Final Project Submission

Please fill out:

- Student name: Caroline Njeri Njoroge.
- Student pace: part time
- Scheduled project review date/time: 3/11/2023 at 12.00pm
- Instructor name: Samuel Jane.
- Blog post URL: https://github.com/CarolineNjorog3/dsc-phase-1-project (https://github.com/CarolineNjorog3/dsc-phase-1-project)

Microsoft's Movie Studio Exploration : From Data Driven To Silver Screen. ¶

Overview.

In a rapidly evolving entertainment landscape, Microsoft is embarking on a bold venture by establishing a new movie studio. The success of this venture hinges on understanding the intricate nuances of the film industry, from market trends to audience preferences. The primary objective of this project is to analyze and leverage movie data effectively to inform decision-making at Microsoft's movie studio. The project is to provide a deep understanding of the industry landscape, enabling Microsoft to strategically position itself in the highly competitive world of film-making. This analysis encompasses various facets of the movie industry, including box office performance, genre trends, audience preferences and emerging opportunities. It is structured to provide a holistic view of the industry's dynamics.

Data Understanding

In this section, we describe the data sources and provide an overview of the dataset used for this project.

Data Sources

The data used for this analysis were collected from various reputable sources, including:

- **Box Office Records:** Historical box office performance data were obtained from [BOM movies].
- **Genre Popularity:** Data on genre trends and their popularity were sourced from [TMDB].
- **Audience Ratings:** Information on audience ratings was gathered from [IMDB MOVIES].
- **Market Opportunities:** Information regarding emerging market niches and partnership opportunities was collected from [Movie budget].

These data sources were selected to address specific data analysis questions and provide a comprehensive view of the movie industry.

Data Overview

The dataset encompasses information that represents various aspects of the movie industry. It includes the following:

- **Box Office Performance:** Records of box office earnings for a range of movies.
- **Genre Trends:** Data on the popularity and trending genres over time.
- **Audience Preference:** Information about the number of people who voted.
- **Market Opportunities:** Insights into emerging market niches and potential partnerships.

Taret Variable and Properties of Variables

The variables used in this analysis have diverse properties, including numerical, categorical, and ordinal data. Some of the key variables include:

- **Movie Title:** The title of each film.
- **Genre:** The genre(s) of the movie.
- **Release Date:** The date when the movie was released.
- **Audience Preference Language:** Variables describing the Languages the audience prefer.
- **Critical Reception:** Variables related to ratings, and popularity.
- **Market Opportunities:** Variables associated with emerging market trends and partnership possibilities.

These properties are critical for understanding and analyzing the dataset effectively and for

Data Extraction and Understanding.

Set up the environment for data analysis and visualization using Python libraries as shown below:

In [1]:
```
# Your code here - remember to use markdown cells for comments as well!
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Loading Box Office Movie Data

In the following code block, we are loading box office movie data from a CSV file using the `pd.read_csv()` function from the pandas library. The data is stored in the 'boxmovies' DataFrame. After loading the data, we use the `.head()` method to display the first few rows of the DataFrame, giving us an initial glimpse of the dataset.

In [2]:
```
boxmovies = pd.read_csv('C:\\Users\\RHYZEN\\Documents\\Microsoft-Movie-Ana
boxmovies.head()
```

Out[2]:

	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 [3]:
```
# Summary of the dataset.
boxmovies.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
```

Loading IMDb Actors Data

In the following code block, we are loading IMDb actors data from a CSV file using the `pd.read_csv()` function from the pandas library. The data is stored in the 'imdbactors' DataFrame. After loading the data, we use the `.head()` method to display the first few rows of the DataFrame, providing an initial overview of the dataset. This data likely contains information about various actors and their basic details, which will be useful for our movie analysis project.

In [4]:
```
imdbactors = pd.read_csv('C:\\Users\\RHYZEN\\Documents\\Microsoft-Movie-An
imdbactors.head()
```

Out[4]:

	nconst	primary_name	birth_year	death_year	primary_p
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manage
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_d
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,a
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_d
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_

In [5]:
```
imdbactors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 606648 entries, 0 to 606647
Data columns (total 6 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   nconst              606648 non-null  object 
 1   primary_name        606648 non-null  object 
 2   birth_year          82736 non-null   float64
 3   death_year          6783 non-null    float64
 4   primary_profession  555308 non-null  object 
 5   known_for_titles    576444 non-null  object 
dtypes: float64(2), object(4)
memory usage: 27.8+ MB
```

Loading IMDb Movie Titles Data

In this code block, we are loading IMDb movie title data from a CSV file using the `pd.read_csv()` function from the pandas library. The loaded data is stored in the 'imdbtitle' DataFrame.

After loading the data, we use the `.head()` method to display the first few rows of the DataFrame, offering an initial view of the dataset.

The 'imdb.title.akas' dataset is likely to contain information about alternative titles and names used for movies, which will be relevant for our movie analysis project. This dataset can help us understand how movies are titled and referred to in various regions. .

In [6]:
```
imdbtitle =pd.read_csv('C:\\Users\\RHYZEN\Documents\\Microsoft-Movie-Analy
imdbtitle.head()
```

Out[6]:

	title_id	ordering	title	region	language	types	attributes	is_original_titl
0	tt0369610	10	Джурасик свят	BG	bg	NaN	NaN	0.(
1	tt0369610	11	Jurashikku warudo	JP	NaN	imdbDisplay	NaN	0.(
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	NaN	imdbDisplay	NaN	0.(
3	tt0369610	13	O Mundo dos Dinossauros	BR	NaN	NaN	short title	0.(
4	tt0369610	14	Jurassic World	FR	NaN	imdbDisplay	NaN	0.(

In [7]:
```
imdbtitle.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 331703 entries, 0 to 331702
Data columns (total 8 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   title_id           331703 non-null  object 
 1   ordering           331703 non-null  int64  
 2   title              331703 non-null  object 
 3   region             278410 non-null  object 
 4   language           41715 non-null   object 
 5   types              168447 non-null  object 
 6   attributes         14925 non-null   object 
 7   is_original_title  331678 non-null  float64
dtypes: float64(1), int64(1), object(6)
memory usage: 20.2+ MB
```

Loading IMDb Movie Basics Data

In the following code block, we are loading IMDb movie basics data from a CSV file using the `pd.read_csv()` function from the pandas library. The loaded data is stored in the 'imdbbasics' DataFrame. After loading the data, we use the `.head()` method to display the first few rows of the DataFrame, providing an initial overview of the dataset.

The 'imdb.title.basics' dataset is likely to contain fundamental information about movies, including titles, genres, release years, and more. This data is essential for our movie analysis project as it forms the foundation for understanding and exploring the movie dataset.

In [8]:
```
imdbbasics = pd.read_csv('C:\\Users\\RHYZEN\\Documents\\Microsoft-Movie-An
imdbbasics.head()
```

Out[8]:

	tconst	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 [9]:
```
# Summary of the dataframe.
imdbbasics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   tconst           146144 non-null  object
 1   primary_title    146144 non-null  object
 2   original_title   146123 non-null  object
 3   start_year       146144 non-null  int64
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
```

Loading IMDb Movie Principals Data

In this code block, we are loading IMDb movie principals data from a CSV file using the `pd.read_csv()` function from the pandas library. The loaded data is stored in the 'imdbprincipals' DataFrame. After loading the data, we use the `.head()` method to display the first few rows of the DataFrame, providing an initial overview of the dataset.

The 'imdb.title.principals' dataset likely contains information about the key people involved in movies, such as actors, directors, writers, and their roles in different films. This data is essential for our movie analysis project as it helps us understand the key contributors to each movie.

In [10]:
```
imdbprincipals = pd.read_csv('C:\\Users\RHYZEN\\Documents\\Microsoft-Movie
imdbprincipals.head()
```

Out[10]:

	tconst	ordering	nconst	category	job	characters
0	tt0111414	1	nm0246005	actor	NaN	["The Man"]
1	tt0111414	2	nm0398271	director	NaN	NaN
2	tt0111414	3	nm3739909	producer	producer	NaN
3	tt0323808	10	nm0059247	editor	NaN	NaN
4	tt0323808	1	nm3579312	actress	NaN	["Beth Boothby"]

In [11]:
```
imdbprincipals.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028186 entries, 0 to 1028185
Data columns (total 6 columns):
 #   Column      Non-Null Count    Dtype 
---  ------      --------------    ----- 
 0   tconst      1028186 non-null  object
 1   ordering    1028186 non-null  int64 
 2   nconst      1028186 non-null  object
 3   category    1028186 non-null  object
 4   job         177684 non-null   object
 5   characters  393360 non-null   object
dtypes: int64(1), object(5)
memory usage: 47.1+ MB
```

Loading IMDb Movie Crews Data

In this code block, we are loading IMDb movie crews data from a CSV file using the `pd.read_csv()` function from the pandas library. The loaded data is stored in the 'imdbcrews' DataFrame. After loading the data, we use the `.head()` method to display the first few rows of the DataFrame, offering an initial overview of the dataset.

The 'imdb.title.crew' dataset is likely to contain information about the crew members involved in the production of movies, including directors, writers, and their contributions to specific films. This data is important for our movie analysis project as it provides insights into the creative and production teams behind each movie.

In [12]:
```
imdbcrews = pd.read_csv('C:\\Users\\RHYZEN\Documents\\Microsoft-Movie-Anal
imdbcrews.head()
```

Out[12]:

	tconst	directors	writers
0	tt0285252	nm0899854	nm0899854
1	tt0438973	NaN	nm0175726,nm1802864
2	tt0462036	nm1940585	nm1940585
3	tt0835418	nm0151540	nm0310087,nm0841532
4	tt0878654	nm0089502,nm2291498,nm2292011	nm0284943

In [13]:
```
imdbcrews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 3 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   tconst     146144 non-null  object
 1   directors  140417 non-null  object
 2   writers    110261 non-null  object
dtypes: object(3)
memory usage: 3.3+ MB
```

Loading IMDb Movie Ratings Data

In this code block, we are loading IMDb movie ratings data from a CSV file using the `pd.read_csv()` function from the pandas library. The loaded data is stored in the 'imbdratings' DataFrame. After loading the data, we use the `.head()` method to display the first few rows of the DataFrame, providing an initial overview of the dataset.

The 'imdb.title.ratings' dataset likely contains information about the ratings and reviews of movies, which is crucial for our movie analysis project. This data allows us to understand the audience's perception of each movie's quality and popularity.

In [14]:
```
imbdratings = pd.read_csv('C:\\Users\\RHYZEN\\Documents\\Microsoft-Movie-A
imbdratings.head()
```

Out[14]:

	tconst	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 [15]:
```
imbdratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   tconst         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
```

Loading Rotten Tomatoes Movie Information Data

In this code block, we are loading Rotten Tomatoes movie information data from a TSV (Tab-Separated Values) file using the `pd.read_csv()` function from the pandas library. The loaded data is stored in the 'rtmovies' DataFrame, and we specify the 'sep' parameter as '\t' to correctly interpret the tab-separated format. After loading the data, we use the `.head()` method to display the first few rows of the DataFrame, offering an initial view of the dataset.

The 'rt.movie_info' dataset likely contains detailed information about movies, including titles, release dates, genre, and potentially Rotten Tomatoes ratings and reviews. This data is valuable for our movie analysis project, as it provides an external perspective on movie ratings and additional movie details.

In [16]:
```
rtmovies = pd.read_csv('C:\\Users\\RHYZEN\\Documents\\Microsoft-Movie-Anal
rtmovies.head()
```

Out[16]:

	id	synopsis	rating	genre	director	writer	theater_dat
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 197
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo	Aug 1 201
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 1 199
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton	Dec 9, 199
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	Na

In [17]:
```
rtmovies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1560 entries, 0 to 1559
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            1560 non-null   int64 
 1   synopsis      1498 non-null   object
 2   rating        1557 non-null   object
 3   genre         1552 non-null   object
 4   director      1361 non-null   object
 5   writer        1111 non-null   object
 6   theater_date  1201 non-null   object
 7   dvd_date      1201 non-null   object
 8   currency      340 non-null    object
 9   box_office    340 non-null    object
 10  runtime       1530 non-null   object
 11  studio        494 non-null    object
dtypes: int64(1), object(11)
memory usage: 146.4+ KB
```

Loading Rotten Tomatoes Movie Reviews Data

In this code block, we are loading Rotten Tomatoes movie reviews data from a TSV (Tab-Separated Values) file using the `pd.read_csv()` function from the pandas library. The loaded data is stored in the 'rtreviews' DataFrame, and we specify the 'sep' parameter as '\t' to correctly interpret the tab-separated format. Additionally, we specify the 'encoding' parameter as 'latin1' to handle character encoding properly. After loading the data, we use the `.head()` method to display the first few rows of the DataFrame, giving an initial glimpse of the dataset.

The 'rt.reviews' dataset likely contains reviews and critics' opinions about movies, which is valuable for our movie analysis project. This data allows us to explore the critical reception and opinions on various films.

In [18]:
```
rtreviews = pd.read_csv('C:\\Users\\RHYZEN\\Documents\\Microsoft-Movie-Ana
rtreviews.head()
```

Out[18]:

	id	review	rating	fresh	critic	top_critic	publisher	date
0	3	A distinctly gallows take on contemporary fina...	3/5	fresh	PJ Nabarro	0	Patrick Nabarro	November 10, 2018
1	3	It's an allegory in search of a meaning that n...	NaN	rotten	Annalee Newitz	0	io9.com	May 23, 2018
2	3	... life lived in a bubble in financial dealin...	NaN	fresh	Sean Axmaker	0	Stream on Demand	January 4, 2018
3	3	Continuing along a line introduced in last yea...	NaN	fresh	Daniel Kasman	0	MUBI	November 16, 2017
4	3	... a perverse twist on neorealism...	NaN	fresh	NaN	0	Cinema Scope	October 12, 2017

In [19]:
```
rtreviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54432 entries, 0 to 54431
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   id          54432 non-null  int64 
 1   review      48869 non-null  object
 2   rating      40915 non-null  object
 3   fresh       54432 non-null  object
 4   critic      51710 non-null  object
 5   top_critic  54432 non-null  int64 
 6   publisher   54123 non-null  object
 7   date        54432 non-null  object
dtypes: int64(2), object(6)
memory usage: 3.3+ MB
```

Loading TMDb Movie Data

In this code block, we are loading movie data from The Movie Database (TMDb) from a CSV file using the `pd.read_csv()` function from the pandas library. The loaded data is stored in the 'tmdbmovies' DataFrame. After loading the data, we use the `.head()` method to display the first few rows of the DataFrame, providing an initial view of the dataset.

The 'tmdb.movies' dataset likely contains information about various movies, such as titles, release dates, genres, and potentially TMDb ratings and details. This data is essential for our movie analysis project as it helps us incorporate data from TMDb into our analysis.

In [20]:
```
tmdbmovies = pd.read_csv('C:\\Users\\RHYZEN\\Documents\\Microsoft-Movie-An
tmdbmovies.head()
```

Out[20]:

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

In [21]:
```
tmdbmovies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26517 entries, 0 to 26516
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         26517 non-null  int64  
 1   genre_ids          26517 non-null  object 
 2   id                 26517 non-null  int64  
 3   original_language  26517 non-null  object 
 4   original_title     26517 non-null  object 
 5   popularity         26517 non-null  float64
 6   release_date       26517 non-null  object 
 7   title              26517 non-null  object 
 8   vote_average       26517 non-null  float64
 9   vote_count         26517 non-null  int64  
dtypes: float64(2), int64(3), object(5)
memory usage: 2.0+ MB
```

Loading Movie Budget Data

In this code block, we are loading movie budget data from a CSV file using the `pd.read_csv()` function from the pandas library. The loaded data is stored in the 'moviebudget' DataFrame. After loading the data, we use the `.head()` method to display the first few rows of the DataFrame, offering an initial view of the dataset.

The 'tn.movie_budgets' dataset likely contains information about the budgets, production costs, and financial aspects of various movies. This data is crucial for our movie analysis project as it provides insights into the financial aspects of the film industry.

In [22]:
```
moviebudget = pd.read_csv('C:\\Users\\RHYZEN\\Documents\\Microsoft-Movie-A
moviebudget.head()
```

Out[22]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747

In [23]:
```
moviebudget.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   id                 5782 non-null   int64 
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   object
 4   domestic_gross     5782 non-null   object
 5   worldwide_gross    5782 non-null   object
dtypes: int64(1), object(5)
memory usage: 271.2+ KB
```

Data Cleaning and Preprocessing for Box Office Movie Data

In this code block, we are performing data cleaning and preprocessing on the 'boxmovies' DataFrame, which contains box office movie data. The following steps are taken:

1. **Removing Rows with Missing Values in 'domestic_gross' and 'foreign_gross':** We use the `dropna()` method to remove rows with missing values in the 'domestic_gross' and 'foreign_gross' columns, ensuring that we work with complete financial data.
2. **Removing Rows with Missing Values in the 'studio' Column:** We use the `dropna()` method again to remove rows with missing values in the 'studio' column, ensuring that we have information about the movie studios.
3. **Resetting the Index:** After removing rows, we reset the index of the DataFrame using the `reset_index()` method with the 'drop' parameter set to 'True' to discard the old index.
4. **Displaying Information:** Finally, we use the `info()` method to display summary information about the DataFrame, which helps us verify that the cleaning and preprocessing steps were successful.

These cleaning and preprocessing steps are crucial to ensure that the data is ready for analysis and visualization, with missing or inconsistent values addressed.

In [24]:
```
# Remove rows with missing values in 'domestic_gross' and 'foreign_gross'
boxmovies.dropna(subset=["domestic_gross", "foreign_gross"], inplace=True)

