# § 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


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/) and OpusData (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/) - Unique Genre Combinations
- IMDb (https://www.imdb.com/) - Individual Genres
- OpusData (https://www.opusdata.com/) - Creative Types
- OpusData (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/) - Total Worldwide Gross Revenue
- The Numbers (https://www.the-numbers.com/) - Total Worldwide Profits
- The Numbers (https://www.the-numbers.com/) - Average Worldwide Gross Revenue
- The Numbers (https://www.the-numbers.com/) - Average Worldwide Profits

I then determined which of the Motion Picture Association (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/), The Movie DataBase (TMDB) (https://www.themoviedb.org/), The Numbers (https://www.the-numbers.com/), Box Office Mojo

(https://www.boxofficemojo.com/), and Rotten Tomatoes (https://www.rottentomatoes.com/). I also applied for and received an academic dataset from OpusData (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/)'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 exlude the Rotten Tomatoes data from my final analysis.

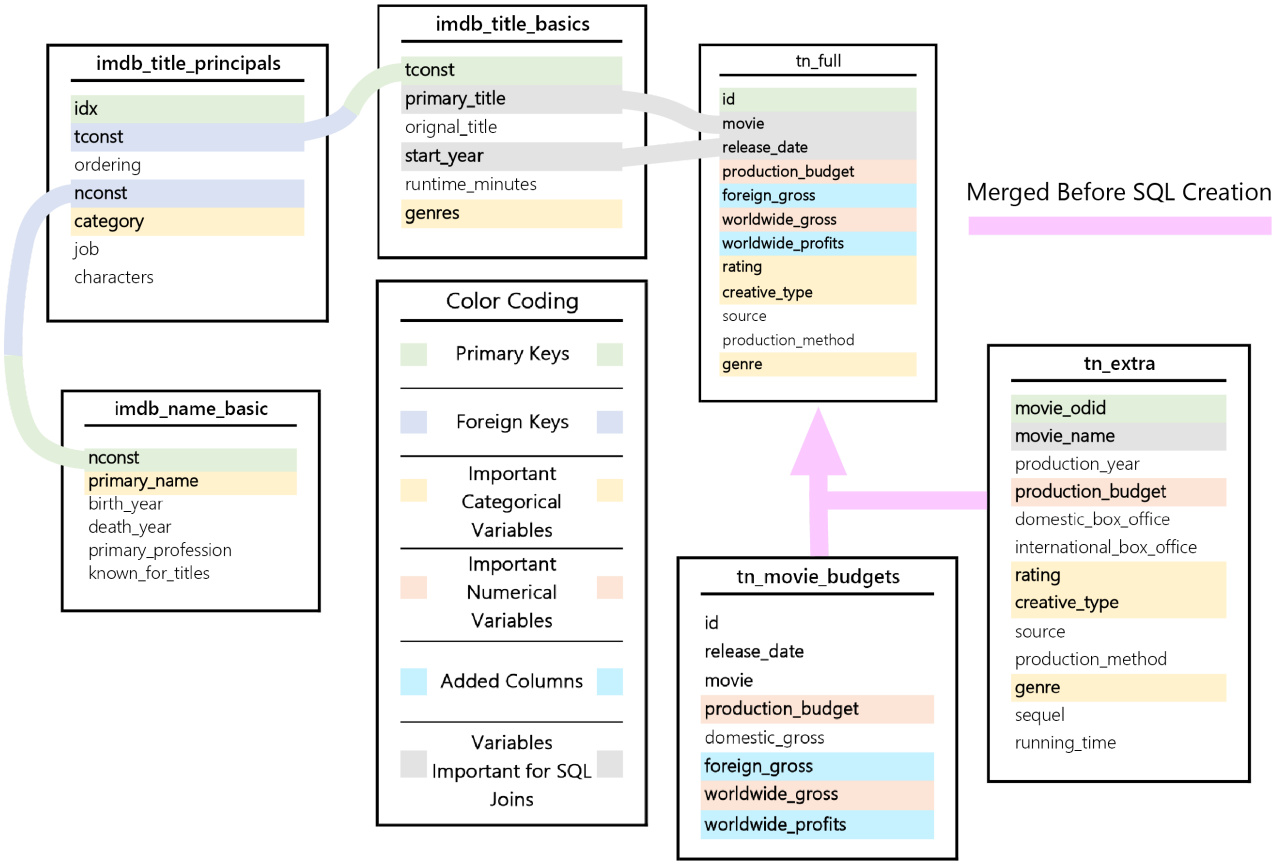
Those reasons are also why I decided to use the data I recieved from OpusData. Not only did it have a `rating` column, but it had even more movies than the entire Rotten Tomatoes dataset, all of which were comparable as it contained a `title` column as well as similar columns to those in The Numbers 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]:

```
# Importing the necessary modules, setting up the mapping function for passing SQL queries into,
# and enabling inline plotting for when I was building the plotting functions
#########################################################################################################
import os
from os import listdir
import string
import numpy as np
import pandas as pd
import sqlite3
#--------------------------------------------------------------------------------------------------------
from pandasql import sqldf
pysqldf = lambda q: sqldf(q, globals())
#--------------------------------------------------------------------------------------------------------
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import matplotlib.patches as mpatches
from matplotlib import cm
from matplotlib.colors import LinearSegmentedColormap
#--------------------------------------------------------------------------------------------------------
import ipywidgets as widgets
#--------------------------------------------------------------------------------------------------------
from PIL import Image
#--------------------------------------------------------------------------------------------------------
%matplotlib inline
```

In [2]:

```
# Importing the data into dataframes
#########################################################################################################
imdb_title_principals = pd.read_csv('data/title.principals.csv')
imdb_title_basics = pd.read_csv('data/title.basics.csv')
imdb_name_basic = pd.read_csv('data/name.basics.csv')
tn_movie_budgets = pd.read_csv('data/tn.movie_budgets.csv')
```

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

In [4]:
```
# Setting up lists of the dataframes and their names to iterate through while checking out the
# data and while creating the SQL database
#########################################################################################################
df_names = ['imdb_title_principals', 'imdb_title_basics', 'imdb_name_basic', 'tn_movie_budgets',
            'tn_extra']
#--------------------------------------------------------------------------------------------------------
all_dfs = [imdb_title_principals, imdb_title_basics, imdb_name_basic, tn_movie_budgets, tn_extra]
#--------------------------------------------------------------------------------------------------------
for name, df in zip(df_names, all_dfs):
    print('\033[1m'+ name + ':'+'\033[0m')
    display(df.head())
    display(df.info())
    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
---  ------      --------------    -----
```

# 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]:

```
# Removing any characters that would get in the way of changing columns with financial data
# to a float dtype
#################################################################################################
finan_cols = ['production_budget', 'domestic_gross', 'worldwide_gross']
#-------------------------------------------------------------------------------------------------
for col in finan_cols:
    tn_movie_budgets[col] = tn_movie_budgets[col].str.strip('$')
    tn_movie_budgets[col] = tn_movie_budgets[col].str.replace(',', '')
#-------------------------------------------------------------------------------------------------
tn_movie_budgets[finan_cols] = tn_movie_budgets[finan_cols].astype(float)
#-------------------------------------------------------------------------------------------------
display(tn_movie_budgets.head(3))
display(tn_movie_budgets.info())
```

| | id | release_date | movie | production_budget | domestic_gross | worldwide_gross |
|---|---|---|---|---|---|---|
| **0** | 1 | Dec 18, 2009 | Avatar | 425000000.0 | 760507625.0 | 2.776345e+09 |
| **1** | 2 | May 20, 2011 | Pirates of the Caribbean: On Stranger Tides | 410600000.0 | 241063875.0 | 1.045664e+09 |
| **2** | 3 | Jun 7, 2019 | Dark Phoenix | 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]:
```
# Adding financial columns that I may want to use in my analysis
#################################################################################################
foreign_gross = tn_movie_budgets['worldwide_gross'] - tn_movie_budgets['domestic_gross']
#-------------------------------------------------------------------------------------------------
tn_movie_budgets.insert(5, 'foreign_gross', foreign_gross)
#-------------------------------------------------------------------------------------------------
tn_movie_budgets['worldwide_profits'] = tn_movie_budgets['worldwide_gross'] - \
                                        tn_movie_budgets['production_budget']
#-------------------------------------------------------------------------------------------------
display(tn_movie_budgets.head())
display(tn_movie_budgets.info())
```

| | id | release_date | movie | production_budget | domestic_gross | foreign_gross | worldwide_gross | worldwide_profits |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | Dec 18, 2009 | Avatar | 425000000.0 | 760507625.0 | 2.015838e+09 | 2.776345e+09 | 2.351345e+09 |
| **1** | 2 | May 20, 2011 | Pirates of the Caribbean: On Stranger Tides | 410600000.0 | 241063875.0 | 8.046000e+08 | 1.045664e+09 | 6.350639e+08 |
| **2** | 3 | Jun 7, 2019 | Dark Phoenix | 350000000.0 | 42762350.0 | 1.070000e+08 | 1.497624e+08 | -2.002376e+08 |
| **3** | 4 | May 1, 2015 | Avengers: Age of Ultron | 330600000.0 | 459005868.0 | 9.440081e+08 | 1.403014e+09 | 1.072414e+09 |
| **4** | 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]:
```
# Changing all columns with any sort of time data to the same format to make comparison easy
#################################################################################################
tn_movie_budgets.release_date = \
pd.to_datetime(tn_movie_budgets.release_date).map(lambda x: x.strftime("%m/%d/%Y"))
#-------------------------------------------------------------------------------------------------
tn_movie_budgets.release_date = \
pd.to_datetime(tn_movie_budgets.release_date, format="%m/%d/%Y")
#-------------------------------------------------------------------------------------------------
display(tn_movie_budgets.release_date.head(3))
print(tn_movie_budgets.release_date.dt.year.min(), tn_movie_budgets.release_date.dt.year.max())
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]:
```
# Rearranging the column order of OpusData dataframe so that 'movie_odid' column was first
###############################################################################################
tn_cols = [tn_extra.columns[2]] + list(tn_extra.columns[:2]) + list(tn_extra.columns[3:])
tn_extra = tn_extra[tn_cols]
#----------------------------------------------------------------------------------------------
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 | Sc F |
| **2** | 34620100 | End of the Spear | 2006 | 10000000 | 11748661 | 175380 | PG-13 | His F |

# Limiting 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]:
```
# Limiting any movies to the past ten years
###############################################################################################
imdb_title_basics = imdb_title_basics.loc[(imdb_title_basics.start_year > 2009) & \
                                          (imdb_title_basics.start_year < 2020)]
#----------------------------------------------------------------------------------------------
tn_movie_budgets = tn_movie_budgets.loc[(tn_movie_budgets.release_date.dt.year > 2009) &
                                        (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]:

```
# Merging The Numbers data with OpusData data
#########################################################################################################
tn_p1 = tn_movie_budgets.set_index('movie')
tn_p2 = tn_extra.set_index('movie_name')
#---------------------------------------------------------------------------------------------------------
tn_full = \
tn_p1.merge(tn_p2, how='outer', left_index=True, right_index=True)\
     .reset_index().set_index('id').reset_index()
#---------------------------------------------------------------------------------------------------------
tn_full = tn_full.rename({'index': 'movie'}, axis=1)
#---------------------------------------------------------------------------------------------------------
tn_full = \
tn_full.drop(tn_full.loc[(tn_full.production_budget_x != tn_full.production_budget_y) & \
                         (tn_full.release_date.dt.year != 2019)].index)
#---------------------------------------------------------------------------------------------------------
tn_full = \
tn_full.drop(tn_full.loc[(tn_full.domestic_gross != tn_full.domestic_box_office) & \
                         (tn_full.foreign_gross != tn_full.international_box_office) & \
                         (tn_full.release_date.dt.year != 2019)].index)
#---------------------------------------------------------------------------------------------------------
cols_to_drop = ['production_year', 'movie_odid', 'production_budget_y', 'domestic_box_office',
                'international_box_office', 'sequel', 'running_time']
#---------------------------------------------------------------------------------------------------------
tn_full = tn_full.drop(columns=cols_to_drop)\
                 .rename({'production_budget_x': 'production_budget'}, axis=1)
#---------------------------------------------------------------------------------------------------------
display(tn_full.head(3))
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 | R |
| **5** | 64.0 | 12 Strong | 2018-01-19 | 35000000.0 | 45819713.0 | 25298665.0 | 71118378.0 | 36118378.0 | R |
| **6** | 18.0 | 12 Years a Slave | 2013-10-18 | 20000000.0 | 56671993.0 | 124353350.0 | 181025343.0 | 161025343.0 | R |

```
<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]:
```
# Recreating the lists for database creation to include tn_full
#########################################################################################################
df_names = ['imdb_title_principals', 'imdb_title_basics', 'imdb_name_basic', 'tn_full']
#---------------------------------------------------------------------------------------------------------
all_dfs = [imdb_title_principals, imdb_title_basics, imdb_name_basic, tn_full]
```

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

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

```
[('imdb_title_principals',), ('imdb_title_basics',), ('imdb_name_basic',), ('tn_full',)]
```

In [14]:
```
# Making sure everything worked
#########################################################################################################
for table, table_name in zip(all_dfs, df_names):
    df = pd.DataFrame(mov_cur.execute('SELECT * FROM ' + table_name + ';').fetchall())
    df.columns = [x[0] for x in mov_cur.description]
    #-----------------------------------------------------------------------------------------------------
    print('\033[1m'+ table_name + ':'+'\033[0m')
    display(df.head(3))
    display(df.info())
    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]:
```
# Setting up a list of spine sides to iterate through to alter their appearances
#########################################################################################################
light_spine_sides = ['top', 'bottom']
dark_spine_sides = ['left', 'right']

# Updating matplotlib rcParams to the desired settings
# I had to include the translation package b/c of one director's name had a non-recognized
# character that caused problems when converting to LaTeX
#########################################################################################################
plt.rcParams.update({'text.usetex': True,
                     'text.latex.preamble': r'\usepackage{ucs}'+\
                                            r'\usepackage[T1]{fontenc}'+\
                                            r'\usepackage[utf8]{inputenc}'+\
                                            r'\usepackage[icelandic,english]{babel}'+\
                                            r'\usepackage{amsmath}',
                     'font.family': ['sans-serif', 'serif'],
                     'font.sans-serif': ['Verdana'],
                     'font.serif': ['New Century Schoolbook'],
                     'mathtext.fontset': 'cm'})

# Uncomment to check and/or change keys
#########################################################################################################
# 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]:
```
# Function to get a color between a given color and pure white based on a number between 0 and 1.
# A higher light_frac results in lighter color.
#########################################################################################################
def get_lighter_color(color_name, light_frac):
    color_map = LinearSegmentedColormap.from_list('', [color_name,'w'], N=9)
    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 subcatgory's performance within each of main categories.

