Movie Analysis

Author: Dan Lee

Overview

This presentation contains data-driven results and recommendations for Microsoft as they consider their entry into the film industry. Given this company's extensive resources, Microsoft has the ability to make a big splash and enter the field at the top. They can make high caliber movies that yield revenue and ratings competitive with the upper-echelon in the industry standard.

As a start, this presentation will recommend that executives aim to make movies of a certain genre, time length, and with specific personnel.

Business Problem

The past 20 years has seen a transition in the film making and production business. With the rise of internet streaming delivery to consumers, tech companies have benefitted from producing their own movies in house. They have cut out the middle man by avoiding outsourcing them to the production companies that have dominated Hollywood for nearly a century.

Microsoft is one of the big players in personal computers, software/operating systems, video games, and accessories. Now they want in on the $100+ billion film industry. In this presentation we will give Microsoft executives some vision and direction in how to approach movie making.

We will focus on three questions:

1. What movie genres earn the most worldwide revenue?
2. How long should a movie be to maximize revenue?
3. What movie personnel will maximize IMDB ratings?

I believe the answers to these high-level questions will give executives an understanding of the current trends in the film industry.

Question 1: What movie genres earn the most worldwide revenue?

We will explore the data to find the top five highest grossing movie genres from 2010-2018.

Q1 A. Data Understanding

We will be using two datasets:

1. imdb.title.basics - This dataset comes from IMDb (Internet Movie Database). The target variable we will use is the genre types of each movie.
2. bom.movie.gross - This dataset comes from BOM (Box Office Mojo). The target variables are domestic (USA) gross and foreign gross.

All of the datasets in this analysis contain comprhensive movie data from the years 2010-2018.

In [1]:
```
# Import standard packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [2]:
```python
imdb_title_basics = pd.read_csv('zippedData/imdb.title.basics.csv.gz')
imdb_title_basics
```

Out[2]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy
...	...	...	...	...	...	...
146139	tt9916538	Kuambil Lagi Hatiku	Kuambil Lagi Hatiku	2019	123.0	Drama
146140	tt9916622	Rodolpho Teóphilo - O Legado de um Pioneiro	Rodolpho Teóphilo - O Legado de um Pioneiro	2015	NaN	Documentary
146141	tt9916706	Dankyavar Danka	Dankyavar Danka	2013	NaN	Comedy
146142	tt9916730	6 Gunn	6 Gunn	2017	116.0	NaN
146143	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	NaN	Documentary

146144 rows × 6 columns

In [3]:
```python
bom_movie_gross = pd.read_csv('zippedData/bom.movie_gross.csv.gz')
bom_movie_gross
```

Out[3]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
3	Inception	WB	292600000.0	535700000	2010
4	Shrek Forever After	P/DW	238700000.0	513900000	2010
...	...	...	...	...	...
3382	The Quake	Magn.	6200.0	NaN	2018
3383	Edward II (2018 re-release)	FM	4800.0	NaN	2018
3384	El Pacto	Sony	2500.0	NaN	2018
3385	The Swan	Synergetic	2400.0	NaN	2018
3386	An Actor Prepares	Grav.	1700.0	NaN	2018

3387 rows × 5 columns

Q1 B. Data Preparation

In this section we will prepare to join these two datasets by:

- Creating a title + year column to merge on
- Cleaning the data from duplicates and null values

In [4]:
```python
# Normalizing title names in preparation for join
imdb_adjusted_titles = []
for title in imdb_title_basics['primary_title']:
    imdb_adjusted_titles.append(
        title.lower().replace(":", "").replace("-", ""))
imdb_title_basics['primary_title'] = imdb_adjusted_titles
imdb_title_basics['primary_title'].head()
```

Out[4]:
```
0                         sunghursh
1     one day before the rainy season
2           the other side of the wind
3                      sabse bada sukh
4            the wandering soap opera
Name: primary_title, dtype: object
```

In [5]:
```python
# Create new column with title + year to minimize duplicates when joining
imdb_title_basics['title_year'] = imdb_title_basics[
    'primary_title'] + " " + imdb_title_basics['start_year'].astype('str')
imdb_title_basics.head()
```

Out[5]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres	title_ye
0	tt0063540	sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama	sunghur 20
1	tt0066787	one day before the rainy season	Ashad Ka Ek Din	2019	114.0	Biography,Drama	one d before t rai seas 20
2	tt0069049	the other side of the wind	The Other Side of the Wind	2018	122.0	Drama	the oth side of t wind 20
3	tt0069204	sabse bada sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama	sab bada su 20
4	tt0100275	the wandering soap opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy	t wanderi so ope 20

In [6]:
```python
imdb_title_basics[imdb_title_basics['start_year'] == 2019].head()
```

Out[6]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres	title_y
1	tt0066787	one day before the rainy season	Ashad Ka Ek Din	2019	114.0	Biography,Drama	one before ra sea 2
26	tt0263814	on kadin	On kadin	2019	NaN	Drama	on ka 2
31	tt0285423	abolição	Abolição	2019	NaN	Documentary	aboli 2
68	tt0385887	motherless brooklyn	Motherless Brooklyn	2019	NaN	Crime,Drama	motherl brook 2
107	tt0437086	alita battle angel	Alita: Battle Angel	2019	122.0	Action,Adventure,Sci-Fi	alita ba angel 2

This data was likely gathered in 2019 as there are some movies from 2019 with null values for runtime minutes. Thus, we will remove these unreleased movies from the dataset:

In [7]:
```python
# Purging unreleased movies based on year and null runtime
imdb_drop_rows = []
for row in imdb_title_basics.index:
    if (np.isnan(imdb_title_basics['runtime_minutes'][row])
            == True) and (imdb_title_basics['start_year'][row] >= 2020):
        imdb_drop_rows.append(row)
imdb_drop_rows[:5]
```

Out[7]:
```
[33, 93, 229, 289, 386]
```

In [8]:
```python
imdb_title_basics.drop(imdb_drop_rows, inplace = True)
```

In [9]:
```python
len(imdb_title_basics)
```

Out[9]:
```
145170
```

In [10]:
```python
# Checking title_year column for duplicates
imdb_title_basics.duplicated(subset=['title_year']).sum()
```

Out[10]:
```
2160
```

In [11]:
```python
# Checking original title column for duplicates
imdb_title_basics.duplicated(subset=['primary_title']).sum()
```

Out[11]:
```
10251
```

In [12]:
```python
# Dropping duplicate titles in title_year column
clean_imdb = imdb_title_basics.drop_duplicates(subset=['title_year'],
                                               keep="first")
clean_imdb.duplicated(subset=['title_year']).sum()
```

Out[12]:
```
0
```

In [13]:
```python
len(clean_imdb)
```

Out[13]:
```
143010
```

Let's apply the same process to the BOM dataset:

In [14]:
```
bom_movie_gross
```

0	Toy Story 3	BV	415000000.0	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
3	Inception	WB	292600000.0	535700000	2010
4	Shrek Forever After	P/DW	238700000.0	513900000	2010
...	...	...	...	...	...
3382	The Quake	Magn.	6200.0	NaN	2018
3383	Edward II (2018 re-release)	FM	4800.0	NaN	2018
3384	El Pacto	Sony	2500.0	NaN	2018
3385	The Swan	Synergetic	2400.0	NaN	2018
3386	An Actor Prepares	Grav.	1700.0	NaN	2018

3387 rows × 5 columns

In [15]:
```
# Normalizing title names in preparation for join
bom_adjusted_titles = []
for title in bom_movie_gross['title']:
    bom_adjusted_titles.append(title.lower().replace(":", "").replace("-",
bom_movie_gross['title'] = bom_adjusted_titles
bom_movie_gross['title'].head()
```

Out[15]:
```
0                                    toy story 3
1                     alice in wonderland (2010)
2    harry potter and the deathly hallows part 1
3                                      inception
4                            shrek forever after
Name: title, dtype: object
```

In [16]:
```python
# Create new column with title + year to minimize duplicates when joining
bom_movie_gross['title_year'] = bom_movie_gross['title'] + \
    " " + bom_movie_gross['year'].astype('str')
bom_movie_gross = bom_movie_gross.reset_index()
bom_movie_gross.head()
```

Out[16]:

	index	title	studio	domestic_gross	foreign_gross	year	title_year
0	0	toy story 3	BV	415000000.0	652000000	2010	toy story 3 2010
1	1	alice in wonderland (2010)	BV	334200000.0	691300000	2010	alice in wonderland (2010) 2010
2	2	harry potter and the deathly hallows part 1	WB	296000000.0	664300000	2010	harry potter and the deathly hallows part 1 2010
3	3	inception	WB	292600000.0	535700000	2010	inception 2010
4	4	shrek forever after	P/DW	238700000.0	513900000	2010	shrek forever after 2010

In [17]:
```python
# Checking original title column for duplicates
bom_movie_gross.duplicated(subset=['title']).sum()
```

Out[17]: 1

In [18]:
```python
# Checking new title_year column for duplicates
bom_movie_gross.duplicated(subset=['title_year']).sum()
```

Out[18]: 0

In [19]:
```python
# Joining imdb with bom
len(bom_movie_gross)
```

Out[19]: 3387

In [20]:
```python
len(clean_imdb)
```

Out[20]: 143010

In [21]:
```python
bom_imdb = pd.merge(
    clean_imdb,
    bom_movie_gross,
    how='inner',
    on ='title_year')
```

In [22]:
```
bom_imdb
```

Out[22]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0315642	wazir	Wazir	2016	103.0	Action,Crime,Drama
1	tt0337692	on the road	On the Road	2012	124.0	Adventure,Drama,Romance
2	tt0359950	the secret life of walter mitty	The Secret Life of Walter Mitty	2013	114.0	Adventure,Comedy,Drama
3	tt0365907	a walk among the tombstones	A Walk Among the Tombstones	2014	114.0	Action,Crime,Drama
4	tt0369610	jurassic world	Jurassic World	2015	124.0	Action,Adventure,Sci-Fi
...	...	...	...	...	...	...
1918	tt8362680	mountain	Mountain	2018	15.0	Documentary
1919	tt8404272	how long will i love u	Chao shi kong tong ju	2018	101.0	Romance
1920	tt8427036	helicopter eela	Helicopter Eela	2018	135.0	Drama
1921	tt9078374	last letter	Ni hao, Zhihua	2018	114.0	Drama,Romance
1922	tt9151704	burn the stage the movie	Burn the Stage: The Movie	2018	84.0	Documentary,Music

1923 rows × 13 columns

In [23]:
```
# Checking merged dataset for duplicates
bom_imdb.duplicated(subset=['index']).sum()
```

Out[23]: 0

In [24]:
```
bom_imdb.duplicated(subset=['tconst']).sum()
```

Out[24]: 0

In [25]:
```python
# Checking for null values
bom_imdb.isna().sum()
```

Out[25]:
```
tconst              0
primary_title       0
original_title      0
start_year          0
runtime_minutes     3
genres              0
title_year          0
index               0
title               0
studio              2
domestic_gross     11
foreign_gross     609
year                0
dtype: int64
```

The null values in the domestic_gross and foreign_gross columns must be dealt with if we are going to find total revenue for currently released movies. We will take a closer look at runtime_minutes in Q2.

In [26]:
```
bom_imdb[bom_imdb['domestic_gross'].isna()]
```

Out[26]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres	title
314	tt1319716	it's a wonderful afterlife	It's a Wonderful Afterlife	2010	100.0	Comedy,Drama,Fantasy	wor a
475	tt1507563	dark tide	Dark Tide	2012	94.0	Action,Adventure,Drama	dai
524	tt1570982	celine through the eyes of the world	Celine: Through the Eyes of the World	2010	120.0	Documentary,Music	th the
584	tt1618421	white lion	White Lion	2010	88.0	Drama,Family	whi
638	tt1658837	the tall man	The Tall Man	2012	106.0	Crime,Drama,Horror	t mar
867	tt1992138	force	Force	2011	137.0	Action,Thriller	
931	tt2106537	matru ki bijlee ka mandola	Matru ki Bijlee ka Mandola	2013	151.0	Comedy,Drama	m bij ma
960	tt2147365	keith lemon the film	Keith Lemon: The Film	2012	85.0	Comedy	lem film
1039	tt2300975	jessabelle	Jessabelle	2014	90.0	Horror,Thriller	jess
1179	tt2597892	viral	Viral	2016	85.0	Drama,Horror,Sci-Fi	vira
1793	tt6108090	secret superstar	Secret Superstar	2017	150.0	Drama,Music	sup

In [27]:
```
# Dropping rows with no domestic gross data because there are only 11, and
# gross impact is relatively tiny
bom_imdb = bom_imdb.dropna(subset = ['domestic_gross'])
```

In [28]:
```
bom_imdb['foreign_gross'][0] == True # Should return false it is a null val
```

Out[28]:
```
False
```

Cleaning runtime data ahead of question 2

This section includes cleaning null values for runtime data, and must be adjusted now before we proceed.

In [29]:
```
# Checking for runtime null values
bom_imdb.isna().sum()
```

Out[29]:
```
tconst              0
primary_title       0
original_title      0
start_year          0
runtime_minutes     3
genres              0
title_year          0
index               0
title               0
studio              0
domestic_gross      0
foreign_gross     609
year                0
dtype: int64
```

In [30]:
```
# Checking domestic gross for movies with no runtime data
titles_null_runtime = {}
for row in bom_imdb.index:
    if pd.isna(bom_imdb['runtime_minutes'][row]) == True:
        titles_null_runtime[bom_imdb['title_year']
                            [row]] = bom_imdb['domestic_gross'][row]
titles_null_runtime
```

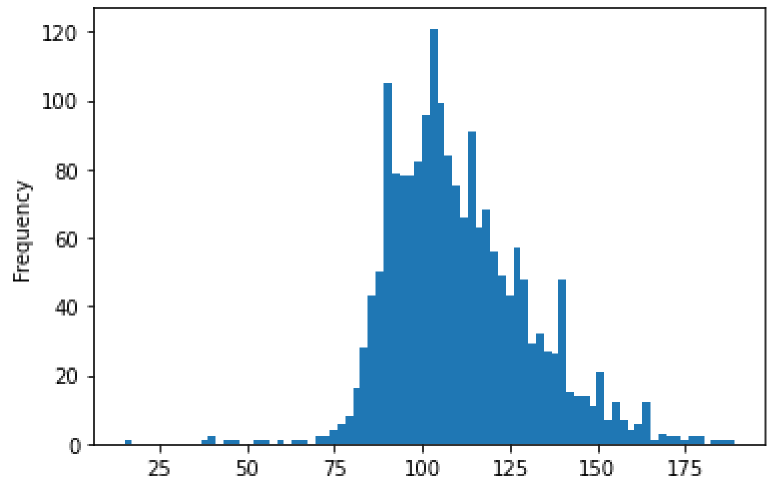
Out[30]:
```
{'upside down 2013': 105000.0,
 'extraction 2015': 16800.0,
 'the other side 2016': 8100.0}
```

We will replace the null runtime for the movies with the median runtime for all movies:

In [31]:
```
# Replacing null runtime values with median
runtime_mean = bom_imdb['runtime_minutes'].mean()
runtime_median = bom_imdb['runtime_minutes'].median()
```

In [32]:
```python
# Plotting distribution of runtime
bom_imdb['runtime_minutes'].plot(kind='hist', bins=80)
print(f"Mean: {bom_imdb['runtime_minutes'].mean()}")
print(f"Median: {bom_imdb['runtime_minutes'].median()}")
```