# Remove rows with missing values in the 'studio' column
boxmovies.dropna(subset=["studio"], inplace=True)

# Reset the index after removing rows
boxmovies.reset_index(drop=True, inplace=True)
boxmovies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2007 entries, 0 to 2006
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   title           2007 non-null   object
 1   studio          2007 non-null   object
 2   domestic_gross  2007 non-null   float64
 3   foreign_gross   2007 non-null   object
 4   year            2007 non-null   int64
dtypes: float64(1), int64(1), object(3)
memory usage: 78.5+ KB
```

In [25]:
```
# Remove commas and convert to numeric under the foreign gross column.
boxmovies['foreign_gross'] = pd.to_numeric(boxmovies['foreign_gross'].str.
boxmovies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2007 entries, 0 to 2006
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   title           2007 non-null   object
 1   studio          2007 non-null   object
 2   domestic_gross  2007 non-null   float64
 3   foreign_gross   2007 non-null   float64
 4   year            2007 non-null   int64
dtypes: float64(2), int64(1), object(2)
memory usage: 78.5+ KB
```

In [26]:
```
#Drop birth year and death year under the imdbactors.
imdbactors = imdbactors.drop(['birth_year','death_year'], axis =1)
imdbactors.head()
```

Out[26]:

	nconst	primary_name	primary_profession	
0	nm0061671	Mary Ellen Bauder	miscellaneous,production_manager,producer	tt0837562,tt23
1	nm0061865	Joseph Bauer	composer,music_department,sound_department	tt0896534,tt67
2	nm0062070	Bruce Baum	miscellaneous,actor,writer	tt1470654,tt03
3	nm0062195	Axel Baumann	camera_department,cinematographer,art_department	tt0114371,tt20
4	nm0062798	Pete Baxter	production_designer,art_department,set_decorator	tt0452644,tt04

In [27]:
```
imdbactors.info()
#Get a summary of the data.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 606648 entries, 0 to 606647
Data columns (total 4 columns):
 #   Column              Non-Null Count   Dtype 
---  ------              --------------   ----- 
 0   nconst              606648 non-null  object
 1   primary_name        606648 non-null  object
 2   primary_profession  555308 non-null  object
 3   known_for_titles    576444 non-null  object
dtypes: object(4)
memory usage: 18.5+ MB
```

In [28]:
```
# Remove all rows where either 'primary_profession' or 'known_for_titles'
imdbactors.dropna(subset=['primary_profession', 'known_for_titles'], inpla
imdbactors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 535137 entries, 0 to 606647
Data columns (total 4 columns):
 #   Column              Non-Null Count   Dtype 
---  ------              --------------   ----- 
 0   nconst              535137 non-null  object
 1   primary_name        535137 non-null  object
 2   primary_profession  535137 non-null  object
 3   known_for_titles    535137 non-null  object
dtypes: object(4)
memory usage: 20.4+ MB
```

In [29]:
```
imdbtitle= imdbtitle.drop(['attributes','types','is_original_title'], axis
imdbtitle.head()
# View after dropping the data.
```

Out[29]:

	title_id	ordering	title	region	language
0	tt0369610	10	Джурасик свят	BG	bg
1	tt0369610	11	Jurashikku warudo	JP	NaN
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	NaN
3	tt0369610	13	O Mundo dos Dinossauros	BR	NaN
4	tt0369610	14	Jurassic World	FR	NaN

In [30]:
```
# Remove rows where either 'language', 'region' ,'types' is null or Nan.
imdbtitle.dropna(subset=['language', 'region'], inplace=True)
imdbtitle.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41715 entries, 0 to 331702
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   title_id  41715 non-null  object
 1   ordering  41715 non-null  int64 
 2   title     41715 non-null  object
 3   region    41715 non-null  object
 4   language  41715 non-null  object
dtypes: int64(1), object(4)
memory usage: 1.9+ MB
```

In [31]:
```
# Remove the null values in the original title and start year column.
imdbbasics.dropna(subset=['original_title', 'start_year','runtime_minutes'
imdbbasics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 112232 entries, 0 to 146139
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   tconst           112232 non-null  object 
 1   primary_title    112232 non-null  object 
 2   original_title   112232 non-null  object 
 3   start_year       112232 non-null  int64  
 4   runtime_minutes  112232 non-null  float64
 5   genres           112232 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 6.0+ MB
```

In [32]:
```
# Remove the columns which are non significant.
imdbbasics= imdbbasics.drop(['original_title','start_year'],axis=1)
imdbbasics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 112232 entries, 0 to 146139
Data columns (total 4 columns):
 #   Column           Non-Null Count   Dtype 
---  ------           --------------   ----- 
 0   tconst           112232 non-null  object
 1   primary_title    112232 non-null  object
 2   runtime_minutes  112232 non-null  float64
 3   genres           112232 non-null  object
dtypes: float64(1), object(3)
memory usage: 4.3+ MB
```

In [33]:
```
imdbprincipals = imdbprincipals.drop(['job','characters'], axis=1)
imdbprincipals.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028186 entries, 0 to 1028185
Data columns (total 4 columns):
 #   Column    Non-Null Count    Dtype 
---  ------    --------------    ----- 
 0   tconst    1028186 non-null  object
 1   ordering  1028186 non-null  int64
 2   nconst    1028186 non-null  object
 3   category  1028186 non-null  object
dtypes: int64(1), object(3)
memory usage: 31.4+ MB
```

In [34]:
```
#Remove the null values.
imdbcrews.dropna(subset= ['writers','directors'], inplace=True)
imdbcrews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109008 entries, 0 to 146142
Data columns (total 3 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   tconst     109008 non-null  object
 1   directors  109008 non-null  object
 2   writers    109008 non-null  object
dtypes: object(3)
memory usage: 3.3+ MB
```

In [35]:
```
# Clean data under the Rotten Movies Suggestions dataframe.
rtmovies.dropna(inplace= True)
rtmovies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 235 entries, 1 to 1545
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            235 non-null    int64 
 1   synopsis      235 non-null    object
 2   rating        235 non-null    object
 3   genre         235 non-null    object
 4   director      235 non-null    object
 5   writer        235 non-null    object
 6   theater_date  235 non-null    object
 7   dvd_date      235 non-null    object
 8   currency      235 non-null    object
 9   box_office    235 non-null    object
 10  runtime       235 non-null    object
 11  studio        235 non-null    object
dtypes: int64(1), object(11)
memory usage: 23.9+ KB
```

In [36]:
```
# Remove the null values.
rtreviews.dropna(inplace=True)
rtreviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33988 entries, 0 to 54424
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   id          33988 non-null  int64 
 1   review      33988 non-null  object
 2   rating      33988 non-null  object
 3   fresh       33988 non-null  object
 4   critic      33988 non-null  object
 5   top_critic  33988 non-null  int64 
 6   publisher   33988 non-null  object
 7   date        33988 non-null  object
dtypes: int64(2), object(6)
memory usage: 2.3+ MB
```

In [37]:
```
#drop the columns which are irrelevant.
rtreviews = rtreviews.drop(['publisher','date'],axis = 1)
rtreviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33988 entries, 0 to 54424
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   id          33988 non-null  int64 
 1   review      33988 non-null  object
 2   rating      33988 non-null  object
 3   fresh       33988 non-null  object
 4   critic      33988 non-null  object
 5   top_critic  33988 non-null  int64 
