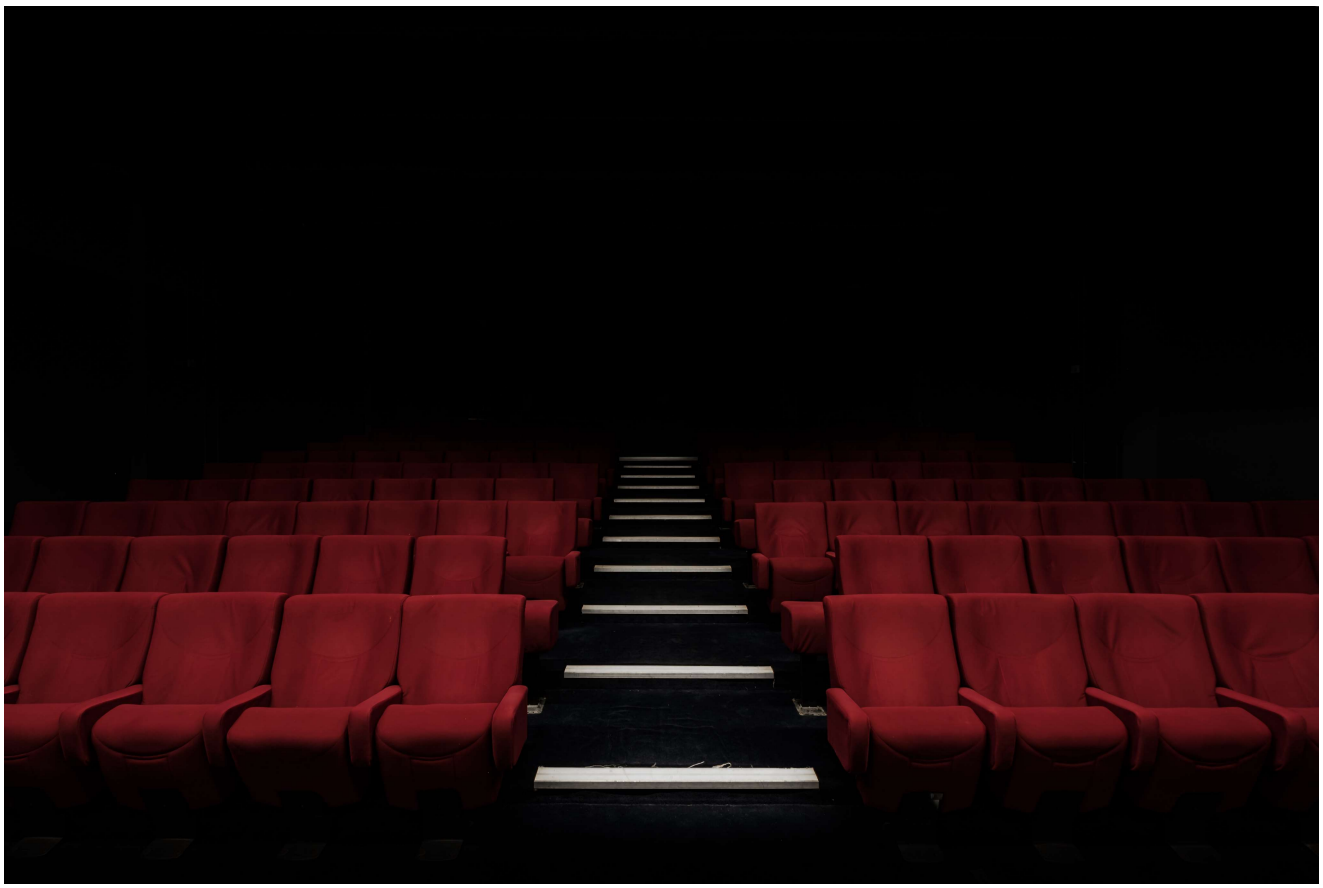


Movie Industry Analysis at Microsoft

Name: Mercy Ronoh

Overview

This project analyzes the movie industry data to provide insights to the Microsoft team which wants to venture into movie production. The objective is to explore the data through analysis, manipulation and exploration to come up with findings that can provide recommendations on where and how to kickstart the process. Microsoft can use this analysis to make decisions in regards to which type of genres to venture into, the budget they can pump into the project and the predictive timelines when they can release the movies/films into the target market. This will in turn led to high gross returns, therefore high profit margins.

Problem Statement

Microsoft is venturing into the lucrative movie-making industry, but they lack a clear starrting point. Their new movie studio is eager to dive in. Unfortunately, Microsoft has no prior knowledge of the movie industry and might have challenges moving forward blindly in creating content.
Through analysis of the recent data and practicies that are foundational for a profitable venture, I have recommended essential features that Microsoft's movie studio can incorporate for maximum revenue generation.

The main factors I have focused on are:

1. The genre type that are most successful.
2. The budget that can result to better gross returns.
3. The seasons when release can generate high gross returns.

Using the data, have described patterns, made predictions and decisions to creating a lucratibe movie industry.This analysis equips Microsoft with the knowledge needed to make well-informed strides into the film industry.

Data Understanding

I sourced data from four primary datasets to comprehensively analyze the film industry:

1. IMDB Data: movies produced, including details on production budgets, revenues, genres, and ratings.
2. Box Office Mojo Data: encompassing domestic, international, and worldwide box office earnings, genres, and more.
3. Rotten Tomatoes Data: Access to audience and critic ratings, reviews, and related insights.
4. The Numbers Data: Data from a reputable movie industry data provider, offering information on box office revenues, budgets, genres, and more for the top 100 movies per year.

These datasets collectively informed my analysis of the film industry's key success factors and guided recommendations for Microsoft's entry into movie production.

