FILM PROJECT:
Box Office Analysis for New Movie Studio

1.BUSINESS UNDERSTANDING

1.1. Business Overview

Our project is to analyze current box office trends and competitor strategies to provide actionable insights for the company's new movie studio.

1.2Business Objectives

Our main objectives are to ensure data-backed decisions on film production, budgeting, and marketing. Also, consider some of the factors clients consider most in the film industry's popularity

1.3 Business Success Criteria

(i)Financial Success
-Target Gross Revenue per Film: E.g., achieve an average worldwide gross of at least 6 million per film.
-Profitability per Film: Aim for a minimum profit margin (e.g., 30% of budget) for at least 70% of released films.
(ii)Audience Engagement & Brand Building
-Audience Satisfaction-Achieve average audience scores above a certain threshold
-Social Media Engagement: Build a strong online presence and engagement around releases
-Brand Recognition: Establish the studio as a recognized and respected name in the industry within 5 years.

2.0 DATA UNDERSTANDING

2.1 Data understanding overview

-Files have different formats. Some are compressed CSV (comma-separated values) or TSV (tab-separated values) files that can be opened using spreadsheet software or pd.read_csv, while the data from IMDB is located in a SQLite database.

2.2 Python Library Importation

In [3]:
```
import pandas as pd
import sqlite3
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

2.3 Files importation

The files we will use in the analysis are in different formats, some in CSV, TSV, and a database file.
We obtained our data from a GitHub account, which you can Visit by clicking. For our projects , we will use more than 2 files

In [8]:
```
movie_gross=pd.read_csv("FILES/bom.movie_gross.csv")
```

Movies gross incomes per year

In [9]:
```
movie_gross.head(10)
```

Out[9]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
3	Inception	WB	292600000.0	535700000	2010
4	Shrek Forever After	P/DW	238700000.0	513900000	2010
5	The Twilight Saga: Eclipse	Sum.	300500000.0	398000000	2010
6	Iron Man 2	Par.	312400000.0	311500000	2010
7	Tangled	BV	200800000.0	391000000	2010
8	Despicable Me	Uni.	251500000.0	291600000	2010
9	How to Train Your Dragon	P/DW	217600000.0	277300000	2010

2.4 Data Information

In [4]:
```
movie_gross.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 [5]:
```
#add the two gross amounts to the new column gross_amount
movie_gross['foreign_gross'] = movie_gross['foreign_gross'].str.replace(',', '', re
movie_gross
```

Out[5]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000.0	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000.0	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000.0	2010
3	Inception	WB	292600000.0	535700000.0	2010
4	Shrek Forever After	P/DW	238700000.0	513900000.0	2010
...	...	...	...	...	...
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

3387 rows × 5 columns

2.5 Shape of the data

In [6]:
```
movie_gross.shape
```

Out[6]:
```
(3387, 5)
```

we found the data has 3387 rows of data and 5 columns

2.6 Finding the missing values

In [7]:
```
movie_gross.isna().idxmax()
```

Out[7]:
```
title              0
studio           210
domestic_gross   230
foreign_gross    222
year               0
dtype: int64
```

2.7 percentage of the missing values

In [8]:
```
total=len(movie_gross)
movie_gross.isna().sum()*100/total
```

Out[8]:
```
title             0.000000
studio            0.147623
domestic_gross    0.826690
foreign_gross    39.858282
year              0.000000
dtype: float64
```

From the data , the column with the highest percentage of the missing values is Foreign_gross with 39.9%

2.8 Checking for the duplicates

In [9]:
```
movie_gross.duplicated().value_counts()
```

Out[9]:
```
False    3387
Name: count, dtype: int64
```

From our Data , we don't have any duplicates

2.9 Filling the null values

In [10]:
```
#beginning with the integer/float data types
# We will fill the missing values with the median
for col in movie_gross.select_dtypes(include='float64').columns:
    if movie_gross[col].isnull().any():
        median_val = movie_gross[col].median()
        movie_gross[col].fillna(median_val, inplace=True)
```

In [11]:
```
# We will fill the missing values in objects with the mode
for col in movie_gross.select_dtypes(include='object').columns:
    if movie_gross[col].isnull().any():
        mode_val = movie_gross[col].mode()[0]
        movie_gross[col].fillna(mode_val, inplace=True)
```

In [12]:
```
movie_gross.isna().idxmax()
```

Out[12]:
```
title             0
studio            0
domestic_gross    0
foreign_gross     0
year              0
dtype: int64
```

We have filled our data appropriately, no null values in our Data

In [13]:
```
movie_gross
```

Out[13]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000.0	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000.0	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000.0	2010
3	Inception	WB	292600000.0	535700000.0	2010
4	Shrek Forever After	P/DW	238700000.0	513900000.0	2010
...	...	...	...	...	...
3382	The Quake	Magn.	6200.0	18700000.0	2018
3383	Edward II (2018 re-release)	FM	4800.0	18700000.0	2018
3384	El Pacto	Sony	2500.0	18700000.0	2018
3385	The Swan	Synergetic	2400.0	18700000.0	2018
3386	An Actor Prepares	Grav.	1700.0	18700000.0	2018

3387 rows × 5 columns

3.1. Movie Popularity

In [7]:
```
tmdb_movies=pd.read_csv("FILES/tmdb.movies.csv", index_col=0)
tmdb_movies.head(10)
```

Out[7]:

	genre_ids	id	original_language	original_title	popularity	release_date	title
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon
2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2
3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story
4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception
5	[12, 14, 10751]	32657	en	Percy Jackson & the Olympians: The Lightning T...	26.691	2010-02-11	Percy Jackson & the Olympians: The Lightning T...
6	[28, 12, 14, 878]	19995	en	Avatar	26.526	2009-12-18	Avatar
7	[16, 10751, 35]	10193	en	Toy Story 3	24.445	2010-06-17	Toy Story 3
8	[16, 10751, 35]	20352	en	Despicable Me	23.673	2010-07-09	Despicable Me
9	[16, 28, 35, 10751, 878]	38055	en	Megamind	22.855	2010-11-04	Megamind

In [15]:
```
tmdb_movies.shape
```

Out[15]: (26517, 9)

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

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

Our data has 26517 row data and 10 colmns

3.1 We need drop other columns

In [16]:
```
tmdb_movies.drop(columns=["original_language","genre_ids","original_title","id"],in
```

In [17]:
```
tmdb_movies
```

Out[17]:

	popularity	release_date	title	vote_average	vote_count
0	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	7.7	10788
1	28.734	2010-03-26	How to Train Your Dragon	7.7	7610
2	28.515	2010-05-07	Iron Man 2	6.8	12368
3	28.005	1995-11-22	Toy Story	7.9	10174
4	27.920	2010-07-16	Inception	8.3	22186
...	...	...	...	...	...
26512	0.600	2018-10-13	Laboratory Conditions	0.0	1
26513	0.600	2018-05-01	_EXHIBIT_84xxx_	0.0	1
26514	0.600	2018-10-01	The Last One	0.0	1
26515	0.600	2018-06-22	Trailer Made	0.0	1
26516	0.600	2018-10-05	The Church	0.0	1

26517 rows × 5 columns

In [18]:
```
tmdb_movies.isna().max()
```

Out[18]:
```
popularity      False
release_date    False
title           False
vote_average    False
vote_count      False
dtype: bool
```

We dont have any null valus from our data set

3.2 Checking of the Duplicates

In [19]:
```
tmdb_movies.duplicated().value_counts()
```

Out[19]:
```
False    25497
True      1020
Name: count, dtype: int64
```

In [20]:
```
#dropping duplicates from the data set
tmdb_movies.drop_duplicates(inplace=True)
```

In [21]:
```
tmdb_movies.duplicated().value_counts()
```

Out[21]:
```
False    25497
Name: count, dtype: int64
```

All duplicated have been drops from the data set

3.3 Merging the two dataFrames, tmdb_movies and movies_gross

In [22]:
```
mg_data=pd.merge(tmdb_movies,movie_gross,on="title")
mg_data
```

Out[22]:

	popularity	release_date	title	vote_average	vote_count	studio	domestic_gro
0	28.734	2010-03-26	How to Train Your Dragon	7.7	7610	P/DW	217600000
1	28.515	2010-05-07	Iron Man 2	6.8	12368	Par.	312400000
2	27.920	2010-07-16	Inception	8.3	22186	WB	292600000
3	24.445	2010-06-17	Toy Story 3	7.7	8340	BV	415000000
4	23.673	2010-07-09	Despicable Me	7.2	10057	Uni.	251500000
...	...	...	...	...	...	...	
2451	2.903	2018-11-30	Elliot: The Littlest Reindeer	3.4	7	Scre.	24300
2452	2.707	2018-02-02	Bilal: A New Breed of Hero	6.8	54	VE	491000
2453	2.550	2018-02-09	La Boda de Valentina	6.3	7	PNT	2800000
2454	2.276	2018-01-12	Mukkabaaz	7.5	18	Eros	75900
2455	0.600	2018-11-09	Last Letter	6.0	1	CL	181000