```
Mean: 110.82818229439498
Median: 107.0
```



In [33]:
```python
bom_imdb['runtime_minutes'] = bom_imdb['runtime_minutes'].fillna(runtime_me
```

```
<ipython-input-33-4655749cc4d4>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  bom_imdb['runtime_minutes'] = bom_imdb['runtime_minutes'].fillna(runtim
e_median)
```

In [34]:
```python
bom_imdb.isna().sum()
```

Out[34]:
```
tconst               0
primary_title        0
original_title       0
start_year           0
runtime_minutes      0
genres               0
title_year           0
index                0
title                0
studio               0
domestic_gross       0
foreign_gross      609
year                 0
dtype: int64
```

Q1 Data Preparation (cont.) - Foreign gross missing data

In order to decide what to do with the foreign_gross null data, we will analyze what movies are missing foreign gross revenue by looking at how much they earned domestically:

In [35]:
```
# Preparing a dictionary to view domestic gross of movies missing foreign g
titles_null_foreign = {}
for row in bom_imdb.index:
    if pd.isna(bom_imdb['foreign_gross'][row]) == True:
        titles_null_foreign[bom_imdb['title_year']
                                    [row]] = bom_imdb['domestic_gross'][row]
```

In [36]:
```
# Finding domestic gross mean as a benchmark
bom_imdb['domestic_gross'].mean()
```

Out[36]:
```
42926502.922594145
```

In [37]:
```python
# Viewing starting from the highest earnings
null_foreign_earnings = {k: v for k, v in sorted(
    titles_null_foreign.items(), key=lambda item: item[1], reverse=True)}
null_foreign_earnings
```

Out[37]:
```
{'book club 2018': 68600000.0,
 'war room 2015': 67800000.0,
 'all eyez on me 2017': 44900000.0,
 '47 meters down 2017': 44300000.0,
 'snitch 2013': 42900000.0,
 'courageous 2011': 34500000.0,
 'wind river 2017': 33800000.0,
 'when the game stands tall 2014': 30100000.0,
 'hostiles 2017': 29800000.0,
 'home again 2017': 27000000.0,
 'winchester 2018': 25100000.0,
 'our idiot brother 2011': 24800000.0,
 'whiskey tango foxtrot 2016': 23100000.0,
 'keanu 2016': 20600000.0,
 'broken city 2013': 19700000.0,
 'the choice 2016': 18700000.0,
 'admission 2013': 18000000.0,
 'they shall not grow old 2018': 18000000.0,
 'bad santa 2 2016': 17800000.0,
 'free solo 2018': 17500000.0,
 'addicted 2014': 17400000.0,
 'norm of the north 2016': 17100000.0,
 'detroit 2017': 16800000.0,
 'hunter killer 2018': 15800000.0,
 'the infiltrator 2016': 15400000.0,
 'the last exorcism part ii 2013': 15200000.0,
 'woodlawn 2015': 14400000.0,
 'wish upon 2017': 14300000.0,
 'jackie 2016': 14000000.0,
 'love & friendship 2016': 14000000.0,
 'playing for keeps 2012': 13100000.0,
 'battle of the sexes 2017': 12600000.0,
 'people like us 2012': 12400000.0,
 'before i fall 2017': 12200000.0,
 'foxcatcher 2014': 12100000.0,
 'no manches frida 2016': 11500000.0,
 'the old man & the gun 2018': 11300000.0,
 'the darkness 2016': 10800000.0,
 'belle 2014': 10700000.0,
 "moms' night out 2014": 10400000.0,
 'prom 2011': 10100000.0,
 'ratchet & clank 2016': 8800000.0,
 'can you ever forgive me? 2018': 8800000.0,
 'amy 2015': 8400000.0,
 'bajrangi bhaijaan 2015': 8199999.0,
 "i'll see you in my dreams 2015": 7400000.0,
 'take me home tonight 2011': 6900000.0,
 'cedar rapids 2011': 6900000.0,
 'the spectacular now 2013': 6900000.0,
 'the collection 2012': 6800000.0,
 'baahubali the beginning 2015': 6700000.0,
 'bajirao mastani 2015': 6600000.0,
```

'the young messiah 2016': 6500000.0,
'little boy 2015': 6500000.0,
'cantinflas 2014': 6400000.0,
'believe 2013': 6200000.0,
'the hurricane heist 2018': 6100000.0,
'leave no trace 2018': 6000000.0,
'the florida project 2017': 5900000.0,
'suburbicon 2017': 5800000.0,
'closed circuit 2013': 5800000.0,
'my hero academia two heroes 2018': 5800000.0,
'20th century women 2016': 5700000.0,
'the man who invented christmas 2017': 5700000.0,
'the skeleton twins 2014': 5300000.0,
"won't back down 2012": 5300000.0,
'chennai express 2013': 5300000.0,
'hunt for the wilderpeople 2016': 5200000.0,
'bleed for this 2016': 5100000.0,
'colette 2018': 5100000.0,
'20 feet from stardom 2013': 4900000.0,
'dilwale 2015': 4900000.0,
'hands of stone 2016': 4700000.0,
'true story 2015': 4700000.0,
'dear white people 2014': 4400000.0,
'gotti 2018': 4300000.0,
'a hologram for the king 2016': 4200000.0,
'45 years 2015': 4200000.0,
'swiss army man 2016': 4200000.0,
'ya veremos 2018': 4200000.0,
'denial 2016': 4099999.0,
'buck 2011': 4000000.0,
'safety not guaranteed 2012': 4000000.0,
'my old lady 2014': 4000000.0,
'kabali 2016': 3900000.0,
'yeh jawaani hai deewani 2013': 3800000.0,
'batman the killing joke 2016': 3800000.0,
'don 2 2011': 3700000.0,
'searching for sugar man 2012': 3700000.0,
'ben is back 2018': 3700000.0,
'spare parts 2015': 3600000.0,
'mommy 2014': 3500000.0,
'everybody wants some!! 2016': 3400000.0,
'bad samaritan 2018': 3400000.0,
'robot & frank 2012': 3300000.0,
'shoplifters 2018': 3300000.0,
'the leisure seeker 2017': 3200000.0,
'the eagle huntress 2016': 3200000.0,
'zindagi na milegi dobara 2011': 3100000.0,
'rosewater 2014': 3100000.0,
'obvious child 2014': 3100000.0,
'compadres 2016': 3100000.0,
'dil dhadakne do 2015': 3100000.0,
'ladrones 2015': 3100000.0,
'tanu weds manu returns 2015': 3000000.0,
'jab tak hai jaan 2012': 3000000.0,
'rock the kasbah 2015': 3000000.0,
'the end of the tour 2015': 3000000.0,
'anthropoid 2016': 3000000.0,

'talaash 2012': 2900000.0,
'barfi! 2012': 2800000.0,
'la boda de valentina 2018': 2800000.0,
'the good lie 2014': 2700000.0,
'youth 2015': 2700000.0,
'chiraq 2015': 2700000.0,
'ramleela 2013': 2700000.0,
'the raid 2 2014': 2600000.0,
'kill the messenger 2014': 2500000.0,
'dabangg 2 2012': 2500000.0,
'kick 2014': 2500000.0,
'mistress america 2015': 2500000.0,
'truth 2015': 2500000.0,
"i'm in love with a church girl 2013": 2400000.0,
'the queen of versailles 2012': 2400000.0,
'the disappointments room 2016': 2400000.0,
'the homesman 2014': 2400000.0,
'dear zindagi 2016': 2400000.0,
'ek tha tiger 2012': 2300000.0,
'love is strange 2014': 2300000.0,
'the christmas candle 2013': 2300000.0,
'elle 2016': 2300000.0,
'ode to my father 2014': 2300000.0,
'krrish 3 2013': 2200000.0,
'austenland 2013': 2200000.0,
'2 states 2014': 2200000.0,
'paterson 2016': 2200000.0,
"i'm not ashamed 2016": 2100000.0,
'agneepath 2012': 2000000.0,
'the handmaiden 2016': 2000000.0,
'good time 2017': 2000000.0,
'english vinglish 2012': 1900000.0,
'everybody loves somebody 2017': 1900000.0,
'along with the gods the two worlds 2017': 1900000.0,
'flipped 2010': 1800000.0,
'the vatican tapes 2015': 1800000.0,
'a better life 2011': 1800000.0,
'bodyguard 2011': 1800000.0,
'housefull 2 2012': 1800000.0,
'tusk 2014': 1800000.0,
'piku 2015': 1800000.0,
'son of saul 2015': 1800000.0,
'the last word 2017': 1800000.0,
'weiner 2016': 1700000.0,
'busco novio para mi mujer 2016': 1700000.0,
'capernaum 2018': 1700000.0,
'a late quartet 2012': 1600000.0,
'race 2 2013': 1600000.0,
'bhaag milkha bhaag 2013': 1600000.0,
'a ghost story 2017': 1600000.0,
'fed up 2014': 1500000.0,
'the diary of a teenage girl 2015': 1500000.0,
'awake the life of yogananda 2014': 1500000.0,
'toni erdmann 2016': 1500000.0,
'the square 2017': 1500000.0,
'marley 2012': 1400000.0,
"the devil's double 2011": 1400000.0,

'the art of getting by 2011': 1400000.0,
'force majeure 2014': 1400000.0,
'welcome back 2015': 1400000.0,
'jai ho 2014': 1300000.0,
'chasing ice 2012': 1300000.0,
'the neon demon 2016': 1300000.0,
'for a good time, call... 2012': 1300000.0,
'the other son 2012': 1300000.0,
'the kings of summer 2013': 1300000.0,
'the wolfpack 2015': 1300000.0,
'churchill 2017': 1300000.0,
'mohenjo daro 2016': 1300000.0,
'housefull 3 2016': 1300000.0,
'the sense of an ending 2017': 1300000.0,
'never look away 2018': 1300000.0,
'the invisible woman 2013': 1200000.0,
'ek main aur ekk tu 2012': 1200000.0,
'miss you already 2015': 1200000.0,
'singham returns 2014': 1200000.0,
'the remaining 2014': 1200000.0,
'3 idiotas 2017': 1200000.0,
'veteran 2015': 1200000.0,
'mandy 2018': 1200000.0,
'andhadhun 2018': 1200000.0,
'wazir 2016': 1100000.0,
'red state 2011': 1100000.0,
'there be dragons 2011': 1100000.0,
'hector and the search for happiness 2014': 1100000.0,
'desi boyz 2011': 1100000.0,
'laggies 2014': 1100000.0,
'elvis & nixon 2016': 1100000.0,
'the captive 2014': 1100000.0,
'the innocents 2016': 1100000.0,
'certain women 2016': 1100000.0,
'kill your darlings 2013': 1000000.0,
'short term 12 2013': 1000000.0,
'haider 2014': 1000000.0,
"film stars don't die in liverpool 2017": 1000000.0,
'little italy 2018': 990000.0,
'baar baar dekho 2016': 982000.0,
'ready 2011': 955000.0,
'faces places 2017': 954000.0,
'neruda 2016': 939000.0,
'tubelight 2017': 930000.0,
'shaadi ke side effects 2014': 921000.0,
'boruto naruto the movie 2015': 920000.0,
'the miseducation of cameron post 2018': 905000.0,
'kedarnath 2018': 901000.0,
'gentleman 2016': 898000.0,
'song of the sea 2014': 858000.0,
'ki & ka 2016': 858000.0,
'generation iron 2013': 850000.0,
'mustang 2015': 845000.0,
'higher ground 2011': 842000.0,
'bombshell the hedy lamarr story 2017': 820000.0,
'kochadaiiyaan 2014': 817000.0,
'dishoom 2016': 813000.0,

'mirai 2018': 813000.0,
'beast 2018': 800000.0,
'darling companion 2012': 794000.0,
'teri meri kahaani 2012': 781000.0,
'son of sardaar 2012': 772000.0,
'satyagraha 2013': 739000.0,
'ek villain 2014': 731000.0,
'mausam 2011': 728000.0,
'walk with me 2017': 727000.0,
'burning 2018': 719000.0,
'khoobsurat 2014': 711000.0,
'hobo with a shotgun 2011': 703000.0,
'tangerine 2015': 702000.0,
'agent vinod 2012': 698000.0,
'shubh mangal saavdhan 2017': 690000.0,
'besharam 2013': 680000.0,
'student of the year 2012': 670000.0,
'the stanford prison experiment 2015': 661000.0,
'frank 2014': 645000.0,
'holy motors 2012': 641000.0,
'the assassin 2015': 633000.0,
'mary kom 2014': 621000.0,
'tickled 2016': 614000.0,
'walking the camino six ways to santiago 2013': 613000.0,
'the midwife 2017': 604000.0,
'lootera 2013': 582000.0,
'freeheld 2015': 573000.0,
'the future 2011': 569000.0,
'tunnel 2016': 569000.0,
'manmarziyaan 2018': 567000.0,
'dum maaro dum 2011': 564000.0,
'vicky donor 2012': 549000.0,
'american pastoral 2016': 544000.0,
'double dhamaal 2011': 544000.0,
'the age of shadows 2016': 542000.0,
'being flynn 2012': 540000.0,
'gabbar is back 2015': 535000.0,
'ai weiwei never sorry 2012': 534000.0,
'fitoor 2016': 529000.0,
'the first monday in may 2016': 527000.0,
'the one i love 2014': 513000.0,
'our president 2017': 507000.0,
'the spy gone north 2018': 501000.0,
'peggy guggenheim art addict 2015': 498000.0,
'mere brother ki dulhan 2011': 496000.0,
'a girl walks home alone at night 2014': 492000.0,
'wienerdog 2016': 477000.0,
'the happy prince 2018': 466000.0,
"let's get married 2015": 463000.0,
'circumstance 2011': 454000.0,
'bombay velvet 2015': 451000.0,
'attack on titan part 1 2015': 450000.0,
'song to song 2017': 444000.0,
'baaghi 2016': 438000.0,
'ferrari ki sawaari 2012': 434000.0,
'beijing love story 2014': 428000.0,
'listen to me marlon 2015': 426000.0,

'sound city 2013': 423000.0,
'heart of a dog 2015': 421000.0,
'grand masti 2013': 414000.0,
'project nim 2011': 411000.0,
'unbranded 2015': 410000.0,
'tiny furniture 2010': 392000.0,
'goodbye to language 2014': 390000.0,
'things to come 2016': 388000.0,
'the armstrong lie 2013': 383000.0,
'humpty sharma ki dulhania 2014': 380000.0,
'khiladi 786 2012': 379000.0,
'tom of finland 2017': 378000.0,
'smashed 2012': 377000.0,
'the second mother 2015': 377000.0,
'shootout at wadala 2013': 370000.0,
'katti batti 2015': 365000.0,
'mountain 2018': 365000.0,
'mr. donkey 2016': 356000.0,
'singham 2011': 351000.0,
'northern limit line 2015': 338000.0,
'i origins 2014': 336000.0,
"god's own country 2017": 336000.0,
'te3n 2016': 332000.0,
'liberal arts 2012': 327000.0,
'shamitabh 2015': 325000.0,
'compliance 2012': 319000.0,
'adore 2013': 319000.0,
'happy ending 2014': 314000.0,
'back to 1942 2012': 313000.0,
'the breadwinner 2017': 312000.0,
'in the land of blood and honey 2011': 304000.0,
"being elmo a puppeteer's journey 2011": 304000.0,
'happy end 2017': 302000.0,
'go goa gone 2013': 298000.0,
'something from nothing the art of rap 2012': 288000.0,
'aquarius 2016': 286000.0,
'the interrupters 2011': 282000.0,
'main tera hero 2014': 275000.0,
'rascals 2011': 274000.0,
'after the storm 2017': 272000.0,
'himmatwala 2013': 271000.0,
'kingsglaive final fantasy xv 2016': 270000.0,
'hitchcock/truffaut 2015': 260000.0,
'what they had 2018': 260000.0,
'the green prince 2014': 258000.0,
'love in the buff 2012': 256000.0,
'the blue room 2014': 255000.0,
'life, animated 2016': 255000.0,
'life in a day 2011': 253000.0,
'lola versus 2012': 253000.0,
'the fortress 2017': 253000.0,
'tazza the hidden card 2014': 252000.0,
'batti gul meter chalu 2018': 250000.0,
'black butler book of the atlantic 2017': 248000.0,
'keep the lights on 2012': 246000.0,
'oasis supersonic 2016': 243000.0,
'the swindlers 2017': 242000.0,

'tevar 2015': 236000.0,
'elena 2012': 233000.0,
'matangi/maya/m.i.a. 2018': 231000.0,
'tezz 2012': 230000.0,
'slow west 2015': 229000.0,
'a royal night out 2015': 228000.0,
'singh saab the great 2013': 226000.0,
'into the abyss 2011': 224000.0,
'pyaar ka punchnama 2 2015': 223000.0,
'abcd (any body can dance) 2013': 222000.0,
'the viral factor 2012': 220000.0,
'fall in love like a star 2015': 213000.0,
'tai chi zero 2012': 212000.0,
'the house i live in 2012': 211000.0,
'dark places 2015': 209000.0,
'the breakup guru 2014': 209000.0,
'the prison 2017': 207000.0,
'heneral luna 2015': 206000.0,
'kaptaan 2016': 204000.0,
'ghanchakkar 2013': 203000.0,
'any day now 2012': 201000.0,
'a good old fashioned orgy 2011': 200000.0,
'listen up philip 2014': 200000.0,
'bol 2011': 189000.0,
'maggie 2015': 187000.0,
'detective k secret of the lost island 2015': 185000.0,
'the priests 2015': 185000.0,
'jig 2011': 184000.0,
'the other side of hope 2017': 184000.0,
'league of gods 2016': 182000.0,
'karwaan 2018': 182000.0,
'marjorie prime 2017': 181000.0,
'last letter 2018': 181000.0,
'the accidental detective 2 in action 2018': 179000.0,
'tere naal love ho gaya 2012': 178000.0,
'casino jack and the united states of money 2010': 177000.0,
'the guardians 2018': 177000.0,
'asura the city of madness 2016': 176000.0,
'that demon within 2014': 172000.0,
"god's pocket 2014": 170000.0,
'joker 2012': 169000.0,
'bellflower 2011': 168000.0,
'r... rajkumar 2013': 167000.0,
'we steal secrets the story of wikileaks 2013': 166000.0,
"c'est si bon 2015": 164000.0,
'the last women standing 2015': 163000.0,
'after the ball 2015': 162000.0,
'shirin farhad ki toh nikal padi 2012': 156000.0,
'experimenter 2015': 156000.0,
'the journey 2017': 155000.0,
'chakravyuh 2012': 152000.0,
'revenge of the electric car 2011': 151000.0,
'blade of the immortal 2017': 151000.0,
'the void 2017': 151000.0,
'museo 2018': 149000.0,
'rangeelay 2013': 148000.0,
'thelma 2017': 147000.0,

