§ 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 which types of movies perform best at the box office, which would aslo provide insights into the viewing preferences of moviegoers, and then produce movies and/or television shows based on the insights gained from such an analysis.

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

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

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

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

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

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

- **Crew Member Types**
  - Directors
  - Writers
  - Actors
  - Actresses

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

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

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

§ Exploring and Preparing the Data

For this project I used data provided to me from IMDb (https://www.imdb.com/) and The Numbers (https://www.the-numbers.com/), by Flatiron School (https://flatironschool.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, for which it provides data services.

Importing and Investigating the Data

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

In [1]:
```
import os
from os import listdir
import string
import numpy as np
import pandas as pd
import sqlite3
#-------------------------------------------------------------------------------
from pandas.io.formats.style import Styler
#-------------------------------------------------------------------------------
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
from IPython.display import clear_output
#-------------------------------------------------------------------------------
%matplotlib inline
```

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

In [2]:
```
# Flatiron School provided data from IMDb and The Numbers
################################################################################
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]:
```
# OpusData data
################################################################################
tn_extra = pd.read_csv('data/Opus Data/MovieData.csv')
```

In [4]:

```
df_names = ['imdb_title_principals', 'imdb_title_basics', 'imdb_name_basic', 'tn_mo
            'tn_extra']
#-------------------------------------------------------------------------------
all_dfs = [imdb_title_principals, imdb_title_basics, imdb_name_basic, tn_movie_budg
#-------------------------------------------------------------------------------
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 
---  ------      --------------    ----- 
 0   tconst      1028186 non-null  object
```

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

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

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

Preparing the Data for Analysis

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

In [5]:

```
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())
```

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
```

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

In [6]:
```
foreign_gross = tn_movie_budgets['worldwide_gross'] - tn_movie_budgets['domestic_gr
#-------------------------------------------------------------------------------
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())
```

3	4	May 1, 2015	Avengers: Age of Ultron	330600000.0	459005868.0	9.440081e+08	1.403014e+09
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000.0	620181382.0	6.965404e+08	1.316722e+09

```
<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
```

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

In [7]:
```
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
print(len(tn_movie_budgets))
```

```
0   2009-12-18
1   2011-05-20
2   2019-06-07
Name: release_date, dtype: datetime64[ns]

1915 2020
5782
```

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

In [8]:
```
tn_cols = [tn_extra.columns[2]] + list(tn_extra.columns[:2]) + list(tn_extra.columr
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_off
0	8220100	Madea's Family Reunion	2006	10000000	63257940	62
1	58540100	Krrish	2006	10000000	1430721	31000
2	34620100	End of the Spear	2006	10000000	11748661	175

Limiting the Year Range

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

In [9]:
```
imdb_title_basics = imdb_title_basics.loc[(imdb_title_basics.start_year > 2009) & \
                                          (imdb_title_basics.start_year < 2019)]
#-------------------------------------------------------------------------------
tn_movie_budgets = tn_movie_budgets.loc[(tn_movie_budgets.release_date.dt.year > 20
                                        (tn_movie_budgets.release_date.dt.year < 20
```

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]:

```
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_
                          (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_b
                '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())
```

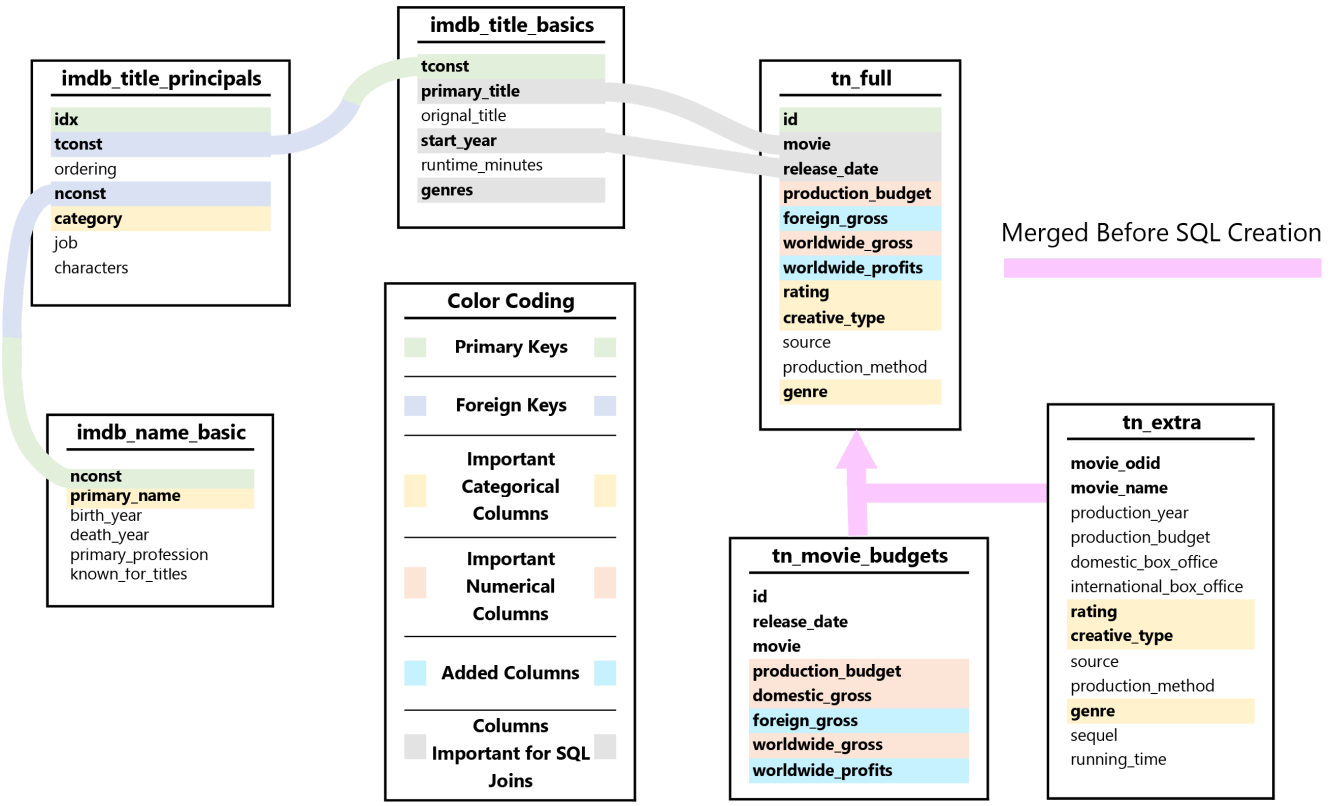
	id	movie	release_date	production_budget	domestic_gross	foreign_gross	worldwide_gross	w
2	48.0	10 Days in a Madhouse	2015-11-11	12000000.0	14616.0	0.0	14616.0	
5	64.0	12 Strong	2018-01-19	35000000.0	45819713.0	25298665.0	71118378.0	
6	18.0	12 Years a Slave	2013-10-18	20000000.0	56671993.0	124353350.0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

Turning the DataFrames into a SQL Database

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

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

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



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

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

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

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

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

In [13]:
```
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)
```

I then checked to make sure everything worked.

In [14]:
```
print(mov_cur.execute('SELECT name FROM sqlite_master').fetchall())
```

```
[('imdb_title_principals',), ('imdb_title_basics',), ('imdb_name_basic',), ('tn_ful
l',)]
```

In [15]:
```
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('----------------------------------------')
```

	idx	tconst	ordering	nconst	category	job	characters
1	1	tt0111414	2	nm0398271	director	None	None
2	2	tt0111414	3	nm3739909	producer	producer	None

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028186 entries, 0 to 1028185
Data columns (total 7 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   idx         1028186 non-null  int64
 1   tconst      1028186 non-null  object
 2   ordering    1028186 non-null  int64
 3   nconst      1028186 non-null  object
 4   category    1028186 non-null  object
 5   job         177684 non-null   object
 6   characters  393360 non-null   object
dtypes: int64(2), object(5)
memory usage: 54.9+ MB
```

§ Creating the Financial Analysis DataFrames

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

Main Financial DataFrame

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

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

In [16]:

```
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=Fal
#-------------------------------------------------------------------------------
display(fin_df.info())
```

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

In [17]:

```
dup_titles = \
fin_df.loc[fin_df.duplicated(['start_year', 'primary_title', 'worldwide_gross'], ke
      .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))
```

```
Int64Index: 975 entries, 0 to 1032
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   tconst             975 non-null    object
 1   start_year         975 non-null    int64
 2   primary_title      975 non-null    object
 3   genre              975 non-null    object
 4   genres             975 non-null    object
 5   rating             975 non-null    object
 6   creative_type      972 non-null    object
 7   production_budget  975 non-null    float64
 8   worldwide_gross    975 non-null    float64
 9   worldwide_profits  975 non-null    float64
dtypes: float64(3), int64(1), object(6)
memory usage: 83.8+ KB

None

Movies dropped:  60
```

Main Financial DataFrame Details

I created tables in markdown format to be displayed in the README file on GitHub, and later in HTML for the notebook markdown on GitHub, that show the details of the main category type and each subcategory type. To do so, I first created functions to format the numerical values accordingly.

In [18]:

```
def markdown_table_currency(val):
    dollar_start = '&dollar; '
    #-------------------------------------------------------------------------------
    if val >= 1e9 or val <= -1e9:
        v = dollar_start + '{:.2f} B'.format(abs(val*1e-9))
    if (1e6 <= val < 1e9) or (-1e6 >= val > -1e9):
        v = dollar_start + '{:.2f} M'.format(abs(val*1e-6))
    if (1e3 <= val < 1e6) or (-1e3 >= val > -1e6):
        v = dollar_start + '{:.2f} Th'.format(abs(val*1e-3))
    #-------------------------------------------------------------------------------
    if 0 < val < 1e3: v = dollar_start + '{:.2f}'.format(abs(val))
    if val == 0: v = 0
    if val < 0: v = '- '+ v
    if np.isnan(val): v = ''
    #-------------------------------------------------------------------------------
    return v
```

In [19]:

```
def notebook_table_currency(val):
    dollar_start = '&#36; '
    #-------------------------------------------------------------------------------
    if val >= 1e9 or val <= -1e9:
        v = dollar_start + '{:.2f} B'.format(abs(val*1e-9))
    if (1e6 <= val < 1e9) or (-1e6 >= val > -1e9):
        v = dollar_start + '{:.2f} M'.format(abs(val*1e-6))
    if (1e3 <= val < 1e6) or (-1e3 >= val > -1e6):
        v = dollar_start + '{:.2f} Th'.format(abs(val*1e-3))
    #-------------------------------------------------------------------------------
    if 0 < val < 1e3: v = dollar_start + '{:.2f}'.format(abs(val))
    if val == 0: v = 0
    if val < 0: v = '- '+ v
    if np.isnan(val): v = ''
    #-------------------------------------------------------------------------------
    return v
```

