Final Project Submission

Please fill out:

- Student name: Linda Hutsal
- Student pace: self paced / part time / full time: self paced
- Scheduled project review date/time:
- Instructor name: Morgan Jones
- Blog post URL:

Import and Settings

First, I'll import the libraries needed and call the appropriate settings

In [1]:
```
import pandas as pd #Alias pandas as pd
import matplotlib.pyplot as plt #Alias matplotlib as plt
import numpy as np #Alias numpy as np
import seaborn as sns #Alias seaborn as sns
from glob import glob
import os

#Magic function to allow plot outputs to appear and be stored in the n
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

pd.options.display.float_format = '{:,}'.format
```

Importing the data and initial viewing

Next is importing the various data sets using the pandas dataframes:

1. Box Office Mojo, which is in a csv file
2. IMDB, which is in a sqlite database file

For each dataset, we'll look at:

1. a sample of the data
2. the info for the dataset, including data types and count of non-null values
3. the describe table for the dataset, which includes statistical summary values

In [2]:
```
#Import and view Box Office Magic data
df_bom = pd.read_csv("zippedData/bom.movie_gross.csv.gz")
print(df_bom.shape)
df_bom.head()
```

```
(3387, 5)
```

Out[2]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415,000,000.0	652000000	2010
1	Alice in Wonderland (2010)	BV	334,200,000.0	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296,000,000.0	664300000	2010
3	Inception	WB	292,600,000.0	535700000	2010
4	Shrek Forever After	P/DW	238,700,000.0	513900000	2010

Now let's look at the data types and number of non-null values

In [3]:
```
df_bom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   title           3387 non-null   object
 1   studio          3382 non-null   object
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object
 4   year            3387 non-null   int64
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

I'll convert the foreign_gross to a float64, in order to get the dollar amounts where there is a value

In [4]:
```python
df_bom['foreign_gross'] = pd.to_numeric(df_bom["foreign_gross"], error
df_bom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   title           3387 non-null   object
 1   studio          3382 non-null   object
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2032 non-null   float64
 4   year            3387 non-null   int64
dtypes: float64(2), int64(1), object(2)
memory usage: 132.4+ KB
```

Now the foreign_gross field is showing as float64, but there are still a lot of Null values compared to the other fields in the table. Let's investigate that further when we get to that step.

In [5]:
```python
df_bom.describe()
```

Out[5]:

	domestic_gross	foreign_gross	year
count	3,359.0	2,032.0	3,387.0
mean	28,745,845.06698422	75,057,041.62549213	2,013.9580749926188
std	66,982,498.23736456	137,529,351.2001863	2.4781410973889657
min	100.0	600.0	2,010.0
25%	120,000.0	3,775,000.0	2,012.0
50%	1,400,000.0	18,900,000.0	2,014.0
75%	27,900,000.0	75,050,000.0	2,016.0
max	936,700,000.0	960,500,000.0	2,018.0

Useful information around the Box Office Mojo dataset: There's 3,359 records, with the films ranging in year from 2010 to 2018, on average from being around 2014. The average domestic gross is $28.8MM and the average foreign gross is $75MM.

There are some very low grossing films, with the smallest beign $100 and $600 for domestic and foreign, respectively. On the opposite spectrum, there are some extremely big numbers for the largest grossing films. They are close to a billion dollars, at $936MM for domestic and $960MM for foreign.

I imported the Rotten Tomatoes dataset to take a look at the information, but ultimately did not use it as I tried to keep the data analysis simple.

In [6]:
```
#Import and view Rotten Tomatoes Movie Info`
df_rt_mi = pd.read_csv("zippedData/rt.movie_info.tsv.gz", sep='\t')
df_rt_mi.head()
```

Out[6]:

	id	synopsis	rating	genre	director	writer	theater_date
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo	Aug 17, 2012
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton	Dec 9, 1994
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	NaN

In [7]:
```
#Import and view Rotten Tomatoes Reviews
#df_rt_rev = pd.read_csv("zippedData/rt.reviews.tsv.gz", sep='\t')
#df_rt_rev.head()

#issue loading the file
#opened in Numbers and showed the encoding is Western (Windows Latin 1
```

In [8]:
```python
#Import and view Rotten Tomatoes Reviews with Latin-1 encoding
df_rt_rev = pd.read_csv("zippedData/rt.reviews.tsv.gz", sep='\t', encc
df_rt_rev.head()
```

Out[8]:

	id	review	rating	fresh	critic	top_critic	publisher	date
0	3	A distinctly gallows take on contemporary fina...	3/5	fresh	PJ Nabarro	0	Patrick Nabarro	November 10, 2018
1	3	It's an allegory in search of a meaning that n...	NaN	rotten	Annalee Newitz	0	io9.com	May 23, 2018
2	3	... life lived in a bubble in financial dealin...	NaN	fresh	Sean Axmaker	0	Stream on Demand	January 4, 2018
3	3	Continuing along a line introduced in last yea...	NaN	fresh	Daniel Kasman	0	MUBI	November 16, 2017
4	3	... a perverse twist on neorealism...	NaN	fresh	NaN	0	Cinema Scope	October 12, 2017

I imported The Movie DB dataset to take a look at the information, but ultimately did not use it as I tried to keep the data analysis simple.

In [9]:
```python
#Import and view The Movie DB
df_movie_db = pd.read_csv("zippedData/tmdb.movies.csv.gz")
df_movie_db.head()
```

Out[9]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	t
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	H Pc and Dea Hallc Pa
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	Hov T Y Dra
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron M
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	S
4	4	[28, 878, 12]	27205	en	Inception	27.92	2010-07-16	Incep

Next I imported The Numbers dataset which I will be using as it has the production budget data.

In [10]:
```python
#Import and view The Numbers
df_the_numbers = pd.read_csv("zippedData/tn.movie_budgets.csv.gz")
df_the_numbers.head()
```

Out[10]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747

In [11]:
```python
df_the_numbers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   id                 5782 non-null   int64 
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   object
 4   domestic_gross     5782 non-null   object
 5   worldwide_gross    5782 non-null   object
dtypes: int64(1), object(5)
memory usage: 271.2+ KB
```

I'll need to update the production_budget, domestic_gross, and worldwide_gross to numbers instead of object

In [12]:
```
# I first tried converting the fields to numerics, but it didn't work
#df_the_numbers['production_budget'] = pd.to_numeric(df_the_numbers["p
#df_the_numbers['domestic_gross'] = pd.to_numeric(df_the_numbers["dome
#df_the_numbers['worldwide_gross'] = pd.to_numeric(df_the_numbers["wor


df_the_numbers['production_budget'] = df_the_numbers['production_budge
df_the_numbers['domestic_gross'] = df_the_numbers['domestic_gross'].st
df_the_numbers['worldwide_gross'] = df_the_numbers['worldwide_gross'].



df_the_numbers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   id                 5782 non-null   int64 
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   int64 
 4   domestic_gross     5782 non-null   int64 
 5   worldwide_gross    5782 non-null   int64 
dtypes: int64(4), object(2)
memory usage: 271.2+ KB
```

The number fields are now formatted correctly. Let's look at a sample of the data

In [13]:
```
df_the_numbers.head()
```

Out[13]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875
2	3	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350
3	4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747

This seems reasonable. Let's take a look at the statistical summary.

In [14]:
```
df_the_numbers.describe()
```

Out[14]:

	id	production_budget	domestic_gross	worldwide_gross
count	5,782.0	5,782.0	5,782.0	5,782.0
mean	50.37236250432376	31,587,757.0965064	41,873,326.867001034	91,487,460.90643376
std	28.821076273431096	41,812,076.82694309	68,240,597.35690415	174,719,968.77890477
min	1.0	1,100.0	0.0	0.0
25%	25.0	5,000,000.0	1,429,534.5	4,125,414.75
50%	50.0	17,000,000.0	17,225,945.0	27,984,448.5
75%	75.0	40,000,000.0	52,348,661.5	97,645,836.5
max	100.0	425,000,000.0	936,662,225.0	2,776,345,279.0

Next let's look at the IMDB data, which will need to be imported via sql as it's in a database format. First, we'll look at the list of the tables.

In [15]:
```
#View IMDB data via sql
import sqlite3
conn = sqlite3.connect('zippedData/im.db')

pd.read_sql(""" SELECT name FROM sqlite_master WHERE type ='table'
        ;""", conn)
```

Out[15]:

	name
0	movie_basics
1	directors
2	known_for
3	movie_akas
4	movie_ratings
5	persons
6	principals
7	writers

First we'll look at the movie_basics table and then the movie_ratings.

In [16]:
```
df_movie_basics = pd.read_sql("""SELECT * FROM movie_basics;""", conn)
df_movie_basics.head()
```

Out[16]:

	movie_id	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

In [17]:
```
df_movie_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         146144 non-null  object 
 1   primary_title    146144 non-null  object 
 2   original_title   146123 non-null  object 
 3   start_year       146144 non-null  int64  
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
```

There are a lot more records here than in our box office data: 146,144 records. All the data looks to be in a useable format.

In [18]:
```
df_movie_basics.describe()
```

Out[18]:

	start_year	runtime_minutes
count	146,144.0	114,405.0
mean	2,014.6217976790015	86.18724706088021
std	2.7335829231921163	166.36059015397228
min	2,010.0	1.0
25%	2,012.0	70.0
50%	2,015.0	87.0
75%	2,017.0	99.0
max	2,115.0	51,420.0

The movies in the database range in years from 2010 to 2115.

In [19]:
```
df_movie_basics[df_movie_basics['start_year']>2023]
```

Out[19]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	ge
2948	tt10300396	Untitled Star Wars Film	Untitled Star Wars Film	2024	nan	N
2949	tt10300398	Untitled Star Wars Film	Untitled Star Wars Film	2026	nan	Fan
52213	tt3095356	Avatar 4	Avatar 4	2025	nan	Action,Adventure,Fan
89506	tt5174640	100 Years	100 Years	2115	nan	Dr
96592	tt5637536	Avatar 5	Avatar 5	2027	nan	Action,Adventure,Fan
105187	tt6149054	Fantastic Beasts and Where to Find Them 5	Fantastic Beasts and Where to Find Them 5	2024	nan	Adventure,Family,Fan

In [20]:
```
df_movie_ratings = pd.read_sql(""" SELECT * FROM movie_ratings; """, c
df_movie_ratings.head()
```

Out[20]:

	movie_id	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

It looks like there are a handful of movies in the database with some projected future start dates.

Now let's look at the movie_ratings table.

In [21]:
```
df_movie_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   movie_id       73856 non-null  object 
 1   averagerating  73856 non-null  float64
 2   numvotes       73856 non-null  int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 1.7+ MB
```

This data looks like it's formatted appropriately.

In [22]:
```
df_movie_ratings.describe()
```

Out[22]:

	averagerating	numvotes
count	73,856.0	73,856.0
mean	6.332728552859619	3,523.6621669194105
std	1.4749783548957056	30,294.022971107453
min	1.0	5.0
25%	5.5	14.0
50%	6.5	49.0
75%	7.4	282.0
max	10.0	1,841,066.0

There are 73,856 records here, so less than in the movies table which would make sense if there's some movies that haven't been released yet or potentially some films that haven't been reviewed. The lowest rating is a 1 and the highest is a 10, with the average being 6.3.

The film with the lowest number of votes has 5 and the highest has 1,841,066. Let's see which film that is once the data is joined.

Data Cleaning

I've now loaded all of the data sources, and while I'd love to use all of the information, in order to keep this focused and efficient with time, I will only use the four dataframes listed below:

1. Box Office Magic (domestic and foreign gross revenue)
2. The Numbers (production budget)
3. IMDB - movie basics (genres)
4. IMDB - movie ratings (ratings)

Data Cleaning Process

1. Check for duplicates in the dataframes I'm focusing on: df_bom df_the_numbers df_movie_basics df_movie_ratings
2. Look for placeholder values
3. Check for duplicates on title, which will be used to combine the datasets

In [23]:
```
print(df_bom.duplicated().any())
print(df_the_numbers.duplicated().any())
print(df_movie_basics.duplicated().any())
print(df_movie_ratings.duplicated().any())
```

