

## § 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](#)
- [Functions Created for Visualizations](#)
- [Standards for Qualifying as Important](#)
- [Analysis Preparation](#)
- [IMDb Category Types](#)
- [OpusData Category Types](#)
- [Final Analysis and Recommendations](#)
  - [Recommendations to Maximize Subscribers](#)
  - [Recommendations to Maximize Profits](#)
  - [Interactive Visualization Feature](#)
- [Markdown Crew Exploration](#)

## § 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 the viewing preferences of moviegoers, and then produce movies and/or television shows based on the insights gained from such an analysis.

I have performed such an analysis based on data from the sources shown above. The data was provided to me by [Flatiron School](https://flatironschool.com/) (<https://flatironschool.com/>) and [OpusData](https://www.opusdata.com/) (<https://www.opusdata.com/>).

I explored the financial performance of movies based on the category types listed below, the sources of which I have included as links and will discuss further in the next section.

- [IMDb](https://www.imdb.com/) (<https://www.imdb.com/>) - Unique Genre Combinations
- [IMDb](https://www.imdb.com/) (<https://www.imdb.com/>) - Individual Genres
- [OpusData](https://www.opusdata.com/) (<https://www.opusdata.com/>) - Creative Types
- [OpusData](https://www.opusdata.com/) (<https://www.opusdata.com/>) - Genres

For each of the category types listed above, I identified and explored the highest performing categories based on the following metrics, the sources of which I have included as links and will discuss further in the next section:

- [The Numbers](https://www.the-numbers.com/) (<https://www.the-numbers.com/>) - Total Worldwide Gross Revenue
- [The Numbers](https://www.the-numbers.com/) (<https://www.the-numbers.com/>) - Total Worldwide Profits
- [The Numbers](https://www.the-numbers.com/) (<https://www.the-numbers.com/>) - Average Worldwide Gross Revenue
- [The Numbers](https://www.the-numbers.com/) (<https://www.the-numbers.com/>) - Average Worldwide Profits

I then determined which of the [Motion Picture Association](https://www.motionpictures.org/) (<https://www.motionpictures.org/>)'s ratings performed the best based on the above metrics for each of the highest performing categories. I also determined the highest performing directors and writers of, as well as the highest performing actors and actresses who performed in, films from those high performing categories.

My analysis will be able to provide the Microsoft team assigned to lead their entry into the streaming market with strategic insights into the viewing tendencies of moviegoers. These insights will guide them as to the type of content they may wish to produce to generate the highest possible levels of financial returns.

## § Exploring and Preparing the Data

---

For this project I was provided data from [IMDb](https://www.imdb.com/) (<https://www.imdb.com/>), [The Movie DataBase \(TMDB\)](https://www.themoviedb.org/) (<https://www.themoviedb.org/>), [The Numbers](https://www.the-numbers.com/) (<https://www.the-numbers.com/>), [Box Office Mojo](https://www.boxofficemojo.com/)

(<https://www.boxofficemojo.com/>), and [Rotten Tomatoes \(<https://www.rottentomatoes.com/>\)](https://www.rottentomatoes.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, to which it provides data services. I incorporated data from IMDb, OpusData, and The Numbers. I chose not to use any of the data from TMDB, Box Office Mojo, or Rotten Tomatoes.

The Numbers and Box Office Mojo datasets both contained financial information, but after exploring each of them I determined that The Numbers data was more useful as it contained a column named `production_budget`, which could be used to determine profits, while the Box Office Mojo data did not have such information, diminishing its value and essentially making its use redundant.

While the data from TMDB could have been useful in analyzing the popularity of, or interest in, the categories within the previously mentioned category types, but as that was not the purpose of this project, I decided not to use it.

I was originally planning to use the Rotten Tomatoes data as it did contain useful data. First, it contained a `rating` column with the [Motion Picture Association \(<https://www.motionpictures.org/>\)](https://www.motionpictures.org/)'s ratings which could have been a valuable categorical variable. It also contained data on critics' reviews, specifically if the review was 'fresh' or 'rotten' (good or bad, respectively), which could then be used to calculate their Tomatometer ranking system, which is based on the percentage of fresh reviews.

However, the number of movies contained within the data from Rotten Tomatoes was much smaller in size by comparison, and an even smaller number of those movies could be directly compared with the insights gained through my financial analysis. This was because there was no column containing the title information for the movies, and any comparison to the financial analysis dataframes I created would have to be performed based purely on the even more limited `theater_date` column. The resulting dataset from such a comparison was about half the length or less when compared with the length of the financial analysis dataset, thereby severely limiting the value of the data, and perhaps even resulting in a misleading analysis. All the reasons stated above led to me to exclude the Rotten Tomatoes data from my final analysis.

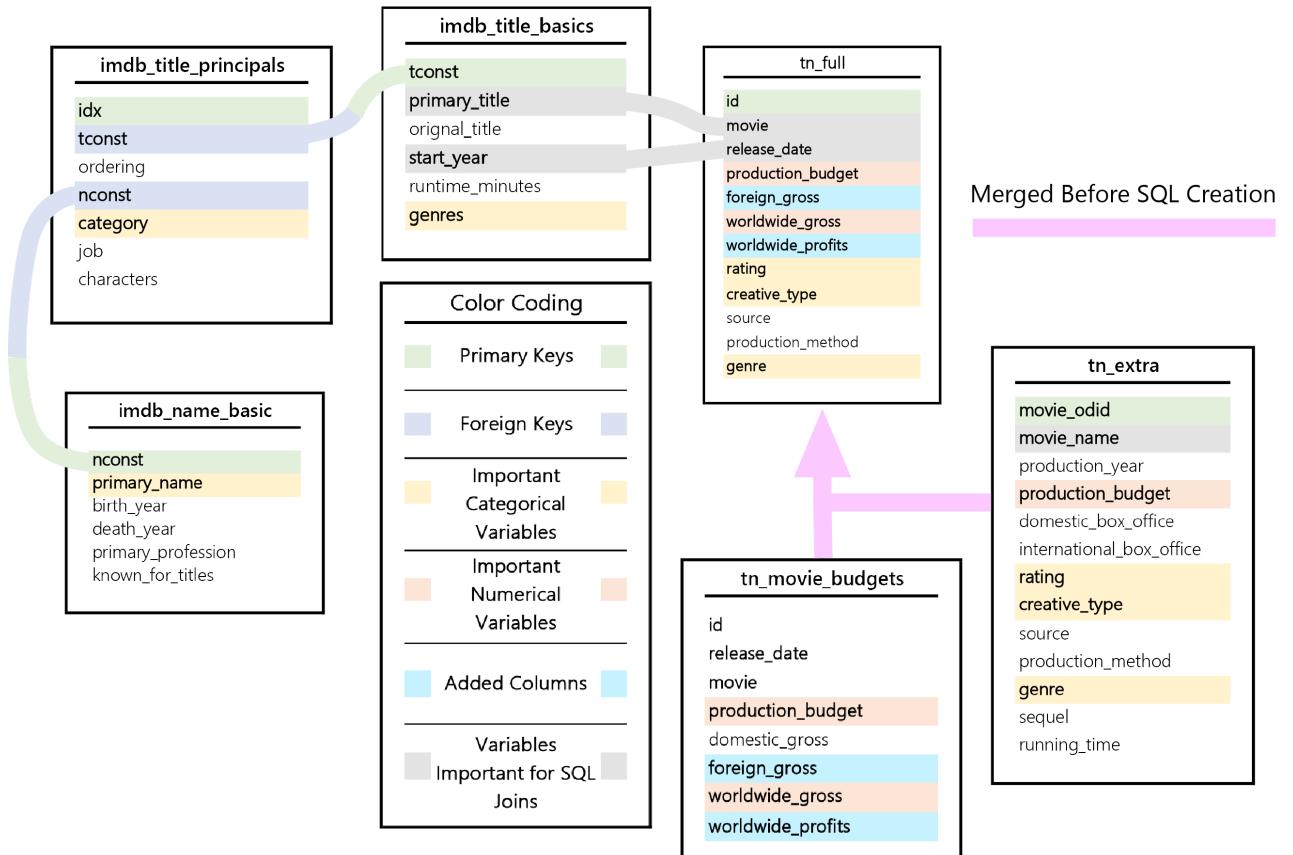
Those reasons are also why I decided to use the data I received from OpusData. Not only did it have a `rating` column, but it had even more movies than the entire Rotten Tomatoes dataset, all of which were comparable as it contained a `title` column as well as similar columns to those in The Numbers data. It also had the added benefit of providing another category type, Creative Type, upon which to analyze performance.

## Importing and Investigating the Data

After importing and investigating the datasets provided to me, I chose to use the data from the tables shown in the image below for my analysis. The image also shows how I used the tables to create an SQL database so that different columns from the various tables could be joined together to build the dataframes that I would need to properly analyze the data and create the appropriate visualizations. Important columns for either joining or analyzing are identified. Also included are any new columns that I created, based on data already in the corresponding table, to enhance my analysis.

There were three other tables containing data from IMDb that I chose not to include in my database. The first table contained movies with the director(s) and/or writer(s) for each movie. As that information was already present in the `imdb_title_principals` table, it was unnecessary to include. The second table contained data regarding the various translations of titles for specific regions of the world, which did not contribute to my analysis. The third table contained online ratings data similar to the data in the TMDB dataset that I chose not to include, which could similarly prove useful for future analysis, but was not useful in a financial analysis.

I merged the `tn_movie_budgets` and `tn_extra` tables into the `tn_full` table prior to creating the SQL database.



```
In [1]: 1 # Importing the necessary modules, setting up the mapping function for passing SQL queries into,
2 # and enabling inline plotting for when I was building the plotting functions
#####
3 import os
4 from os import listdir
5 import string
6 import numpy as np
7 import pandas as pd
8 import sqlite3
9
10 #-----
11 from pandasql import sqldf
12 pysqldf = lambda q: sqldf(q, globals())
13 #-----
14 import matplotlib.pyplot as plt
15 import matplotlib.lines as mlines
16 import matplotlib.patches as mpatches
17 from matplotlib import cm
18 from matplotlib.colors import LinearSegmentedColormap
19 #-----
20 import ipywidgets as widgets
21 #-----
22 from PIL import Image
23 #-----
24 %matplotlib inline
```

```
In [2]: 1 # Importing the data into dataframes
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 # Importing the extra data I received from The Numbers's data service, Opus Data
2 #####
3 tn_extra = pd.read_csv('data/Opus Data/MovieData.csv')
```

```
In [4]: 1 # Setting up lists of the dataframes and their names to iterate through while checking out the
2 # data and while creating the SQL database
3 #####
4 df_names = ['imdb_title_principals', 'imdb_title_basics', 'imdb_name_basic', 'tn_movie_budgets',
5             'tn_extra']
6 #-
7 all_dfs = [imdb_title_principals, imdb_title_basics, imdb_name_basic, tn_movie_budgets, tn_extra]
8 #-
9 for name, df in zip(df_names, all_dfs):
10     print('\033[1m'+ name + ':'+'\033[0m')
11     display(df.head())
12     display(df.info())
13     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 integer
 1   ordering    1028186 non-null integer
 2   nconst      1028186 non-null integer
 3   category    1028186 non-null object 
 4   job         1028186 non-null object 
 5   characters  1028186 non-null object 
```

## Fixing and Adding Data

In order to perform my analysis, changes needed to be made to some of the columns within a few of the dataframes created from the tables shown above. 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. I then could add other financial columns based on those columns that could be of significance to my or future analysis. I also changed any columns containing any date or year data to the same format so that they could be easily compared. Finally, I rearranged the column order of one of the dataframes to make it ready for database construction.

In [5]:

```
1 # Removing any characters that would get in the way of changing columns with financial data
2 # to a float dtype
3 #####
4 finan_cols = ['production_budget', 'domestic_gross', 'worldwide_gross']
5 -----
6 for col in finan_cols:
7     tn_movie_budgets[col] = tn_movie_budgets[col].str.strip('$')
8     tn_movie_budgets[col] = tn_movie_budgets[col].str.replace(',', '')
9 -----
10 tn_movie_budgets[finan_cols] = tn_movie_budgets[finan_cols].astype(float)
11 -----
12 display(tn_movie_budgets.head(3))
13 display(tn_movie_budgets.info())
```

|          | <b>id</b> | <b>release_date</b> | <b>movie</b>                                | <b>production_budget</b> | <b>domestic_gross</b> | <b>worldwide_gross</b> |
|----------|-----------|---------------------|---|--------------------------|-----------------------|------------------------|
| <b>0</b> | 1         | Dec 18, 2009        | Avatar                                      | 425000000.0              | 760507625.0           | 2.776345e+09           |
| <b>1</b> | 2         | May 20, 2011        | Pirates of the Caribbean: On Stranger Tides | 410600000.0              | 241063875.0           | 1.045664e+09           |
| <b>2</b> | 3         | Jun 7, 2019         | Dark Phoenix                                | 350000000.0              | 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 
dtypes: float64(3), int64(1), object(2)
memory usage: 271.2+ KB
```

None

In [6]:

```

1 # Adding financial columns that I may want to use in my analysis
2 #####
3 foreign_gross = tn_movie_budgets['worldwide_gross'] - tn_movie_budgets['domestic_gross']
4 #
5 tn_movie_budgets.insert(5, 'foreign_gross', foreign_gross)
6 #
7 tn_movie_budgets['worldwide_profits'] = tn_movie_budgets['worldwide_gross'] - \
8                                     tn_movie_budgets['production_budget']
9 #
10 display(tn_movie_budgets.head())
11 display(tn_movie_budgets.info())

```

|          | <b>id</b> | <b>release_date</b> | <b>movie</b>                                | <b>production_budget</b> | <b>domestic_gross</b> | <b>foreign_gross</b> | <b>worldwide_gross</b> | <b>worldwide_profits</b> |
|----------|-----------|---------------------|---|--------------------------|-----------------------|----------------------|------------------------|--------------------------|
| <b>0</b> | 1         | Dec 18, 2009        | Avatar                                      | 425000000.0              | 760507625.0           | 2.015838e+09         | 2.776345e+09           | 2.351345e+09             |
| <b>1</b> | 2         | May 20, 2011        | Pirates of the Caribbean: On Stranger Tides | 410600000.0              | 241063875.0           | 8.046000e+08         | 1.045664e+09           | 6.350639e+08             |
| <b>2</b> | 3         | Jun 7, 2019         | Dark Phoenix                                | 350000000.0              | 42762350.0            | 1.070000e+08         | 1.497624e+08           | -2.002376e+08            |
| <b>3</b> | 4         | May 1, 2015         | Avengers: Age of Ultron                     | 330600000.0              | 459005868.0           | 9.440081e+08         | 1.403014e+09           | 1.072414e+09             |
| <b>4</b> | 5         | Dec 15, 2017        | Star Wars Ep. VIII: The Last Jedi           | 317000000.0              | 620181382.0           | 6.965404e+08         | 1.316722e+09           | 9.997217e+08             |

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 8 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   foreign_gross    5782 non-null    float64 
 6   worldwide_gross  5782 non-null    float64 
 7   worldwide_profits 5782 non-null    float64 
dtypes: float64(5), int64(1), object(2)
memory usage: 361.5+ KB

```

None

In [7]:

```

1 # Changing all columns with any sort of time data to the same format to make comparison easy
2 #####
3 tn_movie_budgets.release_date = \
4 pd.to_datetime(tn_movie_budgets.release_date).map(lambda x: x.strftime("%m/%d/%Y"))
5 #
6 tn_movie_budgets.release_date = \
7 pd.to_datetime(tn_movie_budgets.release_date, format="%m/%d/%Y")
8 #
9 display(tn_movie_budgets.release_date.head(3))
10 print(tn_movie_budgets.release_date.dt.year.min(), tn_movie_budgets.release_date.dt.year.max())
11 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

```

In [8]:

```
1 # Rearranging the column order of OpusData dataframe so that 'movie_odid' column was first
2 #####
3 tn_cols = [tn_extra.columns[2]] + list(tn_extra.columns[:2]) + list(tn_extra.columns[3:])
4 tn_extra = tn_extra[tn_cols]
5 -----
6 tn_extra.head(3)
```

Out[8]:

|   | movie_odid | movie_name             | production_year | production_budget | domestic_box_office | international_box_office | rating    | creative  |
|---|------------|------------------------|-----------------|-------------------|---------------------|--------------------------|-----------|-----------|
| 0 | 8220100    | Madea's Family Reunion | 2006            | 10000000          | 63257940            | 62581                    | PG-13     | Contemp F |
| 1 | 58540100   | Krrish                 | 2006            | 10000000          | 1430721             | 31000000                 | Not Rated | Sci F     |
| 2 | 34620100   | End of the Spear       | 2006            | 10000000          | 11748661            | 175380                   | PG-13     | His F     |

## Limits the Year Range

By limiting the year range to the past decade, my analysis would be based on the most relevant date in terms of the current viewing habits of moviegoers. I also did not want any data from the past two years due to the disruptions to the industry caused by COVID-19.

