Final Project Submission

Please fill out:

- Student name: WAFULA SIMIYU
- Student pace: self paced / part time / full time - PART-TIME
- Scheduled project review date/time:6/3/2024
- Instructor name: SAM G/SAM KARU/WINNIE A
- Blog post URL:https://github.com/wafulaandree/PhaseOneProject_Movies

LIGHTS, CAMERA, DATA

In [72]:
```
# Overview:
```

In today's digital age, the entertainment industry continues to evolve rapidly, offering new opportunities for tech giants like Microsoft to diversify their portfolio. This project delves into the feasibility of Microsoft venturing into the dynamic world of movies. Leveraging comprehensive datasets from Box Office Mojo, TheMovieDB, and IMDb, we conduct an in-depth analysis to provide actionable insights and recommendations. By examining top-grossing films, prevalent genres, and key players in the industry, we aim to illuminate the path for Microsoft's potential entry into filmmaking. Additionally, we explore the significance of incorporating modern content strategies, such as social media engagement and live streaming, to remain competitive in an ever-changing landscape.

This is a data driven visionary exploration of Microsoft's potential expansion into the movie industry. As technology continues to redefine entertainment consumption patterns, major players are increasingly diversifying their offerings to stay ahead of the curve. In this project, we embark on a data-driven journey to assess the viability of Microsoft's foray into filmmaking. By analyzing extensive datasets encompassing box office performance, audience preferences, and industry trends, we aim to provide valuable insights that can inform strategic decisions. From uncovering lucrative genres to identifying emerging opportunities in content distribution, our findings aim to equip Microsoft with the knowledge needed to navigate the intricacies of the movie business. Join us as we delve into the intersection of technology and entertainment to envision a future where Microsoft redefines the cinematic landscape.

Business Problem

The key areas of inquiry are as follows:

What are the highest-grossing movies? These insights will guide Microsoft in identifying successful movie types to replicate. How do movies perform across different genres in terms of quantity, revenue, and ratings? This analysis will inform investment decisions by highlighting genres with the most potential. What is the significance of a studio's performance in terms of revenue generation? Microsoft may consider collaborating with high-performing studios or directors for content creation. Data Understanding: Box Office Mojo Dataset: This dataset, provided in CSV format, contains information on the domestic and international gross income of movies. It is crucial for evaluating the financial returns of individual movies over time. The Movie Database Dataset: This dataset offers a snapshot of movie details such as title, genre (identified by

ID), language, release date, and voting statistics. It serves as a valuable resource for assessing box office performance and determining the popularity of various genres. The IMDb Dataset: Comprising multiple CSV files, this dataset includes genre information, reviews, and ratings. Of particular interest are the files imdb.title.basics.csv.gz (which provides genre information and reviews) and imdb.title.ratings.csv.gz.

In [ ]:

Importing necessary modules

To begin our analysis, we'll import the necessary Python modules. We'll also create aliases for these modules to simplify our code.

In [73]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
import sqlite3
```

In [75]:
```
mojo_df = pd.read_csv("zippedData/bom.movie_gross.csv.gz")
mojo_df.head()
```

Out[75]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
3	Inception	WB	292600000.0	535700000	2010
4	Shrek Forever After	P/DW	238700000.0	513900000	2010

In [77]:
```
#Displaying part of the dataframe

with pd.option_context('display.max_rows', 20, 'display.max_columns', 5):
    print(mojo_df)
```

```
                                          title      studio  domestic_gross  \
0                                  Toy Story 3          BV     415000000.0
1                    Alice in Wonderland (2010)          BV     334200000.0
2     Harry Potter and the Deathly Hallows Part 1          WB     296000000.0
3                                     Inception          WB     292600000.0
4                           Shrek Forever After        P/DW     238700000.0
...                                         ...         ...             ...
3382                                The Quake       Magn.           6200.0
3383                 Edward II (2018 re-release)          FM           4800.0
3384                                  El Pacto        Sony           2500.0
3385                                  The Swan  Synergetic           2400.0
3386                           An Actor Prepares       Grav.           1700.0

     foreign_gross  year
0        652000000  2010
1        691300000  2010
2        664300000  2010
3        535700000  2010
4        513900000  2010
...            ...   ...
```

```
3382           NaN  2018
3383           NaN  2018
3384           NaN  2018
3385           NaN  2018
3386           NaN  2018

[3387 rows x 5 columns]
```

In [78]:
```
# a summary of the datafarame created and the datatypes, number of columns and rows, nul

mojo_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   title           3387 non-null   object
 1   studio          3382 non-null   object
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object
 4   year            3387 non-null   int64 
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

Description of the above information

title is the name of the movie.

studio - the production house.

foreign_gross and domestic_gross - income in home market and international market.

year - the year the movie was released.

note that: 1.foreign_gross is a str, should be int. 2.missing values are noted in the studio, domestic_gross and foreign_gross columns.

In [80]:
```
# Missing Data Management

mojo_df.isnull().sum()
```

Out[80]:
```
title              0
studio             5
domestic_gross    28
foreign_gross   1350
year               0
dtype: int64
```

-The above information allows us to assess the amount of missing data we are working with as follows: -For the studio column, due to ths small number the affected rows can be removed as the effect will be insignificant to the overall data. -Foreign_gross has the largest number of null values while the studio the smallest number of null values.

In [81]:
```
# Studio Column

# checking weight in percentage of the missing data in studio column
mojo_df["studio"].isnull().mean() * 100
0.14762326542663123
```

```
# Dropping rows with missing values from the 'studio' column
mojo_df.dropna(subset=['studio'], inplace=True)

# Check if there are any missing values left in the 'studio' column
mojo_df['studio'].isnull().sum()
```

Out[81]:
```
0
```

In the foreign_gross Column, replace missing values with 0 to show no foreign income.

In [82]:
```
mojo_df["foreign_gross"].tail(20)
```

Out[82]:
```
3367    NaN
3368    NaN
3369    NaN
3370    NaN
3371    NaN
3372    NaN
3373    NaN
3374    NaN
3375    NaN
3376    NaN
3377    NaN
3378    NaN
3379    NaN
3380    NaN
3381    NaN
3382    NaN
3383    NaN
3384    NaN
3385    NaN
3386    NaN
Name: foreign_gross, dtype: object
```

In [83]:
```
mojo_df["foreign_gross"].fillna(0, inplace=True)
mojo_df["foreign_gross"].tail(20)
```

Out[83]:
```
3367    0
3368    0
3369    0
3370    0
3371    0
3372    0
3373    0
3374    0
3375    0
3376    0
3377    0
3378    0
3379    0
3380    0
3381    0
3382    0
3383    0
3384    0
3385    0
3386    0
Name: foreign_gross, dtype: object
```

For the domestic_gross Column - replace missing values with 0 to show no foreign income.

In [85]:
```
mojo_df["domestic_gross"].fillna(0, inplace=True)
mojo_df["domestic_gross"].iloc[926:966]
```

Out[85]:
```
928          0.0
```

```
929    2600000.0
930    4300000.0
931    1000000.0
932    4099999.0
934    4000000.0
935    3400000.0
936          0.0
937     355000.0
938     354000.0
939      23400.0
940    3700000.0
941    2000000.0
942      75700.0
943    3300000.0
944    3100000.0
945    3100000.0
946    3000000.0
947      69100.0
948    2900000.0
949    2800000.0
950     192000.0
951    2700000.0
952     151000.0
953    2600000.0
954    2500000.0
955    1500000.0
956    2400000.0
957    2300000.0
958     351000.0
959     304000.0
960    2000000.0
961     898000.0
962    2000000.0
963       3500.0
964    1900000.0
965    1800000.0
966          0.0
967      11000.0
968     138000.0
Name: domestic_gross, dtype: float64
```

In [86]:
```
mojo_df.isnull().sum()
```

Out[86]:
```
title             0
studio            0
domestic_gross    0
foreign_gross     0
year              0
dtype: int64
```

In [87]:
```
#To be able to calculate the income, we will change the datatyoe of foreign_gross column
```

