Movie Analysis

Author: Eric Wehmueller

Overview

This project is the first project for Flatiron School's bootcamp program in Data Science. We are being placed into a hypothetical situation as a Data Scientist and hoping to provide value to our business for the scenario we are given.

Business Problem

Microsoft sees all the big companies creating original video content and they want to get in on the fun. They have decided to create a new movie studio, but they don't know anything about creating movies. Our job is to explore what types of films are currently doing the best at the box office. We must then translate those findings into actionable insights that the head of Microsoft's new movie studio can use to help decide what type of films to create.

```python
# Import standard packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

%matplotlib inline
```

Data Investigation

To start, we will iterate over all the data files in the "data" directory within this notebook and display the first few results. This is to get a feel for what our starting point is and what raw data we have to work with.

```
directory = 'data/' #all data files stored in 'data/' directory of this noteboo
for filename in os.listdir(directory): #iterating over the filenames, read_csv
    print(filename)
    exact_filename = directory + filename

    #adding specific cases for files that need more read_csv parameters
    if filename == 'rt.movie_info.tsv.gz':
        temp_df = pd.read_csv(exact_filename, sep='\t', header=0)
    elif filename == 'rt.reviews.tsv.gz':
         temp_df = pd.read_csv(exact_filename, sep='\t', header=0, encoding='la
    else:
        temp_df = pd.read_csv(exact_filename)

    #display(temp_df.info())
    display(temp_df.head())
    print('\n')
```

```
bom.movie_gross.csv.gz
```

	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

```
imdb.name.basics.csv.gz
```

	nconst	primary_name	birth_year	death_year	
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_ma
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,actor,writer
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinemato
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_depa

```
imdb.title.akas.csv.gz
```

	title_id	ordering	title	region	language	types	attributes	is
0	tt0369610	10	Джурасик свят	BG	bg	NaN	NaN	0.
1	tt0369610	11	Jurashikku warudo	JP	NaN	imdbDisplay	NaN	0.
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	NaN	imdbDisplay	NaN	0.
3	tt0369610	13	O Mundo dos Dinossauros	BR	NaN	NaN	short title	0.
4	tt0369610	14	Jurassic World	FR	NaN	imdbDisplay	NaN	0.

```
imdb.title.basics.csv.gz
```

	tconst	primary_title	original_title	start_year	runtime_minutes	
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Cr
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography
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,I
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,I

```
imdb.title.crew.csv.gz
```

	tconst	directors	writers
0	tt0285252	nm0899854	nm0899854
1	tt0438973	NaN	nm0175726,nm1802864
2	tt0462036	nm1940585	nm1940585
3	tt0835418	nm0151540	nm0310087,nm0841532
4	tt0878654	nm0089502,nm2291498,nm2292011	nm0284943

```
imdb.title.principals.csv.gz
```

	tconst	ordering	nconst	category	job	characters
0	tt0111414	1	nm0246005	actor	NaN	["The Man"]

	tconst	ordering	nconst	category	job	characters
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"]

```
imdb.title.ratings.csv.gz
```

	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

```
rt.movie_info.tsv.gz
```

	id	synopsis	rating	genre	director	writer
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo

	id	synopsis	rating	genre	director	writer
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper

`rt.reviews.tsv.gz`

	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

`tmdb.movies.csv.gz`

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515
3	3	[16, 35, 10751]	862	en	Toy Story	28.005
4	4	[28, 878, 12]	27205	en	Inception	27.920

`tn.movie_budgets.csv.gz`

	id	release_date	movie	production_budget	domestic_gross	worldwid
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

	id	release_date	movie	production_budget	domestic_gross	worldwid
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,3
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

Questions

Now that we've gotten a quick view of the data we have to work with, let's create and define some questions that can be answered for the purpose of providing actions that Microsoft should take. There are 2 paths I believe we can go down, depending on Microsofts goals: quality vs profit. If the priority is to make a film beloved by audiences, what genre is most likely to receive the highest ratings? If the primary focus is profit, what type of movie (genre) is most likely to be the most profitable? It will be our personal goal to provide answers for both through our data investigation.

- 1. Which genre(s) is most likely to receive the highest ratings?
- 2. Which genre(s) is most likely to be the most profitable?

(optional future idea: What is the optimal amount screen time for each (or top genres)?)

Initial Notes on Tables

Here are some notes on the initial findings of the tables from above- the tables relevant to answer these questions are underlined and marked in bold.

<u>bom.movie_gross.csv.gz</u>- Good gross/money Information

imdb.name.basics.csv.gz- Mostly just cast/workers information

imdb.title.akas.csv.gz- title ids/info

<u>imdb.title.basics.csv.gz</u>- movie basics with year and runtime. very important start point for us.

imdb.title.crew.csv.gz- directors and writers information

imdb.title.principals.csv.gz- actors information

<u>imdb.title.ratings.csv.gz</u>- average rating and number of votes for each movie ID

<u>rt.movie_info.tsv.gz</u>- contains genre, runtime, but I'm not sure where the title comes in

<u>rt.reviews.tsv.gz</u>- reviews based on movie 'id'

<u>tmdb.movies.csv.gz</u>- genre with title with ratings, basically a standalone set that gives me everything I need right off the bat

<u>tn.movie_budgets.csv.gz</u>- movie titles with production budget and domestic/worldwide gross income, might be better for determining margains

---

The RT (Rotten Tomatoes), IMDB, and TMDB tables will be good for determining ratings and audience opinion to help answer our first question.

The BOM and TN tables will be good starting points for determining profits, to help answer our second question. However, the inclusion of a production budget in the TN table leads me to believe this will provide more relevant information on the profits of a movie. An example of this might be; a movie might have made a lot of money at the box office, but the production cost may have been extremely high and the movie might have lost money overall. One would not be able to tell if such a profit loss occured, if only looking at the BOM dataset. For this reason, the TN dataset may provide us more value in this situation.

Highest Ratings Genre

Which genre(s) is most likely to receive the highest ratings?

