1 Microsoft in The Movie Business

Table of Contents

1.1 Overview

This project provides insight on the prospects of Microsoft entering the movie making business. In this competitive market, there are a myriad of factors that need to be addressed before fully committing to the project. First, estimating the budget/expenditure for a successful return on investment needs to be investigated. Investigating whether established franchise for making profits or stand-alone movies is the next step. To avoid possible flops, it is important to see if production budget and viewer satisfaction. Success of movies differs based on the genres. Therefore, genres with easy entry and profitability must be examined. Lastly, casting and director recommendations for a successful franchise is made. Microsoft can use preliminary analysis to venture into movie making or streaming service.

1.2 Business Problem

The movie industry is one of the highest return on investment. However, so far, it has been monopolized by few industries. Streaming industries as now making their own franchises that rivals well established companies. Therefore, it is crucial for new company to examine the key aspects of the industry in order to succeed. This project tends to address the following questions:

- What is the current trend on production budget?
- Are there upsides to establishing a franchise?
- Do viewers appreciate the production cost of movies?
- Which genre is the most lucrative?
- Does cast and direction choice influence the gross?

1.3 Data Understanding

Data for this project about production budget, domestic and worldwide gross was taken from The-Numbers (https://www.the-numbers.com/). IMDb (https://www.imdb.com/) has the largest repository for movies, TVs or virtually any kind of media accessible to the public. IMDb dataset (https://datasets.imdbws.com/) include the rating, number of votes, release year, genre, cast and crew information.

The data was scraped from The-Numbers website. This notebook (web_scrape_and_api.ipynb) contians detailed procedure on how to find the elements that could be scraped using BeautifulSoup. IMDb dataset (https://datasets.imdbws.com/) has a list of zipped files that are ready to be downloaded. It contains titles of movies, ratings, principal roles and names of cast and crew.

IMPORTANT NOTE: Before moving on, run the **IMDb Dataset Downloader** section of web_scrape_and_api (web_scrape_and_api.ipynb). The files are too big to upload to GitHub.

In [52]:
```
#Import important libraries
import requests
import pandas as pd
import re
from bs4 import BeautifulSoup
import wikipedia
```

In [53]:
```
#Load the-numbers dataset saved to disk. Run the_numbers_scraper.ipynb first
tn_df = pd.read_csv("zippedData/tn_movie_budgets_updated.csv.gz",
                    compression='gzip')
tn_df.head(3)
```

Out[53]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Apr 23, 2019	Avengers: Endgame	$400,000,000	$858,373,000	$2,797,800,564
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$379,000,000	$241,071,802	$1,045,713,802
2	3	Apr 22, 2015	Avengers: Age of Ultron	$365,000,000	$459,005,868	$1,395,316,979

In [54]:
```
#Load IMDb dataset. For title matching, title.basics.tsv.gz file is used.
#Run IMDb Dataset Downloader section of web_scrape_and_api prior to this cod
imdb_df = pd.read_csv("zippedData/title.basics.tsv.gz",
                      compression='gzip',
                      delimiter='\t',
                      low_memory=False)

#Select rows with 'titleType' == 'movie'
imdb_df = imdb_df[imdb_df['titleType']=='movie']
imdb_df.head(3)
```

Out[54]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	
498	tt0000502	movie	Bohemios	Bohemios	0	1905	\N	100	
570	tt0000574	movie	The Story of the Kelly Gang	The Story of the Kelly Gang	0	1906	\N	70	Ac
587	tt0000591	movie	The Prodigal Son	L'enfant prodigue	0	1907	\N	90	

In [55]:
```
#Add rating
#Run IMDb Dataset Downloader section of web_scrape_and_api prior to this cod
rating_df = pd.read_csv("zippedData/title.ratings.tsv.gz",
                        compression='gzip',
                        delimiter='\t',
                        low_memory=False)
rating_df.head(3)
```

Out[55]:

	tconst	averageRating	numVotes
0	tt0000001	5.7	1827
1	tt0000002	6.0	233
2	tt0000003	6.5	1581

In [56]:
```
#Add principals for each movie
#Run IMDb Dataset Downloader section of web_scrape_and_api prior to this cod
princ_df = pd.read_csv("zippedData/title.principals.tsv.gz",
                       compression='gzip',
                       delimiter='\t')
princ_df.head(3)
```

Out[56]:

	tconst	ordering	nconst	category	job	characters
0	tt0000001	1	nm1588970	self	\N	["Self"]
1	tt0000001	2	nm0005690	director	\N	\N
2	tt0000001	3	nm0374658	cinematographer	director of photography	\N

In [57]:
```
#Find names. We only need names that are in the movies.
#Run IMDb Dataset Downloader section of web_scrape_and_api prior to this cod
names = pd.read_csv("zippedData/name.basics.tsv.gz",
                    compression='gzip',
                    delimiter='\t')
names.head(3)
```

Out[57]:

	nconst	primaryName	birthYear	deathYear	primaryProfession	
0	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt0050419,tt003
1	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt0037382,tt01
2	nm0000003	Brigitte Bardot	1934	\N	actress,soundtrack,music_department	tt0049189,tt005

1.4 Data Preparation and Cleaning

Next several steps were taken to make sure that the right movie was matched. The first is grouping both dataframes using `title` and `year` values. Prior to this, however, the titles had to be modified so that undesired characters could reduce the hits. Next, filler words such as 'the' and 'and' were removed from titles that were not matched in the first round. Third, movies that have matching names but the year was off by a few years. Those were matched. Lastly, movies that do not have release date in `tn_df` were matched with names in `imdb_df` . The final dataset is saved here (zippedData/movies.csv.gz).

This section consolidates data is collected from three sources (The-Numbers (https://www.the-numbers.com/) and IMDb (https://www.imdb.com/)). Links to notebooks that collect/scrape data from The-Numbers (the_numbers_scraper.ipynb) and IMDb (imdb_dataset_download.ipynb) are provided.

In this notebook, datasets from the two sources are merged using movie title and release year. To do this, spaces and special characters are stripped from the movie titles.

For entries that do not match, these additional steps were taken:

- Remove *'the'* and *'and'* from titles on the unmatched datasets
- Match using title and select the nearest year of release (atmost ± 3 years apart)
- For The-Numbers dataset with unknown dataset, match only those with single rows after being merged

In the next step, movie titles are scraped from Wikipedia for names of the companies that distribute the movies. The final dataframe is also merged with other tables from IMDb datasets using the unique movie id.

In [58]:
```
#Create 'startYear' column in tn_df, with similar name to imdb_df
tn_df['startYear'] = tn_df['release_date'].map(lambda x: x.split()[-1])
tn_df.head(3)
```

Out[58]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYear
0	1	Apr 23, 2019	Avengers: Endgame	$400,000,000	$858,373,000	$2,797,800,564	2019
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$379,000,000	$241,071,802	$1,045,713,802	2011
2	3	Apr 22, 2015	Avengers: Age of Ultron	$365,000,000	$459,005,868	$1,395,316,979	2015

In [59]:
```
#Set '\N' values to 'Unknown' (similar to tn_df) and replace numbers in 'sta
imdb_df[imdb_df['startYear']=='\\N']='Unknown'
imdb_df['startYear'] = imdb_df['startYear'].astype(str)
imdb_df.head(3)
```

Out[59]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	
498	tt0000502	movie	Bohemios	Bohemios	0	1905	\N	100	
570	tt0000574	movie	The Story of the Kelly Gang	The Story of the Kelly Gang	0	1906	\N	70	Ac
587	tt0000591	movie	The Prodigal Son	L'enfant prodigue	0	1907	\N	90	

In [60]:
```
#Assign columns to merge on, and columns to choose from imdb_df
cols = ['tconst','primaryTitle','originalTitle','isAdult','runtimeMinutes',
        'genres','simple_title','startYear']
```

In [61]:
```
#Function strips away special characters and additional list of strings
def prep_title(title, replace=False):
    words = ['the','and']
    title = [x.lower() for x in title.split()]
    if replace:
        for word in words:
            if word in title:
                title.remove(word)
    return re.sub('[\W\_]','','.join(title))

#Function that returns rows present only in df_1 but not in d_
def left_outer_merge(df_1,df_2,column):
    left_out = df_1.merge(df_2[column], how="left", on=column,indicator=True
    left_out=left_out[left_out['_merge']=='left_only']
    return left_out.drop('_merge',axis=1)
```

In [62]:
```
#Create a column named 'simple_title' in 'tn_df' for comparing titles
tn_df['simple_title'] = tn_df['movie'].map(lambda x: prep_title(x))
tn_df.head(3)
```

Out[62]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYear	
0	1	Apr 23, 2019	Avengers: Endgame	$400,000,000	$858,373,000	$2,797,800,564	2019	
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$379,000,000	$241,071,802	$1,045,713,802	2011	pir
2	3	Apr 22, 2015	Avengers: Age of Ultron	$365,000,000	$459,005,868	$1,395,316,979	2015	

Now we are ready to merge the two datasets. IMDb dataset has two columns for titles: original and primary. Therefore, titles are prepped from these two columns separately and merged with `tn_df` .

1.4.1 Merge Datasets by Year and Title

In [63]:
```
#Create a new dataframe called movies_df
movies_df = pd.DataFrame(columns=list(tn_df.columns)+cols[:-2])
```

In [64]:

```
#Function that preps the title and concatinates dataframes on orignal and pr
def concat_with_movies(df_1,df_2,
                       merge_cols=['simple_title','startYear'],
                       replace=False,
                       m_df=pd.DataFrame(),
                       cols=cols):
    '''
    df_1: The first dataframe
    df_2: The second dataframe. df_1 and df_2 must share columns
          specified by merge_cols
    merge_cols: list of columns on which the df_1 and df_2
    replace: boolean to replace 'the' and 'and' in the movie titles
    m_df: a dummy dataframe returned by the function, set empty by default
    cols: columns selected from the df_2
    '''
    df_1['simple_title'] = df_1['movie'].map(lambda x: prep_title(x,replace)
    for col in df_2.columns:
        if 'Title' in col:
            df_2['simple_title'] = df_2[col].map(lambda x: prep_title(x,repl
            df = pd.merge(df_1,df_2[cols],on=merge_cols)
            if m_df.empty:
                m_df =df
            else:
                m_df = pd.concat([m_df,df[m_df.columns]])
    return m_df.drop_duplicates()
```

In [65]:
```
#First merge tn_df and imdb_df based on title and release year
movies_df = concat_with_movies(tn_df,imdb_df)
movies_df
```

Out[65]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYe
0	1	Apr 23, 2019	Avengers: Endgame	$400,000,000	$858,373,000	$2,797,800,564	20
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$379,000,000	$241,071,802	$1,045,713,802	20
2	3	Apr 22, 2015	Avengers: Age of Ultron	$365,000,000	$459,005,868	$1,395,316,979	20
3	4	Dec 16, 2015	Star Wars Episode VII: The Force Awakens	$306,000,000	$936,662,225	$2,064,615,817	20
4	5	Apr 25, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,044,540,523	20
...	...	...	...	...	...	...	
4810	5754	Oct 22, 1970	Il conformista	$750,000	$59,656	$89,609	19
4814	5763	Jun 25, 1968	La mariée était en noir	$747,000	$44,566	$44,566	19
4925	6000	Jul 15, 2016	Embrace	$225,000	$412,340	$1,611,592	20
4952	6044	Oct 23, 2015	Lumea e a mea	$168,000	$0	$29,678	20
4992	6122	Oct 27, 1980	Pepi, Luci, Bom y otras chicas del montón	$46,000	$48,363	$48,363	19

5125 rows × 14 columns

1.4.2 Handling Left-out Titles

In [66]:
```
#Find unmerged rows
un_merged_1 = left_outer_merge(tn_df,movies_df,'simple_title')
un_merged_1
```

Out[66]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYea
481	440	Jan 20, 2012	Jin ling shi san chai	$100,000,000	$311,434	$98,227,017	201
511	465	Jan 29, 2015	Kingsman: The Secret Service	$94,000,000	$128,261,724	$404,561,724	201
680	623	Nov 22, 2006	Déjà Vu	$80,000,000	$64,038,616	$181,038,616	200
703	646	Apr 22, 2010	Oceans	$80,000,000	$19,422,319	$86,787,530	201
707	650	Jan 12, 2007	Arthur et les Minimoys	$80,000,000	$15,132,763	$113,325,743	200
...	...	...	...	...	...	...	
6488	6179	Unknown	Red 11	$7,000	$0	$0	Unknow
6489	6180	Apr 2, 1999	Following	$6,000	$48,482	$240,495	199
6490	6181	Jul 13, 2005	Return to the Land of Wonders	$5,000	$1,338	$1,338	200
6491	6182	Sep 29, 2015	A Plague So Pleasant	$1,400	$0	$0	201
6492	6183	Aug 5, 2005	My Date With Drew	$1,100	$181,041	$181,041	200

1153 rows × 8 columns

In [67]:
```
#Remove 'the','and' on the unmerged rows and try again
df = concat_with_movies(un_merged_1,imdb_df,replace=True)
movies_df = pd.concat([movies_df,df])
movies_df
```