In [17]:
```
# Ratings color dictionary
#########################################################################################################
uniq_ratings = tn_full.rating.unique()
#---------------------------------------------------------------------------------------------------------
ratings_colors_dict = {'Not Rated': 'palegreen',
                       'G': 'orchid',
                       'PG': 'xkcd:aqua blue',
                       'PG-13': 'gold',
                       '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]:
```python
# Function for Tick Params Settings for Axes with Numerical Data on X-Axis
#####################################################################################################
def h_tick_params(h_ax, box=None):
    grid_kw = dict(color='dimgrey', linewidth=3, zorder=0)
    maj_gr_kw = dict(length=12, width=2.4, labelsize=25.5, labelcolor='k', color=(0,0,0,.63))
    min_gr_kw = dict(length=9, width=1.2, color=(0,0,0,.63))
    #-------------------------------------------------------------------------------------------------
    maj_reg_kw = dict(length=0, width=2.4, labelsize=22.5, labelcolor='k', color=(0,0,0,.63))
    #-------------------------------------------------------------------------------------------------
    light_spine_kw = dict(color='dimgrey', linewidth=3, zorder=9, alpha=.09)
    dark_spine_kw = dict(color='k', lw=2.4, alpha=.63, zorder=9)
    #-------------------------------------------------------------------------------------------------
    h_ax.minorticks_on()
    h_ax.grid(True, which='major', axis='x', alpha=.09, **grid_kw)
    #-------------------------------------------------------------------------------------------------
    h_ax.tick_params('x', **maj_gr_kw)
    h_ax.tick_params('x', which='minor', **min_gr_kw)
    #-------------------------------------------------------------------------------------------------
    h_ax.tick_params('y', **maj_reg_kw)
    h_ax.tick_params('y', which='minor', left=False)
    #-------------------------------------------------------------------------------------------------
    if not box:
        [h_ax.spines[side].update(light_spine_kw) for side in light_spine_sides]
        [h_ax.spines[side].update(dark_spine_kw) for side in dark_spine_sides]
    if box:
        h_ax.spines['top'].update(light_spine_kw)
        h_ax.spines['right'].update(light_spine_kw)
        h_ax.spines['left'].update(dark_spine_kw)
        h_ax.spines['bottom'].update(dark_spine_kw)
```

In [19]:
```python
# Function for a Tick Params Settings for Axes with a Dual Numerical X Axis on the top
#####################################################################################################
def top_x_tick_params(t_ax):
    grid_kw = dict(color='dimgrey', linewidth=3, zorder=0)
    maj_gr_kw = dict(length=12, width=2.4, labelsize=25.5, labelcolor='k', color=(0,0,0,.63))
    min_gr_kw = dict(length=9, width=1.2, color=(0,0,0,.63))
    #-------------------------------------------------------------------------------------------------
    t_ax.minorticks_on()
    #-------------------------------------------------------------------------------------------------
    t_ax.tick_params('x', **maj_gr_kw)
    t_ax.tick_params('x', which='minor', **min_gr_kw)
    #-------------------------------------------------------------------------------------------------
    t_ax.tick_params('y', which='both', labelleft=False, left=False)
    #-------------------------------------------------------------------------------------------------
    [t_ax.spines[side].set_visible(False) for side in light_spine_sides]
    [t_ax.spines[side].set_visible(False) for side in dark_spine_sides]
```

In [20]:
```python
# Function for a Tick Params Settings for Axes with a Dual Non-Numerical Y Axis on the right
#####################################################################################################
def right_y_tick_params(r_ax):
    maj_reg_kw = dict(length=0, width=2.4, labelsize=24, labelcolor='k', color=(0,0,0,.63))
    #-------------------------------------------------------------------------------------------------
    r_ax.tick_params('y', **maj_reg_kw)
    r_ax.tick_params('y', which='minor', right=False)
    #-------------------------------------------------------------------------------------------------
    r_ax.tick_params('x', which='both', labelbottom=False, bottom=False)
    #-------------------------------------------------------------------------------------------------
    [r_ax.spines[side].set_visible(False) for side in light_spine_sides]
    [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]:
```python
# Function to format currency ticks on the X Axis
#######################################################################################################
def x_currency_formatter(x, pos):
    x_prime = x.copy()
    s_start = r'{\fontsize{24}{0}\selectfont{\scshape{\$}}}'
    #-----------------------------------------------------------------------------------------------
    if x>=1e9 or x<=-1e9:
        s_end = r'{\fontsize{22.5}{0}\selectfont{\scshape{B}}}'
        #-----------------------------------------------------------------------------------------------
        if x>=1e9: x = x*1e-9
        if x<=-1e9: x = x*-1e-9
    #-----------------------------------------------------------------------------------------------
    if 1e6<=x<1e9 or -1e6>=x>-1e9:
        s_end = r'{\fontsize{22.5}{0}\selectfont{\scshape{M}}}'
        #-----------------------------------------------------------------------------------------------
        if 1e6<=x<1e9: x = x*1e-6
        if -1e6>=x>-1e9: x = x*-1e-6
    #-----------------------------------------------------------------------------------------------
    if 1e3<=x<1e6 or -1e3>=x>-1e6:
        s_end = r'{\fontsize{22.5}{0}\selectfont{\scshape{K}}}'
        #-----------------------------------------------------------------------------------------------
        if 1e3<=x<1e6: x = x*1e-3
        if -1e3>=x>-1e6: x = x*-1e-3
    #-----------------------------------------------------------------------------------------------
    if round(x, 2) == int(x): s = s_start + r'$\mathbf{'+'{:.0f}'.format(x)+'}$'
    #-----------------------------------------------------------------------------------------------
    elif round(x, 2) == round(x, 1): s = s_start + r'$\mathbf{'+'{:.1f}'.format(x)+'}$'
    #-----------------------------------------------------------------------------------------------
    else:  s = s_start + r'$\mathbf{'+'{:.2f}'.format(x)+'}$'
    #-----------------------------------------------------------------------------------------------
    if x_prime<-1e3: s = r'$\mathbf{-}$'+ s + s_end
    if x_prime>1e3: s = s + s_end
    #-----------------------------------------------------------------------------------------------
    if x_prime==0: s = r'{$\mathbf{'+str(0)+'}$'
    #-----------------------------------------------------------------------------------------------
    return s
```

In [22]:
```python
# Function to format percentage ticks on an X Axis
#######################################################################################################
def x_percentage_formatter(x, pos):
    s_end = r'{\fontsize{24}{0}\selectfont{$\boldsymbol{\%}$}}'
    #-----------------------------------------------------------------------------------------------
    x_prime = x.copy()
    x = x*1e2
    #-----------------------------------------------------------------------------------------------
    if round(x, 2) == int(x): s = r'$\mathbf{'+'{:.0f}'.format(x)+'}$' + s_end
    #-----------------------------------------------------------------------------------------------
    elif round(x, 2) == round(x, 1): s = r'$\mathbf{'+'{:.1f}'.format(x)+'}$' + s_end
    #-----------------------------------------------------------------------------------------------
    else:  s = r'$\mathbf{'+'{:.2f}'.format(x)+'}$' + s_end
    #-----------------------------------------------------------------------------------------------
    if x_prime==0: s = r'{$\mathbf{'+str(0)+'}$'

    return s
```

In [23]:
```
# Function to format simple numerical tick labels in LaTeX format for use on ticks on the Y Axis
# where simple formatting wouldn't work
#########################################################################################################
def y_labels_to_norm_num(labels):
    tex_num_labels = []
    #-----------------------------------------------------------------------------------------------------
    for y in labels:
        s = r'{$\mathbf{'+str(int(y))+'}$'
        #-------------------------------------------------------------------------------------------------
        tex_num_labels.append(s)
    #-----------------------------------------------------------------------------------------------------
    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]:
```
# Function to find a small positive quantile to be passed into the max tick finder in an
# aggregate type plot
#########################################################################################################
def find_pos_quantile(ser):
    test_value = -1
    test_quant = .06
    last_neg = 0
    i = 0
    while test_value < 0:
        test_value = ser.quantile(test_quant)
        #-------------------------------------------------------------------------------------------------
        if test_value < 0: last_neg = test_quant
        #-------------------------------------------------------------------------------------------------
        if i < 3: test_quant = test_quant + .06
        elif 3 <= i < 6: test_quant = test_quant + .06*2
        else: test_quant = test_quant + .06*(i / 2)
        i += 1
    #-----------------------------------------------------------------------------------------------------
    return test_value, test_quant, last_neg
```

In [25]:
```
# Function to find a min for data ticks
#########################################################################################################
def find_min(new_min, min_step):
    min_found = False
    test_v = min_step
    i = 0
    #-----------------------------------------------------------------------------------------------------
    while not min_found:
        #-------------------------------------------------------------------------------------------------
        if new_min >= int(test_v):
            final_min = int(test_v)
            min_found = True
        #-------------------------------------------------------------------------------------------------
        if i < 3: test_v = test_v + min_step
        elif 3 <= i < 6: test_v = test_v + min_step*2
        else: test_v = test_v + min_step*(i - 3)
        i += 1
    #-----------------------------------------------------------------------------------------------------
    return final_min
```

In [26]:

```
# Function to find a max for data ticks
#########################################################################################################
def find_max(new_max, max_step):
    max_found = False
    test_v = max_step
    i = 0
    #-----------------------------------------------------------------------------------------------------
    while not max_found:
        #-------------------------------------------------------------------------------------------------
        if new_max > 1:
            if new_max <= int(test_v):
                final_max = int(test_v)
                max_found = True
        else:
            if new_max <= test_v:
                final_max = test_v
                max_found = True
        #-------------------------------------------------------------------------------------------------
        if i < 3: test_v = test_v + max_step
        elif 3 <= i < 6: test_v = test_v + max_step*2
        else: test_v = test_v + max_step*(i - 3)
        i += 1
    #-----------------------------------------------------------------------------------------------------
    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]:

```
# Plotting Function for all plots with identifying strings passed through
#########################################################################################################
def cat_plot(plot_df, viz_type, plot_type, box=None):

    # Starting variables & varaibles based on which num_type str was passed thru and if box=True
    #####################################################################################################
    plot_height = 9
    perc_plot = False
    tit_bullet = r'{\fontsize{30}{0}\selectfont{\textbf{ \textbullet\ }}}'
    #---------------------------------------------------------------------------------------------------
    if plot_type=='creative_type': main_cat_col = plot_type
    elif 'primary_name' in plot_df.columns: main_cat_col = 'primary_name'
    else: main_cat_col = 'genres'
    #---------------------------------------------------------------------------------------------------
    if plot_type in ['directors', 'writers', 'actors', 'actresses']: crew_plot = True
    else: crew_plot = False
    #---------------------------------------------------------------------------------------------------
    if viz_type == 'rating':
        rating_plot = True
        sub_cat_col = viz_type
        test_col_str = plot_df.columns[2]
        plot_len = len(plot_df)
        #-----------------------------------------------------------------------------------------------
        if 'tot' in test_col_str:
            cat_freq = []
            for g in plot_df[main_cat_col].unique():
                g_freq = plot_df.set_index(main_cat_col).loc[g].num_titles
                #-------------------------------------------------------------------------------------
                if type(g_freq) in [np.int64, np.float64]: cat_freq.append(g_freq)
                else: cat_freq.append(g_freq[0])
        #-----------------------------------------------------------------------------------------------
        else: cat_freq = plot_df['num_titles']
    #---------------------------------------------------------------------------------------------------
    else:
        rating_plot = False
        test_col_str = plot_df.columns[-1]
        plot_len = len(plot_df[main_cat_col].unique())
        #-----------------------------------------------------------------------------------------------
        if not box: cat_freq = plot_df['num_titles']
    #---------------------------------------------------------------------------------------------------
    if 'gross' in test_col_str: lab_add = 'Worldwide Gross'
    if 'profits' in test_col_str: lab_add = 'Worldwide Profits'
    #---------------------------------------------------------------------------------------------------
    base_x_formatter = x_currency_formatter

    # Setting important variables based on if 'tot', 'avg' in last column or if box=True
    #####################################################################################################
    if not box and 'tot' in test_col_str:
        perc_plot = True
        prime_lab_p1 = 'Total '
        t_lab_p1 = 'Percentage of Total '
        t_lab = t_lab_p1 + lab_add
        #-------------------------------------------------------------------------------------------
        plot_col = plot_df.columns[-2]
        perc_col = plot_df.columns[-1]
        #-------------------------------------------------------------------------------------------
        cat_values = plot_df[plot_col]
        cat_percs = plot_df[perc_col]
        #-------------------------------------------------------------------------------------------
        value_step, v_q_used, v_last_neg_q = find_pos_quantile(cat_values)
        perc_step, p_q_used, p_last_neg_q = find_pos_quantile(cat_percs)
        #-------------------------------------------------------------------------------------------
        if p_q_used != v_q_used:
            if p_q_used > v_q_used: value_step = cat_values.quantile(p_q_used)
            else: perc_step = cat_percs.quantile(v_q_used)
        #-------------------------------------------------------------------------------------------
        if p_last_neg_q != v_last_neg_q and box:
            if p_last_neg_q < v_last_neg_q: neg_value_step = cat_values.quantile(p_last_neg_q)
            else: neg_perc_step = cat_percs.quantile(v_last_neg_q)
        #-------------------------------------------------------------------------------------------
```

```
        if rating_plot:
            cat_values = cat_values.unique()
            cat_percs = cat_percs.unique()
    #-----------------------------------------------------------------------------------------------
    if box or 'avg' in test_col_str:
        prime_lab_p1 = 'Average '
        #-------------------------------------------------------------------------------------------
        if box:
            box_add = 'Boxplots of the '
            if 'tot' in box: prime_lab_p1 = 'Total '
        #-------------------------------------------------------------------------------------------
        plot_col = plot_df.columns[-1]
        cat_values = plot_df[plot_col]
        #-------------------------------------------------------------------------------------------
        if not rating_plot:
            value_step, v_q_used, v_last_neg_q = find_pos_quantile(cat_values)
        #-------------------------------------------------------------------------------------------
        else:
            value_step, v_q_used, v_last_neg_q = find_pos_quantile(plot_df[test_col_str])
        #-------------------------------------------------------------------------------------------
        neg_value_step = cat_values.quantile(v_last_neg_q)
    #-----------------------------------------------------------------------------------------------
    if not box: prime_lab = prime_lab_p1 + lab_add
    if box:
        top_lab = prime_lab_p1 + lab_add
        prime_lab = box_add + lab_add
        #-------------------------------------------------------------------------------------------
        rank_add = 'Average '+ lab_add

    # Setting important variables based on plot type and if box=True
    ###################################################################################################
    if crew_plot:
        y_lab = plot_type.title()
        y_labels = plot_df[main_cat_col]
        tex_y_labels = [r'\textbf{\scshape{'+ lab +'}}' for lab in y_labels]
        #-------------------------------------------------------------------------------------------
        if plot_type == 'actresses': r_lab_insert = y_lab[:-2]
        else: r_lab_insert = y_lab[:-1]
    #-----------------------------------------------------------------------------------------------
    if not crew_plot: y_labels = plot_df[main_cat_col].unique()
    #-----------------------------------------------------------------------------------------------
    if rating_plot and 'avg' in test_col_str: y_labels = plot_df[main_cat_col]

    # Setting important variables for Genre Combination plots based on plot type
    ###################################################################################################
    if viz_type == 'combos':
        r_lab_1 = 'Number of Titles with this Genre'
        #-------------------------------------------------------------------------------------------
        if crew_plot:
            r_lab_2 = 'Combination from each '+ r_lab_insert
            #---------------------------------------------------------------------------------------
            title_add = y_lab +' of' if plot_type in ['directors','writers'] else y_lab +' in'
            #---------------------------------------------------------------------------------------
            c_tit_p1 = r'{\fontsize{30}{0}\selectfont{\textsl{Top '+ str(plot_len) +\
                       r' '+ title_add +r' Movies with the Genre Combination Above}}}'+'\n'
            c_tit_p2 = r'{\fontsize{27}{0}\selectfont{\textsl{Ranked by '+ prime_lab +'}}}'
        #-------------------------------------------------------------------------------------------
        if plot_type == 'genres':
            y_lab = 'Unique Genre Combinations'
            #---------------------------------------------------------------------------------------
            r_lab_1 = 'Number of Titles from each'
            r_lab_2 = y_lab[:-1]
            #---------------------------------------------------------------------------------------
            c_tit_p1 = r'{\fontsize{30}{0}\selectfont{\textsl{Top '+ str(plot_len) +\
                       r' '+y_lab+'}}}'+ tit_bullet
            #---------------------------------------------------------------------------------------
            c_tit_p2 = r'{\fontsize{30}{0}\selectfont{\textsl{Ranked by '+ prime_lab +'}}}'
            #---------------------------------------------------------------------------------------
            tex_y_labels = \
            [r'\textbf{\scshape{'+', '.join(lab.split(','))+'}}' for lab in y_labels]
        #-------------------------------------------------------------------------------------------
```

```
        if box:
            prime_lab = box_add + lab_add
            #-----------------------------------------------------------------------------------------
            c_tit_p2 = r'{\fontsize{30}{0}\selectfont{\textsl{Based on '+ top_lab +'}}}'
            #-----------------------------------------------------------------------------------------
            c_tit_p3 = \
            r'{\fontsize{28.5}{0}\selectfont{\textsl{Ranked by Average '+ lab_add +'}}}'+'\n'+\
            r'{\fontsize{27}{0}\selectfont{\textsl{'+ prime_lab +' of Movies}}}'+\
            r'{\fontsize{27}{0}\selectfont{\textsl{ with these Unique Genre Combinations}}}'
        #-----------------------------------------------------------------------------------------
        if not box: tit_p3 = c_tit_p1 + c_tit_p2
        if box: tit_p3 = c_tit_p1 + c_tit_p2 +'\n'+ c_tit_p3

    # Setting important variables for individual Genre plots based on year range and plot type
    ###############################################################################################
    if viz_type == 'individs' or viz_type == 'genres':
        if viz_type == 'individs': r_lab_1 = 'Number of Titles that Included'
        if viz_type == 'genres': r_lab_1 = 'Number of Titles with'
        #-----------------------------------------------------------------------------------------
        if crew_plot:
            r_lab_2 = 'This Genre from each '+ r_lab_insert
            #-----------------------------------------------------------------------------------------
            title_add = y_lab +' of' if plot_type in ['directors','writers'] else y_lab +' in'
            #-----------------------------------------------------------------------------------------
            if viz_type == 'individs':
                i_tit_p1 = r'{\fontsize{30}{0}\selectfont{\textsl{'+ title_add +\
                           ' Movies that Included the Individual Genre Above}}}'
            #-----------------------------------------------------------------------------------------
            if viz_type == 'genres':
                i_tit_p1 = r'{\fontsize{30}{0}\selectfont{\textsl{'+ title_add +\
                           ' Movies with the Genre Above}}}'
            #-----------------------------------------------------------------------------------------
            i_tit_p2 = r'{\fontsize{27}{0}\selectfont{\textsl{Ranked by '+ prime_lab +'}}}'
        #-----------------------------------------------------------------------------------------
        if plot_type == 'genres':
            if viz_type == 'individs': y_lab = 'Individual Genres'
            if viz_type == 'genres': y_lab = 'Genres'
            #-----------------------------------------------------------------------------------------
            i_p1_insert = 'each '+ y_lab[:-1]
            #-----------------------------------------------------------------------------------------
            if not box:
                if viz_type == 'individs': r_lab_2 = 'Each '+ y_lab[:-1]
                if viz_type == 'genres': r_lab_2 = 'each '+ y_lab[:-1]
                tit_start = r'{\fontsize{30}{0}\selectfont{\textsl{'
            else:
                tit_start = r'{\fontsize{27}{0}\selectfont{\textsl{'
                i_tit_p2 = r'{\fontsize{27}{0}\selectfont{\textsl{Ranked by '+ rank_add +'}}}'
            #-----------------------------------------------------------------------------------------
            if viz_type == 'individs':
                i_tit_p1 = tit_start + prime_lab +' of Movies that Included '+ i_p1_insert +'}}}'
            if viz_type == 'genres':
                i_tit_p1 = tit_start + prime_lab +' of Movies with '+ i_p1_insert +'}}}'
            #-----------------------------------------------------------------------------------------
            tex_y_labels = [r'\textbf{\scshape{'+ lab +'}}' for lab in y_labels]
        #-----------------------------------------------------------------------------------------
        if plot_type != 'genres': tit_p3 = i_tit_p1 +'\n'+ i_tit_p2
        else: tit_p3 = i_tit_p1

    # Setting important variables for individual Genre plots based on year range and plot type
    ###############################################################################################
    if viz_type == 'creative_type':
        r_lab_1 = 'Number of Titles with '
        #-----------------------------------------------------------------------------------------
        if crew_plot:
            r_lab_2 = 'Creative Type from each '+ r_lab_insert
            #-----------------------------------------------------------------------------------------
            title_add = y_lab +' of' if plot_type in ['directors','writers'] else y_lab +' in'
            #-----------------------------------------------------------------------------------------
            i_tit_p1 = r'{\fontsize{30}{0}\selectfont{\textsl{'+ title_add +\
                       ' Movies with the Creative Type Above}}}'
            i_tit_p2 = r'{\fontsize{27}{0}\selectfont{\textsl{Ranked by '+ prime_lab +'}}}'
```

```
        #-------------------------------------------------------------------------------------
        if plot_type == 'creative_type':
            y_lab = 'Creative Types'
            #-------------------------------------------------------------------------------------
            i_p1_insert = 'each '+ y_lab[:-1]
            #-------------------------------------------------------------------------------------
            if not box:
                r_lab_2 = 'Each '+ y_lab[:-1]
                tit_start = r'{\fontsize{30}{0}\selectfont{\textsl{'
            else:
                tit_start = r'{\fontsize{27}{0}\selectfont{\textsl{'
                i_tit_p2 = r'{\fontsize{27}{0}\selectfont{\textsl{Ranked by '+ rank_add +'}}}'
            #-------------------------------------------------------------------------------------
            i_tit_p1 = tit_start + prime_lab +' of Movies with '+ i_p1_insert +'}}}'
            #-------------------------------------------------------------------------------------
            tex_y_labels = [r'\textbf{\scshape{'+ lab +'}}' for lab in y_labels]
        #-------------------------------------------------------------------------------------
        if plot_type != 'creative_type': tit_p3 = i_tit_p1 +'\n'+ i_tit_p2
        else: tit_p3 = i_tit_p1

    # Setting important variables for individual Genre plots based on year range and plot type
    ##########################################################################################
    if rating_plot:
        r_lab_1 = 'Number of Titles with'
        #-------------------------------------------------------------------------------------
        spec_lab = 'MPA Ratings'
        #-------------------------------------------------------------------------------------
        if plot_type=='combos':
            y_lab = 'Unique Genre Combinations'
            tit_p1_add = 'with'
            # cat_tit_add = 'Each Genre Combination with '
            cat_tit_add = ' each Genre Combination with '
            tex_y_labels = \
            [r'\textbf{\scshape{'+', '.join(lab.split(','))+'}}' for lab in y_labels]
        #-------------------------------------------------------------------------------------
        if plot_type=='individs':
            y_lab = 'Individual Genres'
            tit_p1_add = 'that'
            # cat_tit_add = 'Included each Individual Genre with '
            cat_tit_add = ' included each Individual Genre with '
            tex_y_labels = [r'\textbf{\scshape{'+ lab +'}}' for lab in y_labels]
        #-------------------------------------------------------------------------------------
        if plot_type=='genres':
            y_lab = 'Genres'
            tit_p1_add = 'with'
            # cat_tit_add = 'Each Genre with '
            cat_tit_add = ' each Genre with '
            tex_y_labels = [r'\textbf{\scshape{'+ lab +'}}' for lab in y_labels]
        #-------------------------------------------------------------------------------------
        if plot_type=='creative_type':
            y_lab = 'Creative Types'
            tit_p1_add = 'with'
            # cat_tit_add = 'Each Creative Type with '
            cat_tit_add = ' each Creative Type with '
            tex_y_labels = [r'\textbf{\scshape{'+ lab +'}}' for lab in y_labels]
        #-------------------------------------------------------------------------------------
        if 'avg' in test_col_str: plot_height = 18
        #-------------------------------------------------------------------------------------
        i_p2_insert = 'each '+ spec_lab[:-1]
        #-------------------------------------------------------------------------------------
#         if not box:
        if viz_type!='genres' or (rating_plot and 'tot' in test_col_str and plot_type!='genres'):
            r_lab_2 = 'Each '+ y_lab[:-1]

        if viz_type=='genres' or (rating_plot and 'avg' in test_col_str) or \
           (rating_plot and plot_type=='genres'):
            r_lab_2 = 'each '+ y_lab[:-1]



        tit_start = r'{\fontsize{30}{0}\selectfont{\textsl{'
```

```
#         else: tit_start = r'{\fontsize{27}{0}\selectfont{\textsl{'
        #-------------------------------------------------------------------------------------------
        i_tit_p1 = tit_start + prime_lab +' of Movies '+ tit_p1_add +'}}}'
        i_tit_p2 = tit_start + cat_tit_add + i_p2_insert +'}}}'
        #-------------------------------------------------------------------------------------------
#         tit_p3 = i_tit_p1 +'\n'+ i_tit_p2
        tit_p3 = i_tit_p1 + i_tit_p2

    # Setting other variables necesary for plotting
    ###############################################################################################
    num_bars = 1
    bar_height = 3
    group_space = 4.5
    #-------------------------------------------------------------------------------------------
    y_locs = range(plot_len)
    #-------------------------------------------------------------------------------------------
    if (rating_plot and 'tot' in test_col_str) or box:
        y_locs = range(len(plot_df[main_cat_col].unique()))

    # Locations for bars or boxes
    ###############################################################################################
    y_ticks = [num_bars*group_space*y + bar_height*(1.5) for y in y_locs]
    #-------------------------------------------------------------------------------------------
    split_dist = ((y_ticks[1] - y_ticks[0]) / 2)
    split_locs = [(num_bars*group_space)*y + bar_height*(1.5) + split_dist for y in y_locs[0:-1]]

    # Setting up the figure and the axes
    ###############################################################################################
    fig = plt.figure(figsize=(18, plot_height), dpi=300)
    gs = fig.add_gridspec(1)
    #-------------------------------------------------------------------------------------------
    ax = gs.subplots()
    #-------------------------------------------------------------------------------------------
    if perc_plot: t_ax = ax.twiny()
    if not box: r_ax = ax.twinx()
    #-------------------------------------------------------------------------------------------
    renderer = fig.canvas.get_renderer()

    # Plotting the data based on if box=True
    ###############################################################################################
    if not box:
        bar_colors = get_lighter_color('forestgreen', .45)
        #-------------------------------------------------------------------------------------------
        bars_kw = dict(color=bar_colors, ec='k', lw='.54', zorder=3)
        #-------------------------------------------------------------------------------------------
        if not rating_plot or (rating_plot and 'tot' in test_col_str):
            main_bars = ax.barh(y_ticks, cat_values, bar_height, **bars_kw)
    #-------------------------------------------------------------------------------------------
    if rating_plot and not box:
        rats_used_dict = {}
        #-------------------------------------------------------------------------------------------
        if 'tot' in test_col_str:
            for m_cat, y_t, m_bar in zip(plot_df[main_cat_col].unique(), y_ticks, main_bars):
                sub_cats = plot_df.set_index(main_cat_col).loc[m_cat, 'rating']
                sub_cat_values = plot_df.set_index(main_cat_col).loc[m_cat, test_col_str]
                #-------------------------------------------------------------------------------------------
                bar_total = m_bar.get_width()
                #-------------------------------------------------------------------------------------------
                if type(sub_cat_values) != pd.Series:
                    bar_color = get_lighter_color(ratings_colors_dict[sub_cats], .45)
                    #-------------------------------------------------------------------------------------------
                    if sub_cats not in rats_used_dict.keys():
                        rats_used_dict[sub_cats] = bar_color
                    #-------------------------------------------------------------------------------------------
                    bars_kw = dict(color=bar_color, ec='k', lw='.54', zorder=3)
                    #-------------------------------------------------------------------------------------------
                    ax.barh(y_t, sub_cat_values, bar_height, **bars_kw)
                    #-------------------------------------------------------------------------------------------
                    if sub_cat_values < bar_total:
                        unk_label = 'Unknown'
                        unk_color = get_lighter_color('forestgreen', .45)
```

```
                    #-------------------------------------------------------------------------
                    if ink_label not in rats_used_dict.keys():
                        rats_used_dict[unk_label] = unk_color
                #-----------------------------------------------------------------------------
                else:
                    old_val = 0
                    for s_cat, val in zip(sub_cats, sub_cat_values):
                        bar_color = get_lighter_color(ratings_colors_dict[s_cat], .45)
                        #-------------------------------------------------------------------------
                        if s_cat not in rats_used_dict.keys():
                            rats_used_dict[s_cat] = bar_color
                        #-------------------------------------------------------------------------
                        bars_kw = dict(color=bar_color, ec='k', lw='.54', zorder=3)
                        #-------------------------------------------------------------------------
                        ax.barh(y_t, val, bar_height, left=old_val, **bars_kw)
                        #-------------------------------------------------------------------------
                        old_val = old_val + val
                    #-------------------------------------------------------------------------
                    if old_val < bar_total:
                        unk_label = 'Unknown'
                        unk_color = get_lighter_color('forestgreen', .45)
                        #-------------------------------------------------------------------------
                        if unk_label not in rats_used_dict.keys():
                            rats_used_dict[unk_label] = unk_color
        #-----------------------------------------------------------------------------------
        else:
            main_series = plot_df[main_cat_col]
            sub_series = plot_df['rating']
            plot_series = plot_df[test_col_str]
            #-------------------------------------------------------------------------------
            for g, sub_cat, val, y_t in zip(main_series, sub_series, plot_series, y_ticks):
