

# Final Project Submission

Please fill out:

- Student name: **David Githaiga Maina**
- Student pace: **Part time**
- Scheduled project review date/time:
- Instructor name:
- Blog post URL:

## Research Questions

1. Which genres are most popular for all-time?
2. Is movie production budget directly related to the popularity/gross revenue?
3. Which cast and crew are associated with the most popular movies?

```
In [45]: #import required modules  
import gzip  
import requests  
import pandas as pd  
import numpy as np  
import json  
import matplotlib.pyplot as plt  
from datetime import datetime
```

```
In [62]: plt.style.use('seaborn-v0_8')
```

```
In [2]: #Load tmdb.movies.csv.gz file
with gzip.open('zippedData/tmdb.movies.csv.gz', 'r') as f:
    tmdb_movies = pd.read_csv(f, index_col= 0)
```

```
In [3]: tmdb_movies.head()
```

```
Out[3]:
```

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average	vote_count
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	7.7	10788
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon	7.7	7610
2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2	6.8	12368
3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story	7.9	10174
4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception	8.3	22186

```
In [4]: #filter to take only movies subset with vote count that is higher than the average
tmdb_movies = tmdb_movies[tmdb_movies['vote_count'] > tmdb_movies['vote_count'].mean()]
```

```
In [5]: #function for loading api key file that will be used to request tmdb for genres ids to use with the tmdb
def get_keys(path):
    with open(path) as f:
        return json.load(f)

keys = get_keys('/Users/dave/.secret/tmdb_api.json')
api_key = keys['api_key']
```

```
In [6]: #Request tmdb for genres ids
url = "https://api.themoviedb.org/3/genre/movie/list?language=en"

headers = {
    "accept": "application/json",
    "Authorization": "Bearer {}".format(api_key)
}

response = requests.get(url, headers=headers)
```

```
In [7]: #load the json format of the response from the api call into a dataframe
tmdb_genre_ids = pd.DataFrame(response.json()['genres'])
tmdb_genre_ids.head()
```

```
Out[7]:
```

	id	name
0	28	Action
1	12	Adventure
2	16	Animation
3	35	Comedy
4	80	Crime

```
In [8]: #Define a function that processes tmdb_movies['genre_ids'] values, which are currently of object dtype,
def process_genre_ids(index):
    ids = list(index.replace("[", "").replace("]", "").split(','))
    return [int(num.strip()) for num in ids if num.strip()]

# Apply the function to every element in tmdb_movies['genre_ids']
tmdb_movies['genre_ids'] = tmdb_movies['genre_ids'].apply(process_genre_ids)
```

```
In [9]: #Create a new column in tmdb.movies that will hold translated values of genre names from the genre_ids
# Create a function to extract the genre names
def get_genre_names(genre_ids):
    genre_names = []
    for genre_id in genre_ids:
        if genre_id in tmdb_genre_ids['id'].values:
            genre_index = tmdb_genre_ids[tmdb_genre_ids['id'] == genre_id].index[0]
            genre_names.append(tmdb_genre_ids.loc[genre_index, 'name'])
    return genre_names

# Apply the function to the 'genre_ids' column and assign the result to 'genre' column
tmdb_movies['genre'] = tmdb_movies['genre_ids'].apply(lambda x: get_genre_names(x))
```

In [10]: `tmdb_movies.head()`

Out[10]:

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average	vote_count	genre
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	7.7	10788	[Adventure, Fantasy, Family]
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon	7.7	7610	[Fantasy, Adventure, Animation, Family]
2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2	6.8	12368	[Adventure, Action, Science Fiction]
3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story	7.9	10174	[Animation, Comedy, Family]
4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception	8.3	22186	[Action, Science Fiction, Adventure]

```
In [11]: #Load imdb.title.ratings.csv.gz
with gzip.open('zippedData/imdb.title.ratings.csv.gz', 'r') as f:
    imdb_movies_ratings = pd.read_csv(f)
```

```
In [12]: imdb_movies_ratings.head()
```

```
Out[12]:
```

	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

```
In [13]: #Load imdb.title.basics.csv.gz
with gzip.open('zippedData/imdb.title.basics.csv.gz', 'r') as f:
    imdb_title_basics = pd.read_csv(f)
```

```
In [14]: imdb_title_basics.head()
```

```
Out[14]:
```

	tconst	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action, Crime, Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography, Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy, Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy, Drama, Fantasy

```
In [15]: #concatenate imdb_title_basics and imdb.title.ratings on the 'tconst' column
imdb_movies = pd.concat([imdb_title_basics.set_index('tconst'), imdb_movies_ratings.set_index('tconst')])
```

In [16]: `imdb_movies.head()`

Out[16]:

	primary_title	original_title	start_year	runtime_minutes	genres	averagerating	numvotes
tt0063540	Sunghursh	Sunghursh	2013	175.0	Action, Crime, Drama	7.0	77.0
tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography, Drama	7.2	43.0
tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama	6.9	4517.0
tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy, Drama	6.1	13.0
tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy, Drama, Fantasy	6.5	119.0

In [17]: `imdb_movies.shape`

Out[17]: (146144, 7)

In [18]: `imdb_movies = imdb_movies[imdb_movies['numvotes'] > imdb_movies['numvotes'].mean()]`

