Introduction

Final Project Submission

- Student Name: Adam Marianacci
- Student Pace: Flex
- Scheduled project review date/time: 5/23/23 1:00pm EST
- Instructor Name: Morgan Jones

Business Understanding

This project analyzes what types of films Microsoft should create in their new movie studio using various movie data sets. Microsoft currently does not know anything about creating movies and they need help in deciding what films to create. Exploring movie data sets will tell what types of films are doing the best in the box office and this well help Microsoft decide on which creative direction they should go in to be successful.

Data Understanding

Each dataset used in this project contains thousands of entries. The datasets are from

- [Box Office Mojo (https://www.boxofficemojo.com/)](https://www.boxofficemojo.com/)
- [IMDB (https://www.imdb.com/)](https://www.imdb.com/)
- [Rotten Tomatoes (https://www.rottentomatoes.com/)](https://www.rottentomatoes.com/)
- [The Movie DB (https://www.themoviedb.org/)](https://www.themoviedb.org/)
- [The Numbers (https://www.the-numbers.com/)](https://www.the-numbers.com/)

They contain data such as movie titles/release date/domestic and worldwide gross/popularity and genre to name a few. The data used in this research is suitable for the project because it contains various information that shows what movies are doing or have done the best in the box office.

Data Preperation

I started by importing the appropriate libraries to start my data preperation.

In [1]:
```
import pandas as pd
import numpy as np
import sqlite3
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.float_format', lambda x: '%.0f' % x)

```

In [2]:
```
# Loading the 'movie budgets' file.
movie_budgets_df = pd.read_csv('data/tn.movie_budgets.csv.gz', inde
```

In [3]:
```
# Gathering information about the Data Frame.
movie_budgets_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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
```

There are no missing values from the above dataframe.

In [4]:
```
# Previewing the first 5 rows
movie_budgets_df.head()
```

Out[4]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross
id					
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

I needed to format to proper datetime format so that I could eventually filter movies by year and month if needed. Eventually will be looking to see if there are any connections between high grossing movies and when in the calendar year they were released.

In [5]:
```
# Converting the df to proper datetime format
movie_budgets_df['release_date'] = pd.to_datetime(movie_budgets_df[
```

I was running into an issue converting strings to integers for dollar values. I searched the following in google: `converting dollars from str to integers pandas` and found this [solution (https://stackoverflow.com/a/32464612)](https://stackoverflow.com/a/32464612).

In [6]:
```
# Getting rid of the '$' and commas in 'gross' and 'budget' columns
movie_budgets_df['worldwide_gross'] = movie_budgets_df['worldwide_g
    lambda x: int(x.replace('$','').replace(',','')))
movie_budgets_df['production_budget'] = movie_budgets_df['productio
    lambda x: int(x.replace('$','').replace(',','')))
movie_budgets_df['domestic_gross'] = movie_budgets_df['domestic_gro
    lambda x: int(x.replace('$','').replace(',','')))

```

In [7]:
```
# Looking at the highest grossing movies worldwide
sorted_worldwide_gross_movie_budgets_df = movie_budgets_df.sort_val
```

In [8]:
```
# Coverting release_date to proper dates
sorted_worldwide_gross_movie_budgets_df['release_date'] = pd.to_dat

```

In [9]:
```
sorted_worldwide_gross_movie_budgets_df
```

Out[9]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	2009-12-18	Avatar	425000000	760507625	2776345279
43	1997-12-19	Titanic	200000000	659363944	2208208395
6	2015-12-18	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220
7	2018-04-27	Avengers: Infinity War	300000000	678815482	2048134200
34	2015-06-12	Jurassic World	215000000	652270625	1648854864
...	...	...	...	...	...
75	2005-12-31	Insomnia Manica	500000	0	0
74	2012-07-17	Girls Gone Dead	500000	0	0
73	2012-04-03	Enter Nowhere	500000	0	0
72	2010-12-31	Drones	500000	0	0
69	2008-12-12	The Kings of Appletown	7000000	0	0

5782 rows × 5 columns

In [10]:
```
sorted_worldwide_gross_movie_budgets_df = sorted_worldwide_gross_mo
```

In [11]:
```
sorted_worldwide_gross_movie_budgets_df = sorted_worldwide_gross_mo
```

In [12]:
```
sorted_worldwide_gross_movie_budgets_df = sorted_worldwide_gross_mo
```

In [13]:
```
# Getting descriptive statistics on the sorted worldwide gross data
sorted_worldwide_gross_movie_budgets_df.describe()
```

Out[13]:

	production_budget	domestic_gross	worldwide_gross
count	5782	5782	5782
mean	31587757	41873327	91487461
std	41812077	68240597	174719969
min	1100	0	0
25%	5000000	1429534	4125415
50%	17000000	17225945	27984448
75%	40000000	52348662	97645836
max	425000000	936662225	2776345279

The mean for the production budget is around 31,000,000, the median is 17,000,000 which shows that there are certain films pulling the mean higher when in reality the majority of films are around the 17,000,000 mark.This number is in USD. This skew is also happening in the Worldwide gross column. The mean for worldwide gross is around 914,000,000 with the median only at 279,000,000.

In [14]:
```
sorted_worldwide_gross_movie_budgets_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5782 entries, 1 to 6
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   release_date       5782 non-null   datetime64[ns]
 1   movie              5782 non-null   object
 2   production_budget  5782 non-null   int64
 3   domestic_gross     5782 non-null   int64
 4   worldwide_gross    5782 non-null   int64
dtypes: datetime64[ns](1), int64(3), object(1)
memory usage: 271.0+ KB
```

In [15]:
```
# Loading the tmdb movies file.
tmdb_movies_df = pd.read_csv('data/tmdb.movies.csv.gz', index_col=0
```

In [16]:
```
1 tmdb_movies_df.head()
```

Out[16]:

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_ave
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	34	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	29	2010-03-26	How to Train Your Dragon	
2	[12, 28, 878]	10138	en	Iron Man 2	29	2010-05-07	Iron Man 2	
3	[16, 35, 10751]	862	en	Toy Story	28	1995-11-22	Toy Story	
4	[28, 878, 12]	27205	en	Inception	28	2010-07-16	Inception	

I imported the above dataframe because it had valuable information regarding release date and genre_ids which allowed me to draw comparisons between popularity,release date, and genres between films.

In [17]:
```
1 # Gathering information about the Data Frame.
2 tmdb_movies_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26517 entries, 0 to 26516
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   genre_ids          26517 non-null  object
 1   id                 26517 non-null  int64
 2   original_language  26517 non-null  object
 3   original_title     26517 non-null  object
 4   popularity         26517 non-null  float64
 5   release_date       26517 non-null  object
 6   title              26517 non-null  object
 7   vote_average       26517 non-null  float64
 8   vote_count         26517 non-null  int64
dtypes: float64(2), int64(2), object(5)
memory usage: 2.0+ MB
```

In [18]:
```
1 # Converting dates to actual dates.
2 tmdb_movies_df['release_date'] = pd.to_datetime(tmdb_movies_df['rel
```

This will eventually allow me to sort films and look at release dates more clearly.

In [19]:
```
# Previewing the first 5 rows of the Data Frame.
tmdb_movies_df.head()
```

Out[19]:

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_ave
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	34	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	29	2010-03-26	How to Train Your Dragon	
2	[12, 28, 878]	10138	en	Iron Man 2	29	2010-05-07	Iron Man 2	
3	[16, 35, 10751]	862	en	Toy Story	28	1995-11-22	Toy Story	
4	[28, 878, 12]	27205	en	Inception	28	2010-07-16	Inception	

I wanted to see which movies had the highest popularity rating and eventually wanted to look at which genres were associated with the films with the highest popularity ratings.

In [20]:
```
# Sorting movies by most popular.
sorted_popularity_tmdb_movies_df = tmdb_movies_df.sort_values('popu
```

In [21]:
```
import ast
```

I was running into an issue the list to an actual python numeric list. I searched the following in google: `converting lists to numeric in pandas` and found this [solution (https://stackoverflow.com/questions/55144642/python-pandas-convert-list-of-objects-to-a-list-of-integer)](https://stackoverflow.com/questions/55144642/python-pandas-convert-list-of-objects-to-a-list-of-integer). When converting to numeric value I can actually sort numerically which was not possible before.

In [22]:
```
# Convert the string representation of the list to an actual Python
sorted_popularity_tmdb_movies_df['genre_ids'] = sorted_popularity_t

# Convert the list to a numeric type
sorted_popularity_tmdb_movies_df['genre_ids'] = sorted_popularity_t
```

In [23]:
```
type(sorted_popularity_tmdb_movies_df['genre_ids'][0])
```

Out[23]: numpy.ndarray

In [24]:
```
1 genre_conversion_df = sorted_popularity_tmdb_movies_df[['genre_ids'
2 genre_conversion_df.head()
```

Out[24]:

	genre_ids	original_title	release_date
23811	[12, 28, 14]	Avengers: Infinity War	2018-04-27
11019	[28, 53]	John Wick	2014-10-24
23812	[28, 12, 16, 878, 35]	Spider-Man: Into the Spider-Verse	2018-12-14
11020	[28, 12, 14]	The Hobbit: The Battle of the Five Armies	2014-12-17
5179	[878, 28, 12]	The Avengers	2012-05-04

In [25]:
```
1 genre_conversion_df
```

Out[25]:

	genre_ids	original_title	release_date
23811	[12, 28, 14]	Avengers: Infinity War	2018-04-27
11019	[28, 53]	John Wick	2014-10-24
23812	[28, 12, 16, 878, 35]	Spider-Man: Into the Spider-Verse	2018-12-14
11020	[28, 12, 14]	The Hobbit: The Battle of the Five Armies	2014-12-17
5179	[878, 28, 12]	The Avengers	2012-05-04
...	...	...	...
13877	[10749]	Crème Caramel	2014-05-20
13878	[878]	Elegy	2014-09-10
13879	[35]	Jaguar	2014-09-21
13880	[]	Unleashed! A Dog Dancing Story	2014-02-13
26516	[53, 27]	The Church	2018-10-05

26517 rows × 3 columns

Creating a dataframe that displayed genre_ids to corresponding genre titles.