In [414]:
```
# install the necessary packages and libraries on jupyternotebook
!pip install pandas
!pip install matplotlib
!pip install seaborn
!pip install numpy
!pip install missingno
```

```
Requirement already satisfied: pandas in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (1.1.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from pan
das) (2.8.1)
Requirement already satisfied: numpy>=1.15.4 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from pandas) (1.1
8.5)
Requirement already satisfied: pytz>=2017.2 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from pandas) (202
0.1)
Requirement already satisfied: six>=1.5 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from python-dateutil>=
2.7.3->pandas) (1.15.0)
Requirement already satisfied: matplotlib in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (3.3.1)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matpl
otlib) (2.8.1)
Requirement already satisfied: certifi>=2020.06.20 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplo
tlib) (2023.7.22)
Requirement already satisfied: numpy>=1.15 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplotlib)
(1.18.5)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\admin\anaconda3\envs\learn-env\lib\site-
packages (from matplotlib) (2.4.7)
Requirement already satisfied: pillow>=6.2.0 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplotlib)
(8.0.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplotl
ib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplotlib)
(0.10.0)
Requirement already satisfied: six>=1.5 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from python-dateutil>=
2.1->matplotlib) (1.15.0)
Requirement already satisfied: seaborn in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (0.11.0)
Requirement already satisfied: scipy>=1.0 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from seaborn) (1.5.
0)
Requirement already satisfied: pandas>=0.23 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from seaborn) (1.
1.3)
Requirement already satisfied: numpy>=1.15 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from seaborn) (1.1
8.5)
Requirement already satisfied: matplotlib>=2.2 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from seaborn)
(3.3.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from pan
das>=0.23->seaborn) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from pandas>=0.23-
>seaborn) (2020.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplotlib>=
2.2->seaborn) (8.0.0)
Requirement already satisfied: cycler>=0.10 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplotlib>=
2.2->seaborn) (0.10.0)
Requirement already satisfied: certifi>=2020.06.20 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplo
tlib>=2.2->seaborn) (2023.7.22)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\admin\anaconda3\envs\learn-env\lib\site-
packages (from matplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplotl
ib>=2.2->seaborn) (1.2.0)
Requirement already satisfied: six>=1.5 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from python-dateutil>=
2.7.3->pandas>=0.23->seaborn) (1.15.0)
Requirement already satisfied: numpy in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (1.18.5)
Requirement already satisfied: missingno in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (0.5.2)
Requirement already satisfied: numpy in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from missingno) (1.18.5)
Requirement already satisfied: seaborn in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from missingno) (0.11.
0)
Requirement already satisfied: matplotlib in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from missingno) (3.
3.1)
Requirement already satisfied: scipy in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from missingno) (1.5.0)
Requirement already satisfied: pandas>=0.23 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from seaborn->miss
ingno) (1.1.3)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matpl
otlib->missingno) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplotl
ib->missingno) (1.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\admin\anaconda3\envs\learn-env\lib\site-
packages (from matplotlib->missingno) (2.4.7)
Requirement already satisfied: cycler>=0.10 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplotlib->m
issingno) (0.10.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplotlib->
missingno) (8.0.0)
Requirement already satisfied: certifi>=2020.06.20 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from matplo
tlib->missingno) (2023.7.22)
Requirement already satisfied: pytz>=2017.2 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from pandas>=0.23-
>seaborn->missingno) (2020.1)
Requirement already satisfied: six>=1.5 in c:\users\admin\anaconda3\envs\learn-env\lib\site-packages (from python-dateutil>=
2.1->matplotlib->missingno) (1.15.0)
```

In [415]:
```
#loading the libraries/packages for use in the project

import pandas as pd
import numpy as np
import os
import zipfile
import matplotlib.pyplot as plt
import seaborn as sns
from bs4 import BeautifulSoup
import requests

%matplotlib inline
```

In [851]:
```
#reading the  csv file
title_basics= pd.read_csv('zippedData/imdb.title.basics.csv.gz', compression='gzip')
bom_movie= pd.read_csv('zippedData/bom.movie_gross.csv.gz', compression='gzip')
title_ratings = pd.read_csv('zippedData/imdb.title.ratings.csv.gz', compression='gzip')
title_crew = pd.read_csv('zippedData/imdb.title.crew.csv.gz', compression='gzip')
movie_budgets = pd.read_csv('zippedData/tn.movie_budgets.csv.gz', compression='gzip')
movie_info = pd.read_csv('zippedData/rt.movie_info.tsv.gz', compression='gzip', delimiter='\t')
```

In [852]:
```
#previewing the dataframe
title_basics.head()
```

Out[852]:

	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 [853]:
```
#checking the columns details
selected_columns = ["primary_title", "genres"]
title_basics[selected_columns].tail(5)
```

Out[853]:

	primary_title	genres
146139	Kuambil Lagi Hatiku	Drama
146140	Rodolpho Teóphilo - O Legado de um Pioneiro	Documentary
146141	Dankyavar Danka	Comedy
146142	6 Gunn	NaN
146143	Chico Albuquerque - Revelações	Documentary

In [421]:
```
#getting info for the DataFrame
title_basics.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
```

In [855]:
```
#to view the genres columns count content
title_basics['genres'].value_counts().head(5)
```

Out[855]:
```
Documentary     32185
Drama           21486
Comedy           9177
Horror           4372
Comedy,Drama     3519
Name: genres, dtype: int64
```

In [425]:
```
#previewing the dataframe
title_ratings.head()
```

Out[425]:

	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 [426]:
```
#previewing the dataframe
title_ratings.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
```

In [427]:
```
#previewing the dataframe
title_crew.head(5)
```

Out[427]:

	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 [429]:
```
#previewing the dataframe
movie_budgets.head()
```

Out[429]:

	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 [857]:
```
#previewing the dataframe
movie_info.head(3)
```

Out[857]:

	id	synopsis	rating	genre	director	writer	theater_date	dvd_date	currency	box_office	runtime	studio
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971	Sep 25, 2001	NaN	NaN	104 minutes	NaN
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo	Aug 17, 2012	Jan 1, 2013	$	600,000	108 minutes	Entertainment One
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996	Apr 18, 2000	NaN	NaN	116 minutes	NaN

In [424]:
```
#previewing the dataframe information/structure
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
```

In [423]:
```
bom_movie.tail()
```

Out[423]:

	title	studio	domestic_gross	foreign_gross	year
3382	The Quake	Magn.	6200.0	NaN	2018
3383	Edward II (2018 re-release)	FM	4800.0	NaN	2018
3384	El Pacto	Sony	2500.0	NaN	2018
3385	The Swan	Synergetic	2400.0	NaN	2018
3386	An Actor Prepares	Grav.	1700.0	NaN	2018

In [431]:
```
#previewing the dataframe information/structure

movie_info.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
```

Data Preparation

With the understanding of the data,the next step is to clean up, prepare and preprocess for analysis

In [858]:
```
#missing values in the title_basics
missing_values_basics = title_basics.isnull().sum()
missing_values_basics
```

Out[858]:
```
tconst                0
primary_title         0
original_title       21
start_year            0
runtime_minutes    31739
genres              5408
dtype: int64
```

In [863]:
```
#drop the rows in the columns that are null
title_basics.dropna(subset=['original_title', 'runtime_minutes', 'genres'], inplace=True)
```

In [864]:
```
# recheck the missing values in the title_basics
missing_values_basics = title_basics.isnull().sum()
missing_values_basics
```

Out[864]:
```
tconst            0
primary_title     0
original_title    0
start_year        0
runtime_minutes   0
genres            0
dtype: int64
```

In [865]:
```
#check for missing values in title_ratings
missing_values_ratings = title_ratings.isnull().sum()
missing_values_ratings
```

Out[865]:
```
tconst           0
averagerating    0
numvotes         0
dtype: int64
```

In [866]:
```
#check for missing values in title_crew
missing_values_crew = title_crew.isnull().sum()
missing_values_crew
```

Out[866]:
```
tconst           0
directors     5727
writers      35883
dtype: int64
```

In [867]:
```
#missing valyes input 'NaN'

title_crew['directors'].fillna('Unknown', inplace=True)
title_crew['writers'].fillna('Unknown', inplace=True)
```

In [438]:
```
#next we check if the missing values are still there
missing_values_crew = title_crew.isnull().sum()
missing_values_crew
```

Out[438]:
```
tconst       0
directors    0
writers      0
dtype: int64
```

In [868]:
```
#check the number of rows and columns of the data
title_basics.shape
```

Out[868]:
```
(112232, 6)
```

In [869]:
```
# Merge title_basics and title_ratings on 'tconst'
combined_data = title_basics.merge(title_ratings, on='tconst')

# Merge title_crew on 'tconst'
combined_data = combined_data.merge(title_crew, on='tconst')

# Now, combined_data contains all the information combined over 'tconst' and print
combined_data.tail(10)
```