In [9]:

```
1 # Limiting any movies to the past ten years
2 #####
3 imdb_title_basics = imdb_title_basics.loc[(imdb_title_basics.start_year > 2009) & \
4                                              (imdb_title_basics.start_year < 2020)]
5 -----
6 tn_movie_budgets = tn_movie_budgets.loc[(tn_movie_budgets.release_date.dt.year > 2009) &
7                                         (tn_movie_budgets.release_date.dt.year < 2020)]
```

## 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 # Merging The Numbers data with OpusData data
2 #####
3 tn_p1 = tn_movie_budgets.set_index('movie')
4 tn_p2 = tn_extra.set_index('movie_name')
5 #-
6 tn_full = \
7 tn_p1.merge(tn_p2, how='outer', left_index=True, right_index=True)\n8 .reset_index().set_index('id').reset_index()
9 #-
10 tn_full = tn_full.rename({'index': 'movie'}, axis=1)
11 #-
12 tn_full = \
13 tn_full.drop(tn_full.loc[(tn_full.production_budget_x != tn_full.production_budget_y) & \
14 (tn_full.release_date.dt.year != 2019)].index)
15 #-
16 tn_full = \
17 tn_full.drop(tn_full.loc[(tn_full.domestic_gross != tn_full.domestic_box_office) & \
18 (tn_full.foreign_gross != tn_full.international_box_office) & \
19 (tn_full.release_date.dt.year != 2019)].index)
20 #-
21 cols_to_drop = ['production_year', 'movie_omid', 'production_budget_y', 'domestic_box_office',
22 'international_box_office', 'sequel', 'running_time']
23 #-
24 tn_full = tn_full.drop(columns=cols_to_drop)\n25 .rename({'production_budget_x': 'production_budget'}, axis=1)
26 #-
27 display(tn_full.head(3))
28 display(tn_full.info())
```

|   |      | id                    | movie | release_date | production_budget | domestic_gross | foreign_gross | worldwide_gross | worldwide_profits | rating |
|---|------|-----------------------|-------|--------------|-------------------|----------------|---------------|-----------------|-------------------|--------|
| 2 | 48.0 | 10 Days in a Madhouse |       | 2015-11-11   | 12000000.0        | 14616.0        | 0.0           | 14616.0         | -11985384.0       | F      |
| 5 | 64.0 | 12 Strong             |       | 2018-01-19   | 35000000.0        | 45819713.0     | 25298665.0    | 71118378.0      | 36118378.0        | F      |
| 6 | 18.0 | 12 Years a Slave      |       | 2013-10-18   | 20000000.0        | 56671993.0     | 124353350.0   | 181025343.0     | 161025343.0       | F      |

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1285 entries, 2 to 2884
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   id               1285 non-null   float64
 1   movie             1285 non-null   object  
 2   release_date     1285 non-null   datetime64[ns]
 3   production_budget 1285 non-null   float64
 4   domestic_gross   1285 non-null   float64
 5   foreign_gross    1285 non-null   float64
 6   worldwide_gross  1285 non-null   float64
 7   worldwide_profits 1285 non-null   float64
 8   rating            1219 non-null   object  
 9   creative_type    1222 non-null   object  
 10  source            1220 non-null   object  
 11  production_method 1224 non-null   object  
 12  genre             1226 non-null   object  
dtypes: datetime64[ns](1), float64(6), object(6)
memory usage: 140.5+ KB
```

None

## Turning the DataFrames into a SQL Database

Once the dataframes were prepared, I created a SQL database file and populated it with the dataframes. I then checked to make sure everything functioned correctly.

```
In [11]: 1 # Recreating the lists for database creation to include tn_full
2 #####
3 df_names = ['imdb_title_principals', 'imdb_title_basics', 'imdb_name_basic', 'tn_full']
4 #-----
5 all_dfs = [imdb_title_principals, imdb_title_basics, imdb_name_basic, tn_full]
```

```
In [12]: 1 # Creating a connection to a new SQL database
2 #####
3 mov_conn = sqlite3.Connection('movies_db.sqlite')
4 mov_cur = mov_conn.cursor()
```

```
In [13]: 1 # Populating the database with the tables, resetting the index if the first column had any
2 # duplicates
3 #####
4 for t_i, (table, table_name) in enumerate(zip(all_dfs, df_names)):
5     if table[table.columns[0]].duplicated().any():
6         table_copy = table.reset_index().rename({'index':'idx'}, axis=1)
7     else: table_copy = table.copy()
8     #-----
9     table_copy.to_sql(table_name, mov_conn, if_exists='replace', index=False)
10    #
11 print(mov_cur.execute('SELECT name FROM sqlite_master').fetchall())
```

[('imdb\_title\_principals',), ('imdb\_title\_basics',), ('imdb\_name\_basic',), ('tn\_full',)]

```
In [14]: 1 # Making sure everything worked
2 #####
3 for table, table_name in zip(all_dfs, df_names):
4     df = pd.DataFrame(mov_cur.execute('SELECT * FROM ' + table_name + ';').fetchall())
5     df.columns = [x[0] for x in mov_cur.description]
6     #
7     print('\033[1m'+ table_name + ':'+'\033[0m')
8     display(df.head(3))
9     display(df.info())
10    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  

```

## § 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 [15]: 1 # Setting up a list of spine sides to iterate through to alter their appearances
2 #####
3 light_spine_sides = ['top', 'bottom']
4 dark_spine_sides = ['left', 'right']
5
6 # Updating matplotlib rcParams to the desired settings
7 # I had to include the translation package b/c of one director's name had a non-recognized
8 # character that caused problems when converting to LaTeX
9 #####
10 plt.rcParams.update({'text.usetex': True,
11                      'text.latex.preamble': r'\usepackage{ucs}'+\
12                                         r'\usepackage[T1]{fontenc}'+\
13                                         r'\usepackage[utf8]{inputenc}'+\
14                                         r'\usepackage[icelandic,english]{babel}'+\
15                                         r'\usepackage{amsmath}',

16                      'font.family': ['sans-serif', 'serif'],
17                      'font.sans-serif': ['Verdana'],
18                      'font.serif': ['New Century Schoolbook'],
19                      'mathtext.fontset': 'cm'})

20
21 # Uncomment to check and/or change keys
22 #####
23 # 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.

```
In [16]: 1 # Function to get a color between a given color and pure white based on a number between 0 and 1.
2 # A higher light_frac results in lighter color.
3 #####
4 def get_lighter_color(color_name, light_frac):
5     color_map = LinearSegmentedColormap.from_list('', [color_name, 'w'], N=9)
6     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 any visualizations that analyzed the subcategory's performance within each of main categories.

```
In [17]: 1 # Ratings color dictionary
2 #####
3 uniq_ratings = tn_full.rating.unique()
4 -----
5 ratings_colors_dict = {'Not Rated': 'palegreen',
6                         'G': 'orchid',
7                         'PG': 'xkcd:aqua_blue',
8                         'PG-13': 'gold',
9                         'R': 'tomato'}
```

# Tick Params Functions

These functions are 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 plot I needed to create.

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

```
In [19]: 1 # Function for a Tick Params Settings for Axes with a Dual Numerical X Axis on the top
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 [20]: 1 # Function for a Tick Params Settings for Axes with a Dual Non-Numerical Y Axis on the right
2 #####
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,.63))
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 [21]:

```
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{24}{0}\selectfont{\scshape{\$}}}''
6     #
7     if x>=1e9 or x<=-1e9:
8         s_end = r'{\fontsize{22.5}{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{22.5}{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{22.5}{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 [22]:

```
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 [23]: 1 # Function to format simple numerical tick labels in LaTeX format for use on ticks on the Y Axis
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 Tick Finders & Quantile 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 [24]: 1 # Function to find a small positive quantile to be passed into the max tick finder in an
2 # aggregate type plot
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*2
16         else: test_quant = test_quant + .06*(i / 2)
17         i += 1
18     #
19     return test_value, test_quant, last_neg
```

```
In [25]: 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 [26]:

```
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         #
10        if new_max > 1:
11            if new_max <= int(test_v):
12                final_max = int(test_v)
13                max_found = True
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*2
21    else: test_v = test_v + max_step*(i - 3)
22    i += 1
23    #
24 return final_max
```

## Main Visualization Function

This is the only function for creating visualizations that I used for this project, as I used it for every plot that I created. 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 all plotting to run 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 [27]:

```
1 # Plotting Function for all plots with identifying strings passed through
2 #####
3 def cat_plot(plot_df, viz_type, plot_type, box=None):
4
5     # Starting variables & variables based on which num_type str was passed thru and if box=True
6 #####
7     plot_height = 9
8     perc_plot = False
9     tit_bullet = r'{\fontsize{30}{0}\selectfont{\textbf{ \textbullet\ }}}'
10    #
11    if plot_type=='creative_type': main_cat_col = plot_type
12    elif 'primary_name' in plot_df.columns: main_cat_col = 'primary_name'
13    else: main_cat_col = 'genres'
14    #
15    if plot_type in ['directors', 'writers', 'actors', 'actresses']: crew_plot = True
16    else: crew_plot = False
17    #
18    if viz_type == 'rating':
19        rating_plot = True
20        sub_cat_col = viz_type
21        test_col_str = plot_df.columns[2]
22        plot_len = len(plot_df)
23        #
24        if 'tot' in test_col_str:
25            cat_freq = []
26            for g in plot_df[main_cat_col].unique():
27                g_freq = plot_df.set_index(main_cat_col).loc[g].num_titles
28                #
29                if type(g_freq) in [np.int64, np.float64]: cat_freq.append(g_freq)
30                else: cat_freq.append(g_freq[0])
31            #
32        else: cat_freq = plot_df['num_titles']
33    #
34    else:
35        rating_plot = False
36        test_col_str = plot_df.columns[-1]
37        plot_len = len(plot_df[main_cat_col].unique())
38        #
39        if not box: cat_freq = plot_df['num_titles']
40    #
41    if 'gross' in test_col_str: lab_add = 'Worldwide Gross'
42    if 'profits' in test_col_str: lab_add = 'Worldwide Profits'
43    #
44    base_x_formatter = x_currency_formatter
45
46    # Setting important variables based on if 'tot', 'avg' in last column or if box=True
47    #####
48    if not box and 'tot' in test_col_str:
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        plot_col = plot_df.columns[-2]
55        perc_col = plot_df.columns[-1]
56        #
57        cat_values = plot_df[plot_col]
58        cat_percs = plot_df[perc_col]
59        #
60        value_step, v_q_used, v_last_neg_q = find_pos_quantile(cat_values)
61        perc_step, p_q_used, p_last_neg_q = find_pos_quantile(cat_percs)
62        #
63        if p_q_used != v_q_used:
64            if p_q_used > v_q_used: value_step = cat_values.quantile(p_q_used)
65            else: perc_step = cat_percs.quantile(v_q_used)
66        #
67        if p_last_neg_q != v_last_neg_q and box:
68            if p_last_neg_q < v_last_neg_q: neg_value_step = cat_values.quantile(p_last_neg_q)
69            else: neg_perc_step = cat_percs.quantile(v_last_neg_q)
70        #
```

```

71     if rating_plot:
72         cat_values = cat_values.unique()
73         cat_percs = cat_percs.unique()
74     #-----
75     if box or 'avg' in test_col_str:
76         prime_lab_p1 = 'Average '
77     #-----
78     if box:
79         box_add = 'Boxplots of the '
80         if 'tot' in box: prime_lab_p1 = 'Total '
81     #-----
82     plot_col = plot_df.columns[-1]
83     cat_values = plot_df[plot_col]
84     #-----
85     if not rating_plot:
86         value_step, v_q_used, v_last_neg_q = find_pos_quantile(cat_values)
87     #-----
88     else:
89         value_step, v_q_used, v_last_neg_q = find_pos_quantile(plot_df[test_col_str])
90     #-----
91     neg_value_step = cat_values.quantile(v_last_neg_q)
92     #-----
93     if not box: prime_lab = prime_lab_p1 + lab_add
94     if box:
95         top_lab = prime_lab_p1 + lab_add
96         prime_lab = box_add + lab_add
97     #-----
98     rank_add = 'Average ' + lab_add
99
100    # Setting important variables based on plot type and if box=True
101    ######
102    if crew_plot:
103        y_lab = plot_type.title()
104        y_labels = plot_df[main_cat_col]
105        tex_y_labels = [r'\textbf{\textit{scshape}{' + lab + '}}' for lab in y_labels]
106    #-----
107        if plot_type == 'actresses': r_lab_insert = y_lab[:-2]
108        else: r_lab_insert = y_lab[:-1]
109    #-----
110    if not crew_plot: y_labels = plot_df[main_cat_col].unique()
111    #-----
112    if rating_plot and 'avg' in test_col_str: y_labels = plot_df[main_cat_col]
113
114    # Setting important variables for Genre Combination plots based on plot type
115    ######
116    if viz_type == 'combos':
117        r_lab_1 = 'Number of Titles with this Genre'
118    #-----
119        if crew_plot:
120            r_lab_2 = 'Combination from each ' + r_lab_insert
121        #-----
122            title_add = y_lab + ' of' if plot_type in ['directors', 'writers'] else y_lab + ' in'
123        #-----
124            c_tit_p1 = r'{\fontsize{30}{0}\selectfont{\textsf{Top ' + str(plot_len) + \
125                r' ' + title_add + r' Movies with the Genre Combination Above}}}'+ '\n'
126            c_tit_p2 = r'{\fontsize{27}{0}\selectfont{\textsf{Ranked by ' + prime_lab +'}}}'
127        #-----
128        if plot_type == 'genres':
129            y_lab = 'Unique Genre Combinations'
130        #-----
131            r_lab_1 = 'Number of Titles from each'
132            r_lab_2 = y_lab[:-1]
133        #-----
134            c_tit_p1 = r'{\fontsize{30}{0}\selectfont{\textsf{Top ' + str(plot_len) + \
135                r' ' + y_lab +'}}}' + tit_bullet
136        #-----
137            c_tit_p2 = r'{\fontsize{30}{0}\selectfont{\textsf{Ranked by ' + prime_lab +'}}}'
138        #-----
139            tex_y_labels = \
140                [r'\textbf{\textit{scshape}{' + ', '.join(lab.split(',')) + '}}' for lab in y_labels]
141        #-----

