Final Project Submission

Please fill out:

- Student name: Greg Osborne
- Student pace: part time
- Scheduled project review date/time: May 20th 2:00 pm CT
- Instructor name: Claude Fried
- Blog post URL: https://medium.com/@gregosborne (https://medium.com/@gregosborne)

Business Case

Microsoft's strong representation in the fields of personal computing, business computing, and interactive entertainment needs to expand into other forms of popular media to stay competitive and attract new customers. Microsoft wishes to investigate the best way to expand into the blockbuster movie industry.

To break in to this business, Microsoft faces competition from the five titans of the film industry:

1. Walt Disney/20th Century Fox
2. Warner Brothers
3. Sony/Columbia
4. Universal Studios
5. Paramount Pictures

Microsoft commissioned Data Science firm FurPig Inc. to recommend best practices to succeed in this new initiative.

FurPig Inc. assigned Data Scientist Greg Osborne (me) to the project.

Microsoft provided me with a zip file containing database information detailing several different characteristics from thousands of films. My job is to go through the data and make the best recommendations based on three business questions.

Dependent Variable

There are two reasons for this initiative:

1. Make a profit on the films
2. Build a reputation with the public of quality entertainment for Microsoft's brand

Microsoft is not looking to create fodder for the pretentious crowd, generating buzz at film festivals and ignored by the public upon wider release. Microsoft wants its new movie brand—Working title: XBoxOffice—to be recognized by as many people as possible.

Since this is new a initiative for Microsoft, and the whole point is getting as many people to see these films as possible, the driving metric for our analysis will be box office dollars, because that's the metric that's closest to ticket sales.

Our dependent variable will be box office dollars.

Independent Variables

Microsoft contracted FurPig to select three independent variables—that is, three decisions that Microsoft would have complete control over during the selection of which films to produce—and see how changing these three variables yields different box office revenue.

How can we change these variables to make box office revenue as large as possible?

The three variables selected by FurPig are:

1. Genre
2. Release month
3. Creative Personnel (Writer, Director, Producer, Actor, Actress)

Business Questions

Greg selected the following business questions:

1. What genres of film produce the highest box office revenue?
2. What is the best month to release a film to generate the most revenue?
3. What writers, directors, producers, actors and actresses have the highest revenue earning potential?

Python Libraries

The first thing I'll do is import the libraries I need for this project.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import shutil
import sqlite3
import seaborn as sns
import matplotlib.patches as mpatches
pd.set_option("display.max_columns", None)
```

Data Importation

Now, I'll import the data I was given in the zip file

In [2]:
```
ls zippedData
```

```
 Volume in drive C is OS
 Volume Serial Number is 4EE7-277F

 Directory of C:\Users\g_osb\Clones\Phase1\dsc-phase-1-project-v2-4\zippedData

05/19/2022  01:15 PM    <DIR>          .
06/09/2022  11:11 PM    <DIR>          ..
05/08/2022  11:01 PM            53,544 bom.movie_gross.csv.gz
05/08/2022  11:01 PM        67,149,708 im.db.zip
05/08/2022  11:01 PM           107,563 movie_data_erd.jpeg
05/08/2022  11:01 PM           498,202 rt.movie_info.tsv.gz
05/08/2022  11:01 PM         3,402,194 rt.reviews.tsv.gz
05/08/2022  11:01 PM           827,840 tmdb.movies.csv.gz
05/08/2022  11:01 PM           153,218 tn.movie_budgets.csv.gz
               7 File(s)     72,192,269 bytes
               2 Dir(s)  407,640,190,976 bytes free
```

I'm going to convert each csv file into a pandas database. There are five csv files to convert.

In [3]:
```
bom = pd.read_csv('C:/Users/g_osb/Clones/Phase1/dsc-phase-1-project-v2-4/zippedData/bom.mov
rt_info = pd.read_csv('C:/Users/g_osb/Clones/Phase1/dsc-phase-1-project-v2-4/zippedData/rt
rt_reviews = pd.read_csv('C:/Users/g_osb/Clones/Phase1/dsc-phase-1-project-v2-4/zippedData/
tmdb = pd.read_csv('C:/Users/g_osb/Clones/Phase1/dsc-phase-1-project-v2-4/zippedData/tmdb.m
tn = pd.read_csv('C:/Users/g_osb/Clones/Phase1/dsc-phase-1-project-v2-4/zippedData/tn.movie
```

The remaining zip file needs to be unzipped. It contains a a SQL file, which needs to be connected to python so I can run SQL queries.

In [4]:
```
#unzip the zip file
shutil.unpack_archive("C:/Users/g_osb/Clones/Phase1/dsc-phase-1-project-v2-4/zippedData/im
```

The SQL file is named im.db. Connecting it to Python

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

Next, I'll run a script that sets up printing the table names, then I'll print the table names

In [6]:
```
%%script sqlite3 im.db --out tables
.tables
.quit
```

In [7]:
```
#Now listing the tables
print(tables)
```

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

I will now convert each of the SQL tables to a pandas data frame and preview the data. Then I checked each table for fully duplicated rows, and deleted the duplicates. If a table had no duplicates, I deleted the check.

In [8]:
```
imdb_directors = pd.read_sql("""
SELECT *
FROM directors
;
""",conn)

imdb_directors.drop(imdb_directors[imdb_directors.duplicated()].index, inplace=True)
imdb_directors
```

Out[8]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0462036	nm1940585
2	tt0835418	nm0151540
4	tt0878654	nm0089502
5	tt0878654	nm2291498
...	...	...
291169	tt8999974	nm10122357
291170	tt9001390	nm6711477
291171	tt9001494	nm10123242
291172	tt9001494	nm10123248
291173	tt9004986	nm4993825

163535 rows × 2 columns

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

Out[9]:

	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
...	...	...	...	...	...	...	...	...
331698	tt9827784	2	Sayonara kuchibiru	None	None	original	None	1.0
331699	tt9827784	3	Farewell Song	XWW	en	imdbDisplay	None	0.0
331700	tt9880178	1	La atención	None	None	original	None	1.0
331701	tt9880178	2	La atención	ES	None	None	None	0.0
331702	tt9880178	3	The Attention	XWW	en	imdbDisplay	None	0.0

331703 rows × 8 columns

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

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
...	...	...	...
73851	tt9805820	8.1	25
73852	tt9844256	7.5	24
73853	tt9851050	4.7	14
73854	tt9886934	7.0	5
73855	tt9894098	6.3	128

73856 rows × 3 columns

In [11]:
```
imdb_principals = pd.read_sql("""
SELECT *
FROM principals
;
""",conn)
imdb_principals
```

Out[11]:

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

1028186 rows × 6 columns

In [12]:
```
imdb_known_for = pd.read_sql("""
SELECT *
FROM known_for
;
""",conn)
imdb_known_for
```

Out[12]:

	person_id	movie_id
0	nm0061671	tt0837562
1	nm0061671	tt2398241
2	nm0061671	tt0844471
3	nm0061671	tt0118553
4	nm0061865	tt0896534
...	...	...
1638255	nm9990690	tt9090932
1638256	nm9990690	tt8737130
1638257	nm9991320	tt8734436
1638258	nm9991320	tt9615610
1638259	nm9993380	tt8743182

1638260 rows × 2 columns

In [13]:
```
imdb_movie_basics = pd.read_sql("""
SELECT *
FROM movie_basics
;
""",conn)
imdb_movie_basics
```

Out[13]:

	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
...	...	...	...	...	...	...
146139	tt9916538	Kuambil Lagi Hatiku	Kuambil Lagi Hatiku	2019	123.0	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	116.0	None
146143	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	NaN	Documentary

146144 rows × 6 columns

In [14]:
```
imdb_persons = pd.read_sql("""
SELECT *
FROM persons
;
""",conn)
imdb_persons
```

Out[14]:

	person_id	primary_name	birth_year	death_year	primary_profession
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,producer
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_department
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,actor,writer
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_department
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_decorator
...	...	...	...	...	...
606643	nm9990381	Susan Grobes	NaN	NaN	actress
606644	nm9990690	Joo Yeon So	NaN	NaN	actress
606645	nm9991320	Madeline Smith	NaN	NaN	actress
606646	nm9991786	Michelle Modigliani	NaN	NaN	producer
606647	nm9993380	Pegasus Envoyé	NaN	NaN	director,actor,writer

606648 rows × 5 columns

In [15]:
```
imdb_writers = pd.read_sql("""
SELECT *
FROM writers
;
""",conn)

#This had a lot of fully identical duplicated rows, so I deleted them.
imdb_writers.drop(imdb_writers[imdb_writers.duplicated()].index, inplace=True)
imdb_writers
```

Out[15]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0438973	nm0175726
2	tt0438973	nm1802864
3	tt0462036	nm1940585
4	tt0835418	nm0310087
...	...	...
255868	tt8999892	nm10122246
255869	tt8999974	nm10122357
255870	tt9001390	nm6711477
255871	tt9004986	nm4993825
255872	tt9010172	nm8352242

178352 rows × 2 columns

A personal test of the provided database

As a fun exercise, I decided to look for Sanford Gibbons, my friend's Uncle.

In [16]:
```
pd.read_sql("""
SELECT *
FROM persons
WHERE primary_name LIKE '%Gibbons%';
""",conn)
```

Out[16]:

	person_id	primary_name	birth_year	death_year	primary_profession
0	nm0316584	Patrick D. Gibbons	NaN	NaN	assistant_director,producer,miscellaneous
1	nm0316554	Greg Gibbons	NaN	NaN	animation_department,visual_effects,art_director
2	nm0316596	Sanford Gibbons	1933.0	2018.0	actor
3	nm1733301	Dave Gibbons	1949.0	NaN	writer,art_department,producer
4	nm0316531	Billy Gibbons	1949.0	NaN	soundtrack,actor,composer
5	nm2524514	Matt Gibbons	NaN	NaN	actor,writer,director
6	nm1644817	Tyler Gibbons	NaN	NaN	composer,sound_department,soundtrack
7	nm3453179	Pete Gibbons	NaN	NaN	producer
8	nm1668151	Neil Gibbons	NaN	NaN	writer,producer,director
9	nm2971362	Tony Gibbons	1983.0	NaN	actor
10	nm1668152	Rob Gibbons	NaN	NaN	writer,producer,director
11	nm1766037	Michael Gibbons	NaN	NaN	None
12	nm3202120	Lauren Gibbons	NaN	NaN	actress
13	nm4752963	Sally Fitzgibbons	NaN	NaN	None
14	nm3501059	Marlon Gibbons	NaN	NaN	composer,soundtrack
15	nm4613842	Richard Fitzgibbons	NaN	NaN	None
16	nm4548984	Darryn Gibbons	NaN	NaN	actor,writer,director
17	nm3106632	Derek Gibbons	NaN	NaN	actor,producer,director
18	nm4313499	Anthony Gibbons	NaN	NaN	cinematographer
19	nm5854730	Anthony Gibbons	NaN	NaN	actor
20	nm3180477	Michael Gibbons	NaN	NaN	actor
21	nm5785553	Tom Gibbons	NaN	NaN	cinematographer
22	nm4425737	Brendan Gibbons	NaN	NaN	director,writer,producer
23	nm5258424	Maurine Gibbons	NaN	NaN	miscellaneous,actress,writer
24	nm4882012	Akil Gibbons	NaN	NaN	producer,miscellaneous,camera_department
25	nm5903645	Kendyl Gibbons	NaN	NaN	None
26	nm5532424	Brendon Gibbons	NaN	NaN	None
27	nm4958141	Philip Gibbons	NaN	NaN	writer,director,producer
28	nm5964883	Matthew Gibbons	NaN	NaN	sound_department,camera_department,editor
29	nm6367107	John Carroll-Gibbons	NaN	NaN	editor,visual_effects,editorial_department
30	nm9242691	Mark Gibbons	NaN	NaN	actor

Sure enough, Sanford's person_id is nm0316596. Let's see what films Sanford Gibbons has listed in this database.

In [17]:
```
pd.read_sql("""
SELECT *
FROM principals
JOIN movie_basics
    USING(movie_id)
WHERE person_id LIKE 'nm0316596';
""",conn)
```

Out[17]:

	movie_id	ordering	person_id	category	job	characters	primary_title	original_title	start_year	runtime_
0	tt5607782	4	nm0316596	actor	None	["Father James Burk"]	The Covenant	The Covenant	2017	

Data Cleaning / Merging of tables

After previewing all the data provided, I decided on these three business questions (printed above):

1. What genres of film produce the highest box office revenue?
2. What is the best month to release a film in to generate the most revenue?
3. What writers, directors, producers, actors and actresses have the highest revenue earning potential?

For these three questions, I don't need to utilize the information from Rotten Tomatoes. That database does not have any movie titles that can be matched to the other databases. It is primarily internet user film ratings, which is not being considered by my three business questions.

Instead, I will join the remaining four databases together: Box Office Mojo, IMDB, The Numbers, and The Movie Database. I will only keep the films that exist in each database provided. First, I'll start with merging Box Office Mojo and the Internet Movie Database.

After reviewing the data, I found a strange mistake in Box Office Mojo's date for a 2012 movie titled Upside Down. It's date is listed 2013. This caused the data to be inconsistent with other databases. I could just delete it, but, after finding the error, it's just as easy to correct it.

In [18]:
```
bom.at[1298,'year'] = 2012
bom.iloc[1298]['year']
```

Out[18]: 2012

In [19]:
```
#Merging bom and imdb
movies_analyzed = pd.merge(bom,imdb_movie_basics, how='inner',left_on='title', right_on='o
movies_analyzed
```

Out[19]:

	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title	original_title	start_year
0	Toy Story 3	BV	415000000.0	652000000	2010	tt0435761	Toy Story 3	Toy Story 3	2010
1	Inception	WB	292600000.0	535700000	2010	tt1375666	Inception	Inception	2010
2	Shrek Forever After	P/DW	238700000.0	513900000	2010	tt0892791	Shrek Forever After	Shrek Forever After	2010
3	The Twilight Saga: Eclipse	Sum.	300500000.0	398000000	2010	tt1325004	The Twilight Saga: Eclipse	The Twilight Saga: Eclipse	2010
4	Iron Man 2	Par.	312400000.0	311500000	2010	tt1228705	Iron Man 2	Iron Man 2	2010
...	...	...	...	...	...	...	...	...	...
2771	The Escape	IFC	14000.0	NaN	2018	tt6069126	The Escape	The Escape	2017
2772	Souvenir	Strand	11400.0	NaN	2018	tt2387692	Souvenir	Souvenir	2016
2773	Souvenir	Strand	11400.0	NaN	2018	tt2389092	Souvenir	Souvenir	2014
2774	Souvenir	Strand	11400.0	NaN	2018	tt3478898	Souvenir	Souvenir	2014
2775	An Actor Prepares	Grav.	1700.0	NaN	2018	tt5718046	An Actor Prepares	An Actor Prepares	2018

2776 rows × 11 columns

I will review the merged data.

In [20]:
```
movies_analyzed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2776 entries, 0 to 2775
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   title            2776 non-null   object
 1   studio           2773 non-null   object
 2   domestic_gross   2756 non-null   float64
 3   foreign_gross    1778 non-null   object
 4   year             2776 non-null   int64
 5   movie_id         2776 non-null   object
 6   primary_title    2776 non-null   object
 7   original_title   2776 non-null   object
 8   start_year       2776 non-null   int64
 9   runtime_minutes  2610 non-null   float64
 10  genres           2740 non-null   object
dtypes: float64(2), int64(2), object(7)
memory usage: 260.2+ KB
```

I will check for inconsistent debut year information between these two database and drop any films that do not match between the databases.

In [21]:
```
hese films
max_rows", None, "display.max_columns", None)

_analyzed[movies_analyzed['title'].duplicated()]
ies_to_drop[(movies_to_drop['year'] == movies_to_drop['start_year'])].index, inplace=True)
,'studio','year','primary_title','movie_id','start_year','runtime_minutes']]
```