Out[869]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres	averagerating	numvotes	directors
65710	tt9905476	Hand Rolled	Hand Rolled	2019	90.0	Documentary	9.3	13	nm10533931,nm10523880
65711	tt9906218	Unstoppable	Unstoppable	2019	84.0	Documentary	8.1	8	nm0932216
65712	tt9908960	Pliusas	Pliusas	2018	90.0	Comedy	4.2	13	nm10534967
65713	tt9910502	Hayatta Olmaz	Hayatta Olmaz	2019	97.0	Comedy	7.0	9	nm7288238
65714	tt9910930	Jeg ser deg	Jeg ser deg	2019	75.0	Crime,Documentary	6.1	7	nm10536332
65715	tt9911774	Padmavyuhathile Abhimanyu	Padmavyuhathile Abhimanyu	2019	130.0	Drama	8.4	365	nm10536451 nm1(
65716	tt9913056	Swarm Season	Swarm Season	2019	86.0	Documentary	6.2	5	nm1502645
65717	tt9913084	Diabolik sono io	Diabolik sono io	2019	75.0	Documentary	6.2	6	nm0812850
65718	tt9914286	Sokagin Çocuklari	Sokagin Çocuklari	2019	98.0	Drama,Family	8.7	136	nm4394529
65719	tt9916160	Drømmeland	Drømmeland	2019	72.0	Documentary	6.5	11	nm5684093

In [872]:
```
#renaming the columns to have a common name for better manipulation
combined_data = combined_data.rename(columns={'start_year': 'year'})
```

In [873]:
```
#missing values in the title_basics
missing_values_bom = bom_movie.isnull().sum()
missing_values_bom
```

Out[873]:
```
title              0
studio             5
domestic_gross    28
foreign_gross   1350
year               0
dtype: int64
```

In [874]:
```
# Drop rows with missing values in 'foreign gross' and 'domestic gross' columns
bom_movie.dropna(subset=['foreign_gross', 'domestic_gross', 'studio'], inplace=True)
```

In [875]:
```
#missing values in the title_basics
missing_values_bom = bom_movie.isnull().sum()
missing_values_bom
```

Out[875]:
```
title             0
studio            0
domestic_gross    0
foreign_gross     0
year              0
dtype: int64
```

In [879]:
```
# Check the data type of 'foreign_gross', and structure of the columns e.g. name
if bom_movie['foreign_gross'].dtype == 'object':
    # Convert to float after removing commas
    bom_movie['foreign_gross'] = bom_movie['foreign_gross'].str.replace(',', '').astype(float)

bom_movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2007 entries, 0 to 3353
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: 94.1+ KB
```

In [880]:
```
# #rename columns to allow for ease of data
movie_info = movie_info.rename(columns ={'genre': 'genres', 'runtime': 'runtime minutes', 'box office':'world gross'})
```

In [883]:
```
#print the columns
print([col for col in bom_movie.columns])
```

```
['title', 'studio', 'domestic_gross', 'foreign_gross', 'year']
```

In [884]:
```
#combined_data.shape
combined_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 65720 entries, 0 to 65719
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   tconst           65720 non-null  object 
 1   primary_title    65720 non-null  object 
 2   original_title   65720 non-null  object 
 3   year             65720 non-null  int64  
 4   runtime_minutes  65720 non-null  float64
 5   genres           65720 non-null  object 
 6   averagerating    65720 non-null  float64
 7   numvotes         65720 non-null  int64  
 8   directors        65720 non-null  object 
 9   writers          65720 non-null  object 
dtypes: float64(2), int64(2), object(6)
memory usage: 5.5+ MB
```

In [886]:
```
# # Rename columns
# movie_budgets = movie_budgets.rename(columns={'production_budget': 'budgets', 'worldwide_gross': 'world_gross', 'movie': 't

# # Remove dollar signs and commas, and convert to float in one step
# movie_budgets['budgets'] = movie_budgets['budgets'].replace('[\$,]', '', regex=True).astype(float)
# movie_budgets['domestic_gross'] = movie_budgets['domestic_gross'].replace('[\$,]', '', regex=True).astype(float)
# movie_budgets['world_gross'] = movie_budgets['world_gross'].replace('[\$,]', '', regex=True).astype(float)


def clean_data(df):
    # Rename columns
    df = df.rename(columns={'production_budget': 'budgets', 'worldwide_gross': 'world_gross', 'movie': 'title'})

    # Remove dollar signs and commas, and convert to float
    df['budgets'] = df['budgets'].replace('[\$,]', '', regex=True).astype(float)
    df['domestic_gross'] = df['domestic_gross'].replace('[\$,]', '', regex=True).astype(float)
    df['world_gross'] = df['world_gross'].replace('[\$,]', '', regex=True).astype(float)

    return df

# Now you can use the function like this:
cleaned_movie_budgets = clean_data(movie_budgets)
```

In [891]:
```
#check the data structure for the cleaned dataframe for movie_budgets
cleaned_movie_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   title           5782 non-null   object 
 3   budgets         5782 non-null   float64
 4   domestic_gross  5782 non-null   float64
 5   world_gross     5782 non-null   float64
dtypes: float64(3), int64(1), object(2)
memory usage: 271.2+ KB
```

In [892]:
```
# merge combined_data and bom_movie
merged_data = pd.merge(combined_data, bom_movie, left_on=['year', 'primary_title'], right_on=['year', 'title'], how='inner')
merged_data
```

Out[892]:

	tconst	primary_title	original_title	year	runtime_minutes	genres	averagerating	numvotes	directors	
0	tt0337692	On the Road	On the Road	2012	124.0	Adventure,Drama,Romance	6.1	37886	nm0758574	
1	tt0359950	The Secret Life of Walter Mitty	The Secret Life of Walter Mitty	2013	114.0	Adventure,Comedy,Drama	7.3	275300	nm0001774	
2	tt0365907	A Walk Among the Tombstones	A Walk Among the Tombstones	2014	114.0	Action,Crime,Drama	6.5	105116	nm0291082	
3	tt0369610	Jurassic World	Jurassic World	2015	124.0	Action,Adventure,Sci-Fi	7.0	539338	nm1119880	nm0415425,nm0798646,(
4	tt0376136	The Rum Diary	The Rum Diary	2011	119.0	Comedy,Drama	6.2	94787	nm0732430	
...	...	...	...	...	...	...	...	...	...	...
1250	tt7784604	Hereditary	Hereditary	2018	127.0	Drama,Horror,Mystery	7.3	151571	nm4170048	
1251	tt7959026	The Mule	The Mule	2018	116.0	Crime,Drama,Thriller	7.1	58955	nm0000142	
1252	tt8097306	Nobody's Fool	Nobody's Fool	2018	110.0	Comedy,Drama,Romance	4.6	3618	nm1347153	
1253	tt8404272	How Long Will I Love U	Chao shi kong tong ju	2018	101.0	Romance	6.5	607	nm6050764	
1254	tt9151704	Burn the Stage: The Movie	Burn the Stage: The Movie	2018	84.0	Documentary,Music	8.8	2067	nm10201503	

1255 rows × 14 columns

In [903]:
```
print([col for col in merged_data.columns])
```

```
['tconst', 'primary_title', 'original_title', 'year', 'runtime_minutes', 'genres', 'averagerating', 'numvotes', 'directors',
'writers', 'title', 'studio', 'domestic_gross', 'foreign_gross']
```

In [904]:
```
#above merged_data plus  the cleaned_movie_budgets
merged_data1 = pd.merge(merged_data, cleaned_movie_budgets, on='title', how='inner')
merged_data1
```