In [88]:
```
mojo_df["foreign_gross"] = pd.to_numeric(mojo_df["foreign_gross"], errors="coerce")
mojo_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3382 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3382 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3382 non-null   float64
 3   foreign_gross   3377 non-null   float64
 4   year            3382 non-null   int64  
```

```
dtypes: float64(2), int64(1), object(2)
memory usage: 158.5+ KB
```

In [89]:
```
# displaying the affected rows

fg_missing = mojo_df[mojo_df["foreign_gross"].isna()]
fg_missing
```

Out[89]:

	title	studio	domestic_gross	foreign_gross	year
1872	Star Wars: The Force Awakens	BV	936700000.0	NaN	2015
1873	Jurassic World	Uni.	652300000.0	NaN	2015
1874	Furious 7	Uni.	353000000.0	NaN	2015
2760	The Fate of the Furious	Uni.	226000000.0	NaN	2017
3079	Avengers: Infinity War	BV	678800000.0	NaN	2018

In [90]:
```
# creating the new figures to replace NaN
# this will replace the missing figures with correct values

fg_missing_dict = {"Star Wars: The Force Awakens": 1134647993,
                   "Jurassic World": 1018130819,
                   "Furious 7": 1162334379,
                   "The Fate of the Furious": 1009996733,
                   "Avengers: Infinity War": 1373599557}
```

In [91]:
```
# use a for loop to update the values
# it assigns the key to the title and the value to the figure
# locate the key and value in the DataFrame
for index, (key, value) in enumerate(fg_missing_dict.items()):
    mojo_df.loc[mojo_df.title == key, 'foreign_gross'] = value
```

In [92]:
```
# testing the changes
mojo_df[mojo_df['title'] == "The Fate of the Furious"]
```

Out[92]:

	title	studio	domestic_gross	foreign_gross	year
2760	The Fate of the Furious	Uni.	226000000.0	1.009997e+09	2017

In [94]:
```
mojo_df.info()
mojo_df.iloc[1868:1873]
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3382 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   title           3382 non-null   object
 1   studio          3382 non-null   object
 2   domestic_gross  3382 non-null   float64
 3   foreign_gross   3382 non-null   float64
 4   year            3382 non-null   int64
dtypes: float64(2), int64(1), object(2)
memory usage: 158.5+ KB
```

Out[94]:

	title	studio	domestic_gross	foreign_gross	year
1872	Star Wars: The Force Awakens	BV	936700000.0	1.134648e+09	2015
1873	Jurassic World	Uni.	652300000.0	1.018131e+09	2015
1874	Furious 7	Uni.	353000000.0	1.162334e+09	2015
1875	Avengers: Age of Ultron	BV	459000000.0	9.464000e+08	2015

1876	Minions	Uni.	336000000.0	8.234000e+08	2015

foreign_gross is a str. However it should be an int thus the coversion.

In [96]:
```
# converting foreign_gross to int64 for readability

mojo_df["foreign_gross"] = mojo_df["foreign_gross"].astype("int64")
mojo_df.iloc[1868:1873]
```

Out[96]:

	title	studio	domestic_gross	foreign_gross	year
1872	Star Wars: The Force Awakens	BV	936700000.0	1134647993	2015
1873	Jurassic World	Uni.	652300000.0	1018130819	2015
1874	Furious 7	Uni.	353000000.0	1162334379	2015
1875	Avengers: Age of Ultron	BV	459000000.0	946400000	2015
1876	Minions	Uni.	336000000.0	823400000	2015

Creating a new column The new column is "gross_income"

this is done by summing up the "domestic_gross" and "foreign_gross" columns can be beneficial for several reasons:

Data Consolidation: It consolidates related information into a single column, making it easier to analyze and interpret.

Analysis Convenience: It simplifies analysis by providing a single metric (gross income) instead of having to consider multiple separate columns.

Visualization: It enables the visualization of the combined gross income, which may reveal patterns or trends that are not immediately evident when looking at the individual components separately.

Calculation Efficiency: Pre-calculating the gross income and storing it in a new column can improve computational efficiency, especially if you need to perform calculations or analysis involving gross income frequently.Creating a new column like "gross_income" by summing up the "domestic_gross" and "foreign_gross" columns can be beneficial for several reasons:

Data Consolidation: It consolidates related information into a single column, making it easier to analyze and interpret.

Analysis Convenience: It simplifies analysis by providing a single metric (gross income) instead of having to consider multiple separate columns.

Visualization: It enables the visualization of the combined gross income, which may reveal patterns or trends that are not immediately evident when looking at the individual components separately.

Calculation Efficiency: Pre-calculating the gross income and storing it in a new column can improve computational efficiency, especially if you need to perform calculations or analysis involving gross income frequently.

In [98]:
```
#create a new column 'gross_income'
mojo_df["gross_income"] = mojo_df["domestic_gross"] + mojo_df["foreign_gross"]
mojo_df
```

	title	studio	domestic_gross	foreign_gross	year	gross_income

Out[98]:

0	Toy Story 3	BV	415000000.0	652000000	2010	1.067000e+09
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010	1.025500e+09
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010	9.603000e+08
3	Inception	WB	292600000.0	535700000	2010	8.283000e+08
4	Shrek Forever After	P/DW	238700000.0	513900000	2010	7.526000e+08
...	...	...	...	...	...	...
3382	The Quake	Magn.	6200.0	0	2018	6.200000e+03
3383	Edward II (2018 re-release)	FM	4800.0	0	2018	4.800000e+03
3384	El Pacto	Sony	2500.0	0	2018	2.500000e+03
3385	The Swan	Synergetic	2400.0	0	2018	2.400000e+03
3386	An Actor Prepares	Grav.	1700.0	0	2018	1.700000e+03

3382 rows × 6 columns

In [99]:
```
# Convert the "gross_income" column to int64

mojo_df["gross_income"] = pd.to_numeric(mojo_df["gross_income"], errors="coerce", downca

mojo_df
```

Out[99]:

	title	studio	domestic_gross	foreign_gross	year	gross_income
0	Toy Story 3	BV	415000000.0	652000000	2010	1067000000
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010	1025500000
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010	960300000
3	Inception	WB	292600000.0	535700000	2010	828300000
4	Shrek Forever After	P/DW	238700000.0	513900000	2010	752600000
...	...	...	...	...	...	...
3382	The Quake	Magn.	6200.0	0	2018	6200
3383	Edward II (2018 re-release)	FM	4800.0	0	2018	4800
3384	El Pacto	Sony	2500.0	0	2018	2500
3385	The Swan	Synergetic	2400.0	0	2018	2400
3386	An Actor Prepares	Grav.	1700.0	0	2018	1700

3382 rows × 6 columns

Selecting the columns to keep

In [100…
```
# Reorder the columns
mojo_df = mojo_df.loc[:, ["title", "studio", "gross_income"]]

# Display information about the DataFrame
mojo_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3382 entries, 0 to 3386
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype
```

```
---  ------         --------------  -----
 0   title          3382 non-null   object
 1   studio         3382 non-null   object
 2   gross_income   3382 non-null   int32
dtypes: int32(1), object(2)
memory usage: 92.5+ KB
```

Database Dataset Read the CSV file and display the first five rows by using the iloc indexer to select the first five rows after reading the file with pd.read_csv():

In [101…

```
# Read the CSV file
moviedb_df = pd.read_csv("zippedData/tmdb.movies.csv.gz", index_col=0)

# Display the first five rows
moviedb_df.iloc[:5]
```

Out[101]:

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average	vote_co
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	7.7	10
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon	7.7	7
2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2	6.8	12
3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story	7.9	10
4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception	8.3	22

In [102…

```
# Display summary information from the dataframe created
print(moviedb_df.describe())
print(moviedb_df.info())
```

```
                 id    popularity  vote_average    vote_count
count   26517.000000  26517.000000  26517.000000  26517.000000
mean   295050.153260      3.130912      5.991281    194.224837
std    153661.615648      4.355229      1.852946    960.961095
min        27.000000      0.600000      0.000000      1.000000
25%    157851.000000      0.600000      5.000000      2.000000
50%    309581.000000      1.374000      6.000000      5.000000
75%    419542.000000      3.694000      7.000000     28.000000
max    608444.000000     80.773000     10.000000  22186.000000
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26517 entries, 0 to 26516
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   genre_ids          26517 non-null  object
 1   id                 26517 non-null  int64
 2   original_language  26517 non-null  object
 3   original_title     26517 non-null  object
 4   popularity         26517 non-null  float64
 5   release_date       26517 non-null  object
 6   title              26517 non-null  object
 7   vote_average       26517 non-null  float64
 8   vote_count         26517 non-null  int64
dtypes: float64(2), int64(2), object(5)
```

```
memory usage: 2.0+ MB
None
```

