IMDb Movie Rating Preprocessing and EDA

Individual Report – Sahara Ensley

Introduction to Data Mining – DATS6103

Group 2

December 4th, 2021

Table of Contents

Introduction

For our project we decided to use an extensive movies dataset from Kaggle (https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset) containing metadata about ~85000 movies. Through the same source we also had access to information about the cast and crew responsible for producing each movie, as well as rating information about each film. We decided to use this dataset for a few reasons. First, the movie industry is incredibly relevant to modern society and is constantly evolving as we progress through different release formats and advertising strategies. As more money gets poured into the industry, predicting what movies are going to be received well becomes a lucrative question in need of answering. The second reason we chose this dataset builds off this point. This data set contained enough information about the movies, cast, crew, and ratings, that we believed it would give in depth insight not only into the field as a whole, but how the field has progressed through time.

After choosing the dataset, we broke up the work responsibilities. I took the preprocessing tasks, Josh took the modeling tasks, and Adam was responsible for the GUI. We split the report and presentation equally since we were all in charge of different sections. My part of the project was the first that needed to be completed, so a lot of my work was front loaded as we moved through the project.

Background and Description of Work

The first thing that needed to be decided in regards to the preprocessing was what variables we were going to work on. Josh made a very helpful spreadsheet that we all worked on to help us organize what needed to be done to each factor and whether we were going to keep it.

Column Name	Use in Model?
imdb_title_id	N
original_title	N
year	N
language	N
avg_vote	N
votes	N
metascore	N
height	N
date_of_birth	N
% male/female	N
reviews_from_users	N
reviews_from_critics	N
females_allages_avg_vote	Potential Target Label
males_allages_avg_vote	Potential Target Label

weighted_average_vote	Target Label
title	Y
date_published	Y
genre	Y
country	Y
director	Y
writer	Y
production_company	Y
actors	Y
description	Y
duration	Y
budget	Y
worldwide_gross_income	Y
usa_gross_income	Y

Once we had a general strategy for what was going to be done, I began laying out what needed to be completed for preprocessing.

1. The data needed to be loaded

2. The important features needed to be extracted, cleaned if necessary, and merged into one main data frame
3. The data needed to be split into train, test, and validation data sets.
4. Individual feature transformations applied
5. Data returned in a useful way

Each of these steps were able to be neatly packaged into separate subroutines in my code file. This functionalization of the different steps made the code easiest to read and use. I knew that I needed to make the code callable from one line to make Josh and Adam's section the most streamlined, with this in mind I set out to make a single 'Preprocessing' function that then called each individual step and returned a cleaned test, train, and validation test set, as well as anything else that was needed for the GUI, model, and EDA.

My work was put into 2 python files in my own personal code folder (sahara-individual-project/Code), and eventually moved to the main Code folder once it was ready to be used by Josh and Adam. The first python file is preprocessing_utils.py, this file contained all functions that were responsible for cleaning the data. The second file was EDA.py, this is where I put all of the code I used to create plots and run any statistics on the dataset. The code in this file was eventually moved to TestEDA.py in the main Code folder.

Individual Work

PREPROCESSING_UTILS.PY

The most important code I worked on was the preprocessing_utils.py file. This file contained all the necessary functions for preprocessing.

Loading Function

The first thing I wrote was the loading function. Adam was responsible for writing a data download function that pulled data from the internet, while he worked on that I used a temporary loading function that pulled from my local computer. The temporary function I wrote that we used until that was done was as follows:

Function	Output	Location
`load_all(base)`	All five necessary data frames as pandas DataFrame objects	Preprocessing_utils.py

