# IMDB Top 1000: The Recipe to Success

## Aims, objectives, background and ethics

**1.1 Introduction**

Movies are inherently unique in their ability to captivate audiences on both a personal and collective level. The remarkable success of films, particularly the ones which reach the highest echelons of the box office, have been studied and analysed for decades as filmmakers strive to replicate success.

In order to make a blockbuster movie, you need more than just a good story. You also need the right ingredients. But what are these ingredients? What defines a movie's success at the box office? Is the success of a movie defined by the pockets it fills or by the hearts it captures?

With the recent downfall of film theatres due to the pandemic, this topic has become even more pertinent as studios make their way into the digital age. To gain an insight into the success of films, I have set my sight on IMDB's Top 1000 Movies List in the hope of uncovering meaningful patterns and data insights that can shed light on what factors contribute to a movie's success.

**1.1.1 Motivation & Relevance**

This research area has the potential to help filmmakers create movies that are both profitable and well-received by critics and audiences worldwide. Furthermore, this research serves as a guide for aspiring filmmakers and producers, providing insight from those who have come before them and have achieved success in the movie industry. Finally, it may help to identify emerging trends in cinema and allow for more informed decision-making when creating new movies.

## 1.2 Aims and objectives

The primary goal of this research is to identify factors that make up the recipe for success in the movie industry. The project aims to uncover what makes a movie commercially successful and critically acclaimed, by looking at the common characteristics of the IMDB Top 1000.

**Below are some aims for the project:**

- Understanding the characteristics of the movies in the dataset: This would involve analyzing various features of the movies, such as duration, rating, votes, and gross revenue, to understand their distribution and identify patterns or trends within the data.
- Identifying factors that influence a movie's financial success
- Providing insights for decision-making within the film industry: The findings of the data analysis project could be used to inform decisions about future projects, such as identifying the characteristics of successful movies and the factors that drive a movie's financial success.
- Building a predictive model
- Identifying any changes over time: Analyzing the dataset over time, can help us understand how the film industry has changed, if there is a trend in the characteristics of movies, what is the impact of technology on the industry, etc.

**The objectives are to:**

- Discover any trends in movie genres that drive success
- Examine the correlation between critically acclaimed films and its box office success.
- Explore how well various aspects of filmmaking correlate with box office performance and reviews
- Evaluate what elements lead to audience engagement, loyalty and box office success.
- Explore whether the cast or director of a film impacts ratings or gross
- Explore whether the plot of a film impacts ratings or gross.
- Building a model that can predict the gross revenue of a movie based on certain features can be useful for making predictions about future projects and identifying factors that influence a movie's financial success.

## 1.3 Processing Pipeline

**Below is the step-by-step implementation and processing pipeline of this project:**

Brainstorming and Research: Research is the foundation of this project. The research will be conducted by researching existing literature and analyzing relevant datasets, as well as the type of visual encodings that could be used to perform this analysis. Data Acquisition – Acquiring datasets from sources such as IMDb and Kaggle Data Preprocessing and Cleaning: Cleaning, transforming, and organising the dataset. Exploratory Data Analysis: Visualizing and exploring the data to observe patterns, trends, and potential relationships with the use of techniques such as correlation or statistical analysis. Extra Curriculum: Building a predictive model to estimate the success of a movie based on different parameters. Results & Conclusion: Present the results of the analysis and conclusions drawn from it.

## 2 Dataset

**2.1.1 Data Requirements**

To accurately assess the data, it is important to understand what information we need and in what format. The two main components that are necessary for exploratory data analysis on the IMDb Top 1000 movies are:

1. A list of all the movies in the top 1000 and
2. Their associated meta-data. For each movie, we will need to know the title

- year of release
- genre classification
- IMDb rating
- director name(s)
- cast names
- Gross
- Votes
- Plots
- Duration

Gathering this information from the IMDb website is relatively straightforward and can be done using web scraping.

Once the data has been collected and stored in an appropriate database, the next step is to perform basic analysis and visualisation techniques, which will be illustrated below.

**2.1.2 Origin of Data**

The data for this exploratory data analysis project is derived from the IMDb Top 1000 movies list, which is available on the IMDb website. IMDb (Internet Movie Database) is a website that provides information about movies, TV shows, and other media. It is owned by Amazon and is one of the largest and most popular online movie databases in the world. The IMDb Top 1000 movies list is a ranking of the top 1000 movies as rated by IMDb users. It includes a variety of information about each movie, such as the title, year of release, genre, rating, and number of votes.

To acquire this data, I accessed a CSV file first from Kaggle, which is available to the public however, due to some important missing data and information from the data set such as gross, cast, and director, I will be web scraping the IMDB website to acquire the data. The process of web scraping is a technique used to extract large amounts of data from websites in an organised format. It involves writing a program that uses automated processes to collect data from the web, which would otherwise take much longer if done manually.

It is worth noting that the IMDb Top 1000 movies list is based on the ratings and votes of IMDb users, and therefore may be subject to biases or errors. It is important to consider these potential limitations when interpreting the results of the analysis. The limitations and constraints of the data set have been noted in 3.

**2.1.3 Methodology**

This project involves the development of multiple graphical/visual encodings with an aim to identify any common features among top-ranking movies. To structure, clean and process the data for this project, I will be using Python and its distinct libraries such as Pandas, Matplotlib, Seaborn, etc. Additionally, exploratory data analysis (EDA) and natural language processing (NLP) will also be used to create a series of visualisations to uncover any meaningful correlations.

**Types of Visual forms to be used:**

- Graphs
- Charts
- Tables
- Scattered Plots
- Diagrams

# 3 Limitations and constraints of the Data set

There are several limitations and constraints to consider when performing exploratory data analysis on the dataset:

1. Representativeness: The IMDB Top 1000 movies may not be representative of the entire population of movies. This means that any conclusions drawn from the analysis may not apply to all movies.
2. Missing data: There may be missing data for some movies in the dataset. This could lead to biases in the analysis and may affect the accuracy of any conclusions drawn.
3. Quality of data: The quality of the data may vary and may not always be reliable. It is important to carefully assess the quality of the data before using it for analysis.
4. Time period: The IMDB Top 1000 movies may only include movies from a specific time period, which could limit the applicability of the analysis to other time periods.
5. Subjectivity: The ratings and rankings for movies on IMDB are based on user opinions, which can be subjective and may not accurately reflect the quality of the movies.
6. Limited scope: The IMDB Top 1000 movies only includes a small number of movies, which could limit the ability to draw general conclusions about the entire population of movies.

# 4 Ethical considerations

There are certain ethical considerations that must be taken into account when working with the IMDB Top 1000 dataset. Firstly, we must ensure that all data is collected and used in a responsible and ethical manner. As some information provided by the dataset may be sensitive or confidential, it is important to take steps to protect this data from misuse. Secondly, we should consider any legal implications of using the data set such as copyright laws or other restrictions. Finally, we should consider the ethical effects of sharing the results of our analysis; how will people use this information? Will it have a positive or negative impact on society?

#### 4.1.1 Ethics of practices and data

1. Responsible use of data: It is important to use the data in a responsible and ethical manner, and to respect the rights and privacy of the users who have provided the data.
2. Transparency: The methods and processes used in the analysis should be transparent and clearly documented, to allow others to replicate and verify the results.
3. Fairness: The analysis should be conducted in a fair and unbiased manner, without discrimination against any particular group of users.
4. Confidentiality: It is important to respect the confidentiality of the data and not disclose any personal information about the users who have rated the movies.
5. Privacy: It is important to protect the privacy of the users who have rated the movies by not sharing any identifying information about them.
6. Informed consent: It is important to ensure that the users who have rated the movies have given their informed consent for their data to be used for analysis.
7. Accuracy: The analysis should be based on accurate and reliable data, and any conclusions drawn should be supported by the data.
8. Data security: It is important to protect the security of the data and ensure that it is not accessed or used by unauthorized individuals.

#### 4.1.2 Reuse of data

There are several considerations that I will be taking into account when reusing data from the IMDB top 1000 movies list:

Data accuracy: It is important to ensure that the data being reused is accurate and up-to-date. This may involve cross-checking the data against other sources to ensure that it is correct.

Copyright: The data contained within the IMDB top 1000 list is likely to be protected by copyright. It is important to ensure that any reuse of the data complies with copyright laws and obtains the necessary permissions. For this i will review its terms of use. Additionally, the data is not being used for commercial use.

Data context and interpretation: It is important to consider the context in which the data was originally collected and to be aware of any limitations or biases that may be present. This may involve conducting additional research to fully understand the context of the data. For example, i know that rating is conducted by IMDB platform users and the data may be biased.

Also, the data may be open to interpretation and it is important to be mindful of any subjective biases that may influence the way the data is interpreted. It is important to consider multiple viewpoints and to be aware of any potential limitations of the data.

```
!pip install requests --upgrade --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.8/62.8 KB 6.2 MB/s eta 0:00:00
```

```
!pip install beautifulsoup4 --upgrade --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 128.2/128.2 KB 11.6 MB/s eta 0:00:00
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Import libraries and modulesi
import pandas as pd
import nltk
import re
import os
from bs4 import BeautifulSoup
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import RegexpTokenizer
import matplotlib.pyplot as plt
from collections import Counter
import requests
import numpy as np;
import seaborn as sns
import itertools
import collections
from wordcloud import WordCloud, STOPWORDS
from sklearn.linear_model import LinearRegression
from matplotlib.pyplot import figure
```

```
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.model_selection import train_test_split



nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')

stopwords = nltk.corpus.stopwords.words('english')
tokenizer = RegexpTokenizer(r'\w+')

# Current Directory Path
os.getcwd()
os.chdir("/content/drive/MyDrive/PWD/Mid Project/")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

# 5 Webscraping IMDB

Webscraping is the process of extracting data from websites. In this case, I will be using web scraping to retrieve important information such as gross, cast, directors, and more for my dataset which is not currently available on Kaggle.

To scrape the data we need, I will use a function that can extract information like title, gross, year, genre, plot etc. This function can then store all the scraped data in my dataset.

Extracting functions are also used to parse through the downloaded content to identify specific elements such as words and phrases that can be used in my analysis. They may also be used to manipulate the scraped data into a usable format such as CSV.

Before scraping I am defining the list which will store the scraped data and giving the required URL from which we will scrape.

