A Tutorial on Data Science through Movies

> Collaborators: Archit Kambhamettu, Rohit Mukund, Saurabh Gadre, Vineeth Gohimukkula

Table of Contents:

Introduction

In an age where entertainment options are more abundant than ever, choosing the perfect movie to watch can be an impossible task. We've all been there: scrolling endlessly through Netflix and Hulu just to settle on a mediocre movie because you're just too overwhelmed by the options. In the 80s, there were only about a hundred movies released per year. Now, there's over 800 movies made per year in the US alone! Luckily, we've used our knowledge about data science and machine learning to help you make the perfect choice.

The purpose of this project is not just to create a movie recommender, it is mainly to walk you through the data science lifecycle. There are steps that most data scientists follow in a project, which we will The first is data collection, which consists of looking for data that could be interesting to analyze based on a central inquiry. Next comes cleaning and processing the data so that it is in a usable form for analysis. The processed data is then visualized for better understanding of trends and spread. Once the data is analyzed and visualized, it is modelled through ML. Finally, the analyzed data and the results of the model drive new in sights on the central question and policy decisions can be made with greater certainty.

We hope you enjoy this journey into the world of data science and find our results intriguing!

Data Collection/Data Cleaning

Imports for Data Collection

For this project, we imported many libraries that are instrumental in data science. These include pandas, numpy, and Beautiful Soup, which makes it easy to pull data out of HTML files.

In [54]:
```
import requests
import pandas as pd
import numpy as np
from bs4 import BeautifulSoup
import requests
import html5lib
```

Overview of Data Collection

Data collection is the first step in the data science life cycle. To gather the data needed for this project, we scoured the web to look for sources that would give us the most amount of data in a simple format. To start, we looked at the IMDB Top 250 movies list which gave us the title of the movie and the imdb rating in a tabular format. However, we recognized that we would need more information if we were to do real data analysis on an abundance of factors. Also, we felt that 250 entries in a table would simply not be enough data points to train a model.

Webscraping from Wikipedia

So, we needed a comprehensive list of movies from the past decade sorted by year. We decided to do this by webscraping from Wikipedia, since this list is arranged by month and year, as well as giving us additional information on Production Company. This list is messy when scraped, but proper data cleaning in combination with extraneous data from the two API's will generate a list of clean data that supports thorough analysis.

We start by using the requests library to make a GET request to Wikipedia's webpage for American movies by year. Each year's Wikipedia page is split into 5 different tables: 1) The top 10 movies of the year 2) Movies released in January through March 3) Movies released in April through June 4) Movies released in July through September 5) Movies released in October through December

We want to pull all of the table tags out of the HTML using BeautifulSoup. Since all we are seeking is a list of movies in the year, we can ignore the first table. Once we have these tables, we can concetenate them together, making a table for each year, and then do one final concentation to get our complete data set

In [55]:
```
dfs = []
# We can start by collecting one dataframe for each year, and then concatenati
for i in range(2011, 2022):
    # First we pull this specific year's table of off wikipedia.
    # Wikipedia splits its tables between the following
    # 1) A top hits table
    # 2 -> 5) January-March, April-June, July-September, October-December

    # We can use beautiful soup to grab each of these tables, and concatenate a
    r = requests.get(f'https://en.wikipedia.org/wiki/List_of_American_films_of_
    soup = BeautifulSoup(r.text, 'html.parser')
    find_table = soup.find_all('table')
    tables = pd.read_html(str(find_table))
    df = pd.concat([tables[2], tables[3], tables[4], tables[5]])
    # Now, set the year to be this index
    df['year'] = i
    dfs.append(df)
```

In [56]:
```
# We then concatenate all of these years together into one larger dataframe
data = pd.concat(dfs)
data
```

Out[56]:

	Opening	Opening.1	Title	Production company	Cast and crew	.mw-parser-output .tooltip-dotted{border-bottom:1px dotted;cursor:help}Ref.
0	JANUARY	5.0	If I Want to Whistle, I Whistle	Film Movement	Florin Șerban (director); George Piştereanu	[2]
1	JANUARY	5.0	Phil Ochs: There but for Fortune	First Run Features	Kenneth Bowser (director)	[3]
2	JANUARY	7.0	Season of the Witch	Rogue Pictures / Relativity Media	Dominic Sena (director); Bragi Schut (screenpl...	[4]
3	JANUARY	7.0	The Time That Remains	IFC Films	Elia Suleiman (director); Elia Suleiman, Saleh...	[5]
4	JANUARY	14.0	Barney's Version	Sony Pictures Classics	Richard J. Lewis (director); Michael Konyves (...	NaN
...	...	...	...	...	...	...
93	DECEMBER	25.0	The Tragedy of Macbeth	Apple TV+ / A24 / IAC Films	Joel Coen (director/screenplay); Denzel Washin...	NaN
94	DECEMBER	25.0	A Journal for Jordan	Columbia Pictures / Escape Artists / Bron Studios	Denzel Washington (director); Virgil Williams ...	NaN
95	DECEMBER	25.0	American Underdog	Lionsgate	Erwin brothers (directors); Jon Erwin, David A...	NaN
96	DECEMBER	26.0	Memoria	Neon	Apichatpong Weerasethakul (director/acreenplay...	NaN
97	DECEMBER	NaN	NaN	NaN	NaN	NaN

2813 rows × 9 columns

Pulling from the OMDB API

Because we want information on Movies such as Director, Genre, Plot Summary, Rating, and Box Office Score along with IMDB Rating that the Wikipedia page does not contain, we need to find a way to find this information. Luckily, the Internet Movie Database (IMDB) has copious amounts of information regarding movies, and we can access this information by accessing the Online Movie Database(OMDB).

By requesting a key from OMDB, we can use this key to access details from the Application Programming Interface, or API, which will allow us to gather much more information regarding the movies.

We start by developing a function, search_movie, that takes in a movie title and generates a JSON (JavaScript Object Notation) object from the API. This JSON will gives us valuable information on any movie.

In [57]:
```
def search_movie(search_string):
  # We start a request session in order to create a GET Request from this API
  sess = requests.Session()
  # Putting in the API Key, requested from OMDB
  api_key = '9418ad5d'
  # Building the url
  base_url = f'http://www.omdbapi.com/?apikey={api_key}&'
  search_url = f'&t={search_string}'
  # Now, pulling the movie information from the API using the request session
  resp = sess.get(base_url + search_url)
  data = resp.json()
  return data
```

For example, lets search up one of the most popular movies, "The Avengers" and get its information.

In [58]:
```
search_movie('The Avengers')
```

Out[58]:

```
{'Title': 'The Avengers',
 'Year': '2012',
 'Rated': 'PG-13',
 'Released': '04 May 2012',
 'Runtime': '143 min',
 'Genre': 'Action, Sci-Fi',
 'Director': 'Joss Whedon',
 'Writer': 'Joss Whedon, Zak Penn',
 'Actors': 'Robert Downey Jr., Chris Evans, Scarlett Johansson',
 'Plot': "Earth's mightiest heroes must come together and learn to fight as a
team if they are going to stop the mischievous Loki and his alien army from en
slaving humanity.",
 'Language': 'English, Russian',
 'Country': 'United States',
 'Awards': 'Nominated for 1 Oscar. 38 wins & 80 nominations total',
 'Poster': 'https://m.media-amazon.com/images/M/MV5BNDYxNjQyMjAtNTdiOS00NGYwLW
FmNTAtNThmYjU5ZGI2YTI1XkEyXkFqcGdeQXVyMTMxODk2OTU@._V1_SX300.jpg',
 'Ratings': [{'Source': 'Internet Movie Database', 'Value': '8.0/10'},
  {'Source': 'Rotten Tomatoes', 'Value': '91%'},
  {'Source': 'Metacritic', 'Value': '69/100'}],
 'Metascore': '69',
 'imdbRating': '8.0',
 'imdbVotes': '1,411,647',
 'imdbID': 'tt0848228',
 'Type': 'movie',
 'DVD': '25 Sep 2012',
 'BoxOffice': '$623,357,910',
 'Production': 'N/A',
 'Website': 'N/A',
 'Response': 'True'}
```


Creating our Appended DataFrame

Now, we want to create a new DataFrame with this added information. We define a function, append_frame, which takes in a dictionary of sought after columns and fills this dictionary with information from the API, creating a DataFrame out of this information.

The Categories that we want to analyze are as follows: 1) Genre 2) Actors 3) Director 4) Plot 5) Country 6) Language 7) IMDBRating 8) BoxOffice 9) IMDBVotes 10) Metascore 11) Rated (PG, R, PG-13, etc.)

We then make calls to the API for these particular categories, and for each Title in the current database, we add this new information about the title to a column if it exists in the API.

In [59]:
```
# We start with our list of desired columns columns
lst = {"Genre": [], "Actors":[], "Director":[], "Plot":[], "Country":[], "Langu
       "imdbVotes":[], "Metascore":[], "Rated": []}
def append_frame(lst, df):
    # Now for each title in the DataFrame, we want to make an API Request for e
    for title in df['Title']:
        d = search_movie(title)
        # Now, d is our JSON Object. We want to check if there was a response i
        # If there was, we can on to append the features.
        if d['Response'] == 'True':
            # Now for every feature, append the output from the JSON object if
```

```
            for feature in lst.keys():
                if feature in d.keys():
                    lst[feature].append(d[feature])
                else:
                    lst[feature].append(np.nan)
        else:
            # In this case, the title was not found in the dataframe, in which
            for feature in lst.keys():
                lst[feature].append(np.nan)

    # make the dictionary into a DataFrame, and return it
    dataframe = pd.DataFrame(lst)
    return dataframe

# Now, we call this method on our data
api_response = append_frame(lst, data)
```

In [60]:

```
api_response
```

Out[60]:

	Genre	Actors	Director	Plot	Country	Language	imdbRating
0	Drama	George Pistereanu, Ada Condeescu, Mihai Consta...	Florin Serban	Two weeks before his release, a teenage prison...	Romania, Sweden, Germany	Romanian	7.0
1	Documentary, Biography, History	Salvador Allende, Erik Andersen, Joan Baez	Kenneth Bowser	From civil rights to the anti-war movement to ...	United States	English	7.8
2	Action, Adventure, Fantasy	Nicolas Cage, Ron Perlman, Claire Foy	Dominic Sena	14th-century knights transport a suspected wit...	United States	English, Latin	5.4
3	Drama, History	Menashe Noy, Elia Suleiman, Baher Agbariya	Elia Suleiman	An examination of the creation of the state of...	France, Belgium, Italy, United Kingdom, United...	Arabic, Hebrew, English	7.0
4	Comedy, Drama	Paul Giamatti, Rosamund Pike, Jake Hoffman	Richard J. Lewis	The picaresque and touching story of the polit...	Italy, Canada	English, French	7.3
...	...	...	...	...	...	...	...
2808	Drama, Mystery, Thriller	Denzel Washington, Frances McDormand, Alex Has...	Joel Coen	A Scottish lord becomes convinced by a trio of...	United States	English, Persian	7.1
2809	Drama	Michael B. Jordan, Chanté Adams, Jalon Christian	Denzel Washington	1st Sgt. Charles Monroe King, before he is kil...	United States	English	5.9
2810	Biography, Drama, Sport	Zachary Levi, Anna Paquin, Hayden Zaller	Andrew Erwin, Jon Erwin	The story of NFL MVP and Hall of Fame quarterb...	United States	English	7.1
2811	Drama, Mystery, Sci-Fi	Tilda Swinton, Agnes Brekke, Daniel Giménez Cacho	Apichatpong Weerasethakul	A woman from Scotland, while traveling in Colo...	Colombia, Thailand, France, Germany, Mexico, Q...	English, Spanish	6.5

	Genre	Actors	Director	Plot	Country	Language	imdbRating
2812	Comedy	Paul Reid, Emer Hedderman, Rosalie Craig	Josie Rourke	Catherine Tate's iconic character Nan hits the...	United Kingdom	English	4.7

2813 rows × 11 columns

Now, we can combine the two previous dataframes by creating a title column for this DataFrame, which will match exactly with the title column for the previously scraped data from Wikipedia.

In [61]:
```
df = api_response
data = data.reset_index()
# setting the titles to match that of Wikipedia.
df['Title'] = data['Title']
df
```

Out[61]:

	Genre	Actors	Director	Plot	Country	Language	imdbRating
0	Drama	George Pistereanu, Ada Condeescu, Mihai Consta...	Florin Serban	Two weeks before his release, a teenage prison...	Romania, Sweden, Germany	Romanian	7.0
1	Documentary, Biography, History	Salvador Allende, Erik Andersen, Joan Baez	Kenneth Bowser	From civil rights to the anti-war movement to ...	United States	English	7.8
2	Action, Adventure, Fantasy	Nicolas Cage, Ron Perlman, Claire Foy	Dominic Sena	14th-century knights transport a suspected wit...	United States	English, Latin	5.4
3	Drama, History	Menashe Noy, Elia Suleiman, Baher Agbariya	Elia Suleiman	An examination of the creation of the state of...	France, Belgium, Italy, United Kingdom, United...	Arabic, Hebrew, English	7.0
4	Comedy, Drama	Paul Giamatti, Rosamund Pike, Jake Hoffman	Richard J. Lewis	The picaresque and touching story of the polit...	Italy, Canada	English, French	7.3
...	...	...	...	...	...	...	...
2808	Drama, Mystery, Thriller	Denzel Washington, Frances McDormand, Alex Has...	Joel Coen	A Scottish lord becomes convinced by a trio of...	United States	English, Persian	7.1
2809	Drama	Michael B. Jordan, Chanté Adams, Jalon Christian	Denzel Washington	1st Sgt. Charles Monroe King, before he is kil...	United States	English	5.9
2810	Biography, Drama, Sport	Zachary Levi, Anna Paquin, Hayden Zaller	Andrew Erwin, Jon Erwin	The story of NFL MVP and Hall of Fame quarterb...	United States	English	7.1
2811	Drama, Mystery, Sci-Fi	Tilda Swinton, Agnes Brekke, Daniel Giménez Cacho	Apichatpong Weerasethakul	A woman from Scotland, while traveling in Colo...	Colombia, Thailand, France, Germany, Mexico, Q...	English, Spanish	6.5

	Genre	Actors	Director	Plot	Country	Language	imdbRating
2812	Comedy	Paul Reid, Emer Hedderman, Rosalie Craig	Josie Rourke	Catherine Tate's iconic character Nan hits the...	United Kingdom	English	4.7

2813 rows × 12 columns

Data Cleaning: Organizing Relavent Information & Addressing Missing Data

Organizing Relavent Information

Since some information from the original Wikipedia scraped database is valuable, we want to maintain the storage of that information in the new DataFrame, but since some of it is messy, it needs to be organized a bit. First, we keep the Production Company data, which can remain untouched. Then, we want to add the date. In general, for our analysis, we need to maintain the year seperate, so we will avoid creating a datetime object. Instead, we will clean up the months formatting, along with renaming some of the columns to be more relavent to their purpose.