In [19]: `imdb_movies.shape`

Out[19]: (4751, 7)

```
In [20]: #aggregate imdb movies by genre and find the averages of ratings and number of votes per group
imdb_genre_aggregate = imdb_movies[['genres', 'averagerating', 'numvotes']].groupby('genres').agg(
    mean_average_rating=('averagerating', 'mean'),
    mean_num_votes=('numvotes', 'mean')
)
#sort values by average ratings
imdb_genre_aggregate.sort_values(by= imdb_genre_aggregate.columns[0], ascending= False)
```

Out[20]:

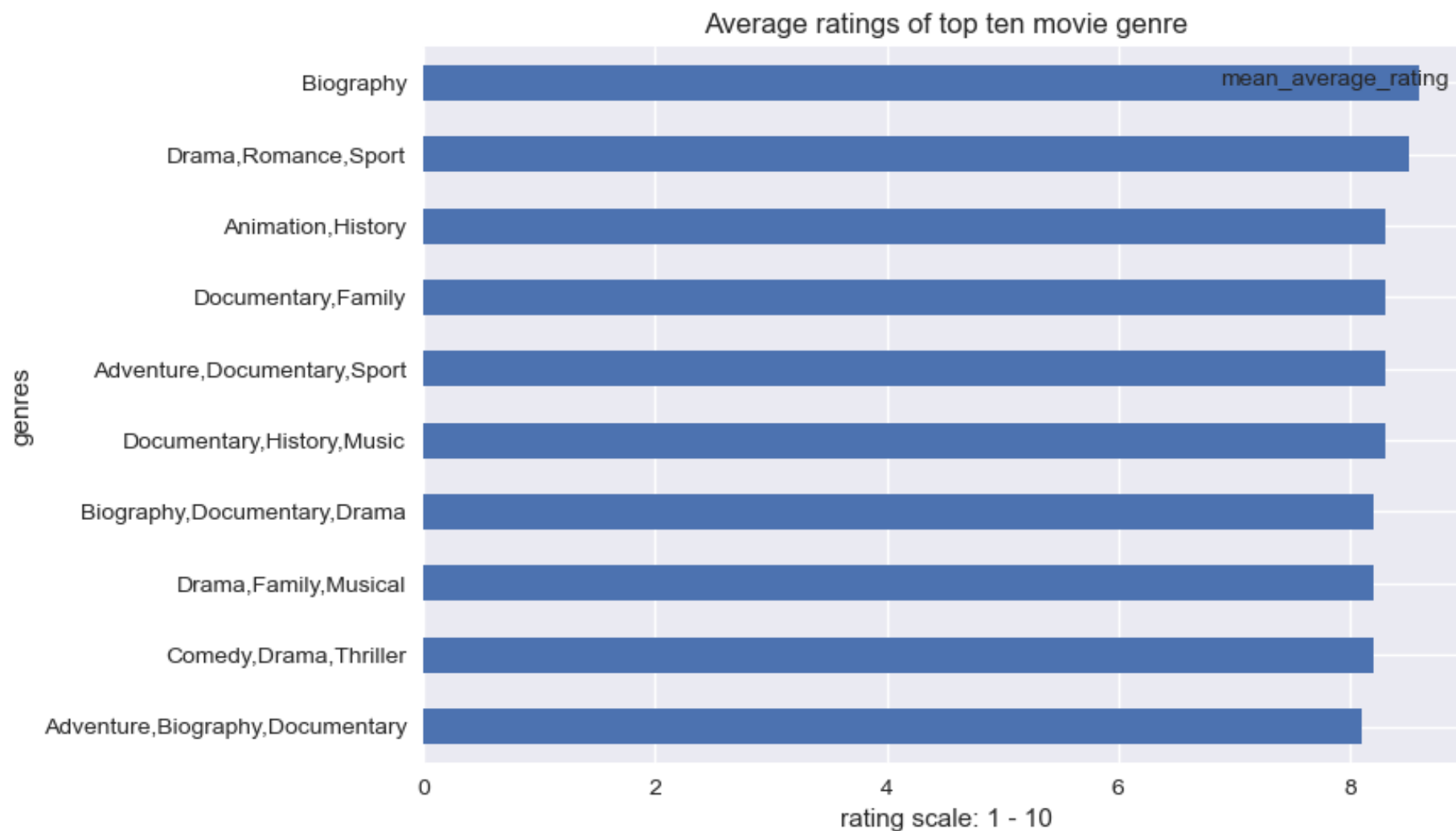
	mean_average_rating	mean_num_votes
genres		
Biography	8.6	3879.0
Drama,Romance,Sport	8.5	4098.0
Animation,History	8.3	7451.0
Documentary,Family	8.3	3820.0
Adventure,Documentary,Sport	8.3	7288.0
...	...	...
Action,Adventure,History	4.5	7523.0
Crime,Romance,Thriller	4.3	8782.0
Comedy,Romance,Sci-Fi	4.2	4953.0
Comedy,Family,Sci-Fi	2.6	4552.0
Crime,Mystery	1.5	26723.0

