Exploratory data analysis

Overview

This project will utilize movie data from various sources to create recommendations for a company to take to create a movie studio. It will draw findings from analysis to determine the best metrics to focus on for the recommendations.

In [2]:
```
# import needed libraries
import pandas as pd
import sqlite3
```

Data Sources

- Box Office Mojo
- Rotten Tomatoes
- The Movie Database
- The Numbers
- IMDB

Let's go through each dataset and see what we have.

BOX OFFICE MOJO

In [3]:
```
bom_movie_gross_df = pd.read_csv('../zippedData/bom.movie_gross.csv.gz', compression = 'gzip')
```

In [4]:
```
bom_movie_gross_df.head()
```

Out[4]:

	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 [5]:
```
bom_movie_gross_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
```

In [6]:
```
bom_movie_gross_df.describe()
```

Out[6]:

	domestic_gross	year
count	3.359000e+03	3387.000000
mean	2.874585e+07	2013.958075
std	6.698250e+07	2.478141
min	1.000000e+02	2010.000000
25%	1.200000e+05	2012.000000
50%	1.400000e+06	2014.000000
75%	2.790000e+07	2016.000000
max	9.367000e+08	2018.000000

There are 3387 records in this dataframe and each represent a movie with title, studio, domestic gross, foriegn gross, and year it was released.

columns:

- title: movie title
- studio: name of the studio
- domestic gross: money made domestically
- foreign gross: money made foreignly
  - third are nulls
- year: year movie was released

In [7]:
```
bom_movie_gross_df.isna().sum()
```

Out[7]:
```
title              0
studio             5
domestic_gross    28
foreign_gross    1350
year               0
dtype: int64
```

In [8]:
```
bom_movie_gross_df = bom_movie_gross_df.dropna()
```

In [9]:
```
bom_movie_gross_df.isna().sum()
```

Out[9]:
```
title             0
studio            0
domestic_gross    0
foreign_gross     0
year              0
dtype: int64
```

In [10]:
```
bom_movie_gross_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2007 entries, 0 to 3353
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           2007 non-null   object 
 1   studio          2007 non-null   object 
 2   domestic_gross  2007 non-null   float64
 3   foreign_gross   2007 non-null   object 
 4   year            2007 non-null   int64  
```

dtypes: float64(1), int64(1), object(3)
memory usage: 94.1+ KB

This clean data set consists of 2007 records.

ROTTEN TOMATOES

In [11]:
```
rt_movie_info = pd.read_csv('../zippedData/rt.movie_info.tsv.gz', compression = 'gzip', sep = '\t')
```

In [12]:
```
rt_movie_info.head()
```

Out[12]:

	id	synopsis	rating	genre	director	writer	theater_date	dvd_date	cur
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971	Sep 25, 2001	
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo	Aug 17, 2012	Jan 1, 2013	
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996	Apr 18, 2000	
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton	Dec 9, 1994	Aug 27, 1997	
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	NaN	NaN	

In [13]:
```
rt_movie_info.shape
```

Out[13]: (1560, 12)

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

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

The Rotten Tomatoes movie info dataset includes 1560 records where each record represents a movie, with id, rating, director, writer, box office, and studio as some of the columns.

There seem to be lots of nulls in important columns such as studio and box office so may not use this dataset.

In [15]:
```
rt_reviews = pd.read_csv('../zippedData/rt.reviews.tsv.gz', compression = 'gzip', sep = '\t', encoding = 'latin') # encoding is utf8?
```

In [16]:
```
rt_reviews.head()
```

Out[16]:

	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 [17]:
```
rt_reviews.shape
```

Out[17]:
```
(54432, 8)
```

In [18]:
```
rt_reviews.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
```

The Rotten Tomatoes reviews dataset includes 54432 records where each record represents a movie.

The columns include id, review, rating, fresh, critic, top critic, publisher, and date.

A better measure of success is box office payout so ratings won't be a specific target metric we will focus on.

THE MOVIE DB

In [19]:
```
tmdb_movies = pd.read_csv('../zippedData/tmdb.movies.csv.gz', compression = 'gzip', index_col  = 0 )
```

In [20]:
```
tmdb_movies.head()
```

Out[20]:

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average	vote_
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	7.7	
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon	7.7	
2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2	6.8	
3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story	7.9	
4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception	8.3	

In [21]:
```
tmdb_movies.shape
```

Out[21]:
```
(26517, 9)
```

In [22]:
```
tmdb_movies.info() # no nulls
```

```
<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
```

The Movie DB dataset includes 26,517 records where each represents a movie.

