Microsoft Media: An Analysis on the Top Successful Movies

When thinking about the data, I wonder what makes a movie successful and how could someone view success by? Different people may have a different answer to this question. To me, I may view a movie success by highest ratings or which movie made the most money or maybe even which movie had the highest return on their investment and production budget. My goal of this project is to filter my data by these success metrics and then answer questions that could help a film company start their business to enter the competitive field strong. Some questions I hope to answer are:

- Is there a similarity in movies between what made the most money and what had the highest return?
- Where were movies filmed? What are the most popular locations for movies to be filmed? Where were the movies with the highest gross filmed? Highest return?
- What genres were the most profitable?

Going forward I will have these questions in mind while I go through the Data Science process of loading my data, cleaning it, performing calculations, and creating my visualizations and final analysis.

First Step: Importing the csv data and previewing the dataframes

In [1]:
```
import requests
import pandas as pd
import numpy as np

budg_df= pd.read_csv(r'C:\Users\melfr\Documents\Flatiron\p1\labs\dsc-phase-1-proj
rate_df= pd.read_csv(r'C:\Users\melfr\Documents\Flatiron\p1\labs\dsc-phase-1-proj
titleaka_df= pd.read_csv(r'C:\Users\melfr\Documents\Flatiron\p1\labs\dsc-phase-1-
titleinfo_df= pd.read_csv(r'C:\Users\melfr\Documents\Flatiron\p1\labs\dsc-phase-1
```

In [2]:
```
budg_df.head()
```

Out[2]:

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

In [3]:
```
rate_df.head()
```

Out[3]:

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

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

Out[4]:

	title_id	ordering	title	region	language	types	attributes	is_original_title
0	tt0369610	10	Джурасик свят	BG	bg	NaN	NaN	0.0
1	tt0369610	11	Jurashikku warudo	JP	NaN	imdbDisplay	NaN	0.0
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	NaN	imdbDisplay	NaN	0.0
3	tt0369610	13	O Mundo dos Dinossauros	BR	NaN	NaN	short title	0.0
4	tt0369610	14	Jurassic World	FR	NaN	imdbDisplay	NaN	0.0

In [6]:
```
titleinfo_df.head()
```

Out[6]:

	tconst	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

Prepping columns to merge by adjusting the column names

In [2]:
```
rate_df= rate_df.rename(columns= {'tconst': 'title_id'})
rate_df.head()
```

Out[2]:

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

In [3]:
```
titleinfo_df= titleinfo_df.rename(columns= {'tconst': 'title_id'})
titleinfo_df
```

Out[3]:

	title_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	NaN
146143	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	NaN	Documentary

146144 rows × 6 columns

Its merging time! I will be merging three dataframes based on the column 'title_id'. These dataframes are titleaka_df, rate_df, and titleinfo_df.

In [4]:
```
merged_df= pd.merge(titleaka_df, rate_df,  on='title_id')
merged_df.head()
```

Out[4]:

	title_id	ordering	title	region	language	types	attributes	is_original_title	avera
0	tt0369610	10	Джурасик свят	BG	bg	NaN	NaN	0.0	
1	tt0369610	11	Jurashikku warudo	JP	NaN	imdbDisplay	NaN	0.0	
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	NaN	imdbDisplay	NaN	0.0	
3	tt0369610	13	O Mundo dos Dinossauros	BR	NaN	NaN	short title	0.0	
4	tt0369610	14	Jurassic World	FR	NaN	imdbDisplay	NaN	0.0	

In [5]:
```
merged_df= pd.merge(merged_df, titleinfo_df,  on='title_id')
merged_df.head()
```

Out[5]:

	title_id	ordering	title	region	language	types	attributes	is_original_title	avera
0	tt0369610	10	Джурасик свят	BG	bg	NaN	NaN	0.0	
1	tt0369610	11	Jurashikku warudo	JP	NaN	imdbDisplay	NaN	0.0	
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	NaN	imdbDisplay	NaN	0.0	
3	tt0369610	13	O Mundo dos Dinossauros	BR	NaN	NaN	short title	0.0	
4	tt0369610	14	Jurassic World	FR	NaN	imdbDisplay	NaN	0.0	

Now that the first 3 dataframes are merged its time to add the last one. The budg_df has the movie column named as 'movie' but the new merged_df has the column as 'primary_title'. I will be renaming budg_df to match and merge on that column.

In [11]:
```
budg_df.head()
```

Out[11]:

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

In [6]:
```
budg_df= budg_df.rename(columns= {'movie': 'primary_title'})
budg_df.head()
```

Out[6]:

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

In [7]:
```
merged_df= pd.merge(merged_df, budg_df,  on='primary_title')
merged_df.head()
```

Out[7]:

	title_id	ordering	title	region	language	types	attributes	is_original_title	avera
0	tt0369610	10	Джурасик свят	BG	bg	NaN	NaN	0.0	
1	tt0369610	11	Jurashikku warudo	JP	NaN	imdbDisplay	NaN	0.0	
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	NaN	imdbDisplay	NaN	0.0	
3	tt0369610	13	O Mundo dos Dinossauros	BR	NaN	NaN	short title	0.0	
4	tt0369610	14	Jurassic World	FR	NaN	imdbDisplay	NaN	0.0	