383 rows × 2 columns

```
In [69]: #plot for the top ten (10) most rated genres on imdb movies
imdb_genre_aggregate.sort_values(by= imdb_genre_aggregate.columns[0], ascending= False, inplace= True)
imdb_genre_aggregate[:10].plot(kind= 'barh', y=imdb_genre_aggregate.columns[0])
```



```
plt.gca().invert_yaxis()  
plt.title('Average ratings of top ten movie genre')  
plt.xlabel('rating scale: 1 - 10');
```



```
In [22]: #Similarly to tmdb.movies, also aggregate tmdb.movies by genre and find the averages of ratings and numb
#First convert the values of the tmdb genre column into a string objects instead of lists so the column
#as that of the imdb.movies genre column
tmdb_movies['genre'] = tmdb_movies['genre'].apply(lambda x: ','.join(x))

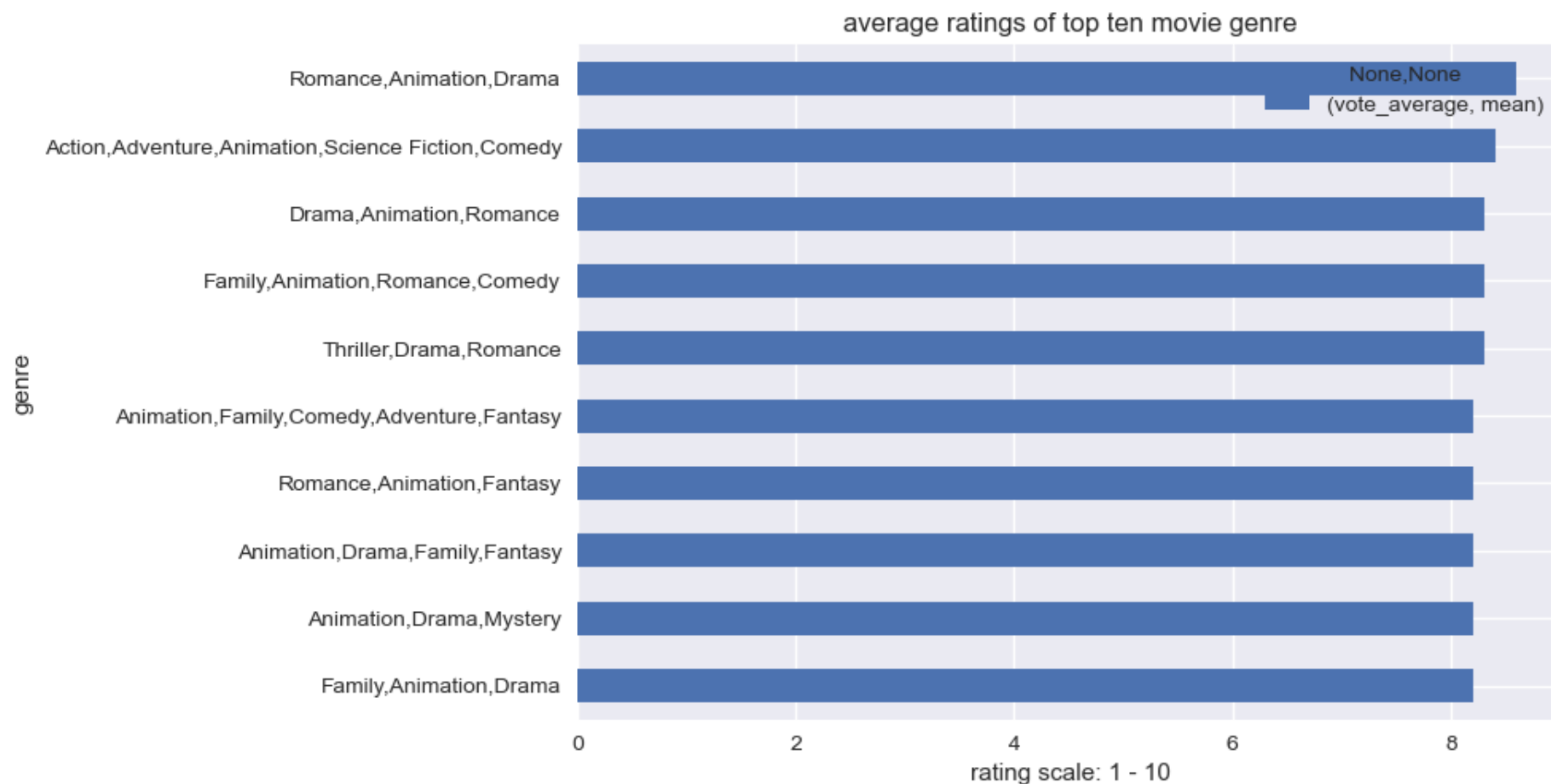
#group by genre
tmdb_genre_aggregate = tmdb_movies[['vote_average', 'vote_count', 'genre']].groupby('genre').agg(
    {'vote_average': ['mean'], 'vote_count': ['mean']})
tmdb_genre_aggregate.sort_values(by= tmdb_genre_aggregate.columns[0], ascending= False)
```