Out[67]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYear
0	1	Apr 23, 2019	Avengers: Endgame	$400,000,000	$858,373,000	$2,797,800,564	2019
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$379,000,000	$241,071,802	$1,045,713,802	2011
2	3	Apr 22, 2015	Avengers: Age of Ultron	$365,000,000	$459,005,868	$1,395,316,979	2015
3	4	Dec 16, 2015	Star Wars Episode VII: The Force Awakens	$306,000,000	$936,662,225	$2,064,615,817	2015
4	5	Apr 25, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,044,540,523	2018
...	...	...	...	...	...	...	...
14	4726	May 23, 1973	Pat Garrett and Billy the Kid	$4,638,783	$8,000,000	$11,000,000	1973
15	5415	Jan 27, 2006	Bubble	$1,600,000	$145,382	$145,382	2006
16	5686	Mar 10, 2015	The Pet	$1,000,000	$0	$0	2015
17	5727	Aug 3, 2012	Celeste and Jesse Forever	$840,000	$3,103,407	$3,787,689	2012
18	5840	Jul 11, 2003	The Journey	$500,000	$19,800	$19,800	2003

5144 rows × 14 columns

1.4.3 Merging Titles with Different Release Years

In [68]:
```
#Find missing rows
un_merged_2 = left_outer_merge(tn_df,movies_df,'simple_title')
un_merged_2
```

Out[68]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYea
481	440	Jan 20, 2012	Jin ling shi san chai	$100,000,000	$311,434	$98,227,017	201
511	465	Jan 29, 2015	Kingsman: The Secret Service	$94,000,000	$128,261,724	$404,561,724	201
680	623	Nov 22, 2006	Déjà Vu	$80,000,000	$64,038,616	$181,038,616	200
703	646	Apr 22, 2010	Oceans	$80,000,000	$19,422,319	$86,787,530	201
707	650	Jan 12, 2007	Arthur et les Minimoys	$80,000,000	$15,132,763	$113,325,743	200
...	...	...	...	...	...	...	
6488	6179	Unknown	Red 11	$7,000	$0	$0	Unknow
6489	6180	Apr 2, 1999	Following	$6,000	$48,482	$240,495	199
6490	6181	Jul 13, 2005	Return to the Land of Wonders	$5,000	$1,338	$1,338	200
6491	6182	Sep 29, 2015	A Plague So Pleasant	$1,400	$0	$0	201
6492	6183	Aug 5, 2005	My Date With Drew	$1,100	$181,041	$181,041	200

1145 rows × 8 columns

Now, let's try to merge `un_merged_2` using titles and check if the release years in `tn_df` and `imdb_df` are within 5 years of each other

In [69]:
```
#Function that returns string of years ± 3 years in 'un_merged_2' dataframe
def list_year(year,gap):
    try:
        year_split = [str(int(year)+i) for i in range(-gap,gap+1)]
        return ','.join(year_split)
    except:
        return year
```

In [70]:
```
#Merge un_merged_2 and imdb_df on title alone first with replacements
mer_2_titl = concat_with_movies(un_merged_2,imdb_df,
                                        replace=True,
                                        merge_cols='simple_title')
mer_2_titl
```

Out[70]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYe
0	465	Jan 29, 2015	Kingsman: The Secret Service	$94,000,000	$128,261,724	$404,561,724	2
1	623	Nov 22, 2006	Déjà Vu	$80,000,000	$64,038,616	$181,038,616	2
2	623	Nov 22, 2006	Déjà Vu	$80,000,000	$64,038,616	$181,038,616	2
3	623	Nov 22, 2006	Déjà Vu	$80,000,000	$64,038,616	$181,038,616	2
4	623	Nov 22, 2006	Déjà Vu	$80,000,000	$64,038,616	$181,038,616	2
...	...	...	...	...	...	...	
1692	6035	Jan 22, 1999	Bacheha-Ye aseman	$180,000	$933,933	$1,628,579	1
1697	6037	Jun 16, 2017	Arrowhead	$180,000	$0	$0	2
1707	6064	Jun 30, 2006	The Blood of My Brother: A Story of Death in Iraq	$120,000	$0	$0	2
1757	6148	Jan 14, 2000	The Terrorist	$25,000	$195,043	$195,043	2
1785	6167	Mar 9, 2001	Dayereh	$10,000	$673,780	$683,509	2

2190 rows × 15 columns

Note that `mer_2_titl` has two `startYear` columns. We now apply `list_year` function to `startYear_x` column and check if `startYear_y` is in `startYear_x` .

In [71]:
```
#Make a list of years ± 3 years in 'startYear_x'
gap = 3
mer_2_titl['startYear_x'] = mer_2_titl['startYear_x'].apply(lambda x: list_y
```

In [72]:
```
#Select values 'startYear_y' that are in 'startYear_x' and store them in col
mer_2_titl['isin'] = [x[0] in x[1] for x in zip(mer_2_titl['startYear_y'],me
mer_2_titl = mer_2_titl[mer_2_titl['isin']]

#Select all columns except 'startYear_x' and 'isin'
mer_2_titl.drop(['startYear_x','isin'],axis=1,inplace=True)

#Rename startYear_y to simply startYear
mer_2_titl.rename(columns = {'startYear_y':'startYear'}, inplace = True)
mer_2_titl
```

```
C:\Users\zeaps\Anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py:
4163: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  return super().drop(
C:\Users\zeaps\Anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py:
4296: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  return super().rename(
```

Out[72]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	465	Jan 29, 2015	Kingsman: The Secret Service	$94,000,000	$128,261,724	$404,561,724
5	623	Nov 22, 2006	Déjà Vu	$80,000,000	$64,038,616	$181,038,616
7	623	Nov 22, 2006	Déjà Vu	$80,000,000	$64,038,616	$181,038,616
10	646	Apr 22, 2010	Oceans	$80,000,000	$19,422,319	$86,787,530
12	795	May 9, 2014	Legends of Oz: Dorothy's Return	$70,000,000	$8,462,347	$20,107,933
...	...	...	...	...	...	...
1675	6006	Jan 18, 1967	Per un pugno di dollari	$200,000	$3,500,000	$3,527,522
1692	6035	Jan 22, 1999	Bacheha-Ye aseman	$180,000	$933,933	$1,628,579
1697	6037	Jun 16, 2017	Arrowhead	$180,000	$0	$0

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	
1707	6064	Jun 30, 2006	The Blood of My Brother: A Story of Death in Iraq	$120,000	$0	$0	bloo
1785	6167	Mar 9, 2001	Dayereh	$10,000	$673,780	$683,509	

1012 rows × 14 columns

Despite the effort, however, it's possible that one title might be merged with multiple titles from `imdb_df`. Therefore, simply groupinig the table based on `id` and then taking the first entry is going to solve this issue.

In [73]:
```
#Take the first in grouped rows by id and then reset the index
mer_2_titl = mer_2_titl.groupby('id').first().reset_index()
mer_2_titl
```

Out[73]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	
0	440	Jan 20, 2012	Jin ling shi san chai	$100,000,000	$311,434	$98,227,017	jinl
1	465	Jan 29, 2015	Kingsman: The Secret Service	$94,000,000	$128,261,724	$404,561,724	kingsmar
2	623	Nov 22, 2006	Déjà Vu	$80,000,000	$64,038,616	$181,038,616	
3	646	Apr 22, 2010	Oceans	$80,000,000	$19,422,319	$86,787,530	
4	650	Jan 12, 2007	Arthur et les Minimoys	$80,000,000	$15,132,763	$113,325,743	arthure
...	...	...	...	...	...	...	
864	6177	May 26, 2006	Cavite	$7,000	$70,071	$71,644	
865	6180	Apr 2, 1999	Following	$6,000	$48,482	$240,495	
866	6181	Jul 13, 2005	Return to the Land of Wonders	$5,000	$1,338	$1,338	returntol
867	6182	Sep 29, 2015	A Plague So Pleasant	$1,400	$0	$0	aplag
868	6183	Aug 5, 2005	My Date With Drew	$1,100	$181,041	$181,041	my

869 rows × 14 columns

Now `mer_2_titl` can be concatenated with `movies_df`.

In [74]:
```
#Contactenate mer_2_titl and movies_df
movies_df = pd.concat([movies_df,mer_2_titl])
movies_df
```

Out[74]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYea
0	1	Apr 23, 2019	Avengers: Endgame	$400,000,000	$858,373,000	$2,797,800,564	201
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$379,000,000	$241,071,802	$1,045,713,802	201
2	3	Apr 22, 2015	Avengers: Age of Ultron	$365,000,000	$459,005,868	$1,395,316,979	201
3	4	Dec 16, 2015	Star Wars Episode VII: The Force Awakens	$306,000,000	$936,662,225	$2,064,615,817	201
4	5	Apr 25, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,044,540,523	201
...	...	...	...	...	...	...	..
864	6177	May 26, 2006	Cavite	$7,000	$70,071	$71,644	200
865	6180	Apr 2, 1999	Following	$6,000	$48,482	$240,495	199
866	6181	Jul 13, 2005	Return to the Land of Wonders	$5,000	$1,338	$1,338	200
867	6182	Sep 29, 2015	A Plague So Pleasant	$1,400	$0	$0	201
868	6183	Aug 5, 2005	My Date With Drew	$1,100	$181,041	$181,041	200

6013 rows × 14 columns

1.4.4 Movies with Unknown Release Year

Now, we match the titles with unknown release data in `tn_df` with `imdb_df` that do not appear in `movies_df` .

In [75]:
```
#Select titles with unknown 'startYear' that are not movies_df
year_unk_df = left_outer_merge(tn_df[tn_df['startYear']=='Unknown'],movies_d

#Merge 'year_unk_df' and 'imdb_df' on title alone
unk_merge = concat_with_movies(year_unk_df,imdb_df,
                               replace=True,
                               merge_cols='simple_title')
unk_merge
```

Out[75]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYea
0	799	Unknown	b	$70,000,000	$0	$0	Unknc
1	1856	Unknown	Black Water Transit	$35,000,000	$0	$0	Unknc
2	1922	Unknown	George and the Dragon	$32,000,000	$0	$0	Unknc
3	1922	Unknown	George and the Dragon	$32,000,000	$0	$0	Unknc
4	2121	Unknown	Die Konferenz der Tiere	$30,000,000	$0	$53,048,539	Unknc
...	...	...	...	...	...	...	
119	5386	Unknown	Deadline	$1,800,000	$0	$0	Unknc
135	5489	Unknown	Kurmanjan datka	$1,400,000	$0	$0	Unknc
152	5849	Unknown	Feng kuang de shi tou	$500,000	$0	$3,000,000	Unknc
157	5863	Unknown	El rey de Najayo	$500,000	$0	$0	Unknc
159	5871	Unknown	Rodeo Girl	$500,000	$0	$0	Unknc

233 rows × 15 columns

Similar to earlier, `unk_merge` will have `startYear_x` and `startYear_y` columns.

In [76]:
```
#Drop 'startYear_x' and rename 'startYear_y' to 'startYear'
unk_merge.drop('startYear_x',axis=1, inplace=True)
unk_merge.rename(columns = {'startYear_y':'startYear'}, inplace = True)

#Take the first in grouped rows by id and then reset the index as a precauti
unk_merge = unk_merge.groupby('id').first().reset_index()
unk_merge
```

Out[76]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	si
0	799	Unknown	b	$70,000,000	$0	$0	
1	1700	Unknown	Obitaemyy ostrov	$36,500,000	$0	$15,000,000	obitae
2	1856	Unknown	Black Water Transit	$35,000,000	$0	$0	blackw
3	1922	Unknown	George and the Dragon	$32,000,000	$0	$0	geo
4	2121	Unknown	Die Konferenz der Tiere	$30,000,000	$0	$53,048,539	diekonfere
...	...	...	...	...	...	...	
90	6154	Unknown	Dry Spell	$22,000	$0	$0	
91	6157	Unknown	Flywheel	$20,000	$0	$0	
92	6163	Unknown	Stories of Our Lives	$15,000	$0	$0	stories
93	6166	Unknown	Tin Can Man	$12,000	$0	$0	
94	6179	Unknown	Red 11	$7,000	$0	$0	

95 rows × 14 columns

In [77]:
```
#Merge unk_merge with movies_df
movies_df = pd.concat([movies_df,unk_merge])
movies_df
```

Out[77]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYear
0	1	Apr 23, 2019	Avengers: Endgame	$400,000,000	$858,373,000	$2,797,800,564	2019
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$379,000,000	$241,071,802	$1,045,713,802	2011
2	3	Apr 22, 2015	Avengers: Age of Ultron	$365,000,000	$459,005,868	$1,395,316,979	2015
3	4	Dec 16, 2015	Star Wars Episode VII: The Force Awakens	$306,000,000	$936,662,225	$2,064,615,817	2015
4	5	Apr 25, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,044,540,523	2018
...	...	...	...	...	...	...	...
90	6154	Unknown	Dry Spell	$22,000	$0	$0	2005
91	6157	Unknown	Flywheel	$20,000	$0	$0	2003
92	6163	Unknown	Stories of Our Lives	$15,000	$0	$0	2014
93	6166	Unknown	Tin Can Man	$12,000	$0	$0	2007
94	6179	Unknown	Red 11	$7,000	$0	$0	2019

6108 rows × 14 columns

1.4.5 Merge Remaining Titles by Title Only

As a last effort, we are going to merge the remaining dataset in `tn_df` .