This dataset appears to be well-structured, with no missing values, indicating that it's relatively clean. However, to gain deeper insights, further exploration is needed to understand the content of the genre_ids column. Each integer in this column likely corresponds to a specific genre category, but without additional information, it's challenging to interpret these values accurately. Therefore, a more in-depth analysis is necessary to decode the genre representation within these lists of integers. Additionally, exploring potential relationships between movie genres and other variables could provide valuable insights for subsequent analysis and decision-making

In [103…
```
moviedb_df.isna().sum()
```

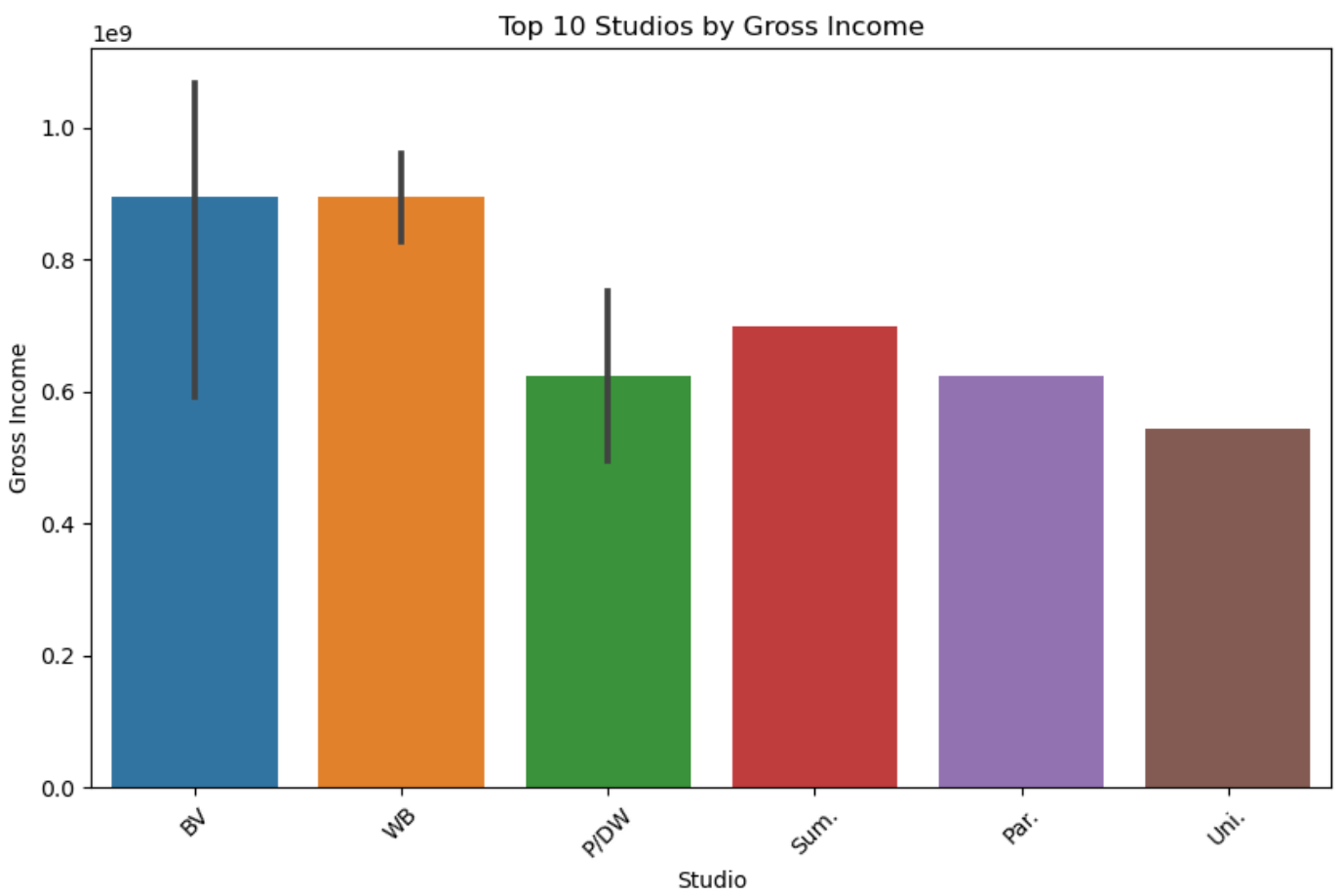
Out[103]:
```
genre_ids            0
id                   0
original_language    0
original_title       0
popularity           0
release_date         0
title                0
vote_average         0
vote_count           0
dtype: int64
```

In [104…
```
#select the columns to keep

moviedb_df = moviedb_df[['title', 'vote_average']]
moviedb_df.info()
```

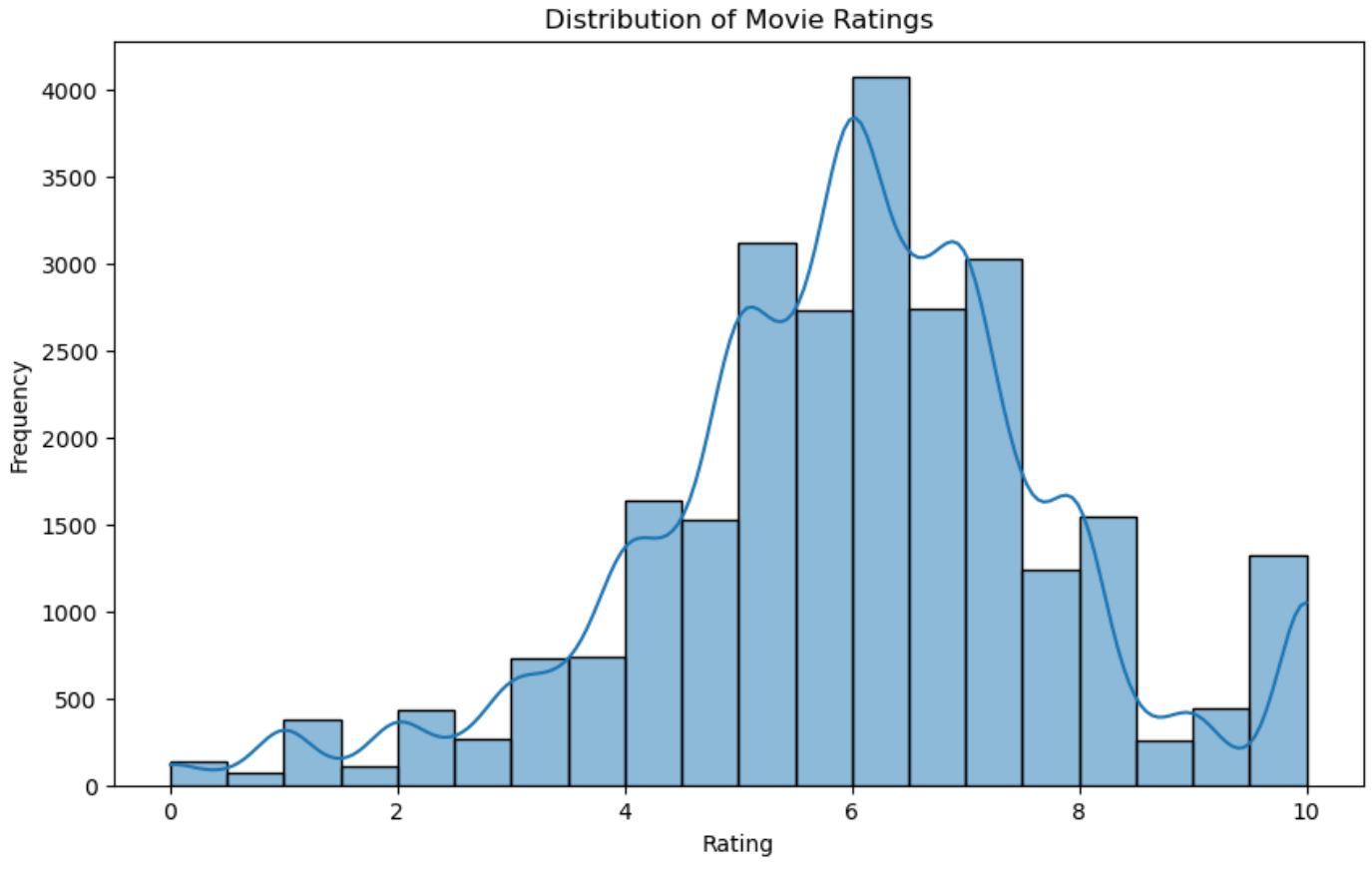
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26517 entries, 0 to 26516
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   title         26517 non-null  object 
 1   vote_average  26517 non-null  float64
dtypes: float64(1), object(1)
memory usage: 621.5+ KB
```

In [105…
```
# Visualization for mojo_df
plt.figure(figsize=(10, 6))
sns.barplot(x='studio', y='gross_income', data=mojo_df.head(10))
plt.title('Top 10 Studios by Gross Income')
plt.xlabel('Studio')
plt.ylabel('Gross Income')
plt.xticks(rotation=45)
plt.show()
```



In [106…

