Final Project Submission

Please fill out:

- Student name: ROBERT KIDAKE KALAFA
- Student pace: PART TIME
- Scheduled project review date/time:
- Instructor name: WINNIE ANYOSO, SAMUEL G. MWANGI and SAMUEL KARU
- Blog post URL:

Overview

Microsoft has assigned me the responsibility of assessing the movie industry's potential and providing recommendations to assist in their decision-making process. This evaluation will entail analyzing the profitability of various genres, with ROI as a central metric. I will thoroughly examine the top-performing studios in the movie box office and explore the genres with the highest viewer ratings to generate insights for my analysis.

Business Problem

Microsoft aims to join the ranks of major companies venturing into original video content creation, prompting them to embark on establishing a new movie studio. However, their lack of experience in the movie-making domain poses a significant challenge. Microsoft endeavors to create content that can compete with established giants like Warner Bros and Walt Disney. To achieve this goal, Microsoft must gain a comprehensive understanding of audience preferences and industry trends. They need to formulate a strategy that effectively balances the costs of content creation with potential revenue streams, such as subscriptions or advertising.

In my assessment, I will focus on determining the most popular genres, identifying the top-

Data Understanding

My analysis entails leveraging data sourced from three distinct movie websites: TMDB, The Numbers, and Box Office Mojo. The dataset "bom.movie_gross.csv" comprises movie titles, studios, domestic and foreign financial earnings, and release years.

In [230]:
```
#importing the relevant libraries

import csv
import pandas as pd
```

In [231]:
```
bom_movie = pd.read_csv('bom.movie_gross.csv')
bom_movie
```

Out[231]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.00	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.00	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.00	664300000	2010
3	Inception	WB	292600000.00	535700000	2010
4	Shrek Forever After	P/DW	238700000.00	513900000	2010
...	...	...	...	...	...
3382	The Quake	Magn.	6200.00	NaN	2018
3383	Edward II (2018 re-release)	FM	4800.00	NaN	2018
3384	El Pacto	Sony	2500.00	NaN	2018
3385	The Swan	Synergetic	2400.00	NaN	2018
3386	An Actor Prepares	Grav.	1700.00	NaN	2018

3387 rows × 5 columns

The second dataset, tn.movie_budgets.csv, provides details on movie releases, encompassing titles, release dates, production budgets, and worldwide gross. The focal point of interest in this dataset is the return on investment, with the monetary data columns serving as the primary rationale for its selection.

In [232]:
```
movies_budgets = pd.read_csv('tn.movie_budgets.csv')
movies_budgets
```

Out[232]:

	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
...	...	...	...	...	...	...
5777	78	Dec 31, 2018	Red 11	$7,000	$0	$0
5778	79	Apr 2, 1999	Following	$6,000	$48,482	$240,495
5779	80	Jul 13, 2005	Return to the Land of Wonders	$5,000	$1,338	$1,338
5780	81	Sep 29, 2015	A Plague So Pleasant	$1,400	$0	$0
5781	82	Aug 5, 2005	My Date With Drew	$1,100	$181,041	$181,041

5782 rows × 6 columns

The third dataset, tmdb.movies.csv, comprises genre codes, original language, original movie titles, popularity metrics, release dates, and votes. It was employed to translate genre codes into genre names, facilitating the identification of trending genres. This dataset serves the purpose of mapping genre codes to genre names sourced from the same website, thereby revealing the most trending genres.

In [233]:
```
tmdb_movies = pd.read_csv('tmdb.movies.csv')
tmdb_movies
```

Out[233]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.53	2010-
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.73	2010-(
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.52	2010-(
3	3	[16, 35, 10751]	862	en	Toy Story	28.00	1995-
4	4	[28, 878, 12]	27205	en	Inception	27.92	2010-(
...	...	...	...	...	...	...	
26512	26512	[27, 18]	488143	en	Laboratory Conditions	0.60	2018-'
26513	26513	[18, 53]	485975	en	_EXHIBIT_84xxx_	0.60	2018-(
26514	26514	[14, 28, 12]	381231	en	The Last One	0.60	2018-'
26515	26515	[10751, 12, 28]	366854	en	Trailer Made	0.60	2018-(
26516	26516	[53, 27]	309885	en	The Church	0.60	2018-'

26517 rows × 10 columns

In [234]:
```
# importing of the necessary packages
import pandas as pd
# setting pandas display to avoid scientific notation in the dataframes
pd.options.display.float_format = '{:.2f}'.format
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
#Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

In [235]:
```
# The first dataset is the bom.movie_gross.csv