Out[904]:

	tconst	primary_title	original_title	year	runtime_minutes	genres	averagerating	numvotes	directors	
0	tt0337692	On the Road	On the Road	2012	124.0	Adventure,Drama,Romance	6.1	37886	nm0758574	
1	tt0359950	The Secret Life of Walter Mitty	The Secret Life of Walter Mitty	2013	114.0	Adventure,Comedy,Drama	7.3	275300	nm0001774	
2	tt0365907	A Walk Among the Tombstones	A Walk Among the Tombstones	2014	114.0	Action,Crime,Drama	6.5	105116	nm0291082	
3	tt0369610	Jurassic World	Jurassic World	2015	124.0	Action,Adventure,Sci-Fi	7.0	539338	nm1119880	nm0415425,nm0798646,nm
4	tt0376136	The Rum Diary	The Rum Diary	2011	119.0	Comedy,Drama	6.2	94787	nm0732430	
...	...	...	...	...	...	...	...	...	...	
945	tt7388562	Paul, Apostle of Christ	Paul, Apostle of Christ	2018	108.0	Adventure,Biography,Drama	6.7	5662	nm1358366	
946	tt7401588	Instant Family	Instant Family	2018	118.0	Comedy,Drama	7.4	46728	nm1890845	
947	tt7535780	The Great Wall	The Great Wall	2017	72.0	Documentary	6.5	24	nm2738464	
948	tt7784604	Hereditary	Hereditary	2018	127.0	Drama,Horror,Mystery	7.3	151571	nm4170048	
949	tt7959026	The Mule	The Mule	2018	116.0	Crime,Drama,Thriller	7.1	58955	nm0000142	

950 rows × 19 columns

In [905]:
```
#renaming the columns
merged_data1['domestic_gross'] = merged_data1['domestic_gross_x'] + merged_data1['domestic_gross_y']
#drop the columns after renaming
merged_data1.drop(columns=['domestic_gross_x', 'domestic_gross_y'], inplace=True)
```

In [896]:
```
#checking the dataframe
merged_data1.head(3)
```

Out[896]:

averagerating	numvotes	directors	writers	title	studio	foreign_gross	id	release_date	budgets	world_gross	domestic_gross
6.1	37886	nm0758574	nm0449616,nm1433580	On the Road	IFC	8000000.0	17	Mar 22, 2013	25000000.0	9313302.0	1464828.0
7.3	275300	nm0001774	nm0175726,nm0862122	The Secret Life of Walter Mitty	Fox	129900000.0	37	Dec 25, 2013	91000000.0	187861183.0	116436838.0
6.5	105116	nm0291082	nm0088747,nm0291082	A Walk Among the Tombstones	Uni.	26900000.0	67	Sep 19, 2014	28000000.0	62108587.0	52317685.0

In [907]:
```
# merge the 2 merged_data1 and movie_info to get the final dataframe used for analysis and visualisations to generate insight
# df = pd.merge(merged_data1, movie_info, on='world_gross', how='outer')
# df.head()

df = pd.concat([merged_data1, movie_info], axis=0, ignore_index=True)
df.head()
```

Out[907]:

	tconst	primary_title	original_title	year	runtime_minutes	genres	averagerating	numvotes	directors	
0	tt0337692	On the Road	On the Road	2012.0	124	Adventure,Drama,Romance	6.1	37886.0	nm0758574	
1	tt0359950	The Secret Life of Walter Mitty	The Secret Life of Walter Mitty	2013.0	114	Adventure,Comedy,Drama	7.3	275300.0	nm0001774	
2	tt0365907	A Walk Among the Tombstones	A Walk Among the Tombstones	2014.0	114	Action,Crime,Drama	6.5	105116.0	nm0291082	
3	tt0369610	Jurassic World	Jurassic World	2015.0	124	Action,Adventure,Sci-Fi	7.0	539338.0	nm1119880	nm0415425,nm0798646,nm
4	tt0376136	The Rum Diary	The Rum Diary	2011.0	119	Comedy,Drama	6.2	94787.0	nm0732430	

5 rows × 25 columns

In [908]:
```
df.columns
```

Out[908]:
```
Index(['tconst', 'primary_title', 'original_title', 'year', 'runtime_minutes',
       'genres', 'averagerating', 'numvotes', 'directors', 'writers', 'title',
       'studio', 'foreign_gross', 'id', 'release_date', 'budgets',
       'world_gross', 'domestic_gross', 'synopsis', 'rating', 'director',
       'writer', 'theater_date', 'dvd_date', 'currency'],
      dtype='object')
```

In [909]:
```
# List of column names to drop
columns_to_drop = ['primary_title', 'original_title', 'id', 'writers', 'directors', 'synopsis', 'currency','dvd_date', 'runti
# Drop the specified columns in one line
df = df.drop(columns=columns_to_drop, axis=1)
```

In [910]:
```
#saving copy of the dataframe as a csv file
df.to_csv('./datadf.csv')
```

In [911]:
```
#preview the dataframe contents
df.head()
```

Out[911]:

	tconst	year	genres	averagerating	numvotes	title	studio	foreign_gross	release_date	budgets	world_gross	domes
0	tt0337692	2012.0	Adventure,Drama,Romance	6.1	37886.0	On the Road	IFC	8000000.0	Mar 22, 2013	25000000.0	9.3133e+06	1.46
1	tt0359950	2013.0	Adventure,Comedy,Drama	7.3	275300.0	The Secret Life of Walter Mitty	Fox	129900000.0	Dec 25, 2013	91000000.0	1.87861e+08	1.16
2	tt0365907	2014.0	Action,Crime,Drama	6.5	105116.0	A Walk Among the Tombstones	Uni.	26900000.0	Sep 19, 2014	28000000.0	6.21086e+07	5.23
3	tt0369610	2015.0	Action,Adventure,Sci-Fi	7.0	539338.0	Jurassic World	Uni.	1019.4	Jun 12, 2015	215000000.0	1.64885e+09	1.30
4	tt0376136	2011.0	Comedy,Drama	6.2	94787.0	The Rum Diary	FD	10800000.0	Oct 28, 2011	45000000.0	2.15447e+07	2.62

In [912]:
```
#drop tconst column
df.drop('tconst', axis=1, inplace=True)
```

In [914]:
```
#change the data type for foreign_gross and world_gross for ease of use to float from the object type

df['foreign_gross'] = df['foreign_gross'].astype(str)
df['foreign_gross'] = df['foreign_gross'].str.replace(',', '').astype(float)

df['world_gross'] = df['world_gross'].astype(str)
df['world_gross'] = df['world_gross'].str.replace(',', '').astype(float)
```

In [916]:
```
#checking the change of the datframe above
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2510 entries, 0 to 2509
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   year            950 non-null    float64
 1   genres          2502 non-null   object
 2   averagerating   950 non-null    float64
 3   numvotes        950 non-null    float64
 4   title           950 non-null    object
 5   studio          1444 non-null   object
 6   foreign_gross   950 non-null    float64
 7   release_date    950 non-null    object
 8   budgets         950 non-null    float64
 9   world_gross     1290 non-null   float64
 10  domestic_gross  950 non-null    float64
 11  rating          1557 non-null   object
 12  director        1361 non-null   object
 13  writer          1111 non-null   object
 14  theater_date    1201 non-null   object
dtypes: float64(7), object(8)
memory usage: 294.3+ KB
```

