Movie Performance

- Author: Bella Scribner
- FLEX Program
- Instructor: Morgan Jones

Overview

Based on data from four different sources, this project explores what types of films have been successful. The analysis shows which genres were the most popular and were the most profitable on average over the past twenty years. Microsoft can utilize this information as a starting point in their new movie-making business venture.

The Situation

Microsoft would like some information concerning what kinds of movies are successful in consideration of creating a new movie studio venture where they create their own original content. In this analysis, we will focus on what genre(s) have been the most popular and have garnered the highest income on average over the past twenty years. Knowing what genres have been successful, will narrow down future research to be done -- such as what directors, writers, or actors to hire, what approximate budgets are required, and more.

Previewing the Data

Data from four different movie tracking sources were provided for this project. The sources are:

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

Box Office Mojo (BOM)

A single CSV file.

```
In [1]: import pandas as pd
```

In [2]:
```
bom_df = pd.read_csv('./zippedData/bom.movie_gross.csv.gz')
bom_df.head()
```

Out[2]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
3	Inception	WB	292600000.0	535700000	2010
4	Shrek Forever After	P/DW	238700000.0	513900000	2010

In [3]:
```
bom_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object 
 4   year            3387 non-null   int64  
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

The domestic gross income by movie would be a good measure of success. Furthermore, the year released will be helpful as we do not want to complete analysis on outdata information.

Internet Movie Database (IMDB)

This SQL database needed to be unzipped prior to utilizing.

In [4]:
```
#import zipfile

#with zipfile.ZipFile('./zippedData/im.db.zip', 'r') as zip:
#    zip.extractall(path='./zippedData/')
```

In [5]:
```
import sqlite3
conn = sqlite3.connect('./zippedData/im.db')
cur = conn.cursor()
```

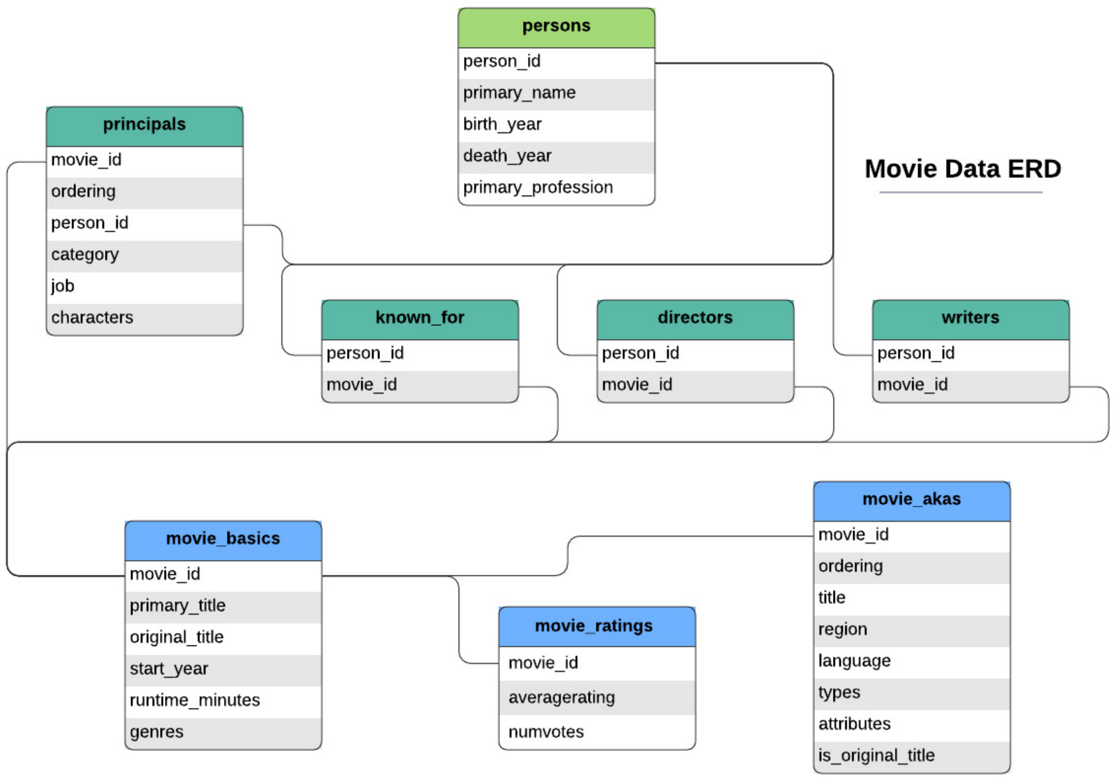
In [6]:
```
cur.execute("""SELECT name FROM sqlite_master WHERE type = 'table';""")
```

Out[6]:
```
<sqlite3.Cursor at 0x18cf11ac5e0>
```

In [7]:
```
table_names = cur.fetchall()
table_names
```

Out[7]:
```
[('movie_basics',),
 ('directors',),
 ('known_for',),
 ('movie_akas',),
 ('movie_ratings',),
 ('persons',),
 ('principals',),
 ('writers',)]
```

See the below given ERD for data set:



After inspecting the ERD above for this database, we will move into further exploration of the movie_akas, movie_ratings, and movie_basics tables. Potential analysis concerning the ratings as a measure of success across variables such as genre, director, and writer.

In [8]:
```
pd.read_sql("""
SELECT *
FROM movie_basics
LIMIT 10;
""", conn)
```

Out[8]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	ge
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Dı
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Dı
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Dı
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Dı
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Far
5	tt0111414	A Thin Life	A Thin Life	2018	75.0	Cor
6	tt0112502	Bigfoot	Bigfoot	2017	NaN	Horror,Tł
7	tt0137204	Joe Finds Grace	Joe Finds Grace	2017	83.0	Adventure,Animation,Cor
8	tt0139613	O Silêncio	O Silêncio	2012	NaN	Documentary,Hi
9	tt0144449	Nema aviona za Zagreb	Nema aviona za Zagreb	2012	82.0	Biogr

In [9]:
```
pd.read_sql("""
SELECT *
FROM movie_akas
LIMIT 5;
""", conn)
```

Out[9]:

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

In [10]:
```
pd.read_sql("""
SELECT *
FROM movie_ratings
LIMIT 5;
""", conn)
```

Out[10]:

	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

While the table movie_akas will not be helpful in further analysis, the movie_basics and movie_ratings tables will be fabulous! Can do potential analysis on average rating of movies based on director, writer, and genre grouping (genre grouping will need a bit of a deeper look). Furthermore, we can use the title or original_title data to maybe merge the genre (or director, or writer information) to another table that has data concerning the average income made by that grouping of movies.

Rotten Tomatoes Data

There are two seperate tabular files from Rotten tomatoes: movie_info and reviews.

In [11]:
```
rt_movie_info_df = pd.read_csv('./zippedData/rt.movie_info.tsv.gz', delimi
rt_movie_info_df.head()
```

Out[11]:

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

In [12]:
```
rt_movie_info_df.info()
```

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

In [13]:
```
rt_reviews_df = pd.read_csv('./zippedData/rt.reviews.tsv.gz', delimiter="\
rt_reviews_df.head()
```

Out[13]:

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

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

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

Based on preview, we could connect these two tables and compare average review ratings by genre, and/or movie rating (G, PG, R, etc). The box_office data seems interesting, but maybe not worth pursuing with so many missing values.

TheMovieDatabase (TMDB)

In [15]:
```
tmdb_df = pd.read_csv('./zippedData/tmdb.movies.csv.gz')
tmdb_df.head()
```

Out[15]:

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

In [16]:
```
tmdb_df.info()
```

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

While the genre_ids information will not not be helpful without further investigation on how TheMovieDataBase transposes the descriptive genre name to the numbers shown, the popularity, vote_average, and vote_count columns could all be useful measures of success.

The Numbers

In [17]:
```
tn_df = pd.read_csv('./zippedData/tn.movie_budgets.csv.gz')
tn_df.head()
```

Out[17]:

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

In [18]:
```
tn_df.info()
```

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

The budget column is very promising! With this we could deduce net income for each entry -- always an important performance statistic to consider.

Prepping the Data for Analysis

Based on the previews above, we will skip over the data provided from Rotten Tomatoes due to the smaller sample size. For this analysis we are going to focus on the performance of the most successful genres based on that rating, production budget, domestic gross income, and domestic net income.

Any movies released prior to 2003 will be dropped from the analysis. Entries missing pertinent data will be dropped as well. Any joins, between data from different sources, that need to occur will be inner joins based on movie titles -- movie titles will be cleaned as best possible prior to the join.

The analysis we would like to pursue, and the data sources that need to be cleaned and prepped, are as follows:

1. Genre by Domestic Gross Income (IMDB and BOM)
2. Genre by Budget (IMDB and The Numbers)
3. Genre by Domestic Gross Income (IMDB and The Numbers)
4. Genre by Net Income (IMDB and The Numbers)
5. Genre by Rating (IMDB database only)
6. Genre by Popularity + Vote Average (IMDB and TMDB)

Genre by Domestic Gross Income (IMDB and BOM)

First, we will check the BOM data to see if any entries need to be dropped based on the release date criteria. Then, seeing as the data will need to be merged based on the title names, will do some investigation on naming conventions for each table and some text cleaning if needed.

In [19]:
```
print(max(bom_df['year']))
print(min(bom_df['year']))
```