bom_movie = pd.read_csv('bom.movie_gross.csv')
bom_movie
```

Out[235]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.00	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.00	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.00	664300000	2010
3	Inception	WB	292600000.00	535700000	2010
4	Shrek Forever After	P/DW	238700000.00	513900000	2010
...	...	...	...	...	...
3382	The Quake	Magn.	6200.00	NaN	2018
3383	Edward II (2018 re-release)	FM	4800.00	NaN	2018
3384	El Pacto	Sony	2500.00	NaN	2018
3385	The Swan	Synergetic	2400.00	NaN	2018
3386	An Actor Prepares	Grav.	1700.00	NaN	2018

3387 rows × 5 columns

The DataFrame 'bom_movie' contains 3387 rows and 5 columns with the following information about movies:

1. title: The title of the movie
2. studio: The studio that produced the movie
3. domestic_gross : The domestic gross revenue of the movie in dollars
4. foreign_gross : The foreign gross revenue of the movie in dollars
5. year: The year in which the movie was released

The first few rows of the DataFrame are also shown in the output.

In [236]:
```
# getting concise summary infomation about the DataFrame
bom_movie.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
```

The title, studio, and foreign_gross columns have object data type, meaning they contain strings or a mixture of strings and other data types. The domestic_gross column has float64 data type, meaning it contains numerical data in decimal format. The year column has int64 data type, meaning it contains integer values. The studio column has 5 missing values, and the domestic_gross and foreign_gross columns have 28 and 1350 missing values, respectively.

In [237]:
```
# Descriptive statistics for domestic box office values
bom_movie['domestic_gross'].describe()
```

Out[237]:
```
count         3359.00
mean      28745845.07
std       66982498.24
min            100.00
25%         120000.00
50%        1400000.00
75%       27900000.00
max      936700000.00
Name: domestic_gross, dtype: float64
```

The output shows the summary statistics of the domestic_gross column of the DataFrame bom_movie which icludes the count, mean.standard deviation, the minimum value, the quatiles and the maximum values of the the domestic gross

1. The mean of the column is approximately 28.75 million dollars.
2. The standard deviation of the column is approximately 66.98 million dollars, indicating that the data is spread out widely.
3. The minimum value of the column is 100 dollars, meaning that there are movies in the dataset that made very little money.
4. The maximum value of the column is approximately 936.7 million dollars, indicating that there are movies in the dataset that made a lot of money domestically.

In [238]:
```
#Descriptive statistics for production budget values
bom_movie['foreign_gross'].describe()
```

Out[238]:
```
count        2037
unique       1204
top       1200000
freq           23
Name: foreign_gross, dtype: object
```

The output shows the summary statistics of the 'foreign_gross' column of the DataFrame 'bom_movie_df':

1. The count of non-null values is 2037, meaning there are 1350 missing values in the column. I will deal with this in the data cleaning section.
2. The unique count of values is 1204, meaning that there are 1204 unique values in the column, which implies that some movies had multiple foreign gross values.
3. The top value in the column is '1200000', which appears 23 times, implying that there are 23 movies that made 1.2 million dollars in foreign markets.
4. The frequency (freq) shows how many times the top value appears in the column.

The second dataset is the datafiles/tn.movie_budgets.csv

In [239]:
```
#Loading the movie budget dataset
movies_budgets = pd.read_csv('tn.movie_budgets.csv')
movies_budgets
```

Out[239]:

	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
...	...	...	...	...	...	...
5777	78	Dec 31, 2018	Red 11	$7,000	$0	$0
5778	79	Apr 2, 1999	Following	$6,000	$48,482	$240,495
5779	80	Jul 13, 2005	Return to the Land of Wonders	$5,000	$1,338	$1,338
5780	81	Sep 29, 2015	A Plague So Pleasant	$1,400	$0	$0
5781	82	Aug 5, 2005	My Date With Drew	$1,100	$181,041	$181,041

5782 rows × 6 columns

The movie_budgets_df dataframe contains 5782 rows and 6 columns. Each row represents a movie with its corresponding budget and gross revenue information. The columns are:

1. id: a unique identifier for each movie
2. release_date: the date when the movie was released in theaters
3. movie: the title of the movie
4. production_budget: the estimated production budget of the movie
5. domestic_gross: the gross revenue of the movie in the domestic market in North America
6. worldwide_gross: the gross revenue of the movie worldwide.

In [240]:
```
# Description for DataFrame: getting concise summary information about the d
movies_budgets.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
```

The initial step involves cleaning and reshaping the movie_budgets_df dataframe. By utilizing the str.replace() method, we can eliminate dollar signs and commas from the production_budget, domestic_gross, and worldwide_gross columns. Additionally, these columns, currently stored as objects, will be converted into numeric data types.

In [241]:
```
# Generating a brief statistics description for numerical columns in the Dat
movies_budgets.describe()
```

Out[241]:

	id
count	5782.00
mean	50.37
std	28.82
min	1.00
25%	25.00
50%	50.00
75%	75.00
max	100.00

Since .describe() automatically picks up integers it will only pick up id column as the production_budget, domestic_gross and worldwide_gross have commas and $ hence are considered objects.

The third dataset is tmdb.movies.csv

In [242]:
```
#Load the dataset
tmdb_movies = pd.read_csv('tmdb.movies.csv')
tmdb_movies
```

Out[242]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.53	2010-
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.73	2010-(
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.52	2010-(
3	3	[16, 35, 10751]	862	en	Toy Story	28.00	1995-
4	4	[28, 878, 12]	27205	en	Inception	27.92	2010-(
...	...	...	...	...	...	...	
26512	26512	[27, 18]	488143	en	Laboratory Conditions	0.60	2018-'
26513	26513	[18, 53]	485975	en	_EXHIBIT_84xxx_	0.60	2018-(
26514	26514	[14, 28, 12]	381231	en	The Last One	0.60	2018-'
26515	26515	[10751, 12, 28]	366854	en	Trailer Made	0.60	2018-(
26516	26516	[53, 27]	309885	en	The Church	0.60	2018-'

26517 rows × 10 columns

This will load the tmdb.movies.csv file into a pandas dataframe called tmdb_movies. The index_col=0 argument specifies that the first column of the csv file should be used as the index of the dataframe. The tmdb_movies dataframe has 26,518 rows and 9 columns. A brief description of the columns is as follows:

1. genre_ids: a list of integers representing the genre of the movie
2. id: unique identifier for the movie
3. original_language: the original language of the movie
4. original_title: the original title of the movie
5. popularity: a measure of the popularity of the movie
6. release_date: the date on which the movie was released
7. title: the title of the movie
8. vote_average: the average rating of the movie
9. vote_count: the number of votes cast for the movie.

In [243]:
```
#Looking at data info to get a concise summary of the DataFrame's structure
tmdb_movies.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
```

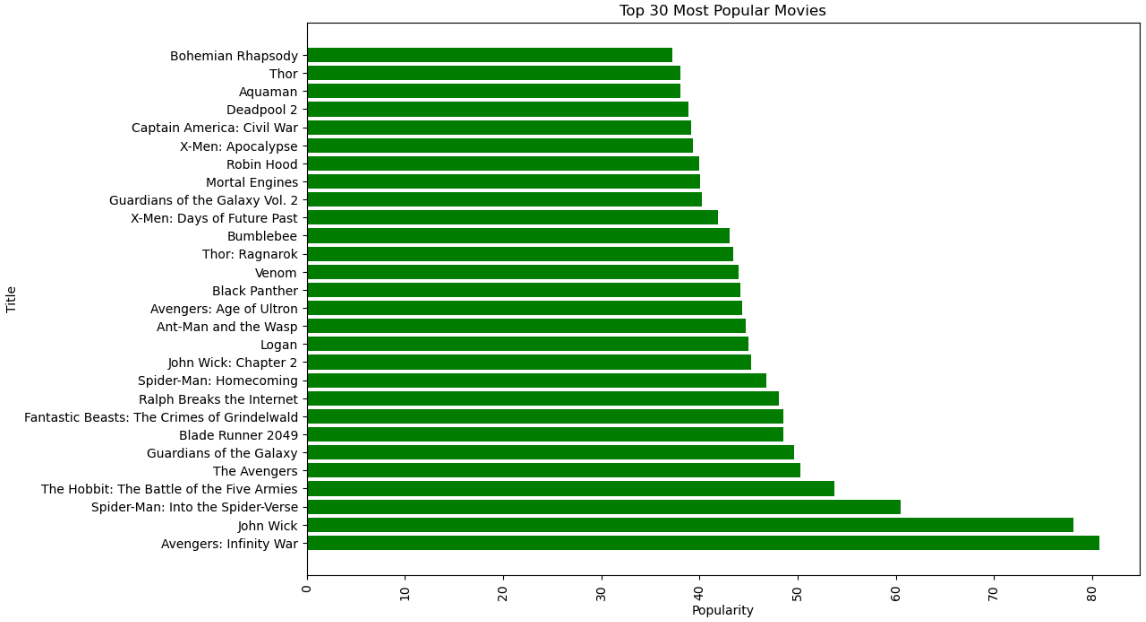
The dataset is complete as it has no missing values.

In [244]:
```
#Sorting by the "popularity" column in ascending order and displaying the fi
tmdb_movies.sort_values(by=["popularity"], ascending=True).head()
```

Out[244]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date
13258	13258	[99]	403294	en	9/11: Simulations	0.60	2014-07-04
11010	11010	[]	203325	en	Slaves Body	0.60	2013-06-25
11011	11011	[99]	186242	en	Re-Emerging: The Jews of Nigeria	0.60	2013-05-17
11012	11012	[99]	116868	en	Occupation: Fighter	0.60	2013-08-02
11013	11013	[99]	85337	en	Wonders Are Many: The Making of Doctor Atomic	0.60	2013-08-07

In [245]:
```
#Generates a horizontal bar plot that visually represents the popularity of
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 8))
top_30 = tmdb_movies.sort_values(by='popularity', ascending=False).head(30)
plt.barh(top_30['title'], top_30['popularity'], color='green')
plt.xlabel('Popularity')
plt.xticks(rotation=90)
plt.ylabel('Title')
plt.title('Top 30 Most Popular Movies')
plt.show()
```



It appears that certain movies in the dataset may have had limited recognition or popularity, indicated by popularity scores as low as 0.6 and vote counts as low as 1, possibly resulting in the low popularity values.

Data Cleaning

Having loaded the data and attempted to comprehend its contents, we can now proceed with the data cleaning process to prepare it for utilization.

Box office mojo

In [246]:
```
# convert "foreign_gross" column to a float
bom_movie['foreign_gross'] = pd.to_numeric(bom_movie['foreign_gross'], error
bom_movie
```

Out[246]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.00	652000000.00	2010
1	Alice in Wonderland (2010)	BV	334200000.00	691300000.00	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.00	664300000.00	2010
3	Inception	WB	292600000.00	535700000.00	2010
4	Shrek Forever After	P/DW	238700000.00	513900000.00	2010
...	...	...	...	...	...
3382	The Quake	Magn.	6200.00	NaN	2018
3383	Edward II (2018 re-release)	FM	4800.00	NaN	2018
3384	El Pacto	Sony	2500.00	NaN	2018
3385	The Swan	Synergetic	2400.00	NaN	2018
3386	An Actor Prepares	Grav.	1700.00	NaN	2018

3387 rows × 5 columns

The pd.to_numeric() method is used to convert the values in the column to numeric data type (float) and the errors='coerce' parameter specifies that if any value can't be converted, it will be set to NaN (Not a Number).

In [247]:
```
#regenerating descriptive statistics for production budget values
bom_movie['foreign_gross'].describe()
```

Out[247]:
```
count          2032.00
mean       75057041.63
std       137529351.20
min             600.00
25%         3775000.00
50%        18900000.00
75%        75050000.00
max       960500000.00
Name: foreign_gross, dtype: float64
```

The output shows the summary statistics of the 'foreign_gross' column of the DataFrame 'bom_movie_df':

1. mean: the mean (average) value of the column.
2. std: the standard deviation of the values in the column.
3. min: the smallest value in the column.
4. max: the largest value in the column.

In [248]:
```
#checking for missing values in the bom_movie_df
bom_movie.isna().sum()
```

Out[248]:
```
title                0
studio               5
domestic_gross      28
foreign_gross     1355
year                 0
dtype: int64
```

In [249]:
```
# Replacing missing values in the "studio" column with the string "None"
bom_movie["studio"].fillna("None", inplace = True)
# replacing missing values in the "domestic_gross" and "foreign_gross" colum
bom_movie["domestic_gross"].fillna(0, inplace = True)
bom_movie["foreign_gross"].fillna(0, inplace = True)
bom_movie
```

Out[249]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.00	652000000.00	2010
1	Alice in Wonderland (2010)	BV	334200000.00	691300000.00	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.00	664300000.00	2010
3	Inception	WB	292600000.00	535700000.00	2010
4	Shrek Forever After	P/DW	238700000.00	513900000.00	2010
...	...	...	...	...	...
3382	The Quake	Magn.	6200.00	0.00	2018
3383	Edward II (2018 re-release)	FM	4800.00	0.00	2018
3384	El Pacto	Sony	2500.00	0.00	2018
3385	The Swan	Synergetic	2400.00	0.00	2018
3386	An Actor Prepares	Grav.	1700.00	0.00	2018

3387 rows × 5 columns

By filling missing values in the "studio" column with the string "None" and replacing missing values in the "domestic_gross" and "foreign_gross" columns with the value 0, I would have handled the missing values in the bom_movie_df dataframe. This will help ensure that my analysis is not affected by missing data.

In [250]:
```
#Rechecking for missing values in the bom_movie
bom_movie.isna().sum()
```

Out[250]:
```
title             0
studio            0
domestic_gross    0
foreign_gross     0
year              0
dtype: int64
```

The numbers movie budgets

In [251]:
```
#Checking for missing values
missing_values_count = movies_budgets.isnull().sum()
print(missing_values_count)
```