2456 rows × 9 columns

In [118…

```
#coverting the dataFrame to excel file
mg_data.to_excel('movie_final.xlsx', index=False)
```

3.4. Swapping title as the main index

In [23]:

```
mg_data.set_index("title").reset_index(drop=False)
```

Out[23]:

	title	popularity	release_date	vote_average	vote_count	studio	domestic_gro
0	How to Train Your Dragon	28.734	2010-03-26	7.7	7610	P/DW	217600000
1	Iron Man 2	28.515	2010-05-07	6.8	12368	Par.	312400000
2	Inception	27.920	2010-07-16	8.3	22186	WB	292600000
3	Toy Story 3	24.445	2010-06-17	7.7	8340	BV	415000000
4	Despicable Me	23.673	2010-07-09	7.2	10057	Uni.	251500000
...	...	...	...	...	...	...	
2451	Elliot: The Littlest Reindeer	2.903	2018-11-30	3.4	7	Scre.	24300
2452	Bilal: A New Breed of Hero	2.707	2018-02-02	6.8	54	VE	491000
2453	La Boda de Valentina	2.550	2018-02-09	6.3	7	PNT	2800000
2454	Mukkabaaz	2.276	2018-01-12	7.5	18	Eros	75900
2455	Last Letter	0.600	2018-11-09	6.0	1	CL	181000

2456 rows × 9 columns

3.4 Production_budget data set

In [12]:
```
movie_budget=pd.read_csv("FILES/tn.movie_budgets.csv",index_col=0)
movie_budget
```

Out[12]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747
...	...	...	...	...	...
78	Dec 31, 2018	Red 11	$7,000	$0	$0
79	Apr 2, 1999	Following	$6,000	$48,482	$240,495
80	Jul 13, 2005	Return to the Land of Wonders	$5,000	$1,338	$1,338
81	Sep 29, 2015	A Plague So Pleasant	$1,400	$0	$0
82	Aug 5, 2005	My Date With Drew	$1,100	$181,041	$181,041

5782 rows × 5 columns

3.5 Removing null in the data sets

In [25]:
```
movie_budget.isna().max()
```

Out[25]:
```
release_date         False
movie                False
production_budget    False
domestic_gross       False
worldwide_gross      False
dtype: bool
```

No null in the data set

3.6 Chcking fo the duplicates

In [26]:
```
#checkin of the data sets
movie_budget.duplicated().value_counts()
```

Out[26]:
```
False    5782
Name: count, dtype: int64
```

We have no duplicates in the data set

3.7 Shape of the Data set

In [27]:
```
movie_budget.shape
```

Out[27]:
```
(5782, 5)
```

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 5782 entries, 1 to 82
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   release_date       5782 non-null   object
 1   movie              5782 non-null   object
 2   production_budget  5782 non-null   object
 3   domestic_gross     5782 non-null   object
 4   worldwide_gross    5782 non-null   object
dtypes: object(5)
memory usage: 271.0+ KB
```

we have a row data set of 5782, and 5 columns

3.8 Merging the Movie budget data set and the merged data set(mg_data)

In [28]:
```
big_mg_data=pd.concat([movie_budget,mg_data],ignore_index=True)
big_mg_data.tail()
```

Out[28]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	popularit
8233	2018-11-30	NaN	NaN	24300.0	NaN	2.90
8234	2018-02-02	NaN	NaN	491000.0	NaN	2.70
8235	2018-02-09	NaN	NaN	2800000.0	NaN	2.55
8236	2018-01-12	NaN	NaN	75900.0	NaN	2.27
8237	2018-11-09	NaN	NaN	181000.0	NaN	0.60

3.9 Dropping of the null columns from the data set

In [29]:
```
big_mg_data.isnull().idxmax()
```

Out[29]:
```
release_date           0
movie               5782
production_budget   5782
domestic_gross         0
worldwide_gross     5782
popularity             0
title                  0
vote_average           0
vote_count             0
studio                 0
foreign_gross          0
year                   0
dtype: int64
```

4.0 Importing of the DATABASE TABLES

In [14]:
```
conn=sqlite3.connect("FILES/im.db")
```

In [15]:
```
cur=conn.cursor()
```

4.1 Reading from the database table person

In [16]:
```
pd.read_sql_query("""SELECT * FROM persons""", conn).head()
```

Out[16]:

	person_id	primary_name	primary_profession
0	nm0061671	Mary Ellen Bauder	miscellaneous,production_manager,producer
1	nm0061865	Joseph Bauer	composer,music_department,sound_department
2	nm0062070	Bruce Baum	miscellaneous,actor,writer
3	nm0062195	Axel Baumann	camera_department,cinematographer,art_department
4	nm0062798	Pete Baxter	production_designer,art_department,set_decorator

4.2 Dropping the NaN columns

In [33]:
```
cur.execute("""ALTER TABLE persons DROP COLUMN death_year """)
```

```
---------------------------------------------------------------------------
OperationalError                          Traceback (most recent call last)
Cell In[33], line 1
----> 1 cur.execute("""ALTER TABLE persons DROP COLUMN death_year """)

OperationalError: no such column: "death_year"
```

In [34]:
```
pd.read_sql_query("""SELECT * FROM persons""", conn).head()
```

Out[34]:

	person_id	primary_name	primary_profession
0	nm0061671	Mary Ellen Bauder	miscellaneous,production_manager,producer
1	nm0061865	Joseph Bauer	composer,music_department,sound_department
2	nm0062070	Bruce Baum	miscellaneous,actor,writer
3	nm0062195	Axel Baumann	camera_department,cinematographer,art_department
4	nm0062798	Pete Baxter	production_designer,art_department,set_decorator

4.3 Reading from the Principals table

In [35]:
```
pd.read_sql_query("""SELECT * FROM principals""", conn).tail()
```

Out[35]:

	movie_id	ordering	person_id	category	job	characters
1028181	tt9692684	1	nm0186469	actor	None	["Ebenezer Scrooge"]
1028182	tt9692684	2	nm4929530	self	None	["Herself","Regan"]
1028183	tt9692684	3	nm10441594	director	None	None
1028184	tt9692684	4	nm6009913	writer	writer	None
1028185	tt9692684	5	nm10441595	producer	producer	None

4.4. Joining the two tables , Persons and principles table

In [36]:
```
pd.read_sql_query("""SELECT * FROM persons
JOIN principals
ON persons.person_id=principals.person_id""",conn).head()
```

Out[36]:

	person_id	primary_name	primary_profession	movie_id	orde
0	nm0061671	Mary Ellen Bauder	miscellaneous,production_manager,producer	tt2398241	
1	nm0061865	Joseph Bauer	composer,music_department,sound_department	tt0433397	
2	nm0061865	Joseph Bauer	composer,music_department,sound_department	tt1681372	
3	nm0061865	Joseph Bauer	composer,music_department,sound_department	tt2281215	
4	nm0061865	Joseph Bauer	composer,music_department,sound_department	tt2387710	

4.5. Looking at the person id with the highest number of Ordering AND category

In [37]:
```
pd.read_sql_query("""SELECT movie_id,COUNT(ordering) AS Ordering,person_id
FROM principals
GROUP BY person_id
ORDER BY Ordering DESC
LIMIT 5""", conn)
```

Out[37]:

	movie_id	Ordering	person_id
0	tt1801509	378	nm1930572
1	tt1274300	160	nm0000636
2	tt1921111	148	nm0000616
3	tt2156899	126	nm0103977
4	tt2414424	103	nm4394575

In [38]:
```
#checking person id with the primary name
pd.read_sql_query("""SELECT primary_name,person_id
FROM persons
WHERE person_id="nm1930572" """,conn)
```

Out[38]:

	primary_name	person_id
0	Kevin MacLeod	nm1930572

In [39]:
```
#cheking  person_id and its category
pd.read_sql_query("""SELECT category,person_id
FROM principals
WHERE person_id="nm1930572"
LIMIT 1 """,conn)
```

Out[39]:

	category	person_id
0	composer	nm1930572

CONC: Composer KEVIN MACLEOD id_no "nm1930572", have the highest ordering films of 378 to clients

4.6 Reading from the table writer
Identifying which person had the highest writer Films

In [40]:
```
#LEFT JOINING OF THE TWO TABLES, PERSONS AND WRITERS
pd.read_sql_query("""SELECT * FROM persons
LEFT JOIN writers
ON persons.person_id=writers.person_id""", conn).head()
```

Out[40]:

	person_id	primary_name	primary_profession	movie_id
0	nm0061671	Mary Ellen Bauder	miscellaneous,production_manager,producer	None
1	nm0061865	Joseph Bauer	composer,music_department,sound_department	None
2	nm0062070	Bruce Baum	miscellaneous,actor,writer	None
3	nm0062195	Axel Baumann	camera_department,cinematographer,art_department	None
4	nm0062798	Pete Baxter	production_designer,art_department,set_decorator	None

In [41]:
```
#
pd.read_sql_query("""SELECT COUNT(person_id) AS person_id_count,person_id
FROM writers
GROUP BY movie_id
ORDER BY person_id_count DESC """, conn).head()
```

Out[41]:

	person_id_count	person_id
0	3818	nm6031788
1	2397	nm0685673
2	2392	nm5437847
3	2013	nm2449187
4	1770	nm0627159

In [42]:
```
#cheking the highest writer's name
pd.read_sql_query("""SELECT person_id,primary_name
FROM persons
WHERE person_id="nm6031788"
LIMIT 1 """,conn)
```

Out[42]:

	person_id	primary_name
0	nm6031788	Frank Appache

CONC: In the Film industry, the highest Writer is **Frank Appache** with ID nm6031788

4.7. Checking at the Best Directors in the industry

In [43]:
```
#reading from the directors tables
pd.read_sql_query("""SELECT* FROM directors
LIMIT 5""",conn)
```

Out[43]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0462036	nm1940585
2	tt0835418	nm0151540
3	tt0835418	nm0151540
4	tt0878654	nm0089502

In [44]:
```
#Directors with the highest number of counts