In [20]:

```
md_table_start = '| '
md_table_mid = ' | '
md_table_end = ' |\n'
#-------------------------------------------------------------------------------
cat_cols = ['rating', 'genre', 'creative_type', ['genre', 'creative_type']]
#-------------------------------------------------------------------------------
main_header_str = '| Main Category Type | Number of Categories |\n'
header_str = '| Subcategory Types | Number of Subcategories |\n'
#-------------------------------------------------------------------------------
align_cols = '| :-: | :-: |\n'
#-------------------------------------------------------------------------------
for c_i, cat_col in enumerate(cat_cols):
    if c_i in [0, 1]: cat_str = cat_col.title() + 's'
    if c_i == 2: cat_str = ' '.join(cat_col.split('_')).title() + 's'
    #---------------------------------------------------------------------------
    if c_i == 3:
        cat_str = 'Genre & Creative Type<br>Combinations'
        #-----------------------------------------------------------------------
        cat_col_1 = cat_col[0]
        cat_col_2 = cat_col[1]
        #-----------------------------------------------------------------------
        cat_df = fin_df[[cat_col_1, cat_col_2]].copy()
        cat_df['combos'] = cat_df[cat_col_1] +', '+ cat_df[cat_col_2]
    #---------------------------------------------------------------------------
    else:
        value_df = fin_df[cat_col].value_counts()
        #-----------------------------------------------------------------------
        value_df.name = cat_str
        #-----------------------------------------------------------------------
        value_df = value_df.reset_index().rename({cat_str: 'Count', 'index': cat_st
                           .set_index(cat_str)
        #-----------------------------------------------------------------------
        all_allign_cols = ['center'] * (len(value_df.columns) + 1)

# Not Necessary for Actual Analysis - Uncomment to print when Needed
###############################################################################
#         print(value_df.to_markdown(tablefmt='pipe', colalign=all_allign_cols))
#         print('\n\n\n')

    cat_num = str(fin_df[cat_col].nunique())
    if c_i == 3: cat_num = str(cat_df.combos.nunique())
    #---------------------------------------------------------------------------
    md_table_str = md_table_start + cat_str + md_table_mid + cat_num + md_table_end
    #---------------------------------------------------------------------------
    if c_i == 0:
        table = main_header_str + align_cols + md_table_str +'\n'

# Not Necessary for Actual Analysis - Uncomment to print when Needed
###############################################################################
#         print(table)

    if c_i == 1: table = header_str + align_cols + md_table_str
    if c_i > 1: table = table + md_table_str

# Not Necessary for Actual Analysis - Uncomment to print when Needed
###############################################################################
# print(table)
```

In [21]:

```
for cat_col in cat_cols[:-1]:
    for col in ['worldwide_gross', 'worldwide_profits']:
        desc_df = fin_df.groupby(cat_col).describe()[[col]]
        #-------------------------------------------------------------------------
        temp_df_p1 = desc_df.iloc[:, :1].astype(int)
        temp_df_p2 = desc_df.iloc[:, 1:].applymap(lambda x: markdown_table_currency
        #-------------------------------------------------------------------------
        md_df = pd.concat([temp_df_p1, temp_df_p2], axis=1)
        #-------------------------------------------------------------------------
        col_tit = ' '.join(col.split('_')).title()
        #-------------------------------------------------------------------------
        md_df.index.name = col_tit +'<br><br>'+ md_df.index.name.title()
        md_df = md_df.rename(columns={col: col_tit}, level=0)
        #-------------------------------------------------------------------------
        md_df.columns = md_df.columns.to_flat_index()
        #-------------------------------------------------------------------------
        md_df.columns = [col[1] for col in md_df.columns]
        #-------------------------------------------------------------------------
        all_allign_cols = ['center'] * (len(md_df.columns) + 1)

# Not Necessary for Actual Analysis - Uncomment to print when Needed
##############################################################################
#         print(md_df.to_markdown(tablefmt='pipe', colalign=all_allign_cols))
#         print('\n\n\n')
```

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

Main Category Type	Number of Categories
Ratings	5

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

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

Click here to see more details for the Rating category type and Genre and Creative Type subcategory types in the README file.
(https://github.com/sarnadpy32/microsoft_productions#extra_details)

Crew Financial DataFrames

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

```
"""
SELECT
    tconst, nconst, primary_name, start_year, primary_title, genre, genres, ra
ting,
    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 [22]:

```
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, rat
        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 d
    ###############################################################################
    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('-----------------------------------------')

    # Changing each df name to an appropriate one
    ###############################################################################
    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]
```

```
Int64Index: 1095 entries, 0 to 1094
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   tconst             1095 non-null   object
 1   nconst             1095 non-null   object
 2   primary_name       1095 non-null   object
 3   start_year         1095 non-null   int64
 4   primary_title      1095 non-null   object
 5   genre              1095 non-null   object
 6   genres             1095 non-null   object
 7   rating             1095 non-null   object
 8   creative_type      1092 non-null   object
 9   production_budget  1095 non-null   float64
 10  worldwide_gross    1095 non-null   float64
 11  worldwide_profits  1095 non-null   float64
dtypes: float64(3), int64(1), object(8)
memory usage: 111.2+ KB

None
```

Crew Financial DataFrames Details

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

In [23]:
```
md_table_start = '| '
md_table_mid = ' | '
md_table_end = ' |\n'
#-------------------------------------------------------------------------------
align_cols = '| :-: | :-: | :-: |\n'
#-------------------------------------------------------------------------------
type_head = 'Crew Member Types'
mov_head = 'Number of Movies<br>with data for each<br>Crew Member Type'
crew_head = 'Number of Crew Members<br>for each<br>Crew Member Type'
#-------------------------------------------------------------------------------
header_str = \
md_table_start + type_head + md_table_mid + mov_head + md_table_mid + crew_head + m
#-------------------------------------------------------------------------------
for c_df, crew_str in zip(crew_dfs, crew_strings):
    crew_orig = crew_str
    #---------------------------------------------------------------------------
    if crew_str == 'actress': crew_tit = crew_str + 'es'
    else: crew_tit = crew_str + 's'
    #---------------------------------------------------------------------------
    crew_tit = crew_tit.title()
    #---------------------------------------------------------------------------
    num_titles = str(c_df.tconst.nunique())
    num_crew = str(c_df.nconst.nunique())
    #---------------------------------------------------------------------------
    md_table_str = md_table_start + crew_tit + md_table_mid + num_titles + md_table
                   num_crew + md_table_end
    #---------------------------------------------------------------------------
    if crew_orig == crew_strings[0]: table = header_str + align_cols + md_table_str
    else: table = table + md_table_str

# Not Necessary for Actual Analysis - Uncomment to print when Needed
################################################################################
# print(table)
```

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

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

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

In [24]:
```
mov_conn.close()
```

§ Visualization Functions

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

Modifying Key rcParams

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

In [25]:
```
light_spine_sides = ['top', 'bottom']
dark_spine_sides = ['left', 'right']

# I had to include the translation package b/c one director's name had a non-recogn
# 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{stix}'+\
                                            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.

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

In [26]:
```
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 their corresponding visualizations.

In [27]:
```
uniq_ratings = tn_full.rating.unique()
#-------------------------------------------------------------------------------
ratings_colors_dict = {'Not Rated': 'palegreen',
                       'G': 'orchid',
                       'PG': 'dodgerblue',
                       'PG-13': 'forestgreen',
                       'R': 'tomato'}
```

Tick Params Functions

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

In [28]:
```
# Function for setting Tick Params for axes with numerical data on X-Axis and categ
# the Y-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
    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
    #---------------------------------------------------------------------------
    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 [29]:
```
# Function for setting Tick Params for axes with a twinned numerical X-Axis on the
##############################################################################
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
    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 [30]:
```
# Function for setting Tick Params for axes with a twinned categorical Y-Axis on th
##############################################################################
def right_y_tick_params(r_ax):
    maj_reg_kw = dict(length=0, width=2.4, labelsize=24, labelcolor='k', color=(0,0
    #-------------------------------------------------------------------------
    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 [31]:

```
# Function to format currency ticks on the X Axis
#########################################################################################
def x_currency_formatter(x, pos):
    x_prime = x.copy()
    s_start = r'{\fontsize{25.5}{0}\selectfont{\scshape{\$}}}\,'
    #-----------------------------------------------------------------------------------
    if x>=1e9 or x<=-1e9:
        s_end = r'{\fontsize{24}{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{24}{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{24}{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 [32]:

```
# 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
    #-----------------------------------------------------------------------------------
    else:  s = r'$\mathbf{'+'{:.2f}'.format(x)+'}$\,' + s_end
    #-----------------------------------------------------------------------------------
    if x_prime==0: s = r'{$\mathbf{'+str(0)+'}$'

    return s
```

In [33]:
```
# Function to format simple numerical tick labels in LaTeX format for use on ticks
# 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 Quantile & Tick Finders

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

In [34]:
```
# Function to find positive and/or negative quantiles to be matched for a twinned a
# passed into the max tick finder
#################################################################################
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 + .06
        else: test_quant = test_quant + .06 + .06 + .06
        i += 1
    #-----------------------------------------------------------------------------
    return test_value, test_quant, last_neg
```

In [35]:
```
# 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 [36]:
```
# 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 + max_step
        else: test_v = test_v + max_step + max_step + max_step
        i += 1
    #-------------------------------------------------------------------------------------
    return final_max
```

Main Visualization Function

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

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

In [37]:

```
def rat_cat_plot(plot_df, plot_type, box=None):

    # Starting variables
    ##########################################################################
    plot_len = len(plot_df)
    plot_height = 9
    #-------------------------------------------------------------------------
    base_x_formatter = x_currency_formatter
    #-------------------------------------------------------------------------
    rat_insert = plot_df.rating[0]
    #-------------------------------------------------------------------------
    tit_bullet = r'{\fontsize{28.5}{0}\selectfont{ \textbullet\ }}'
    tit_start = r'{\fontsize{30}{0}\selectfont{\textrm{'

    # crew_plot, main_cat_col
    ##########################################################################
    if plot_type in ['directors', 'writers', 'actors', 'actresses']: crew_plot = T
    else: crew_plot = False
    #-------------------------------------------------------------------------
    if plot_type in ['creative_type', 'rating']: main_cat_col = plot_type
    elif crew_plot: main_cat_col = 'primary_name'
    else: main_cat_col = 'genres'

    # plot_col, cat_values, cat_percs, cat_freqs and parts of title strs
    ##########################################################################
    if 'avg' in plot_df.columns[-1] or box: plot_col = plot_df.columns[-1]
    else: plot_col = plot_df.columns[-2]
    #-------------------------------------------------------------------------
    if 'gross' in plot_col: lab_add = 'Worldwide Gross'
    if 'profits' in plot_col: lab_add = 'Worldwide Profits'
    #-------------------------------------------------------------------------
    cat_values = plot_df[plot_col]
    #-------------------------------------------------------------------------
    if not box: cat_freqs = plot_df['num_titles']
    #-------------------------------------------------------------------------
    value_step, v_q_used, v_last_neg_q = find_pos_quantile(cat_values)
    #-------------------------------------------------------------------------
    neg_value_step = cat_values.quantile(v_last_neg_q)
    #-------------------------------------------------------------------------
    if box or 'avg' in plot_col:
        perc_plot = False
        prime_lab_p1 = 'Average '
        #---------------------------------------------------------------------
        if box:
            box_add = 'Boxplots of their '
            if 'tot' in box: prime_lab_p1 = 'Total '
    #-------------------------------------------------------------------------
    if (crew_plot or not box) and 'tot' in plot_col:
        perc_plot = True
        prime_lab_p1 = 'Total '
        t_lab_p1 = 'Percentage of Total '
        t_lab = t_lab_p1 + lab_add
        #---------------------------------------------------------------------
        perc_col = plot_df.columns[-1]
        #---------------------------------------------------------------------
        cat_percs = plot_df[perc_col]
        #---------------------------------------------------------------------
        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 box and p_last_neg_q != v_last_neg_q:
            if p_last_neg_q < v_last_neg_q: neg_value_step = cat_values.quantile(p_
            else: neg_perc_step = cat_percs.quantile(v_last_neg_q)
    #-----------------------------------------------------------------------------
    prime_lab = prime_lab_p1 + lab_add
    #-----------------------------------------------------------------------------
    if box:
        top_lab = box_add + lab_add
        #-------------------------------------------------------------------------
        rank_add = 'Average '+ lab_add
    #-----------------------------------------------------------------------------
    else: rank_add = prime_lab
    #-----------------------------------------------------------------------------
    tit_based_on_add = 'Based on '+ prime_lab

