revenue, etc. The following function uses a given movie's TMDB id to retrieve more detailed information on each movie.

```

search_movie_extra <- function(auth, movie_id, fields) {
  parse_field <- function(json_data, field) {
    return(ifelse(!is.null(json_data[[field]]), json_data[[field]], NA))
  }

  url <- paste0("https://api.themoviedb.org/3/movie/", movie_id)

  # Make the GET request with the Authorization header and JSON accept header
  response <- GET(url,
    add_headers('Authorization' = auth),
    content_type("application/octet-stream"),
    accept("application/json"))

  # Check for response status
  if (http_error(response)) {
    stop("Failed to retrieve movie data. HTTP Status: ", status_code(response))
  }

  # Parse the JSON response
  response_content <- content(response, "text")
  json_data <- fromJSON(response_content)

  result <- data.frame(matrix(ncol = length(fields), nrow = 1))
  colnames(result) <- fields

```

```

# Safely extract all the available fields
if (length(json_data) > 0) {
  for (field in fields) {
    result[[field]] = parse_field(json_data, field)
  }

  return(result)
} else {
  return(data.frame())
}
}

```

More specifically, we wanted to retrieve the following columns/data to utilize later.

```

# Define fields to search for
input_fields <- c("adult", "backdrop_path", "belongs_to_collection", "budget", "genres", "homepage", "id", "imdb_id", "original_language", "original_title", "popularity", "poster_path", "production_companies", "production_countries", "release_date", "revenue", "runtime", "spoken_languages", "status", "tagline", "title", "video", "vote_average", "vote_count")

```

Hence, we have all of the qualities of the function necessary to retrieve this data! For example we can run this function to get extra details on Fightclub:

```

# Example: Get details for Fightclub
fightclub <- search_movie_extra(personal_authorization, 550, input_fields)
print(fightclub)

```

```

  adult      backdrop_path belongs_to_collection    budget genres
1 FALSE /hZkgoQYus5vegHoetLkCJzb17zJ.jpg          NA 63000000    18
      homepage id    imdb_id original_language
1 http://www.foxmovies.com/movies/fight-club 550 tt0137523      en
  original_title
1    Fight Club

1 A ticking-time-bomb insomniac and a slippery soap salesman channel primal male aggression :
  popularity      poster_path    production_companies
1    88.976 /pB8BM7pdSp6B6Ih7QZ4DrQ3PmJK.jpg 711, 508, 4700, 25, 20555
  production_countries release_date    revenue runtime spoken_languages    status
1          DE, US    1999-10-15 100853753    139      English Released
      tagline      title video vote_average vote_count
1 Mischief. Mayhem. Soap. Fight Club FALSE      8.437    29486

```

Now that we understand it works, we can run this function for every unique movie that appeared in our initial dataset

So now we have all of the details needed to analyze the Oscar and movie data!

## Data Wrangling

Now, we have all of the data necessary for us to pursue our project goal. That said, the data doesn't exactly come in a simple form for us to easily use in data visualization and in building a model. So, we have to transform it a bit.

First, we want to combine our datasets into one, easily manageable dataframe. So, we join by the film name and since we found some slight discrepancies in the year of release of some films, we accounted for a buffer of 1 in year. Joining this data creates a data frame where each row represents a movie nominated for a specific category of a given year. So, there are repeats of movies since many movies are nominated for several different categories. Note that in the following code we also created a new column called `won_at_least_one` which helps us to track whether a movie won an oscar, even if it wasn't necessarily in the category that the observation belongs to.

```
all_data <- left_join(oscar_nominees, movie_data, by = c("film" = "title")) |>
  filter(abs(year_film - release_year) <= 1) |>
  group_by(film, year_film) |>
  mutate(won_at_least_one = ifelse(sum(winner == "True") >= 1, TRUE, FALSE)) |>
  ungroup()
```

Our next hurdle was the fact that many columns in the dataset (including `genre_ids`, `production_countries`, and `spoken_languages`) were columns of lists. For example, in order to get the genre of a given observation, you could not just call `obs$genre_ids` because that could return to you a list of `genre_ids` associated with that given observation. Additionally (specifically for the genre variable), the vector initially came encoded as an integer vector (where the integers mapped to a specific genre). So, in order to handle this, we created a function to widen the dataset by genre, where a new binary column was made for each possible genre indicating whether or not the movie classified as that genre.

