Contents

[Section 1: Background 2](#_Toc523666323)

[Part 1a: Description/Why are we interested 2](#_Toc523666324)

[Part 1b: Literature Review 2](#_Toc523666325)

[Section 2: Data Gathering 2](#_Toc523666326)

[Section 3: Data Wrangling 3](#_Toc523666327)

[Part 3A: Joining datasets together 3](#_Toc523666328)

[Part 3B: Parsing data 4](#_Toc523666329)

[Part 3C: Merging data 5](#_Toc523666330)

[Part 3D: Cleaning Data, part 1 7](#_Toc523666331)

[Part 3E: Cleaning Data, part 2 8](#_Toc523666332)

[Section 4: Data Preprocessing/Feature generation 9](#_Toc523666333)

[Part 4A: Generating usable features 9](#_Toc523666334)

[Part 4B: Final Dataset 13](#_Toc523666335)

[Section 5: Modeling 13](#_Toc523666336)

[Part 5A: Data exploration 13](#_Toc523666337)

[Part 5B: Feature selection 21](#_Toc523666338)

[Part 5C: Regression Models 21](#_Toc523666339)

[Part 5D: Classification Models 21](#_Toc523666340)

[Section 6: Results Summary 21](#_Toc523666341)

[Section 7: Data Product? 21](#_Toc523666342)

[Section 8: Team member contributions 21](#_Toc523666343)

[Section 9: Appendix? 21](#_Toc523666344)

Predicting Movie Box Office Revenues

# Section 1: Background

## Part 1a: Description/Why are we interested

## Part 1b: Literature Review

# Section 2: Data Gathering

The primary data sources we used were three websites that contained ready-made data sets and two websites that hosted APIs, which allowed users to pull movie information. In addition to these five primary data sources, we also needed to supplement them with other data found online. A description of each of these sources follows:

1. **Source #1:** The first primary website was an account on Kaggle[[1]](#footnote-1), which provided data from the MovieLens database. It included 45,000 movies, along each movie’s revenue, production budget, release date, language, genres, overview, runtime, tagline, production company as well as several accompanying files on movie keywords, credits, and ratings. Because this was the largest and most comprehensive data source we found, we made this our base file and sought out more information to corroborate the data already in these files and to also add any additional features we could find to make it a more complete data set.
2. **Source #2:** The second primary website was another account on Kaggle[[2]](#footnote-2), which provided data on 5,000 movies using The Movie Database. This dataset included the movie’s cast, crew, production budget, genre, keywords, production company, revenue, languages, runtime, and tagline.
3. **Source #3:** The third primary website was the-numbers.com[[3]](#footnote-3). This dataset included just the movie’s release date, revenue, and production budget.
4. **Source #4:** The first of the two movie APIs we used was The Open Movie Database (OMDB)[[4]](#footnote-4). APIs have the potential to have more up-to-date data or have different features or information compared with the static data we found in our website sources. In order to pull from the API, we needed a list of movies to pull. Since we decided to make the MovieLens data our base file (Source #1), we wanted to pull the same list of movies that were already in that large file and then merge any information we pulled from the API. In order to pull data from this API, we used the one of the movie ID columns in the MovieLens data (IMDB ID) and fed those ID’s for movies released in 1995 and onwards. The code associated with this is in the pull\_OMDB\_API.py[[5]](#footnote-5) file. We took the output and stored that as another dataset. Because of the amount of data we needed to pull, we required assistance from the OMDB API owner to give us both his direct server’s URL and a temporary increase in our daily pull limit. Because of this, anyone trying to run the code associated with this API pull will run into problems because we were asked to not publish the server’s URL in any of our code.
5. **Source #5:** The second of the two movie APIs we used was The Movie Database (TMDB)[[6]](#footnote-6). This is where the data from Source #2 came from; however, because of the possibility that pulling data from an API would give us more up-to-date data, we went ahead and also pulled data from this API in order to eventually merge all the data sources together. The code associated with this is in the pull\_TMDB\_API.py[[7]](#footnote-7) file. Pulling data through TMDB was more complicated because we had to do a two-step procedure. The reason for this is because TMDB provides a full set of movie features only when you pull movies using TMDB’s own set of movie IDs. In contrast, if any other recognized movie ID is used, only a portion of the movie’s features would be pulled.

As a result, we first pulled information using the same movie ID we used to pull from the OMDB API. From this partial output, TMDB provided its own movie ID as part of the features. We then used that new movie ID and performed a second pull of data. The new output of movie information included the full list of features TMDB provides to users. With the benefit of hindsight, however, it’s possible this could have been done in just one step if the appropriate movie ID stored in the MovieLens data was used instead. Unfortunately, this was the one of the very first things we did for our project, and we were not as familiar with the data at that point in time.

Lastly, because our base file of movies was the MovieLens database, it did not have good coverage of movies in 2017 or the first half of 2018. As a result, we made a list of all the movies that came out during those one and a half years using a simple Google search. We took the names and release dates of those movies and input them into our two movie API sources in order to pull whatever information existed for those movies. The files associated with this are pull\_OMDB\_API\_newMovies.py[[8]](#footnote-8) and pull\_TMDB\_API\_newMovies.py[[9]](#footnote-9).

# Section 3: Data Wrangling

## Part 3A: Joining datasets together

With all the files we either collected from websites or generated ourselves, there were roughly a dozen files that we needed to join together to begin our process of creating one complete dataset. All the data from the data gathering stage is stored in the data folder in our GitHub account[[10]](#footnote-10). The code associated with joining these files is in the joinDataModule.py[[11]](#footnote-11). Excluding the dataset we created from the-numbers.com, all the datasets from all the sources could be joined relatively easily after it was determined which movie ID each dataset was using.

Joining the dataset from the-numbers.com required a different method because that data did not come with an ID. It only had the movie name and release date as usable joining columns. So the first step was to make movie name in both this dataset and our base file (Source #1) lower case. Then the two data sets were joined using movie name and the year of the release. We believed this was the most reasonable way to join these two datasets. However, there were instances when this joining method resulted in duplicates; for example, if a movie was released in limited release in the winter (and was captured as such in one dataset), but was released widely at the beginning of the next year (and was captured as such in the other dataset). In addition, sometimes a movie name could be spelled differently or have extra characters that would make joining on name impossible. Future work could be done on the best way to join these datasets to avoid such errors.

## Part 3B: Parsing data

Once we joined all the datasets into a single joined dataset, we noticed that several columns were JSON strings of lists of dictionaries. In order to make them into more usable columns, we wrote several functions to first convert those strings into actual lists of dictionaries. Then we either parsed the values into separate columns or we extracted the values and replaced the lists of dictionaries with simple lists of values. The code associated with parsing these files is in the parseColumnsModule.py[[12]](#footnote-12) file. The following columns needed to be parsed:

1. Ratings: Some ratings columns originally looked like this:

*“[{'Source': 'Internet Movie Database', 'Value': '8.3/10'}, {'Source': 'Rotten Tomatoes', 'Value': '100%'}, {'Source': 'Metacritic', 'Value': '95/100'}]”*

After converting this string into an actual list of dictionaries, we parsed through this to extract the rating for IMDB, Rotten Tomatoes, and Metacritic separately and stored them in columns called “Rating\_IMDB”, “Rating\_RT”, and “Rating\_Metacritic”.

1. Genres: Some genre columns originally looked like this:

*“[{'id': 28, 'name': 'Action'}, {'id': 80, 'name': 'Crime'}, {'id': 18, 'name': 'Drama'}, {'id': 53, 'name': 'Thriller'}]”*

After converting this string into an actual list of dictionaries, we parsed through this to extract every genre listed and convert the columns in a simple list of genres: “[‘Action’, ‘Crime’, ‘Drama’, ‘Thriller’]”.

1. Production company: The same method used for genres was applied here to parse through production companies and convert the columns into a simple list of companies.
2. Movie collection: The same method used for genres was applied here to parse through movie collections and convert the columns into a simple list of collections. Generally, there would only be one collection per movie if this column was filled in with data.
3. Cast: The same method used for genres was applied here to parse through actors and convert the columns into a simple list of actors.
4. Keywords: The same method used for genres was applied here to parse through movie keywords and convert the columns into a simple list of keywords.
5. Languages: The same method used for genres was applied here to parse through languages spoken in the movie and convert the columns into a simple list of languages.
6. Crew: A snippet of the various crew columns originally looked something like this:

*“[{'credit\_id': '52fe4284c3a36847f8024f49', 'department': 'Directing', 'gender': 2, 'id': 7879, 'job': 'Director', 'name': 'John Lasseter', 'profile\_path': '/7EdqiNbr4FRjIhKHyPPdFfEEEFG.jpg'}, {'credit\_id': '52fe4284c3a36847f8024f4f', 'department': 'Writing', 'gender': 2, 'id': 12891, 'job': 'Screenplay', 'name': 'Joss Whedon', 'profile\_path': '/dTiVsuaTVTeGmvkhcyJvKp2A5kr.jpg'}, {'credit\_id': '52fe4284c3a36847f8024f55', 'department': 'Writing', 'gender': 2, 'id': 7, 'job': 'Screenplay', 'name': 'Andrew Stanton', 'profile\_path': '/pvQWsu0qc8JFQhMVJkTHuexUAa1.jpg'}, {'credit\_id': '52fe4284c3a36847f8024f5b', 'department': 'Writing', 'gender': 2, 'id': 12892, 'job': 'Screenplay', 'name': 'Joel Cohen', 'profile\_path': '/dAubAiZcvKFbboWlj7oXOkZnTSu.jpg'}, {'credit\_id': '52fe4284c3a36847f8024f61', 'department': 'Writing', 'gender': 0, 'id': 12893, 'job': 'Screenplay', 'name': 'Alec Sokolow', 'profile\_path': '/v79vlRYi94BZUQnkkyznbGUZLjT.jpg'}, {'credit\_id': '52fe4284c3a36847f8024f67', 'department': 'Production', 'gender': 1, 'id': 12894, 'job': 'Producer', 'name': 'Bonnie Arnold', 'profile\_path': None},….]”*

We created specific columns for Director, Writer, and Producer by extracting any names associated with the job “Director”, “Screenplay”, or “Producer”, respectively. One aspect of this data we did not use was the gender data that was embedded in these dictionaries. Doing analysis based on the gender of the cast or crew could be a further enhancement of this project because of the current interest to produce more women-led and women-developed movies.

