Microsoft Studios Needs Analysis

Overview

This project analyzes film data from imdb and other sources based on genre, ratings, and release dates to provide accurate data analyses for Microsoft Studios. Descriptive analysis of the variable data shows certain genres have more favorable ratings and/or greater ROI than others, and certain months have proven to be far more profitable than others. Budgets have also been analyzed to show the most favorable investment range. Microsoft Studios can use this data to plan their first motion picture, using the correct release dates, genre types, and budget set ups.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import os
import calendar

#These are all the packages/libraries I plan on using
```

Data

imdb has a number of files with relational movie data, that can be combined and reformatted to analyze specific trends across a number of variables. This also includes a data series with movie budget values, which I will merge together later

In [5]:
```
#Here I'll be loading all the data for my project
movie_budgets = pd.read_csv('tn.movie_budgets.csv.gz')
movie_titles = pd.read_csv('imdb.title.basics.csv.gz')
movie_ratings = pd.read_csv('imdb.title.ratings.csv.gz')
```

Merging the Data

The next few cells are where I will merge the imported data through, so that I can cross compare values

In [9]:
```
#Combine the data
titles_and_ratings = pd.merge(movie_titles, movie_ratings, on='tconst')
titles_and_ratings.head()
```

Out[9]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres	average
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 [11]:
```
#rename columns I plan on merging through
titles_and_ratings.rename(columns = {'primary_title':'movie'}, inplace=True
titles_and_ratings.head()
```

Out[11]:

	tconst	movie	original_title	start_year	runtime_minutes	genres	averagera
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 [14]:
```
#merge the remaining data
aggregate_data = pd.merge(movie_budgets, titles_and_ratings, on='movie')
aggregate_data.head()
```

Out[14]:

vie	production_budget	domestic_gross	worldwide_gross	tconst	original_title	start_year	runtime
ıtar	$425,000,000	$760,507,625	$2,776,345,279	tt1775309	Abatâ	2011	
s of the an: On ger des	$410,600,000	$241,063,875	$1,045,663,875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	
ark nix	$350,000,000	$42,762,350	$149,762,350	tt6565702	Dark Phoenix	2019	
ers: e of ron	$330,600,000	$459,005,868	$1,403,013,963	tt2395427	Avengers: Age of Ultron	2015	
ers: nity Var	$300,000,000	$678,815,482	$2,048,134,200	tt4154756	Avengers: Infinity War	2018	

In [20]:
```
#convert data to integers so I can do some math to calculate profit later
aggregate_data['production_budget'] = aggregate_data['production_budget'].s
aggregate_data['production_budget'] = aggregate_data['production_budget'].s
aggregate_data['worldwide_gross'] = aggregate_data['worldwide_gross'].str.r
aggregate_data['worldwide_gross'] = aggregate_data['worldwide_gross'].str.r
```

In [22]:
```
#convert data to integers so I can do some math to calculate profit later(c
aggregate_data['worldwide_gross'] = aggregate_data['worldwide_gross'].astyp
aggregate_data['production_budget'] = aggregate_data['production_budget'].a
aggregate_data.head()
```

Out[22]:

vie	production_budget	domestic_gross	worldwide_gross	tconst	original_title	start_year	runtime
ıtar	425000000	$760,507,625	2776345279	tt1775309	Abatâ	2011	
s of the an: On ger des	410600000	$241,063,875	1045663875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	
ark nix	350000000	$42,762,350	149762350	tt6565702	Dark Phoenix	2019	
ers: e of ron	330600000	$459,005,868	1403013963	tt2395427	Avengers: Age of Ultron	2015	
ers: nity Var	300000000	$678,815,482	2048134200	tt4154756	Avengers: Infinity War	2018	

In [24]:
```
#Add ROI column, check results
for ind, row in aggregate_data.iterrows():
    aggregate_data.loc[ind, 'ROI'] = row['worldwide_gross'] - row['producti
aggregate_data['ROI'] = aggregate_data['ROI'].astype('int64')
```

In [25]:
```
aggregate_data.head()
```

Out[25]:

on_budget	domestic_gross	worldwide_gross	tconst	original_title	start_year	runtime_minutes	
425000000	$760,507,625	2776345279	tt1775309	Abatâ	2011	93.0	
410600000	$241,063,875	1045663875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	136.0	Act
350000000	$42,762,350	149762350	tt6565702	Dark Phoenix	2019	113.0	A
330600000	$459,005,868	1403013963	tt2395427	Avengers: Age of Ultron	2015	141.0	A