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45174 entries, 0 to 45173
Data columns (total 20 columns):
title_id             45174 non-null object
ordering             45174 non-null int64
title                45174 non-null object
region               42546 non-null object
language             6938 non-null object
types                32158 non-null object
attributes           2202 non-null object
is_original_title    45174 non-null float64
averagerating        45174 non-null float64
numvotes             45174 non-null int64
primary_title        45174 non-null object
original_title       45174 non-null object
start_year           45174 non-null int64
runtime_minutes      45019 non-null float64
genres               45155 non-null object
id                   45174 non-null int64
release_date         45174 non-null object
production_budget    45174 non-null object
domestic_gross       45174 non-null object
worldwide_gross      45174 non-null object
dtypes: float64(3), int64(4), object(13)
memory usage: 7.2+ MB
```

Doing some inspection on the averagerating column because I will want to do some analysis based on the values here. I will also be filtering some of the values if they do not meet a certain threshhold.

In [15]:
```
merged_df['averagerating'].dtype
```

Out[15]:
```
dtype('float64')
```

In [16]:
```
merged_df['averagerating'].isnull().sum()
```

Out[16]:
```
0
```

In [8]:
```
fltr_ratingdf= merged_df[merged_df['averagerating'] > 6.5]
```

We are assuming that an initial launch that will only be for the United States so we are only including films that were released in the US. Films that were not released here will be excluded from our dataframe.

In [9]:
```
fltr_ratingdf= fltr_ratingdf[fltr_ratingdf['region'] == 'US']
```

Now time to get rid of columns that are not important to the analysis that I plan to do!

In [10]:
```
fltr_ratingdf.drop('attributes', axis=1, inplace=True)
fltr_ratingdf
```

Out[10]:

	title_id	ordering	title	region	language	types	is_original_title	averagera
12	tt0369610	21	Jurassic World 3D	US	NaN	NaN	0.0	
20	tt0369610	29	Jurassic World	US	NaN	NaN	0.0	
21	tt0369610	2	Ebb Tide	US	NaN	NaN	0.0	
28	tt0369610	36	Jurassic Park IV	US	NaN	working	0.0	
37	tt0369610	44	Jurassic Park 4	US	NaN	NaN	0.0	
...	...	...	...	...	...	...	...	
45085	tt5649108	2	Thoroughbreds	US	NaN	NaN	0.0	
45099	tt6139732	13	Aladdin	US	NaN	imdbDisplay	0.0	
45100	tt6139732	13	Aladdin	US	NaN	imdbDisplay	0.0	
45133	tt6139732	29	Aladdin	US	en	NaN	0.0	
45134	tt6139732	29	Aladdin	US	en	NaN	0.0	

1213 rows × 19 columns

In [11]:
```
fltr_ratingdf.drop('types', axis=1, inplace=True)
fltr_ratingdf
```

Out[11]:

	title_id	ordering	title	region	language	is_original_title	averagerating	numvot
12	tt0369610	21	Jurassic World 3D	US	NaN	0.0	7.0	5393
20	tt0369610	29	Jurassic World	US	NaN	0.0	7.0	5393
21	tt0369610	2	Ebb Tide	US	NaN	0.0	7.0	5393
28	tt0369610	36	Jurassic Park IV	US	NaN	0.0	7.0	5393
37	tt0369610	44	Jurassic Park 4	US	NaN	0.0	7.0	5393
...	...	...	...	...	...	...	...	
45085	tt5649108	2	Thoroughbreds	US	NaN	0.0	6.7	250
45099	tt6139732	13	Aladdin	US	NaN	0.0	7.4	575
45100	tt6139732	13	Aladdin	US	NaN	0.0	7.4	575
45133	tt6139732	29	Aladdin	US	en	0.0	7.4	575
45134	tt6139732	29	Aladdin	US	en	0.0	7.4	575

1213 rows × 18 columns

In [12]:
```
fltr_ratingdf.drop('ordering', axis=1, inplace=True)
fltr_ratingdf
```

Out[12]:

	title_id	title	region	language	is_original_title	averagerating	numvotes	prim
12	tt0369610	Jurassic World 3D	US	NaN	0.0	7.0	539338	Juras
20	tt0369610	Jurassic World	US	NaN	0.0	7.0	539338	Juras
21	tt0369610	Ebb Tide	US	NaN	0.0	7.0	539338	Juras