## Part 3C: Merging data

Because several data sources were used to create the consolidated dataset, there were roughly 100 columns of features, many of which were similar because our data sources all provided similar features. [SOME FEATURES WERE UNIQUE]. However, we simply could not just delete any similar sounding column. The benefit of using multiple sources is to try to ensure that some combination of these sources would yield a more complete set of data for each movie. Meaning, one source may only have data on a particular movie’s revenue, budget, and cast, but another source may have that movie’s plot and crew. By having multiple sources, we hoped to have a more complete picture of each movie.

Properly merging the features was the longest part of the data wrangling phase of our project because we had to ensure we were doing it correctly and were using our best judgement. The code associated with merging these columns is in the mergeDataModule.py[[13]](#footnote-13) file.

For each similar set of features, we implemented a different algorithm to merge them into one feature:

1. Name: Pick the first title found among the set of movie name columns.
2. Revenue: Pick the maximum revenue number found among list of movie revenue columns. We made the assumption that a larger number indicated that either, that number came from a more up-to-date data source, or because the largest number most likely represented a global box office revenue number.
3. Release Date: Pick earliest release date among list of movie date columns. This was a choice that may have been decided differently if we had the benefit of hindsight. Initially the idea was the earliest release date may be more likely to indicate a U.S. release date because sometimes movie open on a rolling schedule across the world and our data sets did not provide specifics on what the release date indicated. However, from our analysis at a later stage of this project, it seems like there was inconsistent reporting in release dates and more analysis should be done on this matter. When a movie is first released in a limited number of theaters, that date could potentially have been chosen rather than the wide-release date. Further, some movies may have actually opened in other countries first, which would also result in a less-than-accurate guess on an appropriate release date.
4. Movie Length: Pick the average movie length.
5. Budget: Pick the average movie budget. This was because most movie production budgets are estimates and taking an average would have been better than simply picking the maximum or minimum number.
6. Language: The intent was to only keep movies whose primary language was English. So, for every single language column that appeared in our combined dataset, it had to have “English” as the first language listed. This was a very strict filter because there were many instances when some datasets listed “English” as the primary language while others did not and the movie was in fact not primarily in English.
7. Genre: Create a list and get the union of all genres listed in all movie genre columns.
8. Production company: Create a list and get the union of all companies listed in all movie company columns.
9. Actors/Cast: Create a list and append any actors listed in all actor columns. Because actors are usually listed in order of importance in the movie, we wanted to preserve the ordering and could not use sets or unions.
10. Keywords: Create a list and get the union of all movie keywords listed in all similar movie keyword columns.
11. Movie Collection: Create a list and get the union of all movie collections listed in all similar movie collection columns.
12. Movie Overview: Pick the longest movie overview out of any movie overview columns. Being a text column, we decided to simply pick the longest movie description, so as to have more words for any text analysis model.
13. Movie Tagline: Pick the longest movie tagline out of any movie tagline columns. Being a text column, we decided to simply pick the longest movie tagline, so as to have more words for any text analysis model.
14. Director: Create a list and get the union of all directors listed in all similar movie director columns.
15. Writer: Create a list and get the union of all screenwriters listed in all similar movie writer columns.
16. Producer: Create a list and get the union of all producers listed in all similar movie producer columns.
17. Movie Rating: For any multiple movie rating columns, take the average of each rating’s source. There were multiple IMDB and Metacritic columns, but only one Rotten Tomatoes column. So the IMDB ratings were averaged together, and the Metacritic ratings were averaged together.

[EXPLAIN DATA TYPES AND SHOW SAMPLE DATASET}

Most of the functions used in the parseColumnsModule.py and mergeDataModule.py are contained in the movieFunctions.py[[14]](#footnote-14) file.

## Part 3D: Cleaning Data, part 1

The joinDataModule.py, parseDataModule.py, and mergeDataModuly.py are all run automatically and in order in the cleanData.py[[15]](#footnote-15) file. [merged set stored in PANDAS DATAFRAME] After the three modules are finished running, we make an initial attempt at removing duplicates. First, any row completely identical in all columns are dropped and only the first instance is kept. Then we also decided to drop any rows where the movie name was the same, controlling for the string’s case, and if the movie date was within the same calendar year. Only the first instance is kept. After doing this, we were left with roughly 20,000 movies that were released from 1995 to June 15, 2018. We performed an initial count of how many missing data points there were for each feature. This initial count showed that most columns were missing 10,000 or more data points, including our movie revenue column, which our project ultimately depended on. This was a concern for us, so we decided to split up a few years of data among the team members in order to get a better sense of what the data looked like after the merging process and understand why so much data was still missing.

What we discovered was the vast majority of movies that had revenue missing were in fact simply TV movies, straight-to-video movies, or foreign-language movies. [Documentaries/Budgets/Writers]. For the 2017 and 2018 movies we pulled separately, there were issues beyond that initial finding. Because those movies were relatively recent, even our data pull from the APIs did not provide a lot of information about revenues or budgets. As a result, several legitimate movies from 2017 and 2018 that opened in theaters did not have critical information. Additionally, another hiccup we discovered was for some movies in our 1995-2018 sample period, our data sources were not consistent in reporting domestic box office returns vs. global box office returns. Lastly, some duplicate movies still remained due to different title spellings and due to different data sources providing different release years for the same movie, making it more difficult to write code to automatically drop these duplicates. Because of these inconsistencies, we decided to take the following steps:

1. For the movies in our dataset released in 2017 and the first half of 2018, we would have team members manually go through and input movie revenue numbers of any legitimate movie and attempt to fill in other missing features for those legitimate movies. Any movie that was not released in theaters or not primarily in English would be marked for deletion.
2. For any movie in our dataset prior to 2017, we would automatically drop any movie whose revenue data was missing. We did this because of the finding we made that most of the movies in earlier years without a revenue number were not released in theaters.
3. Time permitting, team members would then go through as many years as possible between 1995 and 2016 to ensure that the remaining movies’ revenue numbers were global revenues, and also attempt to fill in any missing features for those movies. Any obvious foreign-language movie or duplicate movie would be marked for deletion.

Ultimately our team managed to look over and check movies from 2008 – 2018. In many data science projects, missing data may only be dealt with by data imputation or dropping empty rows, because additional data may not exist. However, because our project dealt with movies, information to help fill in missing data can be found using multiple sources. Additionally, many of our columns were text or categorical data types, and there may not be easy ways to perform imputation on them. In order to have as accurate and up-to-date data as possible, we made the decision to take on this manual task of looking up missing data ourselves. This kind of work would have been made easier if a crowd-sourcing service like Amazon Mechanical Turk were used.

Whether it was the right or wrong decision to attempt this manual work ourselves, we admit this used up a significant amount of time during our data gathering and wrangling process. This manual work did serve as a way to get more team members involved in the project, because the lack of coding and technical skills in our team made it impossible to share the work equally. However, we learned firsthand at how difficult it is to make sure every team member followed the same directions on how to fill in missing data, some examples of which will be described in Section 3, Part 3E. In hindsight, it may have been better to end our sample period in 2016 and deal with the other data inconsistencies we described earlier in some other way.

## Part 3E: Cleaning Data, part 2

The files that contain each team members’ work on filling in missing data is stored in the “filled-in-data” folder in our GitHub[[16]](#footnote-16). These files were loaded back into the same cleanData.py file introduced in Section 3, Part 3D. These files were concatenated back into a Pandas dataframe, with any movie marked for deletion removed.

In order to make sure this rebuilt dataset was ready to be used in our data preprocessing code (Section 4), we had to make sure to fix any human errors made in the manually filled in data. Errors largely occurred in columns where the data type was a string containing a list of strings. An example of this data type is: “[‘Sandra Bullock’, ‘Cate Blanchett’, ‘Anne Hathaway’]”. Columns like Actors, Director, Writers, Producers, Genres, Companies, and Keywords all contained this data type. Unfortunately, human error when filling in this type of data was inevitable, as the order of punctuation became haphazard. This had to be fixed before any further data processing could occur. This can be seen towards the end of the cleanData.py file. Once this and other human error issues were fixed, we stored the rebuilt dataframe that contained all our merged data into a table named “cleanedMovies\_20180814” in an SQLite database named “movies.db”. The SQLite database was stored in the “database” folder in GitHub[[17]](#footnote-17).