In [917]:
```
df.loc[df['title'].isnull()]
```

Out[917]:

	year	genres	averagerating	numvotes	title	studio	foreign_gross	release_date	budgets	world_gross	domestic_gross
950	NaN	Action and Adventure|Classics|Drama	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
951	NaN	Drama|Science Fiction and Fantasy	NaN	NaN	NaN	Entertainment One	NaN	NaN	NaN	600000.0	NaN
952	NaN	Drama|Musical and Performing Arts	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
953	NaN	Drama|Mystery and Suspense	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
954	NaN	Drama|Romance	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...
2505	NaN	Action and Adventure|Horror|Mystery and Suspense	NaN	NaN	NaN	New Line Cinema	NaN	NaN	NaN	33886034.0	NaN
2506	NaN	Comedy|Science Fiction and Fantasy	NaN	NaN	NaN	Paramount Vantage	NaN	NaN	NaN	NaN	NaN
2507	NaN	Classics|Comedy|Drama|Musical and Performing Arts	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2508	NaN	Comedy|Drama|Kids and Family|Sports and Fitness	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2509	NaN	Action and Adventure|Art House and Internation...	NaN	NaN	NaN	Columbia Pictures	NaN	NaN	NaN	NaN	NaN

1560 rows × 15 columns

In [920]:
```
#descriptive statistics to get the median of the df
medians = df.median()
medians
```

Out[920]:
```
year                 2013.0
averagerating           6.5
numvotes            94143.0
foreign_gross     45450000.0
budgets           35000000.0
world_gross       65646201.0
domestic_gross    89806013.0
dtype: float64
```

In [921]:
```
#checking the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2510 entries, 0 to 2509
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   year            950 non-null    float64
 1   genres          2502 non-null   object 
 2   averagerating   950 non-null    float64
 3   numvotes        950 non-null    float64
 4   title           950 non-null    object 
 5   studio          1444 non-null   object 
 6   foreign_gross   950 non-null    float64
 7   release_date    950 non-null    object 
 8   budgets         950 non-null    float64
 9   world_gross     1290 non-null   float64
 10  domestic_gross  950 non-null    float64
 11  rating          1557 non-null   object 
 12  director        1361 non-null   object 
 13  writer          1111 non-null   object 
 14  theater_date    1201 non-null   object 
dtypes: float64(7), object(8)
memory usage: 294.3+ KB
```

In [922]:
```
#filling the null values with median
medians = {
    'year': 2013.0,
    'averagerating': 6.5,
    'numvotes': 94143.0,
    'foreign_gross': 45450000.0,
    'budgets': 35000000.0,
    'world_gross': 65646201.0,  # Added a comma here
    'domestic_gross': 89806013.0
}

# Fill the null values with the medians
df.fillna(medians, inplace=True)
```

In [924]:
```
#checking the remaining null values
df.isnull().sum()
```

Out[924]:
```
year                0
genres              8
averagerating       0
numvotes            0
title            1560
studio           1066
foreign_gross       0
release_date     1560
budgets             0
world_gross         0
domestic_gross      0
rating            953
director         1149
writer           1399
theater_date     1309
dtype: int64
```

The few columns remaining in the dataset that not applicable to my analysis to be dropped-runtime_minutes,writer

In [925]:
```
result = df[df['title'].isnull() & df['director'].isnull()]
result
```

Out[925]:

	year	genres	averagerating	numvotes	title	studio	foreign_gross	release_date	budgets	world_gross	domestic_gross
960	2013.0	NaN	6.5	94143.0	NaN	NaN	45450000.0	NaN	35000000.0	65646201.0	89806013.0
961	2013.0	Documentary	6.5	94143.0	NaN	Showtime Documentary Films	45450000.0	NaN	35000000.0	65646201.0	89806013.0
962	2013.0	Documentary|Special Interest	6.5	94143.0	NaN	Seventh Art Releasing	45450000.0	NaN	35000000.0	65646201.0	89806013.0
966	2013.0	Drama	6.5	94143.0	NaN	Sony Pictures	45450000.0	NaN	35000000.0	99165609.0	89806013.0
970	2013.0	Musical and Performing Arts	6.5	94143.0	NaN	NaN	45450000.0	NaN	35000000.0	65646201.0	89806013.0
...	...	...	...	...	...	...	...	...	...	...	...
2493	2013.0	NaN	6.5	94143.0	NaN	NaN	45450000.0	NaN	35000000.0	65646201.0	89806013.0
2496	2013.0	Art House and International|Comedy|Drama	6.5	94143.0	NaN	NaN	45450000.0	NaN	35000000.0	794306.0	89806013.0
2499	2013.0	Art House and International|Drama	6.5	94143.0	NaN	NaN	45450000.0	NaN	35000000.0	65646201.0	89806013.0
2505	2013.0	Action and Adventure|Horror|Mystery and Suspense	6.5	94143.0	NaN	New Line Cinema	45450000.0	NaN	35000000.0	33886034.0	89806013.0
2509	2013.0	Action and Adventure|Art House and Internation...	6.5	94143.0	NaN	Columbia Pictures	45450000.0	NaN	35000000.0	65646201.0	89806013.0

199 rows × 15 columns

In [926]:
```
df.dropna(subset=['title', 'director'], how='all', inplace=True)
```

In [928]:
```
#fill missing release_dates with the most appearing date of release
df['release_date'].fillna(df['release_date'].mode().iloc[0], inplace=True)
```

In [930]:
```
df = df.drop(['writer', 'rating','director','theater_date'], axis=1)
```

In [932]:
```
#drop missing rows in studio
df = df.dropna(subset=['studio'])
df.isnull().sum()
```

Out[932]:
```
year               0
genres             0
averagerating      0
numvotes           0
title            440
studio             0
foreign_gross      0
release_date       0
budgets            0
world_gross        0
domestic_gross     0
dtype: int64
```

In [934]:
```
#drop missing rows in studio
df = df.dropna(subset=['title'])
```

Feature Engineering

Convert the year and release_date into datatype date_time values and evaluate to get the months released

In [938]:
```
#changing release date to datetime values
df['release_date'] = pd.to_datetime(df['release_date'])
df['year'] = pd.to_datetime(df['year'])
```

In [941]:
```
#create a column for release month and change it to dataframe for datetime values
df['release_month'] = df['release_date'].dt.strftime('%B')
df['release_date'] = pd.to_datetime(df['release_date'])
df['release_month'].value_counts()
```

Out[941]:
```
November     116
December     105
October       91
July          88
September     88
June          80
August        75
March         74
May           65
January       58
February      58
April         52
Name: release_month, dtype: int64
```

In [944]:
```
#checking the df dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 950 entries, 0 to 949
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   year            950 non-null    datetime64[ns]
 1   genres          950 non-null    object
 2   averagerating   950 non-null    float64
 3   numvotes        950 non-null    float64
 4   title           950 non-null    object
 5   studio          950 non-null    object
 6   foreign_gross   950 non-null    float64
 7   release_date    950 non-null    datetime64[ns]
 8   budgets         950 non-null    float64
 9   world_gross     950 non-null    float64
 10  domestic_gross  950 non-null    float64
 11  release_month   950 non-null    object
dtypes: datetime64[ns](2), float64(6), object(4)
memory usage: 96.5+ KB
```