```
id                   0
release_date         0
movie                0
production_budget    0
domestic_gross       0
worldwide_gross      0
dtype: int64
```

To clean up this dataframe we replace commas and dollar signs in the worldwide_gross, domestic_gross, and production_budge columns with nothing ('') and then convert them to floats

I will continue to perform data cleaning and conversion tasks on specific columns of the DataFrame. After executing the below code lines, the 'domestic_gross', 'production_budget', and 'worldwide_gross' columns of the DataFrame movies_budgets will contain numeric values, with any currency symbols and commas removed. This makes the data suitable for numerical analysis and calculations, such as computing statistics or creating visualizations.

In [252]:
```
movies_budgets['domestic_gross'] = pd.to_numeric(movies_budgets['domestic_gr
movies_budgets['production_budget'] =pd.to_numeric(movies_budgets['productic
movies_budgets['worldwide_gross'] = pd.to_numeric(movies_budgets['worldwide_

movies_budgets
```

Out[252]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875
2	3	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350
3	4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747
...	...	...	...	...	...	...
5777	78	Dec 31, 2018	Red 11	7000	0	0
5778	79	Apr 2, 1999	Following	6000	48482	240495
5779	80	Jul 13, 2005	Return to the Land of Wonders	5000	1338	1338
5780	81	Sep 29, 2015	A Plague So Pleasant	1400	0	0
5781	82	Aug 5, 2005	My Date With Drew	1100	181041	181041

5782 rows × 6 columns

Applies a lambda function to each column selected above the x.str.replace(',', ''): Replaces commas in the string values with empty strings. This replaces the original string values with float values in the specified columns of the movie_budgets dataframe.

In [253]:
```
#merge the bom_movie_df and movie_budget df on movie titles
merged_df = pd.merge(bom_movie, movies_budgets, how='inner', left_on='title'
#drop title and domestic_gross since they appear in both dataframes
merged_df = merged_df.drop(['domestic_gross_y', 'title'], axis=1)
#preview the merged dataframe
merged_df
```

Out[253]:

	studio	domestic_gross_x	foreign_gross	year	id	release_date	movie	production
0	BV	415000000.00	652000000.00	2010	47	Jun 18, 2010	Toy Story 3	20
1	WB	292600000.00	535700000.00	2010	38	Jul 16, 2010	Inception	16
2	P/DW	238700000.00	513900000.00	2010	27	May 21, 2010	Shrek Forever After	16
3	Sum.	300500000.00	398000000.00	2010	53	Jun 30, 2010	The Twilight Saga: Eclipse	6
4	Par.	312400000.00	311500000.00	2010	15	May 7, 2010	Iron Man 2	17
...	...	...	...	...	...	...	...	
1242	VE	4300000.00	0.00	2018	64	Jun 15, 2018	Gotti	1
1243	RAtt.	3700000.00	0.00	2018	95	Dec 7, 2018	Ben is Back	1
1244	VE	491000.00	1700000.00	2018	100	Feb 2, 2018	Bilal: A New Breed of Hero	3
1245	RLJ	1200000.00	0.00	2018	71	Sep 14, 2018	Mandy	
1246	A24	1200000.00	0.00	2018	13	Apr 6, 2018	Lean on Pete	

1247 rows × 9 columns

I merged the two dataframes on the "title" column. I also dropped the "domestic_gross_y" and "title" columns, which is necessary since they appear in both dataframes.

In [254]:
```
# Filter the DataFrame to include only years above 2013
movie_budgets_filtered_df = merged_df[merged_df['year'] >= 2013]
movie_budgets_filtered_df
```

Out[254]:

	studio	domestic_gross_x	foreign_gross	year	id	release_date	movie	productic
496	BV	400700000.00	875700000.00	2013	56	Nov 22, 2013	Frozen	'
497	BV	409000000.00	805800000.00	2013	48	May 3, 2013	Iron Man 3	2
498	Uni.	368100000.00	602700000.00	2013	22	Jul 3, 2013	Despicable Me 2	
499	WB (NL)	258399999.00	700000000.00	2013	21	Dec 13, 2013	The Hobbit: The Desolation of Smaug	2
500	LGF	424700000.00	440300000.00	2013	38	Nov 22, 2013	The Hunger Games: Catching Fire	'
...	...	...	...	...	...	...	...	
1242	VE	4300000.00	0.00	2018	64	Jun 15, 2018	Gotti	
1243	RAtt.	3700000.00	0.00	2018	95	Dec 7, 2018	Ben is Back	
1244	VE	491000.00	1700000.00	2018	100	Feb 2, 2018	Bilal: A New Breed of Hero	
1245	RLJ	1200000.00	0.00	2018	71	Sep 14, 2018	Mandy	
1246	A24	1200000.00	0.00	2018	13	Apr 6, 2018	Lean on Pete	

751 rows × 9 columns

This code filters the merged_df to only include movies from 2013 onwards and saves it as movie_budgets_filtered_df. Now we have reduced our dataset to 751 rows.

TheMovieDB

Since the dataset is too large, I will sort it in a way that allows me to work with fewer movies. I decided to sort them with their vote_counts.

In [255]:
```
# creating a list of all the vote_counts and sorting them
vote_counts = tmdb_movies['vote_count'].tolist()
vote_counts_sorted = sorted(vote_counts)
```

In [256]:
```
# Define a function to filter a list to values between two numbers
def filter_list(lst, min_val, max_val):
    filtered_list = [x for x in lst if (x > min_val) and (x < max_val)]
    return filtered_list
```

In [257]:
```
# Count the number of movies that have vote counts between 1000 and 23000
num_movies = len(filter_list(vote_counts_sorted, 999, 23000))
num_movies
```

Out[257]: 1108

In [258]:
```
# Filter the DataFrame to only include movies with vote counts of 1000 or mc
filtered_tmdb = tmdb_movies[tmdb_movies['vote_count'] >= 1000]
filtered_tmdb
```

Out[258]:

	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	33.53	2010-11-19
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.73	2010-03-26
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.52	2010-05-07
3	3	[16, 35, 10751]	862	en	Toy Story	28.00	1995-11-22
4	4	[28, 878, 12]	27205	en	Inception	27.92	2010-07-16
...	...	...	...	...	...	...	...
24112	24112	[53, 18, 80, 9648]	446791	en	All the Money in the World	10.94	2017-12-25
24128	24128	[35, 18, 878]	301337	en	Downsizing	10.68	2017-12-22
24169	24169	[16, 18, 9648]	339877	en	Loving Vincent	10.03	2017-09-22
24231	24231	[18]	538362	it	Sulla mia pelle	9.16	2018-09-12
24268	24268	[14, 18]	490	sv	Det sjunde inseglet	8.69	1958-10-13

1108 rows × 10 columns

The code filters the tmdb_movies dataframe to only include movies that have a vote count of 1000 or more, indicating a relatively popular movie. The resulting dataframe is stored in the variable filtered_tmdb.

In [259]:
```
# Find duplicates based on all columns
duplicates = filtered_tmdb.duplicated()