```
C:\Users\g_osb\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py:4163: Sett
ingWithCopyWarning:
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-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  return super().drop(
```

Out[21]:

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
15	Robin Hood	Uni.	2010	Robin Hood	tt2363363	2013	92.0
16	Robin Hood	Uni.	2010	Robin Hood	tt4532826	2018	116.0
17	Robin Hood	Uni.	2010	Robin Hood	tt6858500	2018	NaN
18	Robin Hood	Uni.	2010	Robin Hood	tt8558276	2017	60.0
37	Red	Sum.	2010	Red	tt2294785	2012	26.0
38	Red	Sum.	2010	Red	tt4136848	2014	82.0
39	Red	Sum.	2010	Red	tt4170206	2014	107.0
40	Red	Sum.	2010	Red	tt8851190	2018	90.0
45	Unstoppable	Fox	2010	Unstoppable	tt2951338	2015	53.0
46	Unstoppable	Fox	2010	Unstoppable	tt3070502	2013	65.0
47	Unstoppable	Fox	2010	Unstoppable	tt7098772	2017	12.0
48	Unstoppable	Fox	2010	Unstoppable	tt9499562	2017	52.0
49	Unstoppable	Fox	2010	Unstoppable	tt9906218	2019	84.0
52	The Town	WB	2010	The Town	tt6259458	2012	101.0
66	Life as We Know It	WB	2010	Life as We Know It	tt6341928	2017	NaN
70	Killers	LGF	2010	Killers	tt2409300	2014	137.0
98	Remember Me	Sum.	2010	Remember Me	tt4843358	2016	85.0
99	Remember Me	Sum.	2010	Remember Me	tt4999290	2019	NaN
100	Remember Me	Sum.	2010	Remember Me	tt7990108	2019	NaN
114	Going the Distance	WB (NL)	2010	Going the Distance	tt5127484	2015	65.0
124	The Losers	WB	2010	The Losers	tt3746918	2013	112.0
127	Let Me In	Over.	2010	Let Me In	tt7122852	2017	74.0
134	Buried	LGF	2010	Buried	tt3568352	2014	88.0
135	Buried	LGF	2010	Buried	tt4540326	2011	95.0
154	Conviction	FoxS	2010	Conviction	tt1772252	2013	NaN
156	Stone	Over.	2010	Stone	tt2355884	2012	114.0

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
158	Never Let Me Go	FoxS	2010	Never Let Me Go	tt9776362	2019	NaN
162	The Joneses	RAtt.	2010	The Joneses	tt4360484	2016	80.0
165	Animal Kingdom	SPC	2010	Animal Kingdom	tt7016298	2017	121.0
171	Vision	Zeit.	2010	Vision	tt7566518	2018	109.0
173	The Runaways	App.	2010	The Runaways	tt6168914	2019	108.0
178	Monsters	Magn.	2010	Monsters	tt2620288	2013	45.0
179	Monsters	Magn.	2010	Monsters	tt6449400	2015	NaN
180	Monsters	Magn.	2010	Monsters	tt8254042	2018	97.0
188	Twelve	Hann.	2010	Twelve	tt5001140	2015	NaN
189	Twelve	Hann.	2010	Twelve	tt7033498	2019	92.0
190	Twelve	Hann.	2010	Twelve	tt7343176	2018	NaN
193	Housefull	Eros	2010	Housefull	tt4991978	2013	NaN
197	Flipped	WB	2010	Flipped	tt2388050	2015	90.0
204	Howl	Osci.	2010	Howl	tt2393827	2015	89.0
209	We Are Family	UTV	2010	We Are Family	tt1496040	2017	NaN
219	I'm Still Here	Magn.	2010	I'm Still Here	tt2636364	2013	120.0
224	The Tempest	Mira.	2010	The Tempest	tt3840898	2014	169.0
225	The Tempest	Mira.	2010	The Tempest	tt9151364	2019	NaN
229	The Girl on the Train	Strand	2010	The Girl on the Train	tt3631112	2016	112.0
230	The Girl on the Train	Strand	2010	The Girl on the Train	tt9799088	2018	NaN
233	Eyes Wide Open	NAV	2010	Eyes Wide Open	tt6593242	2018	110.0
235	Waste Land	Arth.	2010	Waste Land	tt2294939	2014	97.0
251	Cherry	Abr.	2010	Cherry	tt9130508	2020	NaN
263	Rio	Fox	2011	Rio	tt2614250	2012	90.0
264	Rio	Fox	2011	Rio	tt5734820	2017	87.0
283	Hugo	Par.	2011	Hugo	tt7496196	2017	80.0
287	The Descendants	FoxS	2011	The Descendants	tt5838908	2015	80.0
293	Limitless	Rela.	2011	Limitless	tt4597838	2015	3.0
294	Limitless	Rela.	2011	Limitless	tt6084030	2017	57.0
305	No Strings Attached	Par.	2011	No Strings Attached	tt6298204	2016	NaN
306	No Strings Attached	Par.	2011	No Strings Attached	tt6598256	2017	73.0
314	The Artist	Wein.	2011	The Artist	tt6684818	2015	53.0
316	Unknown	WB	2011	Unknown	tt2402731	2012	96.0
317	Unknown	WB	2011	Unknown	tt5213284	2015	NaN
318	Unknown	WB	2011	Unknown	tt5877156	2017	NaN
327	Paul	Uni.	2011	Paul	tt5132504	2016	71.0
339	Abduction	LGF	2011	Abduction	tt2335176	2013	NaN
341	Abduction	LGF	2011	Abduction	tt5943940	2017	90.0

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
342	Abduction	LGF	2011	Abduction	tt7867360	2019	106.0
352	The Darkest Hour	Sum.	2011	The Darkest Hour	tt4249090	2014	52.0
357	One Day	Focus	2011	One Day	tt6168298	2016	135.0
358	One Day	Focus	2011	One Day	tt6214734	2017	118.0
362	The Tree of Life	FoxS	2011	The Tree of Life	tt8991250	2018	60.0
372	50/50	Sum.	2011	50/50	tt1734060	2010	77.0
382	Jane Eyre	Focus	2011	Jane Eyre	tt9626910	2019	90.0
390	The Eagle	Focus	2011	The Eagle	tt5130954	2015	11.0
398	Warrior	LGF	2011	Warrior	tt4163390	2015	NaN
408	Melancholia	Magn.	2011	Melancholia	tt7592248	2018	NaN
410	The Conspirator	RAtt.	2011	The Conspirator	tt7520136	2012	NaN
412	Anonymous	Sony	2011	Anonymous	tt6419530	2014	NaN
413	Anonymous	Sony	2011	Anonymous	tt7632298	2017	60.0
418	Win Win	FoxS	2011	Win Win	tt2877476	2013	97.0
424	Last Night	Trib.	2011	Last Night	tt1740794	2010	69.0
425	Last Night	Trib.	2011	Last Night	tt3602660	2014	93.0
426	Last Night	Trib.	2011	Last Night	tt7433980	2017	104.0
427	Last Night	Trib.	2011	Last Night	tt7449324	2017	94.0
428	Last Night	Trib.	2011	Shattered Memories	tt8678108	2018	86.0
439	Force	FoxS	2011	Force	tt4195244	2014	135.0
443	The Way	PDA	2011	The Way	tt1692226	2010	107.0
444	The Way	PDA	2011	The Way	tt6127140	2017	85.0
445	The Way	PDA	2011	The Way	tt6216234	2016	85.0
446	The Way	PDA	2011	The Way	tt7637364	2018	73.0
451	Like Crazy	ParV	2011	Like Crazy	tt8681390	2018	NaN
453	The Double	Imag.	2011	The Double	tt1825157	2013	93.0
465	The Tree	Zeit.	2011	The Tree	tt6206204	2017	95.0
469	Bodyguard	Relbig.	2011	Bodyguard	tt2100411	2012	150.0
470	Bodyguard	Relbig.	2011	Bodyguard	tt5460508	2016	105.0
480	The Whistleblower	Gold.	2011	The Whistleblower	tt8971476	2019	NaN
513	Super	IFC	2011	Super	tt1807022	2010	140.0
514	Super	IFC	2011	Super	tt7456902	2016	92.0
528	Anchor Baby	AGF	2011	Anchor Baby	tt3296300	2013	76.0
534	Octubre	NYer	2011	October	tt1646118	2010	83.0
542	Beautiful Boy	Anch.	2011	Beautiful Boy	tt1533013	2010	100.0
556	Ceremony	Magn.	2011	Ceremony	tt4205114	2014	58.0
557	Ceremony	Magn.	2011	Ceremony	tt4466316	2015	46.0
561	Aurora	CGld	2011	Vanishing Waves	tt2208216	2012	124.0

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
562	Aurora	CGld	2011	Aurora	tt3603470	2014	83.0
563	Aurora	CGld	2011	Aurora	tt5341128	2016	NaN
564	Aurora	CGld	2011	Aurora	tt8095720	2017	68.0
565	Aurora	CGld	2011	Aurora	tt8396182	2018	98.0
566	Aurora	CGld	2011	Aurora	tt8553606	2019	106.0
567	Aurora	CGld	2011	Aurora	tt8821182	2018	110.0
570	Blood Ties	ALP	2011	Blood Ties	tt3882558	2017	120.0
571	Blood Ties	ALP	2011	Blood Ties	tt5621080	2014	NaN
586	Brave	BV	2012	Brave	tt8358722	2014	NaN
601	Lincoln	BV	2012	Lincoln	tt3784440	2014	17.0
610	Safe House	Uni.	2012	Safe House	tt6686996	2016	NaN
618	Flight	Par.	2012	Flight	tt4389440	2017	NaN
634	The Campaign	WB	2012	The Campaign	tt2411684	2013	60.0
636	Project X	WB	2012	Project X	tt5902440	2017	NaN
649	Sinister	LG/S	2012	Sinister	tt7489744	2017	NaN
663	The Three Stooges	Fox	2012	The Three Stooges	tt5108902	2015	NaN
666	Lawless	Wein.	2012	Lawless	tt8781832	2018	90.0
689	The Master	Wein.	2012	The Master	tt4117096	2014	NaN
690	The Master	Wein.	2012	The Master	tt4273106	2014	86.0
691	The Master	Wein.	2012	The Master	tt4777394	2014	NaN
697	Wanderlust	Uni.	2012	Wanderlust	tt5246726	2015	3.0
698	Wanderlust	Uni.	2012	Wanderlust	tt6326352	2017	58.0
699	Wanderlust	Uni.	2012	Wanderlust	tt6851110	2017	NaN
706	Gone	LG/S	2012	Gone	tt1628042	2011	85.0
708	Gone	LG/S	2012	Gone	tt1937473	2011	NaN
709	Gone	LG/S	2012	Gone	tt2156791	2011	45.0
711	Gone	LG/S	2012	Gone	tt2911578	2013	45.0
712	Gone	LG/S	2012	Gone	tt4851420	2016	NaN
713	Gone	LG/S	2012	Gone	tt4852142	2015	5.0
714	Gone	LG/S	2012	Gone	tt6123126	2016	51.0
715	Gone	LG/S	2012	Gone	tt7078824	2017	82.0
716	Gone	LG/S	2012	Gone	tt7474568	2015	54.0
724	People Like Us	BV	2012	People Like Us	tt1758741	2010	55.0
732	Hysteria	SPC	2012	Hysteria	tt1435513	2011	100.0
733	Hysteria	SPC	2012	Hysteria	tt5467982	2016	81.0
735	Bernie	MNE	2012	Bernie	tt5936840	2016	155.0
740	On the Road	IFC	2012	On the Road	tt2404548	2011	90.0
741	On the Road	IFC	2012	On the Road	tt3872966	2013	87.0

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
742	On the Road	IFC	2012	On the Road	tt4339118	2014	89.0
743	On the Road	IFC	2012	On the Road	tt5389486	2015	39.0
744	On the Road	IFC	2012	On the Road	tt5647250	2016	121.0
749	Heroine	UTV	2012	Heroine	tt5921576	2016	87.0
750	Heroine	UTV	2012	Heroine	tt7134728	2015	NaN
758	In Darkness	SPC	2012	In Darkness	tt5164184	2018	100.0
760	Won't Back Down	Fox	2012	Won't Back Down	tt3779570	2014	114.0
762	The Tall Man	Imag.	2012	The Tall Man	tt1864549	2011	79.0
766	Barbara	AF	2012	Barbara	tt6411984	2017	98.0
767	Barbara	AF	2012	Barbara	tt7271942	2018	6.0
771	The Lady	Cohen	2012	The Lady	tt4467734	2014	NaN
772	The Lady	Cohen	2012	The Lady	tt5121606	2015	79.0
774	Branded	RAtt.	2012	Branded	tt3429774	2017	73.0
775	Branded	RAtt.	2012	Branded	tt5000924	2015	73.0
781	Intruders	MNE	2012	Intruders	tt3996044	2017	90.0
782	Intruders	MNE	2012	Intruders	tt6448608	2016	95.0
805	I Wish	Magn.	2012	I Wish	tt5528558	2017	NaN
807	Marley	Magn.	2012	Marley	tt6600382	2017	90.0
808	Marley	Magn.	2012	Marley	tt7551724	2018	NaN
818	Coriolanus	Wein.	2012	Coriolanus	tt1372686	2011	123.0
819	Coriolanus	Wein.	2012	Coriolanus	tt9151404	2019	NaN
851	Elena	Zeit.	2012	Elena	tt1925421	2011	109.0
853	Elena	Zeit.	2012	Elena	tt3747268	2014	NaN
863	Joker	UTV	2012	Joker	tt3002286	2013	94.0
864	Joker	UTV	2012	Joker	tt3010370	2013	49.0
865	Joker	UTV	2012	Joker	tt5611648	2016	130.0
866	Joker	UTV	2012	Joker	tt7286456	2019	NaN
868	Dark Horse	BM&DH	2012	Dark Horse	tt4137902	2015	85.0
893	The First Time	Gold.	2012	The First Time	tt9827712	2018	90.0
901	Take Me Home	Mont.	2012	Take Me Home	tt2105000	2010	60.0
902	Take Me Home	Mont.	2012	Take Me Home	tt5563194	2016	94.0
911	ATM	IFC	2012	ATM	tt6419602	2016	NaN
912	ATM	IFC	2012	ATM	tt8216296	2015	121.0
936	Now You See Me	LG/S	2013	Now You See Me	tt2221640	2012	98.0
941	Oblivion	Uni.	2013	Oblivion	tt1876409	2011	116.0
942	Oblivion	Uni.	2013	Oblivion	tt8826828	2018	NaN
944	Elysium	TriS	2013	Elysium	tt1783264	2010	NaN
967	Mama	Uni.	2013	Mama	tt1868050	2010	71.0

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
969	Mama	Uni.	2013	Mama	tt3843072	2011	110.0
970	Mama	Uni.	2013	Mama	tt5437970	2016	90.0
971	Mama	Uni.	2013	Mama	tt7514806	2017	89.0
972	Mama	Uni.	2013	Mama	tt8323598	2017	NaN
986	Safe Haven	Rela.	2013	Safe Haven	tt5073094	2016	120.0
999	The Call	TriS	2013	The Call	tt2554760	2014	90.0
1000	The Call	TriS	2013	The Call	tt4741754	2014	109.0
1008	The Last Stand	LGF	2013	The Last Stand	tt8774508	2019	93.0
1033	Trance	FoxS	2013	Trance	tt8239184	2018	NaN
1037	Diana	EOne	2013	Diana	tt5531322	2016	88.0
1038	Diana	EOne	2013	Diana	tt8082576	2017	47.0
1039	Diana	EOne	2013	Diana	tt8716014	2018	101.0
1054	Paranoia	Rela.	2013	Paranoia	tt1904995	2012	87.0
1055	Paranoia	Rela.	2013	Paranoia	tt4025788	2014	85.0
1056	Paranoia	Rela.	2013	Paranoia	tt6338350	2016	60.0
1057	Paranoia	Rela.	2013	Paranoia	tt6449960	2016	NaN
1058	Paranoia	Rela.	2013	Paranoia	tt8045822	2015	84.0
1059	Paranoia	Rela.	2013	Paranoia	tt8327742	2018	NaN
1065	The Past	SPC	2013	The Past	tt9649770	2018	53.0
1070	Upside Down	MNE	2012	Upside Down	tt1713561	2011	NaN
1072	Upside Down	MNE	2012	Upside Down	tt3248078	2013	NaN
1073	Upside Down	MNE	2012	Upside Down	tt5105628	2015	70.0
1074	Upside Down	MNE	2012	Upside Down	tt8399650	2015	NaN
1083	Believe	ORF	2013	Believe	tt4700756	2016	119.0
1084	Believe	ORF	2013	Believe	tt7399140	2019	109.0
1085	Believe	ORF	2013	Believe	tt9347476	2016	NaN
1091	White Elephant	Strand	2013	White Elephant	tt6327126	2016	98.0
1096	Mental	Da.	2013	Mental	tt4145834	2014	NaN
1100	Capital	Cohen	2013	Capital	tt3466724	2014	73.0
1111	Disconnect	LD	2013	Disconnect	tt1433811	2012	115.0
1112	Disconnect	LD	2013	Disconnect	tt8413566	2018	107.0
1116	The Invisible Woman	SPC	2013	The Invisible Woman	tt1726787	2010	NaN
1125	Lore	MBox	2013	Lore	tt5038134	2018	NaN
1130	Satyagraha	UTV	2013	Satyagraha	tt6449166	2011	NaN
1135	The Hunt	Magn.	2013	The Hunt	tt3108154	2016	100.0
1136	The Hunt	Magn.	2013	The Hunt	tt5906014	2016	5.0
1137	The Hunt	Magn.	2013	The Hunt	tt8244784	2019	NaN
1138	The Hunt	Magn.	2013	The Hunt	tt9856484	2019	90.0

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
1153	Himmatwala	UTV	2013	Himmatwala	tt2802286	2012	NaN
1155	The Silence	MBox	2013	The Silence	tt4603640	2015	91.0
1156	The Silence	MBox	2013	The Silence	tt6143030	2016	NaN
1157	The Silence	MBox	2013	The Silence	tt7161338	2017	116.0
1158	The Silence	MBox	2013	The Silence	tt7315484	2019	90.0
1188	The Look of Love	IFC	2013	The Look of Love	tt2787052	2012	46.0
1200	Into the White	Magn.	2013	Into the White	tt6278986	2017	14.0
1217	Lucy	Uni.	2014	Lucy	tt5313474	2016	9.0
1218	Lucy	Uni.	2014	Lucy	tt6142034	2016	40.0
1225	Noah	Par.	2014	Noah	tt2109176	2011	105.0
1231	Neighbors	Uni.	2014	Neighbors	tt3638264	2013	22.0
1232	Neighbors	Uni.	2014	Neighbors	tt9386300	2018	74.0
1233	Neighbors	Uni.	2014	Neighbors	tt9392532	2018	90.0
1234	Neighbors	Uni.	2014	Neighbors	tt9702034	2012	NaN
1242	Into the Woods	BV	2014	Into the Woods	tt2191715	2011	NaN
1243	Into the Woods	BV	2014	Into the Woods	tt2201083	2012	NaN
1244	Into the Woods	BV	2014	Into the Woods	tt9703646	2018	NaN
1246	Journey to the West	Magn.	2014	Journey to the West	tt5786990	2016	NaN
1255	Ride Along	Uni.	2014	Ride Along	tt2263520	2012	87.0
1270	Transcendence	WB	2014	Transcendence	tt2339048	2012	45.0
1280	The Judge	WB	2014	The Judge	tt6246092	2017	76.0
1293	Begin Again	Wein.	2014	Begin Again	tt1980929	2013	104.0
1302	Chef	ORF	2014	Chef	tt5745450	2017	133.0
1311	The Gambler	Par.	2014	The Gambler	tt2843832	2013	52.0
1312	The Gambler	Par.	2014	The Gambler	tt9447594	2019	121.0
1334	The Drop	FoxS	2014	The Drop	tt4745746	2015	50.0
1335	The Drop	FoxS	2014	The Drop	tt5177706	2016	NaN
1340	Addicted	LGF	2014	Addicted	tt5396774	2013	55.0
1341	Addicted	LGF	2014	Addicted	tt6949630	2017	54.0
1352	One Chance	Wein.	2014	One Chance	tt2583860	2011	NaN
1354	Belle	FoxS	2014	Belle	tt2404181	2013	100.0
1366	The Immigrant	Wein.	2014	The Immigrant	tt4666530	2015	47.0
1367	The Immigrant	Wein.	2014	The Immigrant	tt6594436	2017	65.0
1377	Mommy	RAtt.	2014	Mommy	tt5152894	2015	135.0
1380	Enemy	A24	2014	Enemy	tt5207964	2015	NaN
1390	Kick	UTV	2014	Kick	tt2402127	2015	76.0
1391	Kick	UTV	2014	Kick	tt2597186	2018	NaN
1415	Jackpot	DR	2014	Jackpot	tt2180421	2012	95.0

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
1416	Jackpot	DR	2014	Jackpot	tt3309662	2013	132.0
1417	Jackpot	DR	2014	Jackpot	tt5094332	2015	11.0
1418	Jackpot	DR	2014	Jackpot	tt8671762	2018	150.0
1431	Frank	Magn.	2014	Frank	tt1815706	2012	94.0
1432	Frank	Magn.	2014	Frank	tt9045760	2019	NaN
1437	Archipelago	KL	2014	Archipelago	tt8045226	2017	61.0
1446	Happy Ending	Eros	2014	Happy Ending	tt4946192	2015	NaN
1447	Happy Ending	Eros	2014	Happy Ending	tt6445354	2016	95.0
1448	Happy Ending	Eros	2014	Happy Ending	tt7413472	2018	96.0
1456	Bethlehem	AF	2014	Bethlehem	tt5065924	2015	NaN
1489	Wolves	KE	2014	Wolves	tt4623856	2016	109.0
1493	Stray Dogs	CGld	2014	Stray Dogs	tt6098740	2017	82.0
1499	Anna	VE	2014	Anna	tt1951077	2011	80.0
1500	Anna	VE	2014	Anna	tt3673036	2015	96.0
1501	Anna	VE	2014	Anna	tt3872918	2015	109.0
1502	Anna	VE	2014	Anna	tt6118134	2016	142.0
1503	Anna	VE	2014	Anna	tt6427854	2017	97.0
1504	Anna	VE	2014	Anna	tt7456310	2019	NaN
1512	Inside Out	BV	2015	Inside Out	tt2064820	2011	NaN
1513	Inside Out	BV	2015	Inside Out	tt2071483	2011	59.0
1515	Inside Out	BV	2015	Inside Out	tt2608638	2013	75.0
1516	Inside Out	BV	2015	Inside Out	tt6419446	2011	NaN
1517	Inside Out	BV	2015	Inside Out	tt8269544	2018	NaN
1523	The Revenant	Fox	2015	The Revenant	tt3300078	2012	80.0
1538	Daddy's Home	Par.	2015	Daddy's Home	tt1617630	2010	90.0
1545	Tomorrowland	BV	2015	Tomorrowland	tt7728220	2017	70.0
1551	Creed	WB (NL)	2015	Creed	tt4354930	2016	80.0
1563	Get Hard	WB	2015	Get Hard	tt8672656	2018	90.0
1568	Sisters	Uni.	2015	Sisters	tt5870006	2016	102.0
1569	Sisters	Uni.	2015	Sisters	tt7326798	2018	NaN
1570	Sisters	Uni.	2015	Sisters	tt9048972	2018	72.0
1571	Sisters	Uni.	2015	Sisters	tt9851050	2019	NaN
1575	Joy	Fox	2015	Joy	tt1465490	2010	75.0
1576	Joy	Fox	2015	Joy	tt2327665	2012	53.0
1578	Joy	Fox	2015	Joy	tt6180144	2016	80.0
1579	Joy	Fox	2015	Joy	tt8731858	2018	4.0
1580	Joy	Fox	2015	Joy	tt8917752	2018	99.0
1583	The Visit	Uni.	2015	The Visit	tt1901018	2010	50.0

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
1584	The Visit	Uni.	2015	The Visit	tt2252032	2012	52.0
1607	Brooklyn	FoxS	2015	Brooklyn	tt3184798	2014	83.0
1614	No Escape	Wein.	2015	No Escape	tt2332790	2011	27.0
1619	The Night Before	Sony	2015	The Night Before	tt6006924	2017	81.0
1629	Legend	Uni.	2015	Legend	tt3409352	2012	34.0
1630	Legend	Uni.	2015	Legend	tt3555036	2014	168.0
1642	Burnt	Wein.	2015	Burnt	tt4607118	2014	NaN
1657	The Gunman	ORF	2015	The Gunman	tt5241920	2016	NaN
1659	Black or White	Rela.	2015	Black or White	tt6270540	2016	135.0
1669	Unfinished Business	Fox	2015	Unfinished Business	tt4998626	2016	NaN
1670	Unfinished Business	Fox	2015	Unfinished Business	tt6893224	2017	NaN
1671	Unfinished Business	Fox	2015	Unfinished Business	tt7316906	2017	60.0
1672	Unfinished Business	Fox	2015	Unfinished Business	tt8323644	2017	NaN
1673	Unfinished Business	Fox	2015	Unfinished Business	tt8324550	2017	NaN
1675	Do You Believe?	PFR	2015	Do You Believe?	tt6960218	2017	71.0
1678	Strange Magic	BV	2015	Strange Magic	tt4319336	2014	NaN
1686	Shanghai	Wein.	2015	Shanghai	tt2072227	2012	120.0
1687	Shanghai	Wein.	2015	Shanghai	tt5424576	2016	120.0
1691	Amy	A24	2015	Amy	tt3007924	2013	94.0
1701	Serena	Magn.	2015	Serena	tt5749696	2016	95.0
1706	Trash	FCW	2015	Trash	tt1846800	2010	NaN
1707	Trash	FCW	2015	Trash	tt1921149	2014	114.0
1708	Trash	FCW	2015	Trash	tt1988822	2010	75.0
1711	True Story	FoxS	2015	True Story	tt3044736	2013	80.0
1728	Youth	FoxS	2015	Youth	tt5671384	2016	124.0
1731	Truth	SPC	2015	Truth	tt1921152	2011	99.0
1732	Truth	SPC	2015	Truth	tt2459920	2013	94.0
1733	Truth	SPC	2015	The Virus	tt3017234	2014	96.0
1740	Cake	CLF	2015	Cake	tt7715988	2018	125.0
1741	Cake	CLF	2015	Cake	tt7993780	2017	104.0
1768	Tangerine	Magn.	2015	Tangerine	tt6212558	2016	127.0
1783	Noble	Asp.	2015	Noble	tt2626090	2014	100.0
1784	Noble	Asp.	2015	Noble	tt5694538	2016	117.0
1797	The Connection	Drft.	2015	The Connection	tt2310270	2012	51.0
1798	The Connection	Drft.	2015	The Connection	tt3856972	2014	72.0
1800	Maggie	RAtt.	2015	Maggie	tt9000062	2018	88.0
1818	Christmas Eve	Ampl.	2015	Christmas Eve	tt7263240	2019	NaN
1827	Eden	BG	2015	Eden	tt1734433	2012	98.0

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
1828	Eden	BG	2015	Eden	tt2063641	2012	101.0
1829	Eden	BG	2015	Eden	tt2132321	2016	71.0
1830	Eden	BG	2015	Eden	tt3032282	2014	90.0
1831	Eden	BG	2015	Eden	tt3090634	2014	131.0
1833	Eden	BG	2015	Eden	tt7539088	2019	85.0
1834	Eden	BG	2015	Eden	tt9174880	2018	97.0
1837	Flowers	MBox	2015	Flowers	tt3110280	2013	NaN
1839	Flowers	MBox	2015	Flowers	tt4534208	2011	107.0
1852	Before We Go	RTWC	2015	Before We Go	tt4453750	2014	84.0
1854	Knock Knock	LGP	2015	Knock Knock	tt6096414	2017	55.0
1860	The Cut	Strand	2015	The Cut	tt2245171	2014	138.0
1867	The Little Death	Magn.	2015	The Little Death	tt2785032	2014	96.0
1868	The Little Death	Magn.	2015	The Little Death	tt8488276	2018	62.0
1875	Extraction	LGP	2015	Extraction	tt7841286	2018	NaN
1877	Eva	Wein.	2015	Eva	tt1298554	2011	94.0
1879	Eva	Wein.	2015	Eva	tt6414890	2018	100.0
1902	Skin Trade	Magn.	2015	Skin Trade	tt1641841	2014	96.0
1926	Dangal	UTV	2016	Dangal	tt5911540	2013	NaN
1935	Inferno	Sony	2016	Inferno	tt3219106	2014	113.0
1936	Inferno	Sony	2016	Inferno	tt9149142	2017	58.0
1942	Arrival	Par.	2016	Arrival	tt7325124	2012	NaN
1954	Lights Out	WB (NL)	2016	Lights Out	tt2611518	2013	NaN
1960	Lion	Wein.	2016	Lion	tt4685968	2015	143.0
1974	The Little Prince	EOne	2016	The Little Prince	tt5317732	2015	NaN
1983	Nerve	LGF	2016	Nerve	tt2495778	2013	86.0
1985	Nerve	LGF	2016	Nerve	tt5324464	2015	62.0
1994	Fences	Par.	2016	Fences	tt3197696	2013	117.0
2020	When the Bough Breaks	SGem	2016	When the Bough Breaks	tt6814096	2017	93.0
2025	Fan	Yash	2016	Fan	tt7442346	2017	623.0
2037	Solace	LGP	2016	Solace	tt2140411	2013	76.0
2038	Solace	LGP	2016	Solace	tt3240102	2018	81.0
2039	Solace	LGP	2016	Solace	tt6687810	2017	66.0
2046	The Choice	LGF	2016	The Choice	tt8266726	2018	50.0
2059	Jackie	FoxS	2016	Jackie	tt1757772	2010	126.0
2060	Jackie	FoxS	2016	Jackie	tt2108546	2012	100.0
2061	Jackie	FoxS	2016	Jackie	tt2180419	2013	108.0
2064	Shut In	EC	2016	Intruders	tt4009278	2015	90.0

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
2071	The Perfect Match	LGF	2016	The Perfect Match	tt3827944	2010	87.0
2086	Genius	RAtt.	2016	Genius	tt6925204	2012	NaN
2087	Genius	RAtt.	2016	Genius	tt7722258	2018	165.0
2088	Genius	RAtt.	2016	Genius	tt8919194	2018	NaN
2115	Elle	SPC	2016	Elle	tt9104092	2018	60.0
2123	A Perfect Day	IFC	2016	A Perfect Day	tt7785272	2015	82.0
2146	City of Gold	IFC	2016	City of Gold	tt4113346	2018	99.0
2153	Viral	W/Dim.	2016	Viral	tt3892200	2015	NaN
2158	Marguerite	Cohen	2016	Marguerite	tt8686424	2019	NaN
2170	The Invitation	Drft.	2016	The Invitation	tt4546370	2015	6.0
2176	Tumbledown	SM	2016	Tumbledown	tt2732210	2013	80.0
2179	Three	WGUSA	2016	Three	tt3914834	2018	NaN
2180	Three	WGUSA	2016	Three	tt4205866	2015	NaN
2198	Fatima	KL	2016	Fatima	tt4466544	2015	79.0
2199	Fatima	KL	2016	Fatima	tt7972082	2017	65.0
2208	Mr. Right	FCW	2016	Mr. Right	tt3812348	2015	79.0
2216	Cosmos	KL	2016	Cosmos	tt4035268	2015	103.0
2218	Lolo	FR	2016	Lolo	tt4085944	2015	99.0
2226	Chronic	MR	2016	Chronic	tt7000030	2017	89.0
2231	The Other Side	FM	2016	The Other Side	tt2921338	2014	103.0
2232	The Other Side	FM	2016	The Other Side	tt3058792	2012	68.0
2233	The Other Side	FM	2016	The Other Side	tt3527966	2014	85.0
2235	The Other Side	FM	2016	The Other Side	tt4280102	2014	19.0
2236	The Other Side	FM	2016	The Other Side	tt4610244	2015	92.0
2238	The Other Side	FM	2016	The Other Side	tt5335244	2018	138.0
2239	The Other Side	FM	2016	The Other Side	tt9000546	2018	100.0
2241	The President	Crnth	2016	The President	tt7200166	2017	70.0
2253	Wonder Woman	WB	2017	Wonder Woman	tt4028068	2014	60.0
2254	Wonder Woman	WB	2017	Wonder Woman	tt4283448	2016	75.0
2277	Split	Uni.	2017	Split	tt3315656	2016	127.0
2278	Split	Uni.	2017	Split	tt3604256	2016	NaN
2279	Split	Uni.	2017	Split	tt4972582	2016	117.0
2280	Split	Uni.	2017	Split	tt5495666	2016	80.0
2281	Split	Uni.	2017	Split	tt6147768	2016	123.0
2301	The Foreigner	STX	2017	The Foreigner	tt2133340	2011	145.0
2302	The Foreigner	STX	2017	The Foreigner	tt2170623	2012	100.0
2327	Snatched	Fox	2017	Snatched	tt1245534	2011	81.0
2329	Snatched	Fox	2017	Snatched	tt5271436	2015	47.0

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
2330	Snatched	Fox	2017	Evil Doctor	tt7130262	2018	90.0
2351	The House	WB (NL)	2017	The House	tt2416192	2013	NaN
2352	The House	WB (NL)	2017	The House	tt2428826	2012	111.0
2354	The House	WB (NL)	2017	The Preachers House	tt7208440	2018	NaN
2358	Sleepless	ORF	2017	Sleepless	tt5657428	2015	120.0
2369	Father Figures	WB	2017	Father Figures	tt2925960	2013	58.0
2372	Born in China	BV	2017	One Child Nation	tt8923482	2019	85.0
2380	Gold	Wein.	2017	Gold	tt2338846	2013	101.0
2381	Gold	Wein.	2017	Gold	tt3134422	2014	88.0
2382	Gold	Wein.	2017	Gold	tt6173990	2018	151.0
2396	Stronger	RAtt.	2017	Stronger	tt7130472	2016	47.0
2410	Collide	ORF	2017	Collide	tt2126235	2016	99.0
2413	The Wall	RAtt.	2017	The Wall	tt4778274	2015	75.0
2414	The Wall	RAtt.	2017	The Wall	tt6060874	2016	73.0
2415	The Wall	RAtt.	2017	The Wall	tt6567616	2016	34.0
2421	Friend Request	ENTMP	2017	Friend Request	tt3352390	2016	92.0
2422	Friend Request	ENTMP	2017	Friend Request	tt4859526	2015	106.0
2427	Colossal	Neon	2017	Colossal	tt4680182	2016	109.0
2429	A Gentleman	FIP	2017	A Gentleman	tt8289324	2018	70.0
2431	Kedi	Osci.	2017	Kedi	tt4420704	2016	79.0
2444	The Square	Magn.	2017	The Square	tt5511124	2016	75.0
2448	Churchill	Cohen	2017	Churchill	tt7258476	2015	65.0
2466	Happy End	SPC	2017	Happy End	tt1821449	2011	97.0
2468	Happy End	SPC	2017	Happy End: Stupid and Stupider 3	tt8697718	2018	85.0
2476	Graduation	IFC	2017	Graduation	tt9778150	2018	112.0
2479	The Journey	IFC	2017	The Journey	tt2099755	2011	13.0
2480	The Journey	IFC	2017	The Journey	tt3025936	2014	112.0
2481	The Journey	IFC	2017	The Journey	tt3144678	2014	103.0
2482	The Journey	IFC	2017	The Journey	tt4826674	2016	94.0
2483	The Journey	IFC	2017	The Journey	tt4900076	2015	91.0
2484	The Journey	IFC	2017	The Journey	tt5270158	2015	26.0
2485	The Journey	IFC	2017	The Journey	tt6067118	2013	90.0
2488	The Journey	IFC	2017	The Journey	tt8189070	2018	46.0
2490	The Void	Scre.	2017	The Void	tt4255304	2016	90.0
2491	The Void	Scre.	2017	The Void	tt6267458	2016	86.0
2492	The Void	Scre.	2017	The Void	tt7282030	2018	NaN
2501	The Mayor	WGUSA	2017	The Mayor	tt3044022	2013	NaN

	title	studio	year	primary_title	movie_id	start_year	runtime_minutes
2506	Tomorrow	UTMW	2017	Tomorrow	tt2831326	2015	115.0
2507	Tomorrow	UTMW	2017	Tomorrow	tt4800294	2016	75.0
2512	The Daughter	KL	2017	The Daughter	tt3922816	2015	96.0
2524	Blind	VE	2017	Blind	tt2616810	2014	96.0
2525	Blind	VE	2017	Blind	tt7390494	2018	NaN
2530	Oro	Sony	2017	Oro	tt2732680	2013	47.0
2532	Oro	Sony	2017	Oro	tt6271528	2016	110.0
2569	The Mule	WB	2018	The Mule	tt2130270	2014	103.0
2581	First Man	Uni.	2018	First Man	tt7027210	2017	92.0
2583	Alpha	Studio 8	2018	Alpha	tt3166734	2015	80.0
2585	Alpha	Studio 8	2018	Alpha	tt6313896	2016	90.0
2589	Truth or Dare	Uni.	2018	Truth or Dare	tt2125681	2013	85.0
2590	Truth or Dare	Uni.	2018	Truth or Dare	tt2762738	2013	84.0
2603	Tag	WB (NL)	2018	Tag	tt4049714	2014	5.0
2604	Tag	WB (NL)	2018	Tag	tt5908018	2016	76.0
2605	Tag	WB (NL)	2018	Tag	tt8430862	2017	46.0
2609	Widows	Fox	2018	Widows	tt4907156	2015	79.0
2621	Life of the Party	WB (NL)	2018	Life of the Party	tt3156890	2013	NaN
2623	Life of the Party	WB (NL)	2018	Life of the Party	tt6897484	2017	115.0
2625	Adrift	STX	2018	Adrift	tt4608626	2015	85.0
2631	Slender Man	SGem	2018	Slender Man	tt5897302	2013	52.0
2666	Upgrade	BH Tilt	2018	Upgrade	tt6739824	2016	NaN
2676	Gringo	STX	2018	Gringo	tt4180008	2016	80.0
2677	Gringo	STX	2018	Gringo	tt4772418	2015	89.0
2718	Beast	RAtt.	2018	Beast	tt1572501	2011	83.0
2719	Beast	RAtt.	2018	Beast	tt4251006	2015	94.0
2720	Beast	RAtt.	2018	Beast	tt5628302	2017	107.0
2723	Foxtrot	SPC	2018	Foxtrot	tt6896536	2017	113.0
2729	Mountain	Greenwich	2018	Mountain	tt6203570	2017	74.0
2737	The Guardians	MBox	2018	The Guardians	tt6901956	2017	46.0
2751	In Between	FM	2018	In Between	tt4303000	2014	26.0
2752	In Between	FM	2018	In Between	tt9028484	2017	NaN
2771	The Escape	IFC	2018	The Escape	tt6069126	2017	101.0
2773	Souvenir	Strand	2018	Souvenir	tt2389092	2014	86.0
2774	Souvenir	Strand	2018	Souvenir	tt3478898	2014	86.0

It's clear that IMDB has a much more exhaustive that the BOM database. The list above shows repeated BOM data on the left side matched to multiple movies from IMDB on the right based on the fact that they have the same title. However, we need to drop these because they are clearly not the same films, as the release years differ, IMDB has multiple entries in their identifier, the movie_id column, and the run_time minutes differ, but they are all listed as the same movie under BOM.

It's worth noting that most of these films are still matched, however, they are matched to matching films. The BOM data is not lost.

I will move forward with the drop.

In [22]:
```
#Resetting row maximum
pd.set_option("display.max_rows", 30, "display.max_columns", None)