```
'luv shuv tey chicken khurana 2012': 144000.0,
'ingrid bergman in her own words 2015': 138000.0,
'1911 2011': 136000.0,
'mississippi grind 2015': 131000.0,
"spirits' homecoming 2016": 126000.0,
'bpm (beats per minute) 2017': 125000.0,
'explosion 2017': 123000.0,
'z for zachariah 2015': 121000.0,
'fire at sea 2016': 121000.0,
'three 2016': 120000.0,
'becoming astrid 2018': 120000.0,
'zero motivation 2014': 116000.0,
'love live! the school idol movie 2015': 116000.0,
'gonjiam haunted asylum 2018': 115000.0,
'ek thi daayan 2013': 112000.0,
'take point 2018': 112000.0,
'memories of the sword 2015': 111000.0,
'the negotiation 2018': 111000.0,
'the beauty inside 2015': 108000.0,
'hello i must be going 2012': 107000.0,
'the english teacher 2013': 105000.0,
'fabricated city 2017': 105000.0,
'namaste england 2018': 104000.0,
'god help the girl 2014': 103000.0,
'the golden era 2014': 103000.0,
'happy phirr bhag jayegi 2018': 103000.0,
'the magic of belle isle 2012': 102000.0,
'mugamoodi 2012': 101000.0,
'ghost in the shell the new movie 2015': 101000.0,
'man of tai chi 2013': 100000.0,
'v/h/s 2012': 100000.0,
'munna michael 2017': 99600.0,
'rock on 2 2016': 98900.0,
'seondal the man who sells the river 2016': 97500.0,
'so young 2 never gone 2016': 96400.0,
'chalo dilli 2011': 94300.0,
'saving mr. wu 2015': 92700.0,
'cuban fury 2014': 92400.0,
'passion 2013': 92200.0,
'christmas eve 2015': 91300.0,
'memoir of a murderer 2017': 91300.0,
'queen of earth 2015': 91200.0,
"the king's case note 2017": 91000.0,
'life after beth 2014': 88300.0,
'the house that jack built 2018': 88000.0,
'advanced style 2014': 87900.0,
'marina abramovic the artist is present 2012': 86600.0,
'author the jt leroy story 2016': 86000.0,
'the rooftop 2013': 85800.0,
'warriors of the dawn 2017': 84500.0,
'cock and bull 2016': 82800.0,
'a street cat named bob 2016': 82700.0,
'the whale 2011': 81900.0,
'we are what we are 2013': 81400.0,
'the nile hilton incident 2017': 81100.0,
'nasty baby 2015': 79800.0,
'pop aye 2017': 78000.0,
```

'just a breath away 2018': 78000.0,
'saala khadoos 2016': 76900.0,
'better living through chemistry 2014': 75100.0,
'keys to the heart 2018': 75100.0,
'helicopter eela 2018': 72000.0,
'dancer 2016': 71900.0,
'puncture 2011': 68900.0,
'phantom detective 2016': 67100.0,
'people places things 2015': 67000.0,
'a beautiful life 2011': 66200.0,
'eden 2015': 65500.0,
'my lucky star 2013': 64400.0,
'heropanti 2014': 63600.0,
'i can speak 2017': 63200.0,
'no tears for the dead 2014': 63100.0,
'flowers 2015': 61600.0,
'only you 2015': 61600.0,
'murder 2 2011': 59500.0,
'time renegades 2016': 59400.0,
'walk of shame 2014': 59200.0,
'vulgaria 2012': 59100.0,
'i smile back 2015': 58700.0,
'rudderless 2014': 58300.0,
'vanishing time a boy who returned 2016': 57800.0,
'the canyons 2013': 56800.0,
'extraordinary mission 2017': 54200.0,
'being 17 2016': 52700.0,
'jodi breakers 2012': 52600.0,
'my annoying brother 2016': 52000.0,
'like for likes 2016': 50800.0,
'painted skin the resurrection 2012': 50400.0,
'the advocate a missing body 2015': 50200.0,
'table no. 21 2013': 46700.0,
'margaret 2011': 46500.0,
'we are x 2016': 45300.0,
'double trouble 2012': 44800.0,
'a melody to remember 2016': 44300.0,
'phantom of the theatre 2016': 44000.0,
'mastizaade 2016': 43400.0,
'bluebeard 2017': 43100.0,
'hell and back again 2011': 40600.0,
'the phone 2015': 40500.0,
'what our fathers did a nazi legacy 2015': 40100.0,
'killing season 2013': 39900.0,
'khodorkovsky 2011': 39500.0,
'lila & eve 2015': 38300.0,
'ip man the final fight 2013': 37900.0,
'girls vs gangsters 2018': 37100.0,
'knock knock 2015': 36300.0,
'welcome to leith 2015': 36000.0,
'until forever 2016': 35800.0,
'camille claudel 1915 2013': 35300.0,
'wild city 2015': 34900.0,
'love in space 2011': 34800.0,
'the forbidden room 2015': 34400.0,
'white bird in a blizzard 2014': 33800.0,
'the hero of color city 2014': 32200.0,

```
'bird people 2014': 32100.0,
'the son of no one 2011': 30700.0,
'the bay 2012': 30700.0,
'the son of joseph 2016': 30600.0,
'muran 2011': 30100.0,
'for a few bullets 2016': 30100.0,
'the last princess 2016': 29300.0,
'the villainess 2017': 27700.0,
'lilting 2014': 27100.0,
"the new year's eve of old lee 2016": 25400.0,
'moscow never sleeps 2017': 25300.0,
'in another country 2012': 25100.0,
'bastards 2013': 24500.0,
'elliot the littlest reindeer 2018': 24300.0,
'the first time 2012': 22800.0,
"salut d'amour 2015": 22500.0,
'tyrannosaur 2011': 22300.0,
'mediterranea 2015': 22100.0,
'sanam teri kasam 2016': 22100.0,
'the workshop 2018': 22100.0,
'bending the arc 2017': 21900.0,
'v/h/s/2 2013': 21800.0,
'a paris education 2018': 21600.0,
'always kabhi kabhi 2011': 21400.0,
'the look of love 2013': 21300.0,
'term life 2016': 21300.0,
'rv resurrected victims 2017': 20400.0,
'guardian angel 2014': 19500.0,
'28 hotel rooms 2012': 18900.0,
'two night stand 2014': 18600.0,
'5 days of war 2011': 17500.0,
'strangerland 2015': 17500.0,
'extraction 2015': 16800.0,
'eva 2015': 16700.0,
'patience (after sebald) 2012': 15900.0,
'remember you 2016': 14800.0,
'a wedding invitation 2013': 14400.0,
'abracadabra 2017': 14000.0,
'the hallow 2015': 13900.0,
'13 sins 2014': 13800.0,
'camp xray 2014': 13300.0,
'red obsession 2013': 13200.0,
'blind 2017': 12500.0,
'kiki, love to love 2016': 12500.0,
'grassroots 2012': 12400.0,
'arabian nights volume 1  the restless one 2015': 12300.0,
'the selfish giant 2013': 12200.0,
'wolves 2014': 12100.0,
'falcon rising 2014': 11800.0,
'charlie countryman 2013': 11700.0,
"don't call me son 2016": 11700.0,
"you're not you 2014": 11500.0,
'the taqwacores 2010': 11400.0,
'cherry 2010': 11400.0,
'so young 2013': 11200.0,
'empire of lust 2015': 10700.0,
'stray dogs 2014': 10400.0,
```

```
 'husbands in goa 2012': 10100.0,
 'in the name of 2013': 9900.0,
 'welcome to the punch 2013': 9700.0,
 'every thing will be fine 2015': 9200.0,
 'the lesson 2015': 9000.0,
 'the last king 2016': 8900.0,
 'magnus 2016': 8900.0,
 'the girl in the book 2015': 8200.0,
 'the other side 2016': 8100.0,
 'the babymakers 2012': 7900.0,
 'overdrive 2017': 7800.0,
 'on the way to school 2015': 7400.0,
 'my piece of the pie 2011': 6900.0,
 'finding mr. right 2013': 6900.0,
 'steve mcqueen the man & le mans 2015': 6800.0,
 'paradise hope 2013': 6700.0,
 'i melt with you 2011': 6400.0,
 'arabian nights volume 2  the desolate one 2015': 6400.0,
 'the carer 2016': 6200.0,
 'the quake 2018': 6200.0,
 'barely lethal 2015': 6100.0,
 "the devil's hand 2014": 5700.0,
 'chic! 2015': 5600.0,
 "jenny's wedding 2015": 4700.0,
 'ashby 2015': 4600.0,
 'the summer of sangaile 2015': 4500.0,
 'arabian nights volume 3  the enchanted one 2015': 4500.0,
 'club life 2015': 4400.0,
 'back to the jurassic 2015': 4400.0,
 'singularity 2017': 4200.0,
 'sleepless night 2012': 3400.0,
 'atm 2012': 3000.0,
 'about cherry 2012': 3000.0,
 'cheerful weather for the wedding 2012': 2700.0,
 'knuckle 2011': 2600.0,
 'the lady in the car with glasses and a gun 2015': 2200.0,
 'love, wedding, marriage 2011': 1900.0,
 'open windows 2014': 1800.0,
 'an actor prepares 2018': 1700.0,
 'i spit on your grave 2 2013': 800.0,
 'down by love 2016': 700.0,
 '222 2017': 400.0,
 'satanic 2016': 300.0,
 'news from planet mars 2016': 300.0}
```

In [38]:
```
len(null_foreign_earnings)
```

Out[38]:
```
609
```

Only 5 of 609 movies with missing foreign gross data earned above or near the average domestic gross of $43 million. Since these movies generally have lower impact on the dataset's revenue, we will replace the null values with 0 rather than median or mean:

In [39]:
```python
# Assigning foreign gross null values to $0
bom_imdb['foreign_gross'] = bom_imdb['foreign_gross'].fillna('0')
```

```
<ipython-input-39-b491f3028515>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  bom_imdb['foreign_gross'] = bom_imdb['foreign_gross'].fillna('0')
```

In [40]:
```python
bom_imdb.dtypes
```

Out[40]:
```
tconst              object
primary_title       object
original_title      object
start_year           int64
runtime_minutes    float64
genres              object
title_year          object
index                int64
title               object
studio              object
domestic_gross     float64
foreign_gross       object
year                 int64
dtype: object
```

In [41]:
```python
# Cleaning foreign_gross so it can be translated from type string to float
foreign_adjusted = []
for gross in bom_imdb['foreign_gross']:
    foreign_adjusted.append(gross.replace(",", ""))
bom_imdb['foreign_gross'] = foreign_adjusted
```

```
<ipython-input-41-3874b9b0c9be>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  bom_imdb['foreign_gross'] = foreign_adjusted
```

In [42]:
```
bom_imdb['foreign_gross'] = bom_imdb['foreign_gross'].astype('float64')
```

```
<ipython-input-42-125bfd2a1a68>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  bom_imdb['foreign_gross'] = bom_imdb['foreign_gross'].astype('float64')
```

In [43]:
```
bom_imdb.dtypes
```

Out[43]:
```
tconst              object
primary_title       object
original_title      object
start_year           int64
runtime_minutes    float64
genres              object
title_year          object
index                int64
title               object
studio              object
domestic_gross     float64
foreign_gross      float64
year                 int64
dtype: object
```

Now let's create a new column combining total revenue by adding domestic and foreign gross together:

In [44]:
```
total_gross = []
for row in bom_imdb.index:
    total_gross.append(bom_imdb['domestic_gross']
                       [row] + bom_imdb['foreign_gross'][row])
bom_imdb['total_gross'] = total_gross
bom_imdb['total_gross']
```