# Filter the DataFrame to show only the duplicate rows
duplicate_rows = filtered_tmdb[duplicates]

# Print the duplicate rows
print(len(duplicate_rows))
```

```
0
```

In [260]:
```
#call it back to show cleaned data
tmdb_movies
```

Out[260]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.53	2010-
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.73	2010-(
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.52	2010-(
3	3	[16, 35, 10751]	862	en	Toy Story	28.00	1995-
4	4	[28, 878, 12]	27205	en	Inception	27.92	2010-(
...	...	...	...	...	...	...	
26512	26512	[27, 18]	488143	en	Laboratory Conditions	0.60	2018-'
26513	26513	[18, 53]	485975	en	_EXHIBIT_84xxx_	0.60	2018-(
26514	26514	[14, 28, 12]	381231	en	The Last One	0.60	2018-'
26515	26515	[10751, 12, 28]	366854	en	Trailer Made	0.60	2018-(
26516	26516	[53, 27]	309885	en	The Church	0.60	2018-'

26517 rows × 10 columns

In [261]:
```
#checking for missing values
missing_values_count = tmdb_movies.isnull().sum()
print(missing_values_count)
```

```
Unnamed: 0           0
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
```

There are no missing values in this dataset. I'll now proceed to obtain data set genres that correspond to their respective genre ids.

In [262]:
```
# genre_ids are list of numbers, actually in a string.
tmdb_movies.iloc[0]['genre_ids']
```

Out[262]:
```
'[12, 14, 10751]'
```

This code will return a list of genre IDs associated with the first movie in the DataFrame.

In [263]:
```
#Create dictionary of genre ID and its associated genre name.
#This information is sourced from tmdb website

genre_dict = {
    28: 'Action',
    12: 'Adventure',
    16: 'Animation',
    35: 'Comedy',
    80: 'Crime',
    99: 'Documentary',
    18: 'Drama',
    10751: 'Family',
    14: 'Fantasy',
    36: 'History',
    27: 'Horror',
    10402: 'Music',
    9648: 'Mystery',
    10749: 'Romance',
    878: 'Science Fiction',
    10770: 'TV Movie',
    53: 'Thriller',
    10752: 'War',
    37: 'Western'
}
```

In [264]:
```
# Creating a dataframe with id and genre columns
genre_df = pd.DataFrame.from_dict(genre_dict, orient='index', columns=['genr
genre_df.index.name = 'id'
genre_df.reset_index(inplace=True)
genre_df
```

Out[264]:

	id	genre
0	28	Action
1	12	Adventure
2	16	Animation
3	35	Comedy
4	80	Crime
5	99	Documentary
6	18	Drama
7	10751	Family
8	14	Fantasy
9	36	History
10	27	Horror
11	10402	Music
12	9648	Mystery
13	10749	Romance
14	878	Science Fiction
15	10770	TV Movie
16	53	Thriller
17	10752	War
18	37	Western

The genre ids are a list of numbers in a string but I want them to be integers.

In [265]:
```
#defining a function for removing the brackets from the string in 'genre_ids
def split_ids(string):
    string = string.replace('[','').replace(']','')
    numbers = string.split(',')
    new_list = []
    for i in numbers:
        if i != '':
            new_list.append(int(i))
    return new_list
```

In [266]:
```
#applying the "split_ids" function to each value in the "genre_ids"
def split_ids(input):
    if isinstance(input, str):
        input = input.replace('[','').replace(']','')
        numbers = input.split(',')
        new_list = []
        for i in numbers:
            if i.isdigit():
                new_list.append(int(i))
        return new_list
    elif isinstance(input, list):
        return input
    else:
        return []
```

In [267]:
```
def split_ids(string):
    if string == '':
        return []
    else:
        string = string.replace('[','').replace(']','')
        numbers = string.split(',')
        new_list = []
        for i in numbers:
            if i.isdigit():
                new_list.append(int(i))
        return new_list

tmdb_movies['genre_names'] = tmdb_movies['genre_ids'].apply(lambda x: split_
tmdb_movies
```

Out[267]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.53	2010-
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.73	2010-(
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.52	2010-(
3	3	[16, 35, 10751]	862	en	Toy Story	28.00	1995-
4	4	[28, 878, 12]	27205	en	Inception	27.92	2010-(
...	...	...	...	...	...	...	
26512	26512	[27, 18]	488143	en	Laboratory Conditions	0.60	2018-'
26513	26513	[18, 53]	485975	en	_EXHIBIT_84xxx_	0.60	2018-(
26514	26514	[14, 28, 12]	381231	en	The Last One	0.60	2018-'
26515	26515	[10751, 12, 28]	366854	en	Trailer Made	0.60	2018-(
26516	26516	[53, 27]	309885	en	The Church	0.60	2018-'

26517 rows × 11 columns

Now we have a new column called 'genre_names' which is mapped onto corresponding 'genre_ids'.

Data Analysis & Visualizations

1. Determining the most prevalent genre.

Merge tmdb with movie budgets

In [268]:
```
#merge the movie_budgets_filtered_df and tmdb_df_filtered on movie titles
final_merged_df = pd.merge(tmdb_movies, movie_budgets_filtered_df , how='inr
final_merged_df
```

Out[268]:

	Unnamed: 0	genre_ids	id_x	original_language	original_title	popularity	release_date_x
0	148	[53]	44363	en	Frozen	9.68	2010-02-05
1	7886	[16, 12, 10751]	109445	en	Frozen	26.18	2013-11-27
2	321	[18, 9648, 10749]	38407	en	Shanghai	6.82	2010-10-02
3	801	[878]	136921	en	Pixels	2.17	2010-04-01
4	14187	[28, 35, 878]	257344	en	Pixels	23.03	2015-07-24
...	...	...	...	...	...	...	...
894	24089	[18, 36, 53]	453201	en	The 15:17 to Paris	11.58	2018-02-09
895	24120	[35]	474335	en	Uncle Drew	10.84	2018-06-29
896	24168	[80, 18, 36, 53]	339103	en	Gotti	10.03	2018-06-15
897	24212	[53, 28, 80]	442064	en	Proud Mary	9.37	2018-01-12
898	25148	[28, 12, 16]	332718	en	Bilal: A New Breed of Hero	2.71	2018-02-02

899 rows × 20 columns

We have finally merged our datasets to obtain a dataframe that we will use for the rest of our analysis. The resulting merged dataframe will contain columns from both dataframes where the 'original_title' column has matching values.

We create a new dataframe with the values that can help us analyse the most popular dataframe

In [269]:
```python
# Create an empty list to store DataFrame objects
genre_popl_list = []

# Iterate through each row of the TMBD+MovieBudgets dataset
for i in range(len(final_merged_df)):
    # Extract the list of genre IDs for each movie and convert it to a list
    ids = final_merged_df.iloc[i]['genre_ids']
    genre_ids_list = ids.strip('][').split(', ') if isinstance(ids, str) els

    # Iterate through each genre ID for the current movie
    for genre_id in genre_ids_list:
        # Check if genre_id is not an empty string
        if genre_id:
            # Extract the relevant information for the current movie and ger
            popularity = final_merged_df.iloc[i]['popularity']
            title = final_merged_df.iloc[i]['original_title']
            avg = final_merged_df.iloc[i]['vote_average']
            genre = int(genre_id)
            budget = final_merged_df.iloc[i]['production_budget']
            revenue = final_merged_df.iloc[i]['worldwide_gross']

            # Calculate the ROI for the current movie and genre
            if budget != 0:
                ROI = ((revenue - budget) / budget) * 100
            else:
                ROI = 0

            # Create a DataFrame with the information for the current movie
            df = pd.DataFrame({
                'popularity': [popularity],
                'title': [title],
                'vote_average': [avg],
                'genre': [genre],
                'ROI': [ROI]
            })