300000000	$678,815,482	2048134200	tt4154756	Avengers: Infinity War	2018	149.0	A

In [29]:
```
#Get data info
aggregate_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2875 entries, 0 to 2874
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 2875 non-null   int64  
 1   release_date       2875 non-null   object 
 2   movie              2875 non-null   object 
 3   production_budget  2875 non-null   int64  
 4   domestic_gross     2875 non-null   object 
 5   worldwide_gross    2875 non-null   int64  
 6   tconst             2875 non-null   object 
 7   original_title     2875 non-null   object 
 8   start_year         2875 non-null   int64  
 9   runtime_minutes    2757 non-null   float64
 10  genres             2867 non-null   object 
 11  averagerating      2875 non-null   float64
 12  numvotes           2875 non-null   int64  
 13  ROI                2875 non-null   int64  
dtypes: float64(2), int64(6), object(6)
memory usage: 416.9+ KB
```

Data Cleaning

To make the data easier to work with, I'll be dropping duplicates and missing values, and grouping movies by individual genres rather than multiple, to better get a sense of each genres impact on the data

In [34]:
```
#Clean the Data
#Drop duplicates and missing values
aggregate_data.drop_duplicates(inplace=True)
aggregate_data.dropna(subset=['genres'], inplace=True)
aggregate_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2867 entries, 0 to 2874
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 2867 non-null   int64  
 1   release_date       2867 non-null   object 
 2   movie              2867 non-null   object 
 3   production_budget  2867 non-null   int64  
 4   domestic_gross     2867 non-null   object 
 5   worldwide_gross    2867 non-null   int64  
 6   tconst             2867 non-null   object 
 7   original_title     2867 non-null   object 
 8   start_year         2867 non-null   int64  
 9   runtime_minutes    2752 non-null   float64
 10  genres             2867 non-null   object 
 11  averagerating      2867 non-null   float64
 12  numvotes           2867 non-null   int64  
 13  ROI                2867 non-null   int64  
dtypes: float64(2), int64(6), object(6)
memory usage: 336.0+ KB
```

In [50]:
```
#double check the genres are strings so they can be seperated
aggregate_data['genres'] = aggregate_data['genres'].astype('str')
```

In [66]:
```
#check for individual genre count
aggregate_data['genres'].str.contains(',')
aggregate_data['genres'].nunique()
```

Out[66]:
```
311
```

In [72]:
```
# Split genres and create a new entry for each of the genre a movie falls i
aggregate_data_split_genre = aggregate_data.copy()
split_genre = aggregate_data['genres'].str.split(',').apply(pd.Series, 1).s
split_genre.name = 'genre_split'
aggregate_data_split_genre = aggregate_data_split_genre.drop(['genres'], ax
```

In [71]:
```
#check that it worked
aggregate_data_split_genre.head()
```

Out[71]:

duction_budget	domestic_gross	worldwide_gross	tconst	original_title	start_year	runtime_minutes
425000000	$760,507,625	2776345279	tt1775309	Abatâ	2011	93.0
410600000	$241,063,875	1045663875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	136.0
410600000	$241,063,875	1045663875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	136.0
410600000	$241,063,875	1045663875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	136.0
350000000	$42,762,350	149762350	tt6565702	Dark Phoenix	2019	113.0

In [73]:
```
#Check the new data info
aggregate_data_split_genre.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6444 entries, 0 to 2874
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 6444 non-null   int64  
 1   release_date       6444 non-null   object 
 2   movie              6444 non-null   object 
 3   production_budget  6444 non-null   int64  
 4   domestic_gross     6444 non-null   object 
 5   worldwide_gross    6444 non-null   int64  
 6   tconst             6444 non-null   object 
 7   original_title     6444 non-null   object 
 8   start_year         6444 non-null   int64  
 9   runtime_minutes    6293 non-null   float64
 10  averagerating      6444 non-null   float64
 11  numvotes           6444 non-null   int64  
 12  ROI                6444 non-null   int64  
 13  genre              6444 non-null   int64  
 14  genre_split        6444 non-null   object 
dtypes: float64(2), int64(7), object(6)
memory usage: 805.5+ KB
```

In [76]:
```
#How many genres are we working with??