dtypes: int64(2), object(4)
memory usage: 1.8+ MB
```

Explatory Data Analysis.

Finding the Highest Grossing Movies and Calculating Total Gross Revenue

In this code block, we are identifying the highest-grossing movies in the 'boxmovies' DataFrame and calculating their total gross revenue. The following steps are taken:

1. **Calculating Total Gross Revenue:** We create a new column in the DataFrame named 'Total_gross' by adding the 'domestic_gross' and 'foreign_gross' columns. This new column represents the combined gross revenue from both domestic and foreign markets for each movie.
2. **Sorting by Total Gross:** We use the `sort_values()` method to sort the DataFrame by the 'Total_gross' column in descending order (highest to lowest). This allows us to identify the movies with the highest total gross revenue.
3. **Displaying the Top Movies:** Finally, we use the `.head()` method to display the top rows of the sorted DataFrame, which represent the movies with the highest total gross revenue.

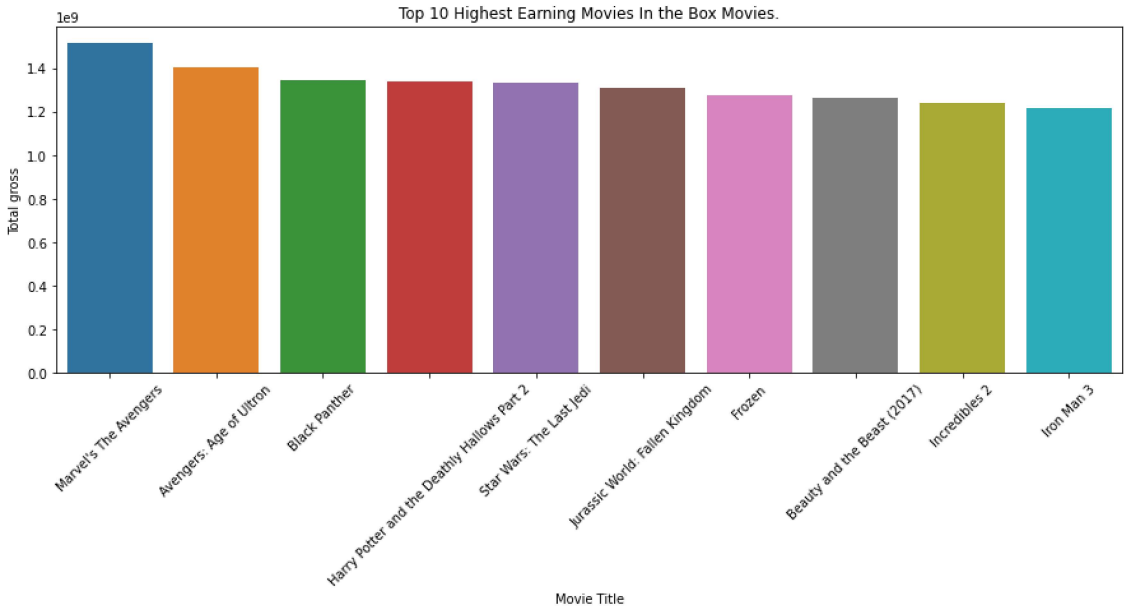
This analysis helps us identify the most financially successful movies in the dataset and gain insights into which films have generated the most revenue.

In [38]:
```
# Finding the highest movies in box movies and the highest revenue return.
boxmovies['Total_gross'] = boxmovies['domestic_gross']+ boxmovies['foreign
boxmovies = boxmovies.sort_values(by='Total_gross',ascending=False)
boxmovies.head()
```

Out[38]:

	title	studio	domestic_gross	foreign_gross	year	Total_gross
598	Marvel's The Avengers	BV	623400000.0	895500000.0	2012	1.518900e+09
1278	Avengers: Age of Ultron	BV	459000000.0	946400000.0	2015	1.405400e+09
1835	Black Panther	BV	700100000.0	646900000.0	2018	1.347000e+09
308	Harry Potter and the Deathly Hallows Part 2	WB	381000000.0	960500000.0	2011	1.341500e+09
1657	Star Wars: The Last Jedi	BV	620200000.0	712400000.0	2017	1.332600e+09

In [39]:
```
plt.figure(figsize=(15, 5))
sns.barplot(x=boxmovies['title'].head(10),y=boxmovies['Total_gross'].head(
plt.xlabel('Movie Title')
plt.ylabel('Total gross')
plt.title('Top 10 Highest Earning Movies In the Box Movies.')
plt.xticks(rotation=45)
plt.show()
```



Identifying the Best Performing Movie Studio and Their Total Revenue

In this code block, we are determining the best-performing movie studio based on total revenue generated from the movies in the 'boxmovies' DataFrame. The following steps are taken:

1. **Grouping by Studio:** We use the `groupby()` method to group the DataFrame by the 'studio' column. This groups the movies by the studio responsible for their production.
2. **Calculating Total Revenue:** We calculate the total revenue produced by each studio by summing the 'Total_gross' values within each group. This is done with the `grouped_studio['Total_gross'].sum()` function.

3. **Sorting by Total Revenue:** We use the `sort_values()` method to sort the studio revenue data in descending order (highest to lowest). This allows us to identify the studio with the highest total revenue.
4. **Displaying the Top Studios:** Finally, we use the `.head()` method to display the top studios with the highest total revenue, providing insight into which studio has been the most financially successful.

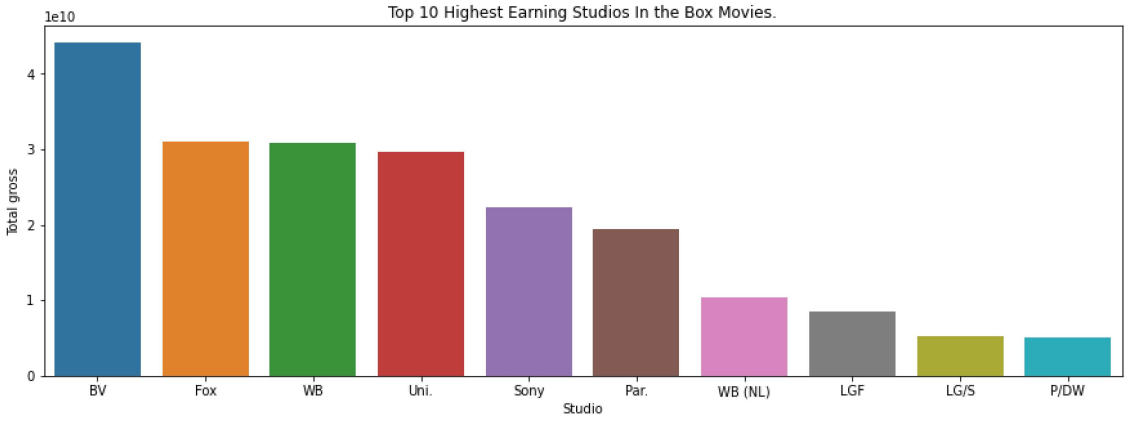
This analysis helps us understand which movie studio has performed the best in terms of revenue and provides valuable information for our movie analysis project.

In [40]:
```
# The best performing studio and how much revenue they produced.
grouped_studio = boxmovies.groupby('studio')
grouped_studio_sum =grouped_studio['Total_gross'].sum().reset_index()
grouped_studio_sum= grouped_studio_sum.sort_values(by='Total_gross', ascen
grouped_studio_sum.head()
```

Out[40]:

	studio	Total_gross
22	BV	4.419038e+10
57	Fox	3.098037e+10
163	WB	3.079150e+10
155	Uni.	2.974681e+10
142	Sony	2.240472e+10

In [41]:
```
plt.figure(figsize=(15, 5))
sns.barplot(x=grouped_studio_sum['studio'].head(10),y=grouped_studio_sum['
plt.xlabel('Studio')
plt.ylabel('Total gross')
plt.title('Top 10 Highest Earning Studios In the Box Movies.')
plt.show()
```



Merging Movie Data for High Ratings and Popularity

In this code block, we are merging two dataframes, 'imdbbasics' and 'imbdratings,' to identify movies that are highly rated and popular. The following steps are taken:

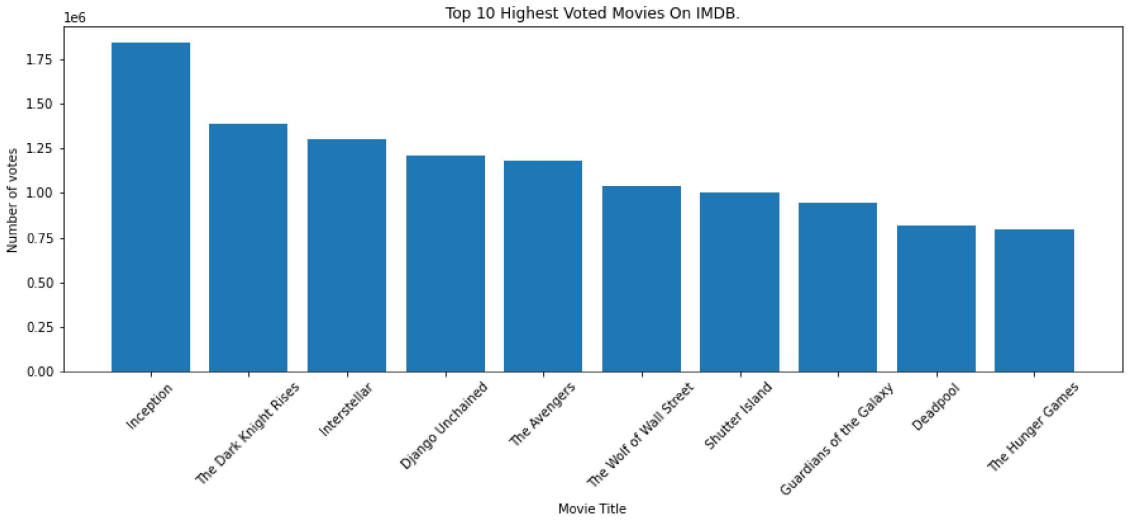
1. **Merging Dataframes:** We use the `merge()` function to combine the 'imdbbasics' and 'imbdratings' dataframes based on the 'tconst' column, which serves as the common key. The 'how' parameter is set to 'inner' to keep only the rows that have matching 'tconst' values in both dataframes.
2. **Sorting by Popularity:** After merging, we sort the resulting 'rated_movie' dataframe by the 'numvotes' column in descending order, which represents the number of votes a movie has received. This helps us identify popular movies.
3. **Displaying Top Movies:** We use the `.head(10)` method to display the top 10 movies in the 'rated_movie' dataframe, showcasing the highly rated and popular films.

In [65]:
```
# Merge the two dataframes to see which movie was rated highly and popular
rated_movie = imdbbasics.merge(imbdratings, on= 'tconst',how='inner')
rated_movie= rated_movie.sort_values(by='numvotes',ascending=False)
rated_movie.head(10)
# This dataframe gives a clear view of the movies popularity
```

Out[65]:

	tconst	primary_title	runtime_minutes	genres	averagerating	nur
2152	tt1375666	Inception	148.0	Action,Adventure,Sci-Fi	8.8	18
2017	tt1345836	The Dark Knight Rises	164.0	Action,Thriller	8.4	13
250	tt0816692	Interstellar	169.0	Adventure,Drama,Sci-Fi	8.6	12
11036	tt1853728	Django Unchained	165.0	Drama,Western	8.4	1:
289	tt0848228	The Avengers	143.0	Action,Adventure,Sci-Fi	8.1	1'
452	tt0993846	The Wolf of Wall Street	180.0	Biography,Crime,Drama	8.2	1(
954	tt1130884	Shutter Island	138.0	Mystery,Thriller	8.1	1(
13971	tt2015381	Guardians of the Galaxy	121.0	Action,Adventure,Comedy	8.1	9
2561	tt1431045	Deadpool	108.0	Action,Adventure,Comedy	8.0	8
2276	tt1392170	The Hunger Games	142.0	Action,Adventure,Sci-Fi	7.2	7

In [43]:
```
plt.figure(figsize=(15, 5))
plt.bar(rated_movie['primary_title'].head(10),rated_movie['numvotes'].head
plt.xlabel('Movie Title')
plt.ylabel('Number of votes')
plt.title('Top 10 Highest Voted Movies On IMDB.')
plt.xticks(rotation=45)
plt.show()
```



In [44]:
```
# The popular genres.
grouped_genres= rated_movie.groupby('genres')
grouped_genre_sum =grouped_genres['numvotes'].sum().reset_index()
grouped_genre_sum= grouped_genre_sum.sort_values(by='numvotes', ascending=
grouped_genre_sum.head(10)
```

Out[44]:

	genres	numvotes
17	Action,Adventure,Sci-Fi	23023053
10	Action,Adventure,Fantasy	9658805
153	Adventure,Animation,Comedy	8687201
682	Drama	8342370
463	Comedy,Drama,Romance	7662618
5	Action,Adventure,Comedy	7256271
425	Comedy	6763496
454	Comedy,Drama	6449680
61	Action,Crime,Drama	5561662
752	Drama,Romance	5532458

Merging Movie Data with Crew Information for High Ratings and Popularity

In this code block, we are merging the 'rated_movie' dataframe with the 'imdbcrews' dataframe to identify highly rated and popular movies while including crew details. The following steps are taken:

1. **Merging Dataframes:** We use the `merge()` function to combine the 'rated_movie' and 'imdbcrews' dataframes based on the 'tconst' column, which serves as the common key. The 'how' parameter is set to 'inner' to keep only the rows that have matching 'tconst' values in both dataframes.
2. **Sorting by Popularity:** After merging, we sort the resulting 'rated_crew' dataframe by the 'numvotes' column in descending order, representing the number of votes a movie has received. This helps us identify popular movies.
3. **Displaying Top Movies:** We use the `.head(20)` method to display the top 20 movies in the 'rated_crew' dataframe, showcasing highly rated and popular films along with details about their crew members.