# Section 4: Data Preprocessing/Feature generation

## Part 4A: Generating usable features

The “cleanedMovies\_20180814” table in the “movies.db” SQLite database is pulled for further processing in order to make it more usable for machine learning models. The code for this portion of the project is in the preprocessData.py[[18]](#footnote-18) file. For most features in our dataset, some level of processing was required:

1. Genre: The merged genre column contains a list of genres associated with the movie. Meaning, each movie in our dataset may have multiple genres, i.e., “[‘Drama’, ‘Romance’]”. We decided to make multiple binary genre-specific columns: “Genre\_Drama”, “Genre\_Comedy”, “Genre\_Action\_Adventure”, “Genre\_Thriller\_Horror”, “Genre\_Romance”, “Genre\_Crime\_Mystery”, “Genre\_Animation”, “Genre\_Scifi”, “Genre\_Documentary”, “Genre\_Other”. Any genre associated with the movie will have that corresponding binary genre column set to 1. Multiple binary genre columns can be set to 1.
2. Rated: The movie rated column indicates how a movie is rated according to the Motion Picture Association of America (MPAA) film rating system. The possible ratings are “G”, “PG”, “PG-13”, “R”, and “NC-17”. We decided to make four binary rated columns: “Rated\_G\_PG”, “Rated\_PG-13”, “Rated\_R”, and “Rated\_Other”. The “Rated\_Other” column was necessary because some movies may been unrated, have spelling errors, or have some other type of rating not captured by the official movie ratings system.
   1. We decided to create a “Rated\_category” column, which would contain “G”, “PG”, “PG-13”, “R”, and “Other” as categories in order to help with data exploration and plotting.
3. Actors: Some data sources included the entire cast of the movie in their actor column, so some of our movies had a merged actor column that had over 100 actors listed. We decided to limit the number of actors in our actor column to just the top five actors of the movie. Because we kept the actors’ order of importance intact during the merging phase, this should still keep the most prominent actors.
4. Awards: The awards feature in our dataset came from IMDB in the form of one or two sentences. For example, some of the awards data looked like this: “Won 6 Oscars. Another 117 wins & 126 nominations.” or “3 wins & 4 nominations.” After looking at multiple variations of this data, IMDB seems to be following a pattern that could be codified and converted into numerical columns. Anytime a movie won or was nominated for a major award, like an Oscar, Golden Globe, BAFTA, etc., IMDB seems to indicate it in the first of two sentences. If the movie has won or been nominated for minor awards, IMDB seems to indicate that in the second of two sentences. If a movie only has minor award wins or nominations, there would only be one sentence. As a result, we wrote code to separate the major and minor awards based on how many sentences there were, and separate between a win and nomination using punctuation. We then extracted the numbers associated with a win or with a nomination based on what word was used. From this, four new features were created: “Major\_Win”, “Major\_Nomination”, “Minor\_Win”, and “Minor\_Nomination”.
5. Collection: A new feature was created called “isCollection”, which would be a 0 if the “Movie\_Collection” column was blank, and a 1 otherwise. This binary variable would help identify if the movie is part of a collection, like a sequel.
6. Release Date: In order to see whether the time of year a movie was released had an impact on the movie’s revenues, we created five seasonal dummy variables: “Spring”, “Summer”, “Fall”, “Winter”, and “Holiday”. These seasons were created using BoxOfficeMojo’s definitions of seasons[[19]](#footnote-19). For example, they define “Holiday” as being roughly the first week in November through the New Year’s weekend, and “Winter” as roughly after the New Year through the first week in March. Initially, we wanted to create a single binary variable that was a 1 or 0 if the movie opened on any major U.S. federal holiday. This task was left to a team member but was not completed. However, because of the discrepancies in some movies’ release dates across data sets (explained in Section 3, Part 3C), it probably worked out better to just use a more general time of year when a movie was released as opposed to a specific weekend.
   1. We also created a “Season” column, which would contain “Winter”, “Spring”, “Summer”, “Fall”, and “Holiday” as categories in order to help with data exploration and plotting.
7. Profit: Several profit-related variables were created for potential use in our models.
   1. A column called “Profit” was created as the simple difference between the Revenue and Budget columns. If Budget was null, Profit would be null.
   2. A column called “Profit\_Bucket” was created to indicate how much more revenue was earned compared with the movie’s production budget. Specifically, if a movie did not even earn as much as its production budget, it would fall under “Profit\_<1x”, to indicate it did not earn a whole multiple of its budget. Under similar rules, the other categories would be “Profit\_[1-2x)”, “Profit\_[2-3x)”, “Profit\_[3-4x)”, “Profit\_[4-5x)”, and “Profit\_>=5x”.
   3. Lastly, in case it was necessary, each of the previously defined profit buckets were also created as individual binary columns.
8. Deflated Revenue, Budget, Profit: In order to take into account general price inflation, we decided to deflate revenue and budget by the monthly headline U.S. Consumer Price Index (CPI)[[20]](#footnote-20). We pulled data using a python package that could pull data from the Federal Reserve Economic Data (FRED) website published by the Federal Reserve Bank of St. Louis. We decided to adjust our revenue and budget numbers by making the base month June 2018. Setting the base month to a recent month can help to give a better handle on what the dollar amounts represent. So for any particular month a movie was released in, the revenue and budget associated with that movie would be converted to June 2018 dollars.

There a definitely arguments to be made about whether to use annual vs. monthly inflation data, or potentially use something different from the headline U.S. CPI. Another option, particularly for movie revenues is to get historical data on the price of a movie ticket. However, for ease, we decided to just use U.S. CPI for both revenue and budget. Lastly, after revenue and budget were deflated, they were stored in new “Revenue\_Real” and “Budget\_Real” columns. These two columns were differenced to get a deflated profit. This was stored in “Profit\_Real”.

1. Production Company: In order to see if the production company that made the movie has any impact on the movie’s revenue, we created several binary columns for each of the major movie companies. Note that movies can have multiple production companies, i.e., “[‘Warner Bros.’, ‘Village Roadshow Pictures’]. So each movie can have a 1 in multiple binary production company columns. The new binary columns are “Comp\_Disney”, “Comp\_DreamWorks”, “Comp\_Fox”, “Comp\_Lionsgate”, “Comp\_MGM”, “Comp\_Miramax”, “Comp\_Paramount”, “Comp\_Sony”, “Comp\_Universal”, “Comp\_WarnerBros”, and “Comp\_Other”.

Because not all data sources correctly spell out production companies, we had to do extra work creating a lookup table of most of the production companies in our data set. For example, sometimes we would see “20th Century Fox” vs. “Fox” or “Warner Brothers Pictures” vs. “Warner Bros”. Beyond inconsistent spelling issues, there are a variety of subsidiaries of these major companies that make movies; however, we believe those movies should still be marked as being made by the major company. For example, Marvel Studios is the production company for several Marvel Comics-based movies, but it is a Disney company.

As a result, we did research to try to make sure subsidiaries were associated with their major-studio counterpart. If we had not done this, it’s possible that there would be too few movies associated with each production company, rendering the production company features useless. The lookup table we created is stored in our data folder in GitHub as “production\_companies.csv”[[21]](#footnote-21). This table was used to fill in the new binary production company columns. More work could have been done here to make sure all subsidiaries were captured, especially with the help of someone with more expertise in the field.

1. Plot/Overview: To help with any potential text analysis, we wanted to make sure there were no empty cells for Movie Plot or Movie Overview. These two features are fairly similar, with the Plot usually a shorter summary of the movie compared with the Overview. However, to make any text analysis easier and to avoid removing rows due to missing data, we decided to fill in any remaining missing cells in the Movie Plot column with the data in the Movie Overview column, if it existed. The same was done if the Movie Overview column had a missing cell, it would be filled with the Movie Plot, if it existed.
2. IMDB votes: The IMDB votes column would be converted to a float and any commas would be deleted.
3. Rotten Tomatoes rating: Any percentage signs (%) would be removed from the Rotten Tomatoes rating column and the resulting number would be converted to a float.
4. Impute Rotten Tomatoes, IMDB, Metacritic ratings: We decided to impute any missing Rotten Tomatoes, IMDB, or Metacritic ratings. Because these were numeric columns, and because these ratings tended to follow a normal distribution, imputation was a reasonable way to fill in any missing data our team was not able to fill in. The imputation went as follows:
   1. We impute any missing ratings based on what other ratings are available in our dataset, and also by a ranking of how important a given ratings source is.
      1. Of the three ratings we have in our dataset, we believe the best ratings site is Rotten Tomatoes. It’s a site that nearly everyone uses and is aware of. The second best ratings site is Metacritic. The last is IMDB. We put IMDB last in our ordering because the ratings we have in our data set is from IMDB users, who may have different tastes compared with professional movie reviewers and may not have a steady-state score before a movie comes out.
      2. Both Rotten Tomatoes and Metacritic are on a 0-100 scale, while IMDB is on a 0-10 scale.
   2. So, if the Rotten Tomatoes score is missing, but Metacritic rating is available in our dataset, we simply use the Metacritic score as the Rotten Tomatoes score.
      1. If Metacritic is also missing, but the IMDB rating is not, we take the IMDB score, multiply it by 10, and use it as the Rotten Tomatoes score.
      2. If both Metacritic and IMDB ratings are missing, then we take the median Rotten Tomatoes score of the dataset.
   3. The same algorithm is used if we start with the Metacritic ratings column or the IMDB ratings column. Note that if IMDB ratings are missing, but the Rotten Tomatoes or Metacritic ratings are not, both would have to be divided by 10 to get it into the IMDB ratings scale.
5. Running sum of movie’s cast and crew past revenue: Based on some literature review and discussion a subject matter expert, we decided to create some new features that could be used to indicate how popular the movie’s cast and crew are. For each movie, we calculate a running sum of the historical movie revenues associated with the top five actors, collectively, of the movie in question that exist in our dataset. This would be stored in “Revenue \_Actor” and “Revenue\_Actor\_Real” for both nominal and real dollar sums. Similarly for each movie, we would calculate the same running sum of the historical movie revenues associated with the director, writers, and producers of the movie in question. These would be stored in “Revenue\_Director”, “Revenue\_Director\_Real”, “Revenue\_Writer”, “Revenue\_Writer\_Real”, “Revenue\_Producer”, and “Revenue\_Producer\_Real”.