```
#  Creating the lists we want to writ/e into
titles = []
years = []
time = []
imdb_ratings = []
metascores = []
votes = []
us_gross = []
directors=[]
casts=[]
votes=[]
gross=[]


topics_url = 'https://www.imdb.com/search/title/?count=100&groups=top_1000&sort=user_rating'
response = requests.get(topics_url)
page_content = response.text


# Parse the webpage now we've checked it's accessible
doc = BeautifulSoup(response.text, 'html.parser')
```

**Extracting web content**

The following content extracts the webpage content as a HTML doc

```
def get_topics_page():
    # Define the url of the page
    topic_url = 'https://www.imdb.com/search/title/?groups=top_1000&sort=user_rating,desc&count=100&start=000&ref_=adv_next'

    # Send a GET request to the url
    response=requests.get(topic_url)

    # check successfull response
    if response.status_code != 200:
        raise Exception(f'Failed to load page {topic_url}')

    # Parse the HTML content of the page using BeautifulSoup
    doc = BeautifulSoup(response.text, 'html.parser')
    return doc
```

```
# Get the top 1000 movie titles page
doc=get_topics_page()
```

**Extraction required information from IMDB**

After we have scraped the data and created our data frame, the next step of web scraping is to extract the required information. The function below allows us to parse through the scraped data and identify specific elements such as movie title, year, genre, plot, director, cast, gross, URL, rating, duration and votes. Once we have identified these elements within our dataset, they can then be manipulated into an array or dictionary format for further analysis or use in applications.

```
# extract movie title
def get_movie_titles(doc):
    selection_class="lister-item-header"
    movie_title_tags=doc.find_all('h3',{'class':selection_class})
    movie_titles=[]

# reading all movie titles in list
    for tag in movie_title_tags:
        title = tag.find('a').text
        movie_titles.append(title)
    return movie_titles

# storing movie titles in a list
titles = get_movie_titles(doc)

# extract movie genre
def get_movie_year(doc):
    year_selector = "lister-item-year text-muted unbold"
    movie_year_tags=doc.find_all('span',{'class':year_selector})
    movie_year_tagss=[]
# reading all movie years in list
    for tag in movie_year_tags:
        movie_year_tagss.append(tag.get_text().strip()[1:5])
    return movie_year_tagss

# storing movie years in a list
years = get_movie_year(doc)

# extract movie genre
def get_movie_genre(doc):
    year_selector = "genre"
    movie_year_tags=doc.find_all('span',{'class':year_selector})
    movie_year_tagss=[]
# reading all movie genres in list
    for tag in movie_year_tags:
        movie_year_tagss.append(tag.get_text().strip())
    return movie_year_tagss

# storing movie genres in a list
genres = get_movie_genre(doc)
len(genres)

# extract movie plots
def get_movie_plots(doc):
    year_selector = "text-muted"
    movie_year_tags=doc.find_all('p',{'class':year_selector})
    movie_year_tagss=[]
    for tag in movie_year_tags:
        movie_year_tagss.append(tag.get_text().strip())
    plots = movie_year_tagss
    t = []
    for x in range(len(plots)):
      if not x%2 == 0:
        t.append(plots[x])
    return t

# stroring movie plots in a list
plots = get_movie_plots(doc)
len(plots)

# extract movie cast and director names
def get_movie_cast_director(doc):
    year_selector = ""
    movie_year_tags=doc.find_all('p',{'class':year_selector})
    movie_year_tagss=[]
    for tag in movie_year_tags:
        movie_year_tagss.append(tag.get_text().strip().replace("\n","").split("|"))
    plots = movie_year_tagss
    director=[]
```

```
        stars=[]
        for x in plots:
          director.append(x[0].replace("Director:",""))
          stars.append(x[1].replace("Stars:","").strip().split(","))

        return director,stars

    # stroring movie cast and directors in a list
    directors,casts = get_movie_cast_director(doc)

    # extract movie gross income
    def get_gross(doc):
        year_selector = "sort-num_votes-visible"
        movie_year_tags=doc.find_all('p',{'class':year_selector})
        movie_year_tagss=[]
        for tag in movie_year_tags:
            movie_year_tagss.append(tag.get_text().strip().replace("\n",""))
        plots = movie_year_tagss

        v= []
        g=[]
        # extracting votes information
        for x in plots:
          if "Votes" in x:
            t = x.split("|")
            for z in t:
              if "Votes" in z:
                v.append(z.replace("Votes:","").strip().replace(",",""))
          else:
            v.append(-1)
        # extracting gross income information
        for x in plots:
          if "Gross" in x:
            t = x.split("|")
            for z in t:
              if "Gross" in z:
                g.append(z.replace("Gross:","").strip().replace(",",""))
          else:
            g.append(-1)

        return v,g
    # stroring movie votes and gross profit in a list
    votes,gross = get_gross(doc)

    def get_movie_url(doc):
        url_selector="lister-item-header"
        movie_url_tags=doc.find_all('h3',{'class':url_selector})
        movie_url_tagss=[]
        base_url = 'https://www.imdb.com/'
        for tag in movie_url_tags:
            movie_url_tagss.append('https://www.imdb.com/' + tag.find('a')['href'])
        return movie_url_tagss

    urls = get_movie_url(doc)
    len(urls)

    # extract movie rating
    def get_movie_rating(doc):
        rating_selector="inline-block ratings-imdb-rating"
        movie_rating_tags=doc.find_all('div',{'class':rating_selector})
        movie_rating_tagss=[]
        for tag in movie_rating_tags:
            movie_rating_tagss.append(tag.get_text().strip())
        return movie_rating_tagss

    # stroring movie ratings in a list
    ratings = get_movie_rating(doc)

    # extract movie duration
    def get_movie_duration(doc):

        selection_class="runtime"
        movie_duration_tags=doc.find_all('span',{'class':selection_class})
        movie_duration=[]

        for tag in movie_duration_tags:
            duration = tag.text[:-4]
            movie_duration.append(duration)


        return movie_duration
```

```
# stroring movie durations in a list
```

**5.1 Scrape data and save to data frame**

After we have created the web scraping function for collecting the necessary data, our next step is to scrape the data and save it into a data frame. A data frame is a two-dimensional structure in which data is stored in rows and columns. This makes the data easy to organize and analyze, as well as easy to export into other formats such as CSV.

To scrape the data, I will use a loop that iterates over for each parameter we require. We will then store this information into an array of dictionaries or objects, where each item contains the scraped data from each element on the page. Once all the relevant elements have been scraped, we can then create our data frame by providing column names and assigning values from our array of dictionaries/objects to them.

Next, I will create a dictionary to store the data with the specified column names. You can use the .append() method to add a new row to the dataframe for each piece of data.

```
def all_pages():
# create a dictionary to store data of all movies
    movies_dict={
        'title':[],
        'genre':[],
        'duration':[],
        'rating':[],
        'year':[],
        'plots':[],
        'url':[],
        'director':[],
        'cast':[],
        'votes':[],
        "gross":[]
    }
  # We have to scrap more than one page so we want urls of all pages with the help of loop we can get all urls
    for i in range(1,2000,100):

        try:
            url = 'https://www.imdb.com/search/title/?groups=top_1000&sort=user_rating,desc&count=100&start='+str(i)+'&ref_=adv_next'
            response = requests.get(url)
        except:
            break

        if response.status_code != 200:
            break

    # Parse using BeautifulSoup
        doc = BeautifulSoup(response.text, 'html.parser')
        titles = get_movie_titles(doc)
        urls = get_movie_url(doc)
        plots = get_movie_plots(doc)
        ratings = get_movie_rating(doc)
        durations = get_movie_duration(doc)
        years = get_movie_year(doc)
        genres = get_movie_genre(doc)
        directors,casts = get_movie_cast_director(doc)
        votes,gross = get_gross(doc)


    # We are adding every movie data to dictionary
        for i in range(len(titles)):
            movies_dict['title'].append(titles[i])
            movies_dict['genre'].append(genres[i])
            movies_dict['duration'].append(durations[i])
            movies_dict['rating'].append(ratings[i])
            movies_dict['year'].append(years[i])
            movies_dict['plots'].append(plots[i])
            movies_dict['url'].append(urls[i])
            movies_dict['director'].append(directors[i])
            movies_dict['cast'].append(casts[i])
            movies_dict['votes'].append(votes[i])
            movies_dict['gross'].append(gross[i])


    return pd.DataFrame(movies_dict)


movies_dict={
    'title':titles,
    'genre':genres,
    'duration':durations,
    'rating':ratings,
    'year':years,
    'plots':plots,
    'url':urls,
```

```
    'director': directors,
    "cast":casts,
    "votes":votes,
    "gross":gross
}

df = pd.DataFrame(movies_dict)
```

#### 5.2 Check scraped data and saving to CSV file

After I have scraped the data and created the data frame, it is important that I check the scraped data to ensure it is accurate and complete. I can do this by running basic tests such as `dataframe.info()` to check for any missing values and verify that all columns are in their correct format.

Once I have verified that all the scraped data is accurate and complete, I can then save it into a CSV file so I can access it from any other application or program. To save the data frame into a CSV file, I will use the `dataframe.to_csv('filename')` command which will create a CSV file with our entire dataset.

