Current Movie Industry Analysis

Author: Akash Ghosh

Overview

I have been tasked with assisting Microsoft in their venture into the movie industry. My goal was to explore what type of films are currently doing the best at the box office and to provide these findings to Microsoft's new movie studio executives. My analysis of the movie industry, achieved by garnering data and utilizing descriptive statistics and visualizations, has shown that a larger budget is correlated with a higher worldwide box office gross. By allocating 75 million to 200 million dollars to produce an animated musical movie released in June or November, or allocating 200 to 400 million dollars to produce a live action superhero movie released in April or May, the data shows that a movie studio will be extremely likely to succeed. I have also given recommendations as to which composers should be hired for an animated musical movie, and which directors should be hired for a superhero movie. Microsoft can use this report to target their production budget, genre, creative type, production method, release-time, and crew members of their upcoming movie endeavors to generate the highest amount of revenue possible.

Business Problem

I have been informed that Microsoft wants a piece of the multi-billion dollar movie-making industry, but that they are unsure of where to begin. The challenge for their new movie studio is that they are ready to jump into the industry but do not have the necessary knowledge to move forward. To assist them with this goal, I have been looking at the movies that performed highest in worldwide box office amounts for the past ten years. By analyzing the movies that have been most successful recently, I can make recommendations about attributes that Microsoft's movies should have in order to achieve the highest revenue. I have based my analysis on four main factors:

- Movie Type (Genre/Creative Type/Production Method): What types of movie content are currently most successful?
- Release Month: When is the most lucrative time of year to release a movie?
- Production Budget: What budget amount tends to achieve the highest box office gross?
- Additional Attributes: Based on these findings, what else do top-grossing movies have in common?

I chose these questions after considering the business problem and combing through the data I obtained. I have determined that the answers to these questions are integral to the steps that should be taken when considering how to produce the most profitable movie in today's world.

Data Understanding

I utilized three different data sources for my analysis in order to have the most comprehensive view of the industry as it currently is.

- OpusData Movie Data: a free dataset available upon request for academic research, comprised of 1,900 movies with a production year from 2006 to 2018, with a production budget greater than or equal to ten million dollars. This dataset contains values for movie name, production budget, domestic and international gross, genre, production method, runtime, and movie board rating.
- Web-scraped data from The-Numbers.com: The Numbers is described as "the premier provider of movie industry data and research services". This website contains domestic, international, and worldwide box office revenue amounts per movie, and allows filtering and ordering of results based on many different criteria. Some of the criteria provided on this site that I found especially useful were the same criteria listed above: title, production budget, domestic and international gross, genre, and production method, in addition to release date and worldwide gross. For the purposes of this project, I generated and scraped reports for the top 100 movies per year, in terms of revenue, from 2010 to 2020.
- The Movie Database (TMDb) API: The Movie Database is a popular database for movies and TV shows. Their API is a system made freely available for data acquisition. There is a very large amount of data available on TMDb; for the purposes of this project, I used it mainly to fill in missing information from my other two datasets as I moved through my analysis.

In [2]:
```
# importing the packages I will be using for this project
import pandas as pd
# setting pandas display to avoid scientific notation in my dataframes
pd.options.display.float_format = '{:.2f}'.format
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from bs4 import BeautifulSoup
import requests

%matplotlib inline
```

OpusData Movie Data

In [3]:
```
# reading the csv file
opus_df = pd.read_csv('/Users/dtunnicliffe/Downloads/MovieData.csv')
# previewing the DataFrame
opus_df.head()
```

Out[3]:

	movie_name	production_year	movie_odid	production_budget	domestic_box_office	internation
0	Madea's Family Reunion	2006	8220100	10000000	63257940	
1	Krrish	2006	58540100	10000000	1430721	
2	End of the Spear	2006	34620100	10000000	11748661	
3	A Prairie Home Companion	2006	24910100	10000000	20342852	
4	Saw III	2006	5840100	10000000	80238724	

In [4]:
```
# getting info for DataFrame
opus_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1936 entries, 0 to 1935
Data columns (total 13 columns):
movie_name                  1936 non-null object
production_year             1936 non-null int64
movie_odid                  1936 non-null int64
production_budget           1936 non-null int64
domestic_box_office         1936 non-null int64
international_box_office    1936 non-null int64
rating                      1913 non-null object
creative_type               1923 non-null object
source                      1915 non-null object
production_method           1925 non-null object
genre                       1926 non-null object
sequel                      1934 non-null float64
running_time                1822 non-null float64
dtypes: float64(2), int64(5), object(6)
memory usage: 196.8+ KB
```

In [5]:
```
# getting counts for each value in genre column
opus_df['genre'].value_counts()
```

Out[5]:
```
Drama                  471
Adventure              334
Comedy                 318
Action                 311
Thriller/Suspense      231
Horror                 104
Romantic Comedy         82
Musical                 25
Black Comedy            24
Western                 15
Concert/Performance      6
Documentary              5
Name: genre, dtype: int64
```

In [6]:
```
# generating descriptive statistics for domestic box office values
opus_df['domestic_box_office'].describe()
```

Out[6]:
```
count        1936.00
mean     64329960.75
std      87724369.60
min             0.00
25%      11003050.25
50%      36329945.00
75%      80069777.50
max     936662225.00
Name: domestic_box_office, dtype: float64
```

In [7]:
```
# generating descriptive statistics for production budget values
opus_df['production_budget'].describe()
```

Out[7]:
```
count         1936.00
mean      53428574.38
std       53685623.15
min       10000000.00
25%       19000000.00
50%       32750000.00
75%       65000000.00
max      425000000.00
Name: production_budget, dtype: float64
```

The-Numbers Web Scraping

2010

My first step was to obtain data for the top 100 grossing movies of 2010. I did this by building a report on The-Numbers.com, scraping this data, and reading it into a pandas DataFrame.

In [8]:
```python
# url for the full customized report of top 100 movies for 2010
url = f"https://www.the-numbers.com/movies/report/All/All/All/All/All/All/All/All/None/None/2010/2010/None/None/None/None/None/None?show-release-date=On&view-order-by=domestic-box-office&show-release-year=On&view-order-direction=desc&show-production-budget=On&show-domestic-box-office=On&show-international-box-office=On&show-worldwide-box-office=On&show-genre=On&show-production-method=On&show-creative-type=On"
response = requests.get(url)
# creating soup
soup = BeautifulSoup(response.text, 'lxml')
# finding table containing report info
table = soup.find('table')
# converting html of table into a string
table_string = f"""{table}"""
# reading html string into pandas
table_read = pd.read_html(table_string)
# converting into DataFrame
numbers_2010 = table_read[0]
# previewing DataFrame
numbers_2010.head()
```

Out[8]:

	Unnamed: 0	Released	Released.1	Title	Genre	ProductionMethod	CreativeType	Pro
0	1	Jun 18, 2010	2010	Toy Story 3	Adventure	Digital Animation	Kids Fiction	
1	2	Mar 5, 2010	2010	Alice in Wonderland	Adventure	Animation/Live Action	Fantasy	
2	3	May 7, 2010	2010	Iron Man 2	Action	Live Action	Super Hero	
3	4	Jun 30, 2010	2010	The Twilight Saga: Eclipse	Drama	Live Action	Fantasy	
4	5	Nov 19, 2010	2010	Harry Potter and the Deathly Hallows:...	Adventure	Animation/Live Action	Fantasy	

Now that I had a DataFrame to work with, I was able to start running some summary statistics and exploring the data.

In [9]:
```
# getting info for DataFrame
numbers_2010.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 11 columns):
Unnamed: 0                    100 non-null int64
Released                      100 non-null object
Released.1                    100 non-null int64
Title                         100 non-null object
Genre                         100 non-null object
ProductionMethod              100 non-null object
CreativeType                  100 non-null object
ProductionBudget              100 non-null object
DomesticBox   Office      100   non-null   object
InternationalBox  Office   100  non-null  object
WorldwideBox   Office     100   non-null   object
dtypes: int64(2), object(9)
memory usage: 8.7+ KB
```

In [10]:
```
# getting descriptive statistics for DataFrame
numbers_2010.describe()
```

Out[10]:

	Unnamed: 0	Released.1
count	100.00	100.00
mean	50.50	2010.00
std	29.01	0.00
min	1.00	2010.00
25%	25.75	2010.00
50%	50.50	2010.00
75%	75.25	2010.00
max	100.00	2010.00

In [11]:
```
# retrieving data type for domestic box office column values
numbers_2010['DomesticBox Office'].dtype
```

Out[11]:
```
dtype('O')
```

I noted that the describe method for this DataFrame was not very helpful at this point because my dollar amounts for domestic, international, and worldwide gross were pulled as objects (not floats or integers). I knew that would require further adjusting in the next stage.

In [12]:
```
# getting value counts for genre column
numbers_2010.Genre.value_counts()
```

Out[12]:
```
Adventure            21
Drama                18
Comedy               16
Action               13
Thriller/Suspense    11
Horror                9
Romantic Comedy       8
Western               1
Documentary           1
Black Comedy          1
Musical               1
Name: Genre, dtype: int64
```

Adventure and Drama were the most common movie genres for top grossing movies in 2010.

In [13]:
```
# getting value counts for production method column
numbers_2010['ProductionMethod'].value_counts()
```

Out[13]:
```
Live Action              85
Animation/Live Action     8
Digital Animation         7
Name: ProductionMethod, dtype: int64
```

For 2010 production methods, Live Action was by far the most common, with 85% of the top grossing movies being of this type.

In [14]:
```
# getting value counts for creative type column
numbers_2010['CreativeType'].value_counts()
```

Out[14]:
```
Contemporary Fiction    56
Fantasy                 14
Kids Fiction            10
Science Fiction          7
Historical Fiction       5
Dramatization            5
Factual                  2
Super Hero               1
Name: CreativeType, dtype: int64
```

Contemporary Fiction was the most common creative type by far for the top movies made in 2010.

Since I knew I'd be scraping data for each year in the exact same way as above, I created a function to speed up the process.

```
In [15]: def number_scraper(year):
    """
    Scrapes 100 top-grossing movies from The-Numbers.com.

    Adds year input to url and scrapes resulting table.

    Parameters:
    year (int): user input 4-digit year for movie gross to be scraped.

    Returns:
    numbers_df (Pandas DataFrame): A dataframe populated with movie gros
s table values.

    """
    # url for the full customized report of top 100 movies for release y
ears in range listed
    url = f"https://www.the-numbers.com/movies/report/All/All/All/All/Al
l/All/All/All/All/None/None/{year}/{year}/None/None/None/None/None/None?
show-release-date=On&view-order-by=domestic-box-office&show-release-year
=On&view-order-direction=desc&show-production-budget=On&show-domestic-bo
x-office=On&show-international-box-office=On&show-worldwide-box-office=O
n&show-genre=On&show-production-method=On&show-creative-type=On"
    response = requests.get(url)
    # creating soup
    soup = BeautifulSoup(response.text, 'lxml')
    # finding table
    table = soup.find('table')
    # converting html of table into string
    table_string = f"""{table}"""
    # reading html string into pandas
    table_read = pd.read_html(table_string)
    # converting into DataFrame
    numbers_df = table_read[0]
    return numbers_df
```

2011

In [16]:
```
# scraping 2011 data and compiling into DataFrame
numbers_2011 = number_scraper(2011)
#previewing DataFrame
numbers_2011.head()
```

Out[16]:

	Unnamed: 0	Released	Released.1	Title	Genre	ProductionMethod	CreativeType	P
0	1	Jul 15, 2011	2011	Harry Potter and the Deathly Hallows:...	Adventure	Animation/Live Action	Fantasy	
1	2	Jun 29, 2011	2011	Transformers: Dark of the Moon	Action	Animation/Live Action	Science Fiction	
2	3	Nov 18, 2011	2011	The Twilight Saga: Breaking Dawn, Part 1	Drama	Live Action	Fantasy	
3	4	May 26, 2011	2011	The Hangover Part II	Comedy	Live Action	Contemporary Fiction	
4	5	May 20, 2011	2011	Pirates of the Caribbean: On Stranger...	Adventure	Live Action	Fantasy	

2012

In [17]:
```
# scraping 2012 data and compiling into DataFrame
numbers_2012 = number_scraper(2012)
# previewing DataFrame
numbers_2012.head()
```

Out[17]:

	Unnamed: 0	Released	Released.1	Title	Genre	ProductionMethod	CreativeTy
0	1	May 4, 2012	2012	The Avengers	Action	Animation/Live Action	Super He
1	2	Jul 20, 2012	2012	The Dark Knight Rises	Action	Live Action	Super He
2	3	Mar 23, 2012	2012	The Hunger Games	Thriller/Suspense	Live Action	Scien Ficti
3	4	Nov 8, 2012	2012	Skyfall	Action	Live Action	Contempora Ficti
4	5	Dec 14, 2012	2012	The Hobbit: An Unexpected Journey	Adventure	Animation/Live Action	Fanta

2013

In [18]:
```
# scraping 2013 data and compiling into DataFrame
numbers_2013 = number_scraper(2013)
# previewing DataFrame
numbers_2013.head()
```

Out[18]:

	Unnamed: 0	Released	Released.1	Title	Genre	ProductionMethod	CreativeType	Pro
0	1	Nov 22, 2013	2013	The Hunger Games: Catching Fire	Adventure	Live Action	Science Fiction	
1	2	May 3, 2013	2013	Iron Man 3	Action	Animation/Live Action	Super Hero	
2	3	Nov 22, 2013	2013	Frozen	Musical	Digital Animation	Kids Fiction	
3	4	Jul 3, 2013	2013	Despicable Me 2	Adventure	Digital Animation	Kids Fiction	
4	5	Jun 14, 2013	2013	Man of Steel	Action	Live Action	Super Hero	

2014

In [19]:
```
# scraping 2014 data and compiling into DataFrame
numbers_2014 = number_scraper(2014)
# previewing DataFrame
numbers_2014.head()
```

Out[19]:

	Unnamed: 0	Released	Released.1	Title	Genre	ProductionMethod	CreativeTyp
0	1	Dec 25, 2014	2014	American Sniper	Drama	Live Action	Dramatizatio
1	2	Nov 21, 2014	2014	The Hunger Games: Mockingjay - Part 1	Thriller/Suspense	Live Action	Scienc Fictio
2	3	Aug 1, 2014	2014	Guardians of the Galaxy	Action	Animation/Live Action	Super Her
3	4	Apr 4, 2014	2014	Captain America: The Winter Soldier	Action	Live Action	Super Her
4	5	Feb 7, 2014	2014	The Lego Movie	Adventure	Digital Animation	Kids Fictio

2015

In [20]:

```
# scraping 2015 data and compiling into DataFrame
numbers_2015 = number_scraper(2015)
# previewing DataFrame
numbers_2015.head()
```

Out[20]:

	Unnamed: 0	Released	Released.1	Title	Genre	ProductionMethod	CreativeType	Prod
0	1	Dec 18, 2015	2015	Star Wars Ep. VII: The Force Awakens	Adventure	Animation/Live Action	Science Fiction	
1	2	Jun 12, 2015	2015	Jurassic World	Action	Live Action	Science Fiction	
2	3	May 1, 2015	2015	Avengers: Age of Ultron	Action	Animation/Live Action	Super Hero	
3	4	Jun 19, 2015	2015	Inside Out	Adventure	Digital Animation	Kids Fiction	
4	5	Apr 3, 2015	2015	Furious 7	Action	Live Action	Contemporary Fiction	

2016

In [21]:
```
# scraping 2016 data and compiling into DataFrame
numbers_2016 = number_scraper(2016)
# previewing the DataFrame
numbers_2016.head()
```

Out[21]:

	Unnamed: 0	Released	Released.1	Title	Genre	ProductionMethod	CreativeType	Produ
0	1	Dec 16, 2016	2016	Rogue One: A Star Wars Story	Adventure	Animation/Live Action	Science Fiction	$
1	2	Jun 17, 2016	2016	Finding Dory	Adventure	Digital Animation	Kids Fiction	$
2	3	May 6, 2016	2016	Captain America: Civil War	Action	Live Action	Super Hero	$
3	4	Jul 8, 2016	2016	The Secret Life of Pets	Adventure	Digital Animation	Kids Fiction	
4	5	Apr 15, 2016	2016	The Jungle Book	Adventure	Animation/Live Action	Fantasy	$

2017

In [22]:
```
# scraping 2017 data and compiling into DataFrame
numbers_2017 = number_scraper(2017)
# previewing the DataFrame
numbers_2017.head()
```

Out[22]:

	Unnamed: 0	Released	Released.1	Title	Genre	ProductionMethod	CreativeType	Prod
0	1	Dec 15, 2017	2017	Star Wars Ep. VIII: The Last Jedi	Adventure	Animation/Live Action	Science Fiction	
1	2	Mar 17, 2017	2017	Beauty and the Beast	Musical	Animation/Live Action	Fantasy	
2	3	Jun 2, 2017	2017	Wonder Woman	Action	Live Action	Super Hero	
3	4	Dec 20, 2017	2017	Jumanji: Welcome to the Jungle	Adventure	Live Action	Science Fiction	
4	5	May 5, 2017	2017	Guardians of the Galaxy Vol 2	Action	Animation/Live Action	Super Hero	

2018

In [23]:
```
# scraping 2018 data and compiling into DataFrame
numbers_2018 = number_scraper(2018)
# previewing the DataFrame
numbers_2018.head()
```

Out[23]:

	Unnamed: 0	Released	Released.1	Title	Genre	ProductionMethod	CreativeType	Prod
0	1	Feb 16, 2018	2018	Black Panther	Action	Live Action	Super Hero	
1	2	Apr 27, 2018	2018	Avengers: Infinity War	Action	Animation/Live Action	Super Hero	
2	3	Jun 15, 2018	2018	Incredibles 2	Adventure	Digital Animation	Kids Fiction	
3	4	Jun 22, 2018	2018	Jurassic World: Fallen Kingdom	Action	Live Action	Science Fiction	
4	5	Dec 21, 2018	2018	Aquaman	Action	Animation/Live Action	Super Hero	

2019

In [24]:
```
# scraping 2019 data and compiling into DataFrame
numbers_2019 = number_scraper(2019)
# previewing the DataFrame
numbers_2019.head()
```

Out[24]:

	Unnamed: 0	Released	Released.1	Title	Genre	ProductionMethod	CreativeType	Prod
0	1	Apr 26, 2019	2019	Avengers: Endgame	Action	Animation/Live Action	Super Hero	
1	2	Jul 19, 2019	2019	The Lion King	Adventure	Animation/Live Action	Kids Fiction	
2	3	Dec 20, 2019	2019	Star Wars: The Rise of Skywalker	Adventure	Animation/Live Action	Science Fiction	
3	4	Nov 22, 2019	2019	Frozen II	Adventure	Digital Animation	Kids Fiction	
4	5	Jun 21, 2019	2019	Toy Story 4	Adventure	Digital Animation	Kids Fiction	

2020

In [25]:
```
# scraping 2020 data and compiling into DataFrame
numbers_2020 = number_scraper(2020)
# previewing the DataFrame
numbers_2020.head()
```

Out[25]:

	Unnamed: 0	Released	Released.1	Title	Genre	ProductionMethod	CreativeType	Pr
0	1	Jan 17, 2020	2020	Bad Boys For Life	Action	Live Action	Contemporary Fiction	
1	2	Feb 14, 2020	2020	Sonic The Hedgehog	Adventure	Animation/Live Action	Kids Fiction	
2	3	Feb 7, 2020	2020	Birds of Prey (And the Fantabulous Em...	Action	Live Action	Super Hero	
3	4	Jan 17, 2020	2020	Dolittle	Adventure	Animation/Live Action	Fantasy	
4	5	Feb 28, 2020	2020	The Invisible Man	Horror	Live Action	Science Fiction	

Data Preparation

Now that my data was scraped ready to go, it was time to clean it up and prepare it for analysis.

OpusData Movie Data

Cleaning

There were a few columns in this dataset that were not relevant to my analysis: 'movie_odid', 'source', 'sequel', 'running-time', and 'rating'. I began by dropping those.

In [26]:
```
# dropping unnecessary columns
opus_df.drop(['movie_odid', 'source', 'sequel', 'running_time', 'rating'
], axis=1, inplace=True)
```

I then renamed some of the column names to make them easier to work with.

In [27]:
```
# renaming certain columns
opus_df = opus_df.rename(columns={'movie_name':'title','production_budge
t':'budget', 'domestic_box_office': 'dom_gross', 'international_box_offi
ce':'int_gross', 'production_method':'prod_method'})
```

In [28]:
```
# showing all column names
opus_df.columns
```

Out[28]:
```
Index(['title', 'production_year', 'budget', 'dom_gross', 'int_gross',
       'creative_type', 'prod_method', 'genre'],
      dtype='object')
```

In [29]:
```
# saving copy of DataFrame as csv file
opus_df.to_csv('./data/opus_df.csv')
```

Scraped data from The-Numbers.com

Cleaning

Due to the fact that I compiled my data from tables that were completely filled in, I was pleased that I had no null values to deal with. I did, however, have an unnecessary column called 'Unnamed: 0', which was used as the index of each table in its original html format. I began by dropping this.

In [30]:
```
numbers_2010.isnull().sum()
```

Out[30]:
```
Unnamed: 0                 0
Released                   0
Released.1                 0
Title                      0
Genre                      0
ProductionMethod           0
CreativeType               0
ProductionBudget           0
DomesticBox Office         0
InternationalBox Office    0
WorldwideBox Office        0
dtype: int64
```

In [31]:
```
# dropping Unnamed column
numbers_2010 = numbers_2010.drop(columns='Unnamed: 0')
```

I then made all the column names lowercase for ease of use.