    # y_labels, y_lab, tex_y_labels, r_lab parts and title strs
    ###############################################################################
    if crew_plot or not box:
        y_labels = plot_df[main_cat_col]
        #-------------------------------------------------------------------------
        tit_start_add = 'Top }}}' + r'{\fontsize{31.5}{0}\selectfont{$\mathrm{' +
                        str(plot_len) +'}$\ }}'
    #-----------------------------------------------------------------------------
    if box:
        y_labels = plot_df[main_cat_col].unique()
        #-------------------------------------------------------------------------
        tit_start_add = 'Top }}}' + r'{\fontsize{31.5}{0}\selectfont{$\mathrm{'+ \
                        str(plot_df[main_cat_col].nunique()) +'}$\ }}'
    #-----------------------------------------------------------------------------
    tex_y_labels = [r'\textbf{\textsc{'+ lab +'}}' for lab in y_labels]
    #-----------------------------------------------------------------------------
    if plot_type=='genres': y_lab = 'Genres'
    #-----------------------------------------------------------------------------
    if plot_type=='creative_type': y_lab = 'Creative Types'
    #-----------------------------------------------------------------------------
    if plot_type=='rating': y_lab = 'MPA Ratings'
    #-----------------------------------------------------------------------------
    if plot_type=='combos':
        y_lab = 'Genre \& Creative Type Combinations'
        #-------------------------------------------------------------------------
        tex_y_labels = \
        [r'\textbf{\textsc{'+ lab.split(',')[0] +'}}'+ '\n' +\
         r'{\fontsize{22}{0}\selectfont{\textbf{\textsc{\&}}'+ '\n' + \
         r'\textbf{\textsc{'+ lab.split(',')[1] +'}}' for lab in y_labels]
    #-----------------------------------------------------------------------------
    if crew_plot:
        y_lab = plot_type.title()
        #-------------------------------------------------------------------------
        r_lab_1_p1 = 'Number of '+ rat_insert +' '
        r_lab_1_p2 = r'\textbf{\textsl{Films '
        #-------------------------------------------------------------------------
        tit_p3_p1 = tit_start + tit_start_add + tit_start + y_lab +' of '+ rat_ins
        tit_p3_end = ' Films'
        #-------------------------------------------------------------------------
```

```
        if ',' in plot_df.index[0]: sub_cat_insert = ' \& '.join(plot_df.index[0].
        #-------------------------------------------------------------------------
        else: sub_cat_insert = plot_df.index[0]
        #-------------------------------------------------------------------------
        r_lab_1 = r_lab_1_p1 + sub_cat_insert +'}}' +'\n'+ r_lab_1_p2
        #-------------------------------------------------------------------------
        tit_p3_p1 = tit_p3_p1 + sub_cat_insert + tit_p3_end
        #-------------------------------------------------------------------------
        if plot_type in ['directors', 'writers']: r_lab_1_add = 'from each '
        else: r_lab_1_add = 'with each '
        #-------------------------------------------------------------------------
        if plot_type == 'actresses': r_lab_1_add2 = y_lab[:-2]
        else: r_lab_1_add2 = y_lab[:-1]
        #-------------------------------------------------------------------------
        r_lab_1 = r_lab_1 + r_lab_1_add + r_lab_1_add2
        #-------------------------------------------------------------------------
        tit_p3_p2 = tit_start + 'Ranked by ' + rank_add +'}}}'
    #-----------------------------------------------------------------------------
    if plot_type == 'rating':
        r_lab_1 = 'Number of Films with each MPA Rating'
        #-------------------------------------------------------------------------
        tit_p3 = \
        tit_start +'MPA Ratings}}}'+ tit_bullet + tit_start +'Ranked by '+ rank_ad
        #-------------------------------------------------------------------------
        if box:
            tit_p3_p1 = tit_start + box_add[:-3] +' '+ lab_add +' of the MPA Ratin
    #-----------------------------------------------------------------------------
    else:
        if not crew_plot:
            r_lab_1 = 'Number of '+ plot_df.rating[0] +' Films with'
            r_lab_2 = 'Each '+ y_lab[:-1]
            #---------------------------------------------------------------------
            tit_p3_p1 = \
            tit_start + tit_start_add + tit_start + y_lab + ' for '+ rat_insert +'
            #---------------------------------------------------------------------
            tit_p3_p2 = tit_start + tit_based_on_add +'}}}'
        #-------------------------------------------------------------------------
        if plot_type == 'combos' or crew_plot:  tit_p3 = tit_p3_p1 +'\n'+ tit_p3_p
        #-------------------------------------------------------------------------
        else: tit_p3 = tit_p3_p1 + tit_bullet + tit_p3_p2
    #-----------------------------------------------------------------------------
    if box:
        tit_box_start = r'{\fontsize{28.5}{0}\selectfont{\textsl{'
        #-------------------------------------------------------------------------
        tit_box = tit_box_start + top_lab + tit_bullet +'Ranked by '+ rank_add +'}
        #-------------------------------------------------------------------------
        if plot_type == 'combos': tit_p3 = tit_p3_p1 +'\n'+ tit_p3_p2 +'\n'+ tit_b
        elif plot_type == 'rating':
            tit_p3 = tit_p3_p1 +'\n'+ tit_box_start +'Ranked by '+ rank_add +'}}}'
        else: tit_p3 = tit_p3_p1 + tit_bullet + tit_p3_p2 +'\n'+ tit_box

    # Setting other variables necesary for plotting
    ###############################################################################
    num_bars = 1
    bar_height = 3
    group_space = 4.5
    #-----------------------------------------------------------------------------
    if not box: y_locs = range(plot_len)
    else: y_locs = range(plot_df[main_cat_col].nunique())
```

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

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

```
    #-----------------------------------------------------------------------
    if not box:
        freq_labels = y_labels_to_norm_num(cat_freqs)
        #-----------------------------------------------------------------------
        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)
        right_y_tick_params(r_ax)
    #-----------------------------------------------------------------------
    else: h_tick_params(ax, True)
    #-----------------------------------------------------------------------
    if perc_plot: top_x_tick_params(t_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:
        if plot_type == 'rating' or crew_plot:
            r_ax.set_ylabel(r'\textbf{\textsl{'+ r_lab_1 +'}}', va='bottom',  rota
                            **y_lab_kw)
        #-----------------------------------------------------------------------
        else:
            r_ax.set_ylabel(r'\textbf{\textsl{'+ r_lab_1 +'}}'+'\n'+\
                            r'\textbf{\textsl{'+ r_lab_2 +'}}', va='bottom',  rota
                            **y_lab_kw)

    # Title
    ###########################################################################
    if perc_plot: tit_kw = dict(pad=36, linespacing=1.8)
    else: tit_kw = dict(pad=18, linespacing=1.8)
    #-----------------------------------------------------------------------
    rat_cat_source = r'{\fontsize{31.5}{0}\selectfont{\textbf{\textsc{OpusData}}}}
    #-----------------------------------------------------------------------
    if plot_type != 'rating':
        m_tit_p1 = \
        r'{\fontsize{31.5}{0}\selectfont{\textbf{\textsc{ Categorical Data}}}}'
    #-----------------------------------------------------------------------
    else: m_tit_p1 = r'{\fontsize{31.5}{0}\selectfont{\textbf{\textsc{ Ratings Dat
    #-----------------------------------------------------------------------
    year_tit = r'2010 \boldsymbol{\rightarrow} 2018'
    #-----------------------------------------------------------------------
    finan_data_source = \
    r'{\fontsize{31.5}{0}\selectfont{\textbf{\textsc{The Numbers Financial Data}}}
    #-----------------------------------------------------------------------
```

```
    year_tit = r'2010 \boldsymbol{\rightarrow} 2018'
    #-----------------------------------------------------------------------
    tit_p2 = r'{\fontsize{33}{0}\selectfont{$\mathbf{'+ year_tit +'}$}}'
    #-----------------------------------------------------------------------
    if not crew_plot:
        tit_p1 = rat_cat_source + m_tit_p1 +'\n'+ 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{\textsc{ '+ y_lab +'
        #-------------------------------------------------------------------
        tit_p1 = rat_cat_source + m_tit_p1 +'\n'+ \
                 crew_source + m_tit_crew + tit_bullet + finan_data_source
    #-----------------------------------------------------------------------
    ax.set_title(tit_p1 +'\n'+ tit_p2 +'\n'+ tit_p3, **tit_kw)

    # Legend
    ###########################################################################
    leg_loc = 'center left'
    #-----------------------------------------------------------------------
    leg_kw = dict(loc=leg_loc, bbox_to_anchor=(1, .5), fontsize=21, handlelength=3
    #-----------------------------------------------------------------------
    frame_kw = dict(lw=.9, ec='k', alpha=1)
    #-----------------------------------------------------------------------
    if box:
        if 'gross' in plot_col: leg = ax.legend(handles=[mean_lines, median_lines]
        #-------------------------------------------------------------------
        else:
            if final_min < 0:
                leg = ax.legend(handles=[zero_line, mean_lines, median_lines], **l
            #---------------------------------------------------------------
            else: leg = ax.legend(handles=[mean_lines, median_lines], **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_ymajortickla
            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_bbo
```

```
            #-------------------------------------------------------------------
            if r_lab_f_bbox_xymax[0] >= max_label_pos_x: max_label_pos_x = r_lab_f
        #-----------------------------------------------------------------------
        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_bbo
    #---------------------------------------------------------------------------
    neg_kw = dict(x=(max_label_neg_x/2) - .0045, ha='center')
    [label.update(neg_kw) for label in ax.get_ymajorticklabels()]
    #---------------------------------------------------------------------------
    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)
    #---------------------------------------------------------------------------
    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_t

    # 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
        #-------------------------------------------------------------------
        div_line = mlines.Line2D([div_left_x, div_right_x], [div_y, div_y], **c_ti
    #---------------------------------------------------------------------------
    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(rendere
    #---------------------------------------------------------------------------
    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_t
    if not box:
        right_vert = mlines.Line2D([[right_vert_x] * 2], [bot_vert_y, top_vert_y],
    #---------------------------------------------------------------------------
    if not box and 'tot' in plot_col:
        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 save accordingly if needed
    ################################################################################
    if not box: return fig, prime_lab
    else: return fig, top_lab
```

§ Standards for Qualifying as Important

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

Standards for Qualifying as an Important Subcategory

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

§ Creating & Plotting the Aggregate DataFrames

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

In [38]:
```
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 also created the dataframe to hold all of the ***Additional Insights*** section data to be processed later.