In [62]:
```
df['Production Company'] = data['Production company']
df['year'] = data['year']
# We want to remove whitespace from the Month,as well as making it lowercase.
df['month'] = data['Opening'].apply(lambda x: x.replace(" ", "").lower())
df['day'] = data['Opening.1']
# Reorganizing the dataframe so that Title is now the first column
df = df[['Title','Genre', 'Actors', 'Director', 'Plot', 'Country', 'Language',
       'imdbRating', 'BoxOffice', 'imdbVotes', 'Metascore', 'month', 'day', 'y
df
```

Out[62]:

	Title	Genre	Actors	Director	Plot	Country	Language	in
0	If I Want to Whistle, I Whistle	Drama	George Pistereanu, Ada Condeescu, Mihai Consta...	Florin Serban	Two weeks before his release, a teenage prison...	Romania, Sweden, Germany	Romanian	
1	Phil Ochs: There but for Fortune	Documentary, Biography, History	Salvador Allende, Erik Andersen, Joan Baez	Kenneth Bowser	From civil rights to the anti-war movement to ...	United States	English	
2	Season of the Witch	Action, Adventure, Fantasy	Nicolas Cage, Ron Perlman, Claire Foy	Dominic Sena	14th-century knights transport a suspected wit...	United States	English, Latin	
3	The Time That Remains	Drama, History	Menashe Noy, Elia Suleiman, Baher Agbariya	Elia Suleiman	An examination of the creation of the state of...	France, Belgium, Italy, United Kingdom, United...	Arabic, Hebrew, English	
4	Barney's Version	Comedy, Drama	Paul Giamatti, Rosamund Pike, Jake Hoffman	Richard J. Lewis	The picaresque and touching story of the polit...	Italy, Canada	English, French	
...	...	...	...	...	...	...	...	
2808	The Tragedy of Macbeth	Drama, Mystery, Thriller	Denzel Washington, Frances McDormand, Alex Has...	Joel Coen	A Scottish lord becomes convinced by a trio of...	United States	English, Persian	
2809	A Journal for Jordan	Drama	Michael B. Jordan, Chanté Adams, Jalon Christian	Denzel Washington	1st Sgt. Charles Monroe King, before he is kil...	United States	English	
2810	American Underdog	Biography, Drama, Sport	Zachary Levi, Anna Paquin, Hayden Zaller	Andrew Erwin, Jon Erwin	The story of NFL MVP and Hall of Fame quarterb...	United States	English	

	Title	Genre	Actors	Director	Plot	Country	Language	in
2811	Memoria	Drama, Mystery, Sci-Fi	Tilda Swinton, Agnes Brekke, Daniel Giménez Cacho	Apichatpong Weerasethakul	A woman from Scotland, while traveling in Colo...	Colombia, Thailand, France, Germany, Mexico, Q...	English, Spanish	
2812	NaN	Comedy	Paul Reid, Emer Hedderman, Rosalie Craig	Josie Rourke	Catherine Tate's iconic character Nan hits the...	United Kingdom	English	

2813 rows × 16 columns

Missing Data

Now, we want to address missing data. We will do this as follows: 1) Any row that has a title missing will be dropped. These rows are pointless to anaylze, since there is no use when we do not know which movie is being discussed 2) Any row with both imdbRating and BoxOffice missing are also dropped. Since these are two key points of analysis, we need not account for any row in our dataframe that has both of these missing.

This will allow our dataset to be prepped for analysis, given that missing data can affect certain methods of analysis.

In [63]:

```
total = df
total["Title"].isnull().sum()
# Finding all of those with Titles missing, or both imdbRating and BoxOffice
total.dropna(subset=['imdbRating', 'BoxOffice']).sum(numeric_only = True)
total = total.dropna(subset=['Title'])
total = total.dropna(subset=['imdbRating', 'BoxOffice'], how='all')
total
```

Out[63]:

	Title	Genre	Actors	Director	Plot	Country	Language	in
0	If I Want to Whistle, I Whistle	Drama	George Pistereanu, Ada Condeescu, Mihai Consta...	Florin Serban	Two weeks before his release, a teenage prison...	Romania, Sweden, Germany	Romanian	
1	Phil Ochs: There but for Fortune	Documentary, Biography, History	Salvador Allende, Erik Andersen, Joan Baez	Kenneth Bowser	From civil rights to the anti-war movement to ...	United States	English	
2	Season of the Witch	Action, Adventure, Fantasy	Nicolas Cage, Ron Perlman, Claire Foy	Dominic Sena	14th-century knights transport a suspected wit...	United States	English, Latin	
3	The Time That Remains	Drama, History	Menashe Noy, Elia Suleiman, Baher Agbariya	Elia Suleiman	An examination of the creation of the state of...	France, Belgium, Italy, United Kingdom, United...	Arabic, Hebrew, English	
4	Barney's Version	Comedy, Drama	Paul Giamatti, Rosamund Pike, Jake Hoffman	Richard J. Lewis	The picaresque and touching story of the polit...	Italy, Canada	English, French	
...	...	...	...	...	...	...	...	
2807	The King's Man	Action, Adventure, Thriller	Ralph Fiennes, Gemma Arterton, Rhys Ifans	Matthew Vaughn	In the early years of the 20th century, the Ki...	United Kingdom, United States	English, Latin, German, French, Russian	
2808	The Tragedy of Macbeth	Drama, Mystery, Thriller	Denzel Washington, Frances McDormand, Alex Has...	Joel Coen	A Scottish lord becomes convinced by a trio of...	United States	English, Persian	
2809	A Journal for Jordan	Drama	Michael B. Jordan, Chanté Adams, Jalon Christian	Denzel Washington	1st Sgt. Charles Monroe King, before he is kil...	United States	English	
2810	American Underdog	Biography, Drama, Sport	Zachary Levi, Anna Paquin, Hayden Zaller	Andrew Erwin, Jon Erwin	The story of NFL MVP and Hall of Fame quarterb...	United States	English	

	Title	Genre	Actors	Director	Plot	Country	Language	in
2811	Memoria	Drama, Mystery, Sci-Fi	Tilda Swinton, Agnes Brekke, Daniel Giménez Cacho	Apichatpong Weerasethakul	A woman from Scotland, while traveling in Colo...	Colombia, Thailand, France, Germany, Mexico, Q...	English, Spanish	

2759 rows × 16 columns

Using the TMDB API

Despite the copious amount of information that we gained from the OMDB API, we are missing two key pieces of information in analyzing movies: Budget and Keywords. With Budget, we can understand how much investment went into a movie. This is vital in seeing if a particular movie lived up to its expectation. Keywords or Tags help us generate some sort of inside into the details in the movie. While genre tells us what sort of movie it is, keywords provide specific information into the subgenre. For example, the Avengers, while classified as action, is specfically a Superhero movie in the Marvel Universe, and such words would be tagged as keywords. This will help us in deciding which movie to recommend to a person.

We can get this information from another databases' API, The Movie DataBase (TMDB). This API has plenty more information on movies, but we will focus on Budget and Keywords as mentioned before. To this point, we will develop functions get_budget and get_keywords that take in a movie's title and generate the Budget and Keywords from the API. Similar to the other API, we start a equest session, form the URL, and use an API Key given by TMDB in order to retrieve the information.

In [64]:
```
def get_budget(title):
    # starting the session in order to get info from the API
    title = title.replace(" ", "+")
    sess = requests.Session()
    # We then form our url with our title, api key and base url
    api_key = '58ea292f930b11d66214fcb7a1bae448'
    base_url = f'https://api.themoviedb.org/3/search/movie?api_key={api_key}&qu
    resp = sess.get(base_url)
    # retrieving data as a json, but since this json does not contain the necce
    # take the id from the json and use it in a different request(if results ca
    # append np.nan
    datas = resp.json()
    if(len(datas['results']) < 1):
        return np.nan
    # use the id as mentioned to get the budget from the JSON in a similar requ
    eyedee = datas['results'][0]['id']
    url = f'https://api.themoviedb.org/3/movie/{eyedee}?api_key=58ea292f930b11d
    r2 = sess.get(url)
    data2 = r2.json()
    return data2['budget']
```

To test, we'll find out what the budget was for "The Avengers"

In [65]:
```
get_budget("The Avengers")
```

Out[65]:
```
220000000
```

Now, we wish to generate the keywords instead of the budget and test it on "The Avengers"

In [66]:
```
# Almost identical to Budget, but instead we wish to generate the keywords ins
def get_keywords(title):
    title = title.replace(" ", "+")
    # starting the session in order to get info from the API
    sess = requests.Session()
    # We then form our url with our title, api key and base url

    api_key = '58ea292f930b11d66214fcb7a1bae448'
    base_url = f'https://api.themoviedb.org/3/search/movie?api_key={api_key}&qu
    resp = sess.get(base_url)

    # retrieving data as a json, but since this json does not contain the necce
    # take the id from the json and use it in a different request(if results ca
    # append np.nan
    datas = resp.json()
    if(len(datas['results']) < 1):
        return np.nan

    # use the id as mentioned to get the budget from the JSON in a similar requ
    eyedee = datas['results'][0]['id']
    url = f'https://api.themoviedb.org/3/movie/{eyedee}/keywords?api_key=58ea29
    r2 = sess.get(url)
    data2 = r2.json()
    return data2['keywords']

get_keywords("The Avengers")
```

Out[66]:
```
[{'id': 242, 'name': 'new york city'},
 {'id': 5539, 'name': 'shield'},
 {'id': 9715, 'name': 'superhero'},
 {'id': 9717, 'name': 'based on comic'},
 {'id': 14909, 'name': 'alien invasion'},
 {'id': 155030, 'name': 'superhero team'},
 {'id': 179430, 'name': 'aftercreditsstinger'},
 {'id': 179431, 'name': 'duringcreditsstinger'},
 {'id': 180547, 'name': 'marvel cinematic universe (mcu)'}]
```

We now apply this to dataframe's title column, generating the budgets and keywords into a Series.

In [67]:
```
info = total['Title'].apply(lambda x: (get_budget(x), get_keywords(x)))
info
```

Out[67]:
```
0                                                  (0, [])
1                                                  (0, [])
2        (40000000, [{'id': 344, 'name': 'inquisition'}...
3            (6500000, [{'id': 537, 'name': 'palestine'}])
4        (30000000, [{'id': 236, 'name': 'suicide'}, {'...
                               ...
2807     (100000000, [{'id': 212, 'name': 'london, engl...
2808     (0, [{'id': 388, 'name': 'scotland'}, {'id': 1...
2809                                               (0, [])
2810     (0, [{'id': 579, 'name': 'american football'},...
2811     (0, [{'id': 155800, 'name': 'atmospheric'}, {'...
Name: Title, Length: 2759, dtype: object
```

Then, we can unpack this series into a dataframe, and then index the dataframe to create our new columns!

In [68]:
```
tmdb_response = info.apply(pd.Series)
total['Budget'] = tmdb_response[0]
total['Keywords'] = tmdb_response[1]
total
```

Out[68]:

	Title	Genre	Actors	Director	Plot	Country	Language	in
0	If I Want to Whistle, I Whistle	Drama	George Pistereanu, Ada Condeescu, Mihai Consta...	Florin Serban	Two weeks before his release, a teenage prison...	Romania, Sweden, Germany	Romanian	
1	Phil Ochs: There but for Fortune	Documentary, Biography, History	Salvador Allende, Erik Andersen, Joan Baez	Kenneth Bowser	From civil rights to the anti-war movement to ...	United States	English	
2	Season of the Witch	Action, Adventure, Fantasy	Nicolas Cage, Ron Perlman, Claire Foy	Dominic Sena	14th-century knights transport a suspected wit...	United States	English, Latin	
3	The Time That Remains	Drama, History	Menashe Noy, Elia Suleiman, Baher Agbariya	Elia Suleiman	An examination of the creation of the state of...	France, Belgium, Italy, United Kingdom, United...	Arabic, Hebrew, English	
4	Barney's Version	Comedy, Drama	Paul Giamatti, Rosamund Pike, Jake Hoffman	Richard J. Lewis	The picaresque and touching story of the polit...	Italy, Canada	English, French	
...	...	...	...	...	...	...	...	
2807	The King's Man	Action, Adventure, Thriller	Ralph Fiennes, Gemma Arterton, Rhys Ifans	Matthew Vaughn	In the early years of the 20th century, the Ki...	United Kingdom, United States	English, Latin, German, French, Russian	
2808	The Tragedy of Macbeth	Drama, Mystery, Thriller	Denzel Washington, Frances McDormand, Alex Has...	Joel Coen	A Scottish lord becomes convinced by a trio of...	United States	English, Persian	
2809	A Journal for Jordan	Drama	Michael B. Jordan, Chanté Adams, Jalon Christian	Denzel Washington	1st Sgt. Charles Monroe King, before he is kil...	United States	English	
2810	American Underdog	Biography, Drama, Sport	Zachary Levi, Anna Paquin, Hayden Zaller	Andrew Erwin, Jon Erwin	The story of NFL MVP and Hall of Fame quarterb...	United States	English	

	Title	Genre	Actors	Director	Plot	Country	Language	in
2811	Memoria	Drama, Mystery, Sci-Fi	Tilda Swinton, Agnes Brekke, Daniel Giménez Cacho	Apichatpong Weerasethakul	A woman from Scotland, while traveling in Colo...	Colombia, Thailand, France, Germany, Mexico, Q...	English, Spanish	

2759 rows × 18 columns

We notice that keywords is formatted awkwardly, so to clean this up a bit, we can unpack the dictionary that is returned so that we have a list of the key words.