```
2018
2010
```

The year span of 2010-2018 is acceptable. No data needs to be dropped due to the release date, however the fact that no data from the past five years exists should be noted in the analysis later on.

Next, let's look at some naming conventions. We saw from the BOM data preview that some of the entries have a note on the release data in the title -- such as `Alice in Wonderland (2010)`. Let's check for parentheses in both the movie_basics table and bom_df.

In [20]:
```
bom_df.duplicated(keep=False).value_counts()
```

Out[20]:
```
False    3387
dtype: int64
```

In [21]:
```
bom_df[bom_df['title'].str.contains("\(")]
```

Out[21]:

	title	studio	domestic_gross	foreign_gross	year
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
10	Clash of the Titans (2010)	WB	163200000.0	330000000	2010
55	A Nightmare on Elm Street (2010)	WB (NL)	63100000.0	52600000	2010
63	Aftershock (Tangshan Dadizhen)	CL	63000.0	100200000	2010
79	If You Are the One 2 (Fei Cheng Wu Rao II)	CL	427000.0	75600000	2010
...	...	...	...	...	...
3341	Unstoppable (2018)	WGUSA	101000.0	NaN	2018
3365	The Apparition (2018)	MBox	28300.0	NaN	2018
3378	Hannah (2018)	PDF	11700.0	NaN	2018
3380	Furious (Legend of Kolovrat)	CARUSEL	10000.0	NaN	2018
3383	Edward II (2018 re-release)	FM	4800.0	NaN	2018

327 rows × 5 columns

In [22]:
```
pd.read_sql("""
SELECT *
FROM movie_basics
WHERE primary_title LIKE "%(%"
;""", conn)
```

Out[22]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	
0	tt0756727	Who Is Harry Nilsson (And Why Is Everybody Tal...	Who Is Harry Nilsson (And Why Is Everybody Tal...	2010	116.0	Biography,Docume
1	tt0860907	Evangelion: 3.0 You Can (Not) Redo	Evangerion shin gekijôban: Kyu	2012	96.0	Action,Anima
2	tt10009978	Bigger Like Me (Extended Director's Cut)	Bigger Like Me (Extended Director's Cut)	2019	100.0	D
3	tt10021804	Ne travaille pas (1968 - 2018)	Ne travaille pas (1968 - 2018)	2018	88.0	D
4	tt10047768	Nikos Dragoumis: Enas zografos sti skia tis is...	Nikos Dragoumis: Enas zografos sti skia tis is...	2015	NaN	Biography,D
...	...	...	...	...	...	
726	tt9890106	Boy Meets Girl (Tholiprema Katha)	Boy Meets Girl (Tholiprema Katha)	2014	NaN	
727	tt9894722	Tipi da spiaggia (la riviera romagnola)	Tipi da spiaggia (la riviera romagnola)	2018	NaN	D
728	tt9900670	Waiting(Words Apart)	Waiting(Words Apart)	2019	NaN	
729	tt9906128	Leak (Penangkeb)	Leak (Penangkeb)	2019	75.0	
730	tt9908592	Filmmakers Unite (FU)	Filmmakers Unite (FU)	2018	78.0	D

731 rows × 6 columns

We can see that there is not a clear naming convention in either database when it comes to using the parentheses, which is unfortunate.

Let's also check for titles that contain colons. Furthermore, let's see if movies with parts in their title, such as the Harry Potter movies, include colons or not and if it is consistent within a specific database, or between the two databases.

In [23]:
```python
bom_df[bom_df['title'].str.contains(":")].sort_values(by='title')
```

Out[23]:

	title	studio	domestic_gross	foreign_gross	year
2418	13 Hours: The Secret Soldiers of Benghazi	Par.	52900000.0	16600000	2016
3264	2001: A Space Odyssey (2018 re-release)	WB	3200000.0	NaN	2018
3078	2:22	Magn.	400.0	NaN	2017
1501	300: Rise of An Empire	WB	106600000.0	231000000	2014
1103	4:44: Last Day on Earth	IFC	17800.0	NaN	2012
...	...	...	...	...	...
1482	X-Men: Days of Future Past	Fox	233900000.0	513900000	2014
345	X-Men: First Class	Fox	146400000.0	207200000	2011
2434	Yo-kai Watch: The Movie	Elev.	257000.0	47600000	2016
1216	Young Detective Dee: Rise of the Sea Dragon	WAMCR	87800.0	72200000	2013
2785	xXx: The Return of Xander Cage	Par.	44900000.0	301200000	2017

273 rows × 5 columns

In [24]:
```
pd.read_sql("""
SELECT primary_title, original_title, start_year
FROM movie_basics
WHERE primary_title LIKE "%:%"
;""", conn)
```

Out[24]:

	primary_title	original_title	start_year
0	Cooper and Hemingway: The True Gen	Cooper and Hemingway: The True Gen	2013
1	Prague: The Restless Heart of Europe	Praha - neklidné srdce Evropy	2018
2	Quantum Quest: A Cassini Space Odyssey	Quantum Quest: A Cassini Space Odyssey	2010
3	Hempsters: Plant the Seed	Hempsters: Plant the Seed	2010
4	Caleuche: El llamado del Mar	Caleuche: El llamado del Mar	2012
...	...	...	...
16163	Footloose in Italy IV: 4 Rimini Tuscany Rome	Footloose in Italy IV: 4 Rimini Tuscany Rome	2016
16164	Footloose in London II: 2 Undiscovered and Unu...	Footloose in London II: 2 Undiscovered and Unu...	2018
16165	The Good Americans: One Revolution, Two Nations	The Good Americans: One Revolution, Two Nations	2019
16166	Doctor Who Augmented Reality: Times Magazine	Doctor Who Augmented Reality: Times Magazine	2013
16167	9/11: Escape from the Towers	9/11: Escape from the Towers	2018

16168 rows × 3 columns

In [25]:
```
bom_df[bom_df['title'].str.contains(": Part")]
```

Out[25]:

	title	studio	domestic_gross	foreign_gross	year
851	Katy Perry: Part of Me	Par.	25300000.0	7400000	2012
2138	Attack on Titan: Part 1	FUN	450000.0	NaN	2015
2156	Attack on Titan: Part 2	FUN	306000.0	NaN	2015

In [26]:
```
pd.read_sql("""
SELECT primary_title, original_title, start_year
FROM movie_basics
WHERE primary_title LIKE "%: Part%"
;""", conn)
```

Out[26]:

	primary_title	original_title	start_year
0	Atlas Shrugged: Part I	Atlas Shrugged: Part I	2011
1	Harry Potter and the Deathly Hallows: Part 1	Harry Potter and the Deathly Hallows: Part 1	2010
2	El Encanto de la Oscuridad: Parte 2	El Encanto de la Oscuridad: Parte 2	2018
3	Bhoot: Part One - The Haunted Ship	Bhoot: Part One - The Haunted Ship	2019
4	Harry Potter and the Deathly Hallows: Part 2	Harry Potter and the Deathly Hallows: Part 2	2011
...	...	...	...
88	The Laws of the Universe: Part I	Uchu no Ho: Reimei-hen	2018
89	The Morning After: Part One	The Morning After: Part One	2018
90	The Art of Mandolin in Crete: part 2	The Art of Mandolin in Crete: part 2	2018
91	Arwah Tumbal Nyai: part Nyai	Arwah Tumbal Nyai: part Nyai	2018
92	Bit X Bit: Part 2 Integration and Regulation	Bit X Bit: Part 2 Integration and Regulation	2019

93 rows × 3 columns