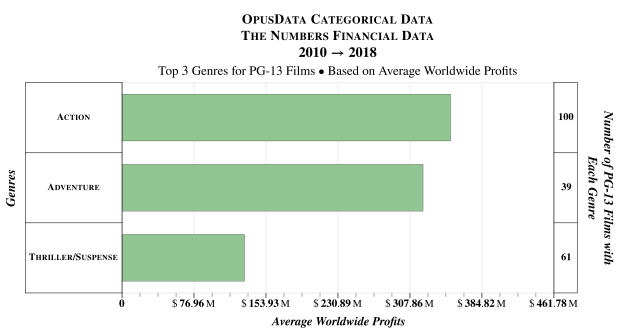
In [39]:
```
g_add_info_df = pd.DataFrame()
#-------------------------------------------------------------------------------
ct_add_info_df = pd.DataFrame()
#-------------------------------------------------------------------------------
combo_add_info_df = pd.DataFrame()
```

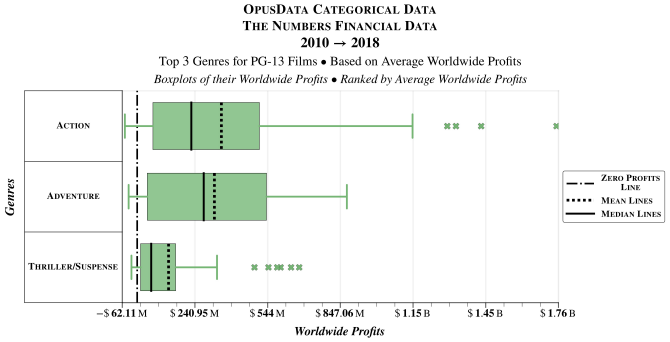
Financially Important Genres

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

Example Visualizations

(Right Click and Open Image in a New Tab)





In [40]:

```
base_dir = 'visuals/'
#-------------------------------------------------------------------------------
out_1 = widgets.Output()
display(out_1)
#-------------------------------------------------------------------------------
for a_i, aka_v in enumerate(all_akas):
    plot_col = aka_v[4:]
    plot_col_split = plot_col.split('_')
    #---------------------------------------------------------------------------
    g_count_df = fin_df.groupby(['rating', 'genre']).count()[['primary_title']]
    #---------------------------------------------------------------------------
    if 'tot' in aka_v:
        aka_start = 'Total'
        #-----------------------------------------------------------------------
        g_agg_df = fin_df.groupby(['rating', 'genre']).sum()[[plot_col]]
    #---------------------------------------------------------------------------
    if 'avg' in aka_v:
        aka_start = 'Average'
        #-----------------------------------------------------------------------
        g_agg_df = fin_df.groupby(['rating', 'genre']).mean()[[plot_col]]
    #---------------------------------------------------------------------------
    all_gs_df = pd.concat([g_count_df, g_agg_df], axis=1).reset_index()\
                  .sort_values(['rating', plot_col], ascending=[True, False])\
                  .rename(columns={'primary_title':'num_titles', 'genre':'genres',
                                   plot_col: aka_v})
    #---------------------------------------------------------------------------
    if 'tot' in aka_v:
        all_gs_df['perc_of_'+ aka_v] = round((all_gs_df[aka_v] / fin_df[plot_col].
    #---------------------------------------------------------------------------
    aka_str = ' '.join([aka_start] + plot_col_split).title()
    #---------------------------------------------------------------------------
    print('\033[1m\033[35mAgg Col: \033[0m', aka_str)
    print('-------------------------------------------------------')

    # Finding the Important Genres and the Genres for the Additional Information s
    ##############################################################################
    aka_add_info_df = pd.DataFrame()
    #---------------------------------------------------------------------------
    for rat in all_gs_df.rating.unique():
        if rat != 'Not Rated': rat_g_df = all_gs_df.loc[all_gs_df.rating == rat]
        else: continue
        #-------------------------------------------------------------------
        rat_copy_df = rat_g_df.copy()
        #-------------------------------------------------------------------
        rat_g_df = rat_g_df.loc[(rat_g_df.num_titles >= 9)].head(3)
        #-------------------------------------------------------------------
        rat_also_df = rat_copy_df.loc[(rat_copy_df.num_titles < 9) &
                                      (rat_copy_df[aka_v] >= rat_g_df[aka_v].max()
        #-------------------------------------------------------------------
        rat_also_df[aka_v] = rat_also_df[aka_v].map(lambda x: x - rat_g_df[aka_v].
        #-------------------------------------------------------------------
        if len(rat_also_df):
            rat_also_df = rat_also_df.set_index(['rating', 'genres', 'num_titles']
            #---------------------------------------------------------------
            aka_add_info_df = pd.concat([aka_add_info_df, rat_also_df])
            #---------------------------------------------------------------
            print('\033[1m\033[37mAdded data to the Additional Information datafra
                  '\033[31mRating\033[0m.')
```

```
            print('   \033[1m\033[31mRating: \033[0m', rat)
            print('-------------------------------------------------------')
        #-----------------------------------------------------------------------
        rat_g_df = rat_g_df.reset_index(drop=True)
        #-----------------------------------------------------------------------
        rat_dir = base_dir + rat +'/'
        rat_cat_dir = rat_dir + 'Important Genres/'
        rat_cat_metric_dir = rat_cat_dir + aka_str +'/'
        #-----------------------------------------------------------------------
        if not os.path.isdir(rat_dir):
            os.mkdir(rat_dir)
        #-----------------------------------------------------------------------
        if not os.path.isdir(rat_cat_dir):
            os.mkdir(rat_cat_dir)
        #-----------------------------------------------------------------------
        if not os.path.isdir(rat_cat_metric_dir):
            os.mkdir(rat_cat_metric_dir)
        #-----------------------------------------------------------------------
        with out_1:
            display(print('\033[1m\033[31mRating: \033[0m', rat), include=[str])
        #-----------------------------------------------------------------------
        if len(rat_g_df) == 1:
            print('\033[1m\033[37mThere was only one associated \033[32mGenre\033[(
                  '\033[1m\033[37mfor this \033[31mRating\033[0m.')
            print('   \033[1m\033[31mRating: \033[0m', rat)
            print('   \033[1m\033[32mGenre: \033[0m', rat_g_df.genres.values[0])
            print('-------------------------------------------------------')

        # Top Genres and their Boxplots Visualizations
        ###############################################################################
        if len(rat_g_df) > 1:
            fig, fig_tit = rat_cat_plot(rat_g_df, 'genres')
            #-------------------------------------------------------------------
            f_name = rat_cat_metric_dir +'1 - Top Genres.jpeg'
            #-------------------------------------------------------------------
            fig.savefig(f_name, bbox_inches='tight')
            plt.close(fig)
            #-------------------------------------------------------------------
            all_box_df = pd.DataFrame()
            import_order_dict = {}
            #-------------------------------------------------------------------
            for g in rat_g_df.genres:
                rat_g_box_df = fin_df.loc[(fin_df.rating == rat) & (fin_df.genre =
                #---------------------------------------------------------------
                rat_g_box_df = rat_g_box_df[['rating', 'genre', plot_col]]
                #---------------------------------------------------------------
                import_order_dict[g] = rat_g_box_df[plot_col].mean()
                #---------------------------------------------------------------
                all_box_df = pd.concat([all_box_df, rat_g_box_df])
            #-------------------------------------------------------------------
            import_order = \
            sorted(import_order_dict.keys(), key=lambda k: import_order_dict[k], r
            #-------------------------------------------------------------------
            all_box_df = \
            all_box_df.sort_values('genre',
                                   key=lambda g_col: g_col.map(lambda x: import_or
            #-------------------------------------------------------------------
            all_box_df = all_box_df.rename(columns={'genre': 'genres', plot_col: a
                                   .reset_index(drop=True)
```

```
            #-------------------------------------------------------------------
            fig, fig_tit = rat_cat_plot(all_box_df, 'genres', aka_v)
            #-------------------------------------------------------------------
            f_name = rat_cat_metric_dir +'2 - '+ fig_tit +'.jpeg'
            #-------------------------------------------------------------------
            fig.savefig(f_name, bbox_inches='tight')
            plt.close(fig)

        # Important Crew Visualizations
        ###############################################################################
        for g in rat_g_df.genres:
            with out_1:
                display(print('\033[1m\033[32mGenre: \033[0m', g), include=[str])
            #-------------------------------------------------------------------
            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 = rat_cat_metric_dir + plot_type.title() +'/'
                #-------------------------------------------------------------------
                if not os.path.isdir(crew_dir):
                    os.mkdir(crew_dir)
                #-------------------------------------------------------------------
                rat_g_crew_df = crew_df.loc[(crew_df.rating == rat) & (crew_df.gen
                #-------------------------------------------------------------------
                if rat_g_crew_df.primary_name.nunique() <= 1:
                    print('\033[1m\033[37mThere was only one crew member associate
                          '\033[32mGenre\033[37m for this \033[31mRating\033[0m.')
                    print('   \033[1m\033[31mRating: \033[0m'+ rat)
                    print('   \033[1m\033[32mGenre: \033[0m'+ g)
                    print('   \033[1m\033[33mCrew Type: \033[0m'+ crew_str.title()
                    print('   \033[1m\033[33mName: \033[0m',rat_g_crew_df.primary_
                    print('-----------------------------------------------------')
                    continue
                #-------------------------------------------------------------------
                else:
                    crew_count_df = \
                    rat_g_crew_df.groupby('primary_name').count()[['primary_title'
                    #-------------------------------------------------------------------
                    if 'tot' in aka_v:
                        crew_agg_df = rat_g_crew_df.groupby('primary_name').sum()[
                    #-------------------------------------------------------------------
                    if 'avg' in aka_v:
                        crew_agg_df = rat_g_crew_df.groupby('primary_name')\
                                                   .mean()[[plot_col]]
                    #-------------------------------------------------------------------
                    c_df = pd.concat([crew_count_df, crew_agg_df], axis=1).reset_i
                             .rename(columns={'primary_title':'num_titles', plot_c
                             .sort_values(aka_v, ascending=False).head(10)
                    #-------------------------------------------------------------------
                    genre_ser = c_df.primary_name.map(lambda x: g).rename('genres'
                    rat_ser = c_df.primary_name.map(lambda x: rat).rename('rating'
                    #-------------------------------------------------------------------
                    c_df.insert(0, 'genres', genre_ser)
                    c_df.insert(1, 'rating', rat_ser)
                    #-------------------------------------------------------------------
                    if 'tot' in aka_v:
                        c_df['perc_of_'+ aka_v] = \
                        round((c_df[aka_v] / fin_df[plot_col].sum()), 4)
```

```
                    #-------------------------------------------------------------
                    c_df = c_df.reset_index(drop=True).set_index('genres')
                    #-------------------------------------------------------------
                    fig, fig_tit = rat_cat_plot(c_df, plot_type)
                    #-------------------------------------------------------------
                    if '/' in g: g_dir = ' - '.join(g.split('/'))
                    else: g_dir = g
                    #-------------------------------------------------------------
                    f_name = crew_dir + g_dir +'.jpeg'
                    #-------------------------------------------------------------
                    fig.savefig(f_name, bbox_inches='tight')
                    plt.close(fig)
            #-----------------------------------------------------------------
            with out_1:
                display(print('\033[1m\033[32mGenre Finsished\033[0m'), include=[s

        # Clearing non-important information after each rating
        ##############################################################################
        with out_1:
            clear_output()
    print()

    # Adding the Additional Information section information
    ##############################################################################
    if not aka_add_info_df.empty:
        g_add_info_df = pd.concat([g_add_info_df, aka_add_info_df[[aka_v]]], axis=
```

```
Agg Col:  Total Worldwide Gross
-------------------------------------------------------
There was only one associated Genre for this Rating.
   Rating:  G
   Genre:  Adventure
-------------------------------------------------------

Agg Col:  Total Worldwide Profits
-------------------------------------------------------
There was only one associated Genre for this Rating.
   Rating:  G
   Genre:  Adventure
-------------------------------------------------------

Agg Col:  Average Worldwide Gross
-------------------------------------------------------
There was only one associated Genre for this Rating.
   Rating:  G
   Genre:  Adventure
-------------------------------------------------------
Added data to the Additional Information dataframe for this Rating.
   Rating:  R
-------------------------------------------------------

Agg Col:  Average Worldwide Profits
-------------------------------------------------------
There was only one associated Genre for this Rating.
   Rating:  G
   Genre:  Adventure
-------------------------------------------------------
Added data to the Additional Information dataframe for this Rating.
```