            # Append the DataFrame to the list
            genre_popl_list.append(df)

# Concatenate all DataFrames in the list to create the final DataFrame
genre_popl = pd.concat(genre_popl_list, ignore_index=True)
```

The purpose of the code is to iterate through each row in the TMBD+MovieBudgets dataset and extract the list of genre IDs for each movie. Subsequently, it will iterate through each genre ID for the current movie, gather relevant details for both the movie and genre, calculate their respective return on investment (ROI), and append a row to the genre_popl DataFrame. This new row will contain information for the current movie-genre combination, including popularity, title, vote average, genre ID, and ROI. Thus, the resulting DataFrame will consist of one row for each movie-genre pair, encompassing the specified details.

In [270]:
```
genre_popl
```

Out[270]:

	popularity	title	vote_average	genre	ROI
0	9.68	Frozen	5.80	53	748.31
1	26.18	Frozen	7.30	16	748.31
2	26.18	Frozen	7.30	12	748.31
3	26.18	Frozen	7.30	10751	748.31
4	6.82	Shanghai	6.10	18	-68.99
...	...	...	...	...	...
2241	9.37	Proud Mary	5.50	28	-27.63
2242	9.37	Proud Mary	5.50	80	-27.63
2243	2.71	Bilal: A New Breed of Hero	6.80	28	-97.84
2244	2.71	Bilal: A New Breed of Hero	6.80	12	-97.84
2245	2.71	Bilal: A New Breed of Hero	6.80	16	-97.84

2246 rows × 5 columns

Next we merge the genre_popl with thw genre_df (the data frame we created containing id and genre)

In [271]:
```
# merge the genre_popl with the genre_df
genre_popl_merged = genre_popl.merge(genre_df, left_on="genre", right_on="id
genre_popl_merged
```

Out[271]:

	popularity	title	vote_average	genre_x	ROI	id	genre_y
0	9.68	Frozen	5.80	53	748.31	53	Thriller
1	24.74	Get Out	7.50	53	5007.36	53	Thriller
2	10.16	The Lazarus Effect	5.10	53	667.19	53	Thriller
3	7.18	Trash	7.10	53	-45.39	53	Thriller
4	10.20	Legend	6.80	53	-5.98	53	Thriller
...	...	...	...	...	...	...	...
2241	0.60	The Judge	7.50	99	52.24	99	Documentary
2242	1.96	Moana	6.50	99	325.01	99	Documentary
2243	0.60	They Will Have to Kill Us First	5.00	99	-98.68	99	Documentary
2244	4.34	City of Ghosts	7.10	99	-98.14	99	Documentary
2245	2.26	Non-Stop	5.60	10770	344.77	10770	TV Movie

2246 rows × 7 columns

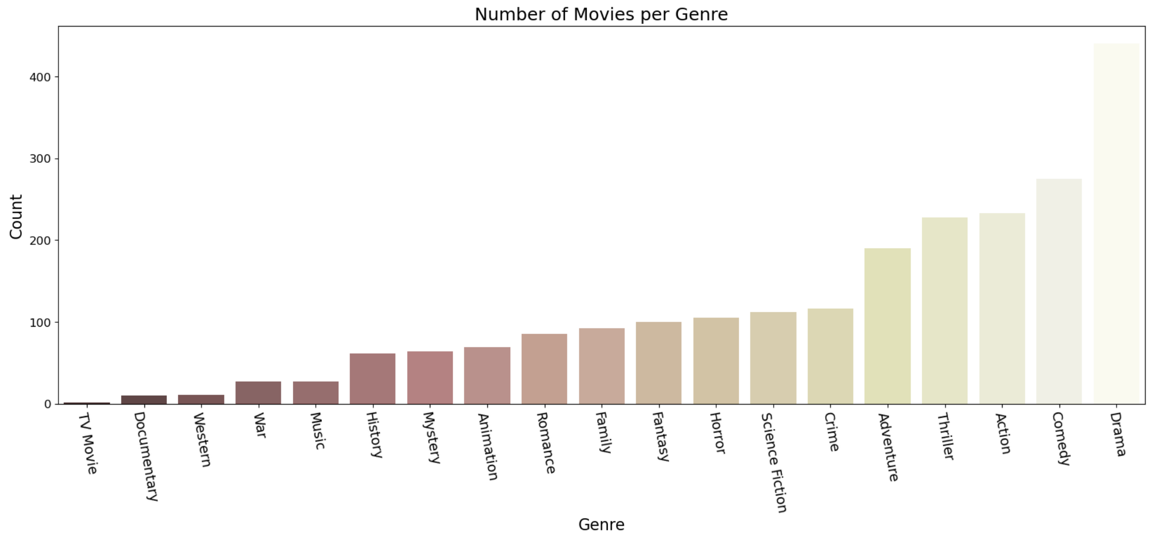
The resulting dataframe genre_popl_merged should have columns for popularity, title, vote_average, genre, ROI, and id, where id corresponds to the genre ID used in the TMDB API and genre corresponds to the actual name of the genre.

In [272]:
```
# getting value counts for genre column
genre_popl_merged['genre_y'].value_counts()
```

Out[272]:
```
genre_y
Drama              440
Comedy             275
Action             233
Thriller           228
Adventure          190
Crime              116
Science Fiction    112
Horror             105
Fantasy            100
Family              92
Romance             85
Animation           69
Mystery             64
History             61
Music               27
War                 27
Western             11
Documentary         10
TV Movie             1
Name: count, dtype: int64
```

This count of values will assist us in identifying genres with the highest occurrence. Our subsequent step entails generating a graph to discern which movie genres boast the highest film count.

In [273]:
```
#Plotting the number of movies per genre in dataset
plt.figure(figsize=(20, 7))

#Sort the genres by ascending count
genre_counts_sorted = genre_popl_merged['genre_y'].value_counts().sort_value
sns.countplot(x='genre_y', data=genre_popl_merged, palette='pink',
order=genre_counts_sorted.index)

#Setting title, labels, and tick sizes
plt.title('Number of Movies per Genre', fontsize=18)
plt.ylabel('Count', fontsize=16)
plt.xlabel('Genre', fontsize=16)
plt.xticks(fontsize=14, rotation=-80)
plt.yticks(fontsize=12)

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



From the graph above we learn that genres with the highest number of movies were:

- Drama
- Action
- Comedy
- Adventure
- Thriller.

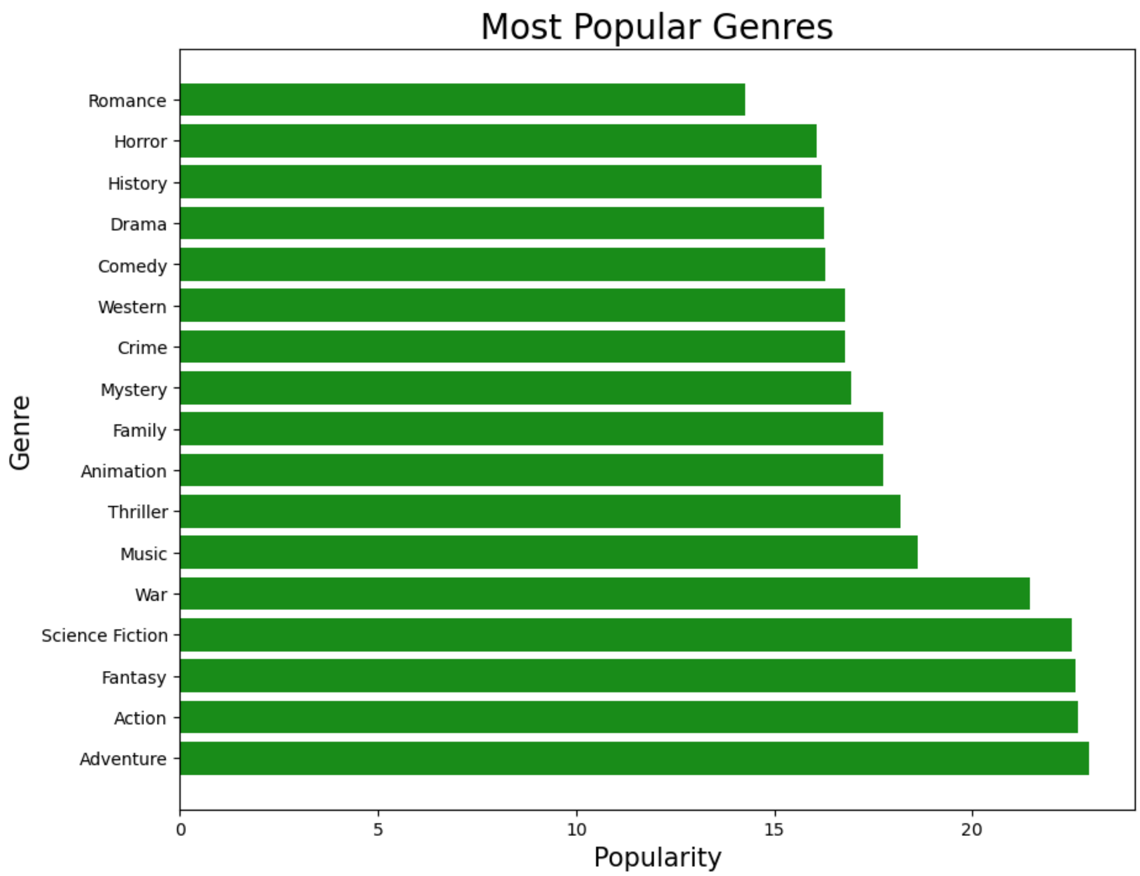
In [274]:
```
import pandas as pd

# Group by genre and calculate the mean of vote_average
top_votes = genre_df.groupby("genre").mean().reset_index()

# Display the resulting DataFrame
print(top_votes)
```

```
              genre       id
0            Action    28.00
1         Adventure    12.00
2         Animation    16.00
3            Comedy    35.00
4             Crime    80.00
5       Documentary    99.00
6             Drama    18.00
7            Family 10751.00
8           Fantasy    14.00
9           History    36.00
10           Horror    27.00
11            Music 10402.00
12          Mystery  9648.00
13          Romance 10749.00
14  Science Fiction   878.00
15         TV Movie 10770.00
16         Thriller    53.00
17              War 10752.00
18          Western    37.00
```