```
# Visualization for moviedb_df
plt.figure(figsize=(10, 6))
sns.histplot(moviedb_df['vote_average'], bins=20, kde=True)
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```



The Numbers Dataset

The Numbers is a film industry data website that tracks box office revenue in a systematic, algorithmic way. This way, the company also conducts research services and forecasts incomes of film project provinding detailed movie financial analysis, including; box office, DVD and Blu-ray sales reports, and release schedules.

The IMDb Dataset

Connecting to dataset

In [107…

```
import zipfile
import sqlite3

# Specify the path to the ZIP file
zip_file_path = "zippedData/im.db.zip"

# Extract the SQLite database file from the ZIP archive
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall("unzipped_data")

# Connect to the SQLite database
try:
    conn = sqlite3.connect("unzipped_data/im.db")
    print("Connected to the database successfully!")
except sqlite3.Error as e:
    print("Error connecting to the database:", e)

# Get a cursor object to execute SQL queries
cur = conn.cursor()

# Execute a SQL query to fetch the list of tables
try:
    cur.execute("SELECT name FROM sqlite_master WHERE type='table';")
    tables = cur.fetchall()
    if tables:
        print("List of tables in the database:")
        for table in tables:
            print(table[0])
    else:
        print("No tables found in the database.")
except sqlite3.Error as e:
    print("Error fetching tables from the database:", e)

# Close the cursor and connection
cur.close()
conn.close()
```

```
Connected to the database successfully!
List of tables in the database:
movie_basics
directors
known_for
movie_akas
movie_ratings
persons
principals
writers
```

In [ ]:

In [108…
```
conn = sqlite3.connect("zippedData/im.db")
cur = conn.cursor()
tables = pd.read_sql("SELECT * FROM sqlite_master WHERE type='table';", conn)
tables
```

Out[108]:

	type	name	tbl_name	rootpage	sql
0	table	movie_basics	movie_basics	2	CREATE TABLE "movie_basics" (\n"movie_id" TEXT...
1	table	directors	directors	3	CREATE TABLE "directors" (\n"movie_id" TEXT,\n...
2	table	known_for	known_for	4	CREATE TABLE "known_for" (\n"person_id" TEXT,\...
3	table	movie_akas	movie_akas	5	CREATE TABLE "movie_akas" (\n"movie_id" TEXT,\...
4	table	movie_ratings	movie_ratings	6	CREATE TABLE "movie_ratings" (\n"movie_id" TEX...
5	table	persons	persons	7	CREATE TABLE "persons" (\n"person_id" TEXT,\n ...
6	table	principals	principals	8	CREATE TABLE "principals" (\n"movie_id" TEXT,\...
7	table	writers	writers	9	CREATE TABLE "writers" (\n"movie_id" TEXT,\n ...

For this analysis, the movie_basics and movie_ratings tables will be used. They will then be joined using the movie_id column as it is common to both.

In [109…
```
# creating df from movie_basics table
imdb_moviebasics_df = pd.read_sql("SELECT * FROM movie_basics", conn)
imdb_moviebasics_df.head()
```

Out[109]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy

In [110…
```
# Function to check if table exists
def table_exists(table_name):
    query = f"SELECT name FROM sqlite_master WHERE type='table' AND name='{table_name}'"
    result = conn.execute(query).fetchone()
    return result is not None

# Check if the table exists before reading from it
if table_exists('movie_basics'):
    imdb_moviebasics_df = pd.read_sql("SELECT * FROM movie_basics", conn)
    print(imdb_moviebasics_df.head())
else:
    print("The table 'movie_basics' does not exist in the database.")
```

```
    movie_id                    primary_title              original_title  \
0  tt0063540                        Sunghursh                  Sunghursh
1  tt0066787  One Day Before the Rainy Season             Ashad Ka Ek Din
2  tt0069049       The Other Side of the Wind  The Other Side of the Wind
3  tt0069204                  Sabse Bada Sukh             Sabse Bada Sukh
4  tt0100275         The Wandering Soap Opera       La Telenovela Errante
```

```
   start_year  runtime_minutes                genres
0        2013            175.0    Action,Crime,Drama
1        2019            114.0       Biography,Drama
2        2018            122.0                 Drama
3        2018              NaN          Comedy,Drama
4        2017             80.0  Comedy,Drama,Fantasy
```

Dataframe: movie_ratings table

In [111…
```
from sqlalchemy import create_engine

# Create SQLAlchemy engine
engine = create_engine('sqlite:///imdb_moviebasics_.db')

try:
    # Execute SQL query and read data into DataFrame
    query = "SELECT * FROM movie_ratings"
    imdb_movieratings_df = pd.read_sql_query(query, engine)

    # Display the first few rows of the DataFrame
    print(imdb_movieratings_df.head())
except Exception as e:
    print("Error:", e)
```

```
Error: (sqlite3.OperationalError) no such table: movie_ratings
[SQL: SELECT * FROM movie_ratings]
(Background on this error at: http://sqlalche.me/e/13/e3q8)
```

Alternatively:

In [112…
```
imdb_movieratings_df = pd.read_sql("SELECT * FROM movie_ratings", conn)
imdb_movieratings_df.head()
```

Out[112]:

	movie_id	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21

In [113…
```
imdb_movieratings_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   movie_id       73856 non-null  object 
 1   averagerating  73856 non-null  float64
 2   numvotes       73856 non-null  int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 1.7+ MB
```

Data Cleaning for movie_id Column Data Cleaning for movie_id Column in imdb_movieratings_df Merging DataFrames

These operations clean and merge the data from the two DataFrames based on the movie_id column, creating a new DataFrame imdb_df containing combined information from both original DataFrames.