pd.read_sql_query("""SELECT COUNT(person_id) AS person_id_count,person_id
FROM directors
GROUP BY movie_id
ORDER BY person_id_count DESC """, conn).head()
```

Out[44]:

	person_id_count	person_id
0	3818	nm4429747
1	2397	nm0294492
2	2392	nm4712424
3	2013	nm7576911
4	1770	nm5074519

In [45]:
```
#Checking at the directors name
pd.read_sql_query("""SELECT person_id,primary_name
FROM persons
WHERE person_id="nm4429747"
LIMIT 1 """,conn)
```

Out[45]:

	person_id	primary_name
0	nm4429747	Liz Salvato

CONC: The best director from the industry is Liz Salvato

4.8. Checking at the Best Movies with the highest number of orders

In [46]:
```
#displaying the movie_basics
pd.read_sql_query("""SELECT* FROM movie_basics
LIMIT 5""",conn)
```

Out[46]:

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

In [47]:
```
#movie id  with the highest number of orders from principals table

pd.read_sql_query("""SELECT movie_id,COUNT(ordering) AS ordering,person_id
FROM principals
GROUP BY person_id
ORDER BY ordering DESC
LIMIT 5""", conn)
```

Out[47]:

	movie_id	ordering	person_id
0	tt1801509	378	nm1930572
1	tt1274300	160	nm0000636
2	tt1921111	148	nm0000616
3	tt2156899	126	nm0103977
4	tt2414424	103	nm4394575

In [49]:
```
#from the movies,  print out the  movie name with the highest number of orders usin
pd.read_sql_query("""SELECT primary_title,original_title,start_year,runtime_minutes
FROM movie_basics
WHERE movie_id="tt1801509" """, conn)
```

Out[49]:

	primary_title	original_title	start_year	runtime_minutes	genres
0	Goles y metas	Goles y metas	2010	6.0	Documentary,Drama,Family

In [50]:
```
#Output the person of the movie
pd.read_sql_query("""SELECT primary_name,person_id FROM persons
WHERE person_id="nm1930572"
""",conn)
```

Out[50]:

	primary_name	person_id
0	Kevin MacLeod	nm1930572

CONC: The movie with the highest order is **Goles y metas**, composed on the year 2010

4.9.Movie with the highest number of ratings

In [51]:
```
#displaying the movie_ratings
pd.read_sql_query("""SELECT* FROM movie_ratings
LIMIT 5""",conn)
```

Out[51]:

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

In [52]:
```
#highest number of vote ratings
pd.read_sql_query("""SELECT  movie_id,MAX(numvotes) AS NUMB_VOTES_ratings
FROM movie_ratings
ORDER BY NUMB_VOTES_ratings DESC
LIMIT 5
 """, conn)
```

Out[52]:

	movie_id	NUMB_VOTES_ratings
0	tt1375666	1841066

In [53]:
```
#Highest number of averaging ratings
pd.read_sql_query("""SELECT  movie_id,MAX(averagerating) AS AVG_VOTES_ratings
FROM movie_ratings
ORDER BY AVG_VOTES_ratings DESC

 """, conn)
```

Out[53]:

	movie_id	AVG_VOTES_ratings
0	tt5390098	10.0

In [54]:
```
#IDENTIFY THE TYPE OF THE FILMS WITH THE HIGHEST NUMBER OF NUMBER OF VOTES RATINGS
pd.read_sql_query("""SELECT original_title,runtime_minutes,genres,movie_id,start_ye
FROM movie_basics
WHERE movie_id="tt1375666" """,conn)
```

Out[54]:

	original_title	runtime_minutes	genres	movie_id	start_year
0	Inception	148.0	Action,Adventure,Sci-Fi	tt1375666	2010

In [55]:
```
#IDENTIFY THE TYPE OF THE FILMS WITH THE HIGHEST NUMBER OF AVG VOTES RATINGS
pd.read_sql_query("""SELECT original_title,runtime_minutes,genres,movie_id,start_ye
FROM movie_basics
WHERE movie_id="tt5390098" """,conn)
```

Out[55]:

	original_title	runtime_minutes	genres	movie_id	start_year
0	Atlas Mountain: Barbary Macaques - Childcaring...	59.0	Documentary	tt5390098	2015

CONC:
From the analysis, the movie with the highest number of votes is **Inception**
The movie with the highest average is **Atlas Mountain: Barbary Macaques - Childcaring./**

5.0 Movie Regional_Popularity

In [56]:
```
#print out the table ,Movies_Akas
pd.read_sql_query("""SELECT * FROM movie_akas
LIMIT 5
""",conn)
```

Out[56]:

	movie_id	ordering	title	region	language	types	attributes	is_original_t
0	tt0369610	10	Джурасик свят	BG	bg	None	None	
1	tt0369610	11	Jurashikku warudo	JP	None	imdbDisplay	None	
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	None	imdbDisplay	None	
3	tt0369610	13	O Mundo dos Dinossauros	BR	None	None	short title	
4	tt0369610	14	Jurassic World	FR	None	imdbDisplay	None	

In [57]:
```
#checking on the movie popularity per Region
pd.read_sql_query("""SELECT title,movie_id,COUNT(region) AS region_count,region
FROM movie_akas
GROUP BY region
ORDER BY region_count DESC
```

```
LIMIT 10
 """, conn)
```

Out[57]:

	title	movie_id	region_count	region
0	Jurassic World 3D	tt0369610	51490	US
1	Jurassic World 3D	tt0369610	18467	XWW
2	Мир Юрского периода	tt0369610	13817	RU
3	Jurassic World 3D	tt0369610	11634	DE
4	Jurassic World	tt0369610	10990	FR
5	Jurassic World	tt0369610	9007	ES
6	Cloud Cuckoo Land	tt0381957	8942	GB
7	Monde jurassique	tt0369610	8871	CA
8	Park jurajski 4	tt0369610	8691	PL
9	John Carter Maaveeran	tt0401729	8435	IN

In [58]:

```
#Region with the highest Order
pd.read_sql_query("""SELECT title,movie_id,MAX(ordering) AS MAX_ORDER,region
FROM movie_akas
GROUP BY region
ORDER BY MAX_ORDER DESC
LIMIT 5
 """, conn)
```

Out[58]:

	title	movie_id	MAX_ORDER	region
0	Žvaigždžiu karai: galia nubunda	tt2488496	61	LT
1	Star Wars: Güç Uyaniyor	tt2488496	60	TR
2	Star Wars: Episódio VII - O Despertar da Força	tt2488496	59	PT
3	Star Wars: O Despertar da Força	tt2488496	58	BR
4	Star Wars: Das Erwachen der Macht	tt2488496	57	DE

In [59]:

```
#movie_ rating
pd.read_sql_query("""SELECT movie_id,averagerating,numvotes
FROM movie_ratings
WHERE movie_id="tt0369610" """,conn)
```

Out[59]:

	movie_id	averagerating	numvotes
0	tt0369610	7.0	539338

CONC:
From the analysis, the region with the highest number of Movie counts is

"US", MOVIE-Jurassic World 3D,
with counts of 51490 AND ratings of (avgrate-7.0), num_votes-539338

5.1. MOVIE WITH THE HIGHEST WRITER

In [60]:
```
#confirmation of the movie with the highest writer
pd.read_sql_query("""SELECT COUNT(person_id) AS COUNT_person,movie_id
FROM writers
GROUP BY movie_id
ORDER BY COUNT_person DESC
LIMIT 5 """, conn)
```

Out[60]:

	COUNT_person	movie_id
0	3818	tt4050462
1	2397	tt3091166
2	2392	tt2249786
3	2013	tt4942694
4	1770	tt3528906

In [61]:
```
#Displaying the movie name with the highest writer
pd.read_sql_query("""SELECT movie_id,primary_title,original_title,start_year,runtim
FROM movie_basics
WHERE movie_id="tt4050462" """,conn)
```

Out[61]:

	movie_id	primary_title	original_title	start_year	runtime_minutes
0	tt4050462	World of Death	World of Death	2016	142.0

In [65]:
```
#understanding the movie ratings
pd.read_sql_query("""SELECT movie_id,averagerating,numvotes
FROM movie_ratings
WHERE movie_id="tt4050462" """,conn)
```

Out[65]:

	movie_id	averagerating	numvotes
0	tt4050462	7.2	31

In [64]:
```
#movie with the highest writer popularity
pd.read_sql_query("""SELECT region,title, ordering, language
FROM movie_akas
WHERE movie_id="tt4050462" """,conn)
```

Out[64]:

	region	title	ordering	language
0	US	World of Death	1	None

In [63]:
```
#understanding the name of the writer

pd.read_sql_query("""SELECT COUNT(person_id) AS WRITER,person_id
FROM writers
WHERE movie_id="tt4050462" """, conn)
```

Out[63]:

	WRITER	person_id
0	3818	nm6031788

In [62]:
```
#printing out the name_writer of
pd.read_sql_query("""SELECT person_id,primary_name,primary_profession
FROM persons
WHERE person_id="nm6031788" """, conn)
```

Out[62]:

	person_id	primary_name	primary_profession
0	nm6031788	Frank Appache	director,writer,editor

CONC:
The movie with the highest writers is **WORLD OF DEATH**

With the Rating of averagely **7.2**

Popularity is in US

Name of the writer is **Frank Appache**

Year of Production is **2016**

In [ ]:
```

```

6.0.DATA EXPLORATORY ANALYSIS

(i)Movie_gross

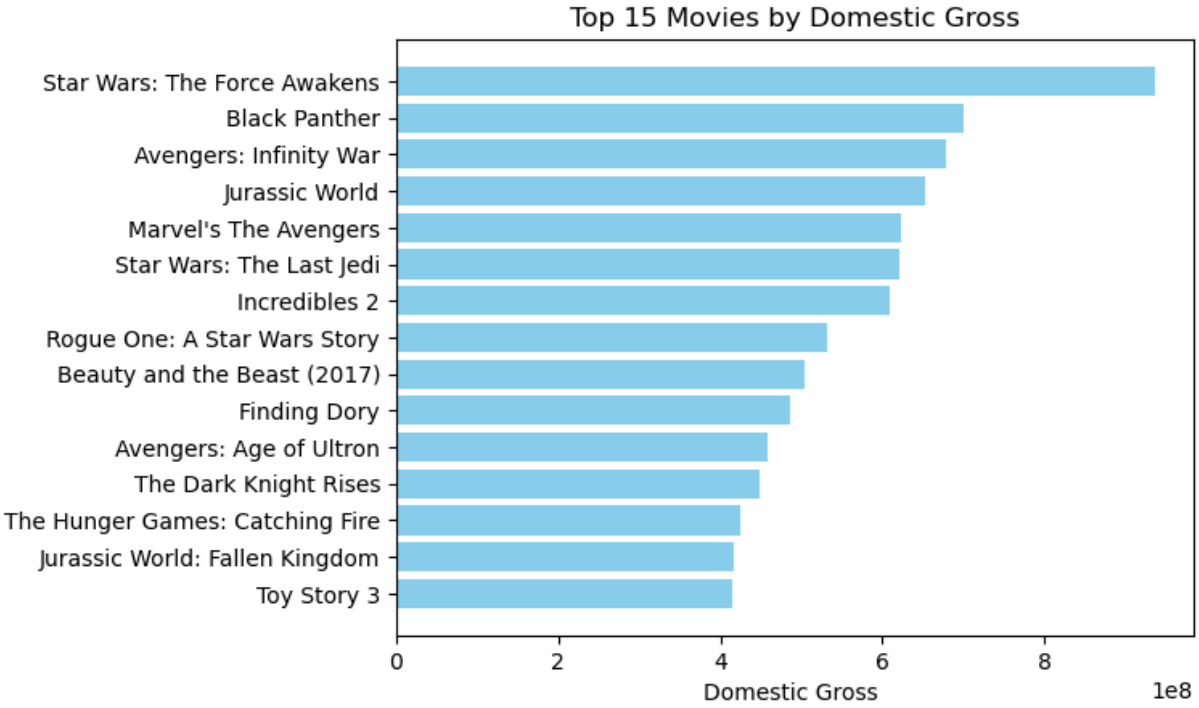
In [66]:
```
movie_gross.head()
```

Out[66]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000.0	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000.0	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000.0	2010
3	Inception	WB	292600000.0	535700000.0	2010
4	Shrek Forever After	P/DW	238700000.0	513900000.0	2010

6.1. Formulating a bar graph movie title with the highest Domestic_gross Income

In [84]:
```
#plotting
top_movies = movie_gross.sort_values(by='domestic_gross', ascending=False).head(15)


#plt.figure(figsize=(12, 8))
plt.barh(top_movies['title'], top_movies['domestic_gross'], color='skyblue')
plt.xlabel('Domestic Gross ')
plt.title('Top 15 Movies by Domestic Gross')
plt.gca().invert_yaxis()  # Highest grossing at the top

plt.show()
```

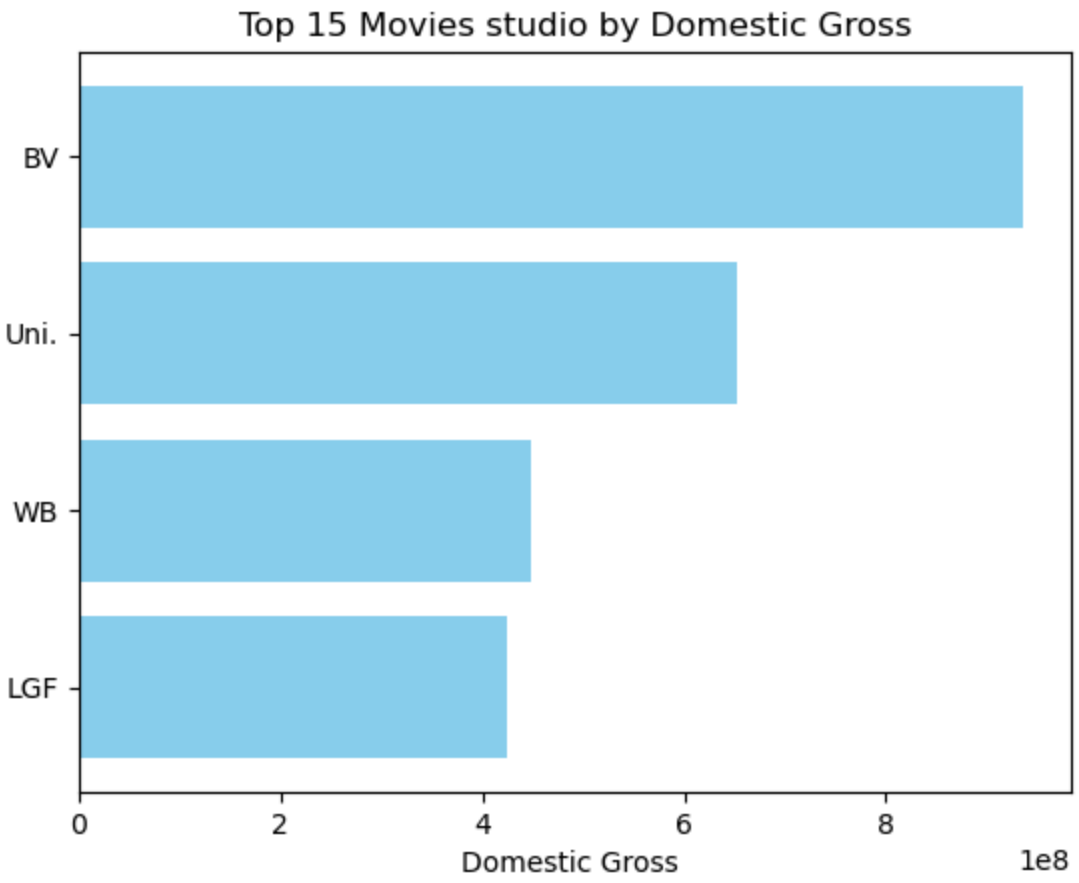


CONC: The Movie title with the highest domestic_Gross is **STAR WARS: The FORCE AWAKENS**

6.2.BAR graph of top studios with the highest domestic_gross

In [87]:
```
#plotting bar grapgh on studio name with the highest domestic gross
top_movies =movie_gross.sort_values(by='domestic_gross', ascending=False).head(15)