The following, `wider_by_genre` does exactly as explained and mutates the column names to be decoded into their string representations.

```
# function for unnesting genre list columns
wider_by_genre <- function(df) {
  # make sure all values are vectors
  df$genre_ids <- map(df$genre_ids, ~ if (is.list(.)) unlist(.) else .)

  # now, change all the number values to be genres(char)
  new_df <- df |>
    unnest(genre_ids) |>
    distinct() |>
    mutate(genre_ids = genres_dict[as.character(genre_ids)]) |>
```

```

filter(!is.na(genre_ids)) |>
mutate(valid_id = 1) |> # create binary value for the pivot_wider to use
pivot_wider(names_from = genre_ids, values_from = valid_id,
             values_fill = 0) # use names_prefix if want "genre_"

return(new_df)
}

```

The `production_countries` and `spoken_languages` columns had a similar format of list, but we did not need to decode its value names. The following function `flatten_lists` does exactly as described, widening the data so we did not have to worry about handling lists any longer. Additionally included in the following code is the idea of `top_vector`. The motivation for and explanation of this will be described in the Random Forest section.

```

# more generic flattening of lists
flatten_lists <- function(df, feature, top_vector, prefix) {
  # make sure all values in the specified column are vectors
  df[[feature]] <- map(df[[feature]], ~ if (is.list(.)) unlist(.) else .)
  df$feature[is.null(df$feature)] <- NA

  # now, change all the number values
  new_df <- df |>
    unnest({{feature}}) |> # unnest the specified column
    mutate(valid_value = 1, # create binary value for pivot_wider
           temp_feature = case_when(
             .data[[feature]] %in% top_vector ~ as.character(.data[[feature]]),
             TRUE ~ "Other")) |>
    select(-{{feature}}, -feature) |>
    distinct() |>
    pivot_wider(names_from = temp_feature, values_from = valid_value,
               names_prefix = prefix, values_fill = 0)

  return(new_df)
}

```

## Visualizations

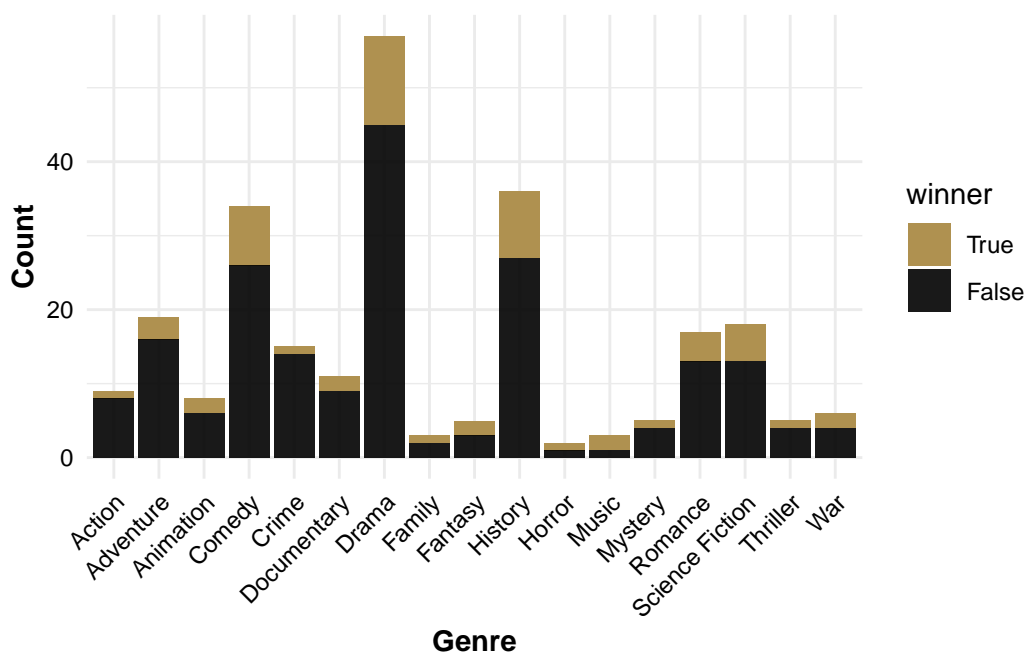
- q4, q5

Now that we have wrangled the data to be in a form that we can easily work with. We look to do some data analyzation. Motivated by this idea of oscar baiting, we initially wanted

to look at the characteristics that were specifically listed as elements of oscar baiting. More specifically, the traits that we had data on included Genre, Budget, and Release Date.