In [32]:
```
# converting column names to all lowercase
numbers_2010.columns = [x.lower() for x in numbers_2010.columns]
# showing all column names
numbers_2010.columns
```

Out[32]:
```
Index(['released', 'released.1', 'title', 'genre', 'productionmethod',
       'creativetype', 'productionbudget', 'domesticbox office',
       'internationalbox office', 'worldwidebox office'],
      dtype='object')
```

I also wanted to rename some of the columns to make them more comprehensive.

```
In [33]: # renaming columns
numbers_2010 = numbers_2010.rename(columns = {'released':'release_date', 
'released.1':'release_year', 'productionmethod':'prod_method', 'domestic
box office':'dom_gross', 'internationalbox office':'int_gross', 'worldwi
debox office':'world_gross', 'creativetype': 'creative_type', 'productio
nbudget': 'budget'})
# showing all column names
numbers_2010.columns
```

```
Out[33]: Index(['release_date', 'release_year', 'title', 'genre', 'prod_method',
       'creative_type', 'budget', 'dom_gross', 'int_gross', 'world_gros
s'],
      dtype='object')
```

Finally, I wanted to convert the dollar amounts to numbers I could actually work with.

```
In [34]: # removing dollar signs and commas from dollar amounts
# converting dollar amounts from strings into integers
numbers_2010['dom_gross'] = numbers_2010['dom_gross'].str.replace(',', 
'').str.replace('$', '').astype(int)
numbers_2010['int_gross'] = numbers_2010['int_gross'].str.replace(',', 
'').str.replace('$', '').astype(int)
numbers_2010['world_gross'] = numbers_2010['world_gross'].str.replace(
',', '').str.replace('$', '').astype(int)
numbers_2010['budget'] = numbers_2010['budget'].str.replace(',', '').str
.replace('$', '').astype(int)
numbers_2010.head()
```

Out[34]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	do
0	Jun 18, 2010	2010	Toy Story 3	Adventure	Digital Animation	Kids Fiction	200000000	41
1	Mar 5, 2010	2010	Alice in Wonderland	Adventure	Animation/Live Action	Fantasy	200000000	33
2	May 7, 2010	2010	Iron Man 2	Action	Live Action	Super Hero	170000000	31
3	Jun 30, 2010	2010	The Twilight Saga: Eclipse	Drama	Live Action	Fantasy	68000000	30
4	Nov 19, 2010	2010	Harry Potter and the Deathly Hallows:...	Adventure	Animation/Live Action	Fantasy	125000000	29

Since this is how I intended to clean all my DataFrames, I wrote a function to execute all the steps I had taken.

In [35]:

```
def clean(df):
    """
    Cleans and modifies a given dataframe according to criteria set for
 this particular project.

    Drops column called 'Unnamed:0'.
    Converts column names to lowercase.
    Renames certain columns to make them shorter/more comprehensive.
    Removes dollar signs and commas from dollar amounts.
    Converts dollar amounts from strings into integers.

    Parameters:
    df (Pandas DataFrame): user input dataframe based on previously scra
ped table values from The-Numbers.com.

    Returns:
    df (Pandas DataFrame): A dataframe cleaned and adjusted as per crite
ria listed above.

    """
    # drop 'Unnamed' column
    df = df.drop(columns='Unnamed: 0')
    # make column names lowercase
    df.columns = [x.lower() for x in df.columns]
    # rename certain columns
    df = df.rename(columns = {'released':'release_date', 'released.1':'r
elease_year', 'productionmethod':'prod_method',
                              'domesticbox office':'dom_gross', 'interna
tionalbox office':'int_gross', 'worldwidebox office':'world_gross',
                              'creativetype':'creative_type', 'productio
nbudget': 'budget'})
    # removing dollar signs and commas from dollar amounts
    # converting dollar amounts from strings into integers
    df['dom_gross'] = df['dom_gross'].str.replace(',', '').str.replace(
'$', '').astype(int)
    df['int_gross'] = df['int_gross'].str.replace(',', '').str.replace(
'$', '').astype(int)
    df['world_gross'] = df['world_gross'].str.replace(',', '').str.repla
ce('$', '').astype(int)
    df['budget'] = df['budget'].str.replace(',', '').str.replace('$', ''
).astype(int)
    return df
```

In [36]:
```
# cleaning data
numbers_2011 = clean(numbers_2011)
# previewing cleaned data
numbers_2011.head()
```

Out[36]:

	release_date	release_year	title	genre	prod_method	creative_type	budget
0	Jul 15, 2011	2011	Harry Potter and the Deathly Hallows:...	Adventure	Animation/Live Action	Fantasy	125000000
1	Jun 29, 2011	2011	Transformers: Dark of the Moon	Action	Animation/Live Action	Science Fiction	195000000
2	Nov 18, 2011	2011	The Twilight Saga: Breaking Dawn, Part 1	Drama	Live Action	Fantasy	127500000
3	May 26, 2011	2011	The Hangover Part II	Comedy	Live Action	Contemporary Fiction	80000000
4	May 20, 2011	2011	Pirates of the Caribbean: On Stranger...	Adventure	Live Action	Fantasy	379000000

In [37]:
```
# cleaning data
numbers_2012 = clean(numbers_2012)
```

In [38]:
```
# cleaning data
numbers_2013 = clean(numbers_2013)
```

In [39]:
```
# cleaning data
numbers_2014 = clean(numbers_2014)
```

In [40]:
```
# cleaning data
numbers_2015 = clean(numbers_2015)
```

In [41]:
```
# cleaning data
numbers_2016 = clean(numbers_2016)
```

In [42]:
```
# cleaning data
numbers_2017 = clean(numbers_2017)
```

In [43]:
```
# cleaning data
numbers_2018 = clean(numbers_2018)
```

In [44]:
```
# cleaning data
numbers_2019 = clean(numbers_2019)
```

```
In [45]: # cleaning data
numbers_2020 = clean(numbers_2020)
```

Combining

Now that all the data had been cleaned, I was ready to combine all my DataFrames into a single DataFrame for further analysis.

```
In [46]: # concatenating my DataFrames for box office data from years 2015-2020
numbers_df = pd.concat([numbers_2010, numbers_2011, numbers_2012, number
s_2013,
                        numbers_2014, numbers_2015, numbers_2016, number
s_2017,
                        numbers_2018, numbers_2019, numbers_2020], axis=
0, ignore_index=True)
# getting info on my new DataFrame
numbers_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1100 entries, 0 to 1099
Data columns (total 10 columns):
release_date     1100 non-null object
release_year     1100 non-null int64
title            1100 non-null object
genre            1099 non-null object
prod_method    1100  non-null  object
creative_type   1100  non-null  object
budget           1100 non-null int64
dom_gross        1100 non-null int64
int_gross     1100   non-null   int64
world_gross    1100  non-null   int64
dtypes: int64(5), object(5)
memory usage: 86.1+ KB
```

At this point I realized that, somehow, one singular null value had snuck in there.

```
In [47]: # checking for null values
numbers_df.loc[numbers_df.genre.isnull()]
```

Out[47]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	dom_gross
1091	Jul 10, 2020	2020	Archive	NaN	Live Action	Science Fiction	0	139593

Since it was just one null value for genre for the movie 'Archive', it was easy enough to perform a quick search on IMDb. The primary genre listed for this movie is Drama, so I filled in the value manually.

In [48]:
```
# for movie with title value Archive, set genre to Drama
numbers_df.loc[numbers_df['title']=='Archive', 'genre'] = 'Drama'
```

In [49]:
```
# checking for successful value change
numbers_df.loc[numbers_df['title'] == 'Archive']
```

Out[49]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	dom_gross
1091	Jul 10, 2020	2020	Archive	Drama	Live Action	Science Fiction	0	139593

Feature Engineering

I was curious about how the timing of a release impacted the revenue. So my next step was to create a column for the month that each movie was released in.

In [50]:
```
# convert release date column to datetime values
numbers_df['release_date'] = pd.to_datetime(numbers_df['release_date'])
# create release month column
numbers_df['release_month'] = numbers_df['release_date'].dt.strftime('%
B')
```

In [51]:
```
# checking for successful column creation
numbers_df['release_month'].value_counts()
```

Out[51]:
```
February     104
December     104
August       104
November     103
July          97
March         92
January       89
June          89
September     86
October       85
May           79
April         68
Name: release_month, dtype: int64
```

In [52]:
```
# saving copy of DataFrame as csv file
numbers_df.to_csv('./data/thenumbers_df.csv')
```

Putting it all together!

In [53]:
```
# show all column names
numbers_df.columns
```

Out[53]:
```
Index(['release_date', 'release_year', 'title', 'genre', 'prod_method',
       'creative_type', 'budget', 'dom_gross', 'int_gross', 'world_gros
s',
       'release_month'],
      dtype='object')
```

In [54]:
```
# show number of rows and columns
numbers_df.shape
```

Out[54]:
```
(1100, 11)
```

In [55]:
```
# show all column names
opus_df.columns
```

Out[55]:
```
Index(['title', 'production_year', 'budget', 'dom_gross', 'int_gross',
       'creative_type', 'prod_method', 'genre'],
      dtype='object')
```

In [56]:
```
# show number of rows and columns of DataFrame
opus_df.shape
```

Out[56]:
```
(1936, 8)
```

In [57]:
```
# merging the DataFrames
merged_df = pd.merge(numbers_df, opus_df, how='outer')
# previewing the new DataFrame
merged_df.shape
```

Out[57]:
```
(2441, 12)
```

In [58]:
```
# filter DataFrame to only include movies with a production year greater
than or equal to 2010 (or null)
merged_df = merged_df[(merged_df['production_year']>=2010.00) | (merged_
df['production_year'].isnull())]
```

In [59]:
```
# show number of rows and columns
merged_df.shape
```

Out[59]:
```
(1745, 12)
```

In [60]:
```
# previewing DataFrame
merged_df.head()
```

Out[60]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	do
0	2010-06-18	2010.00	Toy Story 3	Adventure	Digital Animation	Kids Fiction	200000000	41
1	2010-03-05	2010.00	Alice in Wonderland	Adventure	Animation/Live Action	Fantasy	200000000	33
2	2010-05-07	2010.00	Iron Man 2	Action	Live Action	Super Hero	170000000	31
3	2010-06-30	2010.00	The Twilight Saga: Eclipse	Drama	Live Action	Fantasy	68000000	30
4	2010-11-19	2010.00	Harry Potter and the Deathly Hallows:...	Adventure	Animation/Live Action	Fantasy	125000000	29

In [61]:
```
# show all column names
merged_df.columns
```

Out[61]:
```
Index(['release_date', 'release_year', 'title', 'genre', 'prod_method',
       'creative_type', 'budget', 'dom_gross', 'int_gross', 'world_gros
s',
       'release_month', 'production_year'],
      dtype='object')
```

Cleaning Duplicates

In [62]:
```
# generating rows where movies are duplicates in terms of title
merged_df.loc[merged_df.duplicated(subset=['title'])]
```

Out[62]:

	release_date	release_year	title	genre	prod_method	creative_type	budget
100	2011-07-15	2011.00	Harry Potter and the Deathly Hallows:...	Adventure	Animation/Live Action	Fantasy	125000000
887	2018-11-21	2018.00	Robin Hood	Action	Live Action	Historical Fiction	99000000
1076	2020-10-02	2020.00	The Call	Horror	Live Action	Contemporary Fiction	0
1858	NaT	nan	Midnight in Paris	Romantic Comedy	Live Action	Fantasy	30000000
1866	NaT	nan	A Nightmare on Elm Street	Horror	Live Action	Fantasy	35000000
...	...	...	...	...	...	...	...
2432	NaT	nan	Robin Hood	Action	Live Action	Historical Fiction	99000000
2435	NaT	nan	Mary Poppins Returns	Musical	Live Action	Kids Fiction	130000000
2436	NaT	nan	The Nutcracker and the Four Realms	Adventure	Live Action	Fantasy	132900000
2437	NaT	nan	Aquaman	Action	Live Action	Super Hero	160000000
2438	NaT	nan	Ralph Breaks The Internet	Adventure	Digital Animation	Kids Fiction	175000000

185 rows × 12 columns

In [63]:
```
merged_df.loc[merged_df['title']=='Aquaman']
```

Out[63]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	dom
804	2018-12-21	2018.00	Aquaman	Action	Animation/Live Action	Super Hero	160000000	335
2437	NaT	nan	Aquaman	Action	Live Action	Super Hero	160000000	333

In [64]:
```
merged_df.loc[merged_df['title']=='Ralph Breaks The Internet']
```

Out[64]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	do
813	2018-11-21	2018.00	Ralph Breaks The Internet	Adventure	Digital Animation	Kids Fiction	175000000	20
2438	NaT	nan	Ralph Breaks The Internet	Adventure	Digital Animation	Kids Fiction	175000000	20

In [65]:
```
merged_df.loc[merged_df['title']=='Midnight in Paris']
```

Out[65]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	dom
157	2011-05-20	2011.00	Midnight in Paris	Romantic Comedy	Live Action	Fantasy	30000000	56
1858	NaT	nan	Midnight in Paris	Romantic Comedy	Live Action	Fantasy	30000000	56

After reviewing many of the duplicates, I was able to determine that if a movie was showing up as a duplicate in terms of title, it was truly a duplicate entry of the same movie. I wanted to give priority to the data that had come from numbers_df since it was more current (and had more non-null values) than the data that had come from opus_df. So I opted to keep the first instance of each duplicate and drop the rest.

In [66]:
```
# dropping duplicates
merged_df.drop_duplicates(subset=['title'], inplace=True)
```

In [67]:
```
# checking for null values
merged_df.isnull().sum()
```

Out[67]:
```
release_date        474
release_year        474
title                 0
genre                 1
prod_method           1
creative_type         3
budget                0
dom_gross             0
int_gross             0
world_gross         474
release_month       474
production_year     502
dtype: int64
```

Based on the amount of null values in the world_gross and release_month columns, I could see that there were 474 movies that were in my scraped Numbers dataset but not in my Opus dataset. And based on the amount of null vlaues in the production_year column, I could see that there were 501 movies that were in my Opus dataset but not my Numbers dataset.

In [68]:
```
# creating a list of titles that did not have an associated release_date
value
dateless_titles = merged_df['title'].loc[merged_df['release_date'].isnul
l()]
```

I created a list of the movie titles that originated from the Opus dataset which did not contain release date information. I then ran a function to query The Movies Database (TMDb) via API for each specific title and pull the release date for the title.

In [69]:
```
import json
from datetime import datetime
```

In [70]:
```
def get_keys(path):
    """Retrieves API key from files as api_key."""
    with open(path) as f:
        return json.load(f)
keys = get_keys("/Users/dtunnicliffe/.secret/TMDb_api.json")
api_key = keys['api_key']
```

```
In [71]: def get_date(title):
             """
             Updates release date information for movie in dataframe.

             Queries TMDB for a given movie title.
             Retrieves release date information for title.
             Adds new release date value to movie's row in dataframe.

             Parameters:
             title (str): user input movie title.

             Returns:
             df (Pandas DataFrame): A dataframe cleaned and adjusted as per crite
         ria listed above.

             """
             title_r = title.replace(' ', '+')
             url = f"https://api.themoviedb.org/3/search/movie?api_key={api_key}&
         query={title_r}"
             response = requests.get(url)
             if len(response.json()['results']) > 0:
                 rdate = (response.json()['results'][0]['release_date'])
                 if rdate:
                     x = datetime.strptime(rdate, '%Y-%m-%d').strftime('%b %d, %
         Y')
                     merged_df.loc[merged_df['title']==title, 'release_date'] = x
             else:
                 pass
```

```
In [72]: # getting release dates for list of titles lacking release dates
         for title in dateless_titles:
             get_date(title)
```

```
In [73]: # checking for null values
         merged_df.isnull().sum()
```

```
Out[73]: release_date         6
         release_year       474
         title                0
         genre                1
         prod_method          1
         creative_type        3
         budget               0
         dom_gross            0
         int_gross            0
         world_gross        474
         release_month      474
         production_year    502
         dtype: int64
```

My get_date function successfully took care of almost all the null values for release_date! I had a look at the few null values that remained.

In [74]:
```python
# show rows that contain null values for release date
merged_df[merged_df['release_date'].isnull()]
```

Out[74]:

	release_date	release_year	title	genre	prod_method	creative_type	bu
1962	NaT	nan	Jin líng shí san chai	Drama	Live Action	Historical Fiction	10000
2160	NaT	nan	San cheng ji	Drama	Live Action	Dramatization	1200
2202	NaT	nan	Savva. Serdtse voyna	Adventure	Digital Animation	Kids Fiction	3000
2282	NaT	nan	Baahubali 2: The Conclusion	Action	Live Action	Historical Fiction	3000
2345	NaT	nan	Jìyì dàshī	Thriller/Suspense	Live Action	Contemporary Fiction	2000
2350	NaT	nan	Chāi dàn zhuānjiā	Action	Live Action	Contemporary Fiction	2300

Based on the fact that all these movies had many columns of null values, I opted to drop them from the data.

In [75]:
```python
# dropping rows that have null release dates
merged_df = merged_df.dropna(axis=0, subset=['release_date'])
```

In [76]:
```python
# checking for null values
merged_df.isnull().sum()
```

Out[76]:
```
release_date         0
release_year       468
title                0
genre                1
prod_method          1
creative_type        3
budget               0
dom_gross            0
int_gross            0
world_gross        468
release_month      468
production_year    502
dtype: int64
```

Next, I dealt with movies that did not have a world_gross value.

In [77]:
```
# creating a list of titles that did not have a world_gross value
worldless_titles = merged_df['title'].loc[merged_df['world_gross'].isnul
l()]
worldless_titles
```

Out[77]:
```
1785                    You Got Served: Beat The World
1786                                     A Better Life
1787                                     Cedar Rapids
1788                             Blood Done Sign My Name
1789                                        MacGruber
                             ...
2428                               Dr. Seuss' The Grinch
2433                                      Mortal Engines
2434     How to Train Your Dragon: The Hidden World
2439                         Mission: Impossible—Fallout
2440    Fantastic Beasts: The Crimes of Grindelwald
Name: title, Length: 468, dtype: object
```

To fill in this data, I ran a function to add the domestic gross and international gross column values together for each row to equal the world gross.

In [78]:
```
def worldsum(title):
    """Gets sum of dom_gross and int_gross values for movie title, sets
 as world_gross value"""
    dg = merged_df.loc[merged_df['title'] == title, 'dom_gross']
    ig = merged_df.loc[merged_df['title'] == title, 'int_gross']
    merged_df.loc[merged_df['title'] == title, 'world_gross'] = dg + ig
```

In [79]:
```
# generating world_gross values for all titles lacking world_gross
for title in worldless_titles:
    worldsum(title)
```

In [80]:
```
# checking for null values
merged_df.isnull().sum()
```

Out[80]:
```
release_date         0
release_year       468
title                0
genre                1
prod_method          1
creative_type        3
budget               0
dom_gross            0
int_gross            0
world_gross          0
release_month      468
production_year    502
dtype: int64
```

I no longer needed the production_year column at this point, so I went ahead and dropped that from the DataFrame.

In [81]:
```python
# dropping production_year column
merged_df.drop('production_year', axis=1, inplace=True)
```

I once again made use of the datetime methods I had used previously for numbers_df prior to the merge, in order to fill in the null values for 'month' in merged_df.

In [82]:
```python
# converting release_date column to datetime object
merged_df['release_date'] = pd.to_datetime(merged_df['release_date'])
# generating release_month value based on datetime release_date
merged_df['release_month'] = merged_df['release_date'].dt.strftime('%B')
```

In [83]:
```python
# checking for null values
merged_df.isnull().sum()
```

Out[83]:
```
release_date       0
release_year     468
title              0
genre              1
prod_method        1
creative_type      3
budget             0
dom_gross          0
int_gross          0
world_gross        0
release_month      0
dtype: int64
```

Finally, I had to set the release_year for the titles whose release_dates had been scraped after the merge.

In [84]:
```python
# setting release_year based on datetime release_date values
merged_df['release_year'] = merged_df['release_date'].dt.year
```

In [85]:
```python
# checking for null values
merged_df.isnull().sum()
```

Out[85]:
```
release_date     0
release_year     0
title            0
genre            1
prod_method      1
creative_type    3
budget           0
dom_gross        0
int_gross        0
world_gross      0
release_month    0
dtype: int64
```

In [86]:
```python
# getting counts of each value in release_year column
merged_df.release_year.value_counts()
```

Out[86]:
```
2011    162
2016    158
2013    155
2015    152
2012    148
2014    146
2010    142
2017    133
2018    119
2019    109
2020    102
2009      8
2005      3
2007      2
1968      1
1996      1
1969      1
1982      1
1985      1
1988      1
1994      1
2006      1
1997      1
1999      1
2000      1
2001      1
2004      1
2008      1
1967      1
Name: release_year, dtype: int64
```

When I checked the value counts for the release_year column, I found 31 values where the release year was not within my parameters.