In [69]:
```
total['Keywords'] = total['Keywords'].apply(lambda x: np.nan if type(x) is flo
total
```

Out[69]:

	Title	Genre	Actors	Director	Plot	Country	Language	in
0	If I Want to Whistle, I Whistle	Drama	George Pistereanu, Ada Condeescu, Mihai Consta...	Florin Serban	Two weeks before his release, a teenage prison...	Romania, Sweden, Germany	Romanian	
1	Phil Ochs: There but for Fortune	Documentary, Biography, History	Salvador Allende, Erik Andersen, Joan Baez	Kenneth Bowser	From civil rights to the anti-war movement to ...	United States	English	
2	Season of the Witch	Action, Adventure, Fantasy	Nicolas Cage, Ron Perlman, Claire Foy	Dominic Sena	14th-century knights transport a suspected wit...	United States	English, Latin	
3	The Time That Remains	Drama, History	Menashe Noy, Elia Suleiman, Baher Agbariya	Elia Suleiman	An examination of the creation of the state of...	France, Belgium, Italy, United Kingdom, United...	Arabic, Hebrew, English	
4	Barney's Version	Comedy, Drama	Paul Giamatti, Rosamund Pike, Jake Hoffman	Richard J. Lewis	The picaresque and touching story of the polit...	Italy, Canada	English, French	
...	...	...	...	...	...	...	...	
2807	The King's Man	Action, Adventure, Thriller	Ralph Fiennes, Gemma Arterton, Rhys Ifans	Matthew Vaughn	In the early years of the 20th century, the Ki...	United Kingdom, United States	English, Latin, German, French, Russian	
2808	The Tragedy of Macbeth	Drama, Mystery, Thriller	Denzel Washington, Frances McDormand, Alex Has...	Joel Coen	A Scottish lord becomes convinced by a trio of...	United States	English, Persian	
2809	A Journal for Jordan	Drama	Michael B. Jordan, Chanté Adams, Jalon Christian	Denzel Washington	1st Sgt. Charles Monroe King, before he is kil...	United States	English	
2810	American Underdog	Biography, Drama, Sport	Zachary Levi, Anna Paquin, Hayden Zaller	Andrew Erwin, Jon Erwin	The story of NFL MVP and Hall of Fame quarterb...	United States	English	

	Title	Genre	Actors	Director	Plot	Country	Language	in
2811	Memoria	Drama, Mystery, Sci-Fi	Tilda Swinton, Agnes Brekke, Daniel Giménez Cacho	Apichatpong Weerasethakul	A woman from Scotland, while traveling in Colo...	Colombia, Thailand, France, Germany, Mexico, Q...	English, Spanish	

2759 rows × 18 columns

Exploratory Data Analysis and Visualization

We are now at the stage where we have processed our data into a usable form. Of course, there will be more processing throughout the rest of this project because we need to tailor the information based on the specific needs of the visualization or the model. In the first part of this analysis and visualization we will take a look at a bunch of different factors and explore the data in our table.

For the data we use in the project, we made a .csv file to store the data because the api calls are constantly changing due to the number of people that interact with IMDB, changing the numbers. As you can see, the tables are identical.

In [779…

```
apiData = total.copy()
apiData
```

Out[779]:

	Title	Genre	Actors	Director	Plot	Country	Language	i
0	If I Want to Whistle, I Whistle	Drama	George Pistereanu, Ada Condeescu, Mihai Consta...	Florin Serban	Two weeks before his release, a teenage prison...	Romania, Sweden, Germany	Romanian	
1	Phil Ochs: There but for Fortune	Documentary, Biography, History	Salvador Allende, Erik Andersen, Joan Baez	Kenneth Bowser	From civil rights to the anti-war movement to ...	United States	English	
2	Season of the Witch	Action, Adventure, Fantasy	Nicolas Cage, Ron Perlman, Claire Foy	Dominic Sena	14th-century knights transport a suspected wit...	United States	English, Latin	
3	The Time That Remains	Drama, History	Menashe Noy, Elia Suleiman, Baher Agbariya	Elia Suleiman	An examination of the creation of the state of...	France, Belgium, Italy, United Kingdom, United...	Arabic, Hebrew, English	
4	Barney's Version	Comedy, Drama	Paul Giamatti, Rosamund Pike, Jake Hoffman	Richard J. Lewis	The picaresque and touching story of the polit...	Italy, Canada	English, French	
...	...	...	...	...	...	...	...	
2807	The King's Man	Action, Adventure, Thriller	Ralph Fiennes, Gemma Arterton, Rhys Ifans	Matthew Vaughn	In the early years of the 20th century, the Ki...	United Kingdom, United States	English, Latin, German, French, Russian	
2808	The Tragedy of Macbeth	Drama, Mystery, Thriller	Denzel Washington, Frances McDormand, Alex Has...	Joel Coen	A Scottish lord becomes convinced by a trio of...	United States	English, Persian	
2809	A Journal for Jordan	Drama	Michael B. Jordan, Chanté Adams, Jalon Christian	Denzel Washington	1st Sgt. Charles Monroe King, before he is kil...	United States	English	
2810	American Underdog	Biography, Drama, Sport	Zachary Levi, Anna Paquin, Hayden Zaller	Andrew Erwin, Jon Erwin	The story of NFL MVP and Hall of Fame quarterb...	United States	English	

	Title	Genre	Actors	Director	Plot	Country	Language	i
2811	Memoria	Drama, Mystery, Sci-Fi	Tilda Swinton, Agnes Brekke, Daniel Giménez Cacho	Apichatpong Weerasethakul	A woman from Scotland, while traveling in Colo...	Colombia, Thailand, France, Germany, Mexico, Q...	English, Spanish	

2759 rows × 18 columns

Processing Box Office and Month Columns

In this raw data from the API, the BoxOffice and imdbVotes columns are represented as strings. Additionally, the BoxOffice columns are prepended with a $ sign and have commas indicating thousands of dollars. We remove these symbols and convert both these columns into floats, as we will be applying mathematical functions to them later. Additionally, the months are listed in word format (ie January, Feburary, ..) rather than in numerical format (ie 1,2,3...). We convert these strings to numbers using the datetime import, mapping January → 1, Feburary → 2, and so on.

Creating Profit and Hit Column

As stated above, we define a hit movie as (BOR - BoF) > mean(BOR-BoF). To make the "hit" column of the dataframe, we subtract the budget from the box office to calculate the profit and classify a movie as a "hit" (giving it a score of 1) if it is higher than the average profit of all movies, and a "flop" (giving it a score of 0) if it is lower.

In [780…

```
data = pd.read_csv("roshart.csv")
# data.columns
# data["Title"].isnull().sum()
# data.dropna(subset=['imdbRating', 'BoxOffice']).sum()
# data = data.dropna(subset=['Title'])
# data = data.replace('N/A', np.nan)
# data = data.dropna(subset=['imdbRating', 'BoxOffice'], how='all')
# data.dropna(subset=['Genre', 'Plot'])
# data2 = data.copy()
data.columns
data["Title"].isnull().sum()
data.dropna(subset=['imdbRating', 'BoxOffice']).sum()
data = data.dropna(subset=['Title'])
data = data.dropna(subset=['imdbRating', 'BoxOffice'], how='all')
data.dropna(subset=['Genre', 'Plot'])
data2 = data.copy()
data
```

```
C:\Users\saura\AppData\Local\Temp\ipykernel_20980\2856899449.py:12: FutureWarn
ing: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=
None') is deprecated; in a future version this will raise TypeError.  Select o
nly valid columns before calling the reduction.
  data.dropna(subset=['imdbRating', 'BoxOffice']).sum()
```

Out[780]:

	Unnamed: 0	index	Title	Genre	Actors	Director	Plot	Cou
0	0	0	If I Want to Whistle, I Whistle	Drama	George Pistereanu, Ada Condeescu, Mihai Consta...	Florin Serban	Two weeks before his release, a teenage prison...	Rom Sw Ger
1	1	1	Phil Ochs: There but for Fortune	Documentary, Biography, History	Salvador Allende, Erik Andersen, Joan Baez	Kenneth Bowser	From civil rights to the anti-war movement to ...	U S
2	2	2	Season of the Witch	Action, Adventure, Fantasy	Nicolas Cage, Ron Perlman, Claire Foy	Dominic Sena	14th-century knights transport a suspected wit...	U S
3	3	3	The Time That Remains	Drama, History	Menashe Noy, Elia Suleiman, Baher Agbariya	Elia Suleiman	An examination of the creation of the state of...	Fr Bel U King Uni
4	4	4	Barney's Version	Comedy, Drama	Paul Giamatti, Rosamund Pike, Jake Hoffman	Richard J. Lewis	The picaresque and touching story of the polit...	Ca
...	...	...	...	...	...	...	...	
2746	2746	2807	The King's Man	Action, Adventure, Thriller	Ralph Fiennes, Gemma Arterton, Rhys Ifans	Matthew Vaughn	In the early years of the 20th century, the Ki...	U King U S
2747	2747	2808	The Tragedy of Macbeth	Drama, Mystery, Thriller	Denzel Washington, Frances McDormand, Alex Has...	Joel Coen	A Scottish lord becomes convinced by a trio of...	U S
2748	2748	2809	A Journal for Jordan	Drama	Michael B. Jordan, Chanté Adams, Jalon Christian	Denzel Washington	1st Sgt. Charles Monroe King, before he is kil...	U S
2749	2749	2810	American Underdog	Biography, Drama, Sport	Zachary Levi, Anna Paquin, Hayden Zaller	Andrew Erwin, Jon Erwin	The story of NFL MVP and Hall of Fame quarterb...	U S

	Unnamed: 0	index	Title	Genre	Actors	Director	Plot	Cou
2750	2750	2811	Memoria	Drama, Mystery, Sci-Fi	Tilda Swinton, Agnes Brekke, Daniel Giménez Cacho	Apichatpong Weerasethakul	A woman from Scotland, while traveling in Colo...	Colo Tha Fr Gerr Me

2751 rows × 20 columns

In [781…

```
from datetime import datetime
#data["Title"].isnull()
data = data.dropna(subset = ["Rated"])
data = data[data["BoxOffice"].apply(lambda x: type(x) == str)]
data = data[data["Budget"] != 0.0]

data = data.sort_values("imdbRating", ascending = False)
data["BoxOffice"] = data["BoxOffice"].apply(lambda x: x.split("$")[1])
data["BoxOffice"] = data["BoxOffice"].apply(lambda x: x.replace(",",""))
data["imdbVotes"] = data["imdbVotes"].apply(lambda x: x.replace(",",""))
#data = data.drop('Unnamed: 0', axis = 1)

data = data.astype({"BoxOffice":'float', "imdbVotes":'float', "imdbRating":'fl
data['month'] = data['month'].apply(lambda x: datetime.strptime(x, '%B').month

data["hit"] = data.apply(lambda row : 1 if row["imdbRating"] > 6.5 and row["im
```

Initial Visualizations

First, it would be interesting to see the box office of movies at different months throughout the year. It is generally known that the summer months and the november-december are considered the best time to release movies as the audience is the largest during this time.

In [782…

```
month = data[['month', 'BoxOffice']]
month
```

Out[782]:

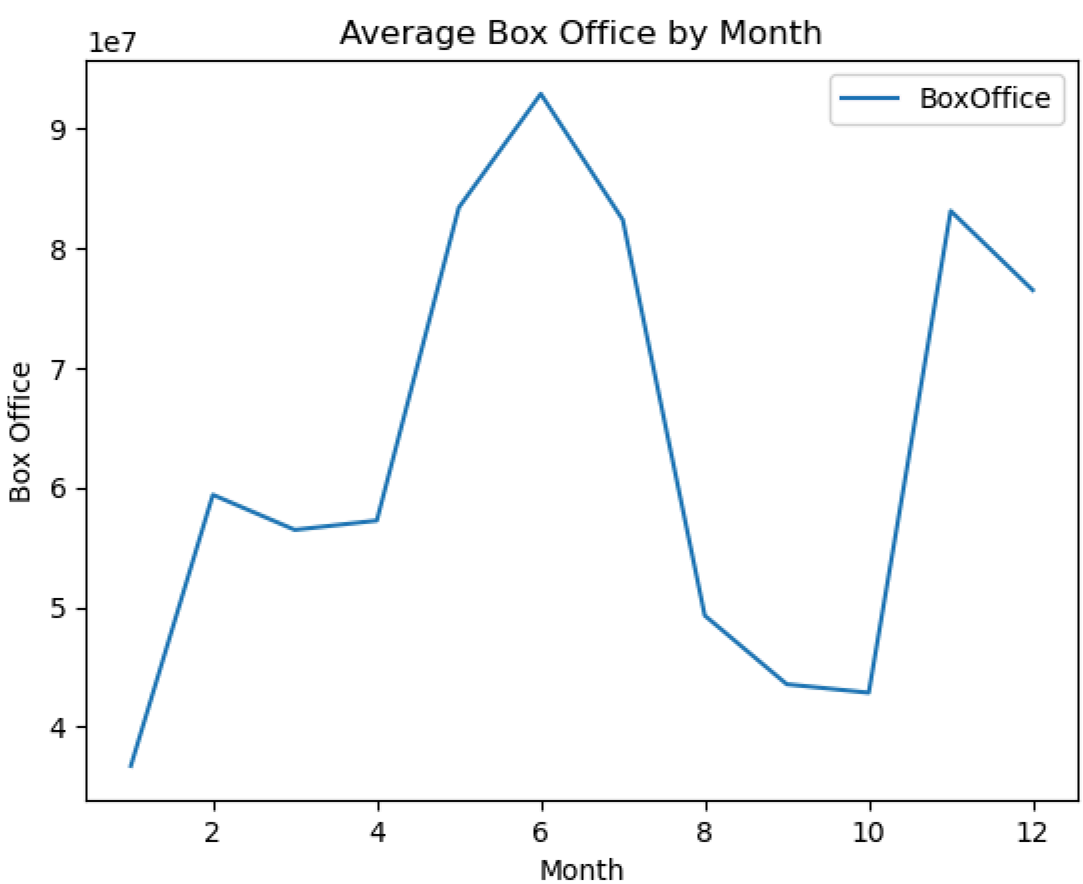
	month	BoxOffice
944	11	188020017.0
928	10	13092000.0
2005	7	422783777.0
1867	12	190241310.0
730	11	707481.0
...	...	...
178	11	74158157.0
1776	8	30569484.0
921	10	14019924.0
2118	12	27166770.0
22	2	73013910.0

1509 rows × 2 columns

In [783…

```
m = month.groupby(['month']).mean().plot()
m.set_title('Average Box Office by Month')
m.set_xlabel('Month')
m.set_ylabel('Box Office')
```

Out[783]: Text(0, 0.5, 'Box Office')



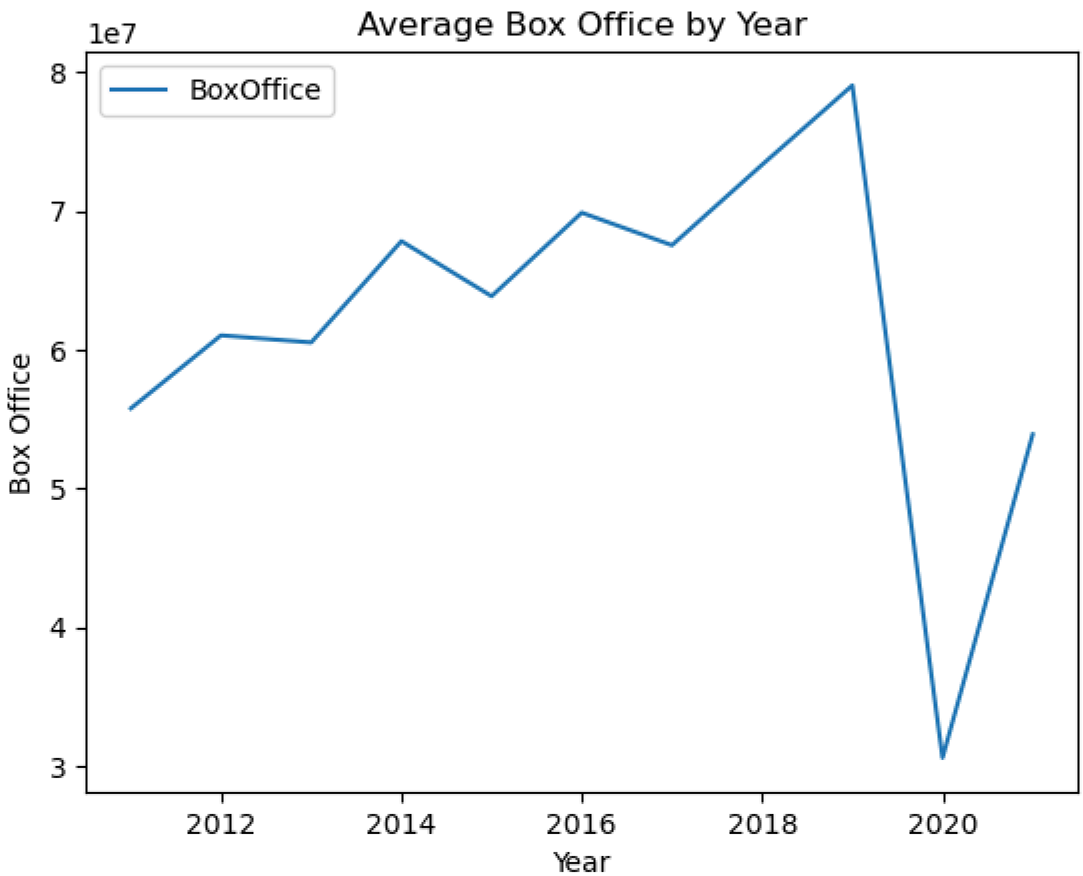
As we can see, we got the expected results. The summer months and the end of the year are the best months for box office. We now have one component that contributes to the success of a movie. Now, lets take a look at the year and how it relates to box office success.