In [26]:
```
1 genre_conversion_df = genre_conversion_df.explode('genre_ids')
```

In [27]:

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

In [28]:

```
for key, value in genre_ids.items():
    print(key, value)
```

```
Action 28
Adventure 12
Animation 16
Comedy 35
Crime 80
Documentary 99
Drama 18
Family 10751
Fantasy 14
History 36
Horror 27
Music 10402
Mystery 9648
Romance 10749
ScienceFiction 878
TVMovie 10770
Thriller 53
War 10752
Western 37
```

In [29]:

```
for key, value in genre_ids.items():
    genre_conversion_df['genre_ids'] = genre_conversion_df['genre_i
```

In [30]:
```
1 genre_conversion_df
```

Out[30]:

	genre_ids	original_title	release_date
23811	Adventure	Avengers: Infinity War	2018-04-27
23811	Action	Avengers: Infinity War	2018-04-27
23811	Fantasy	Avengers: Infinity War	2018-04-27
11019	Action	John Wick	2014-10-24
11019	Thriller	John Wick	2014-10-24
...	...	...	...
13878	ScienceFiction	Elegy	2014-09-10
13879	Comedy	Jaguar	2014-09-21
13880	NaN	Unleashed! A Dog Dancing Story	2014-02-13
26516	Thriller	The Church	2018-10-05
26516	Horror	The Church	2018-10-05

47834 rows × 3 columns

In [31]:
```
# Looking at the top 30 most popular movies in the data set.
sorted_popularity_tmdb_movies_df.head(30)
```

Out[31]:

	genre_ids	id	original_language	original_title	popularity	release_date	title
23811	[12, 28, 14]	299536	en	Avengers: Infinity War	81	2018-04-27	Avengers: Infinity War
11019	[28, 53]	245891	en	John Wick	78	2014-10-24	John Wick
23812	[28, 12, 16, 878, 35]	324857	en	Spider-Man: Into the Spider-Verse	61	2018-12-14	Spider-Man: Into the Spider-Verse
11020	[28, 12, 14]	122917	en	The Hobbit: The Battle of the Five Armies	54	2014-12-17	The Hobbit: The Battle of the Five Armies
5179	[878, 28, 12]	24428	en	The Avengers	50	2012-05-04	The Avengers
11021	[28, 878, 12]	118340	en	Guardians of the Galaxy	50	2014-08-01	Guardians of the Galaxy
20617	[878, 28, 53]	335984	en	Blade Runner 2049	49	2017-10-06	Blade Runner 2049
23813	[878, 28, 53]	335984	en	Blade Runner 2049	49	2017-10-06	Blade Runner 2049
23814	[12]	338952	en	Fantastic Beasts: The Crimes of Grindelwald	49	2018-11-16	Fantastic Beasts: The Crimes of Grindelwald
23815	[10751, 16, 35, 14, 12]	404368	en	Ralph Breaks the Internet	48	2018-11-21	Ralph Breaks the Internet
20618	[28, 12, 878, 18]	315635	en	Spider-Man: Homecoming	47	2017-07-07	Spider-Man: Homecoming
20619	[53, 28, 80]	324552	en	John Wick: Chapter 2	45	2017-02-10	John Wick: Chapter 2
20620	[28, 18, 878]	263115	en	Logan	45	2017-03-03	Logan
23816	[28, 12, 878, 35]	363088	en	Ant-Man and the Wasp	45	2018-07-06	Ant-Man and the Wasp
14169	[28, 12, 878]	99861	en	Avengers: Age of Ultron	44	2015-05-01	Avengers: Age of Ultron
23817	[28, 12, 14, 878]	284054	en	Black Panther	44	2018-02-16	Black Panther
23818	[878, 28]	335983	en	Venom	44	2018-10-05	Venom
20621	[28, 12, 35, 14]	284053	en	Thor: Ragnarok	43	2017-11-03	Thor: Ragnarok
23819	[28, 12, 35, 14]	284053	en	Thor: Ragnarok	43	2017-11-03	Thor: Ragnarok

	genre_ids	id	original_language	original_title	popularity	release_date	title
23820	[28, 12, 878]	424783	en	Bumblebee	43	2018-12-21	Bumblebee
11022	[28, 12, 14, 878]	127585	en	X-Men: Days of Future Past	42	2014-05-23	X-Men: Days of Future Past
20622	[28, 12, 35, 878]	283995	en	Guardians of the Galaxy Vol. 2	40	2017-05-05	Guardians of the Galaxy Vol. 2
23821	[12, 14]	428078	en	Mortal Engines	40	2018-12-14	Mortal Engines
23822	[12, 28, 53]	375588	en	Robin Hood	40	2018-11-21	Robin Hood
17381	[28, 12, 878, 14]	246655	en	X-Men: Apocalypse	39	2016-05-27	X-Men: Apocalypse
17382	[12, 28, 878]	271110	en	Captain America: Civil War	39	2016-05-06	Captain America: Civil War
23823	[28, 35, 12]	383498	en	Deadpool 2	39	2018-05-10	Deadpool 2
23824	[28, 12, 14]	297802	en	Aquaman	38	2018-12-21	Aquaman
2468	[12, 14, 28]	10195	en	Thor	38	2011-05-06	Thor

In [32]:
```
genre_popularity_df = sorted_popularity_tmdb_movies_df[['genre_ids'
```

I had to look up the genre_ids on the Movie Database website https://www.themoviedb.org/talk/5daf6eb0ae36680011d7e6ee (https://www.themoviedb.org/talk/5daf6eb0ae36680011d7e6ee). To see which id's were associated with what genre.

In [33]:
```
# Loading the 'movie gross' file.
movie_gross_df = pd.read_csv('data/bom.movie_gross.csv.gz', index_c
```

In [34]:
```
# Gatherting information about the Data Frame.
movie_gross_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3387 entries, Toy Story 3 to An Actor Prepares
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   studio          3382 non-null   object
 1   domestic_gross  3359 non-null   float64
 2   foreign_gross   2037 non-null   object
 3   year            3387 non-null   int64
dtypes: float64(1), int64(1), object(2)
memory usage: 132.3+ KB
```

Noticing that there are a good amount of missing values from this dataframe.

In [35]:
```python
# Checking for how many missing values there are in each column.
movie_gross_df.isna().sum()
```

Out[35]:
```
studio               5
domestic_gross      28
foreign_gross     1350
year                 0
dtype: int64
```

In [36]:
```python
# Because it is a small amount I decided to drop the missing values
movie_gross_df.dropna(subset=['domestic_gross'], inplace=True)
```

In [37]:
```python
# Checking to see if the missing values were dropped
movie_gross_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3359 entries, Toy Story 3 to An Actor Prepares
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   studio          3356 non-null   object 
 1   domestic_gross  3359 non-null   float64
 2   foreign_gross   2009 non-null   object 
 3   year            3359 non-null   int64  
dtypes: float64(1), int64(1), object(2)
memory usage: 131.2+ KB
```

In [38]:
```python
# Previewing the first 5 rows in the Data Frame.
movie_gross_df.head()
```

Out[38]:

title	studio	domestic_gross	foreign_gross	year
Toy Story 3	BV	415000000	652000000	2010
Alice in Wonderland (2010)	BV	334200000	691300000	2010
Harry Potter and the Deathly Hallows Part 1	WB	296000000	664300000	2010
Inception	WB	292600000	535700000	2010
Shrek Forever After	P/DW	238700000	513900000	2010

In [39]:
```python
# Sorting movies by the highest domestic gross
sorted_movie_gross_domestic_df = movie_gross_df.sort_values('domest
```

In [40]:

```
# Viewing the top 30 results
sorted_movie_gross_domestic_df.head(30)
```

Out[40]:

title	studio	domestic_gross	foreign_gross	year
Star Wars: The Force Awakens	BV	936700000	1,131.6	2015
Black Panther	BV	700100000	646900000	2018
Avengers: Infinity War	BV	678800000	1,369.5	2018
Jurassic World	Uni.	652300000	1,019.4	2015
Marvel's The Avengers	BV	623400000	895500000	2012
Star Wars: The Last Jedi	BV	620200000	712400000	2017
Incredibles 2	BV	608600000	634200000	2018
Rogue One: A Star Wars Story	BV	532200000	523900000	2016
Beauty and the Beast (2017)	BV	504000000	759500000	2017
Finding Dory	BV	486300000	542300000	2016
Avengers: Age of Ultron	BV	459000000	946400000	2015
The Dark Knight Rises	WB	448100000	636800000	2012
The Hunger Games: Catching Fire	LGF	424700000	440300000	2013
Jurassic World: Fallen Kingdom	Uni.	417700000	891800000	2018
Toy Story 3	BV	415000000	652000000	2010
Wonder Woman	WB	412600000	409300000	2017
Iron Man 3	BV	409000000	805800000	2013
Captain America: Civil War	BV	408100000	745200000	2016
The Hunger Games	LGF	408000000	286400000	2012
Jumanji: Welcome to the Jungle	Sony	404500000	557600000	2017
Frozen	BV	400700000	875700000	2013
Guardians of the Galaxy Vol. 2	BV	389800000	473900000	2017
Harry Potter and the Deathly Hallows Part 2	WB	381000000	960500000	2011
The Secret Life of Pets	Uni.	368400000	507100000	2016
Despicable Me 2	Uni.	368100000	602700000	2013
The Jungle Book (2016)	BV	364000000	602500000	2016
Deadpool	Fox	363100000	420000000	2016
Inside Out	BV	356500000	501100000	2015
Furious 7	Uni.	353000000	1,163.0	2015
Transformers: Dark of the Moon	P/DW	352400000	771400000	2011