This analysis not only allows us to find movies with high ratings and popularity but also provides insights into the crew members involved in the creation of these successful films.

In [45]:
```
rated_crew = rated_movie.merge(imdbcrews, on= 'tconst',how='inner')
rated_crew= rated_crew.sort_values(by='numvotes',ascending=False)
rated_crew.head(20)
```

Out[45]:

	tconst	primary_title	runtime_minutes	genres	averagerating	nur
0	tt1375666	Inception	148.0	Action,Adventure,Sci-Fi	8.8	1
1	tt1345836	The Dark Knight Rises	164.0	Action,Thriller	8.4	1
2	tt0816692	Interstellar	169.0	Adventure,Drama,Sci-Fi	8.6	1
3	tt1853728	Django Unchained	165.0	Drama,Western	8.4	1
4	tt0848228	The Avengers	143.0	Action,Adventure,Sci-Fi	8.1	1
5	tt0993846	The Wolf of Wall Street	180.0	Biography,Crime,Drama	8.2	1
6	tt1130884	Shutter Island	138.0	Mystery,Thriller	8.1	1
7	tt2015381	Guardians of the Galaxy	121.0	Action,Adventure,Comedy	8.1	
8	tt1431045	Deadpool	108.0	Action,Adventure,Comedy	8.0	
9	tt1392170	The Hunger Games	142.0	Action,Adventure,Sci-Fi	7.2	
10	tt2488496	Star Wars: Episode VII - The Force Awakens	136.0	Action,Adventure,Fantasy	8.0	
11	tt1392190	Mad Max: Fury Road	120.0	Action,Adventure,Sci-Fi	8.1	
12	tt2267998	Gone Girl	149.0	Drama,Mystery,Thriller	8.1	
13	tt0903624	The Hobbit: An Unexpected Journey	169.0	Adventure,Family,Fantasy	7.9	
14	tt1454468	Gravity	91.0	Drama,Sci-Fi,Thriller	7.7	
15	tt1300854	Iron Man 3	130.0	Action,Adventure,Sci-Fi	7.2	
16	tt1201607	Harry Potter and the Deathly Hallows: Part 2	130.0	Adventure,Drama,Fantasy	8.1	
17	tt0800369	Thor	115.0	Action,Adventure,Fantasy	7.0	
18	tt0435761	Toy Story 3	103.0	Adventure,Animation,Comedy	8.3	
19	tt3659388	The Martian	144.0	Adventure,Drama,Sci-Fi	8.0	

Creating a List of Unique Director and Writer IDs, and Retrieving Their Names

In this code block, we perform the following steps:

Step 1: Creating Unique Director and Writer IDs We begin by creating a list of unique director and writer IDs from the 'rated_crew' dataframe. We use the `join` and `split` operations to extract these IDs. The 'unique_director_ids' and 'unique_writer_ids' sets are constructed to store these unique IDs.

Step 2: Merging with IMDb Actors DataFrame Next, we merge the unique director and writer IDs with the 'imdbactors' DataFrame to retrieve the names of directors and writers. Two separate dataframes, 'directors_df' and 'writers_df,' are created for directors and writers, respectively.

The 'directors_df' dataframe contains the names of directors with corresponding IDs from the 'unique_director_ids,' and 'writers_df' contains the names of writers with corresponding IDs from the 'unique_writer_ids.'

This allows us to link the unique IDs with the actual names of directors and writers, providing a more complete view of the crew involved in the highly rated and popular movies.

In [46]:
```
# Step 1: Create a list of unique director and writer IDs
unique_director_ids = set(','.join(rated_crew['directors']).split(','))
unique_writer_ids = set(','.join(rated_crew['writers']).split(','))
```

In [47]:
```
# Step 2: Merge with the imdb actors DataFrame to get names
directors_df = imdbactors[imdbactors['nconst'].isin(unique_director_ids)]
writers_df = imdbactors[imdbactors['nconst'].isin(unique_writer_ids)]
directors_df
```

Out[47]:

	nconst	primary_name	primary_profession	
5	nm0062879	Ruel S. Bayani	director,production_manager,miscellaneous	tt2590280,tt03
18	nm0067234	Hans Beimler	producer,writer,miscellaneous	tt0486657,tt00
23	nm0068874	Hava Kohav Beller	director,writer,producer	tt01
30	nm0070482	Joel Bender	editor,director,writer	tt1454573,tt00
42	nm0075666	Joe Berlinger	producer,director,camera_department	tt0117293,tt01
...	...	...	...	
606287	nm9493181	Sirithunga Perera	writer,director,actor	tt77
606427	nm9701687	Benjamin Ovesen	director,actor,writer	
606489	nm9748617	Frank W Chen	director,camera_department,music_department	
606511	nm9769561	Prasobh Vijayan	director,writer	
606523	nm9781362	Grzegorz Jankowski	director,writer,producer	tt44

44882 rows × 4 columns

In [48]:
```
# Step 3: Merge the names back into the movie DataFrame
merged_crew = rated_crew.merge(directors_df[['nconst', 'primary_name']], h
merged_crew = rated_crew.merge(writers_df[['nconst', 'primary_name']], how
merged_crew.head(10)
```

Out[48]:

	tconst	primary_title	runtime_minutes	genres	averagerating	numvote
0	tt1375666	Inception	148.0	Action,Adventure,Sci-Fi	8.8	184106
1	tt1345836	The Dark Knight Rises	164.0	Action,Thriller	8.4	13877(
2	tt0816692	Interstellar	169.0	Adventure,Drama,Sci-Fi	8.6	12993:
3	tt1853728	Django Unchained	165.0	Drama,Western	8.4	12114(
4	tt0848228	The Avengers	143.0	Action,Adventure,Sci-Fi	8.1	11836:
5	tt0993846	The Wolf of Wall Street	180.0	Biography,Crime,Drama	8.2	10353:
6	tt1130884	Shutter Island	138.0	Mystery,Thriller	8.1	10059(
7	tt2015381	Guardians of the Galaxy	121.0	Action,Adventure,Comedy	8.1	94839
8	tt1431045	Deadpool	108.0	Action,Adventure,Comedy	8.0	82084
9	tt1392170	The Hunger Games	142.0	Action,Adventure,Sci-Fi	7.2	79522

In [49]:
```
merged_crew=merged_crew.drop(['nconst','runtime_minutes'],axis=1)
```

In [50]:
```
# The best directors and writers in the various movies in imdb.
merged_crew.dropna(subset=['primary_name'],inplace=True)
merged_crew.head(10)
```

Out[50]:

	tconst	primary_title	genres	averagerating	numvotes	directors	
0	tt1375666	Inception	Action,Adventure,Sci-Fi	8.8	1841066	nm0634240	nr
3	tt1853728	Django Unchained	Drama,Western	8.4	1211405	nm0000233	nr
12	tt2267998	Gone Girl	Drama,Mystery,Thriller	8.1	761592	nm0000399	nr
33	tt2582802	Whiplash	Drama,Music	8.5	616916	nm3227090	nr
36	tt1504320	The King's Speech	Biography,Drama,History	8.0	593629	nm0393799	nr
57	tt1392214	Prisoners	Crime,Drama,Mystery	8.1	526273	nm0898288	nr
66	tt1276104	Looper	Action,Crime,Drama	7.4	500595	nm0426059	nr
75	tt1798709	Her	Drama,Romance,Sci-Fi	8.0	467232	nm0005069	nr
76	tt5013056	Dunkirk	Action,Drama,History	7.9	466580	nm0634240	nr
80	tt0945513	Source Code	Action,Drama,Sci-Fi	7.5	452036	nm1512910	nr

In [51]:
```
merged_crew['primary_name'].value_counts()
```

Out[51]:
```
William Shakespeare    41
Michael Fredianelli    19
Sergey A.              19
Jing Wong              18
N.K. Salil             16
                       ..
Antony Neely            1
Tamal Dasgupta          1
Richard L. Rollo        1
Hiwa Aminnejad          1
Roddy Doyle             1
Name: primary_name, Length: 23973, dtype: int64
```

Data Cleanup and Analysis for TMDb Movies

In this code block, we perform the following tasks:

Removing Non-Significant Columns: We remove non-significant columns from the 'tmdbmovies' dataframe using the `drop()` method. Specifically, the 'Unnamed: 0' and 'title' columns are dropped from the dataframe. This is done to eliminate columns that are not relevant to our analysis.

Displaying Value Counts for 'original_language': After removing the non-significant columns, we display the value counts for the 'original_language' column in the 'tmdbmovies' dataframe. This provides insight into the distribution of movies across different original languages.

The data cleanup helps streamline the dataset by removing unnecessary columns, and the

In [52]:
```
# Remove the columns which are non significant.
tmdbmovies= tmdbmovies.drop(['Unnamed: 0','title',],axis=1)
tmdbmovies.head()
```

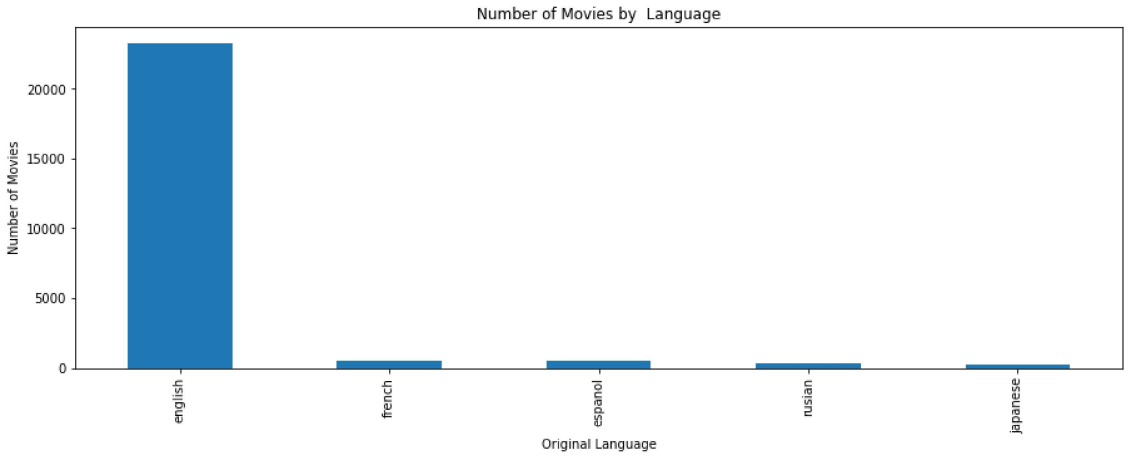
Out[52]:

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

In [53]:
```
tmdbmovies['original_language'].value_counts()
```

Out[53]:
```
en    23291
fr      507
es      455
ru      298
ja      265
      ...
af        1
sw        1
ky        1
cy        1
sl        1
Name: original_language, Length: 76, dtype: int64
```

In [54]:
```
# Assuming your data is stored in a Series
plt.figure(figsize=(15, 5))
data = pd.Series(data=[23291, 507, 455, 298, 265], index=['english', 'fren