In [87]:

```
merged_df.loc[merged_df['release_year']<2010]
```

Out[87]:

	release_date	release_year	title	genre	prod_method	creative_type	bu
1795	2009-09-05	2009	Io sono l'amore	Drama	Live Action	Historical Fiction	1000
1805	2009-03-25	2009	Chloe	Thriller/Suspense	Live Action	Contemporary Fiction	1300
1808	2009-01-18	2009	I Love You, Phillip Morris	Comedy	Live Action	Contemporary Fiction	1300
1820	2009-06-17	2009	Enter the Void	Drama	Live Action	Fantasy	1600
1824	2009-09-11	2009	Youth in Revolt	Comedy	Live Action	Contemporary Fiction	1800
1826	2009-09-04	2009	The Last Station	Drama	Live Action	Dramatization	1800
1833	2009-05-17	2009	Middle Men	Comedy	Live Action	Historical Fiction	2000
1840	2001-11-16	2001	Stone	Drama	Live Action	Contemporary Fiction	2200
1853	2009-08-13	2009	Case 39	Horror	Live Action	Fantasy	2700
1862	2004-08-19	2004	Going the Distance	Romantic Comedy	Live Action	Contemporary Fiction	3200
1880	2000-05-26	2000	Shanghai	Drama	Live Action	Historical Fiction	5000
1902	1969-04-01	1969	Les Intouchables	Comedy	Live Action	Contemporary Fiction	1080
1946	1982-06-25	1982	The Thing	Horror	Live Action	Fantasy	3800
1948	2006-07-28	2006	The Impossible	Drama	Live Action	Dramatization	4000
1996	1988-06-09	1988	Dangerous Liaisons	Drama	Live Action	Historical Fiction	2420
2053	1994-09-23	1994	Redemption	Thriller/Suspense	Live Action	Contemporary Fiction	2300
2095	2007-01-05	2007	Freedom	Drama	Live Action	Historical Fiction	1450
2120	1967-11-27	1967	Viy	Adventure	Live Action	Fantasy	2600
2129	2005-01-01	2005	Aloha	Drama	Live Action	Contemporary Fiction	3700
2161	2005-08-26	2005	Brothers	Action	Live Action	Contemporary Fiction	1300
2166	1997-05-12	1997	Robinson Crusoe	Adventure	Digital Animation	Kids Fiction	1300
2195	1985-07-19	1985	Legend	Thriller/Suspense	Live Action	Dramatization	2500
2210	2008-12-12	2008	Yip Man 3	Action	Live Action	Historical Fiction	3600
2253	1968-01-01	1968	Sultan	Action	Live Action	Contemporary Fiction	1100

	release_date	release_year	title	genre	prod_method	creative_type	bu
2296	2007-03-16	2007	Silence	Drama	Live Action	Historical Fiction	4650
2353	1996-08-02	1996	Matilda	Drama	Live Action	Dramatization	2500
2414	2005-09-03	2005	Serenity	Thriller/Suspense	Live Action	Contemporary Fiction	2500
2428	1999-10-05	1999	Dr. Seuss' The Grinch	Adventure	Digital Animation	Kids Fiction	7500

All of these were movies with release dates prior to 2010 that had been added from my get_date function after my earlier filtering. I went ahead and dropped those as well.

In [88]:
```
# show number of rows and columns before filtering
merged_df.shape
```

Out[88]:
```
(1554, 11)
```

In [89]:
```
# filter DataFrame to only include titles with release_year >= 2010
merged_df = merged_df[merged_df['release_year']>=2010.00]
```

In [90]:
```
# show number of rows and columns after filtering
merged_df.shape
```

Out[90]:
```
(1526, 11)
```

In [91]:
```
merged_df.isnull().sum()
```

Out[91]:
```
release_date      0
release_year      0
title             0
genre             1
prod_method       1
creative_type     3
budget            0
dom_gross         0
int_gross         0
world_gross       0
release_month     0
dtype: int64
```

Looking good! I had exactly 5 null values left in my whole DataFrame.

In [92]:
```
# show columns where value for genre is null
merged_df.loc[merged_df['genre'].isnull()]
```

Out[92]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	dom_gro
1976	2012-04-25	2012	Le prénom	NaN	Live Action	Contemporary Fiction	11000000	616

When I searched this entry on IMDb, there was no useful information on this title, so I dropped this as well.

In [93]:
```
# dropping row with null value for genre
merged_df = merged_df.dropna(axis=0, subset=['genre'])
```

In [94]:
```
# show rows where creative type value is null
merged_df.loc[merged_df['creative_type'].isnull()]
```

Out[94]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	do
1789	2010-05-21	2010	MacGruber	Comedy	Live Action	NaN	10000000	
1807	2010-02-19	2010	The Killer Inside Me	Drama	Live Action	NaN	13000000	
2054	2013-12-18	2013	Dhoom 3	Action	NaN	NaN	24000000	

In [95]:
```
# getting counts for each value in creative_type column
merged_df['creative_type'].value_counts()
```

Out[95]:
```
Contemporary Fiction        690
Dramatization               164
Science Fiction             160
Kids Fiction                154
Historical Fiction          142
Fantasy                     136
Super Hero                   53
Factual                      18
Multiple Creative Types       5
Name: creative_type, dtype: int64
```

In [96]:
```
# filling null values for creative_type with the most common value for t
his column
merged_df['creative_type'].fillna('Contemporary Fiction', inplace=True)
```

In [97]:
```
# checking for null values
merged_df.isnull().sum()
```

Out[97]:
```
release_date     0
release_year     0
title            0
genre            0
prod_method      1
creative_type    0
budget           0
dom_gross        0
int_gross        0
world_gross      0
release_month    0
dtype: int64
```

In [98]:
```
# show rows where value for prod_method is null
merged_df.loc[merged_df['prod_method'].isnull()]
```

Out[98]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	dom_gro
2054	2013-12-18	2013	Dhoom 3	Action	NaN	Contemporary Fiction	24000000	80319

In [99]:
```
# getting value counts for prod_method column
merged_df['prod_method'].value_counts()
```

Out[99]:
```
Live Action                    1283
Digital Animation               124
Animation/Live Action           103
Stop-Motion Animation             8
Hand Animation                    4
Multiple Production Methods       2
Name: prod_method, dtype: int64
```

In [100]:
```
# fill null values for prod_method with most common value for column
merged_df['prod_method'].fillna('Live Action', inplace=True)
```

In [101]:
```
# saving copy of DataFrame as csv
merged_df.to_csv('./data/merged_df.csv')
```

Data Analysis

In order to answer the questions I had posed at the onset of this project, I performed exploratory analyses on the basis of genre, creative type, production method, time of year, and budget. For each of these themes, I utilized statistical methods followed by plotting of visualizations to determine the relationship between each of these key elements and movie revenue (world gross). While I had the data for domestic and international gross for most movies as well, I was most focused on world gross because I wanted the total amount of money that Microsoft could expect to make on a given production.

One thing I realized as soon as I began my analysis was that my dollar amounts were not very plot-friendly. I had to reassess my gross and budget columns at this point and add new columns for budget in millions and world gross in millions.

In [102]:
```
merged_df['budget_in_mil'] = merged_df['budget'] / 1000000
```

In [103]:
```
merged_df['world_gross_mil'] = merged_df['world_gross']/1000000
```

In [104]:
```
merged_df['world_gross_mil'].describe()
```

Out[104]:
```
count    1525.00
mean      180.81
std       265.90
min         0.00
25%        32.40
50%        82.18
75%       211.74
max      2797.80
Name: world_gross_mil, dtype: float64
```

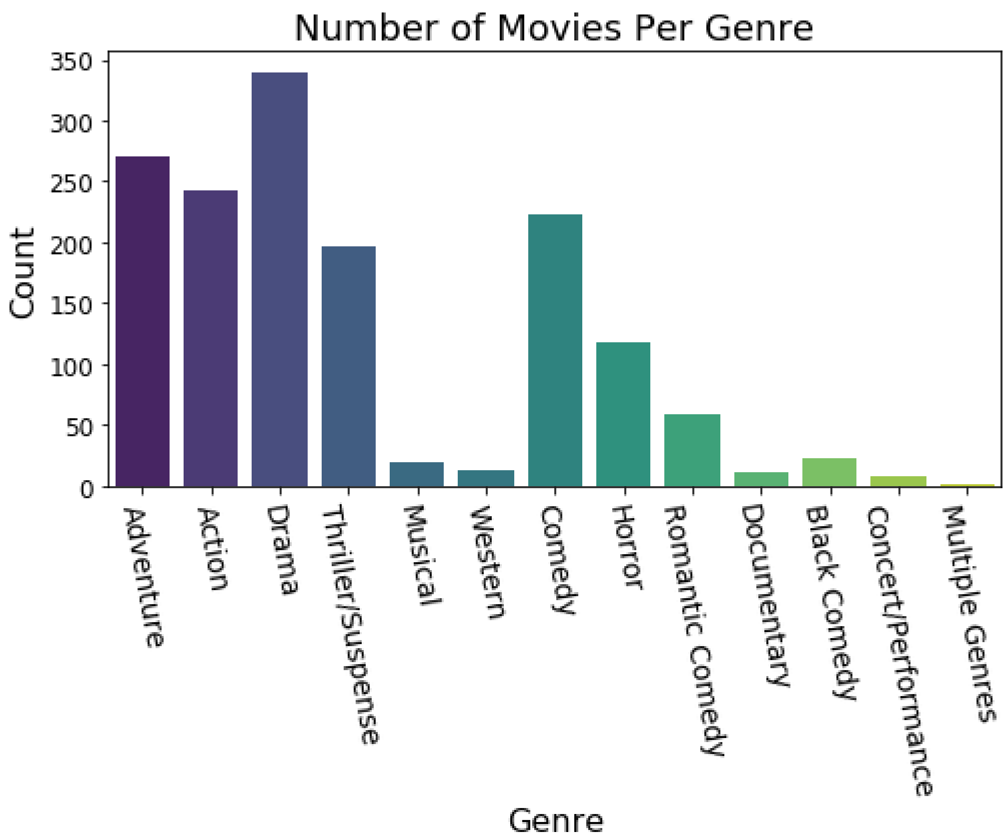
Genre/Creative Type/Production Method: What types of movies are currently most successful?

Genre

In [105]:
```
# getting value counts for genre column
merged_df['genre'].value_counts()
```

Out[105]:
```
Drama                  340
Adventure              270
Action                 243
Comedy                 223
Thriller/Suspense      197
Horror                 118
Romantic Comedy         59
Black Comedy            23
Musical                 19
Western                 12
Documentary             11
Concert/Performance      8
Multiple Genres          2
Name: genre, dtype: int64
```

```
In [106]: # plotting the number of movies per genre in dataset
plt.figure(figsize=(8,4))
sns.countplot(x='genre', data=merged_df, palette='viridis')
plt.title('Number of Movies Per Genre', fontsize=18)
plt.ylabel('Count', fontsize=16)
plt.xlabel('Genre', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12)
plt.xticks(rotation=-80);
#saved in images as fig1
#plt.tight_layout()
#plt.savefig('./images/fig1.png')
```



The Drama genre was most common in this dataset, followed by Adventure and Action.

In [107]:
```
# getting mean and median world gross amounts by genre
genre_stats = merged_df.groupby('genre')['world_gross_mil'].agg(['media
n', 'mean'])
genre_stats.sort_values(by='mean', ascending=False)
```

Out[107]:

genre	median	mean
Musical	392.88	403.18
Adventure	227.53	345.30
Action	181.98	332.11
Western	75.52	124.18
Thriller/Suspense	60.28	114.65
Horror	82.12	105.72
Black Comedy	61.79	97.35
Comedy	70.55	93.16
Romantic Comedy	61.62	91.49
Drama	43.33	84.27
Concert/Performance	27.96	33.47
Documentary	10.53	23.66
Multiple Genres	1.72	1.72

In [108]:
```
# getting Q1 (25th percentile) for world gross of each genre
a = merged_df.groupby('genre')['world_gross_mil'].quantile(0.25)
```

In [109]:
```
# getting Q3 (75th percentile) for world gross of each genre
b = merged_df.groupby('genre')['world_gross_mil'].quantile(0.75)
```

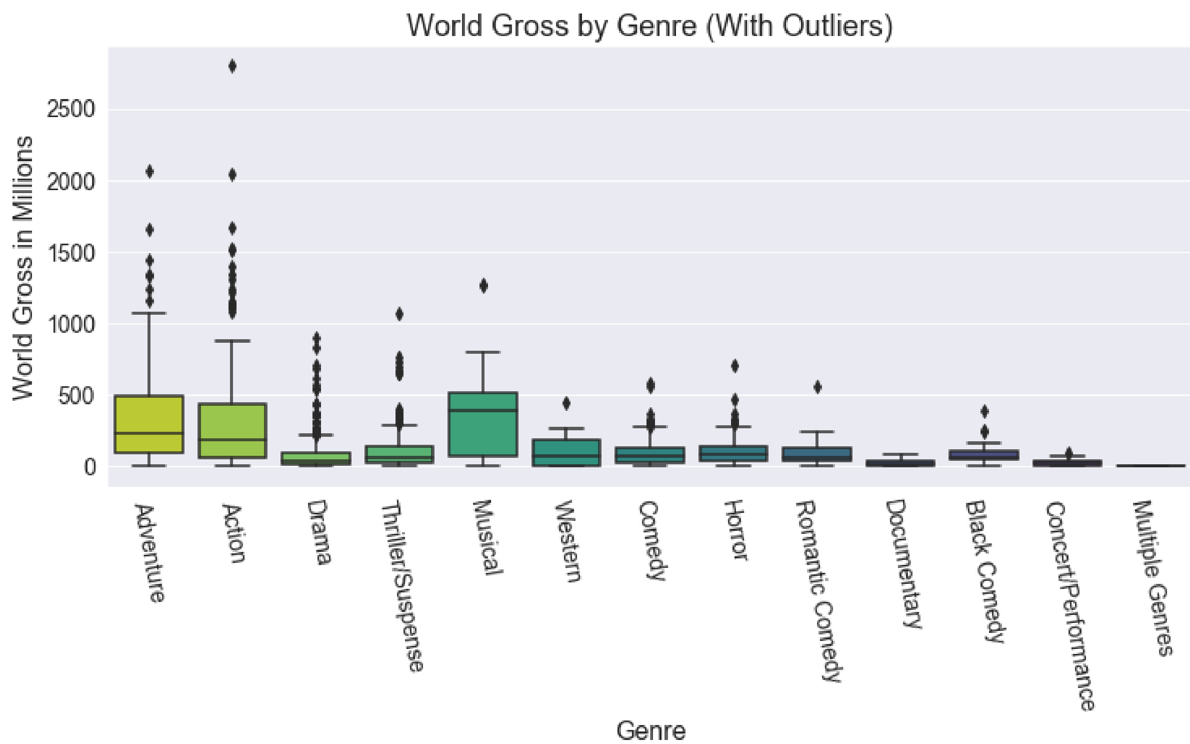
In [110]:
```python
# getting interquartile range (IQR) for world gross of each genre
iqr = b - a
iqr
```

Out[110]:
```
genre
Action                  372.28
Adventure               390.04
Black Comedy             54.42
Comedy                   97.13
Concert/Performance      34.49
Documentary              34.69
Drama                    84.75
Horror                   96.79
Multiple Genres           1.58
Musical                 445.58
Romantic Comedy          93.10
Thriller/Suspense       104.24
Western                 175.06
Name: world_gross_mil, dtype: float64
```

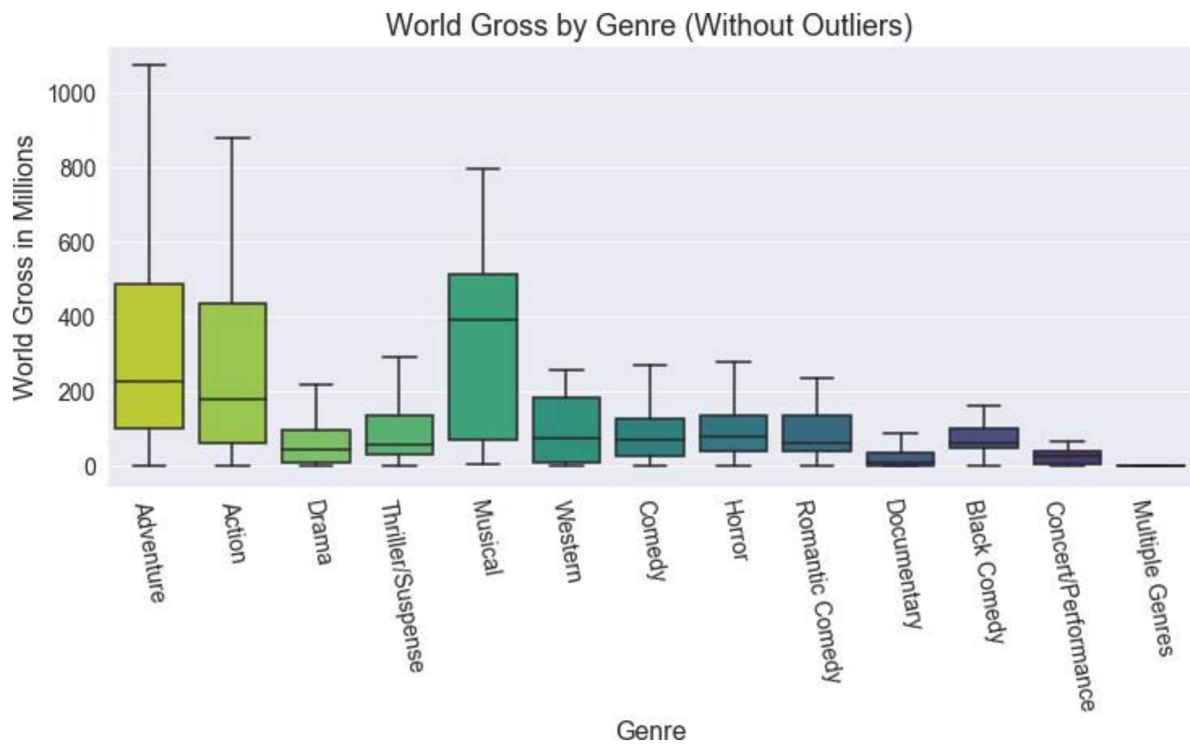
For my visualizations, I found it helpful to show my plots both with and without outliers. While outliers paint the whole picture of the data being analyzed, it can be helpful to see the data more closely and "zoom in" (both literally and figuratively) on the trends without outliers.

In [111]:

```
# generating box plot of world gross statistics per genre
plt.figure(figsize=(12,5))
sns.set_style('darkgrid')
sns.boxplot(x='genre', y='world_gross_mil', data=merged_df, palette='vir
idis_r')
plt.xticks(rotation=-80)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xlabel('Genre', fontsize = 16)
plt.title('World Gross by Genre (With Outliers)', fontsize = 18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14);
#saved in images as fig2
#plt.subplots_adjust(bottom=0.2)
#plt.savefig('./images/fig2.png')
```



In [112]:
```
# generating box plot of world gross statistics per genre
plt.figure(figsize=(12,5))
sns.set_style('darkgrid')
sns.boxplot(x='genre', y='world_gross_mil', data=merged_df, showfliers=False, palette='viridis_r')
plt.xticks(rotation=-80)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xlabel('Genre', fontsize = 16)
plt.title('World Gross by Genre (Without Outliers)', fontsize = 18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14);
#saved in images as fig3
#plt.subplots_adjust(bottom=0.2)
#plt.savefig('./images/fig3.png')
```

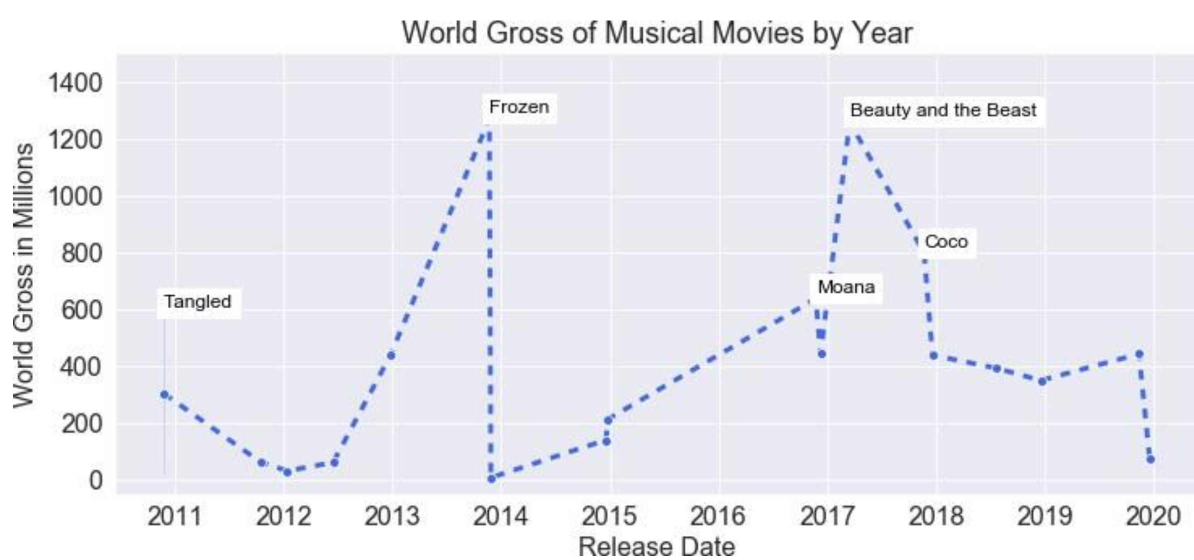


Based on mean and median, Musicals appeared to be the most lucrative genre for the time period I had explored. However, as can be seen from the box plot above, this was also the genre with the largest interquartile range (IQR), meaning that the middle values of world gross for these movies were the most spread out. After Musicals, Action and Adventure were the highest-grossing categories, with both high means and medians. These categories did have high IQR's as well, with a large variance in world gross values. Both Action and Adventure also had many high-grossing outliers, as can be seen in the first box plot above.

In [113]:
```
# creating a copy of merged_df sorted in order of release date
time_df = merged_df.sort_values(by='release_date')
```

In [179]:

```
# plotting the relationship between musical movies, world gross, and release month
sns.set_style('darkgrid')
plt.figure(figsize=(12,5))
sns.lineplot(data=time_df.loc[time_df['genre']=='Musical'], x="release_date", y="world_gross_mil", markers='o', style=True, dashes=[(2,2)], linewidth=3, color='royalblue', legend=False)
plt.title('World Gross of Musical Movies by Year', fontsize=18)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.xlabel('Release Date', fontsize=16)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.ylim(-50, 1500)
for w, x, y, z in zip(time_df['genre'], time_df['release_date'], time_df['world_gross_mil'], time_df['title']):
    if (w == 'Musical') & (y>500):
        plt.text(x = x, y = y+20, s = z, fontsize=12, color='black').set_backgroundcolor('white');
#saved in images as fig4
#plt.tight_layout()
#plt.savefig('./images/fig4.png')
```



For musical movies in the past ten years, there were five titles that were much more successful than any others. There was also an obvious dip in the year 2020, thanks to the global COVID-19 pandemic.