```
movies = all_pages()

movies.to_csv('movie_data.csv')
movies
```

| | title | genre | duration | rating | year | plots | |
|---|---|---|---|---|---|---|---|
| **0** | The Shawshank Redemption | Drama | 142 | 9.3 | 1994 | Over the course of several years, two convicts... | https://www.imdb.com/ |
| **1** | The Godfather | Crime, Drama | 175 | 9.2 | 1972 | The aging patriarch of an organized crime dyna... | https://www.imdb.com/t |
| **2** | The Dark Knight | Action, Crime, Drama | 152 | 9.0 | 2008 | When the menace known as the Joker wreaks havo... | https://www.imdb.com/t |
| **3** | Schindler's List | Biography, Drama, History | 195 | 9.0 | 1993 | In German-occupied Poland during World War II,... | https://www.imdb.com/t |

#### 5.3 Import previously scraped data

Importing previously scraped data can be useful for a number of reasons. It can save time and resources by avoiding the need to scrape the data again and it can help verify that the analysis is replicable If I am able to run the analysis multiple times using the same data and get the same results, this can increase confidence in the validity of my findings.

However, there are also some risks associated with importing previously scraped data. For example with amendments and changes to the HTML of the webpage the data may be outdated or no longer be representative of the current state of the website, this would be the case for this particular dataset as the industry would constantly have new releases climbing the top 1000 charts.

```
# reading movie data
df = pd.read_csv("movie_data.csv")

# setting numbers precisino value for dataframe
pd.set_option('precision', 2)

# print the first five rows of the data
df.head()
```

| | Unnamed: 0 | title | genre | duration | rating | year | plots | |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | The Shawshank Redemption | Drama | 142 | 9.3 | 1994 | Over the course of several years, two convicts... | https://www.imc |
| **1** | 1 | The Godfather | Crime, Drama | 175 | 9.2 | 1972 | The aging patriarch of an organized crime | https://www.imd |

## 6 Overview of dataset

The IMDb top 1000 movies dataset is a list of the top 1000 movies as rated by users of the Internet Movie Database (IMDb). The list is updated on a regular basis, and typically includes a mix of classic and contemporary films.

The dataset includes various details about each movie, such as the title, year of release, IMDb rating, number of ratings, genre, gross, cast, directors and plot.

We can see the average rating is 7.9 with a low deviation of 0.27 as we are viewing the top 1000 movies. The maximum rating is 9.3 and duration is 3 hours 21 minutes with a minium of 7.6 rating and 45 minutes duration. Average number of votes are above 30,000 which is good so that the voting can be considered as part of a larger sample.

```
# getting data sescription and basic stats regarding numberic attributes
df.describe().apply(lambda s: s.apply(lambda x: format(x, 'g')))
```

| | Unnamed: 0 | duration | rating | votes |
|---|---|---|---|---|
| **count** | 1000 | 1000 | 1000 | 1000 |
| **mean** | 499.5 | 123.984 | 7.9675 | 309630 |
| **std** | 288.819 | 28.6978 | 0.276496 | 374603 |
| **min** | 0 | 45 | 7.6 | 25519 |
| **25%** | 249.75 | 103 | 7.8 | 59822 |
| **50%** | 499.5 | 120 | 7.9 | 151669 |
| **75%** | 749.25 | 138 | 8.1 | 428894 |
| **max** | 999 | 321 | 9.3 | 2.68963e+06 |

Explain duration and rating. Average rating, mean, min. As nature of data is top 1000 average rating is 7.9. STD Difference of all movies is 28 mins. No outliers as describe function shows this

## 7 Data cleaning and processing

Data cleaning and processing is essential to ensure accuracy and reliability of any analysis conducted with the data. This involves some of the following techniques which will be used to prep my dataset:

1. Checking for missing values: It is important to identify and handle any missing values in the dataset. This could involve dropping rows with too many missing values, imputing missing values with estimates or averages, or leaving the missing values as is depending on the situation.
2. Removing duplicates: If there are duplicate rows in the dataset, they should be removed to avoid bias or misleading results.
3. Standardising data formats: It is important to make sure that data is in a consistent format. For example, dates should be standardized to a single format, and categorical variables should use a consistent set of labels.
4. Handling outliers: Outliers can have a significant impact on statistical analysis, so it may be necessary to identify and handle outliers in the data. This could involve removing extreme values, capping values at a certain threshold, or transforming the data to make it more symmetrical.
5. Normalising data: Normalizing data can help to make different variables on the same scale, such as converting text to numbers where needed. This also includes lemmatizing, removing stopwords and datatype conversions.

### 7.1 Missing Data

One of the initial steps and perhaps the most important measure to undertake as part of cleaning the data is to check for missing values. Missing data can occur for a variety of reasons, such as errors in data collection or data entry, or because certain values were not applicable or not available.

With the below function we can check the total sum of null or empty values. As you can see this is 28. I can see in the dataframe that there were values in the year column that were not integers hence why the data type is displaying object in year. We can not convert this to an integer unless these values are removed.

There are a few approaches that can be taken when dealing with missing data. One option is to simply drop rows with missing values. This can be appropriate if the missing values are very few and do not represent a significant portion of the dataset. However, if a large number of rows are missing data, this approach may not be feasible as it could result in a significant loss of data.

Another option is to impute missing values with estimates or averages. For example, if a movie's rating is missing, it might be reasonable to fill in the missing value with the average rating of all movies in the dataset. However, this approach can be problematic if the missing values are not missing at random, as it could introduce bias into the data.

A third option is to leave the missing values as is and handle them during the analysis phase. This can be appropriate if the missing values are relatively few and do not significantly impact the analysis.

In this case as the missing values are few and will not skew the findings of my analysis i have decided to remove these values.

```
df['votes'] = df['votes'].fillna(df['votes'].mean())
df['rating'] = df['rating'].fillna(df['rating'].mode()[0])


# checking null values
df.isnull().sum().sum()
```

```
0
```

```
# dropping null values
df = df.dropna()


# checking null values
df.isnull().sum().sum()
```

```
0
```

```
# printing first 50 rows of dataframe
df.head(50)
```

| | Unnamed: 0 | title | genre | duration | rating | year | plots | |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | The Shawshank Redemption | Drama | 142 | 9.3 | 1994 | Over the course of several years, two convicts... | https:/ |
| **1** | 1 | The Godfather | Crime, Drama | 175 | 9.2 | 1972 | The aging patriarch of an organized crime dyna... | https:/ |
| **2** | 2 | The Dark Knight | Action, Crime, Drama | 152 | 9.0 | 2008 | When the menace known as the Joker wreaks havo... | https:/ |
| **3** | 3 | Schindler's List | Biography, Drama, History | 195 | 9.0 | 1993 | In German-occupied Poland during World War II,... | https:/ |
| **4** | 4 | The Godfather Part II | Crime, Drama | 202 | 9.0 | 1974 | The early life and career of Vito Corleone in ... | https:/ |
| **5** | 5 | 12 Angry Men | Crime, Drama | 96 | 9.0 | 1957 | The jury in a New York City murder trial is fr... | https:/ |
| **6** | 6 | The Lord of the Rings: The Return of the King | Action, Adventure, Drama | 201 | 9.0 | 2003 | Gandalf and Aragorn lead the World of Men agai... | https:/ |
| **7** | 7 | Pulp Fiction | Crime, Drama | 154 | 8.9 | 1994 | The lives of two mob hitmen, a boxer, a gangst... | https:// |
| **8** | 8 | 777 Charlie | Adventure, Comedy, Drama | 164 | 8.9 | 2022 | Dharma is stuck in a rut with his negative and... | https:/ |
| **9** | 9 | Inception | Action, Adventure, Sci-Fi | 148 | 8.8 | 2010 | A thief who steals corporate secrets through t... | https:/ |
| **10** | 10 | Fight Club | Drama | 139 | 8.8 | 1999 | An insomniac office worker and a devil-may-car... | https:/ |
| **11** | 11 | The Lord of the Rings: The Fellowship of the Ring | Action, Adventure, Drama | 178 | 8.8 | 2001 | A meek Hobbit from the Shire and eight compani... | https:/ |
| **12** | 12 | Forrest Gump | Drama, Romance | 142 | 8.8 | 1994 | The presidencies of Kennedy and Johnson, the V... | https:/ |
| **13** | 13 | Il buono, il brutto, il cattivo | Adventure, Western | 161 | 8.8 | 1966 | A bounty hunting scam joins two men in an unea... | https:/ |
| **14** | 14 | The Lord of the Rings: The Two Towers | Action, Adventure, Drama | 179 | 8.8 | 2002 | While Frodo and Sam edge closer to Mordor with... | https:/ |
| **15** | 15 | Jai Bhim | Crime, Drama, Mystery | 164 | 8.8 | 2021 | When a tribal man is arrested for a case of al... | https://w |
| | | | | | | | Based on the | |

life of Indian

**7.3 Stopword removal & Lemmatization**

Removing stopwords and lemmatization are important techniques for cleaning and preprocessing data for a few reasons.

First, stopword removal can help to reduce the size of the dataset and make it easier to analyze. By removing common words that are not meaningful for understanding the context or meaning of the text, you can focus on the more important words and concepts.

Second, lemmatization can help to reduce the dimensionality of the dataset and make it easier to analyze. By reducing words to their base form, or lemma, you can reduce the number of word forms that need to be considered, which can make the data more manageable.

Both techniques can be especially useful for the wordclouds analysis as it will allow me to identify the main theme or topic in the plots.

computer

Below is a step by step process of how the lemmatization and stopword function is working:

Load the stopword list and lemmatization library: The first step is to load the list of stopwords and the lemmatization library into your program. The stopword list is typically a pre-defined list of common words that are not useful for understanding the context or meaning of a text, such as "a," "and," "the," and "but." The lemmatization library is a tool that allows you to reduce words to their base form, or lemma.

Tokenize the text: The next step is to tokenize the text into individual words or tokens. This typically involves splitting the text on whitespace and punctuation, and possibly removing any remaining non-alphabetic characters.

Remove stopwords: Once the text is tokenized, the next step is to remove the stopwords. This can be done by iterating through the list of tokens and checking each one against the stopword list. If a token is a stopword, it is removed from the list.

Lemmatize the tokens: After the stopwords are removed, the next step is to lemmatize the remaining tokens. This can be done using the lemmatization library, which takes a token and its part of speech (e.g., noun, verb, adjective) as inputs and returns the lemma of the word.

Reassemble the text: After the stopwords are removed and the tokens are lemmatized, the final step is to reassemble the text from the modified list of tokens. This typically involves joining the tokens with whitespace and possibly adding any necessary punctuation.