# Plot
#plt.figure(figsize=(10, 8))
plt.barh(top_movies["studio"], top_movies['domestic_gross'], color='skyblue')
plt.xlabel('Domestic Gross ')
plt.title('Top 15 Movies studio by Domestic Gross')
plt.gca().invert_yaxis()  # Highest grossing at the top
```

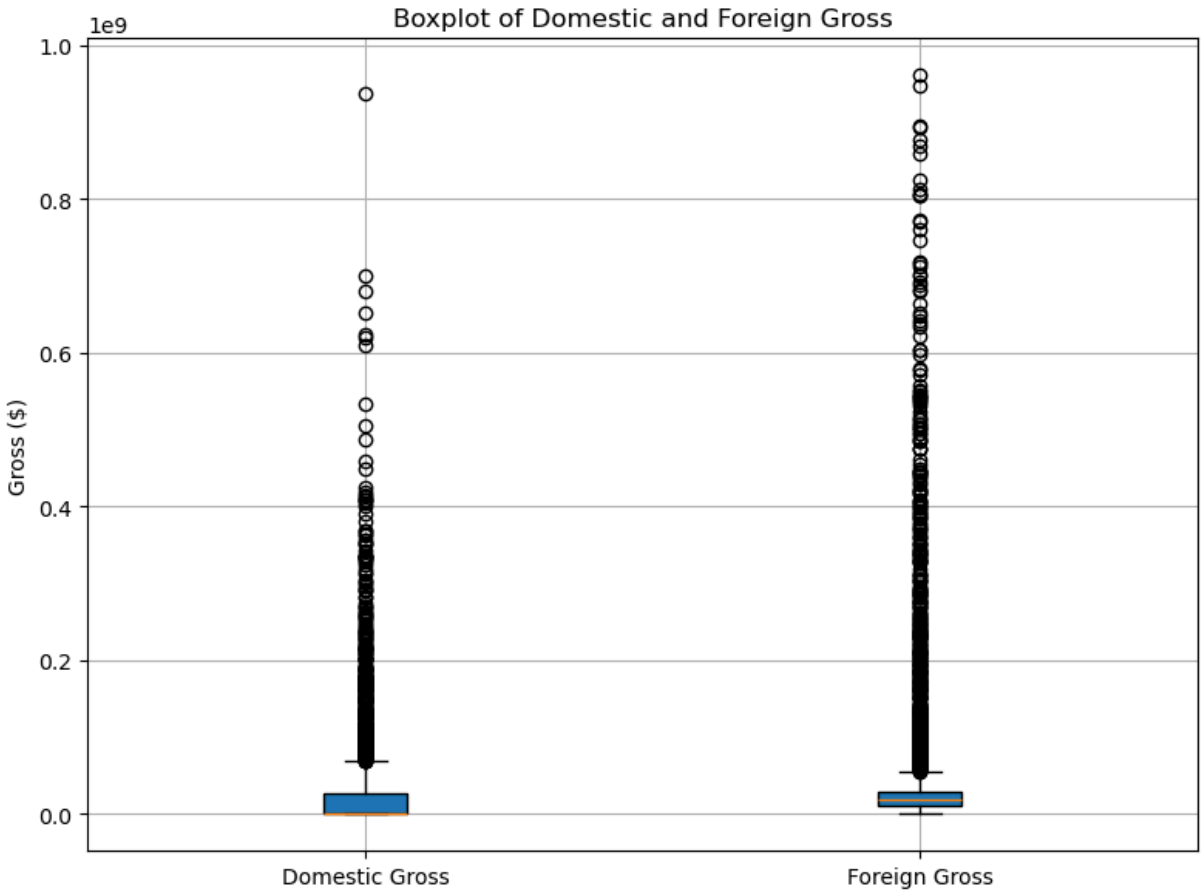
```
plt.show()
```



CONC: The movie_studio with the highest Domestic_gross is **BV**

6.3.Identifing of the outliers from the data set

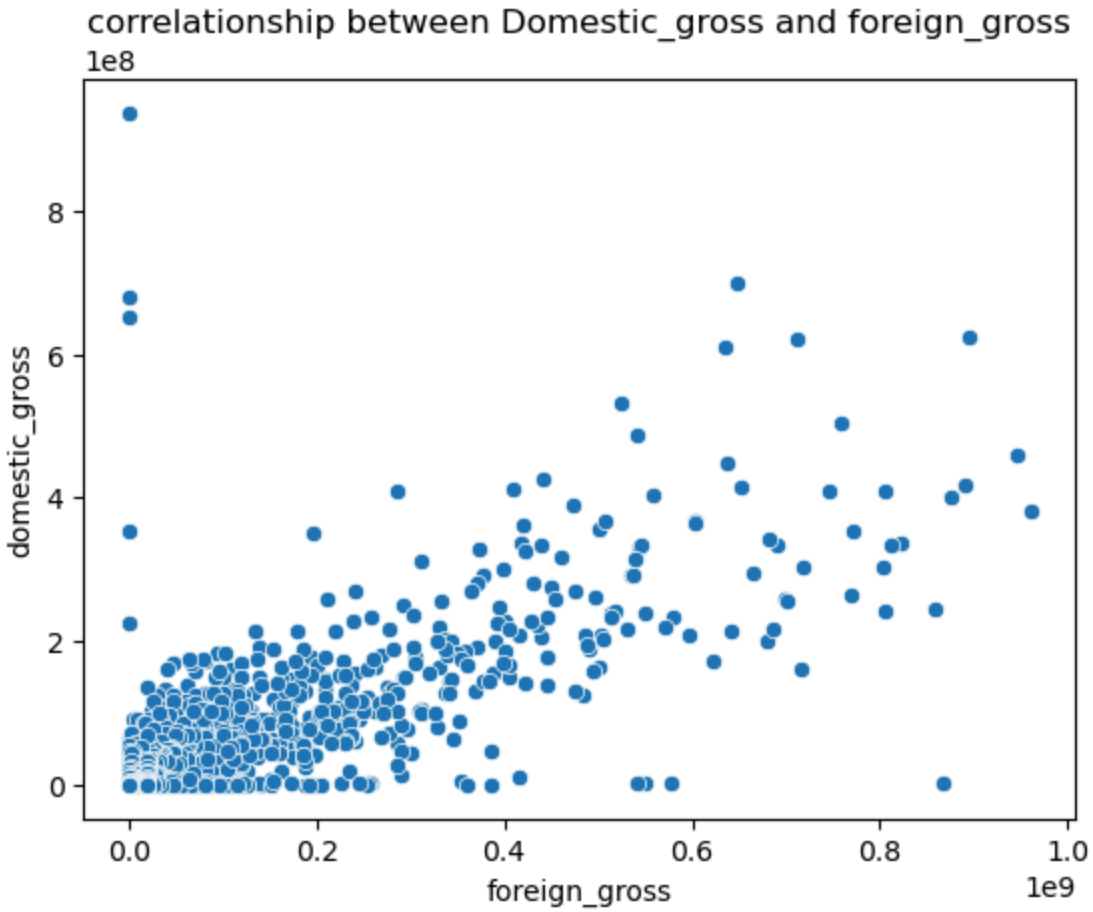
In [92]:
```
#identifying of the data set outliers from the data set
data = [movie_gross['domestic_gross'].dropna(), movie_gross['foreign_gross'].dropna

# Plot
plt.figure(figsize=(8, 6))
plt.boxplot(data, labels=['Domestic Gross', 'Foreign Gross'], patch_artist=True)

# Add titles and labels
plt.title('Boxplot of Domestic and Foreign Gross')
plt.ylabel('Gross ($)')
plt.grid(True)
plt.tight_layout()
plt.show()
```



6.4. Correlationship between domestic and the foreign gross

In [116…