#Dropping the incorrect matches.
movies_analyzed.drop(movies_to_drop.index, inplace=True)
movies_analyzed.drop(movies_analyzed[(movies_analyzed['year'] != movies_analyzed['start_ye
bom_and_imdb = movies_analyzed.reset_index()
movies_analyzed
```

Out[22]:

	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title	original_title	start_year
0	Toy Story 3	BV	415000000.0	652000000	2010	tt0435761	Toy Story 3	Toy Story 3	2010
1	Inception	WB	292600000.0	535700000	2010	tt1375666	Inception	Inception	2010
2	Shrek Forever After	P/DW	238700000.0	513900000	2010	tt0892791	Shrek Forever After	Shrek Forever After	2010
3	The Twilight Saga: Eclipse	Sum.	300500000.0	398000000	2010	tt1325004	The Twilight Saga: Eclipse	The Twilight Saga: Eclipse	2010
4	Iron Man 2	Par.	312400000.0	311500000	2010	tt1228705	Iron Man 2	Iron Man 2	2010
...	...	...	...	...	...	...	...	...	...
2756	The House That Jack Built	IFC	88000.0	NaN	2018	tt4003440	The House That Jack Built	The House That Jack Built	2018
2761	Helicopter Eela	Eros	72000.0	NaN	2018	tt8427036	Helicopter Eela	Helicopter Eela	2018
2765	Oolong Courtyard	CL	37700.0	NaN	2018	tt8549902	Oolong Courtyard: KungFu School	Oolong Courtyard	2018
2768	The Workshop	Strand	22100.0	NaN	2018	tt7405478	The Workshop	The Workshop	2018
2775	An Actor Prepares	Grav.	1700.0	NaN	2018	tt5718046	An Actor Prepares	An Actor Prepares	2018

1667 rows × 11 columns

I'm now going to merge in The Movie Database columns. To do that, I need to rename some of the columns so I can tell where each column came from after the merge.

In [23]:
```
tmdb.rename(columns = {'Unnamed: 0':'tmdb_index','original_title':'tmdb_original_title','ti
```

Now I have to fix another error in the film Upside Down.

In [24]:
```
tmdb.at[7969,'tmdb_release_date'] = '2012-08-31'
```

Now I'm going convert the release dates to release years, in integers, in a new column. This will allow me to merge and compare with the IMDB and Box Office Mojo data.

In [25]:
```
tmdb['tmdb_year'] = tmdb['tmdb_release_date'].map(lambda x : x[0:4])
tmdb['tmdb_year'] = tmdb['tmdb_year'].astype(int)
```

Merging the previously merged dataframe of IMDB and Box Office Mojo data with the tmdb dataframe. I'll now refer to this database as movies_analyzed.

In [26]:
```
movies_analyzed = pd.merge(bom_and_imdb,tmdb, how='inner',left_on='title', right_on='tmdb_t
movies_analyzed
```

Out[26]:

	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title	original_title	sta
0	0	Toy Story 3	BV	415000000.0	652000000	2010	tt0435761	Toy Story 3	Toy Story 3	
1	1	Inception	WB	292600000.0	535700000	2010	tt1375666	Inception	Inception	
2	2	Shrek Forever After	P/DW	238700000.0	513900000	2010	tt0892791	Shrek Forever After	Shrek Forever After	
3	3	The Twilight Saga: Eclipse	Sum.	300500000.0	398000000	2010	tt1325004	The Twilight Saga: Eclipse	The Twilight Saga: Eclipse	
4	4	Iron Man 2	Par.	312400000.0	311500000	2010	tt1228705	Iron Man 2	Iron Man 2	
...	...	...	...	...	...	...	...	...	...	
1770	2739	The Guardians	MBox	177000.0	NaN	2018	tt8150132	The Guardians	The Guardians	
1771	2742	Museo	Vita.	149000.0	NaN	2018	tt4958448	Museo	Museo	
1772	2756	The House That Jack Built	IFC	88000.0	NaN	2018	tt4003440	The House That Jack Built	The House That Jack Built	
1773	2756	The House That Jack Built	IFC	88000.0	NaN	2018	tt4003440	The House That Jack Built	The House That Jack Built	
1774	2775	An Actor Prepares	Grav.	1700.0	NaN	2018	tt5718046	An Actor Prepares	An Actor Prepares	

1775 rows × 23 columns

I will review the merged data.

In [27]:
```
movies_analyzed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1775 entries, 0 to 1774
Data columns (total 23 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   index                1775 non-null   int64  
 1   title                1775 non-null   object 
 2   studio               1775 non-null   object 
 3   domestic_gross       1767 non-null   float64
 4   foreign_gross        1344 non-null   object 
 5   year                 1775 non-null   int64  
 6   movie_id             1775 non-null   object 
 7   primary_title        1775 non-null   object 
 8   original_title       1775 non-null   object 
 9   start_year           1775 non-null   int64  
 10  runtime_minutes      1762 non-null   float64
 11  genres               1773 non-null   object 
 12  tmdb_index           1775 non-null   int64  
 13  genre_ids            1775 non-null   object 
 14  tmdb_id              1775 non-null   int64  
 15  original_language    1775 non-null   object 
 16  tmdb_original_title  1775 non-null   object 
 17  popularity           1775 non-null   float64
 18  tmdb_release_date    1775 non-null   object 
 19  tmdb_title           1775 non-null   object 
 20  vote_average         1775 non-null   float64
 21  vote_count           1775 non-null   int64  
 22  tmdb_year            1775 non-null   int32  
dtypes: float64(4), int32(1), int64(6), object(12)
memory usage: 325.9+ KB
```

Once again, I will check for inconsistent debut year information and drop any films with mistakes.

In [28]:
```
#I need to review all these films
pd.set_option("display.max_rows", None, "display.max_columns", None)

movies_to_drop = movies_analyzed[movies_analyzed['title'].duplicated()]
movies_to_drop.drop(movies_to_drop[(movies_to_drop['year'] == movies_to_drop['tmdb_year'])]
movies_to_drop[['primary_title','movie_id','start_year','runtime_minutes','tmdb_title','tm
```

```
C:\Users\g_osb\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py:4163: Sett
ingWithCopyWarning:
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-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  return super().drop(
```

Out[28]:

	primary_title	movie_id	start_year	runtime_minutes	tmdb_title	tmdb_year
15	Robin Hood	tt0955308	2010	140.0	Robin Hood	2013
16	Robin Hood	tt0955308	2010	140.0	Robin Hood	2018
22	Salt	tt0944835	2010	100.0	Salt	2012
39	Unstoppable	tt0477080	2010	98.0	Unstoppable	2013
65	Edge of Darkness	tt1226273	2010	117.0	Edge of Darkness	2015
102	Leap Year	tt1216492	2010	100.0	Leap Year	2011
107	Let Me In	tt1228987	2010	116.0	Let Me In	2016
113	Another Year	tt1431181	2010	129.0	Another Year	2014
132	Stone	tt1423995	2010	105.0	Stone	2012
146	Monsters	tt1470827	2010	94.0	Monsters	2015
156	Flipped	tt0817177	2010	90.0	Flipped	2015
165	We Are Family	tt1428459	2010	115.0	We Are Family	2012
310	Shame	tt1723811	2011	101.0	Shame	2014
311	Shame	tt1723811	2011	101.0	Shame	1968
331	The Double	tt1646980	2011	98.0	The Double	2012
332	The Double	tt1646980	2011	98.0	The Double	2014
343	A Better Life	tt1554091	2011	98.0	A Better Life	2015
354	The Future	tt1235170	2011	91.0	The Future	2013
433	Project X	tt1636826	2012	88.0	Project X	2016
478	The Master	tt1560747	2012	138.0	The Master	2015
489	Gone	tt2230954	2012	50.0	Gone	2011
499	The Apparition	tt1433822	2012	83.0	The Apparition	2018
508	The Collection	tt1748227	2012	82.0	The Collection	2017
512	Won't Back Down	tt1870529	2012	121.0	Won't Back Down	2014
536	For a Good Time, Call...	tt1996264	2012	85.0	For a Good Time, Call...	2017
552	Elena	tt2475154	2012	80.0	Elena	2014

	primary_title	movie_id	start_year	runtime_minutes	tmdb_title	tmdb_year
553	Elena	tt2475154	2012	80.0	Elena	2014
686	Paranoia	tt1413495	2013	106.0	Paranoia	2016
687	Paranoia	tt1413495	2013	106.0	Paranoia	2011
700	Believe	tt2009606	2013	96.0	Believe	2016
804	The Judge	tt1872194	2014	141.0	The Judge	2018
854	Inherent Vice	tt1791528	2014	148.0	Inherent Vice	2015
880	Kick	tt2372222	2014	146.0	Kick	2015
921	Wolves	tt1403241	2014	91.0	Wolves	2016
922	Wolves	tt1403241	2014	91.0	Wolves	2016
990	Black Mass	tt1355683	2015	123.0	Black Mass	2017
997	Spotlight	tt1895587	2015	129.0	Spotlight	2018
1000	Spotlight	tt7785302	2015	99.0	Spotlight	2018
1021	Brooklyn	tt2381111	2015	117.0	Brooklyn	2017
1041	Gnome Alone	tt3381068	2015	94.0	Legend	1986
1043	Gnome Alone	tt3381068	2015	94.0	Legend	1986
1045	Legend	tt3569230	2015	132.0	Legend	1986
1046	Legend	tt3569230	2015	132.0	Legend	1986
1048	Legend	tt3569230	2015	132.0	Legend	1986
1115	Youth	tt3312830	2015	124.0	Youth	2017
1121	Truth	tt3859076	2015	125.0	Truth	2016
1158	Eden	tt5975878	2015	90.0	Eden	2014
1162	Eden	tt5975878	2015	90.0	Eden	2018
1165	Flowers	tt4248930	2015	79.0	Flowers	2018
1179	Extraction	tt4382872	2015	92.0	Extraction	2013
1275	Nerve	tt3531824	2016	96.0	Nerve	2011
1309	The Forest	tt4982356	2016	109.0	The Forest	2011
1419	The Other Side	tt4942986	2016	NaN	The Other Side	2014
1574	The Wall	tt6845582	2017	5.0	The Wall	2013
1576	The Wall	tt7578246	2017	NaN	The Wall	2013
1582	The Leisure Seeker	tt3741632	2017	112.0	The Leisure Seeker	2018
1606	The Void	tt7318900	2017	110.0	The Void	2016
1607	The Void	tt7318900	2017	110.0	The Void	2016
1617	Singularity	tt7312940	2017	92.0	Singularity	2015
1669	Truth or Dare	tt6772950	2018	100.0	Truth or Dare	2013
1670	Truth or Dare	tt6772950	2018	100.0	Truth or Dare	2014
1671	Truth or Dare	tt6772950	2018	100.0	Truth or Dare	2017
1673	Truth or Dare	tt6869948	2018	92.0	Truth or Dare	2012
1674	Truth or Dare	tt6869948	2018	92.0	Truth or Dare	2013

	primary_title	movie_id	start_year	runtime_minutes	tmdb_title	tmdb_year
1675	Truth or Dare	tt6869948	2018	92.0	Truth or Dare	2014
1676	Truth or Dare	tt6869948	2018	92.0	Truth or Dare	2017
1770	The Guardians	tt8150132	2018	70.0	The Guardians	2012

Once again, the TMDB has other films with the same title that do not match the release year of the now merged BOM/IMDB data. Some of the IMDB data and TMDB could have matched up, but since those films are not in the BOM data I'm still comfortable dropping them. Therefore, these films need to be dropped too.

In [29]:
```
#Resetting row maximum
pd.set_option("display.max_rows", 30, "display.max_columns", None)

movies_analyzed.drop(movies_to_drop.index, inplace=True)
movies_analyzed.drop(movies_analyzed[(movies_analyzed['year'] != movies_analyzed['tmdb_yea
bom_imdb_and_tmdb = movies_analyzed.reset_index()
bom_imdb_and_tmdb
```

Out[29]:

	level_0	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title	origin
0	0	0	Toy Story 3	BV	415000000.0	652000000	2010	tt0435761	Toy Story 3	Toy
1	1	1	Inception	WB	292600000.0	535700000	2010	tt1375666	Inception	In
2	2	2	Shrek Forever After	P/DW	238700000.0	513900000	2010	tt0892791	Shrek Forever After	Forev
3	3	3	The Twilight Saga: Eclipse	Sum.	300500000.0	398000000	2010	tt1325004	The Twilight Saga: Eclipse	The
4	4	4	Iron Man 2	Par.	312400000.0	311500000	2010	tt1228705	Iron Man 2	Iron
...	...	...	...	...	...	...	...	...	...	
1631	1767	2731	Time Freak	Grindstone	10000.0	256000	2018	tt6769280	Time Freak	Time
1632	1768	2732	What They Had	BST	260000.0	NaN	2018	tt6662736	What They Had	Wha
1633	1771	2742	Museo	Vita.	149000.0	NaN	2018	tt4958448	Museo	
1634	1773	2756	The House That Jack Built	IFC	88000.0	NaN	2018	tt4003440	The House That Jack Built	The Th
1635	1774	2775	An Actor Prepares	Grav.	1700.0	NaN	2018	tt5718046	An Actor Prepares	A Pr

1636 rows × 24 columns

Now I'll merge The Numbers into my movies_analyzed dataframe. First I need to edit The Numbers database

so I can merge it with movies_analyzed easily. Like earlier, I'll start by editing the column names.

In [30]:
```
tn.rename(columns = {'id':'tn_id','movie':'tn_title',
                     'domestic_gross':'tn_domestic_gross',
                     'worldwide_gross': 'tn_worldwide_gross',
                     'release_date':'tn_release_date'},inplace=True)
```

Now I'll fix yet another error in the film Upside Down.

In [31]:
```
tn.at[1203,'tn_release_date'] = 'Aug 31 2012'
```

Now, similarly to earlier, I'll convert the dates to the release year as an integer in new column.

In [32]:
```
tn['tn_year'] = tn['tn_release_date'].map(lambda x : x[-4:])
tn['tn_year'] = tn['tn_year'].astype(int)
```

Now, I'll merge the movies_analyzed dataframe with the newly formatted The Numbers dataframe.

In [33]:
```
movies_analyzed = pd.merge(bom_imdb_and_tmdb,tn, how='inner',left_on='title', right_on='tn_
movies_analyzed
```

Out[33]:

	level_0	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title	origi
0	0	0	Toy Story 3	BV	415000000.0	652000000	2010	tt0435761	Toy Story 3	Toy
1	1	1	Inception	WB	292600000.0	535700000	2010	tt1375666	Inception	In
2	2	2	Shrek Forever After	P/DW	238700000.0	513900000	2010	tt0892791	Shrek Forever After	Forev
3	3	3	The Twilight Saga: Eclipse	Sum.	300500000.0	398000000	2010	tt1325004	The Twilight Saga: Eclipse	The
4	4	4	Iron Man 2	Par.	312400000.0	311500000	2010	tt1228705	Iron Man 2	Iron
...	...	...	...	...	...	...	...	...	...	
1157	1741	2682	Suspiria	Amazon	2500000.0	5400000	2018	tt1034415	Suspiria	
1158	1743	2686	The Hurricane Heist	ENTMP	6100000.0	NaN	2018	tt5360952	The Hurricane Heist	H
1159	1745	2688	Destroyer	Annapurna	1500000.0	4000000	2018	tt7137380	Destroyer	D
1160	1749	2694	Gotti	VE	4300000.0	NaN	2018	tt1801552	Gotti	
1161	1759	2710	Mandy	RLJ	1200000.0	NaN	2018	tt6998518	Mandy	

1162 rows × 31 columns

I will review the merged data.

In [34]:
```
movies_analyzed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1162 entries, 0 to 1161
Data columns (total 31 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   level_0              1162 non-null   int64  
 1   index                1162 non-null   int64  
 2   title                1162 non-null   object 
 3   studio               1162 non-null   object 
 4   domestic_gross       1161 non-null   float64
 5   foreign_gross        1044 non-null   object 
 6   year                 1162 non-null   int64  
 7   movie_id             1162 non-null   object 
 8   primary_title        1162 non-null   object 
 9   original_title       1162 non-null   object 
 10  start_year           1162 non-null   int64  
 11  runtime_minutes      1157 non-null   float64
 12  genres               1161 non-null   object 
 13  tmdb_index           1162 non-null   int64  
 14  genre_ids            1162 non-null   object 
 15  tmdb_id              1162 non-null   int64  
 16  original_language    1162 non-null   object 
 17  tmdb_original_title  1162 non-null   object 
 18  popularity           1162 non-null   float64
 19  tmdb_release_date    1162 non-null   object 
 20  tmdb_title           1162 non-null   object 
 21  vote_average         1162 non-null   float64
 22  vote_count           1162 non-null   int64  
 23  tmdb_year            1162 non-null   int32  
 24  tn_id                1162 non-null   int64  
 25  tn_release_date      1162 non-null   object 
 26  tn_title             1162 non-null   object 
 27  production_budget    1162 non-null   object 
 28  tn_domestic_gross    1162 non-null   object 
 29  tn_worldwide_gross   1162 non-null   object 
 30  tn_year              1162 non-null   int32  
dtypes: float64(4), int32(2), int64(8), object(17)
memory usage: 281.4+ KB
```

Once again, I'll check for mistakes based on inconsistent years released data.

In [35]:
```
movies_to_drop = movies_analyzed[movies_analyzed['title'].duplicated()]
movies_to_drop.drop(movies_to_drop[(movies_to_drop['year'] == movies_to_drop['tn_year'])].i
movies_to_drop[['primary_title','movie_id','start_year','runtime_minutes','tn_title','tn_ye
```