There are columns of id, genre ids, original language, original title, popularity, release date, title, vote average and vote count.

There are no nulls in this dataset

THE NUMBERS

In [23]:
```
tn_movie_budgets = pd.read_csv('../zippedData/tn.movie_budgets.csv.gz', compression = 'gzip')
```

In [24]:
```
tn_movie_budgets.head()
```

Out[24]:

	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 [25]:
```
tn_movie_budgets.shape
```

Out[25]:
```
(5782, 6)
```

In [26]:
```
tn_movie_budgets.info() # no nulls
```

```
<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
```

In [27]:
```
tn_movie_budgets.describe()
```

Out[27]:

	id
count	5782.000000
mean	50.372363
std	28.821076
min	1.000000
25%	25.000000
50%	50.000000
75%	75.000000
max	100.000000

The Numbers dataset includes 5782 records where each record represents a movie.

The columns include id, release_date, movie, production_budget, domestic_gross and worldwide_gross.

We can utilize the production budget and gross columns to determine profit.

INTERNET MOVIE DATABASE: IMDB

In [28]:
```
conn = sqlite3.connect('../zippedData/im.db')
```

In [29]:
```
%%bash
sqlite3 /Users/nechamaborisute/Desktop/phase-2-project/zippedData/im.db

.tables
```

```
box_office_mojo   movie_akas       persons         writers
directors         movie_basics     principals
known_for         movie_ratings    tn_movie_budgets
```

In [30]:
```
schema = pd.read_sql("""

SELECT *
FROM sqlite_schema

""", conn)
```

In [31]:
```
schema
```

Out[31]:

	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 ...
8	table	box_office_mojo	box_office_mojo	41369	CREATE TABLE "box_office_mojo" (\n"index" INTE...
9	index	ix_box_office_mojo_index	box_office_mojo	41370	CREATE INDEX "ix_box_office_mojo_index"ON "box...
10	table	tn_movie_budgets	tn_movie_budgets	45071	CREATE TABLE "tn_movie_budgets" (\n"index" INT...
11	index	ix_tn_movie_budgets_index	tn_movie_budgets	45072	CREATE INDEX "ix_tn_movie_budgets_index"ON "tn...

In [32]:
```
# table names
pd.read_sql("""SELECT name FROM sqlite_master WHERE type = 'table';""", conn)
```

Out[32]:

	name
0	movie_basics
1	directors
2	known_for
3	movie_akas
4	movie_ratings
5	persons
6	principals
7	writers
8	box_office_mojo
9	tn_movie_budgets

In [33]:
```
pd.read_sql("""

SELECT *
FROM movie_basics
LIMIT 5

""", conn)
```

Out[33]:

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

In [34]:
```
pd.read_sql("""

SELECT *
FROM movie_basics
WHERE genres IS NULL

""", conn)
```

Out[34]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0187902	How Huang Fei-hong Rescued the Orphan from the...	How Huang Fei-hong Rescued the Orphan from the...	2011	NaN	None
1	tt0253093	Gangavataran	Gangavataran	2018	134.0	None
2	tt0306058	Second Coming	Second Coming	2012	95.0	None
3	tt0326592	The Overnight	The Overnight	2010	88.0	None
4	tt0330811	Regret Not Speaking	Regret Not Speaking	2011	NaN	None
...	...	...	...	...	...	...

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
5403	tt9907396	Footloose in the Cotswolds - Part 1	Footloose in the Cotswolds - Part 1	2016	118.0	None
5404	tt9907608	Footloose in the Cotswolds - Part 2	Footloose in the Cotswolds - Part 2	2016	102.0	None
5405	tt9910922	Doctor Who Augmented Reality: Times Magazine	Doctor Who Augmented Reality: Times Magazine	2013	NaN	None
5406	tt9914942	La vida sense la Sara Amat	La vida sense la Sara Amat	2019	NaN	None
5407	tt9916730	6 Gunn	6 Gunn	2017	116.0	None

5408 rows × 6 columns

In [35]:
```
pd.read_sql("""

SELECT COUNT(*)
FROM movie_basics
WHERE genres IS NULL


""", conn)
```

Out[35]:

	COUNT(*)
0	5408

In [36]:
```
pd.read_sql("""

SELECT COUNT(*) AS title_null_count
FROM movie_basics
WHERE primary_title IS NULL

""", conn)
```

Out[36]:

	title_null_count
0	0

- columns include movie id, primary title, original title, start year, runtime minutes and genres.