In [78]:
```
#Find the titles that are not merged yet.
remaining_df = left_outer_merge(tn_df,movies_df,'movie')
name_only = concat_with_movies(remaining_df,imdb_df,
                               replace=True,
                               merge_cols='simple_title')
name_only
```

Out[78]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYear_
0	1624	Mar 16, 2007	Nomad	$40,000,000	$79,123	$79,123	200
1	1624	Mar 16, 2007	Nomad	$40,000,000	$79,123	$79,123	200
2	1624	Mar 16, 2007	Nomad	$40,000,000	$79,123	$79,123	200
3	1624	Mar 16, 2007	Nomad	$40,000,000	$79,123	$79,123	200
4	1624	Mar 16, 2007	Nomad	$40,000,000	$79,123	$79,123	200
5	1624	Mar 16, 2007	Nomad	$40,000,000	$79,123	$79,123	200
6	1624	Mar 16, 2007	Nomad	$40,000,000	$79,123	$79,123	200
7	3007	Jul 9, 2002	My Spy	$18,000,000	$0	$6,983,461	200
8	3007	Jul 9, 2002	My Spy	$18,000,000	$0	$6,983,461	200
9	3330	Apr 25, 2003	Steal	$15,000,000	$220,944	$240,025	200
10	3330	Apr 25, 2003	Steal	$15,000,000	$220,944	$240,025	200
11	3663	Jun 7, 1997	Madadayo	$11,900,000	$48,856	$49,451	199
12	3848	Oct 20, 2000	The Legend of Drunken Master	$10,000,000	$11,546,543	$11,546,543	200
13	3980	May 19, 2015	Grizzly	$10,000,000	$0	$0	201
14	4066	Jun 17, 2014	Beneath Hill 60	$9,000,000	$0	$3,440,939	201
15	4677	Jun 13, 2003	Tycoon	$5,000,000	$121,016	$121,016	200
16	4677	Jun 13, 2003	Tycoon	$5,000,000	$121,016	$121,016	200
17	4898	Jun 21, 2013	L'attentat	$3,750,000	$1,625,707	$3,237,975	201
18	5048	Feb 28, 1997	Kama Sutra	$3,000,000	$4,109,095	$4,109,095	199
19	5179	Apr 9, 2010	Valley of the Hearts Delight	$2,500,000	$118,666	$118,666	201
20	5194	May 9, 2020	Not Safe For Work	$2,500,000	$0	$0	202
21	5387	Mar 13, 2007	Sublime	$1,800,000	$0	$0	200
22	5396	Apr 23, 2019	Living Dark: The Story of Ted the Caver	$1,750,000	$0	$0	201

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYear_
23	5541	Feb 11, 2011	Certifiably Jonathan	$1,200,000	$7,826	$7,826	201
24	5662	May 15, 2015	Childless	$1,000,000	$1,036	$1,036	201
25	5671	Oct 24, 2003	The Party's Over	$1,000,000	$0	$0	200
26	5671	Oct 24, 2003	The Party's Over	$1,000,000	$0	$0	200
27	5671	Oct 24, 2003	The Party's Over	$1,000,000	$0	$0	200
28	5671	Oct 24, 2003	The Party's Over	$1,000,000	$0	$0	200
29	5758	Oct 11, 2013	All the Boys Love Mandy Lane	$750,000	$0	$1,960,521	201
30	5890	Apr 22, 2021	Kiss Me Deadly	$410,000	$0	$217	202
31	5938	Nov 9, 2012	Nothing But a Man	$300,000	$17,241	$17,241	201
32	6087	Mar 25, 2015	Open Secret	$100,000	$0	$0	201
33	6087	Mar 25, 2015	Open Secret	$100,000	$0	$0	201
34	6087	Mar 25, 2015	Open Secret	$100,000	$0	$0	201
35	6087	Mar 25, 2015	Open Secret	$100,000	$0	$0	201
36	6130	Jul 1, 1991	The Last Waltz	$35,000	$321,952	$322,563	199
37	6130	Jul 1, 1991	The Last Waltz	$35,000	$321,952	$322,563	199
38	6130	Jul 1, 1991	The Last Waltz	$35,000	$321,952	$322,563	199
39	6130	Jul 1, 1991	The Last Waltz	$35,000	$321,952	$322,563	199
40	6144	Dec 24, 1999	Pink Narcissus	$27,000	$8,231	$8,231	199
41	6164	Apr 11, 1997	Pink Flamingos	$12,000	$413,802	$413,802	199
12	4898	Jun 21, 2013	L'attentat	$3,750,000	$1,625,707	$3,237,975	201
13	4898	Jun 21, 2013	L'attentat	$3,750,000	$1,625,707	$3,237,975	201
25	5718	May 5, 2015	Le bonheur d'Elza	$900,000	$0	$0	201

To merge the remaining titles, the nearest values of `startYear_y` to `startYear_x` were chosen.

In [79]:

```
#Identify the unique ids to be iterated
unique_ids = name_only['id'].unique()

#Cast years to int
name_only[['startYear_x','startYear_y']] = name_only[['startYear_x','startYe

#Loop through each unique id, collect df with similar id,
#find the closest year by subtracting startYear_x from startYear_y
# and keep the closest value
df = name_only.copy()
for id_ in unique_ids:
    tmp = name_only[name_only['id']==id_]
    df_idx = tmp.index.tolist()
    #Calculated to differences from 'startYear_x'
    diff = [abs(i-j) for i,j in zip(tmp['startYear_x'],tmp['startYear_y'])]
    idx = df_idx[diff.index(min(diff))]
    df_idx.remove(idx)
    df.drop(df_idx,axis=0,inplace=True)
name_only = df

#Change the years back to str
name_only[['startYear_x','startYear_y']] = name_only[['startYear_x','startYe
name_only
```

Out[79]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYear_
1	1624	Mar 16, 2007	Nomad	$40,000,000	$79,123	$79,123	200
7	3007	Jul 9, 2002	My Spy	$18,000,000	$0	$6,983,461	200
9	3330	Apr 25, 2003	Steal	$15,000,000	$220,944	$240,025	200
11	3663	Jun 7, 1997	Madadayo	$11,900,000	$48,856	$49,451	199
13	3980	May 19, 2015	Grizzly	$10,000,000	$0	$0	201
14	4066	Jun 17, 2014	Beneath Hill 60	$9,000,000	$0	$3,440,939	201
16	4677	Jun 13, 2003	Tycoon	$5,000,000	$121,016	$121,016	200
18	5048	Feb 28, 1997	Kama Sutra	$3,000,000	$4,109,095	$4,109,095	199
19	5179	Apr 9, 2010	Valley of the Hearts Delight	$2,500,000	$118,666	$118,666	201
20	5194	May 9, 2020	Not Safe For Work	$2,500,000	$0	$0	202
21	5387	Mar 13, 2007	Sublime	$1,800,000	$0	$0	200
22	5396	Apr 23, 2019	Living Dark: The Story of Ted the Caver	$1,750,000	$0	$0	201
23	5541	Feb 11, 2011	Certifiably Jonathan	$1,200,000	$7,826	$7,826	201

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYear_
24	5662	May 15, 2015	Childless	$1,000,000	$1,036	$1,036	201
27	5671	Oct 24, 2003	The Party's Over	$1,000,000	$0	$0	200
29	5758	Oct 11, 2013	All the Boys Love Mandy Lane	$750,000	$0	$1,960,521	201
30	5890	Apr 22, 2021	Kiss Me Deadly	$410,000	$0	$217	202
31	5938	Nov 9, 2012	Nothing But a Man	$300,000	$17,241	$17,241	201
35	6087	Mar 25, 2015	Open Secret	$100,000	$0	$0	201
38	6130	Jul 1, 1991	The Last Waltz	$35,000	$321,952	$322,563	199
40	6144	Dec 24, 1999	Pink Narcissus	$27,000	$8,231	$8,231	199
41	6164	Apr 11, 1997	Pink Flamingos	$12,000	$413,802	$413,802	199
13	4898	Jun 21, 2013	L'attentat	$3,750,000	$1,625,707	$3,237,975	201

In [80]:
```
#Drop 'startYear_x' and rename 'startYear_y' to 'startYear'
name_only.drop('startYear_x',axis=1, inplace=True)
name_only.rename(columns = {'startYear_y':'startYear'}, inplace = True)

#Take the first in grouped rows by id and then reset the index as a precauti
name_only = name_only.groupby('id').first().reset_index()

#Merge name_only with movies_df
movies_df = pd.concat([movies_df,name_only])
movies_df
```

Out[80]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	startYear
0	1	Apr 23, 2019	Avengers: Endgame	$400,000,000	$858,373,000	$2,797,800,564	2019
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$379,000,000	$241,071,802	$1,045,713,802	2011
2	3	Apr 22, 2015	Avengers: Age of Ultron	$365,000,000	$459,005,868	$1,395,316,979	2015
3	4	Dec 16, 2015	Star Wars Episode VII: The Force Awakens	$306,000,000	$936,662,225	$2,064,615,817	2015
4	5	Apr 25, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,044,540,523	2018
...	...	...	...	...	...	...	...
18	5938	Nov 9, 2012	Nothing But a Man	$300,000	$17,241	$17,241	1964
19	6087	Mar 25, 2015	Open Secret	$100,000	$0	$0	2011
20	6130	Jul 1, 1991	The Last Waltz	$35,000	$321,952	$322,563	1978
21	6144	Dec 24, 1999	Pink Narcissus	$27,000	$8,231	$8,231	1971
22	6164	Apr 11, 1997	Pink Flamingos	$12,000	$413,802	$413,802	1972

6131 rows × 14 columns

Again, as a precaution, we have to group `movies_df` by `id`, reset the index and take the first value.

In [81]:
```
movies_df = movies_df.groupby('tconst').first().reset_index()

#Remove 'simple_title' column
movies_df.drop('simple_title',axis=1,inplace=True)
movies_df
```

Out[81]:

	tconst	id	release_date	movie	production_budget	domestic_gross	worldwide_gro
0	tt0001141	5984	Unknown	Butterfly	$250,000	$0	
1	tt0004972	6068	Feb 8, 1915	The Birth of a Nation	$110,000	$10,000,000	$11,000,(
2	tt0006333	6004	Dec 24, 1916	20,000 Leagues Under the Sea	$200,000	$8,000,000	$8,000,(
3	tt0006864	5908	Sep 5, 1916	Intolerance	$385,907	$0	
4	tt0011097	5386	Unknown	Deadline	$1,800,000	$0	
...	...	...	...	...	...	...	
6115	tt9801736	5128	Jul 26, 2019	Hvítur, Hvítur Dagur	$2,800,000	$0	$836,9
6116	tt9844522	3240	Jul 1, 2021	Escape Room: Tournament of Champions	$15,000,000	$25,188,958	$51,788,7
6117	tt9845110	5131	Feb 14, 2020	Deux	$2,750,000	$5,495	$558,(
6118	tt9889072	518	Apr 21, 2017	The Promise	$90,000,000	$8,224,288	$10,551,4
6119	tt9893078	5387	Mar 13, 2007	Sublime	$1,800,000	$0	

6120 rows × 13 columns

In [82]:
```
discarded_df = left_outer_merge(tn_df,movies_df,'movie')
print('{} titles were discarded.'.format(len(discarded_df)))
```

```
160 titles were discarded.
```

160 movies out of 6183 is not bad. We will now proceed to merging the `movies_df` with `princ_df`, `rating_df` and `names` using the `tconst` column.

In [83]:
```
movies_df = pd.merge(movies_df,rating_df,on='tconst')
movies_df
```

Out[83]:

	tconst	id	release_date	movie	production_budget	domestic_gross	worldwide_gro
0	tt0004972	6068	Feb 8, 1915	The Birth of a Nation	$110,000	$10,000,000	$11,000,(
1	tt0006333	6004	Dec 24, 1916	20,000 Leagues Under the Sea	$200,000	$8,000,000	$8,000,(
2	tt0006864	5908	Sep 5, 1916	Intolerance	$385,907	$0	
3	tt0011549	6074	Sep 17, 1920	Over the Hill to the Poorhouse	$100,000	$3,000,000	$3,000,(
4	tt0015624	5995	Nov 19, 1925	The Big Parade	$245,000	$11,000,000	$22,000,(
...	...	...	...	...	...	...	
6049	tt9784798	2273	Feb 12, 2021	Judas and the Black Messiah	$26,000,000	$5,453,633	$7,030,5
6050	tt9799878	409	Jun 21, 1996	The Hunchback of Notre Dame	$100,000,000	$100,138,851	$325,500,(
6051	tt9801736	5128	Jul 26, 2019	Hvítur, Hvítur Dagur	$2,800,000	$0	$836,9
6052	tt9844522	3240	Jul 1, 2021	Escape Room: Tournament of Champions	$15,000,000	$25,188,958	$51,788,7
6053	tt9845110	5131	Feb 14, 2020	Deux	$2,750,000	$5,495	$558,(

6054 rows × 15 columns

In [84]:
```
princ_df = pd.merge(princ_df[['tconst','nconst','category']],movies_df['tcon
princ_df.drop_duplicates(inplace=True)
princ_df
```

Out[84]:

	tconst	nconst	category
0	tt0004972	nm0376221	editor
1	tt0004972	nm0001273	actress
2	tt0004972	nm0550615	actress
3	tt0004972	nm0910400	actor
4	tt0004972	nm0178270	actress
...	...	...	...
59737	tt9845110	nm2889273	director
59738	tt9845110	nm11042781	writer
59739	tt9845110	nm0897095	writer
59740	tt9845110	nm2672365	producer
59741	tt9845110	nm2661750	producer

59742 rows × 3 columns

In [85]:
```
names = pd.merge(names[['nconst','primaryName']],princ_df['nconst'],on='ncon
names.drop_duplicates(inplace=True)
names
```

Out[85]:

	nconst	primaryName
0	nm0000001	Fred Astaire
1	nm0000002	Lauren Bacall
5	nm0000003	Brigitte Bardot
6	nm0000004	John Belushi
9	nm0000005	Ingmar Bergman
...	...	...
59736	nm9980882	Alwine Ickert
59737	nm9982380	Gabriel Sky
59738	nm9982544	Khadijha Red Thunder
59739	nm9985209	Christiane Henckel von Donnersmarck
59740	nm9985727	Carlos Peralta