Since the first notion of “oscar baiting” was in 1978 (“The Deet Hunter”), we decided to use data from the last 50 years of the Oscars to analyze the trends of a possible oscar baiting phenomena.

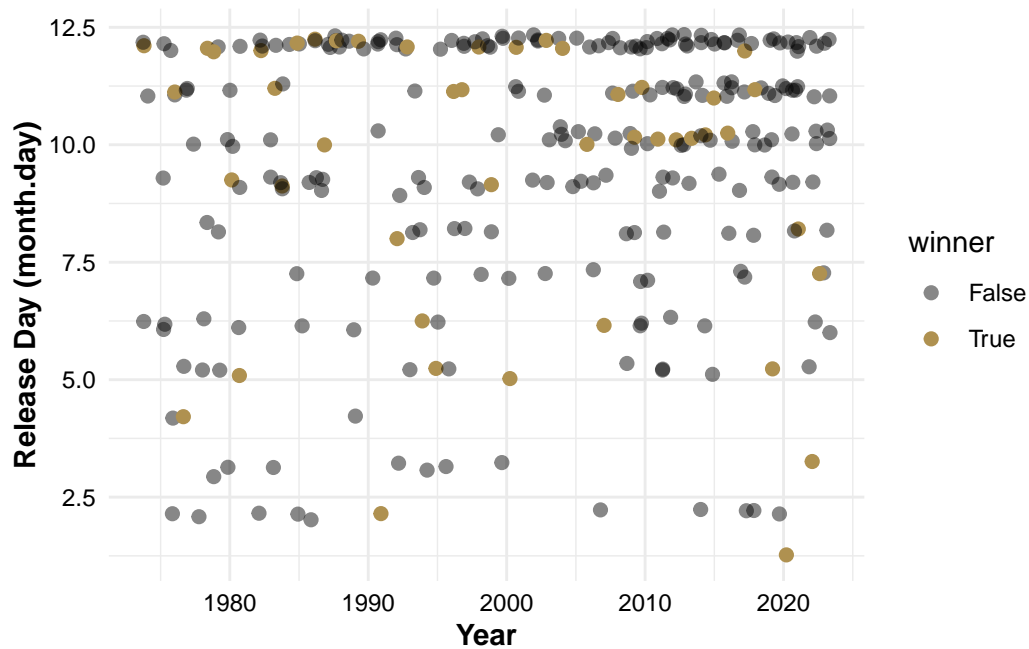
Specifically, we firstlooked at the year 2024 and the movies that were nominated for the 2024 Oscars (ie. they were 2023 films). It is notable how dominant the Drama category is among all of these movies, both in overall nominations and in winners. Additionally, following behind are historical films. Note that these results are consistent with a category of oscar baiting being a dramatic, historical period piece.