```
C:\Users\g_osb\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py:4163: Sett
ingWithCopyWarning:
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-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  return super().drop(
```

Out[35]:

	primary_title	movie_id	start_year	runtime_minutes	tn_title	tn_year
10	The Karate Kid	tt1155076	2010	140.0	The Karate Kid	1984
14	Robin Hood	tt0955308	2010	140.0	Robin Hood	2018
189	Unknown	tt1401152	2011	113.0	Unknown	2006
262	We Need to Talk About Kevin	tt1242460	2011	112.0	We Need to Talk About Kevin	2012
482	Snitch	tt0882977	2013	112.0	Snitch	2012
495	Trance	tt1924429	2013	101.0	Trance	2012
597	The Gambler	tt2039393	2014	111.0	The Gambler	1999
671	Fantastic Four	tt1502712	2015	100.0	Fantastic Four	2005
736	Gnome Alone	tt3381068	2015	94.0	Legend	1986
738	Legend	tt3569230	2015	132.0	Legend	1986
740	Legend	tt3569230	2015	132.0	Legend	1986
745	The Lady in the Van	tt3722070	2015	104.0	The Lady in the Van	2016
793	Eden	tt5975878	2015	90.0	Eden	2016
794	Eden	tt5975878	2015	90.0	Eden	2016
795	Eden	tt5975878	2015	90.0	Eden	2016
892	A Monster Calls	tt3416532	2016	108.0	A Monster Calls	2017
1068	The Square	tt4995790	2017	151.0	The Square	2013
1073	The Journey	tt6956150	2017	NaN	The Journey	2003

Again, same titles, different years. We'll drop these.

In [36]:
```
movies_analyzed.drop(movies_to_drop.index, inplace=True)
movies_analyzed.drop(movies_analyzed[(movies_analyzed['year'] != movies_analyzed['tn_year']
```

I'll check to see what titles are still repeating.

In [37]:
```
movies_analyzed['title'].duplicated().sum()
```

Out[37]:
```
122
```

These are minor quirks of the merged dataframes. I need to preview these to see if I do indeed need to drop them. To do this, I need to create a dataframe with just these values, and preserve the index numbers from movies_analyzed. I'll also create three new columns, duped_title, duped_year and drop. If both the name and year are the same, I'll drop the second of the two rows.

First I'll set the max rows to none so I can review all of the duplicates, then create the new columns.

In [38]:
```
pd.set_option("display.max_rows", None, "display.max_columns", None)

duplicated_titles = movies_analyzed[movies_analyzed['movie_id'].duplicated
                                    (keep=False)].sort_values('title')
duplicated_titles = duplicated_titles.iloc[:,1:].reset_index()
duplicated_titles['duped_title'] = duplicated_titles['movie_id'].duplicated()
duplicated_titles['duped_year'] = duplicated_titles['year'].duplicated().astype(bool)
duplicated_titles['drop'] = False

for x in range(len(duplicated_titles)):
    if (duplicated_titles['duped_title'][x] == True and duplicated_titles['duped_year'][x]
        duplicated_titles['drop'][x] = True


duplicated_titles
```

```
<ipython-input-38-08a1f4107a71>:12: 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-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  duplicated_titles['drop'][x] = True
```

Out[38]:

	level_0	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title
0	1014	2305	A Bad Moms Christmas	STX	72100000.0	58400000	2017	tt6359956	A Bad Moms Christmas
1	1015	2305	A Bad Moms Christmas	STX	72100000.0	58400000	2017	tt6359956	A Bad Moms Christmas
2	615	1350	A Most Violent Year	A24	5700000.0	6300000	2014	tt2937898	A Most Violent Year
3	616	1350	A Most Violent Year	A24	5700000.0	6300000	2014	tt2937898	A Most Violent Year
4	951	2186	A Street Cat Named Bob	Cleopatra	82700.0	NaN	2016	tt3606888	A Street Cat Named Bob
5	952	2186	A Street Cat Named Bob	Cleopatra	82700.0	NaN	2016	tt3606888	A Street Cat Named Bob
6	857	1964	Allied	Par.	40100000.0	79400000	2016	tt3640424	Allied
7	856	1964	Allied	Par.	40100000.0	79400000	2016	tt3640424	Allied
8	427	950	American Hustle	Sony	150100000.0	101100000	2013	tt1800241	American Hustle
9	428	950	American Hustle	Sony	150100000.0	101100000	2013	tt1800241	American Hustle
10	1012	2304	American Made	Uni.	51300000.0	83500000	2017	tt3532216	American Made
11	1013	2304	American Made	Uni.	51300000.0	83500000	2017	tt3532216	American Made

	level_0	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title
12	532	1214	American Sniper	WB	350100000.0	197300000	2014	tt2179136	American Sniper
13	533	1214	American Sniper	WB	350100000.0	197300000	2014	tt2179136	American Sniper
14	779	1698	Anomalisa	Par.	3800000.0	1900000	2015	tt2401878	Anomalisa
15	780	1698	Anomalisa	Par.	3800000.0	1900000	2015	tt2401878	Anomalisa
16	836	1941	Arrival	Par.	100500000.0	102800000	2016	tt2543164	Arrival
17	835	1941	Arrival	Par.	100500000.0	102800000	2016	tt2543164	Arrival
18	464	995	August: Osage County	Wein.	37700000.0	36500000	2013	tt1322269	August: Osage County
19	463	995	August: Osage County	Wein.	37700000.0	36500000	2013	tt1322269	August: Osage County
20	1052	2388	Battle of the Sexes	FoxS	12600000.0	NaN	2017	tt4622512	Battle of the Sexes
21	1053	2388	Battle of the Sexes	FoxS	12600000.0	NaN	2017	tt4622512	Battle of the Sexes
22	530	1212	Big Hero 6	BV	222500000.0	435300000	2014	tt2245084	Big Hero 6
23	529	1212	Big Hero 6	BV	222500000.0	435300000	2014	tt2245084	Big Hero 6
24	986	2282	Blade Runner 2049	WB	92100000.0	167200000	2017	tt1856101	Blade Runner 2049
25	987	2282	Blade Runner 2049	WB	92100000.0	167200000	2017	tt1856101	Blade Runner 2049
26	894	2009	Blair Witch	LGF	20800000.0	24400000	2016	tt1540011	Blair Witch
27	895	2009	Blair Witch	LGF	20800000.0	24400000	2016	tt1540011	Blair Witch
28	936	2091	Bleed for This	ORF	5100000.0	NaN	2016	tt1620935	Bleed for This
29	937	2091	Bleed for This	ORF	5100000.0	NaN	2016	tt1620935	Bleed for This
30	107	146	Blue Valentine	Wein.	9700000.0	2600000	2010	tt1120985	Blue Valentine
31	108	146	Blue Valentine	Wein.	9700000.0	2600000	2010	tt1120985	Blue Valentine
32	672	1553	Bridge of Spies	BV	72300000.0	93200000	2015	tt3682448	Bridge of Spies
33	673	1553	Bridge of Spies	BV	72300000.0	93200000	2015	tt3682448	Bridge of Spies
34	752	1641	Burnt	Wein.	13700000.0	23000000	2015	tt2503944	Burnt
35	751	1641	Burnt	Wein.	13700000.0	23000000	2015	tt2503944	Burnt

	level_0	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title
36	748	1636	Carol	Wein.	12700000.0	27600000	2015	tt2402927	Carol
37	747	1636	Carol	Wein.	12700000.0	27600000	2015	tt2402927	Carol
38	963	2257	Coco	BV	209700000.0	597400000	2017	tt7002100	Coco
39	962	2256	Coco	BV	209700000.0	597400000	2017	tt2380307	Coco
40	964	2257	Coco	BV	209700000.0	597400000	2017	tt7002100	Coco
41	961	2256	Coco	BV	209700000.0	597400000	2017	tt2380307	Coco
42	870	1978	Collateral Beauty	WB (NL)	31000000.0	57500000	2016	tt4682786	Collateral Beauty
43	869	1978	Collateral Beauty	WB (NL)	31000000.0	57500000	2016	tt4682786	Collateral Beauty
44	668	1550	Creed	WB (NL)	109800000.0	63800000	2015	tt3076658	Creed
45	669	1550	Creed	WB (NL)	109800000.0	63800000	2015	tt3076658	Creed
46	710	1597	Crimson Peak	Uni.	31100000.0	43600000	2015	tt2554274	Crimson Peak
47	711	1597	Crimson Peak	Uni.	31100000.0	43600000	2015	tt2554274	Crimson Peak
48	472	1005	Dallas Buyers Club	Focus	27300000.0	27900000	2013	tt0790636	Dallas Buyers Club
49	471	1005	Dallas Buyers Club	Focus	27300000.0	27900000	2013	tt0790636	Dallas Buyers Club
50	1006	2298	Darkest Hour	Focus	56500000.0	94400000	2017	tt4555426	Darkest Hour
51	1007	2298	Darkest Hour	Focus	56500000.0	94400000	2017	tt4555426	Darkest Hour
52	854	1963	Deepwater Horizon	LG/S	61400000.0	60400000	2016	tt1860357	Deepwater Horizon
53	855	1963	Deepwater Horizon	LG/S	61400000.0	60400000	2016	tt1860357	Deepwater Horizon
54	943	2097	Denial	BST	4099999.0	NaN	2016	tt5897002	Denial
55	942	2096	Denial	BST	4099999.0	NaN	2016	tt4645330	Denial
56	941	2096	Denial	BST	4099999.0	NaN	2016	tt4645330	Denial
57	944	2097	Denial	BST	4099999.0	NaN	2016	tt5897002	Denial
58	805	1910	Doctor Strange	BV	232600000.0	445100000	2016	tt1211837	Doctor Strange
59	806	1910	Doctor Strange	BV	232600000.0	445100000	2016	tt1211837	Doctor Strange
60	1030	2335	Downsizing	Par.	24400000.0	30600000	2017	tt1389072	Downsizing

	level_0	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title
61	1031	2335	Downsizing	Par.	24400000.0	30600000	2017	tt1389072	Downsizing
62	1046	2368	Father Figures	WB	17500000.0	8100000	2017	tt1966359	Father Figures
63	1047	2368	Father Figures	WB	17500000.0	8100000	2017	tt1966359	Father Figures
64	882	1993	Fences	Par.	57700000.0	6700000	2016	tt2671706	Fences
65	881	1993	Fences	Par.	57700000.0	6700000	2016	tt2671706	Fences
66	984	2275	Ferdinand	Fox	84400000.0	211700000	2017	tt3411444	Ferdinand
67	985	2275	Ferdinand	Fox	84400000.0	211700000	2017	tt3411444	Ferdinand
68	994	2288	Geostorm	WB	33700000.0	187900000	2017	tt1981128	Geostorm
69	993	2288	Geostorm	WB	33700000.0	187900000	2017	tt1981128	Geostorm
70	676	1555	Goosebumps	Sony	80100000.0	70100000	2015	tt1051904	Goosebumps
71	677	1555	Goosebumps	Sony	80100000.0	70100000	2015	tt1051904	Goosebumps
72	840	1946	Hacksaw Ridge	LGF	67200000.0	108100000	2016	tt2119532	Hacksaw Ridge
73	841	1946	Hacksaw Ridge	LGF	67200000.0	108100000	2016	tt2119532	Hacksaw Ridge
74	828	1932	Hidden Figures	Fox	169600000.0	66300000	2016	tt4846340	Hidden Figures
75	829	1932	Hidden Figures	Fox	169600000.0	66300000	2016	tt4846340	Hidden Figures
76	1032	2336	I, Tonya	Neon	30000000.0	23900000	2017	tt5580036	I, Tonya
77	1033	2336	I, Tonya	Neon	30000000.0	23900000	2017	tt5580036	I, Tonya
78	701	1589	In the Heart of the Sea	WB	25000000.0	68900000	2015	tt1390411	In the Heart of the Sea
79	700	1589	In the Heart of the Sea	WB	25000000.0	68900000	2015	tt1390411	In the Heart of the Sea
80	831	1934	Inferno	Sony	34300000.0	185700000	2016	tt3062096	Inferno
81	830	1934	Inferno	Sony	34300000.0	185700000	2016	tt3062096	Inferno
82	966	2259	It	WB (NL)	327500000.0	372900000	2017	tt1396484	It
83	967	2259	It	WB (NL)	327500000.0	372900000	2017	tt1396484	It
84	842	1948	Jack Reacher: Never Go Back	Par.	58700000.0	103400000	2016	tt3393786	Jack Reacher: Never Go Back

	level_0	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title
85	843	1948	Jack Reacher: Never Go Back	Par.	58700000.0	103400000	2016	tt3393786	Jack Reacher: Never Go Back
86	785	1736	Jem and the Holograms	Uni.	2200000.0	149000	2015	tt3614530	Jem and the Holograms
87	784	1736	Jem and the Holograms	Uni.	2200000.0	149000	2015	tt3614530	Jem and the Holograms
88	955	2248	Jumanji: Welcome to the Jungle	Sony	404500000.0	557600000	2017	tt2283362	Jumanji: Welcome to the Jungle
89	956	2248	Jumanji: Welcome to the Jungle	Sony	404500000.0	557600000	2017	tt2283362	Jumanji: Welcome to the Jungle
90	1058	2400	Just Getting Started	BG	6100000.0	1600000	2017	tt5721088	Just Getting Started
91	1059	2400	Just Getting Started	BG	6100000.0	1600000	2017	tt5721088	Just Getting Started
92	969	2260	Justice League	WB	229000000.0	428900000	2017	tt0974015	Justice League
93	968	2260	Justice League	WB	229000000.0	428900000	2017	tt0974015	Justice League
94	814	1915	La La Land	LG/S	151100000.0	295000000	2016	tt3783958	La La Land
95	813	1915	La La Land	LG/S	151100000.0	295000000	2016	tt3783958	La La Land
96	1022	2318	Lady Bird	A24	49000000.0	30000000	2017	tt4925292	Lady Bird
97	1023	2318	Lady Bird	A24	49000000.0	30000000	2017	tt4925292	Lady Bird
98	737	1628	Legend	Uni.	1900000.0	41100000	2015	tt3381068	Gnome Alone
99	739	1631	Legend	Uni.	1900000.0	41100000	2015	tt3569230	Legend
100	735	1628	Legend	Uni.	1900000.0	41100000	2015	tt3381068	Gnome Alone
101	741	1631	Legend	Uni.	1900000.0	41100000	2015	tt3569230	Legend
102	850	1959	Lion	Wein.	51700000.0	88600000	2016	tt3741834	Lion
103	851	1959	Lion	Wein.	51700000.0	88600000	2016	tt3741834	Lion
104	921	2065	Loving	Focus	7800000.0	5200000	2016	tt4669986	Loving
105	922	2065	Loving	Focus	7800000.0	5200000	2016	tt4669986	Loving
106	875	1987	Manchester by the Sea	RAtt.	47700000.0	31300000	2016	tt4034228	Manchester by the Sea
107	876	1987	Manchester by the Sea	RAtt.	47700000.0	31300000	2016	tt4034228	Manchester by the Sea
108	932	2082	Max Steel	ORF	3800000.0	2500000	2016	tt1472584	Max Steel
109	931	2082	Max Steel	ORF	3800000.0	2500000	2016	tt1472584	Max Steel

	level_0	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title
110	927	2074	Miss Sloane	EC	3500000.0	5600000	2016	tt4540710	Miss Sloane
111	926	2074	Miss Sloane	EC	3500000.0	5600000	2016	tt4540710	Miss Sloane
112	807	1911	Moana	BV	248800000.0	394600000	2016	tt3521164	Moana
113	808	1911	Moana	BV	248800000.0	394600000	2016	tt3521164	Moana
114	541	1224	Noah	Par.	101200000.0	261399999	2014	tt1959490	Noah
115	540	1224	Noah	Par.	101200000.0	261399999	2014	tt1959490	Noah
116	905	2021	Nocturnal Animals	Focus	10700000.0	19600000	2016	tt4550098	Nocturnal Animals
117	904	2021	Nocturnal Animals	Focus	10700000.0	19600000	2016	tt4550098	Nocturnal Animals
118	873	1986	Ouija: Origin of Evil	Uni.	35100000.0	46600000	2016	tt4361050	Ouija: Origin of Evil
119	874	1986	Ouija: Origin of Evil	Uni.	35100000.0	46600000	2016	tt4361050	Ouija: Origin of Evil
120	947	2116	Paterson	BST	2200000.0	NaN	2016	tt5247022	Paterson
121	948	2116	Paterson	BST	2200000.0	NaN	2016	tt5247022	Paterson
122	888	2001	Patriots Day	LGF	31900000.0	18700000	2016	tt4572514	Patriots Day
123	889	2001	Patriots Day	LGF	31900000.0	18700000	2016	tt4572514	Patriots Day
124	1000	2293	Pitch Perfect 3	Uni.	104900000.0	80500000	2017	tt4765284	Pitch Perfect 3
125	999	2293	Pitch Perfect 3	Uni.	104900000.0	80500000	2017	tt4765284	Pitch Perfect 3
126	1050	2386	Roman J. Israel, Esq.	Sony	12000000.0	1100000	2017	tt6000478	Roman J. Israel, Esq.
127	1051	2386	Roman J. Israel, Esq.	Sony	12000000.0	1100000	2017	tt6000478	Roman J. Israel, Esq.
128	753	1643	Room	A24	14700000.0	20700000	2015	tt3170832	Room
129	754	1643	Room	A24	14700000.0	20700000	2015	tt3170832	Room
130	705	1593	Sicario	LGF	46900000.0	38000000	2015	tt3397884	Sicario
131	706	1593	Sicario	LGF	46900000.0	38000000	2015	tt3397884	Sicario
132	809	1912	Sing	Uni.	270400000.0	363800000	2016	tt3470600	Sing
133	810	1912	Sing	Uni.	270400000.0	363800000	2016	tt3470600	Sing
134	640	1510	Spectre	Sony	200100000.0	680600000	2015	tt2379713	Spectre
135	639	1510	Spectre	Sony	200100000.0	680600000	2015	tt2379713	Spectre

	level_0	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title
136	696	1587	Spotlight	ORF	45100000.0	53200000	2015	tt1895587	Spotlight
137	697	1587	Spotlight	ORF	45100000.0	53200000	2015	tt1895587	Spotlight
138	698	1588	Spotlight	ORF	45100000.0	53200000	2015	tt7785302	Spotlight
139	699	1588	Spotlight	ORF	45100000.0	53200000	2015	tt7785302	Spotlight
140	755	1644	Steve Jobs	Uni.	17800000.0	16700000	2015	tt2080374	Steve Jobs
141	756	1644	Steve Jobs	Uni.	17800000.0	16700000	2015	tt2080374	Steve Jobs
142	664	1546	Straight Outta Compton	Uni.	161200000.0	40400000	2015	tt1398426	Straight Outta Compton
143	665	1546	Straight Outta Compton	Uni.	161200000.0	40400000	2015	tt1398426	Straight Outta Compton
144	844	1951	The Accountant	WB	86300000.0	68900000	2016	tt2140479	The Accountant
145	845	1951	The Accountant	WB	86300000.0	68900000	2016	tt2140479	The Accountant
146	1043	2359	The Disaster Artist	A24	21100000.0	8700000	2017	tt3521126	The Disaster Artist
147	1042	2359	The Disaster Artist	A24	21100000.0	8700000	2017	tt3521126	The Disaster Artist
148	913	2043	The Edge of Seventeen	STX	14400000.0	4400000	2016	tt1878870	The Edge of Seventeen
149	914	2043	The Edge of Seventeen	STX	14400000.0	4400000	2016	tt1878870	The Edge of Seventeen
150	1009	2300	The Foreigner	STX	34400000.0	111000000	2017	tt1615160	The Foreigner
151	1010	2300	The Foreigner	STX	34400000.0	111000000	2017	tt1615160	The Foreigner
152	898	2015	The Forest	Focus	26600000.0	11000000	2016	tt3387542	The Forest
153	901	2016	The Forest	Focus	26600000.0	11000000	2016	tt4982356	The Forest
154	900	2016	The Forest	Focus	26600000.0	11000000	2016	tt4982356	The Forest
155	899	2015	The Forest	Focus	26600000.0	11000000	2016	tt3387542	The Forest
156	654	1529	The Good Dinosaur	BV	123100000.0	209100000	2015	tt1979388	The Good Dinosaur
157	653	1529	The Good Dinosaur	BV	123100000.0	209100000	2015	tt1979388	The Good Dinosaur
158	976	2266	The Greatest Showman	Fox	174300000.0	260700000	2017	tt1485796	The Greatest Showman
159	975	2266	The Greatest Showman	Fox	174300000.0	260700000	2017	tt1485796	The Greatest Showman

	level_0	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title
160	675	1554	The Hateful Eight	Wein.	54100000.0	101600000	2015	tt3460252	The Hateful Eight
161	674	1554	The Hateful Eight	Wein.	54100000.0	101600000	2015	tt3460252	The Hateful Eight
162	643	1519	The Hunger Games: Mockingjay - Part 2	LGF	281700000.0	371700000	2015	tt1951266	The Hunger Games: Mockingjay - Part 2
163	642	1519	The Hunger Games: Mockingjay - Part 2	LGF	281700000.0	371700000	2015	tt1951266	The Hunger Games: Mockingjay - Part 2
164	679	1556	The Last Witch Hunter	LG/S	27400000.0	119600000	2015	tt1618442	The Last Witch Hunter
165	678	1556	The Last Witch Hunter	LG/S	27400000.0	119600000	2015	tt1618442	The Last Witch Hunter
166	645	1520	The Martian	Fox	228400000.0	401700000	2015	tt3659388	The Martian
167	644	1520	The Martian	Fox	228400000.0	401700000	2015	tt3659388	The Martian
168	726	1618	The Night Before	Sony	43000000.0	9300000	2015	tt3530002	The Night Before
169	729	1620	The Night Before	Sony	43000000.0	9300000	2015	tt6353886	The Night Before
170	728	1620	The Night Before	Sony	43000000.0	9300000	2015	tt6353886	The Night Before
171	727	1618	The Night Before	Sony	43000000.0	9300000	2015	tt3530002	The Night Before
172	658	1535	The Peanuts Movie	Fox	130199999.0	116100000	2015	tt2452042	The Peanuts Movie
173	659	1535	The Peanuts Movie	Fox	130199999.0	116100000	2015	tt2452042	The Peanuts Movie
174	1002	2294	The Post	Fox	81900000.0	97900000	2017	tt6294822	The Post
175	1001	2294	The Post	Fox	81900000.0	97900000	2017	tt6294822	The Post
176	648	1522	The Revenant	Fox	183600000.0	349300000	2015	tt1663202	The Revenant
177	647	1522	The Revenant	Fox	183600000.0	349300000	2015	tt1663202	The Revenant
178	998	2292	The Shape of Water	FoxS	63900000.0	131400000	2017	tt5580390	The Shape of Water
179	997	2292	The Shape of Water	FoxS	63900000.0	131400000	2017	tt5580390	The Shape of Water
180	693	1585	The Visit	Uni.	65200000.0	33200000	2015	tt3567288	The Visit
181	695	1586	The Visit	Uni.	65200000.0	33200000	2015	tt3833746	The Visit: An Alien Encounter
182	694	1586	The Visit	Uni.	65200000.0	33200000	2015	tt3833746	The Visit: An Alien Encounter

	level_0	index	title	studio	domestic_gross	foreign_gross	year	movie_id	primary_title
183	692	1585	The Visit	Uni.	65200000.0	33200000	2015	tt3567288	The Visit
184	958	2251	Thor: Ragnarok	BV	315100000.0	538900000	2017	tt3501632	Thor: Ragnarok
185	959	2251	Thor: Ragnarok	BV	315100000.0	538900000	2017	tt3501632	Thor: Ragnarok
186	1004	2297	Three Billboards Outside Ebbing, Missouri	FoxS	54500000.0	104700000	2017	tt5027774	Three Billboards Outside Ebbing, Missouri
187	1005	2297	Three Billboards Outside Ebbing, Missouri	FoxS	54500000.0	104700000	2017	tt5027774	Three Billboards Outside Ebbing, Missouri
188	821	1921	Trolls	Fox	153700000.0	193200000	2016	tt1679335	Trolls
189	820	1921	Trolls	Fox	153700000.0	193200000	2016	tt1679335	Trolls
190	757	1645	Victor Frankenstein	Fox	5800000.0	28500000	2015	tt1976009	Victor Frankenstein
191	758	1645	Victor Frankenstein	Fox	5800000.0	28500000	2015	tt1976009	Victor Frankenstein
192	861	1967	Why Him?	Fox	60300000.0	57800000	2016	tt4501244	Why Him?
193	860	1967	Why Him?	Fox	60300000.0	57800000	2016	tt4501244	Why Him?
194	982	2274	Wonder	LGF	132400000.0	173500000	2017	tt2543472	Wonder
195	983	2274	Wonder	LGF	132400000.0	173500000	2017	tt2543472	Wonder
196	770	1676	Woodlawn	PFR	14400000.0	NaN	2015	tt4183692	Woodlawn
197	771	1676	Woodlawn	PFR	14400000.0	NaN	2015	tt4183692	Woodlawn

Resetting the max rows

In [39]:
```
pd.set_option("display.max_rows", 60, "display.max_columns", None)
```

Indeed, if both the title and year are identical, one needs to be dropped. I'll move forward with the drop and check to see if there are still any duplicated movie titles.

In [40]:
```
duplicated_titles.drop(duplicated_titles[duplicated_titles['drop'] == False].index, inplace
movies_analyzed.drop(duplicated_titles['level_0'], inplace=True)