In [784…

```
yearData = data[['year', 'BoxOffice']]
y = yearData.groupby(['year']).mean().plot()
y.set_title('Average Box Office by Year')
y.set_xlabel('Year')
y.set_ylabel('Box Office')
```

Out[784]:

```
Text(0, 0.5, 'Box Office')
```



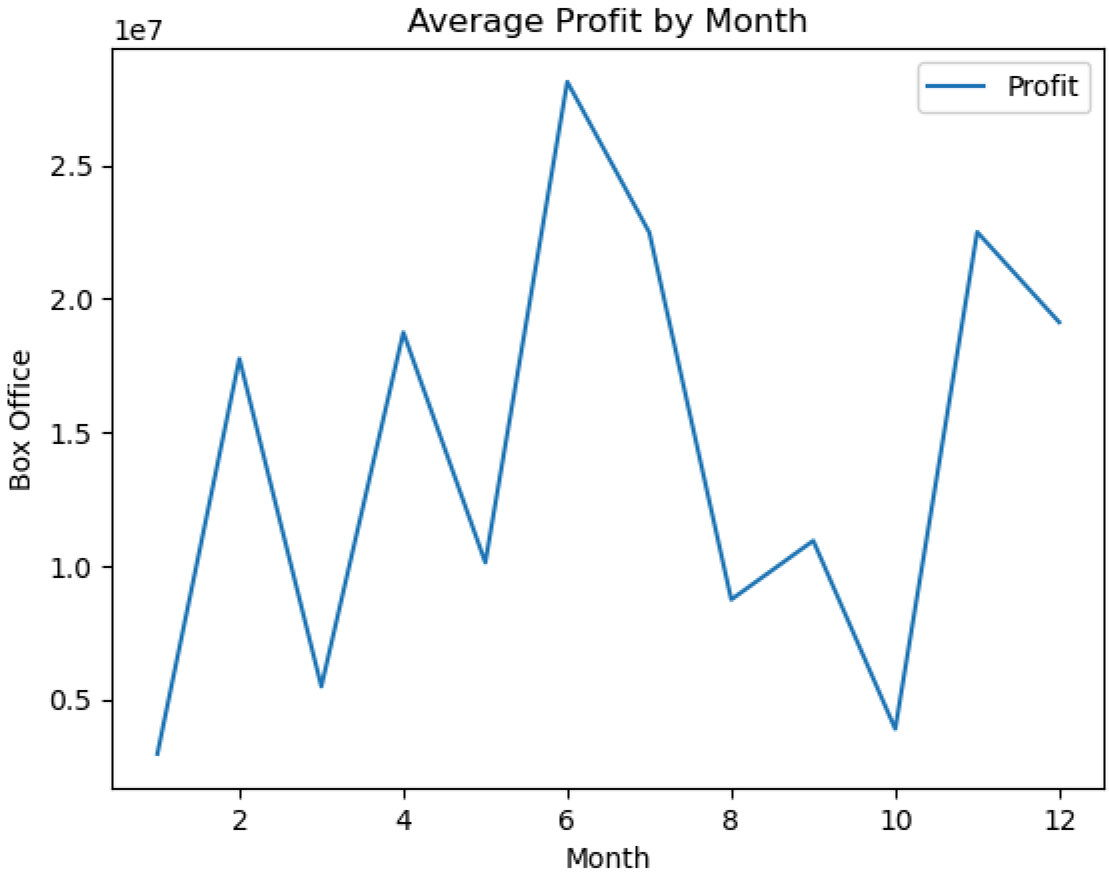
This graph is not too surprising, but there is clearly an effect of the pandemic on movies where box office earnings have plummeted. Other than this, box office earnings have gone up every year. Now, lets look at the total profit and how it changes based on month.

In [785…

```
data['Profit'] = data['BoxOffice'] - data['Budget']
monthlyProfit = data[['month', 'Profit']]
monthlyProfit = monthlyProfit.groupby(['month']).mean().plot()
monthlyProfit.set_title('Average Profit by Month')
monthlyProfit.set_xlabel('Month')
monthlyProfit.set_ylabel('Box Office')
```

Out[785]:

```
Text(0, 0.5, 'Box Office')
```

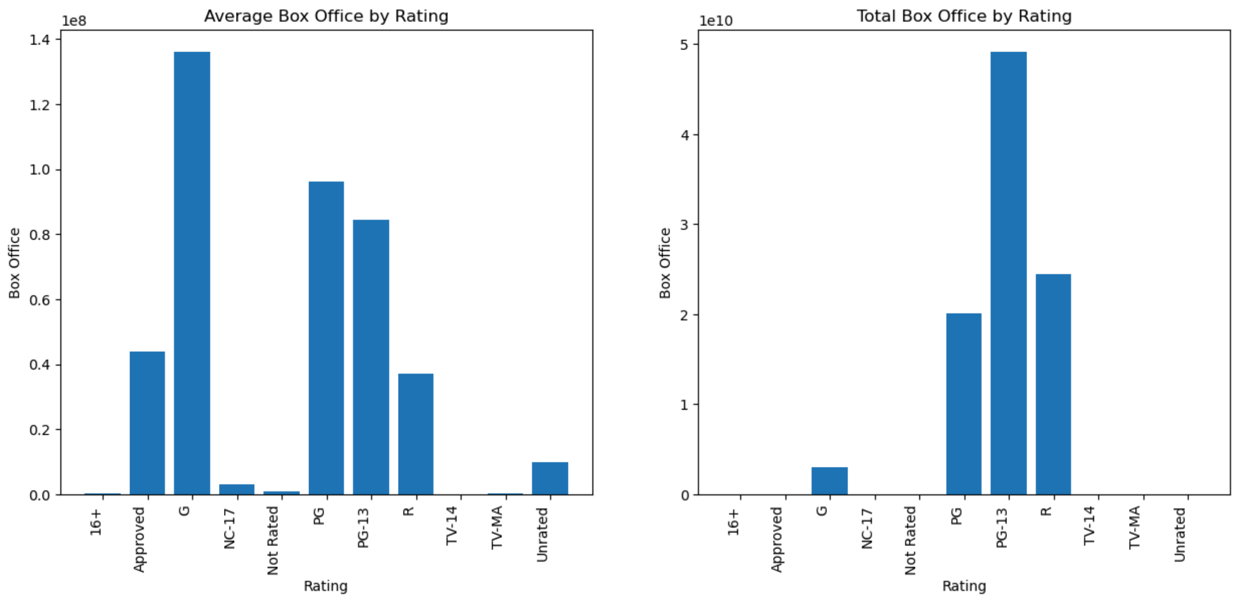


It seems like the profit is dependent on month as well, as this graph's peaks are similar to the box office graph. A month feature would be interesting to look at to calculate whether a movie will be a hit. Another thing to analyze could be the relation between the box office and the aggregate number of imdb votes on the movie.

In [786…

```
plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
rateData = data[['Rated', 'BoxOffice']]
y = rateData.groupby(['Rated']).mean()

plt.bar(y.index, y['BoxOffice'])
plt.title('Average Box Office by Rating')
plt.xlabel('Rating')
plt.ylabel('Box Office')
plt.xticks(rotation = 90, ha = 'right')

plt.subplot(1, 2, 2)
z = rateData.groupby(['Rated']).sum()

plt.bar(z.index, z['BoxOffice'])
plt.title('Total Box Office by Rating')
plt.xlabel('Rating')
plt.ylabel('Box Office')
plt.xticks(rotation = 90, ha = 'right')
```

```
Out[786]: ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
           [Text(0, 0, ''),
            Text(0, 0, ''),
            Text(0, 0, ''),
            Text(0, 0, ''),
            Text(0, 0, ''),
            Text(0, 0, ''),
            Text(0, 0, ''),
            Text(0, 0, ''),
            Text(0, 0, ''),
            Text(0, 0, ''),
            Text(0, 0, '')])
```



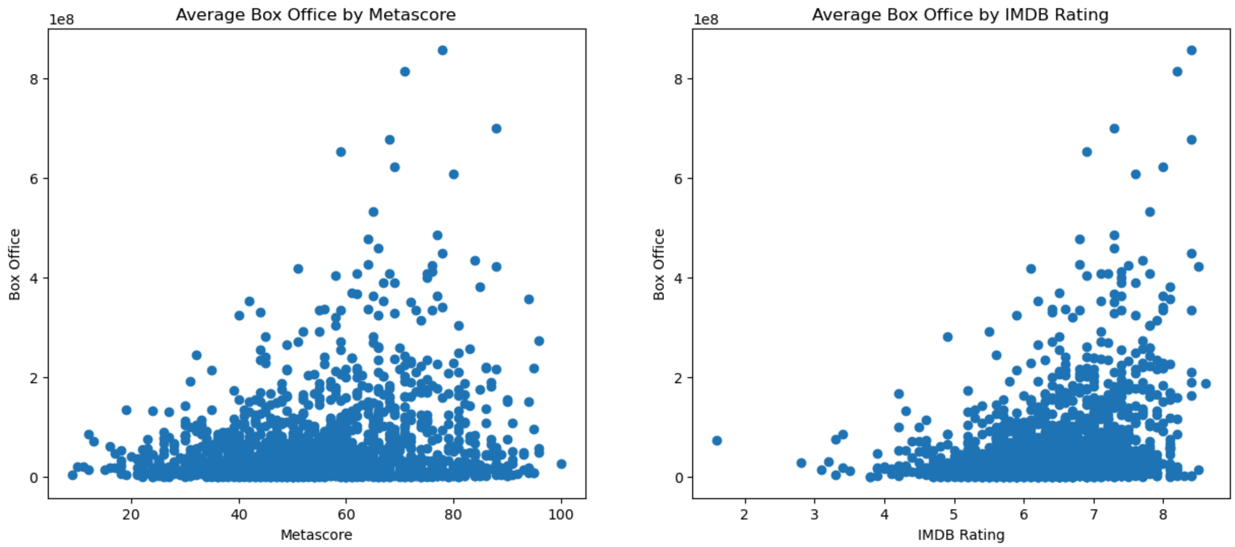
Here is a comparison of two plots. The first one is the average box office of rating. The "G" rating produces the highest box office, followed by the "PG" rating, and then the "PG-13" rating. However, the total box office for G movies is not even comparable to that of PG-13. Nevertheless, we have acquired vital information about another factor in determining whether a movie is a hit or not: rating.

Lastly, we will look at ratings and votes by fans and their relation to the box office. First, I'll define a couple of terms. A metascore is a weighted average of many reviews coming from reputed critics. It is scored from 0 - 100. The [IMDB Rating] is an aggregation of user reviews, which is then averaged. IMDB votes are the sum of the amount of votes that users cast for a particular movie.

```
In [787…  plt.figure(figsize=(15, 6))

          #subplots
          plt.subplot(1, 2, 1)
          plt.scatter(data['Metascore'], data['BoxOffice'])
          plt.title('Average Box Office by Metascore')
          plt.xlabel('Metascore')
          plt.ylabel('Box Office')


          plt.subplot(1, 2, 2)
          plt.scatter(data['imdbRating'], data['BoxOffice'])
```

```
plt.title('Average Box Office by IMDB Rating')
plt.xlabel('IMDB Rating')
plt.ylabel('Box Office')
```

Out[787]: Text(0, 0.5, 'Box Office')



Although there is not a perfect correlation between these ratings and box office, we can point out some observations from these graphs. A movie with a large box office is generally on the higher side of the imdbRating scale. However, this does not seem to be the case with the metascore, where there are movies that have yielded a box office of more than $60,000,000 that have a mediocre metascore rating. All in all, we can take a look at all of these factors and try to come up with a way to determine whether a movie is a hit or not.

Model: Analysis, Hypothesis Testing, & ML

In this section, we will use some of the insights we gained from our visuals to make some predictions using machine learning. During this section, more visuals will be made to gain even further information about the data.

Do you ever hear about the highest grossing box office movies, but wonder whether the audience genuinley enjoyed it? Or whether movie critics are actually reliable sources in determining public perception of movies? How about whether the month a movie releases has an impact on whether people like that movie or not?

Using the features listed above and some others detailed below, we attempt to classify whether a movie is a hit or not. While there is no univeral definition for what classifies a hit film, we define it to be an imdbRating of over 6.5, which is significantly above average, with over 500 imdbVotes. This removes films that might have only a few ratings, which would skew the data.

Because we are interested in applying a label as an output, we decided to use a classification model rather than a regression model, which is generally used for predicting a quantity. More information on the difference between these two types of models can be found here.

In [788…

```
import math

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score


from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from scipy.stats import randint
```

Processing Data through One-Hot Encoding

Before we explain one-hot encoding, it's important to recognize what categorical data is. In statistics, a categorical variable is a variable that can take on a number of possible values, each assigned to a specific group. In our project, a categorical variable could the genre or rating(PG-13, R, etc.) of the movie. The issue with this kind of data is that many machine learning algorithms cannot operate on labeled data directly; inputs and outputs must be numeric. This is where one-hot encoding is useful. It allows us to represent categorical variables as numerical values in a model. The categorical parameters will prepare separate columns for each individual label. For example, whenever there is a comedy movie, the value will be 1 in the new "Comedy" column and 0s for all the other columns. Here is a link to more information on one-hot encoding: What is One Hot Encoding?