> might drop these rows where genre is null since we need the genre unless we could combine with other dataframes to get the genre runtime minutes we could recommend how long the movie should be

In [37]:
```
pd.read_sql("""

SELECT *
FROM directors
LIMIT 5

""", conn)
```

Out[37]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0462036	nm1940585

	movie_id	person_id
2	tt0835418	nm0151540
3	tt0835418	nm0151540
4	tt0878654	nm0089502

In [38]:
```
pd.read_sql("""

SELECT COUNT(*) AS person_id_null_count
FROM directors
WHERE person_id IS NULL

""", conn)
```

Out[38]:

	person_id_null_count
0	0

- the director table includes movie id and person id
- no nulls in director table

In [39]:
```
pd.read_sql("""

SELECT *
FROM known_for

""", conn).head()
```

Out[39]:

	person_id	movie_id
0	nm0061671	tt0837562
1	nm0061671	tt2398241
2	nm0061671	tt0844471
3	nm0061671	tt0118553
4	nm0061865	tt0896534

In [40]:
```
pd.read_sql("""

SELECT COUNT(*) AS movie_id_null_count
FROM known_for
WHERE movie_id IS NULL

""", conn).head()
```

Out[40]:

	movie_id_null_count
0	0

In [41]:
```
pd.read_sql("""

SELECT COUNT(*) AS person_id_null_count
FROM known_for
WHERE person_id IS NULL

""", conn).head()
```

Out[41]:

	person_id_null_count
0	0

- in known for table there are no nulls and it has a person id in one column paired with the movie id of the movie their most known for

In [42]:
```
pd.read_sql("""

SELECT *
FROM movie_akas
LIMIT 5

""", conn).head()
```

Out[42]:

	movie_id	ordering	title	region	language	types	attributes	is_original_title
0	tt0369610	10	Джурасик свят	BG	bg	None	None	0.0
1	tt0369610	11	Jurashikku warudo	JP	None	imdbDisplay	None	0.0
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	None	imdbDisplay	None	0.0
3	tt0369610	13	O Mundo dos Dinossauros	BR	None	None	short title	0.0
4	tt0369610	14	Jurassic World	FR	None	imdbDisplay	None	0.0

In [43]:
```
pd.read_sql("""

SELECT COUNT(*) types_null_count
FROM movie_akas
WHERE types IS NULL

""", conn).head()
```

Out[43]:

	types_null_count
0	163256

In [44]:
```
pd.read_sql("""

SELECT COUNT(*) attrib_null_count
FROM movie_akas
WHERE attributes IS NULL

""", conn).head()
```

Out[44]:

	attrib_null_count
0	316778

- the movie aka table contiains movie id, ordering, title, region, language, types, attributes, is original title
- has bunch of nulls

In [45]:
```
pd.read_sql("""

SELECT *
FROM movie_ratings
```

```
""", conn).head()
```

Out[45]:

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

In [46]:
```
pd.read_sql("""

SELECT COUNT(*) AS avgrating_null_count
FROM movie_ratings
WHERE averagerating IS NULL

""", conn).head()
```

Out[46]:

	avgrating_null_count
0	0

In [47]:
```
pd.read_sql("""

SELECT COUNT(*) AS numvotes_null_count
FROM movie_ratings
WHERE numvotes IS NULL

""", conn).head()
```

Out[47]:

	numvotes_null_count
0	0

In [48]:
```
pd.read_sql("""

SELECT COUNT(*) AS id_null_count
FROM movie_ratings
WHERE movie_id IS NULL

""", conn).head()
```

Out[48]:

	id_null_count
0	0

- the movie ratings table contains movie id, average rating and num votes
- no nulls

In [49]:
```
pd.read_sql("""

SELECT *
FROM persons
LIMIT 5

""", conn)
```

Out[49]:

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

- the persons table contains person id, primary name, birth year, death year and primary profession.
- lots of nulls in birth and death columns

In [50]:
```
pd.read_sql("""

SELECT *
FROM principals
LIMIT 5

""", conn)
```

Out[50]:

	movie_id	ordering	person_id	category	job	characters
0	tt0111414	1	nm0246005	actor	None	["The Man"]
1	tt0111414	2	nm0398271	director	None	None
2	tt0111414	3	nm3739909	producer	producer	None
3	tt0323808	10	nm0059247	editor	None	None
4	tt0323808	1	nm3579312	actress	None	["Beth Boothby"]

In [51]:
```
pd.read_sql("""

SELECT COUNT(*)
FROM principals
WHERE characters IS NULL

""", conn)
```

Out[51]:

	COUNT(*)
0	634826

- the principals table contains movie id, ordering, person id, category, job and characters.
- lots of nulls in job and characters columns