                bar_color = get_lighter_color(ratings_colors_dict[sub_cat], .45)
                #---------------------------------------------------------------------------
                if sub_cat not in rats_used_dict.keys(): rats_used_dict[sub_cat] = bar_color
                #---------------------------------------------------------------------------
                bars_kw = dict(color=bar_color, ec='k', lw='.54', zorder=3)
                #---------------------------------------------------------------------------
                ax.barh(y_t, val, bar_height, **bars_kw)
        #-------------------------------------------------------------------------
        rat_leg = []
        #-------------------------------------------------------------------------------
        for rat, rat_c in rats_used_dict.items():
            sub_cat_patch = \
            mpatches.Patch(label=r'\textbf{\scshape{'+ rat +'}}', color=rat_c)
            #---------------------------------------------------------------------
            rat_leg.append(sub_cat_patch)
    #-----------------------------------------------------------------------------------
    if box:
        c_patch = get_lighter_color('forestgreen', .45)
        c_line = get_lighter_color('forestgreen', .36)
        #-------------------------------------------------------------------------------
        for cat, y_t in zip(plot_df[main_cat_col].unique(), y_ticks):
            #---------------------------------------------------------------------------
            box_kw = dict(fc=c_patch, lw=.9, ec='k')
            means_kw = dict(color='k', ls=(0, (1, 1)), lw=6.3, zorder=6)
            meds_kw = dict(color='k', lw=4.5, zorder=99)
            caps_kw = dict(c=c_line, lw=4.2, solid_capstyle='round', zorder=6)
            whis_kw = dict(c=c_line, lw=4.2, solid_capstyle='round', zorder=6)
            fliers_kw = dict(mfc=c_line, ms=15, mec='k', mew=.3, marker='X')
            #---------------------------------------------------------------------------
            b_kw = dict(widths=bar_height, boxprops=box_kw, medianprops=meds_kw, meanline=True,
                        showmeans=True, meanprops=means_kw, flierprops=fliers_kw,
                        whiskerprops=whis_kw, capprops=caps_kw, patch_artist=True, vert=False)
            #---------------------------------------------------------------------------
            cat_df = plot_df.set_index(main_cat_col).loc[cat]
            #---------------------------------------------------------------------------
            box_dict = ax.boxplot(cat_df[plot_col], positions=[y_t], **b_kw)
            #---------------------------------------------------------------------------
            mean_lines = \
            mlines.Line2D([], [], label=r'\textbf{\scshape{Mean Lines}}', **means_kw)
            #---------------------------------------------------------------------------
```

```
            median_lines = \
            mlines.Line2D([], [], label=r'\textbf{\scshape{Median Lines}}', **meds_kw)

    # AX - X Ticks, TIck Labels, and Bounds
    ###########################################################################################################
    if not rating_plot or 'tot' in test_col_str:
        x_max = cat_values.max()
    else: x_max = plot_df[test_col_str].max()
    #-----------------------------------------------------------------------------------------------------------
    final_max = find_max(x_max, value_step)
    #-----------------------------------------------------------------------------------------------------------
    x_tick_max = final_max
    #-----------------------------------------------------------------------------------------------------------
    if not box:
        x_step = final_max/6
        x_ticks = np.arange(0, x_tick_max +1, x_step)
    #-----------------------------------------------------------------------------------------------------------
    if box:
        x_min = cat_values.min()
        #-------------------------------------------------------------------------------------------------------
        final_min = find_min(x_min, neg_value_step)
        #-------------------------------------------------------------------------------------------------------
        if final_min < 0:
            zero_kw = dict(color='k', lw=3.9, ls='-.')
            #---------------------------------------------------------------------------------------------------
            zero_line = \
            ax.axvline(0, label=r'\textbf{\scshape{Zero Profits Line}}', **zero_kw)
        #-------------------------------------------------------------------------------------------------------
        x_tick_min = final_min
        x_step = (final_max - final_min) / 6
        x_ticks = np.arange(final_min, x_tick_max+1, x_step)
    #-----------------------------------------------------------------------------------------------------------
    ax.set_xticks(x_ticks)
    #-----------------------------------------------------------------------------------------------------------
    ax.xaxis.set_major_formatter(base_x_formatter)
    #-----------------------------------------------------------------------------------------------------------
    ax.set_xbound(x_ticks[0], x_ticks[-1])

    # T_AX - X Ticks, TIck Labels, and Bounds
    ###########################################################################################################
    if perc_plot:
        x_max = cat_percs.max()
        #-------------------------------------------------------------------------------------------------------
        final_max = find_max(x_max, perc_step)
        #-------------------------------------------------------------------------------------------------------
        x_tick_max = final_max
        x_step = final_max/6
        x_ticks = np.arange(0, x_tick_max+1, x_step)
        #-------------------------------------------------------------------------------------------------------
        t_ax.set_xticks(x_ticks)
        t_ax.xaxis.set_major_formatter(x_percentage_formatter)
        #-------------------------------------------------------------------------------------------------------
        t_ax.set_xbound(0, x_tick_max)

    # AX & R_AX - Y Tick, Tick Labels and Bounds
    ###########################################################################################################
    ax.set_yticks(y_ticks)
    ax.set_yticklabels(tex_y_labels, va='center')
    ax.set_ybound((y_ticks[0] - split_dist, y_ticks[-1] + split_dist))
    ax.invert_yaxis()
    #-----------------------------------------------------------------------------------------------------------
    if not box:
        freq_labels = y_labels_to_norm_num(cat_freq)
        #-------------------------------------------------------------------------------------------------------
        r_ax.set_yticks(y_ticks)
        r_ax.set_yticklabels(freq_labels, va='center')
        r_ax.set_ybound((y_ticks[0] - split_dist, y_ticks[-1] + split_dist))
        r_ax.invert_yaxis()

    # Tick Params Functions
    ###########################################################################################################
```

```
    if not box: h_tick_params(ax)
    if box: h_tick_params(ax, True)
    if perc_plot: top_x_tick_params(t_ax)
    if not box: right_y_tick_params(r_ax)

    # Axis Labels
    ##########################################################################################################
    x_lab_kw= dict(size=28.5, labelpad=18)
    y_lab_kw= dict(size=30, labelpad=27, linespacing=1.5)
    #---------------------------------------------------------------------------------------------------------
    if not box: ax.set_xlabel(r'\textbf{\textsl{'+ prime_lab +'}}', **x_lab_kw)
    if box: ax.set_xlabel(r'\textbf{\textsl{'+ lab_add +'}}', **x_lab_kw)
    #---------------------------------------------------------------------------------------------------------
    if perc_plot:
        t_ax.set_xlabel(r'\textbf{\textsl{'+ t_lab +'}}', **x_lab_kw)
    #---------------------------------------------------------------------------------------------------------
    ax.set_ylabel(r'\textbf{\textsl{'+ y_lab +'}}', **y_lab_kw)
    if not box:
        r_ax.set_ylabel(r'\textbf{\textsl{'+ r_lab_1 +'}}'+'\n'+\
                        r'\textbf{\textsl{'+ r_lab_2 +'}}', va='bottom',  rotation=270,
                        **y_lab_kw)
        #-----------------------------------------------------------------------------------------------------
        if (viz_type=='genres' and plot_type=='genres') or \
           (rating_plot and 'avg' in test_col_str) or \
           (rating_plot and 'tot' in test_col_str and plot_type=='genres'):
            r_ax.set_ylabel(r'\textbf{\textsl{'+ r_lab_1 +' '+ r_lab_2 +'}}', va='bottom',
                            rotation=270, **y_lab_kw)

    # Title
    ##########################################################################################################
    if perc_plot: tit_kw = dict(pad=36, linespacing=1.8)
    else: tit_kw = dict(pad=18, linespacing=1.8)
    #---------------------------------------------------------------------------------------------------------
    m_tit_p1 = r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{ Categorical Data}}}}'
    #---------------------------------------------------------------------------------------------------------
    finan_data_source = \
    r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{The Numbers Financial Data}}}}'
    #---------------------------------------------------------------------------------------------------------
    if viz_type in ['combos', 'individs'] or plot_type in ['combos', 'individs']:
        cat_source = r'{\fontsize{31.5}{0}\selectfont{\textbf{IMDb}}}'
        year_tit = r'2010 \boldsymbol{\rightarrow} 2019'
    #---------------------------------------------------------------------------------------------------------
    if viz_type in ['genres', 'creative_type'] or \
       (rating_plot and plot_type in ['genres', 'creative_type']):
        cat_source = r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{OpusData}}}}'
        year_tit = r'2010 \boldsymbol{\rightarrow} 2018'
    #---------------------------------------------------------------------------------------------------------
    if rating_plot:
        if 'tot' in test_col_str: year_tit = r'2010 \boldsymbol{\rightarrow} 2019'
        if 'avg' in test_col_str: year_tit = r'2010 \boldsymbol{\rightarrow} 2018'
        #-----------------------------------------------------------------------------------------------------
        rat_cat_source = r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{OpusData}}}}'
        m_tit_rat = r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{ Ratings Data}}}}'
        #-----------------------------------------------------------------------------------------------------
        tit_p1 = cat_source + m_tit_p1 +'\n'+ rat_cat_source + m_tit_rat + tit_bullet + \
                 finan_data_source
    #---------------------------------------------------------------------------------------------------------
    if crew_plot:
        crew_source = r'{\fontsize{31.5}{0}\selectfont{\textbf{IMDb}}}'
        m_tit_crew = r'{\fontsize{31.5}{0}\selectfont{\textbf{\scshape{ Crew Data}}}}'
        #-----------------------------------------------------------------------------------------------------
        tit_p1 = cat_source + m_tit_p1 +'\n'+ crew_source + m_tit_crew + tit_bullet + \
                 finan_data_source
    #---------------------------------------------------------------------------------------------------------
    if not rating_plot and not crew_plot:
        tit_p1 = cat_source + m_tit_p1 + tit_bullet + finan_data_source
    #---------------------------------------------------------------------------------------------------------
    tit_p2 = r'{\fontsize{33}{0}\selectfont{$\mathbf{'+ year_tit +'}$}}'
    #---------------------------------------------------------------------------------------------------------
    if not crew_plot: ax.set_title(tit_p1 + '\n' + tit_p2 + '\n' + tit_p3, **tit_kw)
    #---------------------------------------------------------------------------------------------------------
```

```
    if crew_plot:
        if viz_type=='combos':
            cat_tit = r'{\fontsize{30}{0}\selectfont{\textbf{\scshape{'+ \
                      ', '.join(plot_df.index[0].split(',')) +r'}}}}'
        #-------------------------------------------------------------------------------------------
        else:
            cat_tit = r'{\fontsize{30}{0}\selectfont{\textbf{\scshape{'+ plot_df.index[0] +'}}}}'
        #-------------------------------------------------------------------------------------------
        ax.set_title(tit_p1 + '\n' + tit_p2 + '\n' + cat_tit + '\n' + tit_p3, **tit_kw)

    # Legend
    ###############################################################################################
#     if viz_type=='rating': leg_loc = 'best'
#     else: leg_loc = 'lower right'
    leg_loc = 'best'
    #-------------------------------------------------------------------------------------------
    leg_kw = dict(loc=leg_loc, fontsize=21, handlelength=3)
    #-------------------------------------------------------------------------------------------
    frame_kw = dict(lw=.9, ec='k', alpha=1)
    #-------------------------------------------------------------------------------------------
    if box:
        if final_min >= 0: leg = ax.legend(handles=[mean_lines, median_lines], **leg_kw)
        else: leg = ax.legend(handles=[zero_line, mean_lines, median_lines], **leg_kw)
        #-------------------------------------------------------------------------------------------
        plt.setp(leg.get_frame(), **frame_kw)
        plt.setp(leg.get_texts(), ma='center')
    #-------------------------------------------------------------------------------------------
    if viz_type=='rating':
        leg = ax.legend(handles=rat_leg, **leg_kw)
        #-------------------------------------------------------------------------------------------
        plt.setp(leg.get_frame(), **frame_kw)
        plt.setp(leg.get_texts(), ma='center')

    # Adding lines between each genre's name (lines between y tick labels)
    ###############################################################################################
    c_tick_kw = dict(clip_on=False, color='k', lw=1.5, alpha=.81, zorder=9)
    #-------------------------------------------------------------------------------------------
    top_spl = y_ticks[-1] + split_dist
    bot_spl = y_ticks[0] - split_dist
    #-------------------------------------------------------------------------------------------
    max_label_neg_x = 0
    max_label_pos_x = 0
    #-------------------------------------------------------------------------------------------
    ax_inv = ax.transAxes.inverted()
    #-------------------------------------------------------------------------------------------
    if not box:
        r_ax_inv = r_ax.transAxes.inverted()
        #-------------------------------------------------------------------------------------------
        for label, r_label in zip(ax.get_ymajorticklabels(), r_ax.get_ymajorticklabels()):
            lab_xymin = label.get_tightbbox(renderer).min
            r_lab_xymax = r_label.get_tightbbox(renderer).max
            #-------------------------------------------------------------------------------------------
            lab_f_bbox_xymin = ax_inv.transform(lab_xymin)
            r_lab_f_bbox_xymax = r_ax_inv.transform(r_lab_xymax)
            #-------------------------------------------------------------------------------------------
            if lab_f_bbox_xymin[0] <= max_label_neg_x: max_label_neg_x = lab_f_bbox_xymin[0]
            #-------------------------------------------------------------------------------------------
            if r_lab_f_bbox_xymax[0] >= max_label_pos_x: max_label_pos_x = r_lab_f_bbox_xymax[0]
        #-------------------------------------------------------------------------------------------
        pos_kw = dict(x=((max_label_pos_x - 1) / 2) + 1.006, ha='center')
        [r_label.update(pos_kw) for r_label in r_ax.get_ymajorticklabels()]
    #-------------------------------------------------------------------------------------------
    else:
        for label in ax.get_ymajorticklabels():
            lab_xymin = label.get_tightbbox(renderer).min
            #-------------------------------------------------------------------------------------------
            lab_f_bbox_xymin = ax_inv.transform(lab_xymin)
            #-------------------------------------------------------------------------------------------
            if lab_f_bbox_xymin[0] <= max_label_neg_x: max_label_neg_x = lab_f_bbox_xymin[0]
    #-------------------------------------------------------------------------------------------
    neg_kw = dict(x=(max_label_neg_x/2) - .0045, ha='center')
```

```
    [label.update(neg_kw) for label in ax.get_ymajorticklabels()]
    #-----------------------------------------------------------------------------------------
    if rating_plot and 'avg' in test_col_str:
        old_label_text = ax.get_ymajorticklabels()[0].get_text()
        label_size = ax.get_ymajorticklabels()[0].get_fontsize()
        #-------------------------------------------------------------------------------------
        same_label_group = [ax.get_ymajorticklabels()[0]]
        #-------------------------------------------------------------------------------------
        for s_loc, label in zip(split_locs, ax.get_ymajorticklabels()[1:]):
            new_label_text = label.get_text()
            #---------------------------------------------------------------------------------
            if new_label_text == old_label_text and label == ax.get_ymajorticklabels()[-1]:
                last_labels_same = True
                same_label_group.append(label)
            else: last_labels_same = False
            #---------------------------------------------------------------------------------
            if new_label_text != old_label_text or last_labels_same:
                if len(same_label_group) > 1:
                    label_x_pos = ax_inv.transform(same_label_group[0].get_tightbbox(renderer))
                    #-------------------------------------------------------------------------
                    lab_x_min = label_x_pos[0][0]
                    lab_x_max = label_x_pos[1][0]
                    #-------------------------------------------------------------------------
                    first_label_y_min = \
                    ax_inv.transform(same_label_group[0].get_tightbbox(renderer).min)[1]
                    #-------------------------------------------------------------------------
                    last_label_y_max = \
                    ax_inv.transform(same_label_group[-1].get_tightbbox(renderer).max)[1]
                    #-------------------------------------------------------------------------
                    mid_x = (lab_x_min  +  lab_x_max) / 2
                    mid_y = ((last_label_y_max  -  first_label_y_min) / 2) + first_label_y_min
                    #-------------------------------------------------------------------------
                    [label.set_visible(False) for label in same_label_group]
                    #-------------------------------------------------------------------------
                    ax.annotate(old_label_text, (mid_x, mid_y), xycoords='axes fraction',
                                size=label_size, va='center', ha='center')
                #-----------------------------------------------------------------------------
                old_label_text = new_label_text
                same_label_group = [label]
                #-----------------------------------------------------------------------------
                if not last_labels_same:
                    ax.axhline(s_loc, 0, max_label_neg_x - .015, **c_tick_kw)
            #---------------------------------------------------------------------------------
            else: same_label_group.append(label)
            #---------------------------------------------------------------------------------
            ax.axhline(s_loc, 1, max_label_pos_x + .015, **c_tick_kw)
    else:
        for s_loc in split_locs:
            ax.axhline(s_loc, 0, max_label_neg_x - .015, **c_tick_kw)
            #-----------------------------------------------------------------------------
            if not box: ax.axhline(s_loc, 1, max_label_pos_x + .015, **c_tick_kw)
    #-----------------------------------------------------------------------------------------
    if not any([label.get_visible() for label in ax.get_ymajorticklabels()]):
        y_label = ax.get_ylabel()
        #-------------------------------------------------------------------------------------
        ax.set_ylabel('', visible=False)
        #-------------------------------------------------------------------------------------
        ax.annotate(y_label, (max_label_neg_x - .036, .5), xycoords='axes fraction', size=30,
                    va='bottom', ha='center', rotation=90, rotation_mode='anchor')
    #-----------------------------------------------------------------------------------------
    bot_ax_line = ax.axhline(bot_spl, 0, max_label_neg_x - .015, **c_tick_kw)
    if not box: ax.axhline(bot_spl, 1, max_label_pos_x + .015, **c_tick_kw)
    #-----------------------------------------------------------------------------------------
    top_ax_line = ax.axhline(top_spl, 0, max_label_neg_x - .015, **c_tick_kw)
    if not box: top_ax_line = ax.axhline(top_spl, 1, max_label_pos_x + .015, **c_tick_kw)

    # Adding other lines to enhance appearance
    #########################################################################################
    div_kw = dict(color='k', lw=1.95)#, alpha=.63)
    #-----------------------------------------------------------------------------------------
    fig_inv = fig.transFigure.inverted()
```

```
    #-------------------------------------------------------------------------------------------
    if perc_plot:
        ax_xymin = t_ax.get_tightbbox(renderer).min
        ax_xymax = t_ax.get_tightbbox(renderer).max
        top_lab_xymax = t_ax.xaxis.get_label().get_tightbbox(renderer).max
        #-------------------------------------------------------------------------------------------
        div_left_x = fig_inv.transform(ax_xymin)[0]
        div_right_x = fig_inv.transform(ax_xymax)[0]
        div_y = fig_inv.transform(top_lab_xymax)[1]
        div_y = div_y + .027 if plot_height==9 else div_y + .009
        #-------------------------------------------------------------------------------------------
        div_line = mlines.Line2D([div_left_x, div_right_x], [div_y, div_y], **c_tick_kw)
    #-------------------------------------------------------------------------------------------
    left_vert_x = fig_inv.transform(bot_ax_line.get_tightbbox(renderer).min)[0]
    if not box: right_vert_x = fig_inv.transform(top_ax_line.get_tightbbox(renderer).max)[0]
    #-------------------------------------------------------------------------------------------
    bot_vert_y = fig_inv.transform(top_ax_line.get_tightbbox(renderer).max)[1]
    top_vert_y = fig_inv.transform(bot_ax_line.get_tightbbox(renderer).min)[1]
    #-------------------------------------------------------------------------------------------
    left_vert = mlines.Line2D([[left_vert_x] * 2], [bot_vert_y, top_vert_y], **c_tick_kw)
    if not box:
        right_vert = mlines.Line2D([[right_vert_x] * 2], [bot_vert_y, top_vert_y], **c_tick_kw)
    #-------------------------------------------------------------------------------------------
    if 'tot' in plot_df.columns[-1] or (not box and 'tot' in test_col_str):
        fig.add_artist(div_line)
    #-------------------------------------------------------------------------------------------
    if not box: [fig.add_artist(art) for art in [left_vert, right_vert]]
    else: fig.add_artist(left_vert)

    # Returning the figure with a label to the save accordingly
    ###########################################################################################
    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 $^{1}/_{3}^{rd}$** 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]:
```
# Preparing important strings to use for the entire financial section
#################################################################################################
sum_aka_1 = 'tot_worldwide_gross'
#-----------------------------------------------------------------------------------------------
sum_aka_2 = 'tot_worldwide_profits'
#-----------------------------------------------------------------------------------------------
avg_aka_1 = 'avg_worldwide_gross'
#-----------------------------------------------------------------------------------------------
avg_aka_2 = 'avg_worldwide_profits'
#-----------------------------------------------------------------------------------------------
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]:

```
# Using a SQL query to get the IMDB movie data with the The Numbers financial information
#################################################################################################
db_query = """
SELECT
    tconst, start_year, primary_title, genre, genres, rating, creative_type,
    production_budget, 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;"""
#-------------------------------------------------------------------------------------------------
fin_df = pysqldf(db_query)
#-------------------------------------------------------------------------------------------------
display(fin_df.head(3))
display(fin_df.info())
#-------------------------------------------------------------------------------------------------
fin_df = fin_df.drop_duplicates(['tconst', 'start_year', 'primary_title'], keep=False)
#-------------------------------------------------------------------------------------------------
display(fin_df.info())
```