```

```

142     if box:
143         prime_lab = box_add + lab_add
144         #-
145         c_tit_p2 = r'{\fontsize{30}{0}\selectfont{\textsl{Based on '+ top_lab +'}}}''
146         #-
147         c_tit_p3 = \
148             r'{\fontsize{28.5}{0}\selectfont{\textsl{Ranked by Average '+ lab_add +'}}}'+'\n'+\
149             r'{\fontsize{27}{0}\selectfont{\textsl{'+ prime_lab +' of Movies}}}'+\
150             r'{\fontsize{27}{0}\selectfont{\textsl{ with these Unique Genre Combinations}}}''
151         #-
152         if not box: tit_p3 = c_tit_p1 + c_tit_p2
153         if box: tit_p3 = c_tit_p1 + c_tit_p2 +'\n'+ c_tit_p3
154
155     # Setting important variables for individual Genre plots based on year range and plot type
156 #####
157     if viz_type == 'individs' or viz_type == 'genres':
158         if viz_type == 'individs': r_lab_1 = 'Number of Titles that Included'
159         if viz_type == 'genres': r_lab_1 = 'Number of Titles with'
160         #-
161         if crew_plot:
162             r_lab_2 = 'This Genre from each '+ r_lab_insert
163             #-
164             title_add = y_lab + ' of' if plot_type in ['directors','writers'] else y_lab + ' in'
165             #-
166             if viz_type == 'individs':
167                 i_tit_p1 = r'{\fontsize{30}{0}\selectfont{\textsl{'+ title_add +\
168                     ' Movies that Included the Individual Genre Above}}}''
169             #-
170             if viz_type == 'genres':
171                 i_tit_p1 = r'{\fontsize{30}{0}\selectfont{\textsl{'+ title_add +\
172                     ' Movies with the Genre Above}}}''
173             #-
174             i_tit_p2 = r'{\fontsize{27}{0}\selectfont{\textsl{Ranked by '+ prime_lab +'}}}''
175         #-
176         if plot_type == 'genres':
177             if viz_type == 'individs': y_lab = 'Individual Genres'
178             if viz_type == 'genres': y_lab = 'Genres'
179             #-
180             i_p1_insert = 'each '+ y_lab[:-1]
181             #-
182             if not box:
183                 if viz_type == 'individs': r_lab_2 = 'Each '+ y_lab[:-1]
184                 if viz_type == 'genres': r_lab_2 = 'each '+ y_lab[:-1]
185                 tit_start = r'{\fontsize{30}{0}\selectfont{\textsl{'
186             else:
187                 tit_start = r'{\fontsize{27}{0}\selectfont{\textsl{'
188                 i_tit_p2 = r'{\fontsize{27}{0}\selectfont{\textsl{Ranked by '+ rank_add +'}}}''
189             #-
190             if viz_type == 'individs':
191                 i_tit_p1 = tit_start + prime_lab +' of Movies that Included '+ i_p1_insert +'}}}''
192             if viz_type == 'genres':
193                 i_tit_p1 = tit_start + prime_lab +' of Movies with '+ i_p1_insert +'}}}''
194             #-
195             tex_y_labels = [r'\textbf{\textsf{\textit{\textbf{\textsf{\textit{lab}}}}}}' for lab in y_labels]
196             #-
197             if plot_type != 'genres': tit_p3 = i_tit_p1 +'\n'+ i_tit_p2
198             else: tit_p3 = i_tit_p1
199
200     # Setting important variables for individual Genre plots based on year range and plot type
201 #####
202     if viz_type == 'creative_type':
203         r_lab_1 = 'Number of Titles with '
204         #-
205         if crew_plot:
206             r_lab_2 = 'Creative Type from each '+ r_lab_insert
207             #-
208             title_add = y_lab + ' of' if plot_type in ['directors','writers'] else y_lab + ' in'
209             #-
210             i_tit_p1 = r'{\fontsize{30}{0}\selectfont{\textsl{'+ title_add +\
211                 ' Movies with the Creative Type Above}}}''
212             i_tit_p2 = r'{\fontsize{27}{0}\selectfont{\textsl{Ranked by '+ prime_lab +'}}}''

```

```

213
214     #-
215     if plot_type == 'creative_type':
216         y_lab = 'Creative Types'
217         #-
218         i_p1_insert = 'each ' + y_lab[:-1]
219         #-
220         if not box:
221             r_lab_2 = 'Each ' + y_lab[:-1]
222             tit_start = r'{\fontsize{30}{0}\selectfont{\textsl{'
223             else:
224                 tit_start = r'{\fontsize{27}{0}\selectfont{\textsl{'
225                 i_tit_p2 = r'{\fontsize{27}{0}\selectfont{\textsl{Ranked by '+ rank_add +'}}}'}
226                 #-
227                 i_tit_p1 = tit_start + prime_lab + ' of Movies with '+ i_p1_insert +'}}}'
228                 #-
229                 tex_y_labels = [r'\textbf{\scshape{'+ lab +'}}' for lab in y_labels]
230             #-
231             if plot_type != 'creative_type': tit_p3 = i_tit_p1 +'\n'+ i_tit_p2
232             else: tit_p3 = i_tit_p1
233
234     # Setting important variables for individual Genre plots based on year range and plot type
235 ######
236     if rating_plot:
237         r_lab_1 = 'Number of Titles with'
238         #-
239         spec_lab = 'MPA Ratings'
240         #-
241         if plot_type=='combos':
242             y_lab = 'Unique Genre Combinations'
243             tit_p1_add = 'with'
244             # cat_tit_add = 'Each Genre Combination with '
245             cat_tit_add = ' each Genre Combination with '
246             tex_y_labels = \
247                 [r'\textbf{\scshape{+', ' .join(lab.split(','))+'}}' for lab in y_labels]
248             #-
249         if plot_type=='individs':
250             y_lab = 'Individual Genres'
251             tit_p1_add = 'that'
252             # cat_tit_add = 'Included each Individual Genre with '
253             cat_tit_add = ' included each Individual Genre with '
254             tex_y_labels = [r'\textbf{\scshape{'+ lab +'}}' for lab in y_labels]
255             #-
256         if plot_type=='genres':
257             y_lab = 'Genres'
258             tit_p1_add = 'with'
259             # cat_tit_add = 'Each Genre with '
260             cat_tit_add = ' each Genre with '
261             tex_y_labels = [r'\textbf{\scshape{'+ lab +'}}' for lab in y_labels]
262             #-
263         if plot_type=='creative_type':
264             y_lab = 'Creative Types'
265             tit_p1_add = 'with'
266             # cat_tit_add = 'Each Creative Type with '
267             cat_tit_add = ' each Creative Type with '
268             tex_y_labels = [r'\textbf{\scshape{'+ lab +'}}' for lab in y_labels]
269             #-
270             if 'avg' in test_col_str: plot_height = 18
271             #-
272             i_p2_insert = 'each ' + spec_lab[:-1]
273             #-
274             if not box:
275                 if viz_type!='genres' or (rating_plot and 'tot' in test_col_str and plot_type!='genres'):
276                     r_lab_2 = 'Each ' + y_lab[:-1]
277
278                 if viz_type=='genres' or (rating_plot and 'avg' in test_col_str) or \
279                     (rating_plot and plot_type=='genres'):
280                     r_lab_2 = 'each ' + y_lab[:-1]
281
282             tit_start = r'{\fontsize{30}{0}\selectfont{\textsl{'

```

```

284 #         else: tit_start = r'{\fontsize{27}{0}\selectfont{\textsl{'
285 #-
286 i_tit_p1 = tit_start + prime_lab + ' of Movies ' + tit_p1_add + '}}}'  

287 i_tit_p2 = tit_start + cat_tit_add + i_p2_insert + '}}}'  

288 #-
289 #         tit_p3 = i_tit_p1 + '\n' + i_tit_p2  

290 tit_p3 = i_tit_p1 + i_tit_p2  

291
292 # Setting other variables necesary for plotting
293 #####  

294 num_bars = 1
295 bar_height = 3
296 group_space = 4.5
297 #-
298 y_locs = range(plot_len)
299 #-
300 if (rating_plot and 'tot' in test_col_str) or box:
301     y_locs = range(len(plot_df[main_cat_col].unique()))
302
303 # Locations for bars or boxes
304 #####  

305 y_ticks = [num_bars*group_space*y + bar_height*(1.5) for y in y_locs]
306 #-
307 split_dist = ((y_ticks[1] - y_ticks[0]) / 2)
308 split_locs = [(num_bars*group_space)*y + bar_height*(1.5) + split_dist for y in y_locs[0:-1]]
309
310 # Setting up the figure and the axes
311 #####  

312 fig = plt.figure(figsize=(18, plot_height), dpi=300)
313 gs = fig.add_gridspec(1)
314 #-
315 ax = gs.subplots()
316 #-
317 if perc_plot: t_ax = ax.twiny()
318 if not box: r_ax = ax.twinx()
319 #-
320 renderer = fig.canvas.get_renderer()
321
322 # Plotting the data based on if box=True
323 #####  

324 if not box:
325     bar_colors = get_lighter_color('forestgreen', .45)
326     #-
327     bars_kw = dict(color=bar_colors, ec='k', lw=.54, zorder=3)
328     #-
329     if not rating_plot or (rating_plot and 'tot' in test_col_str):
330         main_bars = ax.barh(y_ticks, cat_values, bar_height, **bars_kw)
331     #-
332 if rating_plot and not box:
333     rats_used_dict = {}
334     #-
335     if 'tot' in test_col_str:
336         for m_cat, y_t, m_bar in zip(plot_df[main_cat_col].unique(), y_ticks, main_bars):
337             sub_cats = plot_df.set_index(main_cat_col).loc[m_cat, 'rating']
338             sub_cat_values = plot_df.set_index(main_cat_col).loc[m_cat, test_col_str]
339             #-
340             bar_total = m_bar.get_width()
341             #-
342             if type(sub_cat_values) != pd.Series:
343                 bar_color = get_lighter_color(ratings_colors_dict[sub_cats], .45)
344                 #-
345                 if sub_cats not in rats_used_dict.keys():
346                     rats_used_dict[sub_cats] = bar_color
347                 #-
348                 bars_kw = dict(color=bar_color, ec='k', lw=.54, zorder=3)
349                 #-
350                 ax.barh(y_t, sub_cat_values, bar_height, **bars_kw)
351                 #-
352                 if sub_cat_values < bar_total:
353                     unk_label = 'Unknown'
354                     unk_color = get_lighter_color('forestgreen', .45)

```

```

355
356         #-
357         if ink_label not in rats_used_dict.keys():
358             rats_used_dict[unk_label] = unk_color
359     #
360     else:
361         old_val = 0
362         for s_cat, val in zip(sub_cats, sub_cat_values):
363             bar_color = get_lighter_color(ratings_colors_dict[s_cat], .45)
364             #
365             if s_cat not in rats_used_dict.keys():
366                 rats_used_dict[s_cat] = bar_color
367             #
368             bars_kw = dict(color=bar_color, ec='k', lw=.54, zorder=3)
369             #
370             ax.bahr(y_t, val, bar_height, left=old_val, **bars_kw)
371             #
372             old_val = old_val + val
373             #
374             if old_val < bar_total:
375                 unk_label = 'Unknown'
376                 unk_color = get_lighter_color('forestgreen', .45)
377             #
378             if unk_label not in rats_used_dict.keys():
379                 rats_used_dict[unk_label] = unk_color
380     #
381     else:
382         main_series = plot_df[main_cat_col]
383         sub_series = plot_df['rating']
384         plot_series = plot_df[test_col_str]
385     #
386     for g, sub_cat, val, y_t in zip(main_series, sub_series, plot_series, y_ticks):
387         bar_color = get_lighter_color(ratings_colors_dict[sub_cat], .45)
388         #
389         if sub_cat not in rats_used_dict.keys(): rats_used_dict[sub_cat] = bar_color
390         #
391         bars_kw = dict(color=bar_color, ec='k', lw=.54, zorder=3)
392         #
393         ax.bahr(y_t, val, bar_height, **bars_kw)
394     #
395     rat_leg = []
396     #
397     for rat, rat_c in rats_used_dict.items():
398         sub_cat_patch = \
399         mpatches.Patch(label=r'\textbf{\scshape{'+ rat +'}}', color=rat_c)
400     #
401     rat_leg.append(sub_cat_patch)
402     #
403     if box:
404         c_patch = get_lighter_color('forestgreen', .45)
405         c_line = get_lighter_color('forestgreen', .36)
406     #
407     for cat, y_t in zip(plot_df[main_cat_col].unique(), y_ticks):
408     #
409         box_kw = dict(fc=c_patch, lw=.9, ec='k')
410         means_kw = dict(color='k', ls=(0, (1, 1)), lw=6.3, zorder=6)
411         meds_kw = dict(color='k', lw=4.5, zorder=99)
412         caps_kw = dict(c=c_line, lw=4.2, solid_capstyle='round', zorder=6)
413         whis_kw = dict(c=c_line, lw=4.2, solid_capstyle='round', zorder=6)
414         fliers_kw = dict(mfc=c_line, ms=15, mec='k', mew=.3, marker='X')
415     #
416         b_kw = dict(widths=bar_height, boxprops=box_kw, medianprops=meds_kw, meanline=True,
417                     showmeans=True, meanprops=means_kw, flierprops=fliers_kw,
418                     whiskerprops=whis_kw, capprops=caps_kw, patch_artist=True, vert=False)
419     #
420     cat_df = plot_df.set_index(main_cat_col).loc[cat]
421     #
422     box_dict = ax.boxplot(cat_df[plot_col], positions=[y_t], **b_kw)
423     #
424     mean_lines = \
425     mlines.Line2D([], [], label=r'\textbf{\scshape{Mean Lines}}', **means_kw)
426

```

```

426         median_lines = \
427             mlines.Line2D([], [], label=r'\textbf{\scshape{Median Lines}}', **meds_kw)
428
429     # AX - X Ticks, Tick Labels, and Bounds
430     #####
431     if not rating_plot or 'tot' in test_col_str:
432         x_max = cat_values.max()
433     else: x_max = plot_df[test_col_str].max()
434     #
435     final_max = find_max(x_max, value_step)
436     #
437     x_tick_max = final_max
438     #
439     if not box:
440         x_step = final_max/6
441         x_ticks = np.arange(0, x_tick_max +1, x_step)
442     #
443     if box:
444         x_min = cat_values.min()
445         #
446         final_min = find_min(x_min, neg_value_step)
447         #
448         if final_min < 0:
449             zero_kw = dict(color='k', lw=3.9, ls='-.')
450             #
451             zero_line = \
452                 ax.axvline(0, label=r'\textbf{\scshape{Zero Profits Line}}', **zero_kw)
453         #
454         x_tick_min = final_min
455         x_step = (final_max - final_min) / 6
456         x_ticks = np.arange(final_min, x_tick_max+1, x_step)
457     #
458     ax.set_xticks(x_ticks)
459     #
460     ax.xaxis.set_major_formatter(base_x_formatter)
461     #
462     ax.set_xbound(x_ticks[0], x_ticks[-1])
463
464     # T_AX - X Ticks, Tick Labels, and Bounds
465     #####
466     if perc_plot:
467         x_max = cat_percs.max()
468         #
469         final_max = find_max(x_max, perc_step)
470         #
471         x_tick_max = final_max
472         x_step = final_max/6
473         x_ticks = np.arange(0, x_tick_max+1, x_step)
474         #
475         t_ax.set_xticks(x_ticks)
476         t_ax.xaxis.set_major_formatter(x_percentage_formatter)
477         #
478         t_ax.set_xbound(0, x_tick_max)
479
480     # AX & R_AX - Y Tick, Tick Labels and Bounds
481     #####
482     ax.set_yticks(y_ticks)
483     ax.set_yticklabels(tex_y_labels, va='center')
484     ax.set_ybound((y_ticks[0] - split_dist, y_ticks[-1] + split_dist))
485     ax.invert_yaxis()
486     #
487     if not box:
488         freq_labels = y_labels_to_norm_num(cat_freq)
489         #
490         r_ax.set_yticks(y_ticks)
491         r_ax.set_yticklabels(freq_labels, va='center')
492         r_ax.set_ybound((y_ticks[0] - split_dist, y_ticks[-1] + split_dist))
493         r_ax.invert_yaxis()
494
495     # Tick Params Functions
496     #####