len(aggregate_data_split_genre['genre_split'].unique())
```

Out[76]:
```
22
```

In [89]:
```
#Lets see what their names are
(aggregate_data_split_genre['genre_split'].unique())
```

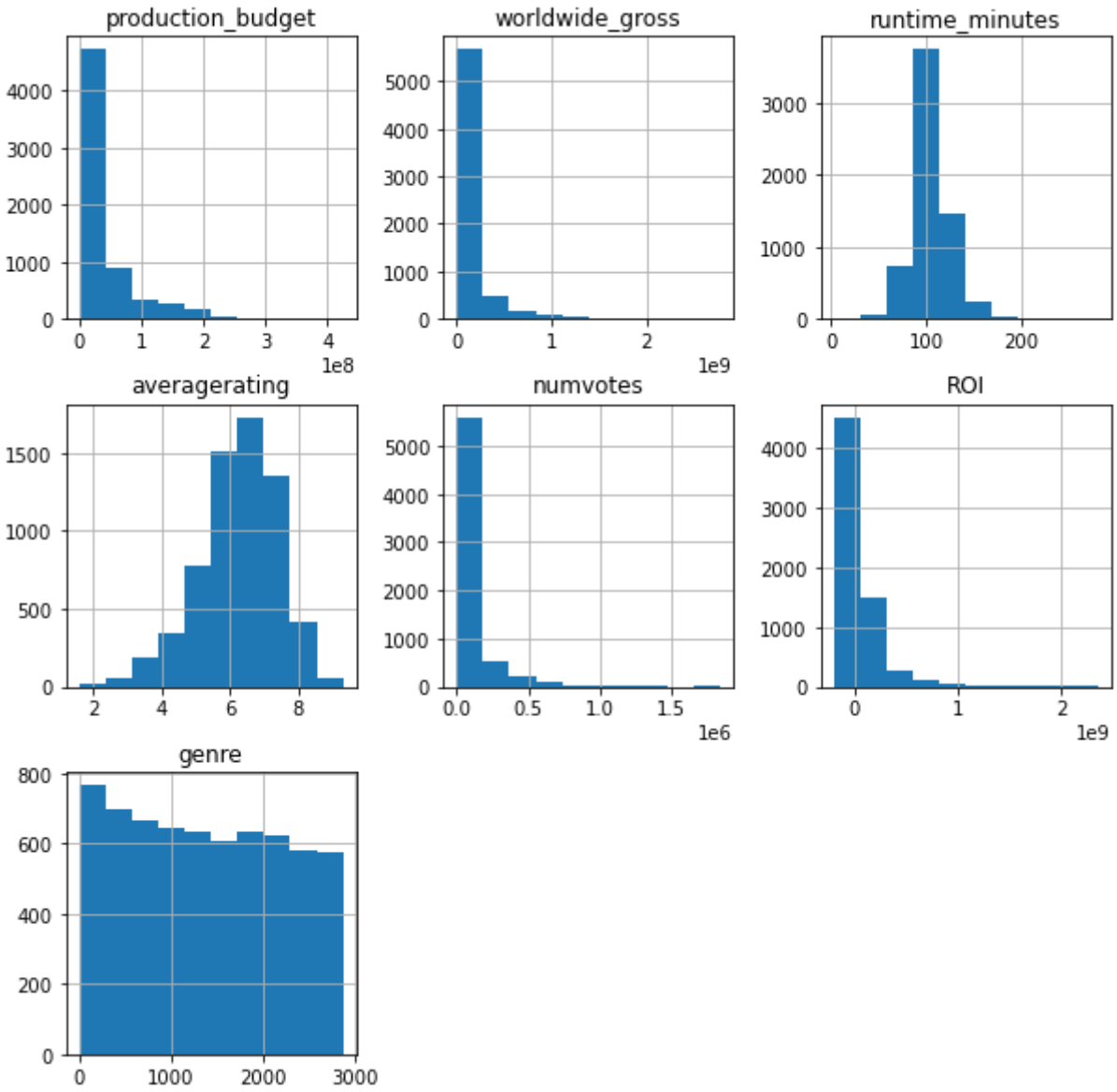
Out[89]:
```
array(['Horror', 'Action', 'Adventure', 'Fantasy', 'Sci-Fi', 'Thriller',
       'Western', 'Animation', 'Comedy', 'Family', 'Crime', 'Drama',
       'Musical', 'Romance', 'Mystery', 'Documentary', 'Biography',
       'History', 'Sport', 'War', 'Music', 'News'], dtype=object)
```

In [77]:
```
#reassign the data
cleaned_data = aggregate_data_split_genre
```

In [81]:
```
#drop the columns that don't tell us useful info
final_data = cleaned_data.drop(['id', 'original_title', 'start_year'], axis
```

Data Analysis

In [82]:
```
#not very telling is it?
final_data.hist(figsize=(10,10));
```



In [101]:
```
#Lets see what the average ROI for all films is
final_data['ROI'].mean()
```

Out[101]:
```
78263220.13407822
```

In [113]:
```
#What about the average ROI be genre?
q1 = final_data.groupby('genre_split').mean('ROI').astype(float)
q1
```

Out[113]:

genre_split	production_budget	worldwide_gross	runtime_minutes	averagerating	numvotes
Action	6.472846e+07	1.904540e+08	110.404568	6.103810	135457.123810
Adventure	9.131197e+07	2.999950e+08	108.925676	6.400893	168058.022321
