

§ Project Overview

For my project, I utilized data from the three sources shown below to gain insight into the film industry in order to provide useful recommendations for Microsoft as they prepare to enter the streaming media market.



§ Table of Contents

- [A Gladiator Wishes to Enter the Arena](#)
- [Exploring and Preparing the Data](#)
- [Creating the Financial Analysis DataFrames](#)
- [Functions Created for Visualizations](#)
 - [Main Visualization Function](#)
- [Standards for Qualifying as Important](#)
- [Creating & Plotting the Aggregate DataFrames](#)
 - [Important Genres Analysis](#)
 - [Important Creative Types Analysis](#)
 - [Important Combinations Analysis](#)
 - [MPA Ratings Priorities](#)
- [Recommendations for Microsoft](#)
 - [Recommendations for PG-13 Content](#)
 - [Recommendations for PG Content](#)
 - [Recommendations for R Content](#)
 - [Recommendations for G Content](#)
 - [Important Subcategories & Crew Interactive Visualization Explorer](#)
- [Additional Insights](#)

§ A Gladiator Wishes to Enter the Arena



Microsoft Productions

Microsoft is developing their own streaming service that will feature content which they intend to produce in-house. As they are entering into an already crowded and competitive market, it would be advantageous for them to analyze which types of movies perform best at the box office, which would also provide insights into the viewing preferences of moviegoers, and then produce movies and/or television shows based on the insights gained from such an analysis.

For this project, I analyzed the financial performance of movies released between 2010 and 2018 based on data from the sources shown above. The data was provided to me by [Flatiron School](#) (<https://flatironschool.com/>) and [OpusData](#) (<https://www.opusdata.com/>). I explored the performance of the movies based on the category and subcategory types listed below.

- **Primary Category Type**
 - [Motion Picture Association](#) (<https://www.motionpictures.org>) (MPA) Ratings
- **Subcategory Types**
 - Genres
 - Creative Types
 - Genre & Creative Type Combinations

By using the MPA Ratings as the main categories, my analysis will allow Microsoft to target the various age and audience groups accordingly. I identified and explored the highest performing subcategories for each of the MPA Ratings, as well as the highest performing MPA Ratings themselves, based on the metrics shown below. The reason for using each of the metrics is also shown.

- **Financial Metrics**
 - **Total Worldwide Gross Revenue**
 - This metric acts as a measure of overall audience interest, as the movies with the highest gross revenues would be the ones that sold the most tickets.
 - **Total Worldwide Profits**

- This metric shows which of the categories or subcategories were associated with the largest profit margins overall, which is of obvious interest.
- **Average Worldwide Gross Revenue**
 - The metric shows which of the categories or subcategories were the most reliable in generating high levels of revenue, which again is a measure of overall audience interest.
- **Average Worldwide Profits**
 - The metric shows which of the categories or subcategories were the most reliable in generating high levels of profit, again of obvious interest.

I also determined the crew members of the types listed below that worked on the high performing films from the high performing subcategories for each of the MPA Ratings.

- **Crew Member Types**

- Directors
- Writers
- Actors
- Actresses

My analysis will be able to provide the Microsoft team assigned to lead their entry into the streaming market with strategically valuable insights into the viewing tendencies of moviegoers. These insights will guide them as to the type of content they may wish to produce to attract and retain subscribers while generating high levels of financial returns.

Of course, the insights gained through my analysis will be limited to the financial performance of **movies** with the category and subcategory types shown above. However, it is not clear if the financial performance of movies is directly correlated with the viewing habits and preferences of people using streaming services, or television viewers in general for that matter. My analysis does not contain any analysis of the viewing habits of streaming service subscribers or regular television viewers for a number of reasons. First, the other streaming services do not publish data on the viewing habits of their users. Second, the data on the [Nielsen](https://global.nielsen.com/) (<https://global.nielsen.com/>) ratings (the standard for television viewing figures that has also recently upgraded its data collection for streaming services as well) must be purchased, and I did not see any mention of a free academic dataset. Third, in the data that I was either provided or was able to easily obtain, there were no useful metrics to measure the actual performance of TV shows, only what people online thought of them.

While the lack of data on the viewing habits of streaming service subscribers or regular television viewers are potential blind spots for my analysis, the project assignment was to analyze **movies based on box office performance**, and the analysis I have performed will still provide the Microsoft team with highly useful insights that can at least serve as a starting point for their productions. They could then analyze the viewing habits of their subscribers once their streaming service was launched to adjust their production priorities accordingly if needed.

§ Exploring and Preparing the Data

For this project I used data provided to me from [IMDb](https://www.imdb.com/) (<https://www.imdb.com/>) and [The Numbers](https://www.the-numbers.com/) (<https://www.the-numbers.com/>), by [Flatiron School](https://flatironschool.com/) (<https://flatironschool.com/>). I also applied for and received an academic dataset from [OpusData](https://www.opusdata.com/) (<https://www.opusdata.com/>), which is owned and operated by the same company as The Numbers, for which it provides data services.

Importing and Investigating the Data

I imported the necessary modules and functions to explore the data, coming back and adding additional modules as needed. I also created a mapping function to handle SQL queries, and enabled inline plotting for building the visualization functions.

In [1]:

```
1 import os
2 from os import listdir
3 import string
4 import numpy as np
5 import pandas as pd
6 import sqlite3
7 #-----
8 from pandas.io.formats.style import Styler
9 #-----
10 from pandasql import sqldf
11 pysqldf = lambda q: sqldf(q, globals())
12 #-----
13 import matplotlib.pyplot as plt
14 import matplotlib.lines as mlines
15 import matplotlib.patches as mpatches
16 from matplotlib import cm
17 from matplotlib.colors import LinearSegmentedColormap
18 #-----
19 import ipywidgets as widgets
20 #-----
21 from PIL import Image
22 from IPython.display import clear_output
23 #-----
24 %matplotlib inline
```

I used `pandas` to import the data into dataframes. I then created lists of the dataframes and their names to iterate through and explore the data with ease.

In [2]:

```
1 # Flatiron School provided data from IMDb and The Numbers
2 #####
3 imdb_title_principals = pd.read_csv('data/title.principals.csv')
4 imdb_title_basics = pd.read_csv('data/title.basics.csv')
5 imdb_name_basic = pd.read_csv('data/name.basics.csv')
6 tn_movie_budgets = pd.read_csv('data/tn.movie_budgets.csv')
```

In [3]:

```
1 # OpusData data
2 #####
3 tn_extra = pd.read_csv('data/Opus Data/MovieData.csv')
```

```
In [4]: 1 df_names = ['imdb_title_principals', 'imdb_title_basics', 'imdb_name_basic', 'tn_movie_budget']
2         'tn_extra']
3 #-----
4 all_dfs = [imdb_title_principals, imdb_title_basics, imdb_name_basic, tn_movie_budget,
5 #-----
6 for name, df in zip(df_names, all_dfs):
7     print('\033[1m' + name + ':' + '\033[0m')
8     display(df.head())
9     display(df.info())
10    print('-----')
```

`imdb_title_principals:`

	tconst	ordering	nconst	category	job	characters
0	tt0111414	1	nm0246005	actor	NaN	["The Man"]
1	tt0111414	2	nm0398271	director	NaN	NaN
2	tt0111414	3	nm3739909	producer	producer	NaN
3	tt0323808	10	nm0059247	editor	NaN	NaN
4	tt0323808	1	nm3579312	actress	NaN	["Beth Boothby"]

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028186 entries, 0 to 1028185
Data columns (total 6 columns):
Column Non-Null Count Dtype
0 tconst 1028186 non-null object

I decided to use the categorical data from the OpusData data, the financial data from the The Numbers data, and the crew information from the IMDb data.

The OpusData data was best suited as a source of categorical data as it contained multiple category types to analyze. Even though the data from OpusData contained financial data, the fact that it contained a 'production_year' column instead of a 'release_date' column meant that the The Numbers data contained meant that it could not be used alone with the IMDb data, which contained a 'start_year' column, as there would be no way of knowing if I was combining data from movies from different years. I therefore decided to use the financial data from the The Numbers data.

As I previously mentioned, the crew data was used after finding the financially important subcategories within each MPA Rating. The crew members who worked on the highest performing films with those subcategories could then be identified. Microsoft will therefore be provided with all of the data necessary to begin producing their own streaming content.

Preparing the Data for Analysis

In order to perform my analysis, changes needed to be made to some of the columns within several dataframes. I first had to remove punctuation from any column with financial data that was in string form, and then change the data type of that column to a numerical data type.

In [5]:

```
1 finan_cols = ['production_budget', 'domestic_gross', 'worldwide_gross']
2 #-----
3 for col in finan_cols:
4     tn_movie_budgets[col] = tn_movie_budgets[col].str.strip('$')
5     tn_movie_budgets[col] = tn_movie_budgets[col].str.replace(',', '')
6 #-----
7 tn_movie_budgets[finan_cols] = tn_movie_budgets[finan_cols].astype(float)
8 #-----
9 display(tn_movie_budgets.head(3))
10 display(tn_movie_budgets.info())
```

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	425000000.0	760507625.0	2.776345e+09
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000.0	241063875.0	1.045664e+09
2	3	Jun 7, 2019	Dark Phoenix	3500000000.0	42762350.0	1.497624e+08


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   id               5782 non-null    int64  
 1   release_date     5782 non-null    object  
 2   movie            5782 non-null    object  
 3   production_budget 5782 non-null    float64
 4   domestic_gross   5782 non-null    float64
 5   worldwide_gross  5782 non-null    float64
```

I then could add other financial columns based on those columns that could be of significance to my analysis.

```
In [6]: 1 foreign_gross = tn_movie_budgets['worldwide_gross'] - tn_movie_budgets['domestic_gr
2 #-----
3 tn_movie_budgets.insert(5, 'foreign_gross', foreign_gross)
4 #-----
5 tn_movie_budgets['worldwide_profits'] = tn_movie_budgets['worldwide_gross'] - \
6 tn_movie_budgets['production_budget']
7 #-----
8 display(tn_movie_budgets.head())
9 display(tn_movie_budgets.info())
```

	id	release_date	movie	production_budget	domestic_gross	foreign_gross	worldwide_gross	wo
0	1	Dec 18, 2009	Avatar	425000000.0	760507625.0	2.015838e+09	2.776345e+09	
			Pirates of the Caribbean: On Stranger Tides					
1	2	May 20, 2011		410600000.0	241063875.0	8.046000e+08	1.045664e+09	
2	3	Jun 7, 2019	Dark Phoenix	350000000.0	42762350.0	1.070000e+08	1.497624e+08	
3	4	May 1, 2015	Avengers: Age of Ultron	330600000.0	459005868.0	9.440081e+08	1.403014e+09	
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last .Jedi	317000000.0	620181382.0	6.965404e+08	1.316722e+09	

I also changed any columns containing any date or year data to the datetime format so that they could be easily compared and used to control the period of time covered by my analysis.

```
In [7]: 1 tn_movie_budgets.release_date = \
2 pd.to_datetime(tn_movie_budgets.release_date).map(lambda x: x.strftime("%m/%d/%Y"))
3 #-----
4 tn_movie_budgets.release_date = \
5 pd.to_datetime(tn_movie_budgets.release_date, format="%m/%d/%Y")
6 #-----
7 display(tn_movie_budgets.release_date.head(3))
8 print(tn_movie_budgets.release_date.dt.year.min(), tn_movie_budgets.release_date.dt.
9 print(len(tn_movie_budgets))
```

0	2009-12-18
1	2011-05-20
2	2019-06-07

Name: release_date, dtype: datetime64[ns]

1915 2020
5782

Finally, I rearranged the column order of one of the dataframes to make it ready for database construction.

```
In [8]: 1 tn_cols = [tn_extra.columns[2]] + list(tn_extra.columns[:2]) + list(tn_extra.columns[-2:])
2 tn_extra = tn_extra[tn_cols]
3 #-----
4 tn_extra.head(3)
```

Out[8]:

	movie_odid	movie_name	production_year	production_budget	domestic_box_office	international_box_office
0	8220100	Madea's Family Reunion	2006	10000000	63257940	62
1	58540100	Krrish	2006	10000000	1430721	31000
2	34620100	End of the Spear	2006	10000000	11748661	175

Limits the Year Range

The data from OpusData, which contained the categorical data that I would analyze for performance, only contained data from 2006 to 2018. I felt that going back to 2006 was unnecessary and that by limiting the year range to the past decade, my analysis would be based on the most relevant data in terms of the current viewing habits of moviegoers. It would have been advantageous to obtain data for 2019 as well, but I would have excluded any data from the past two years anyway due to the disruptions to the industry caused by COVID-19 so the loss of data, and with it any potentially valuable insights, is minimal.

```
In [9]: 1 imdb_title_basics = imdb_title_basics.loc[(imdb_title_basics.start_year > 2009) &
2                                         (imdb_title_basics.start_year < 2019)]
3 #-----
4 tn_movie_budgets = tn_movie_budgets.loc[(tn_movie_budgets.release_date.dt.year > 2009) &
5                                         (tn_movie_budgets.release_date.dt.year < 2019)]
```

Merging The Numbers Data with OpusData Data

I found it easier to merge the data from The Numbers and OpusData together, and deal with any errors that might occur, prior to creating the SQL database, rather than after. It also made sense to combine them first since they are essentially the same data source.

In [10]:

```

1 tn_p1 = tn_movie_budgets.set_index('movie')
2 tn_p2 = tn_extra.set_index('movie_name')
3 #-----
4 tn_full = \
5 tn_p1.merge(tn_p2, how='outer', left_index=True, right_index=True)\ \
6     .reset_index().set_index('id').reset_index()
7 #-----
8 tn_full = tn_full.rename({'index': 'movie'}, axis=1)
9 #-----
10 tn_full = \
11 tn_full.drop(tn_full.loc[(tn_full.production_budget_x != tn_full.production_budget_ \
12                         (tn_full.release_date.dt.year != 2019)].index)
13 #-----
14 tn_full = \
15 tn_full.drop(tn_full.loc[(tn_full.domestic_gross != tn_full.domestic_box_office) & \
16                         (tn_full.foreign_gross != tn_full.international_box_office) & \
17                         (tn_full.release_date.dt.year != 2019)].index)
18 #-----
19 cols_to_drop = ['production_year', 'movie_odid', 'production_budget_y', 'domestic_b \
20                  'international_box_office', 'sequel', 'running_time']
21 #-----
22 tn_full = tn_full.drop(columns=cols_to_drop)\ \
23     .rename({'production_budget_x': 'production_budget'}, axis=1)
24 #-----
25 display(tn_full.head(3))
26 display(tn_full.info())

```

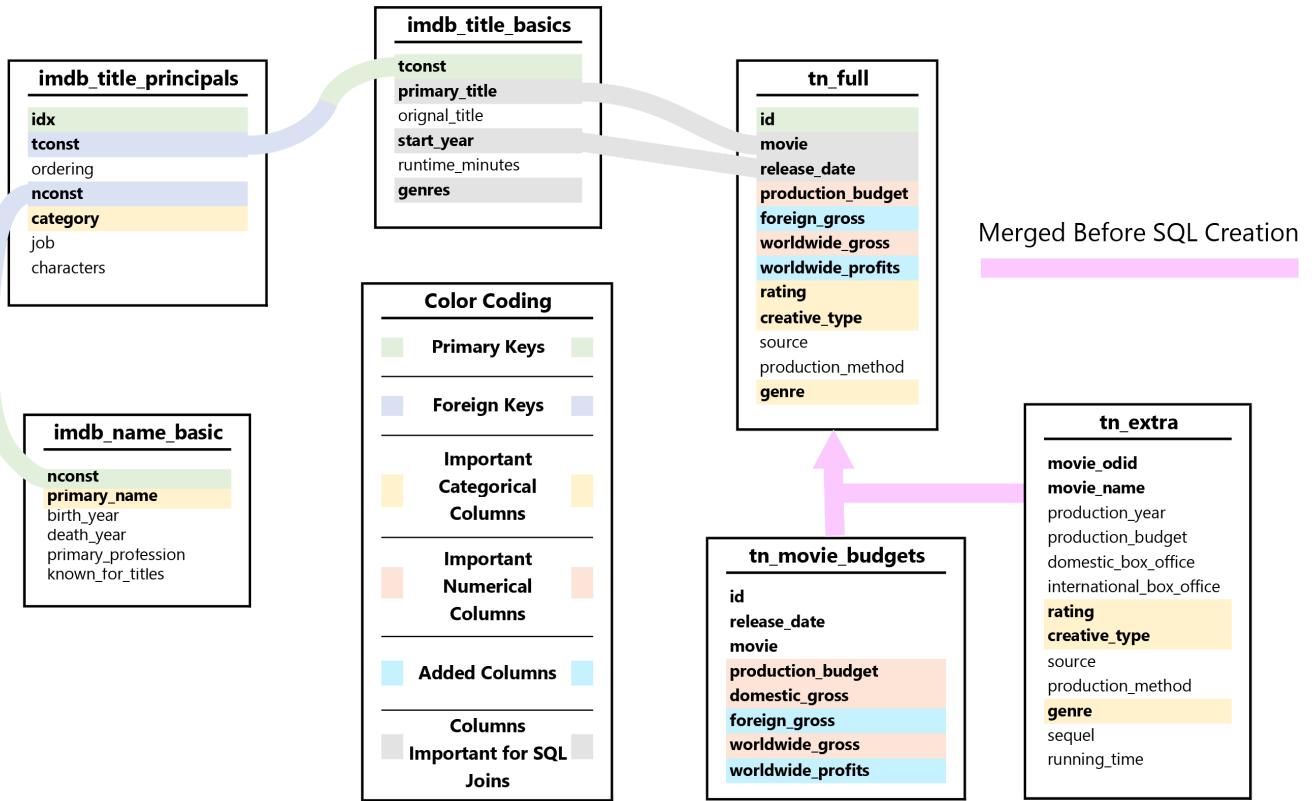
	<u>id</u>	<u>movie</u>	<u>release_date</u>	<u>production_budget</u>	<u>domestic_gross</u>	<u>foreign_gross</u>	<u>worldwide_gross</u>	<u>year</u>
2	48.0	10 Days in a Madhouse	2015-11-11	12000000.0	14616.0	0.0	14616.0	2015
5	64.0	12 Strong	2018-01-19	35000000.0	45819713.0	25298665.0	71118378.0	2018
6	18.0	12 Years a Slave	2013-10-18	20000000.0	56671993.0	124353350.0	181025343.0	2013

Turning the DataFrames into a SQL Database

I chose to use the dataframes shown in the image below to create an SQL database for my analysis as it would allow me to create new dataframes with the desired data with ease. The image also shows how I used the database to take different columns from the dataframes and join them together to build the

new dataframes. The important columns for either joining or analyzing are color coded. Any new columns that I created, based on the data already in the corresponding table, to enhance my analysis, were also given an identifying color.

As previously discussed, I combined the `tn_movie_budgets` and `tn_extra` dataframes into the `tn_full` dataframe prior to creating the SQL database.



In order to create the database, I first updated the lists of the dataframes and their names to iterate through to create the database.

```
In [11]: 1 df_names = ['imdb_title_principals', 'imdb_title_basics', 'imdb_name_basic', 'tn_fu
2 #-----
3 all_dfs = [imdb_title_principals, imdb_title_basics, imdb_name_basic, tn_full]
```

I then created a new database and a connection to it via the `sqlite3` module.

```
In [12]: 1 mov_conn = sqlite3.Connection('movies_db.sqlite')
2 mov_cur = mov_conn.cursor()
```

I could then populate the database with the dataframes I needed. If a dataframe did not have a unique identifier column, I reset the index to create one.

```
In [13]: 1 for t_i, (table, table_name) in enumerate(zip(all_dfs, df_names)):
2     if table[table.columns[0]].duplicated().any():
3         table_copy = table.reset_index().rename({'index':'idx'}, axis=1)
4     else: table_copy = table.copy()
5     #
6     table_copy.to_sql(table_name, mov_conn, if_exists='replace', index=False)
```

I then checked to make sure everything worked.

```
In [14]: 1 print(mov_cur.execute('SELECT name FROM sqlite_master').fetchall())
[('imdb_title_principals',), ('imdb_title_basics',), ('imdb_name_basic',), ('tn_ful
l',)]
```

```
In [15]: 1 for table, table_name in zip(all_dfs, df_names):
2     df = pd.DataFrame(mov_cur.execute('SELECT * FROM ' + table_name + ';').fetchall())
3     df.columns = [x[0] for x in mov_cur.description]
4     #-----
5     print('\033[1m' + table_name + ':' + '\033[0m')
6     display(df.head(3))
7     display(df.info())
8     print('-----')
```

imdb_title_principals:

	idx	tconst	ordering	nconst	category	job	characters
0	0	tt0111414		1 nm0246005	actor	None	["The Man"]
1	1	tt0111414		2 nm0398271	director	None	None
2	2	tt0111414		3 nm3739909	producer	producer	None

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028186 entries, 0 to 1028185
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   idx         1028186 non-null   int64  
 1   tconst      1028186 non-null   object 
 2   ordering    1028186 non-null   int64  
 3   nconst      1028186 non-null   object 
```

§ Creating the Financial Analysis DataFrames

I created five different financial dataframes that I would use to create my aggregate dataframes. For each of the dataframes, I would first create the dataframe with a SQL query and then use `pandas` to manipulate it. Although I decided to use the categorical data from the OpusData data, I found the '`genres`' column from the `imdb_title_basics` dataframe useful in weeding out data that was mistakenly duplicated by being joined to multiple titles and I did not want to simply drop the titles with the same name from the same year. The more data I could retain the more value my analysis would have.

Main Financial DataFrame

The SQL query I used to create the main financial dataframe is shown below.

```
"""
SELECT
    tconst, start_year, primary_title, genre, genres, rating, creative_type,
    worldwide_gross, worldwide_profits
FROM tn_full
JOIN imdb_title_basics
    ON (primary_title = movie AND (release_date LIKE '%' || start_year || '%'))
WHERE genres NOT Null;"""

```

In [16]:

```
1 db_query = """
2 SELECT
3     tconst, start_year, primary_title, genre, genres, rating, creative_type,
4     production_budget, worldwide_gross, worldwide_profits
5 FROM tn_full
6 JOIN imdb_title_basics
7     ON (primary_title = movie AND (release_date LIKE '%' || start_year || '%'))
8 WHERE genres NOT Null;"""
9 #-----
10 fin_df = pysqldf(db_query)
11 #-----
12 display(fin_df.head(3))
13 display(fin_df.info())
14 #-----
15 fin_df = fin_df.drop_duplicates(['tconst', 'start_year', 'primary_title'], keep=False)
16 #-----
17 display(fin_df.info())
```

	tconst	start_year	primary_title	genre	genres	rating	creative_type	production
0	tt0249516	2012	Foodfight!	Adventure	Action,Animation,Comedy	PG	Kids Fiction	4
1	tt0359950	2013	The Secret Life of Walter Mitty	Adventure	Adventure,Comedy,Drama	PG	Contemporary Fiction	9
2	tt0365907	2014	A Walk Among the Tombstones	Action	Action,Crime,Drama	R	Historical Fiction	2

In [17]:

```
1 dup_titles = \
2 fin_df.loc[fin_df.duplicated(['start_year', 'primary_title', 'worldwide_gross'], keep='first')
3         .primary_title.unique())
4 #-----
5 drop_idx = []
6 #-----
7 for tit in dup_titles:
8     tit_df = fin_df.loc[fin_df.primary_title == tit]
9     #
10    if len(tit_df.genres.unique()) > 1:
11        drop_idx.extend(tit_df.index)
12    #
13 fin_df = fin_df.drop(drop_idx)
14 #
15 display(fin_df.head(3))
16 display(fin_df.info())
17 print('Movies dropped: ', len(drop_idx))
```

	tconst	start_year	primary_title	genre	genres	rating	creative_type	production_companies
0	tt0249516	2012	Foodfight!	Adventure	Action,Animation,Comedy	PG	Kids Fiction	2
1	tt0359950	2013	The Secret Life of Walter Mitty	Adventure	Adventure,Comedy,Drama	PG	Contemporary Fiction	9
2	tt0365907	2014	A Walk Among the Tombstones	Action	Action,Crime,Drama	R	Historical Fiction	2


```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 975 entries, 0 to 1032
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   tconst          975 non-null    object 
 1   start_year      975 non-null    int64  
 2   primary_title   975 non-null    object 
 3   genres          975 non-null    object 
 4   rating          975 non-null    object 
 5   creative_type   975 non-null    object 
 6   production_companies 975 non-null  object 
 7   runtime          975 non-null    int64  
 8   start_month     975 non-null    int64  
 9   end_month       975 non-null    int64 
```

Main Financial DataFrame Details

I created tables in markdown format to be displayed in the notebook that show the details of the main category type and each subcategory type. To do so, I first created a function to format the numerical values accordingly.

In [18]:

```
1 def table_currency(val):
2     dollar_start = '&dollar; '
3     #
4     if val >= 1e9 or val <= -1e9:
5         v = dollar_start + '{:.2f}&nbsp;B'.format(abs(val*1e-9))
6     if (1e6 <= val < 1e9) or (-1e6 >= val > -1e9):
7         v = dollar_start + '{:.2f}&nbsp;M'.format(abs(val*1e-6))
8     if (1e3 <= val < 1e6) or (-1e3 >= val > -1e6):
9         v = dollar_start + '{:.2f}&nbsp;Th'.format(abs(val*1e-3))
10    #
11    if 0 < val < 1e3: v = dollar_start + '{:.2f}'.format(abs(val))
12    if val == 0: v = 0
13    if val < 0: v = '- ' + v
14    if np.isnan(val): v = ''
15    #
16    return v
```

In [19]:

```

1 md_table_start = '| '
2 md_table_mid = '| '
3 md_table_end = '| \n'
4 #-
5 cat_cols = ['rating', 'genre', 'creative_type', ['genre', 'creative_type']]
6 #-
7 main_header_str = '| Main Category Type | Number of Categories |\n'
8 header_str = '| Subcategory Types | Number of Subcategories |\n'
9 #-
10 align_cols = '| :-: | :-: |\n'
11 #-
12 for c_i, cat_col in enumerate(cat_cols):
13     if c_i in [0, 1]: cat_str = cat_col.title() + 's'
14     if c_i == 2: cat_str = ' '.join(cat_col.split('_')).title() + 's'
15     #
16     if c_i == 3:
17         cat_str = 'Genre & Creative Type<br>Combinations'
18     #
19     cat_col_1 = cat_col[0]
20     cat_col_2 = cat_col[1]
21     #
22     cat_df = fin_df[[cat_col_1, cat_col_2]].copy()
23     cat_df['combos'] = cat_df[cat_col_1] + ', ' + cat_df[cat_col_2]
24     #
25 else:
26     value_df = fin_df[cat_col].value_counts()
27     #
28     value_df.name = cat_str
29     #
30     value_df = value_df.reset_index().rename({cat_str: 'Count', 'index': cat_str})
31     .set_index(cat_str)
32     #
33     all_align_cols = ['center'] * (len(value_df.columns) + 1)
34     #
35     print(value_df.to_markdown(tablefmt='pipe', colalign=all_align_cols))
36     print('\n\n\n')
37     #
38     cat_num = str(fin_df[cat_col].nunique())
39     if c_i == 3: cat_num = str(cat_df.combos.nunique())
40     #
41     md_table_str = md_table_start + cat_str + md_table_mid + cat_num + md_table_end
42     #
43     if c_i == 0:
44         table = main_header_str + align_cols + md_table_str + '\n'
45         print(table)
46     #
47     if c_i == 1: table = header_str + align_cols + md_table_str
48     if c_i > 1: table = table + md_table_str
49     #
50 print(table)

```

Ratings	Count
PG-13	411
R	382
PG	161
G	11

Not Rated	10
-----------	----

Main Category Type	Number of Categories
-:-	-:-
Ratings	5

Genres	Count
Drama	219

In [20]:

```

1 for cat_col in cat_cols[:-1]:
2     for col in ['worldwide_gross', 'worldwide_profits']:
3         desc_df = fin_df.groupby(cat_col).describe()[[col]]
4         #-
5         temp_df_p1 = desc_df.iloc[:, :1].astype(int)
6         temp_df_p2 = desc_df.iloc[:, 1:1].applymap(lambda x: table_currency(x))
7         #-
8         desc_df = pd.concat([temp_df_p1, temp_df_p2], axis=1)
9         #-
10        col_tit = ' '.join(col.split('_')).title()
11        #-
12        desc_df.index.name = col_tit + '<br><br>' + desc_df.index.name.title()
13        desc_df = desc_df.rename(columns={col: col_tit}, level=0)
14        #-
15        desc_df.columns = desc_df.columns.to_flat_index()
16        #-
17        desc_df.columns = [col[1] for col in desc_df.columns]
18        #-
19        all_align_cols = ['center'] * (len(desc_df.columns) + 1)
20        #-
21        print(desc_df.to_markdown(tablefmt='pipe', colalign=all_align_cols))
22        print('\n\n\n')

```

Worldwide Gross Rating	count	mean
std	min	25%
50%	75%	max
G	11	$369.03&nbspM &dolla r;&nbsp331.65&nbspM $&nbsp27.47&nbspM $&nbsp74.59&nbspM $&nbsp349.09&nbspM $&nbsp526.50&nbspM $&nbsp1.07&nbs p;B
Not Rated	10	$&nbsp54.75&nbspM &dolla r;&nbsp82.57&nbspM $&nbsp247.56&nbspTh $&nbsp10.77&nbspM $&nbsp19.81&nbspM $&nbsp61.43&nbspM $&nbsp263.50&nbs p;M
PG	161	$&nbsp282.71&nbspM &dolla r;&nbsp288.30&nbspM $&nbsp73.71&nbspTh $&nbsp77.23&nbspM $&nbsp183.35&nbspM $&nbsp373.99&nbspM $&nbsp1.27&nbs p;B

The table below shows the number of categories within the main category type.