In [115]:

```python
# creating subset of DataFrame where genre is Musical
musicals = merged_df.loc[merged_df['genre']=='Musical']
musicals.sort_values(by='world_gross_mil', ascending=False).head()
```

Out[115]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	dom_g
302	2013-11-22	2013	Frozen	Musical	Digital Animation	Kids Fiction	150000000	40073
701	2017-03-17	2017	Beauty and the Beast	Musical	Animation/Live Action	Fantasy	160000000	50401
712	2017-11-22	2017	Coco	Musical	Digital Animation	Kids Fiction	175000000	21032
610	2016-11-23	2016	Moana	Musical	Digital Animation	Kids Fiction	150000000	24875
9	2010-11-24	2010	Tangled	Musical	Digital Animation	Kids Fiction	260000000	20082

I wanted to see which other attributes the most profitable movies in the Musical genre had shared. What I found was that, for the top five highest-grossing Musical movies in this time period, 4 out of 5 also happened to be: Digitally Animated, Kids Fiction, and released in November!
Based on my analysis of genre, my recommendation to Microsoft was to consider making a musical movie.

Creative Type

In [116]:

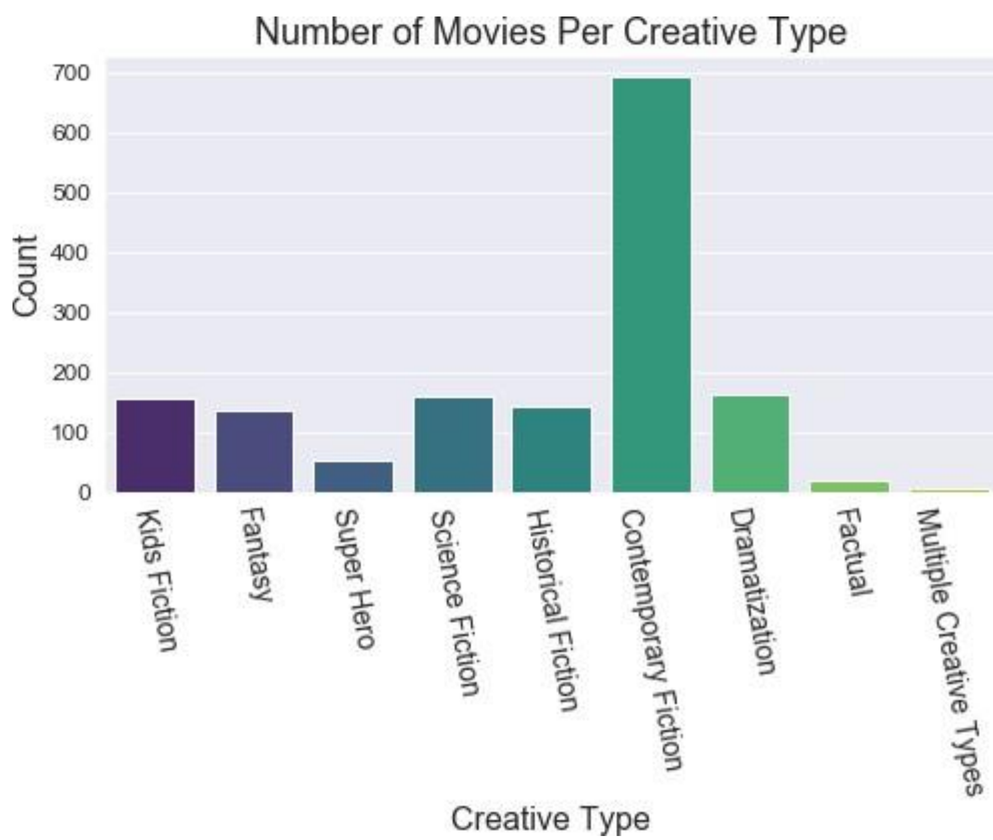
```python
# getting value counts for creative type column
merged_df['creative_type'].value_counts()
```

Out[116]:

```
Contemporary Fiction        693
Dramatization               164
Science Fiction             160
Kids Fiction                154
Historical Fiction          142
Fantasy                     136
Super Hero                   53
Factual                      18
Multiple Creative Types       5
Name: creative_type, dtype: int64
```

In [117]:

```
# plotting the number of movies per creative type in dataset
plt.figure(figsize=(8,4))
sns.countplot(x='creative_type', data=merged_df, palette='viridis')
plt.title('Number of Movies Per Creative Type', fontsize=18)
plt.ylabel('Count', fontsize=16)
plt.xlabel('Creative Type', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12)
plt.xticks(rotation=-80);
#saved in images as fig5
#plt.tight_layout()
#plt.savefig('./images/fig5.png')
```



Contemporary Fiction was the most common creative type in this dataset by far.

```
In [118]: # getting mean and median world gross amounts by creative type
genre_stats = merged_df.groupby('creative_type')['world_gross_mil'].agg
(['median', 'mean'])
genre_stats.sort_values(by='mean', ascending=False)
```

Out[118]:

creative_type	median	mean
Super Hero	668.00	714.09
Kids Fiction	197.54	310.47
Fantasy	156.96	288.07
Science Fiction	171.24	279.50
Contemporary Fiction	62.60	109.09
Historical Fiction	44.55	99.84
Dramatization	53.55	93.46
Factual	32.51	47.64
Multiple Creative Types	31.16	45.12

Super Hero movies did substantially better at the box office than all other creative types, with Kids Fiction coming in second place.

```
In [119]: # getting Q1 (25th percentile) for world gross of each genre
a = merged_df.groupby('creative_type')['world_gross_mil'].quantile(0.25)
```

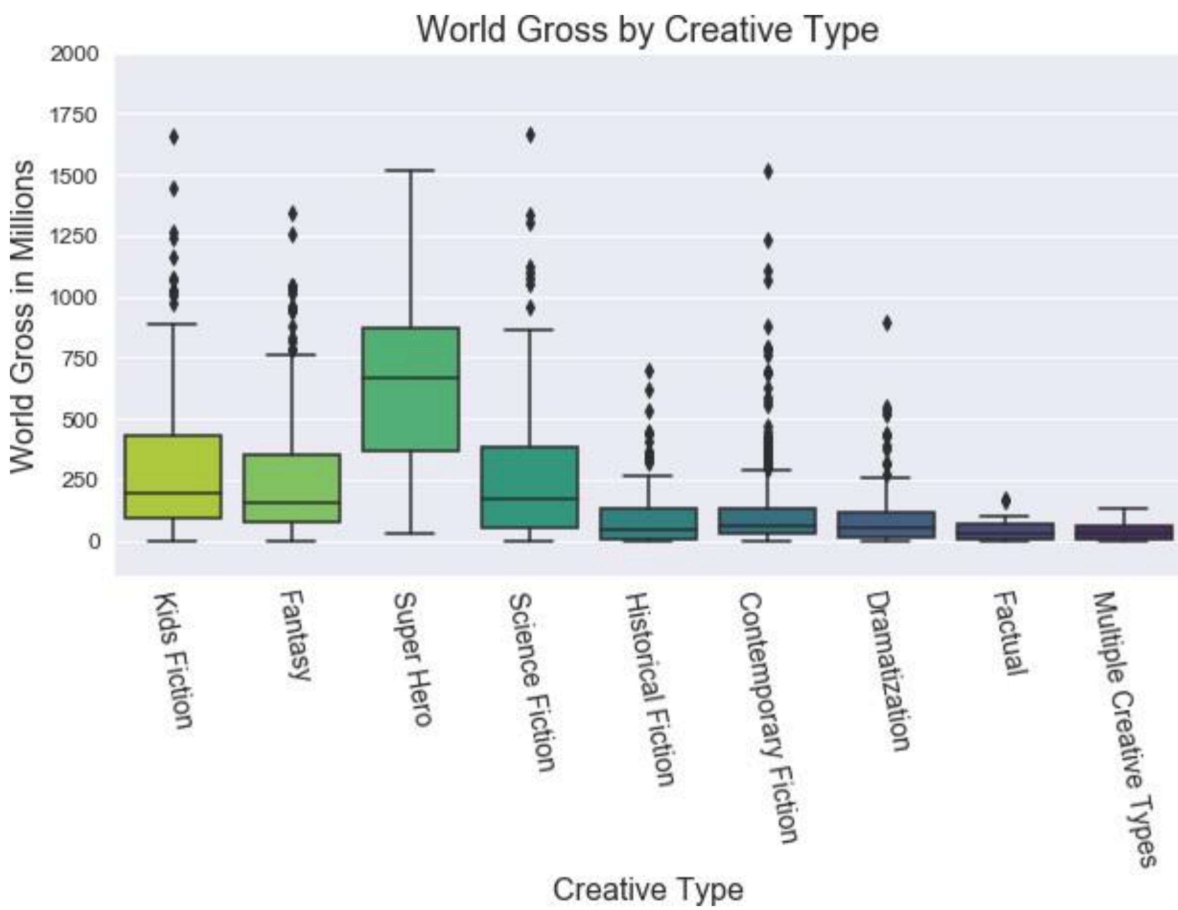
```
In [120]: # getting Q3 (75th percentile) for world gross of each genre
b = merged_df.groupby('creative_type')['world_gross_mil'].quantile(0.75)
```

```
In [121]: # getting interquartile range (IQR) for world gross of each genre
iqr = b - a
iqr
```

```
Out[121]: creative_type
Contemporary Fiction       105.01
Dramatization               99.43
Factual                     59.02
Fantasy                    279.87
Historical Fiction         124.20
Kids Fiction               344.91
Multiple Creative Types     56.62
Science Fiction            329.81
Super Hero                 498.54
Name: world_gross_mil, dtype: float64
```

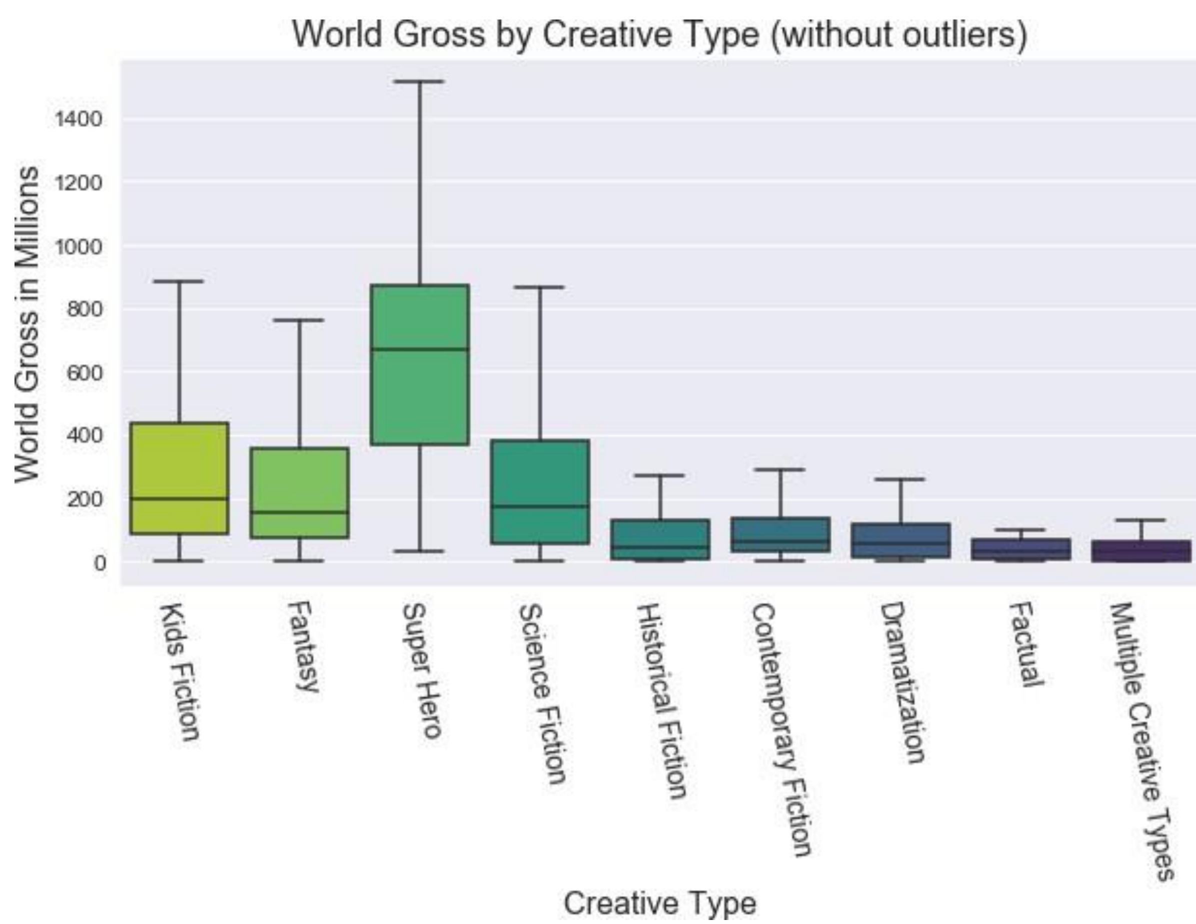
In [122]:

```
# generating box plot of world gross statistics per creative type
plt.figure(figsize=(10,5))
sns.set_style('darkgrid')
sns.boxplot(x='creative_type', y='world_gross_mil', data=merged_df, pale
tte='viridis_r')
plt.xticks(rotation=-80)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xlabel('Creative Type', fontsize = 16)
plt.title('World Gross by Creative Type', fontsize = 18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12)
plt.ylim(None, 2000);
#saved in images as fig6
#plt.subplots_adjust(bottom=0.2)
#plt.savefig('./images/fig6.png')
```



In [123]:

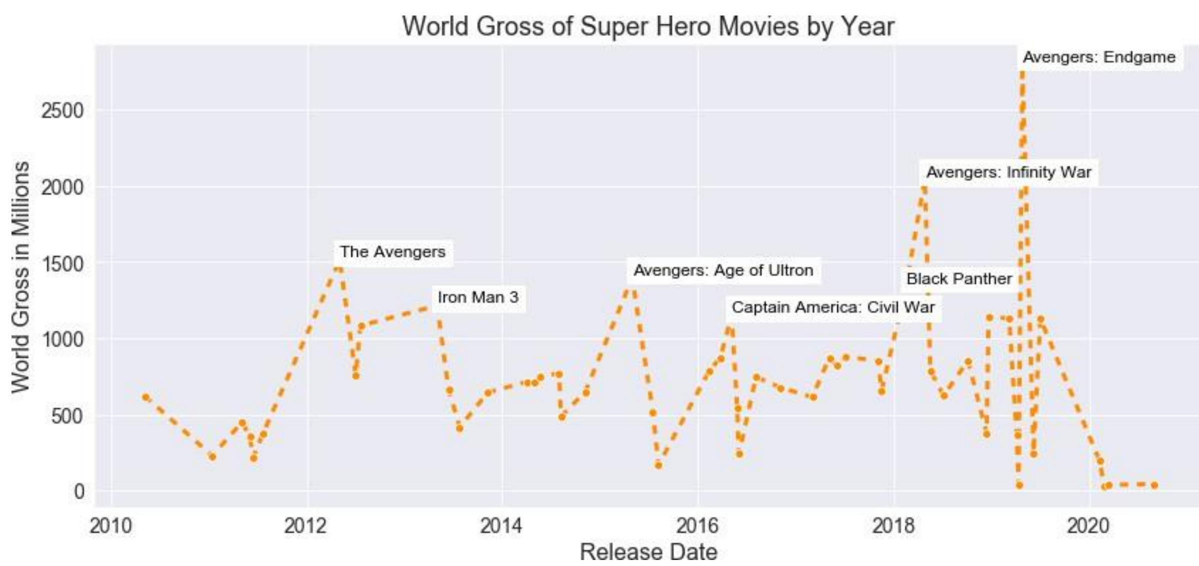
```
# generating box plot of world gross statistics per creative type
plt.figure(figsize=(10,5))
sns.set_style('darkgrid')
sns.boxplot(x='creative_type', y='world_gross_mil', data=merged_df, show
fliers=False, palette='viridis_r')
plt.xticks(rotation=-80)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xlabel('Creative Type', fontsize = 16)
plt.title('World Gross by Creative Type (without outliers)', fontsize =
18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12);
#saved in images as fig7
#plt.subplots_adjust(bottom=0.2)
#plt.savefig('./images/fig7.png')
```



Based on mean and median, Super Hero movies were far above all the other creative types. Kids Fiction was in second place, with many high-grossing outliers (mentioned previously). Science Fiction and Fantasy had relatively high means and medians as well, and both creative types also contained many high-grossing outliers.

In [124]:

```
# plotting relationship between world gross and release month for super
 hero movies
sns.set_style('darkgrid')
plt.figure(figsize=(14,6))
sns.lineplot(data=time_df.loc[time_df['creative_type']=='Super Hero'], x
="release_date", y="world_gross_mil", markers='o', style=True, dashes=[(
2,2)], linewidth=3, color='darkorange', legend=False)
plt.title('World Gross of Super Hero Movies by Year', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.xlabel('Release Date', fontsize=16)
plt.ylabel('World Gross in Millions', fontsize=16)
for w, x, y, z in zip(time_df['creative_type'], time_df['release_date'],
time_df['world_gross_mil'], time_df['title']):
    if (w == 'Super Hero') & (y>1150):
        plt.text(x = x, y = y+20, s = z, fontsize=12, color='black').set
_backgroundcolor('white');
#saved in images as fig8
#plt.tight_layout()
#plt.savefig('./images/fig8.png')
```



Super Hero movies seemed to do consistently well over the past ten years, although the line plot showed some ups and downs. Still, even the lows for Super Hero movies would be considered highs for other types of movies, so perspective is important. The plot showed seven titles that did extremely well for their movie type.

In [125]:
```
# creating subset of DataFrame where creative type is Super Hero
superhero = merged_df.loc[merged_df['creative_type']=='Super Hero']
superhero.sort_values(by='world_gross_mil', ascending=False).head(7)
```

Out[125]:

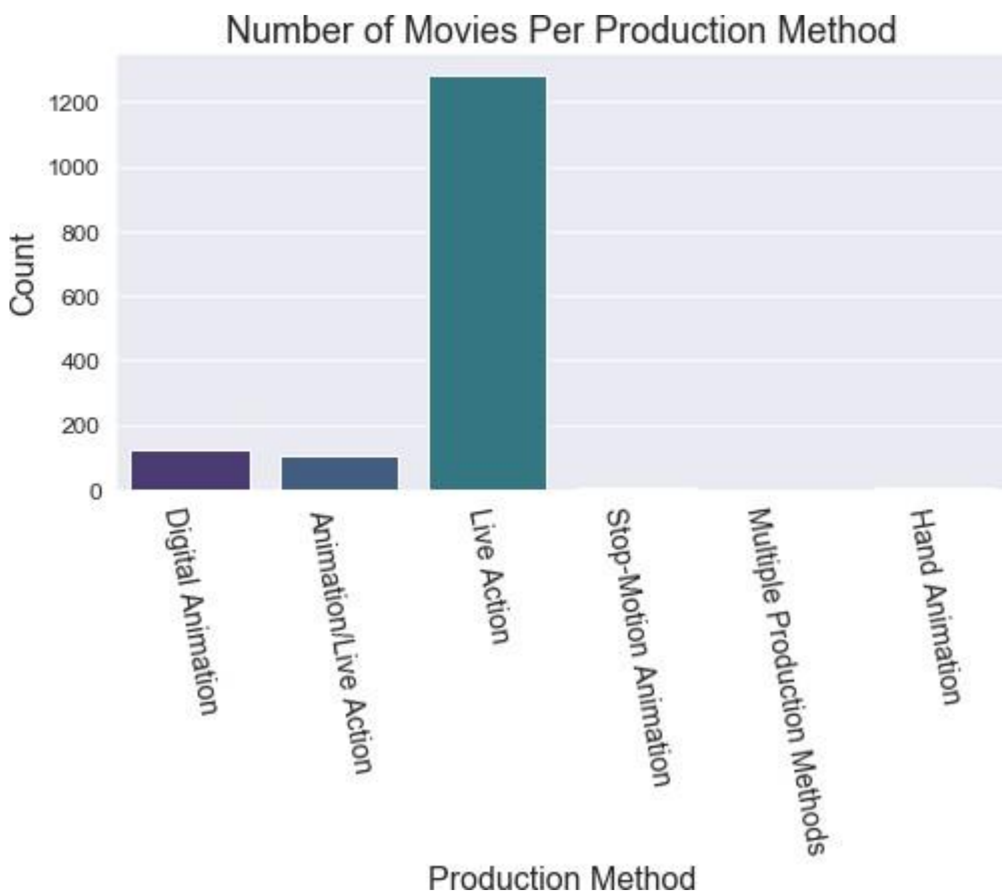
	release_date	release_year	title	genre	prod_method	creative_type	budget	dom_
900	2019-04-26	2019	Avengers: Endgame	Action	Animation/Live Action	Super Hero	400000000	8583
801	2018-04-27	2018	Avengers: Infinity War	Action	Animation/Live Action	Super Hero	300000000	6788
200	2012-05-04	2012	The Avengers	Action	Animation/Live Action	Super Hero	225000000	6233
502	2015-05-01	2015	Avengers: Age of Ultron	Action	Animation/Live Action	Super Hero	365000000	4590
800	2018-02-16	2018	Black Panther	Action	Live Action	Super Hero	200000000	7000
301	2013-05-03	2013	Iron Man 3	Action	Animation/Live Action	Super Hero	200000000	4089
602	2016-05-06	2016	Captain America: Civil War	Action	Live Action	Super Hero	250000000	4080

As I had anticipated, The Avengers movies dominated the Super Hero category. All of these movies were produced by Live Action in the Action genre, and 6 out of 7 were released in either April or May. Based on my analysis of creative type, my recommendation to Microsoft was to explore the idea of making a Super Hero movie.

Production Method

In [126]:

```
# plotting the number of movies per production method in dataset
plt.figure(figsize=(8,4))
sns.countplot(x='prod_method', data=merged_df, palette='viridis')
plt.title('Number of Movies Per Production Method', fontsize=18)
plt.ylabel('Count', fontsize=16)
plt.xlabel('Production Method', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12)
plt.xticks(rotation=-80);
#saved in images as fig9
#plt.subplots_adjust(bottom=0.2)
#plt.savefig('./images/fig9.png')
```



Live Action was the most common production method by far.