Cleaning and Merging

Following this we needed to do initial cleaning and merging of data frames. The initial Movies dataset contained 22 features and 85855 observations while the initial Ratings dataset contained 49 features and 85855 observations. Given that the unique identifiers in these datasets were the same, 'imdb_title_id' we were able to merge these two datasets seamlessly. We also obtained a dataset containing inflation information (https://fred.stlouisfed.org/series/CPIAUCNS). This dataset contained the Consumer Price Index (CPI) by year dating back to 1913. CPI is a measure how the price of a fixed amount of goods and services changes over time. By indexing this dataset to the CPI of January 2021 we were able to get a multiplier that allowed us to convert a US dollar amount from any year into 2021 dollars. Given that we have the year a movie was produced, we were able to merge this inflation multiplier into the full movies dataset by the year column. The Names dataset was then merged with the Title_principals dataset by the unique actor identifier. The merging functions also dropped any unnecessary columns. The columns we kept can be found in the reference to the spreadsheet above. Following this I wrote a function that would perform the train test validation split so it was easier to call and could be done once and preserved through the cleaning process.

The merging and cleaning steps were accomplished with the following functions.

Function	Output	Location
`clean_inflation(inflation)`	Returned inflation data frame with only multiplier and year	Preprocessing_utils.py
`merge_and_clean_movies(movies, ratings, inflation)`	Returned movies data frame merged with ratings and only including necessary columns	Preprocessing_utils.py
`merge_and_clean_names(names, title_principals)`	Returned names data frame merged with title_principals	Preprocessing_utils.py
`get_train_test_val(data, test_size = 0.3, val_size = 0.2)`	Returned data split into train test val split by given parameters	Preprocessing_utils.py

Following this step, the main transformation wrapper was called and the individual features were transformed and then scaled.

TRANSFORMATIONS

First up were the monetary columns. *Budget*, *World Income*, and *USA Income* were all cleaned in the same way. First we removed any value that was not represented in US dollars. This was determined by finding the values that did not start with "$" but instead another identifier. Once this was done the "$" was removed from the values and they were converted to integers. Finally the budget multiplier was applied to the value and the original money columns were removed from the dataset. The formula for converting the money columns is as follows:

$$AdjustedDollars = \frac{CPI_{2021}}{CPI_{year}} * OriginalDollars$$

Where *CPI-year* is defined as the CPI value for the year the movie was released and *CPI-2021* is defined as the CPI value for January 2021.

This was accomplished with the following helper function.

Function	Output	Location
`clean_money(money)`	Monetary value stripped of special characters and casted to an int	Preprocessing_utils.py

Next the *Date* was transformed by expanding it into 3 separate columns, *date_year*, *date_month*, and *date_day*. For example "11/02/2021" would become three values of *date_year* - 2021, *date_month* - 11, and *date_day* - 02. There were no null date values so imputing was not a concern.

This was accomplished with the following helper function.

Function	Output	Location
`expand_date(df, col_to_expand, keep_original = False)`	Date value expanded into 3 columns in the given data frame.	Preprocessing_utils.py

Country was transformed by taking the primary country of origin, defined as the first listed country in the given string, and mapping it to its associated *Region.* This was done by finding the countries ISO code and corresponding region code through the UN database. (https://raw.githubusercontent.com/lukes/ISO-3166-Countries-with-Regional-Codes/master/all/all.csv). This was accomplished with the following functions.

Function	Output	Location
`get_primary_country(x)`	Given a string, returned the first listed country and mapped it to the correct name if necessary	Preprocessing_utils.py
`to_region(df)`	Mapped a given country to it's appropriate region	Preprocessing_utils.py

Genre was originally set up similar to country as a string containing 1 to 3 listed genres. These values were listed alphabetically and therefore could not be reduced to a primary genre. We transformed the string value to a list of 1 to 3 genres. These values were then transformed by taking each unique combination of genres and mapping it to a binarized string using log base 2. This binarized string was then mapped to 10 binary columns in our dataset labeled as *genre_1* through *genre_10*. There were 732 unique genre combinations, since log2(732) = 9.15 all combinations were captured in these 10 columns.