```
# function to remove stop words
def stopwords_removal(text):

  text = tokenizer.tokenize(text.lower())
  text = [i for i in text if i not in stopwords]
  return " ".join(text)

# function to lemmatize the text
wordnet_lemmatizer = WordNetLemmatizer()
def lemmatize(text):

  text = tokenizer.tokenize(text.lower())
  text = [wordnet_lemmatizer.lemmatize(i) for i in text]
  return " ".join(text)
```

```
# cleaning gross values as gross is represented as text with M as million and dollar sign.
def clean_gross(text):
  if(len(text)>3):
    return text[1:-1]
  else:
     return text


# calliong text cleaning, stop word removal and lemmatization functions
df["plots"] = df["plots"].apply(stopwords_removal)
df["plots"] = df["plots"].apply(lemmatize)

df["gross"] = df["gross"].apply(clean_gross)
df.head()
```

| Unnamed: 0 | title | genre | duration | rating | year | plots |
|---|---|---|---|---|---|---|

**7.4 Categorising data types**

The

Categorising data types involves identifying the data type of each column in a dataset and then converting any datatype that may need to be converted for an analysis to be conducted at a later stage. For example, gross would need to be converted from an object to an integer.

The function df.dtypes in pandas below first identifies the data types of each column in the dataset and will return the data type of each column in the dataframe.

I then convert data types as necessary. For example, converting a column of year data that is currently an object data type into an integer or string data type. Below i am using the function df['column'].astype(int) in pandas, which will convert the data in the column to an integer data type and the same for converting gross from an object to a float datatype.

known

```
# converting datatype
df['year'] = pd.to_numeric(df['year'], errors='coerce')
```

```
df.dtypes
```

```
Unnamed: 0      int64
title          object
genre          object
duration        int64
rating        float64
year          float64
plots          object
url            object
director       object
cast           object
votes           int64
gross          object
dtype: object
```

```
# df['year'] = df['year'].astype(int)
df['gross'] = df['gross'].astype(float)
```

As you can see below the datatypes have been converted into its appropriate datatype.

```
# checking datatypes
df.dtypes
```

```
Unnamed: 0      int64
title          object
genre          object
duration        int64
rating        float64
year          float64
plots          object
url            object
director       object
cast           object
votes           int64
gross         float64
dtype: object
```

```
# topreview_df=df.query("rating < 8")
```

# 8 Exploratory data analysis

## 8.1 Pie Chart

There are several benefits to getting the different types of movie genres with this dataset. First, it allows for a more granular analysis of the data, as you can examine how specific genres are performing rather than just looking at the data as a whole. This can help identify trends or patterns within specific genres that may not be apparent when looking at the data as a whole. Additionally, presenting the data in a pie chart allows for a visual representation of the proportion of movies within each genre, making it easier to compare and contrast the different genres. This can be useful for identifying which genres are the most popular or successful, as well as for identifying any imbalances or discrepancies in the data.

```
# Initialize an empty list
cat= []
```

```
# Extract the 'genre' column
genre_list = df['genre']

# Split the genre string at each comma and store the resulting list of genres in genre_list
genre_list= [x.split(',') for x in genre_list]

# Iterate through the list of genres, and add each genre to the 'cat' list
for i in genre_list:
  cat.extend(i)

# Strip whitespace from each genre in the 'cat' list
cat = [x.strip() for x in cat]

# printing uniques genre
print(set(cat))
```

```
{'Comedy', 'Animation', 'Family', 'Crime', 'Adventure', 'War', 'Fantasy', 'Thriller', 'Horror', 'Drama', 'History', 'Action', 'Musi
```

There are several benefits to getting the different types of movie genres with this dataset. First, it allows for a more granular analysis of the data, as you can examine how specific genres are performing rather than just looking at the data as a whole. This can help identify trends or patterns within specific genres that may not be apparent when looking at the data as a whole. Additionally, presenting the data in a pie chart allows for a visual representation of the proportion of movies within each genre, making it easier to compare and contrast the different genres. This can be useful for identifying which genres are the most popular or successful, as well as for identifying any imbalances or discrepancies in the data.

The pie chart shows the distribution of movie genres. From the chart, it is clear that drama is the largest category, followed by comedy, adventure, action, and crime. This suggests that drama movies are the most popular or successful among the top 1000 movies on IMDb.

There are several reasons why drama movies may be more successful than other genres. One reason is that drama movies often have more complex, multi-layered plots and characters, which can engage and resonate with audiences on a deeper level. Additionally, drama movies often tackle real-world issues or explore deeper themes, which can make them more relevant and thought-provoking for audiences.

```
# counter function to get the count for each genre
genre_count = Counter(cat)
labels = []
sizes = []
fig1, ax1 = plt.subplots(figsize=(12, 12))
for x, y in genre_count.items():
    labels.append(x)
    sizes.append(y)

# Plot
plt.pie(sizes, labels=labels)

plt.axis('equal')
plt.show()
```

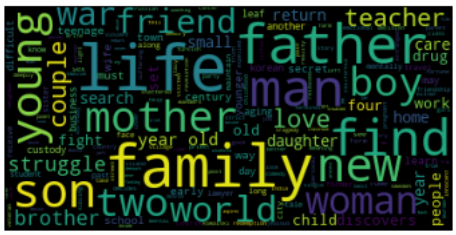

In order to gain a better understanding of the characteristics and themes associated with the movie plots a wordcloud would be a suitable form of visual representation. By using the word cloud we can identify any common theme if any, which may appeal to an audience.

## 8.2 Wordclouds

With the use of wordclouds we can take an indepth look at a genre like Drama for example. Given that its the most popular genre, we can analyse if there is a common theme in plots which could be a contributing factor for the genres dominance in the top 1000 list. By examining the frequency and placement of these words in the wordcloud, we can gain a sense of which themes are the most prominent in the genre and how they may be related to movie success.

Below you can see the wordcloud contains words like "family," "life," and "father "it suggests that these themes are common in drama movies and may be contributing factors.

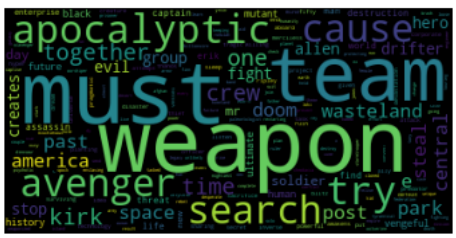
```
# creating wordcloud for movie descriptions
drama_df=df.query("genre == 'Drama'")
wordcloud2 = WordCloud().generate(' '.join(drama_df['plots']))
# displaying wordcloud
plt.imshow(wordcloud2)
plt.axis("off")
plt.show()
```




The wordcloud for the genres (Action, Adventure and Sci-Fi) contains words like "weapon" and "team,". The presence of the word "weapon" in the wordcloud could indicate that action movies often involve the use of weapons or violence as a central element of the plot. The presence of the word "apocalyptic" could indicate that adventure movies often involve scenarios or plotlines that are set in a post-apocalyptic or a disaster-stricken world. The presence of the word "space" could indicate that sci-fi movies often involve stories set in outer space or that involve extraterrestrial beings or technology.

By examining the specific words and themes that are prevalent in the wordclouds, it has given a better insight to identify patterns and trends that may be influencing the popularity and success of movies within the genre. However, there are many other factors that overall equate to success including duration, cast, rating. I will be taking a look at this further through other visual representations.

```
# creating wordcloud for 'Action, Adventure, Sci-Fi' movies
drama_df=df.query("genre == 'Action, Adventure, Sci-Fi'")

wordcloud2 = WordCloud().generate(' '.join(drama_df['plots']))
# displaying wordcloud
plt.imshow(wordcloud2)
plt.axis("off")
plt.show()
```




## 8.3 Boxplot

A box plot (also called a box and whisker plot) is a graphical representation of statistical data based on the minimum, first quartile, median, third quartile, and maximum. It is often used to display the distribution of numerical data.

When analysing the box plot of duration and year, it is evident that the distribution shifts over time, with an upward trend. From 1920 to 2020, it can be observed that the median value of movie duration has increased. This suggests that movies have become longer over time. Additionally, the interquartile range (IQR) of the box plot also appears to have increased, as the size of the box has on average grown. This indicates that the range of movie durations has also increased over time.

Moreover, the width of the whiskers also shows that the spread of the data has increased over time, which means that there are more outliers, and the data is more dispersed.

In summary, movies have become longer over time, and the range of movie durations has also increased. This could indicate that movies have become more diverse in terms of duration, and that movie-goers are interested in different types of movies, not just the traditional '90 minutes' format.