In [275]:
```
# Print column names of both DataFrames
print("top_votes columns:", top_votes.columns)
print("genre_df columns:", genre_df.columns)

# Merge top_votes with genre_df on the "genre" column
highly_voted = top_votes.merge(genre_df, on="genre")

# Display the resulting DataFrame
print(highly_voted)
```

```
top_votes columns: Index(['genre', 'id'], dtype='object')
genre_df columns: Index(['id', 'genre'], dtype='object')
              genre     id_x   id_y
0            Action     28.00     28
1         Adventure     12.00     12
2         Animation     16.00     16
3            Comedy     35.00     35
4             Crime     80.00     80
5       Documentary     99.00     99
6             Drama     18.00     18
7            Family  10751.00  10751
8           Fantasy     14.00     14
9           History     36.00     36
10           Horror     27.00     27
11            Music  10402.00  10402
12          Mystery   9648.00   9648
13          Romance  10749.00  10749
14  Science Fiction    878.00    878
15         TV Movie  10770.00  10770
16         Thriller     53.00     53
17              War  10752.00  10752
18          Western     37.00     37
```

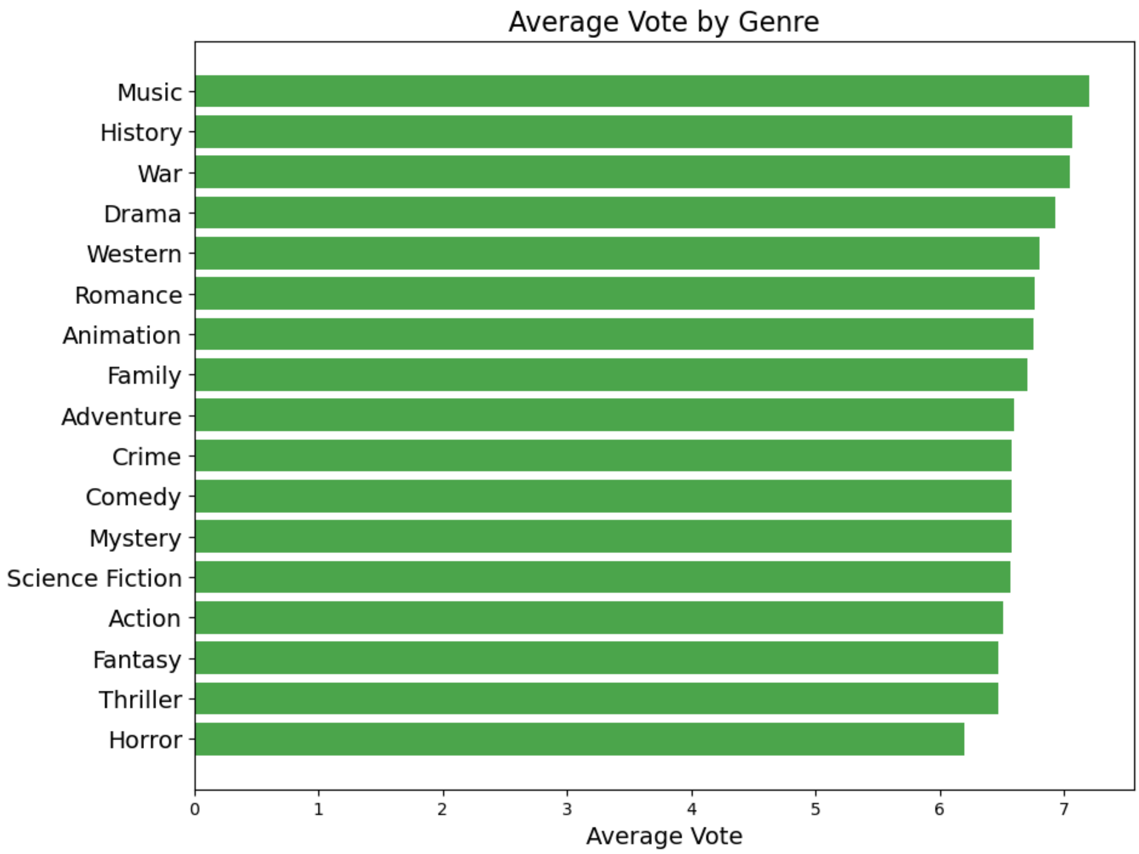
In [276]:
```
# Create the DataFrame
df = pd.DataFrame({'popularity': [22.96, 22.65, 22.61, 22.52, 21.46, 18.62,
                   'genre_y': ['Adventure', 'Action', 'Fantasy', 'Science Fi

# Sort the DataFrame by popularity
most_popular = df.sort_values(by='popularity', ascending=False)

# Create a horizontal bar chart
fig, ax = plt.subplots(figsize=(10,8))
ax.barh(y=most_popular['genre_y'], width=most_popular['popularity'], color='

# Set the x-tick labels
ax.set_xlabel('Popularity', fontsize=15)

# Set the y-tick labels
ax.set_ylabel('Genre', fontsize=15)

# Set the title
ax.set_title('Most Popular Genres', fontsize=20)

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



I examined each movie and categorized them according to their respective genres. Based on my analysis, I identified the seven most commonly occurring genres, which are

1. Adventure
2. Action
3. Fantasy
4. Science Fiction

In [277]:
```
# create the dataframe
data = {'vote_average': [7.20, 7.06, 7.04, 6.93, 6.80, 6.76, 6.75, 6.70, 6.6
        'genre_y': ['Music', 'History', 'War', 'Drama', 'Western', 'Romance',
df = pd.DataFrame(data)

# sort the dataframe by vote_average in descending order
most_popular = df.sort_values(by='vote_average', ascending=False)

# create a horizontal bar chart
fig, ax = plt.subplots(figsize=(10,8))
ax.barh(y=range(len(df)), width=most_popular['vote_average'], color='green',

# set the y-tick labels as the genres
ax.set_yticks(range(len(df)))
ax.set_yticklabels(most_popular['genre_y'], fontsize=14)

# set the x-axis label
ax.set_xlabel('Average Vote', fontsize=14)

# set the title
ax.set_title('Average Vote by Genre', fontsize=16)

# invert the y-axis to display the genres in descending order
ax.invert_yaxis()

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



After analyzing the data, I determined that the top five genres with the highest average rating (in terms of stars) are

1. Music

2. History
3. War
4. Animation
5. Drama

Conclusion

Across the movie industry, different genres exhibit varying performance levels in terms of average ratings. Leveraging the provided data, we have pinpointed the top-performing genres based on ratings. Notably, Documentary and Drama emerge as the highest-performing genres, boasting an average rating exceeding 6. It's advisable to consider genre selection carefully when deciding which movies to produce in the studio, ensuring alignment with target ratings and maximizing business potential.

2. Examining the correlation between production budget and return on investment.

- Trying to find out if the more the company spends the more they get on return on investment

In [278]:
```
#calculating and creating a new column in the dataframe named 'ROI'
movie_budgets_filtered_df['ROI'] = ((movie_budgets_filtered_df['worldwide_gr
movie_budgets_filtered_df.head()
```

Out[278]:

	studio	domestic_gross_x	foreign_gross	year	id	release_date	movie	production_
496	BV	400700000.00	875700000.00	2013	56	Nov 22, 2013	Frozen	15(
497	BV	409000000.00	805800000.00	2013	48	May 3, 2013	Iron Man 3	20(
498	Uni.	368100000.00	602700000.00	2013	22	Jul 3, 2013	Despicable Me 2	7(
499	WB (NL)	258399999.00	700000000.00	2013	21	Dec 13, 2013	The Hobbit: The Desolation of Smaug	25(
500	LGF	424700000.00	440300000.00	2013	38	Nov 22, 2013	The Hunger Games: Catching Fire	13(

Now the movie_budgets_filtered_df dataframe has a new column called "ROI" (Return on Investment) that represents the return on investment percentage for each movie based on its worldwide gross and production budget.