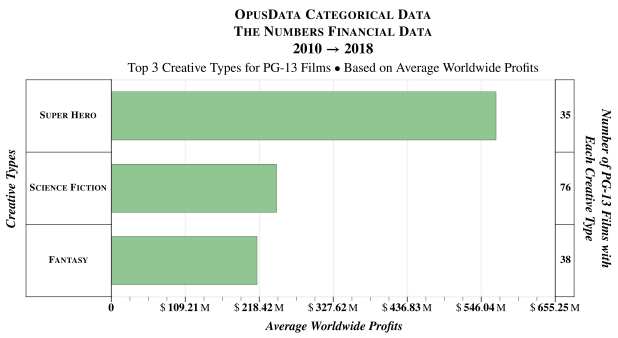
Rating: R

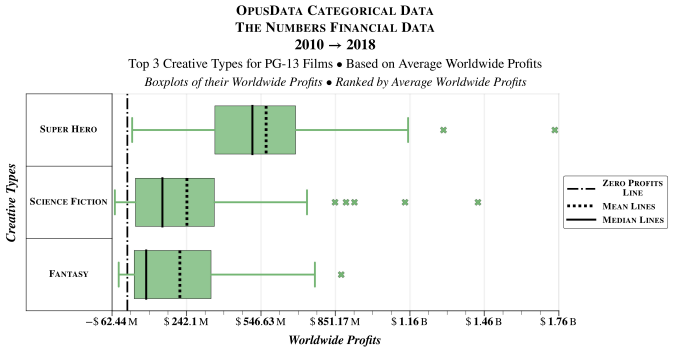
Financially Important Creative Types

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

Example Visualizations

(Right Click and Open Image in a New Tab)





In [41]:

```
base_dir = 'visuals/'
#-------------------------------------------------------------------------------
out_2 = widgets.Output()
display(out_2)
#-------------------------------------------------------------------------------
rat_add_df = pd.DataFrame()
#-------------------------------------------------------------------------------
for a_i, aka_v in enumerate(all_akas):
    plot_col = aka_v[4:]
    plot_col_split = plot_col.split('_')
    #---------------------------------------------------------------------------
    c_count_df = fin_df.groupby(['rating', 'creative_type']).count()[['primary_tit
    #---------------------------------------------------------------------------
    if 'tot' in aka_v:
        aka_start = 'Total'
        #-----------------------------------------------------------------------
        c_agg_df = fin_df.groupby(['rating', 'creative_type']).sum()[[plot_col]]
    #---------------------------------------------------------------------------
    if 'avg' in aka_v:
        aka_start = 'Average'
        #-----------------------------------------------------------------------
        c_agg_df = fin_df.groupby(['rating', 'creative_type']).mean()[[plot_col]]
    #---------------------------------------------------------------------------
    all_cs_df = pd.concat([c_count_df, c_agg_df], axis=1).reset_index()\
                  .sort_values(['rating', plot_col], ascending=[True, False])\
                  .rename(columns={'primary_title':'num_titles', plot_col: aka_v})
    #---------------------------------------------------------------------------
    if 'tot' in aka_v:
        all_cs_df['perc_of_'+ aka_v] = round((all_cs_df[aka_v] / fin_df[plot_col].
    #---------------------------------------------------------------------------
    aka_str = ' '.join([aka_start] + plot_col_split).title()
    #---------------------------------------------------------------------------
    print('\033[1m\033[35mAgg Col: \033[0m', aka_str)
    print('-------------------------------------------------------')

    # Finding the Important Creative Types and the Creative Types for the Additiona
    # section
    ###########################################################################
    aka_add_info_df = pd.DataFrame()
    #---------------------------------------------------------------------------
    for rat in all_cs_df.rating.unique():
        if rat != 'Not Rated': rat_c_df = all_cs_df.loc[all_cs_df.rating == rat]
        else: continue
        #-----------------------------------------------------------------------
        rat_copy_df = rat_c_df.copy()
        #-----------------------------------------------------------------------
        rat_c_df = rat_c_df.loc[(rat_c_df.num_titles >= 9)].head(3)
        #-----------------------------------------------------------------------
        rat_also_df = rat_copy_df.loc[(rat_copy_df.num_titles < 9) &
                                      (rat_copy_df[aka_v] >= rat_c_df[aka_v].max()
        #-----------------------------------------------------------------------
        rat_also_df[aka_v] = rat_also_df[aka_v].map(lambda x: x - rat_c_df[aka_v].
        #-----------------------------------------------------------------------
        if len(rat_also_df):
            rat_also_df = rat_also_df.set_index(['rating', 'creative_type', 'num_t
            #-------------------------------------------------------------------
            aka_add_info_df = pd.concat([aka_add_info_df, rat_also_df])
            #-------------------------------------------------------------------
```

```
            print('\033[1m\033[37mAdded data to the Additional Information datafra
                  '\033[31mRating\033[0m.')
            print('   \033[1m\033[31mRating: \033[0m', rat)
            print('-------------------------------------------------------')
        #-----------------------------------------------------------------------
        rat_c_df = rat_c_df.reset_index(drop=True)
        #-----------------------------------------------------------------------
        rat_dir = base_dir + rat +'/'
        rat_cat_dir = rat_dir + 'Important Creative Types/'
        rat_cat_metric_dir = rat_cat_dir + aka_str +'/'
        #-----------------------------------------------------------------------
        if not os.path.isdir(rat_dir):
            os.mkdir(rat_dir)
        #-----------------------------------------------------------------------
        if not os.path.isdir(rat_cat_dir):
            os.mkdir(rat_cat_dir)
        #-----------------------------------------------------------------------
        if not os.path.isdir(rat_cat_metric_dir):
            os.mkdir(rat_cat_metric_dir)
        #-----------------------------------------------------------------------
        with out_2:
            display(print('\033[1m\033[31mRating: \033[0m', rat), include=[str])
        #-----------------------------------------------------------------------
        if len(rat_c_df) == 1:
            print('\033[1m\033[37mThere was only one associated \033[32mCreative Ty
                  '\033[1m\033[37mfor this \033[31mRating\033[0m.')
            print('   \033[1m\033[31mRating: \033[0m', rat)
            print('   \033[1m\033[32mCreative Type: \033[0m', rat_c_df.creative_ty
            print('-------------------------------------------------------')

        # Top Creative Types and their Boxplots Visualizations
        ##############################################################################
        if len(rat_c_df) > 1:
            fig, fig_tit = rat_cat_plot(rat_c_df, 'creative_type')
            #-------------------------------------------------------------------
            f_name = rat_cat_metric_dir +'1 - Top Creative Types.jpeg'
            #-------------------------------------------------------------------
            fig.savefig(f_name, bbox_inches='tight')
            plt.close(fig)
            #-------------------------------------------------------------------
            all_box_df = pd.DataFrame()
            import_order_dict = {}
            #-------------------------------------------------------------------
            for ct in rat_c_df.creative_type:
                rat_ct_box_df = fin_df.loc[(fin_df.rating == rat) & (fin_df.creati
                #---------------------------------------------------------------
                rat_ct_box_df = rat_ct_box_df[['rating', 'creative_type', plot_col
                #---------------------------------------------------------------
                import_order_dict[ct] = rat_ct_box_df[plot_col].mean()
                #---------------------------------------------------------------
                all_box_df = pd.concat([all_box_df, rat_ct_box_df])
            #-------------------------------------------------------------------
            import_order = \
            sorted(import_order_dict.keys(), key=lambda k: import_order_dict[k], r
            #-------------------------------------------------------------------
            all_box_df = \
            all_box_df.sort_values('creative_type',
                                   key=lambda c_col: c_col.map(lambda x: import_or
            #-------------------------------------------------------------------
```

```
            all_box_df = all_box_df.rename(columns={plot_col: aka_v}).reset_index(
            #-------------------------------------------------------------------
            fig, fig_tit = rat_cat_plot(all_box_df, 'creative_type', aka_v)
            #-------------------------------------------------------------------
            f_name = rat_cat_metric_dir +'2 - '+ fig_tit +'.jpeg'
            #-------------------------------------------------------------------
            fig.savefig(f_name, bbox_inches='tight')
            plt.close(fig)

        # Important Crew Visualizations
        ###############################################################################
        for ct in rat_c_df.creative_type:
            with out_2:
                display(print('\033[1m\033[32mCreative Type: \033[0m', ct), includ
            #-------------------------------------------------------------------
            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 = rat_cat_metric_dir + plot_type.title() +'/'
                #-------------------------------------------------------------------
                if not os.path.isdir(crew_dir):
                    os.mkdir(crew_dir)
                #-------------------------------------------------------------------
                rat_ct_crew_df = crew_df.loc[(crew_df.rating == rat) &
                                             (crew_df.creative_type == ct)]
                #-------------------------------------------------------------------
                if rat_ct_crew_df.primary_name.nunique() <= 1:
                    print('\033[1m\033[37mThere was only one crew member associate
                          '\033[32mCreative Type\033[37m for this \033[31mRating\0
                    print('   \033[1m\033[31mRating: \033[0m'+ rat)
                    print('   \033[1m\033[32mCreative Type: \033[0m', ct)
                    print('   \033[1m\033[33mCrew Type: \033[0m', crew_str.title()
                    print('   \033[1m\033[33mName: \033[0m',
                          rat_ct_crew_df.primary_name.values[0])
                    print('-------------------------------------------------------
                    continue
                #-------------------------------------------------------------------
                else:
                    crew_count_df = \
                    rat_ct_crew_df.groupby('primary_name').count()[['primary_title
                    #---------------------------------------------------------------
                    if 'tot' in aka_v:
                        crew_agg_df = \
                        rat_ct_crew_df.groupby('primary_name').sum()[[plot_col]]
                    #---------------------------------------------------------------
                    if 'avg' in aka_v:
                        crew_agg_df = \
                        rat_ct_crew_df.groupby('primary_name').mean()[[plot_col]]
                    #---------------------------------------------------------------
                    c_df = pd.concat([crew_count_df, crew_agg_df], axis=1).reset_i
                             .rename(columns={'primary_title':'num_titles', plot_c
                             .sort_values(aka_v, ascending=False).head(10)
                    #---------------------------------------------------------------
                    ct_ser = c_df.primary_name.map(lambda x: ct).rename('creative_
                    rat_ser = c_df.primary_name.map(lambda x: rat).rename('rating'
                    #---------------------------------------------------------------
                    c_df.insert(0, 'creative_type', ct_ser)
                    c_df.insert(1, 'rating', rat_ser)
```

```
                    #-------------------------------------------------------------
                    if 'tot' in aka_v:
                        c_df['perc_of_'+ aka_v] = \
                        round((c_df[aka_v] / fin_df[plot_col].sum()), 4)
                    #-------------------------------------------------------------
                    c_df = c_df.reset_index(drop=True).set_index('creative_type')
                    #-------------------------------------------------------------
                    fig, fig_tit = rat_cat_plot(c_df, plot_type)
                    #-------------------------------------------------------------
                    f_name = crew_dir + ct +'.jpeg'
                    #-------------------------------------------------------------
                    fig.savefig(f_name, bbox_inches='tight')
                    plt.close(fig)
                #-----------------------------------------------------------------
                with out_2:
                    display(print('\033[1m\033[32mCreative Type Finished\033[0m'),incl

            # Clearing non-important information after each rating
            ##############################################################################
            with out_2:
                clear_output()
        print()

        # Adding the Additional Information section information
        ##############################################################################
        if not aka_add_info_df.empty:
            ct_add_info_df = pd.concat([ct_add_info_df, aka_add_info_df[[aka_v]]], axi
```

```
Agg Col:  Total Worldwide Gross
---------------------------------------------------------
There was only one associated Creative Type for this Rating.
   Rating:  G
   Creative Type:  Kids Fiction
---------------------------------------------------------

Agg Col:  Total Worldwide Profits
---------------------------------------------------------
There was only one associated Creative Type for this Rating.
   Rating:  G
   Creative Type:  Kids Fiction
---------------------------------------------------------

Agg Col:  Average Worldwide Gross
---------------------------------------------------------
There was only one associated Creative Type for this Rating.
   Rating:  G
   Creative Type:  Kids Fiction
---------------------------------------------------------
Added data to the Additional Information dataframe for this Rating.
   Rating:  PG
---------------------------------------------------------
Added data to the Additional Information dataframe for this Rating.
   Rating:  R
---------------------------------------------------------

Agg Col:  Average Worldwide Profits
---------------------------------------------------------
There was only one associated Creative Type for this Rating.
```