To answer this question we will be looking at the "tmdb.movies.csv.gz" table.

```
#load data for single item
tmdb_filename = 'data/tmdb.movies.csv.gz'
print(tmdb_filename)
df_tmdb = pd.read_csv(tmdb_filename)
df_tmdb.head()
```

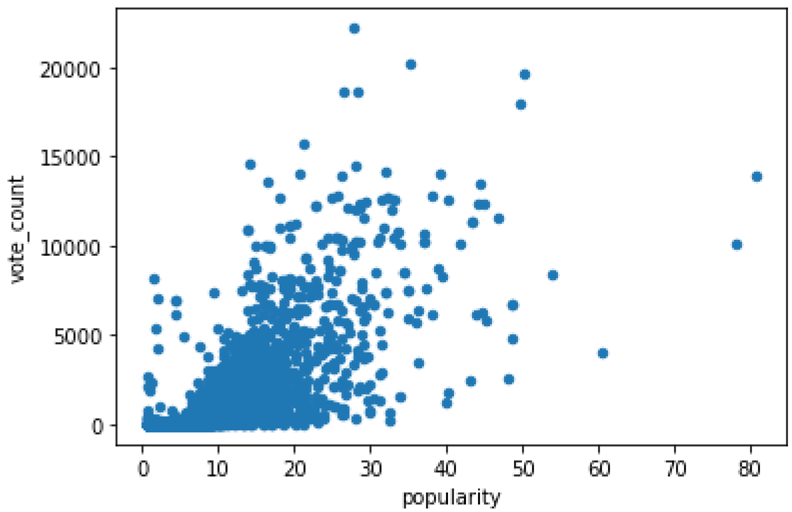
```
data/tmdb.movies.csv.gz
```

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515
3	3	[16, 35, 10751]	862	en	Toy Story	28.005
4	4	[28, 878, 12]	27205	en	Inception	27.920

```
display(df_tmdb.describe())
display(df_tmdb.value_counts('vote_count'))
df_tmdb.plot.scatter(x='popularity', y='vote_count');
```

	Unnamed: 0	id	popularity	vote_average	vote_count
count	26517.00000	26517.000000	26517.000000	26517.000000	26517.000000
mean	13258.00000	295050.153260	3.130912	5.991281	194.224837
std	7654.94288	153661.615648	4.355229	1.852946	960.961095
min	0.00000	27.000000	0.600000	0.000000	1.000000
25%	6629.00000	157851.000000	0.600000	5.000000	2.000000
50%	13258.00000	309581.000000	1.374000	6.000000	5.000000
75%	19887.00000	419542.000000	3.694000	7.000000	28.000000
max	26516.00000	608444.000000	80.773000	10.000000	22186.000000

```
vote_count
1       6541
2       3044
3       1757
4       1347
5        969
        ...
1779       1
1785       1
1787       1
1789       1
1075       1
Length: 1693, dtype: int64
```



Moving forward, there are two ways for us to consider the vote data in this table:

- consider every movie rating to be not weigh more than others (with a minimum # of votes per movie)

or

- consider every single vote from a user to be a rating vote for each of its genres

We'll investigate the first method. Unfortunately there are a decent amount of movies in this database which have a low number of vote counts. Since we are working for Microsoft, our goal is to create movies that many people will love. However, since we are an extremely large company, there probably needs to be a minimum for the amount of interest in our movie. As evidenced by this scatterplot, there is a positive correlation between the popularity and the number of votes. For the time being, we will make a minimum requirement of 5 votes and draw conclusions from that data- we can consider that a minimum for the popularity we expect. We also want to avoid drawing conclusions on movies based around a single review of 10/10, for example. Although this will filter about half of our results, there are many movies in this dataset that very few people have watched.

```
#cleaning out entries with no defined genre_ids
df_tmdb = df_tmdb[df_tmdb['genre_ids'] != '[]']
print(df_tmdb.shape[0])

#confirming our genre_ids table will no longer start with empty values
display(df_tmdb.sort_values('genre_ids', ascending=False, inplace=False).head()
```

```
24038
```

	Unnamed: 0	genre_ids	id	original_language	original_title	popula
13467	13467	[99]	261810	en	Silenced	0.600
15976	15976	[99]	441888	en	America's Greatest Prison Breaks	0.883
15977	15977	[99]	566441	en	The Hunger Games: The Phenomenon	0.882
15988	15988	[99]	562517	en	Birdman: All-Access (A View From the Wings)	0.881
15989	15989	[99]	390455	en	IOM TT	0.881

genre_ids column has been cleaned of NaNs. Now let's filter the results to only contain movies with 5 or more votes.

```
#filtering out entries with less than 5 votes
df_tmdb_5vote_min = df_tmdb[df_tmdb['vote_count'] >= 5]
print(df_tmdb_5vote_min.shape[0])
df_tmdb_5vote_min.head()
```

```
13653
```

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515
3	3	[16, 35, 10751]	862	en	Toy Story	28.005
4	4	[28, 878, 12]	27205	en	Inception	27.920

Now we will need to determine all the unique genre_ids so that we can know all data to be displayed for our genres visualization.

```
#this is meant to interact with our 'genre_ids' strings from the dataframe
#and return these numbers as a list object
def genre_string_to_list(ids_string):
    strcopy = ids_string.replace('[','').replace(']','').replace(' ','')
    return strcopy.split(',')
```

```
#determine what the possible genre_ids are in our dataset
all_genre_ids = [] #empty list start
for index, row in df_tmdb_5vote_min.iterrows(): #iterate over dataframe rows
    genre_string = row['genre_ids']
    genre_list = genre_string_to_list(genre_string) #generate a list for this r
    for genre in genre_list: #iterate over the row, adding to our cumulative li
        if genre not in all_genre_ids: #if it's not already in our list
            all_genre_ids.append(genre)

print(all_genre_ids)
```

```
['12', '14', '10751', '16', '28', '878', '35', '53', '27', '80', '18', '10749', '10
402', '9648', '36', '37', '10770', '10752', '99']
```

Now that we have a unique list for the genre_ids, we need a way to translate these into actual string values. Creating a local dictionary for this seems like a decent option. These values were found from the movie database API.

```
#Lookup from the movie database API for the genre list
genre_ids_dict = {'12':'Adventure', '28':'Action', '16':'Animation', '35':'Come
                  '18':'Drama', '10751':'Family', '14':'Fantasy', '36':'Hist
                  '9648':'Mystery', '10749':'Romance', '878':'SciFi', '10770
                  '10752':'War', '37':'Western'}
```