26989 rows × 2 columns

In [86]:

```
#Merge principals and names
princ_df = pd.merge(princ_df,names,on='nconst')
princ_df.drop('nconst',axis=1,inplace=True)

#Pivot table by 'tconst' and 'category'
pv_prin_df = princ_df.groupby(['tconst','category'])['primaryName'].apply(li
pv_prin_df = pv_prin_df.reset_index()
pv_prin_df = pv_prin_df.pivot(index='tconst', columns='category', values='pr
pv_prin_df = pv_prin_df.reset_index()
pv_prin_df
```

Out[86]:

category	tconst	actor	actress	archive_footage	archive_sound	cinematographer	co
0	tt0004972	[Henry B. Walthall]	[Lillian Gish, Mae Marsh, Miriam Cooper]	NaN	NaN	[G.W. Bitzer]	C
1	tt0006333	[Allen Holubar, Dan Hanlon, Curtis Benton]	[Edna Pendleton]	NaN	NaN	[Eugene Gaudio]	[Ma Ma
2	tt0006864	[Robert Harron, F.A. Turner]	[Lillian Gish, Mae Marsh]	NaN	NaN	NaN	
3	tt0011549	[James Sheridan, Noel Tearle, William Welsh]	[Mary Carr]	NaN	NaN	[Harold S. Sintzenich, George Schneiderman]	
4	tt0015624	[John Gilbert, Hobart Bosworth]	[Renée Adorée, Claire McDowell]	NaN	NaN	NaN	
...	...	...	...	...	...	...	
6045	tt9784798	[LaKeith Stanfield, Jesse Plemons, Daniel Kalu...	[Dominique Fishback]	NaN	NaN	NaN	
6046	tt9799878	[Ilona Strauss, Georg Feils, Armin Drogat]	[Nicole Zimmer]	NaN	NaN	NaN	
6047	tt9801736	[Ingvar Sigurdsson, Hilmir Snær Guðnason]	[Ída Mekkín Hlynsdóttir, Sara Dögg Ásgeirsdóttir]	NaN	NaN	[Maria von Hausswolff]	[

	category	tconst	actor	actress	archive_footage	archive_sound	cinematographer	co
6048		tt9844522	[Logan Miller, Thomas Cocquerel]	[Taylor Russell, Deborah Ann Woll]	NaN	NaN	NaN	
6049		tt9845110	[Jérôme Varanfrain]	[Léa Drucker, Barbara Sukowa, Martine Chevallier]	NaN	NaN	NaN	

6050 rows × 13 columns

Now that our table has list of directors, writers, actor and so on, we need to stretch each role and append that to `movies_df` .

In [87]:
```
#Loop through each columns in 'pv_prin_df', stretch that to multiple columns
# change the column names and append that to 'movies_df'
new_cols = list(pv_prin_df.columns)
roles = pd.DataFrame()
for col in new_cols:
    df = pv_prin_df[col].apply(pd.Series)
    df.columns = [col+'_'+str(int(i)+1) for i in df.columns]
    if roles.empty:
        roles = df
    else:
        roles = pd.concat([roles,df],axis=1)

#Before merging to 'movies_df', the code above appends '_1' to 'tconst'. The
roles.rename(columns = {'tconst_1':'tconst'}, inplace = True)
roles
```

Out[87]:

	tconst	actor_1	actor_2	actor_3	actor_4	actor_5	actor_6	actor_7	actor_8	actor_
0	tt0004972	Henry B. Walthall	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
1	tt0006333	Allen Holubar	Dan Hanlon	Curtis Benton	NaN	NaN	NaN	NaN	NaN	Nal
2	tt0006864	Robert Harron	F.A. Turner	NaN	NaN	NaN	NaN	NaN	NaN	Nal
3	tt0011549	James Sheridan	Noel Tearle	William Welsh	NaN	NaN	NaN	NaN	NaN	Nal
4	tt0015624	John Gilbert	Hobart Bosworth	NaN	NaN	NaN	NaN	NaN	NaN	Nal
...	...	...	...	...	...	...	...	...	...	.
6045	tt9784798	LaKeith Stanfield	Jesse Plemons	Daniel Kaluuya	NaN	NaN	NaN	NaN	NaN	Nal
6046	tt9799878	Ilona Strauss	Georg Feils	Armin Drogat	NaN	NaN	NaN	NaN	NaN	Nal
6047	tt9801736	Ingvar Sigurdsson	Hilmir Snær Guðnason	NaN	NaN	NaN	NaN	NaN	NaN	Nal
6048	tt9844522	Logan Miller	Thomas Cocquerel	NaN	NaN	NaN	NaN	NaN	NaN	Nal
6049	tt9845110	Jérôme Varanfrain	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal

6050 rows × 59 columns

In [88]:
```
#Now, 'movies_df' and roles can be merged using 'tconst'
movies_df = pd.merge(movies_df,roles,on='tconst')

#It is important to avoid repeated rows therefore we need to get rid of thos
movies_df = movies_df.groupby('tconst').first().reset_index()
movies_df = movies_df.sort_values('id')
```

The cleaned data should look like this:

In [89]:
```
movies_df
```

Out[89]:

	tconst	id	release_date	movie	production_budget	domestic_gross	worldwide_gro
5544	tt4154796	1	Apr 23, 2019	Avengers: Endgame	$400,000,000	$858,373,000	$2,797,800,5
4028	tt1298650	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$379,000,000	$241,071,802	$1,045,713,8
5043	tt2395427	3	Apr 22, 2015	Avengers: Age of Ultron	$365,000,000	$459,005,868	$1,395,316,9
5091	tt2488496	4	Dec 16, 2015	Star Wars Episode VII: The Force Awakens	$306,000,000	$936,662,225	$2,064,615,8
5543	tt4154756	5	Apr 25, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,044,540,5
...	...	...	...	...	...	...	
5974	tt7837402	6179	Unknown	Red 11	$7,000	$0	
1401	tt0154506	6180	Apr 2, 1999	Following	$6,000	$48,482	$240,4
3001	tt0461832	6181	Jul 13, 2005	Return to the Land of Wonders	$5,000	$1,338	$1,3
4853	tt2107644	6182	Sep 29, 2015	A Plague So Pleasant	$1,400	$0	
2520	tt0378407	6183	Aug 5, 2005	My Date With Drew	$1,100	$181,041	$181,0

6050 rows × 73 columns

1.5 Analysis

In this section, the visualization aspect of the project is being discussed. In the subsequent code blocks, plots are generated that could be used as key indicators for making important decisions. In this analysis, the following plots were made

- Top 10 high budget movies
- Average production budget since 2000
- Relationship between budget and gross
- Top 10 domestic and worldwide grossing movie
- Budget versus customer satisfaction
- Grossing by genre
- Actors and directors that were involved in cumulatively highest grossing movies

In [90]:
```
#Import libraries
import matplotlib.pyplot as plt
import seaborn as sns
import re
from textwrap import wrap
from collections import Counter
%matplotlib inline
```

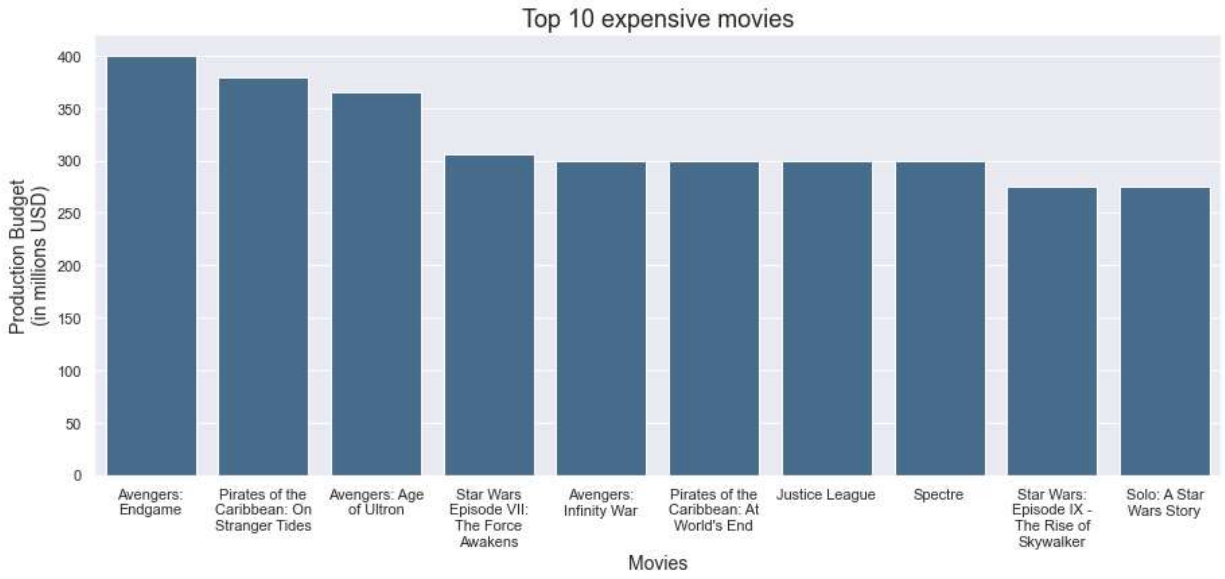
In [91]:
```
#Function to change the production budget, domestic and worldwide gross into
def enum(value):
    try:
        return int(''.join(re.findall(r'\d+',value)))
    except:
        return value

movies_df['production_budget'] = movies_df['production_budget'].map(enum)
movies_df['domestic_gross'] = movies_df['domestic_gross'].map(enum)
movies_df['worldwide_gross'] = movies_df['worldwide_gross'].map(enum)
movies_df['startYear'] = movies_df['startYear'].map(enum)
```

1.5.1 Top Ten High Budget Movies

For starters, let's find out the top 10 movies with the highest budgets.

In [92]:

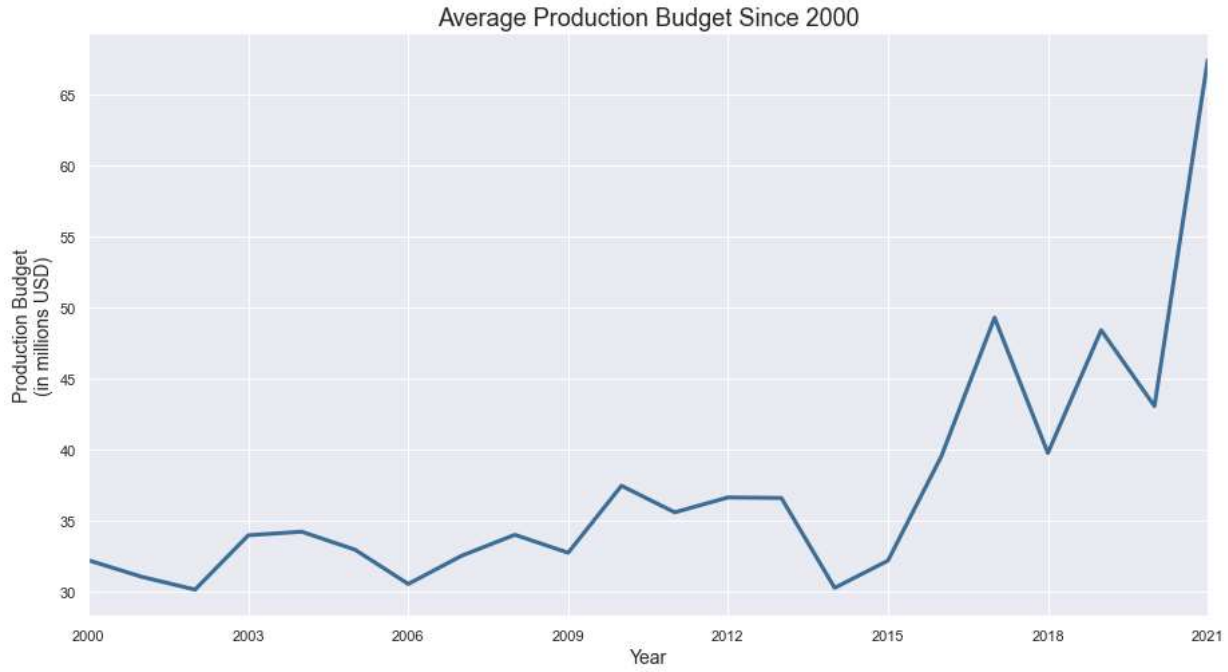
```
#Plot for the top 10 movie budgets
#Set image size
sns.set(rc={"figure.figsize":(15, 6)})
#Top 10 expensive movies to make
df = movies_df.sort_values('production_budget',ascending=False)[:10]
df['production_budget'] = df['production_budget']/1e6
ax = sns.barplot(x='movie',
            y='production_budget',
            data=df[['movie','production_budget']],
            color="#3C6D97")
ax.set_xlabel('Movies',fontsize=14)
ax.set_ylabel('Production Budget\n(in millions USD)',fontsize=14)
ax.set_title('Top 10 expensive movies',fontsize=18);
labels = ax.axes.get_xticklabels()
# fix the labels
for v in labels:
    text = v.get_text()
    text = '\n'.join(wrap(text,15))
    v.set_text(text)
# set the new labels
ax.axes.set_xticklabels(labels);
plt.savefig('images/top_10_expensive.png')
```