movies_analyzed['title'].duplicated().sum()
```

Out[40]: 23

Funny enough, there are twenty three pairs, or 46 films, that have the same title, were released the same year, but yet have different runtimes and different entries on IMDB. These need to be removed selectively. First, I'll look at the 46 films.

In [41]:
```
test = movies_analyzed[movies_analyzed['title'].duplicated(keep=False)].sort_values('title
test
```

Out[41]:

	title	studio	year	movie_id	runtime_minutes	genres
207	Abduction	LGF	2011	tt1600195	106.0	Action,Mystery,Thriller
208	Abduction	LGF	2011	tt2447982	84.0	Horror,Thriller
614	Addicted	LGF	2014	tt3435418	97.0	Documentary,Music
613	Addicted	LGF	2014	tt2205401	106.0	Drama,Thriller
607	Big Eyes	Wein.	2014	tt4317898	NaN	Documentary
606	Big Eyes	Wein.	2014	tt1126590	106.0	Biography,Crime,Drama
53	Burlesque	SGem	2010	tt1126591	119.0	Drama,Music,Musical
54	Burlesque	SGem	2010	tt1586713	NaN	Drama
962	Coco	BV	2017	tt2380307	105.0	Adventure,Animation,Comedy
963	Coco	BV	2017	tt7002100	98.0	Horror
111	Cyrus	FoxS	2010	tt1327709	87.0	Crime,Horror,Mystery
112	Cyrus	FoxS	2010	tt1336617	91.0	Comedy,Drama,Romance
942	Denial	BST	2016	tt4645330	109.0	Biography,Drama
943	Denial	BST	2016	tt5897002	93.0	Documentary
737	Legend	Uni.	2015	tt3381068	94.0	Horror
739	Legend	Uni.	2015	tt3569230	132.0	Biography,Crime,Drama
848	Lights Out	WB (NL)	2016	tt5328340	90.0	Documentary
847	Lights Out	WB (NL)	2016	tt4786282	81.0	Drama,Horror,Mystery
686	Sisters	Uni.	2015	tt1850457	118.0	Comedy
687	Sisters	Uni.	2015	tt4793074	53.0	Biography,Documentary,Music
698	Spotlight	ORF	2015	tt7785302	99.0	Drama
696	Spotlight	ORF	2015	tt1895587	129.0	Crime,Drama
1055	Stronger	RAtt.	2017	tt3881784	119.0	Biography,Drama
1056	Stronger	RAtt.	2017	tt5738152	125.0	Drama
187	The Artist	Wein.	2011	tt1825978	100.0	Thriller
186	The Artist	Wein.	2011	tt1655442	100.0	Comedy,Drama,Romance
41	The Bounty Hunter	Sony	2010	tt1038919	110.0	Action,Comedy,Romance
42	The Bounty Hunter	Sony	2010	tt1472211	NaN	None
898	The Forest	Focus	2016	tt3387542	93.0	Horror,Mystery,Thriller
901	The Forest	Focus	2016	tt4982356	109.0	Drama,Fantasy,Horror
729	The Night Before	Sony	2015	tt6353886	86.0	Documentary
726	The Night Before	Sony	2015	tt3530002	101.0	Adventure,Comedy,Fantasy
138	The Tempest	Mira.	2010	tt1683003	131.0	Drama
137	The Tempest	Mira.	2010	tt1274300	110.0	Comedy,Drama,Fantasy

	title	studio	year	movie_id	runtime_minutes	genres
695	The Visit	Uni.	2015	tt3833746	83.0	Documentary
693	The Visit	Uni.	2015	tt3567288	94.0	Horror,Mystery,Thriller
721	The Walk	TriS	2015	tt3488710	123.0	Adventure,Biography,Drama
720	The Walk	TriS	2015	tt2159988	89.0	Crime,Thriller
1065	The Wall	RAtt.	2017	tt7578246	NaN	Documentary
1064	The Wall	RAtt.	2017	tt6845582	5.0	Documentary
1063	The Wall	RAtt.	2017	tt4218696	88.0	Action,Drama,Thriller
1114	Truth or Dare	Uni.	2018	tt6869948	92.0	Comedy,Drama,Romance
1113	Truth or Dare	Uni.	2018	tt6772950	100.0	Horror,Thriller
510	Upside Down	MNE	2012	tt1374992	109.0	Drama,Fantasy,Romance
511	Upside Down	MNE	2012	tt2105043	81.0	Drama

What we see here is IMDB data that has multiple film entries with the same title and same year. I need to go into the various databases, BOM, IMDB, TMDB and TN, and check to see which IMDB movie matches the other three databases. This won't take too terribly long, but I cannot think of another way to do it other than hard coding which movies I wish to drop.

In [42]:
```
#These are the IMDB movie_id numbers for the films that need to be deleted.
imposter_films = ['tt2447982', 'tt3435418', 'tt4317898', 'tt1586713',
                  'tt7002100', 'tt1327709', 'tt5897002', 'tt3381068',
                  'tt5328340', 'tt4793074', 'tt7785302', 'tt5738152',
                  'tt1825978', 'tt1472211', 'tt4982356', 'tt6353886',
                  'tt1683003', 'tt3833746', 'tt2159988', 'tt7578246',
                  'tt6845582', 'tt6869948', 'tt2105043']

#I'll test to make sure dropping these films will leave me with the ones I want to keep.
to_drop = test[test['movie_id'].isin(imposter_films)].index
test.drop(to_drop, inplace=True)
test.sort_values('title')
```

Out[42]:

	title	studio	year	movie_id	runtime_minutes	genres
207	Abduction	LGF	2011	tt1600195	106.0	Action,Mystery,Thriller
613	Addicted	LGF	2014	tt2205401	106.0	Drama,Thriller
606	Big Eyes	Wein.	2014	tt1126590	106.0	Biography,Crime,Drama
53	Burlesque	SGem	2010	tt1126591	119.0	Drama,Music,Musical
962	Coco	BV	2017	tt2380307	105.0	Adventure,Animation,Comedy
112	Cyrus	FoxS	2010	tt1336617	91.0	Comedy,Drama,Romance
942	Denial	BST	2016	tt4645330	109.0	Biography,Drama
739	Legend	Uni.	2015	tt3569230	132.0	Biography,Crime,Drama
847	Lights Out	WB (NL)	2016	tt4786282	81.0	Drama,Horror,Mystery
686	Sisters	Uni.	2015	tt1850457	118.0	Comedy
696	Spotlight	ORF	2015	tt1895587	129.0	Crime,Drama
1055	Stronger	RAtt.	2017	tt3881784	119.0	Biography,Drama
186	The Artist	Wein.	2011	tt1655442	100.0	Comedy,Drama,Romance
41	The Bounty Hunter	Sony	2010	tt1038919	110.0	Action,Comedy,Romance
898	The Forest	Focus	2016	tt3387542	93.0	Horror,Mystery,Thriller
726	The Night Before	Sony	2015	tt3530002	101.0	Adventure,Comedy,Fantasy
137	The Tempest	Mira.	2010	tt1274300	110.0	Comedy,Drama,Fantasy
693	The Visit	Uni.	2015	tt3567288	94.0	Horror,Mystery,Thriller
721	The Walk	TriS	2015	tt3488710	123.0	Adventure,Biography,Drama
1063	The Wall	RAtt.	2017	tt4218696	88.0	Action,Drama,Thriller
1113	Truth or Dare	Uni.	2018	tt6772950	100.0	Horror,Thriller
510	Upside Down	MNE	2012	tt1374992	109.0	Drama,Fantasy,Romance

The remaining films match all of the databases' entries. Now I'll perform the same action on movies_analyzed.

In [43]:
```
movies_analyzed.drop(to_drop, inplace=True)
movies_analyzed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1008 entries, 0 to 1161
Data columns (total 31 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   level_0              1008 non-null   int64  
 1   index                1008 non-null   int64  
 2   title                1008 non-null   object 
 3   studio               1008 non-null   object 
 4   domestic_gross       1007 non-null   float64
 5   foreign_gross        911 non-null    object 
 6   year                 1008 non-null   int64  
 7   movie_id             1008 non-null   object 
 8   primary_title        1008 non-null   object 
 9   original_title       1008 non-null   object 
 10  start_year           1008 non-null   int64  
 11  runtime_minutes      1008 non-null   float64
 12  genres               1008 non-null   object 
 13  tmdb_index           1008 non-null   int64  
 14  genre_ids            1008 non-null   object 
 15  tmdb_id              1008 non-null   int64  
 16  original_language    1008 non-null   object 
 17  tmdb_original_title  1008 non-null   object 
 18  popularity           1008 non-null   float64
 19  tmdb_release_date    1008 non-null   object 
 20  tmdb_title           1008 non-null   object 
 21  vote_average         1008 non-null   float64
 22  vote_count           1008 non-null   int64  
 23  tmdb_year            1008 non-null   int32  
 24  tn_id                1008 non-null   int64  
 25  tn_release_date      1008 non-null   object 
 26  tn_title             1008 non-null   object 
 27  production_budget    1008 non-null   object 
 28  tn_domestic_gross    1008 non-null   object 
 29  tn_worldwide_gross   1008 non-null   object 
 30  tn_year              1008 non-null   int32  
dtypes: float64(4), int32(2), int64(8), object(17)
memory usage: 244.1+ KB
```

I see we are missing some critical data in both the domestic_gross and foreign_gross categories. We'll address these in a bit. (See cell 53)

After merging all of the data together, we still have 1008 films worth of data. This is a sufficent number of films to perform analysis on.

Now that I've merged all the data together and deleted duplicates and bad entries, I'll rename our database to mark where we are.

In [44]:
```
dataset = movies_analyzed.copy()
```

Now I'll convert the money in string format into integers.

In [45]:
```
dataset['tn_worldwide_gross'] = dataset['tn_worldwide_gross'].str.replace(',', '')
dataset['tn_worldwide_gross'] = dataset['tn_worldwide_gross'].str.replace('$', '').astype(
dataset['tn_domestic_gross'] = dataset['tn_domestic_gross'].str.replace(',', '')
dataset['tn_domestic_gross'] = dataset['tn_domestic_gross'].str.replace('$', '').astype(int
dataset['production_budget'] = dataset['production_budget'].str.replace(',', '')
dataset['production_budget'] = dataset['production_budget'].str.replace('$', '').astype(int
dataset[['tn_worldwide_gross', 'tn_domestic_gross', 'production_budget']]
```

Out[45]:

	tn_worldwide_gross	tn_domestic_gross	production_budget
0	1068879522	415004880	200000000
1	835524642	292576195	160000000
2	756244673	238736787	165000000
3	706102828	300531751	68000000
4	621156389	312433331	170000000
...	...	...	...
1157	7034615	2483472	20000000
1158	30963684	6115824	40000000
1159	3681096	1533324	9000000
1160	6089100	4286367	10000000
1161	1427656	1214525	6000000

1008 rows × 3 columns

Reformatting a few numbers in the foreign_gross column.

In [46]:
```
dataset['foreign_gross'] = dataset['foreign_gross'].str.replace(',', '')
```

I want to compare the numbers between the Box Office Mojo domestic_gross column with the box office totals from The Numbers dataset. However, the Box Office Mojo data is missing some values, so I'll fill those in with the data from The Numbers. Then, I'll see if anything is missing from the Box Office Mojo domestic_gross column.

In [47]:
```
dataset['domestic_gross'] = dataset['domestic_gross'].fillna(dataset['tn_domestic_gross'])
dataset['domestic_gross'] = dataset['domestic_gross'].round(decimals=0).astype(int)
dataset['foreign_gross'] = dataset['foreign_gross'].fillna(dataset['tn_worldwide_gross']).a
dataset['foreign_gross'] = dataset['foreign_gross'].round(decimals=0).astype(int)

dataset.isna().sum()
```

Out[47]:
```
level_0               0
index                 0
title                 0
studio                0
domestic_gross        0
foreign_gross         0
year                  0
movie_id              0
primary_title         0
original_title        0
start_year            0
runtime_minutes       0
genres                0
tmdb_index            0
genre_ids             0
tmdb_id               0
original_language     0
tmdb_original_title   0
popularity            0
tmdb_release_date     0
tmdb_title            0
vote_average          0
vote_count            0
tmdb_year             0
tn_id                 0
tn_release_date       0
tn_title              0
production_budget     0
tn_domestic_gross     0
tn_worldwide_gross    0
tn_year               0
dtype: int64
```

I need to convert runtime_minutes to an integer because every number in the column is a whole number.

In [48]:
```
dataset['runtime_minutes'] = dataset['runtime_minutes'].astype(int)
```

My three independent variables are genre, release month, and creative personnel. So far, the DataFrame I've created only include the first two. I need both the genre info and release months ready for analysis.

I looked for ways to decode the tmdb genre_id categories, but I could not find a simple way to do it. Since IMDB gives up to three genres per film, we'll just use those definitions for any genre comparisons. It's important to note that the genres listed for imdb are in alphabetical order, not order of significance. In light of this, I'll rely solely on imdb for genre information.

I need to split out the imdb genre list into separate columns.

In [49]:
```
dataset['imdb_genre_lst'] = dataset['genres'].str.split(',')
dataset['imdb_genre_1'] = dataset['imdb_genre_lst'].apply(lambda x : x[0])
dataset['imdb_genre_2'] = dataset['imdb_genre_lst'].apply(lambda x : x[1] if len(x)>=2 else
dataset['imdb_genre_3'] = dataset['imdb_genre_lst'].apply(lambda x : x[2] if len(x)==3 else
dataset[['imdb_genre_1','imdb_genre_2','imdb_genre_3','imdb_genre_lst']]
```

Out[49]:

	imdb_genre_1	imdb_genre_2	imdb_genre_3	imdb_genre_lst
0	Adventure	Animation	Comedy	[Adventure, Animation, Comedy]
1	Action	Adventure	Sci-Fi	[Action, Adventure, Sci-Fi]
2	Adventure	Animation	Comedy	[Adventure, Animation, Comedy]
3	Adventure	Drama	Fantasy	[Adventure, Drama, Fantasy]
4	Action	Adventure	Sci-Fi	[Action, Adventure, Sci-Fi]
...	...	...	...	...
1157	Fantasy	Horror	Mystery	[Fantasy, Horror, Mystery]
1158	Action	Adventure	Crime	[Action, Adventure, Crime]
1159	Action	Crime	Drama	[Action, Crime, Drama]
1160	Biography	Crime	Drama	[Biography, Crime, Drama]
1161	Action	Fantasy	Horror	[Action, Fantasy, Horror]

1008 rows × 4 columns

Those columns are now ready for analysis. Now for the release month columns.

I will analyze which movie release months yield the highest revenue, so I will isolate this info from the TN data. For organizational purposes, I'll also store this info as an integer from 1–12.

In [50]:
```
dataset['tn_release_month'] = dataset['tn_release_date'].str[0:3]
dataset[['tn_release_date','tn_release_month']]
```

Out[50]:

	tn_release_date	tn_release_month
0	Jun 18, 2010	Jun
1	Jul 16, 2010	Jul
2	May 21, 2010	May
3	Jun 30, 2010	Jun
4	May 7, 2010	May
...	...	...
1157	Oct 26, 2018	Oct
1158	Mar 9, 2018	Mar
1159	Dec 25, 2018	Dec
1160	Jun 15, 2018	Jun
1161	Sep 14, 2018	Sep

1008 rows × 2 columns

In [51]:
```
dataset['tn_num_month'] = dataset['tn_release_month'].map(lambda x :
                                                          1 if x == 'Jan' else
                                                          2 if x == 'Feb' else
                                                          3 if x == 'Mar' else
                                                          4 if x == 'Apr' else
                                                          5 if x == 'May' else
                                                          6 if x == 'Jun' else
                                                          7 if x == 'Jul' else
                                                          8 if x == 'Aug' else
                                                          9 if x == 'Sep' else
                                                          10 if x == 'Oct' else
                                                          11 if x == 'Nov' else
                                                          12 if x == 'Dec' else
                                                          'error')
```

I need to check to see if my conversions performed correctly. So, I'll compare the value counts for both columns.

In [52]:
```
print(dataset['tn_release_month'].value_counts())
print()
print(dataset['tn_num_month'].value_counts())
```

```
Nov    120
Oct    104
Dec    103
Sep     96
Jul     95
Jun     88
Aug     87
Mar     78
May     67
Feb     61
Apr     55
Jan     54
Name: tn_release_month, dtype: int64