| | tconst | start_year | primary_title | genre | genres | rating | creative_type | production_budget | worldwide_g |
|---|---|---|---|---|---|---|---|---|---|
| **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]:
```
# Checking for duplicated data on a deeper level, trying to save as much data as possible
#########################################################################################################
dup_titles = \
fin_df.loc[fin_df.duplicated(['start_year', 'primary_title', 'worldwide_gross'], keep=False)]\
      .primary_title.unique()
#---------------------------------------------------------------------------------------------------------
drop_idx = []
#---------------------------------------------------------------------------------------------------------
for tit in dup_titles:
    tit_df = fin_df.loc[fin_df.primary_title == tit]
    #-----------------------------------------------------------------------------------------------------
    if len(tit_df.genres.unique()) > 1:
        drop_idx.extend(tit_df.index)
#---------------------------------------------------------------------------------------------------------
fin_df = fin_df.drop(drop_idx)
#---------------------------------------------------------------------------------------------------------
display(fin_df.head(3))
display(fin_df.info())
print('Movies dropped: ', len(drop_idx))
```

| | tconst | start_year | primary_title | genre | genres | rating | creative_type | production_budget | worldwide_g |
|---|---|---|---|---|---|---|---|---|---|
| **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]:

```
# Using an SQL query to create the necessary dataframe
#########################################################################################################
crew_strings = ['director', 'writer', 'actor', 'actress']
#---------------------------------------------------------------------------------------------------------
for crew_str in crew_strings:
    print('\033[1m\033[4m'+ crew_str.title() +' DataFrame:\033[0m')
    print('-------------------------------------------')
    #-----------------------------------------------------------------------------------------------------
    db_query = """
    SELECT
        tconst, nconst, primary_name, start_year, primary_title, genre, genres, rating,
        creative_type, production_budget, 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;"""
    #-----------------------------------------------------------------------------------------------------
    crew_df = pysqldf(db_query)
    #-----------------------------------------------------------------------------------------------------
    crew_df = crew_df.drop_duplicates(['tconst', 'nconst', 'primary_title'])
    #-----------------------------------------------------------------------------------------------------
    print('\033[1m\033[4mBEFORE DUP CHECK:\033[0m')
    display(crew_df.info())

    # Checking for duplicated data on a deeper level, trying to save as much data as possible
    #####################################################################################################
    dup_titles = \
    crew_df.loc[crew_df.duplicated(['start_year', 'primary_title'], keep=False)]\
           .primary_title.unique()
    #-----------------------------------------------------------------------------------------------------
    drop_idx = []
    #-----------------------------------------------------------------------------------------------------
    for tit in dup_titles:
        tit_df = crew_df.loc[crew_df.primary_title == tit]
        #-------------------------------------------------------------------------------------------------
        if len(tit_df.genres.unique()) > 1:
            drop_idx.extend(tit_df.index)
    #-----------------------------------------------------------------------------------------------------
    crew_df = crew_df.drop(drop_idx)
    #-----------------------------------------------------------------------------------------------------
    print('\033[1m\033[4mAFTER DUP CHECK:\033[0m')
    display(crew_df.info())
    print('\033[1mMovies dropped:\033[0m', len(drop_idx))
    print('-------------------------------------------')

    # Checking for duplicated data on a deeper level, trying to save as much data as possible
    #####################################################################################################
    if crew_str=='director': dir_df = crew_df
    if crew_str=='writer': writ_df = crew_df
    if crew_str=='actor': actor_df = crew_df
    if crew_str=='actress': actress_df = crew_df

# List of all crew dataframes
#########################################################################################################
crew_dfs = [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              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

In [49]:
```
# Example visualizations in README.md
```

In [32]:
```
# Preparing empty lists for important genre combinations based on each metric to be added to
#############################################################################################
finan_tot_gross_imp_g_combos = []
finan_tot_profits_imp_g_combos = []
finan_avg_gross_imp_g_combos = []
finan_avg_profits_imp_g_combos = []
```

In [33]:

```
# Creating, plotting, and saving the appropriate dataframes
#####################################################################################################
base_dir = 'visuals/Important Unique Genre Combinations/'
#-----------------------------------------------------------------------------------------------------
for aka_v in all_akas:
    plot_col = aka_v[4:]
    #-------------------------------------------------------------------------------------------------
    g_count_df = fin_df.groupby('genres').count()[['primary_title']]
    #-------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        g_agg_df = fin_df.groupby('genres').sum()[[plot_col]]
    if 'avg' in aka_v:
        g_agg_df = fin_df.groupby('genres').mean()[[plot_col]]
    #-------------------------------------------------------------------------------------------------
    genre_df = pd.concat([g_count_df, g_agg_df], axis=1)
    #-------------------------------------------------------------------------------------------------
    genre_df = genre_df.rename(columns={'primary_title':'num_titles', plot_col: aka_v})
    #-------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        genre_df['perc_of_'+ aka_v] = round((genre_df[aka_v] / fin_df[plot_col].sum()), 4)
    #-------------------------------------------------------------------------------------------------
    genre_df = genre_df.loc[genre_df['num_titles'] >= 3]
    #-------------------------------------------------------------------------------------------------
    all_years_top10 = genre_df.sort_values(aka_v, ascending=False).reset_index().head(10)
    #-------------------------------------------------------------------------------------------------
    top_combo_mapper = \
    fin_df.genres.map(lambda x: x if x in list(all_years_top10.genres) else None)
    #-------------------------------------------------------------------------------------------------
    imp_list = list(all_years_top10.genres)
    #-------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        rat_agg_df = \
        fin_df.groupby([top_combo_mapper, 'rating']).sum()[[plot_col]].reset_index('rating')
        #---------------------------------------------------------------------------------------------
        rat_df = \
        rat_agg_df.merge(all_years_top10.set_index('genres'), left_index=True, right_index=True)\
                  .reset_index().rename({aka_v:'top_cat', plot_col: aka_v}, axis=1)\
                  .sort_values('top_cat', ascending=False)
    #-------------------------------------------------------------------------------------------------
    if 'avg' in aka_v:
        rat_count_df = fin_df.groupby([top_combo_mapper, 'rating']).count()[['primary_title']]
        #---------------------------------------------------------------------------------------------
        rat_agg_df = \
        fin_df.groupby([top_combo_mapper, 'rating']).mean()[[plot_col]]
        #---------------------------------------------------------------------------------------------
        rat_df = \
        rat_agg_df.merge(rat_count_df, left_index=True, right_index=True).reset_index('rating')\
                  .merge(all_years_top10.set_index('genres')[[aka_v]], left_index=True,
                         right_index=True).reset_index()\
                  .rename({'primary_title':'num_titles', aka_v:'top_cat', plot_col: aka_v},
                          axis=1).sort_values(['top_cat', aka_v], ascending=False)
    #-------------------------------------------------------------------------------------------------
    box_df = pd.DataFrame()
    fin_import_order_dict = {}
    #-------------------------------------------------------------------------------------------------
    for g_combo in all_years_top10.genres:
        imp_g_combo_map = fin_df['genres'].map(lambda x: True if g_combo==x else False)
        #-----------------------------------------------------------------------------------------
        imp_genres_df = fin_df.loc[imp_g_combo_map].copy()[['genres', plot_col]]
        #-----------------------------------------------------------------------------------------
        fin_import_order_dict[g_combo] = imp_genres_df[plot_col].mean()
        #-----------------------------------------------------------------------------------------
        box_df = box_df.append(imp_genres_df)
    #-------------------------------------------------------------------------------------------------
    fin_import_order = \
    sorted(fin_import_order_dict.keys(), key=lambda k: fin_import_order_dict[k], reverse=True)
    #-------------------------------------------------------------------------------------------------
    box_df = box_df.sort_values('genres',
                                key=lambda g_col: g_col.map(lambda x: fin_import_order.index(x)))
    #-------------------------------------------------------------------------------------------------
```

```
    fig, fig_tit = cat_plot(all_years_top10, 'combos', 'genres')
    #-----------------------------------------------------------------------------------------------
    if not os.path.isdir(base_dir + fig_tit +'/'):
        os.mkdir(base_dir + fig_tit +'/')
    #-----------------------------------------------------------------------------------------------
    f_name = base_dir + fig_tit +'/1 - Top 10 Genre Combos.jpeg'
    #-----------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-----------------------------------------------------------------------------------------------
    new_dir = base_dir + fig_tit +'/'
    #-----------------------------------------------------------------------------------------------
    fig, fig_tit = cat_plot(box_df, 'combos', 'genres', aka_v)
    #-----------------------------------------------------------------------------------------------
    f_name = new_dir +'2 - '+ fig_tit +'.jpeg'
    #-----------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-----------------------------------------------------------------------------------------------
    fig, fig_tit = cat_plot(rat_df, 'rating', 'combos')
    #-----------------------------------------------------------------------------------------------
    f_name = new_dir +'3 - Top 10 Genre Combos - MPA Ratings.jpeg'
    #-----------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-----------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        if 'gross' in aka_v: finan_tot_gross_imp_g_combos = list(all_years_top10.genres)
        if 'profits' in aka_v: finan_tot_profits_imp_g_combos = list(all_years_top10.genres)
    if 'avg' in aka_v:
        if 'gross' in aka_v: finan_avg_gross_imp_g_combos = list(all_years_top10.genres)
        if 'profits' in aka_v: finan_avg_profits_imp_g_combos = list(all_years_top10.genres)
#-----------------------------------------------------------------------------------------------
# display(fin_combo_all_cols_df)
```

# Financially Important Unique Genre Combinations - Crew

In [49]:

```
# Example visualizations in README.md
```

In [34]:

```
# Creating, plotting, and saving the appropriate dataframes
#####################################################################################################
base_dir = 'visuals/Important Unique Genre Combinations - Crew/'
#-----------------------------------------------------------------------------------------------------
for crew_str, crew_df in zip(crew_strings, crew_dfs):
    if crew_str != 'actress': plot_type = crew_str +'s'
    else: plot_type = crew_str +'es'
    #-------------------------------------------------------------------------------------------------
    crew_dir = base_dir + plot_type.title() +'/'
    #-------------------------------------------------------------------------------------------------
    for aka_v in all_akas:
        plot_col = aka_v[4:]
        #---------------------------------------------------------------------------------------------
        if 'tot' in aka_v:
            if 'gross' in aka_v: imp_g_combos = finan_tot_gross_imp_g_combos
            if 'profits' in aka_v: imp_g_combos = finan_tot_profits_imp_g_combos
        if 'avg' in aka_v:
            if 'gross' in aka_v: imp_g_combos = finan_avg_gross_imp_g_combos
            if 'profits' in aka_v: imp_g_combos = finan_avg_profits_imp_g_combos
        #---------------------------------------------------------------------------------------------
        for g_combo in imp_g_combos:
            g_combo_df = crew_df.set_index('genres').loc[g_combo].copy()
            #-----------------------------------------------------------------------------------------
            if type(g_combo_df) == pd.Series:
                print('\033[1mUnique Genre Combination: \033[0m', g_combo)
                print('\033[1mFinancial Metric: \033[0m', aka_v)
                print('This unique genre combination in this metric was only '+\
                      'associated with one '+ crew_str +'.')
                print("The crew member's name was: ", g_combo_df['primary_name'])
                print('------------------------------------------------------------')
                continue
            #-----------------------------------------------------------------------------------------
            g_combo_df.reset_index(inplace=True)
            #-----------------------------------------------------------------------------------------
            top_crew_num_titles = \
            g_combo_df.groupby('primary_name').count()[['primary_title', 'genres']]
            #-----------------------------------------------------------------------------------------
            if 'tot' in aka_v:
                top_crew_aka_df = g_combo_df.groupby('primary_name').sum()[[plot_col]]
            if 'avg' in aka_v:
                top_crew_aka_df = g_combo_df.groupby('primary_name').mean()[[plot_col]]
            #-----------------------------------------------------------------------------------------
            g_combo_top_crew_df = pd.concat([top_crew_num_titles, top_crew_aka_df], axis=1)\
                                    .sort_values(plot_col, ascending=False).head(10)
            #-----------------------------------------------------------------------------------------
            g_combo_top_crew_df.rename(columns={'primary_title': 'num_titles', plot_col: aka_v},
                                       inplace=True)
            #-----------------------------------------------------------------------------------------
            if 'tot' in aka_v:
                g_combo_top_crew_df['perc_of_' + aka_v] = \
                round((g_combo_top_crew_df[aka_v] / g_combo_df[plot_col].sum()), 4)
            #-----------------------------------------------------------------------------------------
            g_combo_top_crew_df = g_combo_top_crew_df.reset_index()
            g_combo_top_crew_df['genres'] = g_combo_top_crew_df['genres'].map(lambda x: g_combo)
            g_combo_top_crew_df = g_combo_top_crew_df.set_index('genres')
            #-----------------------------------------------------------------------------------------
            fig, fig_tit = \
            cat_plot(g_combo_top_crew_df, 'combos', plot_type)
            #-----------------------------------------------------------------------------------------
            if not os.path.isdir(crew_dir + fig_tit +'/'):
                os.mkdir(crew_dir + fig_tit +'/')
            #-----------------------------------------------------------------------------------------
            f_name = crew_dir + fig_tit +'/'+ g_combo +'.jpeg'
            #-----------------------------------------------------------------------------------------
            fig.savefig(f_name, bbox_inches='tight')
            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.