In [41]:
```
# Getting descriptive statistics on the sorted domestic movie gross
sorted_movie_gross_domestic_df.describe()
```

Out[41]:

	domestic_gross	year
count	3359	3359
mean	28745845	2014
std	66982498	2
min	100	2010
25%	120000	2012
50%	1400000	2014
75%	27900000	2016
max	936700000	2018

In [42]:
```
conn = sqlite3.connect('data/im.db')
cur = conn.cursor()
imdb_tables = pd.read_sql('SELECT * FROM sqlite_master WHERE type =
imdb_tables
```

Out[42]:

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

In [43]:
```
# read in the movie_basics table into a dataframe
movie_basics = pd.read_sql_query("SELECT * from movie_basics", conn

# read in the directors table into a dataframe
directors = pd.read_sql_query("SELECT * from directors", conn)

```

In [44]:
```
# deleting duplicate entries in the dataframe.
directors.drop_duplicates(subset=['movie_id'], keep='first', inplac
```

In [45]:
```
directors
```

Out[45]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0462036	nm1940585
2	tt0835418	nm0151540
4	tt0878654	nm0089502
7	tt0879859	nm2416460
...	...	...
291167	tt8999892	nm10122247
291169	tt8999974	nm10122357
291170	tt9001390	nm6711477
291171	tt9001494	nm10123242
291173	tt9004986	nm4993825

140417 rows × 2 columns

In [46]:
```
# Merge the movie basics and directors tables on the movie_id colum
movie_directors = pd.merge(movie_basics, directors, on='movie_id',

```

In [47]:
```
movie_directors
```

Out[47]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	nan	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80	Comedy,Drama,Fantasy
...	...	...	...	...	...	...
146139	tt9916538	Kuambil Lagi Hatiku	Kuambil Lagi Hatiku	2019	123	Drama
146140	tt9916622	Rodolpho Teóphilo - O Legado de um Pioneiro	Rodolpho Teóphilo - O Legado de um Pioneiro	2015	nan	Documentary
146141	tt9916706	Dankyavar Danka	Dankyavar Danka	2013	nan	Comedy
146142	tt9916730	6 Gunn	6 Gunn	2017	116	None
146143	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	nan	Documentary

146144 rows × 7 columns

In [48]:
```
persons = pd.read_sql_query("SELECT * from persons", conn)
```

In [49]:
```
df_imdb = pd.read_sql('SELECT * FROM persons', conn)
```

In [50]:

```
# Merge the directors table with the persons table
directors_persons = pd.merge(directors, persons, on='person_id', ho
 
# Merge the resulting dataframe with the movie basics table
movie_directors_df = pd.merge(movie_directors, directors_persons, o
 
movie_directors_df
```

Out[50]:

	movie_id_x	primary_title	original_title	start_year	runtime_minutes	genre
0	tt0063540	Sunghursh	Sunghursh	2013	175	Action,Crime,Dran
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114	Biography,Dran
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122	Dran
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	nan	Comedy,Dran
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80	Comedy,Drama,Fanta
...	...	...	...	...	...	
380129	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	nan	Documenta
380130	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	nan	Documenta
380131	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	nan	Documenta
380132	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	nan	Documenta
380133	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	nan	Documenta

380134 rows × 12 columns

In [51]:
```
1 dir_name_title = movie_directors_df[['movie_id_x', 'primary_title',
2 dir_name_title
```

Out[51]:

	movie_id_x	primary_title	person_id	primary_name
0	tt0063540	Sunghursh	nm0712540	Harnam Singh Rawail
1	tt0066787	One Day Before the Rainy Season	nm0002411	Mani Kaul
2	tt0069049	The Other Side of the Wind	nm0000080	Orson Welles
3	tt0069204	Sabse Bada Sukh	nm0611531	Hrishikesh Mukherjee
4	tt0100275	The Wandering Soap Opera	nm0765384	Valeria Sarmiento
...	...	...	...	...
380129	tt9916754	Chico Albuquerque - Revelações	nm9272490	Angela Gurgel
380130	tt9916754	Chico Albuquerque - Revelações	nm9272490	Angela Gurgel
380131	tt9916754	Chico Albuquerque - Revelações	nm9272490	Angela Gurgel
380132	tt9916754	Chico Albuquerque - Revelações	nm9272490	Angela Gurgel
380133	tt9916754	Chico Albuquerque - Revelações	nm9272490	Angela Gurgel

380134 rows × 4 columns

In [52]:
```
1 budget_info = movie_budgets_df[['movie', 'production_budget', 'dome
2 budget_info
```

Out[52]:

	movie	production_budget	domestic_gross	worldwide_gross
id				
1	Avatar	425000000	760507625	2776345279
2	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875
3	Dark Phoenix	350000000	42762350	149762350
4	Avengers: Age of Ultron	330600000	459005868	1403013963
5	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747
...	...	...	...	...
78	Red 11	7000	0	0
79	Following	6000	48482	240495
80	Return to the Land of Wonders	5000	1338	1338
81	A Plague So Pleasant	1400	0	0
82	My Date With Drew	1100	181041	181041

5782 rows × 4 columns

In [53]:
```
1 budget_info.rename(columns={"movie": "primary_title"}, inplace=True
```

In [54]:
```
1 budget_info
```

Out[54]:

id	primary_title	production_budget	domestic_gross	worldwide_gross
1	Avatar	425000000	760507625	2776345279
2	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875
3	Dark Phoenix	350000000	42762350	149762350
4	Avengers: Age of Ultron	330600000	459005868	1403013963
5	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747
...	...	...	...	...
78	Red 11	7000	0	0
79	Following	6000	48482	240495
80	Return to the Land of Wonders	5000	1338	1338
81	A Plague So Pleasant	1400	0	0
82	My Date With Drew	1100	181041	181041

5782 rows × 4 columns

In [55]:
```
1 gross_directors_df = pd.merge(dir_name_title, budget_info, on='prim
```

I wanted to calculate "profit" and wanted to make a new column to get a sense for what films were making the most money.I did this by taking gross and subtracting production budget in the dataframe.

In [56]:

```
1 gross_directors_df['profit'] = gross_directors_df['worldwide_gross'
2 gross_directors_df
```

Out[56]:

	movie_id_x	primary_title	person_id	primary_name	production_budget	domestic_gross
0	tt0249516	Foodfight!	nm0440415	Lawrence Kasanoff	45000000	0
1	tt0249516	Foodfight!	nm0440415	Lawrence Kasanoff	45000000	0
2	tt0293429	Mortal Kombat	nm2585406	Simon McQuoid	20000000	70433227
3	tt0326592	The Overnight	nm1208371	Jed I. Goodman	200000	1109808
4	tt3844362	The Overnight	nm2674307	Patrick Brice	200000	1109808
...	...	...	...	...	...	...
10924	tt9844102	Ray	nm3386933	Riingo Banerjee	40000000	75305995
10925	tt9844102	Ray	nm3386933	Riingo Banerjee	40000000	75305995
10926	tt9844102	Ray	nm3386933	Riingo Banerjee	40000000	75305995
10927	tt9893078	Sublime	nm0349702	Bill Guttentag	1800000	0
10928	tt9893078	Sublime	nm0349702	Bill Guttentag	1800000	0

10929 rows × 8 columns

In [57]:

```
1 sorted_gross_directors_df = gross_directors_df.sort_values(by='worl
```

In [58]:
```
1 sorted_gross_directors_df.head(30)
```

Out[58]:

	movie_id_x	primary_title	person_id	primary_name	production_budget	domestic_gross
4972	tt1775309	Avatar	nm3786927	Atsushi Wada	425000000	760507625
4973	tt1775309	Avatar	nm3786927	Atsushi Wada	425000000	760507625
4974	tt1775309	Avatar	nm3786927	Atsushi Wada	425000000	760507625
7887	tt2495766	Titanic	nm4430776	Pete Meads	200000000	659363944
7888	tt8852130	Titanic	nm10047650	Ravi Punj	200000000	659363944
9621	tt4154756	Avengers: Infinity War	nm0751577	Anthony Russo	300000000	678815482
9622	tt4154756	Avengers: Infinity War	nm0751577	Anthony Russo	300000000	678815482
9625	tt4154756	Avengers: Infinity War	nm0751577	Anthony Russo	300000000	678815482
9623	tt4154756	Avengers: Infinity War	nm0751577	Anthony Russo	300000000	678815482
9624	tt4154756	Avengers: Infinity War	nm0751577	Anthony Russo	300000000	678815482
31	tt0369610	Jurassic World	nm1119880	Colin Trevorrow	215000000	652270625
30	tt0369610	Jurassic World	nm1119880	Colin Trevorrow	215000000	652270625
28	tt0369610	Jurassic World	nm1119880	Colin Trevorrow	215000000	652270625
29	tt0369610	Jurassic World	nm1119880	Colin Trevorrow	215000000	652270625
8347	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020
8353	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020
8352	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020
8350	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020
8349	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020
8348	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020
8351	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020
395	tt0848228	The Avengers	nm0923736	Joss Whedon	225000000	623279547
393	tt0848228	The Avengers	nm0923736	Joss Whedon	225000000	623279547
391	tt0848228	The Avengers	nm0923736	Joss Whedon	225000000	623279547
7712	tt2395427	Avengers: Age of Ultron	nm0923736	Joss Whedon	330600000	459005868

	movie_id_x	primary_title	person_id	primary_name	production_budget	domestic_gross
7713	tt2395427	Avengers: Age of Ultron	nm0923736	Joss Whedon	330600000	459005868
7711	tt2395427	Avengers: Age of Ultron	nm0923736	Joss Whedon	330600000	459005868
5227	tt1825683	Black Panther	nm3363032	Ryan Coogler	200000000	700059566
5226	tt1825683	Black Panther	nm3363032	Ryan Coogler	200000000	700059566