In [119…

```
imdb_moviebasics_df['movie_id'] = imdb_moviebasics_df['movie_id'].astype('int64')
imdb_movieratings_df['movie_id'] = imdb_movieratings_df['movie_id'].astype('int64')

imdb_df = imdb_moviebasics_df.join(imdb_movieratings_df.set_index('movie_id'), on='movie
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [119], line 1
----> 1 imdb_moviebasics_df['movie_id'] = imdb_moviebasics_df['movie_id'].astype('int64'
)
      2 imdb_movieratings_df['movie_id'] = imdb_movieratings_df['movie_id'].astype('int6
4')
      4 imdb_df = imdb_moviebasics_df.join(imdb_movieratings_df.set_index('movie_id'), o
n='movie_id', how='inner')

File ~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\generic.py:5546, in NDFram
e.astype(self, dtype, copy, errors)
   5539     results = [
   5540         self.iloc[:, i].astype(dtype, copy=copy)
   5541         for i in range(len(self.columns))
   5542     ]
   5544 else:
   5545     # else, only a single dtype is given
-> 5546     new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors,)
   5547     return self._constructor(new_data).__finalize__(self, method="astype")
   5549 # GH 33113: handle empty frame or series

File ~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\internals\managers.py:595,
 in BlockManager.astype(self, dtype, copy, errors)
    592 def astype(
    593     self, dtype, copy: bool = False, errors: str = "raise"
    594 ) -> "BlockManager":
--> 595     return self.apply("astype", dtype=dtype, copy=copy, errors=errors)

File ~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\internals\managers.py:406,
 in BlockManager.apply(self, f, align_keys, **kwargs)
    404         applied = b.apply(f, **kwargs)
    405     else:
--> 406         applied = getattr(b, f)(**kwargs)
    407     result_blocks = _extend_blocks(applied, result_blocks)
    409 if len(result_blocks) == 0:

File ~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\internals\blocks.py:595, i
n Block.astype(self, dtype, copy, errors)
    593 vals1d = values.ravel()
    594 try:
--> 595     values = astype_nansafe(vals1d, dtype, copy=True)
    596 except (ValueError, TypeError):
    597     # e.g. astype_nansafe can fail on object-dtype of strings
    598     #  trying to convert to float
    599     if errors == "raise":

File ~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\dtypes\cast.py:972, in ast
ype_nansafe(arr, dtype, copy, skipna)
    968 elif is_object_dtype(arr):
    969
    970     # work around NumPy brokenness, #1987
    971     if np.issubdtype(dtype.type, np.integer):
--> 972         return lib.astype_intsafe(arr.ravel(), dtype).reshape(arr.shape)
    974     # if we have a datetime/timedelta array of objects
    975     # then coerce to a proper dtype and recall astype_nansafe
    977     elif is_datetime64_dtype(dtype):

File pandas\_libs\lib.pyx:614, in pandas._libs.lib.astype_intsafe()
```

```
ValueError: invalid literal for int() with base 10: 'tt0063540'
```

In [120…

```
imdb_df = imdb_df.drop(["original_title", "runtime_minutes", "numvotes"], axis=1)
imdb_df.columns
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In [120], line 1
----> 1 imdb_df = imdb_df.drop(["original_title", "runtime_minutes", "numvotes"], axis=1
)
      2 imdb_df.columns

File ~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py:4163, in DataFram
e.drop(self, labels, axis, index, columns, level, inplace, errors)
   4034 def drop(
   4035     self,
   4036     labels=None,
   (...)
   4042     errors="raise",
   4043 ):
   4044     """
   4045     Drop specified labels from rows or columns.
   4046
   (...)
   4161             weight  1.0     0.8
   4162     """
-> 4163     return super().drop(
   4164         labels=labels,
   4165         axis=axis,
   4166         index=index,
   4167         columns=columns,
   4168         level=level,
   4169         inplace=inplace,
   4170         errors=errors,
   4171     )

File ~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\generic.py:3887, in NDFram
e.drop(self, labels, axis, index, columns, level, inplace, errors)
   3885 for axis, labels in axes.items():
   3886     if labels is not None:
-> 3887         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   3889 if inplace:
   3890     self._update_inplace(obj)

File ~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\generic.py:3921, in NDFram
e._drop_axis(self, labels, axis, level, errors)
   3919         new_axis = axis.drop(labels, level=level, errors=errors)
   3920     else:
-> 3921         new_axis = axis.drop(labels, errors=errors)
   3922     result = self.reindex(**{axis_name: new_axis})
   3924 # Case for non-unique axis
   3925 else:

File ~\anaconda3\envs\learn-env\lib\site-packages\pandas\core\indexes\base.py:5282, in I
ndex.drop(self, labels, errors)
   5280 if mask.any():
   5281     if errors != "ignore":
-> 5282         raise KeyError(f"{labels[mask]} not found in axis")
   5283     indexer = indexer[~mask]
   5284 return self.delete(indexer)

KeyError: "['original_title' 'runtime_minutes' 'numvotes'] not found in axis"
```

In [121…

```
imdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 73856 entries, 0 to 146134
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   title          73856 non-null  object
 1   start_year     73856 non-null  int64
 2   genres         73052 non-null  object
 3   averagerating  73856 non-null  float64
dtypes: float64(1), int64(1), object(2)
memory usage: 2.8+ MB
```

The missing values will be deleted as the movies are not as popular. this means that their influence on decision making is not focused on

Renaming the primary_title column to tile and selecting the columns to keep

In [122…
```
imdb_df.rename(columns = {'primary_title':'title'}, inplace = True)
imdb_df.columns

imdb_df = imdb_df[["title", "start_year", "genres", "averagerating"]]
imdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 73856 entries, 0 to 146134
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   title          73856 non-null  object
 1   start_year     73856 non-null  int64
 2   genres         73052 non-null  object
 3   averagerating  73856 non-null  float64
dtypes: float64(1), int64(1), object(2)
memory usage: 2.8+ MB
```

Create One Df

In [123…
```
movies = imdb_df.merge(moviedb_df, on = "title").merge(mojo_df, on = "title")
movies
```

Out[123]:

	title	start_year	genres	averagerating	vote_average	studio	gross_income
0	Wazir	2016	Action,Crime,Drama	7.1	6.6	Relbig.	1100000
1	On the Road	2012	Adventure,Drama,Romance	6.1	5.6	IFC	8744000
2	On the Road	2014	Drama	6.0	5.6	IFC	8744000
3	On the Road	2016	Drama	5.7	5.6	IFC	8744000
4	The Secret Life of Walter Mitty	2013	Adventure,Comedy,Drama	7.3	7.1	Fox	188100000
...	...	...	...	...	...	...	...
3292	Nobody's Fool	2018	Comedy,Drama,Romance	4.6	5.9	Par.	33500000
3293	Capernaum	2018	Drama	8.5	8.4	SPC	1700000
3294	The Spy Gone North	2018	Drama	7.2	7.3	CJ	501000

3295	How Long Will I Love U	2018	Romance	6.5	7.4	WGUSA	82847000
3296	Last Letter	2018	Drama,Romance	6.4	6.0	CL	181000

3297 rows × 7 columns

In [124…
```
movies.columns = movies.columns.str.title()
movies.columns

movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3297 entries, 0 to 3296
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Title          3297 non-null   object 
 1   Start_Year     3297 non-null   int64  
 2   Genres         3288 non-null   object 
 3   Averagerating  3297 non-null   float64
 4   Vote_Average   3297 non-null   float64
 5   Studio         3297 non-null   object 
 6   Gross_Income   3297 non-null   int32  
dtypes: float64(2), int32(1), int64(1), object(3)
memory usage: 193.2+ KB
```

In [ ]:
```
#note that the Genres column has missingvalues.
#This leads us to drop the rows without a genre
```

In [125…
```
movies = movies.dropna(subset=["Genres"])
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3288 entries, 0 to 3296
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Title          3288 non-null   object 
 1   Start_Year     3288 non-null   int64  
 2   Genres         3288 non-null   object 
 3   Averagerating  3288 non-null   float64
 4   Vote_Average   3288 non-null   float64
 5   Studio         3288 non-null   object 
 6   Gross_Income   3288 non-null   int32  
dtypes: float64(2), int32(1), int64(1), object(3)
memory usage: 192.7+ KB
```

In [126…
```
#inclusion of needed columns
movies = movies.assign(Genre = movies["Genres"].str.split(',')).explode("Genre")
movies.head()
```

Out[126]:

	Title	Start_Year	Genres	Averagerating	Vote_Average	Studio	Gross_Income	Genre
0	Wazir	2016	Action,Crime,Drama	7.1	6.6	Relbig.	1100000	Action
0	Wazir	2016	Action,Crime,Drama	7.1	6.6	Relbig.	1100000	Crime
0	Wazir	2016	Action,Crime,Drama	7.1	6.6	Relbig.	1100000	Drama
1	On the Road	2012	Adventure,Drama,Romance	6.1	5.6	IFC	8744000	Adventure
1	On the	2012	Adventure,Drama,Romance	6.1	5.6	IFC	8744000	Drama