In [127]:

```
# getting mean and median world gross amounts by production method
prod_stats = merged_df.groupby('prod_method')['world_gross_mil'].agg(['median', 'mean'])
prod_stats.sort_values(by='mean', ascending=False)
```

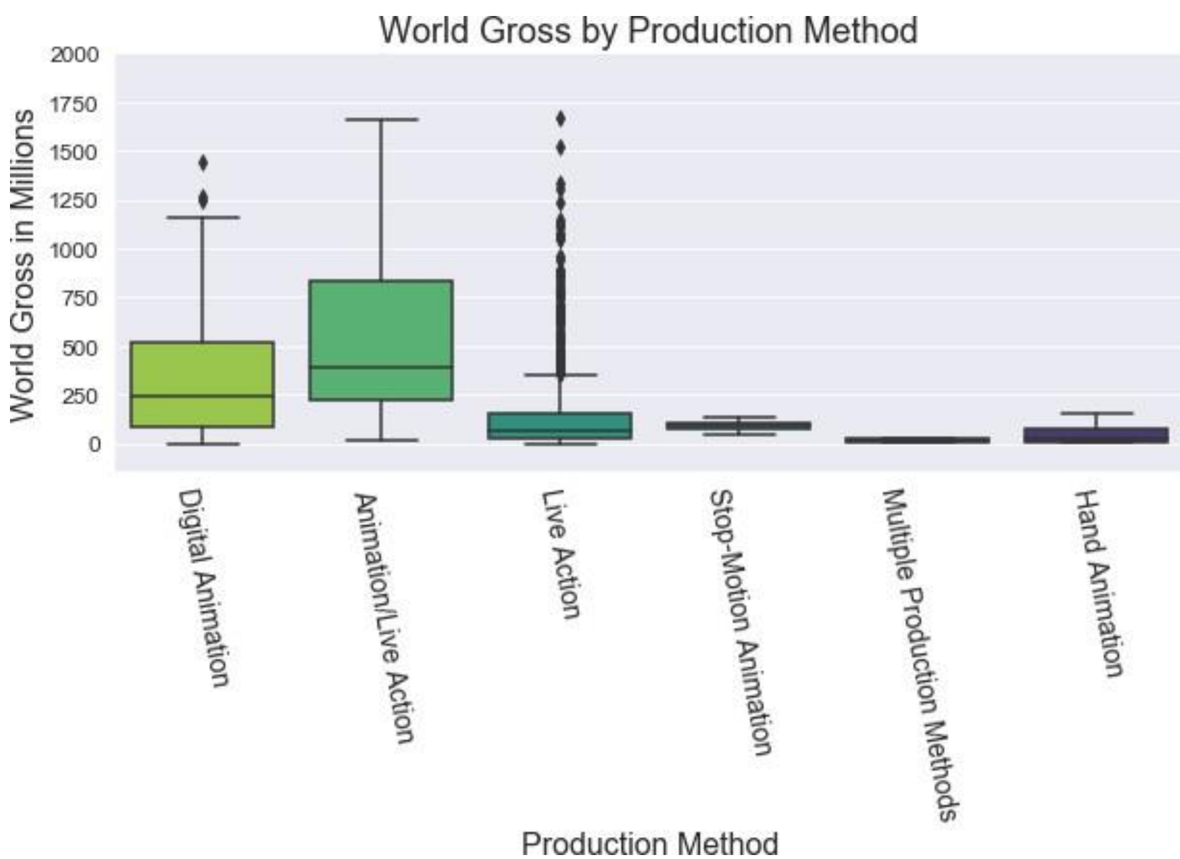
Out[127]:

prod_method	median	mean
Animation/Live Action	393.15	577.76
Digital Animation	247.91	345.38
Live Action	68.27	134.29
Stop-Motion Animation	91.54	90.09
Hand Animation	29.76	54.91
Multiple Production Methods	17.23	17.23

However, the Animation/Live Action category had the highest mean and median, with Digital Animation coming in second.

In [128]:

```
# generating box plot of world gross statistics per production method
plt.figure(figsize=(10,4))
sns.set_style('darkgrid')
sns.boxplot(x='prod_method', y='world_gross_mil', data=merged_df, palett
e='viridis_r')
plt.xticks(rotation=-80)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xlabel('Production Method', fontsize = 16)
plt.title('World Gross by Production Method', fontsize = 18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12)
plt.ylim(None, 2000);
#saved in images as fig10
#plt.subplots_adjust(bottom=0.2)
#plt.savefig('./images/fig10.png')
```



Based on mean and median, Animation/Live Action and Digital Animation appeared to be the most successful production methods for the time period I had explored.

In [129]:

```
# creating subset of DataFrame where prod_method is Animation/Live Actio
n or Digital Animation
anim_df = merged_df.loc[(merged_df['prod_method']=='Animation/Live Actio
n') | (merged_df['prod_method']=='Digital Animation')]
anim_df.sort_values(by='world_gross_mil', ascending=False).head(10)
```

Out[129]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	d
900	2019-04-26	2019	Avengers: Endgame	Action	Animation/Live Action	Super Hero	400000000	
500	2015-12-18	2015	Star Wars Ep. VII: The Force Awakens	Adventure	Animation/Live Action	Science Fiction	306000000	
801	2018-04-27	2018	Avengers: Infinity War	Action	Animation/Live Action	Super Hero	300000000	
901	2019-07-19	2019	The Lion King	Adventure	Animation/Live Action	Kids Fiction	260000000	
200	2012-05-04	2012	The Avengers	Action	Animation/Live Action	Super Hero	225000000	
903	2019-11-22	2019	Frozen II	Adventure	Digital Animation	Kids Fiction	150000000	
502	2015-05-01	2015	Avengers: Age of Ultron	Action	Animation/Live Action	Super Hero	365000000	
1964	2011-07-07	2011	Harry Potter and the Deathly Hallows: Part II	Adventure	Animation/Live Action	Fantasy	125000000	
700	2017-12-15	2017	Star Wars Ep. VIII: The Last Jedi	Adventure	Animation/Live Action	Science Fiction	200000000	
302	2013-11-22	2013	Frozen	Musical	Digital Animation	Kids Fiction	150000000	

I immediately noticed some overlap between this subset and the Musical and Super Hero subsets. Many of the top titles for this animation subset were either musicals or superhero movies as well. I also noted that while Frozen II is generally classified as a musical like the original Frozen, the data had it listed as an adventure movie. I wondered if there may be other children's animated movies in the data that were musical as well, but labeled with a different genre.

In [130]:
```
# creating subset of dataframe where production method is digital animat
ion
diganim_df = merged_df.loc[merged_df['prod_method']=='Digital Animation'
]
diganim_df.sort_values(by='world_gross_mil', ascending=False).head(10)
```

Out[130]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	d
903	2019-11-22	2019	Frozen II	Adventure	Digital Animation	Kids Fiction	150000000	4
302	2013-11-22	2013	Frozen	Musical	Digital Animation	Kids Fiction	150000000	4
802	2018-06-15	2018	Incredibles 2	Adventure	Digital Animation	Kids Fiction	200000000	6
505	2015-07-10	2015	Minions	Adventure	Digital Animation	Kids Fiction	74000000	3
904	2019-06-21	2019	Toy Story 4	Adventure	Digital Animation	Kids Fiction	200000000	4
0	2010-06-18	2010	Toy Story 3	Adventure	Digital Animation	Kids Fiction	200000000	4
708	2017-06-30	2017	Despicable Me 3	Adventure	Digital Animation	Kids Fiction	75000000	2
601	2016-06-17	2016	Finding Dory	Adventure	Digital Animation	Kids Fiction	200000000	4
606	2016-03-04	2016	Zootopia	Adventure	Digital Animation	Kids Fiction	150000000	3
303	2013-07-03	2013	Despicable Me 2	Adventure	Digital Animation	Kids Fiction	76000000	3

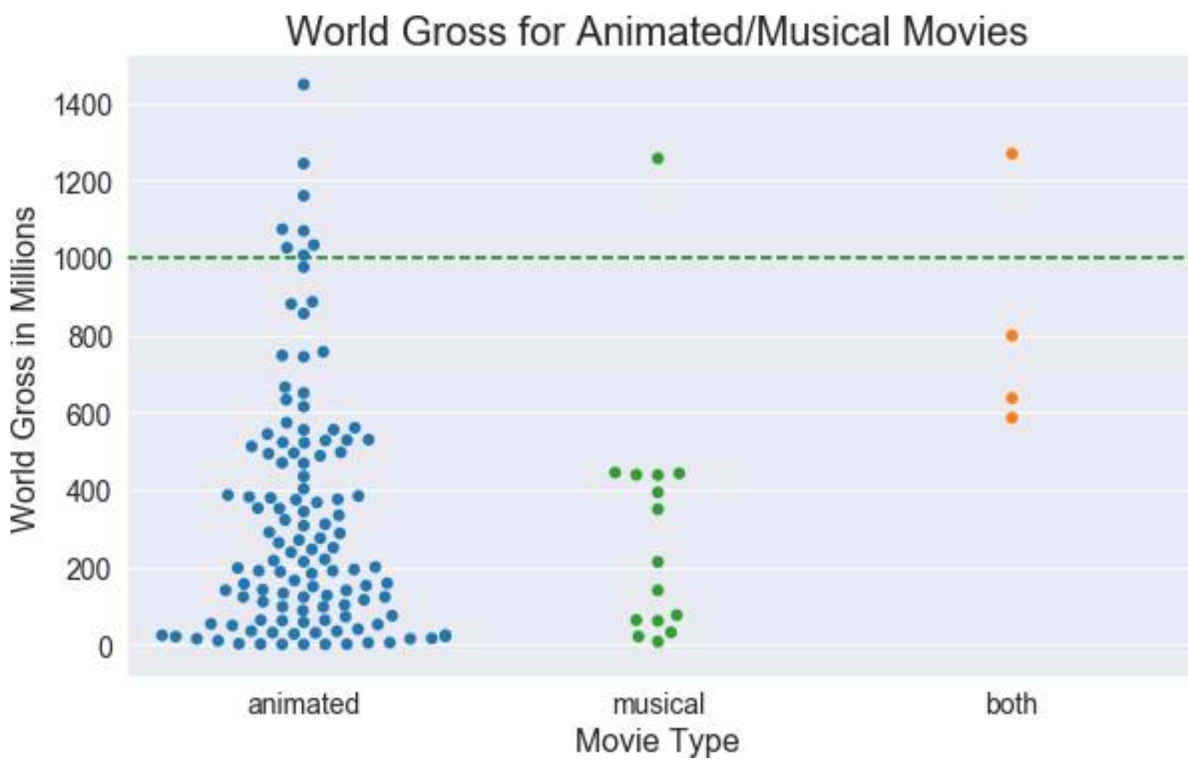
As I had suspected, many of the top films with a production method of digital animation also had musical components, but were labeled with a genre other than Musical. All of these were also Kids Fiction creative type, which was the second most profitable creative type as seen above.

In [131]:
```
# creating column to signify whether a title is animated, musical, or bo
th
merged_df.loc[merged_df['prod_method']=='Digital Animation', 'animated_o
r_musical']='animated'
merged_df.loc[merged_df['genre']=='Musical', 'animated_or_musical']='mus
ical'
merged_df.loc[(merged_df['genre']=='Musical') & (merged_df['prod_method'
]=='Digital Animation'), 'animated_or_musical']='both'
merged_df['animated_or_musical'].value_counts()
```

Out[131]:
```
animated    120
musical      15
both          4
Name: animated_or_musical, dtype: int64
```

In [132]:

```
# plotting distribution and world gross of animated/musical movies
g = sns.catplot(data=merged_df.loc[(merged_df['genre']=='Musical') | (merged_df['prod_method']=='Digital Animation')], kind="swarm", x="animated_or_musical", y="world_gross_mil", hue='animated_or_musical', order=['animated', 'musical', 'both'], s=6)
g.fig.set_size_inches(9,5)
plt.title('World Gross for Animated/Musical Movies', fontsize=20)
plt.xlabel('Movie Type', fontsize=16)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.axhline(y=1000, ls='--', c='green');
#plt.legend(loc='upper left', fontsize=14);
#plt.tight_layout()
#plt.savefig('./images/fig11.png')
```



From reviewing the data above and analyzing this plot, I was able to determine that broadening the recomended movie type from simply *musicals* to *animated movies with a musical component* would ensure that no high-grossing animated movies were excluded. And again, since all of these animated and animated/musical movies were Kids Fiction as well, they all had a combination of factors that were correlated with high world gross.

Based on my analysis of production methods, my recommendation to Microsoft was that they should prioritize animation, whether solely digital animation or in conjunction with live action, in order to achieve the highest possible movie gross.

Movie Type Recommendations

After careful consideration of the findings above, I had two recommendations regarding movie type:

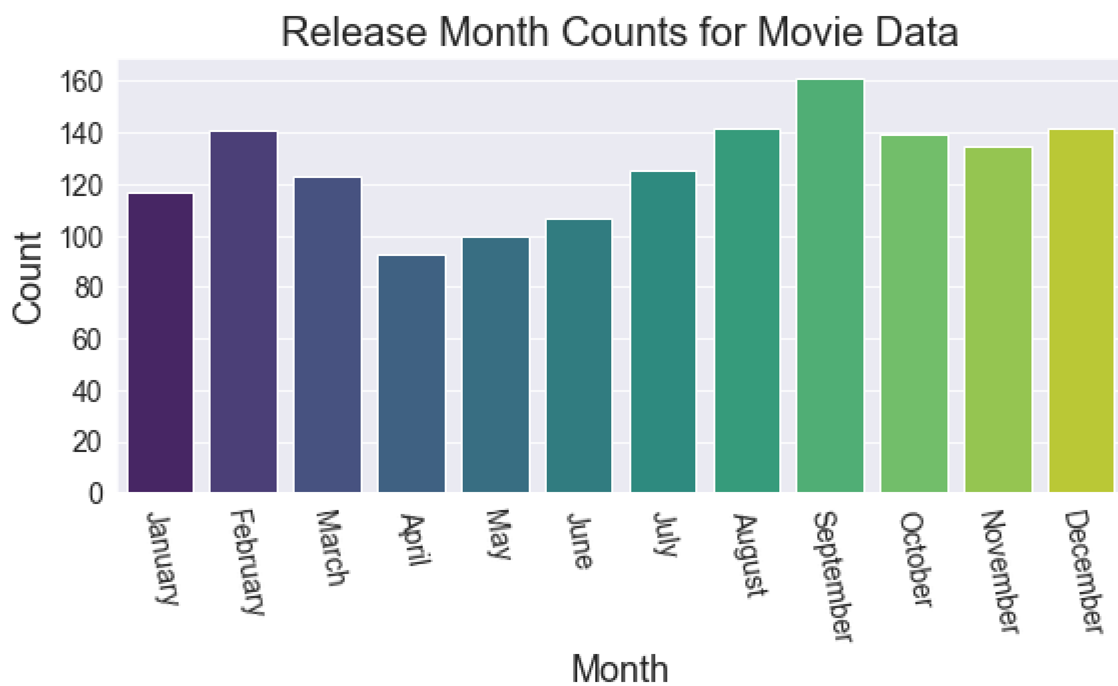
1. Release a super hero movie.
2. Release a an animated kids fiction movie with a musical component.

Release Month: When is the most lucrative time of year to release a movie?

In [133]:
```
# getting value counts for release month column
merged_df['release_month'].value_counts()
```

Out[133]:
```
September    161
August       142
December     142
February     141
October      139
November     135
July         125
March        123
January      117
June         107
May          100
April         93
Name: release_month, dtype: int64
```

In [178]:
```
# plotting the number of movies per release month in dataset
plt.figure(figsize=(9,4))
months = ['January', 'February', 'March', 'April','May','June', 'July',
'August','September', 'October', 'November', 'December']
sns.countplot(x='release_month', data=merged_df, order=months, palette=
'viridis')
plt.title('Release Month Counts for Movie Data', fontsize=20)
plt.ylabel('Count', fontsize=18)
plt.xlabel('Month', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.xticks(rotation=-80);
#plt.subplots_adjust(bottom=0.2)
#plt.savefig('./images/fig28.png')
```



In this dataset, movie release months were fairly evenly distributed throughout the year, with the most releases in September and the least in April.

In [135]:

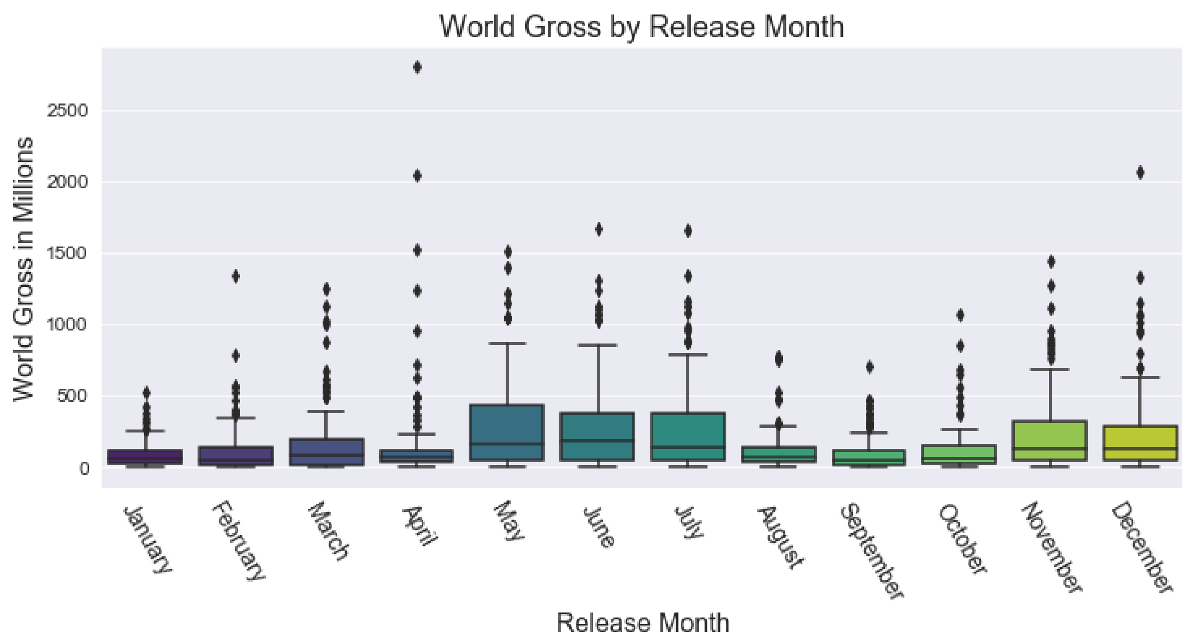
```
# getting mean and median world gross amounts by release month
months_df = merged_df['world_gross_mil'].groupby(merged_df['release_mont
h']).agg(['median', 'mean'])
months_df.sort_values(by='mean', ascending=False)
```

Out[135]:

	median	mean
release_month		
May	169.07	305.07
June	189.62	291.25
July	146.60	275.07
November	128.26	250.93
December	125.04	234.83
April	70.69	197.66
March	87.15	172.53
October	65.25	115.02
February	53.09	114.67
August	73.14	108.64
January	61.64	93.31
September	45.17	89.21

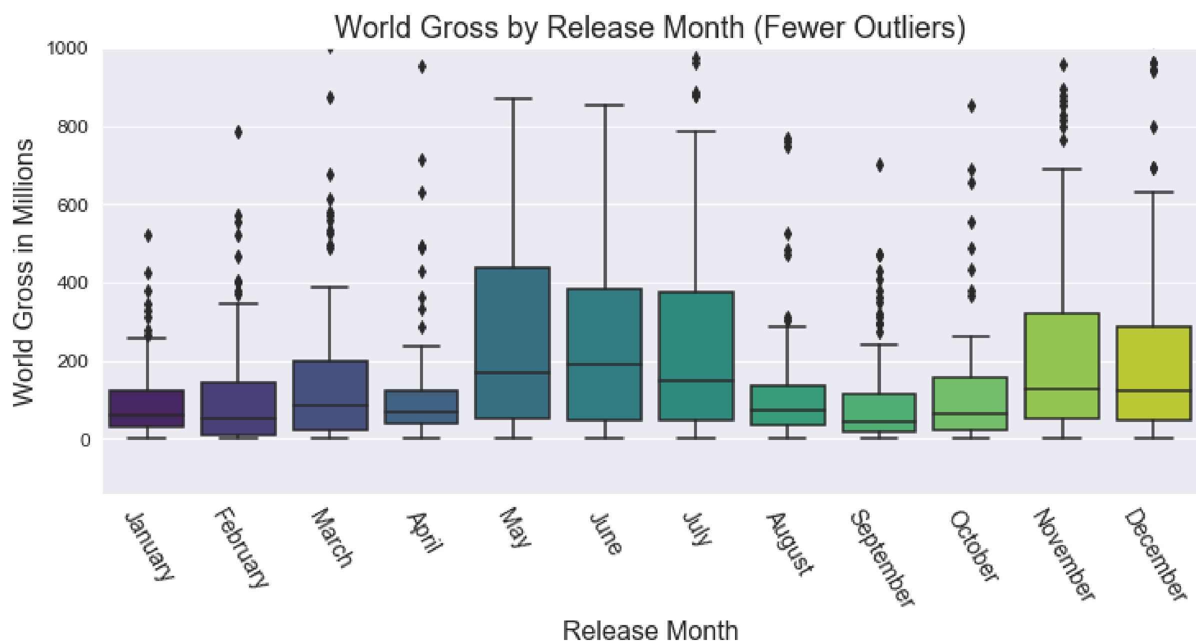
In [136]:

```
# generating boxplots of world gross by release month
plt.figure(figsize=(12, 5))
sns.set_style('darkgrid')
months = ['January', 'February', 'March', 'April','May','June', 'July',
'August','September', 'October', 'November', 'December']
sns.boxplot(x='release_month', y='world_gross_mil', data=merged_df, orde
r=months, palette='viridis')
plt.xticks(rotation=-60)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xlabel('Release Month', fontsize = 16)
plt.title('World Gross by Release Month', fontsize = 18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12);
#saved in images as fig12
#plt.subplots_adjust(bottom=0.2)
#plt.savefig('./images/fig12.png')
```



In [137]:

```
# generating box plots of world gross by release month with fewer outlie
rs
plt.figure(figsize=(12,5))
sns.set_style('darkgrid')
sns.boxplot(x='release_month', y='world_gross_mil', data=merged_df, orde
r=months, palette='viridis')
plt.xticks(rotation=-60)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xlabel('Release Month', fontsize =16)
plt.title('World Gross by Release Month (Fewer Outliers)', fontsize = 18
)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12)
plt.ylim(None, 1000);
#saved in images as fig13
#plt.subplots_adjust(bottom=0.2)
#plt.savefig('./images/fig13.png')
```



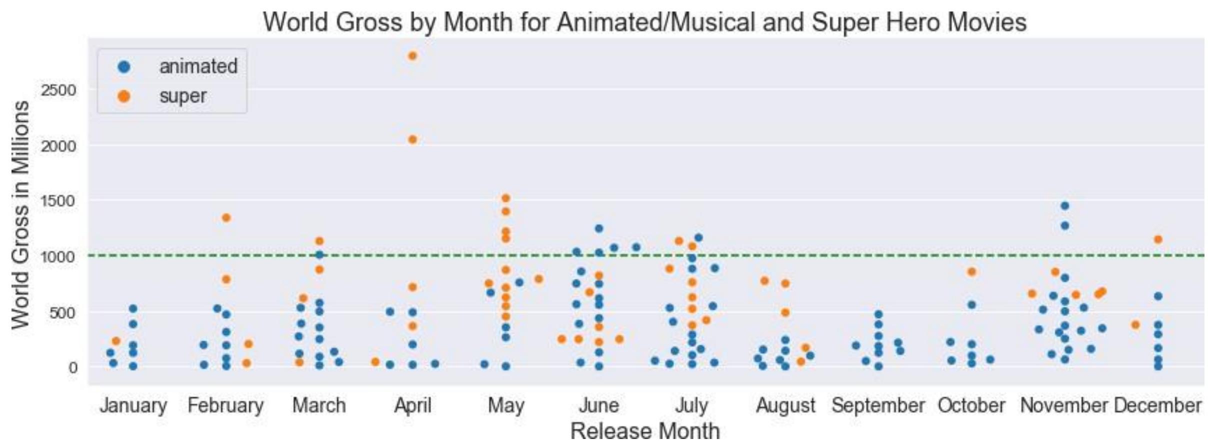
Based on release month alone, May, June, and July seemed to be the most profitable months with the highest mean value for world box office gross. But I wondered, how much does the type of movie impact the time of year that a movie does best?