Out [22]:

	vote_average	vote_count
	mean	mean
genre		
Romance,Animation,Drama	8.6	4161.0
Action,Adventure,Animation,Science Fiction,Comedy	8.4	4048.0
Drama,Animation,Romance	8.3	1034.0
Family,Animation,Romance,Comedy	8.3	684.0
Thriller,Drama,Romance	8.3	1213.0
...	...	...
Action,TV Movie,Science Fiction,Comedy,Adventure	4.3	246.0
Fantasy,Horror,Mystery,Thriller	4.2	2475.0
Action,Horror	4.1	238.0
Mystery,Adventure,Comedy,Crime	4.1	217.0
Science Fiction,TV Movie,Action,Adventure,Comedy	3.8	873.0

822 rows × 2 columns

```
In [70]: #Similarly as done for the imdb data, also plot for the top ten (10) most rated genres on tmdb movies
tmdb_genre_aggregate.sort_values(by= tmdb_genre_aggregate.columns[0], ascending= False, inplace= True)
tmdb_genre_aggregate[:10].plot(kind= 'barh', y= tmdb_genre_aggregate.columns[0])
plt.gca().invert_yaxis()
plt.title('average ratings of top ten movie genre')
plt.xlabel('rating scale: 1 - 10');
```



```
In [28]: #Check to see if there are genres whose ratings have a trend over the years  
#First group by year of release  
imdb_rating_by_year = imdb_movies[['start_year', 'genres', 'averagerating', 'numvotes']].groupby(['genres',  
imdb_rating_by_year.head()
```

Out[28]:

		averagerating	numvotes
		mean	mean
genres	start_year		
Action	2010	3.850000	8532.000000
	2011	5.500000	7592.000000
	2012	5.650000	38569.000000
	2013	5.333333	6815.666667
	2015	7.200000	21092.000000

```
In [29]: #The same with tmdb movies
tmdb_movies['release_date'] = pd.to_datetime(tmdb_movies['release_date'])
tmdb_rating_by_year = tmdb_movies[['release_date', 'vote_average', 'vote_count', 'genre']].groupby(['genre', 'release_date']).head()
```

Out[29]:

		vote_average	vote_count
		mean	mean
genre	release_date		
Action	2012-01-01	6.0	713.0
	2012-11-02	5.3	573.0
	2015-02-27	4.7	223.0
	2015-07-14	4.6	213.0
	2015-07-31	7.1	5242.0

```
In [66]: pivot_imdb = imdb_rating_by_year.pivot_table(index='start_year', columns='genres', values='averagerating')

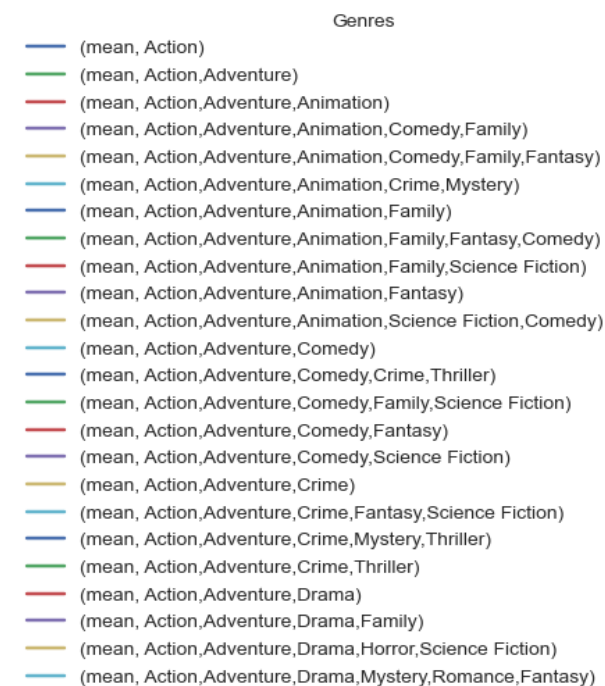
# Plot the data
pivot_imdb.plot(figsize=(10, 6))
plt.title('Mean Average Rating by Year for Each Genre')
plt.xlabel('Year')
plt.ylabel('Mean Average Rating')
plt.legend(title='Genres', loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```

Genres

- (mean, Action)
- (mean, Action, Adventure)
- (mean, Action, Adventure, Animation)
- (mean, Action, Adventure, Biography)
- (mean, Action, Adventure, Comedy)
- (mean, Action, Adventure, Crime)
- (mean, Action, Adventure, Documentary)
- (mean, Action, Adventure, Drama)
- (mean, Action, Adventure, Family)
- (mean, Action, Adventure, Fantasy)
- (mean, Action, Adventure, History)
- (mean, Action, Adventure, Horror)
- (mean, Action, Adventure, Mystery)
- (mean, Action, Adventure, Romance)
- (mean, Action, Adventure, Sci-Fi)
- (mean, Action, Adventure, Thriller)
- (mean, Action, Adventure, Western)
- (mean, Action, Animation)
- (mean, Action, Animation, Comedy)
- (mean, Action, Animation, Crime)
- (mean, Action, Animation, Drama)

```
In [65]: pivot_tmdb = tmdb_rating_by_year.pivot_table(index='release_date', columns='genre', values='vote_average')

# Plot the data
pivot_tmdb.plot(figsize=(10, 6))
plt.xlim('2010-01-01', '2020-01-01')
plt.title('Mean Average Rating by Year for Each Genre')
plt.xlabel('Year')
plt.ylabel('Mean Average Rating')
plt.legend(title='Genres', loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```