Main Category Type Number of Categories

Ratings	5
---------	---

The table below shows the number of subcategories within each subcategory type.

Subcategory Types	Number of Subcategories
Genres	11
Creative Types	9
Genre & Creative Type Combinations	56

Click Here to see more details for the Rating category type and Genre and Creative Type subcategory types.

Click Here to see the explore the Rating category type.

Click Here to see the explore the Genre subcategory type.

Click Here to see the explore the Creative Type subcategory type.

Crew Financial DataFrames

I inserted the different job titles into the SQL query shown below as I iterated through them to create the financial dataframe for each of the previously mentioned crew types.

```
"""
SELECT
    tconst, nconst, primary_name, start_year, primary_title, genre, genres, rating,
    creative_type, worldwide_gross, worldwide_profits
FROM imdb_title_basics
JOIN tn_full
    ON (primary_title = movie AND (release_date LIKE '%' || start_year || '%'))
JOIN imdb_title_principals
    USING(tconst)
JOIN imdb_name_basic AS nb
    USING(nconst)
WHERE genres NOT Null AND category = '"""+ crew_str +"""
ORDER BY start_year, primary_title;"""
```

In [21]:

```

1 crew_strings = ['director', 'writer', 'actor', 'actress']
2 #-----
3 for crew_str in crew_strings:
4     print('\033[1m\033[4m'+ crew_str.title() + ' DataFrame:\033[0m')
5     print('-----')
6 #
7 db_query = """
8     SELECT
9         tconst, nconst, primary_name, start_year, primary_title, genre, genres, rat
10        creative_type, production_budget, worldwide_gross, worldwide_profits
11    FROM imdb_title_basics
12   JOIN tn_full
13      ON (primary_title = movie AND (release_date LIKE '%' || start_year || '%'))
14   JOIN imdb_title_principals
15      USING(tconst)
16   JOIN imdb_name_basic AS nb
17      USING(nconst)
18 WHERE genres NOT Null AND category = '"""+ crew_str +"""
19 ORDER BY start_year, primary_title;"""
20 #
21 crew_df = pysqldf(db_query)
22 #
23 crew_df = crew_df.drop_duplicates(['tconst', 'nconst', 'primary_title'])
24 #
25 print('\033[1m\033[4mBEFORE DUP CHECK:\033[0m')
26 display(crew_df.info())
27
28 # Checking for duplicated data on a deeper level, trying to save as much data as
29 ######
30 dup_titles = \
31     crew_df.loc[crew_df.duplicated(['start_year', 'primary_title'], keep=False)]\
32     .primary_title.unique()
33 #
34 drop_idx = []
35 #
36 for tit in dup_titles:
37     tit_df = crew_df.loc[crew_df.primary_title == tit]
38     #
39     if len(tit_df.genres.unique()) > 1:
40         drop_idx.extend(tit_df.index)
41 #
42 crew_df = crew_df.drop(drop_idx)
43 #
44 print('\033[1m\033[4mAFTER DUP CHECK:\033[0m')
45 display(crew_df.info())
46 print('\033[1mMovies dropped:\033[0m', len(drop_idx))
47 print('-----')
48
49 # Changing each df name to an appropriate one
50 ######
51 if crew_str=='director': dir_df = crew_df
52 if crew_str=='writer': writ_df = crew_df
53 if crew_str=='actor': actor_df = crew_df
54 if crew_str=='actress': actress_df = crew_df
55
56 # List of all crew dataframes
57 ######
58 crew_dfs = [dir_df, writ_df, actor_df, actress_df]

```

```
#   Column      Non-Null Count   Dtype  
---  --          -----          ---  
0   tconst      1095 non-null    object 
1   nconst      1095 non-null    object 
2   primary_name 1095 non-null    object 
3   start_year   1095 non-null    int64  
4   primary_title 1095 non-null    object 
5   genre        1095 non-null    object 
6   genres       1095 non-null    object 
7   rating       1095 non-null    object 
8   creative_type 1092 non-null    object 
9   production_budget 1095 non-null    float64 
10  worldwide_gross 1095 non-null    float64 
11  worldwide_profits 1095 non-null    float64 
dtypes: float64(3), int64(1), object(8) 
memory usage: 111.2+ KB
```

None

AFTER DUP CHECK:

Crew Financial DataFrames Details

I created more tables in markdown format for the details on the crew member types.

In [22]:

```
1 md_table_start = '| | '
2 md_table_mid = '| | '
3 md_table_end = '| \\n'
4 #-
5 align_cols = '| :-: | :-: | :-: | \\n'
6 #-
7 type_head = 'Crew Member Types'
8 mov_head = 'Number of Movies<br>with data for each<br>Crew Member Type'
9 crew_head = 'Number of Crew Members<br>for each<br>Crew Member Type'
10 #-
11 header_str = \
12 md_table_start + type_head + md_table_mid + mov_head + md_table_mid + crew_head + n
13 #-
14 for c_df, crew_str in zip(crew_dfs, crew_strings):
15     crew_orig = crew_str
16     #
17     if crew_str == 'actress': crew_tit = crew_str + 'es'
18     else: crew_tit = crew_str + 's'
19     #
20     crew_tit = crew_tit.title()
21     #
22     num_titles = str(c_df.tconst.nunique())
23     num_crew = str(c_df.nconst.nunique())
24     #
25     md_table_str = md_table_start + crew_tit + md_table_mid + num_titles + md_table_
26             num_crew + md_table_end
27     #
28     if crew_orig == crew_strings[0]: table = header_str + align_cols + md_table_str
29     else: table = table + md_table_str
30 #
31 print(table)
```

Crew Member Types	Number of Movies with data for each Crew Member Type	Number of Crew Members for each Crew Member Type
Actresses	855	700
Actors	951	1096
Writers	882	1491
Directors	954	678

The table below shows the number of movies that had data for each crew member type, as well as the number of crew members that were in each each crew dataframe.

Crew Member Types	Number of Movies with data for each Crew Member Type	Number of Crew Members for each Crew Member Type
Directors	954	678
Writers	882	1491
Actors	951	1096
Actresses	855	700

I closed the SQL query after it was no longer needed. As this was an academic project meant to be run from start to finish, there was no reason to commit and save the changes to the database.

```
In [23]: 1 mov_conn.close()
```

§ Visualization Functions

In this section, I created and stored all of the functions I wrote that were necessary to create my visualizations. I would come back later and modify them as needed.

Modifying Key rcParams

I first made some important changes to the default `rcParams` settings in `matplotlib`. I did this so that I could use, what in my opinion were, more aesthetically appealing fonts. I also created lists of the spine sides to easily change their appearance later.

```
In [24]: 1 light_spine_sides = ['top', 'bottom']
2 dark_spine_sides = ['left', 'right']
3
4 # I had to include the translation package b/c one director's name had a non-recogn
5 # character that caused problems when converting to LaTeX
6 #####
7 plt.rcParams.update({'text.usetex': True,
8                     'text.latex.preamble': r'\usepackage{ucs}+\\
9                               r'\usepackage[T1]{fontenc}+\\
10                             r'\usepackage[utf8]{inputenc}+\\
11                               r'\usepackage[icelandic,english]{babel}\
12                                 r'\usepackage{stix}+\\
13                                   r'\usepackage{amsmath}',\
14                     'font.family': ['sans-serif', 'serif'],
15                     'font.sans-serif': ['Verdana'],
16                     'font.serif': ['New Century Schoolbook'],
17                     'mathtext.fontset': 'cm'})
18
19 # Uncomment to check and/or change keys
20 #####
21 # plt.rcParams.keys()
```

Pastel Color Maker

This function allowed me to take one of the hundreds of colors to choose from by name, and turn it into a pastel of itself, so that when that color would be covering large patches, they would not be overwhelming to the viewer. I could also still use the original colors for any lines or markers, as I would want them to stand out as much as possible.

The function returns a color between a given color and pure white based on a number between 0 and 1. The higher the number, the lighter the color.

```
In [25]: 1 def get_lighter_color(color_name, light_frac):
2     color_map = LinearSegmentedColormap.from_list('', [color_name, 'w'], N=9)
3     return color_map(light_frac)
```

Color Dictionary for MPA Ratings

I created a dictionary with a unique color for each of the the MPA ratings to use for their corresponding visualizations.

```
In [26]: 1 uniq_ratings = tn_full.rating.unique()
2 #-----
3 ratings_colors_dict = {'Not Rated': 'palegreen',
4                         'G': 'orchid',
5                         'PG': 'dodgerblue',
6                         'PG-13': 'forestgreen',
7                         'R': 'tomato'}
```

Tick Params Functions

These functions were used to control details about the grid, ticks, tick labels, and spines for the main plotting axes, or a twinned axes, depending on what kind of visualization I needed to create.

In [27]:

```
1 # Function for setting Tick Params for axes with numerical data on X-Axis and categ
2 # the Y-Axis
3 #####
4 def h_tick_params(h_ax, box=None):
5     grid_kw = dict(color='dimgrey', linewidth=3, zorder=0)
6     maj_gr_kw = dict(length=12, width=2.4, labelsize=25.5, labelcolor='k', color=(0,0,0,.63))
7     min_gr_kw = dict(length=9, width=1.2, color=(0,0,0,.63))
8     #
9     maj_reg_kw = dict(length=0, width=2.4, labelsize=22.5, labelcolor='k', color=(0,0,0,.63))
10    #
11    light_spine_kw = dict(color='dimgrey', linewidth=3, zorder=9, alpha=.09)
12    dark_spine_kw = dict(color='k', lw=2.4, alpha=.63, zorder=9)
13    #
14    h_ax.minorticks_on()
15    h_ax.grid(True, which='major', axis='x', alpha=.09, **grid_kw)
16    #
17    h_ax.tick_params('x', **maj_gr_kw)
18    h_ax.tick_params('x', which='minor', **min_gr_kw)
19    #
20    h_ax.tick_params('y', **maj_reg_kw)
21    h_ax.tick_params('y', which='minor', left=False)
22    #
23    if not box:
24        [h_ax.spines[side].update(light_spine_kw) for side in light_spine_sides]
25        [h_ax.spines[side].update(dark_spine_kw) for side in dark_spine_sides]
26    if box:
27        h_ax.spines['top'].update(light_spine_kw)
28        h_ax.spines['right'].update(light_spine_kw)
29        h_ax.spines['left'].update(dark_spine_kw)
30        h_ax.spines['bottom'].update(dark_spine_kw)
```

In [28]:

```
1 # Function for setting Tick Params for axes with a twinned numerical X-Axis on the
2 #####
3 def top_x_tick_params(t_ax):
4     grid_kw = dict(color='dimgrey', linewidth=3, zorder=0)
5     maj_gr_kw = dict(length=12, width=2.4, labelsize=25.5, labelcolor='k', color=(0,0,0,.63))
6     min_gr_kw = dict(length=9, width=1.2, color=(0,0,0,.63))
7     #
8     t_ax.minorticks_on()
9     #
10    t_ax.tick_params('x', **maj_gr_kw)
11    t_ax.tick_params('x', which='minor', **min_gr_kw)
12    #
13    t_ax.tick_params('y', which='both', labelleft=False, left=False)
14    #
15    [t_ax.spines[side].set_visible(False) for side in light_spine_sides]
16    [t_ax.spines[side].set_visible(False) for side in dark_spine_sides]
```

In [29]:

```
1 # Function for setting Tick Params for axes with a twinned categorical Y-Axis on the right side
2 ##### right_y_tick_params #####
3 def right_y_tick_params(r_ax):
4     maj_reg_kw = dict(length=0, width=2.4, labelsize=24, labelcolor='k', color=(0,0,0))
5     #
6     r_ax.tick_params('y', **maj_reg_kw)
7     r_ax.tick_params('y', which='minor', right=False)
8     #
9     r_ax.tick_params('x', which='both', labelbottom=False, bottom=False)
10    #
11    [r_ax.spines[side].set_visible(False) for side in light_spine_sides]
12    [r_ax.spines[side].set_visible(False) for side in dark_spine_sides]
```

Tick Formatters and Label Creators

These functions were used to either format ticks or create tick label lists that would be aesthetically appealing and in the right format, as well as functional so that large numbers would not overlap.

In [30]:

```

1 # Function to format currency ticks on the X Axis
2 #####
3 def x_currency_formatter(x, pos):
4     x_prime = x.copy()
5     s_start = r'{\fontsize{25.5}{0}\selectfont{\scshape{\$}}}\,'
6     #
7     if x>=1e9 or x<=-1e9:
8         s_end = r'{\fontsize{24}{0}\selectfont{\scshape{B}}}''
9         #
10        if x>=1e9: x = x*1e-9
11        if x<=-1e9: x = x*-1e-9
12        #
13    if 1e6<=x<1e9 or -1e6>=x>-1e9:
14        s_end = r'{\fontsize{24}{0}\selectfont{\scshape{M}}}''
15        #
16        if 1e6<=x<1e9: x = x*1e-6
17        if -1e6>=x>-1e9: x = x*-1e-6
18        #
19    if 1e3<=x<1e6 or -1e3>=x>-1e6:
20        s_end = r'{\fontsize{24}{0}\selectfont{\scshape{K}}}''
21        #
22        if 1e3<=x<1e6: x = x*1e-3
23        if -1e3>=x>-1e6: x = x*-1e-3
24        #
25    if round(x, 2) == int(x): s = s_start + r'$\mathbf{+''{:.0f}}'.format(x)+}''
26        #
27    elif round(x, 2) == round(x, 1): s = s_start + r'$\mathbf{+''{:.1f}}'.format(x)+}''
28        #
29    else: s = s_start + r'$\mathbf{+''{:.2f}}'.format(x)+}''
30        #
31    if x_prime<-1e3: s = r'$\mathbf{-}'+ s + '\,' + s_end
32    if x_prime>1e3: s = s + '\,' + s_end
33        #
34    if x_prime==0: s = r'${\mathbf{+str(0)}}$'
35        #
36    return s

```

In [31]:

```

1 # Function to format percentage ticks on an X Axis
2 #####
3 def x_percentage_formatter(x, pos):
4     s_end = r'{\fontsize{24}{0}\selectfont{\boldsymbol{\%}}}''
5     #
6     x_prime = x.copy()
7     x = x*1e2
8     #
9     if round(x, 2) == int(x): s = r'$\mathbf{+''{:.0f}}'.format(x)+}' + s_end
10    #
11    elif round(x, 2) == round(x, 1): s = r'$\mathbf{+''{:.1f}}'.format(x)+}' + s_end
12    #
13    else: s = r'$\mathbf{+''{:.2f}}'.format(x)+}' + s_end
14    #
15    if x_prime==0: s = r'${\mathbf{+str(0)}}$'
16    #
17    return s