In [945]:
```
#checking the number of rows and columns
df.shape
```

Out[945]:
```
(950, 12)
```

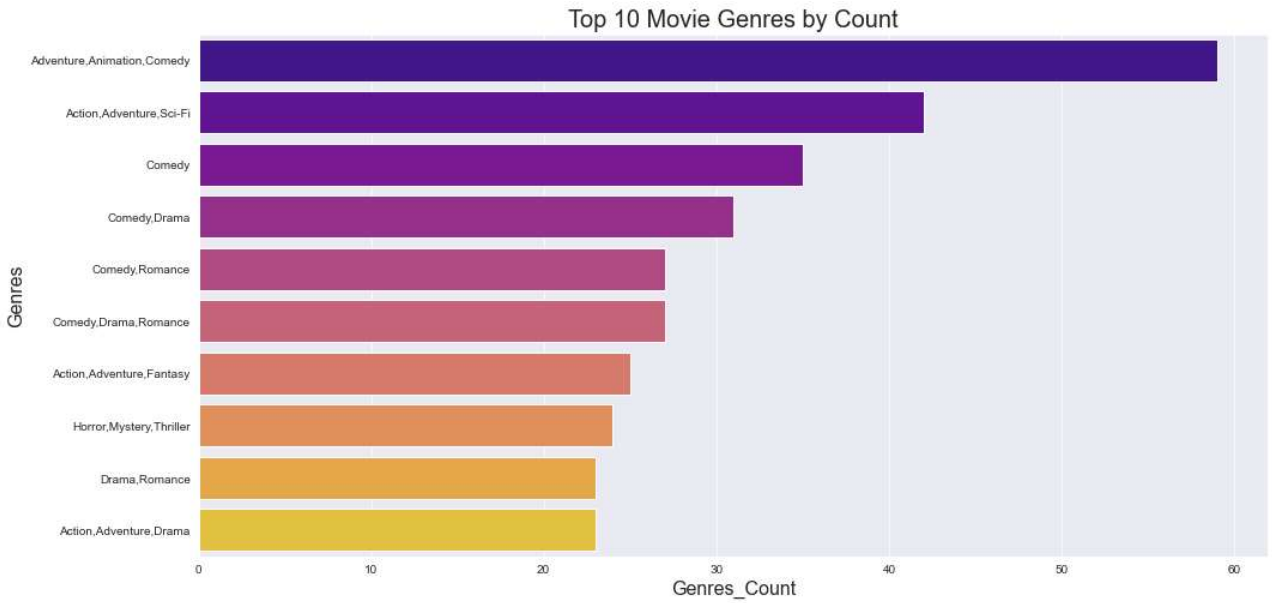
Data Analysis

I performed analysis to examine how factors such as genre, budget, and release timing influence movie revenue, with a primary emphasis on worldwide earnings. I converted the data to million-dollar units to enhance the analysis, all while aiming to estimate Microsoft's potential global revenue.

Genre type

What type of genres has the highest counts

In [946]:
```
# Calculate the top 10 genres and their counts
top_10_genres = df['genres'].value_counts().head(10)

# Create a bar plot
plt.figure(figsize=(16, 8))
sns.barplot(x=top_10_genres.values, y=top_10_genres.index, palette='plasma')
plt.xlabel('Genres_Count', fontsize = 16)
plt.ylabel('Genres',  fontsize = 16)
plt.title('Top 10 Movie Genres by Count',  fontsize = 20)

plt.show()

top_10_genres
```



Out[946]:
```
Adventure,Animation,Comedy    59
Action,Adventure,Sci-Fi       42
Comedy                        35
Comedy,Drama                  31
Comedy,Romance                27
Comedy,Drama,Romance          27
Action,Adventure,Fantasy      25
Horror,Mystery,Thriller       24
Drama,Romance                 23
Action,Adventure,Drama        23
Name: genres, dtype: int64
```

The top 5 Genre with the highest counts that are was the most common in this dataset:

1. Adventure, animation, comedy
2. Action, Adventure and Sci-Fi
3. Comedy
4. Comedy, Drama
5. Comedy, Drama, Romance

Budget

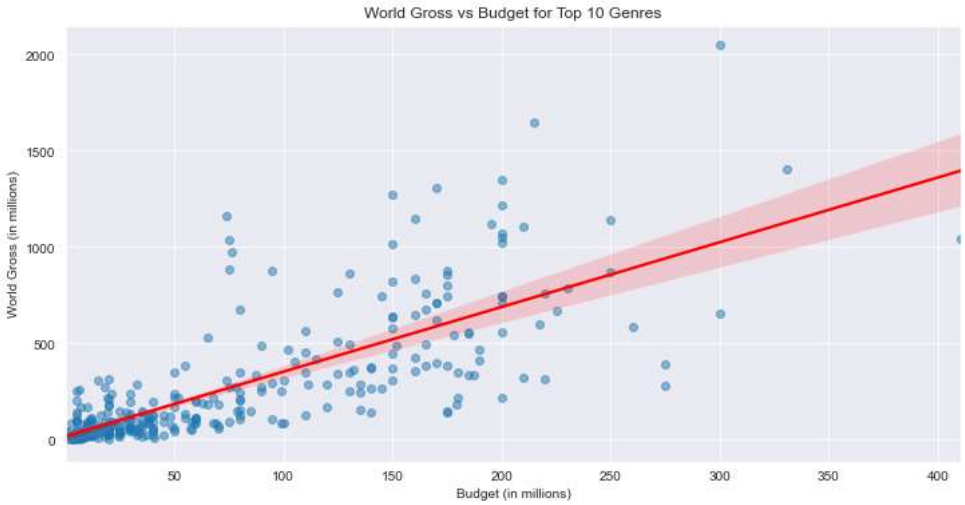
Amount of money allocated production process in comparison with the world gross earnings which is box office returns

In [950]:
```
#make the dollar amounts to be plot friendly
df['budgets_in_mil']= df['budgets']/1000000
df['world_gross_in_mil']= df['world_gross']/1000000
df['domestic_gross_in_mil']= df['domestic_gross']/1000000
df['foreign_gross_in_mil']= df['foreign_gross']/1000000
```

In [951]:
```
#check the change
df['budgets_in_mil'].describe()
```

Out[951]:
```
count    950.000000
mean      56.617487
std       59.915881
min        0.100000
25%       16.000000
50%       35.000000
75%       74.750000
max      410.600000
Name: budgets_in_mil, dtype: float64
```

In [952]:
```
#budget amount that achieves the highest world _gross which happens to be box_office, domestic_gross and foreign_gross
#plotting world gross by budget in terms of genre with line of best fit

top_10_genres = df['genres'].value_counts().head(10).index
genre_data = df[df['genres'].isin(top_10_genres)]

# Create the scatter plot with a line of best fit
plt.figure(figsize=(12, 6))
sns.regplot(x='budgets_in_mil', y='world_gross_in_mil', data=genre_data, scatter_kws={'alpha': 0.5}, line_kws={'color': 'red'
plt.xlabel('Budget (in millions)')
plt.ylabel('World Gross (in millions)')
plt.title('World Gross vs Budget for Top 10 Genres')
plt.show()
```



From the plot above;The regression line in this plot shows general increase in gross amount as the budget of the money spent in production process hence implying that the higher the production budget the higher the box office returns which is interpreted as the world_gross