**Research Question 2:** Is movie production budget directly related to its gross revenue?

```
In [38]: #load tn.movie budgets
with gzip.open('zippedData/tn.movie_budgets.csv.gz') as f:
    tn_movie_budgets = pd.read_csv(f)
tn_movie_budgets.head()
```

```
Out[38]:
```

	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 [39]: #load bom.movie gross for movie revenues
with gzip.open('zippedData/bom.movie_gross.csv.gz') as f:
    bom_movies_gross = pd.read_csv(f)
bom_movies_gross.head()
```

```
Out[39]:
```

	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



```
In [40]: movies_budget_and_revenue = pd.merge(tn_movie_budgets, bom_movies_gross, left_on= 'movie', right_on= 'ti
movies_budget_and_revenue.drop(['domestic_gross_x', 'title'], axis= 1, inplace= True)
movies_budget_and_revenue.rename(columns={'domestic_gross_y': 'domestic_gross'}, inplace= True)
movies_budget_and_revenue.head()
```

```
Out[40]:
```

	id	release_date	movie	production_budget	worldwide_gross	studio	domestic_gross	foreign_gross	year
0	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	\$410,600,000	\$1,045,663,875	BV	241100000.0	804600000	2011
1	4	May 1, 2015	Avengers: Age of Ultron	\$330,600,000	\$1,403,013,963	BV	459000000.0	946400000	2015
2	7	Apr 27, 2018	Avengers: Infinity War	\$300,000,000	\$2,048,134,200	BV	678800000.0	1,369.5	2018
3	9	Nov 17, 2017	Justice League	\$300,000,000	\$655,945,209	WB	229000000.0	428900000	2017
4	10	Nov 6, 2015	Spectre	\$300,000,000	\$879,620,923	Sony	200100000.0	680600000	2015

```
In [41]: movies_budget_and_revenue['production_budget'] = movies_budget_and_revenue['production_budget'].apply(la
movies_budget_and_revenue['worldwide_gross'] = movies_budget_and_revenue['worldwide_gross'].apply(lambda
movies_budget_and_revenue.head()
```

```
Out[41]:
```

	id	release_date	movie	production_budget	worldwide_gross	studio	domestic_gross	foreign_gross	year
0	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	1045663875	BV	241100000.0	804600000	2011
1	4	May 1, 2015	Avengers: Age of Ultron	330600000	1403013963	BV	459000000.0	946400000	2015
2	7	Apr 27, 2018	Avengers: Infinity War	300000000	2048134200	BV	678800000.0	1,369.5	2018
3	9	Nov 17, 2017	Justice League	300000000	655945209	WB	229000000.0	428900000	2017
4	10	Nov 6, 2015	Spectre	300000000	879620923	Sony	200100000.0	680600000	2015

```
In [43]: movies_budget_and_revenue['production_budget'] = movies_budget_and_revenue['production_budget'].astype('float64')
movies_budget_and_revenue['worldwide_gross'] = movies_budget_and_revenue['worldwide_gross'].astype('float64')
movies_budget_and_revenue['year'] = movies_budget_and_revenue['year'].astype('int')
```

```
In [63]: sample_movies = movies_budget_and_revenue.sample(n=100)
plt.scatter(sample_movies['production_budget'], sample_movies['worldwide_gross'])

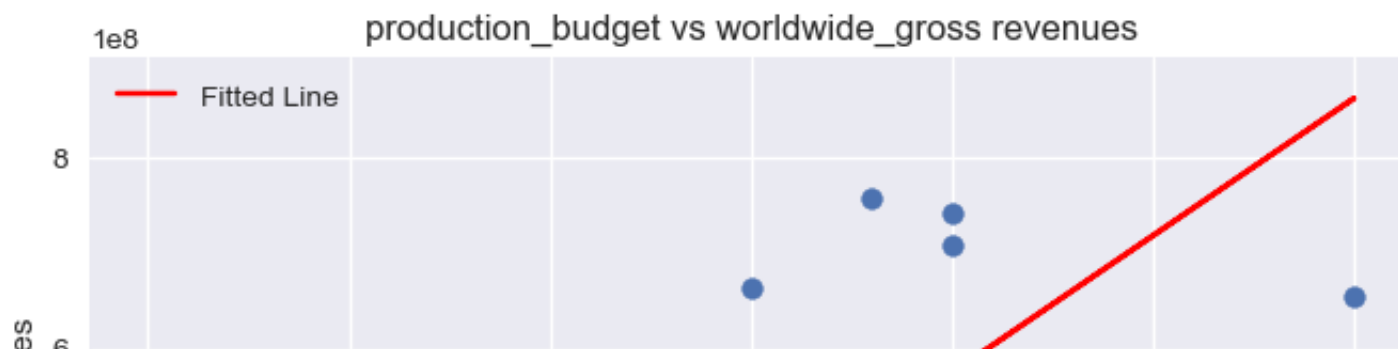
# Define the line equation: y = mx + b
m, b = np.polyfit(sample_movies['production_budget'], sample_movies['worldwide_gross'], 1) # Fit a first degree polynomial