Cast, *crew*, and *production company* were all transformed in the same way. Popularity of cast and crew was approximated by taking a weighted frequency of appearance in the training dataset. The cast and crew were listed in order of importance and their frequency was weighted by this order. Because there was only one production company there was no weighting necessary. Any missing frequencies were filled with the median value, this was important if the test dataset saw a novel name, that frequency was interpreted as the median.

Title and *description* were also transformed the same way. Both features were expanded into 4 columns. The first feature extracted was the number of words, this became *title_n_words* and *description_n_words*. The ratio of long words was calculated by # long words / # total words where long words is defined as having more than 4 letters. This became *title_ratio_long_words* and *description_ratio_long_words*. The ratio of vowels was determined by # vowels / # total letters. This became *title_ratio_vowels* and *description_ratio_vowels*. The ratio capital letters was determined by # capital letters/# total letters. This became *title_ratio_captial_letters* and *description_ratio_capital_letters*. Once these 4 new features were extracted for both *title* and *description*, the original columns were dropped.

Imputing and Scaling

Once the transformation of each column was done, any missing value was imputed. This was done by determining what columns had missing values and imputing the necessary columns. To do this I made a function that calculated the percentage of each column that was missing and split it into numerical and categorical variables in case we needed to use different methods of imputing. This was accomplished with the following function.

Function	Output	Location
`get_missing(df)`	Returned numerical and categorical features and the percentage of missing values	Preprocessing_utils.py

Following imputing, all features were scaled using Sklearn's Standard Scalar fitted to the training dataset and applied to the testing and validation datasets.

Wrapper Functions

The final preprocessing function that gets called by the other modules consists of only 5 function calls. That wrapper function and the final transformation function are as follows.

Function	Output	Location
`autobots_assemble(df_train, df_test, df_val, names, target)`	Transformation function. Returns transformed train test and validation data frames	Preprocessing_utils.py

`preprocess(ratings, movies, names, title_principals, inflation)`	MAIN WRAPPER. This is what gets called by other functions.	Preprocessing_utils.py

TestEDA.py

Once the preprocessing was completed. I began work on our EDA. My goal for this section was to create simple, explanatory, plots that showed how our data was set up and potentially bring to light any interesting patterns before we got into modeling. The file TestEDA.py contains 14 individual functions, 12 of which produce plots. 2 of the functions perform low level statistical analyses that we ran for the report and for the GUI.

Function Descriptions

The following functions can be found in the TestEDA.py file and run to produce plots or statistics.

Function	Description
`plot_duration(df)`	Plots histogram of duration
`plot_budget(df)`	Plots histogram of adjusted budget
`plot_usa_income(df)`	Plots histogram of adjusted USA income
`plot_worldwide_income(df)`	Plots histogram of adjusted worldwide income
`plot_votes(df)`	Plots histogram of weighted average votes
`plot_vote_by_budget(df)`	Plots the average vote by the adjusted budget
`plot_worldwide_income_by_date(df)`	Plots the worldwide income by the date
`plot_USA_income_by_date(df)`	Plots of the USA income by date
`plot_vote_by_decade(df)`	Plots the average vote by the decade the movie was released
`plot_region_count(df)`	Plots the counts per region
`plot_corr(df)`	Plots the numeric correlation matrix