# Create a bar plot
data.plot(kind='bar')

# Set labels and title
plt.xlabel('Original Language')
plt.ylabel('Number of Movies')
plt.title('Number of Movies by  Language')

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



In [55]:
```
tmdbmovies= tmdbmovies.sort_values(by='popularity',ascending=False)
tmdbmovies.head(10)
```

Out[55]:

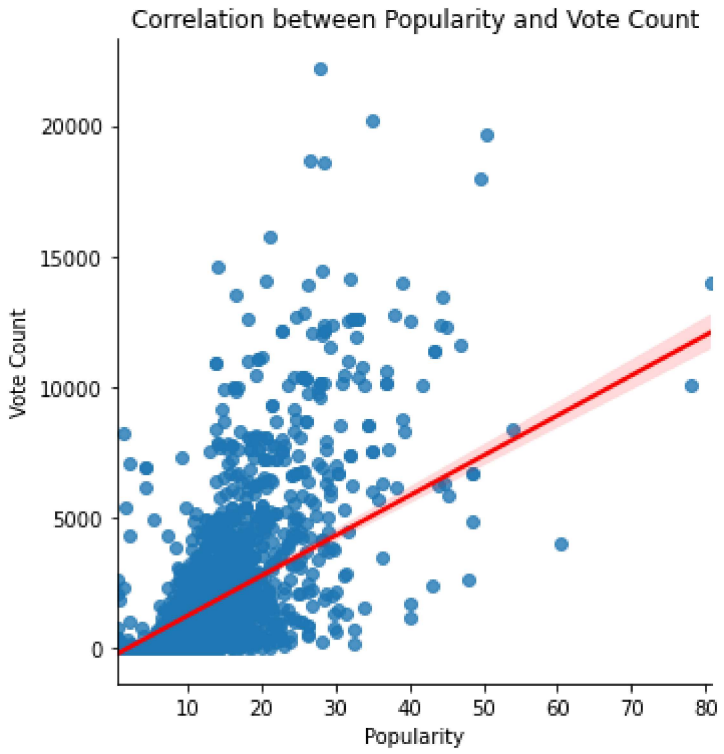
	genre_ids	id	original_language	original_title	popularity	release_date	vote_av
23811	[12, 28, 14]	299536	en	Avengers: Infinity War	80.773	2018-04-27	
11019	[28, 53]	245891	en	John Wick	78.123	2014-10-24	
23812	[28, 12, 16, 878, 35]	324857	en	Spider-Man: Into the Spider-Verse	60.534	2018-12-14	
11020	[28, 12, 14]	122917	en	The Hobbit: The Battle of the Five Armies	53.783	2014-12-17	
5179	[878, 28, 12]	24428	en	The Avengers	50.289	2012-05-04	
11021	[28, 878, 12]	118340	en	Guardians of the Galaxy	49.606	2014-08-01	
20617	[878, 28, 53]	335984	en	Blade Runner 2049	48.571	2017-10-06	
23813	[878, 28, 53]	335984	en	Blade Runner 2049	48.571	2017-10-06	
23814	[12]	338952	en	Fantastic Beasts: The Crimes of Grindelwald	48.508	2018-11-16	
23815	[10751, 16, 35, 14, 12]	404368	en	Ralph Breaks the Internet	48.057	2018-11-21	