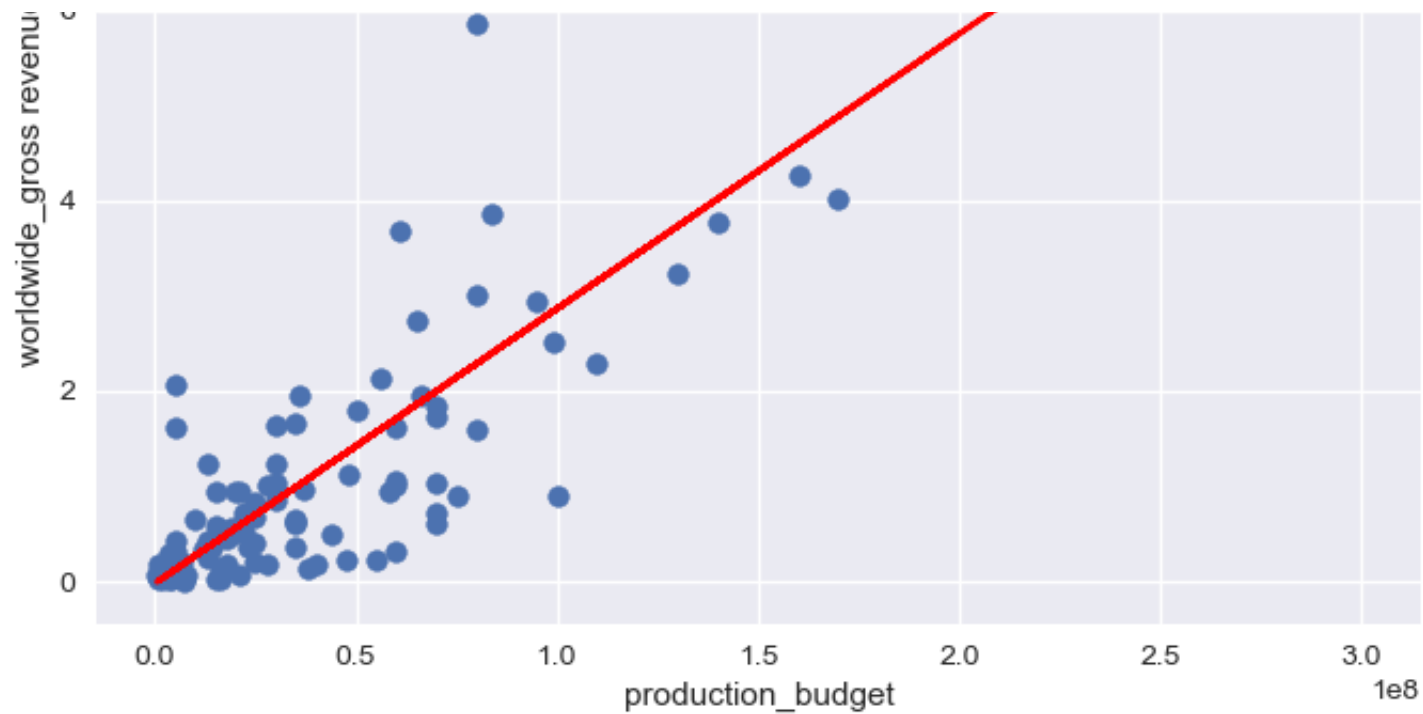
# Create the line
line = m * sample_movies['production_budget'] + b

# Plot the line
plt.plot(sample_movies['production_budget'], line, color='red', label='Fitted Line')

# Add labels and a legend
plt.xlabel('production_budget')
plt.ylabel('worldwide_gross revenues')
plt.title('production_budget vs worldwide_gross revenues')
plt.legend()

# Show the plot
plt.show();
```





**Reserch Question 3:** Which cast and crew are associated with the most popular movies?

```
In [51]: #join imdb.title.principles and imdb.name.basics
with gzip.open('zippedData/imdb.title.principals.csv.gz') as f:
    imdb_title_principals = pd.read_csv(f)
with gzip.open('zippedData/imdb.name.basics.csv.gz') as f:
    imdb_name_basics = pd.read_csv(f)
```

In [52]: `imdb_title_principals.head()`

Out[52]:

	tconst	ordering	nconst	category	job	characters
0	tt0111414	1	nm0246005	actor	NaN	["The Man"]
1	tt0111414	2	nm0398271	director	NaN	NaN
2	tt0111414	3	nm3739909	producer	producer	NaN
3	tt0323808	10	nm0059247	editor	NaN	NaN
4	tt0323808	1	nm3579312	actress	NaN	["Beth Boothby"]

In [53]: `imdb_name_basics.head()`

Out[53]:

	nconst	primary_name	birth_year	death_year	primary_profession	known_for_titles
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,producer	tt0837562,tt2398241,tt0844471,tt0118553
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_department	tt0896534,tt6791238,tt0287072,tt1682940
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,actor,writer	tt1470654,tt0363631,tt0104030,tt0102898
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_department	tt0114371,tt2004304,tt1618448,tt1224387
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_decorator	tt0452644,tt0452692,tt3458030,tt2178256

```
In [54]: #merge imdb.title.principals and imdb_name_basics on the nconst column into a dataframe called movie_pri
movie_principals = pd.merge(imdb_title_principals, imdb_name_basics, on ='nconst', how= 'inner')
movie_principals.head()
```