```
In [27]: bom_df[bom_df['title'].str.contains("Part")]
```

Out[27]:

	title	studio	domestic_gross	foreign_gross	year
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
328	Harry Potter and the Deathly Hallows Part 2	WB	381000000.0	960500000	2011
331	The Twilight Saga: Breaking Dawn Part 1	Sum.	281300000.0	430900000	2011
335	The Hangover Part II	WB	254500000.0	332300000	2011
606	Chillar Party	UTV	6300.0	826000	2011
732	The Twilight Saga: Breaking Dawn Part 2	LG/S	292300000.0	537400000	2012
851	Katy Perry: Part of Me	Par.	25300000.0	7400000	2012
1145	The Hangover Part III	WB	112200000.0	249800000	2013
1280	The Last Exorcism Part II	CBS	15200000.0	NaN	2013
1481	The Hunger Games: Mockingjay - Part 1	LGF	337100000.0	418200000	2014
1651	Alan Partridge: The Movie	Magn.	153000.0	9600000	2014
1880	The Hunger Games: Mockingjay - Part 2	LGF	281700000.0	371700000	2015
2063	Spare Parts	LGF	3600000.0	NaN	2015
2138	Attack on Titan: Part 1	FUN	450000.0	NaN	2015
2156	Attack on Titan: Part 2	FUN	306000.0	NaN	2015
2179	The Farewell Party	Gold.	173000.0	NaN	2015
2382	Sausage Party	Sony	97700000.0	43000000	2016
2393	Office Christmas Party	Par.	54800000.0	59700000	2016
2750	Search Party	FCW	4600.0	NaN	2016
3164	Life of the Party	WB (NL)	53100000.0	12800000	2018
3293	The Party (2017)	RAtt.	750000.0	NaN	2018

As we can see there does not seem to be a clear naming convention even within a specific table. We will have to simply strip the leading and extending whitespace of the titles, merge the two tables on this title, and hope that we do not lose too much data. With increased technical know-how, there would be a more sophisticated way to clean, search for, and join similar text strings which would increase the amount of data kept and lead to more reliable results.

Let's also check what date range of release years the movie_basics table includes.

In [28]:
```
q = """
SELECT start_year, COUNT(*)
FROM movie_basics
GROUP BY start_year
"""
pd.read_sql(q, conn)
```

Out[28]:

	start_year	COUNT(*)
0	2010	11849
1	2011	12900
2	2012	13787
3	2013	14709
4	2014	15589
5	2015	16243
6	2016	17272
7	2017	17504
8	2018	16849
9	2019	8379
10	2020	937
11	2021	83
12	2022	32
13	2023	5
14	2024	2
15	2025	1
16	2026	1
17	2027	1
18	2115	1

As we can see above, there are some incorrect entries in the data seeing as years 2024 and above have not occured yet. Luckily there are no movie entries released prior to 2010 which falls within our limiting criteria of the past 20 years.

In [29]:
```
query = """
SELECT primary_title as title, movie_id, start_year as year, genres
FROM movie_basics
WHERE year < 2024
"""
movie_basics_df = pd.read_sql(query, conn)
movie_basics_df.head()
```

Out[29]:

	title	movie_id	year	genres
0	Sunghursh	tt0063540	2013	Action,Crime,Drama
1	One Day Before the Rainy Season	tt0066787	2019	Biography,Drama
2	The Other Side of the Wind	tt0069049	2018	Drama
3	Sabse Bada Sukh	tt0069204	2018	Comedy,Drama
4	The Wandering Soap Opera	tt0100275	2017	Comedy,Drama,Fantasy

In [30]:
```
movie_basics_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146138 entries, 0 to 146137
Data columns (total 4 columns):
 #   Column    Non-Null Count   Dtype 
---  ------    --------------   ----- 
 0   title     146138 non-null  object
 1   movie_id  146138 non-null  object
 2   year      146138 non-null  int64 
 3   genres    140731 non-null  object
dtypes: int64(1), object(3)
memory usage: 4.5+ MB
```

In [31]:
```
movie_basics_df.duplicated(keep=False).value_counts()
```

Out[31]:
```
False    146138
dtype: int64
```

In [32]:
```
movie_basics_df['titleClean'] = movie_basics_df['title'].str.strip()
```

In [33]:
```
bom_df['titleClean'] = bom_df['title'].str.strip()
```

In [34]:
```
bom_df
```

Out[34]:

	title	studio	domestic_gross	foreign_gross	year	titleClean
0	Toy Story 3	BV	415000000.0	652000000	2010	Toy Story 3
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010	Alice in Wonderland (2010)
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010	Harry Potter and the Deathly Hallows Part 1
3	Inception	WB	292600000.0	535700000	2010	Inception
4	Shrek Forever After	P/DW	238700000.0	513900000	2010	Shrek Forever After
...	...	...	...	...	...	...
3382	The Quake	Magn.	6200.0	NaN	2018	The Quake
3383	Edward II (2018 re-release)	FM	4800.0	NaN	2018	Edward II (2018 re-release)
3384	El Pacto	Sony	2500.0	NaN	2018	El Pacto
3385	The Swan	Synergetic	2400.0	NaN	2018	The Swan
3386	An Actor Prepares	Grav.	1700.0	NaN	2018	An Actor Prepares

3387 rows × 6 columns

In [35]:
```
IMDB_and_BOM_df = bom_df.merge(movie_basics_df, on=["titleClean", "year"],
IMDB_and_BOM_df.head()
```

Out[35]:

	title_x	studio	domestic_gross	foreign_gross	year	titleClean	title_y	movie_id
0	Toy Story 3	BV	415000000.0	652000000	2010	Toy Story 3	Toy Story 3	tt0435761
1	Inception	WB	292600000.0	535700000	2010	Inception	Inception	tt1375666
2	Shrek Forever After	P/DW	238700000.0	513900000	2010	Shrek Forever After	Shrek Forever After	tt0892791
3	The Twilight Saga: Eclipse	Sum.	300500000.0	398000000	2010	The Twilight Saga: Eclipse	The Twilight Saga: Eclipse	tt1325004
4	Iron Man 2	Par.	312400000.0	311500000	2010	Iron Man 2	Iron Man 2	tt1228705

In [36]:
```
IMDB_and_BOM_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1873 entries, 0 to 1872
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title_x         1873 non-null   object 
 1   studio          1871 non-null   object 
 2   domestic_gross  1863 non-null   float64
 3   foreign_gross   1278 non-null   object 
 4   year            1873 non-null   int64  
 5   titleClean      1873 non-null   object 
 6   title_y         1873 non-null   object 
 7   movie_id        1873 non-null   object 
 8   genres          1871 non-null   object 
dtypes: float64(1), int64(1), object(7)
memory usage: 146.3+ KB
```

Wonderful! Now we can group by genre and use that for analysis with domestic gross income (dropping entries that do not include this information).

In [37]:
```
IMDB_and_BOM_df.drop(IMDB_and_BOM_df.loc[:,('studio','foreign_gross', 'yea
```

In [38]:
```
IMDB_and_BOM_df.dropna(inplace=True)
```

In [39]:
```
grouped = IMDB_and_BOM_df.groupby('genres').mean().sort_values(by=['domest
top_ten_gross_BOM = grouped.head(10)
top_ten_gross_BOM
```

Out[39]:

	domestic_gross
genres	
Action,Adventure,Sci-Fi	2.437244e+08
Adventure,Drama,Sci-Fi	2.082000e+08
Adventure,Fantasy	1.929000e+08
Biography,Drama,Musical	1.743000e+08
Action,Adventure,Fantasy	1.530602e+08
Action,Adventure,Mystery	1.509000e+08
Animation,Comedy,Family	1.458669e+08
Comedy,Music	1.446000e+08
Adventure,Animation,Comedy	1.413663e+08
Action,Drama,Family	1.310500e+08

In [40]:
```
top_ten_gross_BOM.reset_index(inplace=True)
top_ten_gross_BOM
```

Out[40]:

	genres	domestic_gross
0	Action,Adventure,Sci-Fi	2.437244e+08
1	Adventure,Drama,Sci-Fi	2.082000e+08
2	Adventure,Fantasy	1.929000e+08
3	Biography,Drama,Musical	1.743000e+08
4	Action,Adventure,Fantasy	1.530602e+08
5	Action,Adventure,Mystery	1.509000e+08
6	Animation,Comedy,Family	1.458669e+08
7	Comedy,Music	1.446000e+08
8	Adventure,Animation,Comedy	1.413663e+08
9	Action,Drama,Family	1.310500e+08

Nice! Now we have the top ten genres by average domestic gross income. We will use this data frame for our analysis later.

Genre by Budget (IMDB and The Numbers)

We will continue to utilize the `movie_basics_df` created in the previous section for subsequent joins. Below, we will review the two dataframes we are planning on merging (`movie_basics_df` and `tn_df` from TheNumbers source) and see if any further data cleaning needs to be accomplished prior to and after said merge.

In [41]:
```
movie_basics_df.head()
```

Out[41]:

	title	movie_id	year	genres	titleClean
0	Sunghursh	tt0063540	2013	Action,Crime,Drama	Sunghursh
1	One Day Before the Rainy Season	tt0066787	2019	Biography,Drama	One Day Before the Rainy Season
2	The Other Side of the Wind	tt0069049	2018	Drama	The Other Side of the Wind
3	Sabse Bada Sukh	tt0069204	2018	Comedy,Drama	Sabse Bada Sukh
4	The Wandering Soap Opera	tt0100275	2017	Comedy,Drama,Fantasy	The Wandering Soap Opera

In [42]:
```
tn_df.head()
```

Out[42]:

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

In [43]:
```
tn_df.duplicated(keep=False).value_counts()
```

Out[43]:
```
False    5782
dtype: int64
```

In [44]:
```
tn_df.info()
```

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

From the review, we can see that the data included in the `production_budget` and `domestic_gross` columns need to be transposed into integers prior to analysis. Furthermore, the date needs to be either transposed, or a new column with just the release year as an integer needs to be created so we can apply the limiting criteria to the data set.

In [45]:
```
tn_df['production_budget_int'] = tn_df['production_budget'].map(lambda x:
```

In [46]:
```
tn_df['domestic_gross_int'] = tn_df['domestic_gross'].map(lambda x: int(x[
```

In [47]:
```
tn_df['year'] = tn_df['release_date'].map(lambda x: int(x[-4:]))
```

In [48]:
```
min(tn_df['year'])
```

Out[48]:
```
1915
```

In [49]:
```
max(tn_df['year'])
```

Out[49]:
```
2020
```

We will drop all entries that have a release date prior to 2003. It is good to see that The Numbers data includes more recent movie entries than information provided by BOM, based on the most recent release date year of 2020 rather than 2018.

In [50]:
```
tn_df = tn_df[tn_df['year'] >= 2003]
tn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3807 entries, 0 to 5781
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   id                     3807 non-null   int64 
 1   release_date           3807 non-null   object
 2   movie                  3807 non-null   object
 3   production_budget      3807 non-null   object
 4   domestic_gross         3807 non-null   object
 5   worldwide_gross        3807 non-null   object
 6   production_budget_int  3807 non-null   int64 
 7   domestic_gross_int     3807 non-null   int64 
 8   year                   3807 non-null   int64 
dtypes: int64(4), object(5)
memory usage: 297.4+ KB
```

In [51]:
```
tn_df['titleClean'] = tn_df['movie'].map(lambda x: x.strip())
```

In [52]:
```
tn_df.head()
```

Out[52]:

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

In [53]:
```
IMDB_and_theNums_df = tn_df.merge(movie_basics_df, on=["titleClean", "year
IMDB_and_theNums_df.head()
```

Out[53]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	produ
0	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875	
1	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350	
2	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963	
3	7	Apr 27, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,048,134,200	
4	9	Nov 17, 2017	Justice League	$300,000,000	$229,024,295	$655,945,209	

In [54]:
```
IMDB_and_theNums_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1547 entries, 0 to 1546
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   id                     1547 non-null   int64 
 1   release_date           1547 non-null   object
 2   movie                  1547 non-null   object
 3   production_budget      1547 non-null   object
 4   domestic_gross         1547 non-null   object
 5   worldwide_gross        1547 non-null   object
 6   production_budget_int  1547 non-null   int64 
 7   domestic_gross_int     1547 non-null   int64 
 8   year                   1547 non-null   int64 
 9   titleClean             1547 non-null   object
 10  title                  1547 non-null   object
 11  movie_id               1547 non-null   object
 12  genres                 1541 non-null   object
dtypes: int64(4), object(9)
memory usage: 169.2+ KB
```

In [55]:
```
grouped = IMDB_and_theNums_df.groupby('genres').mean()
top_ten_budget_theNums = grouped.sort_values(by=['production_budget_int'],
top_ten_budget_theNums
```

Out[55]:

	id	production_budget_int	domestic_gross_int	y
genres				
Adventure,Fantasy	36.333333	2.316667e+08	1.928914e+08	2013.333
Fantasy,Musical	51.000000	2.000000e+08	3.341911e+08	2010.000
Action,Sci-Fi	95.000000	1.780000e+08	1.002063e+08	2014.000
Action,Adventure,Sci-Fi	46.673077	1.731615e+08	2.409982e+08	2014.519
Family,Fantasy,Musical	35.000000	1.600000e+08	5.040142e+08	2017.000
Action,Adventure,Fantasy	53.612903	1.498581e+08	1.296164e+08	2015.032
Adventure,Family,Fantasy	41.928571	1.491714e+08	1.378265e+08	2014.428
Adventure,Drama,Sci-Fi	39.000000	1.365000e+08	2.082258e+08	2014.500
Action,Adventure,Animation	43.000000	1.279333e+08	1.755641e+08	2013.733
Adventure,Mystery,Sci-Fi	75.000000	1.250000e+08	1.264771e+08	2012.000

In [56]:
```
top_ten_budget_theNums.reset_index(inplace=True)
top_ten_budget_theNums.drop(top_ten_budget_theNums.loc[:,('id', 'domestic_
```

In [57]:
```
top_ten_budget_theNums
```

Out[57]:

	genres	production_budget_int
0	Adventure,Fantasy	2.316667e+08
1	Fantasy,Musical	2.000000e+08
2	Action,Sci-Fi	1.780000e+08
3	Action,Adventure,Sci-Fi	1.731615e+08
4	Family,Fantasy,Musical	1.600000e+08
5	Action,Adventure,Fantasy	1.498581e+08
6	Adventure,Family,Fantasy	1.491714e+08
7	Adventure,Drama,Sci-Fi	1.365000e+08
8	Action,Adventure,Animation	1.279333e+08
9	Adventure,Mystery,Sci-Fi	1.250000e+08

Great! Now we have the top ten genres that had the largest average production budgets.

Genre by Domestic Gross Income (IMDB and The Numbers)

Luckily most of the work in cleaning data from both IMDB and The Numbers has already been done, so we will just take the cleaned, joined data frame and widdle down the pertinent information for analysis on domestic gross income rather than production budget.

In [58]:
```
top_ten_gross_theNums = grouped.sort_values(by=['domestic_gross_int'], asc
top_ten_gross_theNums
```

Out[58]:

	id	production_budget_int	domestic_gross_int	y
genres				
Family,Fantasy,Musical	35.000000	1.600000e+08	5.040142e+08	2017.000(
Fantasy,Musical	51.000000	2.000000e+08	3.341911e+08	2010.000(
Action,Adventure,Sci-Fi	46.673077	1.731615e+08	2.409982e+08	2014.519
Adventure,Drama,Sci-Fi	39.000000	1.365000e+08	2.082258e+08	2014.500(
Drama,Family,Fantasy	13.000000	9.500000e+07	2.011514e+08	2015.000(
Adventure,Fantasy	36.333333	2.316667e+08	1.928914e+08	2013.333
Action,Adventure,Animation	43.000000	1.279333e+08	1.755641e+08	2013.733
Animation,Comedy,Family	53.000000	6.960000e+07	1.750288e+08	2013.000(
Biography,Drama,Musical	25.000000	8.400000e+07	1.743402e+08	2017.000(
Adventure,Drama,Western	29.000000	3.500000e+07	1.712430e+08	2010.000(

In [59]:
```
top_ten_gross_theNums.reset_index(inplace=True)
top_ten_gross_theNums.drop(top_ten_gross_theNums.loc[:,('id', 'production_
```

In [60]:
```
top_ten_gross_theNums
```

Out[60]:

	genres	domestic_gross_int
0	Family,Fantasy,Musical	5.040142e+08
1	Fantasy,Musical	3.341911e+08
2	Action,Adventure,Sci-Fi	2.409982e+08
3	Adventure,Drama,Sci-Fi	2.082258e+08
4	Drama,Family,Fantasy	2.011514e+08
5	Adventure,Fantasy	1.928914e+08
6	Action,Adventure,Animation	1.755641e+08
7	Animation,Comedy,Family	1.750288e+08
8	Biography,Drama,Musical	1.743402e+08
9	Adventure,Drama,Western	1.712430e+08

Genre by Domestic Net Income (IMDB and The Numbers)

For this portion, we will make the assumption that the production budget reported equals what the movie actually cost to create. With this assumption, we can deduce the domestic net income for each movie entry, and then the average net income for each genre.

In [61]:
```
IMDB_and_theNums_df['domestic_net'] = IMDB_and_theNums_df['domestic_gross_
IMDB_and_theNums_df.head()
```

Out[61]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	produ
0	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875	
1	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350	
2	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963	
3	7	Apr 27, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,048,134,200	
4	9	Nov 17, 2017	Justice League	$300,000,000	$229,024,295	$655,945,209	

In [62]:
```
grouped = IMDB_and_theNums_df.groupby('genres').mean().sort_values(by=['do
top_ten_net_theNums = grouped.head(10)
top_ten_net_theNums
```

Out[62]:

	id	production_budget_int	domestic_gross_int	year	dome
genres					
Family,Fantasy,Musical	35.0	160000000.0	504014165.0	2017.0	3440
Adventure,Drama,Western	29.0	35000000.0	171243005.0	2010.0	1362
Fantasy,Musical	51.0	200000000.0	334191110.0	2010.0	134
Drama,Family,Fantasy	13.0	95000000.0	201151353.0	2015.0	1061
Animation,Comedy,Family	53.0	69600000.0	175028795.2	2013.0	1054
Action,Comedy,Documentary	98.0	20000000.0	117229692.0	2010.0	972
Biography,Drama,Musical	25.0	84000000.0	174340174.0	2017.0	903
Comedy,Mystery	30.0	40100000.0	127787056.5	2013.0	876
Adventure,Drama,Sci-Fi	39.0	136500000.0	208225778.5	2014.5	717
Action,Mystery,Sci-Fi	60.0	34000000.0	102427862.0	2014.0	684

In [63]:
```
top_ten_net_theNums.reset_index(inplace=True)
```

In [64]:
```
top_ten_net_theNums.drop(['id','year'], axis=1, inplace=True)
```