```

```
In [32]: 1 # Function to format simple numerical tick labels in LaTeX format for use on ticks
2 # where simple formatting wouldn't work
3 #####
4 def y_labels_to_norm_num(labels):
5     tex_num_labels = []
6     #
7     for y in labels:
8         s = r'${\mathbf{'+str(int(y))+'}}$'
9         #
10        tex_num_labels.append(s)
11    #
12    return tex_num_labels
```

Minimum and Maximum Quantile & Tick Finders

These functions were created to find the first possible quantile corresponding to a positive value, based on a set incremental step value, so that matching values (in quantile terms) could be used to find appropriate minimum and maximum tick values for an axes with a twinned axes that needed their scaling to match. If there was no twinned axes, then they were still used as needed to find appropriate minimum and maximum tick values for the axes being plotted. Sometimes no minimum values were necessary, but it was easier to just include that feature in the quantile finder.

```
In [33]: 1 # Function to find positive and/or negative quantiles to be matched for a twinned axis
2 # passed into the max tick finder
3 #####
4 def find_pos_quantile(ser):
5     test_value = -1
6     test_quant = .06
7     last_neg = 0
8     i = 0
9     while test_value <= 0:
10         test_value = ser.quantile(test_quant)
11         #
12         if test_value < 0: last_neg = test_quant
13         #
14         if i < 3: test_quant = test_quant + .06
15         elif 3 <= i < 6: test_quant = test_quant + .06 + .06
16         else: test_quant = test_quant + .06 + .06 + .06
17         i += 1
18     #
19     return test_value, test_quant, last_neg
```

In [34]:

```
1 # Function to find a min for data ticks
2 #####
3 def find_min(new_min, min_step):
4     min_found = False
5     test_v = min_step
6     i = 0
7     #
8     while not min_found:
9         #
10        if new_min >= int(test_v):
11            final_min = int(test_v)
12            min_found = True
13        #
14        if i < 3: test_v = test_v + min_step
15        elif 3 <= i < 6: test_v = test_v + min_step*2
16        else: test_v = test_v + min_step*(i - 3)
17        i += 1
18    #
19    return final_min
```

In [35]:

```
1 # Function to find a max for data ticks
2 #####
3 def find_max(new_max, max_step):
4     max_found = False
5     test_v = max_step
6     i = 0
7     #
8     while not max_found:
9         if new_max > 1:
10            if new_max <= int(test_v):
11                final_max = int(test_v)
12                max_found = True
13            #
14            else:
15                if new_max <= test_v:
16                    final_max = test_v
17                    max_found = True
18                #
19                if i < 3: test_v = test_v + max_step
20                elif 3 <= i < 6: test_v = test_v + max_step + max_step
21                else: test_v = test_v + max_step + max_step + max_step
22                i += 1
23    #
24    return final_max
```

Main Visualization Function

I used the function below to create all of my visualizations. As such, there is a lot of information here. It may seem dense and overwhelming at first, but I think that with the aid of the code comments, it should be understandable. I admit, however, that one might have to view the whole function before realizing just how everything fits together.

I wanted to run all plotting through one function to keep a similar format and style without having to rewrite a slightly different function over and over again. I designed the function so that simple strings could be passed through to be used to determine the necessary plot elements for any type of visualization.

In [36]:

```
1 def rat_cat_plot(plot_df, plot_type, box=None):
2
3     # Starting variables
4 #####
5     plot_len = len(plot_df)
6     plot_height = 9
7     #
8     base_x_formatter = x_currency_formatter
9     #
10    rat_insert = plot_df.rating[0]
11    #
12    tit_bullet = r'{\fontsize{28.5}{0}\selectfont{\textbullet\ }}'
13    tit_start = r'{\fontsize{30}{0}\selectfont{\textrm{'
14
15    # crew_plot, main_cat_col
16 #####
17    if plot_type in ['directors', 'writers', 'actors', 'actresses']: crew_plot = True
18    else: crew_plot = False
19    #
20    if plot_type in ['creative_type', 'rating']: main_cat_col = plot_type
21    elif crew_plot: main_cat_col = 'primary_name'
22    else: main_cat_col = 'genres'
23
24    # plot_col, cat_values, cat_percs, cat_freqs and parts of title strs
25 #####
26    if 'avg' in plot_df.columns[-1] or box: plot_col = plot_df.columns[-1]
27    else: plot_col = plot_df.columns[-2]
28    #
29    if 'gross' in plot_col: lab_add = 'Worldwide Gross'
30    if 'profits' in plot_col: lab_add = 'Worldwide Profits'
31    #
32    cat_values = plot_df[plot_col]
33    #
34    if not box: cat_freqs = plot_df['num_titles']
35    #
36    value_step, v_q_used, v_last_neg_q = find_pos_quantile(cat_values)
37    #
38    neg_value_step = cat_values.quantile(v_last_neg_q)
39    #
40    if box or 'avg' in plot_col:
41        perc_plot = False
42        prime_lab_p1 = 'Average '
43        #
44        if box:
45            box_add = 'Boxplots of their '
46            if 'tot' in box: prime_lab_p1 = 'Total '
47        #
48    if (crew_plot or not box) and 'tot' in plot_col:
49        perc_plot = True
50        prime_lab_p1 = 'Total '
51        t_lab_p1 = 'Percentage of Total '
52        t_lab = t_lab_p1 + lab_add
53        #
54        perc_col = plot_df.columns[-1]
55        #
56        cat_percs = plot_df[perc_col]
57        #
58        perc_step, p_q_used, p_last_neg_q = find_pos_quantile(cat_percs)
```

```

59      #-
60      if p_q_used != v_q_used:
61          if p_q_used > v_q_used: value_step = cat_values.quantile(p_q_used)
62          else: perc_step = cat_percs.quantile(v_q_used)
63      #-
64      if box and p_last_neg_q != v_last_neg_q:
65          if p_last_neg_q < v_last_neg_q: neg_value_step = cat_values.quantile(p_
66          else: neg_perc_step = cat_percs.quantile(v_last_neg_q)
67      #-
68      prime_lab = prime_lab_p1 + lab_add
69      #-
70      if box:
71          top_lab = box_add + lab_add
72          #-
73          rank_add = 'Average ' + lab_add
74      #-
75      else: rank_add = prime_lab
76      #-
77      tit_based_on_add = 'Based on ' + prime_lab
78
79      # y_labels, y_lab, tex_y_labels, r_lab parts and title strs
80      ##########
81      if crew_plot or not box:
82          y_labels = plot_df[main_cat_col]
83          #-
84          tit_start_add = 'Top }}' + r'{\fontsize{31.5}{0}\selectfont{\$\\mathrm{' +
85          str(plot_len) +'}}\$\\ }}'
86      #-
87      if box:
88          y_labels = plot_df[main_cat_col].unique()
89          #-
90          tit_start_add = 'Top }}' + r'{\fontsize{31.5}{0}\selectfont{\$\\mathrm{' +
91          str(plot_df[main_cat_col].nunique()) +'}}\$\\ }}'
92      #-
93      tex_y_labels = [r'\textbf{\textsc{' + lab +'}}' for lab in y_labels]
94      #-
95      if plot_type=='genres': y_lab = 'Genres'
96      #-
97      if plot_type=='creative_type': y_lab = 'Creative Types'
98      #-
99      if plot_type=='rating': y_lab = 'MPA Ratings'
100     #-
101     if plot_type=='combos':
102         y_lab = 'Genre \& Creative Type Combinations'
103         #-
104         tex_y_labels = \
105         [r'\textbf{\textsc{' + lab.split(',')[0] +'}}'+ '\n' +
106         r'{\fontsize{22}{0}\selectfont{\textbf{\textsc{\&}}}' + '\n' +
107         r'\textbf{\textsc{' + lab.split(',')[1] +'}}' for lab in y_labels]
108     #-
109     if crew_plot:
110         y_lab = plot_type.title()
111         #-
112         r_lab_1_p1 = 'Number of ' + rat_insert + ' '
113         r_lab_1_p2 = r'\textbf{\textsl{Films' }
114         #-
115         tit_p3_p1 = tit_start + tit_start_add + tit_start + y_lab + ' of ' + rat_ins
116         tit_p3_end = ' Films'
117         #-

```

```

118     if ',' in plot_df.index[0]: sub_cat_insert = ' \& '.join(plot_df.index[0].values)
119     #-
120     else: sub_cat_insert = plot_df.index[0]
121     #-
122     r_lab_1 = r_lab_1_p1 + sub_cat_insert + '}}' + '\n' + r_lab_1_p2
123     #-
124     tit_p3_p1 = tit_p3_p1 + sub_cat_insert + tit_p3_end
125     #-
126     if plot_type in ['directors', 'writers']: r_lab_1_add = 'from each '
127     else: r_lab_1_add = 'with each '
128     #-
129     if plot_type == 'actresses': r_lab_1_add2 = y_lab[:-2]
130     else: r_lab_1_add2 = y_lab[:-1]
131     #-
132     r_lab_1 = r_lab_1 + r_lab_1_add + r_lab_1_add2
133     #-
134     tit_p3_p2 = tit_start + 'Ranked by ' + rank_add + '}}}'
135     #-
136     if plot_type == 'rating':
137         r_lab_1 = 'Number of Films with each MPA Rating'
138         #-
139         tit_p3 = \
140         tit_start + 'MPA Ratings}}}' + tit_bullet + tit_start + 'Ranked by ' + rank_add
141         #-
142         if box:
143             tit_p3_p1 = tit_start + box_add[:-3] + ' ' + lab_add + ' of the MPA Rating'
144         #-
145     else:
146         if not crew_plot:
147             r_lab_1 = 'Number of ' + plot_df.rating[0] + ' Films with'
148             r_lab_2 = 'Each ' + y_lab[:-1]
149             #-
150             tit_p3_p1 = \
151             tit_start + tit_start_add + tit_start + y_lab + ' for ' + rat_insert +
152             #-
153             tit_p3_p2 = tit_start + tit_based_on_add + '}}}'
154             #-
155             if plot_type == 'combos' or crew_plot: tit_p3 = tit_p3_p1 + '\n' + tit_p3_p2
156             #-
157             else: tit_p3 = tit_p3_p1 + tit_bullet + tit_p3_p2
158         #-
159         if box:
160             tit_box_start = r'{\fontsize{28.5}{0}\selectfont{\textsl{'
161             #-
162             tit_box = tit_box_start + top_lab + tit_bullet + 'Ranked by ' + rank_add + '}'
163             #-
164             if plot_type == 'combos': tit_p3 = tit_p3_p1 + '\n' + tit_p3_p2 + '\n' + tit_box
165             elif plot_type == 'rating':
166                 tit_p3 = tit_p3_p1 + '\n' + tit_box_start + 'Ranked by ' + rank_add + '}}}'
167             else: tit_p3 = tit_p3_p1 + tit_bullet + tit_p3_p2 + '\n' + tit_box
168         #-
169         # Setting other variables necesary for plotting
170 #####
171 num_bars = 1
172 bar_height = 3
173 group_space = 4.5
174 #-
175 if not box: y_locs = range(plot_len)
176 else: y_locs = range(plot_df[main_cat_col].nunique())

```

```

177
178 # Locations for bars or boxes
179 #####
180 y_ticks = [num_bars*group_space*y + bar_height*(1.5) for y in y_locs]
181 #####
182 split_dist = ((y_ticks[1] - y_ticks[0]) / 2)
183 split_locs = [(num_bars*group_space)*y + bar_height*(1.5) + split_dist for y in
184
185 # Setting up the figure and the axes
186 #####
187 fig = plt.figure(figsize=(18, plot_height), dpi=300)
188 gs = fig.add_gridspec(1)
189 #####
190 ax = gs.subplots()
191 #####
192 if perc_plot: t_ax = ax.twiny()
193 if not box: r_ax = ax.twinx()
194 #####
195 renderer = fig.canvas.get_renderer()
196
197 # Plotting the data based on if box=True
198 #####
199 if not box:
200     if plot_type != 'rating':
201         bar_colors = get_lighter_color(ratings_colors_dict[plot_df.rating[0]],
202     else:
203         bar_colors = \
204             [get_lighter_color(ratings_colors_dict[rat], .45) for rat in plot_df.r
205     #####
206     bars_kw = dict(color=bar_colors, ec='k', lw=.54, zorder=3)
207     #####
208     main_bars = ax.banh(y_ticks, cat_values, bar_height, **bars_kw)
209 #####
210 if box:
211     c_patch = bar_colors = get_lighter_color(ratings_colors_dict[plot_df.rating[0]])
212     c_line = bar_colors = get_lighter_color(ratings_colors_dict[plot_df.rating[1]])
213     #####
214     for cat, y_t in zip(plot_df[main_cat_col].unique(), y_ticks):
215         #####
216         box_kw = dict(fc=c_patch, lw=.9, ec='k')
217         means_kw = dict(color='k', ls=(0, (1, 1)), lw=6.3, zorder=6)
218         meds_kw = dict(color='k', lw=4.5, zorder=99)
219         caps_kw = dict(c=c_line, lw=4.2, solid_capstyle='round', zorder=6)
220         whis_kw = dict(c=c_line, lw=4.2, solid_capstyle='round', zorder=6)
221         fliers_kw = dict(mfc=c_line, ms=15, mec='k', mew=.3, marker='X')
222         #####
223         b_kw = dict(widths=bar_height, boxprops=box_kw, medianprops=meds_kw, m
224             showmeans=True, meanprops=means_kw, flierprops=fliers_kw,
225             whiskerprops=whis_kw, capprops=caps_kw, patch_artist=True,
226         #####
227         cat_df = plot_df.set_index(main_cat_col).loc[cat]
228         #####
229         box_dict = ax.boxplot(cat_df[plot_col], positions=[y_t], **b_kw)
230         #####
231         mean_lines = \
232             mlines.Line2D([], [], label=r'\textbf{\textit{Mean Lines}}', **means_k
233         #####
234         median_lines = \
235             mlines.Line2D([], [], label=r'\textbf{\textit{Median Lines}}', **meds_