1.5.2 Production Budget Since 2000

Now, let's plot how production budget has changed since the year 2000

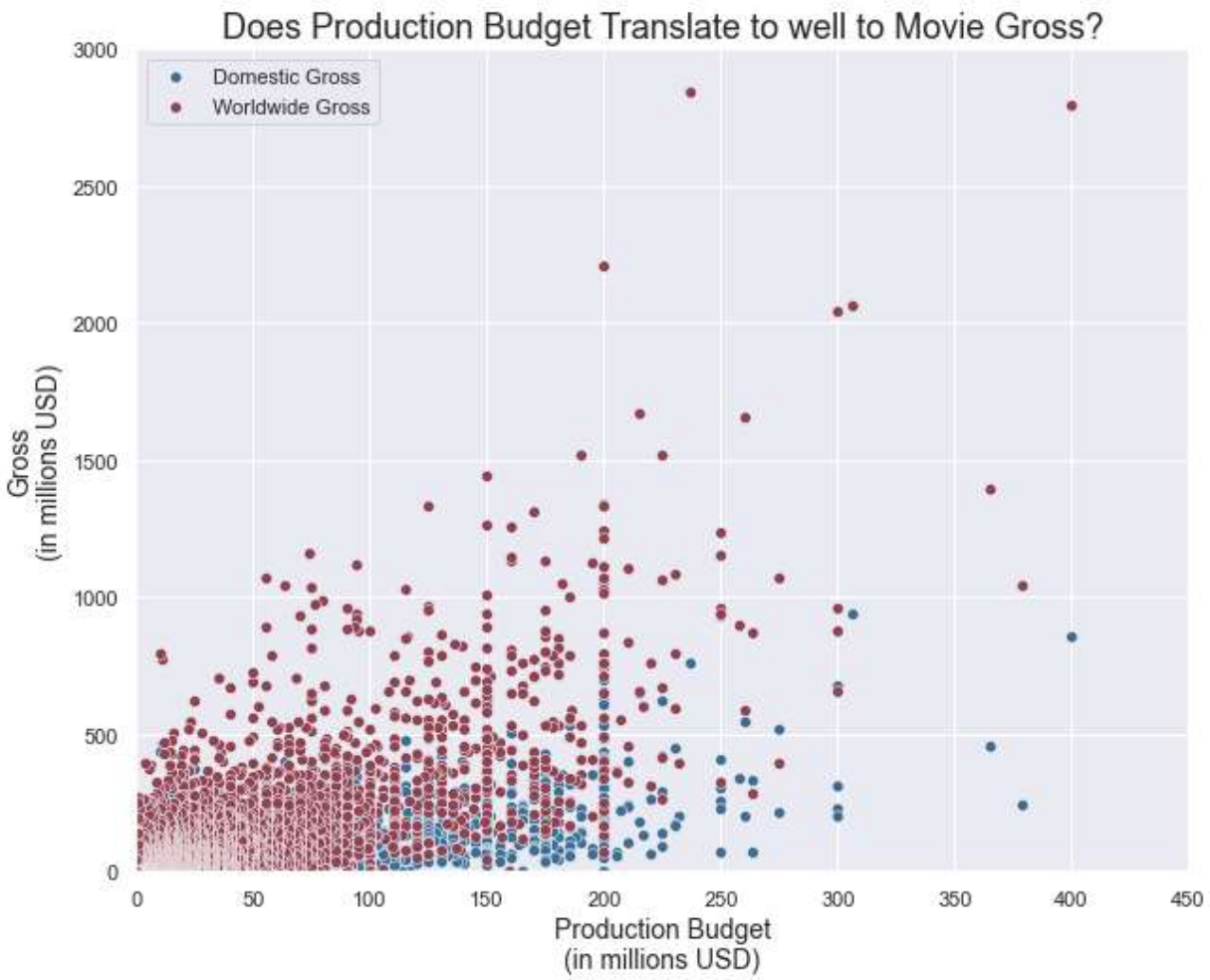
In [93]:
```
#Movies with release year
sns.set(rc={"figure.figsize":(15, 8)})
#Eliminate Unknows
movies_rel = movies_df[movies_df['startYear']!='Unknown']
#Aggregate by year
movie_trends = movies_rel.groupby('startYear')['production_budget'].agg(['me
movie_trends = movie_trends.reset_index()
#Choose movies released after 2000
movie_trends = movie_trends[movie_trends['startYear']>=2000]
#Show values in millions
movie_trends['mean'] = movie_trends['mean']/1e6
ax = sns.lineplot(data=movie_trends, x='startYear',y='mean',color="#3C6D97",
ax.set_xlabel('Year',fontsize=14)
ax.set_ylabel('Production Budget\n(in millions USD)',fontsize=14)
ax.set_title('Average Production Budget Since 2000',fontsize=18);
ax.set(xlim=(2000,2021))
ax.set_xticks(range(2000,2022,3))
plt.savefig('images/budget_by_year.png')
```



1.5.3 Budget versus Gross

Let's try to find out if there is a direct correlation between budget and movie gross.

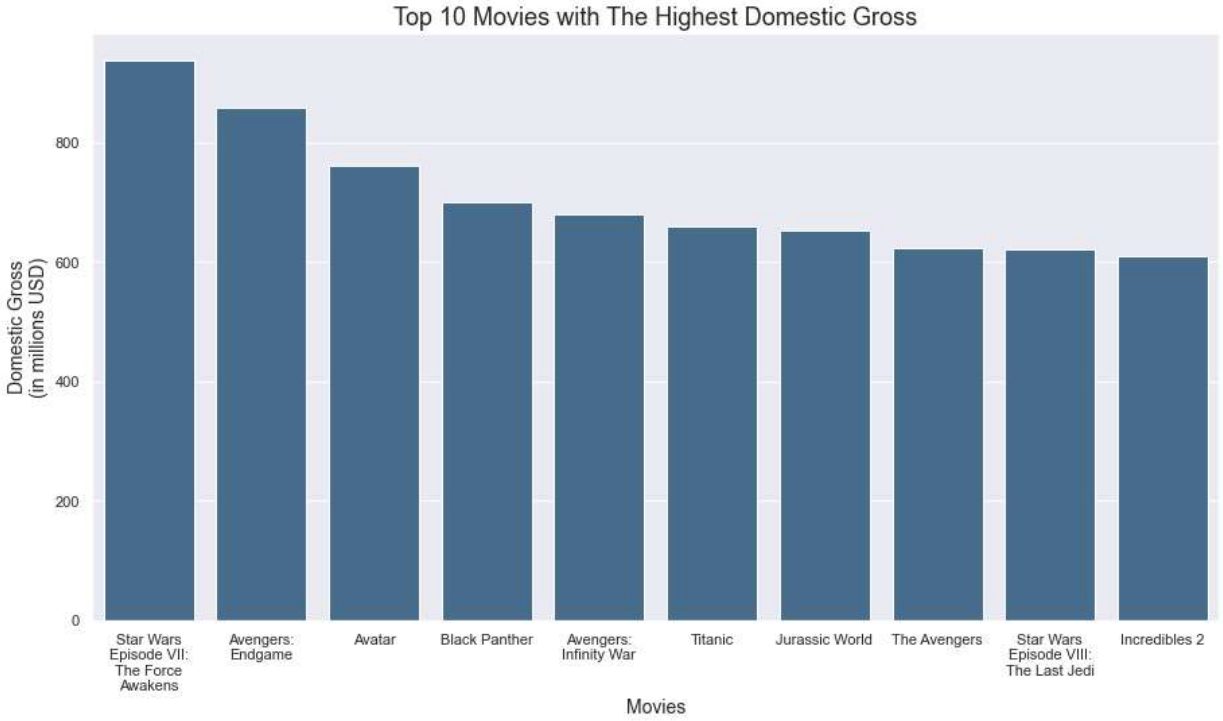
In [94]:
```
#Relationship between budget and gross
#Set image size
sns.set(rc={"figure.figsize":(10, 8)})
df = movies_df.copy()
#Show budget in millions
df['production_budget'] = df['production_budget']/1e6
df['domestic_gross'] = df['domestic_gross']/1e6
df['worldwide_gross'] = df['worldwide_gross']/1e6
ax_1 = sns.scatterplot(data=df, x='production_budget',y='domestic_gross',col
ax_2 = sns.scatterplot(data=df, x='production_budget',y='worldwide_gross',co
ax_2.set_xlabel('Production Budget\n(in millions USD)',fontsize=14)
ax_2.set_ylabel('Gross\n(in millions USD)',fontsize=14)
ax_2.set_title('Does Production Budget Translate to well to Movie Gross?',fo
plt.legend(labels=["Domestic Gross","Worldwide Gross"])
ax_2.set(xlim=(0,450),ylim=(0,3000))
plt.savefig('images/budget_and_gross.png')
```



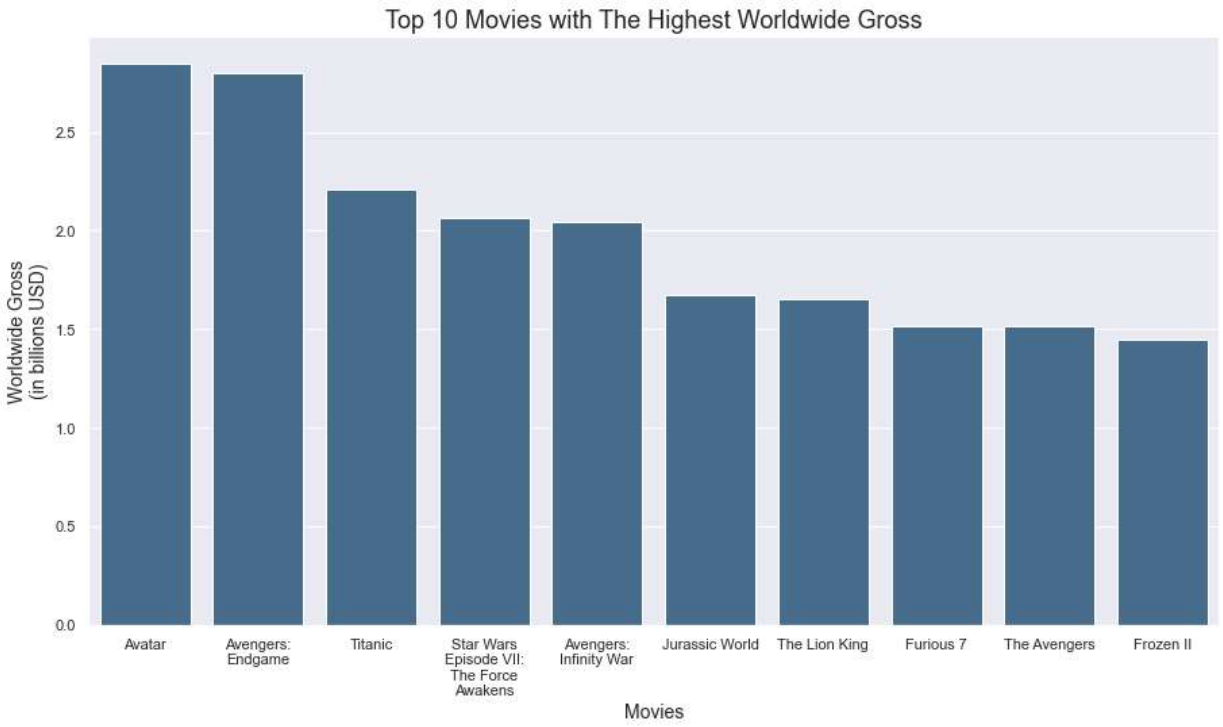
1.5.4 Highest Grossing Movies

To understand why some movies have lucrative gross, lets findout which movies earned the highest in domestic and worldwide gross.

In [95]:
```
#Set image size
sns.set(rc={"figure.figsize":(15, 8)})
#Find movies with the highest domestic gross
df = movies_df.sort_values('domestic_gross',ascending=False)[:10]
#Show values in millions
df['domestic_gross'] = df['domestic_gross']/1e6
ax = sns.barplot(x='movie',
            y='domestic_gross',
            data=df,
            color="#3C6D97")
ax.set_xlabel('Movies',fontsize=14)
ax.set_ylabel('Domestic Gross\n(in millions USD)',fontsize=14)
ax.set_title('Top 10 Movies with The Highest Domestic Gross',fontsize=18);
labels = ax.axes.get_xticklabels()
# fix the labels
for v in labels:
    text = v.get_text()
    text = '\n'.join(wrap(text,15))
    v.set_text(text)
# set the new labels
ax.axes.set_xticklabels(labels);
plt.savefig('images/top_grossing_domestic.png')
```