In [138]:

```
# creating column to signify whether a title is animated or superhero
merged_df.loc[merged_df['prod_method']=='Digital Animation', 'animated_o
r_super']='animated'
merged_df.loc[merged_df['creative_type']=='Super Hero', 'animated_or_sup
er']='super'
```

In [139]:

```
# plotting world gross by month for animated and super hero movies
g = sns.catplot(data=merged_df, kind="swarm", x="release_month", y="worl
d_gross_mil", hue="animated_or_super", order=months, s=6, legend=False)
g.fig.set_size_inches(14,4)
plt.title('World Gross by Month for Animated/Musical and Super Hero Movi
es', fontsize=18)
plt.xlabel('Release Month', fontsize=16)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12)
plt.legend(loc='upper left', fontsize=14)
plt.axhline(y=1000, ls='--', c='green');
#saved in images as fig14
#plt.tight_layout()
#plt.savefig('./images/fig14.png')
```



By plotting world gross by release month for only the types of movies I was recommending, super hero and animated, I was able to determine that there were certain trends that could be observed. While the super hero movies seemed to do best in April and May, the animated movies seemed to do best in June, July, and November.

In [140]:

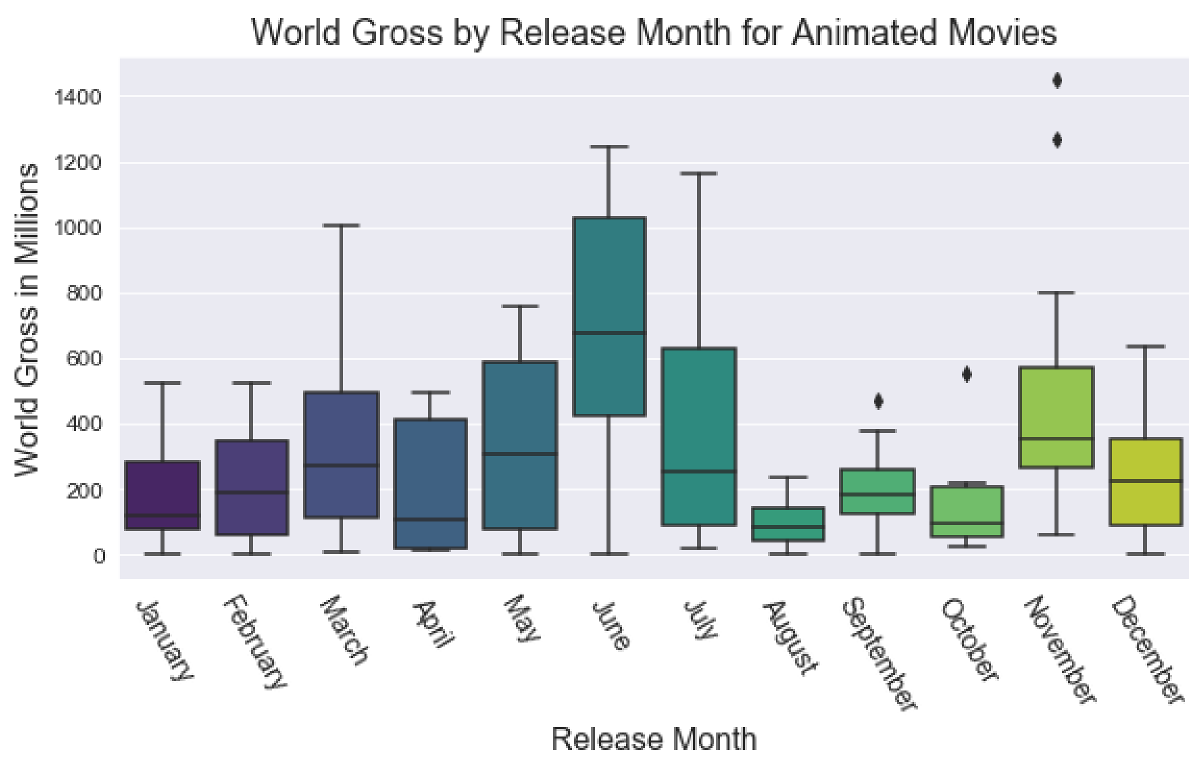
```
# getting mean and median world gross amounts by release month for anima
ted movies
animated_months_df = merged_df.loc[merged_df['animated_or_super']=='anim
ated']
animated_months_gb = animated_months_df['world_gross_mil'].groupby(anima
ted_months_df['release_month']).agg(['median', 'mean'])
animated_months_gb.sort_values(by='mean', ascending=False)
```

Out[140]:

	median	mean
release_month		
June	679.09	656.05
November	355.09	482.04
July	251.73	400.31
May	307.56	342.84
March	269.81	326.02
December	227.54	254.13
February	191.93	222.17
April	110.47	205.19
September	185.62	201.90
January	122.75	195.74
October	97.65	173.41
August	84.24	95.30

In [141]:

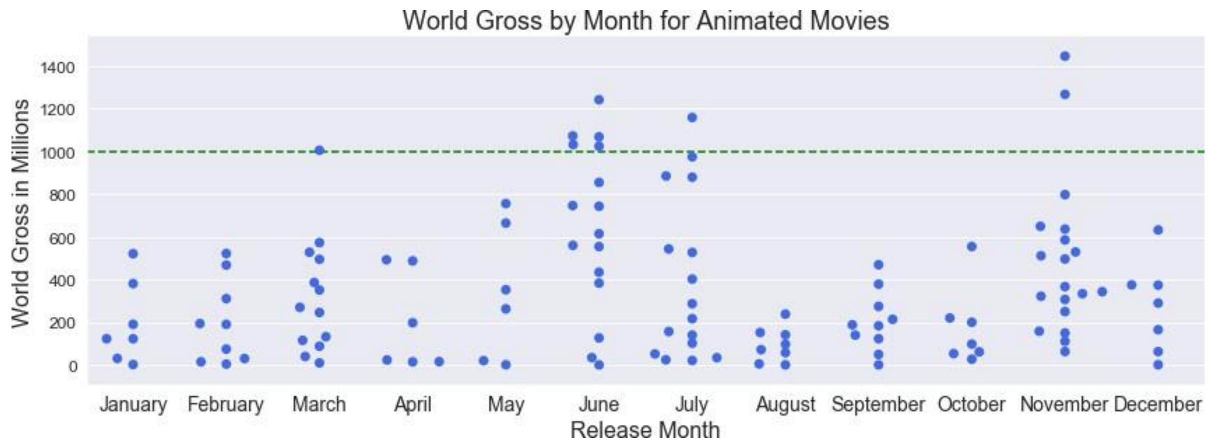
```
# generating boxplots of world gross by release month for animated movie
s
plt.figure(figsize=(10, 5))
sns.set_style('darkgrid')
months = ['January', 'February', 'March', 'April','May','June', 'July',
'August','September', 'October', 'November', 'December']
sns.boxplot(x='release_month', y='world_gross_mil', data=merged_df.loc[m
erged_df['animated_or_super']=='animated'], order=months, palette='virid
is')
plt.xticks(rotation=-60)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xlabel('Release Month', fontsize = 16)
plt.title('World Gross by Release Month for Animated Movies', fontsize =
18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12);
#saved in images as fig15
#plt.subplots_adjust(bottom=0.2)
#plt.savefig('./images/fig15.png')
```



After taking a closer look at release month statistics for animated movies, I was able to confirm that June and November were the highest-grossing months for an animated movie release.

In [142]:

```
# plotting world gross and release month for animated movies
g = sns.catplot(data=merged_df.loc[merged_df['prod_method']=='Digital An
imation'], kind="swarm", x="release_month", y="world_gross_mil", color=
'royalblue', order=months, s=7, legend=False)
g.fig.set_size_inches(14,4)
plt.title('World Gross by Month for Animated Movies', fontsize=18)
plt.xlabel('Release Month', fontsize=16)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12)
plt.axhline(y=1000, ls='--', c='green');
#saved in images as fig16
#plt.tight_layout()
#plt.savefig('./images/fig16.png')
```



Almost all of the animated movies over the one-billion mark for world gross were released in June or November, with the exception of one in July and one right on the mark in March. The highest-grossing ones were released in November.

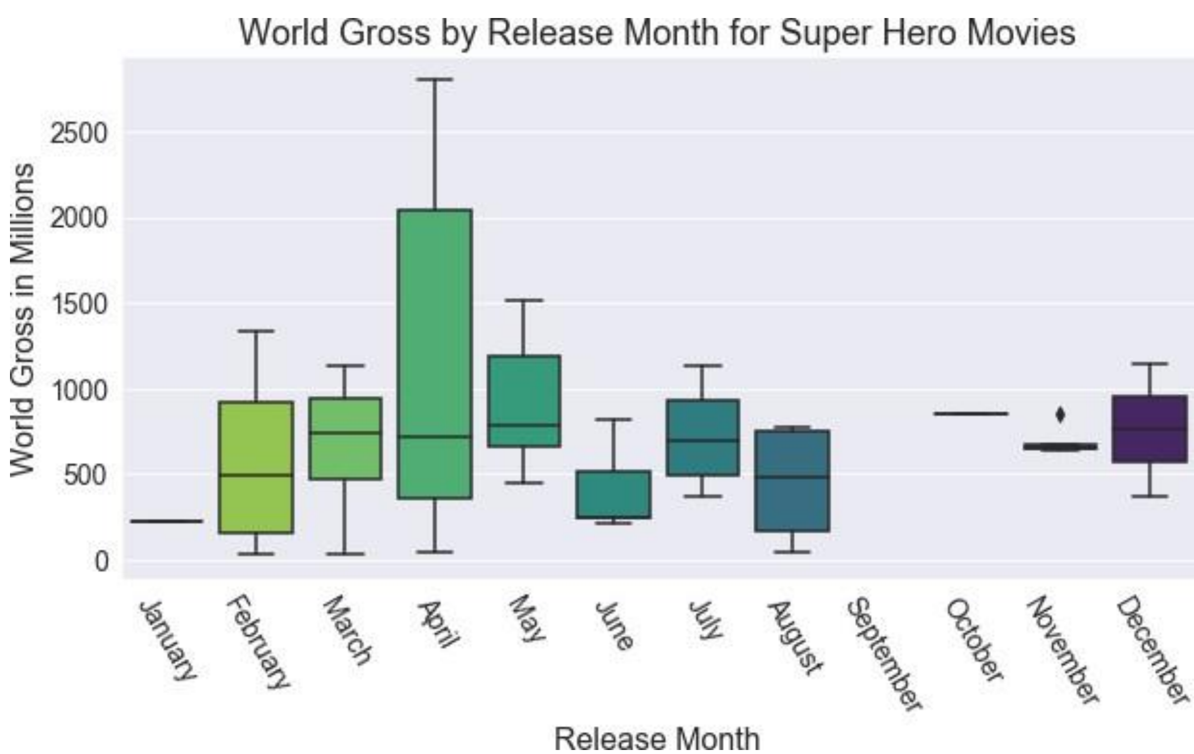
In [143]:

```
# getting mean and median world gross amounts by release month for super
hero movies
super_months_df = merged_df.loc[merged_df['animated_or_super']=='super']
super_months_gb = super_months_df['world_gross_mil'].groupby(super_month
s_df['release_month']).agg(['median', 'mean'])
super_months_gb.sort_values(by='mean', ascending=False)
```

Out[143]:

release_month	median	mean
April	714.40	1192.24
May	786.68	909.47
October	853.63	853.63
December	759.50	759.50
July	690.52	722.41
November	655.95	695.60
March	743.30	663.41
February	493.44	589.13
August	485.00	442.74
June	246.36	399.82
January	229.16	229.16

In [144]:
```
# generating box plots for world gross by relese month of super hero mov
ies
plt.figure(figsize=(10, 5))
sns.set_style('darkgrid')
months = ['January', 'February', 'March', 'April','May','June', 'July',
'August','September', 'October', 'November', 'December']
sns.boxplot(x='release_month', y='world_gross_mil', data=merged_df.loc[m
erged_df['animated_or_super']=='super'], order=months, palette='viridis_
r')
plt.xticks(rotation=-60)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xlabel('Release Month', fontsize = 16)
plt.title('World Gross by Release Month for Super Hero Movies', fontsize
= 18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14);
#saved in images as fig17
#plt.subplots_adjust(bottom=0.2)
#plt.savefig('./images/fig17.png')
```



After taking a closer look at release month statistics for super hero movies, I was able to confirm that April and May were the highest-grossing months for a super hero movie release.

In [145]:
```
# plotting world gross by month for super hero movies
g = sns.catplot(data=merged_df.loc[merged_df['creative_type']=='Super Hero'], kind="swarm", x="release_month", y="world_gross_mil", color='darkorange', order=months, s=6, legend=False)
g.fig.set_size_inches(14,5)
plt.title('World Gross by Month for Super Hero Movies', fontsize=18)
plt.xlabel('Release Month', fontsize=16)
plt.ylabel('World Gross in Millions', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=12)
plt.axhline(y=1000, ls='--', c='green');
#saved in images as fig18
#plt.tight_layout()
#plt.savefig('./images/fig18.png')
```

Most of the super hero movies over the one-billion mark were released in May. The two highest-grossing ones were released in April. There were a couple released in other months as well.

Release Month Recommendations

After careful consideration of the findings above, I had two recommendations regarding movie release months:

1. Release a kids fiction animated movie with a musical component in June or November.
2. Release a super hero movie in April or May.

Production Budget: What budget amount tends to achieve the highest box office gross?

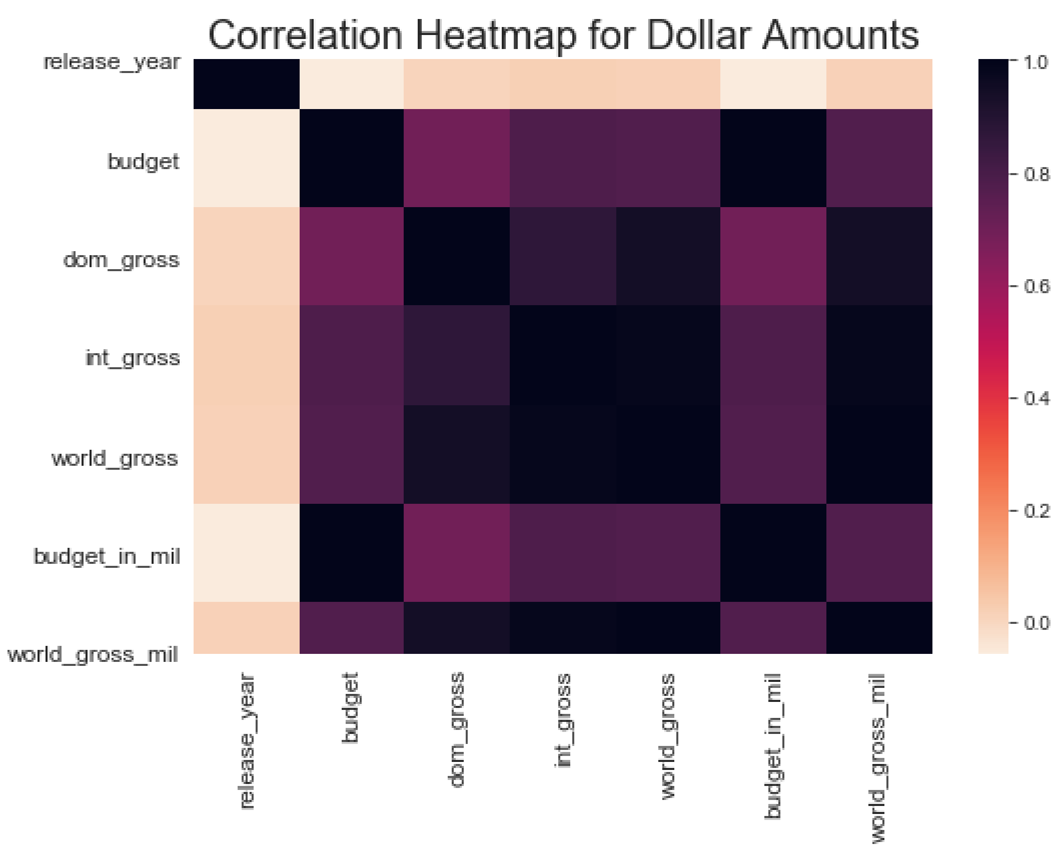
The first thing I did was run the .corr method on merged_df to see if there were any strong correlations in the numerical data. I then generated a heatmap based on these correlations.

In [146]:
```
# generating correlations
corr = merged_df.corr()
corr
```

Out[146]:

	release_year	budget	dom_gross	int_gross	world_gross	budget_in_mil	world_
release_year	1.00	-0.06	0.01	0.02	0.02	-0.06	
budget	-0.06	1.00	0.69	0.78	0.78	1.00	
dom_gross	0.01	0.69	1.00	0.87	0.94	0.69	
int_gross	0.02	0.78	0.87	1.00	0.98	0.78	
world_gross	0.02	0.78	0.94	0.98	1.00	0.78	
budget_in_mil	-0.06	1.00	0.69	0.78	0.78	1.00	
world_gross_mil	0.02	0.78	0.94	0.98	1.00	0.78	

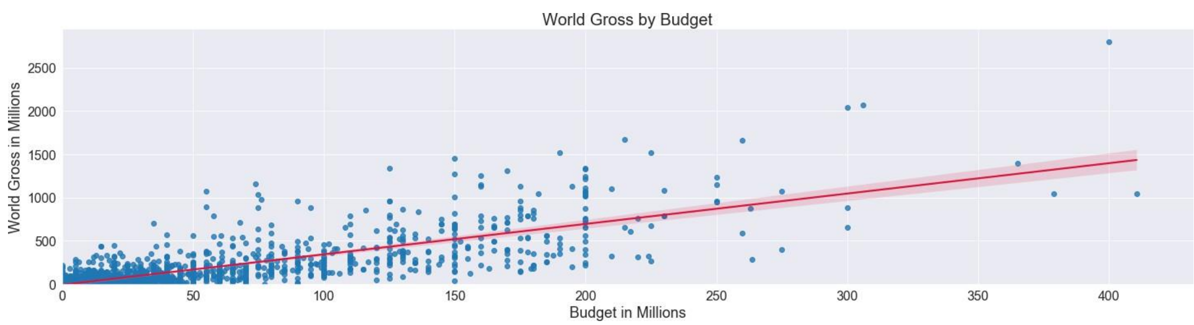
In [147]:
```
# plotting heatmap of correlations
plt.figure(figsize=(8,6))
plt.title("Correlation Heatmap for Dollar Amounts", fontsize=20)
plt.yticks(fontsize=12)
plt.xticks(fontsize=12)
sns.heatmap(corr, cmap='rocket_r');
#saved in images as fig19
plt.tight_layout()
#plt.savefig('./images/fig19.png')
```



The heatmap shows darker colors where two values are highly correlated and lighter colors where there is less of a correlation. It was clear from both the correlation method and the heatmap that there was a strong correlation between budget and world gross, as well as budget and international gross. While the correlation between budget and domestic gross was not quite as strong, it was still right at the threshold of a high correlation. From this visualization, I could conclude that, generally speaking, the more money you put into a movie, the more money you are likely to make from it.