```

```

236
237     # AX - X Ticks, Tick Labels, and Bounds
238 #####
239     x_max = cat_values.max()
240     #
241     final_max = find_max(x_max, value_step)
242     #
243     x_tick_max = final_max
244     #
245     if not box or 'gross' in plot_col:
246         x_step = final_max/6
247         x_ticks = np.arange(0, x_tick_max +1, x_step)
248     #
249     if box and 'profits' in plot_col:
250         x_min = cat_values.min()
251         #
252         final_min = find_min(x_min, neg_value_step)
253         #
254         if final_min < 0:
255             zero_kw = dict(color='k', lw=3.9, ls='-.')
256             #
257             zero_line = \
258                 ax.axvline(0, label=r'\textbf{\textit{Zero Profits}}'+ '\n'+\
259                             r'\textbf{\textit{Line}}}', **zero_kw)
260         #
261     else: final_min = 0
262     #
263     x_tick_min = final_min
264     x_step = (final_max - final_min) / 6
265     x_ticks = np.arange(final_min, x_tick_max+1, x_step)
266     #
267     ax.set_xticks(x_ticks)
268     #
269     ax.xaxis.set_major_formatter(base_x_formatter)
270     #
271     ax.set_xbound(x_ticks[0], x_ticks[-1])
272
273     # T_AX - X Ticks, Tick Labels, and Bounds
274 #####
275     if perc_plot:
276         x_max = cat_percs.max()
277         #
278         final_max = find_max(x_max, perc_step)
279         #
280         x_tick_max = final_max
281         x_step = final_max/6
282         x_ticks = np.arange(0, x_tick_max+1, x_step)
283         #
284         t_ax.set_xticks(x_ticks)
285         t_ax.xaxis.set_major_formatter(x_percentage_formatter)
286         #
287         t_ax.set_xbound(0, x_tick_max)
288
289     # AX & R_AX - Y Tick, Tick Labels and Bounds
290 #####
291     ax.set_yticks(y_ticks)
292     ax.set_yticklabels(tex_y_labels, va='center')
293     ax.set_ybound((y_ticks[0] - split_dist, y_ticks[-1] + split_dist))
294     ax.invert_yaxis()

```

```

295     #-----
296     if not box:
297         freq_labels = y_labels_to_norm_num(cat_freqs)
298         #-----
299         r_ax.set_yticks(y_ticks)
300         r_ax.set_yticklabels(freq_labels, va='center')
301         r_ax.set_ybound((y_ticks[0] - split_dist, y_ticks[-1] + split_dist))
302         r_ax.invert_yaxis()
303
304     # Tick Params Functions
305 #####
306     if not box:
307         h_tick_params(ax)
308         right_y_tick_params(r_ax)
309     #-----
310     else: h_tick_params(ax, True)
311     #-----
312     if perc_plot: top_x_tick_params(t_ax)
313
314     # Axis Labels
315 #####
316     x_lab_kw= dict(size=28.5, labelpad=18)
317     y_lab_kw= dict(size=30, labelpad=27, linespacing=1.5)
318     #-----
319     if not box: ax.set_xlabel(r'\textbf{\textit{'+ prime_lab +'}}', **x_lab_kw)
320     if box: ax.set_xlabel(r'\textbf{\textit{'+ lab_add +'}}', **x_lab_kw)
321     #-----
322     if perc_plot:
323         t_ax.set_xlabel(r'\textbf{\textit{'+ t_lab +'}}', **x_lab_kw)
324     #-----
325     ax.set_ylabel(r'\textbf{\textit{'+ y_lab +'}}', **y_lab_kw)
326     if not box:
327         if plot_type == 'rating' or crew_plot:
328             r_ax.set_ylabel(r'\textbf{\textit{'+ r_lab_1 +'}}', va='bottom', rotation=90, **y_lab_kw)
329         #-----
330     else:
331         r_ax.set_ylabel(r'\textbf{\textit{'+ r_lab_1 +'}}'+ '\n'+ r'\textbf{\textit{'+ r_lab_2 +'}}', va='bottom', rotation=90, **y_lab_kw)
332
333     # Title
334 #####
335     if perc_plot: tit_kw = dict(pad=36, linespacing=1.8)
336     else: tit_kw = dict(pad=18, linespacing=1.8)
337     #-----
338     rat_cat_source = r'{\fontsize{31.5}{0}\selectfont{\textbf{\textit{OpusData}}}}'
339     #-----
340     if plot_type != 'rating':
341         m_tit_p1 = \
342             r'{\fontsize{31.5}{0}\selectfont{\textbf{\textit{Categorical Data}}}}'
343     #-----
344     else: m_tit_p1 = r'{\fontsize{31.5}{0}\selectfont{\textbf{\textit{Ratings Data}}}}'
345     #-----
346     year_tit = r'2010 \boldsymbol{\rightarrow} 2018'
347     #-----
348     finan_data_source = \
349         r'{\fontsize{31.5}{0}\selectfont{\textbf{\textit{The Numbers Financial Data}}}}'
350     #-----
351
352
353

```

```

354     year_tit = r'2010 \boldsymbol{\rightarrow} 2018'
355     #
356     tit_p2 = r'{\fontsize{33}{0}\selectfont{$\mathbf{'+ year_tit +'}$}}'
357     #
358     if not crew_plot:
359         tit_p1 = rat_cat_source + m_tit_p1 + '\n' + finan_data_source
360     #
361     if crew_plot:
362         crew_source = r'{\fontsize{31.5}{0}\selectfont{\textbf{IMDb}}}'
363         m_tit_crew = r'{\fontsize{31.5}{0}\selectfont{\textbf{\textsc{' + y_lab +
364             '#-'
365             tit_p1 = rat_cat_source + m_tit_p1 + '\n' +
366             crew_source + m_tit_crew + tit_bullet + finan_data_source
367             #
368             ax.set_title(tit_p1 +'\n'+ tit_p2 +'\n'+ tit_p3, **tit_kw)
369             #
370             # Legend
371             ######
372             leg_loc = 'center left'
373             #
374             leg_kw = dict(loc=leg_loc, bbox_to_anchor=(1, .5), fontsize=21, handlelength=3
375             #
376             frame_kw = dict(lw=.9, ec='k', alpha=1)
377             #
378             if box:
379                 if 'gross' in plot_col: leg = ax.legend(handles=[mean_lines, median_lines]
380                 #
381             else:
382                 if final_min < 0:
383                     leg = ax.legend(handles=[zero_line, mean_lines, median_lines], **l
384                     #
385                 else: leg = ax.legend(handles=[mean_lines, median_lines], **leg_kw)
386                     #
387                     plt.setp(leg.get_frame(), **frame_kw)
388                     plt.setp(leg.get_texts(), ma='center')
389             #
390             # Adding Lines between each genre's name (lines between y tick labels)
391             ######
392             c_tick_kw = dict(clip_on=False, color='k', lw=1.5, alpha=.81, zorder=9)
393             #
394             top_spl = y_ticks[-1] + split_dist
395             bot_spl = y_ticks[0] - split_dist
396             #
397             max_label_neg_x = 0
398             max_label_pos_x = 0
399             #
400             ax_inv = ax.transAxes.inverted()
401             #
402             if not box:
403                 r_ax_inv = r_ax.transAxes.inverted()
404                 #
405                 for label, r_label in zip(ax.get_ymajorticklabels(), r_ax.get_ymajortickla
406                     lab_xymin = label.get_tightbbox(renderer).min
407                     r_lab_xymin = r_label.get_tightbbox(renderer).max
408                     #
409                     lab_f_bbox_xymin = ax_inv.transform(lab_xymin)
410                     r_lab_f_bbox_xymin = r_ax_inv.transform(r_lab_xymin)
411                     #
412                     if lab_f_bbox_xymin[0] <= max_label_neg_x: max_label_neg_x = lab_f_bbo

```

```

413         #-----
414         if r_lab_f_bbox_xymax[0] >= max_label_pos_x: max_label_pos_x = r_lab_f_
415         #-----
416         pos_kw = dict(x=((max_label_pos_x - 1) / 2) + 1.006, ha='center')
417         [r_label.update(pos_kw) for r_label in r_ax.get_ymajorticklabels()]
418     #-----
419 else:
420     for label in ax.get_ymajorticklabels():
421         lab_xymin = label.get_tightbbox(renderer).min
422         #-----
423         lab_f_bbox_xymin = ax_inv.transform(lab_xymin)
424         #-----
425         if lab_f_bbox_xymin[0] <= max_label_neg_x: max_label_neg_x = lab_f_bbo_
426     #-----
427     neg_kw = dict(x=(max_label_neg_x/2) - .0045, ha='center')
428     [label.update(neg_kw) for label in ax.get_ymajorticklabels()]
429     #-----
430     for s_loc in split_locs:
431         ax.axhline(s_loc, 0, max_label_neg_x - .015, **c_tick_kw)
432         #-----
433         if not box: ax.axhline(s_loc, 1, max_label_pos_x + .015, **c_tick_kw)
434     #-----
435     bot_ax_line = ax.axhline(bot_spl, 0, max_label_neg_x - .015, **c_tick_kw)
436     if not box: ax.axhline(bot_spl, 1, max_label_pos_x + .015, **c_tick_kw)
437     #-----
438     top_ax_line = ax.axhline(top_spl, 0, max_label_neg_x - .015, **c_tick_kw)
439     if not box: top_ax_line = ax.axhline(top_spl, 1, max_label_pos_x + .015, **c_t_
440
441     # Adding other lines to enhance appearance
442 ##### div_kw = dict(color='k', lw=1.95), alpha=.63)
443     #-----
444     fig_inv = fig.transFigure.inverted()
445     #-----
446     if perc_plot:
447         ax_xymin = t_ax.get_tightbbox(renderer).min
448         ax_xymax = t_ax.get_tightbbox(renderer).max
449         top_lab_xymax = t_ax.xaxis.get_label().get_tightbbox(renderer).max
450         #-----
451         div_left_x = fig_inv.transform(ax_xymin)[0]
452         div_right_x = fig_inv.transform(ax_xymax)[0]
453         div_y = fig_inv.transform(top_lab_xymax)[1]
454         div_y = div_y + .027
455         #-----
456         div_line = mlines.Line2D([div_left_x, div_right_x], [div_y, div_y], **c_t_
457     #-----
458     left_vert_x = fig_inv.transform(bot_ax_line.get_tightbbox(renderer).min)[0]
459     if not box: right_vert_x = fig_inv.transform(top_ax_line.get_tightbbox(render_
460     #-----
461     bot_vert_y = fig_inv.transform(top_ax_line.get_tightbbox(renderer).max)[1]
462     top_vert_y = fig_inv.transform(bot_ax_line.get_tightbbox(renderer).min)[1]
463     #-----
464     left_vert = mlines.Line2D([[left_vert_x] * 2], [bot_vert_y, top_vert_y], **c_t_
465     if not box:
466         right_vert = mlines.Line2D([[right_vert_x] * 2], [bot_vert_y, top_vert_y],_
467     #-----
468     if not box and 'tot' in plot_col:
469         fig.add_artist(div_line)
470     #-----
471

```

```
472     if not box: [fig.add_artist(art) for art in [left_vert, right_vert]]
473 else: fig.add_artist(left_vert)
474
475 # Returning the figure with a Label to save accordingly if needed
476 #####
477 if not box: return fig, prime_lab
478 else: return fig, top_lab
```

§ Standards for Qualifying as Important

As I explored the results for each subcategory type, I developed a set of standards for what I would consider to be an important subcategory. Those standards are described below.

Standards for Qualifying as an Important Subcategory

- For a subcategory to even be considered as important, there had to have been **at least nine movies made** with that subcategory within the period covered.
- For each metric, only the **Top 3** subcategories within each subcategory type were considered as financially important subcategories for that metric.
- If there were **less than nine movies made** with a subcategory, but that subcategory's value for a metric was **greater than the maximum value of the Top 3** for that metric, it was added to the [Additional Insights](#) section.

§ Creating & Plotting the Aggregate DataFrames

I first created a list of strings that corresponded to each financial metric to iterate through to create the appropriate aggregate dataframes. I created those aggregate dataframes for each of the subcategory types previously mentioned, as well as the main category type itself.

In [37]:

```
1 sum_aka_1 = 'tot_worldwide_gross'
2 #-----
3 sum_aka_2 = 'tot_worldwide_profits'
4 #-----
5 avg_aka_1 = 'avg_worldwide_gross'
6 #-----
7 avg_aka_2 = 'avg_worldwide_profits'
8 #-----
9 all_akas = [sum_aka_1, sum_aka_2, avg_aka_1, avg_aka_2]
```

I also created the dataframe to hold all of the [Additional Insights](#) section data to be processed later.

In [38]:

```
1 g_add_info_df = pd.DataFrame()  
2 #-----  
3 ct_add_info_df = pd.DataFrame()  
4 #-----  
5 combo_add_info_df = pd.DataFrame()
```

Financially Important Genres

I used the visualizations created in this section to determine my Genres recommendations for each MPA Rating.

[Click here to see example visualizations.](#)

In [39]:

```
1 base_dir = 'visuals/'  
2 #-----  
3 out_1 = widgets.Output()  
4 display(out_1)  
#-----  
5  
6 for a_i, aka_v in enumerate(all_akas):  
7     plot_col = aka_v[4:]  
8     plot_col_split = plot_col.split('_')  
9     #-----  
10    g_count_df = fin_df.groupby(['rating', 'genre']).count()[['primary_title']]  
11    #-----  
12    if 'tot' in aka_v:  
13        aka_start = 'Total'  
14        #-----  
15        g_agg_df = fin_df.groupby(['rating', 'genre']).sum()[[plot_col]]  
16    #-----  
17    if 'avg' in aka_v:  
18        aka_start = 'Average'  
19        #-----  
20        g_agg_df = fin_df.groupby(['rating', 'genre']).mean()[[plot_col]]  
21    #-----  
22    all_gs_df = pd.concat([g_count_df, g_agg_df], axis=1).reset_index()\br/>23        .sort_values(['rating', plot_col], ascending=[True, False])\br/>24        .rename(columns={'primary_title':'num_titles', 'genre':'genres',  
25                    plot_col: aka_v})  
26    #-----  
27    if 'tot' in aka_v:  
28        all_gs_df['perc_of_'+ aka_v] = round((all_gs_df[aka_v] / fin_df[plot_col]).  
29    #-----  
30    aka_str = ' '.join([aka_start] + plot_col_split).title()  
31    #-----  
32    print('\033[1m\033[35mAgg Col: \033[0m', aka_str)  
33    print('-----')  
34  
35    # Finding the Important Genres and the Genres for the Additional Information so  
36    #####  
37    aka_add_info_df = pd.DataFrame()  
38    #-----  
39    for rat in all_gs_df.rating.unique():  
40        if rat != 'Not Rated': rat_g_df = all_gs_df.loc[all_gs_df.rating == rat]  
41        else: continue  
42        #-----  
43        rat_copy_df = rat_g_df.copy()  
44        #-----  
45        rat_g_df = rat_g_df.loc[(rat_g_df.num_titles >= 9)].head(3)  
46        #-----  
47        rat_also_df = rat_copy_df.loc[(rat_copy_df.num_titles < 9) &  
48                                         (rat_copy_df[aka_v] >= rat_g_df[aka_v].max())  
49        #-----  
50        rat_also_df[aka_v] = rat_also_df[aka_v].map(lambda x: x - rat_g_df[aka_v].  
51        #-----  
52        if len(rat_also_df):  
53            rat_also_df = rat_also_df.set_index(['rating', 'genres', 'num_titles'])  
54            #-----  
55            aka_add_info_df = pd.concat([aka_add_info_df, rat_also_df])  
56            #-----  
57            print('\033[1m\033[37mAdded data to the Additional Information datafram  
58            '\033[31mRating\033[0m.')
```

```

59         print('  \033[1m\033[31mRating: \033[0m', rat)
60         print('-----')
61     #
62     rat_g_df = rat_g_df.reset_index(drop=True)
63     #
64     rat_dir = base_dir + rat + '/'
65     rat_cat_dir = rat_dir + 'Important Genres/'
66     rat_cat_metric_dir = rat_cat_dir + aka_str + '/'
67     #
68     if not os.path.isdir(rat_dir):
69         os.mkdir(rat_dir)
70     #
71     if not os.path.isdir(rat_cat_dir):
72         os.mkdir(rat_cat_dir)
73     #
74     if not os.path.isdir(rat_cat_metric_dir):
75         os.mkdir(rat_cat_metric_dir)
76     #
77     with out_1:
78         display(print('\033[1m\033[31mRating: \033[0m', rat), include=[str])
79     #
80     if len(rat_g_df) == 1:
81         print('\033[1m\033[37mThere was only one associated \033[32mGenre\033[0m')
82         print('  \033[1m\033[37mfor this \033[31mRating\033[0m.')
83         print('  \033[1m\033[31mRating: \033[0m', rat)
84         print('  \033[1m\033[32mGenre: \033[0m', rat_g_df.genres.values[0])
85         print('-----')
86
87     # Top Genres and their Boxplots Visualizations
88 #####
89     if len(rat_g_df) > 1:
90         fig, fig_tit = rat_cat_plot(rat_g_df, 'genres')
91         #
92         f_name = rat_cat_metric_dir + '1 - Top Genres.jpeg'
93         #
94         fig.savefig(f_name, bbox_inches='tight')
95         plt.close(fig)
96         #
97         all_box_df = pd.DataFrame()
98         import_order_dict = {}
99         #
100        for g in rat_g_df.genres:
101            rat_g_box_df = fin_df.loc[(fin_df.rating == rat) & (fin_df.genre == g)]
102            #
103            rat_g_box_df = rat_g_box_df[['rating', 'genre', plot_col]]
104            #
105            import_order_dict[g] = rat_g_box_df[plot_col].mean()
106            #
107            all_box_df = pd.concat([all_box_df, rat_g_box_df])
108        #
109        import_order = \
110        sorted(import_order_dict.keys(), key=lambda k: import_order_dict[k], reverse=True)
111        #
112        all_box_df = \
113        all_box_df.sort_values('genre',
114                               key=lambda g_col: g_col.map(lambda x: import_order[x]))
115        #
116        all_box_df = all_box_df.rename(columns={'genre': 'genres', plot_col: 'rating'})
117        .reset_index(drop=True)