11    120
10    104
12    103
9      96
7      95
6      88
8      87
3      78
5      67
2      61
4      55
1      54
Name: tn_num_month, dtype: int64
```

The numbers are consistent. Now, to get the dependent variable ready for analysis.

I need to compare the BOM box office totals to the TN totals. The BOM data separated domestic and foreign totals, while the TN data combined foreign and domestic as worldwide gross. So, I'll make a column of TN foreign gross, and I'll see how the totals of domestic and foreign compare. Once that's done, I'll print the top five rows of the data, and decide on how to proceed from there.

In [53]:
```
dataset['tn_foreign_gross'] = dataset['tn_worldwide_gross'] - dataset['tn_domestic_gross']

comparison = dataset.copy()
comparison['bom_minus_tn_foreign'] = comparison['foreign_gross'] - comparison['tn_foreign_g
comparison['bom_minus_tn_domestic'] = comparison['domestic_gross'] - comparison['tn_domest

comparison[['title','domestic_gross', 'foreign_gross','tn_domestic_gross','tn_worldwide_gr
```

Out[53]:

	title	domestic_gross	foreign_gross	tn_domestic_gross	tn_worldwide_gross	tn_foreign_gross	bom
1074	Avengers: Infinity War	678800000	1370	678815482	2048134200	1369318718	
953	The Fate of the Furious	226000000	1010	225764765	1234846267	1009081502	
636	Jurassic World	652300000	1019	652270625	1648854864	996584239	
45	Dear John	80000000	35000000	80014842	142033509	62018667	
293	Wreck-It Ralph	189400000	281800000	189412677	496511521	307098844	

I was planning on further analysis to see which database provided the best revenue information, but I think I found what I'm looking for in the table above. According to BOM, Avengers Infinity War, Jurassic World, and Fate of the Furious—some of the most profitable film franchises of all time—each made barely over $1000 at the foreign box office. That is not right, so we will rely on the TN database.

Also, there are a lot of extraneous columns in my dataset, so I'll select the columns that are relevant from this point forward and delete the rest. Then I'll preview the top 30 highest grossing films worldwide to see if I like the data left.

In [54]:
```
dataset = dataset[[
                    #bom
                    'title', 'studio','year',
                    #imdb
                    'movie_id', 'runtime_minutes','imdb_genre_1',
                    'imdb_genre_2', 'imdb_genre_3',
                    #tmdb
                    'tmdb_index', 'tmdb_id','original_language','popularity',
                    'tmdb_release_date','vote_average','vote_count',
                    #tn
                    'tn_release_month', 'tn_num_month', 'production_budget',
                    'tn_domestic_gross', 'tn_foreign_gross',
                    'tn_worldwide_gross']]

dataset.sort_values('tn_worldwide_gross', ascending = 0).head(30)
```

Out[54]:

	title	studio	year	movie_id	runtime_minutes	imdb_genre_1	imdb_genre_2	imdb_genre_3	tmd
1074	Avengers: Infinity War	BV	2018	tt4154756	149	Action	Adventure	Sci-Fi	
636	Jurassic World	Uni.	2015	tt0369610	124	Action	Adventure	Sci-Fi	
637	Avengers: Age of Ultron	BV	2015	tt2395427	141	Action	Adventure	Sci-Fi	
1075	Black Panther	BV	2018	tt1825683	134	Action	Adventure	Sci-Fi	
1076	Jurassic World: Fallen Kingdom	Uni.	2018	tt4881806	128	Action	Adventure	Sci-Fi	
401	Frozen	BV	2013	tt2294629	102	Adventure	Animation	Comedy	
1077	Incredibles 2	BV	2018	tt3606756	118	Action	Adventure	Animation	
953	The Fate of the Furious	Uni.	2017	tt4630562	136	Action	Crime	Thriller	
638	Minions	Uni.	2015	tt2293640	91	Adventure	Animation	Comedy	
1078	Aquaman	WB	2018	tt1477834	143	Action	Adventure	Fantasy	
798	Captain America: Civil War	BV	2016	tt3498820	147	Action	Adventure	Sci-Fi	
141	Transformers: Dark of the Moon	P/DW	2011	tt1399103	154	Action	Adventure	Sci-Fi	
283	Skyfall	Sony	2012	tt1074638	143	Action	Adventure	Thriller	
519	Transformers: Age of Extinction	Par.	2014	tt2109248	165	Action	Adventure	Sci-Fi	
284	The Dark Knight Rises	WB	2012	tt1345836	164	Action	Thriller	None	
0	Toy Story 3	BV	2010	tt0435761	103	Adventure	Animation	Comedy	
142	Pirates of the Caribbean: On Stranger Tides	BV	2011	tt1298650	136	Action	Adventure	Fantasy	

	title	studio	year	movie_id	runtime_minutes	imdb_genre_1	imdb_genre_2	imdb_genre_3	tmd
954	Despicable Me 3	Uni.	2017	tt3469046	89	Adventure	Animation	Comedy	
799	Finding Dory	BV	2016	tt2277860	97	Adventure	Animation	Comedy	
800	Zootopia	BV	2016	tt2948356	108	Adventure	Animation	Comedy	
285	The Hobbit: An Unexpected Journey	WB (NL)	2012	tt0903624	169	Adventure	Family	Fantasy	
402	Despicable Me 2	Uni.	2013	tt1690953	98	Adventure	Animation	Comedy	
955	Jumanji: Welcome to the Jungle	Sony	2017	tt2283362	119	Action	Adventure	Comedy	
403	The Hobbit: The Desolation of Smaug	WB (NL)	2013	tt1170358	161	Adventure	Fantasy	None	
520	The Hobbit: The Battle of the Five Armies	WB (NL)	2014	tt2310332	144	Adventure	Fantasy	None	
1079	Bohemian Rhapsody	Fox	2018	tt1727824	134	Biography	Drama	Music	
801	The Secret Life of Pets	Uni.	2016	tt2709768	87	Adventure	Animation	Comedy	
957	Spider-Man: Homecoming	Sony	2017	tt2250912	133	Action	Adventure	Sci-Fi	
286	Ice Age: Continental Drift	Fox	2012	tt1667889	88	Adventure	Animation	Comedy	
640	Spectre	Sony	2015	tt2379713	148	Action	Adventure	Thriller	

Now, since my dependent variable is worldwide box office revenue, I'll add a new column, revenue_rank, and sort the data.

In [55]:
```
dataset.sort_values('tn_worldwide_gross',ascending=False,inplace=True)
dataset.reset_index(inplace=True)
dataset = dataset.iloc[:,1:]
dataset['revenue_rank'] = dataset.index + 1
cols = ['revenue_rank'] + list(dataset.columns[0:-1])
dataset = dataset[cols]
dataset.head(30)
```

Out[55]:

	revenue_rank	title	studio	year	movie_id	runtime_minutes	imdb_genre_1	imdb_genre_2	imdb_
0	1	Avengers: Infinity War	BV	2018	tt4154756	149	Action	Adventure	
1	2	Jurassic World	Uni.	2015	tt0369610	124	Action	Adventure	
2	3	Avengers: Age of Ultron	BV	2015	tt2395427	141	Action	Adventure	
3	4	Black Panther	BV	2018	tt1825683	134	Action	Adventure	
4	5	Jurassic World: Fallen Kingdom	Uni.	2018	tt4881806	128	Action	Adventure	
5	6	Frozen	BV	2013	tt2294629	102	Adventure	Animation	
6	7	Incredibles 2	BV	2018	tt3606756	118	Action	Adventure	A
7	8	The Fate of the Furious	Uni.	2017	tt4630562	136	Action	Crime	
8	9	Minions	Uni.	2015	tt2293640	91	Adventure	Animation	
9	10	Aquaman	WB	2018	tt1477834	143	Action	Adventure	
10	11	Captain America: Civil War	BV	2016	tt3498820	147	Action	Adventure	
11	12	Transformers: Dark of the Moon	P/DW	2011	tt1399103	154	Action	Adventure	
12	13	Skyfall	Sony	2012	tt1074638	143	Action	Adventure	
13	14	Transformers: Age of Extinction	Par.	2014	tt2109248	165	Action	Adventure	
14	15	The Dark Knight Rises	WB	2012	tt1345836	164	Action	Thriller	
15	16	Toy Story 3	BV	2010	tt0435761	103	Adventure	Animation	
16	17	Pirates of the Caribbean: On Stranger Tides	BV	2011	tt1298650	136	Action	Adventure	
17	18	Despicable Me 3	Uni.	2017	tt3469046	89	Adventure	Animation	
18	19	Finding Dory	BV	2016	tt2277860	97	Adventure	Animation	
19	20	Zootopia	BV	2016	tt2948356	108	Adventure	Animation	
20	21	The Hobbit: An Unexpected Journey	WB (NL)	2012	tt0903624	169	Adventure	Family	

	revenue_rank	title	studio	year	movie_id	runtime_minutes	imdb_genre_1	imdb_genre_2	imdb_
21	22	Despicable Me 2	Uni.	2013	tt1690953	98	Adventure	Animation	
22	23	Jumanji: Welcome to the Jungle	Sony	2017	tt2283362	119	Action	Adventure	
23	24	The Hobbit: The Desolation of Smaug	WB (NL)	2013	tt1170358	161	Adventure	Fantasy	
24	25	The Hobbit: The Battle of the Five Armies	WB (NL)	2014	tt2310332	144	Adventure	Fantasy	
25	26	Bohemian Rhapsody	Fox	2018	tt1727824	134	Biography	Drama	
26	27	The Secret Life of Pets	Uni.	2016	tt2709768	87	Adventure	Animation	
27	28	Spider-Man: Homecoming	Sony	2017	tt2250912	133	Action	Adventure	
28	29	Ice Age: Continental Drift	Fox	2012	tt1667889	88	Adventure	Animation	
29	30	Spectre	Sony	2015	tt2379713	148	Action	Adventure	

There's still a long road ahead. In my dataset, I only have genre and release month information. We don't have anything about personnel. I want to analyze what effects five roles have on revenue production: writers, directors actors, actresses and producers.

There's a lot of personnel info in the IMDB data, but I'm going to stick to the personnel in the principals table. This has the big names in the five roles I will analyze. First, I have to make sense of the data in the principals table, so I'll preview the table.

In [56]:
```
imdb_principals
```

Out[56]:

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

1028186 rows × 6 columns

First, I'll select rows from the principals table for the films from my dataset. Then, I'll run the value_counts method on the results to make sure I have the same number of movies in my dataset, 1008.

In [57]:
```
imdb_principals_dataset = imdb_principals[imdb_principals['movie_id'].isin(dataset['movie_i
imdb_principals_dataset['movie_id'].value_counts()
```

Out[57]:
```
tt3691740    10
tt1972571    10
tt1403241    10
tt2184339    10
tt1365519    10
             ..
tt1470827     9
tt2268016     9
tt1508675     8
tt0873886     8
tt7535780     3
Name: movie_id, Length: 1008, dtype: int64
```

I now have the right number of films and the big names attached to them, but I don't want all the names. I just want the names of people from those five categories I'm analyzing (writer, director, actor, actress, producer). I'll see what values are available in the category column.

In [58]:
```
imdb_principals_dataset['category'].value_counts()
```

Out[58]:
```
actor                  2525
writer                 2027
producer               2005
actress                1484
director               1083
composer                472
cinematographer         277
editor                  130
production_designer      38
self                     15
archive_footage           1
archive_sound             1
Name: category, dtype: int64
```

I will select rows that are relevant to the categories I'm analyzing and discard the rest.

In [59]:
```
categories = ['writer', 'director', 'actor', 'actress', 'producer']

imdb_principals_dataset = imdb_principals_dataset[imdb_principals_dataset['category'].isin(
imdb_principals_dataset['category'].value_counts()
```

Out[59]:
```
actor       2525
writer      2027
producer    2005
actress     1484
director    1083
Name: category, dtype: int64
```

Now, I'm curious if each movie has at least one of each category. I want to keep actors and actresses separate for later analysis, but for this analysis, I'll combine the two categories into an actorsneutral column. I'll need to check this more than once without editing any parameters within, so I'll create a function without a parameter call.

In [60]:
```
def missing_categories():
    movies_that_include_writer_director_producer_actorsneutral = []

    categories_no_actors = ['writer', 'director', 'producer']

    for x in categories_no_actors:
        movies_that_include_writer_director_producer_actorsneutral.append(
            len(imdb_principals_dataset[imdb_principals_dataset['category']
            .isin([x])]['movie_id'].value_counts()))

    movies_that_include_writer_director_producer_actorsneutral.append(
        len(imdb_principals_dataset[imdb_principals_dataset['category']
        .isin(['actor','actress'])]['movie_id'].value_counts()))

    print('Movies with writers: ', movies_that_include_writer_director_producer_actorsneutr
    print('Movies with directors: ', movies_that_include_writer_director_producer_actorsneu
    print('Movies with producers: ', movies_that_include_writer_director_producer_actorsneu
    print('Movies with actors (neutral): ', movies_that_include_writer_director_producer_ac
missing_categories()
```

```
Movies with writers:  864
Movies with directors:  980
Movies with producers:  895
Movies with actors (neutral):  1005
```

There are movies in this database with no actors listed. Well, that makes it clear that there are some gaps in the IMDB principals table. I will attempt to fill them in with information from the other IMDB tables. First, I'll see which movies don't have writers.

In [61]:
```
writer = imdb_principals_dataset[imdb_principals_dataset['category'].isin(['writer'])]['mov
no_writer = imdb_principals_dataset[~imdb_principals_dataset['movie_id'].isin(writer).value
no_writer_list = no_writer['movie_id'].unique()
print('There are ' + str(len(no_writer_list)) + ' films missing a writer credit.')
print(no_writer_list)
```

```
There are 144 films missing a writer credit.
['tt0475290' 'tt1403241' 'tt1605783' 'tt1666186' 'tt2083355' 'tt1675192'
 'tt1527186' 'tt1602613' 'tt2215719' 'tt2235108' 'tt1336617' 'tt1650062'
 'tt2184339' 'tt1441326' 'tt0873886' 'tt1213663' 'tt1508675' 'tt1229340'
 'tt1243974' 'tt1549572' 'tt1645080' 'tt1719071' 'tt1764183' 'tt0466893'
 'tt1470827' 'tt1623288' 'tt1702443' 'tt1126591' 'tt1065073' 'tt1535108'
 'tt1535612' 'tt1540133' 'tt1617661' 'tt1020558' 'tt1092026' 'tt0878835'
 'tt1433822' 'tt1421051' 'tt1555064' 'tt2076220' 'tt1869716' 'tt1313092'
 'tt1220634' 'tt1602620' 'tt1615147' 'tt1931533' 'tt1235170' 'tt1182350'
 'tt2170593' 'tt1560747' 'tt1684628' 'tt1853728' 'tt1859650' 'tt1307068'
 'tt2194499' 'tt0872230' 'tt1570989' 'tt1763303' 'tt1316616' 'tt1800246'
 'tt2042568' 'tt2388637' 'tt2334873' 'tt1478964' 'tt1971352' 'tt2229499'
 'tt2401878' 'tt1637688' 'tt1855199' 'tt0938283' 'tt1920849' 'tt1840417'
 'tt1375666' 'tt1710396' 'tt1772288' 'tt1431181' 'tt1655442' 'tt1781827'
 'tt2309260' 'tt1195478' 'tt1333125' 'tt1171222' 'tt2103254' 'tt0478304'
 'tt1878870' 'tt1276104' 'tt1659337' 'tt1855325' 'tt2872718' 'tt3470600'
 'tt3312830' 'tt2334649' 'tt2387433' 'tt3099498' 'tt2690138' 'tt2937898'
 'tt2473794' 'tt3707106' 'tt3850214' 'tt3152624' 'tt2975578' 'tt3783958'
 'tt3606756' 'tt2361509' 'tt2994190' 'tt3760922' 'tt2390361' 'tt2609912'
 'tt2784678' 'tt2884206' 'tt2582802' 'tt2321549' 'tt3460252' 'tt3721936'
 'tt2649554' 'tt3567288' 'tt2872732' 'tt4034228' 'tt4094724' 'tt4925292'
 'tt6000478' 'tt3890160' 'tt5027774' 'tt5052448' 'tt4034354' 'tt5834262'
 'tt5013056' 'tt6265828' 'tt6288250' 'tt5726086' 'tt6791096' 'tt4651520'
 'tt4649416' 'tt5619332' 'tt5758778' 'tt5721088' 'tt4669986' 'tt4761916'
 'tt4624424' 'tt5719700' 'tt7784604' 'tt6499752' 'tt6359956' 'tt6266538']
```

Now I'll do the same for directors.

In [62]:
```
director = imdb_principals_dataset[imdb_principals_dataset['category'].isin(['director'])]
no_director = imdb_principals_dataset[~imdb_principals_dataset['movie_id'].isin(director).v
no_director_list = no_director['movie_id'].unique()
print('There are ' + str(len(no_director_list)) + ' films missing a director.')
print(no_director_list)
```

```
There are 28 films missing a director.
['tt1205537' 'tt1258972' 'tt1562568' 'tt1320253' 'tt1637725' 'tt1321860'
 'tt1235170' 'tt1583420' 'tt2177771' 'tt0840361' 'tt1024648' 'tt1859650'
 'tt1570989' 'tt0359950' 'tt2398231' 'tt2229499' 'tt1124035' 'tt1608290'
 'tt1630036' 'tt3707106' 'tt3521126' 'tt2784678' 'tt2637276' 'tt2671706'
 'tt2870708' 'tt6644200' 'tt5619332' 'tt7959026']
```

Now, I'll check if any of these films are listed in the IMDB writers and directors tables.

In [63]:
```
additional_writers = imdb_writers[imdb_writers['movie_id'].isin(no_writer_list)]
additional_writers['movie_id'].value_counts()
```

Out[63]:
```
tt1333125    20
tt1229340     2
tt2103254     2
tt1666186     2
tt2042568     2
             ..
tt1659337     1
tt2361509     1
tt2184339     1
tt1535612     1
tt1853728     1
Name: movie_id, Length: 142, dtype: int64
```

In [64]:
```
additional_directors = imdb_directors[imdb_directors['movie_id'].isin(no_director_list)]
additional_directors['movie_id'].value_counts()
```

Out[64]:
```
tt1235170    1
tt3521126    1
tt1630036    1
tt2784678    1
tt1562568    1
tt1637725    1
tt0840361    1
tt7959026    1
tt6644200    1
tt1608290    1
tt1859650    1
tt5619332    1
tt2177771    1
tt0359950    1
tt1205537    1
tt3707106    1
tt2637276    1
tt1124035    1
tt2671706    1
tt2870708    1
tt1258972    1
tt1321860    1
tt1320253    1
tt2398231    1
tt1583420    1
tt2229499    1
tt1024648    1
tt1570989    1
Name: movie_id, dtype: int64
```

I can now fill in missing data for 142 missing writers and 28 missing directors into our dataset.

To add this data, I first need to add the column "category" to both of these new dataframes, and use the column to assign the correct category label for both writer and director. Then, I'll just add these new rows to the imdb principles dataset. It's important to note that this new information is missing the data listed in the ordering, job and characters columns. This is not a problem since I'm not going to use these columns for my analysis.

In [65]:
```
additional_writers['category'] = 'writer'
additional_directors['category'] = 'director'

#Adding them to the dataset
imdb_principals_dataset = imdb_principals_dataset.append(additional_writers)
imdb_principals_dataset = imdb_principals_dataset.append(additional_directors)
imdb_principals_dataset
```

```
<ipython-input-65-e6c173939bf1>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  additional_writers['category'] = 'writer'
<ipython-input-65-e6c173939bf1>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  additional_directors['category'] = 'director'
```

Out[65]:

	movie_id	ordering	person_id	category	job	characters
37	tt0475290	1.0	nm0000982	actor	None	["Eddie Mannix"]
38	tt0475290	2.0	nm0000123	actor	None	["Baird Whitlock"]
39	tt0475290	3.0	nm2403277	actor	None	["Hobie Doyle"]
40	tt0475290	4.0	nm0000146	actor	None	["Laurence Laurentz"]
41	tt0475290	5.0	nm0001053	director	None	None
...	...	...	...	...	...	...
231826	tt1235170	NaN	nm0432380	director	NaN	NaN
252452	tt1320253	NaN	nm0000230	director	NaN	NaN
271231	tt7959026	NaN	nm0000142	director	NaN	NaN
281388	tt0359950	NaN	nm0001774	director	NaN	NaN
282970	tt2398231	NaN	nm0000169	director	NaN	NaN

9333 rows × 6 columns

Now I'll rerun the missing_categories function to see if the changes are implemented.

In [66]:
```
missing_categories()
```

```
Movies with writers:  1006
Movies with directors:  1008
Movies with producers:  895
Movies with actors (neutral):  1005
```

I still am missing two writers and a few actors, but I successfully filled in a lot of writers and directors. I could

complete both the writers and actors categories by using Google to look up the writers for the two movies missing that info, and do the same for actors/actresses for the three movies missing that info.

To start, I'll make a pair of dataframes that only include the imdb movie_id for the movies with missing writers and actors.

In [67]:
```
#A dataframe with one row per film to compare with dataframes that are missing data.
films_in_dataset = imdb_principals_dataset[~imdb_principals_dataset['movie_id'].duplicated(

#Now, I'll create a a pair of dataframes of movies with no writer or actorneutral.
films_with_writers = imdb_principals_dataset[imdb_principals_dataset['category'].isin(['wr
films_with_writers = films_with_writers[~films_with_writers.duplicated()]
no_writer = films_in_dataset[~films_in_dataset.isin(films_with_writers.tolist())]

films_with_actor = imdb_principals_dataset[imdb_principals_dataset['category'].isin(['acto
films_with_actor = films_with_actor[~films_with_actor.duplicated()]
no_actor = films_in_dataset[~films_in_dataset.isin(films_with_actor.tolist())]
```

Now I'll print two dataframes that give us the movie titles for the movie_ids missing writers and actors that we just extracted.

Movies with missing writers information

In [68]:
```
dataset[dataset['movie_id'].isin(no_writer.tolist())]
```

Out[68]:

	revenue_rank	title	studio	year	movie_id	runtime_minutes	imdb_genre_1	imdb_genre_2	imdb_genre
433	434	Justin Bieber: Never Say Never	Par.	2011	tt1702443	105	Documentary	Music	No
727	728	Katy Perry: Part of Me	Par.	2012	tt2215719	93	Documentary	Music	No

Movies missing actor information

In [69]:
```
dataset[dataset['movie_id'].isin(no_actor.tolist())]
```

Out[69]:

	revenue_rank	title	studio	year	movie_id	runtime_minutes	imdb_genre_1	imdb_genre_2	imdb_genre
433	434	Justin Bieber: Never Say Never	Par.	2011	tt1702443	105	Documentary	Music	No
727	728	Katy Perry: Part of Me	Par.	2012	tt2215719	93	Documentary	Music	No
895	896	Inside Job	SPC	2010	tt1645089	109	Crime	Documentary	No

Interesting. The two films without writers are concert films, so they certainly don't need writers. They do, however, have stars. So I'll add both Katy Perry and Justin Bieber as an actor/actress to those films.

Inside Job has an A-list Hollywood actor as a narrator, so I'll add him as an actor too.

In [70]:
```
movie_id_name_job = pd.DataFrame({'movie_id': ['tt1645089', 'tt1702443', 'tt2215719'],
                                  'person_id': None,
                                  'category': ['actor', 'actor', 'actress'],
                                  'name': ['Matt Damon', 'Justin Bieber', 'Katy Perry']})

movie_id_name_job['person_id'] = movie_id_name_job['name'].map(
    lambda x : imdb_persons[imdb_persons['primary_name'].str.contains(x)]['person_id'].iloc

imdb_principals_dataset = imdb_principals_dataset.append(movie_id_name_job.iloc[:,0:3])

print('Additional Actors')
print(movie_id_name_job)
print()

missing_categories()
```

```
Additional Actors
    movie_id  person_id category           name
0  tt1645089  nm0000354    actor     Matt Damon
1  tt1702443  nm3595501    actor  Justin Bieber
2  tt2215719  nm2953537  actress     Katy Perry

Movies with writers:  1006
Movies with directors:  1008
Movies with producers:  895
Movies with actors (neutral):  1008
```

I don't want to look up each film missing a producer, but I might be able to find some producer information in the dataset I have. There are some people listed as producers in the job category. It's possible someone is labeled as a producer in the job column and as something else in the category column.

First, I'll make a list of the films missing a producer credit.

In [71]:
```
producer = imdb_principals_dataset[imdb_principals_dataset['category'].isin(['producer'])]
no_producer = imdb_principals_dataset[~imdb_principals_dataset['movie_id'].isin(producer).
no_producer_list = no_producer['movie_id'].unique()
print('There are ' + str(len(no_producer_list)) + ' films missing a producer.')
print(no_producer_list)
```

```
There are 113 films missing a producer.
['tt1228705' 'tt2015381' 'tt2096673' 'tt1772341' 'tt1872181' 'tt0974015'
 'tt1961175' 'tt0892791' 'tt1402488' 'tt0948470' 'tt1075747' 'tt1999890'
 'tt0451279' 'tt1508675' 'tt2247476' 'tt1152822' 'tt1843866' 'tt2267968'
 'tt2279373' 'tt1302067' 'tt1985966' 'tt0881320' 'tt1365519' 'tt1487931'
 'tt2006295' 'tt0808510' 'tt0816711' 'tt1436562' 'tt1628841' 'tt1469304'
 'tt1572315' 'tt1860357' 'tt0478970' 'tt1319716' 'tt1790886' 'tt2243537'
 'tt1809398' 'tt2025690' 'tt2096672' 'tt1449283' 'tt1602620' 'tt1279935'
 'tt2294449' 'tt0800369' 'tt0892769' 'tt1196141' 'tt1198101' 'tt0963966'
 'tt1298650' 'tt1591479' 'tt1679335' 'tt0471042' 'tt0472181' 'tt2177771'
 'tt1587310' 'tt0787474' 'tt0864835' 'tt1397280' 'tt1911658' 'tt2176013'
 'tt1790809' 'tt1596346' 'tt2245084' 'tt2250912' 'tt1877832' 'tt1477834'
 'tt2017020' 'tt1621039' 'tt0837562' 'tt1277953' 'tt1711525' 'tt1981115'
 'tt2234155' 'tt2379713' 'tt1333125' 'tt2296777' 'tt0448694' 'tt1630036'
 'tt2283362' 'tt0790736' 'tt0848537' 'tt1979388' 'tt3501632' 'tt2828996'
 'tt2975590' 'tt3606752' 'tt3707106' 'tt2316204' 'tt3300542' 'tt2473510'
 'tt3385516' 'tt3832914' 'tt3498820' 'tt2910274' 'tt2948356' 'tt4154756'
 'tt2660888' 'tt3522806' 'tt3411444' 'tt3731562' 'tt2692250' 'tt2357291'
 'tt3521164' 'tt4849438' 'tt3416828' 'tt4667094' 'tt3922818' 'tt4981636'
 'tt4871980' 'tt5095030' 'tt6182908' 'tt6306064' 'tt7535780']
```

Now we need to look for producers in imdb_principals_dataset to see if there is anyone listed as a producer for the film in the job column but not in the category column.

In [72]:
```
additional_producers = imdb_principals[imdb_principals['movie_id'].isin(imdb_principals_da
producer_in_job = additional_producers[additional_producers['job'].isin(['producer'])]
print('There are', len(producer_in_job), 'people with a job listed as producer.')
print('Now to see what they are listed as in the category column')
producer_in_job['category'].value_counts()
```