Animation	8.590607e+07	3.176463e+08	93.196850	6.482308	99923.807692
Biography	2.545666e+07	7.231284e+07	111.897436	6.971795	80511.994872
Comedy	3.342504e+07	1.061395e+08	101.518868	6.186280	66854.522427
Crime	2.747678e+07	6.660450e+07	106.462396	6.255801	79236.754144
Documentary	2.261247e+07	5.761226e+07	81.204082	7.171569	2198.710784
Drama	2.356172e+07	6.132325e+07	107.102154	6.437022	54306.093897
Family	5.256183e+07	1.626862e+08	102.812950	6.200000	58717.340278
Fantasy	6.914902e+07	2.150783e+08	108.465116	6.002857	117055.137143
History	3.119592e+07	7.038875e+07	115.309859	6.829577	62827.436620
Horror	1.811885e+07	6.416836e+07	94.709040	5.370556	42102.613889
Music	1.509611e+07	6.285071e+07	104.972222	6.483333	50035.430556
Musical	3.900909e+07	1.773177e+08	108.095238	6.468182	39405.500000
Mystery	2.173123e+07	7.052312e+07	102.131222	6.006726	82368.825112
News	1.660000e+07	3.668208e+07	68.666667	6.800000	20.333333
Romance	2.029487e+07	6.303945e+07	105.694006	6.274233	51569.453988
Sci-Fi	7.018551e+07	2.455817e+08	108.785000	6.145098	197730.406863
Sport	2.382460e+07	7.779175e+07	109.459016	6.591935	41965.370968
Thriller	2.753381e+07	8.089901e+07	102.081136	5.855010	68396.308448
War	2.310256e+07	4.257518e+07	114.871795	6.438462	37215.589744
Western	4.655000e+07	7.666840e+07	110.866667	6.387500	127401.625000

In [409]:
```
#what's the mean rating of all movies?