Road

In [127…
```
#Combine the Averagerating and Vote_Average columns by getting their average.
#Create a new column Rating for the above
movies["Rating"] = (movies["Averagerating"] + movies["Vote_Average"]) / 2
movies.head()
```

Out[127]:

	Title	Start_Year	Genres	Averagerating	Vote_Average	Studio	Gross_Income	Genre
0	Wazir	2016	Action,Crime,Drama	7.1	6.6	Relbig.	1100000	Action
0	Wazir	2016	Action,Crime,Drama	7.1	6.6	Relbig.	1100000	Crime
0	Wazir	2016	Action,Crime,Drama	7.1	6.6	Relbig.	1100000	Drama
1	On the Road	2012	Adventure,Drama,Romance	6.1	5.6	IFC	8744000	Adventure
1	On the Road	2012	Adventure,Drama,Romance	6.1	5.6	IFC	8744000	Drama

In [ ]:
```
# Data Analysis, Visualization and Evaluation

## Top grossing movies
```

Calculating the total gross for each film

In [128…
```
top_grossing_movies = movies.sort_values(by = "Gross_Income", ascending=False).head(20)
print(top_grossing_movies[["Title", "Gross_Income"]])
```

```
                        Title  Gross_Income
2878    Avengers: Infinity War    2052399557
2878    Avengers: Infinity War    2052399557
2878    Avengers: Infinity War    2052399557
6                Jurassic World    1670430819
6                Jurassic World    1670430819
6                Jurassic World    1670430819
2421                 Furious 7    1515334379
2421                 Furious 7    1515334379
2421                 Furious 7    1515334379
2198   Avengers: Age of Ultron    1405400000
2198   Avengers: Age of Ultron    1405400000
2198   Avengers: Age of Ultron    1405400000
1546              Black Panther    1347000000
1545              Black Panther    1347000000
1545              Black Panther    1347000000
1545              Black Panther    1347000000
1546              Black Panther    1347000000
1546              Black Panther    1347000000
2280  Star Wars: The Last Jedi    1332600000
2281  Star Wars: The Last Jedi    1332600000
```

In [54]:
```
top_grossing_movies.iloc[0]
```

Out[54]:
```
Title             Avengers: Infinity War
Start_Year                          2018
Genres           Action,Adventure,Sci-Fi
Averagerating                        8.5
Vote_Average                         8.3
Studio                                BV
Gross_Income                  2052399557
Genre                             Sci-Fi
```

```
Rating                                8.4
Name: 2878, dtype: object
```

Avengers: Infinity War grossed the highest income as per the dataset. It is categorised under the Action, Adventure and Sci-Fi genres. The income grossed to USD 2,052,399,557. Captain America: Civil War grossed the least amount (USD 1,153,300,000). It is listed in genres similar to Avengers: Infinity War, i.e. Action, Adventure and Sci-F

In [129…
```
# descriptive statistics of the top grossing movies
top_grossing_movies.describe()
```

Out[129]:

	Start_Year	Averagerating	Vote_Average	Gross_Income	Rating
count	20.000000	20.000000	20.000000	2.000000e+01	20.000000
mean	2016.550000	7.400000	7.000000	1.533895e+09	7.200000
std	1.468081	0.486664	0.956969	2.516944e+08	0.646855
min	2015.000000	7.000000	5.100000	1.332600e+09	6.200000
25%	2015.000000	7.175000	6.600000	1.347000e+09	6.800000
50%	2017.000000	7.300000	7.300000	1.405400e+09	7.250000
75%	2018.000000	7.300000	7.400000	1.670431e+09	7.350000
max	2018.000000	8.500000	8.300000	2.052400e+09	8.400000

In [130…
```
# reset the index

top_grossing_movies.reset_index(inplace=True)

movie_table = top_grossing_movies.style \
    .background_gradient(cmap='Blues') \
    .highlight_max(color='grey') \
    .set_properties(**{'text-align': 'center'}) \
    .set_caption('Top Grossing Movies') \
    .hide_index()

movie_table
```

Out[130]:

Top Grossing Movies

index	Title	Start_Year	Genres	Averagerating	Vote_Average	Studio	Gross_Income	
2878	Avengers: Infinity War	2018	Action,Adventure,Sci-Fi	8.500000	8.300000	BV	2052399557	S
2878	Avengers: Infinity War	2018	Action,Adventure,Sci-Fi	8.500000	8.300000	BV	2052399557	Adv
2878	Avengers: Infinity War	2018	Action,Adventure,Sci-Fi	8.500000	8.300000	BV	2052399557	A
6	Jurassic World	2015	Action,Adventure,Sci-Fi	7.000000	6.600000	Uni.	1670430819	A
6	Jurassic World	2015	Action,Adventure,Sci-Fi	7.000000	6.600000	Uni.	1670430819	S
6	Jurassic World	2015	Action,Adventure,Sci-Fi	7.000000	6.600000	Uni.	1670430819	Adv
2421	Furious 7	2015	Action,Crime,Thriller	7.200000	7.300000	Uni.	1515334379	Th
2421	Furious 7	2015	Action,Crime,Thriller	7.200000	7.300000	Uni.	1515334379	A

2421	Furious 7	2015	Action,Crime,Thriller	7.200000	7.300000	Uni.	1515334379	C
2198	Avengers: Age of Ultron	2015	Action,Adventure,Sci-Fi	7.300000	7.300000	BV	1405400000	Adv
2198	Avengers: Age of Ultron	2015	Action,Adventure,Sci-Fi	7.300000	7.300000	BV	1405400000	A
2198	Avengers: Age of Ultron	2015	Action,Adventure,Sci-Fi	7.300000	7.300000	BV	1405400000	S
1546	Black Panther	2018	Action,Adventure,Sci-Fi	7.300000	7.400000	BV	1347000000	S
1545	Black Panther	2018	Action,Adventure,Sci-Fi	7.300000	5.100000	BV	1347000000	A
1545	Black Panther	2018	Action,Adventure,Sci-Fi	7.300000	5.100000	BV	1347000000	Adv
1545	Black Panther	2018	Action,Adventure,Sci-Fi	7.300000	5.100000	BV	1347000000	S
1546	Black Panther	2018	Action,Adventure,Sci-Fi	7.300000	7.400000	BV	1347000000	A
1546	Black Panther	2018	Action,Adventure,Sci-Fi	7.300000	7.400000	BV	1347000000	Adv
2280	Star Wars: The Last Jedi	2017	Action,Adventure,Fantasy	7.100000	7.000000	BV	1332600000	A
2281	Star Wars: The Last Jedi	2017	Action,Adventure,Fantasy	7.100000	7.000000	BV	1332600000	A

Grossing by Genre

In [131…

```
genres = movies["Genre"].unique()
print(len(genres))
print(genres)
```

```
22
['Action' 'Crime' 'Drama' 'Adventure' 'Romance' 'Comedy' 'Sci-Fi' 'Family'
 'Animation' 'Thriller' 'Mystery' 'Biography' 'History' 'Horror'
 'Documentary' 'News' 'Fantasy' 'Sport' 'Music' 'War' 'Western' 'Musical']
```

In [132…

```
genres_gross = movies.groupby("Genre").sum()

genres_gross = genres_gross.sort_values("Gross_Income", ascending=False)

# top 10 genres by total gross income
print(genres_gross[["Gross_Income"]].head(10))
```