```
C:\Users\i3scr\anaconda3\envs\learn-env\lib\site-packages\pandas\core\fra
me.py:4163: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
```

In [65]:
```
top_ten_net_theNums
```

Out[65]:

	genres	production_budget_int	domestic_gross_int	domestic_net
0	Family,Fantasy,Musical	160000000.0	504014165.0	344014165.0
1	Adventure,Drama,Western	35000000.0	171243005.0	136243005.0
2	Fantasy,Musical	200000000.0	334191110.0	134191110.0
3	Drama,Family,Fantasy	95000000.0	201151353.0	106151353.0
4	Animation,Comedy,Family	69600000.0	175028795.2	105428795.2
5	Action,Comedy,Documentary	20000000.0	117229692.0	97229692.0
6	Biography,Drama,Musical	84000000.0	174340174.0	90340174.0
7	Comedy,Mystery	40100000.0	127787056.5	87687056.5
8	Adventure,Drama,Sci-Fi	136500000.0	208225778.5	71725778.5
9	Action,Mystery,Sci-Fi	34000000.0	102427862.0	68427862.0

Genre by Rating (IMDB data only)

From the IMDB SQL database, we will join the movie_basics and movie_ratings tables, ensuring only relevant data is present by filtering by release date, dropping entries with null values, and using a limiting factor of at least 300 movies per genre to be considered for analysis.

In [66]:

```
query = """
SELECT genres, AVG(averagerating) AS averagerating_of_genre, SUM(numvotes)
FROM movie_basics
JOIN movie_ratings
    USING(movie_id)
WHERE genres IS NOT NULL and start_year < 2024
GROUP BY genres
HAVING num_entries >=300
ORDER BY averagerating_of_genre DESC
"""

ratings_by_genre_df = pd.read_sql(query, conn)
ratings_by_genre_df  #This is the DataFrame we will use for analysis on ge
```

Out[66]:

	genres	averagerating_of_genre	total_num_votes	num_entries
0	Documentary,Sport	7.500000	114731	318
1	Biography,Documentary,Drama	7.498674	74907	377
2	Documentary,Music	7.478756	393048	579
3	Biography,Documentary,History	7.454071	128555	479
4	Documentary,History	7.389916	97319	476
5	Documentary,Drama	7.332818	141267	582
6	Documentary	7.293794	1785513	10313
7	Biography,Documentary	7.221758	200663	694
8	Drama,Family	6.651464	253346	478
9	Drama	6.494265	8395521	11612
10	Comedy,Drama,Family	6.404180	437322	311
11	Crime,Drama	6.375101	2991931	494
12	Comedy,Drama	6.364119	6462839	2617
13	Drama,Romance	6.294305	5542760	1510
14	Comedy,Drama,Romance	6.292467	7665463	1208
15	Crime,Drama,Thriller	6.128968	3362672	504
16	Drama,Thriller	6.118485	3879354	990
17	Drama,Mystery,Thriller	6.078387	2171057	310
18	Family	6.078004	50028	491
19	Romance	6.051883	139633	717
20	Action,Drama	6.045316	762313	395
21	Action,Crime,Drama	5.989146	5563553	562
22	Animation	5.908621	26477	348
23	Comedy,Romance	5.845631	4752398	1236
24	Comedy	5.777998	6832037	5613
25	Action	5.757712	329057	979
26	Action,Comedy	5.748936	993742	329
27	Thriller	5.704244	440564	1555
28	Action,Thriller	5.658841	4284464	345
29	Comedy,Horror	5.155959	881462	579
30	Drama,Horror,Thriller	5.147191	1019921	356
31	Horror,Mystery,Thriller	5.094444	3902882	378
32	Horror	4.835475	1585933	2692
33	Horror,Thriller	4.811554	3105816	1004

Genre by Popularity + Votes (IMDB and TMDB)

From previewing the data provided by TheMovieDatabase, we can see three columns that will be useful in analysis: `popularity` , `vote_average` , and `vote_count` . After cleaning and merging the two tables, we will find the average popularity, average 'vote average' (which we will be using synonymously with average rating per genre), and the sum of the total votes per genre.

In [67]:
```
tmdb_df.head()
```

Out[67]:

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

In [68]:
```
tmdb_df.duplicated(keep=False).value_counts()
```

Out[68]:
```
False    26517
dtype: int64
```

In [69]:
```
tmdb_df['titleClean'] = tmdb_df['original_title'].str.strip()
```

In [70]:
```
tmdb_df['year'] = tmdb_df['release_date'].map(lambda x: int(x[:4]))
```

In [71]:
```
min(tmdb_df['year'])
```

Out[71]:
```
1930
```

In [72]:
```
max(tmdb_df['year'])
```

Out[72]:
```
2020
```

In [73]:
```
tmdb_df = tmdb_df[tmdb_df['year'] >= 2003]
```

In [74]:
```
tmdb_and_IMDB_df = tmdb_df.merge(movie_basics_df, on=['titleClean', 'year'
tmdb_and_IMDB_df
```

Out[74]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_da
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-
3	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-
4	5	[12, 14, 10751]	32657	en	Percy Jackson & the Olympians: The Lightning T...	26.691	2010-02-
...	...	...	...	...	...	...	
12080	26506	[]	561861	en	Eden	0.600	2018-11-
12081	26507	[99]	545555	ar	Dreamaway	0.600	2018-10-
12082	26509	[27]	502255	en	Closing Time	0.600	2018-02-
12083	26514	[14, 28, 12]	381231	en	The Last One	0.600	2018-10-
12084	26516	[53, 27]	309885	en	The Church	0.600	2018-10-

12085 rows × 15 columns

In [75]:
```
columns_to_drop = ('Unnamed: 0', 'genre_ids', 'id', 'original_language', '
tmdb_and_IMDB_df.drop(tmdb_and_IMDB_df.loc[:,columns_to_drop], axis=1, inp
```

In [76]:
```
tmdb_and_IMDB_df.dropna(inplace=True)
```

In [77]:
```
tmdb_and_IMDB_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12008 entries, 0 to 12084
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   original_title  12008 non-null  object 
 1   popularity      12008 non-null  float64
 2   vote_average    12008 non-null  float64
 3   vote_count      12008 non-null  int64  
 4   titleClean      12008 non-null  object 
 5   year            12008 non-null  int64  
 6   genres          12008 non-null  object 
dtypes: float64(2), int64(2), object(3)
memory usage: 750.5+ KB
```

Fabulous! We now have a merged data frame that we will use to extrapolate the three points of interest we will use in our analysis. First, let's check that we understand the `popularity` column a bit more, and then move right along with creating our three tables for later analysis.

In [78]:
```
min(tmdb_and_IMDB_df['popularity'])
```

Out[78]:
```
0.6
```

In [79]:
```
max(tmdb_and_IMDB_df['popularity'])
```

Out[79]:
```
80.773
```

In [80]:
```
grouped = tmdb_and_IMDB_df.groupby('genres').mean()
grouped.reset_index(inplace=True)
```

In [81]:
```
grouped.drop('year', axis=1, inplace=True)
```

In [82]:
```
top_ten_popular = grouped.sort_values(by='popularity', ascending=False).he
top_ten_by_voteaverage = grouped.sort_values(by='vote_average', ascending=
```

In [83]:
```
grouped = tmdb_and_IMDB_df.groupby('genres').sum()
grouped.reset_index(inplace=True)
```

In [84]:
```
top_ten_by_votecount = grouped.sort_values(by='vote_count', ascending=Fals
```

Analysis

The analysis will focus on what genres of movies were most successful based on a few different exploratory questions. In sum, answering what genres of movies (1) garnered the highest gross income, (2) were most expensive to make, (3) made the most amount of net income, and (4) were the most popular.

In [85]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

1. What genres of movies make the most domestic gross income?

For this analysis we will consider only domestic gross income, rather than digging into the worldwide data, as we would like to focus the launch domestically prior to launching internationally. As we have two sources of data that includes gross domestic income we will compare both results to get a more informed analysis. This is especially helpful as the BOM data did not include any movies from the past 5 most recent years.

In [86]:
```
top_ten_gross_BOM
```

Out[86]:

	genres	domestic_gross
0	Action,Adventure,Sci-Fi	2.437244e+08
1	Adventure,Drama,Sci-Fi	2.082000e+08
2	Adventure,Fantasy	1.929000e+08
3	Biography,Drama,Musical	1.743000e+08
4	Action,Adventure,Fantasy	1.530602e+08
5	Action,Adventure,Mystery	1.509000e+08
6	Animation,Comedy,Family	1.458669e+08
7	Comedy,Music	1.446000e+08
8	Adventure,Animation,Comedy	1.413663e+08
9	Action,Drama,Family	1.310500e+08

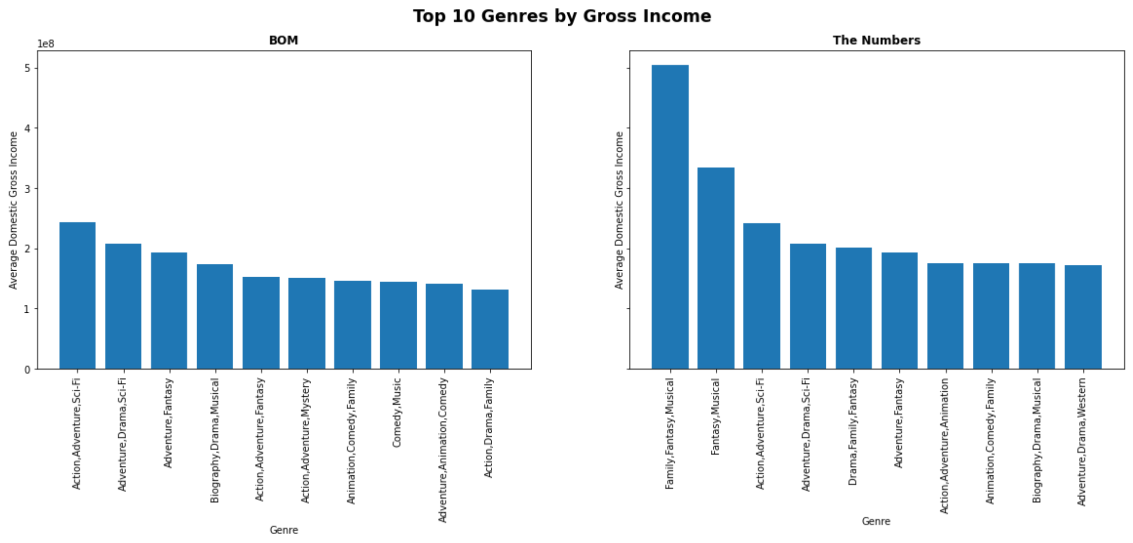
In [87]:
```
top_ten_gross_theNums
```

Out[87]:

	genres	domestic_gross_int
0	Family,Fantasy,Musical	5.040142e+08
1	Fantasy,Musical	3.341911e+08
2	Action,Adventure,Sci-Fi	2.409982e+08
3	Adventure,Drama,Sci-Fi	2.082258e+08
4	Drama,Family,Fantasy	2.011514e+08
5	Adventure,Fantasy	1.928914e+08
6	Action,Adventure,Animation	1.755641e+08
7	Animation,Comedy,Family	1.750288e+08
8	Biography,Drama,Musical	1.743402e+08
9	Adventure,Drama,Western	1.712430e+08

In [88]:
```
fig, ax = plt.subplots(ncols=2, figsize=(20,6), sharey=True)