In [826]:
```
# Create a filter for movies in the specified genres
genres_filter = df['genres'].str.contains('Adventure|Animation|Comedy', case=False, regex=True)

# Filter the DataFrame
filtered_df = df[genres_filter]

# Calculate the statistics for budget_in_mil
stats = filtered_df['budgets_in_mil'].describe()

# Display the statistics
print(stats)
```

```
count    520.000000
mean      75.687404
std       68.605055
min        1.000000
25%       22.000000
50%       50.000000
75%      125.000000
max      410.600000
Name: budgets_in_mil, dtype: float64
```

In [827]:

```
# Create a filter for movies in the specified genres
genres_filter2 = df['genres'].str.contains('Action|Adventure|Sci-Fi', case=False, regex=True)

# Filter the DataFrame
filtered_df2 = df[genres_filter2]
# Calculate the statistics for budget_in_mil
stats2 = filtered_df2['budgets_in_mil'].describe()
# Display the statistics

print(stats2)
```

```
count    441.000000
mean      90.822676
std       69.607254
min        1.800000
25%       35.000000
50%       70.000000
75%      140.000000
max      410.600000
Name: budgets_in_mil, dtype: float64
```

From the statistics above the mean budget for producing titles of the 'Adventure|Animation|Comedy' genre is of 75.687404 million dollars and for 'Action|Adventure|Sci-Fi' being 90.822676 million dollars

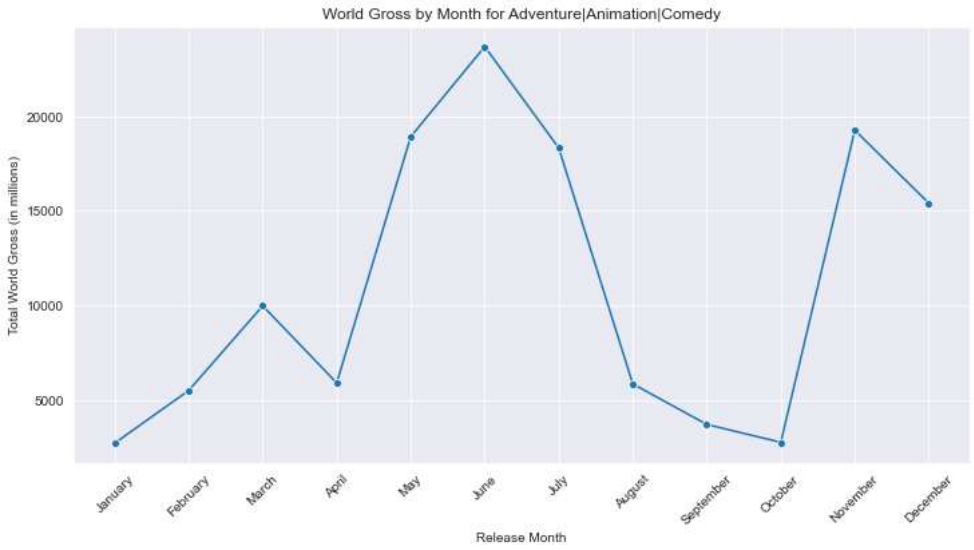
1. Optimal budget being 125million dollars and the 410 million dollars for 'Action|Adventure|Sci-Fi'
2. Optimal budget being 140 million dollars and the 410 million dollars for 'Action|Adventure|Sci-Fi'

The months to release

Checking the months with high gross returns for the top 2 genres with high counts

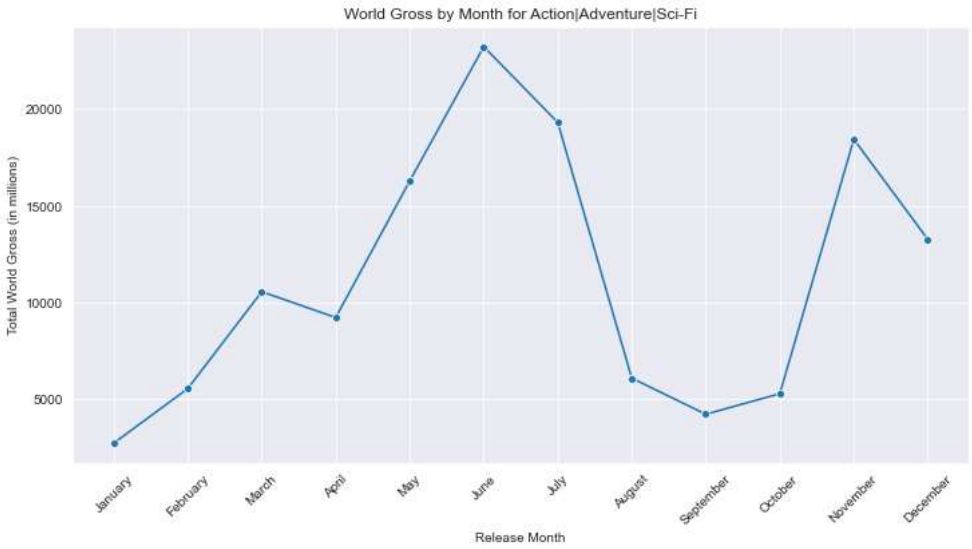
In [955]:
```
#release month with higher gross  we use the
filtered_df = df[df['genres'].str.contains('Adventure|Animation|Comedy', case=False, regex=True)]

# Create a column for the release month
filtered_df['release_month'] = filtered_df['release_date'].dt.strftime('%B')

# Group the data by release month and calculate total world gross
monthly_gross = filtered_df.groupby('release_month')['world_gross_in_mil'].sum().reset_index()

# Sort the months in order
months_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November',
monthly_gross['release_month'] = pd.Categorical(monthly_gross['release_month'], categories=months_order, ordered=True)
monthly_gross = monthly_gross.sort_values('release_month')

# Create a line plot to visualize world gross by month
plt.figure(figsize=(12, 6))
sns.lineplot(x='release_month', y='world_gross_in_mil', data=monthly_gross, marker='o')
plt.xlabel('Release Month')
plt.ylabel('Total World Gross (in millions)')
plt.title('World Gross by Month for Adventure|Animation|Comedy')
plt.xticks(rotation=45)
plt.show()
```

```
<ipython-input-955-8abaffa73d31>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_df['release_month'] = filtered_df['release_date'].dt.strftime('%B')
```



In [956]:
```
#release month with higher gross  we use the
filtered_df = df[df['genres'].str.contains( 'Action|Adventure|Sci-Fi', case=False, regex=True)]

# Create a column for the release month
filtered_df['release_month'] = filtered_df['release_date'].dt.strftime('%B')

# Group the data by release month and calculate total world gross
monthly_gross = filtered_df.groupby('release_month')['world_gross_in_mil'].sum().reset_index()

# Sort the months in order
months_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November',
monthly_gross['release_month'] = pd.Categorical(monthly_gross['release_month'], categories=months_order, ordered=True)
monthly_gross = monthly_gross.sort_values('release_month')

# Create a line plot to visualize world gross by month
plt.figure(figsize=(12, 6))
sns.lineplot(x='release_month', y='world_gross_in_mil', data=monthly_gross, marker='o')
plt.xlabel('Release Month')
plt.ylabel('Total World Gross (in millions)')
plt.title('World Gross by Month for Action|Adventure|Sci-Fi')
plt.xticks(rotation=45)
plt.show()
```

```
<ipython-input-956-0b1587f16899>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_df['release_month'] = filtered_df['release_date'].dt.strftime('%B')
```