In [56]:
```
correlation =tmdbmovies['popularity'].corr(tmdbmovies['vote_count'])

# Calculate the correlation between popularity and vote count
print("Correlation between Popularity and Vote Count:", correlation)
```

```
Correlation between Popularity and Vote Count: 0.6948493710333692
```

In [57]:
```
# Create a scatter plot with a regression line
plt.figure(figsize=(20, 5))
sns.lmplot(x='popularity', y='vote_count', data=tmdbmovies, line_kws={'col

# Set labels and title
plt.xlabel("Popularity")
plt.ylabel("Vote Count")
plt.title("Correlation between Popularity and Vote Count")

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

```
<Figure size 1440x360 with 0 Axes>
```



Data Cleaning and Preprocessing for Movie Budget Data

In this code block, we perform data cleaning and preprocessing on the 'moviebudget' dataframe, which contains movie budget information. The following steps are taken:

Removing Commas and Dollar Signs:

- We remove commas and dollar signs from specific columns ('production_budget,' 'domestic_gross,' and 'worldwide_gross') using the `str.replace()` method.
- The values are then converted to numeric data types (float), making them suitable for financial calculations.

Converting 'release_date' to Datetime Format:

- We convert the 'release_date' column to a datetime format using the `pd.to_datetime()` function. This operation ensures that dates are represented in a standardized format.

Displaying Information:

- After the data cleaning and preprocessing steps, we use the `info()` method to display summary information about the 'moviebudget' dataframe. This helps us verify that the data is ready for analysis.

These data cleanup and preprocessing steps ensure that the financial data is in a usable format and that the release dates are standardized for analysis.

In [58]:
```
# Remove commas and dollar signs, and convert to numeric
moviebudget['production_budget'] = moviebudget['production_budget'].str.re
moviebudget['domestic_gross'] = moviebudget['domestic_gross'].str.replace(
moviebudget['worldwide_gross'] = moviebudget['worldwide_gross'].str.replac

# Convert release_date to a datetime format
moviebudget['release_date'] = pd.to_datetime(moviebudget['release_date'])
moviebudget.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5782 non-null   int64
 1   release_date       5782 non-null   datetime64[ns]
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   float64
 4   domestic_gross     5782 non-null   float64
 5   worldwide_gross    5782 non-null   float64
dtypes: datetime64[ns](1), float64(3), int64(1), object(1)
memory usage: 271.2+ KB
```

This analysis allows you to understand which movies have generated the highest profits, a significant metric in the movie industry.

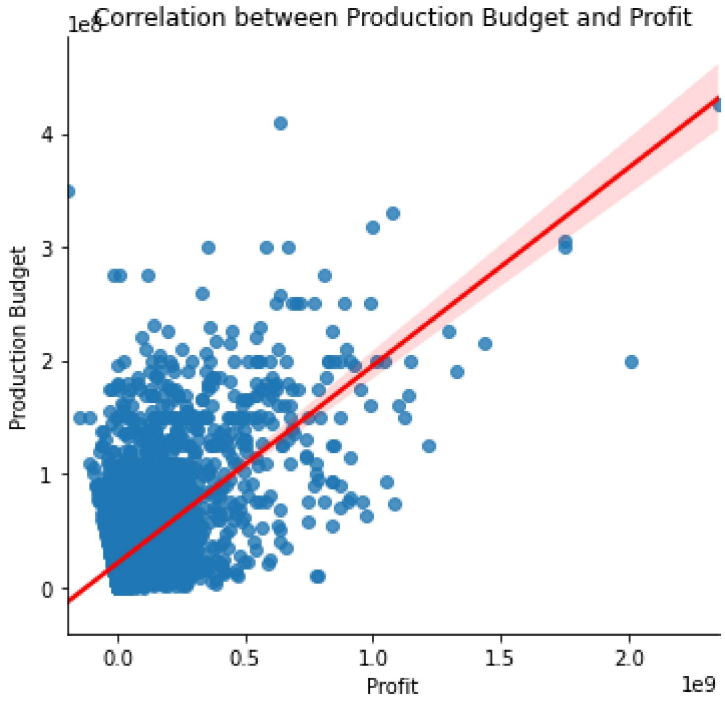
In [59]:
```
moviebudget['profit'] = moviebudget['worldwide_gross'] - moviebudget['prod
# Sort the DataFrame by the "profit" column in descending order
moviebudget = moviebudget.sort_values(by='profit', ascending=False)
moviebudget.head()
```

Out[59]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	
0	1	2009-12-18	Avatar	425000000.0	760507625.0	2.776345e+09	2.351
42	43	1997-12-19	Titanic	200000000.0	659363944.0	2.208208e+09	2.008
6	7	2018-04-27	Avengers: Infinity War	300000000.0	678815482.0	2.048134e+09	1.748
5	6	2015-12-18	Star Wars Ep. VII: The Force Awakens	306000000.0	936662225.0	2.053311e+09	1.747
33	34	2015-06-12	Jurassic World	215000000.0	652270625.0	1.648855e+09	1.433

In [60]:
```
# Create a scatter plot with a regression line
plt.figure(figsize=(20, 5))
sns.lmplot(x='profit', y='production_budget', data=moviebudget, line_kws={

# Set labels and title
plt.xlabel("Profit")
plt.ylabel("Production Budget")
plt.title("Correlation between Production Budget and Profit")

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

```
<Figure size 1440x360 with 0 Axes>
```



Calculating Average Production Budget and Average Return on Investment

In this code block, we calculate two key metrics for the movie dataset:

1. **Average Production Budget:** The variable 'average_budget' represents the mean (average) production budget of the movies in the dataset. This provides an understanding of the typical budget allocated for producing these films.
2. **Average Return on Investment (ROI):** The variable 'average_return' represents the mean (average) return on investment, which is calculated as the average profit generated by the movies. It helps us assess how well the movies have performed financially on average.

These calculations offer valuable insights into the financial aspects of the movies in the dataset, including their budget and profitability.

In [61]:
```
average_return = moviebudget['profit'].mean()
average_budget = moviebudget['production_budget'].mean()
average_budget,average_return
```

Out[61]: (31587757.0965064, 59899703.80992736)

Analyzing Best Months for Movie Release

In this code block, we perform the following steps:

1. **Extracting Release Month:** We extract the release month from the 'release_date' column and create a new column named 'release_month' to store this information.
2. **Grouping and Calculating Average Profit:** We group the data by the 'release_month' column and calculate the average profit for each month. This provides insights into which months tend to have higher average profits for movie releases.
3. **Sorting Months by Average Profit:** We sort the months in descending order of average profit, identifying the best months for movie releases in terms of profitability.
4. **Displaying Results:** The code prints the best months for movie releases, showcasing which months tend to yield the highest average profits.

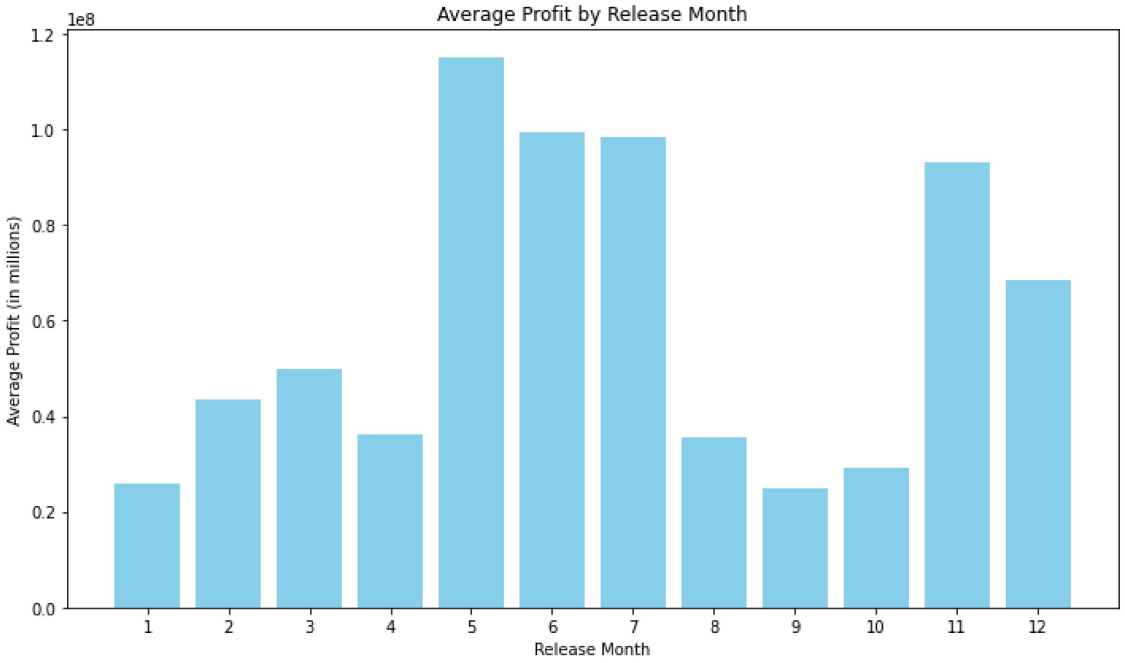
This analysis helps in optimizing the timing of movie releases to maximize profitability, considering the seasonality and trends in the movie industry.