x_var = 'genres'

ax[0].bar(top_ten_gross_BOM[x_var], height=top_ten_gross_BOM['domestic_gro
ax[0].tick_params(axis='x', labelrotation=90)
ax[0].set_xlabel('Genre')
ax[0].set_ylabel('Average Domestic Gross Income')
ax[0].set_title('BOM', fontsize='large', fontweight='bold')

ax[1].bar(top_ten_gross_theNums[x_var], height=top_ten_gross_theNums['dome
ax[1].tick_params(axis='x', labelrotation=90)
ax[1].set_xlabel('Genre')
ax[1].set_ylabel('Average Domestic Gross Income')
ax[1].set_title('The Numbers', fontsize='large', fontweight='bold')

plt.suptitle('Top 10 Genres by Gross Income', fontsize='xx-large', fontwei

;
```

Out[88]:
```
''
```



Analysis Notes

While there are not many exact overlaps of genre categories between the top ten of each dataset, it is notable that Action and Adventure are prevalent in both. It is interesting to see that the average gross income from The Numbers is higher overall; this trend can be attributed to the lack of data from the past five years. The drastic spike for the `Family, Fantasy, Musical` genre which we do not see at all in the top ten for the BOM data set is also notable.

Overall, we can see that movies that include elements of action and adventure tend to have the highest domestic gross income.

2. What genres of movies are the most expensive to

make?

While not directly related to the question 'what movies are most successful', it is important to
keep in mind the cost in making this product. We will investigate further when looking at net

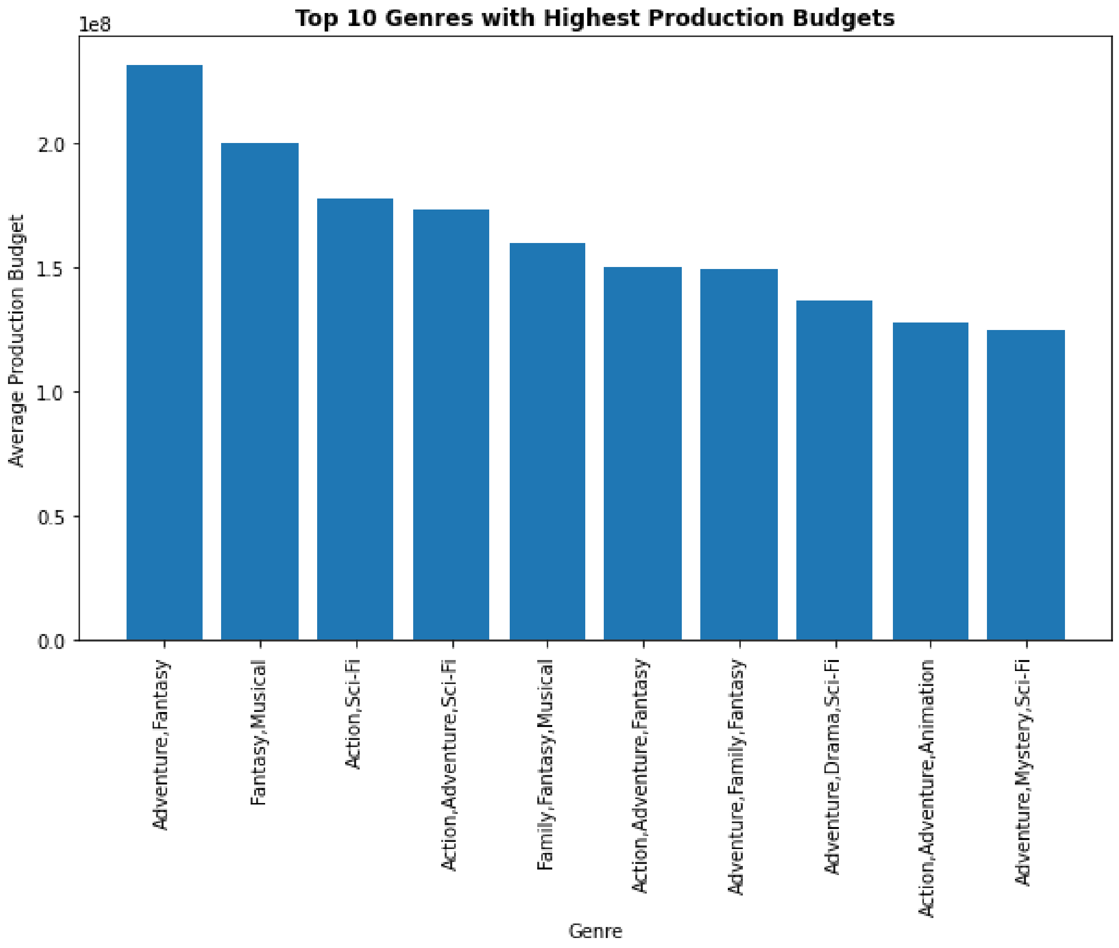
In [89]:
```
top_ten_budget_theNums
```

Out[89]:

	genres	production_budget_int
0	Adventure,Fantasy	2.316667e+08
1	Fantasy,Musical	2.000000e+08
2	Action,Sci-Fi	1.780000e+08
3	Action,Adventure,Sci-Fi	1.731615e+08
4	Family,Fantasy,Musical	1.600000e+08
5	Action,Adventure,Fantasy	1.498581e+08
6	Adventure,Family,Fantasy	1.491714e+08
7	Adventure,Drama,Sci-Fi	1.365000e+08
8	Action,Adventure,Animation	1.279333e+08
9	Adventure,Mystery,Sci-Fi	1.250000e+08

In [90]:
```
fig, ax = plt.subplots(figsize=(10,6))

ax.bar(top_ten_budget_theNums['genres'], height=top_ten_budget_theNums['pr
ax.tick_params(axis='x', labelrotation=90)
ax.set_xlabel('Genre')
ax.set_ylabel('Average Production Budget')
ax.set_title('Top 10 Genres with Highest Production Budgets', fontweight='