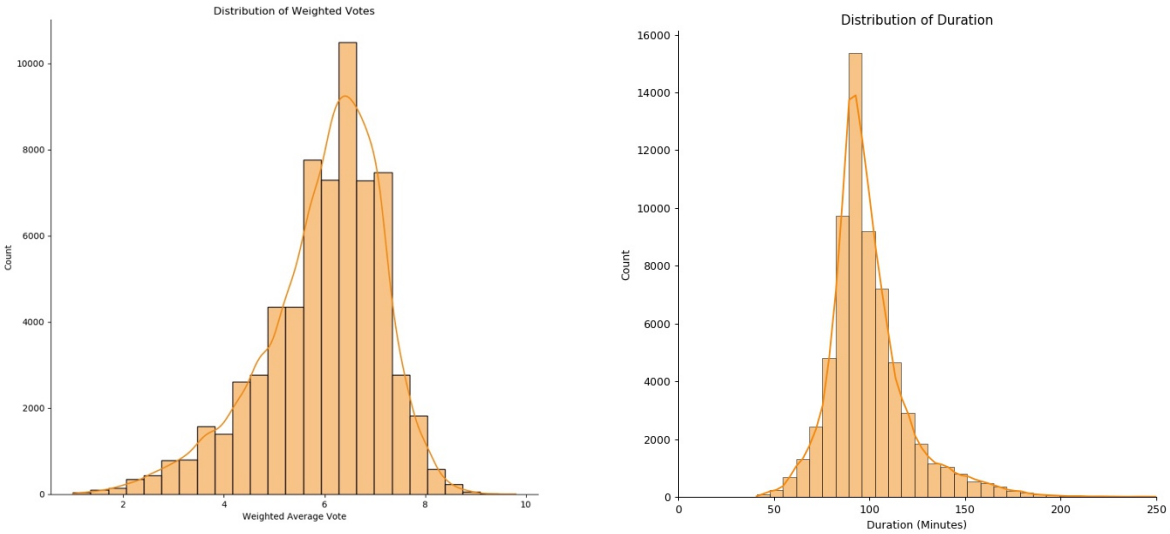
`statsdf(df)`	Creates a statistics data frame with basic stats
`moneytary_plots(df)`	Creates a plot with the average vote plotted against every monetary column
`decade_anova(df)`	Performs the ANOVA on the average vote by decade

RESULTS

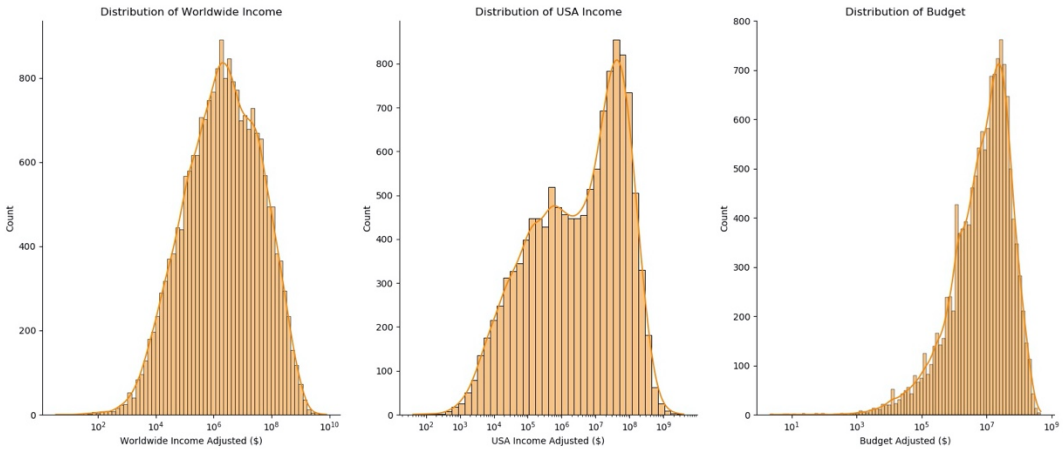
The results from my section of the project are best shown through the EDA plots that I created following the preprocessing.

NUMERICAL/CATEGORICAL VARIABLES

First we have the simple histograms that describe the numeric variables.