In [279]:
```
#merge the movie_budgets_filtered_df and tmdb_df_filtered on movie titles
final_merged_df = pd.merge(tmdb_movies, movie_budgets_filtered_df , how='inn
final_merged_df
```

Out[279]:

	Unnamed: 0	genre_ids	id_x	original_language	original_title	popularity	release_date_x
0	148	[53]	44363	en	Frozen	9.68	2010-02-05
1	7886	[16, 12, 10751]	109445	en	Frozen	26.18	2013-11-27
2	321	[18, 9648, 10749]	38407	en	Shanghai	6.82	2010-10-02
3	801	[878]	136921	en	Pixels	2.17	2010-04-01
4	14187	[28, 35, 878]	257344	en	Pixels	23.03	2015-07-24
...	...	...	...	...	...	...	...
894	24089	[18, 36, 53]	453201	en	The 15:17 to Paris	11.58	2018-02-09
895	24120	[35]	474335	en	Uncle Drew	10.84	2018-06-29
896	24168	[80, 18, 36, 53]	339103	en	Gotti	10.03	2018-06-15
897	24212	[53, 28, 80]	442064	en	Proud Mary	9.37	2018-01-12
898	25148	[28, 12, 16]	332718	en	Bilal: A New Breed of Hero	2.71	2018-02-02

899 rows × 21 columns

In [280]:
```
#drop irrelevant columns since they appear in both dataframes
tmbd_mb_df = final_merged_df.drop(['foreign_gross', 'title', 'original_langu
tmbd_mb_df
```

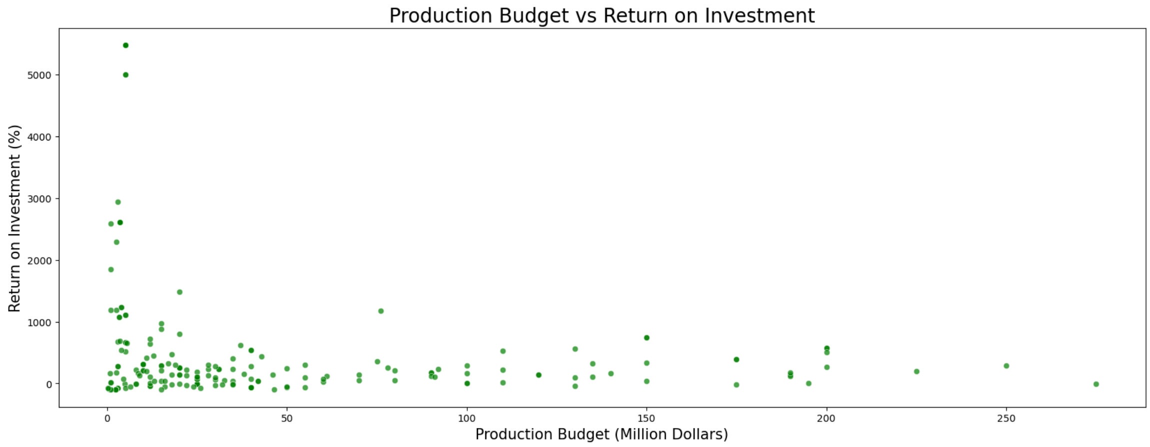
Out[280]:

	Unnamed: 0	genre_ids	original_title	popularity	release_date_x	vote_average	genre_nam
0	148	[53]	Frozen	9.68	2010-02-05	5.80	[5
1	7886	[16, 12, 10751]	Frozen	26.18	2013-11-27	7.30	[1
2	321	[18, 9648, 10749]	Shanghai	6.82	2010-10-02	6.10	[1
3	801	[878]	Pixels	2.17	2010-04-01	7.10	[87
4	14187	[28, 35, 878]	Pixels	23.03	2015-07-24	5.60	[2
...	...	...	...	...	...	...	
894	24089	[18, 36, 53]	The 15:17 to Paris	11.58	2018-02-09	5.30	[1
895	24120	[35]	Uncle Drew	10.84	2018-06-29	6.50	[3
896	24168	[80, 18, 36, 53]	Gotti	10.03	2018-06-15	5.20	[8
897	24212	[53, 28, 80]	Proud Mary	9.37	2018-01-12	5.50	[5
898	25148	[28, 12, 16]	Bilal: A New Breed of Hero	2.71	2018-02-02	6.80	[2

899 rows × 12 columns

I have merged the two dataframes 'tmdb_movies' and 'movie_budgets_filtered_df'. Then created a new dataframe 'tmbd_mb_df' by dropping some columns from the merged dataframe.

In [281]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create figure and axis object
fig, ax = plt.subplots(figsize=(20, 7))

# Convert production_budget to million dollars
tmbd_mb_df['production_budget_million'] = tmbd_mb_df['production_budget'] /

# Plot the scatterplot with the desired color
sns.scatterplot(x='production_budget_million', y='ROI', data=tmbd_mb_df.head

# Set labels and title
ax.set_xlabel('Production Budget (Million Dollars)', fontsize=15)
ax.set_ylabel('Return on Investment (%)', fontsize=15)
ax.set_title('Production Budget vs Return on Investment', fontsize=20)

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



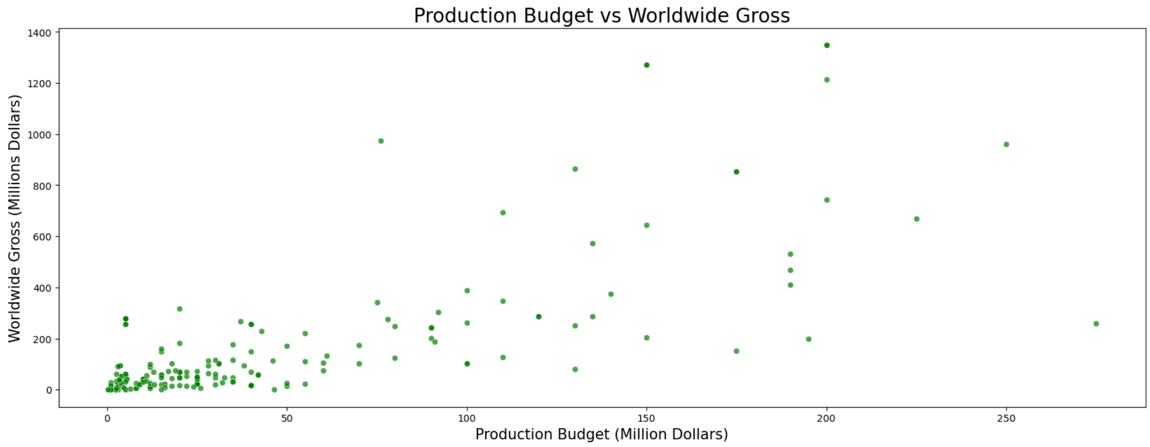
Upon analyzing the scatter plot, it becomes apparent that there is a reverse relationship between the production budget and ROI, albeit not linear. Particularly within the budget range of 0 to 100 million dollars, there's a negative correlation observed between ROI and production budget. However, within the budget range of 100 to 300 million dollars, the correlation between these variables appears to be less discernible.

In [282]:
```
# We can look at the Pearson correlation coefficient between the 'worldwide_
np.corrcoef(tmbd_mb_df['worldwide_gross'], tmbd_mb_df['ROI'])[0,1]
```

Out[282]: 0.06504534930840634

The Pearson correlation coefficient between the 'worldwide_gross' and 'ROI' columns indicates a weak positive correlation between these variables. This implies that there's a slight tendency for movies with higher worldwide grosses to exhibit higher return on investment (RoI), although the correlation isn't particularly strong. Other factors, like production budget and marketing efforts, might exert a more substantial influence on RoI compared to worldwide gross alone. Moreover, outliers within the dataset, such as exceptionally low-budget movies yielding unexpectedly high returns, could potentially impact this correlation.

In [283]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create figure and axis object
fig, ax = plt.subplots(figsize=(20, 7))

# Convert production_budget to million dollars
tmbd_mb_df['production_budget_million'] = tmbd_mb_df['production_budget'] /
tmbd_mb_df['worldwide_gross_million'] = tmbd_mb_df['worldwide_gross'] / 1000

# Plot the scatter plot with the desired color
sns.scatterplot(x='production_budget_million', y='worldwide_gross_million',

# Set labels and title
ax.set_xlabel('Production Budget (Million Dollars)', fontsize=15)
ax.set_ylabel('Worldwide Gross (Millions Dollars)', fontsize=15)
ax.set_title('Production Budget vs Worldwide Gross', fontsize=20)

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



Based on the scatter plot analysis, the worldwide gross tends to increase as production budget increases.