```

```

497     if not box: h_tick_params(ax)
498     if box: h_tick_params(ax, True)
499     if perc_plot: top_x_tick_params(t_ax)
500     if not box: right_y_tick_params(r_ax)
501
502     # Axis Labels
503 #####
504     x_lab_kw= dict(size=28.5, labelpad=18)
505     y_lab_kw= dict(size=30, labelpad=27, linespacing=1.5)
506     #
507     if not box: ax.set_xlabel(r'\textbf{\textsf{'+ prime_lab +'}}', **x_lab_kw)
508     if box: ax.set_xlabel(r'\textbf{\textsf{'+ lab_add +'}}', **x_lab_kw)
509     #
510     if perc_plot:
511         t_ax.set_xlabel(r'\textbf{\textsf{'+ t_lab +'}}', **x_lab_kw)
512     #
513     ax.set_ylabel(r'\textbf{\textsf{'+ y_lab +'}}', **y_lab_kw)
514     if not box:
515         r_ax.set_ylabel(r'\textbf{\textsf{'+ r_lab_1 +'}}'+'\n'+
516                         r'\textbf{\textsf{'+ r_lab_2 +'}}', va='bottom', rotation=270,
517                         **y_lab_kw)
518     #
519     if (viz_type=='genres' and plot_type=='genres') or \
520         (rating_plot and 'avg' in test_col_str) or \
521         (rating_plot and 'tot' in test_col_str and plot_type=='genres'):
522         r_ax.set_ylabel(r'\textbf{\textsf{'+ r_lab_1 +' + '+ r_lab_2 +'}}', va='bottom',
523                         rotation=270, **y_lab_kw)
524
525     # Title
526 #####
527     if perc_plot: tit_kw = dict(pad=36, linespacing=1.8)
528     else: tit_kw = dict(pad=18, linespacing=1.8)
529     #
530     m_tit_p1 = r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{ Categorical Data}}}}'
531     #
532     finan_data_source = \
533     r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{The Numbers Financial Data}}}}'
534     #
535     if viz_type in ['combos', 'individs'] or plot_type in ['combos', 'individs']:
536         cat_source = r'{\fontsize{31.5}{0}\selectfont{\textbf{IMDb}}}'
537         year_tit = r'2010 \boldsymbol{\rightarrow} 2019'
538     #
539     if viz_type in ['genres', 'creative_type'] or \
540         (rating_plot and plot_type in ['genres', 'creative_type']):
541         cat_source = r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{OpusData}}}}'
542         year_tit = r'2010 \boldsymbol{\rightarrow} 2018'
543     #
544     if rating_plot:
545         if 'tot' in test_col_str: year_tit = r'2010 \boldsymbol{\rightarrow} 2019'
546         if 'avg' in test_col_str: year_tit = r'2010 \boldsymbol{\rightarrow} 2018'
547         #
548         rat_cat_source = r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{OpusData}}}}'
549         m_tit_rat = r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{ Ratings Data}}}}'
550         #
551         tit_p1 = cat_source + m_tit_p1 + '\n'+ rat_cat_source + m_tit_rat + tit_bullet + \
552             finan_data_source
553     #
554     if crew_plot:
555         crew_source = r'{\fontsize{31.5}{0}\selectfont{\textbf{IMDb}}}'
556         m_tit_crew = r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{ Crew Data}}}}'
557         #
558         tit_p1 = cat_source + m_tit_p1 + '\n'+ crew_source + m_tit_crew + tit_bullet + \
559             finan_data_source
560     #
561     if not rating_plot and not crew_plot:
562         tit_p1 = cat_source + m_tit_p1 + tit_bullet + finan_data_source
563     #
564     tit_p2 = r'{\fontsize{33}{0}\selectfont{$\mathbf{'+ year_tit +'}$}}'
565     #
566     if not crew_plot: ax.set_title(tit_p1 + '\n' + tit_p2 + '\n' + tit_p3, **tit_kw)
567     #

```

```

568 if crew_plot:
569     if viz_type=='combos':
570         cat_tit = r'{\fontsize{30}{0}\selectfont{\textbf{\textsf{'+ \
571             '}}.\join(plot_df.index[0].split(',')) +r'}}}'
572     #-
573     else:
574         cat_tit = r'{\fontsize{30}{0}\selectfont{\textbf{\textsf{'+ plot_df.index[0] +'}}}}'
575     #-
576     ax.set_title(tit_p1 + '\n' + tit_p2 + '\n' + cat_tit + '\n' + tit_p3, **tit_kw)
577
578     # Legend
579 ######
580     # if viz_type=='rating': leg_loc = 'best'
581     # else: leg_loc = 'Lower right'
582     leg_loc = 'best'
583     #-
584     leg_kw = dict(loc=leg_loc, fontsize=21, handlelength=3)
585     #-
586     frame_kw = dict(lw=.9, ec='k', alpha=1)
587     #-
588     if box:
589         if final_min >= 0: leg = ax.legend(handles=[mean_lines, median_lines], **leg_kw)
590         else: leg = ax.legend(handles=[zero_line, mean_lines, median_lines], **leg_kw)
591     #-
592         plt.setp(leg.get_frame(), **frame_kw)
593         plt.setp(leg.get_texts(), ma='center')
594     #-
595     if viz_type=='rating':
596         leg = ax.legend(handles=rat_leg, **leg_kw)
597     #-
598         plt.setp(leg.get_frame(), **frame_kw)
599         plt.setp(leg.get_texts(), ma='center')
600
601     # Adding lines between each genre's name (lines between y tick labels)
602 ######
603     c_tick_kw = dict(clip_on=False, color='k', lw=1.5, alpha=.81, zorder=9)
604     #-
605     top_spl = y_ticks[-1] + split_dist
606     bot_spl = y_ticks[0] - split_dist
607     #-
608     max_label_neg_x = 0
609     max_label_pos_x = 0
610     #-
611     ax_inv = ax.transAxes.inverted()
612     #-
613     if not box:
614         r_ax_inv = r_ax.transAxes.inverted()
615     #-
616         for label, r_label in zip(ax.get_ymajorticklabels(), r_ax.get_ymajorticklabels()):
617             lab_xymin = label.get_tightbbox(renderer).min
618             r_lab_xymin = r_label.get_tightbbox(renderer).min
619         #-
620             lab_f_bbox_xymin = ax_inv.transform(lab_xymin)
621             r_lab_f_bbox_xymin = r_ax_inv.transform(r_lab_xymin)
622         #-
623             if lab_f_bbox_xymin[0] <= max_label_neg_x: max_label_neg_x = lab_f_bbox_xymin[0]
624         #-
625             if r_lab_f_bbox_xymin[0] >= max_label_pos_x: max_label_pos_x = r_lab_f_bbox_xymin[0]
626         #-
627             pos_kw = dict(x=((max_label_pos_x - 1) / 2) + 1.006, ha='center')
628             [r_label.update(pos_kw) for r_label in r_ax.get_ymajorticklabels()]
629     #-
630     else:
631         for label in ax.get_ymajorticklabels():
632             lab_xymin = label.get_tightbbox(renderer).min
633         #-
634             lab_f_bbox_xymin = ax_inv.transform(lab_xymin)
635         #-
636             if lab_f_bbox_xymin[0] <= max_label_neg_x: max_label_neg_x = lab_f_bbox_xymin[0]
637         #-
638             neg_kw = dict(x=(max_label_neg_x/2) - .0045, ha='center')

```

```

639 [label.update(neg_kw) for label in ax.get_ymajorticklabels()]
640 #
641 if rating_plot and 'avg' in test_col_str:
642     old_label_text = ax.get_ymajorticklabels()[0].get_text()
643     label_size = ax.get_ymajorticklabels()[0].get_fontsize()
644     #
645     same_label_group = [ax.get_ymajorticklabels()[0]]
646     #
647     for s_loc, label in zip(split_locs, ax.get_ymajorticklabels()[1:]):
648         new_label_text = label.get_text()
649         #
650         if new_label_text == old_label_text and label == ax.get_ymajorticklabels()[-1]:
651             last_labels_same = True
652             same_label_group.append(label)
653         else: last_labels_same = False
654         #
655         if new_label_text != old_label_text or last_labels_same:
656             if len(same_label_group) > 1:
657                 label_x_pos = ax_inv.transform(same_label_group[0].get_tightbbox(renderer))
658                 #
659                 lab_x_min = label_x_pos[0][0]
660                 lab_x_max = label_x_pos[1][0]
661                 #
662                 first_label_y_min = \
663                 ax_inv.transform(same_label_group[0].get_tightbbox(renderer).min)[1]
664                 #
665                 last_label_y_max = \
666                 ax_inv.transform(same_label_group[-1].get_tightbbox(renderer).max)[1]
667                 #
668                 mid_x = (lab_x_min + lab_x_max) / 2
669                 mid_y = ((last_label_y_max - first_label_y_min) / 2) + first_label_y_min
670                 #
671                 [label.set_visible(False) for label in same_label_group]
672                 #
673                 ax.annotate(old_label_text, (mid_x, mid_y), xycoords='axes fraction',
674                             size=label_size, va='center', ha='center')
675             #
676             old_label_text = new_label_text
677             same_label_group = [label]
678             #
679             if not last_labels_same:
680                 ax.axhline(s_loc, 0, max_label_neg_x -.015, **c_tick_kw)
681             #
682             else: same_label_group.append(label)
683             #
684             ax.axhline(s_loc, 1, max_label_pos_x + .015, **c_tick_kw)
685         else:
686             for s_loc in split_locs:
687                 ax.axhline(s_loc, 0, max_label_neg_x -.015, **c_tick_kw)
688                 #
689                 if not box: ax.axhline(s_loc, 1, max_label_pos_x + .015, **c_tick_kw)
690             #
691             if not any([label.get_visible() for label in ax.get_ymajorticklabels()]):
692                 y_label = ax.get_ylabel()
693                 #
694                 ax.set_ylabel('', visible=False)
695                 #
696                 ax.annotate(y_label, (max_label_neg_x -.036, .5), xycoords='axes fraction', size=30,
697                             va='bottom', ha='center', rotation=90, rotation_mode='anchor')
698             #
699             bot_ax_line = ax.axhline(bot_spl, 0, max_label_neg_x -.015, **c_tick_kw)
700             if not box: ax.axhline(bot_spl, 1, max_label_pos_x + .015, **c_tick_kw)
701             #
702             top_ax_line = ax.axhline(top_spl, 0, max_label_neg_x -.015, **c_tick_kw)
703             if not box: top_ax_line = ax.axhline(top_spl, 1, max_label_pos_x + .015, **c_tick_kw)
704             #
705             # Adding other lines to enhance appearance
706 #####
707             div_kw = dict(color='k', lw=1.95#, alpha=.63)
708             #
709             fig_inv = fig.transFigure.inverted()

```

```

710  #-
711  if perc_plot:
712      ax_xymin = t_ax.get_tightbbox(renderer).min
713      ax_xymax = t_ax.get_tightbbox(renderer).max
714      top_lab_xymax = t_ax.xaxis.get_label().get_tightbbox(renderer).max
715      #
716      div_left_x = fig_inv.transform(ax_xymin)[0]
717      div_right_x = fig_inv.transform(ax_xymax)[0]
718      div_y = fig_inv.transform(top_lab_xymax)[1]
719      div_y = div_y + .027 if plot_height==9 else div_y + .009
720      #
721      div_line = mlines.Line2D([div_left_x, div_right_x], [div_y, div_y], **c_tick_kw)
722  #
723  left_vert_x = fig_inv.transform(bot_ax_line.get_tightbbox(renderer).min)[0]
724  if not box: right_vert_x = fig_inv.transform(top_ax_line.get_tightbbox(renderer).max)[0]
725  #
726  bot_vert_y = fig_inv.transform(top_ax_line.get_tightbbox(renderer).max)[1]
727  top_vert_y = fig_inv.transform(bot_ax_line.get_tightbbox(renderer).min)[1]
728  #
729  left_vert = mlines.Line2D([[left_vert_x] * 2], [bot_vert_y, top_vert_y], **c_tick_kw)
730  if not box:
731      right_vert = mlines.Line2D([[right_vert_x] * 2], [bot_vert_y, top_vert_y], **c_tick_kw)
732  #
733  if 'tot' in plot_df.columns[-1] or (not box and 'tot' in test_col_str):
734      fig.add_artist(div_line)
735  #
736  if not box: [fig.add_artist(art) for art in [left_vert, right_vert]]
737  else: fig.add_artist(left_vert)
738
739  # Returning the figure with a label to the save accordingly
740 #####
741 return fig, prime_lab

```

## § Standards for Qualifying as Important

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

### IMDb Category Types

#### Standards for Qualifying as an Important Unique Genre Combination

- For a unique genre combination to be considered as an ***Important Unique Genre Combination*** for a metric, there had to have been **at least three movies made** with that specific genre combination within the period covered.
- If a genre combination's value was in the **Top 10** of all of the genre combinations for a metric, it was considered a financially important genre combination for that metric.

#### Standards for Qualifying as an Important Individual Genre

- For an individual genre to be even be considered as an ***Important Individual Genre***, there had to have been **at least nine movies made** that included that individual genre in their 'genres' description within the period covered.
- If an individual genre's value was in the **Top 25%** of all of the individual genres for a metric, it was considered a financially important individual genre for that metric.

### OpusData Category Types

#### Standards for Qualifying as an Important Genre

- For an genre to be even be considered as an ***Important Genre***, there had to have been **at least nine movies made** with that genre within the period covered.
- If a genre's value was in the **Top 25%** of all of the genres for a metric, it was considered a financially important genre for that metric.

## Standards for Qualifying as an Important Creative Type

- For an individual genre to be even be considered as an ***Important Creative Type***, there had to have been **at least nine movies made** with that creative type within the period covered.
- If a creative type's value was in the **Top  $\frac{1}{3}$ rd** of all of the creative types for a metric, it was considered a financially important creative type for that metric.

## § Analysis Preparation

---

I first created a list of strings that I could iterate through to simply create dataframes and visualizations based on each of the financial metrics I previously mentioned.

In [28]:

```

1 # Preparing important strings to use for the entire financial section
2 #####
3 sum_aka_1 = 'tot_worldwide_gross'
4 #-----
5 sum_aka_2 = 'tot_worldwide_profits'
6 #-----
7 avg_aka_1 = 'avg_worldwide_gross'
8 #-----
9 avg_aka_2 = 'avg_worldwide_profits'
10 #-----
11 all_akas = [sum_aka_1, sum_aka_2, avg_aka_1, avg_aka_2]

```

I then used SQL queries to create a base financial dataframe to use in my analysis, as well as financial dataframes for directors, writers, actors, and actresses. I could then use `pandas` to manipulate, analyze and prepare the dataframes I would create based on each category type and financial metric.