```
# Convert the 'year' column to a list
year = df.year.tolist()

# Round the year values to the nearest decade
year = [round((int(x)/10))*10 for x in year]

# Add a new column 'decade' to the dataframe
df["decade"] = year
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-42-61790b1407d3> in <module>
      3
      4 # Round the year values to the nearest decade
----> 5 year = [round((int(x)/10))*10 for x in year]
      6
      7 # Add a new column 'decade' to the dataframe

<ipython-input-42-61790b1407d3> in <listcomp>(.0)
      3
      4 # Round the year values to the nearest decade
----> 5 year = [round((int(x)/10))*10 for x in year]
      6
      7 # Add a new column 'decade' to the dataframe

ValueError: cannot convert float NaN to integer
```

SEARCH STACK OVERFLOW

```
# Create a box plot of the 'duration' column by 'decade'
df.boxplot(column='duration',by='decade')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-43-984ee5e9db5f> in <module>
      1 # Create a box plot of the 'duration' column by 'decade'
----> 2 df.boxplot(column='duration',by='decade')

                                  6 frames
/usr/local/lib/python3.8/dist-packages/pandas/core/groupby/grouper.py in
get_grouper(obj, key, axis, level, sort, observed, mutated, validate, dropna)
    860                 in_axis, level, gpr = False, gpr, None
    861             else:
--> 862                 raise KeyError(gpr)
    863         elif isinstance(gpr, Grouper) and gpr.key is not None:
    864             # Add key to exclusions

KeyError: 'decade'
```

SEARCH STACK OVERFLOW

```
df.boxplot(column='rating',by='decade')
```

## 8.4 Line Plots

In order to get a better understanding of the correlation between two or more variables line plots are great way to visualise this. They also will give us a quick overall pattern in the data and identify any outliers or anomalies.

To get some key insights i have started by creating 2 line plots. I would like to assess whether the duration of a film over time has adjusted course with a transition in modern day audiences, who usually now prefer a lower duration.

To my surprise the line plot below shows the duration of a film on average has increased year by year. My initial thoughts would have been as technology advances and with development of new means of viewing content on the internet for example streaming or youtube audience average watch time would have reduced over the years as it much harder to retain audiences with content being delivered in shorts or small clips.

```
# rating chart with years
axes = plt.gca()
df_sort = df.sort_values(by=['year'])
df_sort.plot(kind='line', x='year', y='duration', ax=axes);
```

The line plot depicts a general upward trend in gross revenue for films over time. This suggests that, on average, films are earning more money year over year. As the lineplot is clustered together the analysis is a little difficult to make as to whether or not the trend is continuing. For this and to make the overall analysis accurate i will be constructing further graphical representations. I will also be redoing these graphs so that the data is not clustered and readibility is much easier.

In the context of assessing the success of movies, creating line plots can help you understand how different factors, such as the total number of productions, total gross collection, average runtime, and votes, have changed over the years.

For example, if you create a line plot showing the total number of productions by year, you can see how the number of movies being produced has changed over time. This can help you understand whether the film industry is growing or shrinking, and can also give you insights into factors that might be influencing the number of productions, such as changes in technology or shifts in cultural tastes.

Similarly, if you create a line plot showing the total gross collection by year, you can see how the financial success of movies has changed over time. This can help you understand whether the film industry is thriving or struggling financially, and can also give you insights into factors that might be influencing the gross collection, such as changes in ticket prices or shifts in audience preferences.

Creating line plots showing the average runtime by year and votes by year can also be useful for understanding trends in the film industry. For example, if the average runtime of movies is increasing over time, it could suggest that filmmakers are choosing to tell more complex and detailed stories. If the number of votes for movies is increasing over time, it could suggest that the film industry is becoming more popular or that people are more engaged with the films being produced.

Overall, creating line plots to assess trends in the film industry can be a useful way to understand how different factors are changing over time and can provide valuable insights into the health and success of the industry.

```
# rating chart with years
axes = plt.gca()
df_sort = df.sort_values(by=['year'])
df_sort.plot(kind='line', x='year', y='gross', ax=axes);
```

Firstly, as we can see in the lineplots as the data is no longer clustered we can clearly now see and measure gross and duration over the years.

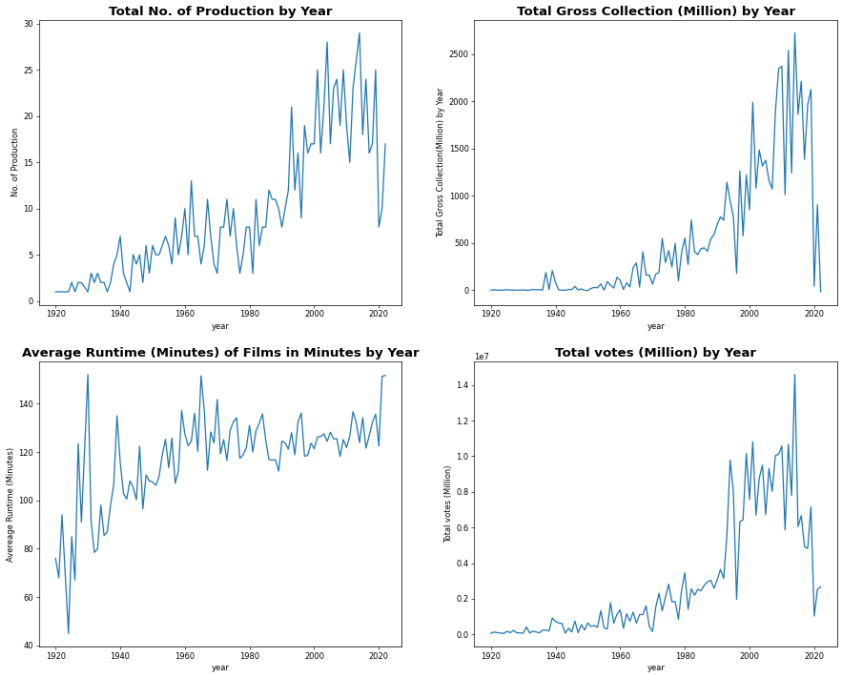
The lineplot shows total number of productions by year increasing with an upward trend. This trend can also be seen in the gross revenue of top 1000 films, with a steady yearly increase visible. The dip in total number of productions and gross collections in 2020 may be due to the pandemic causing production to pause or be delayed. Many film productions were shut down or had to be postponed due to safety concerns and social distancing measures. This could have led to a decrease in the number of productions in 2020. A high majority of film theatres and multiplexes across the globe had to also run at a reduced capacity, leading to fewer ticket sales and lower gross income. The pandemic may also have led to a shift in audience behavior, with more people opting to stay at home and watch movies online rather than going to theaters. The dip in votes in 2020 may also be due to the pandemic causing a reduction in the number of movies being released or making it harder for people to see and vote on movies.

We can however see a constant upward trend in average runtime suggesting that filmmakers are choosing to tell more complex and detailed stories that require longer running times which may have been do a change in audience behaviour.

```
# Add a new column 'Count' with value 1 for every row
df['Count']=1
# number of films by year
df_year_count=df.groupby('year').count()
# Gross collection by year
df_year_gross=df.groupby(['year'])['gross'].sum()
# Average runtime in minutes by year
df_year_ave_runt=round(df.groupby('year')['duration'].mean(),2)
# no. of votes by year
df_year_votes=df.groupby('year')['votes'].sum()

# Create a 2x2 grid of subplots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(figsize=(18,14.5), dpi=60, nrows=2, ncols=2)

# Plot the number of films,gross,runtime and votes by year on the first subplot
sns.lineplot(data=df_year_count['Count'],ax=ax1)
sns.lineplot(data=df_year_gross,ax=ax2)
sns.lineplot(data=df_year_ave_runt,ax=ax3)
sns.lineplot(data=df_year_votes,ax=ax4)

# Add titles to each subplot
ax1.set_title('Total No. of Production by Year', fontsize=16, fontweight ='bold')
ax2.set_title('Total Gross Collection (Million) by Year', fontsize=16, fontweight ='bold')
ax3.set_title('Average Runtime (Minutes) of Films in Minutes by Year', fontsize=16, fontweight ='bold')
ax4.set_title('Total votes (Million) by Year', fontsize=16, fontweight ='bold')
```

```
# Add y-axis labels to each subplot
ax1.set_ylabel('No. of Production')
ax2.set_ylabel('Total Gross Collection(Million) by Year')
ax3.set_ylabel('Avereage Runtime (Minutes)')
ax4.set_ylabel('Total votes (Million)')
```

```
Text(0, 0.5, 'Total votes (Million)')
```




## 8.5 Linear Regression

Linear regression is a statistical technique that can be used to model the relationship between a dependent variable and one or more independent variables. In the context of the IMDb top 1000 dataset, linear regression could be a useful means of analysis for understanding the relationship between different variables and how they influence the ranking of movies on the list.

For example, in this context i would use linear regression to examine the relationship between the rating of a movie and its gross collection, rating, or number of votes, rating and runtime. This could help me understand whether there is a strong relationship between these variables.

Linear regression can provide valuable insights into the factors that influence the ranking of movies on the list. For example, I do have a strong predicament that movies with higher ratings tend to have higher gross collections, longer runtimes, or more votes, suggesting that these factors contribute to the success of a movie. On the other hand, I might find that there is no strong relationship between these variables and the ranking of a movie, suggesting that other factors may be more important in determining a movie's success.

Overall, linear regression can be a useful means of analysis for understanding the relationship between different variables and how they influence the ranking of movies on the IMDb top 1000 list.

The results of the linear regression show that there is a weak positive correlation between rating and number of votes, as indicated by the correlation coefficient of 0.243. This means that there is a slight relationship between these two variables, but it is not a strong relationship.

A weak correlation between rating and number of votes suggests that the number of votes a movie receives is not a strong predictor of its rating. There may be other factors that have a greater influence on a movie's rating, such as the quality of the movie or the preferences of the voters.

The results also show that there is almost no correlation between rating and runtime and rating and gross collection. This means that there is no relationship between these variables and a movie's rating. This could suggest that the runtime and gross collection of a movie do not have a significant impact on its rating.

Overall, the results of the linear regression suggest that the rating of a movie is not strongly influenced by the number of votes it receives, its runtime, or its gross collection. Other factors may have a greater impact on a movie's rating.