```

```

118
119     #-----
120     fig, fig_tit = rat_cat_plot(all_box_df, 'genres', aka_v)
121     #-----
122     f_name = rat_cat_metric_dir + '2 - ' + fig_tit + '.jpeg'
123     #-----
124     fig.savefig(f_name, bbox_inches='tight')
125     plt.close(fig)
126
127     # Important Crew Visualizations
128 #####
129     for g in rat_g_df.genres:
130         with out_1:
131             display(print('\033[1m\033[32mGenre: \033[0m', g), include=[str])
132         #####
133         for crew_str, crew_df in zip(crew_strings, crew_dfs):
134             if crew_str != 'actress': plot_type = crew_str + 's'
135             else: plot_type = crew_str + 'es'
136             #####
137             crew_dir = rat_cat_metric_dir + plot_type.title() + '/'
138             #####
139             if not os.path.isdir(crew_dir):
140                 os.mkdir(crew_dir)
141             #####
142             rat_g_crew_df = crew_df.loc[(crew_df.rating == rat) & (crew_df.gen
143             #####
144             if rat_g_crew_df.primary_name.nunique() <= 1:
145                 print('\033[1m\033[37mThere was only one crew member associated
146                 '\033[32mGenre\033[37m for this \033[31mRating\033[0m.')
147                 print('  \033[1m\033[31mRating: \033[0m' + rat)
148                 print('  \033[1m\033[32mGenre: \033[0m' + g)
149                 print('  \033[1m\033[33mCrew Type: \033[0m' + crew_str.title())
150                 print('  \033[1m\033[33mName: \033[0m', rat_g_crew_df.primary_
151                 print('-----')
152                 continue
153             #####
154             else:
155                 crew_count_df = \
156                     rat_g_crew_df.groupby('primary_name').count()[['primary_title'
157                 #####
158                 if 'tot' in aka_v:
159                     crew_agg_df = rat_g_crew_df.groupby('primary_name').sum()[[
160                 #####
161                 if 'avg' in aka_v:
162                     crew_agg_df = rat_g_crew_df.groupby('primary_name')\
163                         .mean()[[plot_col]]
164                 #####
165                 c_df = pd.concat([crew_count_df, crew_agg_df], axis=1).reset_i
166                     .rename(columns={'primary_title':'num_titles', plot_c
167                     .sort_values(aka_v, ascending=False).head(10)
168                 #####
169                 genre_ser = c_df.primary_name.map(lambda x: g).rename('genres'
170                 rat_ser = c_df.primary_name.map(lambda x: rat).rename('rating'
171                 #####
172                 c_df.insert(0, 'genres', genre_ser)
173                 c_df.insert(1, 'rating', rat_ser)
174                 #####
175                 if 'tot' in aka_v:
176                     c_df['perc_of_'+ aka_v] = \

```

```

177      #-----
178      c_df = c_df.reset_index(drop=True).set_index('genres')
179      #-----
180      fig, fig_tit = rat_cat_plot(c_df, plot_type)
181      #-----
182      if '/' in g: g_dir = ' - '.join(g.split('/'))
183      else: g_dir = g
184      #-----
185      f_name = crew_dir + g_dir + '.jpeg'
186      #-----
187      fig.savefig(f_name, bbox_inches='tight')
188      plt.close(fig)
189      #-----
190      with out_1:
191          display(print('\033[1m\033[32mGenre Finsished\033[0m'), include=[s])
192
193      # Clearing non-important information after each rating
194      ######
195      with out_1:
196          clear_output()
197      print()
198
199      # Adding the Additional Information section information
200      #####
201      if not aka_add_info_df.empty:
202          g_add_info_df = pd.concat([g_add_info_df, aka_add_info_df[[aka_v]]], axis=0)

```

Agg Col: Total Worldwide Gross

There was only one associated **Genre** for this **Rating**.

Rating: G

Genre: Adventure

Agg Col: Total Worldwide Profits

There was only one associated **Genre** for this **Rating**.

Rating: G

Genre: Adventure

Agg Col: Average Worldwide Gross

There was only one associated **Genre** for this **Rating**.

Rating: G

Genre: Adventure

Added data to the Additional Information dataframe for this **Rating**.

Rating: R

Agg Col: Average Worldwide Profits

There was only one associated **Genre** for this **Rating**.

Rating: G

Genre: Adventure

Added data to the Additional Information dataframe for this **Rating**.

Rating: R

Financially Important Creative Types

I used the visualizations created in this section to determine my Creative Types recommendations for each MPA Rating.

[Click here to see Example Visualizations.](#)

In [40]:

```
1 base_dir = 'visuals/'  
2 #-----  
3 out_2 = widgets.Output()  
4 display(out_2)  
5 #-----  
6 rat_add_df = pd.DataFrame()  
7 #-----  
8 for a_i, aka_v in enumerate(all_akas):  
9     plot_col = aka_v[4:]  
10    plot_col_split = plot_col.split('_')  
11    #-----  
12    c_count_df = fin_df.groupby(['rating', 'creative_type']).count()[['primary_tit  
13    #-----  
14    if 'tot' in aka_v:  
15        aka_start = 'Total'  
16        #-----  
17        c_agg_df = fin_df.groupby(['rating', 'creative_type']).sum()[[plot_col]]  
18    #-----  
19    if 'avg' in aka_v:  
20        aka_start = 'Average'  
21        #-----  
22        c_agg_df = fin_df.groupby(['rating', 'creative_type']).mean()[[plot_col]]  
23    #-----  
24    all_cs_df = pd.concat([c_count_df, c_agg_df], axis=1).reset_index()\br/>25        .sort_values(['rating', plot_col], ascending=[True, False])\br/>26        .rename(columns={'primary_title':'num_titles', plot_col: aka_v})  
27    #-----  
28    if 'tot' in aka_v:  
29        all_cs_df['perc_of_'+ aka_v] = round((all_cs_df[aka_v] / fin_df[plot_col].  
30    #-----  
31    aka_str = ' '.join([aka_start] + plot_col_split).title()  
32    #-----  
33    print('\033[1m\033[35mAgg Col: \033[0m', aka_str)  
34    print('-----')  
35    #-----  
36    # Finding the Important Creative Types and the Creative Types for the Additional  
37    # section  
38    #####  
39    aka_add_info_df = pd.DataFrame()  
40    #-----  
41    for rat in all_cs_df.rating.unique():  
42        if rat != 'Not Rated': rat_c_df = all_cs_df.loc[all_cs_df.rating == rat]  
43        else: continue  
44        #-----  
45        rat_copy_df = rat_c_df.copy()  
46        #-----  
47        rat_c_df = rat_c_df.loc[(rat_c_df.num_titles >= 9)].head(3)  
48        #-----  
49        rat_also_df = rat_copy_df.loc[(rat_copy_df.num_titles < 9) &  
50                                         (rat_copy_df[aka_v] >= rat_c_df[aka_v].max())  
51        #-----  
52        rat_also_df[aka_v] = rat_also_df[aka_v].map(lambda x: x - rat_c_df[aka_v].  
53        #-----  
54        if len(rat_also_df):  
55            rat_also_df = rat_also_df.set_index(['rating', 'creative_type', 'num_t  
56            #-----  
57            aka_add_info_df = pd.concat([aka_add_info_df, rat_also_df])  
58            #-----
```

```

59         print('\033[1m\033[37mAdded data to the Additional Information datafram')
60             '\033[31mRating\033[0m.')
61         print(' \033[1m\033[31mRating: \033[0m', rat)
62         print('-----')
63     #-----
64     rat_c_df = rat_c_df.reset_index(drop=True)
65     #-----
66     rat_dir = base_dir + rat + '/'
67     rat_cat_dir = rat_dir + 'Important Creative Types/'
68     rat_cat_metric_dir = rat_cat_dir + aka_str + '/'
69     #-----
70     if not os.path.isdir(rat_dir):
71         os.mkdir(rat_dir)
72     #-----
73     if not os.path.isdir(rat_cat_dir):
74         os.mkdir(rat_cat_dir)
75     #-----
76     if not os.path.isdir(rat_cat_metric_dir):
77         os.mkdir(rat_cat_metric_dir)
78     #-----
79     with out_2:
80         display(print('\033[1m\033[31mRating: \033[0m', rat), include=[str])
81     #-----
82     if len(rat_c_df) == 1:
83         print('\033[1m\033[37mThere was only one associated \033[32mCreative T')
84             '\033[1m\033[37mfor this \033[31mRating\033[0m.')
85         print(' \033[1m\033[31mRating: \033[0m', rat)
86         print(' \033[1m\033[32mCreative Type: \033[0m', rat_c_df.creative_ty
87         print('-----')
88
89     # Top Creative Types and their Boxplots Visualizations
90 #####
91     if len(rat_c_df) > 1:
92         fig, fig_tit = rat_cat_plot(rat_c_df, 'creative_type')
93         #-----
94         f_name = rat_cat_metric_dir + '1 - Top Creative Types.jpeg'
95         #-----
96         fig.savefig(f_name, bbox_inches='tight')
97         plt.close(fig)
98         #-----
99         all_box_df = pd.DataFrame()
100        import_order_dict = {}
101        #-----
102        for ct in rat_c_df.creative_type:
103            rat_ct_box_df = fin_df.loc[(fin_df.rating == rat) & (fin_df.creati
104            #
105            rat_ct_box_df = rat_ct_box_df[['rating', 'creative_type', plot_col
106            #
107            import_order_dict[ct] = rat_ct_box_df[plot_col].mean()
108            #
109            all_box_df = pd.concat([all_box_df, rat_ct_box_df])
110        #-----
111        import_order = \
112            sorted(import_order_dict.keys(), key=lambda k: import_order_dict[k], r
113        #
114        all_box_df = \
115            all_box_df.sort_values('creative_type',
116                                key=lambda c_col: c_col.map(lambda x: import_order[x]))
117        #

```

```

118     all_box_df = all_box_df.rename(columns={plot_col: aka_v}).reset_index()
119
120     #-----
121     fig, fig_tit = rat_cat_plot(all_box_df, 'creative_type', aka_v)
122
123     #-----
124     f_name = rat_cat_metric_dir + '2 - ' + fig_tit + '.jpeg'
125
126     #-----
127     fig.savefig(f_name, bbox_inches='tight')
128     plt.close(fig)
129
130
131     # Important Crew Visualizations
132     #####
133     for ct in rat_c_df.creative_type:
134         with out_2:
135             display(print('\033[1m\033[32mCreative Type: \033[0m', ct), include=True)
136
137         for crew_str, crew_df in zip(crew_strings, crew_dfs):
138             if crew_str != 'actress': plot_type = crew_str + 's'
139             else: plot_type = crew_str + 'es'
140
141             #-----
142             crew_dir = rat_cat_metric_dir + plot_type.title() + '/'
143
144             if not os.path.isdir(crew_dir):
145                 os.mkdir(crew_dir)
146
147             rat_ct_crew_df = crew_df.loc[(crew_df.rating == rat) &
148                                           (crew_df.creative_type == ct)]
149
150             if rat_ct_crew_df.primary_name.nunique() <= 1:
151                 print('\033[1m\033[37mThere was only one crew member associated with this creative type\033[0m')
152                 print('    \033[1m\033[31mRating: \033[0m' + rat)
153                 print('    \033[1m\033[32mCreative Type: \033[0m', ct)
154                 print('    \033[1m\033[33mCrew Type: \033[0m', crew_str.title())
155                 print('    \033[1m\033[33mName: \033[0m',
156                     rat_ct_crew_df.primary_name.values[0])
157                 print('-----')
158                 continue
159
160             else:
161                 crew_count_df = \
162                     rat_ct_crew_df.groupby('primary_name').count()[['primary_title']]
163
164                 if 'tot' in aka_v:
165                     crew_agg_df = \
166                         rat_ct_crew_df.groupby('primary_name').sum()[[plot_col]]
167
168                 if 'avg' in aka_v:
169                     crew_agg_df = \
170                         rat_ct_crew_df.groupby('primary_name').mean()[[plot_col]]
171
172                 c_df = pd.concat([crew_count_df, crew_agg_df], axis=1).reset_index()
173                 .rename(columns={'primary_title': 'num_titles', plot_col: 'rating'})
174                 .sort_values(aka_v, ascending=False).head(10)
175
176                 ct_ser = c_df.primary_name.map(lambda x: ct).rename('creative_type')
177                 rat_ser = c_df.primary_name.map(lambda x: rat).rename('rating')
178
179                 c_df.insert(0, 'creative_type', ct_ser)
180                 c_df.insert(1, 'rating', rat_ser)

```

```
177
178     #-----
179     if 'tot' in aka_v:
180         c_df['perc_of_'+ aka_v] = \
181             round((c_df[aka_v] / fin_df[plot_col].sum()), 4)
182     #-----
183     c_df = c_df.reset_index(drop=True).set_index('creative_type')
184     #-----
185     fig, fig_tit = rat_cat_plot(c_df, plot_type)
186     #-----
187     f_name = crew_dir + ct + '.jpeg'
188     #-----
189     fig.savefig(f_name, bbox_inches='tight')
190     plt.close(fig)
191     #-----
192     with out_2:
193         display(print('\033[1m\033[32mCreative Type Finished\033[0m'),incl
194
195         # Clearing non-important information after each rating
196         ######
197         with out_2:
198             clear_output()
199         print()
200
201         # Adding the Additional Information section information
202         #####
203         if not aka_add_info_df.empty:
204             ct_add_info_df = pd.concat([ct_add_info_df, aka_add_info_df[[aka_v]]], axi
```

Agg Col: Total Worldwide Gross

There was only one associated Creative Type for this Rating.