```
False
False
False
False
```

There are no duplicate records in any of these dataframes.

In [24]:
```
df_bom.isin(['?', '#', 'NaN', 'null', 'N/A', '-', 0, 'nan']).sum()
```

Out[24]:
```
title             0
studio            0
domestic_gross    0
foreign_gross     0
year              0
dtype: int64
```

In [25]:
```
df_bom.isnull().sum()
```

Out[25]:
```
title                0
studio               5
domestic_gross      28
foreign_gross     1355
year                 0
dtype: int64
```

There are no placeholder values, but there are quite a few null values in the Box Office Magic data. Let's udpate the nulls to zeros for the gross revenue fields.

In [26]:
```
df_bom['domestic_gross'].fillna(0, inplace=True)
df_bom['foreign_gross'].fillna(0, inplace=True)
```

In [27]:
```
df_bom.isnull().sum()
```

Out[27]:
```
title             0
studio            5
domestic_gross    0
foreign_gross     0
year              0
dtype: int64
```

In [28]:
```
df_the_numbers.isin(['?', '#', 'NaN', 'null', 'N/A', '-', 0]).sum()
```

Out[28]:
```
id                    0
release_date          0
movie                 0
production_budget     0
domestic_gross      548
worldwide_gross     367
dtype: int64
```

There are some NA's from doing the type conversion in The Numbers data. Let's take a closer look.

In [29]:
```
df_the_numbers[df_the_numbers['domestic_gross'].isin(['?', '#', 'NaN',
```

Out[29]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
194	95	Dec 31, 2020	Moonfall	150000000	0	0
479	80	Dec 13, 2017	Bright	90000000	0	0
480	81	Dec 31, 2019	Army of the Dead	90000000	0	0
535	36	Feb 21, 2020	Call of the Wild	82000000	0	0
617	18	Dec 31, 2012	AstÃ©rix et ObÃ©lix: Au service de Sa MajestÃ©	77600000	0	60680125
...	...	...	...	...	...	...
5761	62	Dec 31, 2014	Stories of Our Lives	15000	0	0
5764	65	Dec 31, 2007	Tin Can Man	12000	0	0
5771	72	May 19, 2015	Family Motocross	10000	0	0
5777	78	Dec 31, 2018	Red 11	7000	0	0
5780	81	Sep 29, 2015	A Plague So Pleasant	1400	0	0

548 rows × 6 columns

It looks like there's movies in the data that are in production and have not yet been released, or potentially are foreign-only films, so this makes sense for the zero values. Let's look at just the other null values now.

In [30]:
```
df_the_numbers[df_the_numbers['domestic_gross'].isin(['?', '#', 'NaN',
```
Out[30]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross

In [31]:
```
df_the_numbers.isnull().sum()
```
Out[31]:
```
id                   0
release_date         0
movie                0
production_budget    0
domestic_gross       0
worldwide_gross      0
dtype: int64
```

There are no other null values, just the zeros so we can continue on.

In [32]:
```
df_movie_basics.isin(['?', '#', 'NaN', 'null', 'N/A', '-', 0]).sum()
```
Out[32]:
```
movie_id           0
primary_title      0
original_title     0
start_year         0
runtime_minutes    0
genres             0
dtype: int64
```

In [33]:
```
df_movie_basics.isnull().sum()
```
Out[33]:
```
movie_id               0
primary_title          0
original_title        21
start_year             0
runtime_minutes    31739
genres              5408
dtype: int64
```

In [34]:
```
df_movie_basics[df_movie_basics['runtime_minutes'].isnull()].head(10)
```

Out[34]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	nan	Comedy,Drama
6	tt0112502	Bigfoot	Bigfoot	2017	nan	Horror,Thriller
8	tt0139613	O Silêncio	O Silêncio	2012	nan	Documentary,History
16	tt0187902	How Huang Fei-hong Rescued the Orphan from the...	How Huang Fei-hong Rescued the Orphan from the...	2011	nan	None
21	tt0250404	Godfather	Godfather	2012	nan	Crime,Drama
26	tt0263814	On kadin	On kadin	2019	nan	Drama
31	tt0285423	Abolição	Abolição	2019	nan	Documentary
33	tt0293429	Mortal Kombat	Mortal Kombat	2021	nan	Action,Adventure,Fantasy
34	tt0297400	Snowblind	Snowblind	2015	nan	Crime,Drama
44	tt0330811	Regret Not Speaking	Regret Not Speaking	2011	nan	None

There are no placeholder values in the Movie Basics data, but there are some null values for the runtime minutes. It seems like these are much smaller films. For now we'll keep these rows but evaluate later when the box office revenue is added in.

In [35]:
```
df_movie_ratings.isin(['?', '#', 'NaN', 'null', 'N/A', '-', 0]).sum()
```

Out[35]:
```
movie_id         0
averagerating    0
numvotes         0
dtype: int64
```

In [36]:
```
df_movie_ratings.isnull().sum()
```

Out[36]:
```
movie_id         0
averagerating    0
numvotes         0
dtype: int64
```

There are no placeholder values in the IMDB movie ratings data

In [37]:
```
df_bom['title'].duplicated().sum()
```

Out[37]:
```
1
```

There's one duplicate value. Let's look at it below and see if it has a material domestic or foreign gross.

In [38]:
```
df_bom[df_bom.duplicated('title', keep=False) == True]
```

Out[38]:

	title	studio	domestic_gross	foreign_gross	year
317	Bluebeard	Strand	33,500.0	5,200.0	2010
3045	Bluebeard	WGUSA	43,100.0	0.0	2017

It looks like the movie is pretty small and not material.

Now let's look at the primary title field in df_movie_basics:

In [39]:
```
df_the_numbers['movie'].duplicated().sum()
```

Out[39]: 84

There's 84 movie titles that are duplicated. Let's take a look at a sample.

In [40]:
```
df_the_numbers[df_the_numbers.duplicated('movie', keep=False) == True]
```

Out[40]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
26	27	May 4, 2012	The Avengers	225000000	623279547	1517935897
38	39	May 14, 2010	Robin Hood	210000000	105487148	322459006
39	40	Dec 14, 2005	King Kong	207000000	218080025	550517357
50	51	Mar 5, 2010	Alice in Wonderland	200000000	334191110	1025491110
64	65	Jun 9, 2017	The Mummy	195000000	80101125	409953905
...	...	...	...	...	...	...
5668	69	Nov 16, 1942	Cat People	134000	4000000	8000000
5676	77	Oct 1, 1968	Night of the Living Dead	114000	12087064	30087064
5677	78	Feb 8, 1915	The Birth of a Nation	110000	10000000	11000000
5699	100	Aug 30, 1972	The Last House on the Left	87000	3100000	3100000
5718	19	Feb 22, 2008	The Signal	50000	251150	406299

165 rows × 6 columns

Let's take a closer look at the Avengers duplicate

In [41]:
```
df_the_numbers[df_the_numbers['movie']=='The Avengers']
```

Out[41]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
26	27	May 4, 2012	The Avengers	225000000	623279547	1517935897
934	35	Aug 14, 1998	The Avengers	60000000	23385416	48585416

The two movies are from two very different years. Let's extract the year and create that as a new columnn so that it can be used to create a unique field for combining the data.

In [42]:
```
df_the_numbers['year'] = df_the_numbers['release_date'].str[-4:]
```

Now let's take a look at the range of the years in the data.

In [43]:
```
df_the_numbers.describe()
```

Out[43]:

	id	production_budget	domestic_gross	worldwide_gross
count	5,782.0	5,782.0	5,782.0	5,782.0
mean	50.37236250432376	31,587,757.0965064	41,873,326.867001034	91,487,460.90643376
std	28.821076273431096	41,812,076.82694309	68,240,597.35690415	174,719,968.77890477
min	1.0	1,100.0	0.0	0.0
25%	25.0	5,000,000.0	1,429,534.5	4,125,414.75
50%	50.0	17,000,000.0	17,225,945.0	27,984,448.5
75%	75.0	40,000,000.0	52,348,661.5	97,645,836.5
max	100.0	425,000,000.0	936,662,225.0	2,776,345,279.0

In [ ]:
```

```

In [44]:
```
df_the_numbers['title_year'] = df_the_numbers['movie']+"_"+df_the_numb
df_the_numbers.head()
```

Out[44]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	year	
0	1	Dec 18, 2009	Avatar	425000000	760507625	2776345279	2009	A
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	2011	P (
2	3	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350	2019	Ph
3	4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963	2015	U
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	2017	St V

Now if we check for duplicates on this new combined field, let's see what we get.

In [45]:
```
df_the_numbers['title_year'].duplicated().sum()
```

Out[45]: 1

Great, now we just have one duplicate. Let's see what movie it is.

In [46]:
```
df_the_numbers[df_the_numbers.duplicated('title_year', keep=False) ==
```

Out[46]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	year	t
3455	56	Jun 5, 2009	Home	12000000	0	0	2009	Ho
5459	60	Apr 23, 2009	Home	500000	15433	44793168	2009	Ho

This movie is not significant and will not tie into the dataset since it's from 2009 and the other data starts after 2010, so it can be ignored.

Previously I did not review the year field on The Numbers data. Let's make it an integer so we can get a sense for the range.
Let's also format the budget and gross numbers so that we can read the values easier.

In [47]:
```
df_the_numbers['year']=df_the_numbers['year'].astype(int)
df_the_numbers.style.format({
    "production_budget": "{:,d}",
    "domestic_gross": "{:,d}",
    "worldwide_gross": "{:,d}"
})
```

Out[47]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gr
0	1	Dec 18, 2009	Avatar	425,000,000	760,507,625	2,776,345,
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410,600,000	241,063,875	1,045,663,
2	3	Jun 7, 2019	Dark Phoenix	350,000,000	42,762,350	149,762,
3	4	May 1, 2015	Avengers: Age of Ultron	330,600,000	459,005,868	1,403,013,
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317,000,000	620,181,382	1,316,721,
5	6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	306,000,000	936,662,225	2,053,311,
6	7	Apr 27, 2018	Avengers: Infinity War	300,000,000	678,815,482	2,048,134,
		May 24	Pirates of the			

In [48]:
```
df_the_numbers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   id                 5782 non-null   int64 
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   int64 
 4   domestic_gross     5782 non-null   int64 
 5   worldwide_gross    5782 non-null   int64 
 6   year               5782 non-null   int64 
 7   title_year         5782 non-null   object
dtypes: int64(5), object(3)
memory usage: 361.5+ KB
```

The year field is now updated to int.

In [49]:
```
df_the_numbers.describe()
```

Out[49]:

	id	production_budget	domestic_gross	worldwide_gross	
count	5,782.0	5,782.0	5,782.0	5,782.0	
mean	50.37236250432376	31,587,757.0965064	41,873,326.867001034	91,487,460.90643376	2,
std	28.821076273431096	41,812,076.82694309	68,240,597.35690415	174,719,968.77890477	1
min	1.0	1,100.0	0.0	0.0	
25%	25.0	5,000,000.0	1,429,534.5	4,125,414.75	
50%	50.0	17,000,000.0	17,225,945.0	27,984,448.5	
75%	75.0	40,000,000.0	52,348,661.5	97,645,836.5	
max	100.0	425,000,000.0	936,662,225.0	2,776,345,279.0	

It looks like the numbers data spans from 1915 to 2020, so a much greater range than most of the other data. Let's see how many films there are from 2010 to 2020.

In [50]:
```
df_the_numbers[df_the_numbers['year']>=2010].count()
```

Out[50]:
```
id                   2194
release_date         2194
movie                2194
production_budget    2194
domestic_gross       2194
worldwide_gross      2194
year                 2194
title_year           2194
dtype: int64
```

It looks like there's 2,194 films from 2010 and onwards.

In [51]:
```
df_movie_basics['primary_title'].duplicated().sum()
```

Out[51]:
```
10073
```

There's a lot more duplicates, which isn't too surprising given how much more data is in the IMDB dataset. Let's first explore the duplicate that we saw in the df_bom: Bluebeard

In [52]:
```
df_movie_basics[df_movie_basics['primary_title']=='Bluebeard']
```

Out[52]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
40404	tt2442772	Bluebeard	Barbazul	2012	98.0	Horror
112563	tt6599340	Bluebeard	Haebing	2017	117.0	Thriller

We see that there are also 2 records, with two different years as well. Once we've combined the data from the two tables, let's again create a combined field to join the datasets on to create more accuracy than simply the title.

First let's join the IMDB data into one dataframe

In [53]:
```
df_imdb = pd.read_sql(""" SELECT * FROM movie_basics
                        JOIN movie_ratings USING(movie_id)

; """, conn)
df_imdb.head()
```

Out[53]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	av
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	

The combined data looks good from the sample above. Now let's see how the data types are setup for the fields.

In [54]:
```
df_imdb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   movie_id         73856 non-null  object
 1   primary_title    73856 non-null  object
 2   original_title   73856 non-null  object
 3   start_year       73856 non-null  int64
 4   runtime_minutes  66236 non-null  float64
 5   genres           73052 non-null  object
 6   averagerating    73856 non-null  float64
 7   numvotes         73856 non-null  int64
dtypes: float64(2), int64(2), object(4)
memory usage: 4.5+ MB
```

Everything here looks as expected. Let's get some more information around the descriptive statistics for the data.

In [55]:
```
df_imdb.describe()
```

Out[55]:

	start_year	runtime_minutes	averagerating	numvotes
count	73,856.0	66,236.0	73,856.0	73,856.0
mean	2,014.276131932409	94.6540400990398	6.332728552859619	3,523.6621669194105
std	2.614807009690716	208.57411133795523	1.4749783548957056	30,294.02297110745
min	2,010.0	3.0	1.0	5.0
25%	2,012.0	81.0	5.5	14.0
50%	2,014.0	91.0	6.5	49.0
75%	2,016.0	104.0	7.4	282.0
max	2,019.0	51,420.0	10.0	1,841,066.0

This combined dataset has 73,856 records which is in line with the number of records in the movie ratings table. Again, we're seeing movies from 2010 to 2019, with an average runtime of 94 minutes, average rating of 6.3 and 3,500 votes.

Next lets check to see how many duplicates there are.

In [56]:
```
df_imdb['primary_title'].duplicated().sum()
```

Out[56]:
```
3863
```

There are still 3,863 duplicates based on title alone, once the movie_basics and movie_ratings have been combined.

In [57]:
```
df_imdb['title_year'] = df_imdb['primary_title']+"_"+df_imdb.start_yea
df_imdb.head()
```

Out[57]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	av
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	

Now let's check to see how many duplicate values we get for the new combined title and year field:

In [58]:
```
df_imdb['title_year'].duplicated().sum()
```

Out[58]: 585

Let's look to see if we can take the record that has the most number of votes. I'll start by making a dataframe that is just the duplicate records, sorting by the combined title and year field, sorted in descending order by number of votes so that we can take the most popular films.

In [59]:
```
df_imdb_dup = df_imdb[df_imdb['title_year'].duplicated(keep=False)].sc
df_imdb_dup.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1135 entries, 56862 to 14576
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         1135 non-null   object 
 1   primary_title    1135 non-null   object 
 2   original_title   1135 non-null   object 
 3   start_year       1135 non-null   int64  
 4   runtime_minutes  1032 non-null   float64
 5   genres           1122 non-null   object 
 6   averagerating    1135 non-null   float64
 7   numvotes         1135 non-null   int64  
 8   title_year       1135 non-null   object 
dtypes: float64(2), int64(2), object(5)
memory usage: 88.7+ KB
```

In [60]:
```
df_imdb_dup.head()
```

Out[60]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genre
56862	tt5815346	Zoom	Zoom	2016	158.0	Comedy,Drama,Romanc
62945	tt6667868	Zoom	Zoom	2016	nan	Horr
49080	tt4842680	Zeus	Zeus	2016	115.0	Biography,Drama,Histo
58771	tt6066078	Zeus	Zeus	2016	105.0	Dran
23668	tt2380333	Worm	Worm	2013	93.0	Horror,Romance,Sci-

It seems like a reasonable approach, to take the films that have the most votes, as they're more likely to sync up with the data for box office revenue.

Next let's get rid of the duplicates with the lessor amount of votes.

In [61]:
```python
df_imdb_dup.drop_duplicates(subset='title_year', keep='first',inplace=
df_imdb_dup.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 550 entries, 56862 to 12111
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         550 non-null    object 
 1   primary_title    550 non-null    object 
 2   original_title   550 non-null    object 
 3   start_year       550 non-null    int64  
 4   runtime_minutes  519 non-null    float64
 5   genres           549 non-null    object 
 6   averagerating    550 non-null    float64
 7   numvotes         550 non-null    int64  
 8   title_year       550 non-null    object 
dtypes: float64(2), int64(2), object(5)
memory usage: 43.0+ KB
```

We're left with 550 entries, which means some of the duplicates had more than 2 records.

Now let's remove the duplicates from the imdb dataframe and append the cleaned up df_imdb_dup dataframe

In [62]:
```python
df_imdb['title_year'].duplicated(keep=False).sum()
```

Out[62]:
```
1135
```

In [63]:
```python
df_imdb.drop_duplicates(subset='title_year', keep=False, inplace=True)
df_imdb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 72721 entries, 0 to 73855
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         72721 non-null  object 
 1   primary_title    72721 non-null  object 
 2   original_title   72721 non-null  object 
 3   start_year       72721 non-null  int64  
 4   runtime_minutes  65204 non-null  float64
 5   genres           71930 non-null  object 
 6   averagerating    72721 non-null  float64
 7   numvotes         72721 non-null  int64  
 8   title_year       72721 non-null  object 
dtypes: float64(2), int64(2), object(5)
memory usage: 5.5+ MB
```

Now we have 72,721 records, versus the 73,856 from earlier, which is a difference of 1,135 (exactly the number of duplicates found above).

Now we'll append the cleaned up 550 entries and will expect a total of 73,271 records

In [64]:
```
df_imdb_clean = df_imdb.append(df_imdb_dup)
df_imdb_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 73271 entries, 0 to 12111
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         73271 non-null  object 
 1   primary_title    73271 non-null  object 
 2   original_title   73271 non-null  object 
 3   start_year       73271 non-null  int64  
 4   runtime_minutes  65723 non-null  float64
 5   genres           72479 non-null  object 
 6   averagerating    73271 non-null  float64
 7   numvotes         73271 non-null  int64  
 8   title_year       73271 non-null  object 
dtypes: float64(2), int64(2), object(5)
memory usage: 5.6+ MB
```

In [65]:
```
df_imdb_clean.describe()
```

Out[65]:

	start_year	runtime_minutes	averagerating	numvotes
count	73,271.0	65,723.0	73,271.0	73,271.0
mean	2,014.2753476818932	94.68806962554966	6.332524463976198	3,550.79122708848
std	2.615655224257939	209.37797851151737	1.4754863362111563	30,413.17698729149
min	2,010.0	3.0	1.0	5.0
25%	2,012.0	81.0	5.5	14.0
50%	2,014.0	91.0	6.5	49.0
75%	2,016.0	104.0	7.4	285.0
max	2,019.0	51,420.0	10.0	1,841,066.0

This looks as we'd expect. Now let's confirm that there's no duplicate values so that we can join on the Box Office Mojo data

In [66]:
```
df_imdb_clean['title_year'].duplicated(keep=False).sum()
```

Out[66]:
```
0
```

Now let's create the combined title and year field for the df_bom data

In [67]:
```
df_bom['title_year'] = df_bom['title']+"_"+df_bom.year.astype(str)
df_bom.head()
```

Out[67]:

	title	studio	domestic_gross	foreign_gross	year	title_year
0	Toy Story 3	BV	415,000,000.0	652,000,000.0	2010	Toy Story 3_2010
1	Alice in Wonderland (2010)	BV	334,200,000.0	691,300,000.0	2010	Alice in Wonderland (2010)_2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296,000,000.0	664,300,000.0	2010	Harry Potter and the Deathly Hallows Part 1_2010
3	Inception	WB	292,600,000.0	535,700,000.0	2010	Inception_2010
4	Shrek Forever After	P/DW	238,700,000.0	513,900,000.0	2010	Shrek Forever After_2010

Check on the Bluebeard title to see how it appears in both datasets before we combine them

In [68]:
```
print(df_imdb_clean[df_imdb_clean['primary_title']=='Bluebeard'])
print(df_bom[df_bom['title']=="Bluebeard"])
```