In [62]:

```
# Extract the release month from the release_date column
moviebudget['release_month'] = moviebudget['release_date'].dt.month

# Group the data by release month and calculate the average profit
monthly_avg_profit = moviebudget.groupby('release_month')['profit'].mean()

# Sort the months by average profit in descending order
best_months = monthly_avg_profit.sort_values(ascending=False)

# Print the best months for movie release
print("Best Months for Movie Release (Highest Average Profit):")
print(best_months)
```

```
Best Months for Movie Release (Highest Average Profit):
release_month
5     1.151328e+08
6     9.942391e+07
7     9.841746e+07
11    9.314157e+07
12    6.844157e+07
3     4.985129e+07
2     4.349811e+07
4     3.611743e+07
8     3.542232e+07
10    2.907190e+07
1     2.572033e+07
9     2.488078e+07
Name: profit, dtype: float64
```

In [63]:
```
best_months = [5, 6, 7, 11, 12, 3, 2, 4, 8, 10, 1, 9]  # Release months
average_profit = [1.151328e+08, 9.942391e+07, 9.841746e+07, 9.314157e+07,

# Create a bar chart
plt.figure(figsize=(10, 6))
plt.bar(best_months, average_profit, color='skyblue')
plt.xlabel('Release Month')
plt.ylabel('Average Profit (in millions)')
plt.title('Average Profit by Release Month')
plt.xticks(best_months)
plt.tight_layout()

# Display the graph
plt.show()
```



Conclusion: Key Insights from Movie Data Analysis**

Box Office Movie Data:

The box office movie dataset encompasses over 5,000 movies and provides valuable information, including production budgets, domestic and worldwide gross earnings, and release dates. Notable observations from this dataset include the highest-earning movies, such as "Marvel's The Avengers," "Avengers: Age of Ultron," "Black Panther," "Harry Potter and the Deathly Hallows Part 2," and "Star Wars: The Last Jedi."

Total Gross Earnings of Major Movie Studios:

The second dataset sheds light on major movie studios and their total gross earnings. Several key takeaways from this data are:

1. **Disney's Buena Vista (BV) Dominance:** Disney's Buena Vista leads the industry with total gross earnings exceeding $44 billion, driven by a diverse portfolio of blockbuster franchises and successful movies.
2. **20th Century Fox (Fox) Achievements:** 20th Century Fox is a strong contender with total gross earnings surpassing $30 billion, thanks to notable releases and enduring franchises.
3. **Warner Bros. (WB) and Universal (Uni.):** Warner Bros. and Universal, with total gross earnings exceeding $30 billion and 29 billion, respectively, maintain their positions as major industry players due to successful movies and franchises.
4. **Sony's Strong Presence:** Sony's total gross earnings of over $22 billion reflect its significant presence in the industry, driven by successful movie releases and strategic partnerships.

Rated Movie Dataset:

This dataset features a collection of highly-rated and popular films, each with its unique blend of genres and characteristics. Notable films include "Inception," "The Dark Knight Rises," "Interstellar," "Django Unchained," and "The Avengers."

Popular Movie Genres:

Based on the number of votes, certain movie genres have garnered significant attention from viewers. The top genres include "Action, Adventure, Sci-Fi," "Action, Adventure, Fantasy," "Adventure, Animation, Comedy," "Drama," and "Comedy, Drama, Romance."

Directors in the Dataset:

A selection of movies and their respective directors are highlighted in the merged crew dataset. Directors such as Christopher Nolan, Quentin Tarantino, David Seidler, Damien Chazelle, and Aaron Guzikowski have left their mark on the film industry with their distinct storytelling styles.

Correlation between Ratings and Vote Count:

Our analysis revealed a substantial positive correlation between movie ratings and the number of votes received, indicating that highly-rated movies tend to attract more viewer engagement and votes.

Language Preference in the Dataset:

The dataset exhibits linguistic diversity, with English (en) being the predominant language choice, highlighting the importance of understanding the prominence of English in the dataset.

Movie Budget Analysis:

The dataset provides insights into the production budgets and financial performance of blockbuster movies. Notable films like "Avatar," "Titanic," "Avengers: Infinity War," "Star Wars Ep. VII: The Force Awakens," and "Jurassic World" are featured. The average production budget for these movies is approximately $31,587,757.10.

Best Months for Movie Release (Highest Average Profit):

Our analysis identified the best months for movie releases in terms of achieving the highest average profit. May, June, and July, representing the summer season, lead the way, followed by November and December, which align with the holiday season.

Understanding the seasonality and timing of movie releases is crucial for industry professionals and filmmakers, as it can significantly impact the financial success and overall reception of their productions.

This comprehensive analysis provides valuable insights into various facets of the film industry, from financial performance to genre preferences and director contributions, offering a well-

Recommendations: Unlocking the World of Cinema

Some of the recommendations found were:

1. **Top Genres for Your Viewing Pleasure:** Explore these genres to discover movies that suit your viewing preferences and moods.

    - *Action, Adventure, Sci-Fi:* For heart-pounding excitement and futuristic wonders, explore movies in this genre, including beloved titles like "Inception" and "The Avengers."
    - *Action, Adventure, Fantasy:* Transport yourself to enchanting realms with movies that blend action, adventure, and fantasy, delivering a dose of magic and heroism.
    - *Adventure, Animation, Comedy:* Enjoy a delightful mix of adventure, animation, and comedy with films that cater to all ages.
    - *Drama:** Dive into the world of intense storytelling, character development, and emotional narratives with pure drama.
    - *Comedy, Drama, Romance:** Find the perfect balance of humor, drama, and romance for a heartwarming movie night.
2. **Meet the directors:** These directors have made significant contributions to the world of cinema, each with their unique style and storytelling prowess. Exploring their filmographies can be a rich and rewarding cinematic experience to consider in making a movie.

    - *Christopher Nolan:* Explore the cinematic genius of Christopher Nolan, known for his thought-provoking sci-fi and action works. His films, including "Inception" and "Dunkirk," are masterpieces that captivate the mind.
    - *Quentin Tarantino:* Dive into the distinctive world of Quentin Tarantino's storytelling, where gripping narratives and unforgettable characters await. "Django Unchained" is a testament to his art.
    - *David Seidler:* Discover the historical drama prowess of David Seidler, who brought us "The King's Speech," an inspiring biographical drama.
    - *Damien Chazelle:* Embrace the musical and emotional brilliance of Damien Chazelle, as seen in "Whiplash," a riveting music-themed drama.
    - *Aaron Guzikowski:* Explore the world of suspense and crime with Aaron Guzikowski, the director behind "Prisoners."
3. **Engage with Highly-Rated Movies:**

    - Our analysis revealed a strong correlation between ratings and the number of votes. Highly-rated movies tend to attract more viewer engagement and votes, reflecting their quality and popularity.In essence, the number of votes acts as a stabilizing factor, making movie ratings a more reliable indicator of a film's overall appeal and

merit.Together, they provide a more comprehensive understanding of a movie's impact and appeal.

4. **Language Diversity:**

   - While the majority of entries prefered are in English (en), the dataset features diverse languages. Explore the world of cinema beyond borders and experience stories from various cultures.

5. **Budget vs. Performance:**

   - The movie budget analysis showcases the potential of high production budgets to yield substantial financial returns. Titles like "Avatar," "Titanic," "Avengers: Infinity War," "Star Wars Ep. VII: The Force Awakens," and "Jurassic World" exemplify this trend.
   - The budget range that optimally balances production investment and financial gain, resulting in substantial profit, falls within the range of approximately $31.6 million to a return profit of$59.9 million.

7. **Best Months for Movie Release:**

   - If you're interested in the business side of filmmaking, our analysis identified the best months for movie releases in terms of achieving the highest average profit. Plan your movie-watching schedule accordingly to align with the most successful release months.The best release months were:
   - May
   - June
   - July

In conclusion, our movie recommendations cater to a broad spectrum of preferences,what directors are prefered, release months, average budget ensuring an enriching cinematic experience. Whether you're a sci-fi enthusiast, thriller aficionado, or a fan of drama and

Type *Markdown* and LaTeX: α^2