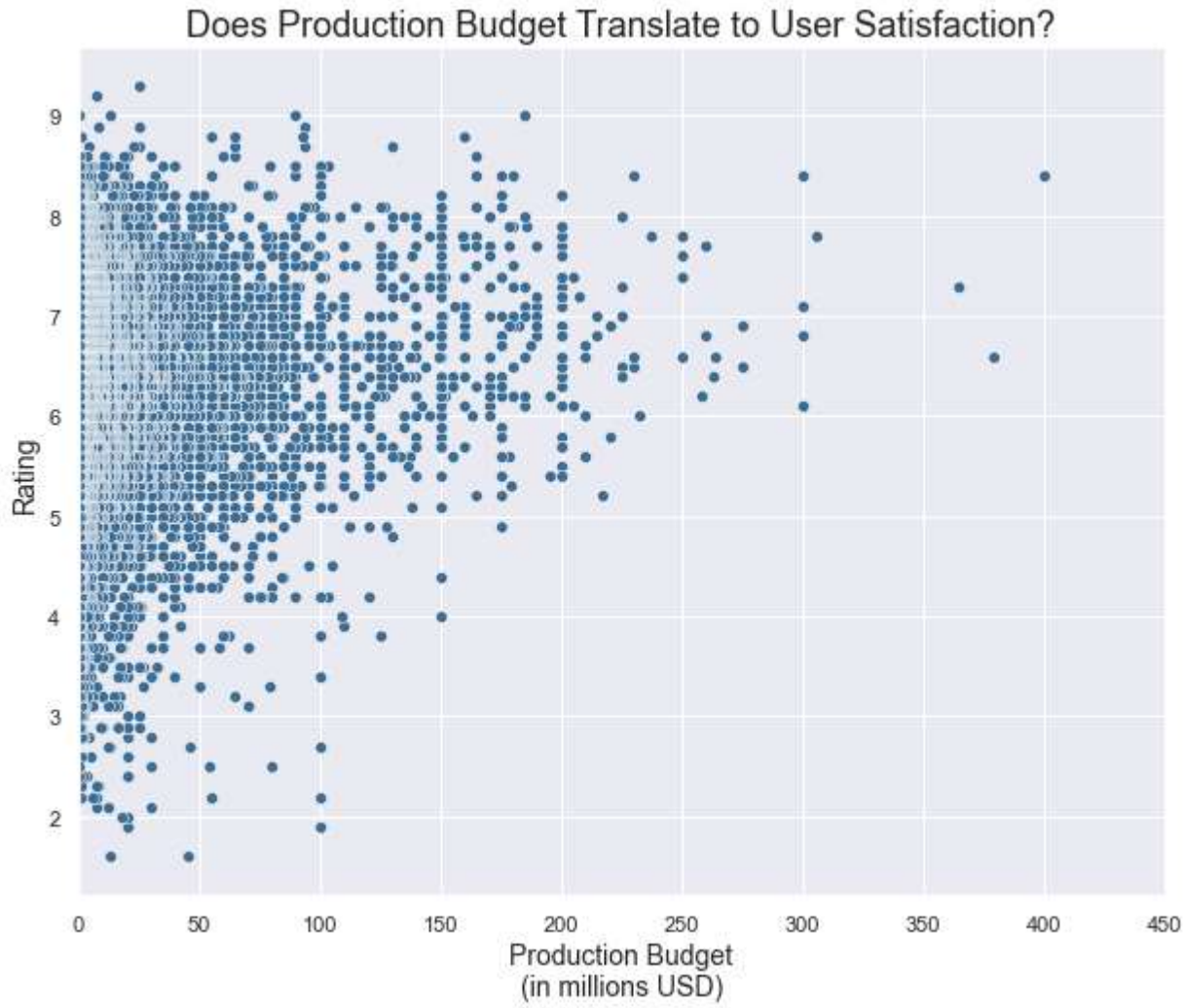
In [96]:
```
#Set image size
sns.set(rc={"figure.figsize":(15, 8)})
#Find movies with the highest worldwide gross
df = movies_df.sort_values('worldwide_gross',ascending=False)[:10]
#Show values in billions
df['worldwide_gross'] = df['worldwide_gross']/1e9
ax = sns.barplot(x='movie',
            y='worldwide_gross',
            data=df,
            color="#3C6D97")
ax.set_xlabel('Movies',fontsize=14)
ax.set_ylabel('Worldwide Gross\n(in billions USD)',fontsize=14)
ax.set_title('Top 10 Movies with The Highest Worldwide Gross',fontsize=18);
labels = ax.axes.get_xticklabels()
# fix the labels
for v in labels:
    text = v.get_text()
    text = '\n'.join(wrap(text,15))
    v.set_text(text)
# set the new labels
ax.axes.set_xticklabels(labels);
plt.savefig('images/top_grossing_world.png')
```



1.5.5 Rating versus Production Budget

Here, we need to understand if budget has strong correlation with rating.

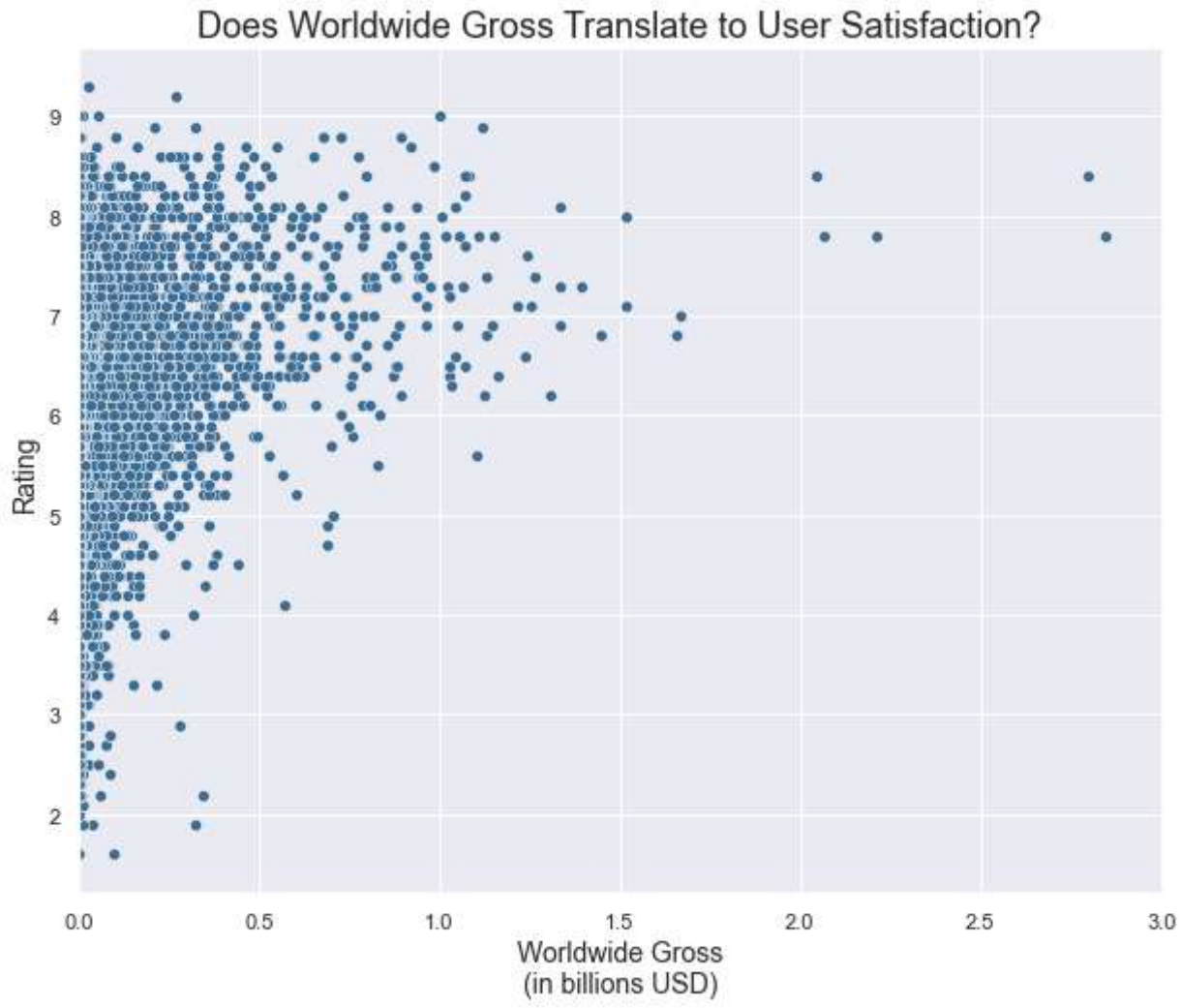
In [97]:
```
#Set image size
sns.set(rc={"figure.figsize":(10, 8)})
df = movies_df.copy()
#Show budget in millions
df['production_budget'] = df['production_budget']/1e6
ax_2 = sns.scatterplot(data=df, x='production_budget',y='averageRating',colo
ax_2.set_ylabel('Rating',fontsize=14)
ax_2.set_xlabel('Production Budget\n(in millions USD)',fontsize=14)
ax_2.set_title('Does Production Budget Translate to User Satisfaction?',font
ax_2.set(xlim=(0,450))
plt.savefig('images/budge_vs_rating.png')
```



1.5.6 Viewer Satisfaction versus Gross

We also need to verify that if high sales correspond to higher rating. It might sound intuitive but showing the plot can help us understand the problem better.

In [98]:
```
#Set image size
sns.set(rc={"figure.figsize":(10, 8)})
df = movies_df.copy()
#Show values in billions
df['worldwide_gross'] = df['worldwide_gross']/1e9
ax = sns.scatterplot(data=df, y='averageRating',x='worldwide_gross',color="#
ax.set_ylabel('Rating',fontsize=14)
ax.set_xlabel('Worldwide Gross\n(in billions USD)',fontsize=14)
ax.set_title('Does Worldwide Gross Translate to User Satisfaction?',fontsize
ax.set(xlim=(0,3))
plt.savefig('images/worldwide_gross_vs_rating.png')
```



Rating is controversial. Small amount of good ratings can have skewed effect. Therefore, we need to show if we can show the overall customer satisfaction. This is done through weighted average between rates and number of votes.

In [99]:
```
#Set image size
sns.set(rc={"figure.figsize":(10, 8)})
df = movies_df.copy()
#Show values in millions
df['viewer_satisfaction'] = df['averageRating']*df['numVotes']/1e6
df['production_budget'] = df['production_budget']/1e6
ax = sns.scatterplot(data=df, x='viewer_satisfaction',y='production_budget',
ax.set_xlabel('Rating x Number of Votes\n(in millions)',fontsize=14)
ax.set_ylabel('Production Budget\n(in millions USD)',fontsize=14)
ax.set_title('Does Production Budget Translate to User Satisfaction?',fontsi
ax.set(xlim=(0,25),ylim=(0,450))
plt.savefig('images/satify_vs_budget.png')
```

1.5.7 Genres with Domestic and Worldwide Gross

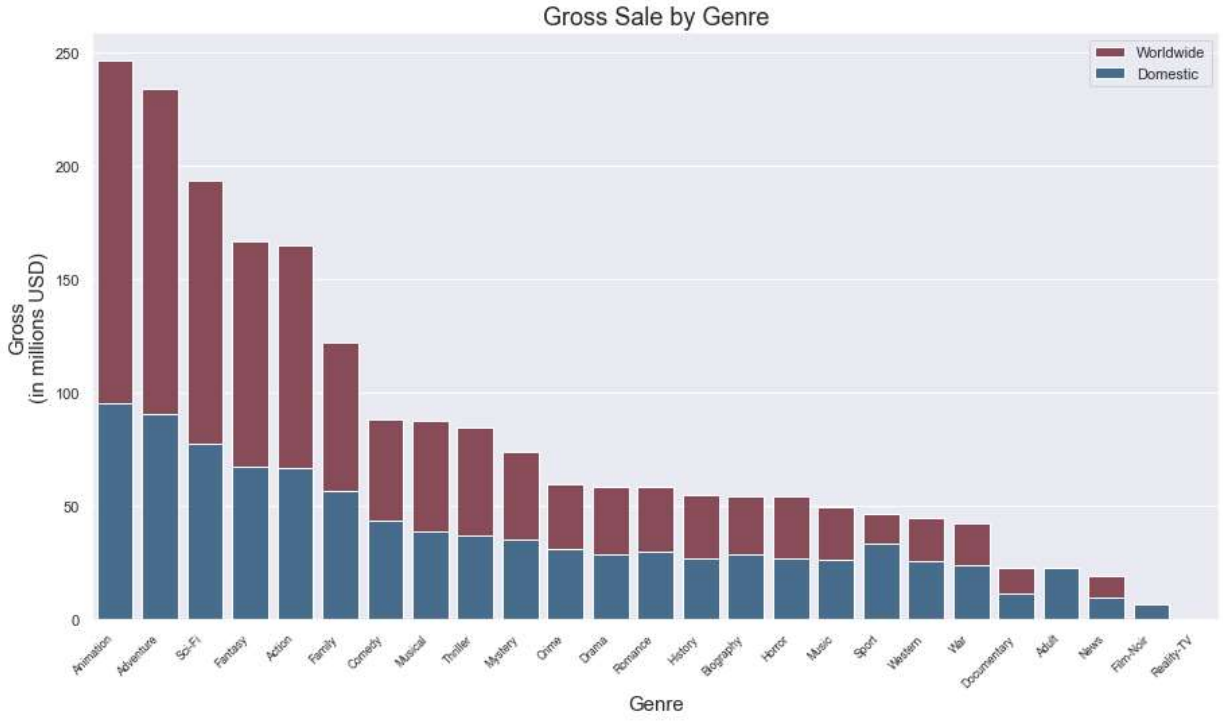
In order to understand which genre has the highest gross, we first need to create a genre boolean dataframe.

In [100]:

```
#Function that returns booleans for a Series of lists
#(Source: https://towardsdatascience.com/dealing-with-list-values-in-pandas-
def boolean_df(item_lists, unique_items):
    # Create empty dict
    bool_dict = {}
    # Loop through all the tags
    for i, item in enumerate(unique_items):
        # Apply boolean mask
        bool_dict[item] = item_lists.apply(lambda x: item in x)
    # Return the results as a dataframe
    return pd.DataFrame(bool_dict)

#Split genres to list
genres = movies_df['genres'].map(lambda x: x.split(','))
#Convert genres to a dataframe
genres_split = genres.apply(pd.Series)
#Get unique items from each column
unique_items = [list(genres_split[c].unique()) for c in genres_split]

#Collect all items and find the unique items
unique_items = Counter([v for s in unique_items for v in s])
#Remove nan column
unique_items = list(unique_items.keys())
unique_items = [x for x in unique_items if str(x) != 'nan']
#Generate a boolean dataframe that contains all genres
genres_bool = boolean_df(genres,unique_items)
genres_bool.drop('\\N',axis=1,inplace=True)
```

Now we can plot the domestic and worldwide gross based on genres.