```
# compute r^2 values

y=df['rating']
# fitting LR for different features to check the relation with movie rating
lin_reg_1=LinearRegression().fit(df[['duration']], y)
lin_reg_2=LinearRegression().fit(df[['votes']], y)
lin_reg_3=LinearRegression().fit(df[['gross']], y)

# computing correlation of different features with movie rating
r2_1=lin_reg_1.score(df[['duration']], y)
r2_2=lin_reg_2.score(df[['votes']], y)
r2_3=lin_reg_3.score(df[['gross']], y)

# printing correlation of different features with movie rating
print(f'r^2 of Rating VS Runtime in Minutes = {round(r2_1,3)}.')
print(f'r^2 of Rating VS No. of Votes = {round(r2_2,3)}.')
print(f'r^2 of Rating VS Gross_Collection(Million) = {round(r2_3,3)}.')
```

```
r^2 of Rating VS Runtime in Minutes = 0.075.
r^2 of Rating VS No. of Votes = 0.238.
r^2 of Rating VS Gross_Collection(Million) = 0.008.
```

## 8.6 Bar charts

Bar charts are a type of graphical representation that can be useful for understanding the distribution and frequency of different categories of data. In the context of the IMDb top 1000 dataset, bar charts can be a useful means of analysis for understanding the number of movies in the list and the directors, actors, and years they were released.

I will be creating a bar chart with the number of movies in the list and their directors. I hope to see a pattern with the popularity and success of different directors, as well as the types of movies they have directed. I will also use this bar chart to compare the number of movies on the list for different directors and see which directors have had the most success.

Secondly, I will create a bar chart with actors and the number of movies they have on the list. This will give me insights into the popularity and success of different actors, as well as the types of movies they have appeared in. I will also be using this bar chart to compare the number of movies on the list for different actors and see which actors have had the most success and whether an A+ cast would make a difference to movie success.

Finally, although the linear plot did show us a trend in the number of films released and the years, the visual representation did not show us individually each year. A bar chart would be a better representation for this so that i can see whether there is any forecasted trend emerging from the number of movies being released on a year to year basis.

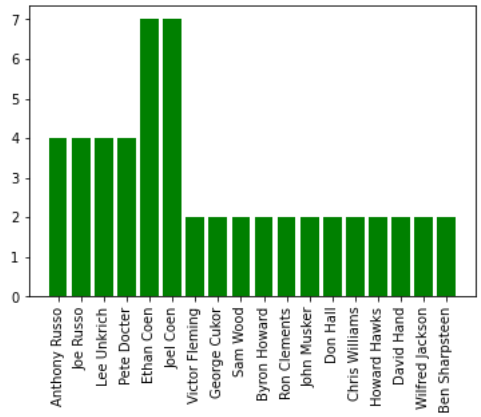
```
# getting director list
directors = df["director"].tolist()
directors = [x.replace("Directors:","") for x in directors if "Directors:" in x ]
# getting individual directore list
directors  = [x.split(",") for x in directors if "," in x]
directors = list(itertools.chain(*directors))
directors = [x.strip() for x in directors]
c = Counter()
# counting movies for each director
x = Counter(directors)
x = dict(x)

dir=[]
num_of_movies=[]
for i in x.keys():
  if x[i] > 1:
    dir.append(i)
    num_of_movies.append(x[i])
```

The bar chart results show that Ethan Coen and Joel Coen are among the directors with the most movies on the IMDb top 1000 list, with a total of 7 movies each. Anthony Russo, Joe Russo, and Lee Unkrich also have a significant number of movies on the list, with 4 movies each.

This suggests that Ethan Coen, Joel Coen, Anthony Russo, Joe Russo, and Lee Unkrich are successful directors who have had a number of popular and highly rated movies. It could be that these directors have a particular style or approach to filmmaking that resonates with audiences and critics, leading to the success of their movies.

It is important to consider the limitations of the data and to examine the full dataset and other relevant factors when trying to understand the factors that contribute to the success of a director. For example, it could be that Ethan Coen, Joel Coen, Anthony Russo, Joe Russo, and Lee Unkrich have had the opportunity to work with talented actors, writers, and other members of the crew, which could have contributed to the success of their movies. In order to get a better insight i will be seeing which actor has had the most amount of movies in the dataset next.

```
# displaying barchart of number of movies for each direto
plt.bar(dir, num_of_movies, color='g')
plt.xticks(rotation = 90)
plt.show()
```




The bar chart results show that Robert De Niro, Tom Hanks, and Al Pacino are among the top casted actors with the most movies on the IMDb top 1000 list. Specifically, Robert De Niro has 16 movies on the list, Tom Hanks has 13 movies on the list, and Al Pacino has 12 movies on the list.

From the result although we are able to deduce the actors with most movies in the top 1000, it is important to see whether there is correlation with directors and actors working together in a particular movie. This would help us to identify which directors are likely to more successful films and the actors who will be crucial for their success.

```
# getting list of cast for each movie
cast = df["cast"]
cast = [x[1:-1].replace("'","").split(",") for x in cast]
cast = list(itertools.chain(*cast))
cast = [x.strip() for x in cast]

# counting number of movies for each star
c = Counter()
x = Counter(cast)
x = dict(x)

top_cast=[]
num_of_movies=[]
for i in x.keys():
  if x[i] > 5:
    top_cast.append(i)
    num_of_movies.append(x[i])


len(top_cast)
```

```
53
```

```
# displaying number of movies for each star
figure(figsize=(12, 4), dpi=80)
plt.bar(top_cast, num_of_movies, color='g')
plt.xticks(rotation = 90)
plt.show()
```

The bar chart results show that the number of films being released has fluctuated over time, with 1994 and 1996 having the highest number of films being released and a downward trend after 2019. The results also suggest that there has been no stable trend in the number of films being released since the 2000s.

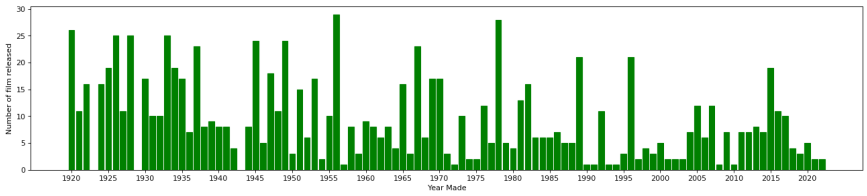
One possible explanation for the high number of films being released in 1994 and 1996 is that these years were particularly successful or popular for the film industry. It could be that a combination of factors, such as successful movies, changes in technology or distribution platforms, or shifts in cultural tastes, contributed to the high number of films being released in these years.

The fluctuation in the number of films being released since the 2000s could be due to a variety of factors, such as changes in the economy, shifts in audience preferences, or the introduction of new distribution platforms. It is difficult to pinpoint a specific cause for the fluctuation without more information.

The downward trend in the number of films being released after 2019 could be due to the impact of the COVID-19 pandemic on the film industry. The pandemic has caused disruptions to production schedules, led to theater closures and reduced capacity, and influenced audience behavior, all of which could have contributed to the downward trend in the number of films being released.

It would be interesting to see whether the dip is likely to be reversed as countries start to ease restrictions and the industry starts to move back towards normalcy.

```
# displaying year wise number of movies
film_release = df.groupby('year')['year'].agg('count')
year = df['year'].unique()