```
        movie_id primary_title original_title  start_year  runtime_mi
nutes  \
24962  tt2442772     Bluebeard       Barbazul        2012
98.0
62638  tt6599340     Bluebeard        Haebing        2017
117.0

         genres  averagerating  numvotes      title_year
24962    Horror            6.1        19  Bluebeard_2012
62638  Thriller            6.4      1269  Bluebeard_2017
          title  studio  domestic_gross  foreign_gross  year       tit
le_year
317   Bluebeard  Strand        33,500.0        5,200.0  2010  Bluebea
rd_2010
3045  Bluebeard   WGUSA        43,100.0            0.0  2017  Bluebea
rd_2017
```

It looks like we'll only get one combined record on the Bluebeard from 2017, which is a good result. Now merge the imdb dataframe with the Box Office Mojo dataframe

In [69]:
```
df_combine_1 =  pd.merge(left=df_bom, right=df_imdb_clean, on="title_y
df_combine_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1822 entries, 0 to 1821
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   title            1822 non-null   object 
 1   studio           1820 non-null   object 
 2   domestic_gross   1822 non-null   float64
 3   foreign_gross    1822 non-null   float64
 4   year             1822 non-null   int64  
 5   title_year       1822 non-null   object 
 6   movie_id         1822 non-null   object 
 7   primary_title    1822 non-null   object 
 8   original_title   1822 non-null   object 
 9   start_year       1822 non-null   int64  
 10  runtime_minutes  1822 non-null   float64
 11  genres           1822 non-null   object 
 12  averagerating    1822 non-null   float64
 13  numvotes         1822 non-null   int64  
dtypes: float64(4), int64(3), object(7)
memory usage: 213.5+ KB
```

Next let's add in The Numbers data for the production budget data.

In [70]:
```
df_combined = pd.merge(left=df_combine_1, right=df_the_numbers, on="ti
df_combined.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1025 entries, 0 to 1024
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   title              1025 non-null   object 
 1   studio             1025 non-null   object 
 2   domestic_gross_x   1025 non-null   float64
 3   foreign_gross      1025 non-null   float64
 4   year_x             1025 non-null   int64  
 5   title_year         1025 non-null   object 
 6   movie_id           1025 non-null   object 
 7   primary_title      1025 non-null   object 
 8   original_title     1025 non-null   object 
 9   start_year         1025 non-null   int64  
 10  runtime_minutes    1025 non-null   float64
 11  genres             1025 non-null   object 
 12  averagerating      1025 non-null   float64
 13  numvotes           1025 non-null   int64  
 14  id                 1025 non-null   int64  
 15  release_date       1025 non-null   object 
 16  movie              1025 non-null   object 
 17  production_budget  1025 non-null   int64  
 18  domestic_gross_y   1025 non-null   int64  
 19  worldwide_gross    1025 non-null   int64  
 20  year_y             1025 non-null   int64  
dtypes: float64(4), int64(8), object(9)
memory usage: 176.2+ KB
```

We have a few fields that had the same name, so I'll have to clean that up. However, all of the runtime_minutes are non-null values, so no need to review that further. First let's take a look at a few rows of the combined data set.

In [71]:
```
df_combined.head()
```

Out[71]:

	title	studio	domestic_gross_x	foreign_gross	year_x	title_year	movie_id	primar
0	Toy Story 3	BV	415,000,000.0	652,000,000.0	2010	Toy Story 3_2010	tt0435761	Toy S
1	Inception	WB	292,600,000.0	535,700,000.0	2010	Inception_2010	tt1375666	Inc
2	Shrek Forever After	P/DW	238,700,000.0	513,900,000.0	2010	Shrek Forever After_2010	tt0892791	Foreve
3	The Twilight Saga: Eclipse	Sum.	300,500,000.0	398,000,000.0	2010	The Twilight Saga: Eclipse_2010	tt1325004	The T E
4	Iron Man 2	Par.	312,400,000.0	311,500,000.0	2010	Iron Man 2_2010	tt1228705	Iron

5 rows × 21 columns

Let's reformat the numbers data again to make it easier to read.

In [72]:
```
df_combined.style.format({
    "production_budget": "{:,d}",
    "domestic_gross_y": "{:,d}",
    "worldwide_gross": "{:,d}"
})
```

Out[72]:

	title	studio	domestic_gross_x	foreign_gross	year_x	title
0	Toy Story 3	BV	415000000.000000	652000000.000000	2010	Toy Story 3
1	Inception	WB	292600000.000000	535700000.000000	2010	Inceptior
2	Shrek Forever After	P/DW	238700000.000000	513900000.000000	2010	Shrek F After
3	The Twilight Saga: Eclipse	Sum.	300500000.000000	398000000.000000	2010	The Twilight Eclipse
4	Iron Man 2	Par.	312400000.000000	311500000.000000	2010	Iron Man 2
5	Tangled	BV	200800000.000000	391000000.000000	2010	Tangled
6	Despicable Me	Uni.	251500000.000000	291600000.000000	2010	Despicable Me
7	How to Train Your Dragon	P/DW	217600000.000000	277300000.000000	2010	How to Trai Dragon
8	The Chronicles of Narnia: The	Fox	104400000.000000	311300000.000000	2010	The Chroni Narnia: The Voy

Let's take a closer look at the revenue data as that's a key field for the analysis.

In [73]:

In [73]:

```
df_combined.sort_values('domestic_gross_x', ascending=False).head(20)
```

Out[73]:

	title	studio	domestic_gross_x	foreign_gross	year_x	title_year	movie_id	pr
939	Black Panther	BV	700,100,000.0	646,900,000.0	2018	Black Panther_2018	tt1825683	
938	Avengers: Infinity War	BV	678,800,000.0	0.0	2018	Avengers: Infinity War_2018	tt4154756	
617	Jurassic World	Uni.	652,300,000.0	0.0	2015	Jurassic World_2015	tt0369610	
941	Incredibles 2	BV	608,600,000.0	634,200,000.0	2018	Incredibles 2_2018	tt3606756	In
733	Rogue One: A Star Wars Story	BV	532,200,000.0	523,900,000.0	2016	Rogue One: A Star Wars Story_2016	tt3748528	F A
734	Finding Dory	BV	486,300,000.0	542,300,000.0	2016	Finding Dory_2016	tt2277860	Fi
619	Avengers: Age of Ultron	BV	459,000,000.0	946,400,000.0	2015	Avengers: Age of Ultron_2015	tt2395427	Ag
275	The Dark Knight Rises	WB	448,100,000.0	636,800,000.0	2012	The Dark Knight Rises_2012	tt1345836	K
396	The Hunger Games: Catching Fire	LGF	424,700,000.0	440,300,000.0	2013	The Hunger Games: Catching Fire_2013	tt1951264	T
940	Jurassic World: Fallen Kingdom	Uni.	417,700,000.0	891,800,000.0	2018	Jurassic World: Fallen Kingdom_2018	tt4881806	W
0	Toy Story 3	BV	415,000,000.0	652,000,000.0	2010	Toy Story 3_2010	tt0435761	
858	Wonder Woman	WB	412,600,000.0	409,300,000.0	2017	Wonder Woman_2017	tt0451279	
393	Iron Man 3	BV	409,000,000.0	805,800,000.0	2013	Iron Man 3_2013	tt1300854	
732	Captain America: Civil War	BV	408,100,000.0	745,200,000.0	2016	Captain America: Civil War_2016	tt3498820	
280	The Hunger Games	LGF	408,000,000.0	286,400,000.0	2012	The Hunger Games_2012	tt1392170	T
855	Jumanji: Welcome to the Jungle	Sony	404,500,000.0	557,600,000.0	2017	Jumanji: Welcome to the Jungle_2017	tt2283362	V

392	Frozen	BV	400,700,000.0	875,700,000.0	2013	Frozen_2013	tt2294629
736	The Secret Life of Pets	Uni.	368,400,000.0	507,100,000.0	2016	The Secret Life of Pets_2016	tt2709768
394	Despicable Me 2	Uni.	368,100,000.0	602,700,000.0	2013	Despicable Me 2_2013	tt1690953
738	Deadpool	Fox	363,100,000.0	420,000,000.0	2016	Deadpool_2016	tt1431045

20 rows × 21 columns

It looks like there's an issue with the foreign gross revenue for some very important films, like Avengers: Infinity War and Jurrasic World. Let's take a closer look where the foreign_gross has a zero value.

In [74]:
```
df_combined[df_combined['foreign_gross']==0].sort_values('domestic_gro
```

Out[74]:

	title	studio	domestic_gross_x	foreign_gross	year_x	title_year	movie_id
938	Avengers: Infinity War	BV	678,800,000.0	0.0	2018	Avengers: Infinity War_2018	tt4154756
617	Jurassic World	Uni.	652,300,000.0	0.0	2015	Jurassic World_2015	tt0369610
618	Furious 7	Uni.	353,000,000.0	0.0	2015	Furious 7_2015	tt2820852
853	The Fate of the Furious	Uni.	226,000,000.0	0.0	2017	The Fate of the Furious_2017	tt4630562
990	Book Club	Par.	68,600,000.0	0.0	2018	Book Club_2018	tt6857166
673	War Room	TriS	67,800,000.0	0.0	2015	War Room_2015	tt3832914
913	All Eyez on Me	LG/S	44,900,000.0	0.0	2017	All Eyez on Me_2017	tt1666185
472	Snitch	LG/S	42,900,000.0	0.0	2013	Snitch_2013	tt0882977
230	Courageous	TriS	34,500,000.0	0.0	2011	Courageous_2011	tt1630036
588	When the Game Stands Tall	TriS	30,100,000.0	0.0	2014	When the Game Stands Tall_2014	tt2247476
920	Home Again	ORF	27,000,000.0	0.0	2017	Home Again_2017	tt5719700
1007	Winchester	LGF	25,100,000.0	0.0	2018	Winchester_2018	tt1072748
238	Our Idiot Brother	Wein.	24,800,000.0	0.0	2011	Our Idiot Brother_2011	tt1637706
820	Whiskey Tango Foxtrot	Par.	23,100,000.0	0.0	2016	Whiskey Tango Foxtrot_2016	tt3553442
822	Keanu	WB (NL)	20,600,000.0	0.0	2016	Keanu_2016	tt4139124
486	Broken City	Fox	19,700,000.0	0.0	2013	Broken City_2013	tt1235522
488	Admission	Focus	18,000,000.0	0.0	2013	Admission_2013	tt1814621
596	Addicted	LGF	17,400,000.0	0.0	2014	Addicted_2014	tt2205401
824	Norm of the North	LGF	17,100,000.0	0.0	2016	Norm of the North_2016	tt1594972
826	The Infiltrator	BG	15,400,000.0	0.0	2016	The Infiltrator_2016	tt1355631

20 rows × 21 columns

The zero values are concerning. Let's add a few fields to make the comparison easier: a combined gross field and a delta field.

In [75]:
```
df_combined['combined_gross'] = df_combined['domestic_gross_x']+df_com
df_combined['gross_delta'] = df_combined['worldwide_gross']-df_combine
df_combined.head(10)
```

Out[75]:

	title	studio	domestic_gross_x	foreign_gross	year_x	title_year	movie_id	prima
0	Toy Story 3	BV	415,000,000.0	652,000,000.0	2010	Toy Story 3_2010	tt0435761	Toy
1	Inception	WB	292,600,000.0	535,700,000.0	2010	Inception_2010	tt1375666	Ir
2	Shrek Forever After	P/DW	238,700,000.0	513,900,000.0	2010	Shrek Forever After_2010	tt0892791	Forev
3	The Twilight Saga: Eclipse	Sum.	300,500,000.0	398,000,000.0	2010	The Twilight Saga: Eclipse_2010	tt1325004	The
4	Iron Man 2	Par.	312,400,000.0	311,500,000.0	2010	Iron Man 2_2010	tt1228705	Iro
5	Tangled	BV	200,800,000.0	391,000,000.0	2010	Tangled_2010	tt0398286	
6	Despicable Me	Uni.	251,500,000.0	291,600,000.0	2010	Despicable Me_2010	tt1323594	Des
7	How to Train Your Dragon	P/DW	217,600,000.0	277,300,000.0	2010	How to Train Your Dragon_2010	tt0892769	How Your
8	The Chronicles of Narnia: The Voyage of the Da...	Fox	104,400,000.0	311,300,000.0	2010	The Chronicles of Narnia: The Voyage of the Da...	tt0980970	Chron Nar Vo t
9	The Karate Kid	Sony	176,600,000.0	182,500,000.0	2010	The Karate Kid_2010	tt1155076	The

10 rows × 23 columns

Again, let's format the numbers so it's easier to read.

In [76]:

```
df_combined[['title', 'domestic_gross_x', 'foreign_gross','combined_gr
    "production_budget": "{:,d}",
    "domestic_gross_y": "{:,d}",
    "worldwide_gross": "{:,d}",
    "domestic_gross_x": "{:,.0f}",
    "foreign_gross": "{:,.0f}"

})
```

Out[76]:

	title	domestic_gross_x	foreign_gross	combined_gross	domestic_gross_y	worldv
939	Black Panther	700,100,000	646,900,000	1347000000.000000	700,059,566	1,3
938	Avengers: Infinity War	678,800,000	0	678800000.000000	678,815,482	2,0
617	Jurassic World	652,300,000	0	652300000.000000	652,270,625	1,6
941	Incredibles 2	608,600,000	634,200,000	1242800000.000000	608,581,744	1,2
733	Rogue One: A Star Wars Story	532,200,000	523,900,000	1056100000.000000	532,177,324	1,0
734	Finding Dory	486,300,000	542,300,000	1028600000.000000	486,295,561	1,0
619	Avengers: Age of Ultron	459,000,000	946,400,000	1405400000.000000	459,005,868	1,4
275	The Dark Knight Rises	448,100,000	636,800,000	1084900000.000000	448,139,099	1,0
396	The Hunger Games: Catching Fire	424,700,000	440,300,000	865000000.000000	424,668,047	8
940	Jurassic World: Fallen Kingdom	417,700,000	891,800,000	1309500000.000000	417,719,760	1,3
0	Toy Story 3	415,000,000	652,000,000	1067000000.000000	415,004,880	1,0
858	Wonder Woman	412,600,000	409,300,000	821900000.000000	412,563,408	8
393	Iron Man 3	409,000,000	805,800,000	1214800000.000000	408,992,272	1,2
732	Captain America: Civil War	408,100,000	745,200,000	1153300000.000000	408,084,349	1,1

280	The Hunger Games	408,000,000	286,400,000	694400000.000000	408,010,692	6
855	Jumanji: Welcome to the Jungle	404,500,000	557,600,000	962100000.000000	404,508,916	9
392	Frozen	400,700,000	875,700,000	1276400000.000000	400,738,009	1,2
736	The Secret Life of Pets	368,400,000	507,100,000	875500000.000000	368,384,330	8
394	Despicable Me 2	368,100,000	602,700,000	970800000.000000	368,065,385	9
738	Deadpool	363,100,000	420,000,000	783100000.000000	363,070,709	8

Now let's sort by the largest differences in the gross_delta column

In [77]:
```python
df_combined.sort_values('gross_delta', ascending=False).head(20)
```

Out[77]:

	title	studio	domestic_gross_x	foreign_gross	year_x	title_year	movie_id
938	Avengers: Infinity War	BV	678,800,000.0	0.0	2018	Avengers: Infinity War_2018	tt4154756
618	Furious 7	Uni.	353,000,000.0	0.0	2015	Furious 7_2015	tt2820852
853	The Fate of the Furious	Uni.	226,000,000.0	0.0	2017	The Fate of the Furious_2017	tt4630562
617	Jurassic World	Uni.	652,300,000.0	0.0	2015	Jurassic World_2015	tt0369610
717	Bajrangi Bhaijaan	Eros	8,199,999.0	0.0	2015	Bajrangi Bhaijaan_2015	tt3863552
502	Yeh Jawaani Hai Deewani	Eros	3,800,000.0	0.0	2013	Yeh Jawaani Hai Deewani_2013	tt2178470
42	Dear John	SGem	80,000,000.0	35,000,000.0	2010	Dear John_2010	tt0989757
284	Wreck-It Ralph	BV	189,400,000.0	281,800,000.0	2012	Wreck-It Ralph_2012	tt1772341
624	The Martian	Fox	228,400,000.0	401,700,000.0	2015	The Martian_2015	tt3659388
315	Step Up Revolution	LG/S	35,100,000.0	105,400,000.0	2012	Step Up Revolution_2012	tt1800741
470	Grudge Match	WB	29,800,000.0	15,100,000.0	2013	Grudge Match_2013	tt1661382
1021	The Hurricane	ENTMP	6,100,000.0	0.0	2018	The Hurricane	tt5360952

	Heist					Heist_2018	
285	Django Unchained	Wein.	162,800,000.0	262,600,000.0	2012	Django Unchained_2012	tt1853728
651	Pan	WB	35,100,000.0	93,300,000.0	2015	Pan_2015	tt3332064
828	Jackie	FoxS	14,000,000.0	0.0	2016	Jackie_2016	tt1619029
990	Book Club	Par.	68,600,000.0	0.0	2018	Book Club_2018	tt6857166
364	The Master	Wein.	16,399,999.0	11,900,000.0	2012	The Master_2012	tt1560747
722	Youth	FoxS	2,700,000.0	0.0	2015	Youth_2015	tt3312830
321	Abraham Lincoln: Vampire Hunter	Fox	37,500,000.0	79,000,000.0	2012	Abraham Lincoln: Vampire Hunter_2012	tt1611224
464	Delivery Man	BV	30,700,000.0	19,300,000.0	2013	Delivery Man_2013	tt2387559

20 rows × 23 columns

Now let's look at the largest differences in the other direction.

In [78]:
```
df_combined.sort_values('gross_delta', ascending=True).head(20)
```

Out[78]:

	title	studio	domestic_gross_x	foreign_gross	year_x	title_year	movie_id
452	Dhoom 3	Yash	8,000,000.0	80,000,000.0	2013	Dhoom 3_2013	tt1833673
868	The Greatest Showman	Fox	174,300,000.0	260,700,000.0	2017	The Greatest Showman_2017	tt1485796
152	Real Steel	BV	85,500,000.0	213,800,000.0	2011	Real Steel_2011	tt0433035
436	Escape Plan	LG/S	25,100,000.0	112,200,000.0	2013	Escape Plan_2013	tt1211956
398	Gravity	WB	274,100,000.0	449,100,000.0	2013	Gravity_2013	tt1454468
866	Dunkirk	WB	188,000,000.0	337,200,000.0	2017	Dunkirk_2017	tt5013056
456	August: Osage County	Wein.	37,700,000.0	36,500,000.0	2013	August: Osage County_2013	tt1322269
782	Office Christmas Party	Par.	54,800,000.0	59,700,000.0	2016	Office Christmas Party_2016	tt1711525
756	Alice Through the Looking Glass	BV	77,000,000.0	222,400,000.0	2016	Alice Through the Looking Glass_2016	tt2567026
437	Last Vegas	CBS	63,900,000.0	70,500,000.0	2013	Last	tt1204975

						Vegas_2013	
221	Shark Night 3D	Rela.	18,900,000.0	21,300,000.0	2011	Shark Night 3D_2011	tt1633356
165	War Horse	BV	79,900,000.0	97,700,000.0	2011	War Horse_2011	tt1568911
745	La La Land	LG/S	151,100,000.0	295,000,000.0	2016	La La Land_2016	tt3783958
449	The Mortal Instruments: City of Bones	SGem	31,200,000.0	64,200,000.0	2013	The Mortal Instruments: City of Bones_2013	tt1538403
970	The Commuter	LGF	36,300,000.0	83,600,000.0	2018	The Commuter_2018	tt1590193
629	San Andreas	WB (NL)	155,200,000.0	318,800,000.0	2015	San Andreas_2015	tt2126355
290	Journey 2: The Mysterious Island	WB (NL)	103,900,000.0	231,400,000.0	2012	Journey 2: The Mysterious Island_2012	tt1397514
865	The Boss Baby	Fox	175,000,000.0	353,000,000.0	2017	The Boss Baby_2017	tt3874544
687	Mortdecai	LGF	7,700,000.0	39,600,000.0	2015	Mortdecai_2015	tt3045616
280	The Hunger Games	LGF	408,000,000.0	286,400,000.0	2012	The Hunger Games_2012	tt1392170

20 rows × 23 columns

The domestic and worldwide gross numbers from The Numbers data seems more consistent than the data from IMDB, so I will go ahead and use those columns and rename the other columns to drop the x and y from the title.

In [79]:
```
df_combined.drop(columns=['domestic_gross_x', 'year_y', 'id', 'foreign
```

In [80]:
```
df_combined.rename(columns={"domestic_gross_y": "domestic_gross", "yea
```

In [81]:
```
df_combined.describe()
```

Out[81]:

	year	start_year	runtime_minutes	averagerating	
count	1,025.0	1,025.0	1,025.0	1,025.0	
mean	2,013.658536585366	2,013.658536585366	109.93463414634147	6.459609756097561	146,
std	2.546920591017949	2.546920591017949	17.840346120384602	0.9422731842788351	178,0
min	2,010.0	2,010.0	41.0	1.6	
25%	2,011.0	2,011.0	97.0	5.9	
50%	2,014.0	2,014.0	107.0	6.5	
75%	2,016.0	2,016.0	120.0	7.1	
max	2,018.0	2,018.0	180.0	8.8	

There are now 1,025 records after starting with these original datasets:
Box Office Mojo: 3,387
IMDB movie_ratings: 73,856
IMDB movie_basics: 146,144
IMDB clean: 73,271
The Numbers: 5,782

Clearly I've lost a lot of the movies that were in the IMDB database, but the meaningful data is in the box office revenues, so this will be fine to go ahead with the analysis

In [82]:
```
df_combined.head()
```

Out[82]:

	title	studio	year	title_year	movie_id	primary_title	original_title	start_year	run
0	Toy Story 3	BV	2010	Toy Story 3_2010	tt0435761	Toy Story 3	Toy Story 3	2010	
1	Inception	WB	2010	Inception_2010	tt1375666	Inception	Inception	2010	
2	Shrek Forever After	P/DW	2010	Shrek Forever After_2010	tt0892791	Shrek Forever After	Shrek Forever After	2010	
3	The Twilight Saga: Eclipse	Sum.	2010	The Twilight Saga: Eclipse_2010	tt1325004	The Twilight Saga: Eclipse	The Twilight Saga: Eclipse	2010	
4	Iron Man 2	Par.	2010	Iron Man 2_2010	tt1228705	Iron Man 2	Iron Man 2	2010	

In [83]:
```
#check for null values
df_combined.isnull().sum()
```

Out[83]:
```
title                0
studio               0
year                 0
title_year           0
movie_id             0
primary_title        0
original_title       0
start_year           0
runtime_minutes      0
genres               0
averagerating        0
numvotes             0
release_date         0
movie                0
production_budget    0
domestic_gross       0
worldwide_gross      0
dtype: int64
```

In [84]:
```
#check for duplicates
df_combined['title'].duplicated().sum()
```

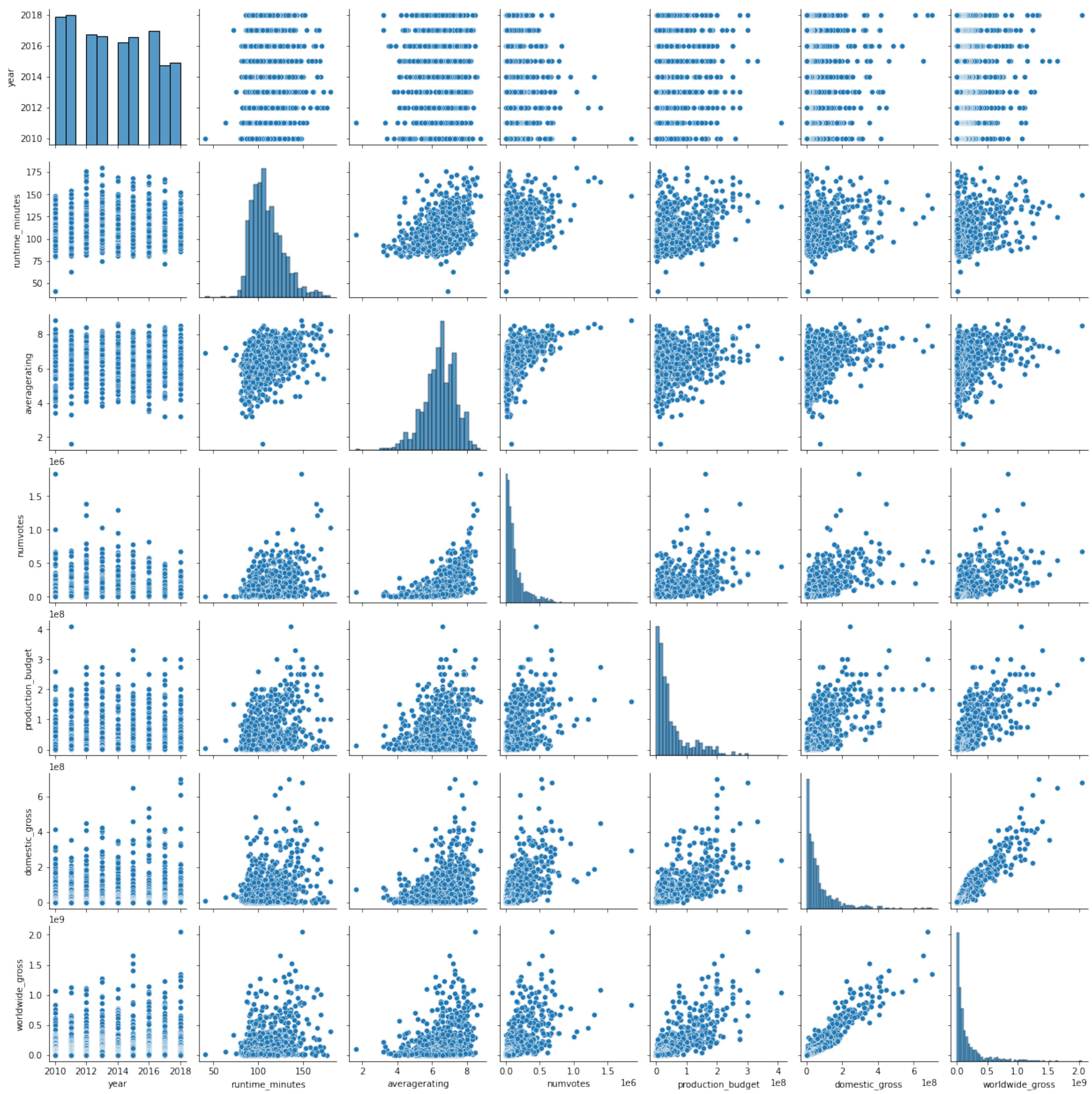
Out[84]:
```
0
```

Now there's no records that have a null value. Let's plot the info and see what it looks like. First we'll drop a couple of the columns that aren't necessary for graphing purposes.

In [85]:
```
df_data_plot = df_combined.drop(columns=['studio', 'title_year', 'star
```

In [86]:
```
sns.pairplot(df_data_plot)
```

Out[86]: <seaborn.axisgrid.PairGrid at 0x7f7ff07f27f0>



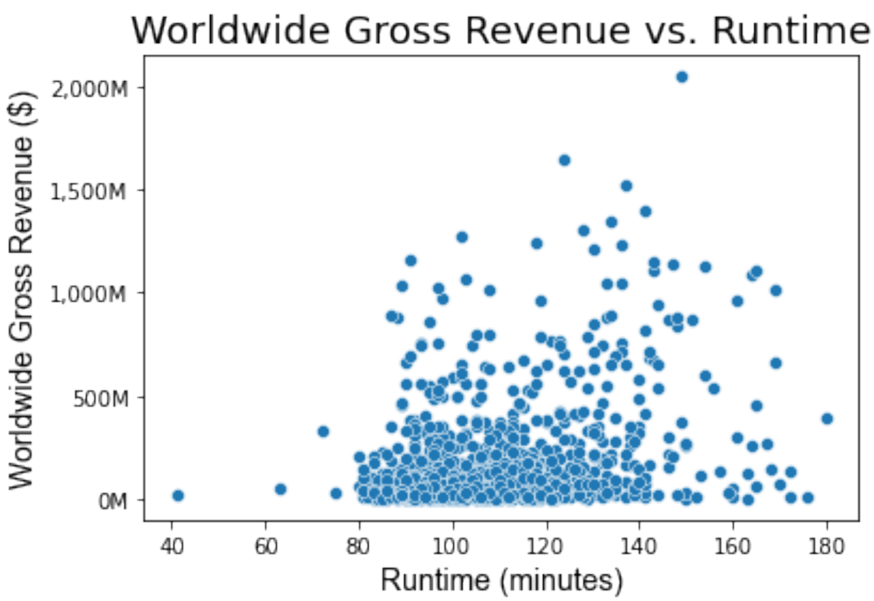
There's a lot of information here. Let's focus on a few of the key attributes for the presentation: genre, gross revenue, runtime and production budget for return on investment.

Runtime Analysis

Graph for Runtime Minutes versus Combined Gross

In [87]:
```python
font1 = {'family':'arial','color':'black','size':14}