Rating: G

Creative Type: Kids Fiction

Agg Col: Total Worldwide Profits

There was only one associated Creative Type for this Rating.

Rating: G

Creative Type: Kids Fiction

Agg Col: Average Worldwide Gross

There was only one associated Creative Type for this Rating.

Rating: G

Creative Type: Kids Fiction

Added data to the Additional Information dataframe for this Rating.

Rating: PG

Added data to the Additional Information dataframe for this Rating.

Rating: R

Agg Col: Average Worldwide Profits

There was only one associated Creative Type for this Rating.

Rating: G

Creative Type: Kids Fiction

Added data to the Additional Information dataframe for this **Rating**.

Rating: PG

Added data to the Additional Information dataframe for this **Rating**.

Rating: R

Financially Important Genre & Creative Type Combinations

I used the visualizations created in this section to determine my Genre & Creative Type Combinations recommendations for each MPA Rating.

[Click here to see Example Visualizations.](#)

In [41]:

```
1 base_dir = 'visuals/'  
2 #-----  
3 out_3 = widgets.Output()  
4 display(out_3)  
5 #-----  
6 rat_add_df = pd.DataFrame()  
7 #-----  
8 for a_i, aka_v in enumerate(all_akas):  
9     plot_col = aka_v[4:]  
10    plot_col_split = plot_col.split('_')  
11    #-----  
12    combo_count_df = \  
13        fin_df.groupby(['rating', 'genre', 'creative_type']).count()[['primary_title']]  
14    #-----  
15    if 'tot' in aka_v:  
16        aka_start = 'Total'  
17        #-----  
18        combo_agg_df = fin_df.groupby(['rating', 'genre', 'creative_type']).sum()  
19    #-----  
20    if 'avg' in aka_v:  
21        aka_start = 'Average'  
22        #-----  
23        combo_agg_df = fin_df.groupby(['rating', 'genre', 'creative_type']).mean()  
24    #-----  
25    all_combos_df = pd.concat([combo_count_df, combo_agg_df], axis=1).reset_index()  
26        .sort_values(['rating', plot_col], ascending=[True, False])\  
27        .rename(columns={'primary_title':'num_titles', 'genre':'genres',  
28                           plot_col: aka_v})  
29    #-----  
30    if 'tot' in aka_v:  
31        all_combos_df['perc_of_'+ aka_v] = \  
32            round((all_combos_df[aka_v] / fin_df[plot_col].sum()), 4)  
33    #-----  
34    aka_str = ' '.join([aka_start] + plot_col_split).title()  
35    #-----  
36    print('\033[1m\033[35mAgg Col: \033[0m', aka_str)  
37    print('-----')  
38  
39    # Finding the Important Combos and the Combos for the Additional Information so  
40    #####  
41    all_combos_df = \  
42    all_combos_df.loc[all_combos_df.rating.map(lambda x: True if x!='Not Rated' else False),]  
43    all_combos_df = all_combos_df.sort_values(aka_v, ascending=False)  
44    #-----  
45    aka_add_info_df = pd.DataFrame()  
46    #-----  
47    for rat in all_combos_df.rating.unique():  
48        rat_df = all_combos_df.loc[(all_combos_df.rating == rat) &  
49                                      (all_combos_df.num_titles >= 9)].copy()  
50        #-----  
51        new_genres = rat_df.apply(lambda x: ', '.join([x['genres'], x['creative_type']]))  
52        #-----  
53        rat_df = rat_df.drop(columns=['genres', 'creative_type']).head(3)  
54        #-----  
55        rat_also_df = all_combos_df.loc[(all_combos_df.rating == rat) &  
56                                         (all_combos_df.num_titles < 9) &  
57                                         (all_combos_df[aka_v] >= rat_df[aka_v].max)]  
58        #-----
```



```

118      #-
119      rat_g_ct_box_df = \
120      rat_g_ct_box_df.drop(columns='genres').rename({'genre': 'genres'},
121      #-
122      rat_g_ct_box_df['genres'] = [', '.join([g, ct])] * len(rat_g_ct_box)
123      #-
124      rat_g_ct_box_df = rat_g_ct_box_df[['rating', 'genres', plot_col]]
125      #-
126      import_order_dict[g+' '+ct] = rat_g_ct_box_df[plot_col].mean()
127      #-
128      all_box_df = pd.concat([all_box_df, rat_g_ct_box_df])
129      #-
130      import_order = \
131      sorted(import_order_dict.keys(), key=lambda k: import_order_dict[k], reverse=True)
132      #-
133      all_box_df = \
134      all_box_df.sort_values('genres',
135                             key=lambda g_col: g_col.map(lambda x: import_order[x]))
136      #-
137      all_box_df = all_box_df.rename(columns={plot_col: aka_v}).reset_index()
138      #-
139      fig, fig_tit = rat_cat_plot(all_box_df, ' combos', aka_v)
140      #-
141      f_name = rat_cat_metric_dir + '2 - ' + fig_tit + '.jpeg'
142      #-
143      fig.savefig(f_name, bbox_inches='tight')
144      plt.close(fig)
145
146      # Important Crew Visualizations
147      #####
148      for g in rat_df.genres:
149          ct = g.split(',')[1]
150          g = g.split(',')[0]
151          #-
152          with out_3:
153              display(print('\u033[1m\u033[32mCat Combo: \u033[0m', g + ', ' + ct),
154                  include=[str]))
155          #-
156          for crew_str, crew_df in zip(crew_strings, crew_dfs):
157              if crew_str != 'actress': plot_type = crew_str + 's'
158              else: plot_type = crew_str + 'es'
159              #
160              crew_dir = rat_cat_metric_dir + plot_type.title() + '/'
161              #
162              if not os.path.isdir(crew_dir):
163                  os.mkdir(crew_dir)
164              #
165              rat_crew_df = crew_df.loc[(crew_df.rating == rat) & (crew_df.genre == g) & (crew_df.creative_type == ct)]
166              #
167              if rat_crew_df.primary_name.nunique() <= 1:
168                  print('\u033[1m\u033[31mRating: \u033[0m', rat)
169                  print('\u033[1m\u033[32mCat Combo: \u033[0m', g + ', ' + ct)
170                  print('\u033[1m\u033[33mCrew Type: \u033[0m', crew_str.title())
171                  print('\u033[1m\u033[33mName: \u033[0m', rat_crew_df.primary_name)
172                  print('-----')
173                  continue
174              #
175          else:
176

```

```

177     crew_count_df = \
178     rat_crew_df.groupby('primary_name').count()[['primary_title']]
179     #
180     if 'tot' in aka_v:
181         crew_agg_df = rat_crew_df.groupby('primary_name').sum()[[p
182     #
183     if 'avg' in aka_v:
184         crew_agg_df = rat_crew_df.groupby('primary_name')\
185                         .mean()[[plot_col]]
186     #
187     c_df = pd.concat([crew_count_df, crew_agg_df], axis=1).reset_i
188         .rename(columns={'primary_title':'num_titles', plot_c
189         .sort_values(aka_v, ascending=False).head(10)
190     #
191     genre_ser = c_df.primary_name.map(lambda x: g+'_'+ct).rename('
192     rat_ser = c_df.primary_name.map(lambda x: rat).rename('rating'
193     #
194     c_df.insert(0, 'genres', genre_ser)
195     c_df.insert(1, 'rating', rat_ser)
196     #
197     if 'tot' in aka_v:
198         c_df['perc_of_'+ aka_v] = \
199             round((c_df[aka_v] / fin_df[plot_col].sum()), 4)
200     #
201     c_df = c_df.reset_index(drop=True).set_index('genres')
202     #
203     fig, fig_tit = rat_cat_plot(c_df, plot_type)
204     #
205     if '/' in g: g_dir = ' - '.join(g.split('/'))
206     else: g_dir = g
207     #
208     if '/' in ct: ct_dir = ' - '.join(ct.split('/'))
209     else: ct_dir = ct
210     #
211     f_name = crew_dir + g_dir +', '+ ct_dir +'.jpeg'
212     #
213     fig.savefig(f_name, bbox_inches='tight')
214     plt.close(fig)
215     #
216     with out_3:
217         display(print('\033[1m\033[32mCat Combo Finsished\033[0m'), includ
218
219     # Clearing non-important information after each rating
220     ######
221     with out_3:
222         clear_output()
223     print()
224
225     # Adding the Additional Information section information
226     ######
227     if not aka_add_info_df.empty:
228         combo_add_info_df = pd.concat([combo_add_info_df, aka_add_info_df[[aka_v]]]

```

Agg Col: Total Worldwide Gross

There was only one associated **Cat Combo** for this **Rating**.

Rating: G

Cat Combo: Adventure,Kids Fiction

Agg Col: Total Worldwide Profits

There was only one associated Cat Combo for this Rating.

Rating: G

Cat Combo: Adventure, Kids Fiction

Agg Col: Average Worldwide Gross

Added data to the Additional Information dataframe for this Rating.

Rating: R

Added data to the Additional Information dataframe for this Rating.

Rating: PG

There was only one associated Cat Combo for this Rating.

Rating: G

Cat Combo: Adventure, Kids Fiction

Agg Col: Average Worldwide Profits

Added data to the Additional Information dataframe for this Rating.

Rating: R

Added data to the Additional Information dataframe for this Rating.

Rating: PG

There was only one associated Cat Combo for this Rating.

Rating: G

Cat Combo: Adventure, Kids Fiction

MPA Ratings Priorities

I used the visualizations created in this section to determine the financial importance of the MPA Ratings to order my recommendations accordingly.

[Click here to see the MPA Ratings visualizations.](#)

```

In [42]: 1  for aka_v in all_akas:
2      plot_col = aka_v[4:]
3      #
4      rat_count_df = fin_df.groupby(['rating']).count()[['primary_title']].drop('Not
5      #
6      if 'tot' in aka_v:
7          aka_start = 'Total'
8          #
9          rat_agg_df = fin_df.groupby(['rating']).sum()[[plot_col]].drop('Not Rated')
10     #
11     if 'avg' in aka_v:
12         aka_start = 'Average'
13         #
14         rat_agg_df = fin_df.groupby(['rating']).mean()[[plot_col]].drop('Not Rated')
15     #
16     rats_df = pd.concat([rat_count_df, rat_agg_df], axis=1).reset_index()\n
17             .sort_values(['rating', plot_col], ascending=[True, False])\
18             .rename(columns={'primary_title':'num_titles', plot_col: aka_v})\n
19             .sort_values(aka_v, ascending=False)
20     #
21     if 'tot' in aka_v:
22         rats_df['perc_of_'+ aka_v] = round((rats_df[aka_v] / fin_df[plot_col].sum())
23     #
24     fig, fig_tit = rat_cat_plot(rats_df, 'rating')
25     #
26     all_rats_dir = 'visuals/All Ratings/'
27     #
28     if not os.path.isdir(all_rats_dir):
29         os.mkdir(all_rats_dir)
30     #
31     f_name = all_rats_dir + fig_tit + '.jpeg'
32     #
33     fig.savefig(f_name, bbox_inches='tight')
34     plt.close(fig)

```

§ Recommendations for Microsoft

Based on my analysis of the financially important categories and subcategories, Microsoft should produce content with the genres, creative types, or specific genre and creative combinations shown, in the order of priority, for each MPA Rating in this section. The ratings themselves are shown in the order of priority that I recommend. By breaking my recommendations down by MPA Rating, my analysis will allow Microsoft to target the various age and audience groups accordingly.

I also created a function to easily access all the Important Crew visualizations that were created, as they were also part of my recommendations. They can use it to find the recommended crew members for the subcategories shown below for each MPA Rating, or those simply deemed important by my analysis. By doing so, they will be maximizing the benefits of the critical insights I gained through my analysis.

Recommendations for PG-13 Content

Priority	Genres	Priority	Creative Types	Priority	Genre & Creative Type Combinations
1.	Action	1.	Super Hero	1.	Action & Super Hero
2.	Adventure	2.	Science Fiction	2.	Adventure & Science Fiction
3.	Thriller / Suspense	3.	Fantasy	3.	Adventure & Fantasy
				4.	Action & Science Fiction
				5.	Action & Contemporary Fiction

[Click here to view the visualizations on which I based the PG-13 recommendations.](#)

Recommendations for PG Content

PG					
Priority	Genres	Priority	Creative Types	Priority	Genre & Creative Type Combinations
1.	Musical	1.	Kids Fiction	1.	Adventure & Kids Fiction
2.	Adventure	2.	Fantasy	2.	Adventure & Fantasy

[Click here to view the visualizations on which I based the PG recommendations.](#)

Recommendations for R Content

R					
Priority	Genres	Priority	Creative Types	Priority	Genre & Creative Type Combinations
1.	Horror	1.	Science Fiction	1.	Action & Science Fiction
2.	Black Comedy	2.	Contemporary Fiction	2.	Comedy & Contemporary Fiction
3.	Action			3.	Horror & Fantasy
4.	Comedy			4.	Action & Contemporary Fiction

[Click here to view the visualizations on which I based the R recommendations.](#)

Recommendations for G Content

G	
Priority	Genres
1.	Adventure
Priority	Creative Types
1.	Kids Fiction
Priority	Genre & Creative Type Combinations
1.	Adventure & Kids Fiction

There was only one important subcategory for each subcategory type, and therefore no need for exploratory visualizations.

Important Subcategories & Crew Interactive Visualization Explorer

My final recommendation for Microsoft is to use this feature to find the most important crew members who have directed, written, or acted in the recommended subcategories for each of MPA Ratings shown above.