```
Out[54]:
```

	tconst	ordering	nconst	category	job	characters	primary_name	birth_year	death_year	primary_profession
0	tt0111414	1	nm0246005	actor	NaN	["The Man"]	Tommy Dysart	NaN	NaN	
1	tt0111414	2	nm0398271	director	NaN	NaN	Frank Howson	1952.0	NaN	actor,writer,producer
2	tt5573596	5	nm0398271	director	NaN	NaN	Frank Howson	1952.0	NaN	actor,writer,producer
3	tt0111414	3	nm3739909	producer	producer	NaN	Barry Porter-Robinson	NaN	NaN	producer,art_department
4	tt0323808	10	nm0059247	editor	NaN	NaN	Sean Barton	1944.0	NaN	editor,editorial_department,assistant_director

```
In [55]: #load imdb.title.akas
with gzip.open('zippedData/imdb.title.akas.csv.gz') as f:
    imdb_title_akas = pd.read_csv(f)
imdb_title_akas.head()
```

```
Out[55]:
```

	title_id	ordering	title	region	language	types	attributes	is_original_title
0	tt0369610	10	Джурасик свят	BG	bg	NaN	NaN	0.0
1	tt0369610	11	Jurashikku warudo	JP	NaN	imdbDisplay	NaN	0.0
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	NaN	imdbDisplay	NaN	0.0
3	tt0369610	13	O Mundo dos Dinossauros	BR	NaN	NaN	short title	0.0
4	tt0369610	14	Jurassic World	FR	NaN	imdbDisplay	NaN	0.0

```
In [56]: #include imdb.title.akas in the merge
movie_principals = pd.merge(movie_principals,imdb_title_akas, left_on= 'tconst', right_on= 'title_id', h
movie_principals = movie_principals[['tconst', 'nconst', 'primary_name', 'primary_profession', 'known_fo
movie_principals.head()
```

```
Out[56]:
```

	tconst	nconst	primary_name	primary_profession	known_for_titles	title
0	tt0111414	nm0246005	Tommy Dysart	actor	tt0093120,tt0076974,tt0084296,tt0077064	A Thin Life
1	tt0111414	nm0398271	Frank Howson	actor,writer,producer	tt0104271,tt0094789,tt0102076,tt0111414	A Thin Life
2	tt0111414	nm3739909	Barry Porter-Robinson	producer,art_department	tt0290884,tt0101374,tt0111414,tt1566940	A Thin Life
3	tt5573596	nm0398271	Frank Howson	actor,writer,producer	tt0104271,tt0094789,tt0102076,tt0111414	Remembering Nigel
4	tt5573596	nm0000476	Sally Kirkland	actress,producer,miscellaneous	tt0315327,tt0092569,tt3216866,tt0102138	Remembering Nigel

In [57]: *#check out a sample of the ultimate table*  
 tmdb\_movies.sample(n=10)

Out [57]:

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average	vote_count	
17512	[28, 16]	379291	en	Justice League vs. Teen Titans	11.817	2016-03-29	Justice League vs. Teen Titans	6.9	364	Action
11172	[9648, 53, 28]	241848	en	The Guest	10.101	2014-09-17	The Guest	6.5	954	Mystery
5222	[18]	84892	en	The Perks of Being a Wallflower	15.148	2012-09-21	The Perks of Being a Wallflower	7.8	5938	
14782	[35, 18, 10402, 10749]	206296	en	The Last Five Years	5.079	2015-02-13	The Last Five Years	5.9	230	Comedy,Drama,M
20747	[18]	393457	en	Fences	12.753	2016-12-16	Fences	6.8	1465	
11394	[27]	254194	en	Starry Eyes	7.145	2014-11-14	Starry Eyes	6.0	219	
17433	[14, 28, 12, 878]	274854	en	The Last Witch Hunter	16.685	2015-10-23	The Last Witch Hunter	5.8	2193	Fantasy>Action,Adv
14386	[35, 18, 10749]	306952	en	Naomi and Ely's No Kiss List	8.794	2015-09-18	Naomi and Ely's No Kiss List	6.0	762	Comedy,Dr
8070	[18, 80]	96936	en	The Bling Ring	8.935	2013-06-21	The Bling Ring	5.4	2069	
14206	[12, 27, 35]	257445	en	Goosebumps	18.957	2015-10-16	Goosebumps	6.2	2147	Adventure,I

```
In [58]: #merge some columns of tmdb.movies and select some columns from the entire dataframe resulting from the
movie_principals = pd.merge(movie_principals,tmdb_movies[['title','vote_average','vote_count','genre']],
movie_principals.head())
```