## 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 [29]:

```
1 # Using a SQL query to get the IMDB movie data with the The Numbers financial information
2 #####
3 db_query = """
4 SELECT
5     tconst, start_year, primary_title, genre, genres, rating, creative_type,
6     production_budget, worldwide_gross, worldwide_profits
7 FROM tn_full
8 JOIN imdb_title_basics
9     ON (primary_title = movie AND (release_date LIKE '%' || start_year || '%'))
10 WHERE genres NOT Null;"""
11 #-
12 fin_df = pysqldf(db_query)
13 #-
14 display(fin_df.head(3))
15 display(fin_df.info())
16 #-
17 fin_df = fin_df.drop_duplicates(['tconst', 'start_year', 'primary_title'], keep=False)
18 #-
19 display(fin_df.info())
```

|   | tconst    | start_year | primary_title                   | genre     | genres                  | rating | creative_type        | production_budget | worldwide_gross |
|---|-----------|------------|---------------------------------|-----------|-------------------------|--------|----------------------|-------------------|-----------------|
| 0 | tt0249516 | 2012       | Foodfight!                      | Adventure | Action,Animation,Comedy | PG     | Kids Fiction         | 45000000.0        | 73              |
| 1 | tt0359950 | 2013       | The Secret Life of Walter Mitty | Adventure | Adventure,Comedy,Drama  | PG     | Contemporary Fiction | 91000000.0        | 187861          |
| 2 | tt0365907 | 2014       | A Walk Among the Tombstones     | Action    | Action,Crime,Drama      | R      | Historical Fiction   | 28000000.0        | 62108           |

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1076 entries, 0 to 1075  
Data columns (total 10 columns):  
 # Column Non-Null Count Dtype  
---  
 0 tconst 1076 non-null object  
 1 start\_year 1076 non-null int64  
 2 primary\_title 1076 non-null object  
 3 genre 1041 non-null object  
 4 genres 1076 non-null object  
 5 rating 1041 non-null object  
 6 creative\_type 1038 non-null object  
 7 production\_budget 1076 non-null float64  
 8 worldwide\_gross 1076 non-null float64  
 9 worldwide\_profits 1076 non-null float64  
dtypes: float64(3), int64(1), object(6)  
memory usage: 84.2+ KB

None

<class 'pandas.core.frame.DataFrame'>  
Int64Index: 1076 entries, 0 to 1075  
Data columns (total 10 columns):  
 # Column Non-Null Count Dtype  
---  
 0 tconst 1076 non-null object  
 1 start\_year 1076 non-null int64  
 2 primary\_title 1076 non-null object  
 3 genre 1041 non-null object  
 4 genres 1076 non-null object  
 5 rating 1041 non-null object  
 6 creative\_type 1038 non-null object  
 7 production\_budget 1076 non-null float64  
 8 worldwide\_gross 1076 non-null float64  
 9 worldwide\_profits 1076 non-null float64  
dtypes: float64(3), int64(1), object(6)  
memory usage: 92.5+ KB

None

```
In [30]: 1 # Checking for duplicated data on a deeper level, trying to save as much data as possible
2 #####
3 dup_titles = \
4 fin_df.loc[fin_df.duplicated(['start_year', 'primary_title', 'worldwide_gross'], keep=False)]\n5 .primary_title.unique()
6 #-----
7 drop_idx = []
8 #-----
9 for tit in dup_titles:
10     tit_df = fin_df.loc[fin_df.primary_title == tit]
11     #
12     if len(tit_df.genres.unique()) > 1:
13         drop_idx.extend(tit_df.index)
14 #-----
15 fin_df = fin_df.drop(drop_idx)
16 #-----
17 display(fin_df.head(3))
18 display(fin_df.info())
19 print('Movies dropped: ', len(drop_idx))
```

|   | tconst    | start_year | primary_title                   | genre     | genres                  | rating | creative_type        | production_budget | worldwide_gross |
|---|-----------|------------|---------------------------------|-----------|-------------------------|--------|----------------------|-------------------|-----------------|
| 0 | tt0249516 | 2012       | Foodfight!                      | Adventure | Action,Animation,Comedy | PG     | Kids Fiction         | 45000000.0        | 73              |
| 1 | tt0359950 | 2013       | The Secret Life of Walter Mitty | Adventure | Adventure,Comedy,Drama  | PG     | Contemporary Fiction | 91000000.0        | 187861          |
| 2 | tt0365907 | 2014       | A Walk Among the Tombstones     | Action    | Action,Crime,Drama      | R      | Historical Fiction   | 28000000.0        | 62108           |

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1016 entries, 0 to 1074
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   tconst            1016 non-null   object 
 1   start_year        1016 non-null   int64  
 2   primary_title     1016 non-null   object 
 3   genre             981 non-null   object 
 4   genres            1016 non-null   object 
 5   rating            981 non-null   object 
 6   creative_type     978 non-null   object 
 7   production_budget 1016 non-null   float64
 8   worldwide_gross   1016 non-null   float64
 9   worldwide_profits 1016 non-null   float64
dtypes: float64(3), int64(1), object(6)
memory usage: 87.3+ KB
```

None

Movies dropped: 60

## Crew Financial DataFrames

I inserted the different job titles as I iterate through them into the SQL Query shown below 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 [31]:

```
1 # Using an SQL query to create the necessary dataframe
2 #####
3 crew_strings = ['director', 'writer', 'actor', 'actress']
4 #
5 for crew_str in crew_strings:
6     print('\033[1m\033[4m'+ crew_str.title() + ' DataFrame:\033[0m')
7     print('-----')
8 #
9 db_query = """
10    SELECT
11        tconst, nconst, primary_name, start_year, primary_title, genre, genres, rating,
12        creative_type, production_budget, worldwide_gross, worldwide_profits
13    FROM imdb_title_basics
14    JOIN tn_full
15        ON (primary_title = movie AND (release_date LIKE '%' || start_year || '%'))
16    JOIN imdb_title_principals
17        USING(tconst)
18    JOIN imdb_name_basic AS nb
19        USING(nconst)
20    WHERE genres NOT Null AND category = '"""+ crew_str +"""
21    ORDER BY start_year, primary_title;"""
22 #
23 crew_df = pysqldf(db_query)
24 #
25 crew_df = crew_df.drop_duplicates(['tconst', 'nconst', 'primary_title'])
26 #
27 print('\033[1m\033[4mBEFORE DUP CHECK:\033[0m')
28 display(crew_df.info())
29
30 # Checking for duplicated data on a deeper level, trying to save as much data as possible
31 #####
32 dup_titles = \
33 crew_df.loc[crew_df.duplicated(['start_year', 'primary_title'], keep=False)]\
34     .primary_title.unique()
35 #
36 drop_idx = []
37 #
38 for tit in dup_titles:
39     tit_df = crew_df.loc[crew_df.primary_title == tit]
40     #
41     if len(tit_df.genres.unique()) > 1:
42         drop_idx.extend(tit_df.index)
43 #
44 crew_df = crew_df.drop(drop_idx)
45 #
46 print('\033[1m\033[4mAFTER DUP CHECK:\033[0m')
47 display(crew_df.info())
48 print('\033[1mMovies dropped:\033[0m', len(drop_idx))
49 print('-----')
50
51 # Checking for duplicated data on a deeper level, trying to save as much data as possible
52 #####
53 if crew_str=='director': dir_df = crew_df
54 if crew_str=='writer': writ_df = crew_df
55 if crew_str=='actor': actor_df = crew_df
56 if crew_str=='actress': actress_df = crew_df
57
58 # List of all crew dataframes
59 #####
60 crewdfs = [dir_df, writ_df, actor_df, actress_df]
```

**Director DataFrame:**

**BEFORE DUP CHECK:**

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1140 entries, 0 to 1139
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  -- 
 0   tconst          1140 non-null   object 
 1   nconst          1140 non-null   object 
 2   primary_name    1140 non-null   object 
 3   start_year      1140 non-null   int64  
 4   primary_title   1140 non-null   object 
 5   genre           1140 non-null   object 
 6   genres          1140 non-null   object 
 7   rating          1140 non-null   float64
 8   creative_type   1140 non-null   object 
 9   production_budget 1140 non-null   float64
 10  worldwide_gross 1140 non-null   float64
 11  worldwide_profits 1140 non-null   float64
```

```
0 tconst          1140 non-null  object
1 nconst          1140 non-null  object
2 primary_name    1140 non-null  object
3 start_year      1140 non-null  int64
4 primary_title   1140 non-null  object
5 genre           1101 non-null  object
6 genres          1140 non-null  object
7 rating          1101 non-null  object
8 creative_type   1098 non-null  object
9 production_budget 1140 non-null  float64
10 worldwide_gross 1140 non-null  float64
```

## § IMDb Category Types

---

### Financially Important Unique Genre Combinations

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

---

In [32]:

```
1 # Preparing empty lists for important genre combinations based on each metric to be added to
2 ######
3 finan_tot_gross_imp_g_combos = []
4 finan_tot_profits_imp_g_combos = []
5 finan_avg_gross_imp_g_combos = []
6 finan_avg_profits_imp_g_combos = []
```

In [33]:

```
1 # Creating, plotting, and saving the appropriate dataframes
2 #####
3 base_dir = 'visuals/Important Unique Genre Combinations/'
4 #
5 for aka_v in all_akas:
6     plot_col = aka_v[4:]
7     #
8     g_count_df = fin_df.groupby('genres').count()[['primary_title']]
9     #
10    if 'tot' in aka_v:
11        g_agg_df = fin_df.groupby('genres').sum()[[plot_col]]
12    if 'avg' in aka_v:
13        g_agg_df = fin_df.groupby('genres').mean()[[plot_col]]
14    #
15    genre_df = pd.concat([g_count_df, g_agg_df], axis=1)
16    #
17    genre_df = genre_df.rename(columns={'primary_title':'num_titles', plot_col: aka_v})
18    #
19    if 'tot' in aka_v:
20        genre_df['perc_of_'+ aka_v] = round((genre_df[aka_v] / fin_df[plot_col].sum()), 4)
21    #
22    genre_df = genre_df.loc[genre_df['num_titles'] >= 3]
23    #
24    all_years_top10 = genre_df.sort_values(aka_v, ascending=False).reset_index().head(10)
25    #
26    top_combo_mapper = \
27    fin_df.genres.map(lambda x: x if x in list(all_years_top10.genres) else None)
28    #
29    imp_list = list(all_years_top10.genres)
30    #
31    if 'tot' in aka_v:
32        rat_agg_df = \
33        fin_df.groupby([top_combo_mapper, 'rating']).sum()[[plot_col]].reset_index('rating')
34    #
35        rat_df = \
36        rat_agg_df.merge(all_years_top10.set_index('genres'), left_index=True, right_index=True)\ \
37            .reset_index().rename({aka_v:'top_cat', plot_col: aka_v}, axis=1)\ \
38            .sort_values('top_cat', ascending=False)
39    #
40    if 'avg' in aka_v:
41        rat_count_df = fin_df.groupby([top_combo_mapper, 'rating']).count()[['primary_title']]
42    #
43        rat_agg_df = \
44        fin_df.groupby([top_combo_mapper, 'rating']).mean()[[plot_col]]
45    #
46        rat_df = \
47        rat_agg_df.merge(rat_count_df, left_index=True, right_index=True).reset_index('rating')\ \
48            .merge(all_years_top10.set_index('genres')[[aka_v]], left_index=True,\ \
49                    right_index=True).reset_index()\ \
50            .rename({'primary_title':'num_titles', aka_v:'top_cat', plot_col: aka_v},\ \
51                    axis=1).sort_values(['top_cat', aka_v], ascending=False)
52    #
53    box_df = pd.DataFrame()
54    fin_import_order_dict = {}
55    #
56    for g_combo in all_years_top10.genres:
57        imp_g_combo_map = fin_df['genres'].map(lambda x: True if g_combo==x else False)
58    #
59        imp_genres_df = fin_df.loc[imp_g_combo_map].copy()[['genres', plot_col]]
60    #
61        fin_import_order_dict[g_combo] = imp_genres_df[plot_col].mean()
62    #
63        box_df = box_df.append(imp_genres_df)
64    #
65    fin_import_order = \
66    sorted(fin_import_order_dict.keys(), key=lambda k: fin_import_order_dict[k], reverse=True)
67    #
68    box_df = box_df.sort_values('genres', key=lambda g_col: g_col.map(lambda x: fin_import_order.index(x)))
69    #
70
```

```

71 fig, fig_tit = cat_plot(all_years_top10, 'combos', 'genres')
72 #
73 if not os.path.isdir(base_dir + fig_tit + '/'):
74     os.mkdir(base_dir + fig_tit + '/')
75 #
76 f_name = base_dir + fig_tit + '/1 - Top 10 Genre Combos.jpeg'
77 #
78 fig.savefig(f_name, bbox_inches='tight')
79 plt.close(fig)
80 #
81 new_dir = base_dir + fig_tit + '/'
82 #
83 fig, fig_tit = cat_plot(box_df, 'combos', 'genres', aka_v)
84 #
85 f_name = new_dir + '2 - ' + fig_tit + '.jpeg'
86 #
87 fig.savefig(f_name, bbox_inches='tight')
88 plt.close(fig)
89 #
90 fig, fig_tit = cat_plot(rat_df, 'rating', 'combos')
91 #
92 f_name = new_dir + '3 - Top 10 Genre Combos - MPA Ratings.jpeg'
93 #
94 fig.savefig(f_name, bbox_inches='tight')
95 plt.close(fig)
96 #
97 if 'tot' in aka_v:
98     if 'gross' in aka_v: finan_tot_gross_imp_g_combos = list(all_years_top10.genres)
99     if 'profits' in aka_v: finan_tot_profits_imp_g_combos = list(all_years_top10.genres)
100 if 'avg' in aka_v:
101     if 'gross' in aka_v: finan_avg_gross_imp_g_combos = list(all_years_top10.genres)
102     if 'profits' in aka_v: finan_avg_profits_imp_g_combos = list(all_years_top10.genres)
103 #
104 # display(fin_combo_all_cols_df)

```

## Financially Important Unique Genre Combinations - Crew

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

---

In [34]:

```
1 # Creating, plotting, and saving the appropriate dataframes
2 #####
3 base_dir = 'visuals/Important Unique Genre Combinations - Crew/'
4 #
5 for crew_str, crew_df in zip(crew_strings, crew_dfs):
6     if crew_str != 'actress': plot_type = crew_str + 's'
7     else: plot_type = crew_str + 'es'
8 #
9     crew_dir = base_dir + plot_type.title() + '/'
10 #
11 for aka_v in all_akas:
12     plot_col = aka_v[4:]
13 #
14     if 'tot' in aka_v:
15         if 'gross' in aka_v: imp_g_combos = finan_tot_gross_imp_g_combos
16         if 'profits' in aka_v: imp_g_combos = finan_tot_profits_imp_g_combos
17     if 'avg' in aka_v:
18         if 'gross' in aka_v: imp_g_combos = finan_avg_gross_imp_g_combos
19         if 'profits' in aka_v: imp_g_combos = finan_avg_profits_imp_g_combos
20 #
21     for g_combo in imp_g_combos:
22         g_combo_df = crew_df.set_index('genres').loc[g_combo].copy()
23 #
24     if type(g_combo_df) == pd.Series:
25         print('\033[1mUnique Genre Combination: \033[0m', g_combo)
26         print('\033[1mFinancial Metric: \033[0m', aka_v)
27         print('This unique genre combination in this metric was only '+\
28             'associated with one '+ crew_str+'.')
29         print("The crew member's name was: ", g_combo_df['primary_name'])
30         print('-----')
31         continue
32 #
33     g_combo_df.reset_index(inplace=True)
34 #
35     top_crew_num_titles = \
36     g_combo_df.groupby('primary_name').count()[['primary_title', 'genres']]
37 #
38     if 'tot' in aka_v:
39         top_crew_aka_df = g_combo_df.groupby('primary_name').sum()[[plot_col]]
40     if 'avg' in aka_v:
41         top_crew_aka_df = g_combo_df.groupby('primary_name').mean()[[plot_col]]
42 #
43     g_combo_top_crew_df = pd.concat([top_crew_num_titles, top_crew_aka_df], axis=1)\.
44         .sort_values(plot_col, ascending=False).head(10)
45 #
46     g_combo_top_crew_df.rename(columns={'primary_title': 'num_titles', plot_col: aka_v},\
47         inplace=True)
48 #
49     if 'tot' in aka_v:
50         g_combo_top_crew_df['perc_of_' + aka_v] = \
51             round((g_combo_top_crew_df[aka_v] / g_combo_df[plot_col].sum()), 4)
52 #
53     g_combo_top_crew_df = g_combo_top_crew_df.reset_index()
54     g_combo_top_crew_df['genres'] = g_combo_top_crew_df['genres'].map(lambda x: g_combo)
55     g_combo_top_crew_df = g_combo_top_crew_df.set_index('genres')
56 #
57     fig, fig_tit = \
58     cat_plot(g_combo_top_crew_df, 'combos', plot_type)
59 #
60     if not os.path.isdir(crew_dir + fig_tit + '/'):
61         os.mkdir(crew_dir + fig_tit + '/')
62 #
63     f_name = crew_dir + fig_tit + '/' + g_combo + '.jpeg'
64 #
65     fig.savefig(f_name, bbox_inches='tight')
66     plt.close(fig)
```

Unique Genre Combination: Adventure,Fantasy  
Financial Metric: avg\_worldwide\_gross

This unique genre combination in this metric was only associated with one actress.  
The crew member's name was: Cate Blanchett

-----  
**Unique Genre Combination:** Adventure,Fantasy

**Financial Metric:** avg\_worldwide\_profits

This unique genre combination in this metric was only associated with one actress.