I created this feature, which I have duplicated in VBA in my presentation, to allow anyone running this to easily find those important crew members. In order to reset the search menus and explore another important category, subcategory, or crew member type, all one would have to do is just re-run the cell.

If you are unable to run the code or open the presentation to use this feature, I recreated the crew-finding capabilities of the feature, for the categories that I recommended at least, in the section below in the Markdown file to make it easier. [Click here to go to that section now.](#) (_____)

By using this feature, Microsoft can take full advantage of the insights gained through my analysis and they will be provided with all of the information necessary to make a forceful entry into the streaming market.

In [43]:

```
1 rat_choices = ['Choose a MPA Rating from the choices below...',  
2                 'PG-13', 'PG', 'R', 'G']  
3 #-----  
4 cat_choices = ['Choose a category type from the choices below...',  
5                 'Important Genres', 'Important Creative Types',  
6                 'Important Genre & Creative Type Combinations']  
7 #-----  
8 metric_choices = ['Choose a financial metric from the choices below...',  
9                     'Total Worldwide Gross', 'Total Worldwide Profits',  
10                    'Average Worldwide Gross', 'Average Worldwide Profits']  
11 #-----  
12 crew_choices = ['Choose a type of crew member from the choices below...',  
13                  'Directors', 'Writers', 'Actors', 'Actresses']  
14 #-----  
15 empty_slot = ['N/A until a selection is made in dropdown menu above']  
16 #-----  
17 menu_layout = widgets.Layout(width='initial', height='30px')  
18 #-----  
19 first_menu = \  
20 widgets.Dropdown(options=rat_choices, value=rat_choices[0], disabled=False, layout=menu_layout)  
21 #-----  
22 second_menu = \  
23 widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)  
24 #-----  
25 third_menu = \  
26 widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)  
27 #-----  
28 fourth_menu = \  
29 widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)  
30 #-----  
31 fifth_menu = \  
32 widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)  
33 #-----  
34 sixth_menu = \  
35 widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)  
36  
37 # Function monitoring changes to the widgets  
38 #####  
39 def f(a, b, c, d, e, f):  
40  
41     # First Menu  
42     #####  
43     if a not in [rat_choices[0], empty_slot[0]]:  
44         f_path_p1 = a + '/'  
45         #-----  
46         second_menu.options = cat_choices  
47         first_menu.disabled = True  
48     #-----  
49     else: second_menu.options = empty_slot  
50  
51     # Second Menu  
52     #####  
53     if b not in [cat_choices[0], empty_slot[0]]:  
54         if 'Combinations' in b: f_path_p2 = 'Important Combinations/'  
55         else: f_path_p2 = b + '/'  
56         #-----  
57         third_menu.options = metric_choices  
58         second_menu.disabled = True
```

```

59      #-----
60      else: third_menu.options = empty_slot
61
62      # Third Menu
63      #####
64      fourth_choice_1_a = 'Do you wish to explore the '
65      fourth_choice_1_b = ' themselves, or the Important Crew from the films with the'
66      fourth_choice_1_c = '?'
67      #-----
68      if c not in [metric_choices[0], empty_slot[0]]:
69          fourth_choice_insert = b.lstrip('Important ')
70          #-
71          fourth_choice_1 = fourth_choice_1_a + fourth_choice_insert + fourth_choice_
72                      fourth_choice_insert + fourth_choice_1_c
73          #-
74          fourth_choices = [fourth_choice_1, fourth_choice_insert, 'Important Crew']
75          #-
76          if a == rat_choices[-1]:
77              fourth_choice_1 = 'Click here and choose Important Crew, as there were
78                          'exploratory visualizations for this Rating.'
79          #-
80          fourth_choices = [fourth_choice_1, 'Important Crew']
81          #-
82          fourth_menu.options = fourth_choices
83          #-
84          f_path_p3 = c + '/'
85          #-
86          third_menu.disabled = True
87          #-
88      else:
89          fourth_menu.options = empty_slot
90          fourth_choices = ['empty']
91
92      # Fourth Menu
93      #####
94      if d in fourth_choices:
95          if a != rat_choices[-1] and d == fourth_choices[1]:
96              file_path = 'visuals/' + f_path_p1 + f_path_p2 + f_path_p3
97              file_choices = listdir(file_path)[:2]
98              #-
99              file_choices.insert(0, 'Please choose a file from the choices below...')
100             #
101             fifth_menu.options = file_choices
102             fourth_menu.disabled = True
103             #-
104             if (a == rat_choices[-1] and d == fourth_choices[1]) or \
105                 (a != rat_choices[-1] and d == fourth_choices[2]):
106                 fifth_menu.options = crew_choices
107                 fourth_menu.disabled = True
108             #-
109             else: fifth_menu.options = empty_slot
110
111             # Fifth Menu
112             #####
113             if '.jpeg' in e:
114                 file_name = file_path + e
115                 #-
116                 image = Image.open(file_name)
117                 display(image)

```

```

118      #-
119      sixth_menu.disabled = True
120      #-
121      elif e in crew_choices[1:]:
122          file_path = 'visuals/' + f_path_p1 + f_path_p2 + f_path_p3 + e + '/'
123          file_choices = listdir(file_path)
124          #-
125          file_choices.insert(0, 'Please choose a file from the choices below... ')
126          #-
127          sixth_menu.options = file_choices
128          fifth_menu.disabled = True
129          #-
130      else: sixth_menu.options = empty_slot
131
132      # Sixth Menu
133      #####
134      if '.jpeg' in f:
135          file_name = file_path + f
136          #-
137          image = Image.open(file_name)
138          display(image)
139
140      # Creating the interactive widget
141      #####
142      ui = widgets.VBox([first_menu, second_menu, third_menu, fourth_menu, fifth_menu, sixth_menu])
143      #-
144      out = widgets.interactive_output(f, {'a':first_menu, 'b':second_menu, 'c':third_menu,
145                                         'd':fourth_menu, 'e':fifth_menu, 'f':sixth_menu})
146      #-
147      print('\n\033[1m    Follow the instructions that appear in each dropdown menu to a
148          'visualization with the desired properties.\n\n    To start a new search, '+
149          'just re-run the cell.\n')
150      display(ui, out)

```

Follow the instructions that appear in each dropdown menu to access the visualization with the desired properties.

To start a new search, just re-run the cell.

PG-13

Important Genre & Creative Type Combinations

Total Worldwide Profits

Important Crew

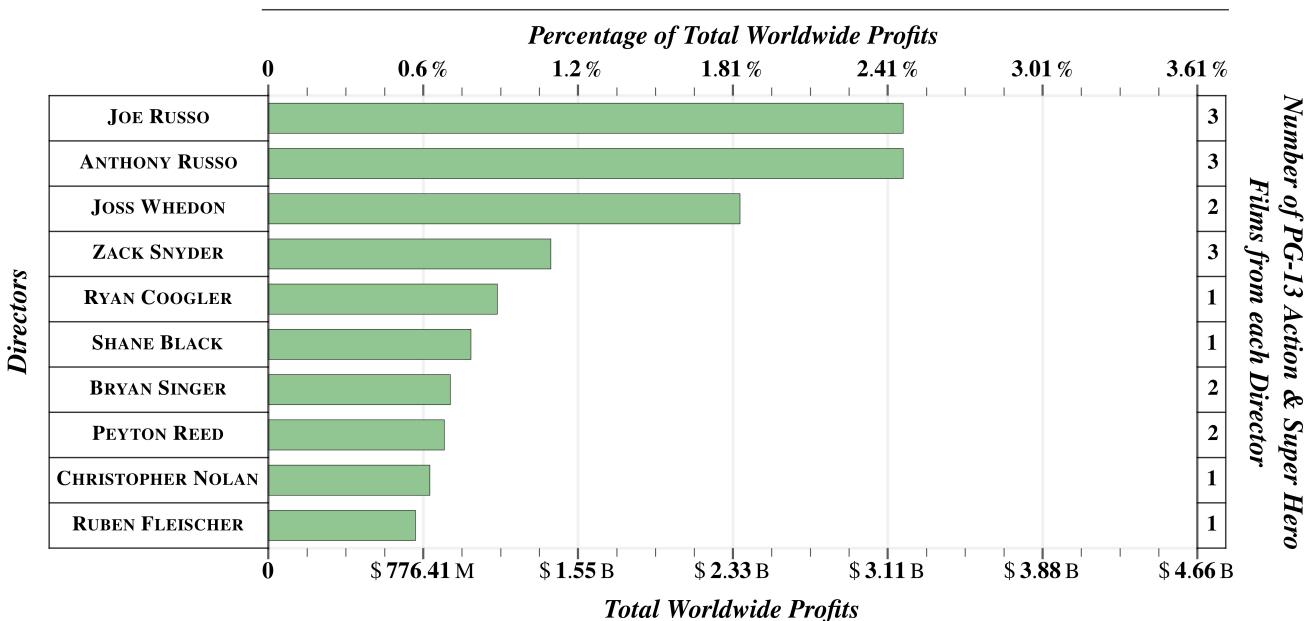
Directors

Action, Super Hero.jpeg

OPUSDATA CATEGORICAL DATA
IMDb DIRECTORS DATA • THE NUMBERS FINANCIAL DATA
2010 → 2018

Top 10 Directors of PG-13 Action & Super Hero Films

Ranked by Total Worldwide Profits



§ Additional Insights

As I mentioned in the [Standards for Qualifying as Important](#) section, if there were less than nine movies made with a subcategory, but that subcategory's value for a metric was **greater than the max value of the Top 3** for that metric, it was added to this section. I again created more markdown tables to be display the information in the notebook.

In [44]:

```

1 pg_add_info_df = pd.DataFrame()
2 r_add_info_df = pd.DataFrame()
3 #-
4 subcat_add_infos = [g_add_info_df, ct_add_info_df, combo_add_info_df]
5 #-
6 new_aka_cols = ['Amount Greater<br>than the<br>Top Average<br>Worldwide Gross',
7                 'Amount Greater<br>than the<br>Top Average<br>Worldwide Profits']
8 rest_cols = ['rating', 'genres', 'creative_type', 'num_titles']
9 #-
10 for df_i, add_info_df in enumerate(subcat_add_infos):
11     add_info_df = add_info_df.reset_index()
12     #-
13     aka_cols = [col for col in add_info_df.columns if col in all_akas]
14     idx_cols = [col for col in add_info_df.columns if col in rest_cols]
15     #-
16     for rat in add_info_df.rating.unique():
17         rat_add_df = add_info_df.loc[add_info_df.rating == rat][idx_cols + aka_cols]
18         #-
19         rat_add_df = \
20             rat_add_df.sort_values(rat_add_df.columns[-1], ascending=False).reset_index()
21         #-
22         if df_i == 2:
23             combo_ser = \
24                 rat_add_df.apply(lambda x: ' & '.join((x['genres'], x['creative_type'])))
25             #-
26             rat_add_df.insert(1, 'Genre & Creative Type<br>Combinations', combo_ser)
27             #-
28             rat_add_df = rat_add_df.drop(columns=['genres', 'creative_type'])
29         #-
30         for col in rest_cols:
31             if col in rest_cols[:2]: new_col = col.title()
32             if col == rest_cols[2]: new_col = ' '.join(col.split('_')).title()
33             if col == rest_cols[3]: new_col = 'Number of Titles'
34             #-
35             rat_add_df = rat_add_df.rename(columns={col: new_col})
36         #-
37         rat_add_df = rat_add_df.rename(columns={all_akas[2]: new_aka_cols[0],
38                                             all_akas[3]: new_aka_cols[1]})
39         #-
40         rat_add_df[new_aka_cols] = rat_add_df[new_aka_cols].applymap(lambda x: tab)
41         #-
42         priority = [1] if len(rat_add_df) == 1 else range(1, len(rat_add_df) + 1)
43         priority = [str(p) + '&period;' for p in priority]
44         rat_add_df.insert(0, 'Priority', priority)
45         #-
46         rat_add_df = rat_add_df.set_index('Priority')
47         #-
48         align_cols = ['center'] * (len(rat_add_df.columns) + 1)
49         #-
50         print(rat_add_df.to_markdown(tablefmt='pipe', colalign=align_cols))
51         print('\n\n\n')

```

Priority	Rating	Genres	Number of Titles	Amount Greater than the Top Average Worldwide Gross	Amount Greater than the Top Average Worldwide Profits

Additional Insights for PG Content

If any subcategory with the PG Rating had less than nine titles, but a greater value in a metric than the top subcategory in the corresponding visualization for that metric, its value was included in the corresponding table below.

Priority	Rating	Creative Type	Number of Titles	Amount Greater than the Top Average Worldwide Gross	Amount Greater than the Top Average Worldwide Profits
1.	PG	Super Hero	1	\$ 288.15 M	\$ 216.13 M
Priority	Rating	Genre & Creative Type Combinations	Number of Titles	Amount Greater than the Top Average Worldwide Gross	Amount Greater than the Top Average Worldwide Profits
1.	PG	Adventure & Super Hero	1	\$ 294.52 M	\$ 220.36 M
2.	PG	Musical & Kids Fiction	4	\$ 271.63 M	\$ 199.96 M
3.	PG	Musical & Fantasy	3	\$ 135.60 M	\$ 148.53 M
4.	PG	Musical & Dramatization	1	\$ 29.06 M	\$ 35.90 M

Additional Insights for R Content

If any subcategory with the R Rating had less than nine titles, but a greater value in a metric than the top subcategory in the corresponding visualization for that metric, its value was included in the corresponding table below.

Priority	Rating	Genres	Number of Titles	Amount Greater than the Top Average Worldwide Gross	Amount Greater than the Top Average Worldwide Profits
1.	R	Adventure	7	\$ 96.68 M	\$ 79.83 M
Priority	Rating	Creative Type	Number of Titles	Amount Greater than the Top Average Worldwide Gross	Amount Greater than the Top Average Worldwide Profits

Priority	Rating	Creative Type	Number of Titles	Amount Greater than the Top Average Worldwide Gross	Amount Greater than the Top Average Worldwide Profits
1.	R	Super Hero	3	\$ 600.49 M	\$ 554.98 M
Priority	Rating	Genre & Creative Type Combinations	Number of Titles	Amount Greater than the Top Average Worldwide Gross	Amount Greater than the Top Average Worldwide Profits
1.	R	Action & Super Hero	3	\$ 559.88 M	\$ 523.74 M
2.	R	Horror & Historical Fiction	2	\$ 177.74 M	\$ 212.43 M
3.	R	Adventure & Science Fiction	1	\$ 227.94 M	\$ 165.13 M
4.	R	Adventure & Dramatization	2	\$ 146.44 M	\$ 126.13 M
5.	R	Black Comedy & Dramatization	3	\$ 59.34 M	\$ 71.20 M
6.	R	Comedy & Factual	1		\$ 39.37 M
7.	R	Horror & Science Fiction	4	\$ 9.64 M	\$ 24.70 M
8.	R	Adventure & Contemporary Fiction	4		\$ 2.96 M

END OF PROJECT