In [49]:
```
# Example visualizations in README.md
```

In [35]:
```
# Getting a list of unique genres
#########################################################################################################
genres_split = fin_df['genres'].map(lambda x: x.split(','))
all_genres = [g for g_list in genres_split for g in g_list]
unique_genres = sorted(set(all_genres))
#---------------------------------------------------------------------------------------------------------
print('\033[1m'+'Number of Unique Genres:'+'\033[0m', len(unique_genres))
print('-------------------------------------------------------------------------------')
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', 'Thrille
r', 'War', 'Western']
```

In [36]:
```
# Preparing empty lists for important individual genres based on each metric to be added to
#########################################################################################################
finan_tot_gross_imp_individ_gs = []
finan_tot_profits_imp_individ_gs = []
finan_avg_gross_imp_individ_gs = []
finan_avg_profits_imp_individ_gs = []
```

In [37]:

```
# Creating, plotting, and saving the appropriate dataframe
#########################################################################################################
base_dir = 'visuals/Important Individual Genres/'
#---------------------------------------------------------------------------------------------------------
for aka_v in all_akas:
    plot_col = aka_v[4:]
    #-----------------------------------------------------------------------------------------------------
    individ_gs_df = pd.DataFrame()
    #-----------------------------------------------------------------------------------------------------
    for g in unique_genres:
        g_map = fin_df['genres'].map(lambda x: g if g in x else None)
        #-------------------------------------------------------------------------------------------------
        g_df_p1 = fin_df.groupby(g_map).count()[['primary_title']]
        #-------------------------------------------------------------------------------------------------
        if 'tot' in aka_v:
            g_df_p2 = fin_df.groupby(g_map).sum()[[ plot_col]]
        #-------------------------------------------------------------------------------------------------
        if 'avg' in aka_v:
            g_df_p2 = fin_df.groupby(g_map).mean()[[plot_col]]
        #-------------------------------------------------------------------------------------------------
        g_df = pd.concat([g_df_p1, g_df_p2], axis=1).reset_index()
        #-------------------------------------------------------------------------------------------------
        g_df.rename({'primary_title': 'num_titles', plot_col: aka_v}, axis=1, inplace=True)
        #-------------------------------------------------------------------------------------------------
        individ_gs_df = individ_gs_df.append(g_df)
    #-----------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        individ_gs_df['perc_of_'+ aka_v] = \
        round((individ_gs_df[aka_v] / fin_df[plot_col].sum()), 4)
    #-----------------------------------------------------------------------------------------------------
    individ_gs_df = individ_gs_df.loc[individ_gs_df['num_titles'] >= 9]
    import_gs_df = individ_gs_df.loc[individ_gs_df[aka_v] >= individ_gs_df[aka_v].quantile(.75)]
    #-----------------------------------------------------------------------------------------------------
    import_gs_df = import_gs_df.sort_values(aka_v, ascending=False)
    #-----------------------------------------------------------------------------------------------------
    top_gs_rat_df = pd.DataFrame()
    #-----------------------------------------------------------------------------------------------------
    box_df = pd.DataFrame()
    #-----------------------------------------------------------------------------------------------------
    for g in import_gs_df.genres:
        top_individ_gs_mapper = fin_df.genres.map(lambda x: g if g in x else None)
        #-------------------------------------------------------------------------------------------------
        if 'tot' in aka_v:
            rat_agg_df = \
            fin_df.groupby([top_individ_gs_mapper, 'rating']).sum()[[plot_col]]\
                  .reset_index('rating')
            #---------------------------------------------------------------------------------------------
            rat_df = \
            rat_agg_df.merge(import_gs_df.set_index('genres'),left_index=True,right_index=True)\
                      .reset_index().rename({aka_v:'top_cat', plot_col: aka_v}, axis=1)\
                      .sort_values('top_cat', ascending=False)
        #-------------------------------------------------------------------------------------------------
        if 'avg' in aka_v:
            rat_count_df = \
            fin_df.groupby([top_individ_gs_mapper, 'rating']).count()[['primary_title']]
            #---------------------------------------------------------------------------------------------
            rat_agg_df = \
            fin_df.groupby([top_individ_gs_mapper, 'rating']).mean()[[plot_col]]
            #---------------------------------------------------------------------------------------------
            rat_df = \
            rat_agg_df.merge(rat_count_df, left_index=True, right_index=True)\
                      .reset_index('rating')\
                      .merge(import_gs_df.set_index('genres')[[aka_v]], left_index=True,
                             right_index=True).reset_index()\
                      .rename({'primary_title':'num_titles', aka_v:'top_cat', plot_col: aka_v},
                              axis=1).sort_values(aka_v, ascending=False)
        #-------------------------------------------------------------------------------------------------
        g_box_df = fin_df.copy()
        g_box_df['genres'] = top_individ_gs_mapper
        g_box_df = g_box_df.dropna(subset=['genres'])[['genres', plot_col]]
```

```
        #-------------------------------------------------------------------------------------------
        fin_import_order_dict[g] = g_box_df[plot_col].mean()
        #-------------------------------------------------------------------------------------------
        top_gs_rat_df = top_gs_rat_df.append(rat_df)
        #-------------------------------------------------------------------------------------------
        box_df = box_df.append(g_box_df)
    #-----------------------------------------------------------------------------------------------
    fin_import_order = \
    sorted(fin_import_order_dict.keys(), key=lambda k: fin_import_order_dict[k], reverse=True)
    #-----------------------------------------------------------------------------------------------
    box_df = \
    box_df.sort_values('genres',
                       key=lambda g_col: g_col.map(lambda x: fin_import_order.index(x)))
    #-----------------------------------------------------------------------------------------------
    fig, fig_tit = cat_plot(import_gs_df, 'individs', 'genres')
    #-----------------------------------------------------------------------------------------------
    if not os.path.isdir(base_dir + fig_tit +'/'):
        os.mkdir(base_dir + fig_tit +'/')
    #-----------------------------------------------------------------------------------------------
    f_name = base_dir + fig_tit +'/1 - Important Individual Genres.jpeg'
    #-----------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-----------------------------------------------------------------------------------------------
    new_dir = base_dir + fig_tit +'/'
    #-----------------------------------------------------------------------------------------------
    fig, fig_tit = cat_plot(box_df, 'individs', 'genres', aka_v)
    #-----------------------------------------------------------------------------------------------
    f_name = new_dir +'2 - '+ fig_tit +'.jpeg'
    #-----------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-----------------------------------------------------------------------------------------------
    fig, fig_tit = cat_plot(top_gs_rat_df, 'rating', 'individs')
    #-----------------------------------------------------------------------------------------------
    f_name = new_dir + '3 - Important Individual Genres - MPA Ratings.jpeg'
    #-----------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-----------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        if 'gross' in aka_v: finan_tot_gross_imp_individ_gs = list(import_gs_df.genres)
        if 'profits' in aka_v: finan_tot_profits_imp_individ_gs = list(import_gs_df.genres)
    if 'avg' in aka_v:
        if 'gross' in aka_v: finan_avg_gross_imp_individ_gs = list(import_gs_df.genres)
        if 'profits' in aka_v: finan_avg_profits_imp_individ_gs = list(import_gs_df.genres)
#---------------------------------------------------------------------------------------------------
# display(fin_individ_all_cols_df)
```

# Financially Important Individual Genres - Crew

In [49]:

```
# Example visualizations in README.md
```

In [38]:
```
# Creating, plotting, and saving the appropriate dataframes
#####################################################################################################
base_dir = 'visuals/Important Individual Genres - Crew/'
#-----------------------------------------------------------------------------------------------------
for crew_str, crew_df in zip(crew_strings, crew_dfs):
    if crew_str != 'actress': plot_type = crew_str +'s'
    else: plot_type = crew_str +'es'
    #-------------------------------------------------------------------------------------------------
    crew_dir = base_dir + plot_type.title() +'/'
    #-------------------------------------------------------------------------------------------------
    for aka_v in all_akas:
        plot_col = aka_v[4:]
        #---------------------------------------------------------------------------------------------
        if 'tot' in aka_v:
            if 'gross' in aka_v: imp_individ_gs = finan_tot_gross_imp_individ_gs
            if 'profits' in aka_v: imp_individ_gs = finan_tot_profits_imp_individ_gs
        if 'avg' in aka_v:
            if 'gross' in aka_v: imp_individ_gs = finan_avg_gross_imp_individ_gs
            if 'profits' in aka_v: imp_individ_gs = finan_avg_profits_imp_individ_gs
        #---------------------------------------------------------------------------------------------
        for g in imp_individ_gs:
            g_map = crew_df['genres'].map(lambda x: g if g in x else None)
            #-----------------------------------------------------------------------------------------
            crew_num_titles = crew_df.groupby(['primary_name', g_map]).count()[['primary_title']]
            #-----------------------------------------------------------------------------------------
            if 'tot' in aka_v:
                crew_aka_v = crew_df.groupby(['primary_name', g_map]).sum()[[plot_col]]
            if 'avg' in aka_v:
                crew_aka_v = crew_df.groupby(['primary_name', g_map]).mean()[[plot_col]]
            #-----------------------------------------------------------------------------------------
            g_crew_df = pd.concat([crew_num_titles, crew_aka_v], axis=1)\
                          .sort_values(plot_col, ascending=False)
            #-----------------------------------------------------------------------------------------
            g_crew_df.rename(columns={'primary_title': 'num_titles',
                                      plot_col: aka_v}, inplace=True)
            #-----------------------------------------------------------------------------------------
            if 'tot' in aka_v:
                g_crew_df['perc_of_'+ aka_v] = \
                round((g_crew_df[aka_v] / g_crew_df[aka_v].sum()), 4)
            #-----------------------------------------------------------------------------------------
            g_top_crew_df = g_crew_df.reset_index('primary_name').head(10)
            #-----------------------------------------------------------------------------------------
            fig, fig_tit = \
            cat_plot(g_top_crew_df, 'individs', plot_type)
            #-----------------------------------------------------------------------------------------
            if not os.path.isdir(crew_dir + fig_tit +'/'):
                os.mkdir(crew_dir + fig_tit +'/')
            #-----------------------------------------------------------------------------------------
            f_name = crew_dir + fig_tit +'/'+ g +'.jpeg'
            #-----------------------------------------------------------------------------------------
            fig.savefig(f_name, bbox_inches='tight')
            plt.close(fig)
```

# § OpusData Category Types

## Financially Important Genres

In [49]:
```
# Example visualizations in README.md
```

In [39]:

```
# Preparing empty lists for important genres based on each metric to be added to
#############################################################################################################
finan_tot_gross_imp_genres = []
finan_tot_profits_imp_genres = []
finan_avg_gross_imp_genres = []
finan_avg_profits_imp_genres = []
```

In [40]:

```
# Creating, plotting, and saving the appropriate dataframes
#####################################################################################################
base_dir = 'visuals/Important Genres/'
#-----------------------------------------------------------------------------------------------------
for aka_v in all_akas:
    plot_col = aka_v[4:]
    #-------------------------------------------------------------------------------------------------
    g_count_df = fin_df.groupby('genre').count()[['primary_title']]
    #-------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        g_agg_df = fin_df.groupby('genre').sum()[[plot_col]]
    if 'avg' in aka_v:
        g_agg_df = fin_df.groupby('genre').mean()[[plot_col]]
    #-------------------------------------------------------------------------------------------------
    genre_df = pd.concat([g_count_df, g_agg_df], axis=1)
    #-------------------------------------------------------------------------------------------------
    genre_df = genre_df.rename(columns={'primary_title': 'num_titles', plot_col: aka_v})
    #-------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        genre_df['perc_of_'+ aka_v] = round((genre_df[aka_v] / fin_df[plot_col].sum()), 4)
    #-------------------------------------------------------------------------------------------------
    genre_df = genre_df.loc[genre_df['num_titles'] >= 9]
    genre_df = genre_df.loc[genre_df[aka_v] >= genre_df[aka_v].quantile(.75)]
    #-------------------------------------------------------------------------------------------------
    all_years_top10 = genre_df.sort_values(aka_v, ascending=False).reset_index()\
                              .rename(columns={'genre': 'genres'}).head(10)
    #-------------------------------------------------------------------------------------------------
    top_combo_mapper = \
    fin_df.genre.map(lambda x: x if x in list(all_years_top10.genres) else None)
    #-------------------------------------------------------------------------------------------------
    imp_list = list(all_years_top10.genres)
    #-------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        rat_agg_df = \
        fin_df.groupby([top_combo_mapper, 'rating']).sum()[[plot_col]].reset_index('rating')
        #---------------------------------------------------------------------------------------------
        rat_df = \
        rat_agg_df.merge(all_years_top10.set_index('genres'), left_index=True, right_index=True)\
                  .reset_index().rename({aka_v:'top_cat', plot_col: aka_v}, axis=1)\
                  .sort_values('top_cat', ascending=False)
    #-------------------------------------------------------------------------------------------------
    if 'avg' in aka_v:
        rat_count_df = fin_df.groupby([top_combo_mapper, 'rating']).count()[['primary_title']]
        #---------------------------------------------------------------------------------------------
        rat_agg_df = \
        fin_df.groupby([top_combo_mapper, 'rating']).mean()[[plot_col]]
        #---------------------------------------------------------------------------------------------
        rat_df = \
        rat_agg_df.merge(rat_count_df, left_index=True, right_index=True).reset_index('rating')\
                  .merge(all_years_top10.set_index('genres')[[aka_v]], left_index=True,
                         right_index=True).reset_index()\
                  .rename({'primary_title':'num_titles', aka_v:'top_cat', plot_col: aka_v},
                          axis=1).sort_values(['top_cat', aka_v], ascending=False)
    #-------------------------------------------------------------------------------------------------
    rat_df = rat_df.rename(columns={'index': 'genres'})
    #-------------------------------------------------------------------------------------------------
    box_df = pd.DataFrame()
    fin_import_order_dict = {}
    #-------------------------------------------------------------------------------------------------
    for genre in all_years_top10.genres:
        imp_genre_map = fin_df['genre'].map(lambda x: True if genre==x else False)
        #---------------------------------------------------------------------------------------------
        imp_genres_df = fin_df.loc[imp_genre_map].copy()[['genre', plot_col]]
        #---------------------------------------------------------------------------------------------
        fin_import_order_dict[genre] = imp_genres_df[plot_col].mean()
        #---------------------------------------------------------------------------------------------
        box_df = box_df.append(imp_genres_df)
    #-------------------------------------------------------------------------------------------------
    fin_import_order = \
    sorted(fin_import_order_dict.keys(), key=lambda k: fin_import_order_dict[k], reverse=True)
```

```
    #-------------------------------------------------------------------------------------------------
    box_df = box_df.sort_values('genre',
                                key=lambda g_col: g_col.map(lambda x: fin_import_order.index(x)))
    #-------------------------------------------------------------------------------------------------
    box_df = box_df.rename(columns={'genre': 'genres'})
    #-------------------------------------------------------------------------------------------------
    fig, fig_tit = cat_plot(all_years_top10, 'genres', 'genres')
    #-------------------------------------------------------------------------------------------------
    if not os.path.isdir(base_dir + fig_tit +'/'):
        os.mkdir(base_dir + fig_tit +'/')
    #-------------------------------------------------------------------------------------------------
    f_name = base_dir + fig_tit +'/1 - Top Genres.jpeg'
    #-------------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-------------------------------------------------------------------------------------------------
    new_dir = base_dir + fig_tit +'/'
    #-------------------------------------------------------------------------------------------------
    fig, fig_tit = cat_plot(box_df, 'genres', 'genres', aka_v)
    #-------------------------------------------------------------------------------------------------
    f_name = new_dir +'2 - '+ fig_tit +'.jpeg'
    #-------------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-------------------------------------------------------------------------------------------------
    fig, fig_tit = cat_plot(rat_df, 'rating', 'genres')
    #-------------------------------------------------------------------------------------------------
    f_name = new_dir +'3 - Top Genres - MPA Ratings.jpeg'
    #-------------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        if 'gross' in aka_v: finan_tot_gross_imp_genres = list(all_years_top10.genres)
        if 'profits' in aka_v: finan_tot_profits_imp_genres = list(all_years_top10.genres)
    if 'avg' in aka_v:
        if 'gross' in aka_v: finan_avg_gross_imp_genres = list(all_years_top10.genres)
        if 'profits' in aka_v: finan_avg_profits_imp_genres = list(all_years_top10.genres)
```