;
```

Out[90]:
```
''
```



Analysis Notes

From above we noted that Action and Adventure movies earned the highest average gross income, however it appears that they also tend to be the most expensive to create, which could be challenging for a new company and could possibly hinder overall net profits.

The other notable genre we see is Science Fiction, though it is not surprising that movies that fall into this overarching genre would be more expensive to create.

3. What genres of movies make the most amount of

profit?

Based on the data gained from IMDB and The Numbers, we can see the top ten movie genres

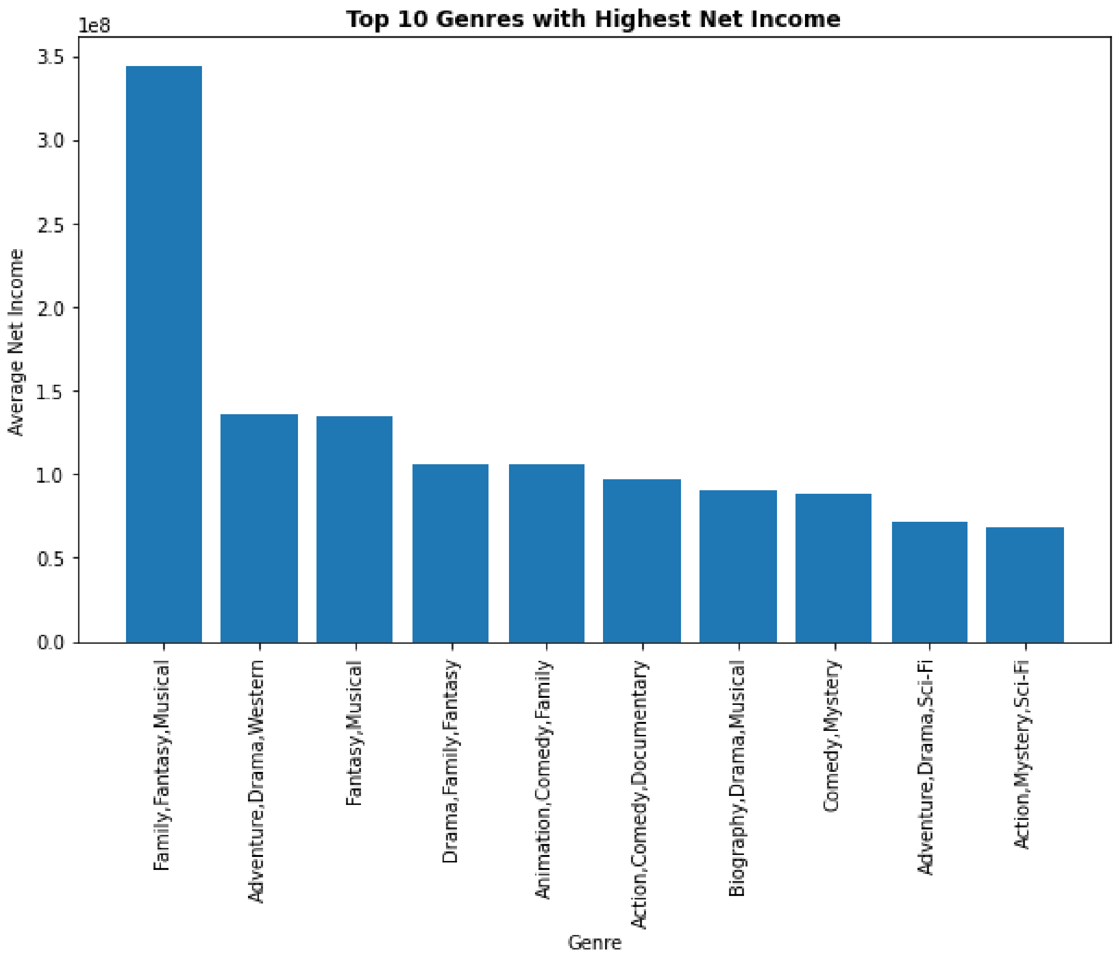
In [91]:
```
top_ten_net_theNums
```

Out[91]:

	genres	production_budget_int	domestic_gross_int	domestic_net
0	Family,Fantasy,Musical	160000000.0	504014165.0	344014165.0
1	Adventure,Drama,Western	35000000.0	171243005.0	136243005.0
2	Fantasy,Musical	200000000.0	334191110.0	134191110.0
3	Drama,Family,Fantasy	95000000.0	201151353.0	106151353.0
4	Animation,Comedy,Family	69600000.0	175028795.2	105428795.2
5	Action,Comedy,Documentary	20000000.0	117229692.0	97229692.0
6	Biography,Drama,Musical	84000000.0	174340174.0	90340174.0
7	Comedy,Mystery	40100000.0	127787056.5	87687056.5
8	Adventure,Drama,Sci-Fi	136500000.0	208225778.5	71725778.5
9	Action,Mystery,Sci-Fi	34000000.0	102427862.0	68427862.0

In [92]:
```
fig, ax = plt.subplots(figsize=(10,6))

ax.bar(top_ten_net_theNums['genres'], height=top_ten_net_theNums['domestic
ax.tick_params(axis='x', labelrotation=90)
ax.set_xlabel('Genre')
ax.set_ylabel('Average Net Income')
ax.set_title('Top 10 Genres with Highest Net Income', fontweight='bold')

;
```

Out[92]:
```
''
```



Analysis Notes

Aside from our outlier, we can see that the other top nine genres are all within the same approximate range of average net income. We do see Action and Adventure included in these genres which leads to the conclusion that even though they are among the most expensive to create, the payoff could be worth the endeavor.

The other notable genres are Drama and Comedy. Both were not in the most expensive categories to create (assuming the `Adventure, Drama, Sci-Fi` genre within our top ten most expensive was due to the adventure and science-fiction aspects rather than the drama aspect), and were both present in the top ten highest domestic income charts.

Overall, the company can pursue (a) `Family, Fantasy, Musical` types of movies in the hopes of competing with the giants in that specific industry (ie Disney, Pixar, etc), (b) creating action or adventure movies if the starting budgets are ample enough, or (c) starting off with creating movies in the drama or comedy field. Or of course, assigning teams to each category

4. What genres of movies are the most popular?

We will be utilizing the `ratings_by_genre_df` which includes data only from IMDB, as well as the data from TMDB and IMDB to gauge popularity. We will rate popularity by the average rating of the genre, but we will also look at the genres with the greatest interaction based on number of votes. We will also consider the genres with the greatest number of movies in that category, as well as the 'popularity' rating from the IMDB data and TMDB respectively.

In [93]:
```
top_ten_by_averagerating = ratings_by_genre_df.sort_values(by=['averagerat
```

In [94]:
```
top_ten_by_averagerating
```

Out[94]:

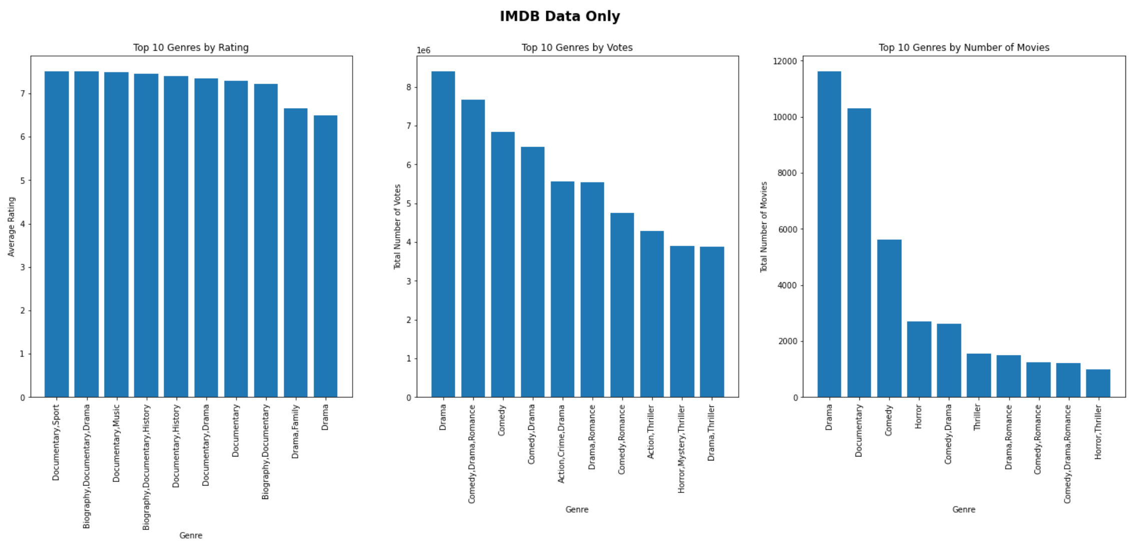
	genres	averagerating_of_genre	total_num_votes	num_entries
0	Documentary,Sport	7.500000	114731	318
1	Biography,Documentary,Drama	7.498674	74907	377
2	Documentary,Music	7.478756	393048	579
3	Biography,Documentary,History	7.454071	128555	479
4	Documentary,History	7.389916	97319	476
5	Documentary,Drama	7.332818	141267	582
6	Documentary	7.293794	1785513	10313
7	Biography,Documentary	7.221758	200663	694
8	Drama,Family	6.651464	253346	478
9	Drama	6.494265	8395521	11612

In [95]:
```
top_ten_by_votes = ratings_by_genre_df.sort_values(by=['total_num_votes'],
```

In [96]:
```
top_ten_by_entries = ratings_by_genre_df.sort_values(by=['num_entries'], a
```

In [97]:
```
fig, ax = plt.subplots(ncols=3, figsize=(25,8))

x_var = 'genres'

ax[0].bar(top_ten_by_averagerating[x_var], height=top_ten_by_averagerating
ax[0].tick_params(axis='x', labelrotation=90)
ax[0].set_xlabel('Genre')
ax[0].set_ylabel('Average Rating')
ax[0].set_title('Top 10 Genres by Rating')