There should be caution when using “Revenue\_Producer” or “Revenue\_Producer\_Real”. Many movies in our dataset are missing the names of the producers [how many], and our team did not put effort into filling these in if they were missing. As a result, the running sum may not be accurate. Further, there will be issues with any of these variables if some data sources spell people’s names differently over time or if there are spelling mistakes in any previous instance of a person’s name in our data set. Lastly, because these are running sums based on only the movies we have in our dataset, the first few years of movies in our sample will inevitably show no historical revenues for the actors, directors, writers, and producers because we don’t have information prior to 1995. Some further refinements of this variable could be to only do a running sum of the past five years of data, to potentially be a more accurate indicator of the current popularity and success of the cast and/or crew.

## Part 4B: Final Dataset

In addition to these variables that we created there are few features leftover from our data merging process, like a ratings or popularity scores developed by MovieLens or TMDB. These are not usable features because it is not guaranteed that these scores would be available in time for any new movie. However, these features are still left in our final dataset. After all the data wrangling and feature engineering, we have 86 columns of data and 6,411 movies. It should be noted that this dataset will be further refined during the modeling phase of the project, as described in Section 5. The column names in our dataset were renamed to slightly shorter and more understandable names. Finally, this dataset was stored in a table called “finalMovies\_20180814” in our previously created SQLite Database “movies.db”[[22]](#footnote-22).

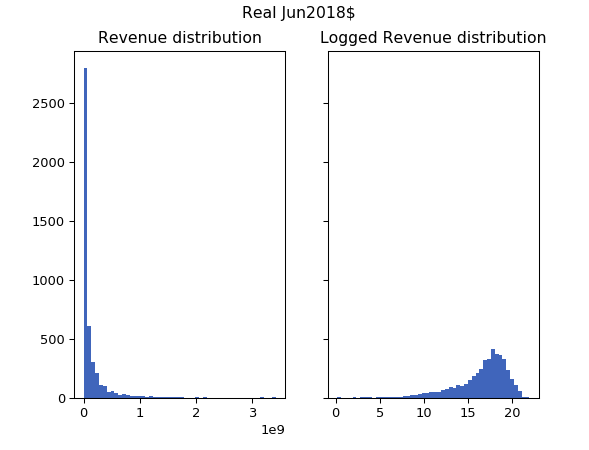
# Section 5: Modeling

## Part 5A: Data exploration

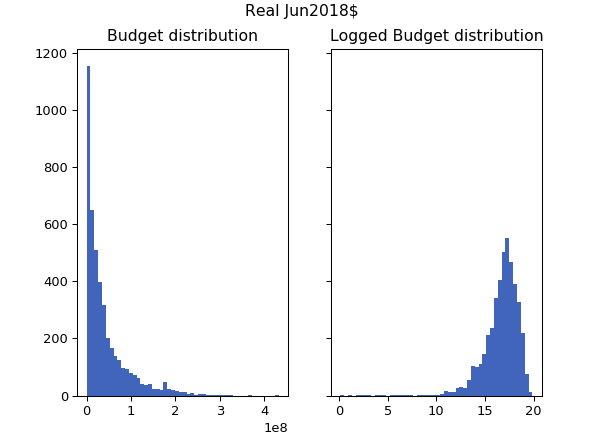
Starting with Section 5, we started coding in Jupyter Notebook. The first part of this phase of the project is to finally explore the data set we created by merging and processing all the data we obtained from our sources. The data exploration notebook is called “exploreMoviesData.ipynb”[[23]](#footnote-23). We used the data stored in table “finalMovies\_20180814” in the SQLite database “movies.db”. As mentioned in Section 4, Part 4B, the final data set had 86 columns and 6,411 movies. All these movies have a revenue number based on the work explained in Section 3. However, because we would like to perform analysis on movies that have budgets filled in and a variety of other factors filled in, we will limit the data set even further by restricting our data set to just those rows with a non-null Budget and non-null Length columns. Any column could have been chosen; choosing to filter on Length, in addition to Budget, helped to further filter out movies that had very few features filled in. Based on our knowledge from looking at the data, as described in Section 3, Part 3D, virtually all rows of data that were sparsely populated were also missing Length.

With this additional filter, we were left with a data set of 86 columns and 4,414 movies. From this, we began exploring several of the features and their correlations.

The figure below shows a histogram of the movies’ revenues, in both levels and logs. Level revenues are highly skewed, so taking the log of the number helps make the distribution look more normal.

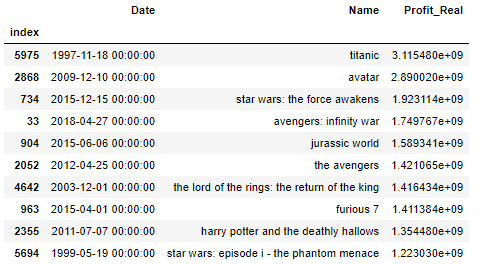


Similarly, the figure below shows movies’ budgets, in both levels and logs. Level budgets are highly skewed, so taking the log of the number helps make the distribution look more normal.

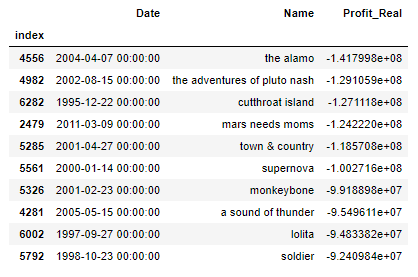


For personal interest, we wanted to look at the most profitable and least profitable movies, on a real June 2018 dollar basis, in our data set.

Top ten most profitable movies:

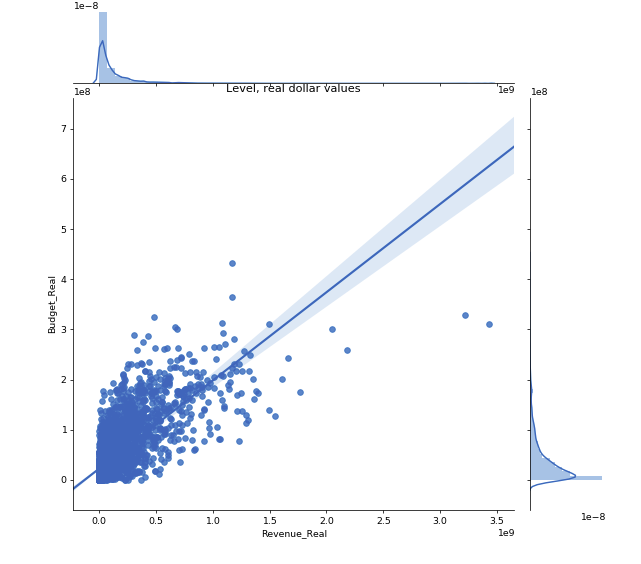
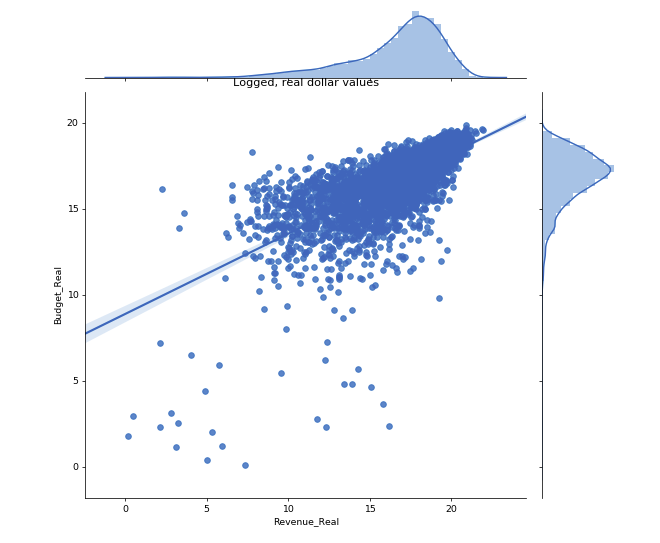


Top ten least profitable movies:

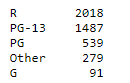
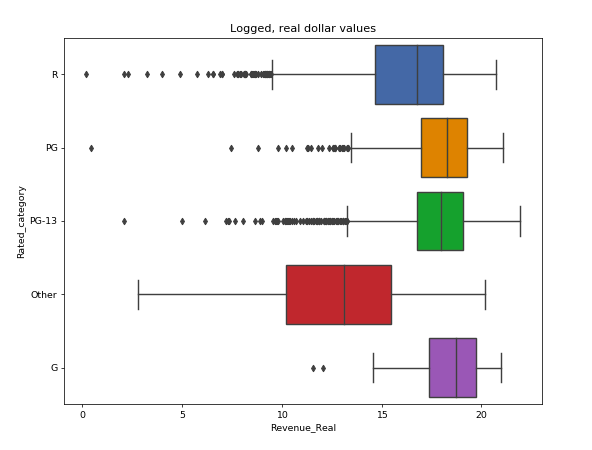
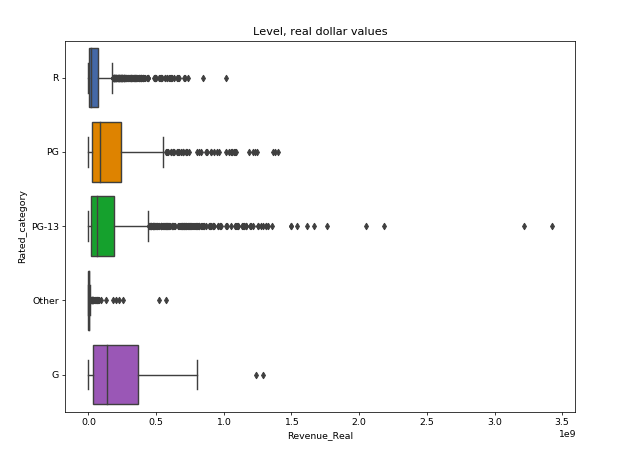


These movies generally correspond to lists found online about the worst or best performing movies. Some differences in movies or rankings could occur based on what inflation rate others used to deflate dollar values and also what the base year was used.