```
There are 2002 people with a job listed as producer.
Now to see what they are listed as in the category column
```

Out[72]:
```
producer    2002
Name: category, dtype: int64
```

It is with a heavy heart that I conclude that there is no missing producer information in this dataset. I'll have to run my analysis on this data as it is.

Now I'll add the names of the people to the imdb principals database.

In [73]:
```
imdb_principals_dataset = pd.merge(imdb_principals_dataset,imdb_persons[['person_id','prima
imdb_principals_dataset
```

Out[73]:

	movie_id	ordering	person_id	category	job	characters	primary_name	death_year
0	tt0475290	1.0	nm0000982	actor	None	["Eddie Mannix"]	Josh Brolin	NaN
1	tt1075747	1.0	nm0000982	actor	None	["Jonah Hex"]	Josh Brolin	NaN
2	tt1182350	3.0	nm0000982	actor	None	["Roy"]	Josh Brolin	NaN
3	tt1403865	4.0	nm0000982	actor	None	["Tom Chaney"]	Josh Brolin	NaN
4	tt3397884	2.0	nm0000982	actor	None	["Matt Graver"]	Josh Brolin	NaN
...	...	...	...	...	...	...	...	...
9331	tt1333125	NaN	nm0765563	writer	NaN	NaN	Olle Sarri	NaN
9332	tt1333125	NaN	nm1856892	writer	NaN	NaN	Jacob Fleisher	NaN
9333	tt1333125	NaN	nm0698119	writer	NaN	NaN	Greg Pritikin	NaN
9334	tt1333125	NaN	nm2695453	writer	NaN	NaN	Steve Baker	NaN
9335	tt1702443	NaN	nm3595501	actor	NaN	NaN	Justin Bieber	NaN

9336 rows × 8 columns

When I analyze the different personnel in this dataset, I'll need a dataframe for each type of personnel analyzed. I'll create these now.

In [74]:
```
writers = imdb_principals_dataset[imdb_principals_dataset['category'].isin(['writer'])]
directors = imdb_principals_dataset[imdb_principals_dataset['category'].isin(['director'])]
producers = imdb_principals_dataset[imdb_principals_dataset['category'].isin(['producer'])]
actors = imdb_principals_dataset[imdb_principals_dataset['category'].isin(['actor'])]
actresses = imdb_principals_dataset[imdb_principals_dataset['category'].isin(['actress'])]
actorsneutral = imdb_principals_dataset[imdb_principals_dataset['category'].isin(['actor',

#Resetting the index and sorting them.
writers.sort_values('movie_id',inplace=True)
writers.reset_index(inplace=True)
directors.sort_values('movie_id',inplace=True)
directors.reset_index(inplace=True)
producers.sort_values('movie_id',inplace=True)
producers.reset_index(inplace=True)
actors.sort_values('movie_id',inplace=True)
actors.reset_index(inplace=True)
actresses.sort_values('movie_id',inplace=True)
actresses.reset_index(inplace=True)
actorsneutral.sort_values('movie_id',inplace=True)
actorsneutral.reset_index(inplace=True)
```

```
<ipython-input-74-6f20f1fc4bd5>:9: 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-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  writers.sort_values('movie_id',inplace=True)
<ipython-input-74-6f20f1fc4bd5>:11: 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-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  directors.sort_values('movie_id',inplace=True)
<ipython-input-74-6f20f1fc4bd5>:13: 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-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
```

Business Questions (Reprise)

Now, I can finally move on to the real work: Answering my 3 business questions:

1. What genres of film produce the highest box office revenue?
2. What is the best month to release a film in to generate the most revenue?
3. What writers, directors, producers, actors and actresses have the highest revenue earning potential?

Question 1

Question 1: What genres of film produce the highest box office revenue?

I'll start by generating a list of all the various genres available in the dataset. I should note that the genre columns 1, 2, and 3 are in alphabetical order per film. Each film can have one, two or three genres.

I will count the genres for more than one set of data, so I'll define a function that can do this with a dataframe parameter.

In [75]:
```
def genre_counts(df):

    #Counting the genres listed in each of the three columns
    genre_col1_count = df['imdb_genre_1'].value_counts()
    genre_col2_count = df['imdb_genre_2'].value_counts()
    genre_col3_count = df['imdb_genre_3'].value_counts()

    #Merging the three dataframes with the genre value counts
    temp = genre_col1_count.append(genre_col2_count.append(genre_col3_count))

    #This will be a dictionary that adds the genre counts together
    genre_count = {}

    for i in range(len(temp)):
        #The index of temp is the genre name
        key = temp.index[i]
        #The value of the single cell in the row is the count of films
        #in that genre in its respective column
        value = temp[i]
        #This tests to see if the genre (key)already has a value in the
        #dictionary. If it doesn't, it creates one.
        if key not in genre_count:
            genre_count[key] = value
        #If it does, then it adds the new value to the existing value.
        else:
            genre_count[key] = int(genre_count[key] + value)

    #Finally, this creates the dataframe to return.
    genre_count_df = pd.DataFrame.from_dict(genre_count,orient = 'index')
    genre_count_df.rename(columns = {0:'count'},inplace=True)
    genre_count_df.sort_values('count', ascending=False,inplace=True)
    return genre_count_df
dataset_genre_counts = genre_counts(dataset)
dataset_genre_counts
```

Out[75]:

	count
Drama	494
Comedy	378
Action	319
Adventure	271
Thriller	176
Crime	156
Romance	140
Biography	103
Horror	103
Sci-Fi	94
Fantasy	88
Animation	82
Mystery	80
Family	66

	count
Music	29
History	29
Sport	20
Documentary	7
War	7
Western	6
Musical	3

So Drama, Comedy, Action and Adventure are the four most numerous genres in the dataset.

There are many factors that make a successful movie. We all know that bad movies exist in all genres. There are superhero films that are among the best selling films ever made, like Avengers Endgame, and superhero films that are so poorly executed that the public is barely aware of them, like The Specials. Therefore, it would be foolish to take an average of all the films in each genre and declaring the highest average as the most successful genre. It would much more valuable to select a list of the best performing films in our dataset and analyzing what genres are represented in those films.

To do this, I'll set a revenue-earned threshold. If I set it to 490 million dollars or above, I'll get exactly 100 films, making percentages effortless to calculate. This number is so close to 500 million, I'll refer to these films as half-billion dollar films.

In [76]:
```
revenue_threshold = 490000000
cols = ['revenue_rank', 'title', 'studio', 'year', 'runtime_minutes',
        'imdb_genre_1', 'imdb_genre_2', 'imdb_genre_3', 'tn_worldwide_gross']
half_billion = dataset[dataset['tn_worldwide_gross'].map(lambda x :
                                    True if x>=revenue_threshold else False)]
half_billion[cols]
```

Out[76]:

	revenue_rank	title	studio	year	runtime_minutes	imdb_genre_1	imdb_genre_2	imdb_genre_3	tn_w
0	1	Avengers: Infinity War	BV	2018	149	Action	Adventure	Sci-Fi	
1	2	Jurassic World	Uni.	2015	124	Action	Adventure	Sci-Fi	
2	3	Avengers: Age of Ultron	BV	2015	141	Action	Adventure	Sci-Fi	
3	4	Black Panther	BV	2018	134	Action	Adventure	Sci-Fi	
4	5	Jurassic World: Fallen Kingdom	Uni.	2018	128	Action	Adventure	Sci-Fi	
...	...	...	...	...	...	...	...	...	
95	96	The Boss Baby	Fox	2017	97	Adventure	Animation	Comedy	
96	97	Dunkirk	WB	2017	106	Action	Drama	History	
97	98	Wreck-It Ralph	BV	2012	101	Adventure	Animation	Comedy	
98	99	How to Train Your Dragon	P/DW	2010	98	Action	Adventure	Animation	
99	100	Rio 2	Fox	2014	101	Adventure	Animation	Comedy	

100 rows × 9 columns

Of these 100 films, I'll see what genres are represented.

In [77]:
```
half_billion_counts = genre_counts(half_billion)
half_billion_counts = half_billion_counts.reset_index()
half_billion_counts
```

Out[77]:

	index	count
0	Adventure	86
1	Action	61
2	Comedy	34
3	Animation	33
4	Sci-Fi	27
5	Fantasy	17
6	Drama	10
7	Thriller	8
8	Family	5
9	Biography	3
10	Horror	3
11	Crime	3
12	Mystery	1
13	Romance	1
14	Music	1
15	History	1

From this, I can see that the most numerous genres are Adventure, Action, Comedy, Animation, and Sci-Fi.

It's important that I define what these genre definitions mean to IMDB, which is where I got the genre information. The data below is copied straight from IMDB's website. In order of popularity:

Adventure: Should contain numerous consecutive and inter-related scenes of characters participating in hazardous or exciting experiences for a specific goal. Often include searches or expeditions for lost continents and exotic locales, characters embarking in treasure hunt or heroic journeys, travels, and quests for the unknown. Not to be confused with Action, and should only sometimes be supplied with it. Subjective. Examples: The Goonies (1985) |The Lord of The Rings: The Fellowship of the Ring (2001) | Life of Pi (2012)

Action: Should contain numerous scenes where action is spectacular and usually destructive. Often includes non-stop motion, high energy physical stunts, chases, battles, and destructive crises (floods, explosions, natural disasters, fires, etc.) Note: if a movie contains just one action scene (even if prolonged, i.e. airplane-accident) it does not qualify. Subjective. Examples: Die Hard (1988) |The Avengers (2012) | Wonder Woman (2019)

Comedy: Virtually all scenes should contain characters participating in humorous or comedic experiences. The comedy can be exclusively for the viewer, at the expense of the characters in the title, or be shared with them. Please submit qualifying keywords to better describe the humor (i.e. spoof, parody, irony, slapstick, satire, black-comedy etc). If the title does not conform to the 'virtually all scenes' guideline then please do not add the comedy genre; instead, submit the same keyword variations described above to signify the comedic elements of the title. Subjective. Examples: Some Like it Hot (1959) |When Harry Met Sally... (1989) | Bridesmaids (2011)

Animation: Over 75% of the title's running time should have scenes that are wholly, or part-animated. Any form of animation is acceptable, e.g., hand-drawn, computer-generated, stop-motion, etc. Puppetry does not count as animation, unless a form of animation such as stop-motion is also applied. Incidental animated sequences should be indicated with the keywords part-animated or animated-sequence instead. Although the overwhelming majority of video games are a form of animation it's okay to forgo this genre when adding them as this is implied by the title type. Objective. Examples: Spirited Away (2001) |The Lion King (1994) | "The Simpsons" (1987)

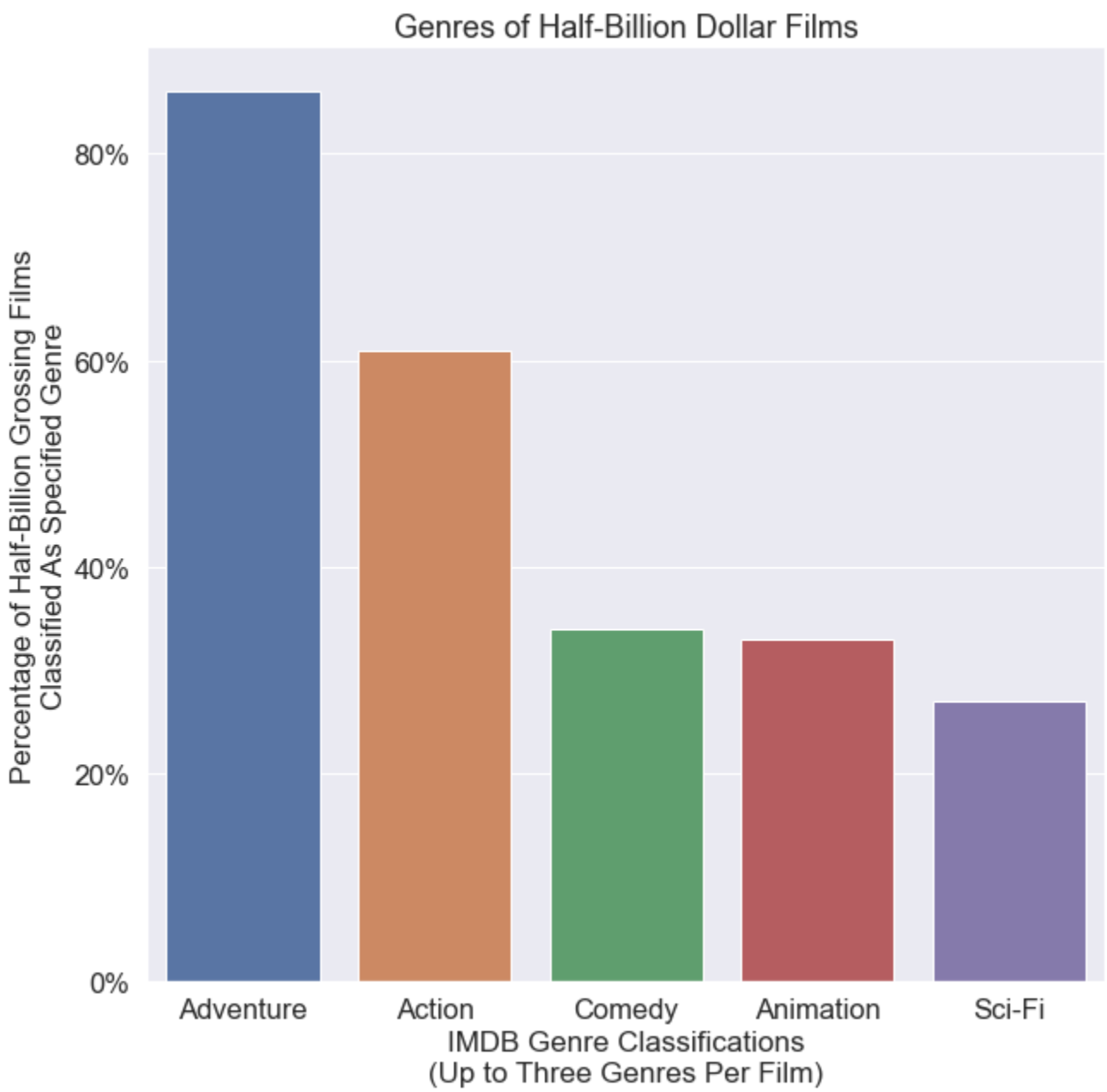
Sci-Fi: Numerous scenes, and/or the entire background for the setting of the narrative, should be based on speculative scientific discoveries or developments, environmental changes, space travel, or life on other planets. Subjective. Examples: Star Wars (1977) |The Matrix (1999) | Alien (1979)

Graphic 1

For this graphic I want to show what genres are the most numerous in the films of our dataset that reach half a billion dollars. I'll use a bar chart to do this.

In [78]:
```
half_billion_counts_percent = half_billion_counts.copy()
genres, percent = half_billion_counts_percent.columns

half_billion_counts_percent[percent] = half_billion_counts_percent[percent]/len(half_billi

plt.figure(figsize=(10, 10))
sns.set(font_scale = 1.5)

xlabel = 'IMDB Genre Classifications\n(Up to Three Genres Per Film)'
ylabel = 'Percentage of Half-Billion Grossing Films\nClassified As Specified Genre'
graphic_title = 'Genres of Half-Billion Dollar Films'

q1_ax = sns.barplot(y=percent,x=genres, data=half_billion_counts_percent.head())

#Formatting
q1_ax.set_xlabel(xlabel, fontsize = 17)
q1_ax.set_ylabel(ylabel, fontsize = 17)
q1_ax.set(title=graphic_title)
current_values = q1_ax.get_yticks()

q1_ax.set_yticklabels(['{:,.0%}'.format(x) for x in current_values]);
```

```
<ipython-input-78-39fa2209ab71>:21: UserWarning: FixedFormatter should only be used toget
her with FixedLocator
  q1_ax.set_yticklabels(['{:,.0%}'.format(x) for x in current_values]);
```



The results show that over half of the half-billion grossing films included the genres action and adventure.

Question 1 Answered?

Question 1: What genres of film produce the highest box office revenue?

My analysis has answered this question as well as it can be, however the question itself has flaws. By IMDB's own admission, most film genres are subjective. What one person finds funny, another might find disturbing. What one person finds adventurous, another finds timid. But if we accept the widely understood definitions of these classifications, then the analysis is sound. Action and adventure films do the best at the box office.

To parse genre considerations beyond generalities is an exercise in art criticism, not business analysis.

Question 2

Question 2: What is the best month to release a film in to generate the most revenue?

This question requires far less analysis. I'd like to look at it in two different ways. The first will be a histogram to see what months the half-billion grossing movies were released in. The second is a simple average of worldwide gross of all the films in the dataset binned by month.

I'll count the films by release month in two different dataframes, so first I'll define a function that performs this.

In [79]:
```
def release_month_count(df,div,sig):
    rt = df.groupby('tn_release_month')['tn_num_month','tn_domestic_gross',
                                        'tn_foreign_gross',
                                        'tn_worldwide_gross'].mean()

    #These two rows will be used in the next graphic
    #Count the total films in each month
    rt['count'] = df.groupby('tn_release_month')['title'].count()
    #Lets reduce the significant digits to make the graph more readable.
    #We'll go with millions of dollars.
    rt['count_percent'] = (rt['count']/len(df)).round(decimals=sig+2)


    cols = ['tn_domestic_gross','tn_foreign_gross','tn_worldwide_gross']

    rt[cols[0:3]] = ((rt[cols[0:3]]/div).round()).astype(int)
    rt.sort_values('tn_num_month',inplace=True)
    rt.reset_index(inplace=True)
    rt.drop(columns='tn_num_month',inplace=True)
    return rt

#The last two columns are not relevant to this graphic, so I'll skip them.
binned_by_month = release_month_count(dataset,1000000,1).iloc[:,0:4]
binned_by_month
```

```
<ipython-input-79-0f1e97393514>:2: FutureWarning: Indexing with multiple keys (implicitly
converted to a tuple of keys) will be deprecated, use a list instead.
  rt = df.groupby('tn_release_month')['tn_num_month','tn_domestic_gross',
```

Out[79]:

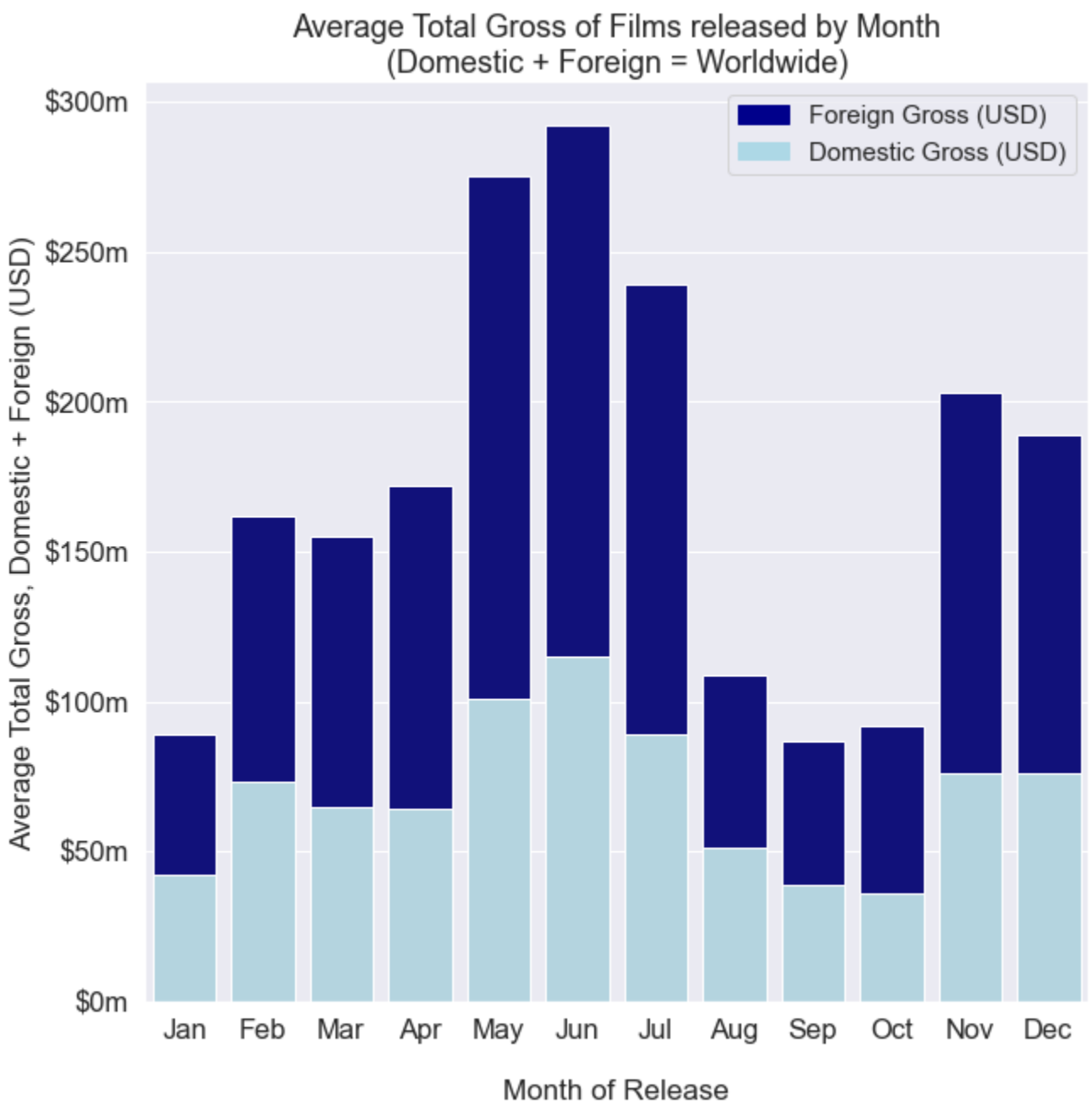
	tn_release_month	tn_domestic_gross	tn_foreign_gross	tn_worldwide_gross
0	Jan	42	47	89
1	Feb	73	89	162
2	Mar	65	89	155
3	Apr	64	108	172
4	May	101	174	275
5	Jun	115	177	292
6	Jul	89	150	239
7	Aug	51	58	109
8	Sep	39	48	87
9	Oct	36	57	92
10	Nov	76	127	203
11	Dec	76	113	189

This shows that the biggest money makers of the year are released in the summer months of May, June and July.

Graphic 2

In [80]:
```
#Shorthand for the column names
mon, dom, fgn, wor = binned_by_month.columns

#Labels
title = "Average Total Gross of Films released by Month\n(Domestic + Foreign = Worldwide)"
xlabel = '\nMonth of Release'
ylabel = 'Average Total Gross, Domestic + Foreign (USD)'