In [59]:

```
# Convert the data type of 'id' column in the second dataframe
sorted_popularity_tmdb_movies_df['id'] = sorted_popularity_tmdb_mov

# Perform the merge operation
genre_directors_df = sorted_gross_directors_df.merge(sorted_popular

# Drop the duplicate 'id' column
genre_directors_df.drop('id', axis=1, inplace=True)
```

In [60]:

```
# Perform a merge operation
genre_directors_df = sorted_gross_directors_df.merge(sorted_popular

# Drop the duplicate 'id' column
genre_directors_df.drop('id', axis=1, inplace=True)

```

In [61]:
```
genre_directors_df.head(50)
```

Out[61]:

	movie_id_x	primary_title	person_id	primary_name	production_budget	domestic_gross	w
0	tt1775309	Avatar	nm3786927	Atsushi Wada	425000000	760507625	
1	tt1775309	Avatar	nm3786927	Atsushi Wada	425000000	760507625	
2	tt1775309	Avatar	nm3786927	Atsushi Wada	425000000	760507625	
3	tt2495766	Titanic	nm4430776	Pete Meads	200000000	659363944	
4	tt8852130	Titanic	nm10047650	Ravi Punj	200000000	659363944	
5	tt4154756	Avengers: Infinity War	nm0751577	Anthony Russo	300000000	678815482	
6	tt4154756	Avengers: Infinity War	nm0751577	Anthony Russo	300000000	678815482	
7	tt4154756	Avengers: Infinity War	nm0751577	Anthony Russo	300000000	678815482	
8	tt4154756	Avengers: Infinity War	nm0751577	Anthony Russo	300000000	678815482	
9	tt4154756	Avengers: Infinity War	nm0751577	Anthony Russo	300000000	678815482	
10	tt0369610	Jurassic World	nm1119880	Colin Trevorrow	215000000	652270625	
11	tt0369610	Jurassic World	nm1119880	Colin Trevorrow	215000000	652270625	
12	tt0369610	Jurassic World	nm1119880	Colin Trevorrow	215000000	652270625	
13	tt0369610	Jurassic World	nm1119880	Colin Trevorrow	215000000	652270625	
14	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020	
15	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020	
16	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020	
17	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020	
18	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020	
19	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020	
20	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020	
21	tt0848228	The Avengers	nm0923736	Joss Whedon	225000000	623279547	
22	tt0848228	The Avengers	nm0923736	Joss Whedon	225000000	623279547	
23	tt0848228	The Avengers	nm0923736	Joss Whedon	225000000	623279547	
24	tt2395427	Avengers: Age of Ultron	nm0923736	Joss Whedon	330600000	459005868	

	movie_id_x	primary_title	person_id	primary_name	production_budget	domestic_gross	v
25	tt2395427	Avengers: Age of Ultron	nm0923736	Joss Whedon	330600000	459005868	
26	tt2395427	Avengers: Age of Ultron	nm0923736	Joss Whedon	330600000	459005868	
27	tt1825683	Black Panther	nm3363032	Ryan Coogler	200000000	700059566	
28	tt1825683	Black Panther	nm3363032	Ryan Coogler	200000000	700059566	
29	tt1825683	Black Panther	nm3363032	Ryan Coogler	200000000	700059566	
30	tt4881806	Jurassic World: Fallen Kingdom	nm1291105	J.A. Bayona	170000000	417719760	
31	tt4881806	Jurassic World: Fallen Kingdom	nm1291105	J.A. Bayona	170000000	417719760	
32	tt4881806	Jurassic World: Fallen Kingdom	nm1291105	J.A. Bayona	170000000	417719760	
33	tt1323045	Frozen	nm1697112	Adam Green	150000000	400738009	
34	tt1323045	Frozen	nm1697112	Adam Green	150000000	400738009	
35	tt1323045	Frozen	nm1697112	Adam Green	150000000	400738009	
36	tt1323045	Frozen	nm1697112	Adam Green	150000000	400738009	
37	tt1323045	Frozen	nm1697112	Adam Green	150000000	400738009	
38	tt1611845	Frozen	nm0477213	Chi-kin Kwok	150000000	400738009	
39	tt1611845	Frozen	nm0477213	Chi-kin Kwok	150000000	400738009	
40	tt1611845	Frozen	nm0477213	Chi-kin Kwok	150000000	400738009	
41	tt2294629	Frozen	nm0118333	Chris Buck	150000000	400738009	
42	tt2294629	Frozen	nm0118333	Chris Buck	150000000	400738009	
43	tt9173998	Beauty and the Beast	nm10211048	Zane Burden	160000000	504014165	
44	tt2771200	Beauty and the Beast	nm0174374	Bill Condon	160000000	504014165	
45	tt2771200	Beauty and the Beast	nm0174374	Bill Condon	160000000	504014165	
46	tt2771200	Beauty and the Beast	nm0174374	Bill Condon	160000000	504014165	
47	tt2316801	Beauty and the Beast	nm0304521	Christophe Gans	160000000	504014165	
48	tt2771200	Beauty and the Beast	nm0174374	Bill Condon	160000000	504014165	

I wanted to check if there were any duplicate entries in the dataframe.

In [62]:
```
sorted_gross_directors_df.info
```

Out[62]:
```
<bound method DataFrame.info of       movie_id_x           primary_t
itle   person_id    primary_name  \
4972   tt1775309                    Avatar   nm3786927    Atsushi Wada
4973   tt1775309                    Avatar   nm3786927    Atsushi Wada
4974   tt1775309                    Avatar   nm3786927    Atsushi Wada
7887   tt2495766                   Titanic   nm4430776      Pete Meads
7888   tt8852130                   Titanic  nm10047650       Ravi Punj
...          ...                       ...         ...             ...
5340   tt1836212  All Superheroes Must Die   nm1875808     Jason Trost
5339   tt1836212  All Superheroes Must Die   nm1875808     Jason Trost
5338   tt1836212  All Superheroes Must Die   nm1875808     Jason Trost
5258   tt1827354                  Airborne   nm2403079   Dominic Burns
10928  tt9893078                   Sublime   nm0349702  Bill Guttentag

       production_budget  domestic_gross  worldwide_gross      profit
4972           425000000       760507625       2776345279  2351345279
4973           425000000       760507625       2776345279  2351345279
4974           425000000       760507625       2776345279  2351345279
7887           200000000       659363944       2208208395  2008208395
7888           200000000       659363944       2208208395  2008208395
...                  ...             ...              ...         ...
5340               20000               0                0      -20000
5339               20000               0                0      -20000
5338               20000               0                0      -20000
5258             1200000               0                0    -1200000
10928            1800000               0                0    -1800000

[10929 rows x 8 columns]>
```

In [63]:
```
# I wanted to drop duplicates in the below dataframe.
sorted_gross_directors_df.drop_duplicates(subset=['movie_id_x', 'pe
```

In [64]:
```
1 sorted_gross_directors_df.head(60)
```

Out[64]:

	movie_id_x	primary_title	person_id	primary_name	production_budget	domestic_gross
4972	tt1775309	Avatar	nm3786927	Atsushi Wada	425000000	760507625
7887	tt2495766	Titanic	nm4430776	Pete Meads	200000000	659363944
7888	tt8852130	Titanic	nm10047650	Ravi Punj	200000000	659363944
9621	tt4154756	Avengers: Infinity War	nm0751577	Anthony Russo	300000000	678815482
31	tt0369610	Jurassic World	nm1119880	Colin Trevorrow	215000000	652270625
8347	tt2820852	Furious 7	nm1490123	James Wan	190000000	353007020
395	tt0848228	The Avengers	nm0923736	Joss Whedon	225000000	623279547
7712	tt2395427	Avengers: Age of Ultron	nm0923736	Joss Whedon	330600000	459005868
5227	tt1825683	Black Panther	nm3363032	Ryan Coogler	200000000	700059560
10014	tt4881806	Jurassic World: Fallen Kingdom	nm1291105	J.A. Bayona	170000000	417719760
2425	tt1323045	Frozen	nm1697112	Adam Green	150000000	400738009
2427	tt1611845	Frozen	nm0477213	Chi-kin Kwok	150000000	400738009
2430	tt2294629	Frozen	nm0118333	Chris Buck	150000000	400738009
7472	tt9173998	Beauty and the Beast	nm10211048	Zane Burden	160000000	504014165
7470	tt2771200	Beauty and the Beast	nm0174374	Bill Condon	160000000	504014165
7458	tt2316801	Beauty and the Beast	nm0304521	Christophe Gans	160000000	504014165
9169	tt3606756	Incredibles 2	nm0083348	Brad Bird	200000000	608581744
9889	tt4630562	The Fate of the Furious	nm0336620	F. Gary Gray	250000000	225764765
2325	tt1300854	Iron Man 3	nm0000948	Shane Black	200000000	408992272
7357	tt2293640	Minions	nm1853544	Pierre Coffin	74000000	336045770
3157	tt1477834	Aquaman	nm1490123	James Wan	160000000	335061807
9021	tt3498820	Captain America: Civil War	nm0751577	Anthony Russo	250000000	408084349
2737	tt1399103	Transformers: Dark of the Moon	nm0000881	Michael Bay	195000000	352390543
9620	tt4154664	Captain Marvel	nm0281396	Ryan Fleck	175000000	426525952