28	tt0369610	Jurassic Park IV	US	NaN	0.0	7.0	539338	Juras
37	tt0369610	Jurassic Park 4	US	NaN	0.0	7.0	539338	Juras
...	...	...	...	...	...	...	...	
45085	tt5649108	Thoroughbreds	US	NaN	0.0	6.7	25098	Thoro
45099	tt6139732	Aladdin	US	NaN	0.0	7.4	57549	
45100	tt6139732	Aladdin	US	NaN	0.0	7.4	57549	
45133	tt6139732	Aladdin	US	en	0.0	7.4	57549	
45134	tt6139732	Aladdin	US	en	0.0	7.4	57549	

1213 rows × 17 columns

In [13]:
```
fltr_ratingdf.drop('id', axis=1, inplace=True)
fltr_ratingdf
```

Out[13]:

	title_id	title	region	language	is_original_title	averagerating	numvotes	prim
12	tt0369610	Jurassic World 3D	US	NaN	0.0	7.0	539338	Juras
20	tt0369610	Jurassic World	US	NaN	0.0	7.0	539338	Juras
21	tt0369610	Ebb Tide	US	NaN	0.0	7.0	539338	Juras
28	tt0369610	Jurassic Park IV	US	NaN	0.0	7.0	539338	Juras
37	tt0369610	Jurassic Park 4	US	NaN	0.0	7.0	539338	Juras
...	...	...	...	...	...	...	...	
45085	tt5649108	Thoroughbreds	US	NaN	0.0	6.7	25098	Thoro
45099	tt6139732	Aladdin	US	NaN	0.0	7.4	57549	
45100	tt6139732	Aladdin	US	NaN	0.0	7.4	57549	
45133	tt6139732	Aladdin	US	en	0.0	7.4	57549	
45134	tt6139732	Aladdin	US	en	0.0	7.4	57549	

1213 rows × 16 columns

In [14]:
```
fltr_ratingdf.drop('is_original_title', axis=1, inplace=True)
fltr_ratingdf
```

Out[14]:

	title_id	title	region	language	averagerating	numvotes	primary_title	origir
12	tt0369610	Jurassic World 3D	US	NaN	7.0	539338	Jurassic World	Jurassi
20	tt0369610	Jurassic World	US	NaN	7.0	539338	Jurassic World	Jurassi
21	tt0369610	Ebb Tide	US	NaN	7.0	539338	Jurassic World	Jurassi
28	tt0369610	Jurassic Park IV	US	NaN	7.0	539338	Jurassic World	Jurassi
37	tt0369610	Jurassic Park 4	US	NaN	7.0	539338	Jurassic World	Jurassi
...	...	...	...	...	...	...	...	
45085	tt5649108	Thoroughbreds	US	NaN	6.7	25098	Thoroughbreds	Thorou
45099	tt6139732	Aladdin	US	NaN	7.4	57549	Aladdin	
45100	tt6139732	Aladdin	US	NaN	7.4	57549	Aladdin	
45133	tt6139732	Aladdin	US	en	7.4	57549	Aladdin	
45134	tt6139732	Aladdin	US	en	7.4	57549	Aladdin	

1213 rows × 15 columns

Now we will be searching for duplicate values based off the values in title_id. After we have the duplicates they will be deleted from the df.

In [15]:
```
fltrdup_df = fltr_ratingdf[fltr_ratingdf['title_id'].duplicated()]
fltrdup_df.head()
```

Out[15]:

	title_id	title	region	language	averagerating	numvotes	primary_title	original_title	star
20	tt0369610	Jurassic World	US	NaN	7.0	539338	Jurassic World	Jurassic World	
21	tt0369610	Ebb Tide	US	NaN	7.0	539338	Jurassic World	Jurassic World	
28	tt0369610	Jurassic Park IV	US	NaN	7.0	539338	Jurassic World	Jurassic World	
37	tt0369610	Jurassic Park 4	US	NaN	7.0	539338	Jurassic World	Jurassic World	
50	tt0401729	A Princess of Mars	US	NaN	6.6	241792	John Carter	John Carter	

In [16]:
```
nodupe_df = fltr_ratingdf.drop_duplicates(subset=['title_id'])
nodupe_df.head()
```

Out[16]:

	title_id	title	region	language	averagerating	numvotes	primary_title	original_title	st
12	tt0369610	Jurassic World 3D	US	NaN	7.0	539338	Jurassic World	Jurassic World	
49	tt0401729	John Carter of Mars	US	NaN	6.6	241792	John Carter	John Carter	
189	tt1194173	Marcher	US	NaN	6.7	268678	The Bourne Legacy	The Bourne Legacy	
204	tt1219289	The Dark Fields	US	NaN	7.4	492490	Limitless	Limitless	
234	tt4597838	Limitless	US	NaN	6.7	10	Limitless	Limitless	

A movie where the world gross is 0.00 is not a movie that we feel a film company would be interested in so we will be deleting any that have that value.

In [17]:
```
zerosum_df = nodupe_df[(nodupe_df['worldwide_gross'] == '$0')].index
len(zerosum_df)
```

Out[17]: 37

In [18]:
```
nodupe_df.drop(zerosum_df, inplace = True)