# Financially Important Genres - Crew

In [49]:

```
# Example visualizations in README.md
```

In [41]:

```
# Creating, plotting, and saving the appropriate dataframes
#########################################################################################################
base_dir = 'visuals/Important Genres - Crew/'
#---------------------------------------------------------------------------------------------------------
for crew_str, crew_df in zip(crew_strings, crew_dfs):
    if crew_str != 'actress': plot_type = crew_str +'s'
    else: plot_type = crew_str +'es'
    #-----------------------------------------------------------------------------------------------------
    crew_dir = base_dir + plot_type.title() +'/'
    #-----------------------------------------------------------------------------------------------------
    for aka_v in all_akas:
        plot_col = aka_v[4:]
        #-------------------------------------------------------------------------------------------------
        if 'tot' in aka_v:
            if 'gross' in aka_v: imp_genres = finan_tot_gross_imp_genres
            if 'profits' in aka_v: imp_genres = finan_tot_profits_imp_genres
        if 'avg' in aka_v:
            if 'gross' in aka_v: imp_genres = finan_avg_gross_imp_genres
            if 'profits' in aka_v: imp_genres = finan_avg_profits_imp_genres
        #-------------------------------------------------------------------------------------------------
        for genre in imp_genres:
            genre_df = crew_df.set_index('genre').loc[genre].copy().drop(columns={'genres'})
            #---------------------------------------------------------------------------------------------
            if type(genre_df) == pd.Series:
                print('\033[1mGenre: \033[0m', genre)
                print('\033[1mFinancial Metric: \033[0m', aka_v)
                print('This genre in this metric was only '+\
                      'associated with one '+ crew_str +'.')
                print("The crew member's name was: ", genre_df['primary_name'])
                print('-------------------------------------------------------------')
                continue
            #---------------------------------------------------------------------------------------------
            genre_df = genre_df.reset_index().rename(columns={'genre': 'genres'})
            #---------------------------------------------------------------------------------------------
            top_crew_num_titles = \
            genre_df.groupby('primary_name').count()[['primary_title', 'genres']]
            #---------------------------------------------------------------------------------------------
            if 'tot' in aka_v:
                top_crew_aka_df = genre_df.groupby('primary_name').sum()[[plot_col]]
            if 'avg' in aka_v:
                top_crew_aka_df = genre_df.groupby('primary_name').mean()[[plot_col]]
            #---------------------------------------------------------------------------------------------
            genre_top_crew_df = pd.concat([top_crew_num_titles, top_crew_aka_df], axis=1)\
                                  .sort_values(plot_col, ascending=False).head(10)
            #---------------------------------------------------------------------------------------------
            genre_top_crew_df.rename(columns={'primary_title': 'num_titles', plot_col: aka_v},
                                     inplace=True)
            #---------------------------------------------------------------------------------------------
            if 'tot' in aka_v:
                genre_top_crew_df['perc_of_' + aka_v] = \
                round((genre_top_crew_df[aka_v] / genre_df[plot_col].sum()), 4)
            #---------------------------------------------------------------------------------------------
            genre_top_crew_df = genre_top_crew_df.reset_index()
            genre_top_crew_df['genres'] = genre_top_crew_df['genres'].map(lambda x: genre)
            genre_top_crew_df = genre_top_crew_df.set_index('genres')
            #---------------------------------------------------------------------------------------------
            fig, fig_tit = \
            cat_plot(genre_top_crew_df, 'genres', plot_type)
            #---------------------------------------------------------------------------------------------
            if not os.path.isdir(crew_dir + fig_tit +'/'):
                os.mkdir(crew_dir + fig_tit +'/')
            #---------------------------------------------------------------------------------------------
            if '/' in genre:
                genre = ' - '.join(genre.split('/'))
            #---------------------------------------------------------------------------------------------
            f_name = crew_dir + fig_tit +'/'+ genre +'.jpeg'
            #---------------------------------------------------------------------------------------------
            fig.savefig(f_name, bbox_inches='tight')
            plt.close(fig)
```

# Financially Important Creative Types

In [49]:
```
# Example visualizations in README.md
```

In [42]:
```
# Preparing empty lists for important creative types based on each metric to be added to
#############################################################################################
finan_tot_gross_imp_creative_types = []
finan_tot_profits_imp_creative_types = []
finan_avg_gross_imp_creative_types = []
finan_avg_profits_imp_creative_types = []
```

In [43]:

```
# Creating, plotting, and saving the appropriate dataframe
#########################################################################################################
base_dir = 'visuals/Important Creative Types/'
#---------------------------------------------------------------------------------------------------------
for aka_v in all_akas:
    plot_col = aka_v[4:]
    #-----------------------------------------------------------------------------------------------------
    ct_count_df = fin_df.groupby('creative_type').count()[['primary_title']]
    #-----------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        ct_agg_df = fin_df.groupby('creative_type').sum()[[plot_col]]
    if 'avg' in aka_v:
        ct_agg_df = fin_df.groupby('creative_type').mean()[[plot_col]]
    #-----------------------------------------------------------------------------------------------------
    creative_df = pd.concat([ct_count_df, ct_agg_df], axis=1)
    #-----------------------------------------------------------------------------------------------------
    creative_df = creative_df.rename(columns={'primary_title':'num_titles', plot_col: aka_v})
    #-----------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        creative_df['perc_of_'+ aka_v] = round((creative_df[aka_v] / fin_df[plot_col].sum()), 4)
    #-----------------------------------------------------------------------------------------------------
    if aka_v == sum_aka_1:
        fin_creative_df = creative_df.copy()
    elif aka_v == sum_aka_2:
        fin_creative_df = \
        pd.concat([fin_creative_df, creative_df[[aka_v, 'perc_of_'+ aka_v]]], axis=1)
    else:
        fin_creative_df = pd.concat([fin_creative_df, creative_df[aka_v]], axis=1)
    #-----------------------------------------------------------------------------------------------------
    creative_df = creative_df.loc[creative_df['num_titles'] >= 9]
    creative_df = creative_df.loc[creative_df[aka_v] >= creative_df[aka_v].quantile(2/3)]
    #-----------------------------------------------------------------------------------------------------
    all_years_top10 = creative_df.sort_values(aka_v, ascending=False).reset_index().head(10)
    #-----------------------------------------------------------------------------------------------------
    top_c_type_mapper = \
    fin_df.creative_type.map(lambda x: x if x in list(all_years_top10.creative_type) else None)
    #-----------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        rat_agg_df = \
        fin_df.groupby([top_c_type_mapper, 'rating']).sum()[[plot_col]].reset_index('rating')
        #-------------------------------------------------------------------------------------------------
        rat_df = \
        rat_agg_df.merge(all_years_top10.set_index('creative_type'), left_index=True,
                         right_index=True)\
                  .reset_index().rename({aka_v:'top_cat', plot_col: aka_v}, axis=1)\
                  .sort_values('top_cat', ascending=False)
    #-----------------------------------------------------------------------------------------------------
    if 'avg' in aka_v:
        rat_count_df = fin_df.groupby([top_c_type_mapper, 'rating']).count()[['primary_title']]
        #-------------------------------------------------------------------------------------------------
        rat_agg_df = \
        fin_df.groupby([top_c_type_mapper, 'rating']).mean()[[plot_col]]
        #-------------------------------------------------------------------------------------------------
        rat_df = \
        rat_agg_df.merge(rat_count_df, left_index=True, right_index=True).reset_index('rating')\
                  .merge(all_years_top10.set_index('creative_type')[[aka_v]], left_index=True,
                         right_index=True).reset_index()\
                  .rename({'primary_title':'num_titles', aka_v:'top_cat', plot_col: aka_v},
                          axis=1).sort_values(['top_cat', aka_v], ascending=False)
    #-----------------------------------------------------------------------------------------------------
    box_df = pd.DataFrame()
    fin_import_order_dict = {}
    #-----------------------------------------------------------------------------------------------------
    for c_type in all_years_top10.creative_type:
        imp_c_type_map = fin_df['creative_type'].map(lambda x: True if c_type==x else False)
        #-------------------------------------------------------------------------------------------------
        imp_c_type_df = fin_df.loc[imp_c_type_map].copy()[['creative_type', plot_col]]
        #-------------------------------------------------------------------------------------------------
        fin_import_order_dict[c_type] = imp_c_type_df[plot_col].mean()
        #-------------------------------------------------------------------------------------------------
```

```
        box_df = box_df.append(imp_c_type_df)
    #-------------------------------------------------------------------------------------------------
    fin_import_order = \
    sorted(fin_import_order_dict.keys(), key=lambda k: fin_import_order_dict[k], reverse=True)
    #-------------------------------------------------------------------------------------------------
    box_df = box_df.sort_values('creative_type',
                                key=lambda g_col: g_col.map(lambda x: fin_import_order.index(x)))
    #-------------------------------------------------------------------------------------------------
    fig, fig_tit = cat_plot(all_years_top10, 'creative_type', 'creative_type')
    #-------------------------------------------------------------------------------------------------
    if not os.path.isdir(base_dir + fig_tit +'/'):
        os.mkdir(base_dir + fig_tit +'/')
    #-------------------------------------------------------------------------------------------------
    f_name = base_dir + fig_tit +'/1 - Top Creative Types.jpeg'
    #-------------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-------------------------------------------------------------------------------------------------
    new_dir = base_dir + fig_tit +'/'
    #-------------------------------------------------------------------------------------------------
    fig, fig_tit = cat_plot(box_df, 'creative_type', 'creative_type', aka_v)
    #-------------------------------------------------------------------------------------------------
    f_name = new_dir +'2 - '+ fig_tit +'.jpeg'
    #-------------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-------------------------------------------------------------------------------------------------
    fig, fig_tit = cat_plot(rat_df, 'rating', 'creative_type')
    #-------------------------------------------------------------------------------------------------
    f_name = new_dir +'3 - Top Creative Types - MPA Ratings.jpeg'
    #-------------------------------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
    #-------------------------------------------------------------------------------------------------
    if 'tot' in aka_v:
        if 'gross' in aka_v:
            finan_tot_gross_imp_creative_types = list(all_years_top10.creative_type)
        if 'profits' in aka_v:
            finan_tot_profits_imp_creative_types = list(all_years_top10.creative_type)
    if 'avg' in aka_v:
        if 'gross' in aka_v:
            finan_avg_gross_imp_creative_types = list(all_years_top10.creative_type)
        if 'profits' in aka_v:
            finan_avg_profits_imp_creative_types = list(all_years_top10.creative_type)
#-------------------------------------------------------------------------------------------------
# display(fin_creative_df)
```