```
#finding out the relationship between domestic and the foreign gross
sns.scatterplot(x='foreign_gross', y='domestic_gross', data=movie_gross)
plt.title("correlationship between Domestic_gross and foreign_gross")
plt.show()
```

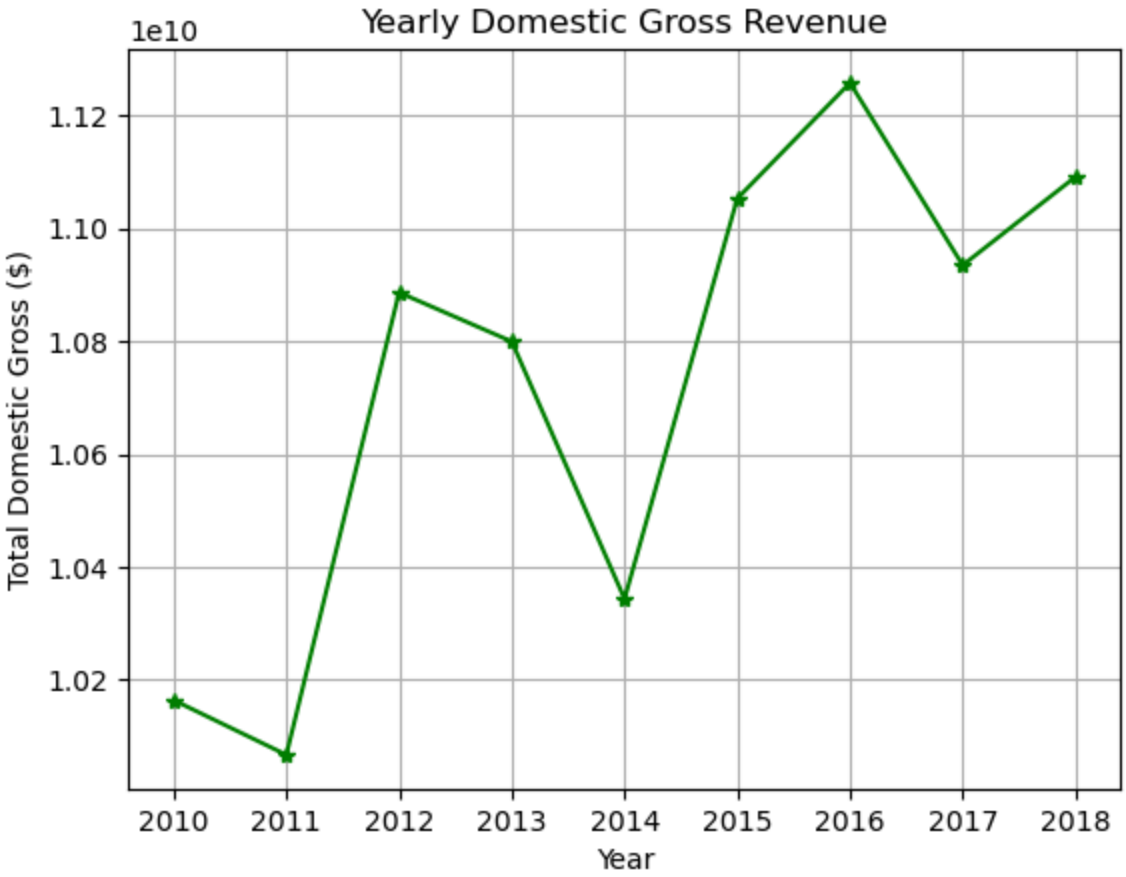


CONC: There is a **positive correlationship** between domestic and foreign gross

6.5. line graph showing the trends in domestic_gross per anum

In [117…

```
yearly_domestic = movie_gross.groupby('year')['domestic_gross'].sum().reset_index()

plt.plot(yearly_domestic['year'], yearly_domestic['domestic_gross'], marker='*', li
plt.title('Yearly Domestic Gross Revenue')
plt.xlabel('Year')
plt.ylabel('Total Domestic Gross ($)')
plt.grid(True)
plt.show()
```



CONC: There is a gradual increase in the domestic gross per year

6.6.
(ii) movie popularity

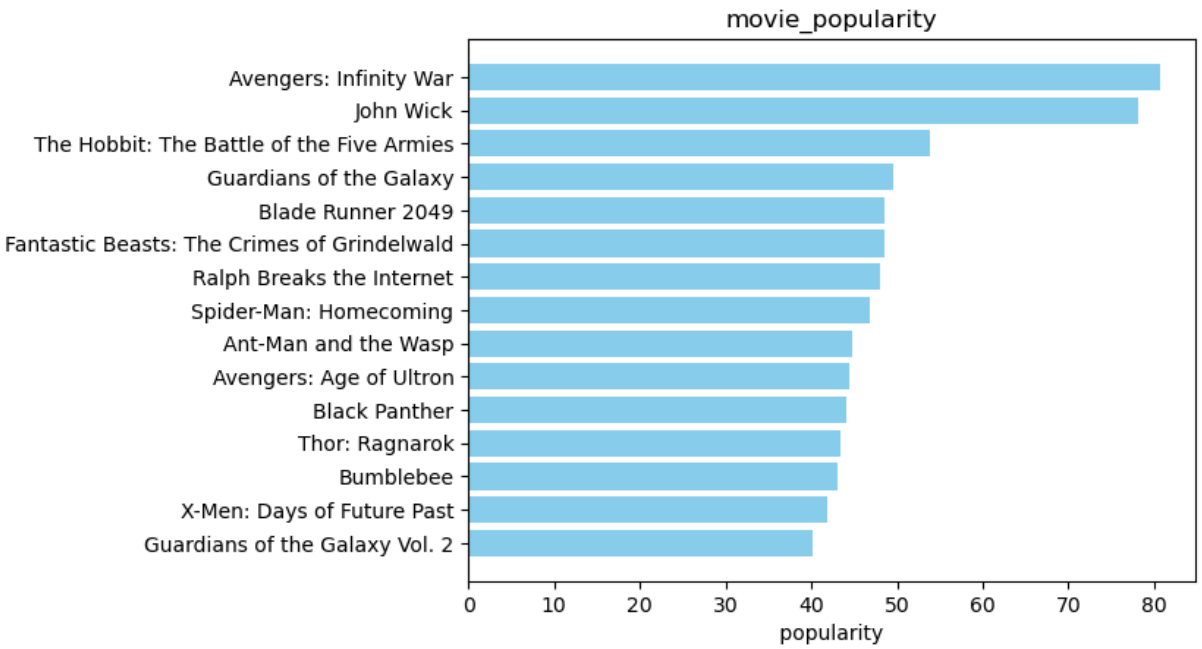
In [123…
```
mg_data.head()
```

Out[123…

	popularity	release_date	title	vote_average	vote_count	studio	domestic_gross
0	28.734	2010-03-26	How to Train Your Dragon	7.7	7610	P/DW	217600000.0
1	28.515	2010-05-07	Iron Man 2	6.8	12368	Par.	312400000.0
2	27.920	2010-07-16	Inception	8.3	22186	WB	292600000.0
3	24.445	2010-06-17	Toy Story 3	7.7	8340	BV	415000000.0
4	23.673	2010-07-09	Despicable Me	7.2	10057	Uni.	251500000.0

In [ ]:

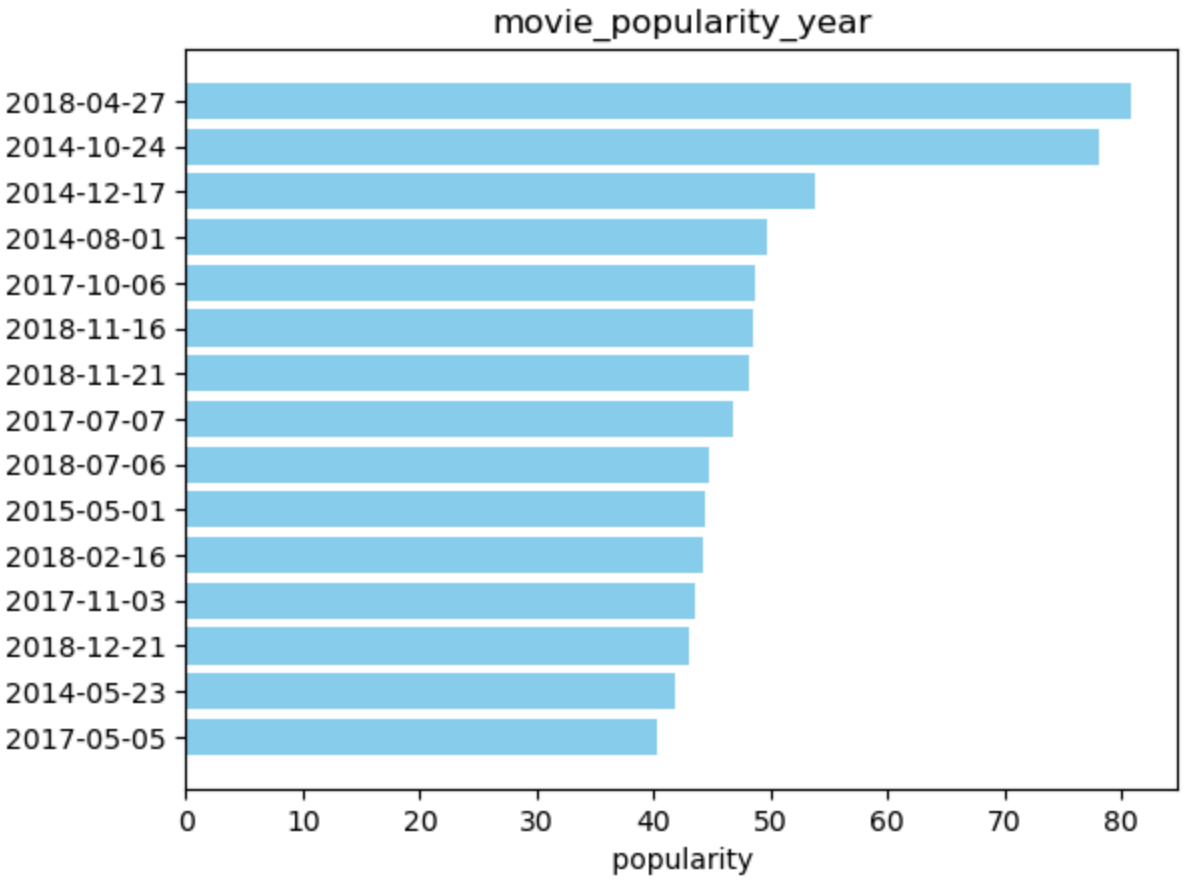
In [127…
```
#plotting bar grapgh on studio name with the highest popularity
top_movie_popularity =mg_data.sort_values(by='popularity', ascending=False).head(15