	movie_id_x	primary_title	person_id	primary_name	production_budget	domestic_gros
1568	tt1074638	Skyfall	nm0005222	Sam Mendes	200000000	30436027
6632	tt2109248	Transformers: Age of Extinction	nm0000881	Michael Bay	210000000	24543907
2509	tt1345836	The Dark Knight Rises	nm0634240	Christopher Nolan	275000000	44813909
70	tt0435761	Toy Story 3	nm0881279	Lee Unkrich	200000000	41500488
9294	tt3748528	Rogue One: A Star Wars Story	nm2284484	Gareth Edwards	200000000	53217732
2320	tt1298650	Pirates of the Caribbean: On Stranger Tides	nm0551128	Rob Marshall	410600000	24106387
8988	tt3469046	Despicable Me 3	nm0049633	Kyle Balda	75000000	26462430
860	tt2049386	Alice in Wonderland	nm0288188	James Fotopoulos	200000000	33419111
848	tt1926979	Alice in Wonderland	nm2483359	Giuseppe Malpasso	200000000	33419111
844	tt1014759	Alice in Wonderland	nm0000318	Tim Burton	200000000	33419111
7323	tt2277860	Finding Dory	nm0004056	Andrew Stanton	200000000	48629556
8475	tt2948356	Zootopia	nm1158544	Jared Bush	150000000	34126824
443	tt0903624	The Hobbit: An Unexpected Journey	nm0001392	Peter Jackson	250000000	30300356
10465	tt6105098	The Lion King	nm0269463	Jon Favreau	79300000	42178528
4502	tt1690953	Despicable Me 2	nm1853544	Pierre Coffin	76000000	36806538
7341	tt2283362	Jumanji: Welcome to the Jungle	nm0440458	Jake Kasdan	90000000	40450891
7112	tt3040964	The Jungle Book	nm0269463	Jon Favreau	175000000	36400112
7100	tt2226178	The Jungle Book	nm0266255	Jun Falkenstein	175000000	36400112
1798	tt1170358	The Hobbit: The Desolation of Smaug	nm0001392	Peter Jackson	250000000	25836685

	movie_id_x	primary_title	person_id	primary_name	production_budget	domestic_gros
7425	tt2310332	The Hobbit: The Battle of the Five Armies	nm0001392	Peter Jackson	250000000	25511978
4758	tt1727824	Bohemian Rhapsody	nm0001741	Bryan Singer	55000000	21630333
8237	tt2709768	The Secret Life of Pets	nm0719208	Chris Renaud	75000000	36838433
7224	tt2250912	Spider-Man: Homecoming	nm1218281	Jon Watts	175000000	33420114
4409	tt1667889	Ice Age: Continental Drift	nm0862211	Mike Thurmeier	95000000	16132184
7637	tt2379713	Spectre	nm0005222	Sam Mendes	300000000	20007417
8511	tt2975590	Batman v Superman: Dawn of Justice	nm0811583	Zack Snyder	250000000	33036019
5897	tt1951264	The Hunger Games: Catching Fire	nm1349376	Francis Lawrence	130000000	42466804
4190	tt8269544	Inside Out	nm3970750	Russell Davidson	175000000	35646171
4186	tt2096673	Inside Out	nm0230032	Pete Docter	175000000	35646171
4182	tt1865538	Inside Out	nm4341391	Vaggelis Rigas	175000000	35646171
4179	tt1640486	Inside Out	nm0541703	Artie Mandelberg	175000000	35646171
4183	tt2064820	Inside Out	nm3034358	Nasir Rahim	175000000	35646171
4189	tt6419446	Inside Out	NaN	NaN	175000000	35646171

In [65]:
```
1 directors_persons
```

Out[65]:

	movie_id	person_id	primary_name	birth_year	death_year	primary_professic
0	tt0285252	nm0899854	Tony Vitale	1964	nan	producer,director,writ
1	tt0462036	nm1940585	Bill Haley	nan	nan	director,writer,produc
2	tt0835418	nm0151540	Jay Chandrasekhar	1968	nan	director,actor,writ
3	tt0859635	nm0151540	Jay Chandrasekhar	1968	nan	director,actor,writ
4	tt0878654	nm0089502	Albert Pyun	1954	nan	director,writer,produc
...	...	...	...	...	...	
140411	tt8998302	nm10121510	Daryl Boman	nan	nan	producer,director,writ
140412	tt8999892	nm10122247	C. Damon Adcock	nan	nan	Nor
140413	tt8999974	nm10122357	Daysi Burbano	nan	nan	director,writer,cinematograph
140414	tt9001390	nm6711477	Bernard Lessa	nan	nan	director,writer,cinematograph
140415	tt9001494	nm10123242	Tate Nova	nan	nan	director,produc

140416 rows × 6 columns

In [66]:
```
1 # Calling movie_directors df
2 len(movie_directors['primary_title'].unique())
```

Out[66]: 136071

In [67]:
```
1 movie_directors
```

Out[67]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	nan	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80	Comedy,Drama,Fantasy
...	...	...	...	...	...	...
146139	tt9916538	Kuambil Lagi Hatiku	Kuambil Lagi Hatiku	2019	123	Drama
146140	tt9916622	Rodolpho Teóphilo - O Legado de um Pioneiro	Rodolpho Teóphilo - O Legado de um Pioneiro	2015	nan	Documentary
146141	tt9916706	Dankyavar Danka	Dankyavar Danka	2013	nan	Comedy
146142	tt9916730	6 Gunn	6 Gunn	2017	116	None
146143	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	nan	Documentary

146144 rows × 7 columns

In [68]:
```
movie_basics = pd.read_sql_query("SELECT * from movie_basics", conn

# read in the directors table into a dataframe
directors = pd.read_sql_query("SELECT * from directors", conn)

# join the two dataframes on the movie_id column
df = pd.merge(movie_basics, directors, on="movie_id")

# close the database connection
conn.close()

# view the resulting dataframe
print(df.head())
```

```
    movie_id                    primary_title   original_title  start_
year  \
0  tt0063540                        Sunghursh        Sunghursh
2013
1  tt0063540                        Sunghursh        Sunghursh
2013
2  tt0063540                        Sunghursh        Sunghursh
2013
3  tt0063540                        Sunghursh        Sunghursh
2013
4  tt0066787  One Day Before the Rainy Season  Ashad Ka Ek Din
2019

   runtime_minutes              genres  person_id
0              175  Action,Crime,Drama  nm0712540
1              175  Action,Crime,Drama  nm0712540
2              175  Action,Crime,Drama  nm0712540
3              175  Action,Crime,Drama  nm0712540
4              114     Biography,Drama  nm0002411
```

In [69]:

```python
# Establish a connection to the database
conn = sqlite3.connect('data/im.db')

# Execute the SQL query to join the tables and convert the result s
df_imdb = pd.read_sql_query('SELECT * FROM directors JOIN persons O

# Preview the first 10 rows of the DataFrame
print(df_imdb.head(10))

# Close the connection
conn.close()
```

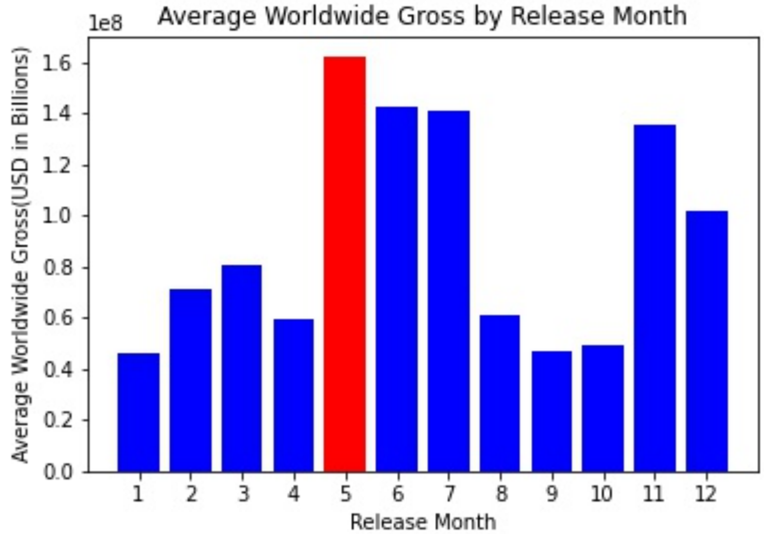
```
    movie_id  person_id  person_id       primary_name  birth_year  dea
th_year  \
0  tt0285252  nm0899854  nm0899854        Tony Vitale        1964
nan
1  tt0462036  nm1940585  nm1940585         Bill Haley         nan
nan
2  tt0835418  nm0151540  nm0151540  Jay Chandrasekhar        1968
nan
3  tt0835418  nm0151540  nm0151540  Jay Chandrasekhar        1968
nan
4  tt0878654  nm0089502  nm0089502        Albert Pyun        1954
nan
5  tt0878654  nm2291498  nm2291498          Joe Baile         nan
nan
6  tt0878654  nm2292011  nm2292011       Howie Askins         nan
nan
7  tt0879859  nm2416460  nm2416460    Eric Manchester         nan
nan
8  tt0996958  nm2286991  nm2286991      Tara Cardinal        1978
nan
9  tt0996958  nm2286991  nm2286991      Tara Cardinal        1978
nan

                    primary_profession
0            producer,director,writer
1            director,writer,producer
2               director,actor,writer
3               director,actor,writer
4            director,writer,producer
5  producer,director,camera_department
6       editor,director,cinematographer
7                     director,writer
8             actress,writer,producer
9             actress,writer,producer
```