figure(figsize=(20, 4), dpi=80)
plt.bar(year[:-1],film_release, color='g',  edgecolor='g')
plt.xticks(np.arange(min(year), max(year), 5.0))
plt.xlabel("Year Made")
plt.ylabel("Number of film released")
plt.show()
```



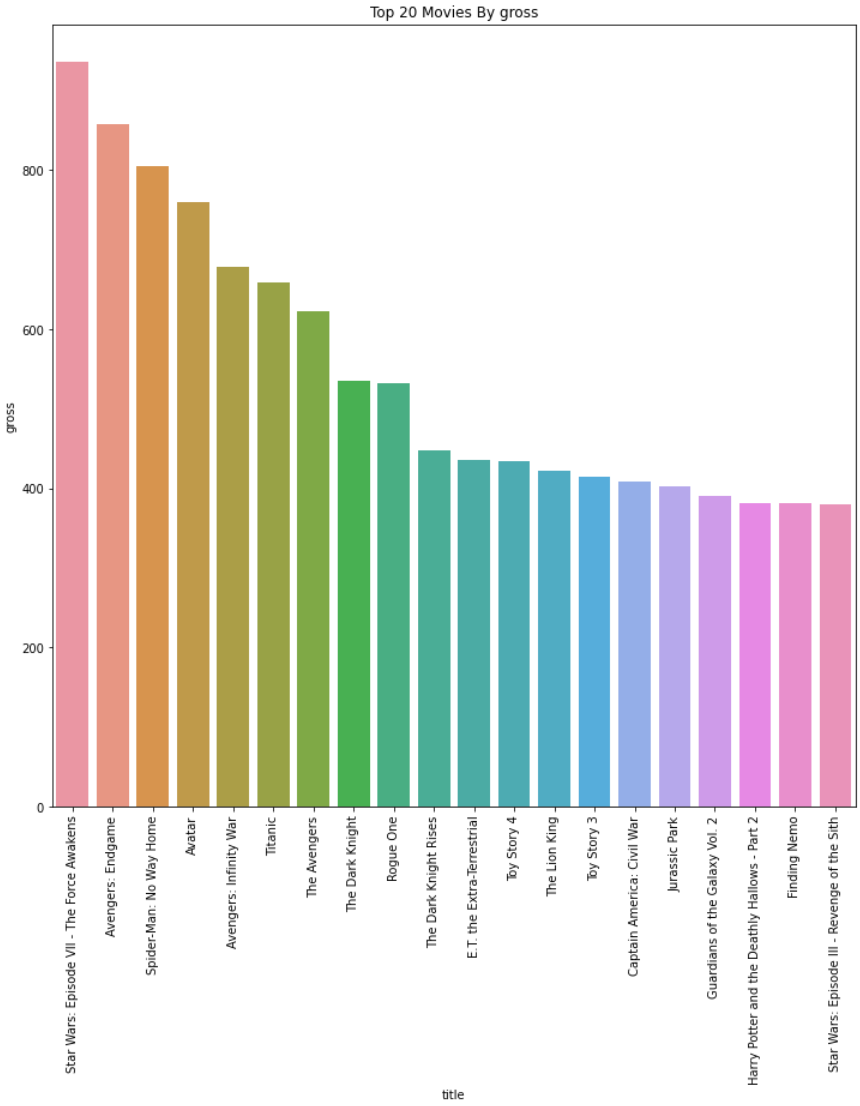

The bar chart results show that the highest grossing films are "Star Wars: The Force Awakens," "Avengers: Endgame," "Spider-Man," and "Avatar." These movies are all in the action, adventure, and sci-fi genres.

The results also show that 7 of the highest grossing films are in the superhero genre, specifically Marvel and DC. This suggests that superhero movies are also particularly popular and successful and that Marvel and DC have had success in this genre.

One possible explanation for the success of action, adventure, sci-fi, and superhero movies is that these genres tend to appeal to a wide audience, including both male and female viewers. These genres also often feature exciting and thrilling storylines and special effects, which can be appealing to audiences.

Another factor that may contribute to the success of these movies is the strength of the characters and their appeal to audiences. Many of the highest grossing movies feature iconic characters with compelling storylines and strong performances by the actors.

```
# top movies with gross income
Top = df[['title', 'gross']].sort_values(by="gross",ascending=False)
Top = Top.head(20)
Top.head(10)
plt.figure(figsize=(12,12))
graph=sns.barplot(y='gross',x='title',data=Top)
graph.set_title('Top 20 Movies By gross')
plt.xticks(rotation=90)
plt.show()
```




The bar chart provided offers a visual representation of various aspects of the IMDB Top 1000 movies. By analyzing this chart, we can gain a deeper understanding of the distribution of certain features within the dataset.

Firstly, the chart indicates that the majority of movies within the dataset fall within the duration range of 90-150 minutes. This suggests that this range is a prevalent duration among the top 1000 movies on IMDB. Additionally, it allows us to infer that this duration range may be considered as an ideal length for a movie.

Furthermore, the chart also illustrates the average rating of the movies in the dataset, which is around 7.75 to 8. This implies that the majority of movies in the dataset are well-received by viewers.

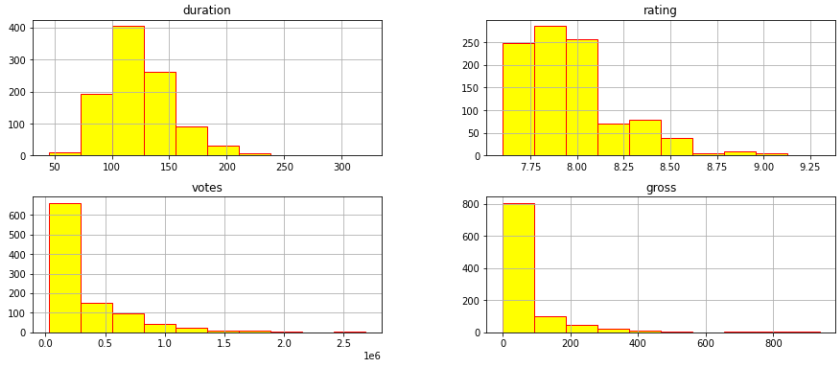
Moreover, the chart reveals that the average frequency of votes for the movies is relatively low. This could be an indication that a limited number of viewers have contributed to the ratings of the movies in the dataset.

Lastly, the chart shows the average gross revenue of the movies in the dataset, which ranges from 0 to 100. This indicates that the top 1000 movies on IMDB may not always be the most financially successful. This insight can be valuable for film industry professionals and investors who are interested in understanding the correlation between the rating of a movie and its commercial performance.

```
df.corr()
```

| | Unnamed: 0 | duration | rating | year | votes | gross | Count |
|---|---|---|---|---|---|---|---|
| **Unnamed: 0** | 1.00 | -0.27 | -0.94 | 0.08 | -0.41 | -0.09 | NaN |
| **duration** | -0.27 | 1.00 | 0.27 | 0.22 | 0.16 | 0.12 | NaN |
| **rating** | -0.94 | 0.27 | 1.00 | -0.08 | 0.49 | 0.09 | NaN |
| **year** | 0.08 | 0.22 | -0.08 | 1.00 | 0.24 | 0.23 | NaN |
| **votes** | -0.41 | 0.16 | 0.49 | 0.24 | 1.00 | 0.61 | NaN |
| **gross** | -0.09 | 0.12 | 0.09 | 0.23 | 0.61 | 1.00 | NaN |
| **Count** | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

```
numerical_attributes = ['duration', 'rating', 'votes','gross']
df[numerical_attributes].hist(figsize = (15, 6), color = 'yellow', edgecolor = 'red', layout = (2, 2));
```




## 8.7 FacetGrid

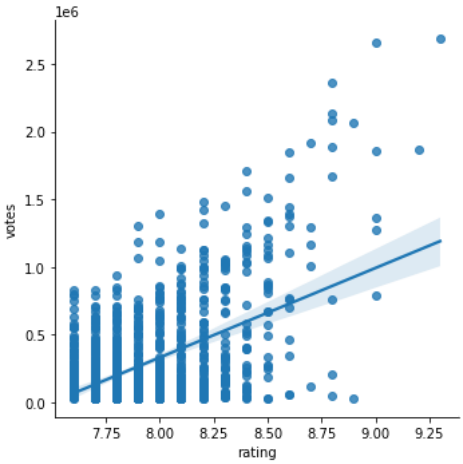
FacetGrid is a useful tool in data visualization because it allows you to easily create subplots and compare multiple variables within a dataset. In addition to the line plots and bar graphs this can be useful to identify trends.

Below are some insights that i will be hoping to gain:

- By displaying rating and votes, you can see how the number of votes a movie receives is related to its rating. I may find that movies with higher ratings tend to have more votes, or that there is no clear relationship between the two variables.
- By displaying year and votes, you can see how the number of votes a movie receives changes over time. This can help me understand how the popularity of a movie changes as it ages.
- By displaying year and rating, you can see how the ratings of movies change over time. This can help me understand if there is a trend in the quality of movies released in different years.
- By displaying gross and rating, you can see if there is a relationship between a movie's box office gross and its rating. I may find that highly rated movies tend to have higher grosses, or that there is no clear relationship between the two variables.
- By displaying duration and rating, you can see if there is a relationship between a movie's runtime and its rating. I may find that shorter or longer movies tend to have higher ratings, or that there is no clear relationship between the two variables.

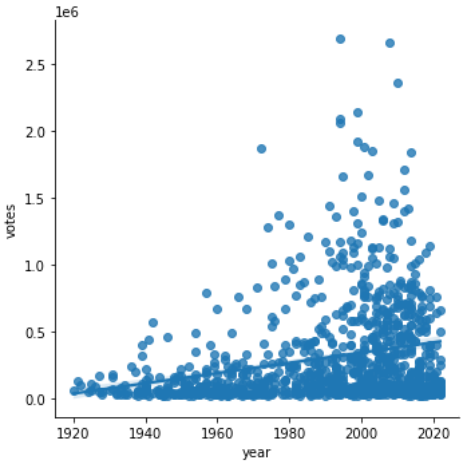
Below i can see a positive correlation between rating and number of votes suggesting that movies with higher ratings tend to have more votes.

```
sns.lmplot(x="rating", y="votes", data=df);
```



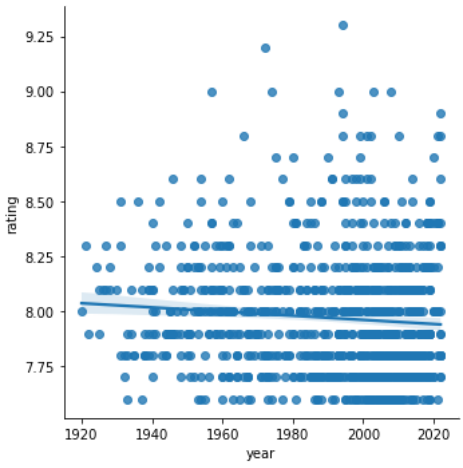

Below although it is not completely clear whether there is an emerging trend with the year and number of votes we can see a cluster of higher votes as the years progress. This may be due to easy access to internet and devices allowing more users to participate as supposed to the 1900's.