sns.scatterplot(data=df_data_plot, x="runtime_minutes", y="worldwide_g
plt.title("Worldwide Gross Revenue vs. Runtime", fontsize=18)
plt.xlabel("Runtime (minutes)", fontdict=font1)
plt.ylabel("Worldwide Gross Revenue ($)", fontdict=font1)
current_values = plt.gca().get_yticks()
plt.gca().set_yticklabels([format(x/1000000,'1,.0f')+'M' for x in curr

plt.tick_params(axis='x', which='major', labelsize=10)
plt.tick_params(axis='y', which='major', labelsize=10)

plt.show
```

Out[87]: <function matplotlib.pyplot.show(close=None, block=None)>



Now let's create a dataframe that sorts by worldwide gross revenue and take a look at some of the top films.

In [88]:
```
top_gross = df_combined.sort_values(by="worldwide_gross", ascending=Fa
top_gross.head(10)
```

Out[88]:

	title	studio	year	title_year	movie_id	primary_title	original_title	start_year
938	Avengers: Infinity War	BV	2018	Avengers: Infinity War_2018	tt4154756	Avengers: Infinity War	Avengers: Infinity War	2018
617	Jurassic World	Uni.	2015	Jurassic World_2015	tt0369610	Jurassic World	Jurassic World	2015
618	Furious 7	Uni.	2015	Furious 7_2015	tt2820852	Furious 7	Furious Seven	2015
619	Avengers: Age of Ultron	BV	2015	Avengers: Age of Ultron_2015	tt2395427	Avengers: Age of Ultron	Avengers: Age of Ultron	2015
939	Black Panther	BV	2018	Black Panther_2018	tt1825683	Black Panther	Black Panther	2018
940	Jurassic World: Fallen Kingdom	Uni.	2018	Jurassic World: Fallen Kingdom_2018	tt4881806	Jurassic World: Fallen Kingdom	Jurassic World: Fallen Kingdom	2018
392	Frozen	BV	2013	Frozen_2013	tt2294629	Frozen	Frozen	2013
941	Incredibles 2	BV	2018	Incredibles 2_2018	tt3606756	Incredibles 2	Incredibles 2	2018
853	The Fate of the Furious	Uni.	2017	The Fate of the Furious_2017	tt4630562	The Fate of the Furious	The Fate of the Furious	2017
393	Iron Man 3	BV	2013	Iron Man 3_2013	tt1300854	Iron Man 3	Iron Man Three	2013

Next let's create a dataframe that is just the top 100, as these are the most successful films.

In [89]:
```
top_100_gross = top_gross.head(100)
```

In [90]:
```
top_100_gross.describe()
```

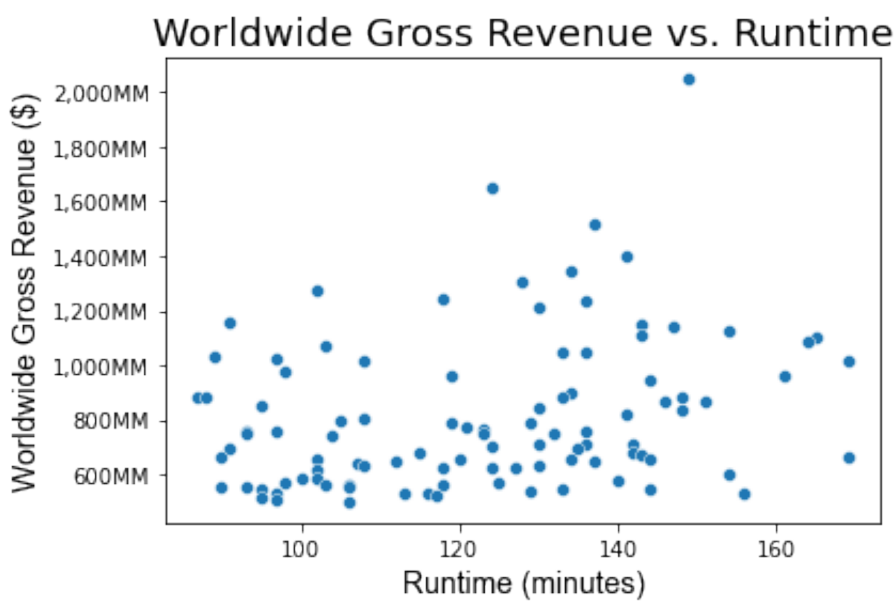
Out[90]:

	year	start_year	runtime_minutes	averagerating	
count	100.0	100.0	100.0	100.0	
mean	2,014.41	2,014.41	122.62	7.156999999999998	
std	2.478574859336174	2.478574859336174	21.40555011574278	0.7996533339800783	274,864
min	2,010.0	2,010.0	87.0	4.1	
25%	2,012.0	2,012.0	103.0	6.6	
50%	2,014.5	2,014.5	124.0	7.2	
75%	2,017.0	2,017.0	137.75	7.8	
max	2,018.0	2,018.0	169.0	8.8	

We're seeing that it is in fact 100 films, with an average worldwide gross of $817MM, a runtime of 122 minutes, and a average production budget of $164MM.
Next let's graph the data for the top 100 for revenue vs runtime.

In [91]:
```
font1 = {'family':'arial','color':'black','size':14}

sns.scatterplot(data=top_100_gross, x="runtime_minutes", y="worldwide_
plt.title("Worldwide Gross Revenue vs. Runtime", fontsize=18)
plt.xlabel("Runtime (minutes)", fontdict=font1)
plt.ylabel("Worldwide Gross Revenue ($)", fontdict=font1)
current_values = plt.gca().get_yticks()
plt.gca().set_yticklabels([format(x/1000000,'1,.0f')+'MM' for x in cur

plt.tick_params(axis='x', which='major', labelsize=10)
plt.tick_params(axis='y', which='major', labelsize=10)

plt.show
```

Out[91]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



This makes sense relative to the previous graph with all of the films. Let's print out some of the statistics.

In [92]:
```
top_gross['year'].max()
```

Out[92]:
```
2018
```

In [93]:
```
top_gross['year'].min()
```

Out[93]:
```
2010
```

In [94]:
```
top_gross["averagerating"].max()
```

Out[94]:
```
8.8
```

In [95]:
```
top_gross["averagerating"].min()
```
Out[95]:
```
1.6
```
In [96]:
```
# Which movie has the highest rating
top_gross.loc[top_gross['averagerating'].idxmax()]
```
Out[96]:
```
title                             Inception
studio                                   WB
year                                   2010
title_year                   Inception_2010
movie_id                          tt1375666
primary_title                     Inception
original_title                    Inception
start_year                             2010
runtime_minutes                       148.0
genres              Action,Adventure,Sci-Fi
averagerating                           8.8
numvotes                            1841066
release_date                   Jul 16, 2010
movie                             Inception
production_budget                 160000000
domestic_gross                    292576195
worldwide_gross                   835524642
Name: 1, dtype: object
```
I was suprised to see that Inception had the highest rating as The Runaways had a 9.2 rating. Let's take a look at the original data

In [97]:
```
df_imdb.loc[df_imdb['averagerating']=="9.2"]
df_imdb[df_imdb['primary_title']=='The Runaways']
```
Out[97]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
699	tt1017451	The Runaways	The Runaways	2010	106.0	Biography,Drama,Music
59761	tt6168914	The Runaways	The Runaways	2019	108.0	Adventure

In [98]:
```
df_bom[df_bom['title']=="The Runaways"]
```
Out[98]:

	title	studio	domestic_gross	foreign_gross	year	title_year
198	The Runaways	App.	3,600,000.0	1,100,000.0	2010	The Runaways_2010

In [99]:
```
df_combined[df_combined['title']=="The Runaways"]
```

Out[99]:

	title	studio	year	title_year	movie_id	primary_title	original_title	start_year
117	The Runaways	App.	2010	The Runaways_2010	tt1017451	The Runaways	The Runaways	2010

It looks like There were two movies called the Runaways in the IMDB data, but the 2010 movie is the one that is also in the Box Office Mojo data. That movie only had a 6.6 average rating, so it looks like this was a good assumption to use (combining data based on title and year)

While it seems helpful to get a high rating, it's not something that can be controlled for when creating a movie, so we will set that aside for now.

Genre analysis

Now let's breakdown the genre data. There's currently one column which can have multiple values. I'll need a way to view the data that allows the data to be broken out.
1. Get a list of the genres
2. Split the genres in the column into a list of values
3. Populate a set with the list of genres, since a set cannot have duplicate values
4. Create a column for each genre and add a value of 1, where the movie is in that genre

This section was tricky and my instructor provided an example of the code to use to be able to parse the genre code and do some some counts and calculations on it.

In [100]:
```
top_gross["genres"].unique()
```

Out[100]:
```
array(['Action,Adventure,Sci-Fi', 'Action,Crime,Thriller',
       'Adventure,Animation,Comedy', 'Action,Adventure,Animation',
       'Action,Adventure,Fantasy', 'Action,Adventure,Thriller',
       'Action,Thriller', 'Adventure,Family,Fantasy',
       'Action,Adventure,Comedy', 'Adventure,Fantasy',
       'Biography,Drama,Music', 'Action,Adventure,Family',
       'Action,Adventure,Drama', 'Adventure,Drama,Fantasy',
       'Horror,Thriller', 'Drama,Sci-Fi,Thriller',
       'Adventure,Drama,Sci-Fi', 'Animation,Comedy,Family',
       'Comedy,Mystery', 'Drama,Romance,Thriller', 'Comedy,Fantasy',
       'Action,Biography,Drama', 'Action,Adventure,Crime',
       'Action,Adventure,Biography', 'Action,Adventure,Horror',
       'Action,Horror,Sci-Fi', 'Action,Drama,History',
       'Action,Drama,Sci-Fi', 'Action,Sci-Fi,Thriller', 'Drama,Wester
n',
       'Comedy,Drama,Music', 'Adventure,Mystery,Sci-Fi',
       'Biography,Crime,Drama', 'Biography,Drama,Musical',
```