The crew member's name was: Cate Blanchett

## Financially Important Individual Genres

in order to get data on the performance of each individual genre, I would have to use a custom groupby mapper function in pandas . To build such a mapper function, I first had to obtain a list of all of the unique individual genres that make up the multitude of unique genre combinations in the genres column.

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

---

In [35]:

```
1 # Getting a list of unique genres
2 #####
3 genres_split = fin_df['genres'].map(lambda x: x.split(','))
4 all_genres = [g for g_list in genres_split for g in g_list]
5 unique_genres = sorted(set(all_genres))
6 -----
7 print('\u033[1m'+'Number of Unique Genres:'+'\u033[0m', len(unique_genres))
8 print('-----')
9 print(unique_genres)
```

Number of Unique Genres: 21

-----  
['Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Family', 'Fantasy', 'History', 'Horror', 'Music', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Sport', 'Thriller', 'War', 'Western']

In [36]:

```
1 # Preparing empty lists for important individual genres based on each metric to be added to
2 #####
3 finan_tot_gross_imp_individ_gs = []
4 finan_tot_profits_imp_individ_gs = []
5 finan_avg_gross_imp_individ_gs = []
6 finan_avg_profits_imp_individ_gs = []
```

In [37]:

```
1 # Creating, plotting, and saving the appropriate dataframe
2 #####base_dir = 'visuals/Important Individual Genres/'
3 #
4 for aka_v in all_akas:
5     plot_col = aka_v[4:]
6     #
7     individ_gs_df = pd.DataFrame()
8     #
9     for g in unique_genres:
10         g_map = fin_df['genres'].map(lambda x: g if g in x else None)
11         #
12         g_df_p1 = fin_df.groupby(g_map).count()[['primary_title']]
13         #
14         if 'tot' in aka_v:
15             g_df_p2 = fin_df.groupby(g_map).sum()[[plot_col]]
16         #
17         if 'avg' in aka_v:
18             g_df_p2 = fin_df.groupby(g_map).mean()[[plot_col]]
19         #
20         g_df = pd.concat([g_df_p1, g_df_p2], axis=1).reset_index()
21         #
22         g_df.rename({'primary_title': 'num_titles', plot_col: aka_v}, axis=1, inplace=True)
23         #
24         individ_gs_df = individ_gs_df.append(g_df)
25     #
26     if 'tot' in aka_v:
27         individ_gs_df['perc_of_'+aka_v] = \
28             round((individ_gs_df[aka_v] / fin_df[plot_col].sum()), 4)
29     #
30     individ_gs_df = individ_gs_df.loc[individ_gs_df['num_titles'] >= 9]
31     import_gs_df = individ_gs_df.loc[individ_gs_df[aka_v] >= individ_gs_df[aka_v].quantile(.75)]
32     #
33     import_gs_df = import_gs_df.sort_values(aka_v, ascending=False)
34     #
35     top_gs_rat_df = pd.DataFrame()
36     #
37     box_df = pd.DataFrame()
38     #
39     for g in import_gs_df.genres:
40         top_individ_gs_mapper = fin_df.genres.map(lambda x: g if g in x else None)
41         #
42         if 'tot' in aka_v:
43             rat_agg_df = \
44                 fin_df.groupby([top_individ_gs_mapper, 'rating']).sum()[[plot_col]]\
45                     .reset_index('rating')
46         #
47         rat_df = \
48             rat_agg_df.merge(import_gs_df.set_index('genres'), left_index=True, right_index=True) \
49                 .reset_index().rename({aka_v: 'top_cat', plot_col: aka_v}, axis=1) \
50                     .sort_values('top_cat', ascending=False)
51         #
52         if 'avg' in aka_v:
53             rat_count_df = \
54                 fin_df.groupby([top_individ_gs_mapper, 'rating']).count()[['primary_title']]
55         #
56         rat_agg_df = \
57             fin_df.groupby([top_individ_gs_mapper, 'rating']).mean()[[plot_col]]
58         #
59         rat_df = \
60             rat_agg_df.merge(rat_count_df, left_index=True, right_index=True) \
61                 .reset_index('rating') \
62                     .merge(import_gs_df.set_index('genres')[[aka_v]], left_index=True, \
63                         right_index=True).reset_index() \
64                             .rename({'primary_title': 'num_titles', aka_v: 'top_cat', plot_col: aka_v}, \
65                                 axis=1).sort_values(aka_v, ascending=False)
66         #
67         g_box_df = fin_df.copy()
68         g_box_df['genres'] = top_individ_gs_mapper
69         g_box_df = g_box_df.dropna(subset=['genres'])[['genres', plot_col]]
```

```

71      #-
72      fin_import_order_dict[g] = g_box_df[plot_col].mean()
73      #
74      top_gs_rat_df = top_gs_rat_df.append(rat_df)
75      #
76      box_df = box_df.append(g_box_df)
77      #
78      fin_import_order = \
79      sorted(fin_import_order_dict.keys(), key=lambda k: fin_import_order_dict[k], reverse=True)
80      #
81      box_df = \
82      box_df.sort_values('genres',
83                          key=lambda g_col: g_col.map(lambda x: fin_import_order.index(x)))
84      #
85      fig, fig_tit = cat_plot(import_gs_df, 'individs', 'genres')
86      #
87      if not os.path.isdir(base_dir + fig_tit + '/'):
88          os.mkdir(base_dir + fig_tit + '/')
89      #
90      f_name = base_dir + fig_tit + '/1 - Important Individual Genres.jpeg'
91      #
92      fig.savefig(f_name, bbox_inches='tight')
93      plt.close(fig)
94      #
95      new_dir = base_dir + fig_tit + '/'
96      #
97      fig, fig_tit = cat_plot(box_df, 'individs', 'genres', aka_v)
98      #
99      f_name = new_dir + '2 - ' + fig_tit + '.jpeg'
100     #
101     fig.savefig(f_name, bbox_inches='tight')
102     plt.close(fig)
103     #
104    fig, fig_tit = cat_plot(top_gs_rat_df, 'rating', 'individs')
105    #
106    f_name = new_dir + '3 - Important Individual Genres - MPA Ratings.jpeg'
107    #
108    fig.savefig(f_name, bbox_inches='tight')
109    plt.close(fig)
110    #
111    if 'tot' in aka_v:
112        if 'gross' in aka_v: finan_tot_gross_imp_individ_gs = list(import_gs_df.genres)
113        if 'profits' in aka_v: finan_tot_profits_imp_individ_gs = list(import_gs_df.genres)
114    if 'avg' in aka_v:
115        if 'gross' in aka_v: finan_avg_gross_imp_individ_gs = list(import_gs_df.genres)
116        if 'profits' in aka_v: finan_avg_profits_imp_individ_gs = list(import_gs_df.genres)
117    #
118    # display(fin_individ_all_cols_df)

```

## Financially Important Individual Genres - Crew

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

---

In [38]:

```
1 # Creating, plotting, and saving the appropriate dataframes
2 #####
3 base_dir = 'visuals/Important Individual Genres - Crew'
4 #
5 for crew_str, crew_df in zip(crew_strings, crew_dfs):
6     if crew_str != 'actress': plot_type = crew_str + 's'
7     else: plot_type = crew_str + 'es'
8 #
9     crew_dir = base_dir + plot_type.title() + '/'
10 #
11 for aka_v in all_akas:
12     plot_col = aka_v[4:]
13 #
14     if 'tot' in aka_v:
15         if 'gross' in aka_v: imp_individ_gs = finan_tot_gross_imp_individ_gs
16         if 'profits' in aka_v: imp_individ_gs = finan_tot_profits_imp_individ_gs
17     if 'avg' in aka_v:
18         if 'gross' in aka_v: imp_individ_gs = finan_avg_gross_imp_individ_gs
19         if 'profits' in aka_v: imp_individ_gs = finan_avg_profits_imp_individ_gs
20 #
21     for g in imp_individ_gs:
22         g_map = crew_df['genres'].map(lambda x: g if g in x else None)
23 #
24     crew_num_titles = crew_df.groupby(['primary_name', g_map]).count()[['primary_title']]
25 #
26     if 'tot' in aka_v:
27         crew_aka_v = crew_df.groupby(['primary_name', g_map]).sum()[[plot_col]]
28     if 'avg' in aka_v:
29         crew_aka_v = crew_df.groupby(['primary_name', g_map]).mean()[[plot_col]]
30 #
31     g_crew_df = pd.concat([crew_num_titles, crew_aka_v], axis=1) \
32         .sort_values(plot_col, ascending=False)
33 #
34     g_crew_df.rename(columns={'primary_title': 'num_titles',
35                         plot_col: aka_v}, inplace=True)
36 #
37     if 'tot' in aka_v:
38         g_crew_df['perc_of_'+ aka_v] = \
39             round((g_crew_df[aka_v] / g_crew_df[aka_v].sum()), 4)
40 #
41     g_top_crew_df = g_crew_df.reset_index('primary_name').head(10)
42 #
43     fig, fig_tit = \
44         cat_plot(g_top_crew_df, 'individs', plot_type)
45 #
46     if not os.path.isdir(crew_dir + fig_tit + '/'):
47         os.mkdir(crew_dir + fig_tit + '/')
48 #
49     f_name = crew_dir + fig_tit + '/' + g + '.jpeg'
50 #
51     fig.savefig(f_name, bbox_inches='tight')
52     plt.close(fig)
```

## § OpusData Category Types

---

### Financially Important Genres

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

---

In [39]:

```
1 # Preparing empty lists for important genres based on each metric to be added to
2 #####
3 finan_tot_gross_imp_genres = []
4 finan_tot_profits_imp_genres = []
5 finan_avg_gross_imp_genres = []
6 finan_avg_profits_imp_genres = []
```

In [40]:

```
1 # Creating, plotting, and saving the appropriate dataframes
2 #####
3 base_dir = 'visuals/Important Genres/'
4 #
5 for aka_v in all_akas:
6     plot_col = aka_v[4:]
7     #
8     g_count_df = fin_df.groupby('genre').count()[['primary_title']]
9     #
10    if 'tot' in aka_v:
11        g_agg_df = fin_df.groupby('genre').sum()[[plot_col]]
12    if 'avg' in aka_v:
13        g_agg_df = fin_df.groupby('genre').mean()[[plot_col]]
14    #
15    genre_df = pd.concat([g_count_df, g_agg_df], axis=1)
16    #
17    genre_df = genre_df.rename(columns={'primary_title': 'num_titles', plot_col: aka_v})
18    #
19    if 'tot' in aka_v:
20        genre_df['perc_of_'+ aka_v] = round((genre_df[aka_v] / fin_df[plot_col].sum()), 4)
21    #
22    genre_df = genre_df.loc[genre_df['num_titles'] >= 9]
23    genre_df = genre_df.loc[genre_df[aka_v] >= genre_df[aka_v].quantile(.75)]
24    #
25    all_years_top10 = genre_df.sort_values(aka_v, ascending=False).reset_index()\
26                                .rename(columns={'genre': 'genres'}).head(10)
27    #
28    top_combo_mapper = \
29    fin_df.genre.map(lambda x: x if x in list(all_years_top10.genres) else None)
30    #
31    imp_list = list(all_years_top10.genres)
32    #
33    if 'tot' in aka_v:
34        rat_agg_df = \
35        fin_df.groupby([top_combo_mapper, 'rating']).sum()[[plot_col]].reset_index('rating')
36    #
37        rat_df = \
38        rat_agg_df.merge(all_years_top10.set_index('genres'), left_index=True, right_index=True)\\
39                        .reset_index().rename({aka_v: 'top_cat', plot_col: aka_v}, axis=1)\\
40                        .sort_values('top_cat', ascending=False)
41    #
42    if 'avg' in aka_v:
43        rat_count_df = fin_df.groupby([top_combo_mapper, 'rating']).count()[['primary_title']]
44    #
45        rat_agg_df = \
46        fin_df.groupby([top_combo_mapper, 'rating']).mean()[[plot_col]]
47    #
48        rat_df = \
49        rat_agg_df.merge(rat_count_df, left_index=True, right_index=True).reset_index('rating')\
50                        .merge(all_years_top10.set_index('genres')[[aka_v]], left_index=True,
51                                         right_index=True).reset_index()\\
52                                         .rename({'primary_title':'num_titles', aka_v:'top_cat', plot_col: aka_v},
53                                         axis=1).sort_values(['top_cat', aka_v], ascending=False)
54    #
55        rat_df = rat_df.rename(columns={'index': 'genres'})
56    #
57        box_df = pd.DataFrame()
58        fin_import_order_dict = {}
59    #
60    for genre in all_years_top10.genres:
61        imp_genre_map = fin_df['genre'].map(lambda x: True if genre==x else False)
62    #
63        imp_genres_df = fin_df.loc[imp_genre_map].copy()[['genre', plot_col]]
64    #
65        fin_import_order_dict[genre] = imp_genres_df[plot_col].mean()
66    #
67        box_df = box_df.append(imp_genres_df)
68    #
69        fin_import_order = \
70        sorted(fin_import_order_dict.keys(), key=lambda k: fin_import_order_dict[k], reverse=True)
```

```

71  #-
72  box_df = box_df.sort_values('genre',
73      key=lambda g_col: g_col.map(lambda x: fin_import_order.index(x)))
74  #-
75  box_df = box_df.rename(columns={'genre': 'genres'})
76  #-
77  fig, fig_tit = cat_plot(all_years_top10, 'genres', 'genres')
78  #-
79  if not os.path.isdir(base_dir + fig_tit + '/'):
80      os.mkdir(base_dir + fig_tit + '/')
81  #-
82  f_name = base_dir + fig_tit + '1 - Top Genres.jpeg'
83  #-
84  fig.savefig(f_name, bbox_inches='tight')
85  plt.close(fig)
86  #-
87  new_dir = base_dir + fig_tit + '/'
88  #-
89  fig, fig_tit = cat_plot(box_df, 'genres', 'genres', aka_v)
90  #-
91  f_name = new_dir + '2 - ' + fig_tit + '.jpeg'
92  #-
93  fig.savefig(f_name, bbox_inches='tight')
94  plt.close(fig)
95  #-
96  fig, fig_tit = cat_plot(rat_df, 'rating', 'genres')
97  #-
98  f_name = new_dir + '3 - Top Genres - MPA Ratings.jpeg'
99  #-
100 fig.savefig(f_name, bbox_inches='tight')
101 plt.close(fig)
102 #-
103 if 'tot' in aka_v:
104     if 'gross' in aka_v: finan_tot_gross_imp_genres = list(all_years_top10.genres)
105     if 'profits' in aka_v: finan_tot_profits_imp_genres = list(all_years_top10.genres)
106 if 'avg' in aka_v:
107     if 'gross' in aka_v: finan_avg_gross_imp_genres = list(all_years_top10.genres)
108     if 'profits' in aka_v: finan_avg_profits_imp_genres = list(all_years_top10.genres)