```
sns.lmplot(x="year", y="votes", data=df);
```



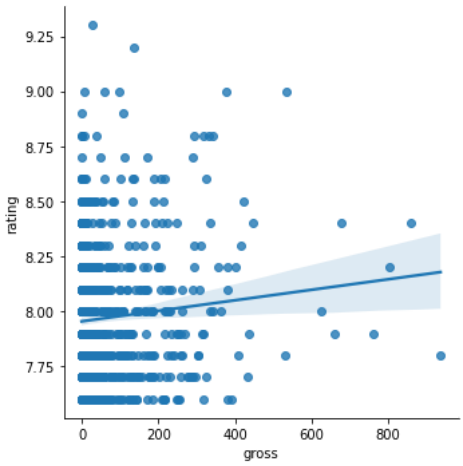

Below we can see a weak negative correlation between years and ratings. This means that there has been a decline in ratings as the years progress.

```
sns.lmplot(x="year", y="rating", data=df);
```



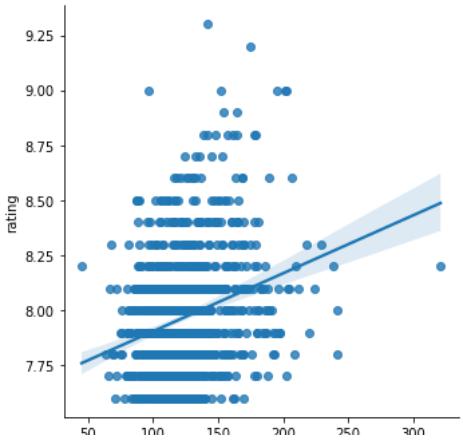

In the grid below we can see a weak positive correlation between gross and rating. Which means that the rating of a film does not entirely impact the gross of a film. We can also use the graph to see majority of the highest grossing films are not very highly ranked compared to other films which are. For example the highest rated film did not gross much yet it gained a rating above 9.25

```
sns.lmplot(x="gross", y="rating", data=df);
```




In the grid below we can see a weak positive correlation between duration and rating. This also gives an approximate avergae of duration at a glance along with its ratings.

```
sns.lmplot(x="duration", y="rating", data=df);
```

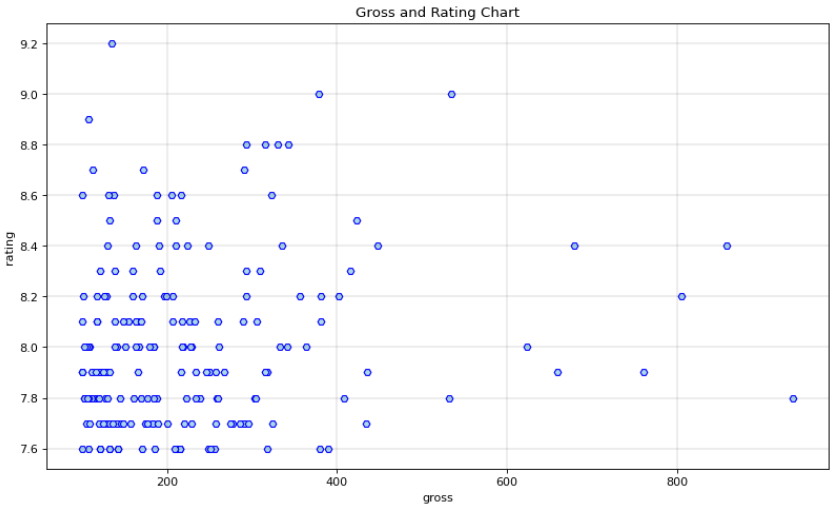



## 8.8 Scatter Plot

A scatter plot is a visualization tool that uses Cartesian coordinates to display values for two variables for a set of data. Each data point in the set is represented by a dot plotted on the graph, with the position of the dot determined by the values of the two variables. Scatter plots are useful for visualizing the relationship between two continuous variables and identifying trends or patterns in the data.

Below, i have created another visual form to see the correlation between gross and rating. We can clearly see there is no correlation between gross and rating. This indicates that rating has little or no impact on a movies gross. We can still use this visual form as an indicator that high grossing films do not necessarily have high ratings and high rated films would not necessarily have a higher gross.

```
# Create a new figure
plt.figure(num=None, figsize=(12,7), dpi=80)
movie=df[df.gross>100]
plt.scatter(movie['gross'], movie['rating'],marker ="H",edgecolors = 'b',facecolor='lightblue') #Marker= hexagon, Edgecolor=Blue, fillcol

# Create a scatter plot of gross revenue and rating
plt.xlabel("gross") #Labelling x
plt.ylabel("rating") #Labelling y
plt.title("Gross and Rating Chart")

# Add grid lines
plt.grid(color='black', linestyle='-', linewidth=0.25, alpha=0.5)  ##adds major gridlines
plt.show()
```




## 8.9 Machine Learning Model - Prediction of Gross

As an additional component of my research, my objective was to construct a predictive model capable of forecasting movie gross revenue based on relevant features.

I trained an XGBRegressor model to predict the gross of a movie based on features such as votes, duration, year, and rating. The dataset used was the IMDB Top 1000 movies and was split into 85% training and 15% testing data.

The results of the model were promising, with a testing accuracy of 0.73. This suggests that the XGBRegressor model is an effective method for predicting the gross of a movie. However, this is only a preliminary analysis and further research could be conducted to test the model's performance on larger and more diverse datasets. Additionally, exploring other features that could be used to improve the model's accuracy could also be a valuable next step in this research. Overall, this model shows great potential for predicting the gross of movies and could have practical applications for the film industry. Further research would be required in this domain.

```
# Splitting data into training and test set
X_train, X_test, y_train, y_test = train_test_split(df,df["gross"],test_size=0.15)


# Define the columns to consider
cols_to_std = ['votes','duration','year','rating']

# Create a StandardScaler object and fit the scaler on training data
scaler=StandardScaler()
scaler.fit(X_train[cols_to_std])

# Apply the scaler on the training data and testing data
X_train[cols_to_std] = scaler.transform(X_train[cols_to_std])
X_test[cols_to_std] = scaler.transform(X_test[cols_to_std])


# Create the xgboost model
model = XGBRegressor()

# Train the model
model.fit(X_train[cols_to_std], y_train)

# Make predictions on the test data
Y_pred = model.predict(X_test[cols_to_std])

# Evaluate the model's performance on the training data
score = model.score(X_train[cols_to_std], y_train)
print('Training Score:', score)

# Evaluate the model's performance on the test data
score = model.score(X_test[cols_to_std], y_test)
print('Testing Score:', score)

# Create a DataFrame with the predicted values
output = pd.DataFrame({'Predicted':Y_pred})
```

```
[12:43:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Training Score: 0.8344575853396136
Testing Score: 0.15116182180050142
```

# 9 Summary

### 9.1 Conclusions

In conclusion, this comprehensive study of the IMDB Top 1000 movies dataset has given us a few beneficial insights regarding the characteristics of the movies included in this list.

One of the key insights from the study was that certain genres were more popular among the top 1000 movies. For example, drama and comedy were the two most popular genres, with a large number of movies falling into one of these categories. This trend could be attributed to the fact that these genres tend to be more universally appealing, and are therefore able to attract a wider audience.

Another key insight from the study was the prevalence of certain themes within the plots of the top 1000 movies. For example, a significant number of movies in the top 1000 list were found to have themes of love, family, and friendship. This trend could be due to the fact that these themes are also universal and relatable to a wide audience.

In terms of the directors and actors who had the most representation in the top 1000, we found that a handful of individuals had multiple movies included in the list. For example, Ethan Coen and Joel had a total of 7 movies each. Anthony Russo, Joe Russo, and Lee Unkrich also have a significant number of movies on the list, with 4 movies each. In terms of actors, Robert De Niro had 16 movies on the list, Tom Hanks had 13 movies on the list, and Al Pacino had 12 movies on the list.

In terms of the correlation between different variables, the study found that there was no significant correlation between rating and gross or rating and votes. This suggests that these variables are not necessarily linked, and that the success of a movie cannot be accurately predicted based solely on these factors.

Furthermore, the study also revealed the potential for machine learning models to contribute towards movie success. The XGBRegressor model that was created to predict the gross of a film could serve as a foundation for further research in this field. By using this model as a starting point, it would be possible to conduct more in-depth analysis on other factors that may influence a movie's performance at the box office, such as plot, cast, year, competition, duration, and rating. This could lead to the development of more advanced models that are capable of providing even more accurate predictions of a movie's gross revenue.

Overall, the project "Analysing IMDB Top 1000" has provided a wealth of insights into the top movies listed on IMDB, and has shed light on the genres, themes, and individuals that are most prevalent in this list.

# 10 References and resources

**10.1 References**

https://www.imdb.com/search/title/?count=100&groups=top_1000&sort=user_rating

**10.2 Resources used**

GeeksforGeeks. (2021, May 31). Removing stop words with NLTK in Python [Online]. Available: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

Webscraping lecture and lab, Dr Sean McGrath

TutorialsPoint. Stemming & Lemmatization [Online]. Available: https://www.tutorialspoint.com/natural_language_toolkit/natural_language_toolkit_stemming_lemmatization.htm

K. Pykes. (2021, Mar. 17). A Guide To Cleaning Text in Python [Online]. Available: https://towardsdatascience.com/a-guide-to-cleaning-text-in-python-943356ac86ca

https://www.geeksforgeeks.org/bar-plot-in-matplotlib/

https://www.kaggle.com/code/anandakshay44/linear-regression-analysis-of-imdb-dataset

https://towardsdatascience.com/simple-linear-regression-in-python-numpy-only-130a988c0212

https://thispointer.com/pandas-tutorial-part-10-add-remove-modify-dataframe/