```
<ipython-input-44-943edd358b1e>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  bom_imdb['total_gross'] = total_gross
```

Out[44]:
```
0         1100000.0
1         8744000.0
2       188100000.0
3        53200000.0
4       652301019.4
            ...
1918       365000.0
1919     82847000.0
1920        72000.0
1921       181000.0
1922     20300000.0
Name: total_gross, Length: 1912, dtype: float64
```

In [45]:
```
bom_imdb['total_gross'].isna().sum()
```

Out[45]:
```
0
```

Parsing Out Genre Information

In order to analyze this movie data by genre, we must create new columns indicating if a movie falls into any or multiple genres. Firstly we will find a unique list of genres:

In [46]:
```python
# Retrieving unique genres list
unique_genres_list = []
for genre_details in bom_imdb['genres']:
    genres_list = genre_details.split(",")
    for genre in genres_list:
        unique_genres_list.append(genre)

unique_genres_list = sorted(list(set(unique_genres_list)))
unique_genres_list
```

Out[46]:
```
['Action',
 'Adventure',
 'Animation',
 'Biography',
 'Comedy',
 'Crime',
 'Documentary',
 'Drama',
 'Family',
 'Fantasy',
 'History',
 'Horror',
 'Music',
 'Musical',
 'Mystery',
 'News',
 'Romance',
 'Sci-Fi',
 'Sport',
 'Thriller',
 'War',
 'Western']
```

In [47]:
```python
# Creating and populating a column for each genre
for genre in unique_genres_list:
    bom_imdb[genre] = bom_imdb['genres'].apply(
        lambda x: genre in x).astype('int')
bom_imdb.head()
```

	tconst	primary_title	original_title	start_year	runtime_minutes	genres	t
0	tt0315642	wazir	Wazir	2016	103.0	Action,Crime,Drama	wa
1	tt0337692	on the road	On the Road	2012	124.0	Adventure,Drama,Romance	on
2	tt0359950	the secret life of walter mitty	The Secret Life of Walter Mitty	2013	114.0	Adventure,Comedy,Drama	th wa
3	tt0365907	a walk among the tombstones	A Walk Among the Tombstones	2014	114.0	Action,Crime,Drama	ar ton
4	tt0369610	jurassic world	Jurassic World	2015	124.0	Action,Adventure,Sci-Fi	wo

5 rows × 36 columns

In [48]:
```python
bom_imdb.sort_values(by=['total_gross'], ascending=False).head(50)
```

Out[48]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genre
1111	tt2395427	avengers age of ultron	Avengers: Age of Ultron	2015	141.0	Action,Adventure,Sci-F
761	tt1825683	black panther	Black Panther	2018	134.0	Action,Adventure,Sci-F
212	tt1201607	harry potter and the deathly hallows part 2	Harry Potter and the Deathly Hallows: Part 2	2011	130.0	Adventure,Drama,Fantas
1157	tt2527336	star wars the last jedi	Star Wars: Episode VIII - The Last Jedi	2017	152.0	Action,Adventure,Fantas

In [49]:
```python
# Viewing mean total gross by genre
mean_total_gross = []
for genre in unique_genres_list:
    genre_gross = bom_imdb.loc[bom_imdb[genre] == 1]['total_gross'].mean()
    mean_total_gross.append((genre, genre_gross))
mean_total_gross
```

Out[49]:
```
[('Action', 190190685.60348624),
 ('Adventure', 320389676.60716176),
 ('Animation', 314138988.5934959),
 ('Biography', 58189028.70769231),
 ('Comedy', 107011699.4011713),
 ('Crime', 62274617.34909091),
 ('Documentary', 10447066.657407407),
 ('Drama', 55344769.441955194),
 ('Family', 154735784.07954547),
 ('Fantasy', 223227357.56060606),
 ('History', 57728077.0),
 ('Horror', 78360422.5165563),
 ('Music', 61411912.16216216),
 ('Musical', 84580666.66666667),
 ('Mystery', 88703551.07194245),
 ('News', 13200.0),
 ('Romance', 43765660.18446602),
 ('Sci-Fi', 339625634.71339285),
 ('Sport', 43154185.71428572),
 ('Thriller', 105161419.42123288),
 ('War', 31476324.0),
 ('Western', 90589508.33333333)]
```

Q1. Data Modeling

Plotting the Top 5 Genres

Microsoft is worldwide brand and and a household company name. With their name recognition and resources, Microsoft can and should aim to make a splash by making movies in the top 5 revenue earning genres.

In this section we will:

- Identify the top 5 genres by total revenue
- Plot each genre's mean revenue per year from 2010-2018 to analyze trends

In [50]:
```python
# Finding top 5 genres by mean total gross
def sort_tuple(tup):
    return (sorted(tup, key=lambda x: x[1], reverse=True))
```

In [153]:
```python
sort_tuple(mean_total_gross)
```

Out[153]:
```
[('Sci-Fi', 339625634.71339285),
 ('Adventure', 320389676.60716176),
 ('Animation', 314138988.5934959),
 ('Fantasy', 223227357.56060606),
 ('Action', 190190685.60348624),
 ('Family', 154735784.07954547),
 ('Comedy', 107011699.4011713),
 ('Thriller', 105161419.42123288),
 ('Western', 90589508.33333333),
 ('Mystery', 88703551.07194245),
 ('Musical', 84580666.66666667),
 ('Horror', 78360422.5165563),
 ('Crime', 62274617.34909091),
 ('Music', 61411912.16216216),
 ('Biography', 58189028.70769231),
 ('History', 57728077.0),
 ('Drama', 55344769.441955194),
 ('Romance', 43765660.18446602),
 ('Sport', 43154185.71428572),
```

We will target the top 5 genres based on revenue because:

1. Their revenue is significantly more than the remaining genres
2. There are obvious overlaps in the types of movies in the top 5

In [51]:
```python
top_5_mean_total_gross = sort_tuple(mean_total_gross)[:5]
top_5_mean_total_gross
```

Out[51]:
```
[('Sci-Fi', 339625634.71339285),
 ('Adventure', 320389676.60716176),
 ('Animation', 314138988.5934959),
 ('Fantasy', 223227357.56060606),
 ('Action', 190190685.60348624)]
```

Analyzing the data: From 2010-2018, on average a movie released under the Sci-Fi genre would have earned $339,625,635.

Next, let's plot the average to see how Sci-Fi films performed in each year:

In [52]:
```python
# Preparing a dataframe(sci-fi) for plotting
scifi_drop_rows = []
for row in bom_imdb.index:
    if bom_imdb['Sci-Fi'][row] == 0:
        scifi_drop_rows.append(row)
scifi_drop_rows[:5]  # Delete all non science fiction movies
```

Out[52]:
```
[0, 1, 2, 3, 5]
```

In [53]:
```
scifi_plot = bom_imdb.drop(scifi_drop_rows)
print(len(scifi_plot))
```

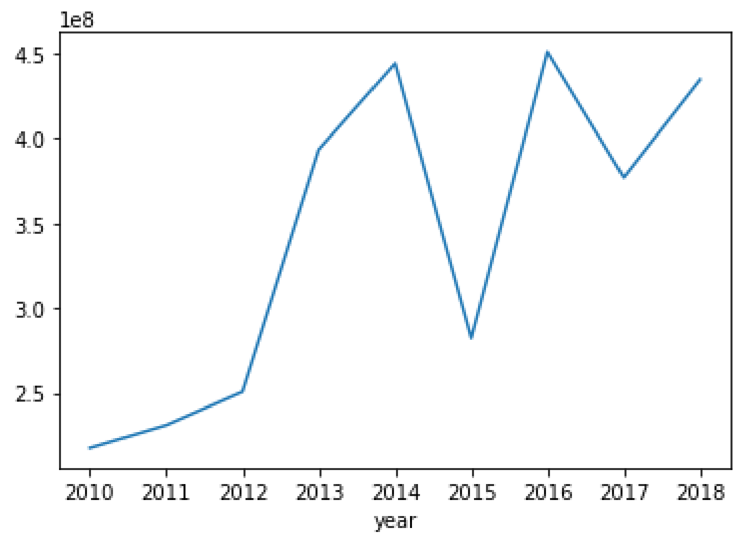
```
112
```

In [54]:
```
# Finding Sci-Fi average revenue per year
scifi_plot.groupby(['year'])['total_gross'].mean()
```

Out[54]:
```
year
2010    2.178546e+08
2011    2.312033e+08
2012    2.508202e+08
2013    3.933600e+08
2014    4.441169e+08
2015    2.823964e+08
2016    4.508778e+08
2017    3.770833e+08
2018    4.345847e+08
Name: total_gross, dtype: float64
```

In [55]:
```
# Plotting Sci-Fi average revenue per year
scifi_plot.groupby(['year'])['total_gross'].mean().plot()
```

Out[55]:
```
<AxesSubplot:xlabel='year'>
```



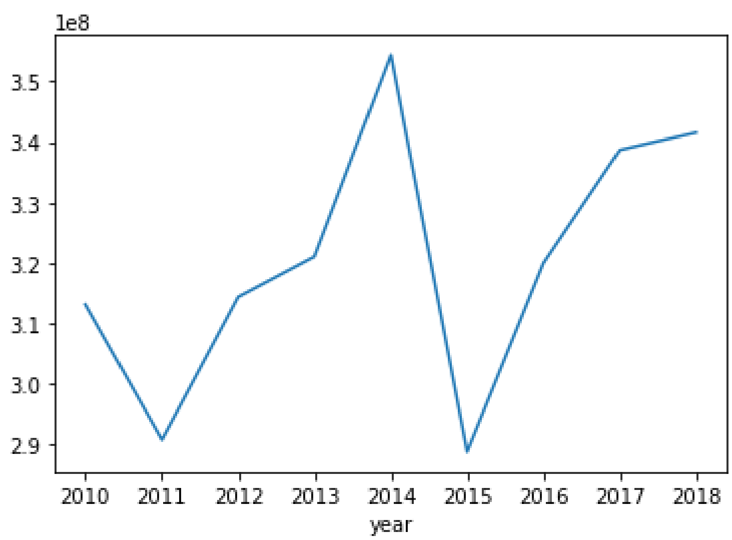
Looks like Sci-Fi films have been gradually performing better from 2010 to 2018! Let's look at the remaining 4 top genres:

In [56]:
```python
# Define a function to plot genre by total revenue over time
def genre_gross_plot(genre, df):
    genre_drop_rows = []
    for row in df.index:
        if df[genre.capitalize()][row] == 0:
            genre_drop_rows.append(row)

    genre_plot = df.drop(genre_drop_rows)
    return genre_plot.groupby(['year'
                              ])['total_gross'].mean(), genre_plot.groupby
                                 (['year'])['total_gross'].mean().plot()
```

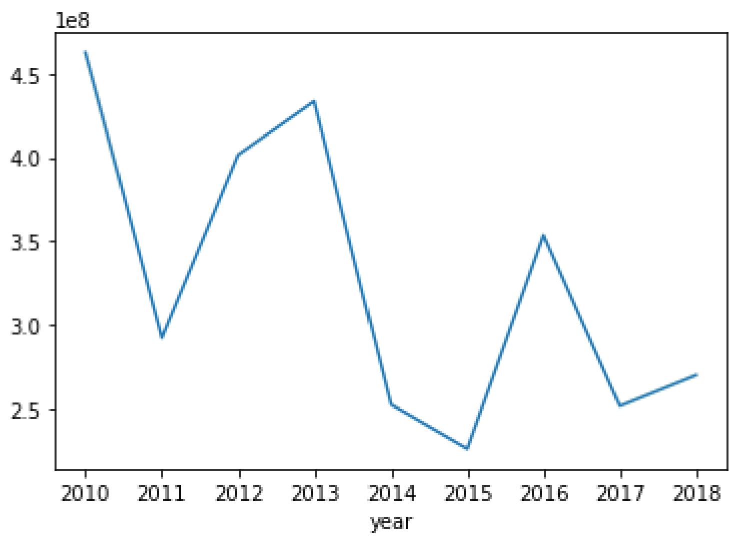
In [57]:
```python
# Plotting Adventure average revenue per year
genre_gross_plot('adventure', bom_imdb)
```

Out[57]:
```
(year
 2010    3.131605e+08
 2011    2.908265e+08
 2012    3.144131e+08
 2013    3.210872e+08
 2014    3.543894e+08
 2015    2.888620e+08
 2016    3.199951e+08
 2017    3.386920e+08
 2018    3.416055e+08
 Name: total_gross, dtype: float64,
 <AxesSubplot:xlabel='year'>)
```



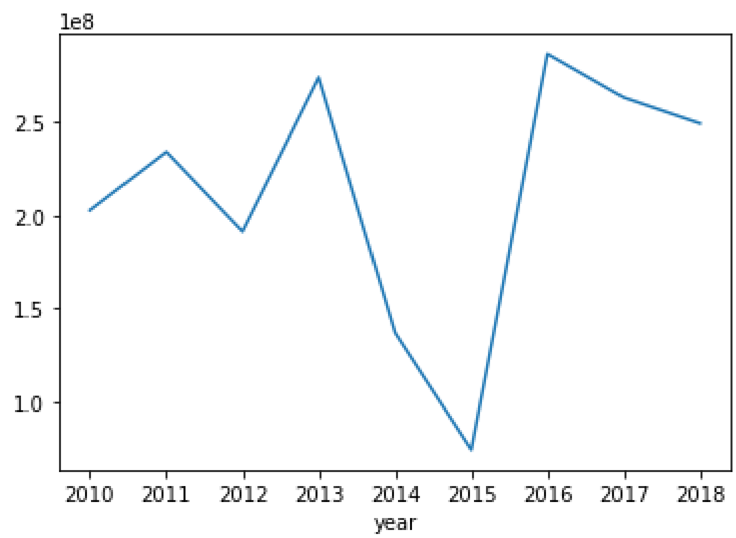
In [58]:
```
# Plotting Animation average revenue per year
genre_gross_plot('animation', bom_imdb)
```

Out[58]:
```
(year
 2010    4.626000e+08
 2011    2.922837e+08
 2012    4.014222e+08
 2013    4.335668e+08
 2014    2.524545e+08
 2015    2.260832e+08
 2016    3.535400e+08
 2017    2.516094e+08
 2018    2.699567e+08
 Name: total_gross, dtype: float64,
 <AxesSubplot:xlabel='year'>)
```



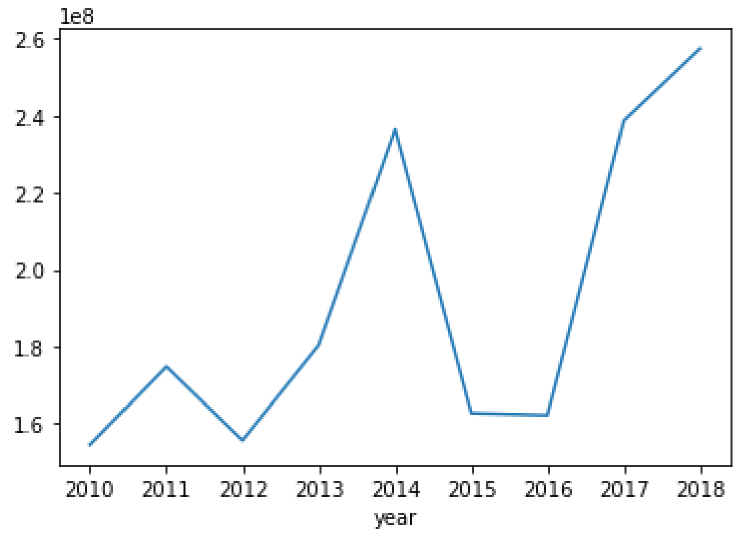
```
In [59]: # Plotting Fantasy average revenue per year
genre_gross_plot('fantasy', bom_imdb)
```