In [115]:
```
#Set image size
sns.set(rc={"figure.figsize":(15, 8)})
#Get the columns in the boolean genres that we created
genre_col = genres_bool.columns
#Set a dictionary to record the gross per genre
gross_by_genre = {x:[] for x in ['domestic','worldwide','genre']}
for i,col in enumerate(genre_col):
    domestic = movies_df['domestic_gross']
    domestic = domestic[genres_bool[col]].mean()
    worldwide = movies_df['worldwide_gross']
    worldwide = worldwide[genres_bool[col]].mean()
    gross_by_genre['domestic'].append(round(domestic)/1e6)
    gross_by_genre['worldwide'].append(round(worldwide)/1e6)
    gross_by_genre['genre'].append(col)

#Cast the dictionary to a dataframe
df = pd.DataFrame(gross_by_genre)
df = df.sort_values('worldwide',ascending=False)
#Overlay the two plots
s2 = sns.barplot(x='genre',y='worldwide',data=df,color='#914252',label='Worl
s1 = sns.barplot(x='genre',y = 'domestic',data=df,color='#3C6D97',label='Dom
s1.set_xticklabels(s1.get_xmajorticklabels(),fontsize =8.5,rotation=45,ha='r
s1.set_xlabel('Genre',fontsize=15)
s1.set_ylabel('Gross\n(in millions USD)',fontsize=15)
s1.set_title('Gross Sale by Genre',fontsize=18);
s1.legend();
plt.savefig('images/gross_by_genre.png')
```



1.5.8 Ranking Crew by Highest Gross

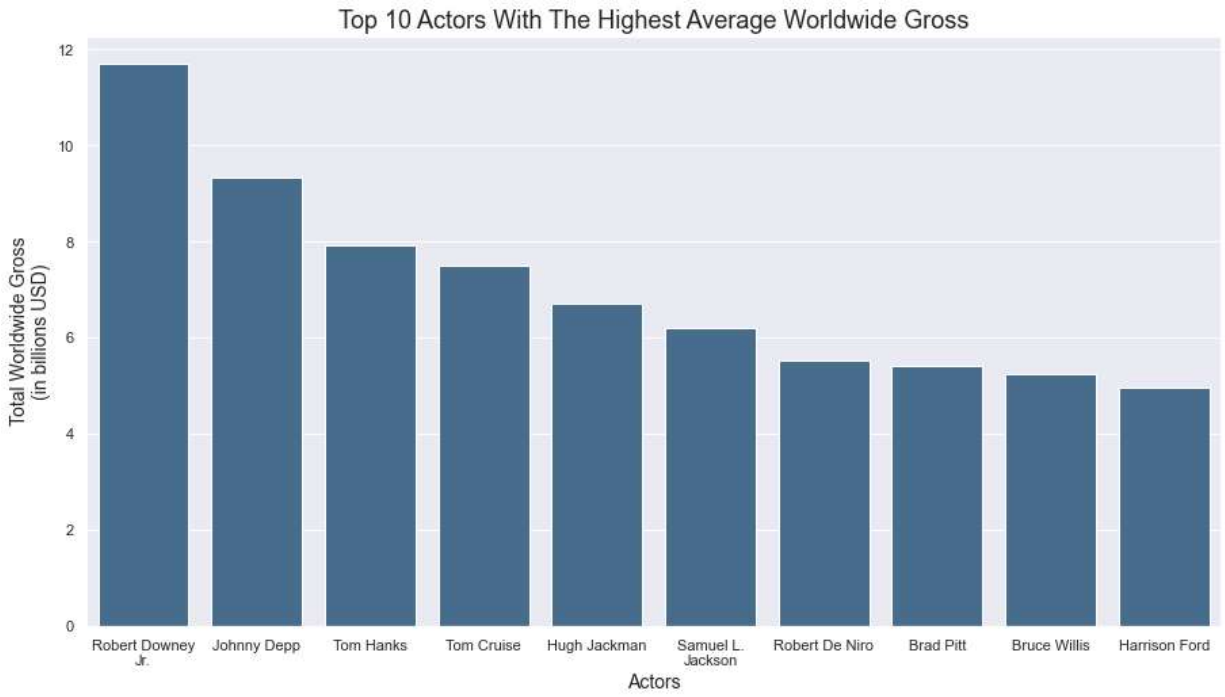
In this last visualization, we are trying to determine which actors and actresses and directors are involved in cumulative and average grossing. This could help in casting and directing choices.

In [116]:
```
#Load only actors, actresses, directors and writers in leading roles
top_roles = ['actor_1','actress_1','director_1','writer_1']

for role in top_roles:
    #Get a list of leading roles
    ppl = list(movies_df[role].unique())
    #Create dictionaries with name as roles
    exec(role+" = {}")
    for person in ppl:
        gross = movies_df[movies_df[role]==person]['worldwide_gross'].sum()
        exec(role+"[person] = gross")
    #Sort the roles by the amount of worldwide gross of movies they particip
    exec(role+"=dict(sorted("+role+".items(), key=lambda item: item[1], reve
    #Take the first 10 of the sorted roles
    exec(role+"={k: "+role+"[k] for k in list("+role+")[:10]}")
```

We now have the first 10 roles and the total amount of gross. Let's create plots and start with actors.

In [117]:

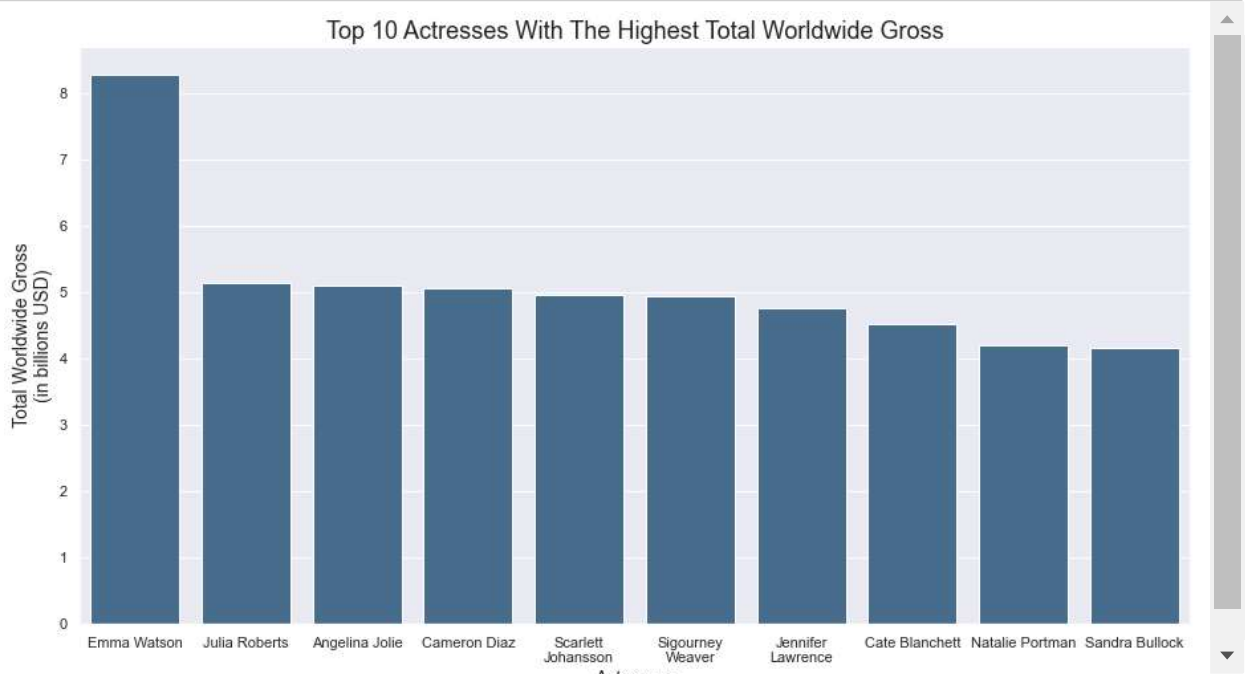
```
#Set image size
sns.set(rc={"figure.figsize":(15, 8)})
#Get keys and values from the actor_1 dictionary
keys = list(actor_1.keys())
vals = list(actor_1.values())
vals = [i/1e9 for i in vals]
ax = sns.barplot(x=keys,
            y=vals,
            color="#3C6D97")
ax.set_xlabel('Actors',fontsize=14)
ax.set_ylabel('Total Worldwide Gross\n(in billions USD)',fontsize=14)
ax.set_title('Top 10 Actors With The Highest Average Worldwide Gross',fontsi
labels = ax.axes.get_xticklabels()
# fix the labels
for v in labels:
    text = v.get_text()
    text = '\n'.join(wrap(text,15))
    v.set_text(text)
# set the new labels
ax.axes.set_xticklabels(labels);
plt.savefig('images/top_grossing_actors_total.png')
```



The female actors who starred roles in top grossing movies are generated as follows.

In [118]:

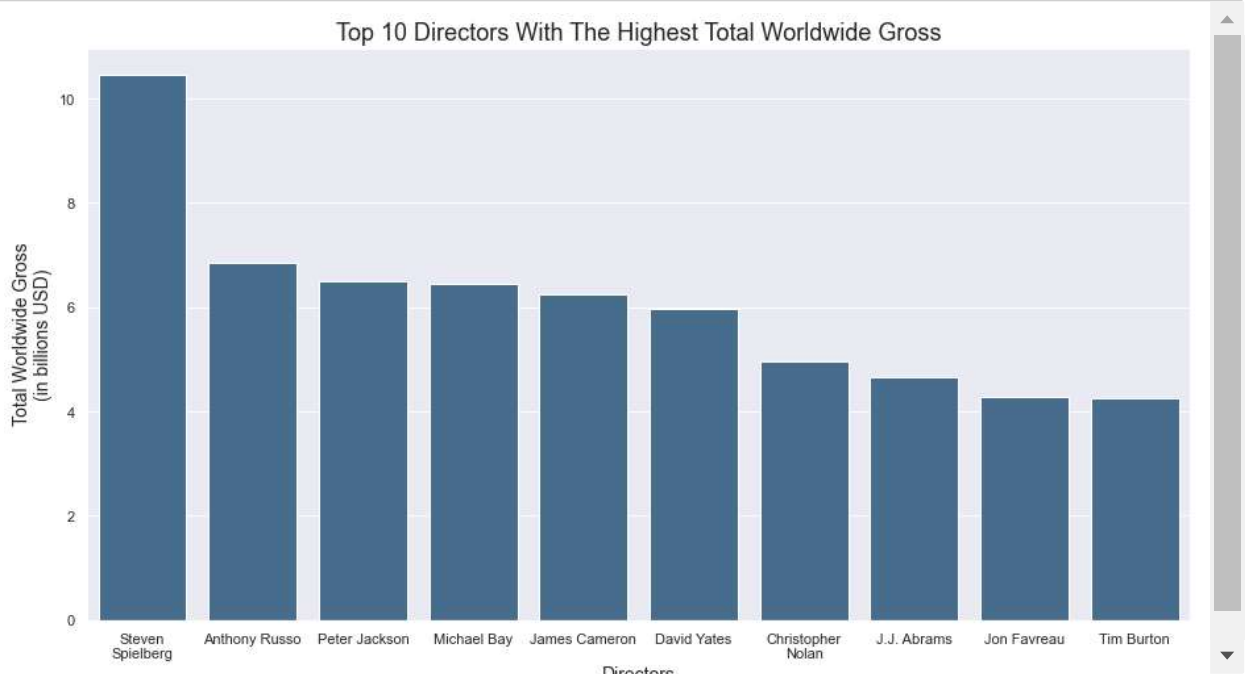
```
#Set image size
sns.set(rc={"figure.figsize":(15, 8)})
#Get keys and values from the actress_1 dictionary
keys = list(actress_1.keys())
vals = list(actress_1.values())
vals = [i/1e9 for i in vals]
ax = sns.barplot(x=keys,
                 y=vals,
                 color="#3C6D97")
ax.set_xlabel('Actresses',fontsize=14)
ax.set_ylabel('Total Worldwide Gross\n(in billions USD)',fontsize=14)
ax.set_title('Top 10 Actresses With The Highest Total Worldwide Gross',fonts
labels = ax.axes.get_xticklabels()
# fix the labels
for v in labels:
    text = v.get_text()
    text = '\n'.join(wrap(text,15))
    v.set_text(text)
# set the new labels
ax.axes.set_xticklabels(labels);
plt.savefig('images/top_grossing_actresses_total.png')
```



Lastly, the directors with the highest collective worldwide gross.

In [119]:

```
#Set Image size
sns.set(rc={"figure.figsize":(15, 8)})
#Get keys and values from the director_1 dictionary
keys = list(director_1.keys())
vals = list(director_1.values())
vals = [i/1e9 for i in vals]
ax = sns.barplot(x=keys,
                 y=vals,
                 color="#3C6D97")
ax.set_xlabel('Directors',fontsize=14)
ax.set_ylabel('Total Worldwide Gross\n(in billions USD)',fontsize=14)
ax.set_title('Top 10 Directors With The Highest Total Worldwide Gross',fonts
labels = ax.axes.get_xticklabels()
# fix the labels
for v in labels:
    text = v.get_text()
    text = '\n'.join(wrap(text,15))
    v.set_text(text)
# set the new labels
ax.axes.set_xticklabels(labels);
plt.savefig('images/top_grossing_directors_total.png')
```



Similarly, we now are going to do the same but this time, we are going to take the average cast and director roles.