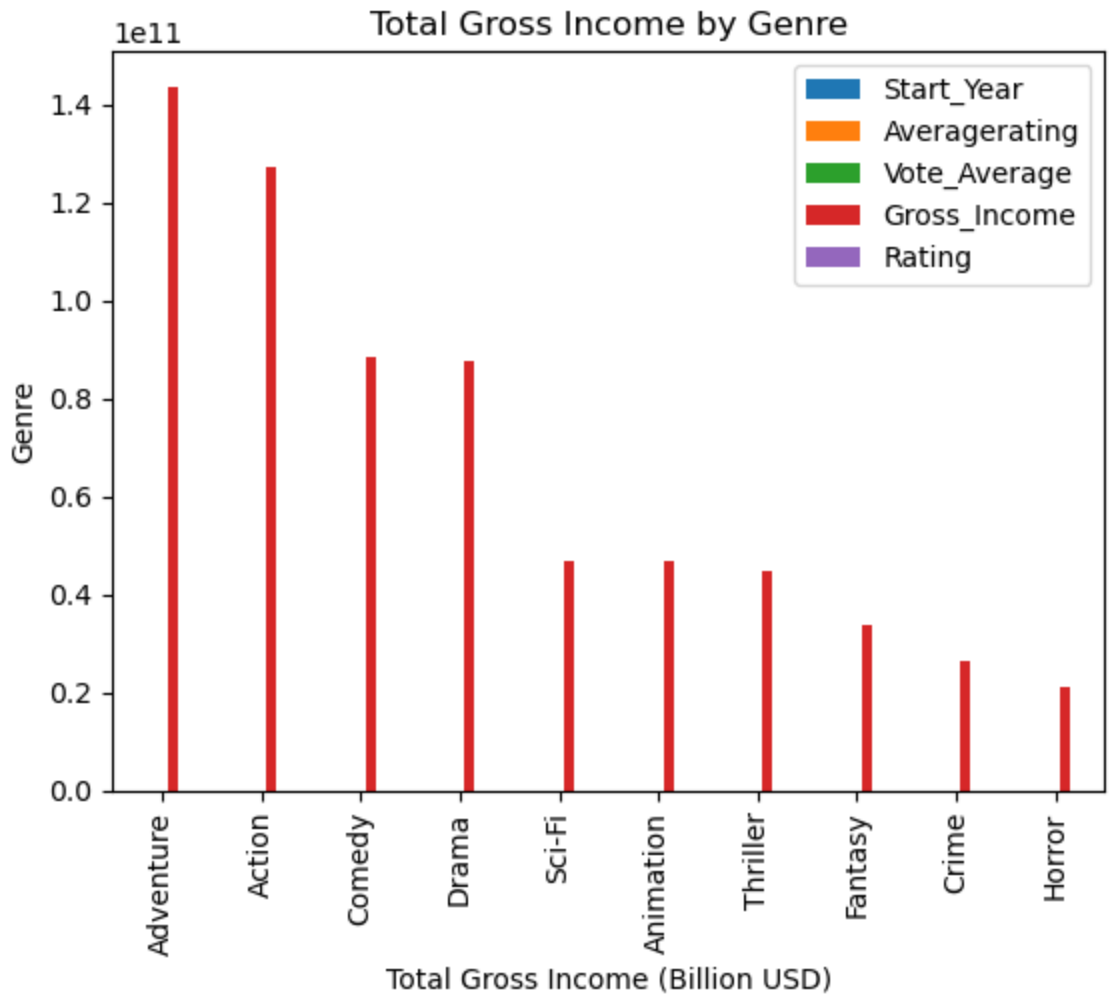
```
           Gross_Income
Genre
Adventure  1.436709e+11
Action     1.272839e+11
Comedy     8.869139e+10
Drama      8.781041e+10
Sci-Fi     4.704390e+10
Animation  4.694738e+10
Thriller   4.493712e+10
Fantasy    3.391112e+10
```

```
Crime      2.665714e+10
Horror     2.107995e+10
```

Graph plot

In [133…
```
genres_gross.head(10).plot(kind='bar')
plt.xlabel('Total Gross Income (Billion USD)')
plt.ylabel('Genre')
plt.title('Total Gross Income by Genre')

plt.legend()
plt.show()
```



Movies with a rating above 5

In [134…
```
rating = movies.groupby("Genre").Rating.agg(["count","mean"]).sort_values(
    'mean', ascending = False)
rating[rating["count"]>=5]
```

Out[134]:

Genre	count	mean
Documentary	231	7.031602
Biography	329	6.886474
War	46	6.805435
History	149	6.801678

Animation	158	6.731646
Music	103	6.673301
Sport	65	6.536923
Western	22	6.531818
Drama	1937	6.524574
Adventure	491	6.439308
Crime	433	6.410508
Sci-Fi	160	6.383750
Romance	481	6.366008
Family	124	6.310887
Musical	15	6.310000
Mystery	243	6.265638
Comedy	958	6.254958
Action	661	6.245840
Fantasy	194	6.236598
Thriller	539	6.110390
Horror	315	5.763492

Datat visualisation

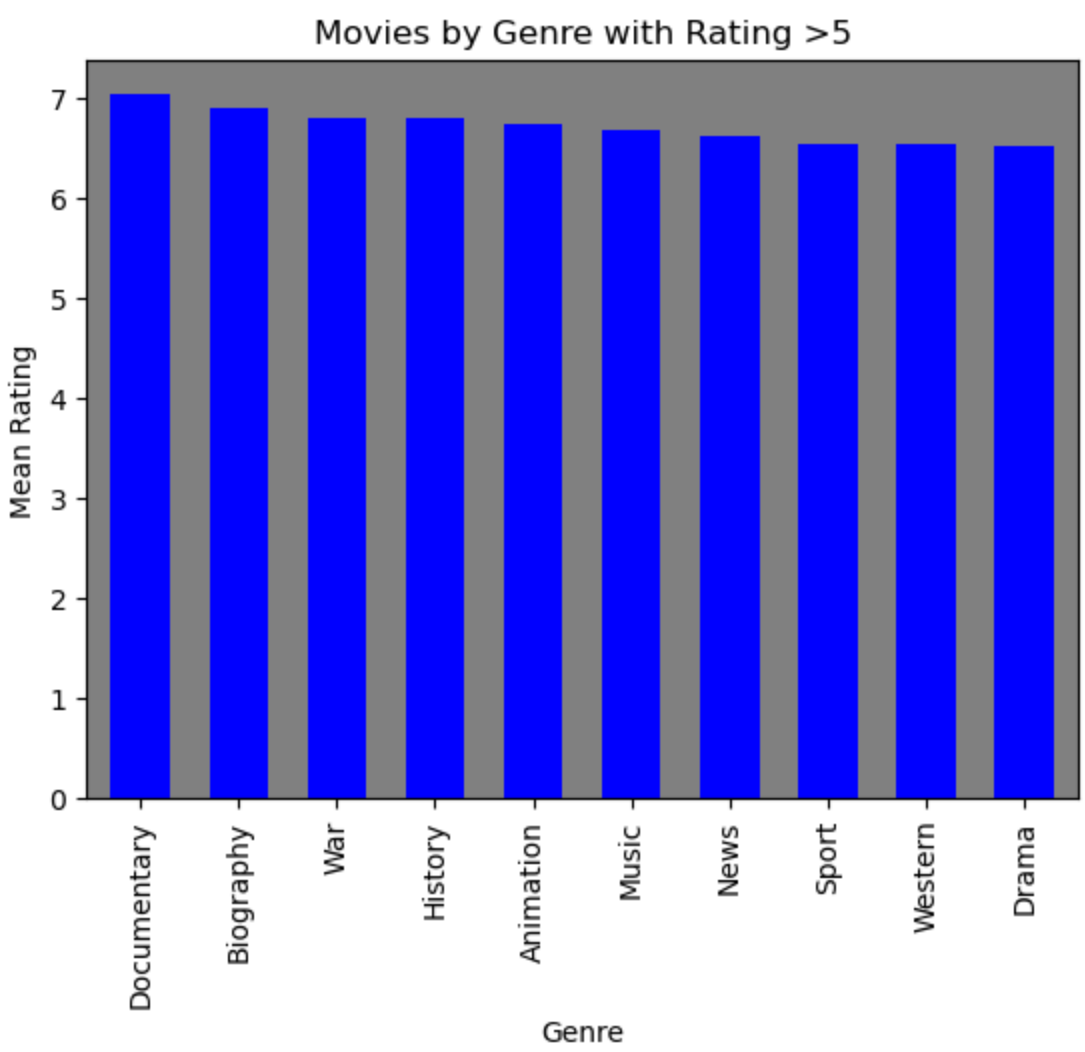
In [135…

```
ax = rating['mean'].head(10).plot(kind='bar', color='blue', width=0.6)

plt.title("Movies by Genre with Rating >5")
plt.xlabel("Genre")
plt.ylabel("Mean Rating")

ax.patch.set_facecolor('grey')

plt.show()
```