# Financially Important Creative Type - Crew

In [49]:

```
# Example visualizations in README.md
```

In [44]:

```
# Creating, plotting, and saving the appropriate dataframes
#####################################################################################################
base_dir = 'visuals/Important Creative Types - Crew/'
#-----------------------------------------------------------------------------------------------------
for crew_str, crew_df in zip(crew_strings, crew_dfs):
    if crew_str != 'actress': plot_type = crew_str +'s'
    else: plot_type = crew_str +'es'
    #-------------------------------------------------------------------------------------------------
    crew_dir = base_dir + plot_type.title() +'/'
    #-------------------------------------------------------------------------------------------------
    for aka_v in all_akas:
        plot_col = aka_v[4:]
        #---------------------------------------------------------------------------------------------
        if 'tot' in aka_v:
            if 'gross' in aka_v: imp_creative_types = finan_tot_gross_imp_creative_types
            if 'profits' in aka_v: imp_creative_types = finan_tot_profits_imp_creative_types
        if 'avg' in aka_v:
            if 'gross' in aka_v: imp_creative_types = finan_avg_gross_imp_creative_types
            if 'profits' in aka_v: imp_creative_types = finan_avg_profits_imp_creative_types
        #---------------------------------------------------------------------------------------------
        for creative_type in imp_creative_types:
            creative_type_df = crew_df.set_index('creative_type').loc[creative_type].copy()
            #-----------------------------------------------------------------------------------------
            creative_type_df.reset_index(inplace=True)
            #-----------------------------------------------------------------------------------------
            top_crew_num_titles = \
            creative_type_df.groupby('primary_name').count()[['primary_title', 'creative_type']]
            #-----------------------------------------------------------------------------------------
            if 'tot' in aka_v:
                top_crew_aka_df = creative_type_df.groupby('primary_name').sum()[[plot_col]]
            if 'avg' in aka_v:
                top_crew_aka_df = creative_type_df.groupby('primary_name').mean()[[plot_col]]
            #-----------------------------------------------------------------------------------------
            creative_type_top_crew_df = \
            pd.concat([top_crew_num_titles, top_crew_aka_df], axis=1)\
              .sort_values(plot_col, ascending=False).head(10)
            #-----------------------------------------------------------------------------------------
            creative_type_top_crew_df.rename(columns={'primary_title': 'num_titles',
                                                      plot_col: aka_v}, inplace=True)
            #-----------------------------------------------------------------------------------------
            if 'tot' in aka_v:
                creative_type_top_crew_df['perc_of_' + aka_v] = \
                round((creative_type_top_crew_df[aka_v] / creative_type_df[plot_col].sum()), 4)
            #-----------------------------------------------------------------------------------------
            creative_type_top_crew_df = creative_type_top_crew_df.reset_index()
            #-----------------------------------------------------------------------------------------
            creative_type_top_crew_df['creative_type'] = \
            creative_type_top_crew_df['creative_type'].map(lambda x: creative_type)
            #-----------------------------------------------------------------------------------------
            creative_type_top_crew_df = creative_type_top_crew_df.set_index('creative_type')
            #-----------------------------------------------------------------------------------------
            fig, fig_tit = \
            cat_plot(creative_type_top_crew_df, 'creative_type', plot_type)
            #-----------------------------------------------------------------------------------------
            if not os.path.isdir(crew_dir + fig_tit +'/'):
                os.mkdir(crew_dir + fig_tit +'/')
            #-----------------------------------------------------------------------------------------
            f_name = crew_dir + fig_tit +'/'+ creative_type +'.jpeg'
            #-----------------------------------------------------------------------------------------
            fig.savefig(f_name, bbox_inches='tight')
            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 Catgegory Types** | **Recommended Categories** ***(In order of Importance)*** | **Recommended MPA Ratings for each Category** ***(In order of Importance for each Category)*** |
|---|---|---|
| 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 |
| | Action, Adventure, Comedy | PG-13 • R |
| Individual Genres | Sci-Fi | PG-13 |
| | Adventure | PG-13 • PG |
| | Action | PG-13 • R • PG |
| | Comedy | PG • PG-13 • R |
| | Drama | PG-13 • R |

| **OpusData Catgegory Types** | **Recommended Categories** ***(In order of Importance)*** | **Recommended MPA Ratings for each Category** ***(In order of Importance for each Category)*** |
|---|---|---|
| Genres | Adventure | PG • PG-13 |
| | Action | PG-13 • R |
| | Drama | PG-13 • R |
| Creative Types | Kids Fiction | PG • G |
| | Science Fiction | PG-13 • R |
| | Contemporary Fiction | PG-13 • R |

## Important Average Gross Revenue Categories

| **IMDb Catgegory Types** | **Recommended Categories** ***(In order of Importance)*** | **Recommended MPA Ratings for each Category** ***(In order of Importance for each Category)*** |
|---|---|---|
| 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 |
| | Adventure, Family, Fantasy | PG-13 • PG |
| Individual Genres | Sci-Fi | PG-13 • R |
| | Animation | G • PG • R |
| | Adventure | PG-13 • G • PG • R |
| | Fantasy | PG-13 • PG • R |
| | Action | PG-13 • PG • R |

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

| OpusData Catgegory Types | Recommended Categories *(In order of Importance)* | Recommended MPA Ratings for each Category *(In order of Importance for each Category)* |
|---|---|---|
| Genres | Action | PG-13 • R |
| | Adventure | PG • PG-13 |
| | Drama | PG-13 • R |
| Creative Types | Super Hero | PG-13 |
| | Kids Fiction | PG • G |
| | Contemporary Fiction | PG-13 • R |

## Important Average Profits Categories

| IMDb Catgegory Types | Recommended Categories *(In order of Importance)* | Recommended MPA Ratings for each Category *(In order of Importance for each Category)* |
|---|---|---|
| Unique Genre Combinations | Adventure, Drama, Fantasy | PG-13 • PG • R |
| | Adventure, Fantasy | PG-13 |
| | Animation, Comedy, Family | PG |
| | Drama, Sci-Fi, Thriller | PG-13 |
| | Action, Adventure, Animation | PG |
| | Adventure, Family, Fantasy | PG-13 • PG |
| | Adventure, Animation, Comedy | PG • G • R |
| | Action, Adventure, Sci-Fi | PG-13 |
| Individual Genres | Animation | G • PG • R |
| | Sci-Fi | PG-13 • R |
| | Adventure | PG-13 • G • PG • R |
| | Fantasy | PG-13 • PG • R |
| | Action | PG-13 • PG • R |

| OpusData Catgegory Types | Recommended Categories *(In order of Importance)* | Recommended MPA Ratings for each Category *(In order of Importance for each Category)* |
|---|---|---|
| Genres | Adventure | PG-13 • G • PG • R |
| | Action | PG-13 • PG • R |
| | Musical | PG • PG-13 |
| Creative Types | Super Hero | R • PG-13 • PG |
| | Kids Fiction | G • PG |
| | Science Fiction | 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]:

```
# Creating interactive dropdown menus that would allow users to access any of the vizualisations
# created for any of the important genres or genre combinations with ease
#########################################################################################################
data_choices = ['Choose a data source from the choices below...',
                'IMDb', 'OpusData']
#--------------------------------------------------------------------------------------------------------
imdb_choices = ['Choose a category type from the choices below...',
                'Important Unique Genre Combinations', 'Important Individual Genres']
#--------------------------------------------------------------------------------------------------------
od_choices = ['Choose a category type from the choices below...',
              'Important Genres', 'Important Creative Types']
#--------------------------------------------------------------------------------------------------------
genre_choices = ['Important Unique Genre Combinations', 'Important Individual Genres',
                 'Important Genres']
#--------------------------------------------------------------------------------------------------------
ct_choices = ['Important Creative Types']
#--------------------------------------------------------------------------------------------------------
metric_choices = ['Choose a financial metric from the choices below...',
                  'Total Worldwide Gross', 'Total Worldwide Profits',
                  'Average Worldwide Gross','Average Worldwide Profits']
#--------------------------------------------------------------------------------------------------------
crew_choices = ['Choose a type of crew member from the choices below...',
                'Directors', 'Writers', 'Actors', 'Actresses']
#--------------------------------------------------------------------------------------------------------
empty_slot = ['N/A unitl a selection is made in dropdown menu above']
#--------------------------------------------------------------------------------------------------------
menu_layout = widgets.Layout(width='99%', height='30px')
#--------------------------------------------------------------------------------------------------------
first_menu = \
widgets.Dropdown(options=data_choices, value=data_choices[0], disabled=False, layout=menu_layout)
#--------------------------------------------------------------------------------------------------------
second_menu = \
widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False,layout=menu_layout)
#--------------------------------------------------------------------------------------------------------
third_menu = \
widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)
#--------------------------------------------------------------------------------------------------------
fourth_menu = \
widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)
#--------------------------------------------------------------------------------------------------------
fifth_menu = \
widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)
#--------------------------------------------------------------------------------------------------------
sixth_menu = \
widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=menu_layout)

# Function monitoring changes to the widgets
#########################################################################################################
def f(a, b, c, d, e, f):

    # First Menu
    #####################################################################################################
    if a == data_choices[1]:
        second_menu.options = imdb_choices
        first_menu.disabled = True
    #--------------------------------------------------------------------------------------------------
    elif a == data_choices[2]:
        second_menu.options = od_choices
        first_menu.disabled = True
    #--------------------------------------------------------------------------------------------------
    else: second_menu.options = empty_slot

    # Second Menu
    #####################################################################################################
    third_choice_1_a = 'Do you wish to explore the '
    third_choice_1_b = ' themselves, or the Important Crew from those '
    third_choice_1_c = '?'
    #--------------------------------------------------------------------------------------------------
    if b in genre_choices or b in ct_choices:
        third_choice_insert = b
```

```
        #-------------------------------------------------------------------------------------------
        third_choice_1 = third_choice_1_a + third_choice_insert + third_choice_1_b +\
                         third_choice_insert + third_choice_1_c
        #-------------------------------------------------------------------------------------------
        third_choices = [third_choice_1, third_choice_insert, third_choice_insert + ' - Crew']
        #-------------------------------------------------------------------------------------------
        third_menu.options = third_choices
        second_menu.disabled = True
    #-----------------------------------------------------------------------------------------------
    else:
        third_menu.options = empty_slot
        third_choices = ['empty']

    # Third Menu
    ###############################################################################################
    if c in third_choices:
        if c == third_choices[1]:
            f_path_p1 = third_choices[1] + '/'
            #-------------------------------------------------------------------------------------------
            fourth_menu.options = metric_choices
            third_menu.disabled = True
        if c == third_choices[2]:
            f_path_p1 = third_choices[2] + '/'
            #-------------------------------------------------------------------------------------------
            fourth_menu.options = crew_choices
            third_menu.disabled = True
    #-----------------------------------------------------------------------------------------------
    else: fourth_menu.options = empty_slot


    # Fourth Menu
    ###############################################################################################
    if d in metric_choices[1:]:
        file_path = 'visuals/' + f_path_p1 + d + '/'
        file_choices = listdir(file_path)
        #-------------------------------------------------------------------------------------------
        file_choices.insert(0, 'Please choose a file from the choices below...')
        #-------------------------------------------------------------------------------------------
        fifth_menu.options = file_choices
        fourth_menu.disabled = True
    #-----------------------------------------------------------------------------------------------
    elif d in crew_choices[1:]:
        f_path_p2 = d + '/'
        #-------------------------------------------------------------------------------------------
        fifth_menu.options = metric_choices
        fourth_menu.disabled = True
    #-----------------------------------------------------------------------------------------------
    else: fifth_menu.options = empty_slot


    # Fifth Menu
    ###############################################################################################
    if '.jpeg' in e:
        file_name = file_path + e
        #-------------------------------------------------------------------------------------------
        image = Image.open(file_name)
        display(image)
        #-------------------------------------------------------------------------------------------
        sixth_menu.disabled = True
    #-----------------------------------------------------------------------------------------------
    elif e in metric_choices[1:]:
        file_path = 'visuals/' + f_path_p1 + f_path_p2 + e + '/'
        file_choices = listdir(file_path)
        #-------------------------------------------------------------------------------------------
        file_choices.insert(0, 'Please choose a file from the choices below...')
        #-------------------------------------------------------------------------------------------
        sixth_menu.options = file_choices
        fifth_menu.disabled = True
    #-----------------------------------------------------------------------------------------------
    else: sixth_menu.options = empty_slot

```

```
    # Sixth Menu
    ##########################################################################################################
    if '.jpeg' in f:
        file_name = file_path + f
        #-------------------------------------------------------------------------------------------
        image = Image.open(file_name)
        display(image)

# Creaing the interactive widget
##########################################################################################################
ui = widgets.VBox([first_menu, second_menu, third_menu, fourth_menu, fifth_menu, sixth_menu])
#----------------------------------------------------------------------------------------------------------
out = widgets.interactive_output(f, {'a':first_menu, 'b':second_menu, 'c':third_menu,
                                     'd':fourth_menu, 'e':fifth_menu, 'f':sixth_menu})
#----------------------------------------------------------------------------------------------------------
print('\n\033[1m    Follow the instructions that appear in each dropdown menu to access the '+\
      'visualization with the desired properties.\n\n    To start a new search, '+\
      'just re-run the cell.\n')
display(ui, out)
```

**Follow the instructions that appear in each dropdown menu to access the visualization with the des ired properties.**

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

OpusData

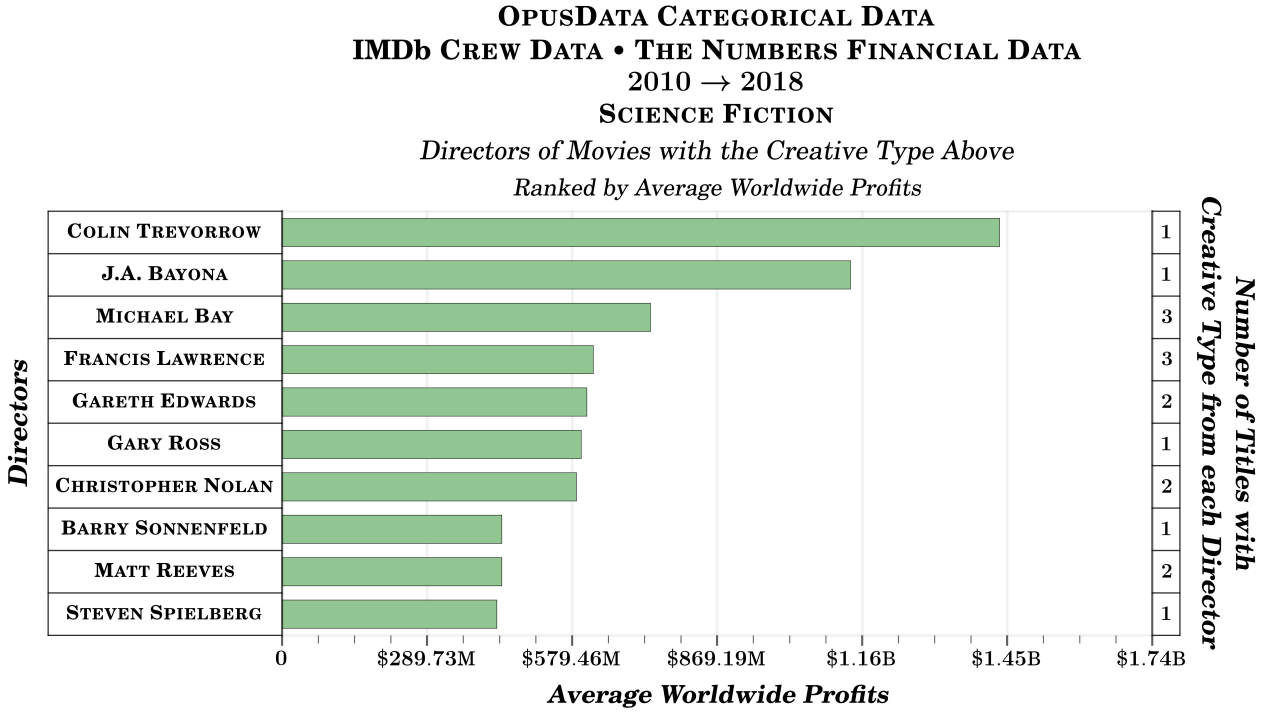
Important Creative Types

Important Creative Types - Crew

Directors

Average Worldwide Profits

Science Fiction.jpeg




# Closing Database Connection

In [46]:

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

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

In [47]:

```
# Printing information about the category types to be copied into my presentation
#########################################################################################################
imdb_source = 'IMDb Category Types'
od_source = 'OpusData Category Types'
#---------------------------------------------------------------------------------------------------------
g_combos_cat = 'Unique Genre Combinations'
individ_gs_cat = 'Individual Genres'
genres_cat = 'Genres'
ct_cat = 'Creative Type'
#---------------------------------------------------------------------------------------------------------
imdb_cat_types = [g_combos_cat, individ_gs_cat]
od_cat_types = [genres_cat, ct_cat]
#---------------------------------------------------------------------------------------------------------
num_combos = ['Number of Different Combinations: '+ str(len(fin_df.genres.unique()))]
od_genres = list(filter(None, fin_df.genre.unique()))
od_cts = list(filter(None, fin_df.creative_type.unique()))
#---------------------------------------------------------------------------------------------------------
imdb_cat_lists = [num_combos, unique_genres]
od_cat_lists = [od_genres, od_cts]
#---------------------------------------------------------------------------------------------------------
def print_categories(data_source, cat_types, cat_lists):
    print_out = '\033[1m'+ data_source +'\033[0m\n\n'
    #-----------------------------------------------------------------------------------------------------
    for cat_type, cat_list in zip(cat_types, cat_lists):
        print_out = print_out +'\033[1m'+ cat_type +'\033[0m\n'
        #-------------------------------------------------------------------------------------------------
        for cat in cat_list:
            print_out = print_out + cat +'\n'
        #-------------------------------------------------------------------------------------------------
        print_out = print_out +'\n'
        #-------------------------------------------------------------------------------------------------
    print_out = print_out +'\n'
    #-----------------------------------------------------------------------------------------------------
    print(print_out)
#---------------------------------------------------------------------------------------------------------
print_categories(imdb_source, imdb_cat_types, imdb_cat_lists)
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

In [48]:
```
1 # This section is in the README.md file only
```

## END OF PROJECT