In [52]:
```
pd.read_sql("""

SELECT *
FROM writers AS w
LIMIT 5

""", conn)
```

Out[52]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0438973	nm0175726

	movie_id	person_id
2	tt0438973	nm1802864
3	tt0462036	nm1940585
4	tt0835418	nm0310087

In [53]:
```
pd.read_sql("""

SELECT COUNT (*) writer_null_count
FROM writers AS w
WHERE person_id IS NULL

""", conn)
```

Out[53]:

	writer_null_count
0	0

In [54]:
```
pd.read_sql("""

SELECT COUNT (*) movie_id_null_count
FROM writers AS w
WHERE movie_id IS NULL

""", conn)
```

Out[54]:

	movie_id_null_count
0	0

- writers table contains movie id and person id
- no nulls

The IMDB database is a relatively clean database with tables movie_basics, directors, known_for, movie_akas, movie_ratings, persons, principals and writers.

In [55]:
```
pd.read_sql("""SELECT name FROM sqlite_master WHERE type = 'table';""", conn)
```

Out[55]:

	name
0	movie_basics
1	directors
2	known_for
3	movie_akas
4	movie_ratings
5	persons
6	principals
7	writers
8	box_office_mojo
9	tn_movie_budgets

Add dataframes we're going to use into the database.

In [ ]:
```
bom_movie_gross_df.to_sql("box_office_mojo", conn)
tn_movie_budgets.to_sql("tn_movie_budgets", conn)
```

In [57]:
```
pd.read_sql("""
SELECT *
FROM box_office_mojo
""", conn)
```

Out[57]:

	index	title	studio	domestic_gross	foreign_gross	year
0	0	Toy Story 3	BV	415000000.0	652000000	2010
1	1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
2	2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
3	3	Inception	WB	292600000.0	535700000	2010
4	4	Shrek Forever After	P/DW	238700000.0	513900000	2010
...	...	...	...	...	...	...
2002	3275	I Still See You	LGF	1400.0	1500000	2018
2003	3286	The Catcher Was a Spy	IFC	725000.0	229000	2018
2004	3309	Time Freak	Grindstone	10000.0	256000	2018
2005	3342	Reign of Judges: Title of Liberty - Concept Short	Darin Southa	93200.0	5200	2018
2006	3353	Antonio Lopez 1970: Sex Fashion & Disco	FM	43200.0	30000	2018

2007 rows × 6 columns

In [60]:
```
pd.read_sql("""

SELECT DISTINCT studio
FROM box_office_mojo

""", conn)
```

Out[60]:

	studio
0	BV
1	WB
2	P/DW
3	Sum.
4	Par.
...	...
167	Blue Fox
168	Aviron
169	VE
170	Grindstone
171	Darin Southa

172 rows × 1 columns

In [61]:
```
pd.read_sql("""
SELECT COUNT(*)
FROM box_office_mojo
WHERE foreign_gross IS NULL

""", conn)
```

Out[61]:

	COUNT(*)
0	0

In [62]:
```
pd.read_sql("""

SELECT *
FROM tn_movie_budgets
LIMIT 15
""", conn)
```

Out[62]:

	index	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	0	1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
3	3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
4	4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747
5	5	6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	$306,000,000	$936,662,225	$2,053,311,220
6	6	7	Apr 27, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,048,134,200
7	7	8	May 24, 2007	Pirates of the Caribbean: At Worldâ€ s End	$300,000,000	$309,420,425	$963,420,425
8	8	9	Nov 17, 2017	Justice League	$300,000,000	$229,024,295	$655,945,209
9	9	10	Nov 6, 2015	Spectre	$300,000,000	$200,074,175	$879,620,923
10	10	11	Jul 20, 2012	The Dark Knight Rises	$275,000,000	$448,139,099	$1,084,439,099
11	11	12	May 25, 2018	Solo: A Star Wars Story	$275,000,000	$213,767,512	$393,151,347
12	12	13	Jul 2, 2013	The Lone Ranger	$275,000,000	$89,302,115	$260,002,115
13	13	14	Mar 9, 2012	John Carter	$275,000,000	$73,058,679	$282,778,100
14	14	15	Nov 24, 2010	Tangled	$260,000,000	$200,821,936	$586,477,240

In [63]:
```
conn.close()
```

Summary

As we can see, some these datasets include very useful information that we can use to draw helpful insights from, yet some are beyond the scope of the metrics we wish to focus on. For that reason, in the final notebook we will be analyzing and visualizing the data and drawing subsequent conclusions from the relevant datasets.