For our data, there are problems beyond just categorical data that require processing. The genre labeling that is returned by the API lists some of the genres as a combination of multiple genres. There are many entries that look like "comedy, action." Movies with this genre must be listed in the "comedy" and "action" columns as 1s in the processed table. Categorical variable In statistics, a categorical variable (also called qualitative variable) is a variable that can take on one of a limited, and usually fixed, number of possible values, assigning each individual or other unit of observation to a particular group or nominal category on the basis of some qualitative property. In computer science and some branches o... What is One Hot Encoding? Why and When Do You Have to Use it? | Hac... One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. box office and month columns:

In [789…

```
all_generes = set()
for i in data["Genre"].unique():
    genre = i.split(",")
    for g in genre:
        b = g.replace(" ", "")
        all_generes.add(b)

for genre in all_generes:
     data[genre] = data["Genre"].apply(lambda x: 1 if genre in x else 0)

all_ratings = set()
for i in data["Rated"].unique():
    all_ratings.add(i)

for rat in all_ratings:
     data[rat] = data["Rated"].apply(lambda x: 1 if rat == x else 0)

all_ratings = list(all_ratings)
all_generes = list(all_generes)


lst = all_ratings+ all_generes  +[ "Budget","BoxOffice", "month", "Metascore"]


data[lst].head(5)
```

Out[789]:

	G	PG	TV-MA	Approved	Unrated	R	NC-17	PG-13	Not Rated	TV-14	...	Family	Biography	Wes
944	0	0	0	0	0	0	0	1	0	0	...	0	0	
928	0	0	0	0	0	1	0	0	0	0	...	0	0	
2005	1	0	0	0	0	0	0	0	0	0	...	0	0	
1867	0	1	0	0	0	0	0	0	0	0	...	0	0	
730	0	0	0	0	0	1	0	0	0	0	...	0	0	

5 rows × 36 columns

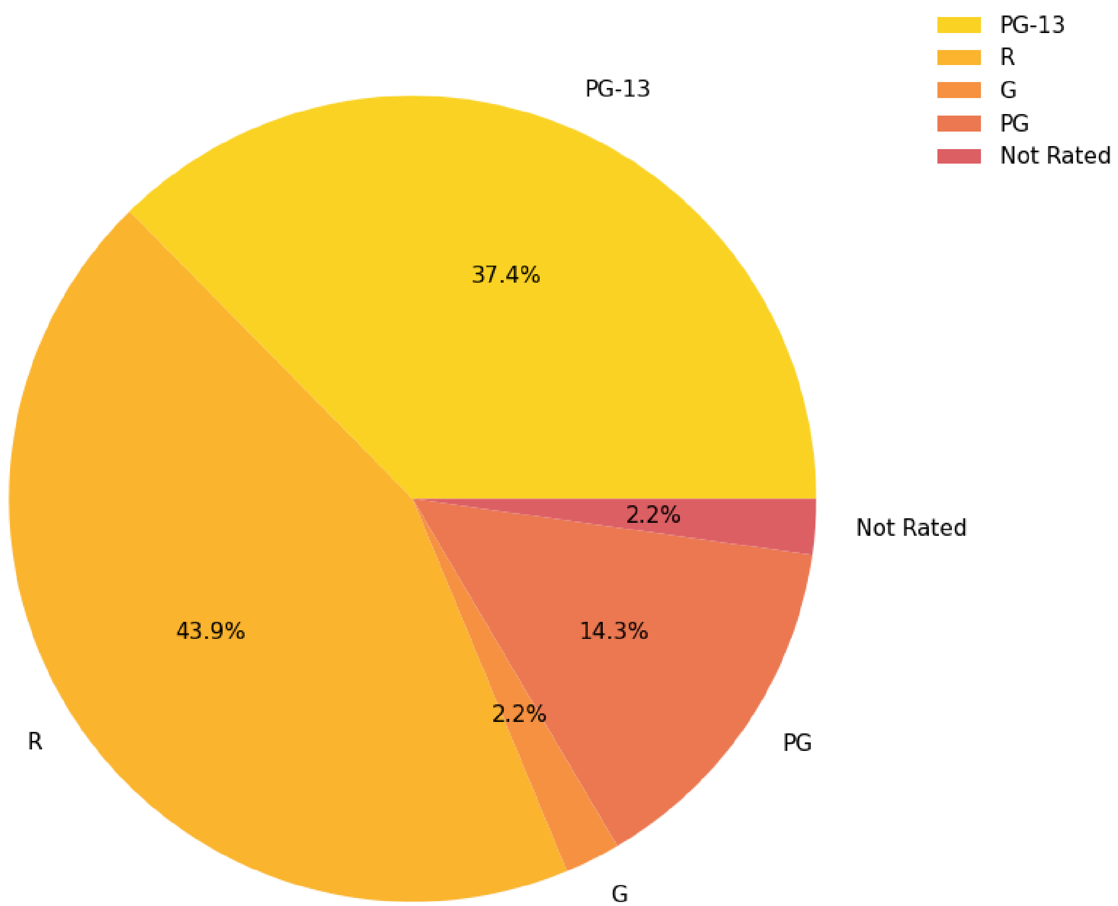
Preliminary Exploration of Data for the Model

We first begin with extrapolating the different features we will put into the model. From the data we explored above, we decided to explore this classification using Metascore, Budget, Box Office, month, genre, and rating (R, PG-13...) as features. While our dataframe contains other information such as actors, production company, and plot, we realize that it is simply impractical to include this information in our model as there are far too many actors and production companies that have been involved in the provided movies. We instead use this information in the movie recommender!

First, we visualize which Ratings and which Genres have the most hits with the two pie charts below. We see that rated Rmovies have the highest percentage of hits, followed by PG-13. Additionally, we notice that Drama movies and Comedy movies have the highest percentage of hits, closely followed by Action and Adventure movies.

In [790…

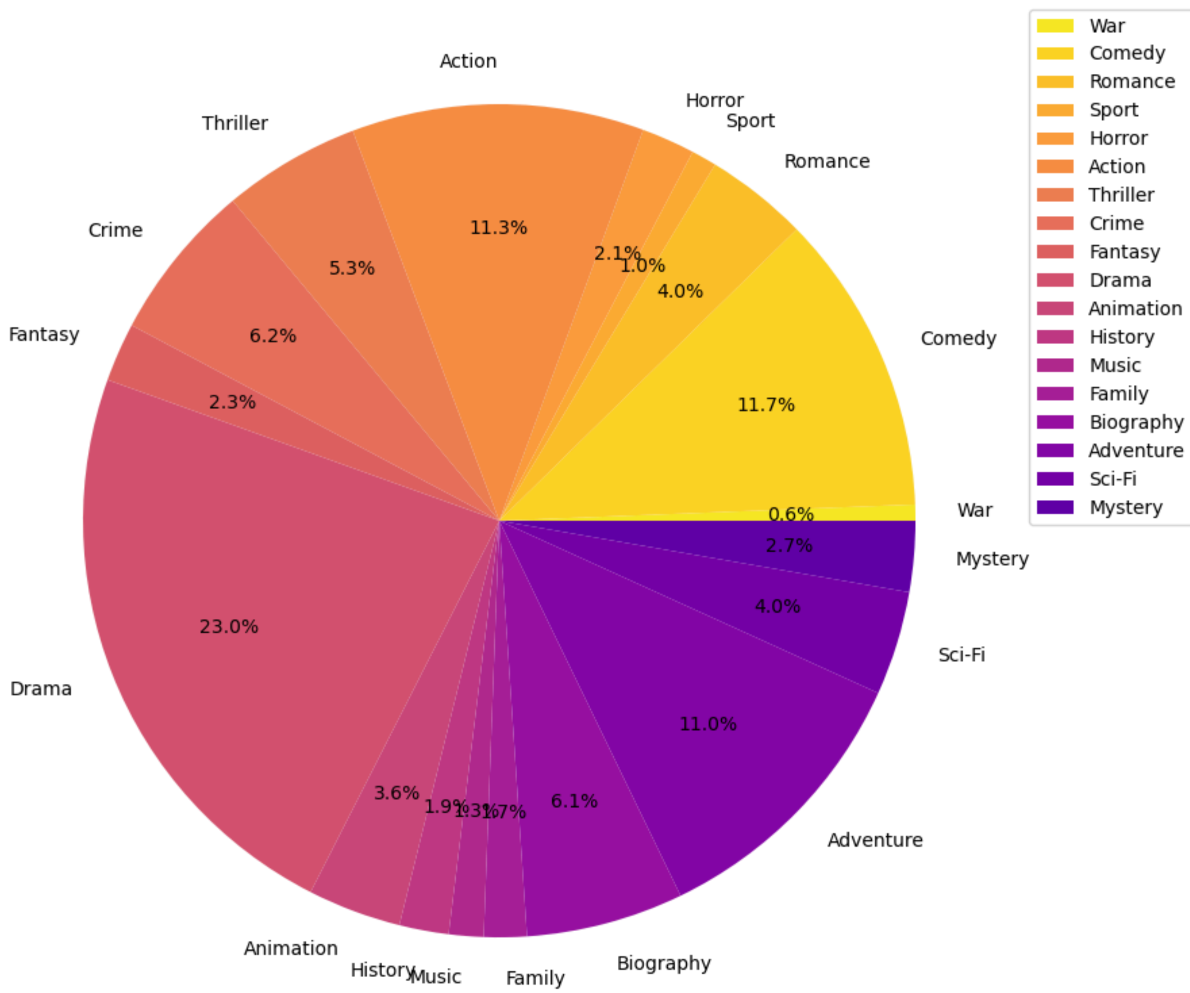
```
h = {}
hits = len(data["hit"])
ratings = data["Rated"].unique()
for i in ratings:
    x = len(data.loc[(data[i] == 1) & (data["hit"] == 1)])
    if x > 2:
        h[i] = x/hits
h
df = pd.DataFrame(h.items(), columns = ["Ratings", "Percent"])

pal_ = list(sns.color_palette(palette='plasma_r',
                              n_colors=len(ratings)).as_hex())
#plot a pie chart
plt.figure(figsize=(9, 9))
plt.rcParams.update({'font.size': 11})
plt.pie(df.Percent,
        labels= df.Ratings,
        colors=pal_, autopct='%1.1f%%',
        pctdistance=.6)
plt.legend(bbox_to_anchor=(1, 1), loc=2, frameon=False)
plt.show()
```



In [791…

```
h = {}
hits = len(data["hit"])
ratings = all_generes
for i in ratings:
```

```
    x = len(data.loc[(data[i] == 1) & (data["hit"] == 1)])
    if x > 10:
        h[i] = x/hits
h
df = pd.DataFrame(h.items(), columns = ["Genre", "Percent"])

pal_ = list(sns.color_palette(palette='plasma_r',
                              n_colors=len(ratings)).as_hex())
#plot a pie chart
plt.figure(figsize=(10, 10))
plt.rcParams.update({'font.size': 10})
plt.pie(df.Percent,
        labels= df.Genre,
        colors=pal_, autopct='%1.1f%%',
        pctdistance=.7)
plt.legend(bbox_to_anchor=(1, 1), loc=2, frameon=True)
plt.show()
```



Feature Importance

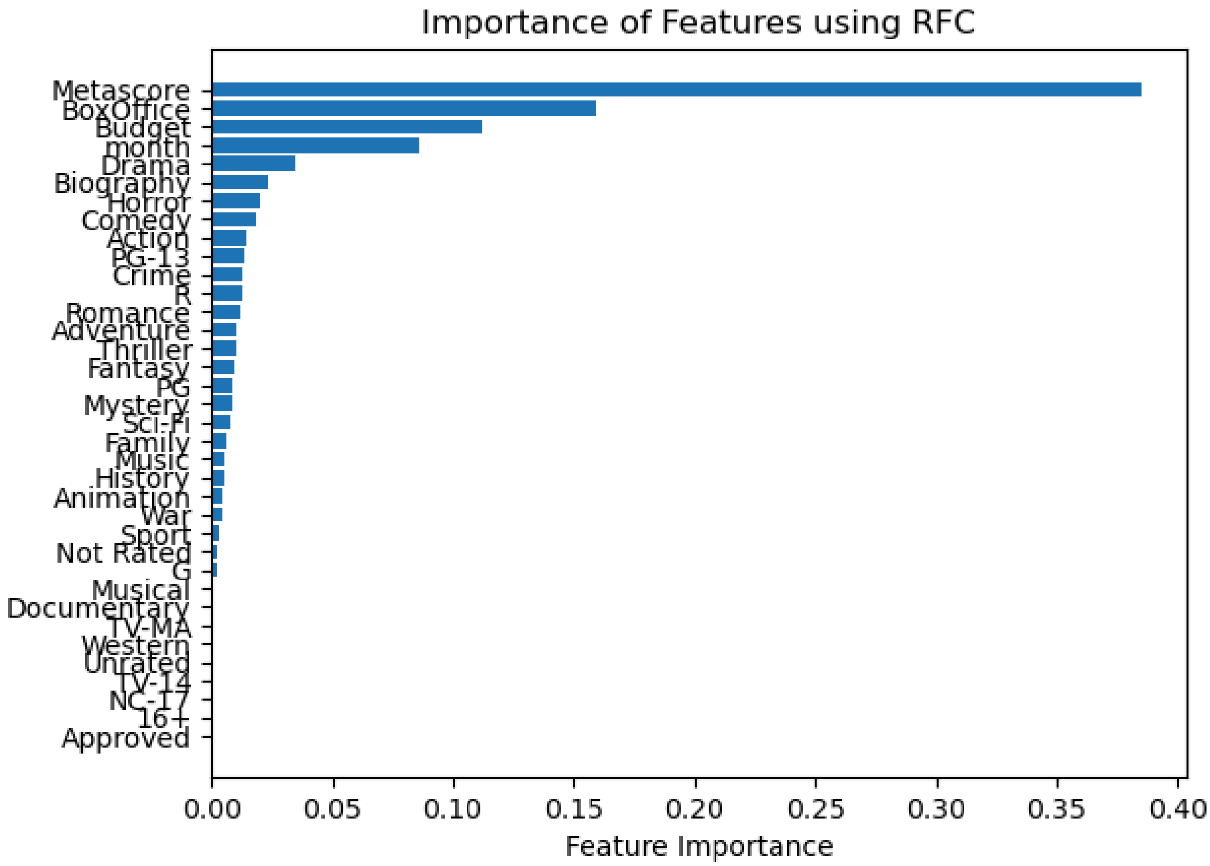
We further analyze just how impactful each genre and each rating is by determining how important each feature is to having a hit movie. We do this by running a RandomForestClassifier, one of the most popular feature importance techniques.

The algorithm involves constructing decision trees for each sample and generating output based off this tree. The final output is a majority vote on whether the trees resulted in a 1 or 0, in this case a hit or not. To learn more about RandomForestClassification and its use on finding feature importance, please refer to this resource here. The feature with the largest importance is Metascore, followed by Box Office, Budget, and month.