```

## Financially Important Genres - Crew

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

---

In [41]:

```
1 # Creating, plotting, and saving the appropriate dataframes
2 #####
3 base_dir = 'visuals/Important Genres - Crew/'
4 #
5 for crew_str, crew_df in zip(crew_strings, crew_dfs):
6     if crew_str != 'actress': plot_type = crew_str + 's'
7     else: plot_type = crew_str + 'es'
8 #
9     crew_dir = base_dir + plot_type.title() + '/'
10 #
11 for aka_v in all_akas:
12     plot_col = aka_v[4:]
13 #
14     if 'tot' in aka_v:
15         if 'gross' in aka_v: imp_genres = finan_tot_gross_imp_genres
16         if 'profits' in aka_v: imp_genres = finan_tot_profits_imp_genres
17     if 'avg' in aka_v:
18         if 'gross' in aka_v: imp_genres = finan_avg_gross_imp_genres
19         if 'profits' in aka_v: imp_genres = finan_avg_profits_imp_genres
20 #
21     for genre in imp_genres:
22         genre_df = crew_df.set_index('genre').loc[genre].copy().drop(columns={'genres'})
23 #
24         if type(genre_df) == pd.Series:
25             print('\033[1mGenre: \033[0m', genre)
26             print('\033[1mFinancial Metric: \033[0m', aka_v)
27             print('This genre in this metric was only '+'\
28                   'associated with one '+ crew_str+'.')
29             print("The crew member's name was: ", genre_df['primary_name'])
30             print('-----')
31             continue
32 #
33         genre_df = genre_df.reset_index().rename(columns={'genre': 'genres'})
34 #
35         top_crew_num_titles = \
36         genre_df.groupby('primary_name').count()[['primary_title', 'genres']]
37 #
38         if 'tot' in aka_v:
39             top_crew_aka_df = genre_df.groupby('primary_name').sum()[[plot_col]]
40         if 'avg' in aka_v:
41             top_crew_aka_df = genre_df.groupby('primary_name').mean()[[plot_col]]
42 #
43         genre_top_crew_df = pd.concat([top_crew_num_titles, top_crew_aka_df], axis=1)\.
44             .sort_values(plot_col, ascending=False).head(10)
45 #
46         genre_top_crew_df.rename(columns={'primary_title': 'num_titles', plot_col: aka_v},
47                                   inplace=True)
48 #
49         if 'tot' in aka_v:
50             genre_top_crew_df['perc_of_' + aka_v] = \
51             round((genre_top_crew_df[aka_v] / genre_df[plot_col].sum()), 4)
52 #
53         genre_top_crew_df = genre_top_crew_df.reset_index()
54         genre_top_crew_df['genres'] = genre_top_crew_df['genres'].map(lambda x: genre)
55         genre_top_crew_df = genre_top_crew_df.set_index('genres')
56 #
57         fig, fig_tit = \
58         cat_plot(genre_top_crew_df, 'genres', plot_type)
59 #
60         if not os.path.isdir(crew_dir + fig_tit + '/'):
61             os.mkdir(crew_dir + fig_tit + '/')
62 #
63         if '/' in genre:
64             genre = ' - '.join(genre.split('/'))
65 #
66         f_name = crew_dir + fig_tit + '/' + genre + '.jpeg'
67 #
68         fig.savefig(f_name, bbox_inches='tight')
69         plt.close(fig)
```

# Financially Important Creative Types

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

---

```
In [42]: 1 # Preparing empty Lists for important creative types based on each metric to be added to
2 #####finan_tot_gross_imp_creative_types = []
3 finan_tot_profits_imp_creative_types = []
4 finan_avg_gross_imp_creative_types = []
5 finan_avg_profits_imp_creative_types = []
```

In [43]:

```
1 # Creating, plotting, and saving the appropriate dataframe
2 #####base_dir = 'visuals/Important Creative Types/'
3 #
4 for aka_v in all_akas:
5     plot_col = aka_v[4:]
6     #
7     ct_count_df = fin_df.groupby('creative_type').count()[['primary_title']]
8     #
9     if 'tot' in aka_v:
10         ct_agg_df = fin_df.groupby('creative_type').sum()[[plot_col]]
11     if 'avg' in aka_v:
12         ct_agg_df = fin_df.groupby('creative_type').mean()[[plot_col]]
13     #
14     creative_df = pd.concat([ct_count_df, ct_agg_df], axis=1)
15     #
16     creative_df = creative_df.rename(columns={'primary_title':'num_titles', plot_col: aka_v})
17     #
18     if 'tot' in aka_v:
19         creative_df['perc_of_'+ aka_v] = round((creative_df[aka_v] / fin_df[plot_col].sum()), 4)
20     #
21     if aka_v == sum_aka_1:
22         fin_creative_df = creative_df.copy()
23     elif aka_v == sum_aka_2:
24         fin_creative_df = \
25             pd.concat([fin_creative_df, creative_df[[aka_v, 'perc_of_'+ aka_v]]], axis=1)
26     else:
27         fin_creative_df = pd.concat([fin_creative_df, creative_df[aka_v]], axis=1)
28     #
29     creative_df = creative_df.loc[creative_df['num_titles'] >= 9]
30     creative_df = creative_df.loc[creative_df[aka_v] >= creative_df[aka_v].quantile(2/3)]
31     #
32     all_years_top10 = creative_df.sort_values(aka_v, ascending=False).reset_index().head(10)
33     #
34     top_c_type_mapper = \
35         fin_df.creative_type.map(lambda x: x if x in list(all_years_top10.creative_type) else None)
36     #
37     if 'tot' in aka_v:
38         rat_agg_df = \
39             fin_df.groupby([top_c_type_mapper, 'rating']).sum()[[plot_col]].reset_index('rating')
40         #
41         rat_df = \
42             rat_agg_df.merge(all_years_top10.set_index('creative_type'), left_index=True,
43                               right_index=True)\n43                 .reset_index().rename({aka_v:'top_cat', plot_col: aka_v}, axis=1)\n44                 .sort_values('top_cat', ascending=False)
45     #
46     if 'avg' in aka_v:
47         rat_count_df = fin_df.groupby([top_c_type_mapper, 'rating']).count()[['primary_title']]
48         #
49         rat_agg_df = \
50             fin_df.groupby([top_c_type_mapper, 'rating']).mean()[[plot_col]]
51         #
52         rat_df = \
53             rat_agg_df.merge(rat_count_df, left_index=True, right_index=True).reset_index('rating')\n54                 .merge(all_years_top10.set_index('creative_type')[[aka_v]], left_index=True,
55                               right_index=True).reset_index()\n56                 .rename({'primary_title':'num_titles', aka_v:'top_cat', plot_col: aka_v},
57                         axis=1).sort_values(['top_cat', aka_v], ascending=False)
58     #
59     box_df = pd.DataFrame()
60     fin_import_order_dict = {}
61     #
62     for c_type in all_years_top10.creative_type:
63         imp_c_type_map = fin_df['creative_type'].map(lambda x: True if c_type==x else False)
64         #
65         imp_c_type_df = fin_df.loc[imp_c_type_map].copy()[['creative_type', plot_col]]
66         #
67         fin_import_order_dict[c_type] = imp_c_type_df[plot_col].mean()
68         #
69         #
```

```

71     box_df = box_df.append(imp_c_type_df)
72 #-
73 fin_import_order = \
74 sorted(fin_import_order_dict.keys(), key=lambda k: fin_import_order_dict[k], reverse=True)
75 #-
76 box_df = box_df.sort_values('creative_type',
77                             key=lambda g_col: g_col.map(lambda x: fin_import_order.index(x)))
78 #-
79 fig, fig_tit = cat_plot(all_years_top10, 'creative_type', 'creative_type')
80 #-
81 if not os.path.isdir(base_dir + fig_tit + '/'):
82     os.mkdir(base_dir + fig_tit + '/')
83 #-
84 f_name = base_dir + fig_tit + '/1 - Top Creative Types.jpeg'
85 #-
86 fig.savefig(f_name, bbox_inches='tight')
87 plt.close(fig)
88 #-
89 new_dir = base_dir + fig_tit + '/'
90 #-
91 fig, fig_tit = cat_plot(box_df, 'creative_type', 'creative_type', aka_v)
92 #-
93 f_name = new_dir + '2 - ' + fig_tit + '.jpeg'
94 #-
95 fig.savefig(f_name, bbox_inches='tight')
96 plt.close(fig)
97 #-
98 fig, fig_tit = cat_plot(rat_df, 'rating', 'creative_type')
99 #-
100 f_name = new_dir + '3 - Top Creative Types - MPA Ratings.jpeg'
101 #-
102 fig.savefig(f_name, bbox_inches='tight')
103 plt.close(fig)
104 #-
105 if 'tot' in aka_v:
106     if 'gross' in aka_v:
107         finan_tot_gross_imp_creative_types = list(all_years_top10.creative_type)
108     if 'profits' in aka_v:
109         finan_tot_profits_imp_creative_types = list(all_years_top10.creative_type)
110 if 'avg' in aka_v:
111     if 'gross' in aka_v:
112         finan_avg_gross_imp_creative_types = list(all_years_top10.creative_type)
113     if 'profits' in aka_v:
114         finan_avg_profits_imp_creative_types = list(all_years_top10.creative_type)
115 #
116 # display(fin_creative_df)

```

## Financially Important Creative Type - Crew

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

---

In [44]:

```
1 # Creating, plotting, and saving the appropriate dataframes
2 #####
3 base_dir = 'visuals/Important Creative Types - Crew/'
4 #
5 for crew_str, crew_df in zip(crew_strings, crew_dfs):
6     if crew_str != 'actress': plot_type = crew_str + 's'
7     else: plot_type = crew_str + 'es'
8 #
9     crew_dir = base_dir + plot_type.title() + '/'
10 #
11 for aka_v in all_akas:
12     plot_col = aka_v[4:]
13 #
14     if 'tot' in aka_v:
15         if 'gross' in aka_v: imp_creative_types = finan_tot_gross_imp_creative_types
16         if 'profits' in aka_v: imp_creative_types = finan_tot_profits_imp_creative_types
17     if 'avg' in aka_v:
18         if 'gross' in aka_v: imp_creative_types = finan_avg_gross_imp_creative_types
19         if 'profits' in aka_v: imp_creative_types = finan_avg_profits_imp_creative_types
20 #
21     for creative_type in imp_creative_types:
22         creative_type_df = crew_df.set_index('creative_type').loc[creative_type].copy()
23 #
24         creative_type_df.reset_index(inplace=True)
25 #
26         top_crew_num_titles = \
27             creative_type_df.groupby('primary_name').count()[['primary_title', 'creative_type']]
28 #
29         if 'tot' in aka_v:
30             top_crew_aka_df = creative_type_df.groupby('primary_name').sum()[[plot_col]]
31         if 'avg' in aka_v:
32             top_crew_aka_df = creative_type_df.groupby('primary_name').mean()[[plot_col]]
33 #
34         creative_type_top_crew_df = \
35             pd.concat([top_crew_num_titles, top_crew_aka_df], axis=1)\.
36                 .sort_values(plot_col, ascending=False).head(10)
37 #
38         creative_type_top_crew_df.rename(columns={'primary_title': 'num_titles',
39                                              plot_col: aka_v}, inplace=True)
40 #
41         if 'tot' in aka_v:
42             creative_type_top_crew_df['perc_of_' + aka_v] = \
43                 round((creative_type_top_crew_df[aka_v] / creative_type_df[plot_col].sum()), 4)
44 #
45         creative_type_top_crew_df = creative_type_top_crew_df.reset_index()
46 #
47         creative_type_top_crew_df['creative_type'] = \
48             creative_type_top_crew_df['creative_type'].map(lambda x: creative_type)
49 #
50         creative_type_top_crew_df = creative_type_top_crew_df.set_index('creative_type')
51 #
52         fig, fig_tit = \
53             cat_plot(creative_type_top_crew_df, 'creative_type', plot_type)
54 #
55         if not os.path.isdir(crew_dir + fig_tit + '/'):
56             os.mkdir(crew_dir + fig_tit + '/')
57 #
58         f_name = crew_dir + fig_tit + '/' + creative_type + '.jpeg'
59 #
60         fig.savefig(f_name, bbox_inches='tight')
61         plt.close(fig)
```

## § Final Analysis and Recommendations

Any of the categories within each category type that were deemed important for any of the metrics could be considered as a recommendation for that metric. However, I also explored the resulting visualizations to develop my own specific recommendations, based on the features of their boxplots, of the important categories and subcategories for each metric.

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 could then use the categories I recommended to guide them, or develop their own if they actually run the code, and fully explore those important categories to see the recommended crew members. By doing so, they will be maximizing the benefits of the critical insights I gained through my analysis.

## Recommendations to Maximize Subscribers

Movies with the categories and subcategories shown below have resulted in the highest number of ticket sales, or the highest average number of ticket sales. My first recommendation for Microsoft is to produce content with these categories and subcategories, as they will attract the highest number of subscribers to their new streaming service.

### Important Total Gross Revenue Categories

| IMDb Category Types       | Recommended Categories<br><i>(In order of Importance)</i> | Recommended MPA Ratings for each Category<br><i>(In order of Importance for each Category)</i> |
|---------------------------|---|--|
| Unique Genre Combinations | Action, Adventure, Sci-Fi                                 | PG-13  |
|                           | Action, Adventure, Animation                              | PG   |
|                           | Adventure, Family, Fantasy                                | PG-13 • PG   |
|                           | Adventure, Animation, Comedy                              | PG • G   |
|                           | Action, Adventure, Fantasy                                | PG-13  |
| Individual Genres         | Action, Adventure, Comedy                                 | PG-13 • R  |
|                           | Sci-Fi  | PG-13  |
|                           | Adventure   | PG-13 • PG   |
|                           | Action  | PG-13 • R • PG   |
|                           | Comedy  | PG • PG-13 • R   |
| OpusData Catgory Types    | Drama   | PG-13 • R  |
|                           | Adventure   | PG • PG-13   |
|                           | Action  | PG-13 • R  |
|                           | Drama   | PG-13 • R  |
|                           | Kids Fiction  | PG • G   |
| Creative Types            | Science Fiction   | PG-13 • R  |
|                           | Contemporary Fiction                                      | PG-13 • R  |

### Important Average Gross Revenue Categories

| IMDb Category Types       | Recommended Categories<br><i>(In order of Importance)</i> | Recommended MPA Ratings for each Category<br><i>(In order of Importance for each Category)</i> |
|---------------------------|---|--|
| Unique Genre Combinations | Action, Adventure, Sci-Fi                                 | PG-13 • R  |
|                           | Adventure, Fantasy  | PG-13  |
|                           | Action, Adventure, Animation                              | PG   |
|                           | Animation, Comedy, Family                                 | PG   |
|                           | Adventure, Drama, Fantasy                                 | PG-13 • PG-13 • R  |
| Individual Genres         | Adventure, Family, Fantasy                                | PG-13 • PG   |
|                           | Sci-Fi  | PG-13 • R  |
|                           | Animation   | G • PG • R   |
|                           | Adventure   | PG-13 • G • PG • R   |
|                           | Fantasy   | PG-13 • PG • R   |
| OpusData Catgory Types    | Action  | PG-13 • PG • R   |
|                           | Musical   | PG • PG-13   |
|                           | Adventure   | PG-13 • G • PG • R   |
|                           | Action  | PG-13 • PG • R   |
|                           | Super Hero  | PG-13 • R • PG   |
| Creative Types            | Kids Fiction  | G • PG • PG-13   |
|                           | Science Fiction   | PG-13 • R • PG   |

## Recommendations to Maximize Profits

Movies with the categories and subcategories shown below have resulted in the highest profits, or the highest average profits. My second recommendation for Microsoft is to produce content with these categories and subcategories, as they can maximize profits and limit their exposure to risk.