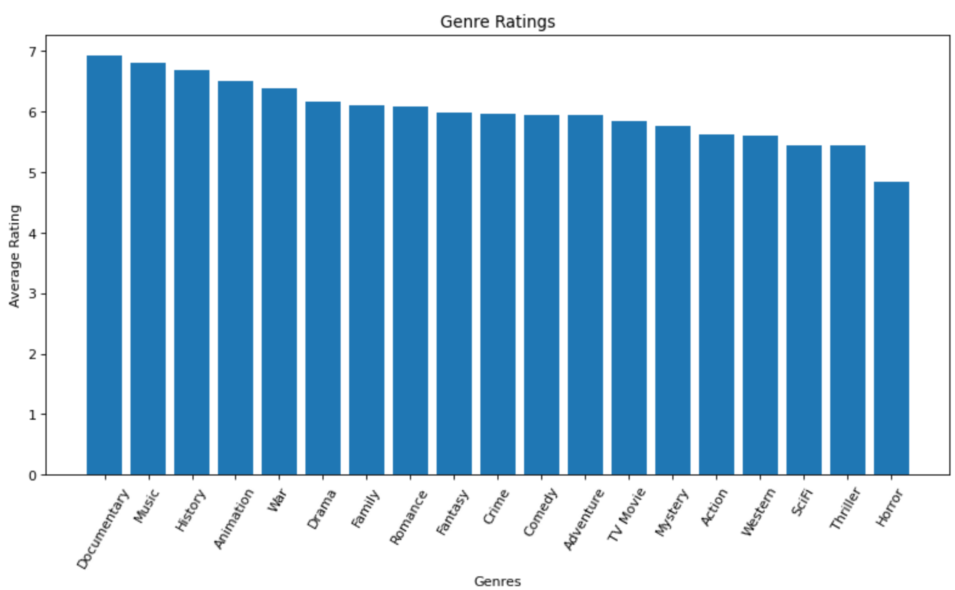
Now let's start filtering and finding values. We'll loop over all unique genre_ids that we just created, and filter for each genre_id. We can then take the mean vote_average of the remaining rows to determine the middle ground for each genre. We'll also print out standard deviation for more helpful information and to make sure this value isn't out of line for one of our genres.

```
#Find the vote averge per genre, place into a dictionary
genre_counts_dict = {}
for genre in all_genre_ids:
    temp_df = df_tmdb_5vote_min[df_tmdb_5vote_min['genre_ids'].str.contains(ger
    vote_average = temp_df['vote_average'].mean()
    genre_string = genre_ids_dict.get(genre) #lookup genre name based on ID
    genre_counts_dict[genre_string] = vote_average #add average rating for this
    print(genre_string+ ":" + str(vote_average) + "  stdev:"+str(temp_df['vote_
```

```
Adventure:5.935914811229429  stdev:1.2266412674198564
Fantasy:5.987247474747474  stdev:1.265940683754582
Family:6.094617563739376  stdev:1.0459568931907952
Animation:6.499438832772166  stdev:1.0264314309735072
Action:5.627225939269172  stdev:1.2271704636962277
SciFi:5.434268707482993  stdev:1.3807395400733886
Comedy:5.937260428410372  stdev:1.0861553038379457
Thriller:5.433604689026375  stdev:1.1057198041873642
Horror:4.840957202024851  stdev:1.1646763097731763
Crime:5.960220994475138  stdev:0.9752898881168476
Drama:6.159110130538701  stdev:1.02402781719999
Romance:6.078312537136066  stdev:0.9903693228905223
Music:6.7956896551724135  stdev:1.0483262184461106
Mystery:5.760859728506788  stdev:1.0953757152920491
History:6.673815461346633  stdev:0.9137492302926166
Western:5.592920353982301  stdev:1.313713168177411
TV Movie:5.834434561626431  stdev:0.9930527273578579
War:6.377685950413223  stdev:1.175404060713961
Documentary:6.920766590389016  stdev:0.8936576669095091
```

We have our data, and it's a little hard to tell what exactly all these numbers mean at first glace. Let's create a visualization to assist us.

```
plt.figure(num=None, figsize=(12, 6), dpi=80, facecolor='w', edgecolor='k')

#sorting dict for visualization
genre_counts_dict = dict(sorted(genre_counts_dict.items(), key=lambda item: ite
keys = genre_counts_dict.keys()
values = genre_counts_dict.values()

y_pos = np.arange(len(keys))
plt.xticks(y_pos, keys, rotation=60)
plt.bar(keys,values)
plt.title("Genre Ratings")
plt.xlabel("Genres")
plt.ylabel("Average Rating")
plt.show()
```



As we can see, Documentaries have the best average ratings across all movies with 5 or more ratings in this TMDB dataset. It is also an extremely good sign that this genre has the lowest standard deviation out of all genres, solidifying its statistic as the best genre in this case (most consistent). It is also worth mentioning that this data indicates that perhaps it is best to avoid Horror movies, as this genre clearly stands out as the worst-rated genre by a significant margain.

Best Budget for Avoiding Profit Losses

Now we will attempt to answer the question: **Which genre(s) is most likely to be the most profitable?**

To calculate profits, the "tn.movie_budgets.csv.gz" contains exactly what we need. In calculating the profits, I believe we will be able to answer another question and gain new insights:

Which budget range is best for avoiding profit losses?

We'll need to attempt to import the genre from another table to find our Most Profitable Genre(s). Unfortunately there is not a single table that contains both profits AND genre.

For now, we'll begin by investigating the "The Numbers" dataset more closely.

```
#investigate and clean this data
tn_filename = 'data/tn.movie_budgets.csv.gz'
print(tn_filename)
df_tn = pd.read_csv(tn_filename)
df_tn.sort_values('production_budget', ascending=False).head()
```

```
data/tn.movie_budgets.csv.gz
```

	id	release_date	movie	production_budget	domestic_gross	worldwi
406	7	Nov 6, 2015	The Peanuts Movie	$99,000,000	$130,178,411	$250,091,
407	8	Feb 8, 2019	The LEGO Movie 2: The Second Part	$99,000,000	$105,806,508	$190,325,
408	9	Nov 21, 2018	Robin Hood	$99,000,000	$30,824,628	$84,747,4
5326	27	Jun 1, 2007	And Then Came Love	$989,000	$8,158	$8,158
409	10	May 4, 2001	The Mummy Returns	$98,000,000	$202,007,640	$435,040,

Immediately I see that we are going to need to convert production_budget, domestic_gross, and worldwide_gross to an actual number format. Sorting by production_budget shows values out of order (see 4th entry "And Then Came Love"), so clearly this needs to be fixed. Let's write a method to convert these strings into number values for us.

```
def convert_money_to_value(money):
    money = money.replace('$', '')
    money = money.replace(',', '')
    money = int(money) #this could be an issue if we pass the integer value lin
    return money

convert_money_to_value('$99,000,000') #quick test
```

```
99000000
```

```
#Only run once***
#apply lambda function on money columns
df_tn['prod_budget_int'] = df_tn.apply(lambda x: convert_money_to_value(x['proc
df_tn['dom_gross_int'] = df_tn.apply(lambda x: convert_money_to_value(x['domest
df_tn['ww_gross_int'] = df_tn.apply(lambda x: convert_money_to_value(x['worldwi
df_tn.drop(columns = ['production_budget', 'domestic_gross', 'worldwide_gross']
df_tn.head()
```

	id	release_date	movie	prod_budget_int	dom_gross_int	ww_gross_i
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