```
        'Drama,Romance', 'Action,Sci-Fi', 'Drama,Mystery,Thriller',
        'Adventure,Comedy,Family', 'Action,Animation,Comedy',
        'Action,Drama,Family', 'Action,Mystery,Sci-Fi',
        'Crime,Mystery,Thriller', 'Comedy,Family,Fantasy',
        'Drama,Horror,Sci-Fi', 'Documentary', 'Action,Comedy,Crime',
        'Drama,Thriller', 'Biography,Comedy,Drama',
        'Horror,Mystery,Thriller', 'Comedy,Romance', 'Drama,Family',
        'Mystery,Thriller', 'Comedy,Drama,Romance',
        'Action,Mystery,Thriller', 'Comedy,Music',
        'Action,Adventure,Mystery', 'Biography,Drama,History', 'Comedy
',
        'Comedy,Crime', 'Action,Drama', 'Adventure,Animation,Family',
        'Action,Adventure,Western', 'Drama,Mystery,Sci-Fi',
        'Mystery,Sci-Fi,Thriller', 'Crime,Drama',
        'Adventure,Drama,Western', 'Action,Comedy,Sci-Fi',
        'Biography,Drama', 'Horror,Sci-Fi,Thriller',
        'Comedy,Fantasy,Horror', 'Action,Crime,Drama',
        'Biography,Drama,Thriller', 'Action,Drama,Fantasy', 'Drama,Spo
rt',
        'Drama', 'Adventure,Comedy,Drama', 'Adventure,Comedy',
        'Action,Drama,Thriller', 'Comedy,Drama', 'Adventure,Drama,Fami
ly',
        'Horror', 'Adventure,Comedy,Crime', 'Crime,Drama,Thriller',
        'Comedy,Family,Romance', 'Drama,History,Thriller',
        'Drama,Music,Romance', 'Biography,Drama,Sport',
        'Comedy,Fantasy,Romance', 'Comedy,Drama,History',
        'Action,Comedy,Romance', 'Drama,History,War',
        'Comedy,Crime,Romance', 'Drama,Horror,Mystery',
        'Crime,Drama,Mystery', 'Drama,Fantasy,Horror', 'Drama,Romance,
War',
        'Action,Fantasy,Horror', 'Action,Biography,Comedy',
        'Action,Drama,Mystery', 'Romance,Sci-Fi,Thriller',
        'Biography,Drama,Romance', 'Action,Comedy,Drama', 'Comedy,Fami
ly',
        'Action,Drama,Romance', 'Drama,Horror,Thriller',
        'Comedy,Drama,Family', 'Comedy,Music,Romance',
        'Comedy,Horror,Romance', 'Action,Comedy,Family',
        'Adventure,Drama,Thriller', 'Drama,Fantasy,Romance',
        'Action,Crime,Mystery', 'Crime,Horror,Mystery',
        'Adventure,Comedy,Sci-Fi', 'Documentary,Music',
        'Comedy,Crime,Drama', 'Action,Comedy', 'Horror,Mystery',
        'Fantasy,Horror,Mystery', 'Adventure,Comedy,Music',
        'Drama,Music,Musical', 'Comedy,Drama,Fantasy',
        'Comedy,Drama,Mystery', 'Comedy,Horror', 'Drama,Fantasy,Music'
,
        'Drama,War', 'Adventure,Comedy,Romance', 'Biography,Drama,Fami
ly',
        'Action,Crime', 'Action,Comedy,Sport', 'Action,Drama,Sport',
        'Crime,Drama,Horror', 'Comedy,Drama,Sport', 'Action,Comedy,Hor
ror',
        'Drama,Romance,Sci-Fi', 'Comedy,Sci-Fi', 'Drama,Fantasy',
        'Adventure,Biography,Drama', 'Comedy,Drama,Musical',
        'Crime,Thriller', 'Drama,Sci-Fi', 'Adventure,Comedy,Fantasy',
```

```
       'Comedy,Crime,Thriller', 'Comedy,Sport', 'Biography,Drama,Fant
asy',
       'Adventure,Family,Sci-Fi', 'Action,Crime,Sci-Fi',
       'Action,Fantasy,Thriller', 'Drama,Music', 'Horror,Mystery,Sci-
Fi',
       'Adventure,Biography,Comedy', 'Comedy,Documentary',
       'Drama,Family,Sport', 'Action', 'Comedy,Romance,Sport',
       'Action,Family,Fantasy', 'Drama,Fantasy,Mystery',
       'Adventure,Drama,Romance', 'Drama,Mystery,Romance',
       'Drama,Mystery', 'Comedy,Mystery,Sci-Fi', 'Biography,Comedy,Cr
ime',
       'Drama,History', 'Music', 'Action,Horror,Thriller',
       'Action,Fantasy,Western', 'Crime,Drama,History',
       'Crime,Documentary', 'Drama,Horror', 'Drama,Family,Music',
       'Drama,History,Romance', 'Animation,Comedy,Drama',
       'Biography,Drama,War', 'Action,Biography,Crime',
       'Action,Crime,Horror', 'Comedy,Drama,Horror',
       'Crime,Drama,Romance', 'Action,Drama,War', 'Action,Adventure']
,
      dtype=object)
```

In [101]:
```
top_gross["genres"] = top_gross["genres"].apply(lambda x: x.split(",")
top_gross.head()
```

Out[101]:

	title	studio	year	title_year	movie_id	primary_title	original_title	start_year	ru
938	Avengers: Infinity War	BV	2018	Avengers: Infinity War_2018	tt4154756	Avengers: Infinity War	Avengers: Infinity War	2018	
617	Jurassic World	Uni.	2015	Jurassic World_2015	tt0369610	Jurassic World	Jurassic World	2015	
618	Furious 7	Uni.	2015	Furious 7_2015	tt2820852	Furious 7	Furious Seven	2015	
619	Avengers: Age of Ultron	BV	2015	Avengers: Age of Ultron_2015	tt2395427	Avengers: Age of Ultron	Avengers: Age of Ultron	2015	
939	Black Panther	BV	2018	Black Panther_2018	tt1825683	Black Panther	Black Panther	2018	

In [102]:
```
all_genres = set()

for genres in top_gross["genres"]:
    if genres:
        all_genres.update(genres)

all_genres
```

Out[102]:
```
{'Action',
 'Adventure',
 'Animation',
 'Biography',
 'Comedy',
 'Crime',
 'Documentary',
 'Drama',
 'Family',
 'Fantasy',
 'History',
 'Horror',
 'Music',
 'Musical',
 'Mystery',
 'Romance',
 'Sci-Fi',
 'Sport',
 'Thriller',
 'War',
 'Western'}
```

In [103]:
```
len(all_genres)
```

Out[103]:
```
21
```

There are 21 total genre categories. Now I'll add these as columns to the end of the dataset, with a preset zero value.

In [104]:
```
for genre in all_genres:
    top_gross[genre] = np.zeros(shape=top_gross.shape[0])

top_gross.head()
```

Out[104]:

	title	studio	year	title_year	movie_id	primary_title	original_title	start_year	ru
938	Avengers: Infinity War	BV	2018	Avengers: Infinity War_2018	tt4154756	Avengers: Infinity War	Avengers: Infinity War	2018	
617	Jurassic World	Uni.	2015	Jurassic World_2015	tt0369610	Jurassic World	Jurassic World	2015	
618	Furious 7	Uni.	2015	Furious 7_2015	tt2820852	Furious 7	Furious Seven	2015	
619	Avengers: Age of Ultron	BV	2015	Avengers: Age of Ultron_2015	tt2395427	Avengers: Age of Ultron	Avengers: Age of Ultron	2015	
939	Black Panther	BV	2018	Black Panther_2018	tt1825683	Black Panther	Black Panther	2018	

5 rows × 38 columns

Now we'll loop through the data and add a value of 1 into the genre column for each genre the movie has.

In [105]:
```python
for index, row in top_gross.iterrows():
    if row['genres']:
        for genre in row['genres']:
            top_gross.loc[index, genre] = 1

top_gross.head()
```

Out[105]:

	title	studio	year	title_year	movie_id	primary_title	original_title	start_year	ru
938	Avengers: Infinity War	BV	2018	Avengers: Infinity War_2018	tt4154756	Avengers: Infinity War	Avengers: Infinity War	2018	
617	Jurassic World	Uni.	2015	Jurassic World_2015	tt0369610	Jurassic World	Jurassic World	2015	
618	Furious 7	Uni.	2015	Furious 7_2015	tt2820852	Furious 7	Furious Seven	2015	
619	Avengers: Age of Ultron	BV	2015	Avengers: Age of Ultron_2015	tt2395427	Avengers: Age of Ultron	Avengers: Age of Ultron	2015	
939	Black Panther	BV	2018	Black Panther_2018	tt1825683	Black Panther	Black Panther	2018	

5 rows × 38 columns

Let's do a spot check with the Crime genre.

In [106]:
```python
crime = top_gross[top_gross['Crime']==1]
crime.head()
```

Out[106]:

	title	studio	year	title_year	movie_id	primary_title	original_title	start_year	r
618	Furious 7	Uni.	2015	Furious 7_2015	tt2820852	Furious 7	Furious Seven	2015	
853	The Fate of the Furious	Uni.	2017	The Fate of the Furious_2017	tt4630562	The Fate of the Furious	The Fate of the Furious	2017	
139	Fast Five	Uni.	2011	Fast Five_2011	tt1596343	Fast Five	Fast Five	2011	
144	Sherlock Holmes: A Game of Shadows	WB	2011	Sherlock Holmes: A Game of Shadows_2011	tt1515091	Sherlock Holmes: A Game of Shadows	Sherlock Holmes: A Game of Shadows	2011	
406	The Wolf of Wall Street	Par.	2013	The Wolf of Wall Street_2013	tt0993846	The Wolf of Wall Street	The Wolf of Wall Street	2013	

5 rows × 38 columns

In [107]:
```python
top_gross.loc[top_gross['Crime'] == 1, 'worldwide_gross'].sum()
```

Out[107]:
```
16043319228
```

As a reference, we know the total Crime worldwide gross revenue is $16 Billion. Now let's create dictionaries for the counts of genres and worldwide gross revenues by genre.

In [108]:
```python
#first create a list of the column names from the dataframe
cols = list(top_gross.columns)
```

In [109]:
```python
#Then just take the column names where the genres start
genre_cols = cols[18:]
```

In [110]:
```
#initialize the variables as dictionaries
genre_count = {}
genre_sum = {}

# Iterate through the columns associated with genres, setting the name
# column as the keys in the dictionary.
for col in genre_cols:
    # Get the total of all the genre counts where the value equaled 1
    count = np.sum(top_gross[col] == 1).sum()
    # Get the total worldwide gross revenue where the value in that ge
    g_sum = top_gross.loc[top_gross[col] == 1, 'worldwide_gross'].sum(
    # Set the values of the dictionary equal to the total of the count
    genre_count[col] = count
    # Set the value of the dictionary equal to the sum of the worldwic
    genre_sum[col] = g_sum
```

In [111]:
```
genre_count
```

Out[111]:
```
{'Comedy': 387,
 'Crime': 159,
 'Fantasy': 87,
 'Mystery': 80,
 'Horror': 103,
 'Drama': 500,
 'War': 7,
 'Family': 66,
 'History': 30,
 'Sport': 21,
 'Documentary': 7,
 'Musical': 4,
 'Thriller': 178,
 'Animation': 84,
 'Music': 31,
 'Western': 6,
 'Action': 327,
 'Adventure': 278,
 'Romance': 140,
 'Biography': 104}
```

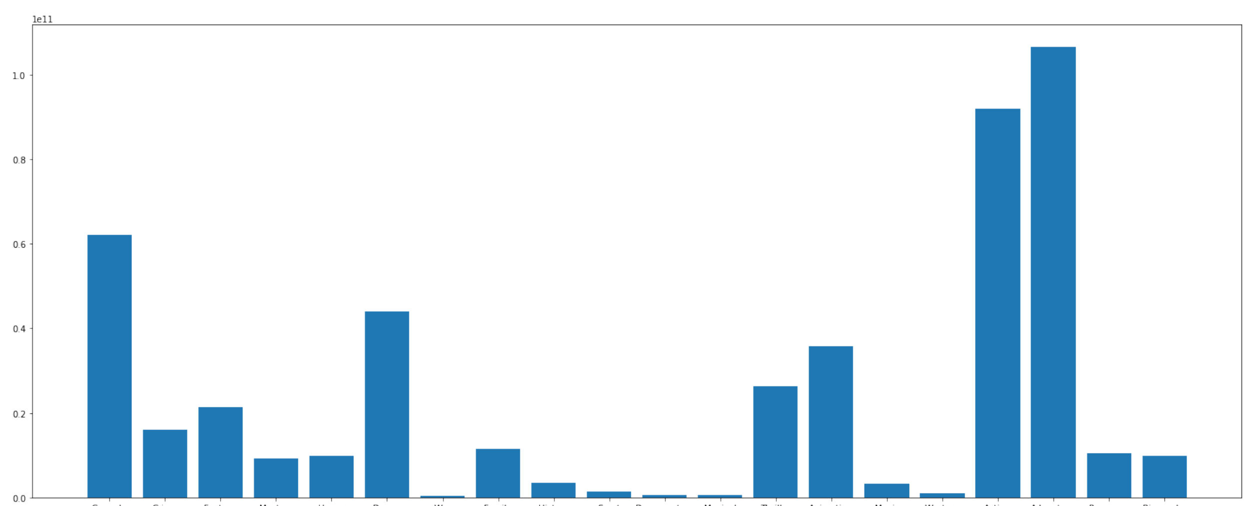
In [112]:
```
genre_sum
```

Out[112]:
```
{'Comedy': 62169129485,
 'Crime': 16043319228,
 'Fantasy': 21336877054,
 'Mystery': 9220981439,
 'Horror': 9846689252,
 'Drama': 44046387128,
 'War': 396914884,
 'Family': 11586549673,
 'History': 3359403547,
 'Sport': 1426148188,
 'Documentary': 523035998,
 'Musical': 589077623,
 'Thriller': 26279116624,
 'Animation': 35713785832,
 'Music': 3192918932,
 'Western': 980176569,
 'Action': 91879047195,
 'Adventure': 106570747616,
 'Romance': 10417029874,
 'Biography': 9889665283}
```

Now we see that the Crime genre had the same $16B of worldwide gross revenue as we calculated earlier, so I feel good about these numbers.

Now let's convert the data into lists so that we can easily plot the information.

In [113]:
```
names = list(genre_sum.keys())
values = list(genre_sum.values())