## Important Total Profits Categories

| IMDb Category Types       | Recommended Categories<br>(In order of Importance)   | Recommended MPA Ratings for each Category<br>(In order of Importance for each Category) |
|---------------------------|--|---|
| Unique Genre Combinations | Action, Adventure, Animation<br>Adventure, Animation, Comedy<br>Action, Adventure, Comedy<br>Adventure, Family, Fantasy<br>Action, Adventure, Fantasy<br>Action, Adventure, Sci-Fi | PG<br>PG • G<br>PG-13 • R<br>PG-13 • PG<br>PG-13<br>PG-13                               |
| Individual Genres         | Sci-Fi<br>Adventure<br>Action<br>Comedy<br>Drama   | PG-13<br>PG-13 • PG<br>PG-13 • R • PG<br>PG • PG-13 • R<br>PG-13 • R                    |
| OpusData Category Types   | Recommended Categories<br>(In order of Importance)   | Recommended MPA Ratings for each Category<br>(In order of Importance for each Category) |
| Genres                    | Action<br>Adventure<br>Drama   | PG-13 • R<br>PG • PG-13<br>PG-13 • R  |
| Creative Types            | Super Hero<br>Kids Fiction<br>Contemporary Fiction   | PG-13<br>PG • G<br>PG-13 • R  |

## Important Average Profits Categories

| IMDb Category Types       | Recommended Categories<br>(In order of Importance)   | Recommended MPA Ratings for each Category<br>(In order of Importance for each Category) |
|---------------------------|--|---|
| Unique Genre Combinations | Adventure, Drama, Fantasy<br>Adventure, Fantasy<br>Animation, Comedy, Family<br>Drama, Sci-Fi, Thriller<br>Action, Adventure, Animation<br>Adventure, Family, Fantasy<br>Adventure, Animation, Comedy<br>Action, Adventure, Sci-Fi | PG-13 • PG • R<br>PG-13<br>PG<br>PG-13<br>PG<br>PG-13 • PG<br>PG • G • R<br>PG-13       |
| Individual Genres         | Animation<br>Sci-Fi<br>Adventure<br>Fantasy<br>Action  | G • PG • R<br>PG-13 • R<br>PG-13 • G • PG • R<br>PG-13 • PG • R<br>PG-13 • PG • R       |
| OpusData Category Types   | Recommended Categories<br>(In order of Importance)   | Recommended MPA Ratings for each Category<br>(In order of Importance for each Category) |
| Genres                    | Adventure<br>Action<br>Musical   | PG-13 • G • PG • R<br>PG-13 • PG • R<br>PG • PG-13                                      |
| Creative Types            | Super Hero<br>Kids Fiction<br>Science Fiction  | R • PG-13 • PG<br>G • PG<br>PG-13 • R   |

## Interactive Visualization Feature

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 categories 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.

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

By using this feature, one could, for example, easily access the visualization of the best directors of Sci-Fi movies ranked by Average Worldwide Profits with just a few selections in the interactive dropdown menus that appear when the cell below is run.

In order to reset the search menus and explore another important genre or genre combination, all one would have to do is just re-run the cell.

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

In [45]:

```
1 # Creating interactive dropdown menus that would allow users to access any of the vizualisations
2 # created for any of the important genres or genre combinations with ease
3 #####
4 data_choices = ['Choose a data source from the choices below...', 
5                 'IMDb', 'OpusData']
6 #####
7 imdb_choices = ['Choose a category type from the choices below...', 
8                  'Important Unique Genre Combinations', 'Important Individual Genres']
9 #####
10 od_choices = ['Choose a category type from the choices below...', 
11                'Important Genres', 'Important Creative Types']
12 #####
13 genre_choices = ['Important Unique Genre Combinations', 'Important Individual Genres', 
14                   'Important Genres']
15 #####
16 ct_choices = ['Important Creative Types']
17 #####
18 metric_choices = ['Choose a financial metric from the choices below...', 
19                    'Total Worldwide Gross', 'Total Worldwide Profits',
20                    'Average Worldwide Gross', 'Average Worldwide Profits']
21 #####
22 crew_choices = ['Choose a type of crew member from the choices below...', 
23                  'Directors', 'Writers', 'Actors', 'Actresses']
24 #####
25 empty_slot = ['N/A until a selection is made in dropdown menu above']
26 #####
27 menu_layout = widgets.Layout(width='99%', height='30px')
28 #####
29 first_menu = \
30 widgets.Dropdown(options=data_choices, value=data_choices[0], disabled=False, layout=menu_layout)
31 #####
32 second_menu = \
33 widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)
34 #####
35 third_menu = \
36 widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)
37 #####
38 fourth_menu = \
39 widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)
40 #####
41 fifth_menu = \
42 widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)
43 #####
44 sixth_menu = \
45 widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)
46 #####
47 # Function monitoring changes to the widgets
48 #####
49 def f(a, b, c, d, e, f):
50
51     # First Menu
52     #####
53     if a == data_choices[1]:
54         second_menu.options = imdb_choices
55         first_menu.disabled = True
56     #####
57     elif a == data_choices[2]:
58         second_menu.options = od_choices
59         first_menu.disabled = True
60     #####
61     else: second_menu.options = empty_slot
62
63     # Second Menu
64     #####
65     third_choice_1_a = 'Do you wish to explore the '
66     third_choice_1_b = ' themselves, or the Important Crew from those '
67     third_choice_1_c = '?'
68     #####
69     if b in genre_choices or b in ct_choices:
70         third_choice_insert = b
```

```

71      #-
72      third_choice_1 = third_choice_1_a + third_choice_insert + third_choice_1_b +\
73          third_choice_insert + third_choice_1_c
74      #-
75      third_choices = [third_choice_1, third_choice_insert, third_choice_insert + ' - Crew']
76      #-
77      third_menu.options = third_choices
78      second_menu.disabled = True
79      #-
80  else:
81      third_menu.options = empty_slot
82      third_choices = ['empty']
83
84  # Third Menu
85 #####
86 if c in third_choices:
87     if c == third_choices[1]:
88         f_path_p1 = third_choices[1] + '/'
89         #-
90         fourth_menu.options = metric_choices
91         third_menu.disabled = True
92     if c == third_choices[2]:
93         f_path_p1 = third_choices[2] + '/'
94         #-
95         fourth_menu.options = crew_choices
96         third_menu.disabled = True
97     #-
98 else: fourth_menu.options = empty_slot
99
100
101 # Fourth Menu
102 #####
103 if d in metric_choices[1:]:
104     file_path = 'visuals/' + f_path_p1 + d + '/'
105     file_choices =.listdir(file_path)
106     #-
107     file_choices.insert(0, 'Please choose a file from the choices below...')
108     #-
109     fifth_menu.options = file_choices
110     fourth_menu.disabled = True
111     #-
112 elif d in crew_choices[1:]:
113     f_path_p2 = d + '/'
114     #-
115     fifth_menu.options = metric_choices
116     fourth_menu.disabled = True
117     #-
118 else: fifth_menu.options = empty_slot
119
120
121 # Fifth Menu
122 #####
123 if '.jpeg' in e:
124     file_name = file_path + e
125     #-
126     image = Image.open(file_name)
127     display(image)
128     #-
129     sixth_menu.disabled = True
130     #-
131 elif e in metric_choices[1:]:
132     file_path = 'visuals/' + f_path_p1 + f_path_p2 + e + '/'
133     file_choices =.listdir(file_path)
134     #-
135     file_choices.insert(0, 'Please choose a file from the choices below...')
136     #-
137     sixth_menu.options = file_choices
138     fifth_menu.disabled = True
139     #-
140 else: sixth_menu.options = empty_slot
141

```

```

142 # Sixth Menu
143 #####
144 if '.jpeg' in f:
145     file_name = file_path + f
146     #
147     image = Image.open(file_name)
148     display(image)
149
150 # Creating the interactive widget
151 #####
152 ui = widgets.VBox([first_menu, second_menu, third_menu, fourth_menu, fifth_menu, sixth_menu])
153 #
154 out = widgets.interactive_output(f, {'a':first_menu, 'b':second_menu, 'c':third_menu,
155                                     'd':fourth_menu, 'e':fifth_menu, 'f':sixth_menu})
156 #
157 print('\n\033[1m Follow the instructions that appear in each dropdown menu to access the '+\
158       'visualization with the desired properties.\n\n To start a new search, '+'\
159       'just re-run the cell.\n')
160 display(ui, out)

```

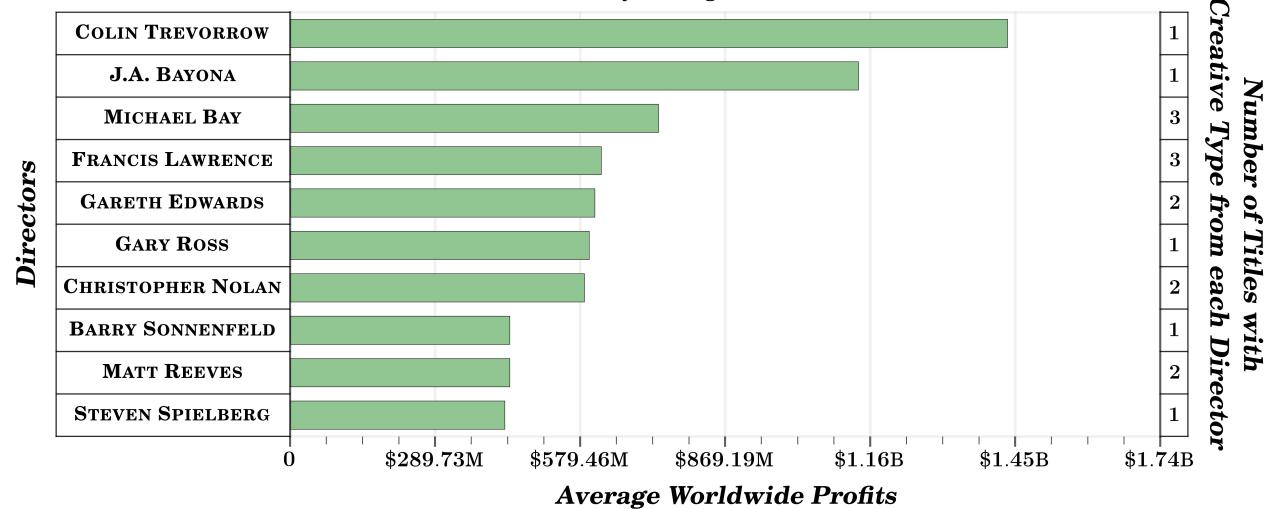
Follow the instructions that appear in each dropdown menu to access the visualization with the desired properties.

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



**OPUSDATA CATEGORICAL DATA**  
**IMDb CREW DATA • THE NUMBERS FINANCIAL DATA**  
**2010 → 2018**  
**SCIENCE FICTION**

*Directors of Movies with the Creative Type Above  
 Ranked by Average Worldwide Profits*



## Closing Database Connection

In [46]:

```
1 # As this is an academic project designed to be run from start to finish, there is no need
2 # to commit and save changes to the database and the file itself is too big for GitHub anyway
3 #####
4 mov_conn.close()
```

## Printing Categories for All of the Category Types to put them in the Presentation

In [47]:

```
1 # Printing information about the category types to be copied into my presentation
2 #####
3 imdb_source = 'IMDb Category Types'
4 od_source = 'OpusData Category Types'
5 #-
6 g_combos_cat = 'Unique Genre Combinations'
7 individ_gs_cat = 'Individual Genres'
8 genres_cat = 'Genres'
9 ct_cat = 'Creative Type'
10 #-
11 imdb_cat_types = [g_combos_cat, individ_gs_cat]
12 od_cat_types = [genres_cat, ct_cat]
13 #-
14 num_combos = ['Number of Different Combinations: ' + str(len(fin_df.genres.unique()))]
15 od_genres = list(filter(None, fin_df.genre.unique()))
16 od_cts = list(filter(None, fin_df.creative_type.unique()))
17 #-
18 imdb_cat_lists = [num_combos, unique_genres]
19 od_cat_lists = [od_genres, od_cts]
20 #-
21 def print_categories(data_source, cat_types, cat_lists):
22     print_out = '\033[1m' + data_source + '\033[0m\n\n'
23     #
24     for cat_type, cat_list in zip(cat_types, cat_lists):
25         print_out = print_out + '\033[1m' + cat_type + '\033[0m\n'
26     #
27     for cat in cat_list:
28         print_out = print_out + cat + '\n'
29     #
30     print_out = print_out + '\n'
31     #
32     print_out = print_out + '\n'
33     #
34     print(print_out)
35 #-
36 print_categories(imdb_source, imdb_cat_types, imdb_cat_lists)
37 print_categories(od_source, od_cat_types, od_cat_lists)
```

## IMDb Category Types

### Unique Genre Combinations

Number of Different Combinations: 183

### Individual Genres

Action  
Adventure  
Animation  
Biography  
Comedy  
Crime  
Documentary  
Drama  
Family  
Fantasy  
History  
Horror  
Music  
Musical  
Mystery  
Romance  
Sci-Fi  
Sport  
Thriller  
War  
Western

## OpusData Category Types

**Genres**

Adventure  
Action  
Drama  
Musical  
Comedy  
Horror  
Thriller/Suspense  
Romantic Comedy  
Black Comedy  
Western  
Concert/Performance

**Creative Type**

Kids Fiction  
Contemporary Fiction  
Historical Fiction  
Science Fiction  
Fantasy  
Dramatization  
Super Hero  
Factual  
Multiple Creative Types

## § Markdown Crew Exploration

---

### Important Unique Genre Combinations

[Click Here to explore the Important Crew of the Important Unique Genre Combinations based on Total Worldwide Gross.](#)

---

[Click Here to explore the Important Crew of the Important Unique Genre Combinations based on Total Worldwide Profits.](#)

---

[Click Here to explore the Important Crew of the Important Unique Genre Combinations based on Average Worldwide Gross.](#)

---

[Click Here to explore the Important Crew of the Important Unique Genre Combinations based on Average Worldwide Profits.](#)

---

### Important Individual Genres

[Click Here to explore the Important Crew of the Important Individual Genres based on Total Worldwide Gross.](#)

---

[Click Here to explore the Important Crew of the Important Individual Genres based on Total Worldwide Profits.](#)

---

[Click Here to explore the Important Crew of the Important Individual Genres based on Average Worldwide Gross.](#)

---

[Click Here to explore the Important Crew of the Important Individual Genres based on Average Worldwide Profits.](#)

---

## Important Genres

[Click Here to explore the Important Crew of the Important Genres based on Total Worldwide Gross.](#)

---

[Click Here to explore the Important Crew of the Important Genres based on Total Worldwide Profits.](#)

---

[Click Here to explore the Important Crew of the Important Genres based on Average Worldwide Gross.](#)

---

[Click Here to explore the Important Crew of the Important Genres based on Average Worldwide Profits.](#)

---

## Important CreativeTypes

[Click Here to explore the Important Crew of the Important Creative Types based on Total Worldwide Gross.](#)

---

[Click Here to explore the Important Crew of the Important Creative Types based on Total Worldwide Profits.](#)

---

[Click Here to explore the Important Crew of the Important Creative Types based on Average Worldwide Gross.](#)

---

[Click Here to explore the Important Crew of the Important Creative Types based on Average Worldwide Profits.](#)

---

**END OF PROJECT**