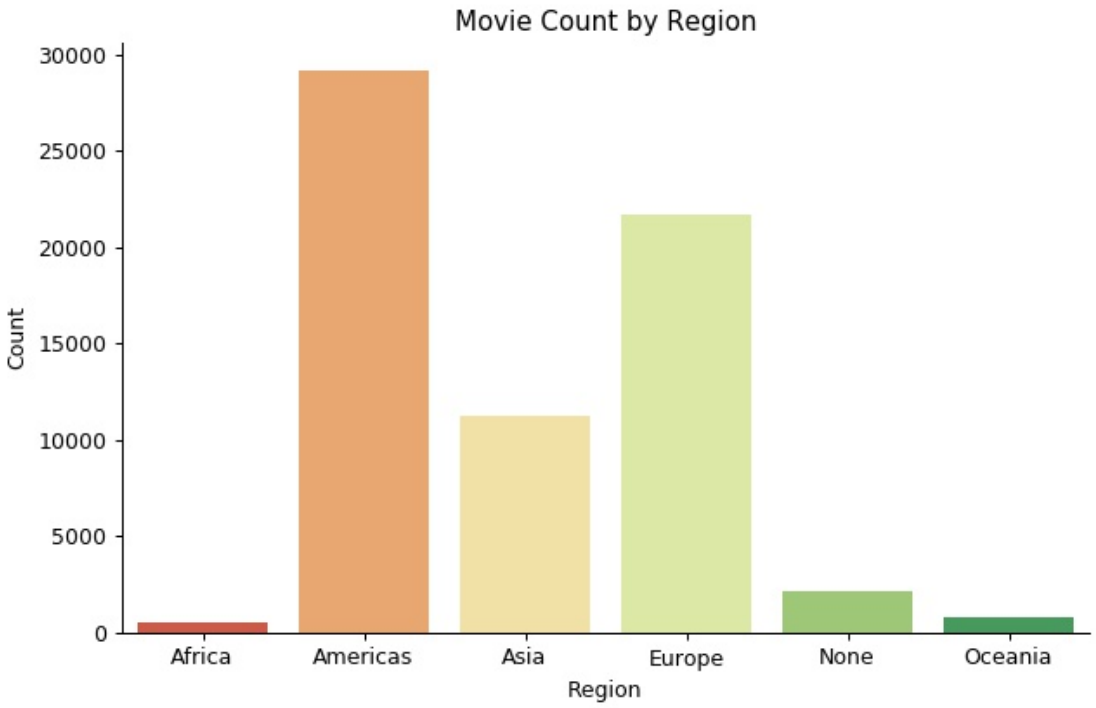


On the left is our dependent variable, weighted average votes. This highlights the normal structure of our dependent variable but I did note that it is slightly skewed towards votes above 5 ($\bar{x}$=5.96, σ=1.19). This gave us confidence going towards our model given that our dependent variable is normal. On the right is the duration distribution. Similarly this variable was normally distributed ($\bar{x}$=99.66, σ=22.71). Moving onto the monetary columns I again plotted them as histograms.



The relatively normal shape of these variables, worldwide income ($\bar{x}$=42461743, σ=155374066), USA income ($\bar{x}$=39540564, σ=103658832), and adjusted budget ($\bar{x}$=25411175, σ=39822572), was encouraging for their use in the model. I plotted these variables on a log scale because they were drastically skewed towards very large values. By scaling them in this way it brings out a normal structure.