In [148]:

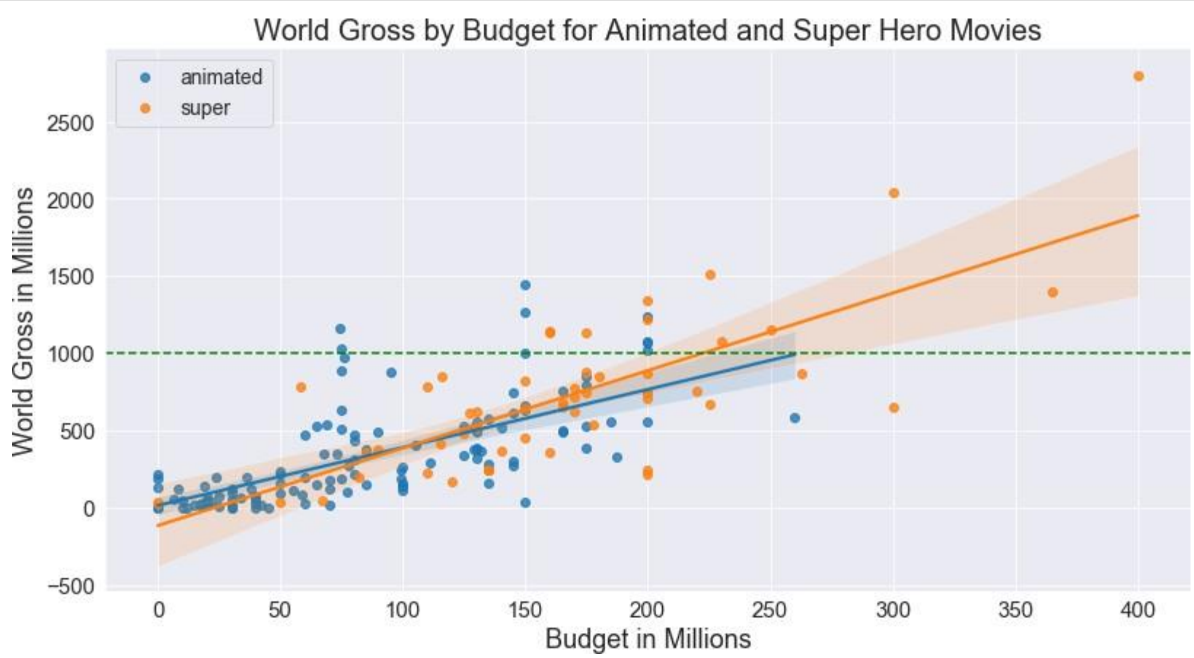
```
# plotting world gross by budget in terms of genre with line of best fit
sns.lmplot(x='budget_in_mil', y='world_gross_mil', data=merged_df, aspec
t=4, line_kws={'color': 'crimson'})
plt.title('World Gross by Budget', fontsize=20)
plt.xlabel('Budget in Millions', fontsize=18)
plt.ylabel('World Gross in Millions', fontsize=18)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.ylim(0,None)
plt.xlim(0, None);
#saved in images as fig20
#plt.tight_layout()
#plt.savefig('./images/fig20.png')
```



The regression line in this plot shows the general increase in movie gross as the budget increases. While this varies greatly from movie to movie, it was a good starting point as I worked towards combining my findings and forming my recommendations. This plot confirmed my hypothesis that a higher production budget typically leads to a higher box office gross.

In [149]:

```
# plotting world gross by budget for animated and super hero movies
sns.set_style('darkgrid')
g = sns.lmplot(x='budget_in_mil', y='world_gross_mil', data=merged_df.lo
c[(merged_df['prod_method']=='Digital Animation') | (merged_df['creative
_type']=='Super Hero')], hue='animated_or_super', aspect=3, legend=False
)
plt.title('World Gross by Budget for Animated and Super Hero Movies', fo
ntsize=20)
plt.xlabel('Budget in Millions', fontsize=18)
plt.ylabel('World Gross in Millions', fontsize=18)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
g.fig.set_size_inches(11,6)
plt.legend(loc='upper left', fontsize=14)
#for w, x, y, z in zip(merged_df['animated_or_super'], merged_df['budget
_in_mil'], merged_df['world_gross_mil'], merged_df['title']):
    #if (w == 'animated') & (y>1250):
        #plt.text(x = x+5, y = y-15, s = z.upper(), fontsize=14, color
='black')
#for w, x, y, z in zip(merged_df['animated_or_super'], merged_df['budget
_in_mil'], merged_df['world_gross_mil'], merged_df['title']):
    #if (w == 'super') & (y>1250):
        #plt.text(x = x+5, y = y-15, s = z.upper(), fontsize=14, color
='black')
plt.axhline(y=1000, ls='--', c='green');
#saved in images as fig21
#plt.tight_layout()
#plt.savefig('./images/fig21.png')
```

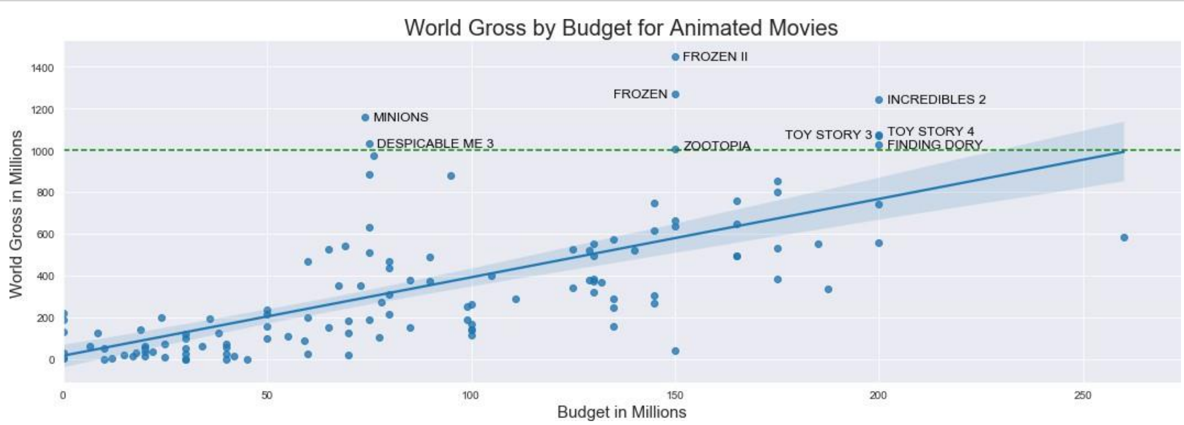


For the specific movie types we are looking at, we can see that super hero movies benefit from a very large budget, while animated movies benefit from a moderately large budget. Both regression lines show that as the budget increases, the world gross tends to increase as well.

In [150]:
```
# getting descriptive stats for animated movie budgets
merged_df.loc[merged_df['prod_method']=='Digital Animation', 'budget_in_
mil'].describe()
```

Out[150]:
```
count    124.00
mean      87.68
std       59.33
min        0.00
25%       37.50
50%       76.75
75%      135.00
max      260.00
Name: budget_in_mil, dtype: float64
```

In [151]:
```
# plotting world gross by budget for animated movies
sns.set_style('darkgrid')
sns.lmplot(x='budget_in_mil', y='world_gross_mil', data=merged_df.loc[me
rged_df['prod_method']=='Digital Animation'], aspect=3)
plt.title('World Gross by Budget for Animated Movies', fontsize=20)
plt.xlabel('Budget in Millions', fontsize=15)
plt.ylabel('World Gross in Millions', fontsize=15)
plt.xlim(0, None)
plt.axhline(y=1000, ls='--', c='green')
for v, w, x, y, z in zip(merged_df['genre'], merged_df['prod_method'], m
erged_df['budget_in_mil'], merged_df['world_gross_mil'], merged_df['titl
e']):
    if (z=='Frozen'):
        plt.text(x=x-15, y=y-15, s=z.upper(), fontsize=12, color='black'
)
    elif  z=='Toy Story 3':
        plt.text(x=x-23, y=y-10, s=z.upper(), fontsize=12, color='black'
)
    elif  z=='Zootopia':
        plt.text(x=x+2, y=y, s=z.upper(), fontsize=12, color='black')
    elif (z=='Toy Story 4'):
        plt.text(x = x+2, y = y, s = z.upper(), fontsize=12, color='blac
k')
    elif (w=='Digital Animation') and y>1000:
        plt.text(x = x+2, y = y-15, s = z.upper(), fontsize=12, color='b
lack');
#saved in images as fig22
#plt.tight_layout()
#plt.savefig('./images/fig22.png')
```



The mean budget for animated movies is 87.68 million dollars. For the highest-grossing animated movies, the optimal budget is between 75 million dollars and 200 million dollars, with the two highest-grossing movies having a budget of 150 million dollars.

In [152]:

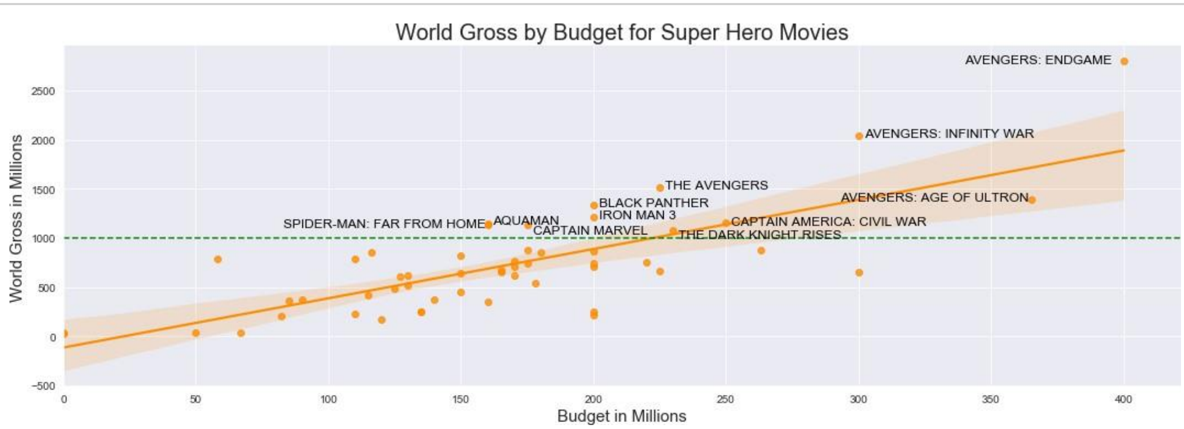
```
# getting descriptive statistics for super hero movie budgets
merged_df.loc[merged_df['creative_type']=='Super Hero', 'budget_in_mil']
.describe()
```

Out[152]:

```
count     53.00
mean     165.21
std       76.16
min        0.00
25%      125.00
50%      165.00
75%      200.00
max      400.00
Name: budget_in_mil, dtype: float64
```

In [153]:

```
# plotting world gross by budget for super hero movies
sns.set_style('darkgrid')
sns.lmplot(x='budget_in_mil', y='world_gross_mil', data=merged_df.loc[me
rged_df['creative_type']=='Super Hero'], line_kws={'color': 'darkorange'
}, scatter_kws={'color': 'darkorange'}, aspect=3)
plt.title('World Gross by Budget for Super Hero Movies', fontsize=20)
plt.xlabel('Budget in Millions', fontsize=15)
plt.ylabel('World Gross in Millions', fontsize=15)
plt.xlim(0, None)
plt.axhline(y=1000, ls='--', c='green');
for w, x, y, z in zip(merged_df['animated_or_super'], merged_df['budget_
in_mil'], merged_df['world_gross_mil'], merged_df['title']):
    if (z=='Avengers: Endgame'):
        plt.text(x=x-60, y=y-15, s=z.upper(), fontsize=12, color='black'
)
    elif (z=='Avengers: Infinity War'):
        plt.text(x=x+2, y=y-15, s=z.upper(), fontsize=12, color='black')
    elif (z=='Avengers: Age of Ultron'):
        plt.text(x=x-72, y=y-15, s=z.upper(), fontsize=12, color='black'
)
    elif (z=='The Avengers'):
        plt.text(x=x+2, y=y-15, s=z.upper(), fontsize=12, color='black')
    elif (z=='Spider-Man: Far From Home'):
        plt.text(x=x-77, y=y-15, s=z.upper(), fontsize=12, color='black'
)
    elif (z=='Aquaman'):
        plt.text(x=x+2, y=y, s=z.upper(), fontsize=12, color='black')
    elif (z=='Captain Marvel'):
        plt.text(x=x+2, y=y-90, s=z.upper(), fontsize=12, color='black')
    elif (z=='The Dark Knight Rises'):
        plt.text(x=x+2, y=y-80, s=z.upper(), fontsize=12, color='black')
    elif (w == 'super') & (y>1000):
        plt.text(x = x+2, y = y-15, s = z.upper(), fontsize=12, color='b
lack')
#saved in images as fig23
#plt.tight_layout()
#plt.savefig('./images/fig23.png')
```



The mean budget for super hero movies is 165.21 million dollars--almost double the mean budget for an animated movie. For the highest-grossing super hero movies, the optimal budget is between 200 million dollars and 400 million dollars, with the two highest-grossing movies having budgets between 300 and 400 million dollars.

Budget Recommendations

After careful consideration of the findings above, my main observation was that a higher production budget leads to a higher gross. I had two specific recommendations regarding production budget for the preferred movie types:

1. For the highest box office gross, an animated move should have a budget of 75 to 200 million dollars.
2. For the highest box office gross, a super hero movie should have a budget of 200 to 400 million dollars.

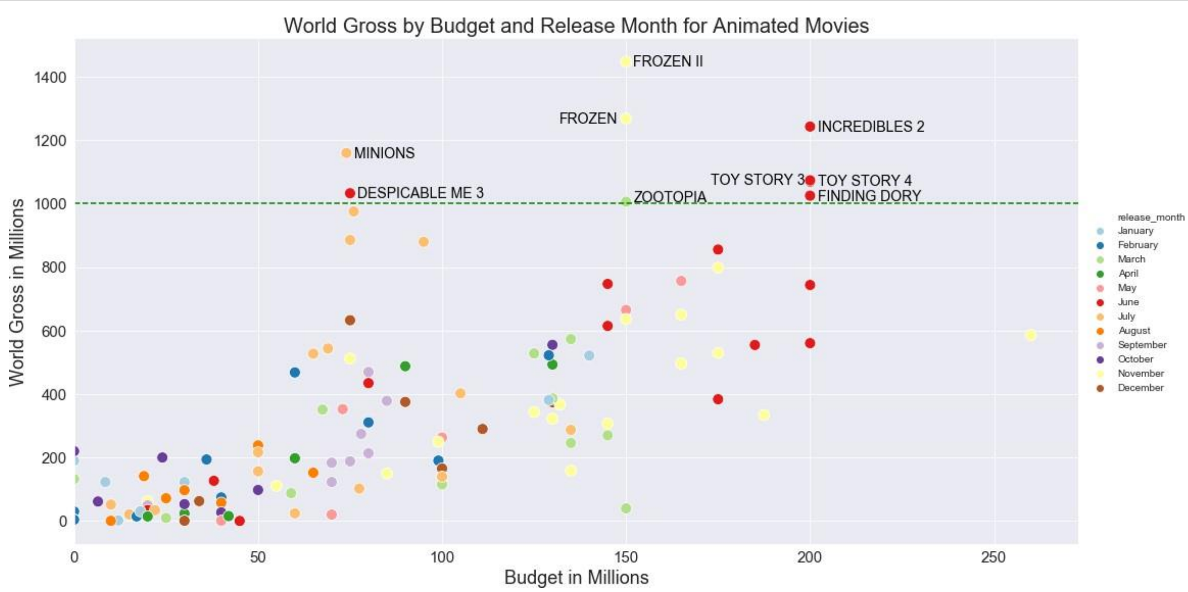
Putting It All Together!

Now that I had answers to my three main questions regarding the business problem, I wanted to put them all together into one visualization.

This plot shows the specifics of the top-grossing movies in the animation category. As discussed previously, the highest-grossing movies were mostly released in June or November and had budgets between 75 and 200 million dollars.

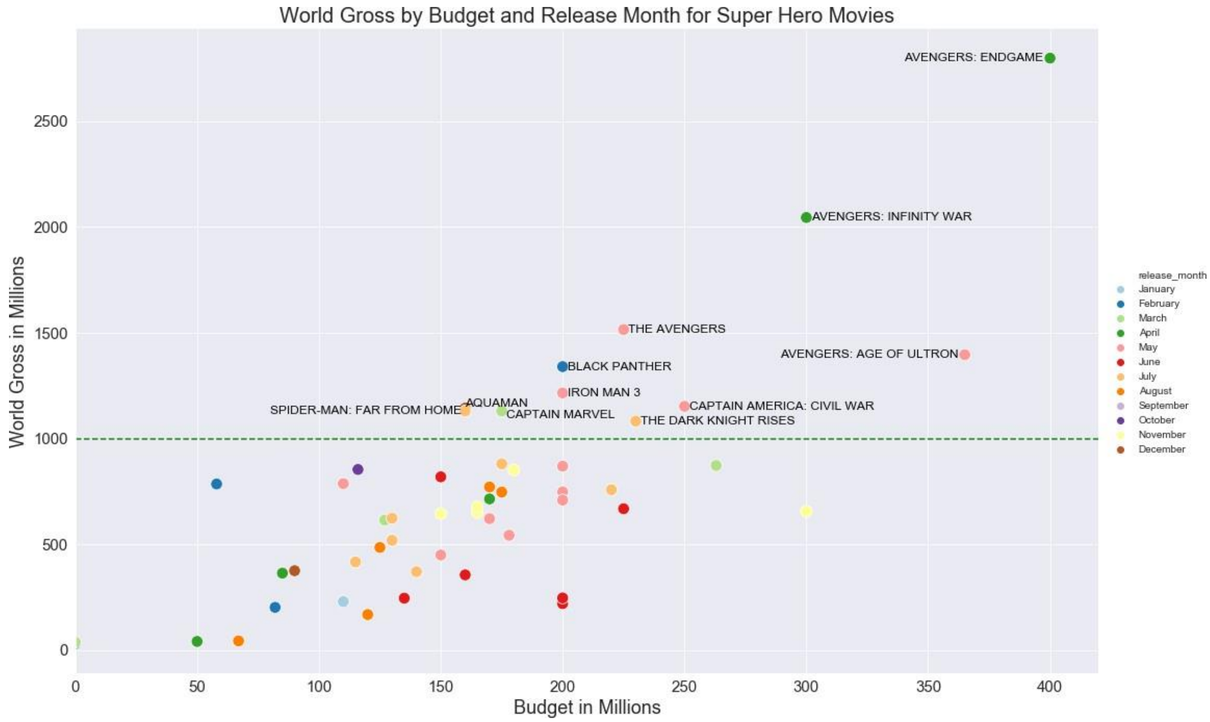
In [173]:

```
# plotting world gross by budget and release month for animated
sns.set_style('darkgrid')
g = sns.relplot(x='budget_in_mil', y='world_gross_mil', data=merged_df.l
oc[(merged_df['prod_method']=='Digital Animation')], hue='release_month'
, hue_order=['January', 'February', 'March', 'April', 'May', 'June', 'Ju
ly', 'August', 'September', 'October', 'November', 'December'], s=120, a
spect=3, palette='Paired')
plt.title('World Gross by Budget and Release Month for Animated Movies',
fontsize=20)
plt.xlabel('Budget in Millions', fontsize=18)
plt.ylabel('World Gross in Millions', fontsize=18)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlim(0, None)
g.fig.set_size_inches(16,8)
for v, w, x, y, z in zip(merged_df['genre'], merged_df['prod_method'], m
erged_df['budget_in_mil'], merged_df['world_gross_mil'], merged_df['titl
e']):
    if (z=='Frozen'):
        plt.text(x=x-18, y=y-15, s=z.upper(), fontsize=14, color='black'
)
    elif  z=='Toy Story 3':
        plt.text(x=x-27, y=y-10, s=z.upper(), fontsize=14, color='black'
)
    elif  z=='Zootopia':
        plt.text(x=x+2, y=y, s=z.upper(), fontsize=14, color='black')
    elif (w=='Digital Animation') and y>1000:
        plt.text(x = x+2, y = y-15, s = z.upper(), fontsize=14, color='b
lack')
plt.axhline(y=1000, ls='--', c='green');
#saved in images as fig24
#plt.tight_layout()
#plt.savefig('./images/fig24.png')
```



In [175]:

```
# plotting world gross by budget and release month for super hero movies
sns.set_style('darkgrid')
g = sns.relplot(x='budget_in_mil', y='world_gross_mil', data=merged_df.l
oc[merged_df['creative_type']=='Super Hero'], hue='release_month', hue_o
rder=['January', 'February', 'March', 'April', 'May', 'June', 'July', 'A
ugust', 'September', 'October', 'November', 'December'], s=130, aspect=3
, palette='Paired')
plt.title('World Gross by Budget and Release Month for Super Hero Movie
s', fontsize=20)
plt.xlabel('Budget in Millions', fontsize=18)
plt.ylabel('World Gross in Millions', fontsize=18)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlim(0, None)
g.fig.set_size_inches(16,10)
for w, x, y, z in zip(merged_df['animated_or_super'], merged_df['budget_
in_mil'], merged_df['world_gross_mil'], merged_df['title']):
    if (z=='Avengers: Endgame'):
        plt.text(x=x-60, y=y-15, s=z.upper(), fontsize=12, color='black'
)
    elif (z=='Avengers: Infinity War'):
        plt.text(x=x+2, y=y-15, s=z.upper(), fontsize=12, color='black')
    elif (z=='Avengers: Age of Ultron'):
        plt.text(x=x-76, y=y-15, s=z.upper(), fontsize=12, color='black'
)
    elif (z=='The Avengers'):
        plt.text(x=x+2, y=y-15, s=z.upper(), fontsize=12, color='black')
    elif (z=='Spider-Man: Far From Home'):
        plt.text(x=x-80, y=y-15, s=z.upper(), fontsize=12, color='black'
)
    elif (z=='Aquaman'):
        plt.text(x=x, y=y+10, s=z.upper(), fontsize=12, color='black')
    elif (z=='Captain Marvel'):
        plt.text(x=x+2, y=y-35, s=z.upper(), fontsize=12, color='black')
    elif (w == 'super') & (y>1000):
        plt.text(x = x+2, y = y-15, s = z.upper(), fontsize=12, color='b
lack')
plt.axhline(y=1000, ls='--', c='green');
#saved in images as fig25
#plt.tight_layout()
#plt.savefig('./images/fig25.png')
```



This plot shows the specifics of the top-grossing movies in the super hero category. As discussed previously, the highest-grossing movies were mostly released in April, May, or July and most had budgets between 200 and 400 million dollars.