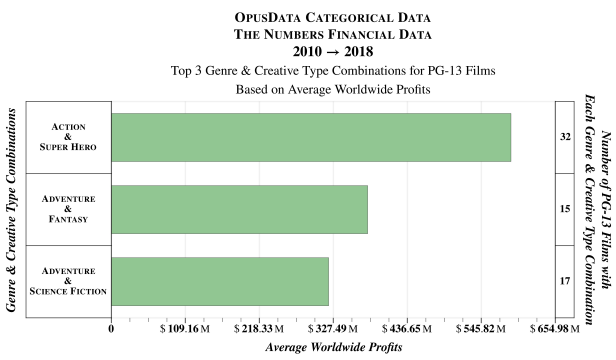
```
   Rating:  G
   Creative Type:  Kids Fiction
----------------------------------------------------------
Added data to the Additional Information dataframe for this Rating.
   Rating:  PG
----------------------------------------------------------
Added data to the Additional Information dataframe for this Rating.
   Rating:  R
----------------------------------------------------------
```

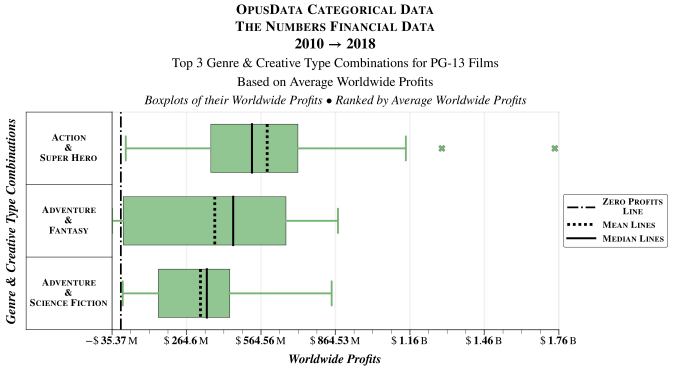
Financially Important Genre & Creative Type Combinations

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

Example Visualizations

(Right Click and Open Image in a New Tab)





In [42]:

```
base_dir = 'visuals/'
#-------------------------------------------------------------------------------
out_3 = widgets.Output()
display(out_3)
#-------------------------------------------------------------------------------
rat_add_df = pd.DataFrame()
#-------------------------------------------------------------------------------
for a_i, aka_v in enumerate(all_akas):
    plot_col = aka_v[4:]
    plot_col_split = plot_col.split('_')
    #---------------------------------------------------------------------------
    combo_count_df = \
    fin_df.groupby(['rating', 'genre', 'creative_type']).count()[['primary_title']
    #---------------------------------------------------------------------------
    if 'tot' in aka_v:
        aka_start = 'Total'
        #-----------------------------------------------------------------------
        combo_agg_df = fin_df.groupby(['rating', 'genre', 'creative_type']).sum()[
    #---------------------------------------------------------------------------
    if 'avg' in aka_v:
        aka_start = 'Average'
        #-----------------------------------------------------------------------
        combo_agg_df = fin_df.groupby(['rating', 'genre', 'creative_type']).mean()
    #---------------------------------------------------------------------------
    all_combos_df = pd.concat([combo_count_df, combo_agg_df], axis=1).reset_index(
                  .sort_values(['rating', plot_col], ascending=[True, False])\
                  .rename(columns={'primary_title':'num_titles', 'genre':'genres',
                                   plot_col: aka_v})
    #---------------------------------------------------------------------------
    if 'tot' in aka_v:
        all_combos_df['perc_of_'+ aka_v] = \
        round((all_combos_df[aka_v] / fin_df[plot_col].sum()), 4)
    #---------------------------------------------------------------------------
    aka_str = ' '.join([aka_start] + plot_col_split).title()
    #---------------------------------------------------------------------------
    print('\033[1m\033[35mAgg Col: \033[0m', aka_str)
    print('-----------------------------------------------------')

    # Finding the Important Combos and the Combos for the Additional Information s
    ###############################################################################
    all_combos_df = \
    all_combos_df.loc[all_combos_df.rating.map(lambda x: True if x!='Not Rated' el
    all_combos_df = all_combos_df.sort_values(aka_v, ascending=False)
    #---------------------------------------------------------------------------
    aka_add_info_df = pd.DataFrame()
    #---------------------------------------------------------------------------
    for rat in all_combos_df.rating.unique():
        rat_df = all_combos_df.loc[(all_combos_df.rating == rat) &
                                   (all_combos_df.num_titles >= 9)].copy()
        #-----------------------------------------------------------------------
        new_genres = rat_df.apply(lambda x: ','.join([x['genres'], x['creative_typ
        #-----------------------------------------------------------------------
        rat_df = rat_df.drop(columns=['genres', 'creative_type']).head(3)
        #-----------------------------------------------------------------------
        rat_also_df = all_combos_df.loc[(all_combos_df.rating == rat) &
                                        (all_combos_df.num_titles < 9) &
                                        (all_combos_df[aka_v] >= rat_df[aka_v].max
        #-----------------------------------------------------------------------
```

```
        rat_also_df[aka_v] = rat_also_df[aka_v].map(lambda x: x - rat_df[aka_v].ma
        #-------------------------------------------------------------------------
        if len(rat_also_df):
            rat_also_df = \
            rat_also_df.set_index(['rating', 'genres', 'creative_type', 'num_title
            #---------------------------------------------------------------------
            aka_add_info_df = pd.concat([aka_add_info_df, rat_also_df])
            #---------------------------------------------------------------------
            print('\033[1m\033[37mAdded data to the Additional Information datafra
                  '\033[31mRating\033[0m.')
            print('   \033[1m\033[31mRating: \033[0m', rat)
            print('-------------------------------------------------------')
        #-------------------------------------------------------------------------
        rat_df.insert(1, 'genres', new_genres)
        #-------------------------------------------------------------------------
        rat_df = rat_df.reset_index(drop=True)
        #-------------------------------------------------------------------------
        rat_dir = base_dir + rat +'/'
        rat_cat_dir = rat_dir + 'Important Combinations/'
        rat_cat_metric_dir = rat_cat_dir + aka_str +'/'
        #-------------------------------------------------------------------------
        if not os.path.isdir(rat_dir):
            os.mkdir(rat_dir)
        #-------------------------------------------------------------------------
        if not os.path.isdir(rat_cat_dir):
            os.mkdir(rat_cat_dir)
        #-------------------------------------------------------------------------
        if not os.path.isdir(rat_cat_metric_dir):
            os.mkdir(rat_cat_metric_dir)
        #-------------------------------------------------------------------------
        with out_3:
            display(print('\033[1m\033[31mRating: \033[0m', rat), include=[str])
        #-------------------------------------------------------------------------
        if len(rat_df) == 1:
            print('\033[1m\033[37mThere was only one associated \033[32mCat Combo\(
                  '\033[1m\033[37mfor this \033[31mRating\033[0m.')
            print('   \033[1m\033[31mRating: \033[0m', rat)
            print('   \033[1m\033[32mCat Combo: \033[0m', rat_df.genres.values[0])
            print('-------------------------------------------------------')

        # Top Combinations and their Boxplots Visualizations
        ############################################################################
        if len(rat_df) > 1:
            fig, fig_tit = rat_cat_plot(rat_df, 'combos')
            #-----------------------------------------------------------------
            f_name = rat_cat_metric_dir +'1 - Top Combinations.jpeg'
            #-----------------------------------------------------------------
            fig.savefig(f_name, bbox_inches='tight')
            plt.close(fig)
            #-----------------------------------------------------------------
            all_box_df = pd.DataFrame()
            import_order_dict = {}
            #-----------------------------------------------------------------
            for g in rat_df.genres:
                ct = g.split(',')[1]
                g = g.split(',')[0]
                #-------------------------------------------------------------
                rat_g_ct_box_df = fin_df.loc[(fin_df.rating == rat) & (fin_df.genr
                                             (fin_df.creative_type == ct)]
```

```
                #-------------------------------------------------------------
                rat_g_ct_box_df = \
                rat_g_ct_box_df.drop(columns='genres').rename({'genre': 'genres'},
                #-------------------------------------------------------------
                rat_g_ct_box_df['genres'] = [','.join([g, ct])] * len(rat_g_ct_box
                #-------------------------------------------------------------
                rat_g_ct_box_df = rat_g_ct_box_df[['rating', 'genres', plot_col]]
                #-------------------------------------------------------------
                import_order_dict[g+','+ct] = rat_g_ct_box_df[plot_col].mean()
                #-------------------------------------------------------------
                all_box_df = pd.concat([all_box_df, rat_g_ct_box_df])
            #-----------------------------------------------------------------
            import_order = \
            sorted(import_order_dict.keys(), key=lambda k: import_order_dict[k], r
            #-----------------------------------------------------------------
            all_box_df = \
            all_box_df.sort_values('genres',
                                   key=lambda g_col: g_col.map(lambda x: import_or
            #-----------------------------------------------------------------
            all_box_df = all_box_df.rename(columns={plot_col: aka_v}).reset_index(
            #-----------------------------------------------------------------
            fig, fig_tit = rat_cat_plot(all_box_df, 'combos', aka_v)
            #-----------------------------------------------------------------
            f_name = rat_cat_metric_dir +'2 - '+ fig_tit +'.jpeg'
            #-----------------------------------------------------------------
            fig.savefig(f_name, bbox_inches='tight')
            plt.close(fig)

        # Important Crew Visualizations
        ###############################################################################
        for g in rat_df.genres:
            ct = g.split(',')[1]
            g = g.split(',')[0]
            #-------------------------------------------------------------
            with out_3:
                display(print('\033[1m\033[32mCat Combo: \033[0m', g +', '+ ct),
                        include=[str])
            #-------------------------------------------------------------
            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 = rat_cat_metric_dir + plot_type.title() +'/'
                #-------------------------------------------------------------
                if not os.path.isdir(crew_dir):
                    os.mkdir(crew_dir)
                #-------------------------------------------------------------
                rat_crew_df = crew_df.loc[(crew_df.rating == rat) & (crew_df.genre
                                          (crew_df.creative_type == ct)]
                #-------------------------------------------------------------
                if rat_crew_df.primary_name.nunique() <= 1:
                    print('\033[1m\033[31mRating: \033[0m', rat)
                    print('\033[1m\033[32mCat Combo: \033[0m', g +', '+ ct)
                    print('\033[1m\033[33mCrew Type: \033[0m', crew_str.title())
                    print('\033[1m\033[33mName: \033[0m', rat_crew_df.primary_name
                    print('-------------------------------------------------------
                    continue
                #-------------------------------------------------------------
                else:
```

```
                    crew_count_df = \
                    rat_crew_df.groupby('primary_name').count()[['primary_title']]
                    #-------------------------------------------------------------
                    if 'tot' in aka_v:
                        crew_agg_df = rat_crew_df.groupby('primary_name').sum()[[p
                    #-------------------------------------------------------------
                    if 'avg' in aka_v:
                        crew_agg_df = rat_crew_df.groupby('primary_name')\
                                                 .mean()[[plot_col]]
                    #-------------------------------------------------------------
                    c_df = pd.concat([crew_count_df, crew_agg_df], axis=1).reset_i
                             .rename(columns={'primary_title':'num_titles', plot_c
                             .sort_values(aka_v, ascending=False).head(10)
                    #-------------------------------------------------------------
                    genre_ser = c_df.primary_name.map(lambda x: g+','+ct).rename('
                    rat_ser = c_df.primary_name.map(lambda x: rat).rename('rating'
                    #-------------------------------------------------------------
                    c_df.insert(0, 'genres', genre_ser)
                    c_df.insert(1, 'rating', rat_ser)
                    #-------------------------------------------------------------
                    if 'tot' in aka_v:
                        c_df['perc_of_'+ aka_v] = \
                        round((c_df[aka_v] / fin_df[plot_col].sum()), 4)
                    #-------------------------------------------------------------
                    c_df = c_df.reset_index(drop=True).set_index('genres')
                    #-------------------------------------------------------------
                    fig, fig_tit = rat_cat_plot(c_df, plot_type)
                    #-------------------------------------------------------------
                    if '/' in g: g_dir = ' - '.join(g.split('/'))
                    else: g_dir = g
                    #-------------------------------------------------------------
                    if '/' in ct: ct_dir = ' - '.join(ct.split('/'))
                    else: ct_dir = ct
                    #-------------------------------------------------------------
                    f_name = crew_dir + g_dir +', '+ ct_dir +'.jpeg'
                    #-------------------------------------------------------------
                    fig.savefig(f_name, bbox_inches='tight')
                    plt.close(fig)
            #-----------------------------------------------------------------
            with out_3:
                display(print('\033[1m\033[32mCat Combo Finsished\033[0m'), includ

        # Clearing non-important information after each rating
        ##############################################################################
        with out_3:
            clear_output()
    print()

    # Adding the Additional Information section information
    ##############################################################################
    if not aka_add_info_df.empty:
        combo_add_info_df = pd.concat([combo_add_info_df, aka_add_info_df[[aka_v]]
```

```
Agg Col:  Total Worldwide Gross
---------------------------------------------------
There was only one associated Cat Combo for this Rating.
   Rating:  G
   Cat Combo:  Adventure,Kids Fiction
```

```
-----------------------------------------------------

Agg Col:  Total Worldwide Profits
-----------------------------------------------------
There was only one associated Cat Combo for this Rating.
   Rating:  G
   Cat Combo:  Adventure,Kids Fiction
-----------------------------------------------------

Agg Col:  Average Worldwide Gross
-----------------------------------------------------
Added data to the Additional Information dataframe for this Rating.
   Rating:  R
-----------------------------------------------------
Added data to the Additional Information dataframe for this Rating.
   Rating:  PG
-----------------------------------------------------
There was only one associated Cat Combo for this Rating.
   Rating:  G
   Cat Combo:  Adventure,Kids Fiction
-----------------------------------------------------

Agg Col:  Average Worldwide Profits
-----------------------------------------------------
Added data to the Additional Information dataframe for this Rating.
   Rating:  R
-----------------------------------------------------
Added data to the Additional Information dataframe for this Rating.
   Rating:  PG
-----------------------------------------------------
There was only one associated Cat Combo for this Rating.
   Rating:  G
   Cat Combo:  Adventure,Kids Fiction
-----------------------------------------------------
```