The following graphs are scatter plots of both logged and level values for deflated budget and revenue data. They show a distinct, positive correlation.

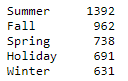


The following graphs are box plots of logged and level values for deflated revenues by movie rated categories. Because movies can only have one MPAA rating, there is no double counting. In our dataset, nearly 80% of movies were rated R or PG-13. Movies rated PG-13 seem to earn the most revenue, with some extreme outliers. Movies rated R or Other seem to have lower revenues. This confirms both literature and intuition because movies that are rated R will have limited audiences and movies rated Other may be those rated as an even more strict NC-17 or may not be rated at all.

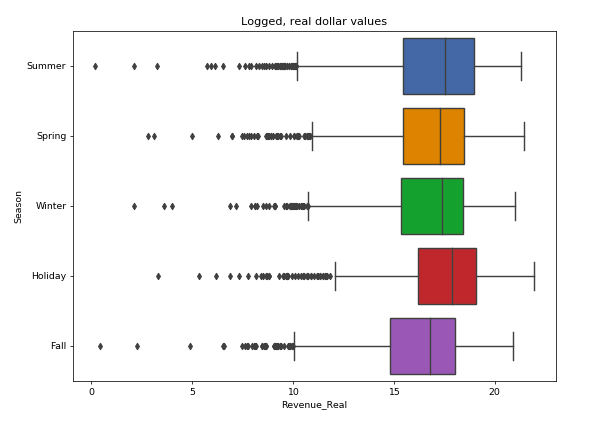
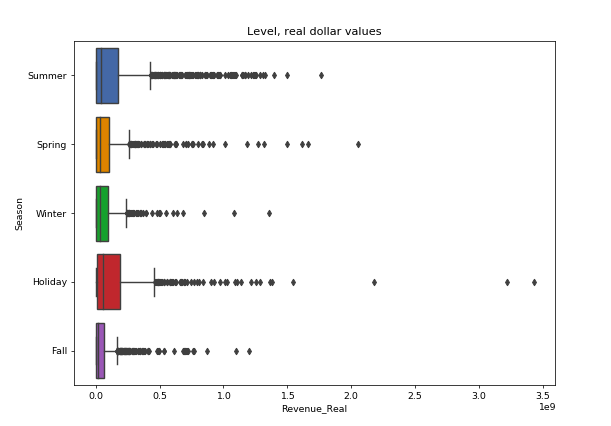


Rated # movies

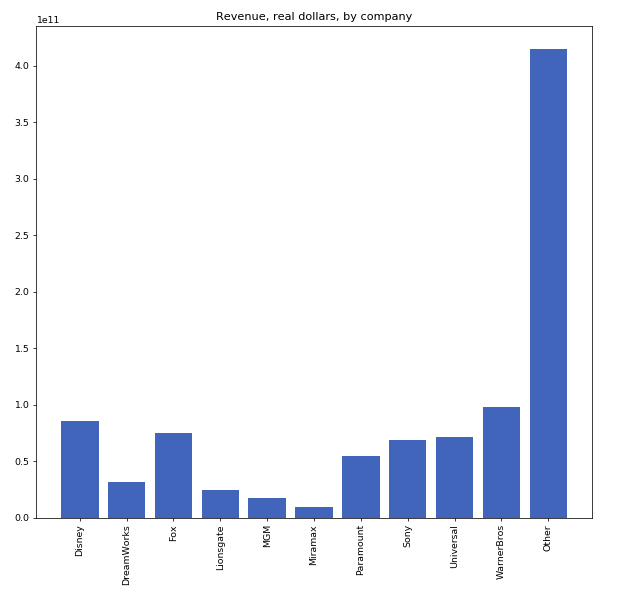
The following graphs are box plots of logged and level values for deflated revenues by the time of year the movies were released. Nearly a third of all movies were released in the summer time. There are some extreme outliers in the movies released in the Holiday season, while movies released in the Fall seem to have the lowest revenues.



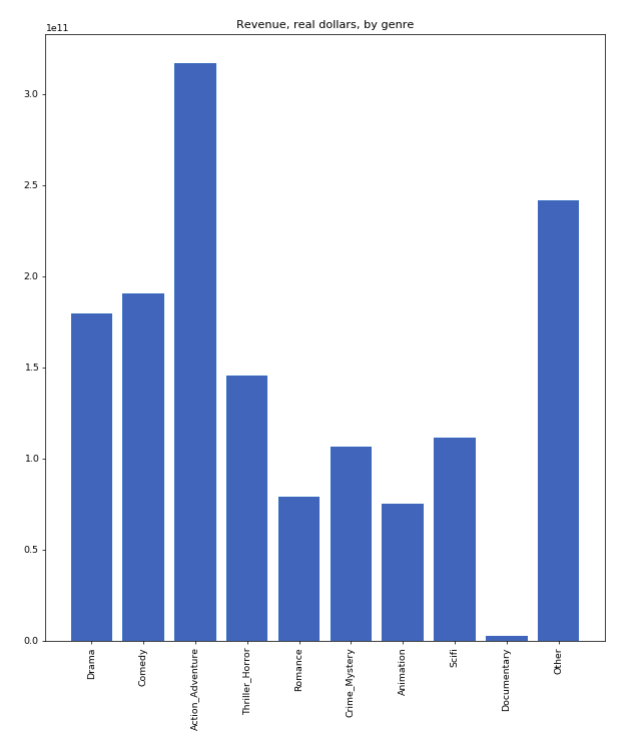
Season # movies



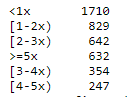
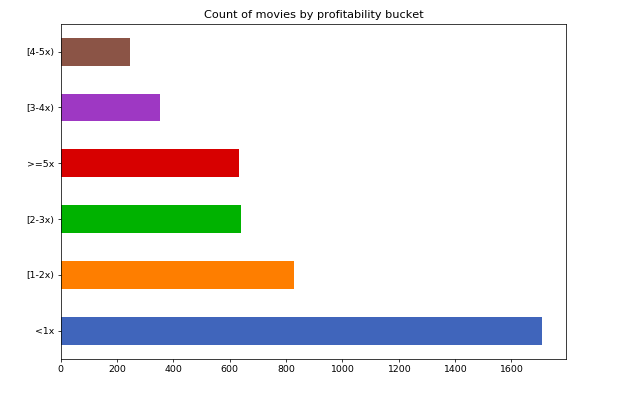
The following graph shows revenue by production company. Because movies can have multiple production companies associated with it, there is double counting in these numbers. Based on our data, major production companies have released a few hundred movies over the entire time span of our data set. That generally matches what can be found on BoxOfficeMojo on the market share of production companies[[24]](#footnote-24). In their data, major companies produce movies that number in the low teens or twenties each year.



The following graph shows the revenue by genre. Because movies can have multiple genres associated with it, there is double counting in these numbers. Action and/or Adventure movies earn the most amount of money in our data set.

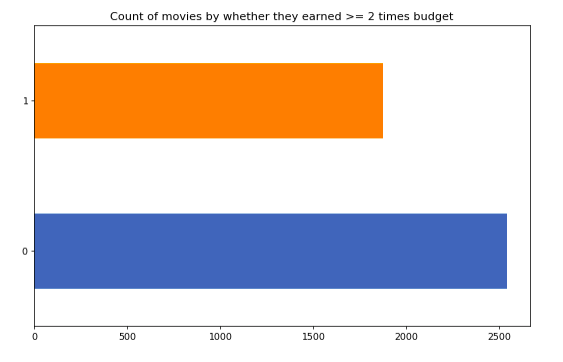


Lastly, we looked at a count of movies by their profitability. We use the Profit Bucket feature generated in Section 4, Part 4A, where movies are categorized by how much revenue was earned in relation to its production budget. Based on our dataset, a plurality of movies (almost 40%) do not earn back enough to even offset just their production budget.



Bucket # movies

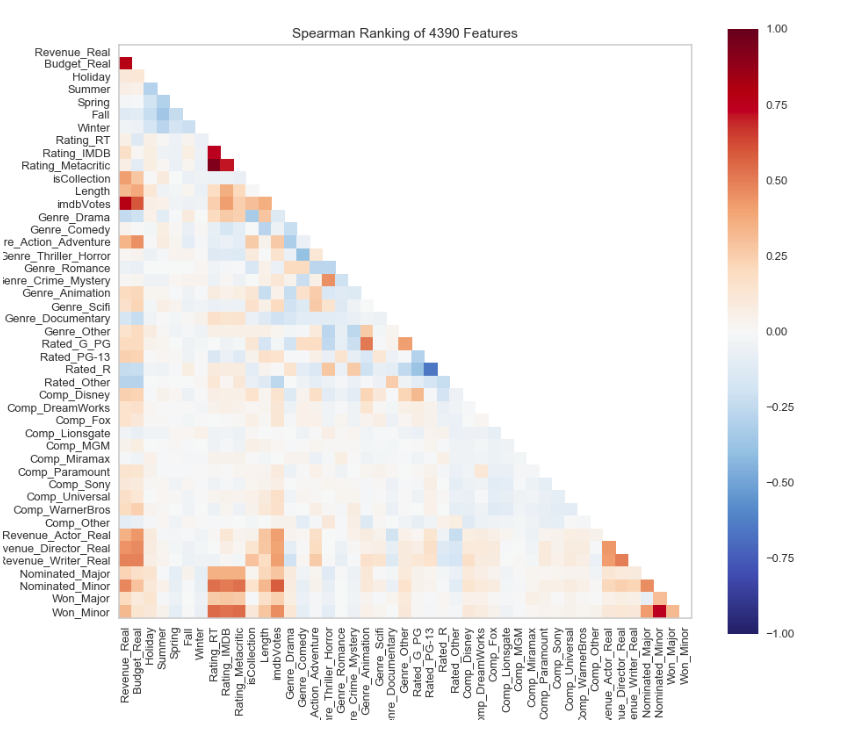
A further grouping that we performed, and will eventually use in our classification models, is to group movies based on whether they earned at least twice as much as their production budget. This feature is called Profit Bucket Binary. This follows the literature described earlier on how to measure whether a movie actually made a profit, after considering additional costs such as marketing and distribution. A 1 indicates the movie earned at least twice as much, while a 0 indicates otherwise. Still, a majority of movies did not make a profit under this classification.