plt.figure(figsize=(25, 10))
plt.bar(range(len(genre_sum)), values, tick_label=names)
plt.show()
```



The information is helpful, but let's sort the data so it's a little easier to view.

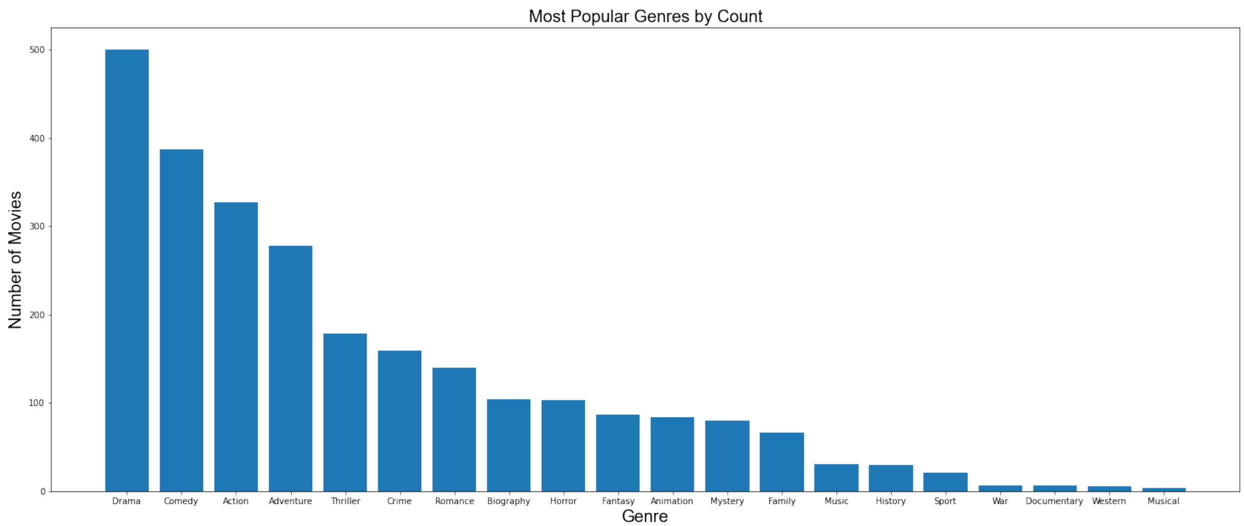
In [114]:
```
sorted_list_count = dict(sorted(genre_count.items(), key = lambda x:x[
sorted_list_count
```

Out[114]:
```
{'Drama': 500,
 'Comedy': 387,
 'Action': 327,
 'Adventure': 278,
 'Thriller': 178,
 'Crime': 159,
 'Romance': 140,
 'Biography': 104,
 'Horror': 103,
 'Fantasy': 87,
 'Animation': 84,
 'Mystery': 80,
 'Family': 66,
 'Music': 31,
 'History': 30,
 'Sport': 21,
 'War': 7,
 'Documentary': 7,
 'Western': 6,
 'Musical': 4}
```

Let's go ahead and plot that sorted data.

In [115]:

```
names = list(sorted_list_count.keys())
values = list(sorted_list_count.values())

plt.figure(figsize=(25, 10))
plt.bar(range(len(sorted_list_count)), values, tick_label=names)
font1 = {'family':'arial','color':'black','size':20}
plt.xlabel("Genre", fontdict=font1)
plt.ylabel("Number of Movies", fontdict=font1)
plt.title("Most Popular Genres by Count", fontdict=font1)
plt.show()
```



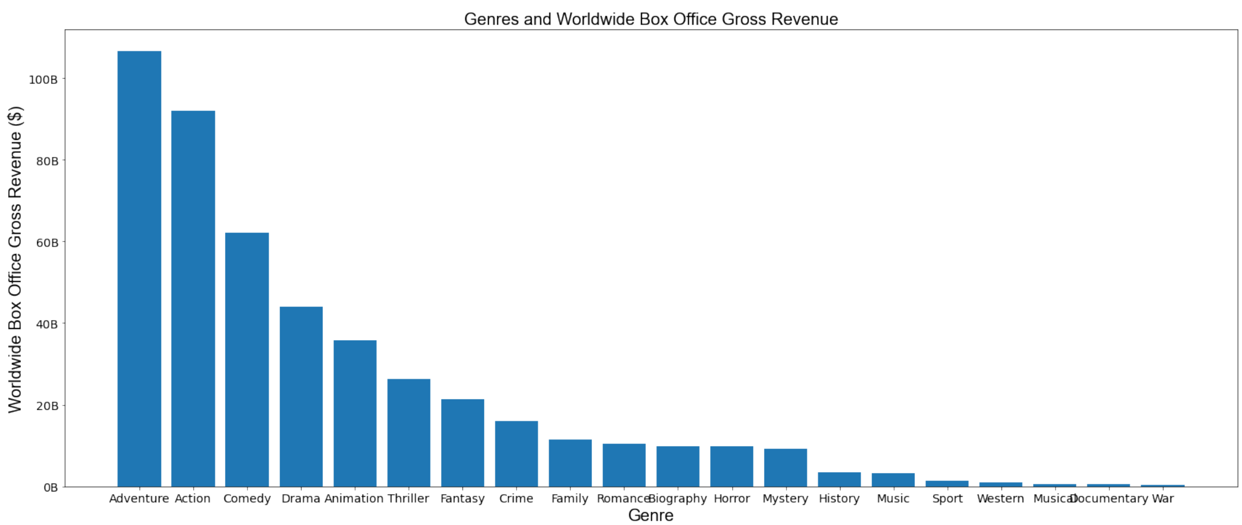
That is much easier to view the largest and smallest values. But since we're looking at revenue as a key factor, let's view the worldwide gross revenue data.

In [116]:
```
sorted_list_sum = dict(sorted(genre_sum.items(), key = lambda x:x[1],
sorted_list_sum
```

Out[116]:
```
{'Adventure': 106570747616,
 'Action': 91879047195,
 'Comedy': 62169129485,
 'Drama': 44046387128,
 'Animation': 35713785832,
 'Thriller': 26279116624,
 'Fantasy': 21336877054,
 'Crime': 16043319228,
 'Family': 11586549673,
 'Romance': 10417029874,
 'Biography': 9889665283,
 'Horror': 9846689252,
 'Mystery': 9220981439,
 'History': 3359403547,
 'Music': 3192918932,
 'Sport': 1426148188,
 'Western': 980176569,
 'Musical': 589077623,
 'Documentary': 523035998,
 'War': 396914884}
```

Now that it's sorted, let's plot the data for revenue, adjust the labels and formatting for use in the presentation.

In [117]:
```
names = list(sorted_list_sum.keys())
values = list(sorted_list_sum.values())



plt.figure(figsize=(25, 10))
plt.bar(range(len(sorted_list_sum)), values, tick_label=names)
font1 = {'family':'arial','color':'black','size':20}

plt.xlabel("Genre", fontdict=font1)
plt.ylabel("Worldwide Box Office Gross Revenue ($)", fontdict=font1)
plt.title("Genres and Worldwide Box Office Gross Revenue", fontdict=fc
current_values = plt.gca().get_yticks()
plt.gca().set_yticklabels([format(x/1000000000,'1,.0f')+'B' for x in c
plt.tick_params(axis='x', which='major', labelsize=14)
plt.tick_params(axis='y', which='major', labelsize=14)
plt.show()
```

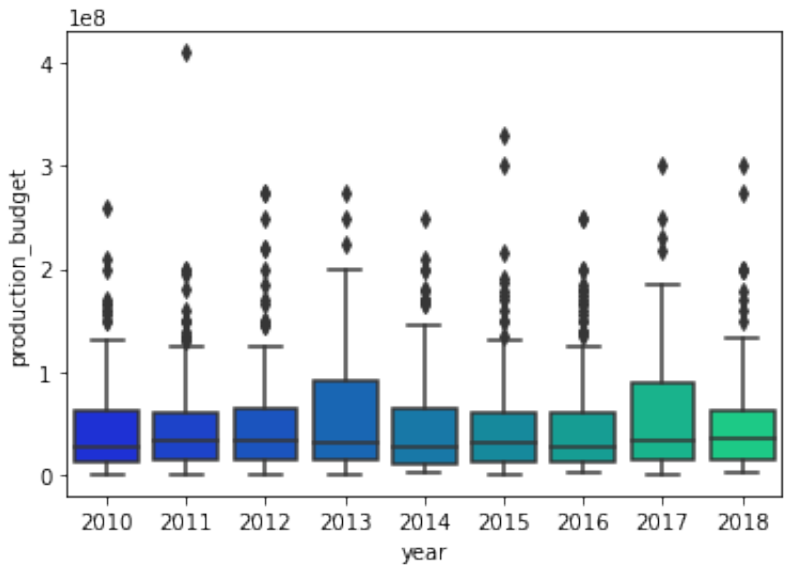


Production Budget

Next let's take a look at the production budget data, graphing that by year.

In [118]:
```
#plt.box(top_gross['production_budget'])
#plt.show

sns.boxplot(top_gross['year'], top_gross['production_budget'], palette
plt.show
```

Out[118]: <function matplotlib.pyplot.show(close=None, block=None)>



It looks like the range of budgets is generally similar, although 2013 and 2017 had some higher budgets.

Next let's look at some statistics around the dataset.

In [119]:
```
top_gross.describe()
```

Out[119]:

	year	start_year	runtime_minutes	averagerating	
count	1,025.0	1,025.0	1,025.0	1,025.0	
mean	2,013.658536585366	2,013.658536585366	109.93463414634147	6.459609756097561	146,
std	2.546920591017949	2.546920591017949	17.840346120384602	0.9422731842788351	178,0
min	2,010.0	2,010.0	41.0	1.6	
25%	2,011.0	2,011.0	97.0	5.9	
50%	2,014.0	2,014.0	107.0	6.5	
75%	2,016.0	2,016.0	120.0	7.1	
max	2,018.0	2,018.0	180.0	8.8	

8 rows × 29 columns

The production budget data ranges from $13MM to $68MM for 25th to 75th percentile, so let's create some buckets around that data.

In [120]:
```
top_gross['budget_cat'], cut_bin = pd.qcut(top_gross['production_budge
top_gross.head()
```

Out[120]:

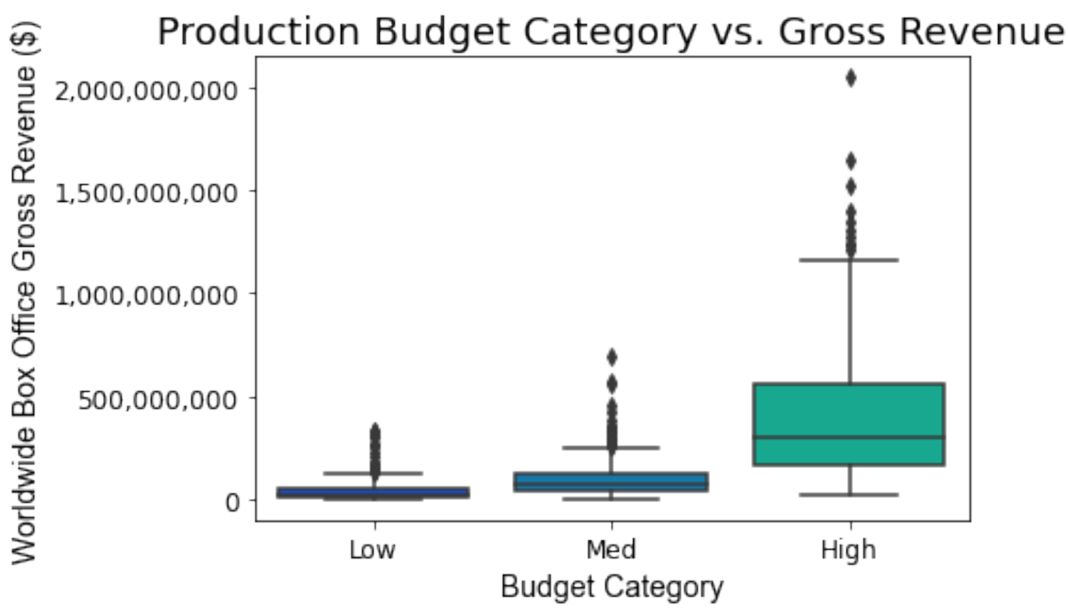
	title	studio	year	title_year	movie_id	primary_title	original_title	start_year	ru
938	Avengers: Infinity War	BV	2018	Avengers: Infinity War_2018	tt4154756	Avengers: Infinity War	Avengers: Infinity War	2018	
617	Jurassic World	Uni.	2015	Jurassic World_2015	tt0369610	Jurassic World	Jurassic World	2015	
618	Furious 7	Uni.	2015	Furious 7_2015	tt2820852	Furious 7	Furious Seven	2015	
619	Avengers: Age of Ultron	BV	2015	Avengers: Age of Ultron_2015	tt2395427	Avengers: Age of Ultron	Avengers: Age of Ultron	2015	
939	Black Panther	BV	2018	Black Panther_2018	tt1825683	Black Panther	Black Panther	2018	

5 rows × 39 columns

Now let's plot it

In [121]:
```
font1 = {'family':'arial','color':'black','size':14}


sns.boxplot(top_gross['budget_cat'], top_gross['worldwide_gross'], pal
plt.title("Production Budget Category vs. Gross Revenue", fontsize=18)
plt.xlabel("Budget Category", fontdict=font1)
plt.ylabel("Worldwide Box Office Gross Revenue ($)", fontdict=font1)
current_values = plt.gca().get_yticks()

plt.gca().set_yticklabels(['{:,.0f}'.format(x) for x in current_values
plt.tick_params(axis='x', which='major', labelsize=12)
plt.tick_params(axis='y', which='major', labelsize=12)