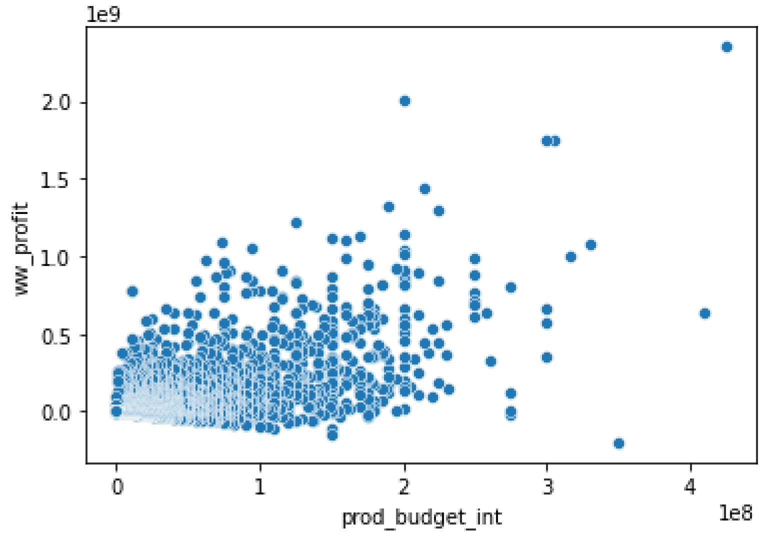
For simplicity's sake, let's just look at the most realistic profit value- worldwide, for now.

```
df_tn['ww_profit'] = df_tn['ww_gross_int'] - df_tn['prod_budget_int']
df_tn.head()
```

	id	release_date	movie	prod_budget_int	dom_gross_int	ww_gross_i
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

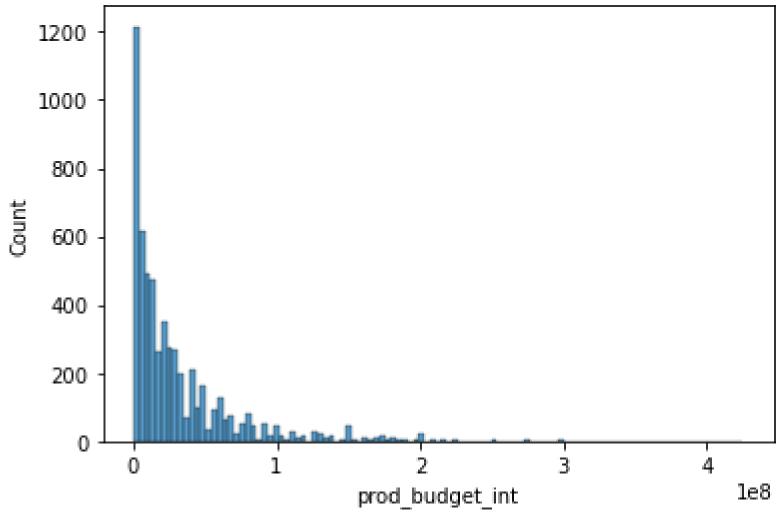
At this moment, we have everything we need to see if bigger budget movies are more prone to suffer profit losses. This would give us some imporant insight about potential budget risks for a movie. Let's visualize it.

```
#Budget vs Profit
sns.scatterplot(data=df_tn, x="prod_budget_int", y="ww_profit");
```



A generally positive correlation here, otherwise there would be no point to creating bigger and better movies. This makes sense. Note that there are a decent amount of values below the profit line (x axis).

```
sns.histplot(data=df_tn, x="prod_budget_int");
```



Based on this, I've determined some resonabily-sized buckets for us to work with.

Budget:

- <5mil
- 5-10mil
- 10-20mil
- 20-200mil
- 200mil+

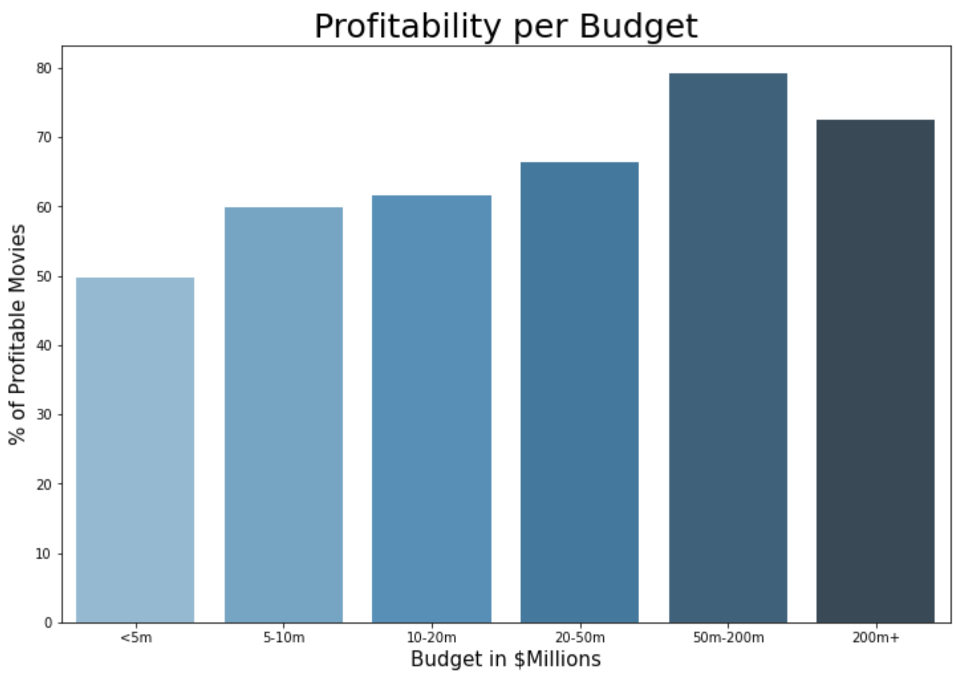
Let's create a method that will find our profit percentage, taking in a dataframe, lower limit value, and upper limit value.

```
#a method that finds our profit percentage- taking in a dataframe, lower limit
def find_profit_percentage(dataframe, lower_limit=0, upper_limit=1e100): #set t
    temp_df_total = dataframe[(dataframe['prod_budget_int'] <= upper_limit) & (
    temp_df_profit = temp_df_total[temp_df_total['ww_profit'] > 0] #profitable
    x = temp_df_profit.shape[0]/temp_df_total.shape[0] #divide the number of ro
    return x*100 #put into percentage
```