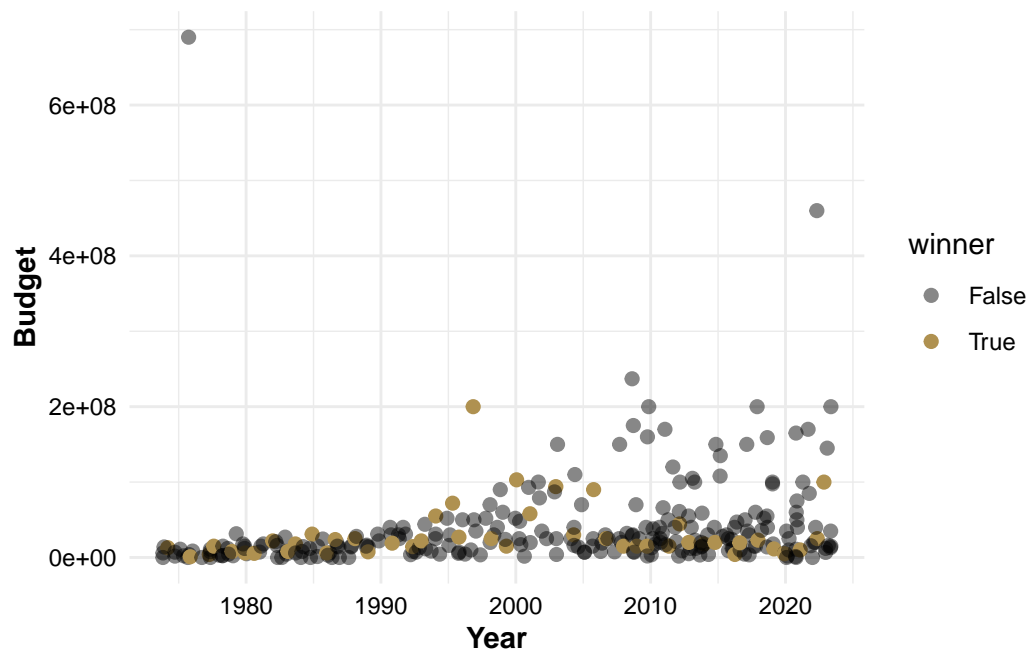
Now that we can acknowledge there is definitely some type of commonality between oscar nominees/winners, we can also look at these variables/traits over time to see how these trends have progressed throughout the years of the Oscars. So as to not make the graph too convoluted, we filtered the following graphs to only consist of nominees/winners of the “Best Picture” category over the last 50 years.

The following graph exhibits nominations over the years in terms of release day. As we can tell, while in 1975 release day of films were generally very scattered, as years passed, films nominated for “Best Picture” definitely showed a tendency of being released later in the year, around the time of the cutoff of movies for Oscar nominations.





Similarly, the following plot shows nominations over the years in terms of budget. Note that the budget data is not normalized. Though this trend is not as significantly obvious as the previous two, we still continue to see a trend of nominated films having a higher and higher budget (meaning higher production) for a movie.



**\*\***It is important to note that these visualizations are not entirely accurate as they may have some corrupted values due to IMDB being a user informed site. For example, there is a movie called “Taxi Driver” that is listed as having a budget of about 600 million dollars (observation seen in the top left corner of the graph above) whereas its real budget is somewhere closer to about 2 million dollars.

From all of these visualizations, we were able to understand and convince ourselves that oscar baiting is honestly a very real phenomena. That is, many Oscar nominated (and awarded) films have oddly similar traits in regards to their genre, budget, and release day of the year.

## Random Forest

- q4, q5

As we saw in the above visualizations, there are definitely trends in what kind of movies get nominated for Oscars. The next question we wanted to answer was whether we could train a random forest model to be able to predict which of those nominees actually wins the Oscar.

To use our data, we first had to do some preprocessing. Much of this preprocessing was done prior (details in the data wrangling section) but more specifically, making this data more feedable to a random forest was essential. In order to do this, we decided to add the `top_vector` variable in our `flatten_lists` function. The purpose of this variable was to help make a column for only the top 5 or 6 features of `spoken_languages` and `production countries`. That is, we created a new binary column for only the top 6 (most-occurring) production companies and the top 5 (most-occurring) spoken languages and for all movies that did not have a highly occurring production country or spoken language, we classified their country and language as “other”.

The motivation behind this was to simplify the dataset that we hoped to feed our random forest model enough so that it would not be thrown off by any outliers.

The following vectors contain the countries and languages that we found to be most-occurring.

```
top_countries <- c("US", "GB", "FR", "DE", "IT", "CA")
top_languages <- c("English", "French", "German", "Spanish", "Italian")
```

Now, we apply all of this data preprocessing to render a dataframe that the random forest will easily be able to understand and work with.

```
# preprocess data for nominees since 1975
rf_data <- all_data |>
  # first make df wider (flatten all data)
  mutate(genre_ids = genres) |>
  select(-genres) |>
  wider_by_genre() |>
  flatten_lists("production_countries", top_countries, "country_") |>
  flatten_lists("spoken_languages", top_languages, "lang_") |>
  select(-belongs_to_collection, -production_companies) |>
  filter(year_ceremony >= 1975) # filter for last 50 years
```