Studio income

In [136…

```
# gross_per_studio = total gross_income for each studio

gross_per_studio = movies.groupby(["Studio"]).sum()

gross_per_studio = gross_per_studio.sort_values("Gross_Income", ascending=False)

# top 5 studios by total gross income
print(gross_per_studio[["Gross_Income"]].head())

# bottom 5 studios by total gross income
print(gross_per_studio[["Gross_Income"]].tail())
```

```
        Gross_Income
Studio
BV      1.698523e+11
Uni.    1.171470e+11
Fox     9.679151e+10
WB      8.285893e+10
Sony    5.953190e+10
          Gross_Income
Studio
ICir           29300.0
ALP            16800.0
TAFC           13800.0
KS             11800.0
EpicPics       11300.0
```

In [137…

```
gross_per_studio = gross_per_studio.reset_index()
```

```
gross_per_studio
```

Out[137]:

	Studio	Start_Year	Averagerating	Vote_Average	Gross_Income	Rating
0	BV	630366	2190.9	2126.3	1.698523e+11	2158.60
1	Uni.	1091697	3328.9	3316.9	1.171470e+11	3322.90
2	Fox	805626	2577.8	2530.9	9.679151e+10	2554.35
3	WB	729070	2388.0	2351.5	8.285893e+10	2369.75
4	Sony	485345	1494.4	1468.7	5.953190e+10	1481.55
...	...	...	...	...	...	...
184	ICir	2011	7.5	6.6	2.930000e+04	7.05
185	ALP	12078	40.5	36.6	1.680000e+04	38.55
186	TAFC	6042	19.5	20.4	1.380000e+04	19.95
187	KS	4026	13.2	14.2	1.180000e+04	13.70
188	EpicPics	2015	4.7	4.5	1.130000e+04	4.60

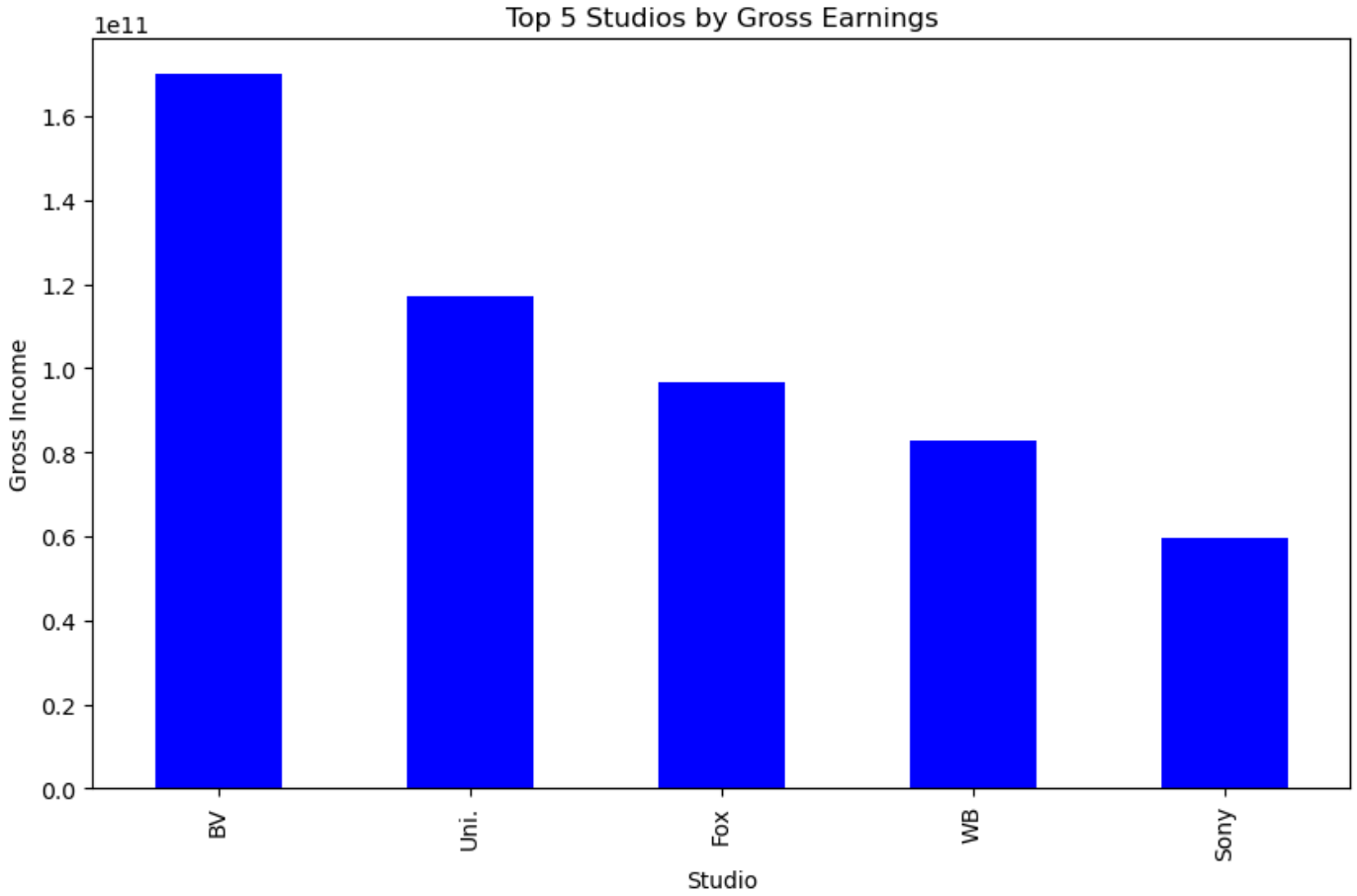
189 rows × 6 columns

In [138…

```
#Plotting the bar chart to show the top 5 studios
gross_per_studio.head().plot(kind="bar", x="Studio", y="Gross_Income",
                      figsize=(10,6), legend=None, color="blue")

# set the labels and title of the plot
plt.xlabel("Studio")
plt.ylabel("Gross Income")
plt.title('Top 5 Studios by Gross Earnings')

# show the plot
plt.show()
```



ASSESMENT

Throughout this project, I've conducted a comprehensive analysis of various datasets related to the movie industry, aiming to provide valuable insights and recommendations for Microsoft's potential expansion into filmmaking.

Data Collection and Preparation I collected data from multiple sources, including Box Office Mojo, The Movie Database (TMDB), and IMDb, and then performed data cleaning and preprocessing steps to handle missing values, convert data types, and merge relevant datasets.

Data Analysis In my analysis, I identified the highest-grossing movies to understand successful movie types that Microsoft could potentially replicate. Additionally, I delved into genre-wise analysis to evaluate movie performance across different genres in terms of quantity, revenue, and ratings. This analysis helped me highlight genres with the most potential for investment. Moreover, I explored the significance of studio performance in terms of revenue generation, providing insights into potential collaborations or partnerships for content creation.

Visualization To present my findings effectively, I utilized visualizations such as bar plots and histograms. For example, I visualized the top studios by gross earnings and movies by genre with ratings above 5.

Key Findings Based on my analysis, Avengers: Infinity War emerged as the highest-grossing movie, with significant earnings in the Action, Adventure, and Sci-Fi genres. I also identified lucrative genres such as Action, Adventure, and Sci-Fi, indicating potential areas for Microsoft's investment. Additionally, I highlighted top-performing studios, including Buena Vista, Universal Pictures, Fox Studios, Warner Brothers, and Sony Pictures, based on their gross earnings.

Recommendations

Moving forward, Microsoft could consider replicating successful movie types identified through my analysis, with a focus on genres like Action, Adventure, and Sci-Fi. Exploring collaboration or partnership opportunities with top-performing studios could also be beneficial to leverage their expertise and resources for content creation. By investing in genres with high potential and forging strategic partnerships, Microsoft can enhance its position in the movie industry.

Conclusion

In conclusion, my analysis provides valuable insights into the movie industry landscape, guiding Microsoft's potential entry into filmmaking. By leveraging data-driven strategies and partnerships, Microsoft can navigate the competitive landscape and capitalize on emerging opportunities in the entertainment industry.