In [70]:
```
# Establish a connection to the database
conn = sqlite3.connect('data/im.db')

# Execute the SQL query to join the tables and convert the result s
df_imdb = pd.read_sql_query('SELECT * FROM directors JOIN movie_bas

# Preview the first 10 rows of the DataFrame
print(df_imdb.head(10))

# Close the connection
conn.close()

```

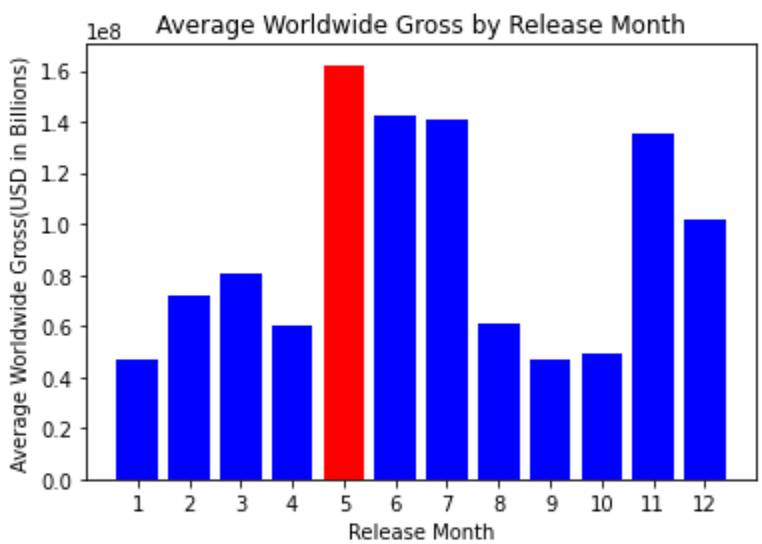
```
    movie_id  person_id   movie_id                    primary_title  \
0  tt0285252  nm0899854  tt0285252                   Life's a Beach
1  tt0462036  nm1940585  tt0462036  Steve Phoenix: The Untold Story
2  tt0835418  nm0151540  tt0835418                   The Babymakers
3  tt0835418  nm0151540  tt0835418                   The Babymakers
4  tt0878654  nm0089502  tt0878654                       Bulletface
5  tt0878654  nm2291498  tt0878654                       Bulletface
6  tt0878654  nm2292011  tt0878654                       Bulletface
7  tt0879859  nm2416460  tt0879859                             Torn
8  tt0996958  nm2286991  tt0996958        Legend of the Red Reaper
9  tt0996958  nm2286991  tt0996958        Legend of the Red Reaper

                    original_title  start_year  runtime_minutes  \
0                   Life's a Beach        2012              100
1  Steve Phoenix: The Untold Story        2012              110
2                   The Babymakers        2012               95
3                   The Babymakers        2012               95
4                       Bulletface        2010               82
5                       Bulletface        2010               82
6                       Bulletface        2010               82
7                             Torn        2010              nan
8         Legend of the Red Reaper        2013               99
9         Legend of the Red Reaper        2013               99

                    genres
0                   Comedy
1                    Drama
2                   Comedy
3                   Comedy
4                 Thriller
5                 Thriller
6                 Thriller
7                 Thriller
8  Action,Adventure,Fantasy
9  Action,Adventure,Fantasy
```

Exploratory Data Analysis

In [71]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
```



In [72]:
```
# Extract the month from the release date
sorted_worldwide_gross_movie_budgets_df['release_month'] = pd.Datet

# Group the data by release month and calculate the average worldwi
grouped = sorted_worldwide_gross_movie_budgets_df.groupby('release_

# Define a list of colors for each bin
colors = ['blue', 'blue', 'blue', 'blue', 'red', 'blue', 'blue', 'b


# Create a histogram of the average worldwide gross by release mont
plt.bar(grouped.index, grouped, color=colors)
plt.xlabel('Release Month')
plt.ylabel('Average Worldwide Gross(USD in Billions)')
plt.title('Average Worldwide Gross by Release Month')
# Set the tick locations and labels for the x-axis
plt.xticks(range(1, 13), range(1, 13))

plt.savefig('images/ReleaseDate.jpg')
plt.show()
```

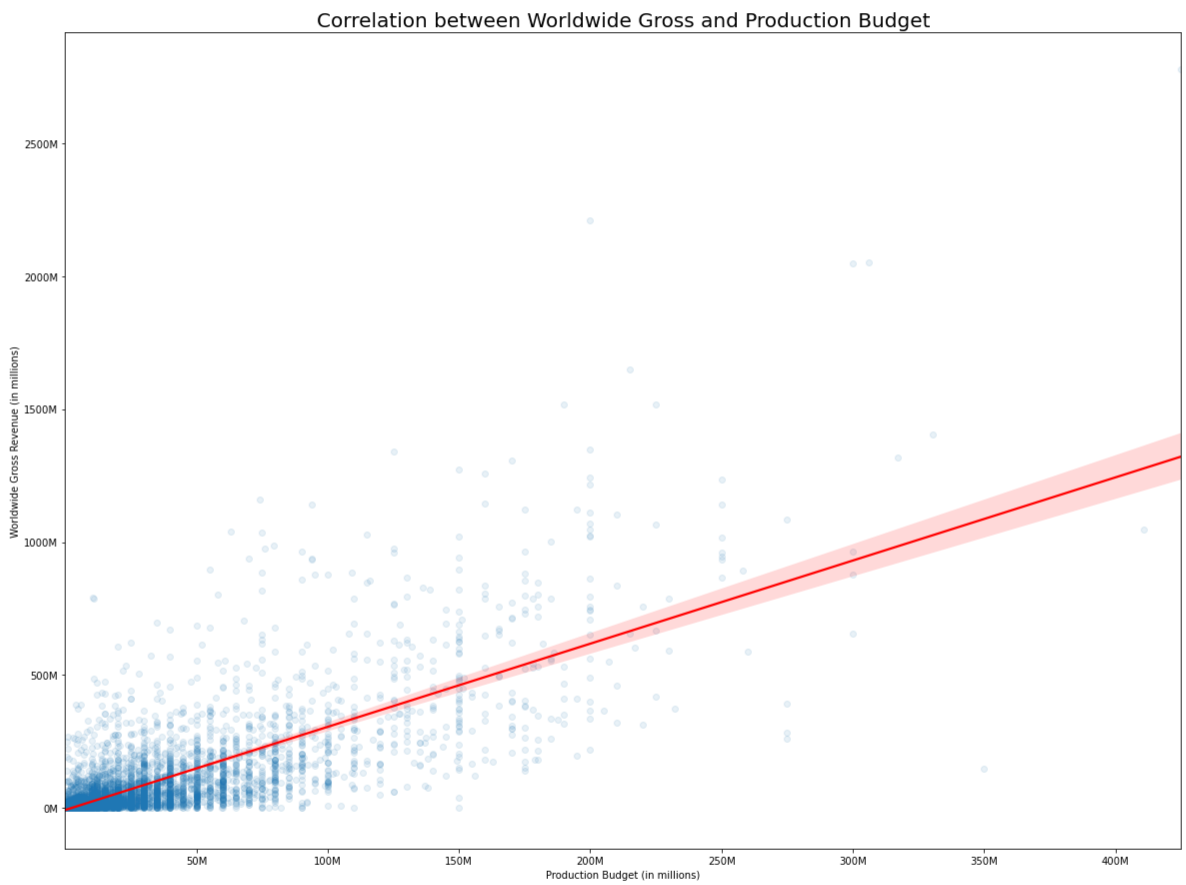


In [73]:
```
genre_popularity_df = genre_popularity_df.explode('genre_ids').grou

# sort the genres by popularity in descending order
genre_popularity_df = genre_popularity_df.sort_values(by='popularit

# select the top three genres by popularity
top_three_genres = genre_popularity_df['genre_ids'].head(3).tolist(

# print the top three genres
print(top_three_genres)

```

```
[12, 28, 14]
```

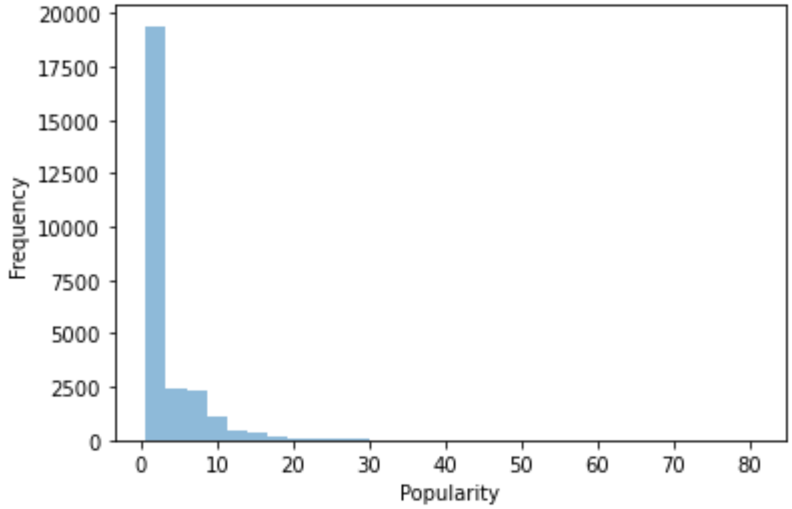
In [74]:

```
plt.figure(figsize = (20, 15))
plt.ticklabel_format(style='plain')
sns.regplot(
    data = sorted_worldwide_gross_movie_budgets_df,
    x = "production_budget",
    y = "worldwide_gross",
    scatter_kws = {"alpha": 0.1},
    line_kws = {"color": "red"}
)
plt.savefig("budget_wgross.jpg")

plt.xlabel('Production Budget (in millions)')
plt.gca().xaxis.set_major_formatter(ticker.FuncFormatter
        (lambda x, pos: f'{x/1000000:.0f}M'))

# Set the y-axis label and format the tick labels in millions
plt.ylabel('Worldwide Gross Revenue (in millions)')
plt.gca().yaxis.set_major_formatter(ticker.FuncFormatter
        (lambda x, pos: f'{x/1000000:.0f}M'))
plt.title('Correlation between Worldwide Gross and Production Budge


plt.savefig("budget_wgross.jpg")
```



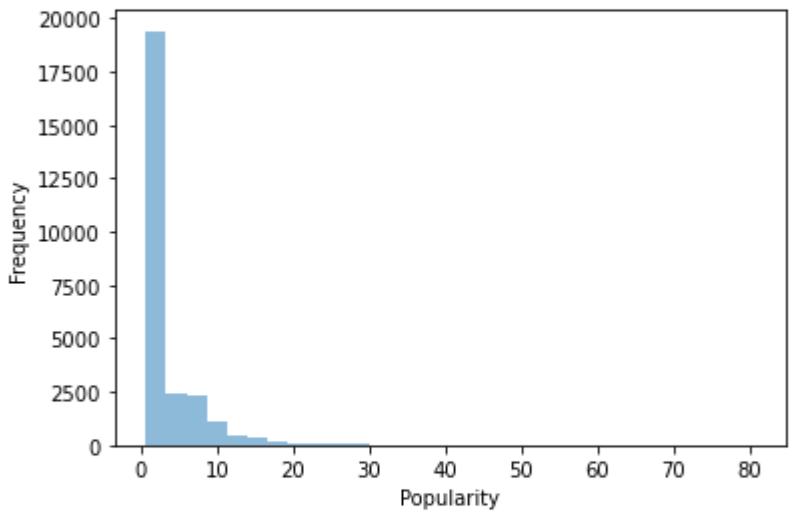
In [75]:
```
# Graphing descriptive statistics for the data frame (histogram)
ax = sorted_popularity_tmdb_movies_df["popularity"].plot.hist(bins=
ax.set_xlabel("Popularity")
ax.set_ylabel("Frequency")

```

Out[75]: Text(0, 0.5, 'Frequency')



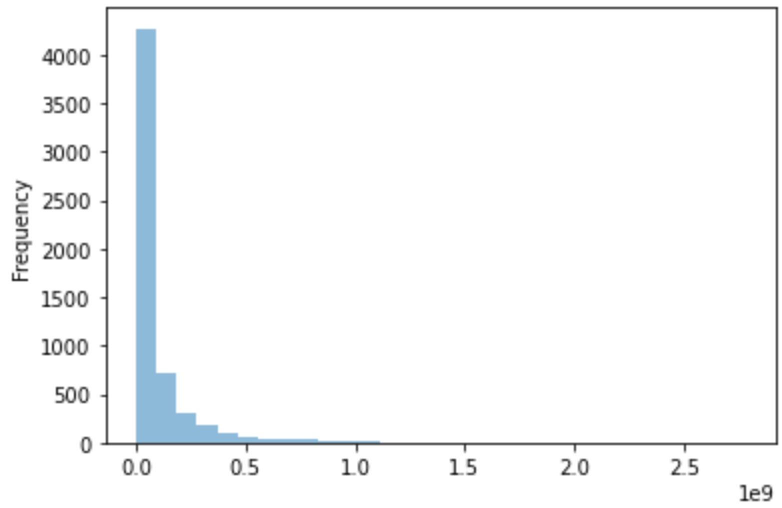
In [76]:
```
# Graphing descriptive statistics for the data frame (histogram) Le
ax = sorted_popularity_tmdb_movies_df["popularity"].plot.hist(bins=
ax.set_xlabel("Popularity")
ax.set_ylabel("Frequency")

```

Out[76]: Text(0, 0.5, 'Frequency')



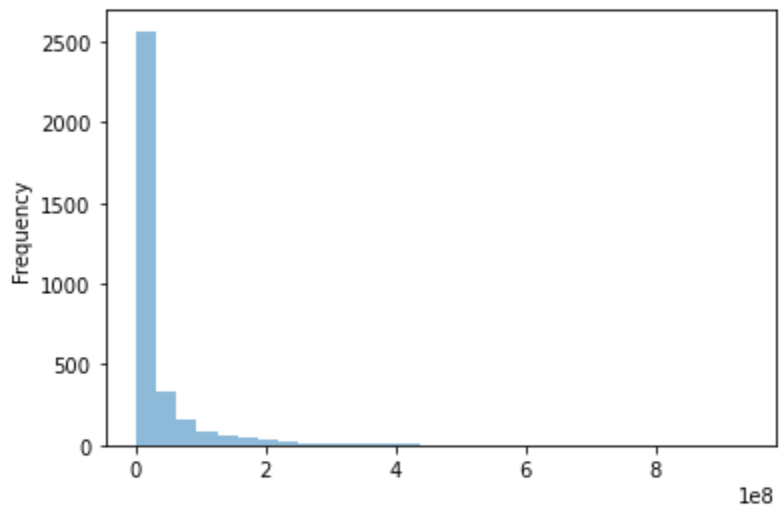
In [77]:
```
# Graphing descriptive statistics for the data frame (histogram) Le
sorted_worldwide_gross_movie_budgets_df["worldwide_gross"].plot.his
ax.set_xlabel("worldwide_gross")
ax.set_ylabel("Frequency")
```

Out[77]: Text(17.200000000000003, 0.5, 'Frequency')



In [78]:
```
# Graphing descriptive statistics for the data frame (histogram) Le
sorted_movie_gross_domestic_df["domestic_gross"].plot.hist(bins=30,
ax.set_xlabel("domestic_gross")
ax.set_ylabel("Frequency")
```

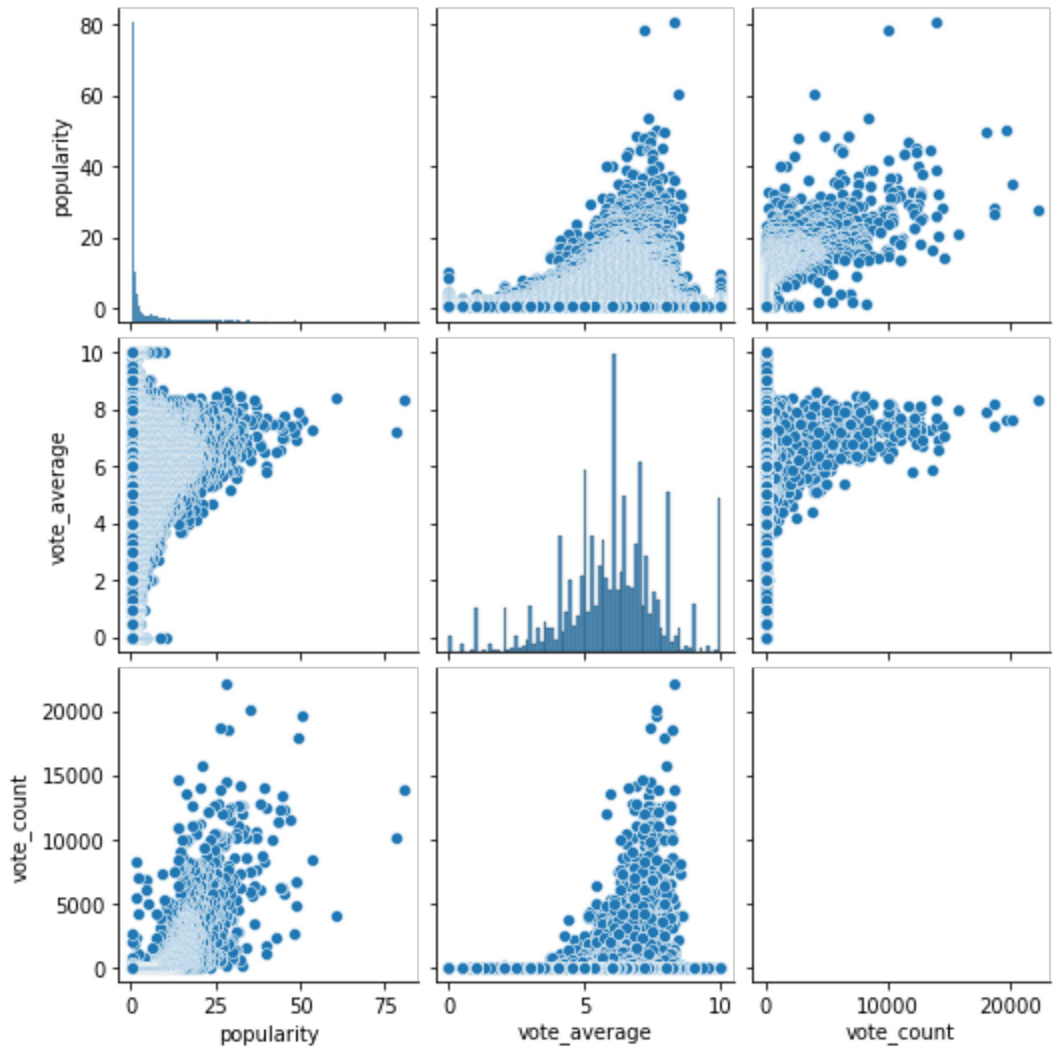
Out[78]: Text(17.200000000000003, 0.5, 'Frequency')



We can see that the relationship between popularity and vote average is fairly normally distributed, as well as vote count and vote average.

In [79]:
```
1 sns.pairplot(sorted_popularity_tmdb_movies_df)
```

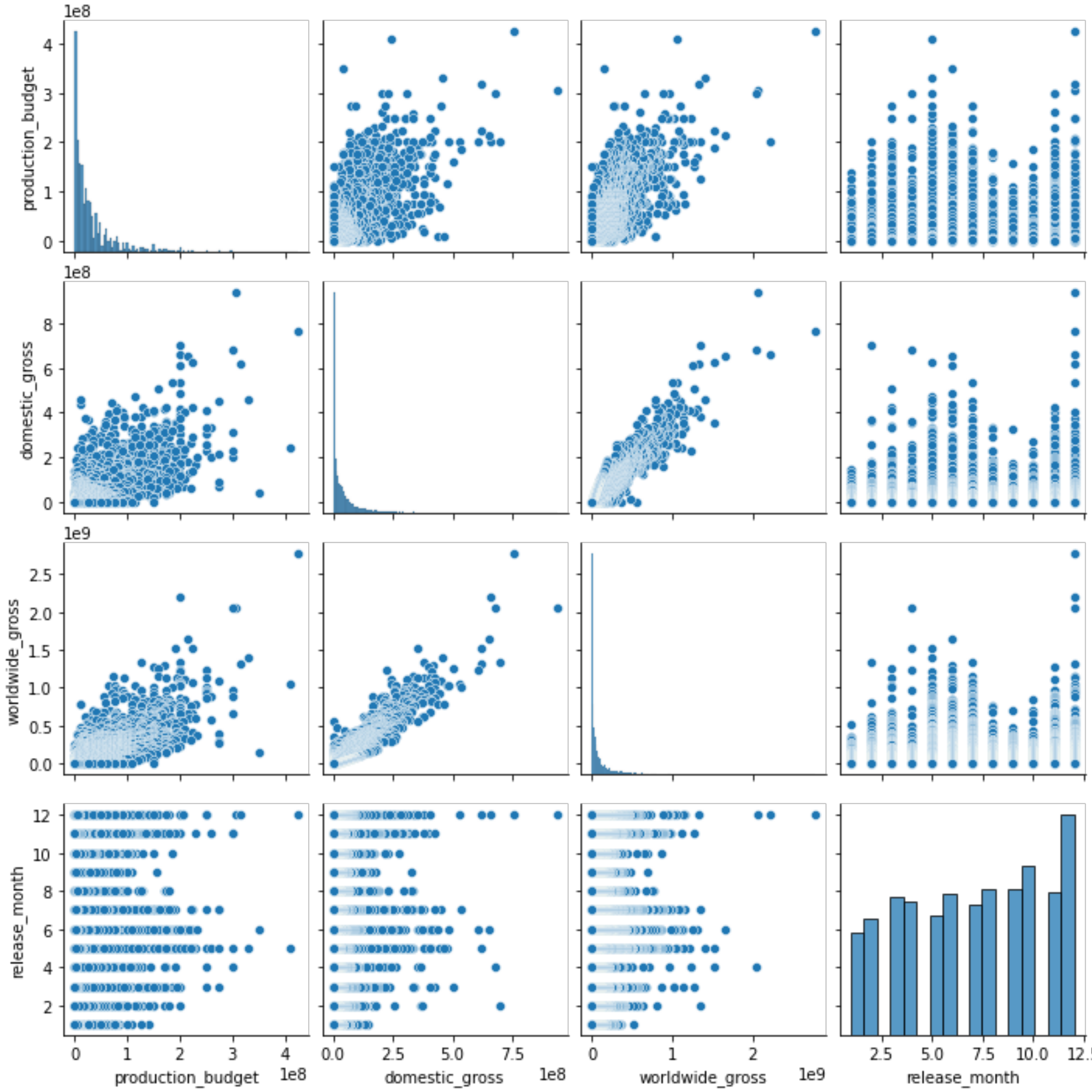
Out[79]: <seaborn.axisgrid.PairGrid at 0x7fbd990a3100>



In general we can see below that there is a positive correlation between production budget and domestic and worldwide gross of a movie.

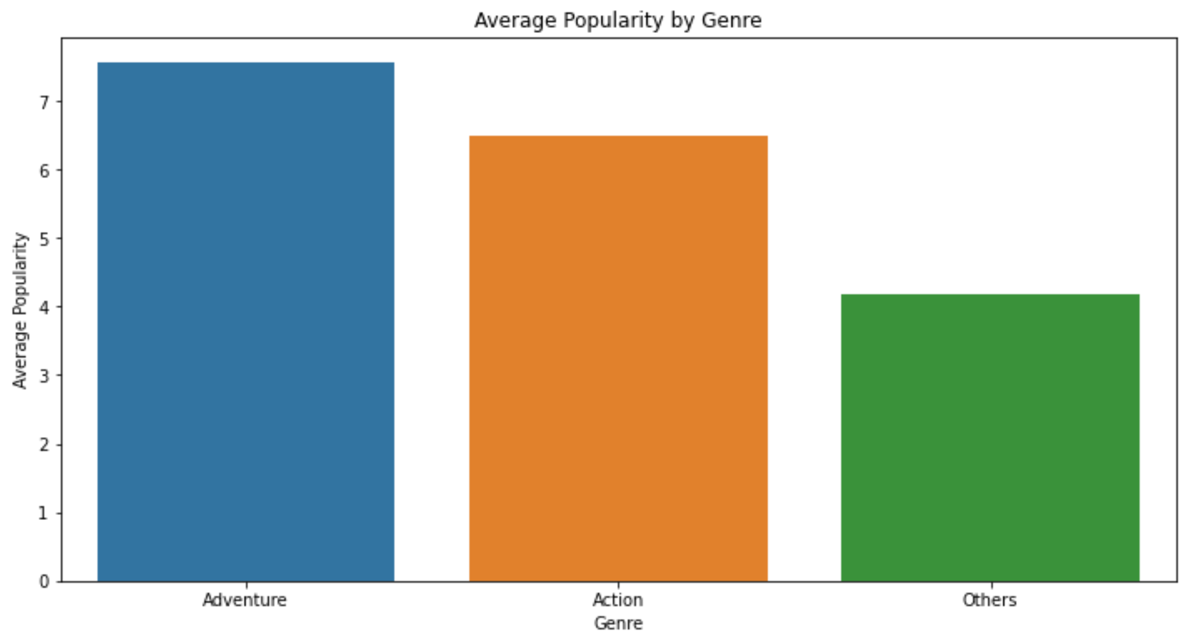
In [80]:
```
sns.pairplot(sorted_worldwide_gross_movie_budgets_df)
```

Out[80]: <seaborn.axisgrid.PairGrid at 0x7fbd966e9f10>



In [81]:
```
pd.options.display.float_format = '{:.2f}'.format

```

In [82]:
```
others_popularity = genre_popularity_df[~genre_popularity_df['genre
others = pd.DataFrame({'genre_ids': ['Others'], 'popularity': [othe