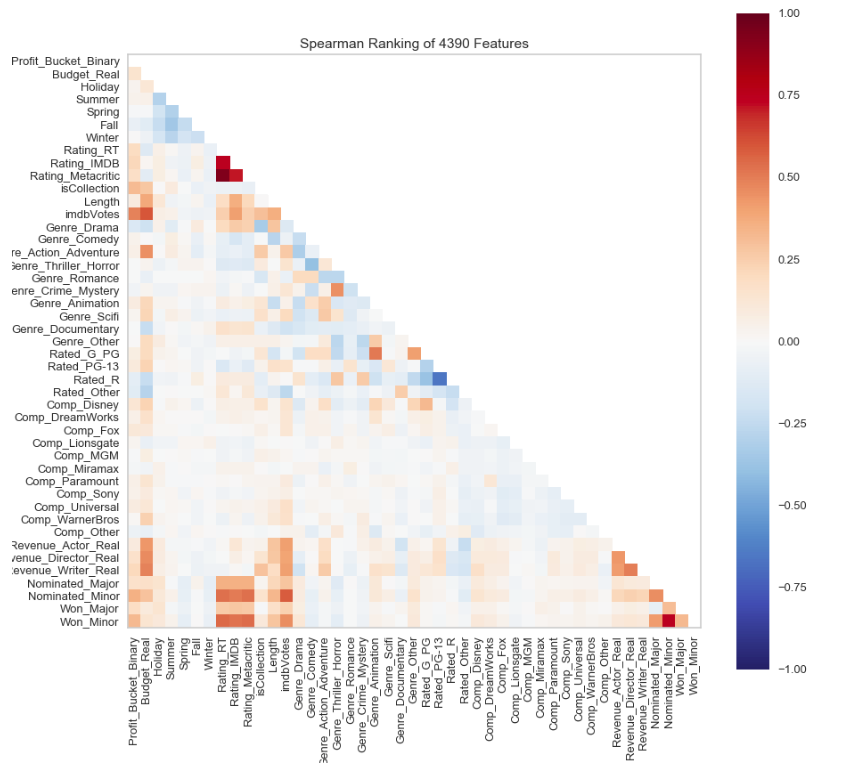


Bucket # movies

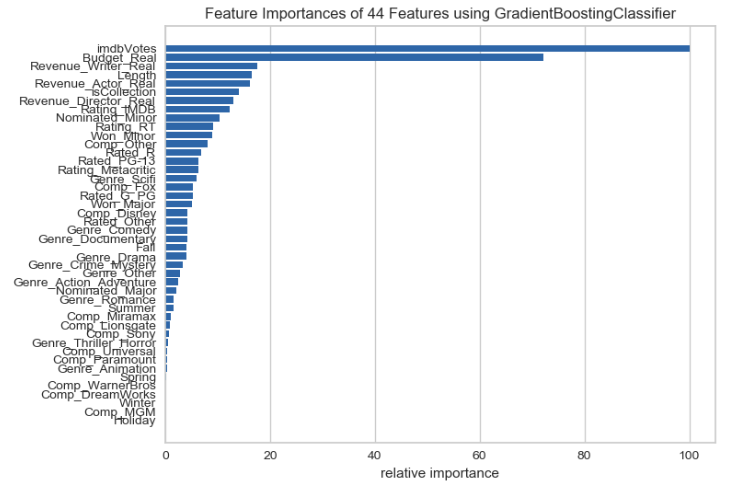
## Part 5B: Feature selection

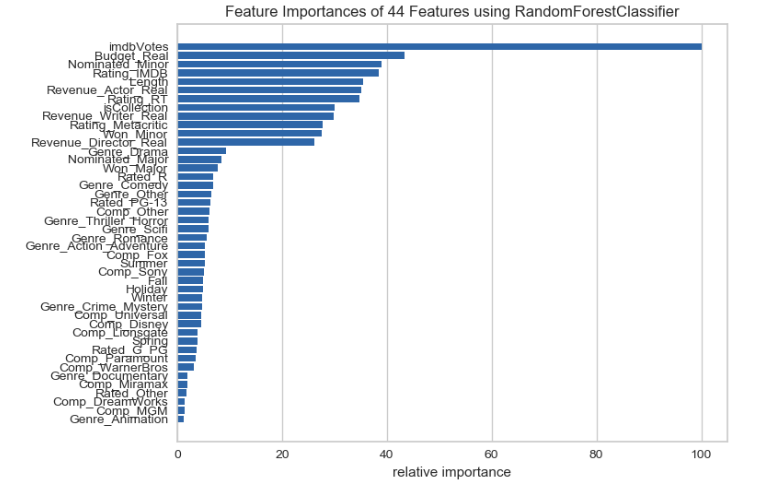
Our feature selection notebook is called featureSelectionMoviesData.ipynb[[25]](#footnote-25). The first step of feature selection was creating a Rank2D visualization from the YellowBrick package of all the variables to see what correlated with Revenue. Revenue is the first column in the following Rank2D visualization. The most positively correlated features were Budget, IMDB votes, whether the movie is a part of a collection, past revenues of a movie’s writers and directors, how many minor award nominations the movie received, and whether the genre was Action and/or Adventure. The more negatively correlated features were if a movie was rated R or Other, if the genre was Documentary or Drama, or if the movie was released in the Fall.

In terms of correlations among the features themselves, as expected, the Rotten Tomatoes rating, IMDB rating, and Metacritic rating are highly correlated with each other. In addition, the number of award nominations or wins is correlated with movie ratings. 

Another Rank2D visualization was done using the Profit Bucket Binary variable, which equaled a 1 if the movie earned back at least twice its production budget. The correlations are less strong, but this may not be the most accurate way to test the correlation of a binary variable.

We also used the YellowBrick feature importance visualization. At this early stage, we decided to look at all the variables we had and use the Gradient Boosting Classifier and the Random Forest Classifier models. We used the Profit Bucket Binary variable as the two classifications to solve for. It can be seen in the following visualizations that for both these ensemble-type models, the two most important features are the number of votes on IMDB and the Budget. Other important variables include historical cast and crew revenues, the movie length, award nominations, and whether the movie is part of a collection.





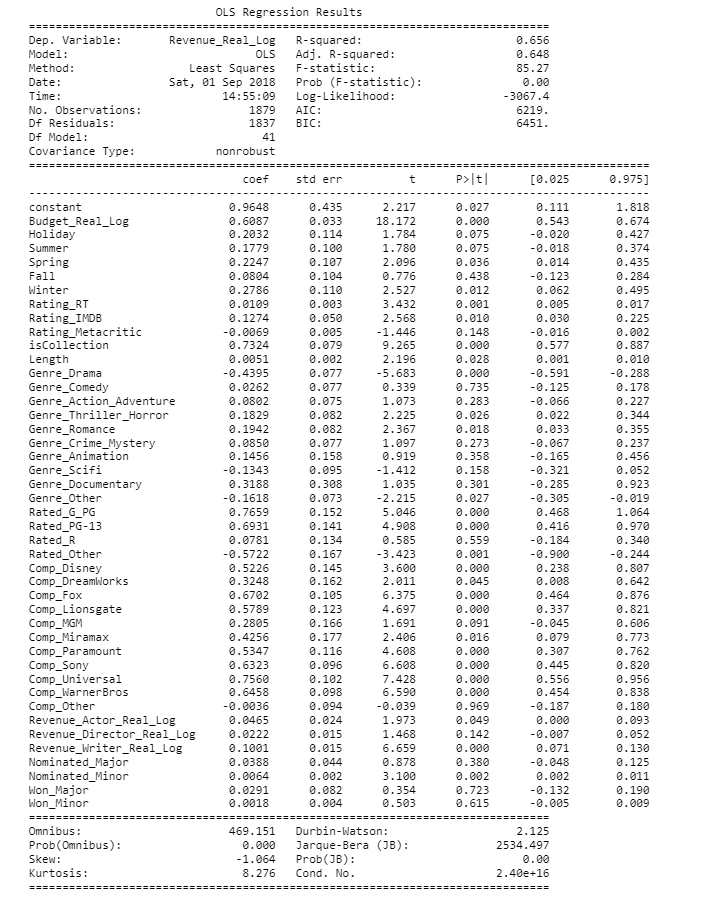
An important note to consider is that the number of votes a movie has on IMDB changes drastically during and after a movie’s run because the people voting are regular users who vote after they watch movie. A higher number means more people have watched the movie, leading to the high correlation with revenue. However, this means an accurate number of votes would not be available prior to the movie’s release, which is when we would like to perform our predictions.

In the same realm, movies tend to be nominated or win awards after they have been released and have been seen by the public or members of some voting academy. So these numbers would not exist before a movie has premiered.