# Plot
#plt.figure(figsize=(10, 8))
plt.barh(top_movie_popularity["title"], top_movie_popularity['popularity'], color='
plt.xlabel(' popularity ')
plt.title('movie_popularity')
plt.gca().invert_yaxis()  # Highest grossing at the top

plt.show()
```



CONC: The most popular movie is Avengers: infinity awar

6.7.movie_popularity per Year

In [132…
```
#plotting bar graph on studio name with the highest popularity
top_movie_popularity_year =mg_data.sort_values(by='popularity', ascending=False).he

# Plot
#plt.figure(figsize=(10, 8))
plt.barh(top_movie_popularity["release_date"], top_movie_popularity['popularity'],
plt.xlabel(' popularity ')
plt.title('movie_popularity_year')
plt.gca().invert_yaxis()  # Highest grossing at the top

plt.show()
```

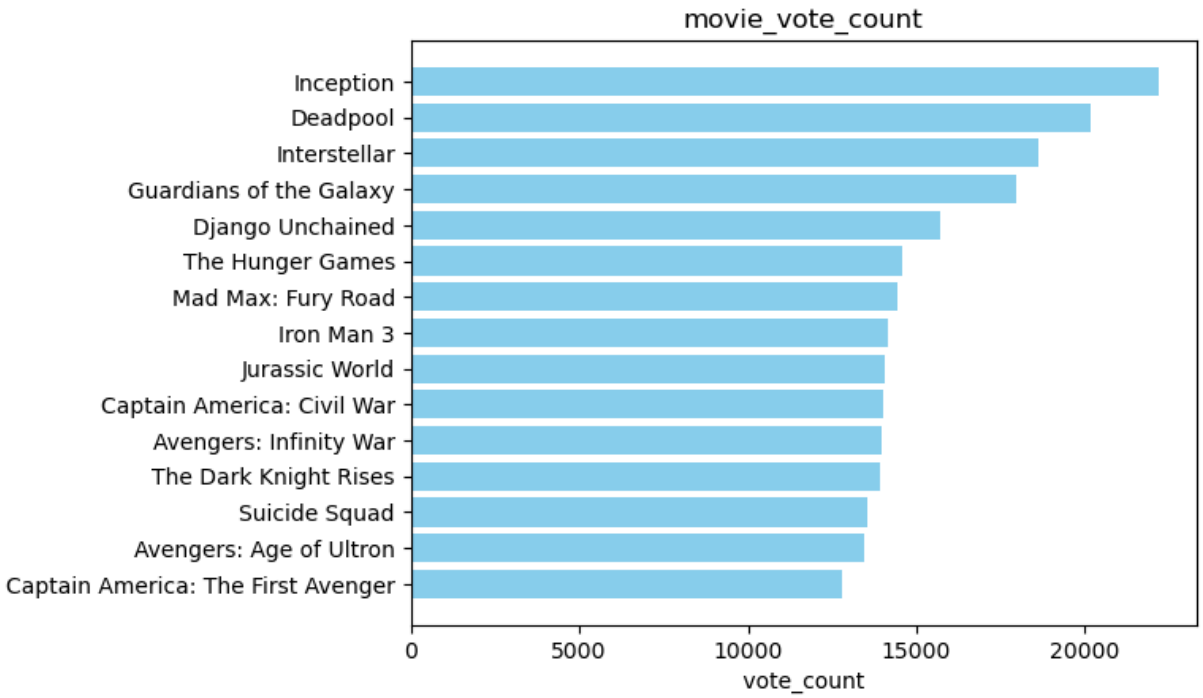


CONC: The year with the highest popularity is 2018

6.8.Identifying the movie vote counts

In [136…

```
#plotting bar grapgh on studio name with the highest vote_counts
top_movie_popularity =mg_data.sort_values(by='vote_count', ascending=False).head(15

# Plot
#plt.figure(figsize=(10, 8))
plt.barh(top_movie_popularity["title"], top_movie_popularity['vote_count'], color='
plt.xlabel(' vote_count ')
plt.title('movie_vote_count')
plt.gca().invert_yaxis()  # Highest grossing at the top

plt.show()
```



CONT:The movie title with the highest votecount is INCEPTION

6.9.

(iii)Movie budget

In [141…
```
movie_budget.head()
```

Out[141…

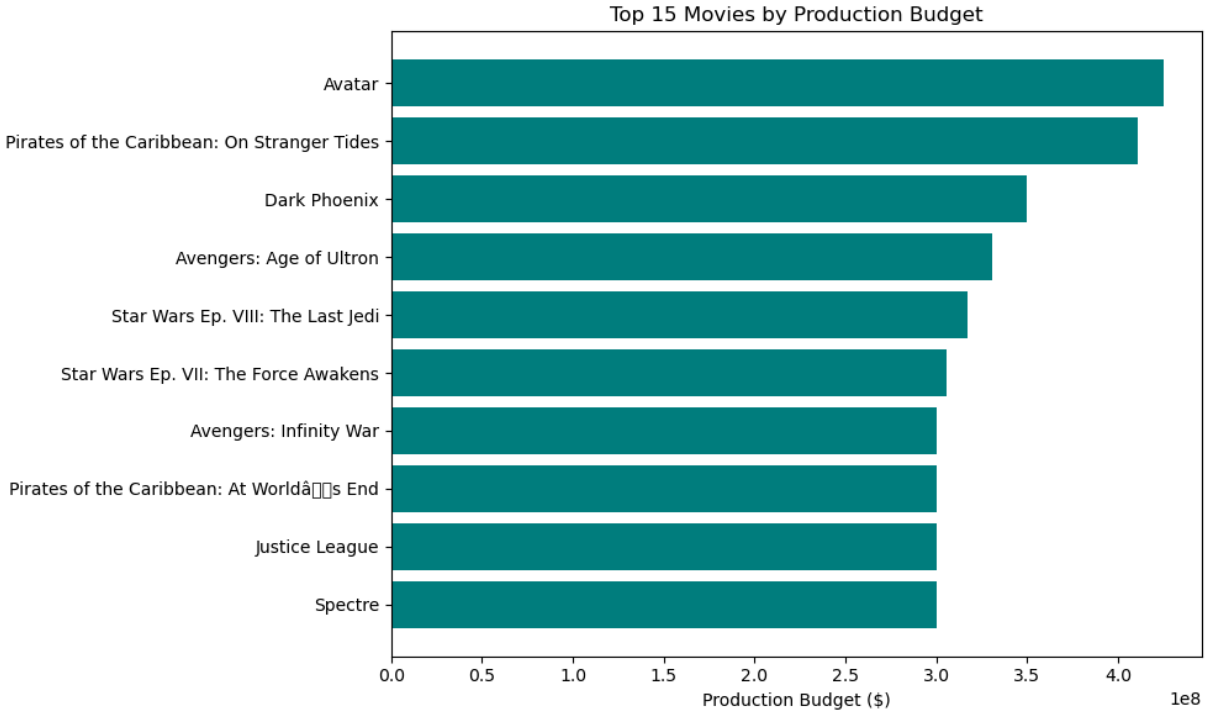
id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
4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

7.0.bar grapgh of movie with the highest production budget

In [150…
```
# Sort by budget and select top 15 for readability
top_budget_movies = movie_budget.sort_values(by='production_budget', ascending=Fals
```