nodupe_df
```

Out[18]:

	title_id	title	region	language	averagerating	numvotes	primary_title	orig
12	tt0369610	Jurassic World 3D	US	NaN	7.0	539338	Jurassic World	Juras
49	tt0401729	John Carter of Mars	US	NaN	6.6	241792	John Carter	Jo
189	tt1194173	Marcher	US	NaN	6.7	268678	The Bourne Legacy	Th
204	tt1219289	The Dark Fields	US	NaN	7.4	492490	Limitless	
234	tt4597838	Limitless	US	NaN	6.7	10	Limitless	

We will now be creating a new column that is the value of 'worldwide_gross' divied by

'production_budget'. To do this, we will be using list comprehension.

In [19]:
```
nodupe_df['profit_margin'] = [int(x.strip('$').replace(',', '')) / int(y.strip('$
            for x, y in zip(nodupe_df['worldwide_gross'], nodupe_df['production_b
nodupe_df
```

```
C:\Users\melfr\Anaconda3\envs\learn-env\lib\site-packages\ipykernel_launcher.p
y: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: http://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
```

Out[19]:

	title_id	title	region	language	averagerating	numvotes	primary_title	origina
12	tt0369610	Jurassic World 3D	US	NaN	7.0	539338	Jurassic World	Jurassic
49	tt0401729	John Carter of Mars	US	NaN	6.6	241792	John Carter	John
189	tt1194173	Marcher	US	NaN	6.7	268678	The Bourne Legacy	The B l
204	tt1219289	The Dark Fields	US	NaN	7.4	492490	Limitless	Li
234	tt4597838	Limitless	US	NaN	6.7	10	Limitless	Li
...	...	...	...	...	...	...	...	
44915	tt5164432	Simon vs. the Homo Sapiens Agenda	US	NaN	7.6	81575	Love, Simon	Love,
44939	tt6053438	First Reformed	US	NaN	7.1	33542	First Reformed	First Ref
45063	tt5462602	The Big Sick	US	NaN	7.6	104008	The Big Sick	The B
45079	tt5649108	Thoroughbred	US	NaN	6.7	25098	Thoroughbreds	Thorough
45099	tt6139732	Aladdin	US	NaN	7.4	57549	Aladdin	A

742 rows × 16 columns

Any move that has a value less than 1 in 'profit_margin' has a low return that a film company may not be interested in, so we will be filtering out and getting rid of movies that have this value.

In [20]:
```
no_profit = nodupe_df[(nodupe_df['profit_margin'] <= 1)].index
len(no_profit)
```

Out[20]: 170

In [21]:
```
nodupe_df.drop(no_profit, inplace = True)
nodupe_df
```

Out[21]:

	title_id	title	region	language	averagerating	numvotes	primary_title	original_title
12	tt0369610	Jurassic World 3D	US	NaN	7.0	539338	Jurassic World	Jurassic World
49	tt0401729	John Carter of Mars	US	NaN	6.6	241792	John Carter	John Carter
189	tt1194173	Marcher	US	NaN	6.7	268678	The Bourne Legacy	The Bourne Legacy
204	tt1219289	The Dark Fields	US	NaN	7.4	492490	Limitless	Limitless
234	tt4597838	Limitless	US	NaN	6.7	10	Limitless	Limitless
...	...	...	...	...	...	...	...	...
44873	tt5117670	Peter Rabbit	US	NaN	6.6	27908	Peter Rabbit	Peter Rabbit
44915	tt5164432	Simon vs. the Homo Sapiens Agenda	US	NaN	7.6	81575	Love, Simon	Love, Simon
44939	tt6053438	First Reformed	US	NaN	7.1	33542	First Reformed	First Reformed
45063	tt5462602	The Big Sick	US	NaN	7.6	104008	The Big Sick	The Big Sick
45099	tt6139732	Aladdin	US	NaN	7.4	57549	Aladdin	Aladdin

572 rows × 16 columns

In [22]:
```
top100_df = nodupe_df.sort_values(by =['profit_margin', 'worldwide_gross',], asce
```

In [23]:
```
top100_df
```

Out[23]:

	title_id	title	region	language	averagerating	numvotes	primary_title	original_title
22295	tt1591095	The Further	US	NaN	6.9	254197	Insidious	Insidious
21013	tt2519480	Benji	US	NaN	7.4	101	Benji	Benji
22087	tt4972582	Untitled M. Night Shyamalan Project	US	NaN	7.3	358543	Split	Split
40606	tt5052448	Get Out	US	NaN	7.7	400474	Get Out	Get Out
41896	tt4975722	Moonlight	US	NaN	7.4	227964	Moonlight	Moonlight
...	...	...	...	...	...	...	...	...
8772	tt1490017	Lego: The Piece of Resistance	US	NaN	7.8	304179	The Lego Movie	The Lego Movie
26505	tt2229499	Don Jon's Addiction	US	NaN	6.6	216350	Don Jon	Don Jon
12227	tt2194499	About Time	US	NaN	7.8	263136	About Time	About Time
19642	tt1270797	Antidote	US	NaN	6.7	275406	Venom	Venom
44704	tt6112220	The Butterfly Effect	US	en	6.9	104	The Butterfly Effect	Vidas Em Paralelo

100 rows × 16 columns

This API will use the title ID from the title_id column to get the filming location information of the movie that matches that ID from the IMDB website. After we have this data we will be creating a new column and putting the values received from the API as entries.

This API has a cap of 500 and is nearly at that cap. It should not be run again because it is nearly at that cap. Later down the CSV is saved with these values so it does not need to be run.

In [24]:
```
def getrequest(ID):
    url = "https://rapidapi.p.rapidapi.com/title/get-filming-locations"
    querystring = {"tconst":ID}

    headers = {
            'x-rapidapi-host': "imdb8.p.rapidapi.com",
            'x-rapidapi-key': "9d73fcadbamshcf027aa66df7320p145039jsn1bc386586e4c
            }

    response = requests.request("GET", url, headers=headers, params=querystring)
    if 'locations' in response.json():
        return response.json()['locations']
    else:
        return None

movie_locations = []