In addition to flattening the lists, we made sure to get rid of any and all information that we did not want to use as predictors. For instance, any identifiers and repetitive data is what we got rid of.

```
rf_data <- rf_data |>
  mutate(won_at_least_one = as.factor(won_at_least_one))

rf_data <- rf_data |>
  ungroup() |>
  distinct(film, .keep_all = TRUE) |>
  select(-year_film, -year_ceremony, -ceremony, -category, -name, -film, -winner,
        -adult, -backdrop_path, -homepage, -id, -imdb_id, -original_title, -overview, -poster)
```

Now that our data has been preprocessed, we are ready to train a model. Keep in mind that our model will be predicting the variable `won_at_least_one`. Again, since we adjusted the dataset earlier to account for any model that won at least one Oscar award, the model will be predicting whether or not it will win *any* award. We also perform cross-validation on the `mtry`, `trees`, and `min_n` parameters.

```
library(tidymodels)

# Partition data into train / test sets
set.seed(47)
movie_split <- initial_split(rf_data)
movie_train <- training(movie_split)
movie_test <- testing(movie_split)

# Recipe
recipe <-
  recipe(won_at_least_one ~ . ,
```

```

      data = movie_train)

# Model
movie_rf <- rand_forest(mtry = tune(),
                      trees = tune(),
                      min_n = tune()) |>
  set_engine("ranger", importance = "permutation") |>
  set_mode("classification")

# Workflow
movie_rf_wflow <- workflow() |>
  add_model(movie_rf) |>
  add_recipe(recipe)

# CV
movie_folds <- vfold_cv(movie_train, v = 4)

# Tuning parameters
movie_grid <- grid_regular(mtry(range = c(1,40)),
                          trees(range = c(10,500)),
                          min_n(range = c(1,20)),
                          levels = 5)

# Tune (CV)
movie_rf_tune <-
  movie_rf_wflow |>
  tune_grid(resamples = movie_folds,
            grid = movie_grid)

select_best(movie_rf_tune, metric = "accuracy")

# A tibble: 1 x 4
   mtry trees min_n .config
  <int> <int> <int> <chr>
1     10   132    15 Preprocessor1_Model082

```

After performing CV, we can select the best model and finalize it.

```

movie_rf_best <- finalize_model(
  movie_rf,
  select_best(movie_rf_tune, metric = "accuracy"))

```

```

movie_rf_final <-
  workflow() |>
  add_model(movie_rf_best) |>
  add_recipe(recipe) |>
  fit(data = movie_train)

movie_rf_final

```

```

== Workflow [trained] =====
Preprocessor: Recipe
Model: rand_forest()

-- Preprocessor -----
0 Recipe Steps

-- Model -----
Ranger result

Call:
  ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~10L,      x), num.trees = ~.

Type:                                Probability estimation
Number of trees:                      132
Sample size:                          1455
Number of independent variables:      40
Mtry:                                  10
Target node size:                     15
Variable importance mode:              permutation
Splitrule:                            gini
OOB prediction error (Brier s.):      0.1748917

```

Now that we have finalized our model, we can predict on our test data to see how well our model does on unseen data.

```

# Test on testing data
preds <- movie_rf_final |>
  predict(new_data = movie_test) |>
  cbind(movie_test)

preds |>

```

```
metrics(truth = won_at_least_one, estimate = .pred_class) |>
filter(.metric == "accuracy")
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>    <chr>         <dbl>
1 accuracy binary         0.716
```

```
conf_matrix <- conf_mat(preds, truth = won_at_least_one, estimate = .pred_class)
conf_matrix
```

	Truth	
Prediction	FALSE	TRUE
FALSE	306	117
TRUE	21	42

We observe that our model has a 72% accuracy rate on our test data. From our confusion matrix, we see that the true positive rate is 74% and the true negative rate is 94%. Note that there are much fewer movies that win Oscars than are nominated, so as seen in the confusion matrix, there are much more “False” observations in our data than “True” observations, causing the skew towards “False”.