```
# Plotting
plt.figure(figsize=(10, 6))
plt.barh(top_budget_movies['movie'], top_budget_movies['production_budget'], color=
plt.xlabel('Production Budget ($)')
plt.title('Top 15 Movies by Production Budget')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_3971/825145010.py:13: UserWarning: Glyph 128 (\x80) missing from curr
ent font.
  plt.tight_layout()
/tmp/ipykernel_3971/825145010.py:13: UserWarning: Glyph 153 (\x99) missing from curr
ent font.
  plt.tight_layout()
```

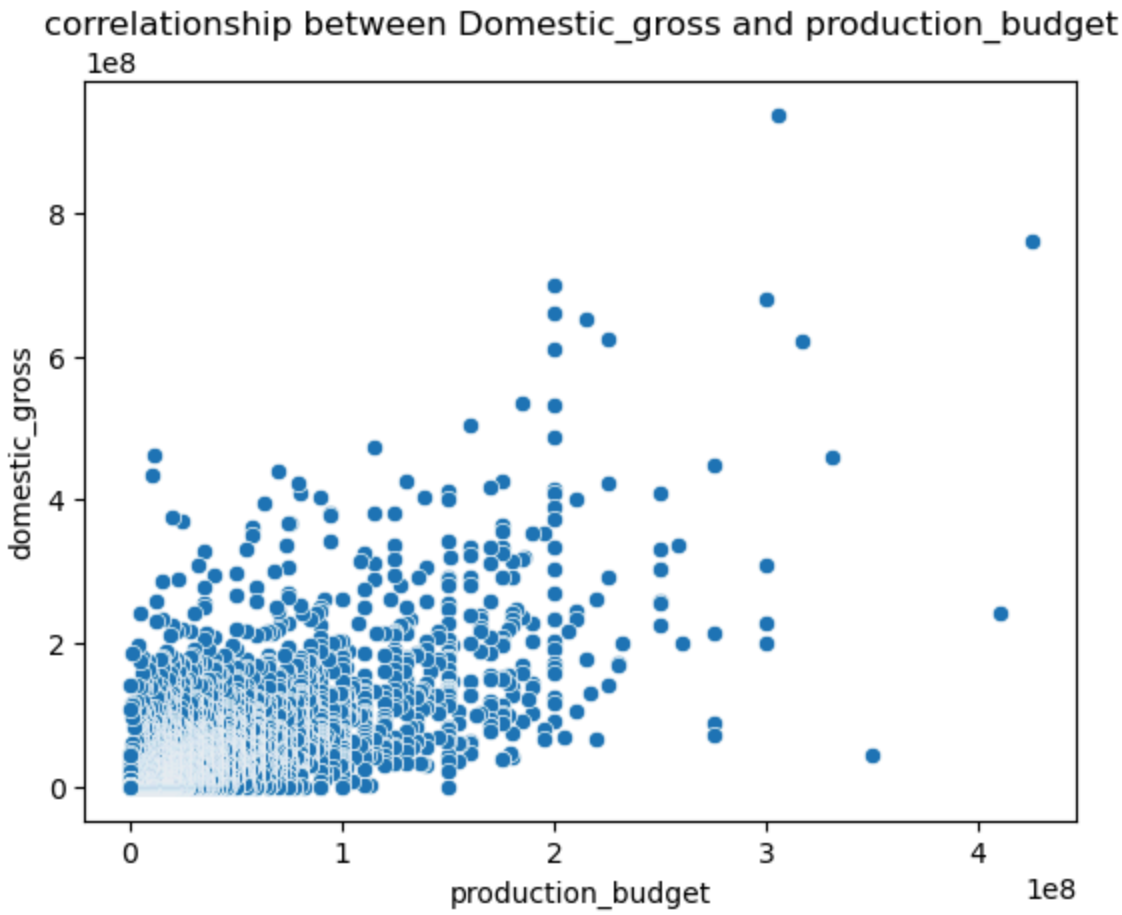


CONC: The movie with the highest production budget is **AVATAR**

In [ ]:

In [163…

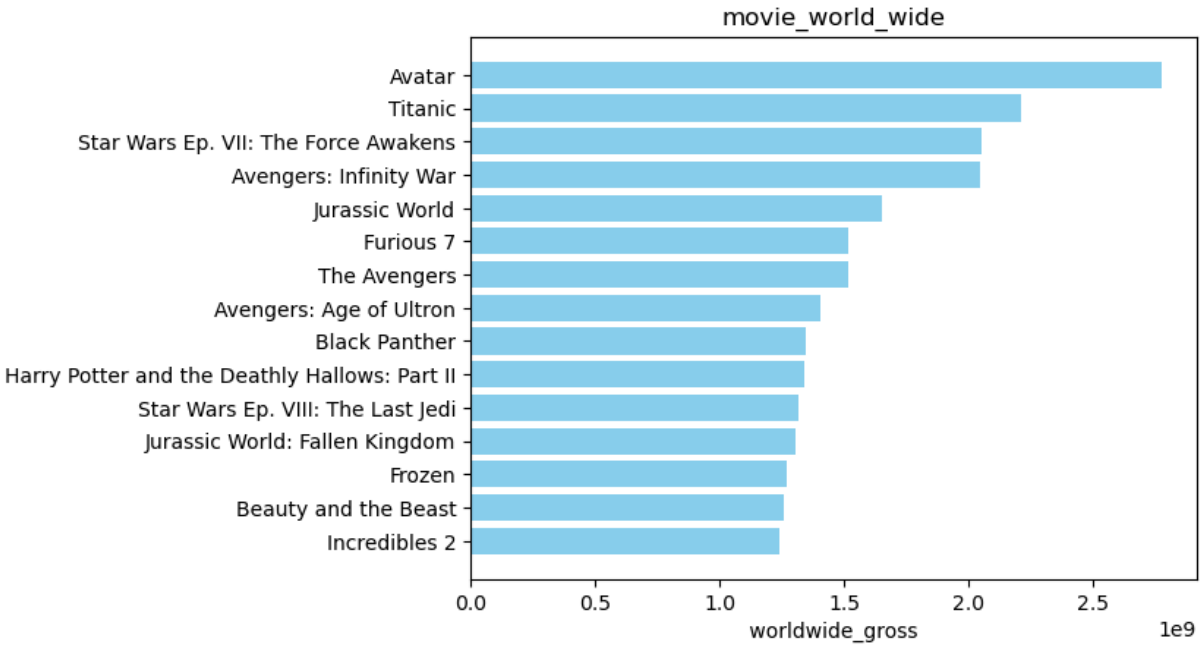
```
#finding out the relationship between domestic and the production_budget
sns.scatterplot(x='production_budget', y='domestic_gross', data=movie_budget)
plt.title("correlationship between Domestic_gross and production_budget")
plt.show()
```



CONC: There is a positive correletionship between the production_budget and the domestic_budget

In [166…

```
#plotting bar grapgh on movie title with the highest worldwide gross
top_movie_world =movie_budget.sort_values(by='worldwide_gross', ascending=False).he

# Plot
#plt.figure(figsize=(10, 8))
plt.barh(top_movie_world["movie"], top_movie_world['worldwide_gross'], color='skybl
plt.xlabel(' worldwide_gross ')
plt.title('movie_world_wide')
plt.gca().invert_yaxis()  # Highest grossing at the top

plt.show()
```



CONC: In the world wide gross rating, Avatar is the highly consumed

In [ ]:

CONCLUTION:

1. Financial Success is Driven by Genre & Talent

2.International Markets Offer Major Revenue Potential

3.Audience Engagement & Brand Building Are Critical

4.The most populr movie world wide is Avatar with the highest budget in production

5. The writers, composers and the directors affect the impact to the clients

6.BV is the highest producing Studio

7.There is a positive correlationship of the domestic and foreign gross