```
Out[59]: (year
 2010    2.028074e+08
 2011    2.341724e+08
 2012    1.913293e+08
 2013    2.741364e+08
 2014    1.367720e+08
 2015    7.391667e+07
 2016    2.867496e+08
 2017    2.632766e+08
 2018    2.493849e+08
 Name: total_gross, dtype: float64,
 <AxesSubplot:xlabel='year'>)
```



In [60]:
```
# Plotting Action average revenue per year
genre_gross_plot('action', bom_imdb)
```

Out[60]:
```
(year
 2010    1.545023e+08
 2011    1.747712e+08
 2012    1.556103e+08
 2013    1.803327e+08
 2014    2.365366e+08
 2015    1.626865e+08
 2016    1.621667e+08
 2017    2.387588e+08
 2018    2.575156e+08
 Name: total_gross, dtype: float64,
 <AxesSubplot:xlabel='year'>)
```




```
![example](images/avg_rev.png)
```

Q1 D. Data Evaluation

Interpreting the Results:

Of the top 5, the Sci-Fi genre seems to have the strongest upward momentum. The Adventure genre has also plateaud at a high level above almost every other genre. I would recommend Microsoft to explore making these types of movies and obtaining the intellectual property rights to sci-fi and adventure franchises (book series, video games, etc.).

Question 2: How long should a movie be to maximize revenue?

In this section we will attempt to see if there are movie runtime "sweet spots" in each genre for movie makers to aim for.

Q2 A. Data Understanding

We will be using the same datasets from the previous question:

1. imdb.title.basics - This dataset comes from IMDb. We will again use the genre information, and additionally will focus on movie runtime in minutes.
2. bom.movie.gross - This dataset comes from BOM. The target variables are domestic (USA) gross and foreign gross.

Q2 B. Data Preparation

We have previously replaced missing runtime data with the median runtime for all movies during the data preparation section of Q1. Next we will create groups by runtime. This will be achieved by organizing groups into bin intervals of 5 minutes:

In [62]:
```
# Finding min/max of all movies' runtime minutes
bom_imdb['runtime_minutes'].min()
```

Out[62]: 15.0

In [63]:
```
bom_imdb['runtime_minutes'].max()
```

Out[63]: 189.0

In [64]:
```
# Dividing all movies into 5-minute invervals to make interpretation of res
# This step makes the applicable data a tad less granular.
minutes = 15

while minutes <= 185:
    bom_imdb.loc[(bom_imdb['runtime_minutes'] >= minutes)
                 & (bom_imdb['runtime_minutes'] < (minutes + 5)),
                 'runtime_minutes'] = minutes
    minutes += 5
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  isetter(loc, value)
/Users/dan/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/pandas/core/indexing.py:1765: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  isetter(loc, value)
/Users/dan/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/pan

das/core/indexing.py:1765: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

In [65]:
```
bom_imdb['runtime_minutes'].head()
```

Out[65]:
```
0    100.0
1    120.0
2    110.0
3    110.0
4    120.0
Name: runtime_minutes, dtype: float64
```

Q2 C. Data Modeling

From here we will use steps very similar to the above Q1 to plot out the relationship between runtime and revenue:

In [66]:
```
# Preparing a dataframe(sci-fi) for plotting
scifi_runtime_drop_rows = []
for row in bom_imdb.index:
    if bom_imdb['Sci-Fi'][row] == 0:
        scifi_runtime_drop_rows.append(row)
scifi_runtime_drop_rows[:5]
```

Out[66]:
```
[0, 1, 2, 3, 5]
```

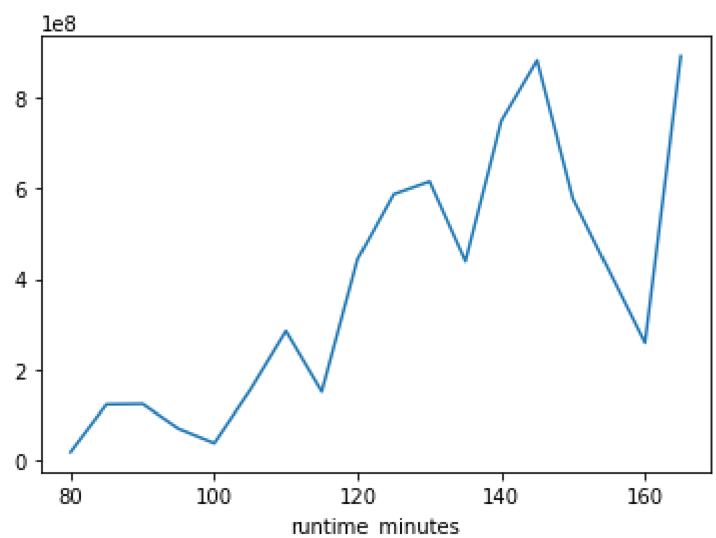
In [67]:
```
scifi_runtime_plot = bom_imdb.drop(scifi_runtime_drop_rows)
print(len(scifi_runtime_plot))
```

```
112
```

In [68]:
```
# Preparing sci-fi plot
scifi_runtime_plot.groupby(['runtime_minutes'])['total_gross'].mean()
```

Out[68]:
```
runtime_minutes
80.0      1.921535e+07
85.0      1.237195e+08
90.0      1.247438e+08
95.0      7.067762e+07
100.0     3.858375e+07
105.0     1.553780e+08
110.0     2.856672e+08
115.0     1.518143e+08
120.0     4.444001e+08
125.0     5.872500e+08
130.0     6.152038e+08
135.0     4.393000e+08
140.0     7.476714e+08
145.0     8.813503e+08
150.0     5.771667e+08
160.0     2.593000e+08
165.0     8.907000e+08
Name: total_gross, dtype: float64
```

In [69]:
```
scifi_runtime_plot.groupby(['runtime_minutes'])['total_gross'].mean().plot(
```

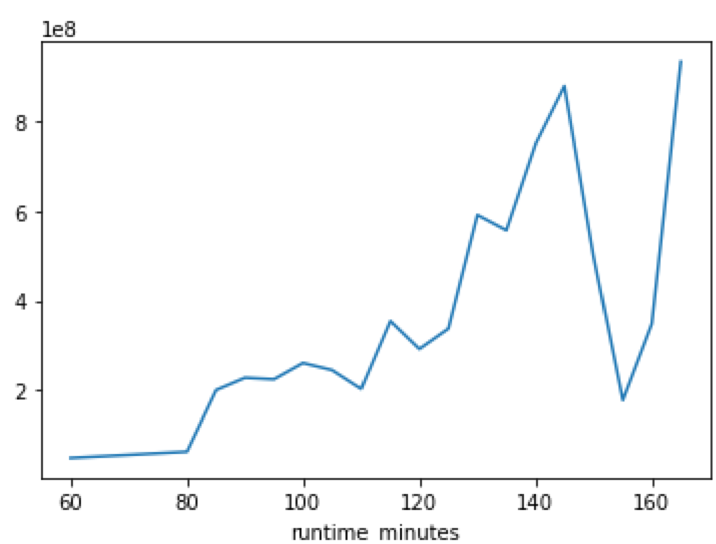


In [70]:
```
# def a function to plot genre by average revenue over runtime minutes
def genre_gross_time_plot(genre, df):
    genre_drop_rows = []
    for row in df.index:
        if df[genre.capitalize()][row] == 0:
            genre_drop_rows.append(row)

    genre_plot = df.drop(genre_drop_rows)
    return (genre_plot.groupby(['runtime_minutes'])['total_gross'].mean(),
            genre_plot.groupby(['runtime_minutes'
                                ])['total_gross'].mean().plot())
```

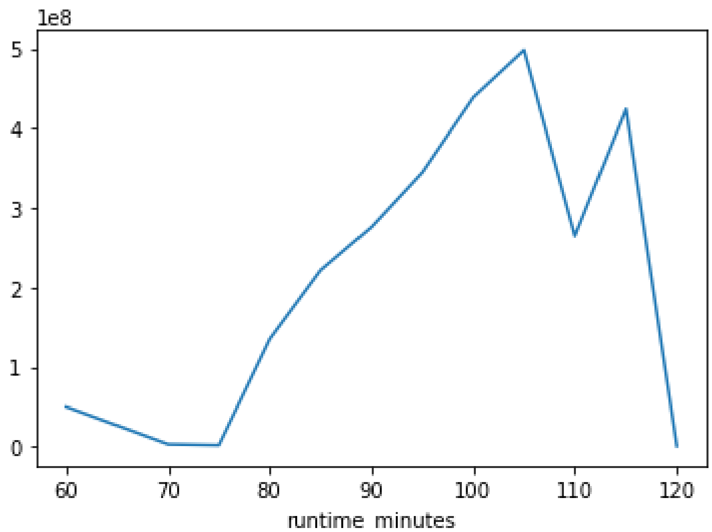
In [71]:
```
genre_gross_time_plot('adventure', bom_imdb)
```

Out[71]:
```
(runtime_minutes
 60.0      4.990000e+07
 80.0      6.382967e+07
 85.0      2.009186e+08
 90.0      2.283674e+08
 95.0      2.249872e+08
 100.0     2.613353e+08
 105.0     2.455996e+08
 110.0     2.036958e+08
 115.0     3.550379e+08
 120.0     2.927202e+08
 125.0     3.381840e+08
 130.0     5.922708e+08
 135.0     5.581400e+08
 140.0     7.493786e+08
 145.0     8.796573e+08
 150.0     5.009557e+08
 155.0     1.782993e+08
 160.0     3.492210e+08
 165.0     9.341667e+08
 Name: total_gross, dtype: float64,
 <AxesSubplot:xlabel='runtime_minutes'>)
```



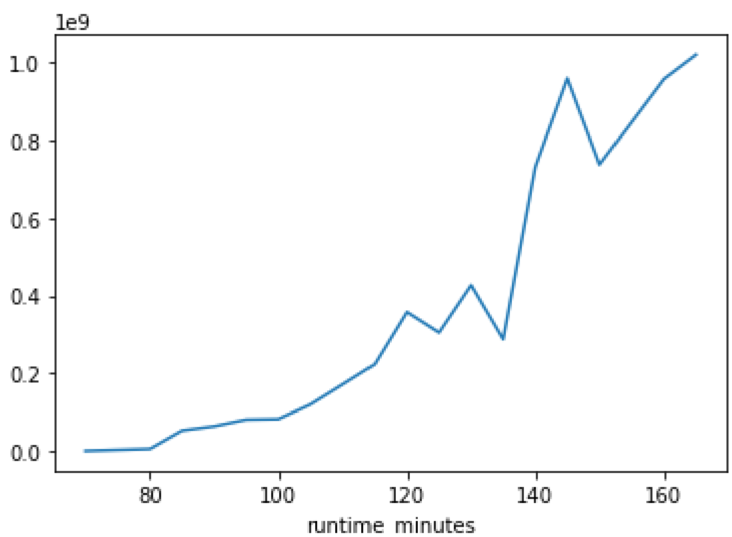
In [72]:
```
genre_gross_time_plot('animation', bom_imdb)
```

Out[72]:
```
(runtime_minutes
 60.0     4.990000e+07
 70.0     3.055000e+06
 75.0     1.916100e+06
 80.0     1.350000e+08
 85.0     2.215555e+08
 90.0     2.760704e+08
 95.0     3.448203e+08
 100.0    4.386921e+08
 105.0    4.981300e+08
 110.0    2.647350e+08
 115.0    4.242625e+08
 120.0    8.170000e+05
 Name: total_gross, dtype: float64,
 <AxesSubplot:xlabel='runtime_minutes'>)
```



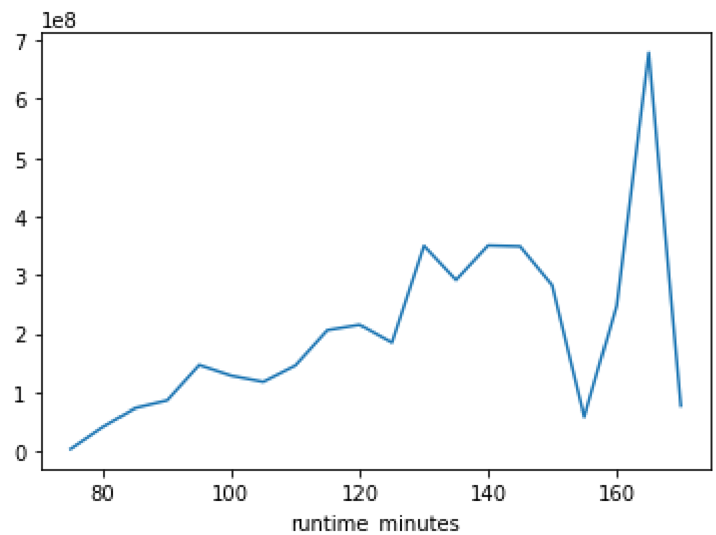
In [73]:
```
genre_gross_time_plot('fantasy', bom_imdb)
```

Out[73]:
```
(runtime_minutes
 70.0      3.900000e+05
 80.0      5.479333e+06
 85.0      5.264990e+07
 90.0      6.353583e+07
 95.0      8.063195e+07
 100.0     8.198467e+07
 105.0     1.200900e+08
 110.0     1.714619e+08
 115.0     2.233414e+08
 120.0     3.577679e+08
 125.0     3.050040e+08
 130.0     4.271787e+08
 135.0     2.877113e+08
 140.0     7.314628e+08
 145.0     9.603000e+08
 150.0     7.380667e+08
 160.0     9.584000e+08
 165.0     1.021100e+09
 Name: total_gross, dtype: float64,
 <AxesSubplot:xlabel='runtime_minutes'>)
```



In [74]:
```
genre_gross_time_plot('action', bom_imdb)
```

Out[74]:
```
(runtime_minutes
 75.0     3.800000e+06
 80.0     4.134267e+07
 85.0     7.306688e+07
 90.0     8.612586e+07
 95.0     1.465068e+08
 100.0    1.282422e+08
 105.0    1.177240e+08
 110.0    1.455689e+08
 115.0    2.060293e+08
 120.0    2.151620e+08
 125.0    1.847785e+08
 130.0    3.500680e+08
 135.0    2.917108e+08
 140.0    3.506070e+08
 145.0    3.491165e+08
 150.0    2.827474e+08
 155.0    5.845390e+07
 160.0    2.476655e+08
 165.0    6.791000e+08
 170.0    7.695000e+07
 Name: total_gross, dtype: float64,
 <AxesSubplot:xlabel='runtime_minutes'>)
```



In [75]:
```python
# Viewing movie data in the 165-minute spike
bom_imdb.loc[bom_imdb['runtime_minutes'] == 165]
```