In [957]:
```
# Create a column for the release month
filtered_df['release_month'] = filtered_df['release_date'].dt.strftime('%B')

# Group the data by release month and calculate the mean and median world gross
monthly_stats = filtered_df.groupby('release_month')['world_gross_in_mil'].agg(['mean', 'median']).reset_index()

# Sort the months in order
months_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November',
monthly_stats['release_month'] = pd.Categorical(monthly_stats['release_month'], categories=months_order, ordered=True)
monthly_stats = monthly_stats.sort_values('release_month')

monthly_stats
#
```

```
<ipython-input-957-18b24917521d>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_df['release_month'] = filtered_df['release_date'].dt.strftime('%B')
```

Out[957]:

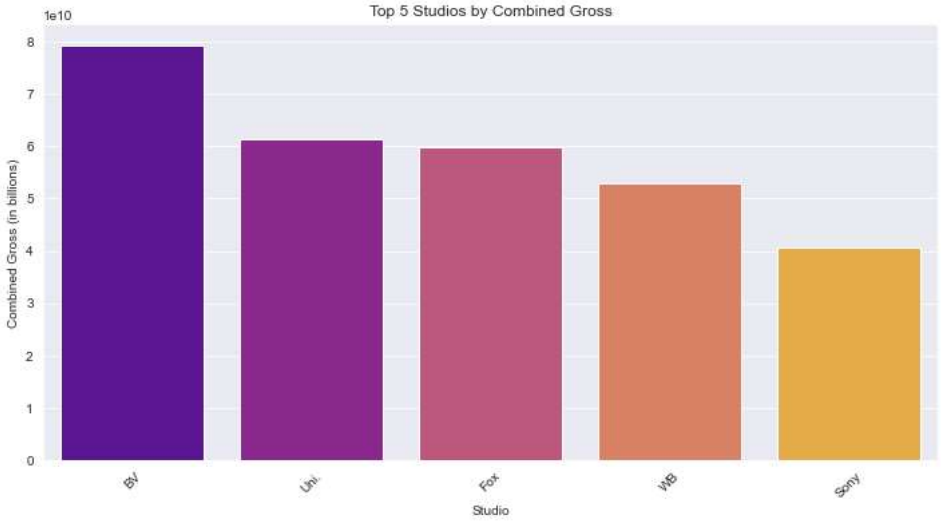
	release_month	mean	median
4	January	100.133434	76.347393
3	February	190.624952	108.957098
7	March	234.476968	163.981261
0	April	418.740878	193.118089
8	May	428.609442	337.393446
6	June	455.797018	355.408305
5	July	357.773605	297.383014
1	August	164.104515	82.182803
11	September	120.097268	84.393749
10	October	169.901474	140.783360
9	November	341.616789	191.587839
2	December	340.079831	249.517956

Release for a movie title to achieve high gross amounts should be done between April and July for the first peak season with the second peak being between November and December

Supplementary features

What else do the top genres performing have in common

In [974]:
```
# Calculate the combined gross (domestic + foreign + world) for each studio
df['combined_gross'] = df['domestic_gross'] + df['foreign_gross'] + df['world_gross']

# Group the data by studio and calculate the total combined gross
studio_gross = df.groupby('studio')['combined_gross'].sum().reset_index()

# Sort the studios by combined gross in descending order
studio_gross = studio_gross.sort_values(by='combined_gross', ascending=False)

# Select the top 5 studios
top_5_studios = studio_gross.head(5)

# Create a histogram to visualize the combined gross for the top 5 studios
plt.figure(figsize=(12, 6))
sns.barplot(x='studio', y='combined_gross', data=top_5_studios, palette='plasma')
plt.xlabel('Studio')
plt.ylabel('Combined Gross (in billions)')
plt.title('Top 5 Studios by Combined Gross')
plt.xticks(rotation=45)
plt.show()
```



In [973]:
```
# Group the data by studio and calculate the mean combined gross
mean_combined_gross = top_5_studios.groupby('studio')['combined_gross'].mean().reset_index()

# Sort the studios by mean combined gross in descending order
mean_combined_gross = mean_combined_gross.sort_values(by='combined_gross', ascending=False)

# Create a new column for mean combined gross in millions
mean_combined_gross['mean_combined_gross_in_mil'] = mean_combined_gross['combined_gross'] / 1000000

print(mean_combined_gross)
```

```
  studio  combined_gross  mean_combined_gross_in_mil
0     BV    7.931747e+10                79317.474431
3   Uni.    6.137832e+10                61378.315085
1    Fox    5.991252e+10                59912.518177
4     WB    5.301350e+10                53013.504166
2   Sony    4.063669e+10                40636.689046
```

Microsoft to focus on the competitors data as the BV (Buerna Vista), Universal, Fox, Warner Brothers, Sony and check into their data or benchmark and even hire personnel from the studios as they have potential to generate total gross of upto a maximum of 79317 million dollars

Conclusion

In conclusion, the focus should be on producing the top five genres with the highest counts, which are the most common in this dataset:

Adventure, Animation, Comedy Action, Adventure, and Sci-Fi Comedy Comedy, Drama Comedy, Drama, Romance

Based on the statistics, the mean budget for producing titles in the Adventure,Animation,Comedy' genre is approximately 75.69 million, while for Action,Adventure,Sci-Fi as it it is about 90.82 million.

Optimal budgets for achieving successful outcomes vary, with an optimal budget of around 125 million for Adventure,Animation,Comedy and 410 million for Action,Adventure,Sci-Fi.

Analyzing the plot, the regression line illustrates a general increase in gross amounts as the production budget increases. This indicates that a higher production budget tends to result in higher box office returns, interpreted as world gross revenue.

Furthermore, our analysis of release months suggests that for a movie title to achieve high gross amounts, it should be released between April and July, which marks the first peak season, with a second peak occurring between November and December.

In light of these findings, Microsoft should consider breaking into the movie industry with strategic insights. Additionally, it's worth noting that top-performing studios like BV (Buena Vista), Universal, Fox, Warner Brothers, and Sony have shown remarkable success in generating substantial total gross revenue, with BV leading at approximately $79,317 million. Microsoft could explore benchmarking against these industry leaders and consider personnel hiring from these studios to leverage their expertise and further enhance its prospects in the movie industry."

Future Considerations

In future, for Microsoft to accelerate the prospects for growth, they could consider adding additional data to allow for below considerations to be analysed:

Tapping into Emerging Genres and Viewer Preferences: Microsoft Movie Studio should continue to explore evolving market trends. This involves staying attuned to emerging movie genres and shifts in viewer preferences. By identifying and capitalizing on these trends, the studio can remain at the forefront of audience demands. Conducting regular market research and audience surveys can provide invaluable insights into what viewers are looking for in their entertainment.

Optimizing Marketing Strategies: To maximize revenue, Microsoft should delve into the effectiveness of its advertising and promotional strategies. Analyzing the impact of various marketing campaigns on box office performance is crucial. Implementing data-driven advertising and tailoring promotional efforts to specific audience segments can lead to a more efficient allocation of resources and higher returns on investment.

Leveraging the Power of Remakes: Another avenue for Microsoft is to explore opportunities for remakes. Remaking classic movies or reimagining existing content can tap into nostalgic and timeless themes that resonate with audiences. By breathing new life into beloved stories, the studio can appeal to both existing fans and a new generation of viewers. Identifying which properties have remake potential and applying a fresh creative vision can be a winning strategy.

Incorporating these future analyses can help Microsoft Movie Studio navigate the dynamic landscape of the film industry and continue to produce successful and captivating movies.