# set the figure size
plt.figure(figsize=(10, 10))

# top bar → take only worldwide values from the data
total = binned_by_month[[mon,wor]]

# bar chart 1 → top bars (group of worldwide total)
q2_ax1 = sns.barplot(x=mon,  y=wor, data=total, color='darkblue')

# bottom bar → take only domestic values from the data
domestic = binned_by_month[[mon,dom,wor]]

# bar chart 2 → bottom bars (group of domestic)
q2_ax2 = sns.barplot(x=mon, y=dom, data=domestic, estimator=sum, ci=None,  color='lightblue

# add legend
top_bar = mpatches.Patch(color='darkblue', label='Foreign Gross (USD)')
bottom_bar = mpatches.Patch(color='lightblue', label='Domestic Gross (USD)')
plt.legend(handles=[top_bar, bottom_bar], fontsize = 15)
q2_ax1.set_xlabel(xlabel, fontsize = 17)
q2_ax1.set_ylabel(ylabel, fontsize = 17)
q2_ax1.set(title=title)

#Setting up the Y-axis tick marks
current_values = q2_ax1.get_yticks()
current_values[:]=current_values[:]
q2_ax1.set_yticklabels(['${:.0f}m'.format(x) for x in current_values]);

# show the graph
plt.show()
```

```
<ipython-input-80-ae61bc86d138>:35: UserWarning: FixedFormatter should only be used toget
her with FixedLocator
  q2_ax1.set_yticklabels(['${:.0f}m'.format(x) for x in current_values]);
```



The graph shows foreign and domestic, which add together to get the worldwide gross.

There are two seasons a year that offer the greatest revenue potential for a movie release:

1. May–July
2. November–December.

Now I'm going to show how many of the half a billion films were released for each month of the year.

Even though we've used all bar charts so far, this plot should also be a bar chart because it fits the data.

In [81]:
```
binned_by_month_half_billion = release_month_count(half_billion,1000000,0)[['tn_release_mo
binned_by_month_half_billion
```

```
<ipython-input-79-0f1e97393514>:2: FutureWarning: Indexing with multiple keys (implicitly
converted to a tuple of keys) will be deprecated, use a list instead.
  rt = df.groupby('tn_release_month')['tn_num_month','tn_domestic_gross',
```

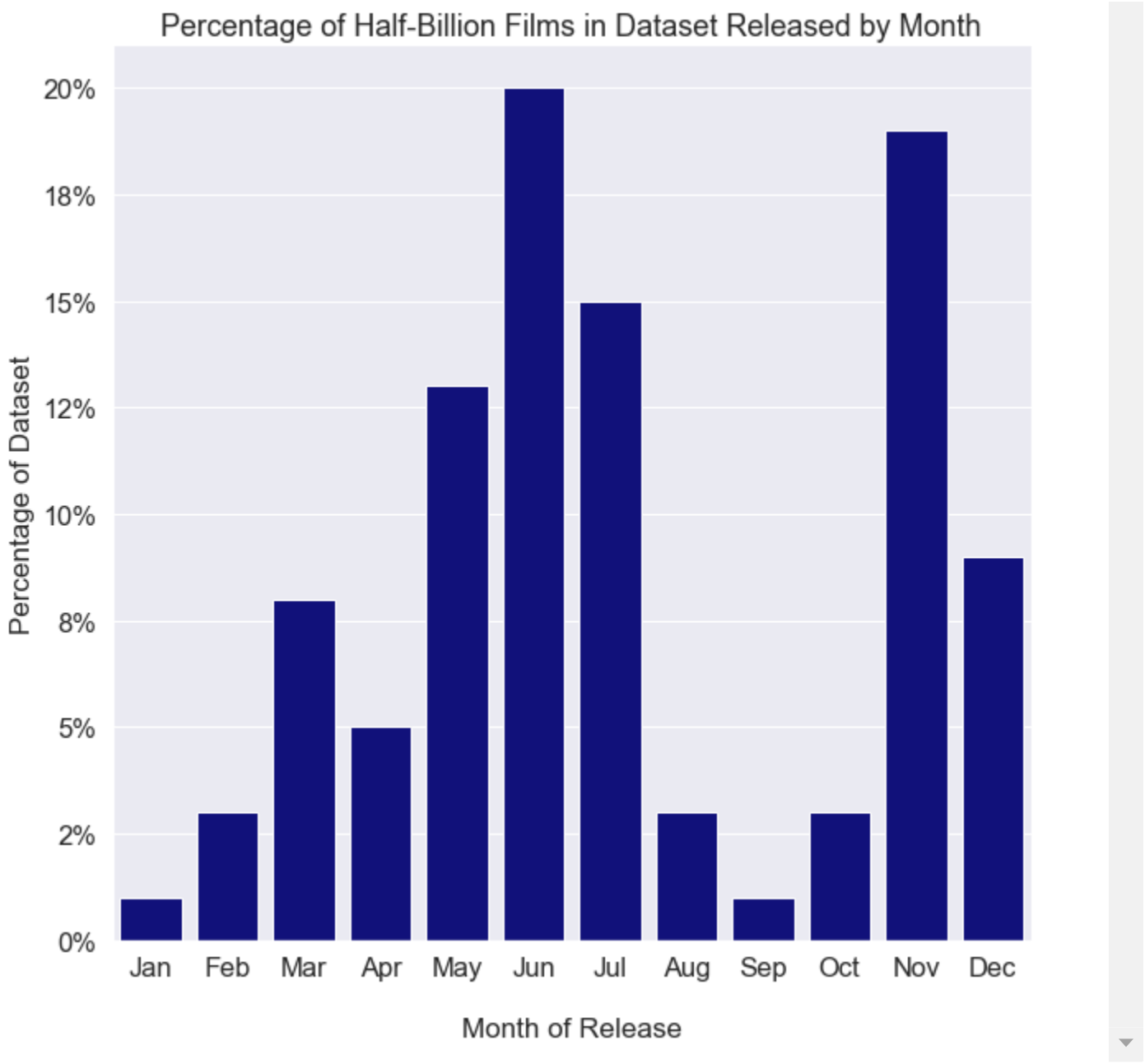
Out[81]:

	tn_release_month	count_percent
0	Jan	0.01
1	Feb	0.03
2	Mar	0.08
3	Apr	0.05
4	May	0.13
5	Jun	0.20
6	Jul	0.15
7	Aug	0.03
8	Sep	0.01
9	Oct	0.03
10	Nov	0.19
11	Dec	0.09

Graphic 3

In [82]:
```
#Shorthand for column names
mon, per = binned_by_month_half_billion.columns

#Labels
title = "Percentage of Half-Billion Films in Dataset Released by Month"
xlabel = '\nMonth of Release'
ylabel = 'Percentage of Dataset'

# set the figure size
plt.figure(figsize=(10, 10))

# bar chart 1 -> top bars (group of worldwide total)
q2_ax2 = sns.barplot(x=mon,  y=per, data=binned_by_month_half_billion, color='darkblue')

#Formatting
q2_ax2.set_xlabel(xlabel, fontsize = 17)
q2_ax2.set_ylabel(ylabel, fontsize = 17)
q2_ax2.set(title=title)
current_values = q2_ax2.get_yticks()
q2_ax2.set_yticklabels(['{:,.0%}'.format(x) for x in current_values]);

# show the graph
plt.show();
```

```
<ipython-input-82-0c81eb6accd6>:20: UserWarning: FixedFormatter should only be used toget
her with FixedLocator
  q2_ax2.set_yticklabels(['{:,.0%}'.format(x) for x in current_values]);
```



The half-billion dollar films follow the same release season trends.

However, it's worth noting that there are films that made half a billion dollars that were released during the slowest months of the year. So it is possible to make half a billion even in off season months.

Question 2 Answered?

Question 2: What is the best month to release a film to generate the most revenue?

The business question has been answered as well as it can be with the data given, however, more analysis needs to be done to choose a release date for a film.

This question ignores a lot of outside factors regarding release dates. A more thorough analysis would be to separate the films by week instead of month, but even that doesn't go far enough. There are so many events that occur outside of a regular calendar year. It could be that a popular video game was released one week, so

people stayed home rather than pay for a film. Also, choosing a release date needs to consider what the competition is doing. We don’t want to release a movie that targets the same audience as a Marvel film the same week a Marvel film debuts.

Question 3

Question 3: What writers, directors, producers, actors and actresses have the highest revenue earning potential?

To answer this question, once again, I need to answer it two ways. The first is which category is most consistent, writer, director, producer or actor. Then, I can list the top ten talent with the highest revenue yield in each category.

The writers job has a lot of bad information in it. It contains people who get credit for creating characters that are featured in the mega popular superhero films, but the character creators didn't write the movie at all. Unfortunately, there's no way to isolate the screenwriters, so instead I'll just take people who have at least three films on their resume. I'll do this for all creatives.

So, I'll add a column that I'll use to count the films a person has worked on, and then use that to filter the data.

In [83]:
```
def counting_films(df):
    new = df.groupby('person_id')['movie_id','primary_name'].count()
    new.rename(columns = {'movie_id' :'film_count'}, inplace=True)
    df = pd.merge(df,new['film_count'], how='inner',left_on='person_id', right_on='person_
    return df

writers = counting_films(writers)
directors = counting_films(directors)
producers = counting_films(producers)
actors = counting_films(actors)
actresses = counting_films(actresses)
actorsneutral = counting_films(actorsneutral)
writers
```

```
<ipython-input-83-bfdda0a7229d>:2: FutureWarning: Indexing with multiple keys (implicitly
converted to a tuple of keys) will be deprecated, use a list instead.
  new = df.groupby('person_id')['movie_id','primary_name'].count()
```

Out[83]:

	index	movie_id	ordering	person_id	category	job	characters	primary_name	death_year	film
0	6787	tt0359950	6.0	nm0862122	writer	based on the short story by	None	James Thurber	1961.0	
1	6784	tt0359950	5.0	nm0175726	writer	screenplay by	None	Steve Conrad	NaN	
2	6785	tt2358925	6.0	nm0175726	writer	written by	None	Steve Conrad	NaN	
3	6786	tt2543472	6.0	nm0175726	writer	screenplay by	None	Steve Conrad	NaN	
4	1898	tt0365907	6.0	nm0088747	writer	based on the novel by	None	Lawrence Block	NaN	
...	...	...	...	...	...	...	...	...	...	
2203	9258	tt7349662	9.0	nm9259302	writer	based on the book by	None	Ron Stallworth	NaN	
2204	9310	tt7388562	6.0	nm3885256	writer	None	None	Terence Berden	NaN	
2205	9293	tt7535780	3.0	nm0434525	writer	short story	None	Franz Kafka	1924.0	
2206	9240	tt7784604	NaN	nm4170048	writer	NaN	NaN	Ari Aster	NaN	
2207	9274	tt7959026	5.0	nm10095627	writer	inspired by the New York Times Magazine Articl...	None	Sam Dolnick	NaN	

2208 rows × 10 columns

Now I need to add the film grossing information to that personnel data.

In [84]:
```
def add_movie_stats(org,cls,div):
    new = pd.merge(org,dataset[cls+['movie_id']], how='inner',left_on='movie_id', right_on=
    new[cls] = (new[cls]/div).round(0)
    return new

cols = ['tn_worldwide_gross']
divisor = 1000000
writers = add_movie_stats(writers,cols,divisor)
directors = add_movie_stats(directors,cols,divisor)
producers = add_movie_stats(producers,cols,divisor)
actors  = add_movie_stats(actors,cols,divisor)
actresses = add_movie_stats(actresses,cols,divisor)
actorsneutral = add_movie_stats(actorsneutral,cols,divisor)
```

We can't hire dead people, so let's get them out of the dataset.

In [85]:
```
writers_alive = writers[writers['death_year'].isna()]
directors_alive = directors[directors['death_year'].isna()]
producers_alive = producers[producers['death_year'].isna()]
actors_alive = actors[actors['death_year'].isna()]
actresses_alive = actresses[actresses['death_year'].isna()]
actorsneutral_alive = actorsneutral[actorsneutral['death_year'].isna()]
```

Now I need to see how the top creatives' averages differ.

In [86]:
```
def top(df,floor,num,label):
    new = df[df['film_count'].map(lambda x : True if x>=floor else False)]
    tops = new.groupby('primary_name')[['tn_worldwide_gross']].mean().sort_values('tn_worl
    tops['tn_worldwide_gross'] = tops['tn_worldwide_gross'].round(0).astype(int)
    last = pd.merge(tops,df[['primary_name','film_count']], how='left',left_on='primary_nar
    last.rename(columns={'primary_name':label+"'s name",'tn_worldwide_gross' : label+"'s Gr
    last = last[~last.duplicated()].reset_index(drop=True)
    return last.reset_index()

#Here I define the minimum number of films a creative must have credited to
#them in order to be included in the dataset.
the_floor = 3
#This variable decides the number of creatives analyzed for each category.
the_top = 50

writers_top = top(writers_alive,the_floor,the_top,'Writer')
directors_top = top(directors_alive,the_floor,the_top,'Director')
producers_top = top(producers_alive,the_floor,the_top,'Producer')
actors_top = top(actors_alive,the_floor,the_top,'Actor')
actresses_top = top(actresses_alive,the_floor,the_top,'Actress')
actorsneutral_top = top(actorsneutral_alive,the_floor,the_top,'(Gender Inclusive) Actor')
```

Now I'll put the various averages into a single table.

In [87]:
```
#Since each dataframe I'm merging has exactly 50 rows, we can merge by index.
def merging(df1,df2):
    tem = pd.merge(df1,df2,how='inner',left_on='index',right_on='index')
    return tem

#Now I'll merge all six dataframes together so I can perform calculations
#on all three at once.
temp1 = merging(writers_top,directors_top)
temp2 = merging(producers_top,actors_top)
temp3 = merging(actresses_top,actorsneutral_top)
temp4 = merging(temp1,temp2)
all_creatives = merging(temp4,temp3)

#This defines which rows I need for the graphic
des_rows = ["Writer's Gross","Director's Gross","Producer's Gross",
            "Actor's Gross","Actress's Gross",
            "(Gender Inclusive) Actor's Gross"]

#Now I'll average all the totals together
all_creatives_average = pd.DataFrame(all_creatives[des_rows].mean()
                                     .round(0).astype(int))
all_creatives_average.reset_index(inplace=True)
all_creatives_average.rename(columns={'index':'Creative Jobs',
                                      0:"Top "+str(the_top)+
                                      " Average gross ($MM)"},inplace=True)
all_creatives_average
```

Out[87]:

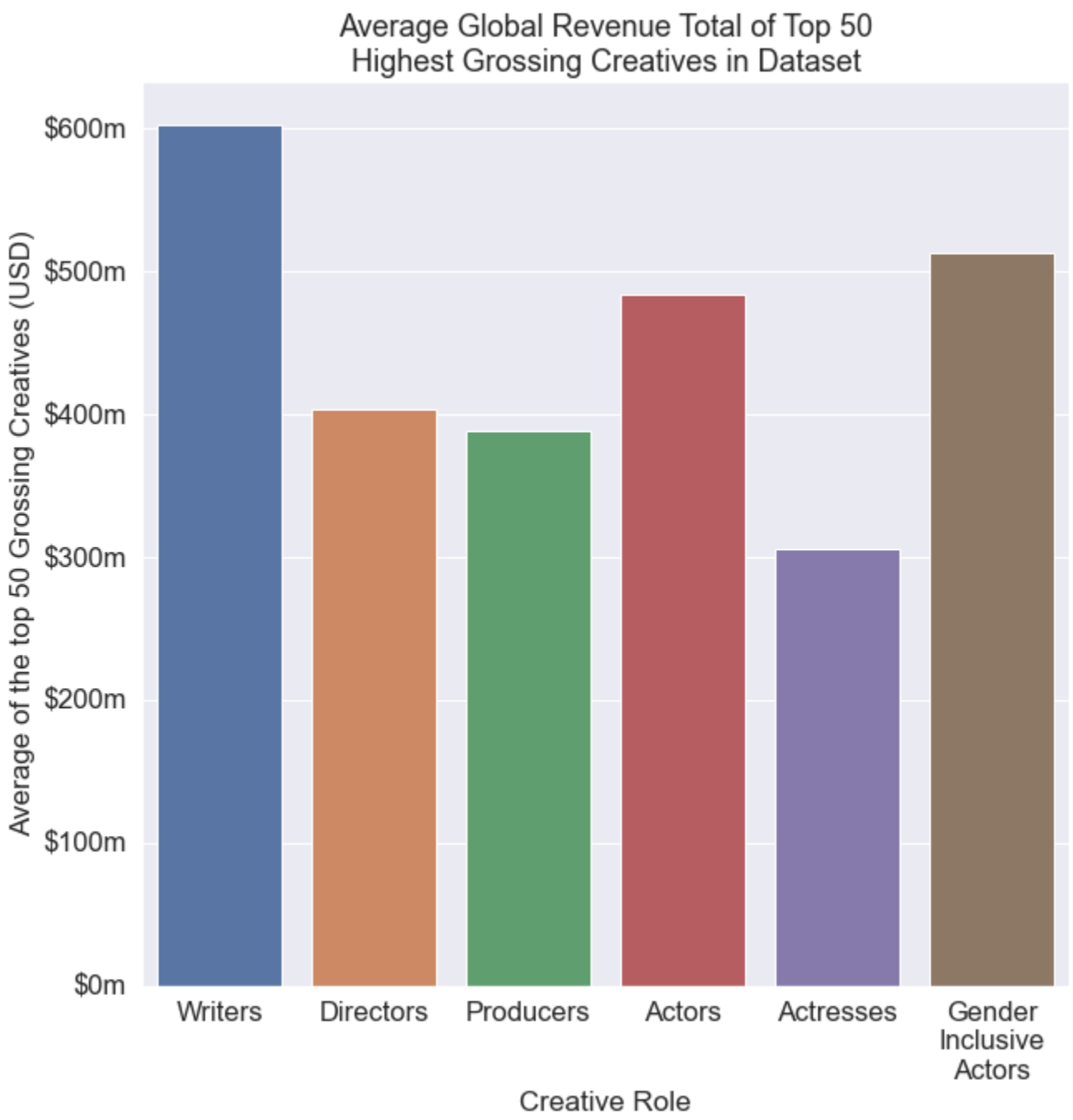
	Creative Jobs	Top 50 Average gross ($MM)
0	Writer's Gross	603
1	Director's Gross	403
2	Producer's Gross	388
3	Actor's Gross	484
4	Actress's Gross	306
5	(Gender Inclusive) Actor's Gross	513

Graphic 4

Now for the graphic

In [88]:
```
#Shorthand column labels
job, grs = all_creatives_average.columns

#Copy so I can change the dataframe just for this graphic
creatives_gross_graphic = all_creatives_average.copy()

#Labels (abb is abbreviated)
abb_rows = ['Writers','Directors','Producers','Actors',
            'Actresses','Gender\nInclusive\nActors']
#This for loop and corresponding dataframe command labels the graphic
#as specified above
row_change = {}
for i in range(len(des_rows)):
    row_change[des_rows[i]]=abb_rows[i]
creatives_gross_graphic['Creative Jobs'] = creatives_gross_graphic['Creative Jobs'].map(la

#More labeling
xlabel = "Creative Role"
ylabel = "Average of the top " +str(the_top)+" Grossing Creatives (USD)"
title = "Average Global Revenue Total of Top " +str(the_top)+"\nHighest Grossing Creatives

#Setting the plot size
plt.figure(figsize=(10, 10))
sns.set(font_scale = 1.5)

#Creating the plot
q3_ax = sns.barplot(y=grs,x=job, data=creatives_gross_graphic)

#Formatting
q3_ax.set_xlabel(xlabel, fontsize = 17)
q3_ax.set_ylabel(ylabel, fontsize = 17)
q3_ax.set(title=title)

#Setting up the Y-axis tick marks
current_values = q3_ax.get_yticks()
current_values[:]=current_values[:]
q3_ax.set_yticklabels(['${:.0f}m'.format(x) for x in current_values]);
```

```
<ipython-input-88-558eaa8ba329>:37: UserWarning: FixedFormatter should only be used toget
her with FixedLocator
  q3_ax.set_yticklabels(['${:.0f}m'.format(x) for x in current_values]);
```



According to the data, writers and actors have the largest impact on a film's total gross, so I'd recommend hiring some big names in those two categories. Despite the data, I highly reccomend pay equality between actors and actresses. We cannot make popular movies without actresses. We should pay them well. Also, if it gets out that your film studio does not have pay equality between men and women, the public will not be happy.

Anyway, with this in mind, I'll list the top ten highest grossing creatives in the categories of writers, actors and actresses.

Graphic 5

In [89]:
```
all_creatives.rename(columns = {'index':'Rank'},inplace=True)
all_creatives['Rank'] = all_creatives['Rank'] + 1
top_ten = all_creatives[["Rank","Writer's name","Actor's name","Actress's name"]].head(10)
top_ten
```

Out[89]:

	Rank	Writer's name	Actor's name	Actress's name
0	1	Guillermo del Toro	Richard Armitage	Sandra Bullock
1	2	Christopher Markus	Ian McKellen	Bryce Dallas Howard
2	3	Stephen McFeely	Robert Downey Jr.	Evangeline Lilly
3	4	Derek Connolly	Chris Evans	Scarlett Johansson
4	5	David S. Goyer	Chris Pratt	Eloise Mumford
5	6	Chris McKenna	Chris Hemsworth	Anne Hathaway
6	7	Erik Sommers	Benjamin Bratt	Holly Hunter
7	8	Philippa Boyens	Andy Serkis	Judi Dench
8	9	Fran Walsh	Mark Ruffalo	Angelina Jolie
9	10	Suzanne Collins	Martin Freeman	Emily Mortimer

These are the highest earning writers, actors and actresses in our dataset.

It's worth noting that each film project has different factors for choosing the right creatives. You cannot just pick a few of the names at the top and expect a smash success.

Question 3 Answered?

Question 3: What writers, directors, producers, actors and actresses have the highest revenue earning potential?

Similar to month, there are many other factors to consider when hiring writers and actors. For starters, is someone who pays to go see Hobbit 2: Desolation of Smaug really going because Evangeline Lilly is in the film, or are they going to see a fantasy adventure film set in the same world as the Lord of the Rings films they saw when they were eleven? Considering how much pull a writer or actor has on an audience has its natural limits.

That being said, we do know which creatives had starring roles in the highest grossing films, but we don’t know how prominent a role the actors played in the film, or how the marketing pushed each film.

We do know which creatives had starring roles in the highest grossing films, but we don’t know how prominent a role the actors played in the film, or how the marketing advertised each film. Was the film billed as the latest thriller from Steven Spielberg, the latest Star Wars Movie, or Angelina Jolie’s role of her career?

We know which people were attached to the highest-grossing films ever released, but we don’t know how much draw their names had to the audience.

Conclusion

The analysis I've done leads to these three conclusions:

1. The public likes action and adventure films.
2. The best release windows are May–July and November–December.
3. Writers and actors are the best creative personnel to increase a films revenue potential.

This analysis is limited by time and budget. There are many other factors to consider when trying to launch a film business. Some areas to consider looking into before producing a film are:

1. How much do existing franchises influence the public's film choices?
2. If existing film franchises have a lot of draw, does the public have the appetite for more after they've already spent a lot of time and money on the existing franchises?
3. Should Microsoft find an existing property and option it for film or start new?
4. Is streaming a better option for building the brand and generating revenue?
5. Should Microsoft purchase one of the big studios instead of entering the competition?

I think more time spent in answering the questions above would go a long way in helping Microsoft be successful with their film aspirations.