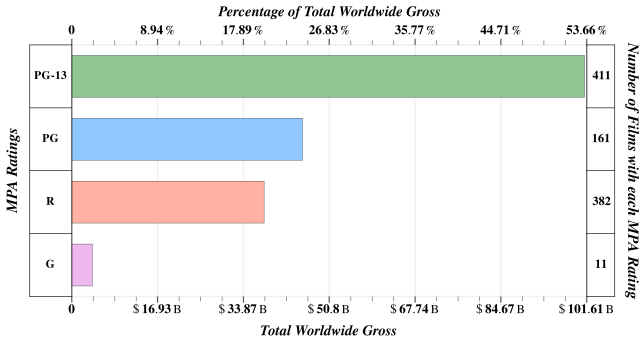
MPA Ratings Priorities

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

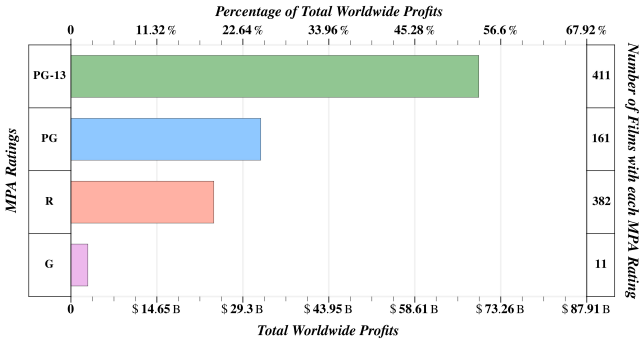
MPA Ratings Visualizations

(Right Click and Open Image in a New Tab)

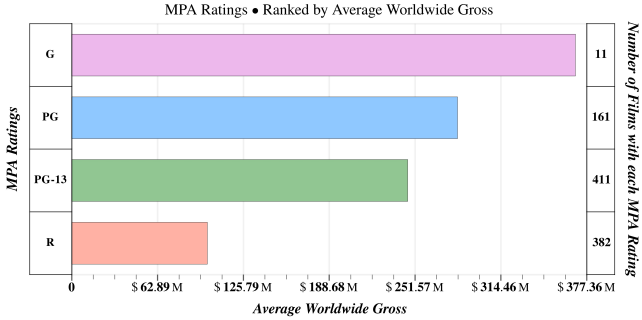
OpusData Ratings Data
The Numbers Financial Data
2010 → 2018
MPA Ratings • Ranked by Total Worldwide Gross



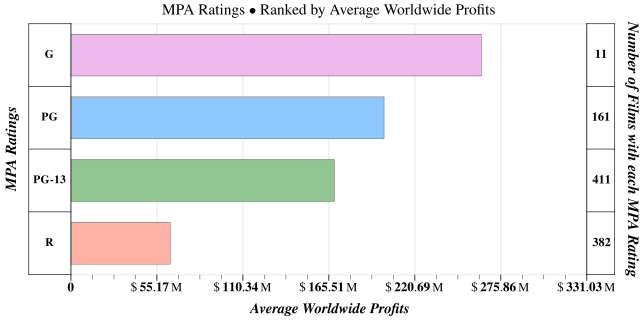
OpusData Ratings Data
The Numbers Financial Data
2010 → 2018
MPA Ratings • Ranked by Total Worldwide Profits



OpusData Ratings Data
The Numbers Financial Data
2010 → 2018
MPA Ratings • Ranked by Average Worldwide Gross



OpusData Ratings Data
The Numbers Financial Data
2010 → 2018
MPA Ratings • Ranked by Average Worldwide Profits



In [43]:

```
for aka_v in all_akas:
    plot_col = aka_v[4:]
    #-----------------------------------------------------------------------------
    rat_count_df = fin_df.groupby(['rating']).count()[['primary_title']].drop('Not
    #-----------------------------------------------------------------------------
    if 'tot' in aka_v:
        aka_start = 'Total'
        #-------------------------------------------------------------------------
        rat_agg_df = fin_df.groupby(['rating']).sum()[[plot_col]].drop('Not Rated')
    #-----------------------------------------------------------------------------
    if 'avg' in aka_v:
        aka_start = 'Average'
        #-------------------------------------------------------------------------
        rat_agg_df = fin_df.groupby(['rating']).mean()[[plot_col]].drop('Not Rated'
    #-----------------------------------------------------------------------------
    rats_df = pd.concat([rat_count_df, rat_agg_df], axis=1).reset_index()\
                .sort_values(['rating', plot_col], ascending=[True, False])\
                .rename(columns={'primary_title':'num_titles', plot_col: aka_v})\
                .sort_values(aka_v, ascending=False)
    #-----------------------------------------------------------------------------
    if 'tot' in aka_v:
        rats_df['perc_of_'+ aka_v] = round((rats_df[aka_v] / fin_df[plot_col].sum()
    #-----------------------------------------------------------------------------
    fig, fig_tit = rat_cat_plot(rats_df, 'rating')
    #-----------------------------------------------------------------------------
    all_rats_dir = 'visuals/All Ratings/'
    #-----------------------------------------------------------------------------
    if not os.path.isdir(all_rats_dir):
        os.mkdir(all_rats_dir)
    #-----------------------------------------------------------------------------
    f_name = all_rats_dir + fig_tit +'.jpeg'
    #-----------------------------------------------------------------------------
    fig.savefig(f_name, bbox_inches='tight')
    plt.close(fig)
```

§ Recommendations for Microsoft

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

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

Recommendations for PG-13 Content

PG-13

Priority	Genres
1.	Action
2.	Adventure
3.	Thriller / Suspense

Priority	Creative Types
1.	Super Hero
2.	Science Fiction
3.	Fantasy

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

Recommendations for PG Content

PG

Priority	Genres
1.	Musaical
2.	Adventure

Priority	Creative Types
1.	Kids Fiction
2.	Fantasy

Priority	Genre & Creative Type Combinations
1.	Adventure & Kids Fiction
2.	Adventure & Fantasy

Recommendations for R Content

PG-13

Priority	Genres
1.	Horror
2.	Black Comedy
3.	Action
4.	Comedy

Priority	Creative Types
1.	Science Fiction
2.	Contemporary Fiction

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

Recommendations for G Content

PG

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

Important Subcategories & Crew Interactive Visualization Explorer

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

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

If you are unable to run the code or open the presentation to use this feature, I recreated the crew-finding capabilities of the feature, for the categories that I recommended at least, in the section below in the Markdown file to make it easier. **Click here to go to that section now.** (https://github.com/sarnadpy32/microsoft_productions#sub_sect_markdown_crew)