Let's put this method to use and visualize the data it produces.

```
bins= ['<5m', '5-10m', '10-20m', '20-50m', '50m-200m', '200m+']
percentages_with_profits = []
percentages_with_profits.append(find_profit_percentage(df_tn, upper_limit=50000
percentages_with_profits.append(find_profit_percentage(df_tn, lower_limit=50000
percentages_with_profits.append(find_profit_percentage(df_tn, lower_limit=10000
percentages_with_profits.append(find_profit_percentage(df_tn, lower_limit=20000
percentages_with_profits.append(find_profit_percentage(df_tn, lower_limit=50000
percentages_with_profits.append(find_profit_percentage(df_tn, lower_limit=20000
```

```
fig, ax = plt.subplots(figsize=(12, 8))
ax = sns.barplot(x= bins, y=percentages_with_profits, palette="Blues_d")
ax.set_title("Profitability per Budget", fontsize= 25)
ax.set_xlabel("Budget in $Millions", fontsize=15)
ax.set_ylabel("% of Profitable Movies", fontsize=15);
```



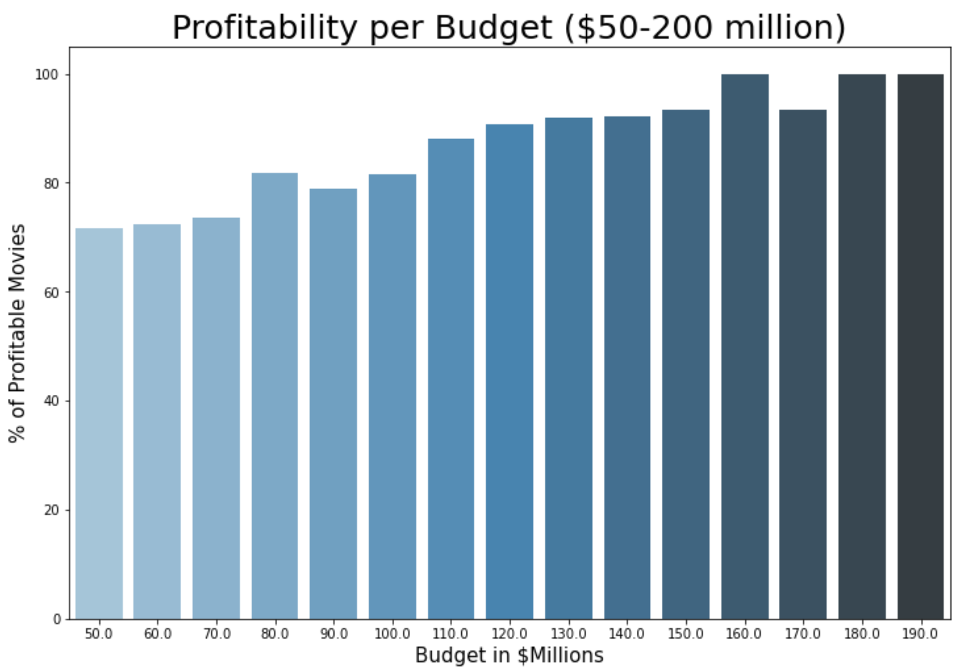
This is excellent. It appears that this trends upwards, which makes sense. Movies that are given a higher budget generally have a better chance of being profitable- to a point. We do see some falloff, with the peak being in the 50m-200m range. This is a large range, so let's take a look at this specific range more closely to see if we can find the best 10m budget range for our movie, based on the data.

We're going to need a better method. We had to call our last method many times and put in many numbers, so let's try to automate that process by giving our method a number of iterations to run on.

```
def find_profit_percentage_equal_limit_increase(dataframe, starting_lower_limit
    percentages_with_profits = []
    num_iterations = 0
    current_lower_limit = starting_lower_limit

    while num_iterations < max_iterations:
        num_iterations+=1 #loop structure for #iterations parameter
        upper_limit = current_lower_limit + increment_value #updating our upper
        temp_df_total = dataframe[(dataframe['prod_budget_int'] <= upper_limit)
        temp_df_profit = temp_df_total[temp_df_total['ww_profit'] > 0]

        denominator = temp_df_total.shape[0]
        if temp_df_total.shape[0] == 0: #avoiding division by zero
            denominator = 1
        pct = temp_df_profit.shape[0]/denominator #dividing # of profit entries
        percentages_with_profits.append(pct*100) #percentage value

        current_lower_limit = upper_limit #setting new lower limit for next loo

    return percentages_with_profits
```

Let's use this method to iterate between 50mil and 200mil, with a step size of 10k (which makes 15 iterations) and plot the results.

```
fifty_plus_bins= np.arange(50000000,200000000,10000000)
fifty_plus_bins_percentages = find_profit_percentage_equal_limit_increase(df_tr
fig, ax = plt.subplots(figsize=(12, 8))
ax = sns.barplot(x= fifty_plus_bins/1000000, y=fifty_plus_bins_percentages, pal
ax.set_title("Profitability per Budget ($50-200 million)", fontsize= 25)
ax.set_xlabel("Budget in $Millions", fontsize=15)
ax.set_ylabel("% of Profitable Movies", fontsize=15);
```



The higher up the budget goes, the less likely the movie is to fail- even when just looking at this 50-200m range. It appears that there are upward bumps in this trend at the 80-90m and 160-170m dollar budget ranges. Although the sample size is getting very small the further up we go, these seem to be "sweet spots" for budget as it relates to profitability.