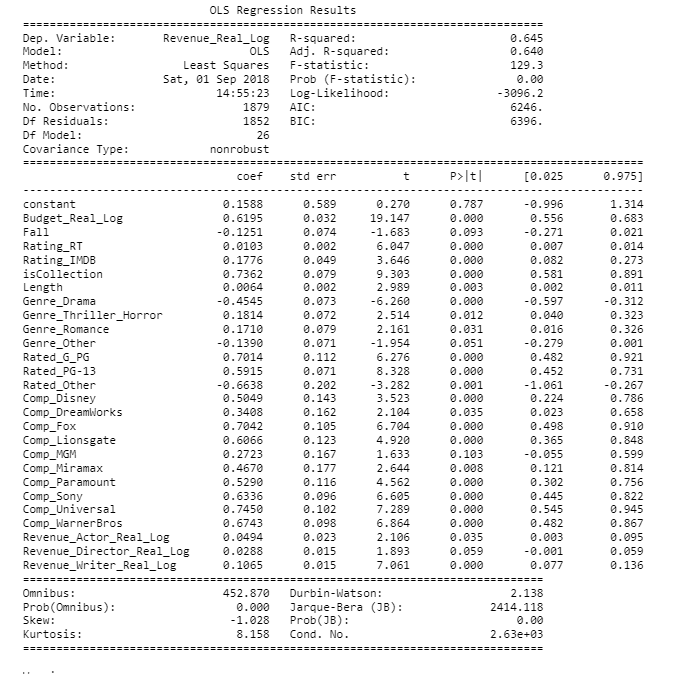
Because of this, we decided to avoid including IMDB votes or any related awards features in our modeling. However, it is still an interesting finding that these variables are relatively good predictors of revenue and profit.

## Part 5C: Regression Models

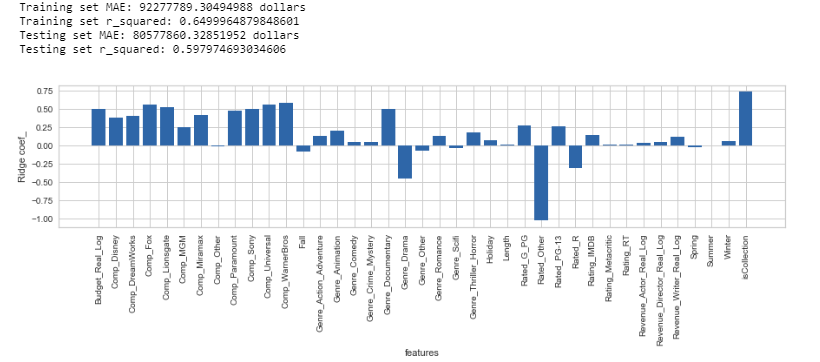
The first type of model we attempted to use was a regression model. The regressions are stored in the modelMoviesData\_Regression.ipynb[[26]](#footnote-26) notebook. Initially we simply include all features, excluding the ones mentioned in Section 5, Part 5B that would not be available prior to a movie’s release. The movie Revenue, Budget, and historical cast’s and crew’s revenues were logged to help with the skewed nature of those features. Doing this, however, prevents us from using any rows where the historical cast or crew revenues are 0. As a result, this limits our data set down to 1,879 movies, and prevents analysis on movies that may employ newer talents with little to no revenue history. An intercept was added to the regression. From this, an R-squared of 0.66 was achieved.

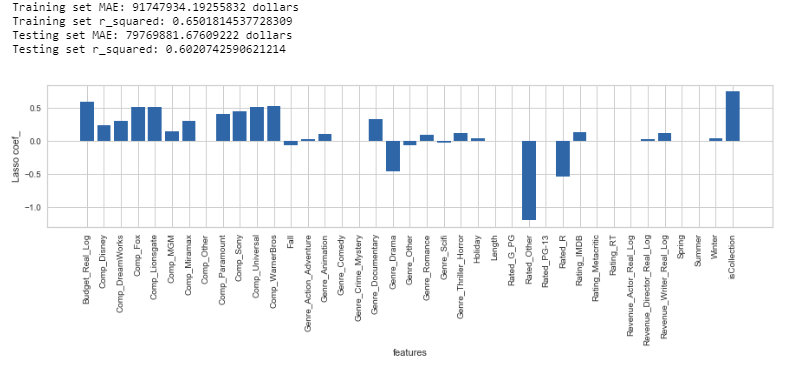


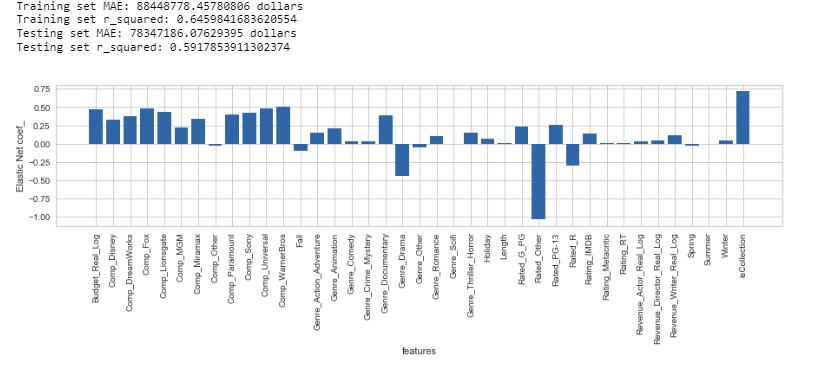
From this, we experimented with dropping some non-statistically significant variables and to avoid the dummy variable trap. This effort produced a regression with an R-squared of 0.65. Interestingly, no seasonal variable (to indicate the time of year a movie was released) seemed to be significant. Fall was significant only at the 10% level, and had a negative coefficient, which matches the finding in the movie exploration phase. Movies that were Dramas had a statistically significant negative coefficient. Lastly, one of the most interesting findings is that the writers involved in the movie had a bigger impact on the movie’s revenue than did the actors or directors involved.



The next step was to experiment with regularization models and use Ridge, Lasso, and ElasticNet regressions to see if those models could provide a more robust way of choosing important variables. We split our data set into training and testing sets using an 80/20 split. We then use the RidgeCV, LassoCV, and ElasticNetCV models in Scikit-learn. The following diagrams show the R-squared and mean absolute error of each of these models, along with the relative importances of the features.

RidgeCV:

LassoCV:

ElasticNetCV:

For each of these three models, the test set R-squared is around 0.60 and the mean absolute error is around $80,000,000. Generally, for all these models, the most important variables that contributed positively were if the movie was a part of a collection, the Budget, whether the movie was a Documentary, and, to varying levels, the production company associated with the movie. The most important variables that contributed negatively was whether the movie was rated Other or if the movie was a Drama.

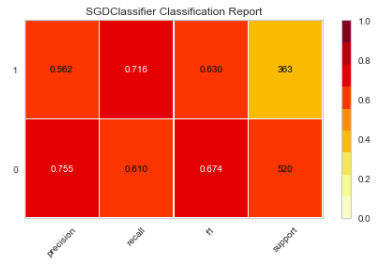
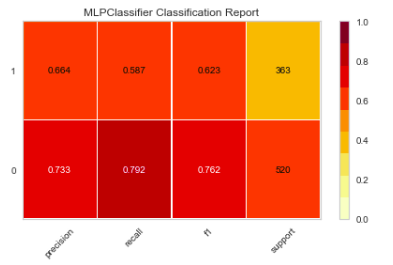
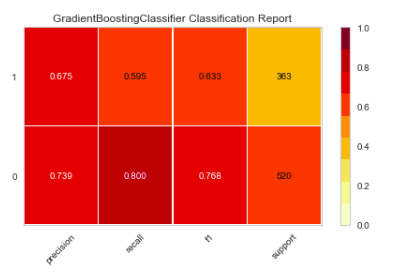
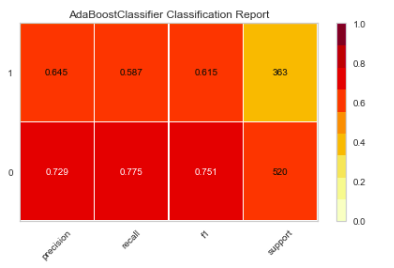
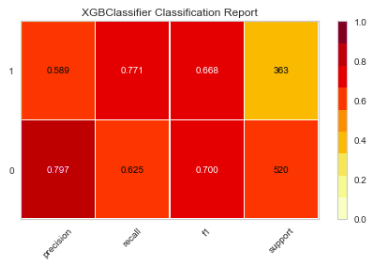
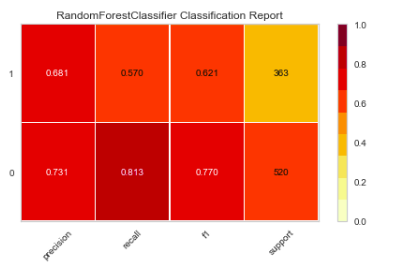
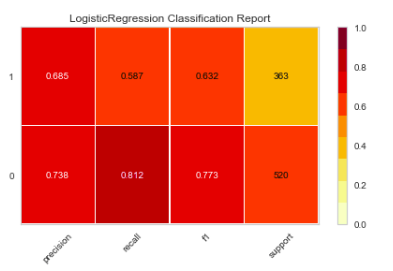
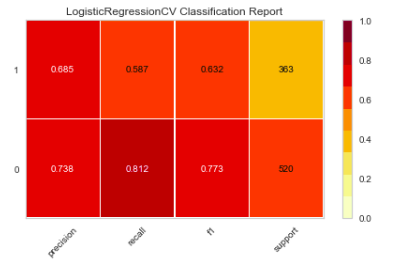
We perform one last test on these models and implement a 10-fold cross validation test to see which model has the highest R-squared. These models all had R-squared very close to each other at 0.62, but we choose LassoCV which had the slightly higher R-squared. This model was stored in the “final\_models” folder in our GitHub as “lasso\_20180901.pkl”[[27]](#footnote-27).

[talk about post 2008?]