The bar graphs below show the importance of each feature we use in our classification. Surprisingly, the movie pundits might be onto something, as the Metascore has the highest importance out of all the features we analyze. The second and third highest importance are the BoxOffice and Budget respectivly. This also aligns with what we expected, as when referencing popular movies, we tend to reference movies that are BoxOffice hits and have large production budgets. Additionally, as shown in the bar plot in the Visualization section, the month has a realtivley high feature importance on whether a movie is a hit or a flop. From there, we noticed that each genre and rating had a reduced feature importance in comparision to the aforementioned four features. This is because each genre and rating is considered its own feature, rather than collectivley determining the feature importance for genres and ratings. The importance of genres as a whole and ratings is shown in the second bar plot, which displays that genres actually have the second highest feature importance out of the given features.

In [792…

```
tot = lst + ["hit"]
data = data.dropna(subset=tot)

X = data[lst]
X.columns = X.columns.astype(str)

y = data["hit"]
rf = RandomForestClassifier(n_estimators = 150)
rf.fit(X, y)
sort = rf.feature_importances_.argsort()
plt.barh(X.columns[sort], rf.feature_importances_[sort])
plt.title("Importance of Features using RFC")
plt.xlabel("Feature Importance")


X.columns[sort]
rf.feature_importances_[sort]
all_features = {}
for i in range(len(X.columns[sort])):
    all_features[X.columns[sort][i]] = rf.feature_importances_[sort][i]
```

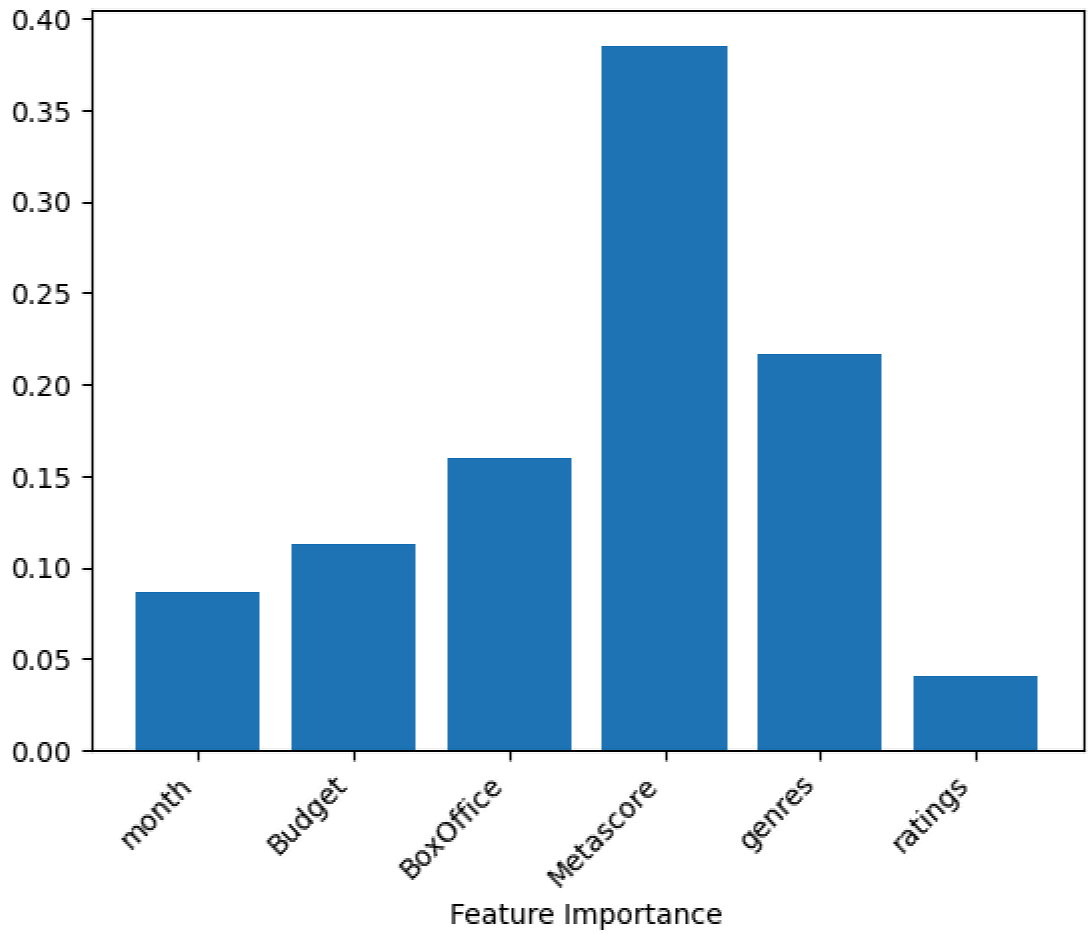


In [793…

```
all_features["genres"] = 0
for key in all_features.keys():
    if key in all_generes:
        all_features["genres"] += float(all_features[key])

for i in all_generes:
    del all_features[i]
all_features

all_features["ratings"] = 0
for key in all_features.keys():
    if key in all_ratings:
        all_features["ratings"] += float(all_features[key])

for i in all_ratings:
    del all_features[i]
all_features


plt.bar(range(len(all_features)), list(all_features.values()))
plt.xticks(range(len(all_features)), list(all_features.keys()))
plt.xticks(rotation = 45, ha = 'right')

plt.xlabel("Feature Importance")
```

Out[793]:

```
Text(0.5, 0, 'Feature Importance')
```



Now that we confirmed that the features we hypothesized would have an impact on movie success, we run 3 different classification algorithms to determine whether we can correctly predict whether a movie would be a hit or not. We first split 80% of our data into a training set, and allocate the remaining 20 as the testing set. To determine which of the classification algorithms perform the best, we compare the accuracy score, which takes the total number of correctly predicted data points and divides it by the total number of data points. Another metric we use in determining the validity of the models is the f1 score, which takes the weighted average of precision and recall. It has more of a focus on false negatives and false positives. We additionally display the confusion matrix which displays the number of correct predictions, the number of Type 1 error predictions, and the number Type 2 error predictions. The formulas for both metrics and a description of the confusion matrix are detailed below.

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

In [794…

```
#Run for all data
lst = all_ratings+ all_generes  +[ "Budget","BoxOffice", "month", "Metascore"]
X = data[lst]
X.columns = X.columns.astype(str)
y = data["hit"]
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state = 0, test_
```

K Nearest Neighbors

The first model we run is a K Nearest Neighbor Classifier (KNN). This classifier computes the euclidean distance between the test point and all the other data points and compares the K nearest neighbors to see whether to classify the test point as a 1 or 0. More information on the KNN model can be found here. After trial and error and reading reputable sources We take K to be the sqrt of the number of data points we have.

Our accuracy score for the KNN model is ~70.80% which for a real-world data set is very good. According to the data science community an accuracy score of above ~70% is considered good. However, the f1 score, ~67.89%, is realtively low for the KNN model, and the confusion matrix shows clarity as to why. The model has a higher amount of false positives, perhaps because there are more non-hit movies than hit movies.

In [795…

```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
```

```
X_test = sc_X.transform(X_test)
n = (int(math.sqrt(len(y_test)) - 1))
classifier = KNeighborsClassifier(n_neighbors = n, p = 2, metric = 'euclidean'
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print("Accuracy Score: ", accuracy_score(y_test, y_pred))
print("F1 Score: ", f1_score(y_test, y_pred))
print("Confusion Matrix: \n", cm)
```

```
Accuracy Score:  0.7080536912751678
F1 Score:  0.6789667896678967
Confusion Matrix:
 [[119  35]
 [ 52  92]]
```

```
C:\Users\saura\anaconda3\lib\site-packages\sklearn\neighbors\_classification.p
y:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis
`), the default behavior of `mode` typically preserves the axis it acts along.
In SciPy 1.11.0, this behavior will change: the default value of `keepdims` wi
ll become False, the `axis` over which the statistic is taken will be eliminat
ed, and the value None will no longer be accepted. Set `keepdims` to True or F
alse to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

The second model we run is a Support Vector Machine Classification (SVC). This classifier finds a hyper plane in an n-dimensional (where n is the number of features) plane that distincly classifies the data points. While there could be multiple hyperplanes, the SCV model chooses the hyperplane that maximizes the distance ebtween the data points of both classes. For more information on the SCV model, refer here.

Our accuracy score for the SVC model is ~78.85% which is better much better than our KNN classifier. Additionally, the f1 score is also very high for the SCV model, which is even higher than the accuracy score. This means the model has a higher weighted average of precision and recall. As shown by the confusion matrix, the model only has ~39 False Negatives and ~24 False Positives.

In [796…

```
svm_model = SVC()
svm_model = svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print("Accuracy Score: ", accuracy_score(y_test, y_pred))
print("F1 Score: ", f1_score(y_test, y_pred))
print("Confusion Matrix: \n", cm)
```

```
Accuracy Score:  0.7885906040268457
F1 Score:  0.7920792079207922
Confusion Matrix:
 [[115  39]
 [ 24 120]]
```

Random Forest Classifier

The third classification model we run is a Random Forest Classifier. The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

The accuracy score for the Random Forest Classifier is an ~81.67%, which is a higher accuracy score than the aforementioned two models. Because the accuracy score can be so varied for Random Forest, we average the accuracy and f1 scores over 15 different runs to find the mean scores. This is a better representation of the actual scores of the model, rather than simply running the model once, as Random Forest can have different scores for different runs due to the randomness of the model.

In [797…

```
a = 0
f1 = 0
n = 15
for i in range(n):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    rf = RandomForestClassifier()
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1 +  f1_score(y_test, y_pred)
    a = a + accuracy
print("Accuracy: ", a/n)
print("F1 Score: ", f1/n)
```

```
Accuracy:  0.812080536912752
F1 Score:  0.7999897536117881
```

Logistic Regression Classification

The fourth and final model we run is a Logistic Regression Classification. Logistic Regression, despite its name, is a classification model. In its simplest form, it models a binary output. Examples given by Saishruthi Swaminathan of TowardsDataScience are, whether an email is spam (0) or not (1), or whether a tumor is malignant (0) or not (1). Logistic Regression is a transformation of linear regression that uses the sigmoid function , a function that lies strictly between 0 and 1. The output of Logistic Regression is the probability of each binary output.

Our accuracy score for the Logistic Regression Classification is ~56.71%, which is moderately high for our data. The f1 score is similar, at about 57.7%. This shows that the Logistic Regression Classification is the worst classifier out of all the ones we tested, as it has the lowest accuracy score and the lowest f1 score. Additionally, the confusion matrix shows that it misclassifed ~89 as False Positives and ~40 as False Negatives.

In [798…

```
lg_model = LogisticRegression()
lg_model = lg_model.fit(X_train, y_train)
y_pred = lg_model.predict(X_test)
accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print("Accuracy Score: ", accuracy_score(y_test, y_pred))
print("F1 Score: ", f1_score(y_test, y_pred))
print("Confusion Matrix: \n", cm)
```

```
Accuracy Score:  0.587248322147651
F1 Score:  0.6328358208955224
Confusion Matrix:
 [[ 69  73]
 [ 50 106]]
```

Movie Recommender

In this section, we will construct a method to find out which movie the user should watch after they have inputed a movie that they like.

In [799…

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity, euclidean_distances
```

In [800…

```
def combine_info(row):
    return str(row["Plot"]) + " " + str(row["Genre"])

print(data2["Plot"].isnull().sum())

data2["combined"] = data2.apply(combine_info, axis = 1)
data2[data2['Title'] == 'A Quiet Place Part II']
data2 = data2.drop(['index'], axis = 1)
```

```
2
```

In [801…

```
tfidf = TfidfVectorizer(max_features=2000)
feature = tfidf.fit_transform(data2["combined"])
movie_title = pd.Series(data2.index, index=data2["Title"])
```

In [802…

```
idx = movie_title["A Quiet Place Part II"]
print(idx)
vector = feature[idx]
reccomendations = cosine_similarity(vector, feature)
reccomendations = reccomendations.flatten()
```

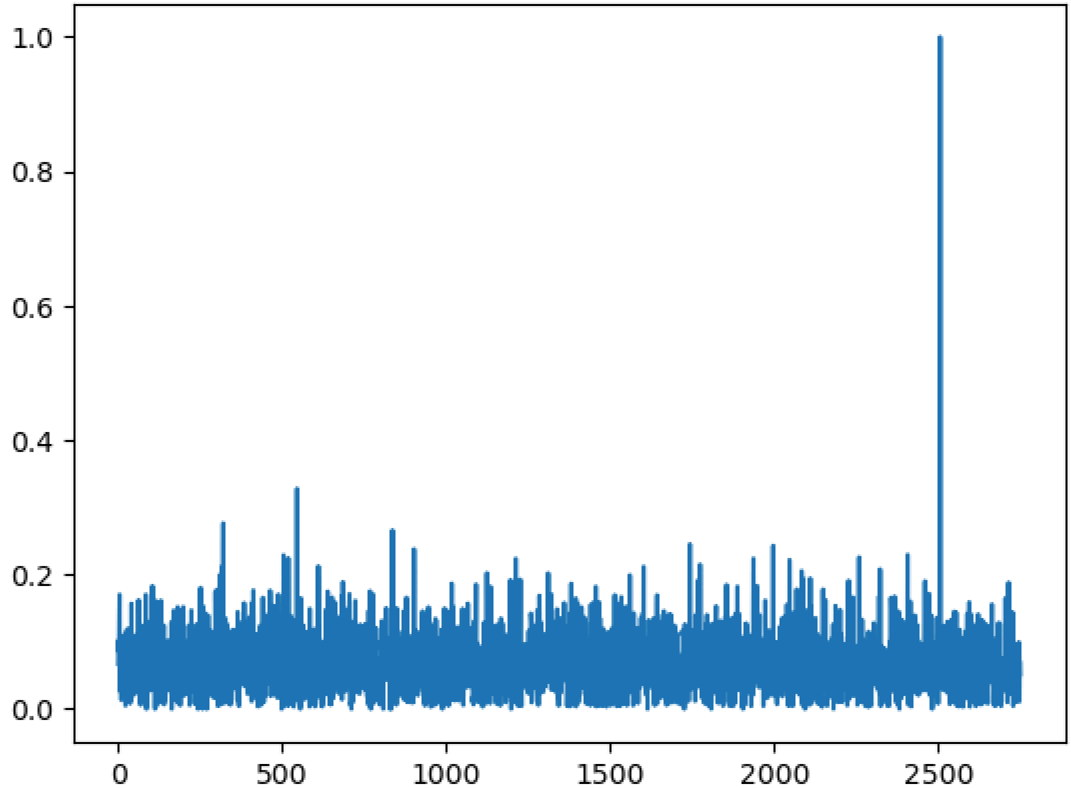
```
2506
```

In [803…

```
plt.plot(reccomendations)
plt.figure(figsize=(10,6))
```

Out[803]:

```
<Figure size 1000x600 with 0 Axes>
```



<Figure size 1000x600 with 0 Axes>

In [804…

```
(-reccomendations).argsort()
final_recs = (-reccomendations).argsort()[1:50]
final_recs
data2.iloc[final_recs]["Title"]
```

Out[804]:
```
545                                          G.I. Joe: Retaliation
320                                                     Prometheus
836                                                       Godzilla
1743                                                 The Domestics
1997                                     Spider-Man: Far From Home
902                                             As Above, So Below
2407                                                    Wrong Turn
504     The Haunting in Connecticut 2: Ghosts of Georgia
2259                                                         Tenet
518                                                     Dark Skies
1937                                                   The Silence
1212                                             Eddie the Eagle
2047                                                     Bloodline
1774                                                       The Meg
318                                                    Piranha 3DD
610                                               This Is the End
1602                                              Pitch Perfect 3
2323                                     Tremors: Shrieker Island
2084                                                  Doctor Sleep
1124                               Maze Runner: The Scorch Trials
1311                                                       Snowden
312                                             Chernobyl Diaries
1560                                    Boo 2! A Madea Halloween
2110                                     Jumanji: The Next Level
2088                                                        Midway
1226                        Batman v Superman: Dawn of Justice
1196                                              Kung Fu Panda 3
1767                                                    Extinction
2227                                                         Relic
2460                                            Godzilla vs. Kong
685                                         Insidious: Chapter 2
2715                                                       Encanto
1017                                              Out of the Dark
1381                           Resident Evil: The Final Chapter
1091                                                     Self/Less
1855                                    Ralph Breaks the Internet
1948                                                    Fast Color
104                                               Horrible Bosses
1888                                                After Darkness
1135                                                   Steve Jobs
1456                          King Arthur: Legend of the Sword
251                                                  Silent House
2069                                     Zombieland: Double Tap
2149                                                      Downhill
301                                                     Battleship
412                                                       Sinister
463                                             Zero Dark Thirty
768                                                   Android Cop
639                                                       V/H/S/2
Name: Title, dtype: object
```

Movie Recommender II

In this section of the movie recommendation system, our objective is to generate a description for a movie that is similar to its original summary using Natural Language Processing techniques. We will use Markov Models as the underlying principle to generate the summary. After generating the summary, we will compare its similarity with the summary scraped from Wikipedia for the same movie using cosine similarity to evaluate the effectiveness of our model.