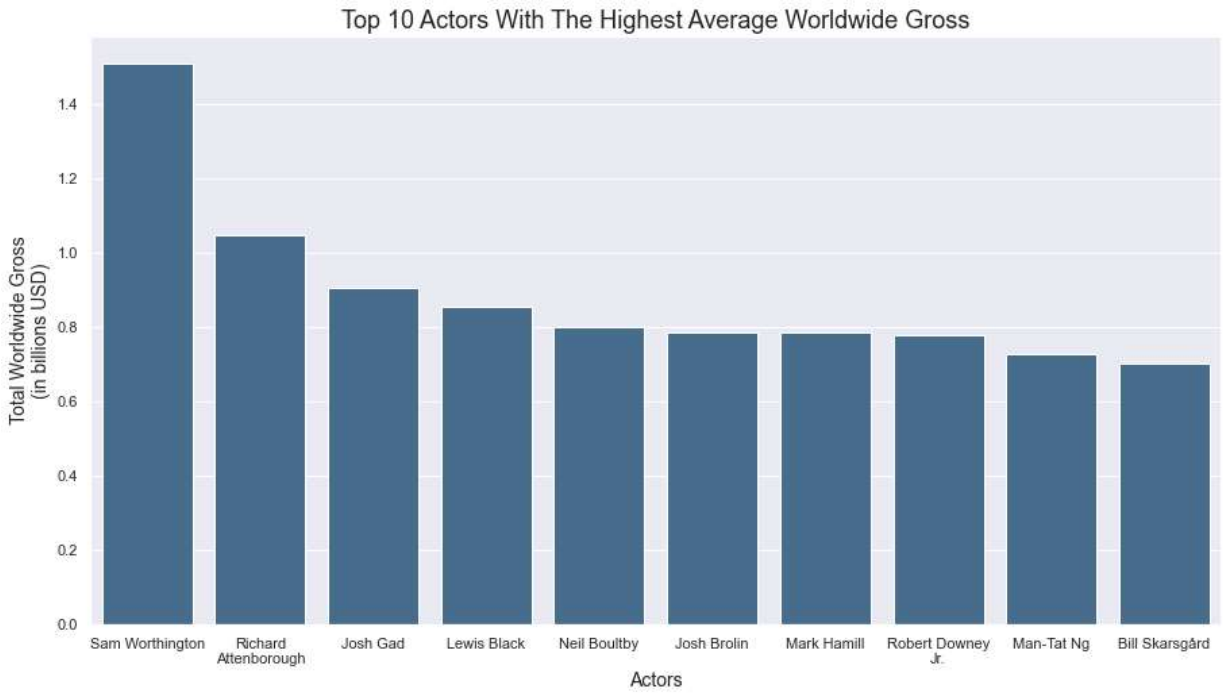
In [120]:

```
#Load only actors, actresses, directors and writers in leading roles
top_roles = ['actor_1','actress_1','director_1','writer_1']

for role in top_roles:
    #Get a list of leading roles
    ppl = list(movies_df[role].unique())
    #Create dictionaries with name as roles
    exec(role+" = {}")
    for person in ppl:
        gross = movies_df[movies_df[role]==person]['worldwide_gross'].mean()
        exec(role+"[person] = gross")
    #Sort the roles by the amount of worldwide gross of movies they particip
    exec(role+"=dict(sorted("+role+".items(), key=lambda item: item[1], reve
    #Take the first 10 of the sorted roles
    exec(role+"={k: "+role+"[k] for k in list("+role+")[:10]}")
```

Let's create plots and start with actors.

In [121]:

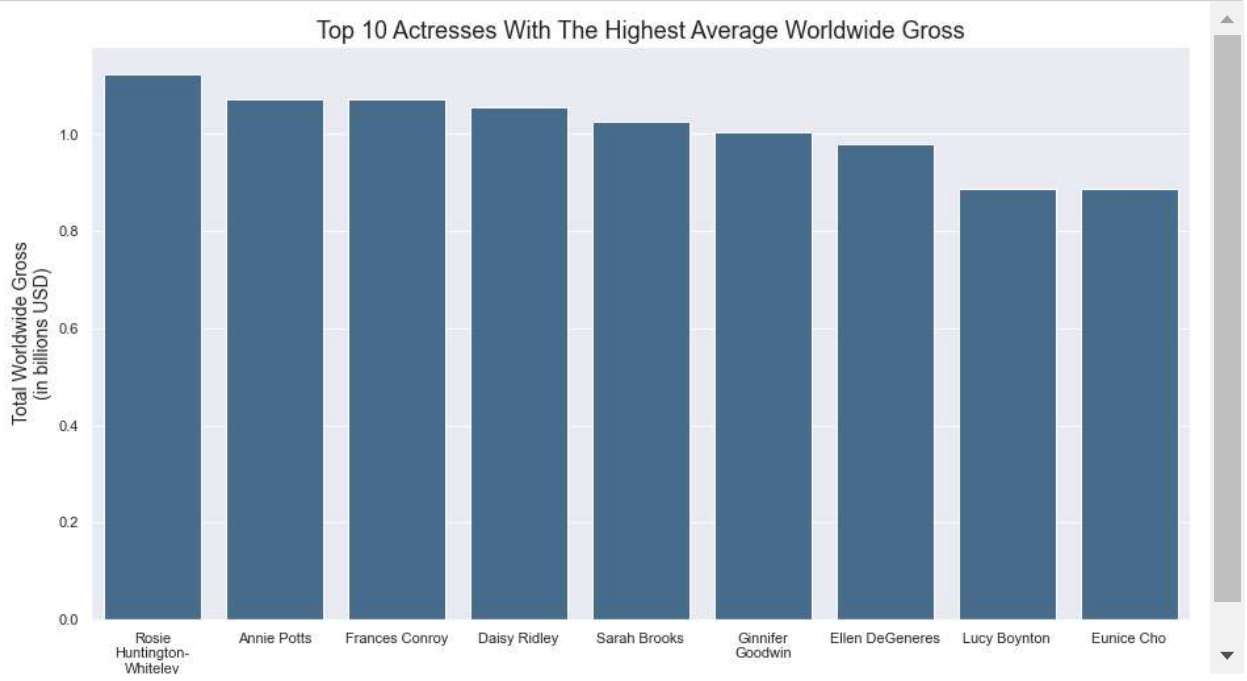
```
#Set image size
sns.set(rc={"figure.figsize":(15, 8)})
#Get keys and values from the actor_1 dictionary
keys = list(actor_1.keys())
vals = list(actor_1.values())
vals = [i/1e9 for i in vals]
ax = sns.barplot(x=keys,
            y=vals,
            color="#3C6D97")
ax.set_xlabel('Actors',fontsize=14)
ax.set_ylabel('Total Worldwide Gross\n(in billions USD)',fontsize=14)
ax.set_title('Top 10 Actors With The Highest Average Worldwide Gross',fontsi
labels = ax.axes.get_xticklabels()
# fix the labels
for v in labels:
    text = v.get_text()
    text = '\n'.join(wrap(text,15))
    v.set_text(text)
# set the new labels
ax.axes.set_xticklabels(labels);
plt.savefig('images/top_grossing_actors_mean.png')
```



The female actors who starred roles in top averaged grossing movies are generated as follows.

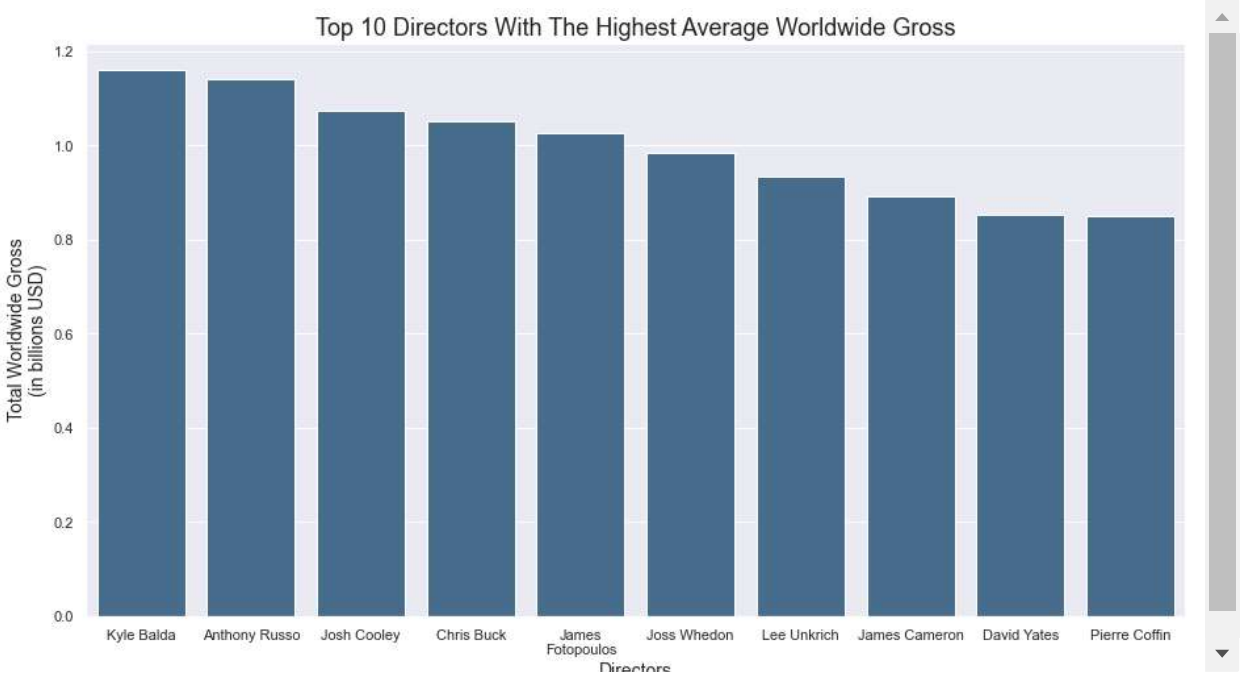
In [122]:

```
#Set image size
sns.set(rc={"figure.figsize":(15, 8)})
#Get keys and values from the actress_1 dictionary
keys = list(actress_1.keys())
vals = list(actress_1.values())
vals = [i/1e9 for i in vals]
ax = sns.barplot(x=keys,
            y=vals,
            color="#3C6D97")
ax.set_xlabel('Actresses',fontsize=14)
ax.set_ylabel('Total Worldwide Gross\n(in billions USD)',fontsize=14)
ax.set_title('Top 10 Actresses With The Highest Average Worldwide Gross',fon
labels = ax.axes.get_xticklabels()
# fix the labels
for v in labels:
    text = v.get_text()
    text = '\n'.join(wrap(text,15))
    v.set_text(text)
# set the new labels
ax.axes.set_xticklabels(labels);
plt.savefig('images/top_grossing_actresses_mean.png')
```



Lastly, the directors with the highest average worldwide gross.

In [123]:
```
#Set Image size
sns.set(rc={"figure.figsize":(15, 8)})
#Get keys and values from the director_1 dictionary
keys = list(director_1.keys())
vals = list(director_1.values())
vals = [i/1e9 for i in vals]
ax = sns.barplot(x=keys,
            y=vals,
            color="#3C6D97")
ax.set_xlabel('Directors',fontsize=14)
ax.set_ylabel('Total Worldwide Gross\n(in billions USD)',fontsize=14)
ax.set_title('Top 10 Directors With The Highest Average Worldwide Gross',fon
labels = ax.axes.get_xticklabels()
# fix the labels
for v in labels:
    text = v.get_text()
    text = '\n'.join(wrap(text,15))
    v.set_text(text)
# set the new labels
ax.axes.set_xticklabels(labels);
plt.savefig('images/top_grossing_directors_mean.png')
```



1.6 Conclusions

This analysis leads to the following conclusions:

- **New movie franchise requires massive budget, specially since the last couple of years.** While movies generally had upward trend production budget with time, there has been a very noticeable uptick over the last two year.
- **Sequels are generally favored by audience.** Audience are captivated by sequels to well-established franchises. This remains to be true despite the depreciated quality. Some, however, have been building up their stories to create mass anticipation.
- **Viewers favorably appreciate the production scale of the project.** Audience and movie goers generally appreciate the effort made to elevate the production budget. This is might

seem unsurprising but there are some caveats that could reverse this trend, such as sequels not meeting viewers' expectations.

- **Some genres (animation, adventure and sci-fi) earn more profit above all others.** Animations have some of the highest grossing. This can reflect two things. First, animations usually have audiences of all ages. Second, animations usually have high overhead cost with rendering farms and with increased technology, attention to details has become increasingly important.
- **Movies tend to be successful based on the choice of director and cast.** As mentioned, recognizable characters garner more audience than those in a stand-alone movie. Although there were more seasoned actors in the list, those involved in high grossing movies maintained the character throughout the franchise. That could have an influence on how well the movies are received.

1.7 Next Steps

Further analyses could yield additional insights to make better recommendation for Microsoft:

- **Identify metrics to predict the success rate of movies.** Besides the data that has been presented, there is still more information that could be gathered from the data collected, such as duration of movies to reviews and so on. If more information is needed to predict the success rate of movies, additional data must be collected.
- **Gather more data on the companies that distribute the movies.** The current dataset suffers from one important metric: which is the name of the distributer. This would further enhance the analysis if streaming services have better success than well-established companies.
- **Include critics' reviews.** Up until this point, only onine reviews were gathered. But there are other sources for more curated review such as Rotten Tomatoes (https://www.rottentomatoes.com/) and other souces that could further enrich the data.
- **Implement sentiment analysis techniques to gather feedback from reviews.** The next step would be to implement a form of sentiment analysis to automate if review are generally positive, negative or neutral.