## Part 5D: Classification Models

In addition to using regression models, we can also use classification models. Our Profit Bucket Binary variable serves as a two classification scheme, where a movie is classified on whether it earned at least twice as much as its production budget. The notebook that contains the classification models is modelMoviesData\_Classification.ipynb[[28]](#footnote-28). We start by including all the features we have, except those we removed based on their availability before a movie is released. Then, the features are scaled using the MinMaxScaler in the Scikit-learn Preprocessing package.

Then we start with a collection of classifiers: Logistic Regression, Random Forest, XGBoost, Extra Trees, Ada Boost, Gradient Boosting, MultinomialNB, GaussianNB, SGD Classifier, Neural Network, Support Vector Classification, and K-Nearest Neighbors. The data set was split into training and testing sets using an 80/20 split. We use the YellowBrick function to calculate the Classification Reports and ROC AUC curves for each of these classifiers. These classifiers were then initially reduced down to those whose F1 scores were at least 0.60 when predicting movies that earned at least twice as much as their budgets (i.e., classification = 1). This was a simple way to start reducing the number of classifiers we were using:



Next, 10-fold cross validation was used to further check how these selected models perform. All the selected models still give at least a 0.60 F1 score, except for SGDClassifier, which we subsequently drop. So we begin hyperparameter tuning using the GridSearchCV function from Scikit-learn on some of the remaining models. Of the models and parameters tested, these were the optimal values using F1 as the scoring metric:

Best parameters set found for Random Forest:

{'class\_weight': 'balanced', 'max\_depth': 10, 'max\_features': 'log2', 'n\_estimators': 60, 'oob\_score': False}

Best parameters set found for Logistic Regression:

{'C': 10, 'class\_weight': None, 'penalty': 'l1'}

Best parameters set found for AdaBoost:

{'n\_estimators': 120}

Best parameters set found for XGBoost:

{'scale\_pos\_weight': 2}

Best parameters set found for Gradient Boost:

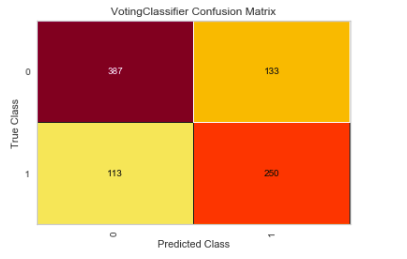
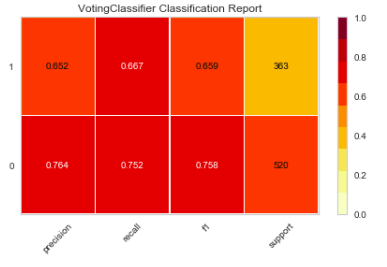
{'criterion': 'friedman\_mse', 'loss': 'exponential', 'max\_depth': 5, 'max\_features': 'log2', 'n\_estimators': 120}

From these optimal values, we then decide to use the VotingClassifier model available in Scikit-learn to determine the best combination of models instead of trying to pick one. We perform hyperparameter tuning on the VotingClassifier model by including the previous five models’ results from hyperparameter tuning and add the LogisticRegressionCV model and the Neural Network model, for a total of seven models. The parameters we test through GridSearchCV is whether to set “voting” to hard or soft and also what combination of models to include, by setting “weights” to either a 1 or 0 for each model. The GridSearchCV results indicated that only the Random Forest, XGBoost, and Neural Network should be included in the VotingClassifier.

Best parameters set found for Voting Classifier:

{'voting': 'hard', 'weights': [0, 0, 1, 1, 0, 0, 1]}

This VotingClassifier model’s Classification Report and Confusion Matrix are:

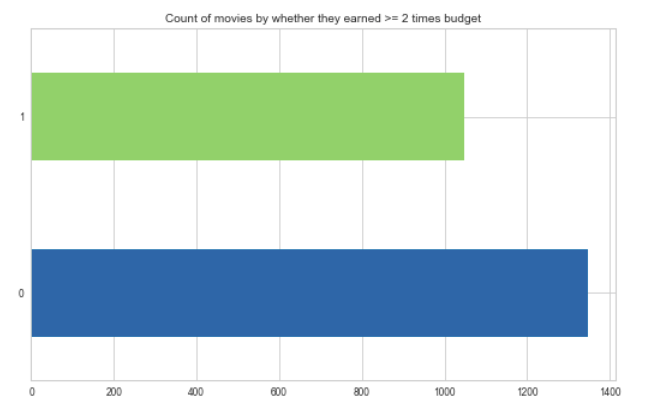


We pick this VotingClassifier model as our classification model and store it in the “final\_models” folder in our GitHub as “votingClassifier\_20180901.pkl”[[29]](#footnote-29).

## Part 5E: Classification Models, restricted period

We wanted to see if these results would change if we restricted the sample of movies in our data set to just those from 2008 and onwards. A few reasons why we wanted to test a shorter time period include the number of years of data our team members were able to manually review, possible structural changes to the movie industry, and the inability to calculate historical revenues of the cast and crew for movies in the early years in our data set.

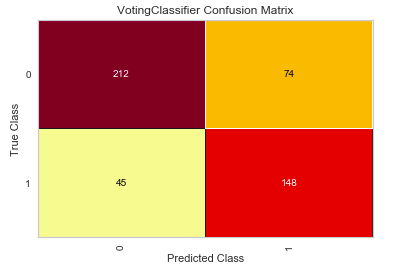
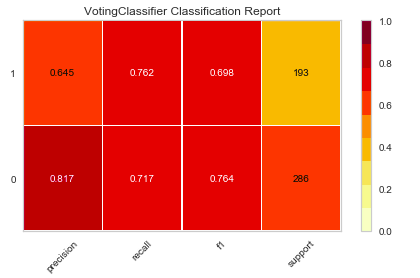
By restricting our data set, we were left with 2,395 movies, with a majority of these movies still not earning at least twice as much as its budget.



Bucket # movies

We run through the exact same procedure as described in Section 5, Part 5D. These models actually perform better, with a lot of models reaching nearly 0.70 F1 score for classification = 1 and close to 0.80 F1 score for classification = 0. After performing a GridSearchCV on the VotingClassifier with the new models, we see the best parameter for “voting” is soft, and the best weights for the models indicate that the best combination of models are Random Forest, XGBoost, and Ada Boost.

This VotingClassifier model’s Classification Report and Confusion Matrix are:



We pick this VotingClassifier as another potential classification model and store it in the “final\_models” folder in our GitHub as “votingClassifierPost2008\_20180901.pkl”[[30]](#footnote-30).

# Section 6: Results Summary

# Section 7: Data Product?

# Section 8: Team member contributions

# Section 9: Appendix?

1. <https://www.kaggle.com/rounakbanik/the-movies-dataset> [↑](#footnote-ref-1)
2. <https://www.kaggle.com/tmdb/tmdb-movie-metadata/home> [↑](#footnote-ref-2)
3. <https://www.the-numbers.com/movie/budgets/> [↑](#footnote-ref-3)
4. <http://www.omdbapi.com/> [↑](#footnote-ref-4)
5. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/API_pulls/pull_OMDB_API.py> [↑](#footnote-ref-5)
6. <https://www.themoviedb.org/> [↑](#footnote-ref-6)
7. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/API_pulls/pull_TMDB_API.py> [↑](#footnote-ref-7)
8. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/API_pulls/pull_OMDB_API_newMovies.py> [↑](#footnote-ref-8)
9. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/API_pulls/pull_TMDB_API_newMovies.py> [↑](#footnote-ref-9)
10. <https://github.com/georgetown-analytics/Box-Office/tree/master/data> [↑](#footnote-ref-10)
11. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/Data_wrangling_code/joinDataModule.py> [↑](#footnote-ref-11)
12. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/Data_wrangling_code/parseColumnsModule.py> [↑](#footnote-ref-12)
13. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/Data_wrangling_code/mergeDataModule.py> [↑](#footnote-ref-13)
14. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/Data_wrangling_code/movieFunctions.py> [↑](#footnote-ref-14)
15. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/Data_wrangling_code/cleanData.py> [↑](#footnote-ref-15)
16. <https://github.com/georgetown-analytics/Box-Office/tree/master/filled_in_data> [↑](#footnote-ref-16)
17. <https://github.com/georgetown-analytics/Box-Office/tree/master/database> [↑](#footnote-ref-17)
18. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/Data_preprocessing_code/preprocessData.py> [↑](#footnote-ref-18)
19. <https://www.boxofficemojo.com/seasonal/> [↑](#footnote-ref-19)
20. <https://fred.stlouisfed.org/series/CPIAUCSL> [↑](#footnote-ref-20)
21. <https://github.com/georgetown-analytics/Box-Office/tree/master/data> [↑](#footnote-ref-21)
22. <https://github.com/georgetown-analytics/Box-Office/tree/master/database> [↑](#footnote-ref-22)
23. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/Data_modeling_notebook/exploreMoviesData.ipynb> [↑](#footnote-ref-23)
24. <https://www.boxofficemojo.com/studio/?view=company&view2=yearly&yr=2017&p=.htm> [↑](#footnote-ref-24)
25. https://github.com/georgetown-analytics/Box-Office/blob/master/codes/Data\_modeling\_notebook/featureSelectionMoviesData.ipynb [↑](#footnote-ref-25)
26. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/Data_modeling_notebook/modelMoviesData_Regression.ipynb> [↑](#footnote-ref-26)
27. <https://github.com/georgetown-analytics/Box-Office/tree/master/final_models> [↑](#footnote-ref-27)
28. <https://github.com/georgetown-analytics/Box-Office/blob/master/codes/Data_modeling_notebook/modelMoviesData_Classification.ipynb> [↑](#footnote-ref-28)
29. <https://github.com/georgetown-analytics/Box-Office/tree/master/final_models> [↑](#footnote-ref-29)
30. <https://github.com/georgetown-analytics/Box-Office/tree/master/final_models> [↑](#footnote-ref-30)