for ID in top100_df['title_id']:
    locations_info = getrequest(ID)
    locations = []
    if locations_info:
        for location in locations_info:
            locations.append(location['location'])
    movie_locations.append(locations)
```

In [25]:
```
top100_df['movie_locations'] = movie_locations
top100_df.head()
```

Out[25]:

	title_id	title	region	language	averagerating	numvotes	primary_title	original_title
22295	tt1591095	The Further	US	NaN	6.9	254197	Insidious	Insidious
21013	tt2519480	Benji	US	NaN	7.4	101	Benji	Benji
22087	tt4972582	Untitled M. Night Shyamalan Project	US	NaN	7.3	358543	Split	Split
40606	tt5052448	Get Out	US	NaN	7.7	400474	Get Out	Get Out
41896	tt4975722	Moonlight	US	NaN	7.4	227964	Moonlight	Moonlight

In [139]:
```
top100_df['movie_locations'].values
```

Out[139]:
```
array([list(['Herald Examiner - 1111 S. Broadway, Downtown, Los Angeles, Cali
fornia, USA', '1153 S Point View St, Los Angeles, California, USA', '4350 Vic
toria Park Drive, Los Angeles, California, USA', 'Los Angeles, California, US
A']),
       list([]),
       list(['Philadelphia Zoo - 3400 W. Girard Avenue, Philadelphia, Pennsyl
vania, USA', 'Silk City Diner Bar & Lounge - 435 Spring Garden St, Philadelph
ia, Pennsylvania, USA', 'Sun Center Studios, Aston, Pennsylvania, USA', '30th
Street Station - 3001 Market Street, Philadelphia, Pennsylvania, USA', 'Fishe
rman Restaurant, 440 Schuylkill Rd, Phoenixville, Pennsylvania, USA', 'New Je
rsey, USA', 'Philadelphia, Pennsylvania, USA', 'Pennsylvania, USA', 'USA']),
       list(['Fairhope, Alabama, USA', '6892 Heathcroft Lane, Fairhope, Alaba
ma, USA', 'Alabama, USA', 'Mobile, Alabama, USA', 'Ryan Avenue, Mobile, Alaba
ma, USA']),
       list(['Miramar High School, Miramar, Florida, USA', "Jimmy's Eastside
Diner, 7201 Biscayne Blvd., Miami, Florida, USA", 'Miami, Florida, USA', 'Lib
erty City, Florida, USA', 'Virginia Key, Florida, USA']),
       list(['Pune, Maharashtra, India', 'Ludhiana, Punjab, India', 'Delhi, I
ndia']),
```

This API will get the coordinates of each location from our 'movie_locations' column in the top100_df by using an API provided by Google.

In [26]:
```
googleAPIkey= 'use key here'

def getcoords(address):
    url = "https://maps.googleapis.com/maps/api/geocode/json?address=" + address
    response = requests.request("GET", url)
    if len(response.json()['results']) > 0:
        return response.json()['results'][0]['geometry']['location']
    else:
        return None

AddressCoords = []
for locations in top100_df['movie_locations']:
    MovieLocations = []
    for Address in locations:
        coords = getcoords(Address)
        MovieLocations.append(coords)
    AddressCoords.append(MovieLocations)
```

In [27]:
```
top100_df['location_coords'] = AddressCoords
top100_df.head()
```

		Further						
21013	tt2519480	Benji	US	NaN	7.4	101	Benji	Benj
22087	tt4972582	Untitled M. Night Shyamalan Project	US	NaN	7.3	358543	Split	Spli
40606	tt5052448	Get Out	US	NaN	7.7	400474	Get Out	Get Ou
41896	tt4975722	Moonlight	US	NaN	7.4	227964	Moonlight	Moonligh

Splitting the genres into their own columns by using list comprehension again.

In [147]:
```
for number in range(3):
    top100_df['genre' + str(number + 1)]= [genre.split(",")[number] if len(genre.
                                           for genre in top100_df['genres']]

top100_df.head()
```

Out[147]:

e_minutes	...	release_date	production_budget	domestic_gross	worldwide_gross	profit_margin	mov
103.0	...	Apr 1, 2011	$1,500,000	$54,009,150	$99,870,886	66.580591	[He
79.0	...	Nov 15, 1974	$500,000	$31,559,560	$31,559,560	63.119120	
117.0	...	Jan 20, 2017	$5,000,000	$138,141,585	$278,964,806	55.792961	[Phi - 3
104.0	...	Feb 24, 2017	$5,000,000	$176,040,665	$255,367,951	51.073590	A 68
111.0	...	Oct 21, 2016	$1,500,000	$27,854,931	$65,245,512	43.497008	Sch Flo

Now that we have our dataframe set up with content that a film company may find interesting. We are ready to do more analysis and create data visualizations. We start by importing various libraries that give us many different options on how we'd like to create our visualizations.

In [176]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import plotly
import plotly.offline as py
import plotly.graph_objs as go
import plotly.express as px
import plotly.io as pio
from plotly.subplots import make_subplots
plotly.offline.init_notebook_mode()
import cufflinks as cf
import plotly.offline
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)
```

Because some of the numerical values in our top100_df are strings and have non-number values, we will need to convert these to integers. We will also create a new dataframe that contains all the columns of interest in our original df that has any numerical values as integers to make calculations easier. To convert the values we made a function that uses list comprehension and will call that function to convert old columns and put them into our new dataframe.