plt.show
```

Out[121]: <function matplotlib.pyplot.show(close=None, block=None)>



In [122]:
```
top_gross[top_gross['budget_cat']=="High"].describe()
```

Out[122]:

	year	start_year	runtime_minutes	averagerating	
count	322.0	322.0	322.0	322.0	
mean	2,013.7173913043478	2,013.7173913043478	115.9223602484472	6.593478260869565	253,
std	2.526669343427132	2.526669343427132	19.783962399975604	0.913478689256249	238,
min	2,010.0	2,010.0	72.0	3.3	
25%	2,012.0	2,012.0	100.0	6.0	
50%	2,014.0	2,014.0	114.0	6.6	
75%	2,016.0	2,016.0	130.0	7.2	
max	2,018.0	2,018.0	180.0	8.8	

8 rows × 29 columns

If we take a look at just the high budget films, the average budget is $122MM, with the min being $50MM and the max being a staggering $410MM.

Let's move onto calculating Return on Investment, definining it as Gross Revenue divided by Production Budget.

In [123]:
```
top_gross['ROI'] = top_gross['worldwide_gross']/top_gross['production_
top_gross.head()
```

Out[123]:

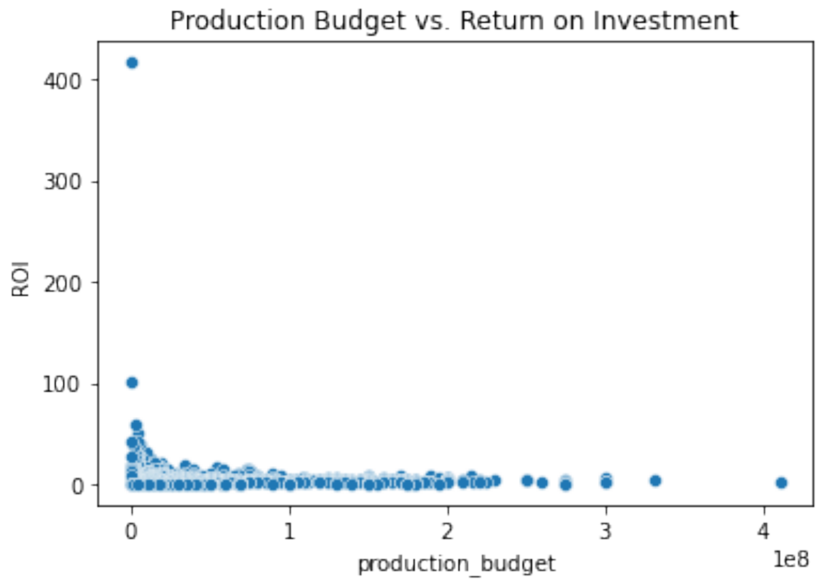
	title	studio	year	title_year	movie_id	primary_title	original_title	start_year	ru
938	Avengers: Infinity War	BV	2018	Avengers: Infinity War_2018	tt4154756	Avengers: Infinity War	Avengers: Infinity War	2018	
617	Jurassic World	Uni.	2015	Jurassic World_2015	tt0369610	Jurassic World	Jurassic World	2015	
618	Furious 7	Uni.	2015	Furious 7_2015	tt2820852	Furious 7	Furious Seven	2015	
619	Avengers: Age of Ultron	BV	2015	Avengers: Age of Ultron_2015	tt2395427	Avengers: Age of Ultron	Avengers: Age of Ultron	2015	
939	Black Panther	BV	2018	Black Panther_2018	tt1825683	Black Panther	Black Panther	2018	

5 rows × 40 columns

Let's see how the data looks when plotted.

In [124]:
```
sns.scatterplot(x='production_budget', y='ROI', data=top_gross).set(ti
plt.show
```

Out[124]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In general, the data seems to be clustered between 0 and 1, with a few large outliers. It doesn't seem like the larger budgets are really translating to exponentially higher gross revenues.

Let's add some category bins for the production data.

In [125]:
```python
bins = [0,100000000,200000000, 300000000, 400000000, 900000000]
labels = ['0-100MM', '100-200MM', '200-300MM', '300-400MM','400MM+']

top_gross['prod_budget_bin'] = pd.cut(top_gross.production_budget, bir
top_gross.head(10)
```

Out[125]:

	title	studio	year	title_year	movie_id	primary_title	original_title	start_year
938	Avengers: Infinity War	BV	2018	Avengers: Infinity War_2018	tt4154756	Avengers: Infinity War	Avengers: Infinity War	2018
617	Jurassic World	Uni.	2015	Jurassic World_2015	tt0369610	Jurassic World	Jurassic World	2015
618	Furious 7	Uni.	2015	Furious 7_2015	tt2820852	Furious 7	Furious Seven	2015
619	Avengers: Age of Ultron	BV	2015	Avengers: Age of Ultron_2015	tt2395427	Avengers: Age of Ultron	Avengers: Age of Ultron	2015
939	Black Panther	BV	2018	Black Panther_2018	tt1825683	Black Panther	Black Panther	2018
940	Jurassic World: Fallen Kingdom	Uni.	2018	Jurassic World: Fallen Kingdom_2018	tt4881806	Jurassic World: Fallen Kingdom	Jurassic World: Fallen Kingdom	2018
392	Frozen	BV	2013	Frozen_2013	tt2294629	Frozen	Frozen	2013
941	Incredibles 2	BV	2018	Incredibles 2_2018	tt3606756	Incredibles 2	Incredibles 2	2018
853	The Fate of the Furious	Uni.	2017	The Fate of the Furious_2017	tt4630562	The Fate of the Furious	The Fate of the Furious	2017
393	Iron Man 3	BV	2013	Iron Man 3_2013	tt1300854	Iron Man 3	Iron Man Three	2013

10 rows × 41 columns

Now let's sort the data by the bin for graphing purposes.

In [126]:
```
top_gross_bins = top_gross.sort_values(by="prod_budget_bin", ascending
top_gross_bins.head()
```

Out[126]:

	title	studio	year	title_year	movie_id	primary_title	original_title	start_year
616	Falcon Rising	Free	2014	Falcon Rising_2014	tt2295722	Falcon Rising	Falcon Rising	2014
984	Hereditary	A24	2018	Hereditary_2018	tt7784604	Hereditary	Hereditary	2018
567	Think Like a Man Too	SGem	2014	Think Like a Man Too_2014	tt2239832	Think Like a Man Too	Think Like a Man Too	2014
459	The Call	TriS	2013	The Call_2013	tt1911644	The Call	The Call	2013
470	Grudge Match	WB	2013	Grudge Match_2013	tt1661382	Grudge Match	Grudge Match	2013

5 rows × 41 columns

Now I'll do a groupby in order to get the sum, count and average by production budget bin.

In [127]:
```
df_groupby_prod_sum = top_gross_bins.groupby('prod_budget_bin').sum()
df_groupby_prod_sum.head()
```

Out[127]:

	year	start_year	runtime_minutes	averagerating	numvotes	produc
prod_budget_bin						
0-100MM	1695449	1695449	90,668.0	5,374.600000000002	91380835	2
100-200MM	294028	294028	17,020.0	985.8999999999995	42592312	2
200-300MM	64447	64447	4,301.0	224.89999999999998	13536865	
300-400MM	8065	8065	558.0	29.1	2018159	
400MM+	2011	2011	136.0	6.6	447624	

5 rows × 30 columns

In [128]:
```
df_groupby_prod_count = top_gross_bins.groupby(['prod_budget_bin']).co
df_groupby_prod_count.head()
```

Out[128]:

	title	studio	year	title_year	movie_id	primary_title	original_title	start_year
prod_budget_bin								
0-100MM	842	842	842	842	842	842	842	842
100-200MM	146	146	146	146	146	146	146	146
200-300MM	32	32	32	32	32	32	32	32
300-400MM	4	4	4	4	4	4	4	4
400MM+	1	1	1	1	1	1	1	1

5 rows × 40 columns

In [129]:
```
df_groupby_prod_avg = top_gross_bins.groupby('prod_budget_bin').mean()
df_groupby_prod_avg.head()
```

Out[129]:

	year	start_year	runtime_minutes	averager
prod_budget_bin				
0-100MM	2,013.5973871733968	2,013.5973871733968	107.68171021377673	6.38313539192
100-200MM	2,013.890410958904	2,013.890410958904	116.57534246575342	6.75273972602
200-300MM	2,013.96875	2,013.96875	134.40625	7.02812499999
300-400MM	2,016.25	2,016.25	139.5	
400MM+	2,011.0	2,011.0	136.0	

5 rows × 30 columns

The data looks great, now I just need to adjust the prod_budget_bin to be a column value instead of just an index.

In [130]:
```python
df_groupby_prod_sum.reset_index(inplace=True)
df_groupby_prod_sum.head()
```

Out[130]:

	prod_budget_bin	year	start_year	runtime_minutes	averagerating	numvotes	proc
0	0-100MM	1695449	1695449	90,668.0	5,374.600000000002	91380835	
1	100-200MM	294028	294028	17,020.0	985.8999999999995	42592312	
2	200-300MM	64447	64447	4,301.0	224.89999999999998	13536865	
3	300-400MM	8065	8065	558.0	29.1	2018159	
4	400MM+	2011	2011	136.0	6.6	447624	

5 rows × 31 columns

In [131]:
```python
df_groupby_prod_count.reset_index(inplace=True)
df_groupby_prod_count.head()
```

Out[131]:

	prod_budget_bin	title	studio	year	title_year	movie_id	primary_title	original_title	start_ye
0	0-100MM	842	842	842	842	842	842	842	8
1	100-200MM	146	146	146	146	146	146	146	1
2	200-300MM	32	32	32	32	32	32	32	
3	300-400MM	4	4	4	4	4	4	4	
4	400MM+	1	1	1	1	1	1	1	

5 rows × 41 columns

In [132]:
```python
df_groupby_prod_avg.reset_index(inplace=True)
df_groupby_prod_avg.head()
```

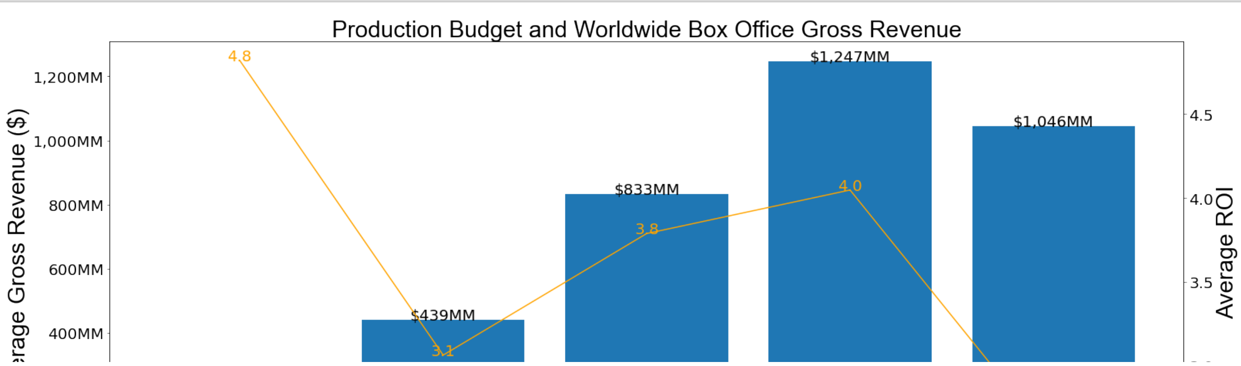
Out[132]:

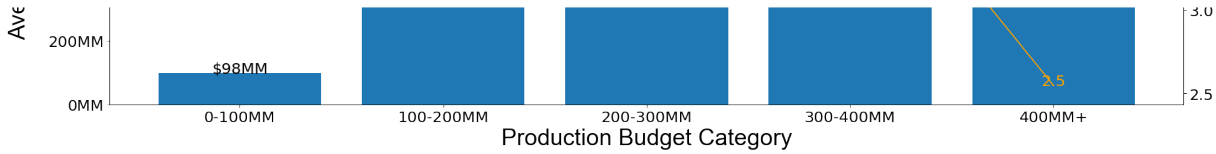
	prod_budget_bin	year	start_year	runtime_minutes	avera
0	0-100MM	2,013.5973871733968	2,013.5973871733968	107.68171021377673	6.38313539
1	100-200MM	2,013.890410958904	2,013.890410958904	116.57534246575342	6.75273972
2	200-300MM	2,013.96875	2,013.96875	134.40625	7.02812499
3	300-400MM	2,016.25	2,016.25	139.5	
4	400MM+	2,011.0	2,011.0	136.0	

5 rows × 31 columns

Now that the data is updated, it's ready to be plotted. I'll add in ROI as a secondary axis and graph it in orange. It will also be helpful to have the datapoints labeled.

In [133]:
```
plt.figure(figsize=(25, 10))

#create bar graph
plt.bar(df_groupby_prod_avg['prod_budget_bin'], df_groupby_prod_avg['w

#add labels with appropriate font and sizing
font1 = {'family':'arial','color':'black','size':30}

plt.xlabel("Production Budget Category", fontdict=font1)
plt.ylabel("Average Gross Revenue ($)", fontdict=font1)
plt.title("Production Budget and Worldwide Box Office Gross Revenue",

#add labels for the tick values in the $MM format
current_values = plt.gca().get_yticks()
plt.gca().set_yticklabels([format(x/1000000,'1,.0f')+'MM'for x in curr
plt.tick_params(axis='x', which='major', labelsize=20)
plt.tick_params(axis='y', which='major', labelsize=20)

#function to add the values in the $MM format
def addlabels_M(x,y):
    for i in range(len(x)):
        plt.text(i, y[i], "${:,.0f}MM".format(y[i]/1000000), ha = 'cen

#Label gross revenue bars
addlabels_M(df_groupby_prod_avg['prod_budget_bin'], df_groupby_prod_av

#Plot ROI on a secondary axis
df_groupby_prod_avg['ROI'].plot(secondary_y=True, color='orange')
plt.ylabel('Average ROI',fontdict=font1)
plt.tick_params(axis='y', which='major', labelsize=20)

#function to label data points for ROI with 1 decimal place
def addlabels_R(x,y):
    for i in range(len(x)):
        plt.text(i, y[i], "{:,.1f}".format(y[i]), ha = 'center', fonts

addlabels_R(df_groupby_prod_avg['prod_budget_bin'], df_groupby_prod_av

plt.show()
```





Great, now we have everything needed to put our presentation together.