Additionally, we note that our model is limited by the fact that it was trained on data/observations of which were *all nominated* for an Oscar. That is to say that our model is quite uninformed on all of the qualities of movies that are not nominated, and hence cannot distinguish exactly what makes a winner a *true* winner. Further, in our Shiny app, when you “predict” if a movie is going to win an Oscar, it is likely going to be False a significant more amount since it is likely being fed movies that have not yet been recognized as an Oscar nominee.

## Ethical Considerations

- q6

## Conclusion

- q7

## References

- cite packages, kaggle dataset (need to cite source of the kaggle data)

### 1. Why should anyone care about this?

Evaluating the media industry and the quality/replicability/authenticity of the media we consume

recent years in hollywood, (ex. writers' strike led to more reality tv, covid), is there more oscar baiting than usual, - putting money into movies (expensive sound/visual effects, cgi, alist actors, etc), want to get it back... how do you get that money back? - is there a trend in what kind of movies people are making nowadays to get the money back (ex. more period pieces) - when are the "best" movies made? and what are the "best" movies - what movies are better - how much money do oscar nominees/wins make back? (in comparison to the budget) - how much is a win worth compared to a nomination? - classification model of who wins and who doesn't - predicting 2025 oscar winners / nominees

overarching questions: who wins the oscars? (oscar formula and international movies), what is an oscar worth? (motivation), how true is our current idea of the oscar formula? (period pieces, big budget, released late in the year)

- quantifying the Oscar formula
- seeing trends in genres of winners/nominees since 1940s
- sentiment analysis on descriptions to see if more positive/negative themes are more likely to win
- genres
- budget / revenue
- release date (is it late in the year)
- runtime
- vote average / vote count
- popularity
- blockbuster vs oscar
- run a random forest on who will win oscars (by category?)
  - \* sentiment analysis on overviews?
  - preprocessing: split release date into year/month
  - how many years of data do we wanna use and which categories?
  - shiny app to make an interface where ppl can "predict" the winners(?)
  - slider graphs

- insights on budget/revenue/popularity/voteAverage → motivates why people wanna win an oscar
- 2. What is the project about? Do not assume that your readers have any domain knowledge! The burden of explanation as to what you are talking about is on you! For example, if your project involves phylogenetic trees, do not assume that your audience has anything other than a basic, lay understanding of genetics.
- 
- 2. Where did your data come from? What kind of data was it? Is there a link to the data or some other way for the reader to follow up on your work? Be sure to cite / reference where your data came from!!
- 
- 3. What are your findings? What kind of statistical computations (if any) have you done to support those conclusions? Again, while the R code will show how the calculation was performed, it is up to you to interpret, in English sentences, the results of these calculations. Do not forget about units, axis labels, etc.
- 4. What are the limitations of your work? Be clear so that others do not misinterpret your findings.
- 5. To what population do your results apply? Do they generalize? Could your work be extended with more data or computational power or time to analyze? How could your study be improved? Suggesting plausible extensions don't weaken your work – they strengthen it by connecting it to future work.
- Population = Movies that are nominated for an Oscar Award, no as of now they do not generalize
- With more data, we'd be able to understand more aspects of movies and come up with better model
  - ability to investigate more aspects of the movie (e.g. data on the actors/directors/producers involved in the film)
  - gathering data on movies that were NOT nominated would also be really helpful
-



### **Our study can be improved:**

6. What ethical considerations that came about during the project. How were you or weren't you able to address those issues?
  - questioning the authenticity of the media we consume and the media we consider to be "so good"
  - what does the academy think is a good movie?
    - the academy gets to dictate what the masses watch!
  - what movie gets to be guilded?
7. What did you learn beyond what we've covered in class?
  - how to work with live data from API, API calls
  - how to create a fully publishable shiny app that works dynamically with data and API
  -

Franklin, Marcy. 2019. "The 3 Tropes of an Oscar-Bait Trailer." *Vox*. <https://www.vox.com/ad/20974225/oscar-nominees-movie-trailers-tropes-formula>.

Hoffer, Robert. 2010. "The First Oscar Consultant." *Variety*. <https://variety.com/2010/film/news/the-first-oscar-consultant-1118015532/>.

"Hooked on Oscar Bait – Is There a Hack to Winning an Oscar?" 2021. *Sinema.SG*. <https://www.sinema.sg/2021/04/21/hooked-on-oscar-bait/>.