Most Profitable Genre

Let's get back to our initial question, trying to find insights about the profitability per genre. In order to "import" our genres into this profits table, we are going to need to clean the "movie" titles column in TN, as well as parts of the "imdb.title.basics.csv.gz" table to maximize our number of row joins.

```
df_tn.head()
```

	id	release_date	movie	prod_budget_int	dom_gross_int	ww_gross_i
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

```
df_tn_clean = df_tn.copy() #copy because we loaded this a while ago

#cleaning titles with these methods: strip, lower, no . or : or '
#we will need to apply the same techniques to our IMDB table to join on titles
df_tn_clean['movie'] = df_tn_clean.movie.str.strip()
df_tn_clean['movie'] = df_tn_clean.movie.str.lower()
df_tn_clean['movie'] = df_tn_clean.movie.str.replace('.','').str.replace(':',''
df_tn_clean.head()
```

	id	release_date	movie	prod_budget_int	dom_gross_int	ww_gross_in
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

```
imdb_filename = 'data/imdb.title.basics.csv.gz'
df_imdb = pd.read_csv(imdb_filename)
display(df_imdb.head())
display(df_imdb.shape[0])
```

	tconst	primary_title	original_title	start_year	runtime_minutes	
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Cr
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography
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,I
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,I

```
146144
```

```
#cleaning in the same ways
df_imdb['primary_title'] = df_imdb.primary_title.str.strip()
df_imdb['primary_title'] = df_imdb.primary_title.str.lower()
df_imdb['primary_title'] = df_imdb.primary_title.str.replace('.','').str.replac

#drop nans before we do our genre calc
df_imdb = df_imdb.dropna(subset=['genres'])
df_imdb['genres'] = df_imdb['genres'].astype(str)
df_imdb.head()
```

	tconst	primary_title	original_title	start_year	runtime_minutes	
0	tt0063540	sunghursh	Sunghursh	2013	175.0	Action,Cr
1	tt0066787	one day before the rainy season	Ashad Ka Ek Din	2019	114.0	Biograph
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,I
4	tt0100275	the wandering soap opera	La Telenovela Errante	2017	80.0	Comedy,I

Now that the main title columns are cleaned for both, we should create another way to look at the multiple genres we are about to import. A good idea would be to separate each individual genre type into it's own column, providing binary data to inform the user if 1=yes, the movie is this genre or 0=no, the movie is not this genre. To start, we'll need to create a method that'll get all the unique genres found in the 'genres' column.

```
def get_all_unique_genres(dataframe, genre_col_name):
    ## Get list of unique genres
    # Join all the (unique) genres values into one big string
    var = dataframe[genre_col_name].unique()
    list_all_genres = ','.join(var)
    # Get a set of all unique genres (no duplicates)
    unique_genres = sorted(set(list_all_genres.split(',')))
    return unique_genres
```

```
def make_genre_columns(dataframe, genre_col_name='genres', drop_genres_col=True
    '''Creates a new DataFrame of a column for each genres from the genres colu
    Input:
        dataframe: Original DataFrame
        genres_col_name: Name of the column of genres (values look like "Actior
        drop_genres_col: Flag to drop the original genres column
    Returns:
        A copy of the original DataFrame with a column for each genres from the
    '''
    unique_genres = get_all_unique_genres(dataframe, genre_col_name)

    print(unique_genres)
    ## Create new columns with the genres & populate with 0 & 1
    # Make a safe copy
    new_dataframe = dataframe.copy(deep=True)
    for genre in unique_genres:
        new_dataframe[genre] = new_dataframe[genre_col_name].map(lambda val: 1
    # Drop the unused `genre_col_name` column
    if drop_genres_col:
        new_dataframe = new_dataframe.drop([genre_col_name], axis=1)
    return new_dataframe
```

```
df_imdb_genres = make_genre_columns(df_imdb, genre_col_name='genres', drop_genr

pd.options.display.max_columns = 50 #updating this for this table's 33 columns
df_imdb_genres.head()
```

```
['Action', 'Adult', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime', 'Docu
mentary', 'Drama', 'Family', 'Fantasy', 'Game-Show', 'History', 'Horror', 'Music',
'Musical', 'Mystery', 'News', 'Reality-TV', 'Romance', 'Sci-Fi', 'Short', 'Sport',
'Talk-Show', 'Thriller', 'War', 'Western']
```

	tconst	primary_title	original_title	start_year	runtime_minutes	
0	tt0063540	sunghursh	Sunghursh	2013	175.0	Action,Cr
1	tt0066787	one day before the rainy season	Ashad Ka Ek Din	2019	114.0	Biograph
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,I
4	tt0100275	the wandering soap opera	La Telenovela Errante	2017	80.0	Comedy,I

We've now succesfully added our binary columns confirming if the movie is or is not the genre, for all unique genres. Let's try merging this table with our profit data.

```
df_profit_genre = df_tn_clean.merge(df_imdb_genres, how='left', left_on='movie'
df_profit_genre = df_profit_genre.dropna(subset=['genres'])

print(df_profit_genre.shape[0])
df_profit_genre.head()
```

```
3861
```

	id	release_date	movie	prod_budget_int	dom_gross_int	ww_gross_in
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
6	7	Apr 27, 2018	avengers infinity war	300000000	678815482	2048134200

Upon the initial merge attempt, there are ~3900 entries left with valid genre values. However, I've noticed an issue- "Avatar" is marked as being a horror movie. We know this to not be the Avatar movie that did so well at the box office. It was keyed upon a foreign film 'Abata'. I think the best way to proceed is to create a concatenated column containing both the release date/start date year of the movie for both and joining. We are likely to lose more data, but we won't end up with incorrect joined genre data thinking Horror movies did better than they actually did (with our Abata example).

```
#convert start_year to int from float column type- it appeared to be float type
df_imdb_genres['release_year_imdb'] = df_imdb_genres.apply(lambda x: str(x['sta
df_imdb_genres.head()

#now we are going to need to extract the year from the release_date column of t
#final 4 string characters of each 'release_date' should suffice
df_tn_clean['release_year_tn'] =  df_tn_clean.apply(lambda x: x['release_date']
print(sorted(df_tn_clean['release_year_tn'].unique())) #confirmation that we do

#now let's create the two columns we are going to 'join' on
#year+movie title concatenation
df_tn_clean['year_plus_movie_tn'] = df_tn_clean['release_year_tn'] + df_tn_clea
df_imdb_genres['year_plus_movie_imdb'] = df_imdb_genres['release_year_imdb'] +

display(df_tn_clean.head())
display(df_imdb_genres.head())
```

```
['1915', '1916', '1920', '1925', '1927', '1929', '1930', '1931', '1933', '1934', '1
935', '1936', '1937', '1938', '1939', '1940', '1941', '1942', '1943', '1944', '194
5', '1946', '1947', '1948', '1949', '1950', '1951', '1952', '1953', '1954', '1955',
'1956', '1957', '1958', '1959', '1960', '1961', '1962', '1963', '1964', '1965', '19
66', '1967', '1968', '1969', '1970', '1971', '1972', '1973', '1974', '1975', '197
6', '1977', '1978', '1979', '1980', '1981', '1982', '1983', '1984', '1985', '1986',
'1987', '1988', '1989', '1990', '1991', '1992', '1993', '1994', '1995', '1996', '19
97', '1998', '1999', '2000', '2001', '2002', '2003', '2004', '2005', '2006', '200
7', '2008', '2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017',
'2018', '2019', '2020']
```

	id	release_date	movie	prod_budget_int	dom_gross_int	ww_gross_in
0	1	Dec 18, 2009	avatar	425000000	760507625	2776345279
1	2	May 20, 2011	pirates of the caribbean on stranger tides	410600000	241063875	1045663875

	id	release_date	movie	prod_budget_int	dom_gross_int	ww_gross_in
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

	tconst	primary_title	original_title	start_year	runtime_minutes	
0	tt0063540	sunghursh	Sunghursh	2013	175.0	Action,Cr
1	tt0066787	one day before the rainy season	Ashad Ka Ek Din	2019	114.0	Biograph
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,I
4	tt0100275	the wandering soap opera	La Telenovela Errante	2017	80.0	Comedy,I