ax[1].bar(top_ten_by_votes[x_var], height=top_ten_by_votes['total_num_vote
ax[1].tick_params(axis='x', labelrotation=90)
ax[1].set_xlabel('Genre')
ax[1].set_ylabel('Total Number of Votes')
ax[1].set_title('Top 10 Genres by Votes')

ax[2].bar(top_ten_by_entries[x_var], height=top_ten_by_entries['num_entrie
ax[2].tick_params(axis='x', labelrotation=90)
ax[2].set_xlabel('Genre')
ax[2].set_ylabel('Total Number of Movies')
ax[2].set_title('Top 10 Genres by Number of Movies')

plt.suptitle('IMDB Data Only', fontsize='xx-large', fontweight='heavy')
;
```

Out[97]:
```
''
```



In [98]:
```
top_ten_popular
```

Out[98]:

	genres	popularity	vote_average	vote_count
155	Adventure,Fantasy,Mystery	33.533000	7.700000	10788.000000
478	Family,Fantasy,Musical	31.793000	6.900000	11023.000000
76	Action,Fantasy,War	23.680000	6.000000	3870.000000
14	Action,Adventure,Sci-Fi	19.238205	5.779487	5619.217949
12	Action,Adventure,Mystery	17.279667	6.266667	5294.000000
9	Action,Adventure,Fantasy	16.982544	5.571930	3702.087719
75	Action,Fantasy,Thriller	16.387000	5.050000	1499.000000
15	Action,Adventure,Thriller	16.384680	5.788000	2770.360000
231	Biography,Drama,Musical	16.378000	7.266667	3485.666667
147	Adventure,Family,Fantasy	16.310800	5.925000	3663.800000

In [99]:
```
top_ten_by_voteaverage
```

Out[99]:

	genres	popularity	vote_average	vote_count
237	Biography,Family,History	0.918	10.0	1.0
196	Animation,Short	0.600	10.0	1.0
422	Drama,Family,War	0.600	10.0	1.0
80	Action,History,War	3.892	10.0	1.0
379	Documentary,Fantasy,History	0.600	10.0	1.0
107	Adventure,Biography	1.001	9.5	4.0
154	Adventure,Fantasy,Musical	0.878	9.0	1.0
204	Biography,Comedy,Music	0.661	9.0	2.0
371	Documentary,Drama,Thriller	0.600	9.0	1.0
223	Biography,Documentary,War	0.600	9.0	2.0

In [100]:

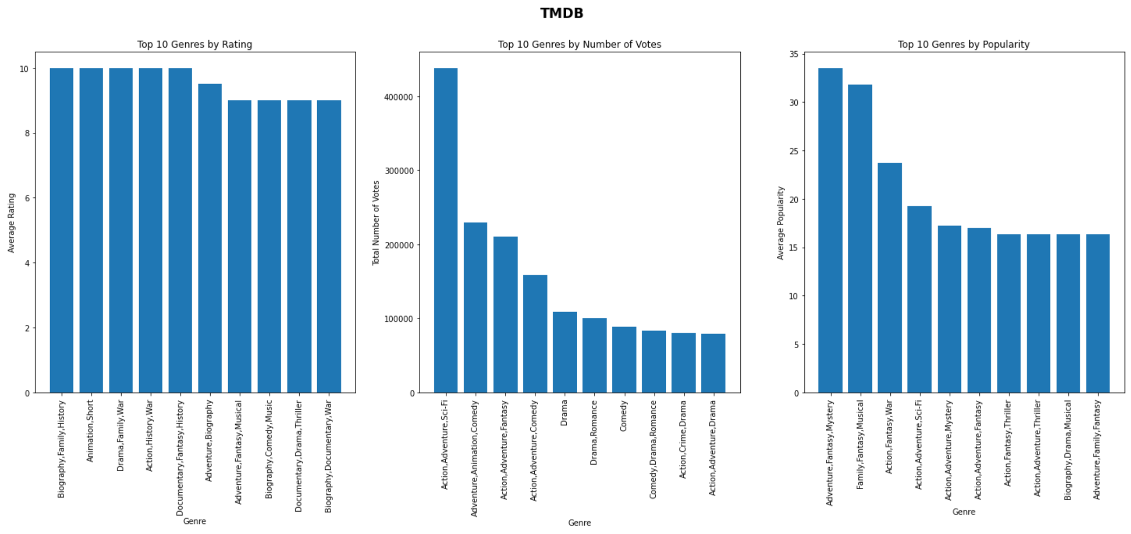
```
top_ten_by_votecount
```

Out[100]:

	genres	popularity	vote_average	vote_count	year
14	Action,Adventure,Sci-Fi	1500.580	450.8	438299	157145
102	Adventure,Animation,Comedy	1365.176	619.7	229345	199446
9	Action,Adventure,Fantasy	968.005	317.6	211019	114831
4	Action,Adventure,Comedy	823.795	421.2	158898	155165
412	Drama	2593.841	6096.5	108956	2066356
461	Drama,Romance	1175.639	1475.0	100040	495410
244	Comedy	1889.150	3044.9	89167	1166083
272	Comedy,Drama,Romance	1298.623	1708.3	83806	573951
44	Action,Crime,Drama	1053.759	790.0	80451	280015
7	Action,Adventure,Drama	672.362	391.4	79499	147049

In [101]:
```
fig, ax = plt.subplots(ncols=3, figsize=(25,8))

x_var = 'genres'

ax[0].bar(top_ten_by_voteaverage[x_var], height=top_ten_by_voteaverage['vo
ax[0].tick_params(axis='x', labelrotation=90)
ax[0].set_xlabel('Genre')
ax[0].set_ylabel('Average Rating')
ax[0].set_title('Top 10 Genres by Rating')

ax[1].bar(top_ten_by_votecount[x_var], height=top_ten_by_votecount['vote_c
ax[1].tick_params(axis='x', labelrotation=90)
ax[1].set_xlabel('Genre')
ax[1].set_ylabel('Total Number of Votes')
ax[1].set_title('Top 10 Genres by Number of Votes')

ax[2].bar(top_ten_popular[x_var], height=top_ten_popular['popularity'])
ax[2].tick_params(axis='x', labelrotation=90)
ax[2].set_xlabel('Genre')
ax[2].set_ylabel('Average Popularity')
ax[2].set_title('Top 10 Genres by Popularity')

plt.suptitle('TMDB', fontsize='xx-large', fontweight='heavy')
;
```

Out[101]:
```
''
```



Analysis

We have ratings and total sum of votes from two different sources, which shows some similarities and differences. It is interesting that the top ten average ratings are all around 7 from the IMDB data, and closer to 10 from the TMDB data. However, the general trend and rate of decrease between the top ten are about the same in each set. We can see that both of these charts include Action, Adventure, and Drama as aspects in the top ten genres by average rating.

For total number of votes by genre we have an outlier from the TMDB data -- the `Action, Adventure, Sci-Fi` genre is drastically higher than the rest of the genres, and not at all present in the IMDB chart for total votes. We could infer that science-fiction fans are very active on one site but not the other, but further investigation most definitely needs to be done. Aside from this one outlier, we do see similar genres that people are interactive with via their votes: Adventure, Action, and Drama are consistent aspects of these genres.

Lastly, we have the sum of movies per genre as well as the average 'popularity' from the IMDB and TMDB data respectively. It is notable that Drama and Documentary are the largest categories simply in the number of movies made. For the number of movies made, it is notable that Drama and Documentary genres have so many entries above all other genres, even the third most robust genre, Comedy, which is notably higher than the rest of the genres. From the popularity data we continue to see Action and Adventure prevalent in the top ten results.

Conclusion

Putting the results all together, there are a few different directions Microsoft could direct their movie studio to. The surest best seems to be to pursue movies in the action and adventure genres. While they are on the higher end of the budget spectrum, movies with these two categories incorporated in their genres have been popular and made profits over the past 20 years.

They could also look into creating dramas, as they would cost less money to make, would still make decent profit and gain traction and interaction with audiences as they tend to be quite popular. Although they would be competing with a wide scope of movies already in this genre.

If interested in more of a niche market, Documentaries would be the way to go as they would be generally very well received by viewers, and would not be as expensive to create, even if profits would also be less as well.

Lastly, Microsoft could pursue the family-friendly types of movies. The positives of going in this direction is that these movies tend to be very popular, and have the potential to garner the highest net income by leaps and bounds. However, this would also be a rather competitive market, going up against companies like Disney and Pixar that have, and continue to dominate the market.

Next Steps

Further research and planning of course must be completed prior to the launch of this new venture. Other than just a genre study, information on what directors and writers have been successful would be imperative to see who to potentially hire. Furthermore, investigation on trends over time would be helpful, both in identifying trends genres that do well when released in a specific time of year (such as releasing a holiday themed drama in November or December rather than in March or June), and in historical trends over the years. It would also be good to dig into more recent data as well, looking only at the past five years rather than averages over the past 20. Lastly, investigating how all of these results change when talking on an