In [197]:
```
def CleanNumbers(column):
    return [int(x.strip('$').replace(',', '')) for x in top100_df[column]]
```

In [201]:
```
plotdf = pd.DataFrame()
plotdf['Title'] = top100_df['primary_title']
plotdf['Ratings'] = top100_df['averagerating']
plotdf['Reviews'] = top100_df['numvotes']
plotdf['Runtime'] = top100_df['runtime_minutes']
plotdf['Budget'] = CleanNumbers('production_budget')
plotdf['World_Gross'] = CleanNumbers('worldwide_gross')
plotdf['Profit_Return'] = top100_df['profit_margin']
plotdf['Genre1'] = top100_df['genre1']
plotdf['Genre2'] = top100_df['genre2']
plotdf['Genre3'] = top100_df['genre3']

plotdf.head()
```

Out[201]:

	Title	Ratings	Reviews	Runtime	Budget	World_Gross	Profit_Return	Genre1	G
22295	Insidious	6.9	254197	103.0	1500000	99870886	66.580591	Horror	M
21013	Benji	7.4	101	79.0	500000	31559560	63.119120	Documentary	
22087	Split	7.3	358543	117.0	5000000	278964806	55.792961	Horror	
40606	Get Out	7.7	400474	104.0	5000000	255367951	51.073590	Horror	M
41896	Moonlight	7.4	227964	111.0	1500000	65245512	43.497008	Drama	

Our first visualization will contain the movie data for the top 10 movies that had the highest world gross values.

In [318]:
```
Top10Gross = plotdf.sort_values('World_Gross', ascending = False).head(10)
Top10Gross
```

Out[318]:

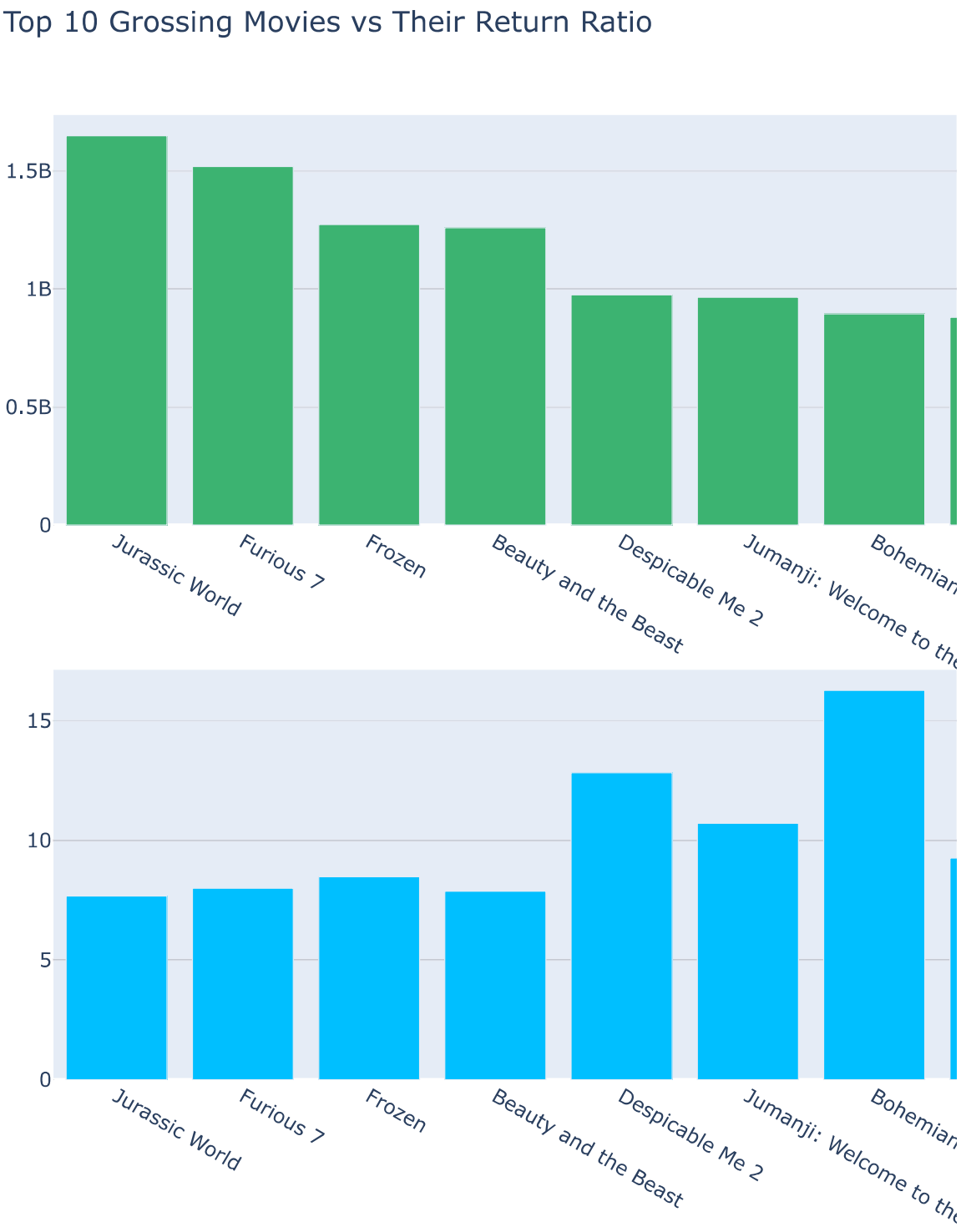
	Title	Ratings	Reviews	Runtime	Budget	World_Gross	Profit_Return	Genre1
12	Jurassic World	7.0	539338	124.0	215000000	1648854864	7.669092	Action
31782	Furious 7	7.2	335074	137.0	190000000	1518722794	7.993278	Action
3102	Frozen	7.5	516998	102.0	150000000	1272469910	8.483133	Adventure
17550	Beauty and the Beast	7.2	238325	129.0	160000000	1259199706	7.869998	Family
6495	Despicable Me 2	7.4	344230	98.0	76000000	975216835	12.831800	Adventure
17403	Jumanji: Welcome to the Jungle	7.0	242735	119.0	90000000	964496193	10.716624	Action
8014	Bohemian Rhapsody	8.0	345466	134.0	55000000	894985342	16.272461	Biography
3456	Ice Age: Continental Drift	6.6	175601	88.0	95000000	879765137	9.260686	Adventure
19642	Venom	6.7	275406	112.0	116000000	853628605	7.358867	Action
28506	Deadpool	8.0	820847	108.0	58000000	801025593	13.810786	Action