Awesome. It looks like we now have something to join on more accurately. We extracted the release_year from both the IMDB and TN tables and concatenated it with their respective cleaned titles. Let's try left joining on our TN table once more. Again, the purpose of this is to "import" our genres into profit data that we already have.

```
df_profit_genre_concat = df_tn_clean.merge(df_imdb_genres, how='left', left_on=
df_profit_genre_concat = df_profit_genre_concat.dropna(subset=['genres'])

print(df_profit_genre_concat.shape[0])
df_profit_genre_concat.head()
```

```
1595
```

	id	release_date	movie	prod_budget_int	dom_gross_int	ww_gross_in
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
6	7	Apr 27, 2018	avengers infinity war	300000000	678815482	2048134200
8	9	Nov 17, 2017	justice league	300000000	229024295	655945209

There are 1595 entires now. As expected, this shrunk down our datasize by over half once again. However, there are many overlaps on movie titles and we don't want to get incorrect genre data imported. We now have a table with both profit and genre data in each row. We can start visualizing to extract useful information for our investors.

```
df_profit_genre_concat.describe()
```

	id	prod_budget_int	dom_gross_int	ww_gross_int	ww_pr
count	1595.000000	1.595000e+03	1.595000e+03	1.595000e+03	1.595000e+
mean	50.555486	4.419429e+07	5.533940e+07	1.380743e+08	9.387997e+
std	28.777909	5.562454e+07	8.390396e+07	2.307890e+08	1.901067e+
min	1.000000	1.500000e+04	0.000000e+00	0.000000e+00	-2.002376e
25%	26.000000	8.000000e+06	2.518277e+06	7.306242e+06	-9.080450e
50%	51.000000	2.200000e+07	2.683950e+07	4.985846e+07	2.185381e+
75%	75.000000	5.500000e+07	6.723922e+07	1.543725e+08	1.008324e+
max	100.000000	4.106000e+08	7.000596e+08	2.048134e+09	1.748134e+

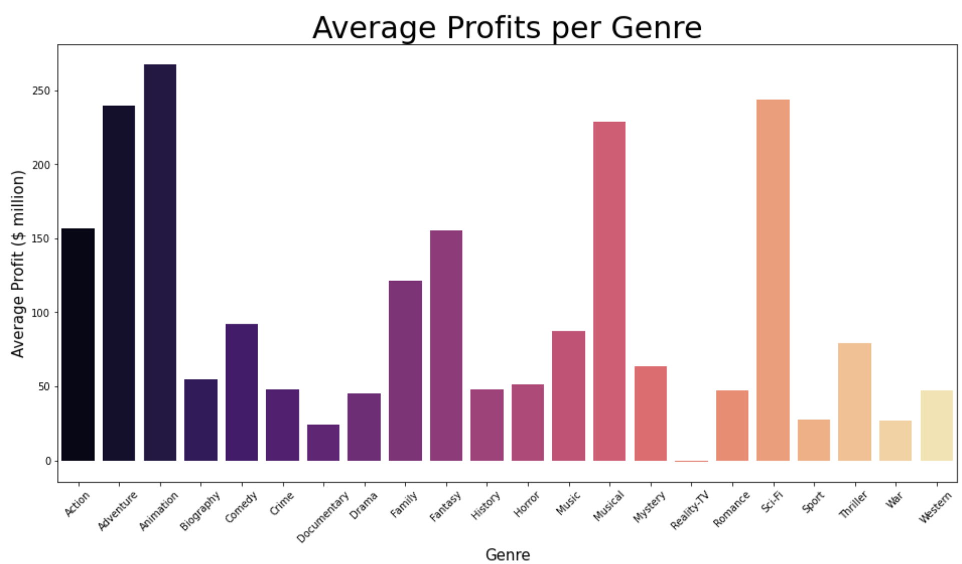
As we can see from this describe call, some of the genres have a zero max, meaning this column consists entirely of zeros. We should probably remove these genre columns from our unique_genre_names list if we are going to iterate over it, so that we avoid any NaNs.

```
unique_genre_names = get_all_unique_genres(df_imdb, 'genres')

#No genres for these with profit data, no need to calc means and medians for th
genres_to_remove = ['Adult', 'Game-Show', 'News', 'Short', 'Talk-Show']
for genre in genres_to_remove:
    unique_genre_names.remove(genre)

genre_means = []
genre_medians = []
#for each genre in names--- filter and find the average and median profits
for genre in unique_genre_names:
    temp_df_test = df_profit_genre_concat[(df_profit_genre_concat[genre] == 1)]
    genre_means.append(temp_df_test['ww_profit'].mean()/1000000) #dividing by 1
    genre_medians.append(temp_df_test['ww_profit'].median()/1000000)

print(unique_genre_names)
print(genre_means)
print(genre_medians)
```

```
['Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime', 'Documentary',
'Drama', 'Family', 'Fantasy', 'History', 'Horror', 'Music', 'Musical', 'Mystery',
'Reality-TV', 'Romance', 'Sci-Fi', 'Sport', 'Thriller', 'War', 'Western']
[156.48882969871795, 239.59186351373629, 267.73512306542057, 54.4482134057971, 92.1
7199526481481, 47.76751733054394, 24.49314892982456, 45.020343217847774, 121.109661
12244898, 155.24786890000001, 48.04698073170732, 51.359353804123714, 87.22141514062
5, 228.6135039, 63.807046916666664, -1.0, 47.093505570707066, 244.08030169064747, 2
7.435814058823528, 79.16955595255475, 27.183357722222222, 47.18224416666666]
[50.972756, 125.2320115, 178.84793, 17.7494135, 29.8884195, 12.966716, 1.495262, 1
1.1889285, 37.0107085, 48.2334175, 11.187026, 17.9662105, 11.7504075, 17.2763375, 3
2.05427, -1.0, 17.135547, 111.68179, 12.0427885, 26.294152, -2.102223, -1.620152]
```