q1['averagerating'].mean()
```

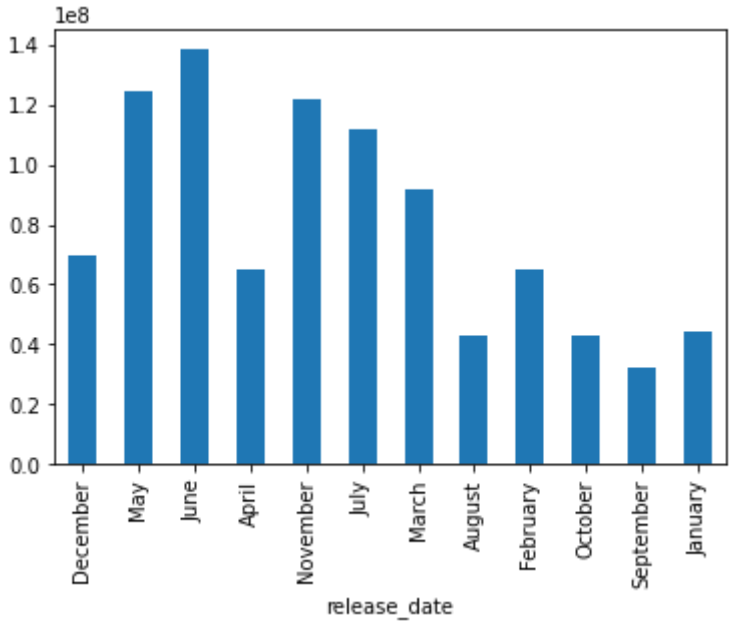
Out[409]: 6.357406649060606

In [410]:
```
#What's the average ROI of every all movies?
q1['ROI'].mean().astype(int)
```

Out[410]:
```
80404806
```

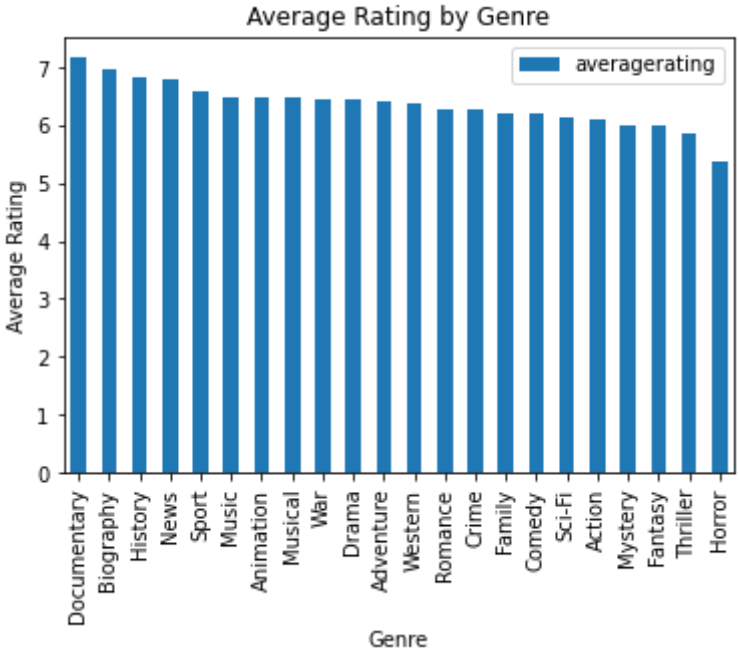
In [98]:
```
#convert dates to datetime so they can be grouped later
final_data['release_date'] = pd.to_datetime(final_data['release_date'], day
final_data.groupby([final_data['release_date'].dt.month_name()], sort=False
  .plot(kind='bar')
```

Out[98]:
```
<AxesSubplot:xlabel='release_date'>
```



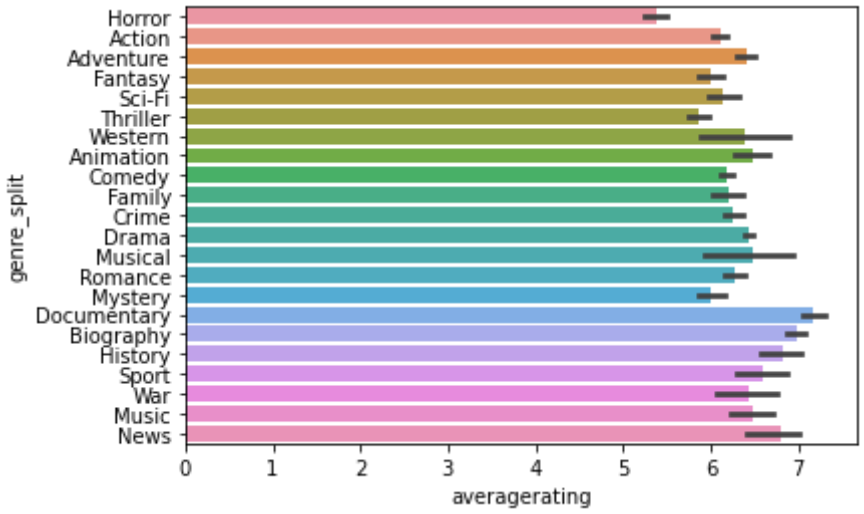
In [282]:
```
#Lets create our first research question graph, What is the average rating
q1_bar = q1.sort_values('averagerating', ascending=False).reset_index().plo
q1_bar
plt.title("Average Rating by Genre")
plt.xlabel("Genre")
plt.ylabel("Average Rating")
```

Out[282]:
```
Text(0, 0.5, 'Average Rating')
```



In [353]:
```
#The above plot works! but lets step it up a bit