Out[75]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres	
75	tt0816692	interstellar	Interstellar	2014	165.0	Adventure,Drama,Sci-Fi	
100	tt0903624	the hobbit an unexpected journey	The Hobbit: An Unexpected Journey	2012	165.0	Adventure,Family,Fantasy	u
154	tt1065073	boyhood	Boyhood	2014	165.0	Drama	
199	tt1188996	my name is khan	My Name Is Khan	2010	165.0	Drama	n
779	tt1853728	django unchained	Django Unchained	2012	165.0	Drama,Western	
803	tt1891757	bol	Bol	2011	165.0	Drama	
933	tt2109248	transformers age of extinction	Transformers: Age of Extinction	2014	165.0	Action,Adventure,Sci-Fi	trɛ
1364	tt3460252	the hateful eight	The Hateful Eight	2015	165.0	Crime,Drama,Mystery	
1629	tt4849438	baahubali 2 the conclusion	Baahubali 2: The Conclusion	2017	165.0	Action,Drama	b (

9 rows × 36 columns

In [76]:
```python
bom_imdb.loc[bom_imdb['runtime_minutes'] == 165]['total_gross']
```

Out[76]:
```
75      6.774000e+08
100     1.021100e+09
154     4.450000e+07
199     4.230000e+07
779     4.254000e+08
803     1.890000e+05
933     1.104000e+09
1364    1.557000e+08
1629    2.542000e+08
Name: total_gross, dtype: float64
```

The Hobbit and Transformers both made over $1 billion! It is worth looking at the individual movies within the 165-169 min group. However we will not be treating them as outliers

In [77]:

```
# Viewing movie data in the 150-159 minute valley
bom_imdb.loc[(bom_imdb['runtime_minutes'] >= 150) &
             (bom_imdb['runtime_minutes'] < 160)]
```

Out[77]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genre
18	tt0443272	lincoln	Lincoln	2012	150.0	Biography,Drama,Histo
31	tt0466893	margaret	Margaret	2011	150.0	Dram
137	tt1029231	krrish 3	Krrish 3	2013	150.0	Action,Adventure,Sci-l
142	tt1034415	suspiria	Suspiria	2018	150.0	Fantasy,Horror,Myste
220	tt1210819	the lone ranger	The Lone Ranger	2013	150.0	Action,Adventure,Weste
356	tt1375789	race 2	Race 2	2013	150.0	Action,Crime,Thrill
369	tt1392214	prisoners	Prisoners	2013	150.0	Crime,Drama,Myste
371	tt1395025	agent vinod	Agent Vinod	2012	155.0	Action,Adventure,Thrill

In [78]:
```python
bom_imdb.loc[(bom_imdb['runtime_minutes'] >= 150) & (
    bom_imdb['runtime_minutes'] < 160)]['total_gross']
```

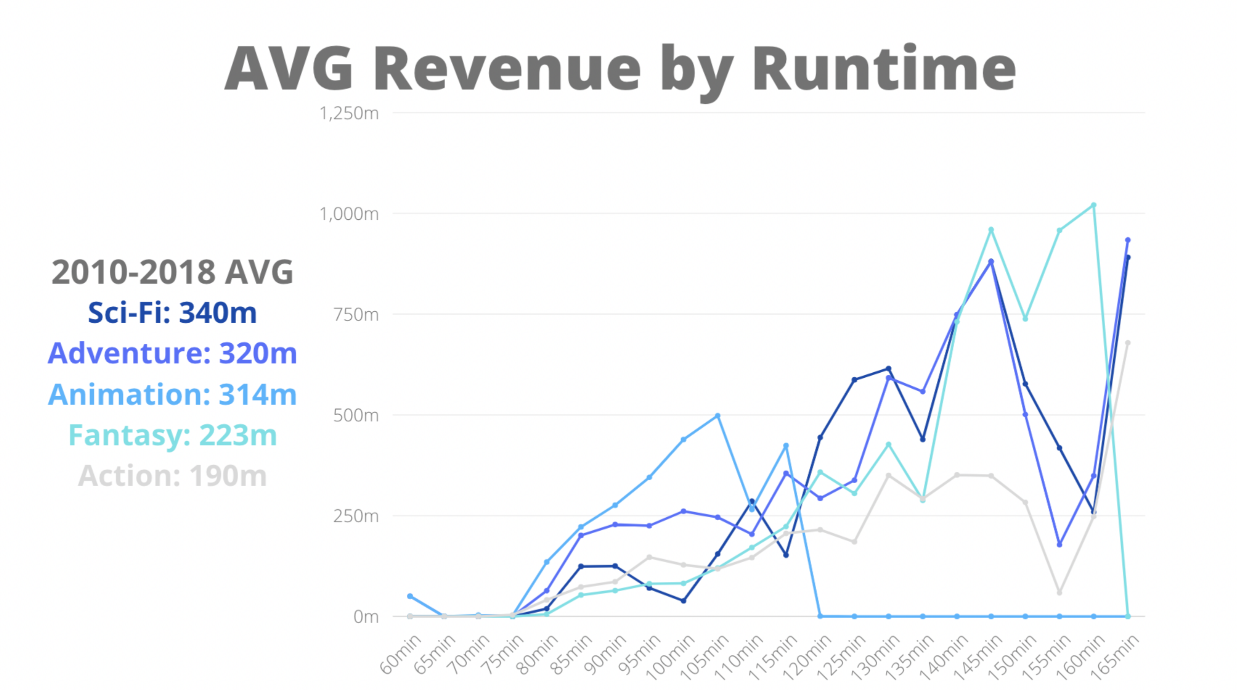
Out[78]:
```
18      2.753000e+08
31      4.650000e+04
137     2.200000e+06
142     7.900000e+06
220     2.605000e+08
356     1.600000e+06
369     1.221000e+08
371     6.980000e+05
377     1.123800e+09
434     1.041000e+06
470     1.616000e+06
484     2.682000e+08
515     2.035000e+06
516     3.100000e+06
643     5.329000e+08
700     2.880000e+07
739     1.328000e+08
801     9.430000e+04
913     2.800000e+06
1017    2.600000e+06
1023    7.390000e+05
1032    1.520000e+05
1040    3.700000e+05
1071    2.710000e+05
1085    1.010000e+05
1146    2.220000e+07
1157    1.332600e+09
1182    6.700000e+06
1222    5.434000e+06
1241    4.200000e+07
1260    8.737000e+08
1305    1.400000e+06
1346    6.055000e+08
1353    2.360000e+05
1432    6.600000e+06
1458    3.250000e+05
1464    1.300000e+06
1483    8.800000e+04
1558    4.900000e+06
1563    2.840000e+07
1649    1.500000e+06
1662    3.900000e+06
1687    2.210000e+04
1709    2.700000e+06
1720    5.670000e+05
1780    2.400000e+06
1809    1.285900e+07
1826    7.150000e+07
Name: total_gross, dtype: float64
```

Target Demographic

It seems there's a large swath of foreign films in this runtime group that have brought the average revenue down. We will not treat these films as outliers nor delete them. Instead it is important to take away that there are still some very high performing films in this time frame such as Transformers 1.1B, Star Wars 1.3B, Batman vs. Superman 874M. Additionally, this only confirms the 165-169 runtime group is not an outlier for this dataset!

What should this tell us about the demographic Microsoft should be targetting? If Microsoft wants to maximize their revenue, they should be targetting domestic rather than foreign audiences.

Insert graph from non-tech pres



Q2 D. Data Evaluation

In [79]:
```python
# Finding mean runtime for all movies
bom_imdb['runtime_minutes'].mean()
```

Out[79]:
```
108.90428870292887
```

Runtime Recommendations

- Sci-Fi: 140-169 minutes
- Adventure: 140-169 minutes
- Animation: 100-119 minutes
- Fantasy: 140-164 minutes
- Action: 130-169 minutes

Longer Movies?

Though longer movies typically cost more to make, the data shows that if Microsoft wants to enter the movie industry at the top, they should be making blockbuster films with longer than average runtimes, as is the trend for many of these top revenue making films.

Question 3: What movie personnel will maximize IMDB ratings?

Users of the popular movie website IMDB are able to rate movies they've viewed on a scale from 0-10. In this section we will explore which specific cast and crew personnel associate with higher scoring movies.

Q3 A. Data Understanding

In this section we will be using three new datasets:

1. imdb.title.ratings - This dataset comes from IMDB. The 'Average Rating' variable will provide our success metric, while the 'Number of Votes' variable will help us weigh which average ratings to include and exclude.
2. imdb_title_principals - This dataset comes from IMDB. It associates personnel with the movie they worked on, in addition to their role.
3. imdb_name_basics - This dataset comes from IMDB. It provides the full name of individual personnel and the movie titles (max 6) they are most known for.

In [80]:
```
imdb_title_ratings = pd.read_csv('zippedData/imdb.title.ratings.csv.gz')
imdb_title_ratings
```

Out[80]:

	tconst	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21
...	...	...	...
73851	tt9805820	8.1	25
73852	tt9844256	7.5	24
73853	tt9851050	4.7	14
73854	tt9886934	7.0	5
73855	tt9894098	6.3	128

73856 rows × 3 columns

In [81]:
```python
imdb_title_principals = pd.read_csv('zippedData/imdb.title.principals.csv.g
imdb_title_principals
```

Out[81]:

	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"]
...	...	...	...	...	...	...
1028181	tt9692684	1	nm0186469	actor	NaN	["Ebenezer Scrooge"]
1028182	tt9692684	2	nm4929530	self	NaN	["Herself","Regan"]
1028183	tt9692684	3	nm10441594	director	NaN	NaN
1028184	tt9692684	4	nm6009913	writer	writer	NaN
1028185	tt9692684	5	nm10441595	producer	producer	NaN

In [82]:
```python
imdb_name_basics = pd.read_csv('zippedData/imdb.name.basics.csv.gz')
imdb_name_basics
```

Out[82]:

	nconst	primary_name	birth_year	death_year	primary_pr
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,|
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_de|
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,ac
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_de|
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_c
...	...	...	...	...	
606643	nm9990381	Susan Grobes	NaN	NaN	
606644	nm9990690	Joo Yeon So	NaN	NaN	
606645	nm9991320	Madeline Smith	NaN	NaN	

Q3 B. Data Preparation

In this section we will:

- Clean our datasets in preparation to merge
- Consider films familiar to an average domestic movie watcher

First we will drop all the columns we don't need:

In [83]:
```
imdb_title_principals.drop(labels=['ordering', 'job', 'characters'],
                           axis=1,
                           inplace=True)
imdb_title_principals
```

Out[83]:

	tconst	nconst	category
0	tt0111414	nm0246005	actor
1	tt0111414	nm0398271	director
2	tt0111414	nm3739909	producer
3	tt0323808	nm0059247	editor
4	tt0323808	nm3579312	actress
...	...	...	...
1028181	tt9692684	nm0186469	actor
1028182	tt9692684	nm4929530	self
1028183	tt9692684	nm10441594	director
1028184	tt9692684	nm6009913	writer
1028185	tt9692684	nm10441595	producer

1028186 rows × 3 columns

In [84]:
```
imdb_name_basics.drop(labels=['primary_profession', 'birth_year'],
                      axis=1,
                      inplace=True)
imdb_name_basics
```

Out[84]:

	nconst	primary_name	death_year	known_for_titles
0	nm0061671	Mary Ellen Bauder	NaN	tt0837562,tt2398241,tt0844471,tt0118553
1	nm0061865	Joseph Bauer	NaN	tt0896534,tt6791238,tt0287072,tt1682940
2	nm0062070	Bruce Baum	NaN	tt1470654,tt0363631,tt0104030,tt0102898
3	nm0062195	Axel Baumann	NaN	tt0114371,tt2004304,tt1618448,tt1224387
4	nm0062798	Pete Baxter	NaN	tt0452644,tt0452692,tt3458030,tt2178256
...	...	...	...	...
606643	nm9990381	Susan Grobes	NaN	NaN
606644	nm9990690	Joo Yeon So	NaN	tt9090932,tt8737130
606645	nm9991320	Madeline Smith	NaN	tt8734436,tt9615610
606646	nm9991786	Michelle Modigliani	NaN	NaN
606647	nm9993380	Pegasus Envoyé	NaN	tt8743182

606648 rows × 4 columns

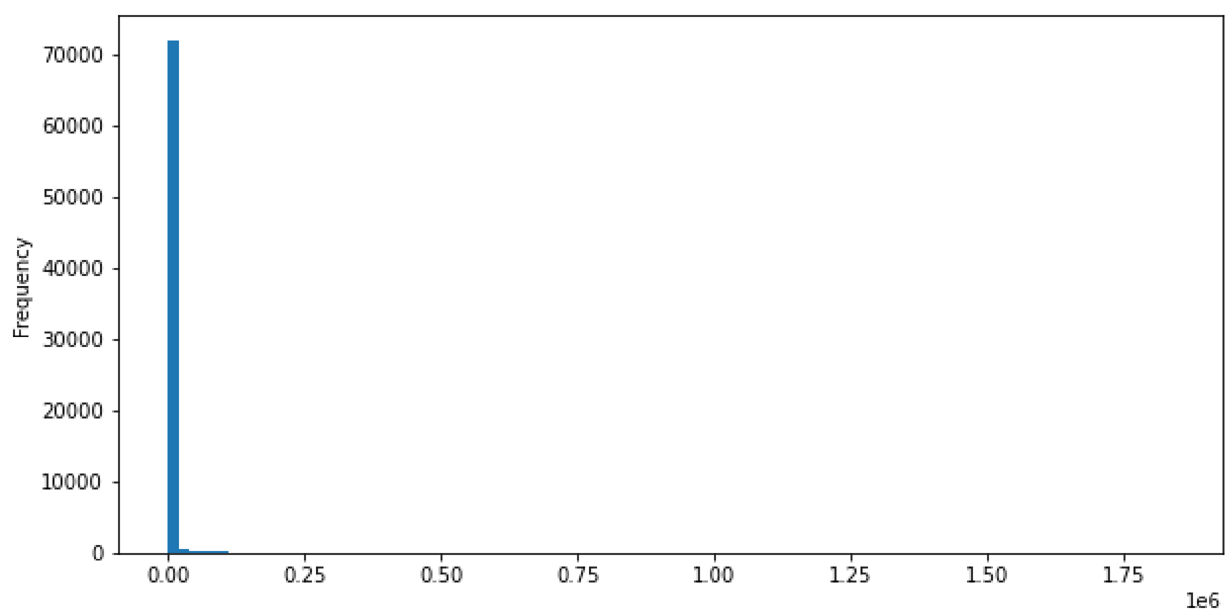
In [85]:
```python
imdb_title_ratings['numvotes'].isna().sum()
```
Out[85]:
```
0
```

In [86]:
```python
# Analyzing distribution of number of votes for each movie
imdb_title_ratings['numvotes'].median()
```
Out[86]:
```
49.0
```

In [87]:
```python
imdb_title_ratings['numvotes'].max()
```
Out[87]:
```
1841066
```

In [88]:
```python
imdb_title_ratings.numvotes.plot(kind='hist', bins=100, figsize=(10, 5))

percentile = .5
while percentile <= 1:
    print(
        f"{percentile:.2f} percentile: {int(imdb_title_ratings.numvotes.qua
    )
    percentile += .05
```
```
0.50 percentile: 49 number of votes
0.55 percentile: 67 number of votes
0.60 percentile: 92 number of votes
0.65 percentile: 130 number of votes
0.70 percentile: 190 number of votes
0.75 percentile: 282 number of votes
0.80 percentile: 441 number of votes
0.85 percentile: 767 number of votes
0.90 percentile: 1587 number of votes
0.95 percentile: 5598 number of votes
```



This is a unique situation because the kind of movie Microsoft wants to release is an outlier when compared to all movies in the IMDB database. What we must now do is make the most rated movies (outliers in this dataset with many numbers of votes) the normal.