We now have mean and median profit data per genre- exactly what we set out to do originally. Let's visualize this and see what conclusions we can draw.

```
fig, ax = plt.subplots(figsize=(16, 8))
ax = sns.barplot(x= unique_genre_names, y=genre_means, palette="magma")

plt.xticks(rotation=45)
plt.title("Average Profits per Genre", fontsize=30)
ax.set_xlabel("Genre", fontsize=15)
ax.set_ylabel("Average Profit ($ million)", fontsize=15);
```

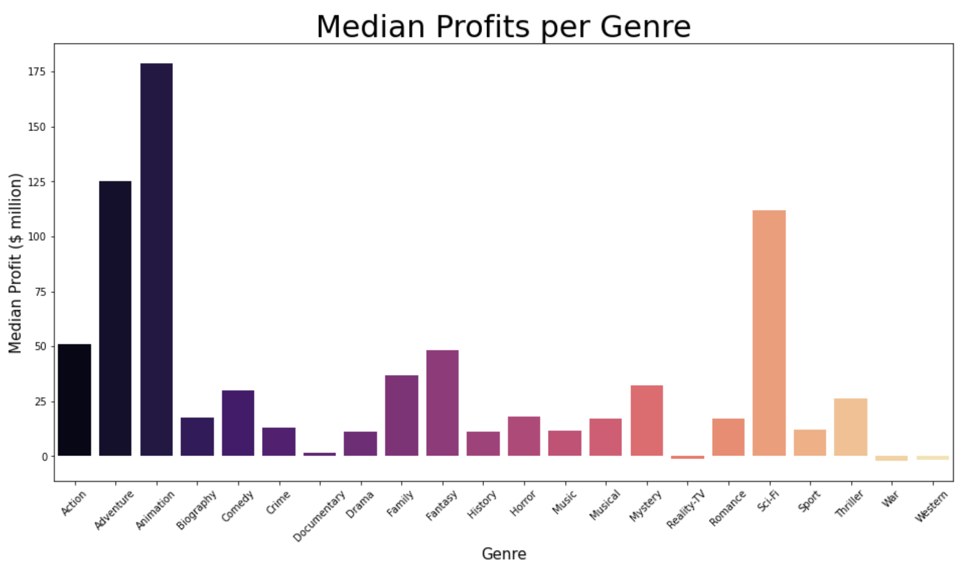


Looking at our top 4 genres, we have several clear winners for "Average Profit"

- 1. Animation
- 2. Sci-Fi
- 3. Adventure
- 4. Musical

This is very useful information. However, a single movie in one specific genre could have been an enormous hit and could be potentially skewing these results. Let's visualize the medians per genre to see if this is the case.

```
fig, ax = plt.subplots(figsize=(16, 8))
ax = sns.barplot(x= unique_genre_names, y=genre_medians, palette="magma")

plt.xticks(rotation=45)
plt.title("Median Profits per Genre", fontsize=30)
ax.set_xlabel("Genre", fontsize=15)
ax.set_ylabel("Median Profit ($ million)", fontsize=15);
```



As we can see, our top 4 for "Median Profit" has become

- 1. Animation
- 2. Adventure
- 3. Sci-Fi
- 4. Action

We have essentially the same top 3, but what in the world happened to our "musical" category we were looking at just a moment ago? From the visualization, we can see that calculating the median of these movies revealed what we believed to be true; your "average" popularity musical film does not seem to do so well as far as profits are concerned. Some outliers in this category are skewing the mean heavily.

Conclusions

Our goal for this project was to provide useful insights for investors, by answering these questions:

- 1. Which genre(s) is most likely to receive the highest ratings?
- 2. Which budget range is best for avoiding profit losses?
- 3. Which genre(s) is most likely to be the most profitable?

---

1. From our analysis, our top 4 highest-rated genres are (in order):

- **Documentary**
- **Music**
- **History**
- **Animation**

Unfortunately, there are some limitations to this. This conclusion was only determined based on a single dataset. To add more robustness to this conclusion, I would recommend that another ratings dataset be cleaned and visualized; the IMDB dataset would be excellent for this. Additionally, there was a minimum ratings filter set to 5 for this visualization. In doing so, this removed about half our datapoints. However, since we are an extremely large company (Microsoft) trying to get into the movies space, we are likely targeting larger scale movies. If our movies are recieving less than 5 ratings, we might be in trouble. So, perhaps the data we filtered out is not as relevant to us and there is no issue with our methodology here.

---

2. From our analysis, I believe the best budget ranges are between either **80-90 million dollars or 160-170million dollars**. In general, we observed that the higher up the budget goes, the less likely the movie is to flop (net profit loss)- up until the 200+ million dollar range. It appears that there are upward bumps in this trend at the 80-90m and 160-170m dollar budget ranges, which indicates

these ranges are positively profiting more on average than expected. Although the sample size is getting very small the further up we go, these seem to be "sweet spots" for budget as it relates to profitability.

---

3. From our analysis, top 3 most profitable genres are clearly:

- **Animation**
- **Adventure**
- **Sci-Fi**

With our initial findings using the mean, it appeared that the Musical genre looked promising at first. Upon inspection of median values, this turned out to not be the case. This is a great lesson in why the median can be so valuable in situations like this- a single movie in one specific genre could have been an enormous hit with massive profits and could be potentially skewing these results. This is exactly what happened for the Musical genre.
Both the means and medians of **Animation, Adventure, and Sci-Fi** are very high above all other genres, indicating that these are generally the most profitable genres of movies to make, based on our data. I believe it is also worth noting that **Reality-TV, War, and Western** movies lose money most of the time (50% = median, this profit value is negative). It is probably best to avoid these genres if possible.

Future Improvements

- Return on investment data would be extremely useful for investors. Unfortunately, my data has limitations- I only calculated if movies for a certain genre at least broke even (aka technically profitable). If a movie budget was 100 million dollars and we only made a profit of one dollar, it's TECHNICALLY still profitable, but that is probably not considered a successful box office killing. Return on investment would be more relevant for scenarios like this.
- The dataset I used for ratings was somewhat limiting. Investigating another dataset such as the IMDB would provide another perspective to potentially confirm this data. Different types of users browse, use, and give different ratings on different sites. There's no telling what the differences in preference might be.