Before delving into the coding aspect, it's crucial to comprehend the Markov Models, how it relates to the subject learned in class, and why we are utilizing it.

In class, we were introduced to the Naive Bayes model, which is a probabilistic model based on Bayes' theorem. This model calculates the probability of an event given some information, also known as conditional probability. Although Markov Models and Bayes' theorem are related, Markov Models are grounded on the underlying Markov assumption, which is based on the concept of modeling sequences where each state is only reliant on the preceding state.

The Markov model's formal notation is P(x at time t | x at time t - 1, x at time t-2, ...) = p(x at time t | x at time t-1). We plan to utilize Markov Models to predict the most probable word to appear next, given the previous one. If we represent each word as a state (s), P((s at time t) = y|(s at time t-1) = z) would be our objective.

Programmatically, we will store the probabilities in a 2-D matrix, where Xyz = P((s at time t) = y|(s at time t-1) = z). Since we are predicting the next word based on the current word, we need to define something for the initial word since it doesn't have a previous word to rely on.

Therefore, we define the initial state to be the number of times each sequence began with state x divided by the total number of sequences in our dataset. Our Xyz would then be the number of times we transitioned from state y to state z divided by the total number of times we were in state y.

For instance, let's consider the phrase "CMSC320 Rocks." We are interested in the probability of seeing the word "Rocks" following the word "CMSC320." Our Xyz would be the number of times we observe the sequence "CMSC320 Rocks" divided by the number of times we observe the sequence "CMSC320."

Generally, the more conditionals you add to Xyz = P((s at time t) = y|(s at time t-1) = z), the more accurate your predictions will be. Therefore, we plan to enhance our model by adding one more conditional or creating a "Second Order Markov Model" by adding one more, so Xyza = P(st = a| st-1 = z, st-2 = y).

In [805…

```
import requests
import nltk
from nltk.corpus import stopwords
import pandas as pd
import numpy as np
import string
from bs4 import BeautifulSoup
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.metrics.pairwise import cosine_similarity, euclidean_distances
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.metrics import r2_score
nltk.download('stopwords')

r = requests.get("https://en.wikipedia.org/wiki/The_Avengers_(2012_film)")
soup = BeautifulSoup(r.text, 'html.parser')
txt = [soup.find_all('p')[i].getText().replace("\'",'') for i in range(4,8)]
text = " ".join(txt)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\saura\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

First we have to to pick a movie and scrape the summary associated with it. We are going to pick The avengers movie as the movie whose summary we are going to compare. This code retrieves the Wikipedia page for the 2012 film "The Avengers" using the requests library. Then it uses the BeautifulSoup library to parse the HTML text of the page and extract the text of the 4th to 7th paragraphs of the article. The reason for using only the text between the 4th and 7th paragraphs is that the full summary on the Wikipedia page may be too long and contain irrelevant information for the movie recommendation system. By limiting the text to these specific paragraphs, we can focus on the most relevant and concise summary of the movie. The resulting text is then cleaned up and the paragraphs are then joined together into a single string variable called "text".

In [806…

```
initial = {} # stores initial state destribution
first_order = {} # first order transition probabilities but only for the secon
second_order = {} #stores second_order probabilities
```

In this section of code, we are initializing three dictionaries, namely, the initial state dictionary, first_order dictionary, and second_order dictionary. These dictionaries are used for generating text using the Markov Model. The first-order dictionary is used to compute the second-order dictionary. The dictionaries will later store the count of each token and the probability of transitioning from one token to another.

In [807…

```
for line in txt:
    line = line.strip().lower()
    words = line.translate(str.maketrans('', '', string.punctuation)).split()
    #This line removes all punctuation from a given string using Python's buil
    for i in range(len(words)):
    # t would be the first word
        word1 = words[i]
        if i == 0:
        # first word so update initial state distribution
          initial[word1] = initial.get(word1, 0) + 1
        else:
        #not in initial state distribution
            x = words[i-1]
       #since it is not first state and second order we need to grab previous 
            if i == len(words) - 1:
                second_order.setdefault((x, word1), []).append('STOP')
```

```
            # measure probability of ending the line
            #if we are at the end of sentence we create a fake word to let
            #still pass the last word,
        if i == 1:
    # measure distribution of second word given only first word

    #looking at second word in statement so update the first_order which w
            first_order.setdefault(x, []).append(word1)

        else:
    #not looking at the first or second word so we have at least two words
            word2 = words[i-2]
            second_order.setdefault((word2, x), []).append(word1)
```

This code processes a list of text lines by cleaning and tokenizing each line, and then uses the resulting tokens to update various dictionaries. The initial dictionary keeps track of the first word in each line, while the first_order dictionary tracks the probability of the second word given the first word. The second_order dictionary tracks the probability of the third word given the previous two words. Additionally, if the current token is the last one in the line, a special "END" token is added to the second_order dictionary to signify the end of the line. The dictionaries are updated using the dicionary's inbuilt setdefault function, which simply adds the given value to the list of values associated with the given key in the dictionary. Specifically the purpose of that [dictionary].setdefault line is to use a dictionary with a key and a corresponding value that is a list of possible next words. For example, if we have the phrases "I love school, I love cars, I love science", we can translate it into the dictionary {"I love": ["school", "cars", "science"]}. The goal is to later convert this into probabilities of each word appearing.

In [808…

```
#initial total has all the counts as of now so we normalize it
initial_total = sum(initial.values())
for key in initial:
    initial[key] /= initial_total
```

This code block performs normalization on the initial state distribution by dividing each count by the total count of all the initial words. The initial state distribution is a probability distribution of the first word of each line in the text corpus. By dividing each count by the total count, we convert the counts into probabilities that sum to 1, which represents the likelihood of each initial word occurring at the beginning of a line in the text corpus.

In [809…

```
def prob(words):
    prob = {}

    # Iteration
    for item in words:
        # increments key in dictionary by one
        prob[item] = prob.get(item, 0) + 1
    # Normalizing the values
    prob = {item: count/len(words) for item, count in prob.items()}
    return prob
```

This code defines a function called "prob" that takes in a list as an argument. The function then creates an empty dictionary to hold the probabilities of each item in the list. It iterates over each item in the list, and if the item is not already in the dictionary, it adds it with a count of 1. If the item is already in the dictionary, it increments its count by 1. After counting all the items, the function then normalizes the counts to obtain the probabilities of each item, by dividing each count by the total length of the list. Finally, it returns the resulting probability dictionary.

In [810…
```
first_order = {x: prob(y) for x, y in first_order.items()}
second_order = {x: prob(y) for x, y in second_order.items()}
```

This section of code calculates the probabilities of the items in the original dictionaries and assigns the probabilities to the keys for both first_order and second_order dictionaries.

In [811…
```
import random

def random_word(d):
    x = random.random()
    for t, p in d.items():
        if x < p:
            return t
        x -= p
```

The function takes a dictionary of probabilities and returns a word based on a random sample. It iterates over the items in the dictionary, keeping a cumulative sum of their probabilities. If the random number is less than the cumulative sum, it returns the corresponding word. If the loop completes without finding a word, it raises a ValueError.

In [812…
```
def create_summary():
    result = []
    for i in range(4): # generate 4 lines
        sentence = []

        # Initial state word
        first = random_word(initial)
        sentence.append(first)

        # first_order word
        second = random_word(first_order[first])
        sentence.append(second)

    # Continuing to do second_order words until the end
        while True:
            third = random_word(second_order[(first, second)])
            if third == 'STOP':
                break
            sentence.append(third)
            first = second
            second = third
        result.append(' '.join(sentence))
    return result
```

In this function, a list of 4 sentences is generated using a first-order Markov chain. The initial word of each sentence is always "The", and subsequent words are chosen probabilistically based on the previous word in the chain using the random_word function. The process continues until the word "FINAL" is generated, at which point the sentence is complete. The resulting sentences are returned as a list.

In [813…

```
sample_text = create_summary()
sample_text = ' '.join(sample_text)
```

Let us see what we generated compared to what we scraped.

In [814…

```
print("Generated Text: \n" + sample_text)
print("Original Text: \n"  +  text)
```