```
Out[58]:
```

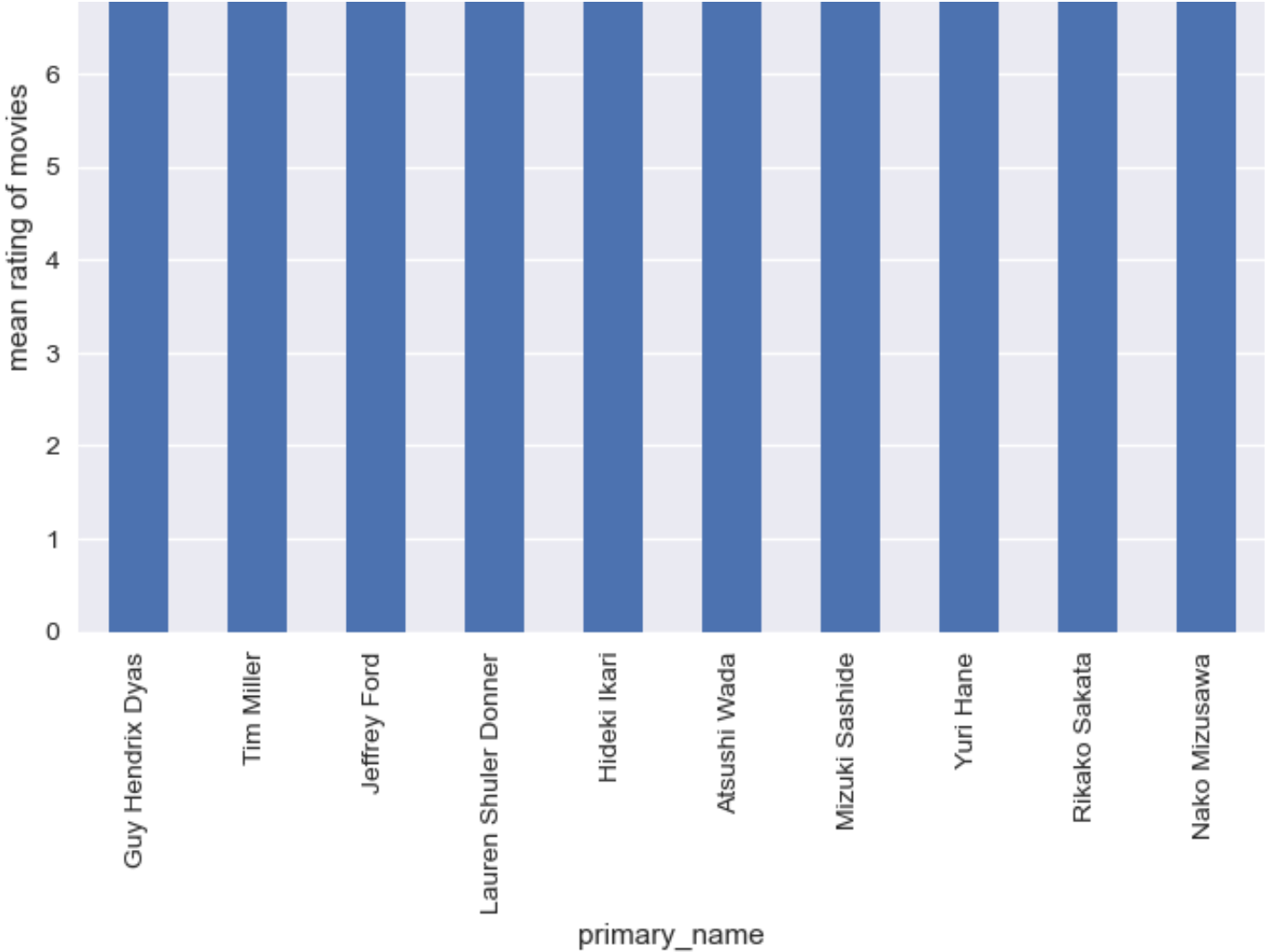
	tconst	nconst	primary_name	primary_profession	known_for_titles	title	vote_average
0	tt1639075	nm0532721	Luis Machín	actor,cinematographer	tt1085786,tt3360828,tt1795747,tt0396356	Criminal	5.9
1	tt1639075	nm0079065	Mariela Besuievsky	producer,actress,executive	tt1816608,tt6792282,tt1305806,tt0488604	Criminal	5.9
2	tt1639075	nm0324409	Bárbara Goenaga	actress	tt0374164,tt0798435,tt3867816,tt0480669	Criminal	5.9
3	tt1639075	nm0324184	Lucio Godoy	composer,music_department,soundtrack	tt0319769,tt0220580,tt1772250,tt0230600	Criminal	5.9
4	tt1639075	nm0380547	Gerardo Herrero	producer,director,writer	tt0120311,tt1305806,tt0260315,tt0436385	Criminal	5.9

```
In [59]: #aggregate using primary names of the movie principals
movie_principals_grouped = movie_principals.groupby(by='primary_name').agg({'vote_average':['mean'],'vote_count':['mean']})
movie_principals_grouped.sort_values(by=movie_principals_grouped.columns[1], ascending=False, inplace=True)
```

```
In [72]: #Plot the top ten principals asociated with highest rated movies
movie_principals_grouped.iloc[:10].plot(kind='bar', y=movie_principals_grouped.columns[0])
plt.title('Top ten principals asociated with highest rated movies')
plt.ylabel('mean rating of movies');
```







## Reccomendations

1. Based on the analysis, Microsoft should source more movies with a bias for Romance, Drama, and Animation genres for their new movie studio.
2. Microsoft new movie studio should make block buster films (i.e. high budget movies) since there is a direct relationship between production budget and gross revenues raked in by such movies
3. The movies should star top movie principals especialy: Guy Hendric Dyas Tim Miller Jeffrey Ford Lauren Shuler Donner Hideki Ikari Atsushi Wada Mizuki Sashide Yuri Hane Rikako Sakata Nako Mizusawa