In [294]:
```
FigGross = make_subplots(rows=2, cols=1)

FigGross.append_trace(go.Bar(x= Top10Gross['Title'], y= Top10Gross['World_Gross']
                             marker_color= "mediumseagreen"), row=1, col=1)

FigGross.append_trace(go.Bar(x= Top10Gross['Title'], y= Top10Gross['Profit_Return
                             marker_color= "deepskyblue"), row=2, col=1)

FigGross.update_layout(height=800, width=1000, title_text="Top 10 Grossing Movies
FigGross.show()
```



The next visualization will contain the movie data for the top 10 movies with the highest profit return ratio.

In [220]:
```
Top10Returns= plotdf.sort_values('Profit_Return', ascending = False).head(11)
Top10Returns
```

Out[220]:

	Title	Ratings	Reviews	Runtime	Budget	World_Gross	Profit_Return	Genre1
22295	Insidious	6.9	254197	103.0	1500000	99870886	66.580591	Horror
21013	Benji	7.4	101	79.0	500000	31559560	63.119120	Documentary
22087	Split	7.3	358543	117.0	5000000	278964806	55.792961	Horror
40606	Get Out	7.7	400474	104.0	5000000	255367951	51.073590	Horror
41896	Moonlight	7.4	227964	111.0	1500000	65245512	43.497008	Drama
39525	Dangal	8.5	123638	161.0	9500000	294654618	31.016276	Action
10851	Lights Out	8.6	22	NaN	5000000	148806510	29.761302	Drama
10852	Lights Out	6.6	10	NaN	5000000	148806510	29.761302	Drama
35182	Sinister	6.8	198345	110.0	3000000	87727807	29.242602	Horror
40121	A Ghost Story	6.8	46280	92.0	100000	2769782	27.697820	Drama
21037	You're Next	6.6	79451	95.0	1000000	26887177	26.887177	Action

While we did filter out most of the duplicates much earlier, there were two instances of the movie "Lights Out" that had the same budget and world gross. The only differences were the ratings and reviews. In this instance we decided to drop the row, however we could have also gotten the mean of the ratings and summed the reviews together to combine these two rows.

In [228]:
```
Top10Returns = Top10Returns.drop(10852)
Top10Returns
```

Out[228]:

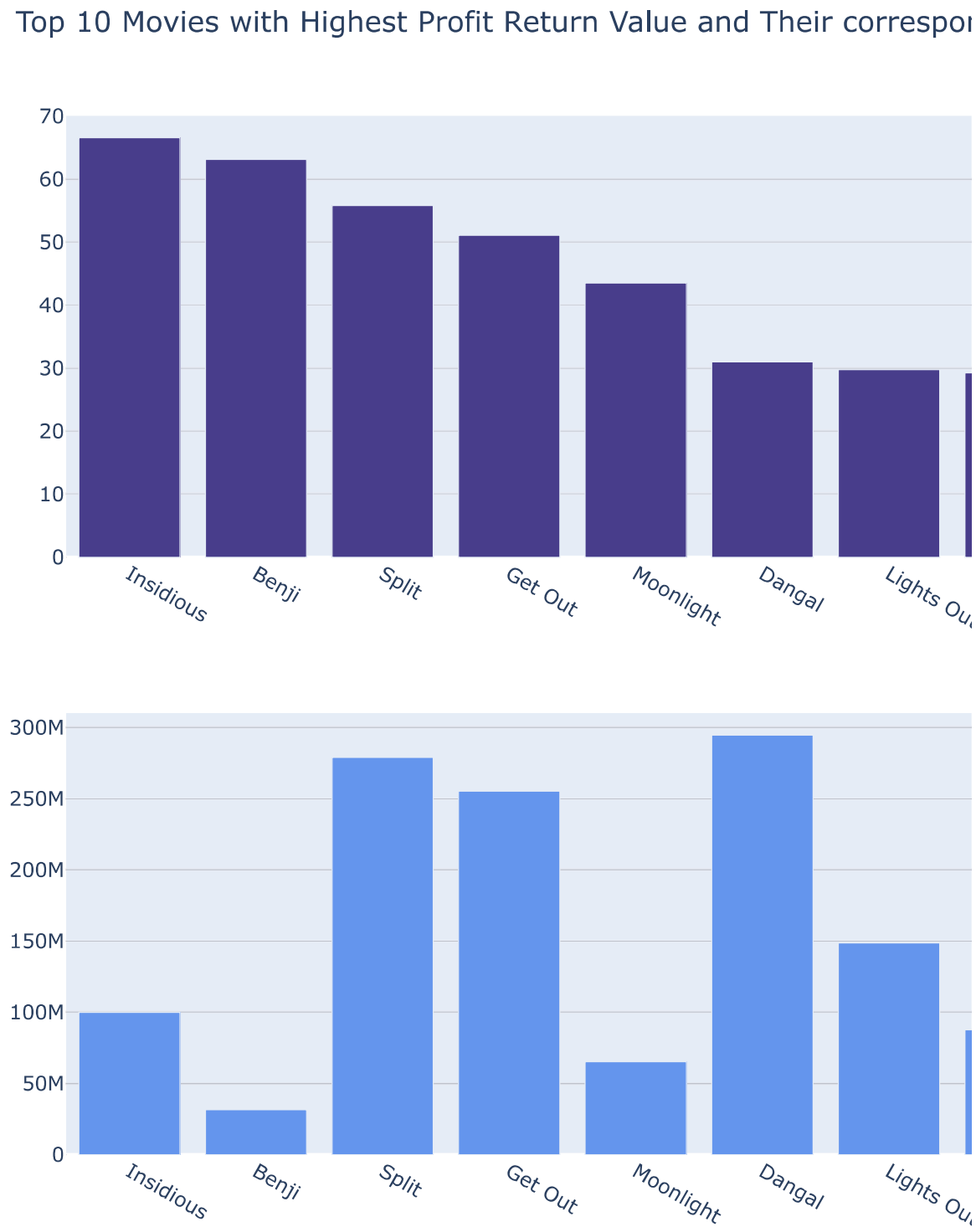
	Title	Ratings	Reviews	Runtime	Budget	World_Gross	Profit_Return	Genre1
22295	Insidious	6.9	254197	103.0	1500000	99870886	66.580591	Horror
21013	Benji	7.4	101	79.0	500000	31559560	63.119120	Documentary
22087	Split	7.3	358543	117.0	5000000	278964806	55.792961	Horror
40606	Get Out	7.7	400474	104.0	5000000	255367951	51.073590	Horror
41896	Moonlight	7.4	227964	111.0	1500000	65245512	43.497008	Drama
39525	Dangal	8.5	123638	161.0	9500000	294654618	31.016276	Action
10851	Lights Out	8.6	22	NaN	5000000	148806510	29.761302	Drama
35182	Sinister	6.8	198345	110.0	3000000	87727807	29.242602	Horror
40121	A Ghost Story	6.8	46280	92.0	100000	2769782	27.697820	Drama
21037	You're Next	6.6	79451	95.0	1000000	26887177	26.887177	Action

In [316]:
```
FigReturn = make_subplots(rows=2, cols=1)

FigReturn.append_trace(go.Bar(x= Top10Returns['Title'], y= Top10Returns['Profit_R
                              marker_color= "darkslateblue"), row=1, col=1)

FigReturn.append_trace(go.Bar(x= Top10Returns['Title'], y= Top10Returns['World_Gr
                              marker_color= "cornflowerblue"), row=2, col=1)

FigReturn.update_layout(height=800, width=1000, title_text="Top 10 Movies with Hi
FigReturn.show()
```



To create visualizations for our location data, we will be making a DF and adding content from our top100_df. We want the DF to be in long format with each latitude/longitude pair to have its own row. This starts with creating an empty dataframe and then filling it with a loop.

In [159]:
```
df = pd.DataFrame()
df['Title'] = []
df['Latitude'] = []
df['Longitude']= []
df['Profit_Return'] = []

df
```

Out[159]:

Title	Latitude	Longitude	Profit_Return

In [160]:
```
for index, row in top100_df.iterrows():
    for dictcoords in row['location_coords']:
        if dictcoords:
            df = df.append({'Title': row['primary_title'], 'Latitude': dictcoords
                            'Profit_Return': row['profit_margin']}, ignore_index

df.head()
```

Out[160]:

	Title	Latitude	Longitude	Profit_Return
0	Insidious	34.039345	-118.259639	66.580591
1	Insidious	34.054792	-118.369091	66.580591
2	Insidious	34.046743	-118.330458	66.580591
3	Insidious	34.052234	-118.243685	66.580591
4	Split	39.972083	-75.195849	55.792961

Here is a map of filming locations of the top 100 movies with the most profit return.

In [89]:
```
fig = px.scatter_mapbox(df, lat="Latitude", lon="Longitude", hover_name="Title",
                        color_discrete_sequence=["green"], zoom=3, height=300)
fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

We now add a third condition from the Profit_Return column. We can now see the movies that had the highest profit return (worldwide gross divided by production budget) more brightly colored in yellow, and the movies that had a lower return in a deep purple.

In [319]:
```
fig = go.Figure(go.Densitymapbox(lat=df.Latitude, lon=df.Longitude, z=df.Profit_R
                                 radius=10))
fig.update_layout(mapbox_style="stamen-terrain", mapbox_center_lon=180)
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

The new dataframes we have created are now saved for further analysis.

In [317]:
```
top100_df.to_csv(r'C:\Users\melfr\Documents\Flatiron\p1\labs\dsc-phase-1-project-
df.to_csv(r'C:\Users\melfr\Documents\Flatiron\p1\labs\dsc-phase-1-project-online\
plotdf.to_csv(r'C:\Users\melfr\Documents\Flatiron\p1\labs\dsc-phase-1-project-onl
```