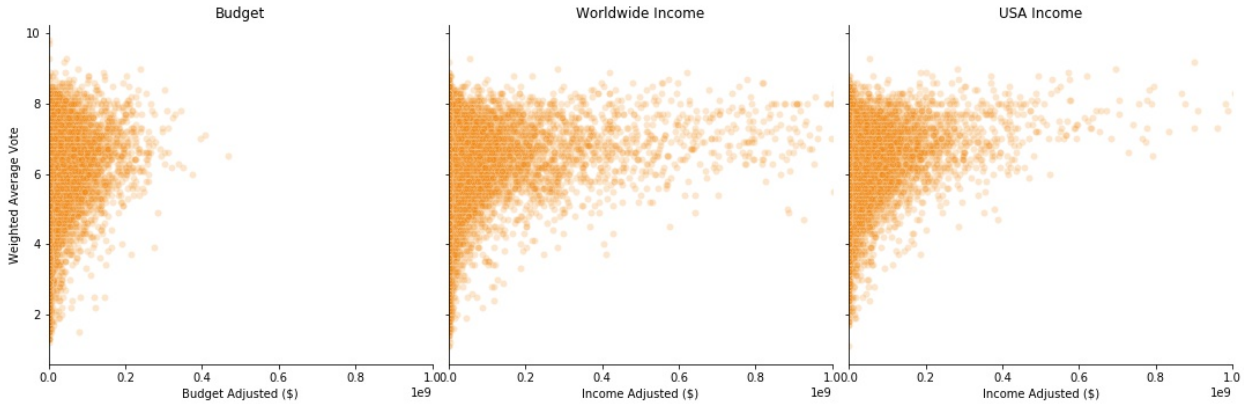
Next we can look at some of the categorical variables. First, I made a simple bar plot showing the counts of movies made in each region.



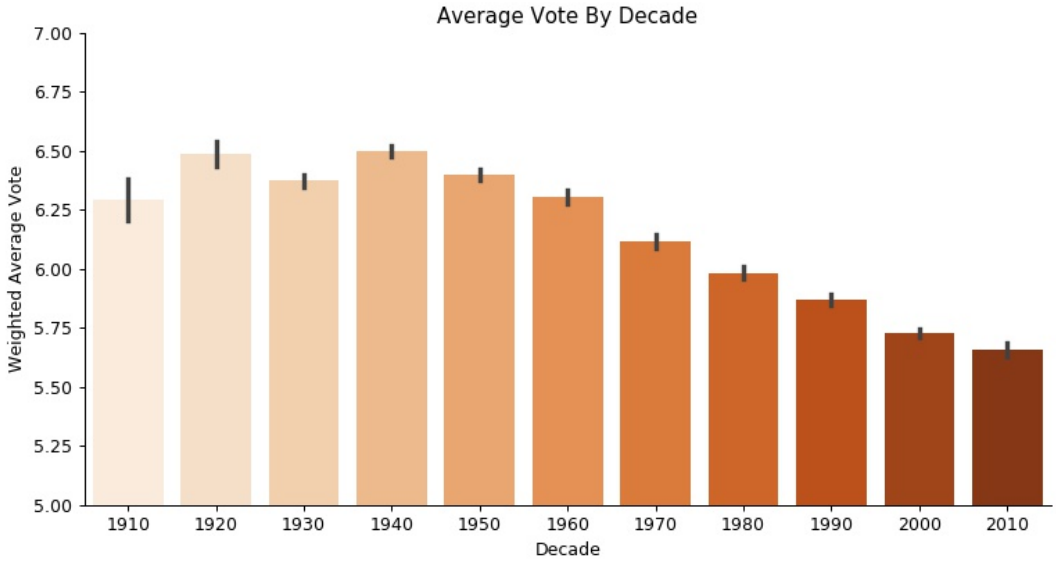
It's clear that the Americas, Asia, and Europe have created the vast majority of the movies in the data set. This can be explained by these regions having the largest GDPs of the world and therefore having the resources to finance the most movies.

Effects

Next I wanted to look at low level effects on our dependent variable. First I looked at the monetary effects.