The Pearson correlation coefficient between the 'production_budget_million' and 'worldwide_gross_million' columns indicates a robust positive correlation between these variables. This implies that as the production budget of a movie rises, its worldwide gross tends to increase as well. The substantial correlation strength suggests a consistent association across the dataset, although it does not establish causation. Other elements, like the movie's quality or marketing efforts, may also influence the relationship between production budget and worldwide gross.

3. What are the best performing studios at the movie box office?

In [284]:
```
#create a new DataFrame called studio_df with the columns studio, foreign_gr
studio_df = final_merged_df[['studio', 'foreign_gross', 'domestic_gross_x',
studio_df
```

Out[284]:

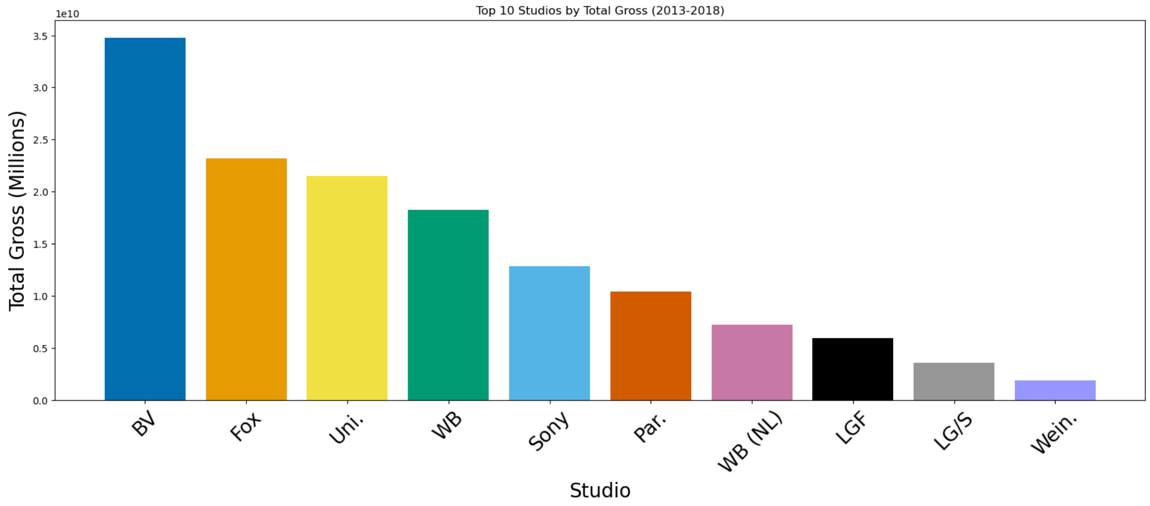
	studio	foreign_gross	domestic_gross_x	production_budget
0	BV	875700000.00	400700000.00	150000000
1	BV	875700000.00	400700000.00	150000000
2	Wein.	9200000.00	46400.00	50000000
3	Sony	166100000.00	78700000.00	90000000
4	Sony	166100000.00	78700000.00	90000000
...	...	...	...	...
894	WB	20800000.00	36300000.00	30000000
895	LG/S	4200000.00	42500000.00	18000000
896	VE	0.00	4300000.00	10000000
897	SGem	876000.00	20900000.00	30000000
898	VE	1700000.00	491000.00	30000000

899 rows × 4 columns

The breakdown of what each column in studio_df represents:

- studio: The name of the movie studio that produced the movie.
- foreign_gross: The gross revenue earned from the movie in foreign markets.
- domestic_gross_x: The gross revenue earned from the movie in the domestic (U.S.) market.
- production_budget: The production budget of the movie.

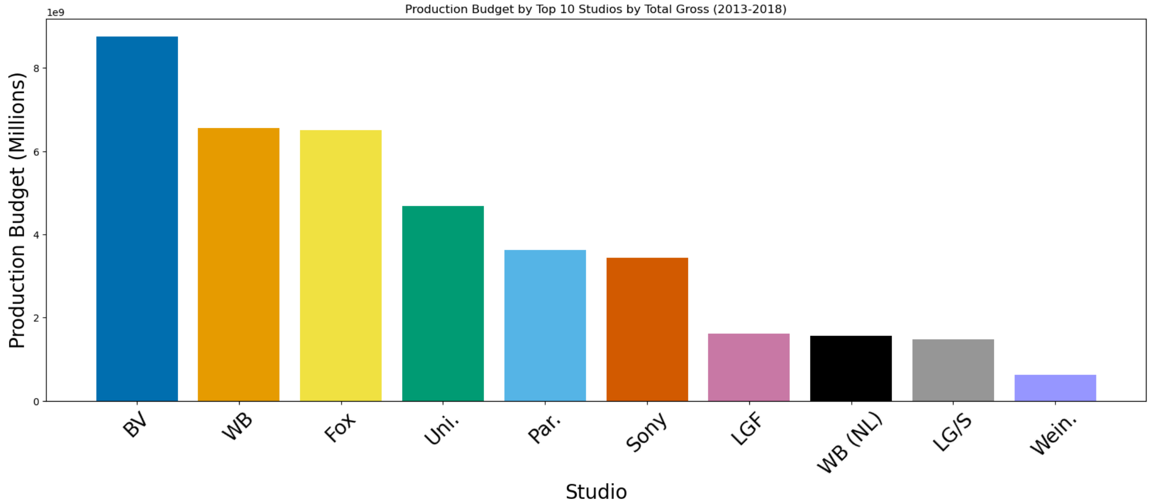
In [285]:
```
# Calculate total gross for each studio
studio_df['total_gross'] = studio_df['domestic_gross_x'] + studio_df['foreig
studio_totals = studio_df.groupby('studio')['total_gross'].sum().sort_values

# Plot bar graph
plt.figure(figsize=(20, 7))
plt.bar(studio_totals.index, studio_totals.values, color=['#0072b2', '#e69f0
plt.xticks(rotation=45, fontsize=20)
plt.xlabel('Studio', fontsize=20)
plt.ylabel('Total Gross (Millions)', fontsize=20)
plt.title('Top 10 Studios by Total Gross (2013-2018)')
plt.show()
```



The top 5 studios in terms of gross income are

- Walt Disney Studios
- 20th Century Fox
- Universal Pictures
- Warner Bros. Pictures
- Sony Pictures Entertainment

In [286]:
```
# Filter by top ten studios by total gross
top_ten_studios = studio_df.groupby('studio')['total_gross'].sum().sort_valu
studio_df_top_ten = studio_df[studio_df['studio'].isin(top_ten_studios.index

# Calculate production budget for each studio
production_df = studio_df_top_ten.groupby('studio')['production_budget'].sum

# Create a bar plot of production budget
plt.figure(figsize=(20, 7))
plt.bar(production_df.index, production_df.values, color=['#0072b2', '#e69f0
plt.xticks(rotation=45, fontsize=20)
plt.xlabel('Studio', fontsize=20)
plt.ylabel('Production Budget (Millions)', fontsize=20)
plt.title('Production Budget by Top 10 Studios by Total Gross (2013-2018)')
plt.show()
```



The top 5 studios in terms of gross income are

- Walt Disney Studios
- 20th Century Fox
- Warner Bros. Pictures
- Universal Pictures
- Paramount Pictures

Final Findings

- My analysis reveals a weakly positive correlation between production budget and return on investment, indicating that higher production budgets don't always guarantee higher returns. Therefore, Microsoft should be cautious in managing its production costs and investments to ensure profitable returns.
- There exists a strong positive correlation between worldwide gross and production budget, suggesting that films with higher budgets tend to achieve broader reach and generate higher box office revenue. This underscores the potential for Microsoft to invest in high-budget productions to maximize revenue.
- The genres of 'Horror' and 'Music' demonstrate higher return on investment, while 'Action' and 'Adventure' emerge as the most popular genres. This insight implies that Microsoft may benefit from focusing on film production within these genres to enhance

profitability.
- Based on my analysis, Microsoft could enter the film industry by acquiring intellectual property rights from leading movie studios. However, given its lack of experience in film production, Microsoft may encounter challenges adapting to the industry's unique dynamics.

In conclusion, while Microsoft has the opportunity to enter the film industry through intellectual property acquisitions, it should prioritize cost management and high-budget productions for revenue optimization. Additionally, considering profitable genres like Horror, Music, Action, and Adventure could further enhance its profitability.

Recommendations

To effectively penetrate the film industry, Microsoft should consider the following steps:

1. Undertake comprehensive market analysis.
2. Collaborate with seasoned film producers.
3. Formulate a well-defined investment plan encompassing aspects like genre trends, production budgeting, and revenue forecasts.
4. Prioritize high-budget film projects.
5. Explore production opportunities in genres that are both popular and financially rewarding.
6. Implement measures to safeguard its intellectual property rights, ensuring protection of investments within the film sector.