```
Generated Text:
rogers stark and rogers work to restart the damaged engine and thor attempts t
o stop the hulks rampage romanoff knocks barton unconscious breaking lokis min
d control loki escapes after killing coulson and fury uses coulsons death to m
otivate the avengers become divided over how to approach loki and the stress c
auses banner to transform into the hulk saves him while romanoff uses lokis sc
epter to close the wormhole in the aftermath thor returns with loki and the st
ress causes banner to transform into the hulk saves him while romanoff uses lo
kis scepter can shut down the generator furys superiors from the world securit
y council attempt to end the invasion by launching a nuclear missile at midtow
n manhattan stark intercepts the missile detonates destroying the chitauri the
hulk beats loki into submission romanoff makes her way to the chitauri mothers
hip and disabling their forces on earth starks suit loses power and he goes in
to freefall but the hulk saves him while romanoff uses lokis scepter can shut
down the generator where selvig freed from lokis mind control loki escapes aft
er killing coulson and fury uses coulsons death to motivate the avengers battl
e the chitauri mothership and disabling their forces on earth starks suit lose
s power and he goes into freefall but the hulk saves him while romanoff uses l
okis scepter to close the wormhole toward the chitauri in exchange for retriev
ing the tesseractc a powerful energy source of unknown potential the other pro
mises loki an army with which he can subjugate earth nick fury director of the
espionage agency shield arrives at a remote research facility where physicist
dr erik selvig is leading a team loki uses the tesseract to asgard where loki
will face their justice rogers stark romanoff barton thor and the revelation t
hat shield plans to harness the tesseract and a wormhole above stark tower to
the chitauri mothership and disabling their forces on earth starks suit loses
power and he goes into freefall but the hulk saves him while romanoff uses lok
is scepter can shut down the generator furys superiors from the world security
council attempt to end the invasion by launching a nuclear missile at midtown
manhattan stark intercepts the missile detonates destroying the chitauri mothe
rship and disabling their forces on earth starks suit loses power and he goes
into freefall but the hulk saves him while romanoff uses lokis scepter to ensl
ave selvig and other agents including clint barton to aid him in response fury
reactivates the avengers initiative agent natasha romanoff heads to kolkata to
recruit dr bruce banner to transform into the hulk saves him while romanoff us
es lokis scepter to close the wormhole toward the chitauri the hulk stark and
rogers work to restart the damaged engine and thor attempts to stop the hulks
rampage romanoff knocks barton unconscious breaking lokis mind control loki es
capes after killing coulson and fury uses coulsons death to motivate the aveng
ers become divided over how to approach loki and the revelation that shield pl
ans to harness the tesseract and uses his scepter to enslave selvig and other
agents including clint barton to aid him in response fury reactivates the aven
gers battle the chitauri fleet the missile and takes it through the wormhole t
oward the chitauri fleet the missile detonates destroying the chitauri fleet t
he missile detonates destroying the chitauri the hulk stark and rogers work to
restart the damaged engine and thor attempts to stop the hulks rampage romanof
f knocks barton unconscious breaking lokis mind control loki escapes after kil
ling coulson and fury uses coulsons death to motivate the avengers into workin
g as a team experimenting on the tesseract to develop powerful weapons as a te
am loki uses the tesseract suddenly activates and opens a wormhole allowing lo
ki to reach earth loki steals the iridium needed to stabilize the tesseracts p
ower leading to a confrontation with rogers stark romanoff barton thor and the
revelation that shield plans to harness the tesseract through its gamma radiat
ion emissions fury approaches steve rogers to retrieve the tesseract and agent
phil coulson visits tony stark to have him check selvigs research loki is in s
tuttgart where barton steals the iridium needed to stabilize the tesseracts po
wer leading to a confrontation with rogers stark romanoff barton thor and the
tesseract and agent phil coulson visits tony stark to have him check selvigs r
esearch loki is taken to shields flying aircraft carrier the helicarrier where
he is imprisoned
```

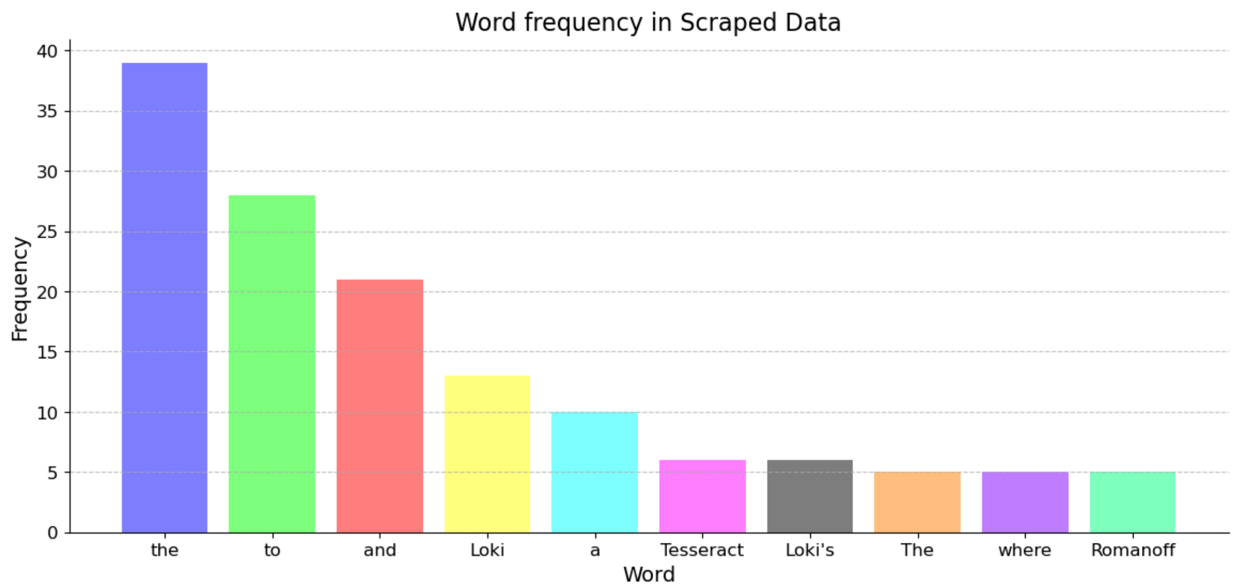
```
Original Text:
The Asgardian Loki encounters the Other, the leader of an extraterrestrial rac
e known as the Chitauri. In exchange for retrieving the Tesseract,[c] a powerf
ul energy source of unknown potential, the Other promises Loki an army with wh
ich he can subjugate Earth. Nick Fury, director of the espionage agency S.H.I.
E.L.D., arrives at a remote research facility, where physicist Dr. Erik Selvig
is leading a team experimenting on the Tesseract. The Tesseract suddenly activ
ates and opens a wormhole, allowing Loki to reach Earth. Loki steals the Tesse
ract and uses his scepter to enslave Selvig and other agents, including Clint
Barton, to aid him.
 In response, Fury reactivates the "Avengers Initiative". Agent Natasha Romano
ff heads to Kolkata to recruit Dr. Bruce Banner to trace the Tesseract through
its gamma radiation emissions. Fury approaches Steve Rogers to retrieve the Te
sseract, and Agent Phil Coulson visits Tony Stark to have him check Selvig's r
esearch. Loki is in Stuttgart, where Barton steals the iridium needed to stabi
lize the Tesseract's power, leading to a confrontation with Rogers, Stark, and
Romanoff that ends with Loki's surrender. While Loki gets escorted to S.H.I.E.
L.D., his adoptive brother Thor arrives and frees him, hoping to convince him
to abandon his plan and return to Asgard. Stark and Rogers intervene and Loki
is taken to S.H.I.E.L.D.'s flying aircraft carrier, the Helicarrier, where he
is imprisoned.
 The Avengers become divided over how to approach Loki and the revelation that
S.H.I.E.L.D. plans to harness the Tesseract to develop powerful weapons as a d
eterrent against hostile extraterrestrials. As they argue, Loki's agents attac
k the Helicarrier, and the stress causes Banner to transform into the Hulk. St
ark and Rogers work to restart the damaged engine, and Thor attempts to stop t
he Hulk's rampage. Romanoff knocks Barton unconscious, breaking Loki's mind co
ntrol. Loki escapes after killing Coulson and Fury uses Coulson's death to mot
ivate the Avengers into working as a team. Loki uses the Tesseract and a wormh
ole generator Selvig built to open a wormhole above Stark Tower to the Chitaur
i fleet in space, launching his invasion.
 Rogers, Stark, Romanoff, Barton, Thor, and the Hulk rally in defense of New Y
ork City, and together the Avengers battle the Chitauri. The Hulk beats Loki i
nto submission. Romanoff makes her way to the generator, where Selvig, freed f
rom Loki's mind control, reveals that Loki's scepter can shut down the generat
or. Fury's superiors from the World Security Council attempt to end the invasi
on by launching a nuclear missile at Midtown Manhattan. Stark intercepts the m
issile and takes it through the wormhole toward the Chitauri fleet. The missil
e detonates, destroying the Chitauri mothership and disabling their forces on
Earth. Stark's suit loses power and he goes into freefall, but the Hulk saves
him, while Romanoff uses Loki's scepter to close the wormhole. In the aftermat
h, Thor returns with Loki and the Tesseract to Asgard, where Loki will face th
eir justice.
```

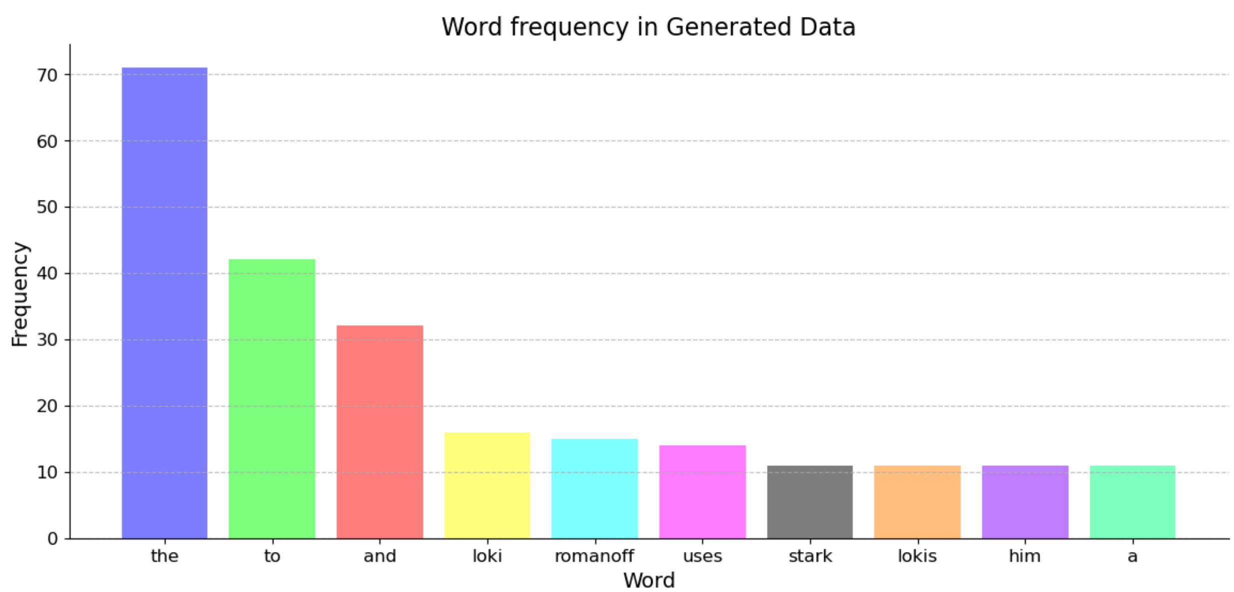
There seems to be slight differences within the original and generated texts. To effectively compare both texts it is neccessary to get rid of stopwords. Let us plot the word counts of both texts to see why.

In [815…

```python
from collections import defaultdict

def plot(sample_text, title):
    sentence_words = defaultdict(int)
    for words in sample_text.split(" "):
        sentence_words[words] += 1
    sentence_words = dict(sorted(sentence_words.items(), key=lambda x:x[1], re
    sentence_words
    plt.figure(figsize=(14, 6))
    names = list(sentence_words.keys())
```

```
    values = list(sentence_words.values())
    colors = ['#7F7FFF', '#7FFF7F', '#FF7F7F', '#FFFF7F', '#7FFFFF', '#FF7FFF'
    plt.bar(range(10), values[:10], tick_label=names[:10], color=colors)
    plt.title(title, fontsize=16)
    plt.xlabel('Word', fontsize=14)
    plt.ylabel('Frequency', fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.gca().spines['top'].set_visible(False)
    plt.gca().spines['right'].set_visible(False)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()
plot(" ".join(txt), "Word frequency in Scraped Data")
plot(sample_text, "Word frequency in Generated Data")
```





As seen when we display the words in the generated paragraph, "the", "to", "and", "a" are among the most common words in both summaries and thus might skew the similarity results.

In order for us to compare with our generated text with our original text let us further clean both these texts by getting rid of the stopwords

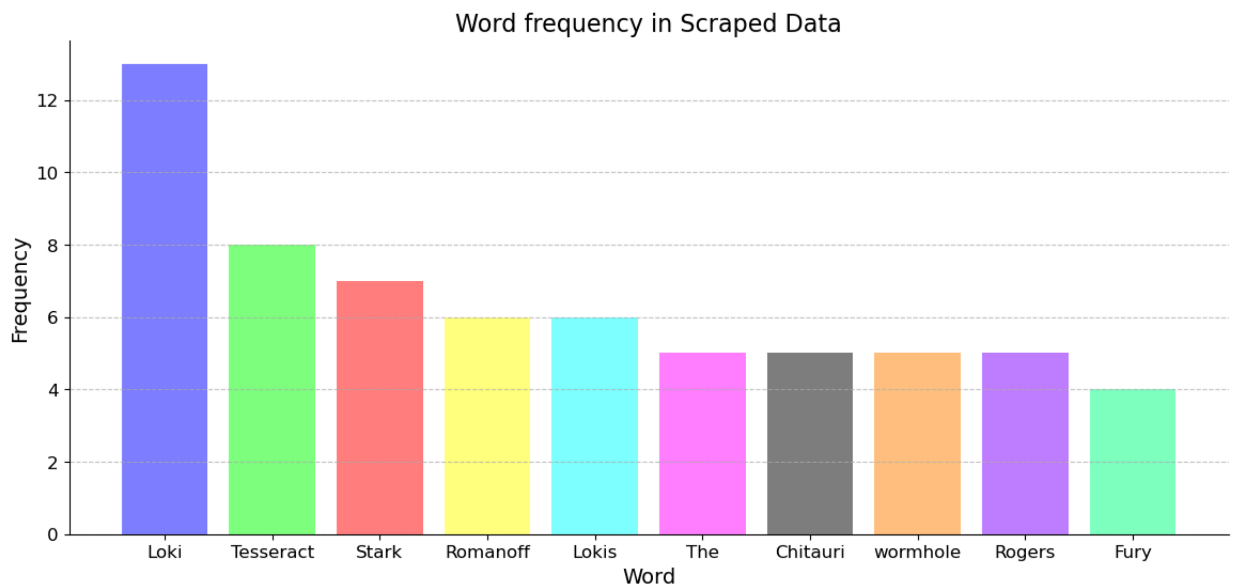
In [816…

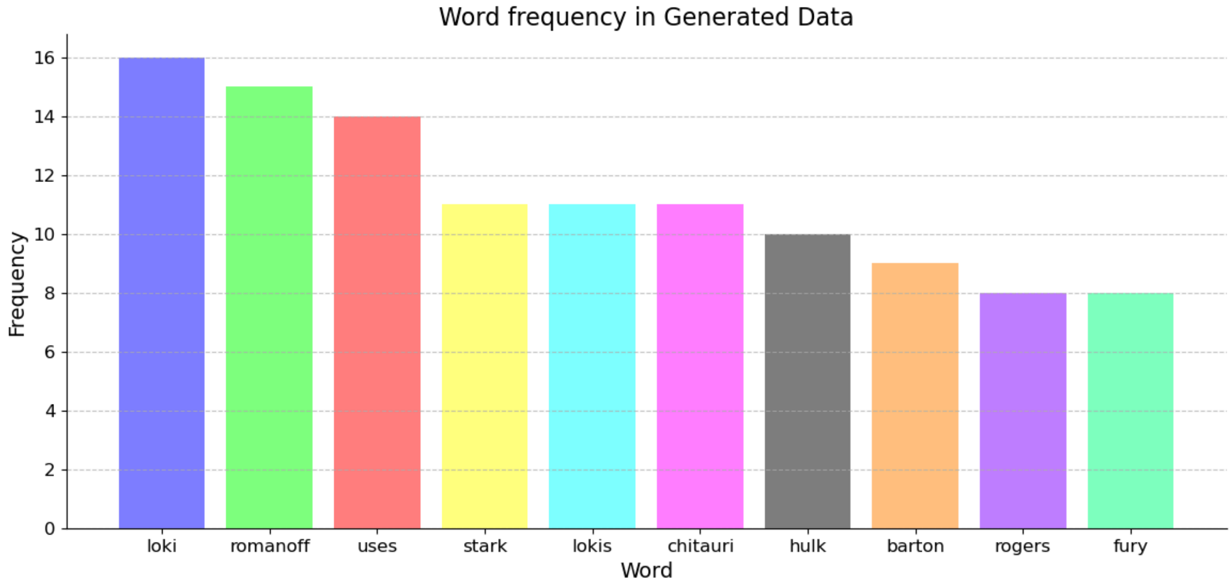
```
import nltk
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
stop_words = set(stopwords.words("english"))

def cleaning2(two):
    final_descrip = []
    for descrip in two:
        descrip = descrip.translate (str.maketrans('', '', string.punctuation)
        tokens = word_tokenize(descrip)
        sentence = [word for word in tokens if word.lower() not in stop_words]
        final = []
        for word in tokens:
            if word not in stop_words:
                final.append(word)
        final_descrip.append(" ".join(final))
    return final_descrip
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\saura\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [817…

```
orig_text = " ".join(cleaning2(txt))
generated_text = " ".join( cleaning2([sample_text]))
plot(orig_text, "Word frequency in Scraped Data")
plot(generated_text, "Word frequency in Generated Data")
```





Here we call out cleaning function on both the original text as well as the generated text to get rid of stopwords and punctation that might affect the cosine similarity process. Now the cosine similarity will be able to more accurately compare the similarity between both texts.

In [818…

```
vectorizer = TfidfVectorizer()
tfidf = vectorizer.fit_transform([orig_text, generated_text])
cosine_sim = (cosine_similarity(tfidf[0], tfidf[1])[0][0]).reshape(1,-1)
print(cosine_sim[0][0])
```

```
0.8398694708765687
```

A cosine similarity score of ~0.80 indicates a relatively high level of similarity between the summary of the movie obtained online and the summary generated using Markov models.However, it is to be noted that the generated text isn't the same every time it is ran. In fact, after running this multiple times the lowest similartiy I got for the same text was 0.53.

Thus, it can be said that there is a lot more that goes into text generated. Markov Models work by picking the sequence of words that result in the highest probability. Because of this, if one were to read the generated text, some sentences are not coherent and because of this some sentences dont make since. However, Markov Models serve as a great way to introduce the idea of text generation.

The cosine similarity score of around 0.80 indicates that there is a relatively high level of similarity between the movie summary obtained online and the summary generated using Markov models. However, it should be noted that the generated text is not the same every time it is run. In fact, when the same text was run multiple times, the lowest similarity score obtained was 0.53. This suggests that text generation involves a lot more than just using Markov models.

Markov models generate text by selecting the sequence of words that have the highest probability. As a result, some sentences in the generated text may not be coherent, and some may not make sense when read. Nevertheless, Markov models provide a useful way to introduce the concept of text generation.

Interpretation: Insight & Policy Decision