The reasoning here is in the world of entertainment, there are countless movies a typical person would have never watched nor ever heard of. A normal movie goer would likely only know movies from the pool of the top echelon of user-rated films. Let's parse out the distribution at the top:

In [89]:
```
percentile = .95
while percentile <= 1:
    print(
        f"{percentile:.3f} percentile: {int(imdb_title_ratings.numvotes.qua
    )
    percentile += .005
```

```
0.950 percentile: 5598 number of votes
0.955 percentile: 6781 number of votes
0.960 percentile: 8295 number of votes
0.965 percentile: 10507 number of votes
0.970 percentile: 14247 number of votes
0.975 percentile: 19384 number of votes
0.980 percentile: 30367 number of votes
0.985 percentile: 46755 number of votes
0.990 percentile: 83518 number of votes
0.995 percentile: 167959 number of votes
1.000 percentile: 1841066 number of votes
```

Thus, we will select our personnel from movies in the top 1%. Let's see how many movies will be in consideration:

In [90]:
```
len(imdb_title_ratings[imdb_title_ratings['numvotes'] > 83518])
```

Out[90]:
```
739
```

In [91]:
```
# Dropping movies less than 83,518 votes to weed out less popular movies
drop_fewer_votes = []
for row in imdb_title_ratings.index:
    if imdb_title_ratings['numvotes'][row] < 83518:
        drop_fewer_votes.append(row)
len(drop_fewer_votes)
```

Out[91]:
```
73117
```

In [92]:
```
imdb_title_ratings.drop(drop_fewer_votes, inplace=True)
len(imdb_title_ratings)
```

Out[92]:
```
739
```

In [93]:
```
imdb_title_principals
```

0	tt0111414	nm0246005	actor
1	tt0111414	nm0398271	director
2	tt0111414	nm3739909	producer
3	tt0323808	nm0059247	editor
4	tt0323808	nm3579312	actress
...	...	...	...
1028181	tt9692684	nm0186469	actor
1028182	tt9692684	nm4929530	self
1028183	tt9692684	nm10441594	director
1028184	tt9692684	nm6009913	writer
1028185	tt9692684	nm10441595	producer

1028186 rows × 3 columns

In [94]:
```
imdb_title_principals.duplicated(subset=['nconst']).sum()
```

Out[94]:
```
423640
```

In [95]:
```
imdb_title_principals.duplicated().sum()
```

Out[95]:
```
35
```

In [96]:
```
imdb_title_principals.drop_duplicates(inplace=True)
```

In [97]:
```
imdb_title_principals.isna().sum()
```

Out[97]:
```
tconst      0
nconst      0
category    0
dtype: int64
```

In [98]:
```
# getting a unique category list
unique_category_list = []
for category_details in imdb_title_principals['category']:
    unique_category_list.append(category_details)

unique_category_list = sorted(list(set(unique_category_list)))
unique_category_list
```

Out[98]:
```
['actor',
 'actress',
 'archive_footage',
 'archive_sound',
 'cinematographer',
 'composer',
 'director',
 'editor',
 'producer',
 'production_designer',
 'self',
 'writer']
```

In [99]:
```
imdb_principals_ratings = pd.merge(imdb_title_principals,
                                   imdb_title_ratings,
                                   how='inner',
                                   on='tconst')
imdb_principals_ratings
```

Out[99]:

	tconst	nconst	category	averagerating	numvotes
0	tt0475290	nm0005683	cinematographer	6.3	111422
1	tt0475290	nm0000982	actor	6.3	111422
2	tt0475290	nm0000123	actor	6.3	111422
3	tt0475290	nm2403277	actor	6.3	111422
4	tt0475290	nm0000146	actor	6.3	111422
...	...	...	...	...	...
7374	tt6628394	nm1206844	director	7.1	86318
7375	tt6628394	nm1436246	producer	7.1	86318
7376	tt6628394	nm0315974	composer	7.1	86318
7377	tt6628394	nm0568974	cinematographer	7.1	86318
7378	tt6628394	nm0489809	editor	7.1	86318

In [100]:
```
imdb_name_basics
```

0	nm0061671	Mary Ellen Bauder	NaN	tt0837562,tt2398241,tt0844471,tt0118553
1	nm0061865	Joseph Bauer	NaN	tt0896534,tt6791238,tt0287072,tt1682940
2	nm0062070	Bruce Baum	NaN	tt1470654,tt0363631,tt0104030,tt0102898
3	nm0062195	Axel Baumann	NaN	tt0114371,tt2004304,tt1618448,tt1224387
4	nm0062798	Pete Baxter	NaN	tt0452644,tt0452692,tt3458030,tt2178256
...	...	...	...	...
606643	nm9990381	Susan Grobes	NaN	NaN
606644	nm9990690	Joo Yeon So	NaN	tt9090932,tt8737130
606645	nm9991320	Madeline Smith	NaN	tt8734436,tt9615610
606646	nm9991786	Michelle Modigliani	NaN	NaN
606647	nm9993380	Pegasus Envoyé	NaN	tt8743182

606648 rows × 4 columns

In [101]:
```
imdb_name_basics.duplicated().sum()
```

Out[101]: 0

In [102]:
```
imdb_name_basics.isna().sum()
```

Out[102]:
```
nconst                   0
primary_name             0
death_year          599865
known_for_titles     30204
dtype: int64
```

In [103]:
```python
imdb_names_roles = pd.merge(imdb_name_basics,
                            imdb_principals_ratings,
                            how='inner',
                            on='nconst')
imdb_names_roles
```

Out[103]:

	nconst	primary_name	death_year	known_for_titles	tconst	cate
0	nm0070822	Terry Benedict	NaN	tt0088247,tt2119532,tt0117280,tt0302427	tt2119532	proc
1	nm0093589	Matt Bomer	NaN	tt1637688,tt2268016,tt1915581,tt1684226	tt3799694	؛
2	nm0125336	Jez Butterworth	NaN	tt0977855,tt2379713,tt1355683,tt1631867	tt1355683	v
3	nm0125336	Jez Butterworth	NaN	tt0977855,tt2379713,tt1355683,tt1631867	tt1631867	v
4	nm0125336	Jez Butterworth	NaN	tt0977855,tt2379713,tt1355683,tt1631867	tt2379713	v
...	...	...	...	...	...	
7374	nm7222287	Christian Stevens	NaN	tt0918940	tt0918940	؛
7375	nm7448575	C.H. Vijay Kumar	NaN	tt1281841,tt0944185,tt2631186,tt4849438	tt2631186	v
7376	nm7887725	Fionn Whitehead	NaN	tt9664108,tt9495224,tt6040662,tt5013056	tt5013056	؛
7377	nm7887725	Fionn Whitehead	NaN	tt9664108,tt9495224,tt6040662,tt5013056	tt9495224	؛
7378	nm8075925	Millicent Simmonds	NaN	tt7966868,tt6644200,tt5208216,tt5195114	tt6644200	ac

7379 rows × 8 columns

In [104]:
```python
# creating and populating category columns
for category in unique_category_list:
    imdb_names_roles[category] = imdb_names_roles['category'].apply(
        lambda x: category in x).astype('int')
imdb_names_roles
```

Out[104]:

	nconst	primary_name	death_year	known_for_titles	tconst	cate
0	nm0070822	Terry Benedict	NaN	tt0088247,tt2119532,tt0117280,tt0302427	tt2119532	proc
1	nm0093589	Matt Bomer	NaN	tt1637688,tt2268016,tt1915581,tt1684226	tt3799694	a
2	nm0125336	Jez Butterworth	NaN	tt0977855,tt2379713,tt1355683,tt1631867	tt1355683	v
3	nm0125336	Jez Butterworth	NaN	tt0977855,tt2379713,tt1355683,tt1631867	tt1631867	v
4	nm0125336	Jez Butterworth	NaN	tt0977855,tt2379713,tt1355683,tt1631867	tt2379713	v
...	...	...	...	...	...	
7374	nm7222287	Christian Stevens	NaN	tt0918940	tt0918940	a
7375	nm7448575	C.H. Vijay Kumar	NaN	tt1281841,tt0944185,tt2631186,tt4849438	tt2631186	v
7376	nm7887725	Fionn Whitehead	NaN	tt9664108,tt9495224,tt6040662,tt5013056	tt5013056	a
7377	nm7887725	Fionn Whitehead	NaN	tt9664108,tt9495224,tt6040662,tt5013056	tt9495224	a
7378	nm8075925	Millicent Simmonds	NaN	tt7966868,tt6644200,tt5208216,tt5195114	tt6644200	ac

7379 rows × 20 columns

In [105]:
```
# Dropping deceased people
imdb_names_roles[imdb_names_roles['death_year'].isna() == False]
```

Out[105]:

	nconst	primary_name	death_year	known_for_titles	tconst	cat
5	nm0132168	John W. Campbell Jr.	1971.0	tt0043238,tt0905372,tt0084787,tt0044121	tt0905372	
84	nm0000341	Michael Crichton	2008.0	tt0070909,tt0117998,tt0108757,tt0107290	tt0369610	
85	nm0000341	Michael Crichton	2008.0	tt0070909,tt0117998,tt0108757,tt0107290	tt4881806	
88	nm0000450	Philip Seymour Hoffman	2014.0	tt1560747,tt0379725,tt0472062,tt0918927	tt1210166	
89	nm0000450	Philip Seymour Hoffman	2014.0	tt1560747,tt0379725,tt0472062,tt0918927	tt1560747	
...	...	...	...	...	...	
7111	nm5293055	Jeffrey Zaslow	2012.0	tt0123338,tt3263904	tt3263904	
7194	nm5410196	Tony Mendez	2019.0	tt5177262,tt4478356,tt3142234,tt1024648	tt1024648	

In [152]:
```
pd.isna(imdb_names_roles['death_year'][5])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2894             try:
-> 2895                 return self._engine.get_loc(casted_key)
   2896             except KeyError as err:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()
```

In [107]:
```
pd.isna(imdb_names_roles['death_year'][6])
```

Out[107]: True

In [108]:
```
drop_deceased_rows = []
for row in imdb_names_roles.index:
    if pd.isna(imdb_names_roles['death_year'][row]) == False:
        drop_deceased_rows.append(row)
len(drop_deceased_rows)
```

Out[108]: 221

In [109]:
```
imdb_names_roles.drop(drop_deceased_rows, inplace=True)
len(imdb_names_roles)
```

Out[109]: 7158

Setting Parameters

Our goal is to discover what movie personnel are associated with movies that have been highly rated (on a scale from 0-10) on the IMDB website. Here is the strategy we will use:

1. If a movie worker is not known for 3 or more movies, they will not be considered
2. Of the movies the person is known for, we will calculate the average IMDB rating and rank the top 30 by role. For example: top 30 actors, top 30 directors, etc.

Keep in mind, we have already narrowed down the movies in the top 1% for number of user votes. Given these parameters, we will surely find the most influential people in the movie business!

In [112]:
```
# Dropping null values in col 'known_for_titles'
imdb_names_roles['known_for_titles'].isna().sum()
```

Out[112]: 3

In [113]:
```
drop_known_titles = []
for row in imdb_names_roles.index:
    if pd.isna(imdb_names_roles['known_for_titles'][row]):
        drop_known_titles.append(row)
len(drop_known_titles)
```

Out[113]: 3

In [114]:
```
imdb_names_roles.drop(drop_known_titles, inplace=True)
len(imdb_names_roles)
```

Out[114]: 7155

In [115]:
```
# Creating new col for number of known titles
known_titles_len = []
for title_details in imdb_names_roles['known_for_titles']:
    title_list = title_details.split(",")
    num_titles = len(title_list)
    known_titles_len.append(num_titles)
len(known_titles_len)
```

Out[115]: 7155

In [116]:
```
imdb_names_roles['known_titles_len'] = known_titles_len
```

In [117]:
```
drop_less_than_three = []
for row in imdb_names_roles.index:
    if imdb_names_roles['known_titles_len'][row] < 4:
        drop_less_than_three.append(row)
len(drop_less_than_three)
```

Out[117]:
```
219
```

In [118]:
```
imdb_names_roles.drop(drop_less_than_three, inplace=True)
len(imdb_names_roles)
```

Out[118]:
```
6936
```

In [119]:
```
unique_category_list
```

Out[119]:
```
['actor',
 'actress',
 'archive_footage',
 'archive_sound',
 'cinematographer',
 'composer',
 'director',
 'editor',
 'producer',
 'production_designer',
 'self',
 'writer']
```

In [120]:
```
imdb_names_roles.head(20)
```

Out[120]:

	nconst	primary_name	death_year	known_for_titles	tconst	categ
0	nm0070822	Terry Benedict	NaN	tt0088247,tt2119532,tt0117280,tt0302427	tt2119532	produ
1	nm0093589	Matt Bomer	NaN	tt1637688,tt2268016,tt1915581,tt1684226	tt3799694	a
2	nm0125336	Jez Butterworth	NaN	tt0977855,tt2379713,tt1355683,tt1631867	tt1355683	w
3	nm0125336	Jez Butterworth	NaN	tt0977855,tt2379713,tt1355683,tt1631867	tt1631867	w
4	nm0125336	Jez Butterworth	NaN	tt0977855,tt2379713,tt1355683,tt1631867	tt2379713	w
6	nm0157787	Stephen Chin	NaN	tt1619805,tt2005151,tt0119237,tt0127722	tt2005151	w
7	nm0161834	Derek Cianfrance	NaN	tt0176573,tt1120985,tt1817273,tt2547584	tt1817273	dire
8	nm0161834	Derek Cianfrance	NaN	tt0176573,tt1120985,tt1817273,tt2547584	tt1120985	dire

Now that we have personnel from generally well known movies, we will delete duplicates and remove the rows we don't plan to use:

In [121]:
```python
imdb_names_roles.drop_duplicates(['nconst'], inplace=True)
```

In [122]:
```python
imdb_names_roles.drop(
    labels=['death_year', 'tconst', 'averagerating', 'numvotes'],
    axis=1,
    inplace=True)
imdb_names_roles
```

Out[122]:

	nconst	primary_name	known_for_titles	category	actor	actress
0	nm0070822	Terry Benedict	tt0088247,tt2119532,tt0117280,tt0302427	producer	0	0
1	nm0093589	Matt Bomer	tt1637688,tt2268016,tt1915581,tt1684226	actor	1	0
2	nm0125336	Jez Butterworth	tt0977855,tt2379713,tt1355683,tt1631867	writer	0	0
6	nm0157787	Stephen Chin	tt1619805,tt2005151,tt0119237,tt0127722	writer	0	0
7	nm0161834	Derek Cianfrance	tt0176573,tt1120985,tt1817273,tt2547584	director	0	0
...	...	...	...	...	...	...
7368	nm8314228	Cailee Spaeny	tt2557478,tt4669788,tt6628394,tt6266538	actress	0	1
7373	nm9545038	Julien Rey	tt2938956,tt2872732,tt2404311,tt2239822	editor	0	0
7375	nm7448575	C.H. Vijay Kumar	tt1281841,tt0944185,tt2631186,tt4849438	writer	0	0

It is necessary to have access to the original dataset of IMDB movie ratings. This is because movies the personnel are known for may not be in the top 1% for number of votes, so we want full access to the data we lost when merging to calculate a person's average rating.