top_movies = genre_popularity_df[genre_popularity_df['genre_ids'].i
top_movies['genre_ids'] = top_movies['genre_ids'].replace({12: 'Adv
top_movies = pd.concat([top_movies, others])

plt.figure(figsize=(12, 6))
sns.barplot(data=top_movies, x='genre_ids', y='popularity')
plt.xlabel('Genre')
plt.ylabel('Average Popularity')
plt.title('Average Popularity by Genre')
plt.show()
```



Conclusion

Through my analysis I found that the most popular movies in the box office over the past 10 years have been in the genre's of "Action" and "Adventure".

In general there is a positive correlation between production budget and worldwide/domestic gross.

On average the highest grossing movies have been released in the month of May.

Recommendations

Based on my findings I would recommend that Microsoft should create movies that are within the "Action" and "Adventure" genres as they have the highest popularity rating.

I would recommend that Microsoft be prepared to investment more money into the production budget. Higher production budgets are more likely to lead to a higher grossing film.

Releasing a movie in one of these 3 months: May,June,July. Which have shown to produce the highest grossing films on average.

Limitations

One limitation that I found while investigating the data sets was that there were too many missing values in certain data sets which effected my ability to use it. The biggest limitation was the lack of information on net profit for production companies. Having this information would help in determining a movie's a success. Looking into how much money went into advertising for example could be an element that would effect total profit. Having numbers on production budget and worldwide gross are very valuable but it still does not tell the whole story.

Next Steps

Exploring more data sets is always helpful. The more information we have at our fingertips will only strenghten knowledge on a particular subject and lead to the best course of action. I would look into data sets focusing more on production companies to try and gather more information on what goes into making a successful film. Looking at data sets that have information on how much money went into advertising and also how long the movie was actively advertised for would be crucial to look at to ensure the best steps are being taken to produce a successful movie. Also looking at merchandising sales for a movie would be another helpful factor in determining success.