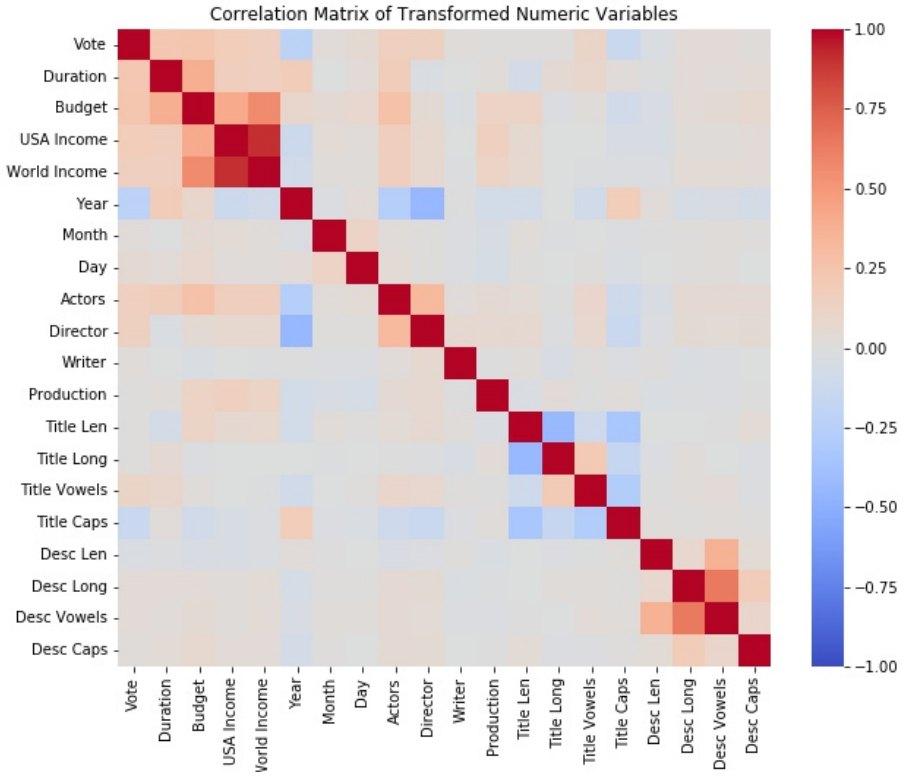
In this graph we can see that there is a slight positive trend between the average vote and the budget (pearson's r=0.23), worldwide income (pearson's r=0.16), and usa income (pearson's r=0.18). Next I looked the decade the movie was released and the votes.



This plot highlights a negative trend with the year the movie was released and the weighted average vote. This can potentially be explained by the volume of movies released increasing with time leaving the

possibility for more bad movies to bring down the average. In order to see if this was a significant finding I ran a one-way ANOVA on decade released and average vote. This confirmed that the decade did significantly affect the weighted average vote of a movie (p=0.0).

The final EDA plot that I created was a correlation matrix to see if any other effects jumped out.



This confirmed our findings from the previous plots that there is a slight positive correlation between the average vote and the monetary features, and a slight negative correlation with year. After this EDA we felt comfortable moving to modeling the weighted average vote of the movies.

Summary

The conclusion of my portion of the project is that by and large our features are trustworthy and were able to be cleaned in a relatively straight forward manner. Not only this, but we were able to use feature

engineering on many of our variables which will hopefully give the model more insight into movie ratings. I was also able to discover effects with the average vote against other features with no modeling. This made me very confident in our ability to predict the success of a movie given all of our features going into the modeling section. Overall, once the preprocessing and EDA sections were complete I was confident in passing the data onto the modeling section.

Code Percentage

I took 0% of my written code from the internet outside of syntax assistance for python packages which I have cited below. However, Josh did help with parts of the categorical variable encoding which I discussed in this report. For reference, I only included function names that I wrote entirely myself so any other functions in the code files I have mentioned were not written by me.

References

1. https://www.visualcapitalist.com/global-gdp-by-region-distribution-map/
2. https://help.imdb.com/article/imdb/track-movies-tv/weighted-average-ratings/GWT2DSBYVT2F25SK?ref_=helpsect_pro_2_8#
3. https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset
4. https://github.com/lukes/ISO-3166-Countries-with-Regional-Codes
5. https://fred.stlouisfed.org/series/CPIAUCNS
6. https://pandas.pydata.org/docs/
7. https://numpy.org/doc/1.21/
8. https://seaborn.pydata.org/
9. https://docs.scipy.org/doc/scipy/reference/
10. https://scikit-learn.org/stable/