Another important note is we do not have access to all the movie ratings, only those from 2010-2018. If a person is known for a movie released before 2010, we will not be able to include it in their average. We will only consider personnel who have at least 3 movies to consider towards their average. This will give us movie personnel who have made the most recent impact, giving Microsoft the best chance of success in making a highly rated movie.

In [123]:
```python
# Importing original ratings dataset
ratings_catalog = pd.read_csv('zippedData/imdb.title.ratings.csv.gz')
```

In [125]:
```python
# Preparing known_for_title column for iteration by splitting into individu
new_titles_col = []
for titles in imdb_names_roles['known_for_titles']:
    new_titles_col.append(titles.split(","))
len(new_titles_col)
```

Out[125]: 3589

In [126]:
```
new_titles_col[:5]
```

Out[126]:
```
[['tt0088247', 'tt2119532', 'tt0117280', 'tt0302427'],
 ['tt1637688', 'tt2268016', 'tt1915581', 'tt1684226'],
 ['tt0977855', 'tt2379713', 'tt1355683', 'tt1631867'],
 ['tt1619805', 'tt2005151', 'tt0119237', 'tt0127722'],
 ['tt0176573', 'tt1120985', 'tt1817273', 'tt2547584']]
```

In [127]:
```
imdb_names_roles['known_for_titles'] = new_titles_col
```

In [128]:
```
imdb_names_roles['known_for_titles']
```

Out[128]:
```
0       [tt0088247, tt2119532, tt0117280, tt0302427]
1       [tt1637688, tt2268016, tt1915581, tt1684226]
2       [tt0977855, tt2379713, tt1355683, tt1631867]
6       [tt1619805, tt2005151, tt0119237, tt0127722]
7       [tt0176573, tt1120985, tt1817273, tt2547584]
                            ...
7368    [tt2557478, tt4669788, tt6628394, tt6266538]
7373    [tt2938956, tt2872732, tt2404311, tt2239822]
7375    [tt1281841, tt0944185, tt2631186, tt4849438]
7376    [tt9664108, tt9495224, tt6040662, tt5013056]
7378    [tt7966868, tt6644200, tt5208216, tt5195114]
Name: known_for_titles, Length: 3589, dtype: object
```

In [129]:
```
# The original ratings data will be our reference
ratings_catalog
```

Out[129]:

	tconst	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21
...	...	...	...
73851	tt9805820	8.1	25
73852	tt9844256	7.5	24
73853	tt9851050	4.7	14
73854	tt9886934	7.0	5
73855	tt9894098	6.3	128

In [130]:
```
# Referencing the rating for 1 movie
imdb_names_roles['known_for_titles'][0]
```

Out[130]:
```
['tt0088247', 'tt2119532', 'tt0117280', 'tt0302427']
```

In [131]:
```python
ratings_catalog[ratings_catalog['tconst'] == 'tt0837562']['tconst'].iloc[0]
```

Out[131]: 'tt0837562'

In [132]:
```python
ratings_catalog[ratings_catalog['tconst'] == 'tt0837562']['averagerating'].
```

Out[132]: 7.1

In [133]:
```python
# Averaging the rating of each person's known movie titles
mean_known_rating = []
for titles in imdb_names_roles['known_for_titles']:
    ratings_list = []
    for title in titles:
        try:
            title == ratings_catalog[ratings_catalog['tconst'] ==
                                     title]['tconst'].iloc[0]
            ratings_list.append(ratings_catalog[
                ratings_catalog['tconst'] == title]['averagerating'].iloc[0
        except:
            pass
    if len(ratings_list) < 3:  # Excluding any people who cannot contribute
        mean_known_rating.append(0)
    else:
        mean_known_rating.append(sum(ratings_list) / len(ratings_list))
mean_known_rating[:10]
```

Out[133]: [0, 6.133333333333333, 7.1, 0, 7.3, 5.675, 6.966666666666666, 0, 0, 0]

In [134]:
```python
len(mean_known_rating)
```

Out[134]: 3589

In [135]:
```python
rounded = []
for value in mean_known_rating:
    rounded.append(round(value, 2))
mean_known_rating = rounded
```

In [136]:
```
# Creating new column for average rating
imdb_names_roles['mean_known_rating'] = mean_known_rating
imdb_names_roles
```

Out[136]:

	nconst	primary_name	known_for_titles	category	actor	actress	archive_footage	archiv
0	nm0070822	Terry Benedict	[tt0088247, tt2119532, tt0117280, tt0302427]	producer	0	0	0	
1	nm0093589	Matt Bomer	[tt1637688, tt2268016, tt1915581, tt1684226]	actor	1	0	0	
2	nm0125336	Jez Butterworth	[tt0977855, tt2379713, tt1355683, tt1631867]	writer	0	0	0	
6	nm0157787	Stephen Chin	[tt1619805, tt2005151, tt0119237, tt0127722]	writer	0	0	0	
7	nm0161834	Derek Cianfrance	[tt0176573, tt1120985, tt1817273, tt2547584]	director	0	0	0	
...	...	...	...	...	...	...	...	
7368	nm8314228	Cailee Spaeny	[tt2557478, tt4669788, tt6628394, tt6266538]	actress	0	1	0	
7373	nm9545038	Julien Rey	[tt2938956, tt2872732, tt2404311, tt2239822]	editor	0	0	0	
7375	nm7448575	C.H. Vijay Kumar	[tt1281841, tt0944185, tt2631186, tt4849438]	writer	0	0	0	
7376	nm7887725	Fionn Whitehead	[tt9664108, tt9495224, tt6040662, tt5013056]	actor	1	0	0	
7378	nm8075925	Millicent Simmonds	[tt7966868, tt6644200, tt5208216, tt5195114]	actress	0	1	0	

3589 rows × 18 columns

In [137]:
```python
# Def a function to return top 30 ranked by movie role
def category_top_rating(category, df):
    # Filtering into movie roles
    category_drop_rows = []
    for row in df.index:
        if df[category.lower()][row] == 0:
            category_drop_rows.append(row)

    category_cleaned = df.drop(category_drop_rows)

    # Sorting to view highest ratings
    sort_category = category_cleaned[['primary_name', 'mean_known_rating'
                                      ]].sort_values(by=['mean_known_rating
                                                     ascending=False)
    indexed = sort_category.reset_index().drop(columns='index')
    indexed.reset_index(inplace=True)

    # Creating rank column
    rank = []
    for value in indexed['index']:
        rank.append(value + 1)
    indexed['index'] = rank
    indexed.columns = ['rank', 'primary_name', 'mean_known_rating']
    top_30 = indexed.head(30)
    return top_30
```

Q3 C. Data Modeling

Here we will see the top 30 most influential movie personnel organized by their role:

Top 30 Actors

In [138]:
```
actor_top_30 = category_top_rating('actor', imdb_names_roles)
actor_top_30
```

Out[138]:

	rank	primary_name	mean_known_rating
0	1	Sunny Pawar	8.20
1	2	Rana Daggubati	8.12
2	3	Tom Hardy	8.10
3	4	Murat Arkin	8.00
4	5	Sathyaraj	8.00
5	6	Bruce Dern	7.97
6	7	Yayan Ruhian	7.87
7	8	Prabhas	7.87
8	9	Mads Mikkelsen	7.87
9	10	Tom Holland	7.85
10	11	Adrien Brody	7.83
11	12	Sebastian Stan	7.82
12	13	Lucas Hedges	7.80
13	14	Ben Affleck	7.80
14	15	Timothée Chalamet	7.80
15	16	Jeremy Renner	7.80
16	17	Chadwick Boseman	7.78
17	18	Ben Mendelsohn	7.75
18	19	Ryan Gosling	7.73
19	20	Benedict Wong	7.72
20	21	Rami Malek	7.70
21	22	Ken Stott	7.70
22	23	Jon Bernthal	7.68
23	24	Dave Bautista	7.68
24	25	Michael Keaton	7.67
25	26	Nicholas Hoult	7.67
26	27	Irrfan Khan	7.67
27	28	Chris Evans	7.65
28	29	Chris Pratt	7.63
29	30	Joseph Gordon-Levitt	7.62

In [139]:
```
#actor_top_30.to_excel('zippedData/Actors.xlsx')
```

Top 30 Actresses

```
In [140]: actress_top_30 = category_top_rating('actress', imdb_names_roles)
          actress_top_30
```

Out[140]:

	rank	primary_name	mean_known_rating
0	1	Tabu	8.20
1	2	Danai Gurira	8.20
2	3	Sanya Malhotra	7.95
3	4	Marion Cotillard	7.87
4	5	Rachel House	7.80
5	6	Emma Stone	7.72
6	7	Lupita Nyong'o	7.70
7	8	Farrah Mackenzie	7.63
8	9	Melissa Benoist	7.63
9	10	Elizabeth Olsen	7.62
10	11	Brie Larson	7.58
11	12	Jessica Chastain	7.58
12	13	Ana Wagener	7.57
13	14	Anne Hathaway	7.57
14	15	Karin Konoval	7.57
15	16	Sonoya Mizuno	7.53
16	17	Mone Kamishiraishi	7.52
17	18	Lucy Boynton	7.50
18	19	Sally Hawkins	7.50
19	20	Scarlett Johansson	7.47
20	21	Rooney Mara	7.45
21	22	Jennifer Lawrence	7.45
22	23	Olivia Colman	7.43
23	24	Viola Davis	7.43
24	25	Octavia Spencer	7.42
25	26	Evangeline Lilly	7.40
26	27	Sarah Paulson	7.40
27	28	Tilda Swinton	7.38
28	29	Anushka Sharma	7.37
29	30	Mindy Kaling	7.37

```
In [141]: #actress_top_30.to_excel('zippedData/Actress.xlsx')
```

Top 30 Directors

In [142]:
```
director_top_30 = category_top_rating('director', imdb_names_roles)
director_top_30
```

Out[142]:

	rank	primary_name	mean_known_rating
0	1	Anthony Russo	8.23
1	2	Joe Russo	8.22
2	3	S.S. Rajamouli	8.07
3	4	Alper Caglar	7.98
4	5	Asghar Farhadi	7.97
5	6	Denis Villeneuve	7.95
6	7	Dean DeBlois	7.83
7	8	Phil Lord	7.77
8	9	Christopher Miller	7.77
9	10	Taika Waititi	7.75
10	11	Thomas Vinterberg	7.67
11	12	Matthew Vaughn	7.67
12	13	Chris Williams	7.63
13	14	Bob Persichetti	7.60
14	15	Rich Moore	7.60
15	16	Alessandro Carloni	7.60
16	17	Nitesh Tiwari	7.58
17	18	Tom Hooper	7.57
18	19	Morten Tyldum	7.53
19	20	Dan Scanlon	7.53
20	21	Jean-Marc Vallée	7.50
21	22	Chad Stahelski	7.47
22	23	Oriol Paulo	7.47
23	24	Steve McQueen	7.43
24	25	Matt Reeves	7.40
25	26	Ryan Coogler	7.40
26	27	James Wan	7.37
27	28	David Yates	7.37
28	29	Edgar Wright	7.37
29	30	Drew Goddard	7.37

```
In [143]: #director_top_30.to_excel('zippedData/Director.xlsx')
```

Top 30 Writers

```
In [144]: writer_top_30 = category_top_rating('writer', imdb_names_roles)
writer_top_30
```

18	19	Nicolás Giacobone	7.55
19	20	Herbert Kretzmer	7.53
20	21	Wajdi Mouawad	7.53
21	22	Stephen McFeely	7.50
22	23	Larry Lieber	7.47
23	24	Adrian Molina	7.47
24	25	Tobias Lindholm	7.43
25	26	Rick Jaffa	7.42
26	27	Amanda Silver	7.42
27	28	Valérie Beaugrand-Champagne	7.40
28	29	Michael H. Weber	7.40

```
In [145]: #writer_top_30.to_excel('zippedData/Writer.xlsx')
```

Top 30 Cinematographers

In [146]:
```
cinematographer_top_30 = category_top_rating('cinematographer', imdb_names_
cinematographer_top_30
```

Out[146]:

	rank	primary_name	mean_known_rating
0	1	Hoyte Van Hoytema	8.17
1	2	Danny Cohen	7.93
2	3	Emmanuel Lubezki	7.80
3	4	Roger Deakins	7.80
4	5	Trent Opaloch	7.73
5	6	Ben Davis	7.70
6	7	Yves Bélanger	7.62
7	8	Andrew Dunn	7.53
8	9	Rob Hardy	7.47
9	10	Gil Zimmerman	7.43
10	11	Jody Lee Lipes	7.40
11	12	Sofian El Fani	7.40
12	13	Greig Fraser	7.35
13	14	Eduard Grau	7.30
14	15	Benoît Delhomme	7.30
15	16	Charlotte Bruus Christensen	7.28
16	17	Erik Wilson	7.23
17	18	Matthew Jensen	7.23
18	19	John Guleserian	7.22
19	20	Terry Stacey	7.13
20	21	Guillaume Schiffman	7.13
21	22	Ken Seng	7.08
22	23	Sam Levy	7.03
23	24	Thimios Bakatakis	7.00
24	25	Thomas Townend	7.00
25	26	Larry Smith	7.00
26	27	Roman Vasyanov	6.93
27	28	Pawel Pogorzelski	6.93
28	29	Salvatore Totino	6.93
29	30	Manuel Alberto Claro	6.93

In [147]:
```
#cinematographer_top_30.to_excel('zippedData/Cinematographer.xlsx')
```

Top 30 Composers

In [148]:
```
composer_top_30 = category_top_rating('composer', imdb_names_roles)
composer_top_30
```

Out[148]:

	rank	primary_name	mean_known_rating
0	1	Justin Hurwitz	7.97
1	2	Lorne Balfe	7.85
2	3	Tyler Bates	7.83
3	4	Benjamin Wallfisch	7.78
4	5	Alexandre Desplat	7.77
5	6	Kristen Anderson-Lopez	7.70
6	7	The Newton Brothers	7.70
7	8	Carter Burwell	7.67
8	9	Jon Thor Birgisson	7.67
9	10	Mark Mothersbaugh	7.63
10	11	Steven Price	7.63

In [149]:
```
#composer_top_30.to_excel('zippedData/Composer.xlsx')
```

Q3 D. Evaluation

Recommendation:

My business recommendation would be to hire these movie personnel as users of IMDB love their movies.

Acknowledging Limitations

Here we can see how the limitations of the data have a profound effect on our final result. The outcome would have been very different if we had access to the ratings of movies before 2010, and personnel who have had a long-term impact on the movie business would have emerged more prominently. However, only taking data from post 2010 ensures that we consider who has made the most *recent* influence.

Conclusions

To summarize:

1. Make movies in the following genres: Sci-Fi, Adventure, Animation, Fantasy, and Action. To make the most revenue, focus on Sci-Fi and Adventure.

2. Make movies in the recommended ~30 minute windows to maximuze revenue. Make longer movies.
3. Hire personnel associated with the top ratings as voted on by users of IMDb

A few caveats:

- Analyzing data that is more comprehensive (this study was limited to the years 2010-2018) can give additional potential insight
- To be critical, the movie time recommendation seems a bit contrived. Obviously directors and editors should try to tell the best story they can while being efficient and/or stylistic with their time. The reason longer movies seem to do so well is they are commonly associated with intellectual property of familiar franchises, and audiences are willing to sit through longer movies that have a reputation that proceeds them.

Good luck and godspeed!

In [ ]:
```

```