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

In [44]:

```
rat_choices = ['Choose a MPA Rating from the choices below...',
               'PG-13', 'PG', 'R', 'G']
#-------------------------------------------------------------------------------
cat_choices = ['Choose a category type from the choices below...',
               'Important Genres', 'Important Creative Types',
               'Important Genre & Creative Type Combinations']
#-------------------------------------------------------------------------------
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='initial', height='30px')
#-------------------------------------------------------------------------------
first_menu = \
widgets.Dropdown(options=rat_choices, value=rat_choices[0], disabled=False, layout
#-------------------------------------------------------------------------------
second_menu = \
widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False,layout=me
#-------------------------------------------------------------------------------
third_menu = \
widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=m
#-------------------------------------------------------------------------------
fourth_menu = \
widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=m
#-------------------------------------------------------------------------------
fifth_menu = \
widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=m
#-------------------------------------------------------------------------------
sixth_menu = \
widgets.Dropdown(options=empty_slot, value=empty_slot[0], disabled=False, layout=m

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

    # First Menu
    ###########################################################################
    if a not in [rat_choices[0], empty_slot[0]]:
        f_path_p1 = a +'/'
        #-----------------------------------------------------------------------
        second_menu.options = cat_choices
        first_menu.disabled = True
    #---------------------------------------------------------------------------
    else: second_menu.options = empty_slot

    # Second Menu
    ###########################################################################
    if b not in [cat_choices[0], empty_slot[0]]:
        if 'Combinations' in b: f_path_p2 = 'Important Combinations/'
        else: f_path_p2 = b +'/'
        #-----------------------------------------------------------------------
        third_menu.options = metric_choices
        second_menu.disabled = True
```

```
    #-------------------------------------------------------------------------
    else: third_menu.options = empty_slot

    # Third Menu
    ##########################################################################
    fourth_choice_1_a = 'Do you wish to explore the '
    fourth_choice_1_b = ' themselves, or the Important Crew from the films with th
    fourth_choice_1_c = '?'
    #-------------------------------------------------------------------------
    if c not in [metric_choices[0], empty_slot[0]]:
        fourth_choice_insert = b.lstrip('Important ')
        #---------------------------------------------------------------------
        fourth_choice_1 = fourth_choice_1_a + fourth_choice_insert + fourth_choice_
                          fourth_choice_insert + fourth_choice_1_c
        #---------------------------------------------------------------------
        fourth_choices = [fourth_choice_1, fourth_choice_insert, 'Important Crew']
        #---------------------------------------------------------------------
        if a == rat_choices[-1]:
            fourth_choice_1 = 'Click here and choose Important Crew, as there were
                              'exploratory visualizations for this Rating.'
            #-----------------------------------------------------------------
            fourth_choices = [fourth_choice_1, 'Important Crew']
        #---------------------------------------------------------------------
        fourth_menu.options = fourth_choices
        #---------------------------------------------------------------------
        f_path_p3 = c +'/'
        #---------------------------------------------------------------------
        third_menu.disabled = True
    #-------------------------------------------------------------------------
    else:
        fourth_menu.options = empty_slot
        fourth_choices = ['empty']

    # Fourth Menu
    ##########################################################################
    if d in fourth_choices:
        if a != rat_choices[-1] and d == fourth_choices[1]:
            file_path = 'visuals/' + f_path_p1 + f_path_p2 + f_path_p3
            file_choices = listdir(file_path)[:2]
            #-----------------------------------------------------------------
            file_choices.insert(0, 'Please choose a file from the choices below...
            #-----------------------------------------------------------------
            fifth_menu.options = file_choices
            fourth_menu.disabled = True
        #---------------------------------------------------------------------
        if (a == rat_choices[-1] and d == fourth_choices[1]) or \
           (a != rat_choices[-1] and d == fourth_choices[2]):
            fifth_menu.options = crew_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 crew_choices[1:]:
        file_path = 'visuals/' + f_path_p1 + f_path_p2 + f_path_p3 + 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, s
#-------------------------------------------------------------------------------
out = widgets.interactive_output(f, {'a':first_menu, 'b':second_menu, 'c':third_me
                                     'd':fourth_menu, 'e':fifth_menu, 'f':sixth_me
#-------------------------------------------------------------------------------
print('\n\033[1m    Follow the instructions that appear in each dropdown menu to a
      '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 visualizat
ion with the desired properties.**

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

PG-13

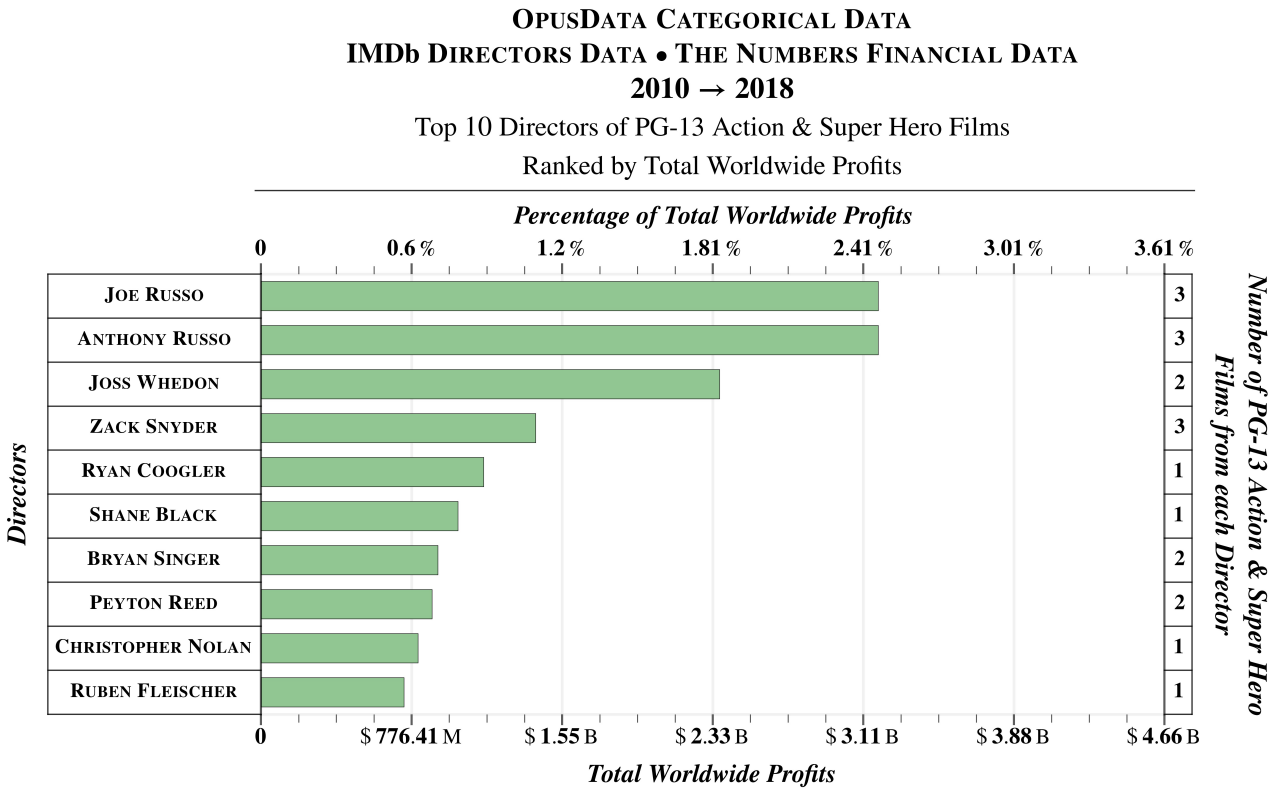
Important Genre & Creative Type Combinations

Total Worldwide Profits

Important Crew

Directors

Action, Super Hero.jpeg



§ Additional Insights

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

In [45]:

```
pg_add_info_df = pd.DataFrame()
r_add_info_df = pd.DataFrame()
#-------------------------------------------------------------------------------
subcat_add_infos = [g_add_info_df, ct_add_info_df, combo_add_info_df]
#-------------------------------------------------------------------------------
new_aka_cols = ['Amount Greater<br>than the<br>Top Average<br>Worldwide Gross',
                'Amount Greater<br>than the<br>Top Average<br>Worldwide Profits']
rest_cols = ['rating', 'genres', 'creative_type', 'num_titles']
#-------------------------------------------------------------------------------
for df_i, add_info_df in enumerate(subcat_add_infos):
    add_info_df = add_info_df.reset_index()
    #---------------------------------------------------------------------------
    aka_cols = [col for col in add_info_df.columns if col in all_akas]
    idx_cols = [col for col in add_info_df.columns if col in rest_cols]
    #---------------------------------------------------------------------------
    for rat in add_info_df.rating.unique():
        md_rat_add_df = add_info_df.loc[add_info_df.rating == rat][idx_cols + aka_c
        #-----------------------------------------------------------------------
        md_rat_add_df = \
        md_rat_add_df.sort_values(md_rat_add_df.columns[-1], ascending=False)\
                     .reset_index(drop=True)
        #-----------------------------------------------------------------------
        if df_i == 2:
            combo_ser = \
            md_rat_add_df.apply(lambda x: ' & '.join((x['genres'], x['creative_type
            #-------------------------------------------------------------------
            md_rat_add_df.insert(1, 'Genre & Creative Type<br>Combinations', combo_
            #-------------------------------------------------------------------
            md_rat_add_df = md_rat_add_df.drop(columns=['genres', 'creative_type'])
        #-----------------------------------------------------------------------
        for col in rest_cols:
            if col in rest_cols[:2]: new_col = col.title()
            if col == rest_cols[2]: new_col = ' '.join(col.split('_')).title()
            if col == rest_cols[3]: new_col = 'Number of Titles'
            #-------------------------------------------------------------------
            md_rat_add_df = md_rat_add_df.rename(columns={col: new_col})
        #-----------------------------------------------------------------------
        md_rat_add_df = md_rat_add_df.rename(columns={all_akas[2]: new_aka_cols[0],
                                                      all_akas[3]: new_aka_cols[1]})
        nb_rat_add_df = md_rat_add_df.copy()
        #-----------------------------------------------------------------------
        nb_rat_add_df[new_aka_cols] = \
        md_rat_add_df[new_aka_cols].applymap(lambda x: notebook_table_currency(x))
        #-----------------------------------------------------------------------
        md_rat_add_df[new_aka_cols] = \
        md_rat_add_df[new_aka_cols].applymap(lambda x: markdown_table_currency(x))
        #-----------------------------------------------------------------------
        priority = [1] if len(md_rat_add_df) == 1 else range(1, len(md_rat_add_df)
        md_priority = [str(p) + '&period;' for p in priority]
        nb_priority = [str(p) + '.' for p in priority]
        md_rat_add_df.insert(0, 'Priority', md_priority)
        nb_rat_add_df.insert(0, 'Priority', nb_priority)
        #-----------------------------------------------------------------------
        md_rat_add_df = md_rat_add_df.set_index('Priority')
        #-----------------------------------------------------------------------
        align_cols = ['center'] * (len(md_rat_add_df.columns) + 1)

# Not Necessary for Actual Analysis - Uncomment to print when Needed
```

```
59 ##########################################################################################
60 #         print(md_rat_add_df.to_markdown(tablefmt='pipe', colalign=align_cols))
61 #         print('\n')
62 #         print(nb_rat_add_df.to_html(index=False, justify="center", escape=False))
63 #         print('\n\n\n')
```

Additional Insights for PG Content

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

Priority	Rating	Creative Type	Number of Titles	Amount Greater than the Top Average Worldwide Gross	Amount Greater than the Top Average Worldwide Profits
1.	PG	Super Hero	1	$ 288.15 M	$ 216.13 M

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

Additional Insights for R Content

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

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

Priority	Rating	Creative Type	Number of Titles	Amount Greater than the Top Average Worldwide Gross	Amount Greater than the Top Average Worldwide Profits
1.	R	Super Hero	3	$ 600.49 M	$ 554.98 M

Priority	Rating	Genre & Creative Type Combinations	Number of Titles	Amount Greater than the Top Average Worldwide Gross	Amount Greater than the Top Average Worldwide Profits
1.	R	Action & Super Hero	3	$ 559.88 M	$ 523.74 M

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

END OF PROJECT