These visualizations show the relationship between all the key factors that I analyzed: movie type, release month, and production budget. Based on the top movies that meet these criteria, super hero movies do best when given a very high budget and are released in April or May. Animated musicals find the most success when released in June or November, and don't necessarily need quite as high of a budget to achieve a high gross, although their gross is also comparatively lower than that of super hero movies.

Additional Attributes: Based on these findings, what else do top-grossing movies have in common?

Directors

Now that we know that a high-budget super hero movie released in April or May is the most rewarding combination of factors, how can we make this movie even better and ensure that it reaches its full potential? One way that we can increase our chances of success is by hiring the right director for this type of movie.

To further this point, I utilized the TMDb API to obtain the director names for all the super hero movies in my dataset.

In [156]:
```python
def get_director(title):
    """
    Updates director information for movie in dataframe.

    Queries TMDB for a given movie title.
    Retrieves TMDB movie_id for title.
    Retrieves director information based on movie_id.
    Adds director information to a list.
    Converts director information from list to string.
    Adds new director value as string to movie's row in dataframe.

    Parameters:
    title (str): user input movie title.

    Returns:
    Updated cells in Pandas DataFrame.

    """
    title_r = title.replace(' ', '+')
    url = f"https://api.themoviedb.org/3/search/movie?api_key={api_key}&
query={title_r}"
    response = requests.get(url)
    if len(response.json()['results']) > 0:
        movie_id = response.json()['results'][0]['id']
        url2 = f"https://api.themoviedb.org/3/movie/{movie_id}/credits?a
pi_key={api_key}"
        response2 = requests.get(url2)
        crew = response2.json()['crew']
        directors = []
        for member in crew:
            if member['job'] == 'Director':
                directors.append(member['name'])
                d = str(directors)
                d = d.replace('[', '').replace(']', '').replace("'","")
                merged_df.loc[merged_df['title']==title, 'director'] = d
    else:
        pass
```

In [157]:
```python
# creating a list of all the super hero movie titles
superhero_titles = [title for title in superhero['title']]
```

In [158]:
```python
# getting director info for movies in list and updating data accordingly
for title in superhero_titles:
    get_director(title)
```

In [159]:
```
# getting director value counts
merged_df.director.value_counts()
```

Out[159]:
```
Anthony Russo, Joe Russo     4
Zack Snyder                  3
James Gunn                   2
Marc Webb                    2
Bryan Singer                 2
James Mangold                2
Jon Watts                    2
Joss Whedon                  2
Dave Green                   2
Taika Waititi                2
Peyton Reed                  2
Tim Story                    1
Ruben Fleischer              1
Martin Campbell              1
Alan Taylor                  1
Patty Jenkins                1
Ryan Coogler                 1
Jon Favreau                  1
Shane Black                  1
Laurent Bouzereau            1
Josh Boone                   1
Ryan Fleck, Anna Boden       1
Jonathan Liebesman           1
Scott Derrickson             1
Simon Kinberg                1
James Wan                    1
David Ayer                   1
David Leitch                 1
Dave Wilson                  1
Neil Marshall                1
Tim Miller                   1
David F. Sandberg            1
Christopher Nolan            1
Michel Gondry                1
Joe Johnston                 1
Chris Williams, Don Hall     1
Name: director, dtype: int64
```

In [160]:

```
# getting mean and median stats for directors
director_stats = merged_df.groupby('director')['world_gross_mil'].agg([
'median', 'mean']).reset_index()
director_stats.sort_values(by='mean', ascending=False)
```

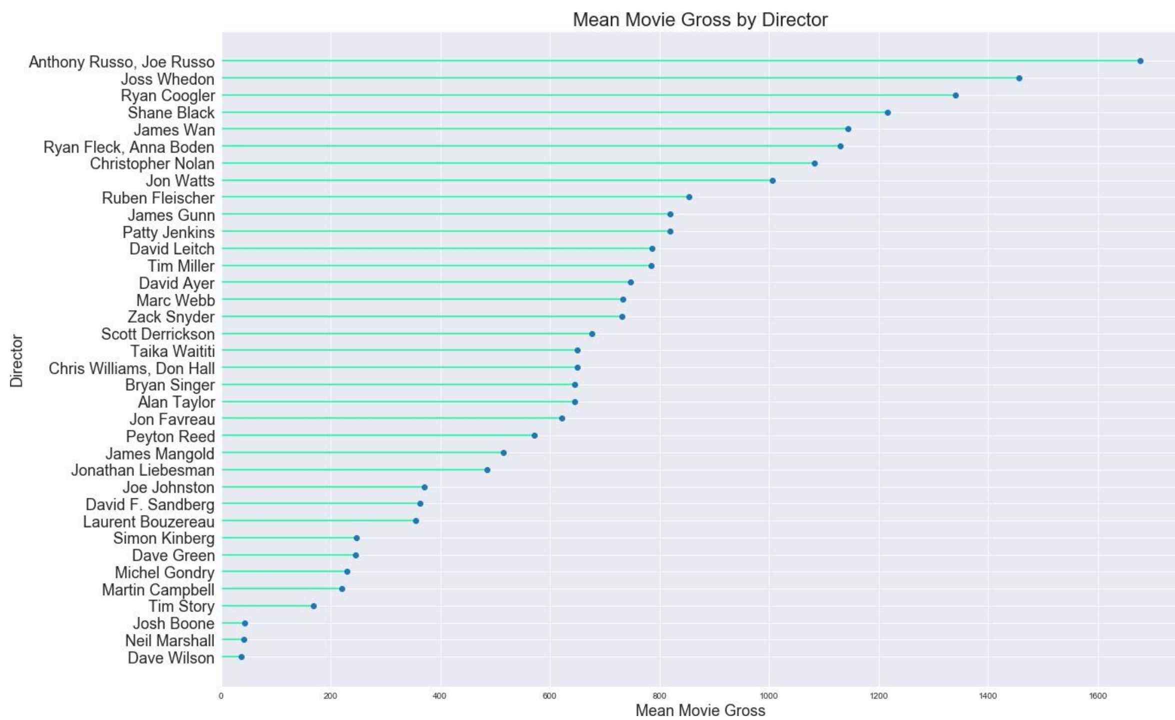
Out[160]:

	director	median	mean
1	Anthony Russo, Joe Russo	1598.23	1677.17
18	Joss Whedon	1455.60	1455.60
27	Ryan Coogler	1339.73	1339.73
30	Shane Black	1215.39	1215.39
12	James Wan	1143.97	1143.97
28	Ryan Fleck, Anna Boden	1129.73	1129.73
4	Christopher Nolan	1082.23	1082.23
15	Jon Watts	1005.06	1005.06
26	Ruben Fleischer	853.63	853.63
10	James Gunn	819.98	819.98
24	Patty Jenkins	818.79	818.79
9	David Leitch	786.68	786.68
33	Tim Miller	785.03	785.03
7	David Ayer	746.85	746.85
20	Marc Webb	733.44	733.44
35	Zack Snyder	668.00	732.11
29	Scott Derrickson	676.35	676.35
32	Taika Waititi	650.36	650.36
3	Chris Williams, Don Hall	649.69	649.69
2	Bryan Singer	645.20	645.20
0	Alan Taylor	644.60	644.60
14	Jon Favreau	621.16	621.16
25	Peyton Reed	571.00	571.00
11	James Mangold	515.33	515.33
16	Jonathan Liebesman	485.00	485.00
13	Joe Johnston	370.57	370.57
8	David F. Sandberg	363.66	363.66
19	Laurent Bouzereau	355.41	355.41
31	Simon Kinberg	246.36	246.36
5	Dave Green	245.33	245.33
22	Michel Gondry	229.16	229.16
21	Martin Campbell	219.54	219.54
34	Tim Story	167.85	167.85
17	Josh Boone	43.15	43.15

	director	median	mean
23	Neil Marshall	40.79	40.79
6	Dave Wilson	37.32	37.32

In [161]:

```
# plotting directors in order of highest-grossing, descending
plt.figure(figsize=(20, 14))
ordered_df = director_stats.sort_values(by='mean')
my_range=range(1,len(director_stats.index)+1)
plt.hlines(y=my_range, xmin=0, xmax=ordered_df['mean'], color='mediumspr
inggreen')
plt.plot(ordered_df['mean'], my_range, "o")
plt.yticks(my_range, ordered_df['director'], fontsize=18)
plt.title("Mean Movie Gross by Director", fontsize=22)
plt.xlabel('Mean Movie Gross', fontsize=18)
plt.ylabel('Director', fontsize=18)
plt.xlim(0, 1750);
#saved in images as fig26
#plt.tight_layout()
#plt.savefig('./images/fig26.png')
```



In [162]:

```
# seeing which movies were directed by Anthony Russo, Joe Russo
merged_df.loc[merged_df['director']=='Anthony Russo, Joe Russo']
```

Out[162]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	dom_
403	2014-04-04	2014	Captain America: The Winter Soldier	Action	Live Action	Super Hero	170000000	2597
602	2016-05-06	2016	Captain America: Civil War	Action	Live Action	Super Hero	250000000	4080
801	2018-04-27	2018	Avengers: Infinity War	Action	Animation/Live Action	Super Hero	300000000	6788
900	2019-04-26	2019	Avengers: Endgame	Action	Animation/Live Action	Super Hero	400000000	8583

In [163]:

```
# seeing which movies were directed by Joss Whedon
merged_df.loc[merged_df['director']=='Joss Whedon']
```

Out[163]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	dom_
200	2012-05-04	2012	The Avengers	Action	Animation/Live Action	Super Hero	225000000	6233
502	2015-05-01	2015	Avengers: Age of Ultron	Action	Animation/Live Action	Super Hero	365000000	4590

As both the director_stats DataFrame and the Mean Movie Gross by Director plot show, for the past ten years, the top five directors (or combinations of directors) for super hero movies by mean world gross are:

1. Anthony Russo, Joe Russo
2. Joss Whedon
3. Ryan Coogler
4. Shane Black
5. James Wan

Hiring one of these top directors to work on a super hero movie can further increase the chances of a successful movie venture.

Composers

For the animated movie, I had found that the musical ones tended to have the highest box office gross. To that end, I again used the TMDb API, this time to obtain composer information for all the animated movies in my dataset. Because not every animated movie was a musical one, they did not all have composer information available. Only the titles that had an associated composer value available were used in the following analysis.

In [164]:

```python
def get_composer(title):
    """
    Updates composer information for movie in dataframe.

    Queries TMDB for a given movie title.
    Retrieves TMDB movie_id for title.
    Retrieves composer information based on movie_id.
    Adds composer information to a list.
    Converts composer information from list to string.
    Adds new composer value as string to movie's row in dataframe.

    Parameters:
    title (str): user input movie title.

    Returns:
    Updated cells in Pandas DataFrame.

    """
    title_r = title.replace(' ', '+')
    url = f"https://api.themoviedb.org/3/search/movie?api_key={api_key}&query={title_r}"
    response = requests.get(url)
    if len(response.json()['results']) > 0:
        movie_id = response.json()['results'][0]['id']
        url2 = f"https://api.themoviedb.org/3/movie/{movie_id}/credits?api_key={api_key}"
        response2 = requests.get(url2)
        crew = response2.json()['crew']
        composers = []
        for member in crew:
            if member['job'] == 'Original Music Composer':
                composers.append(member['name'])
                c = str(composers)
                c = c.replace('[', '').replace(']', '').replace("'","")
                merged_df.loc[merged_df['title']==title, 'composer'] = c
    else:
        pass
```

In [165]:

```python
# creating a list of animated titles
animation_titles = [title for title in merged_df['title'].loc[merged_df['prod_method'] == 'Digital Animation']]
```

In [166]:

```python
# obtaining composer information for animated titles
for title in animation_titles:
    get_composer(title)
```

In [167]:

```
composer_stats = merged_df.groupby('composer')['world_gross_mil'].agg([
'median', 'mean']).reset_index()
composer_stats.sort_values(by='mean', ascending=False)
```

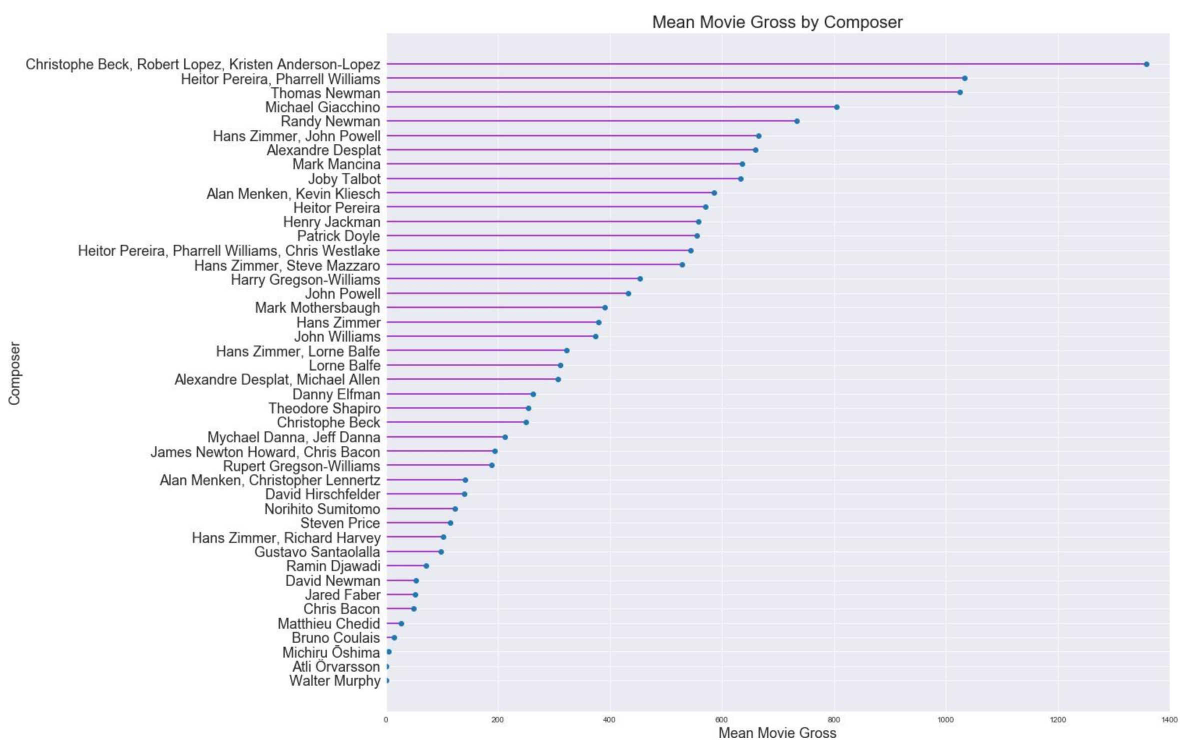
Out[167]:

	composer	median	mean
8	Christophe Beck, Robert Lopez, Kristen Anderso...	1357.67	1357.67
20	Heitor Pereira, Pharrell Williams	1032.60	1032.60
42	Thomas Newman	1025.01	1025.01
32	Michael Giacchino	826.70	804.89
38	Randy Newman	743.59	733.51
14	Hans Zimmer, John Powell	664.84	664.84
2	Alexandre Desplat	659.88	659.88
29	Mark Mancina	636.34	636.34
25	Joby Talbot	632.48	632.48
1	Alan Menken, Kevin Kliesch	585.73	585.73
19	Heitor Pereira	352.33	570.57
22	Henry Jackman	542.14	557.62
36	Patrick Doyle	554.61	554.61
21	Heitor Pereira, Pharrell Williams, Chris Westlake	543.46	543.46
17	Hans Zimmer, Steve Mazzaro	527.97	527.97
18	Harry Gregson-Williams	452.98	452.98
26	John Powell	490.18	432.12
30	Mark Mothersbaugh	423.30	390.96
13	Hans Zimmer	383.45	379.62
27	John Williams	373.99	373.99
15	Hans Zimmer, Lorne Balfe	321.89	321.89
28	Lorne Balfe	310.57	310.57
3	Alexandre Desplat, Michael Allen	306.90	306.90
9	Danny Elfman	262.79	262.79
41	Theodore Shapiro	254.25	254.25
7	Christophe Beck	250.09	250.09
34	Mychael Danna, Jeff Danna	191.64	212.32
23	James Newton Howard, Chris Bacon	193.74	193.74
39	Rupert Gregson-Williams	187.89	187.89
0	Alan Menken, Christopher Lennertz	141.34	141.34
10	David Hirschfelder	139.72	139.72
35	Norihito Sumitomo	122.75	122.75
40	Steven Price	115.12	115.12
16	Hans Zimmer, Richard Harvey	102.03	102.03

	composer	median	mean
12	Gustavo Santaolalla	97.65	97.65
37	Ramin Djawadi	71.59	71.59
11	David Newman	53.05	53.05
24	Jared Faber	51.62	51.62
6	Chris Bacon	48.96	48.96
31	Matthieu Chedid	27.00	27.00
5	Bruno Coulais	14.53	14.53
33	Michiru Ōshima	4.00	4.00
4	Atli Örvarsson	0.65	0.65
43	Walter Murphy	0.07	0.07

In [168]:

```
plt.figure(figsize=(18, 16))
ordered_df2 = composer_stats.sort_values(by='mean')
my_range=range(1,len(composer_stats.index)+1)
plt.hlines(y=my_range, xmin=0, xmax=ordered_df2['mean'], color='darkviol
et')
plt.plot(ordered_df2['mean'], my_range, "o")
plt.yticks(my_range, ordered_df2['composer'], fontsize=18)
plt.title("Mean Movie Gross by Composer", fontsize=22)
plt.xlabel('Mean Movie Gross', fontsize=18)
plt.ylabel('Composer', fontsize=18)
plt.xlim(0, 1400);
#saved in images as fig27
#plt.tight_layout()
#plt.savefig('./images/fig27.png')
```



In [169]:

```
# seeing which movies had music composed by Christophe Beck, Robert Lope
z, Kristen Anderson-Lopez
merged_df.loc[merged_df['composer']=='Christophe Beck, Robert Lopez, Kri
sten Anderson-Lopez']
```

Out[169]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	dom_
302	2013-11-22	2013	Frozen	Musical	Digital Animation	Kids Fiction	150000000	4007
903	2019-11-22	2019	Frozen II	Adventure	Digital Animation	Kids Fiction	150000000	4773

In [170]:

```
# seeing which movies had music composed by Christophe Beck, Robert Lope
z, Kristen Anderson-Lopez
merged_df.loc[merged_df['composer']=='Heitor Pereira, Pharrell Williams'
]
```

Out[170]:

	release_date	release_year	title	genre	prod_method	creative_type	budget	do
708	2017-06-30	2017	Despicable Me 3	Adventure	Digital Animation	Kids Fiction	75000000	26

As both the composer_stats DataFrame and the Mean Movie Gross by Composer plot show, for the past ten years, the top five composers (or combinations of composers) for animated movies by mean world gross are:

1. Christophe Beck, Robert Lopez, Kristen Anderson-Lopez
2. Heitor Pereira, Pharrell Williams
3. Thomas Newman
4. Michael Giacchino
5. Randy Newman

These composers have a proven track record of success and should be hired to work on an animated musical to increase box office success.

Now that I had my general answers to each of the primary business questions that were laid out at the onset of this project, it was time to combine them into an actionable plan for Microsoft's up-and-coming movie studio.

Evaluation

Microsoft's ticket to worldwide box office success can be attained by releasing a Super Hero movie in April or May, or by releasing an animated children's musical movie in June or November. My findings have shown that the more funds a studio invests in their movie production, the more money (worldwide box office gross) they are likely to receive as a result. Though there are some outliers, the majority of high-grossing movies for these particular types of movies are ones with higher budgets. For an animated movie, this amount is between 75 and 200 million dollars. For a super hero movie, that amount is between 200 and 400 million dollars. Throughout my analysis, I have found that the optimal time to release a movie depends on the type of movie that it is. While animated musical movies tend to fare very well in November and June, super hero movies have seen immense box office success in April and May. I chose to investigate a bit further and narrow down some additional attributes that may increase a movie's value, such as the highest-grossing composers and directors for the past ten years, based on mean world gross.

I am confident that the results I extrapolated from this analysis would generalize beyond the data that I have, with the exception of this year and next year due to the COVID-19 pandemic. By looking at the data up until this year, the trends and correlations I found were true for the past ten years, so I am confident that they will again be true once the world returns to some semblance of normalcy.

If the recommendations that I made are put to use, I am confident that Microsoft will have a successful break into the movie-making industry. From the data, it is clear that all the attributes I have discussed are correlated with high worldwide box office gross, which is exactly what Microsoft will want for their first movies and beyond.

Conclusion

In conclusion, I would recommend that Microsoft release one of the following two movies, each with four specific recommendations that have proven to be successful combinations:

Movie Option #1

- an animated kids fiction movie
- with a production budget of 75 to 200 million dollars
- released in June or November
- containing songs by a high-grossing composer with a track record of successful work in digital animation movies, such as Christophe Beck, Robert Lopez, and Kristen Anderson-Lopez, or Heitor Pereira and Pharrell Williams

Movie Option #2

- a live action/animation superhero movie
- with a production budget of 200 to 400 million dollars
- released in April or May
- directed by a top-grossing director with a history of proven successful superhero movies, such as Anthony Russo, Joe Russo, or Joss Whedon

While the past ten years of data show that this should be a good recipe for success, one limitation is that we are currently in a global pandemic, which has negatively affected many facets of the global economy. The visualizations above displaying movie gross over time clearly show a significant drop in movie gross for this year (2020). However, since movies take quite a bit of time to produce, the expectation is that the market will be trending in the right direction by the time a future movie is released.

In the future, this analysis could be improved by adding additional data as it becomes available. It could also be expanded upon by determining how much money there is to be made on a streaming platform movie release while theaters remain at low audience capacity.