sns.barplot(y='genre_split', x='averagerating', data=final_data)
```

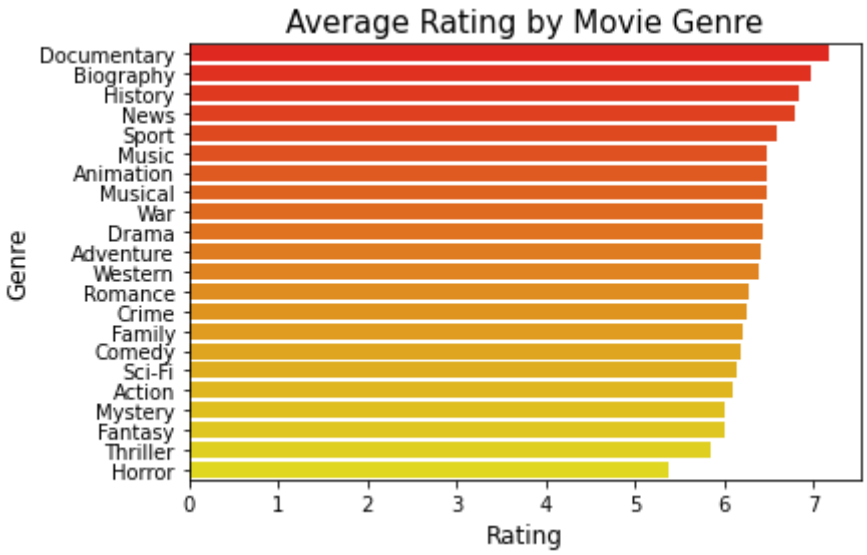
Out[353]:
```
<AxesSubplot:xlabel='averagerating', ylabel='genre_split'>
```



Graphing the Rating Data Based on Genre

In [411]:
```
#looks good! Let's make it a bit more comprehensive, organize the data, and
sns.barplot(y='genre_split', x='averagerating', data=final_data, palette='a
            order=['Documentary', 'Biography', 'History', 'News', 'Sport',
                   'War', 'Drama', 'Adventure', 'Western', 'Romance', 'Crime
                   'Action', 'Mystery', 'Fantasy', 'Thriller', 'Horror'], ci
plt.title('Average Rating by Movie Genre', size= 15)
plt.ylabel('Genre', size = 12)
plt.xlabel('Rating', size = 12)
```

Out[411]:
```
Text(0.5, 0, 'Rating')
```



In [383]:
```
#Group our data for our second research question
q2 = final_data.groupby('genre_split').mean('ROI').astype(int)
q2
```

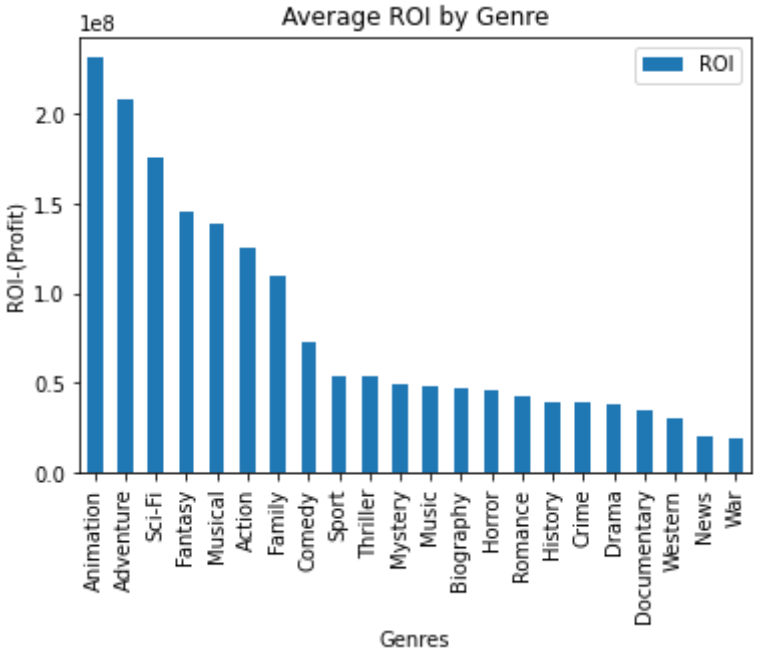
Out[383]:

genre_split	production_budget	worldwide_gross	runtime_minutes	averagerating	numvotes	
Action	64728460	190454040	110	6	135457	1257
Adventure	91311973	299994950	108	6	168058	2086
Animation	85906071	317646291	93	6	99923	2317
Biography	25456658	72312842	111	6	80511	468
Comedy	33425035	106139479	101	6	66854	727
Crime	27476777	66604501	106	6	79236	391
Documentary	22612467	57612261	81	7	2198	349
Drama	23561723	61323247	107	6	54306	377
Family	52561829	162686210	102	6	58717	1101
Fantasy	69149019	215078312	108	6	117055	1459
History	31195915	70388753	115	6	62827	391
Horror	18118852	64168364	94	5	42102	460
Music	15096111	62850710	104	6	50035	477
Musical	39009090	177317674	108	6	39405	1383
Mystery	21731233	70523122	102	6	82368	487
News	16600000	36682077	68	6	20	200
Romance	20294868	63039450	105	6	51569	427
Sci-Fi	70185511	245581691	108	6	197730	1753
Sport	23824596	77791752	109	6	41965	539
Thriller	27533808	80899007	102	5	68396	533
War	23102564	42575176	114	6	37215	194
Western	46550000	76668399	110	6	127401	301

Graphing the Profit Based on Genre

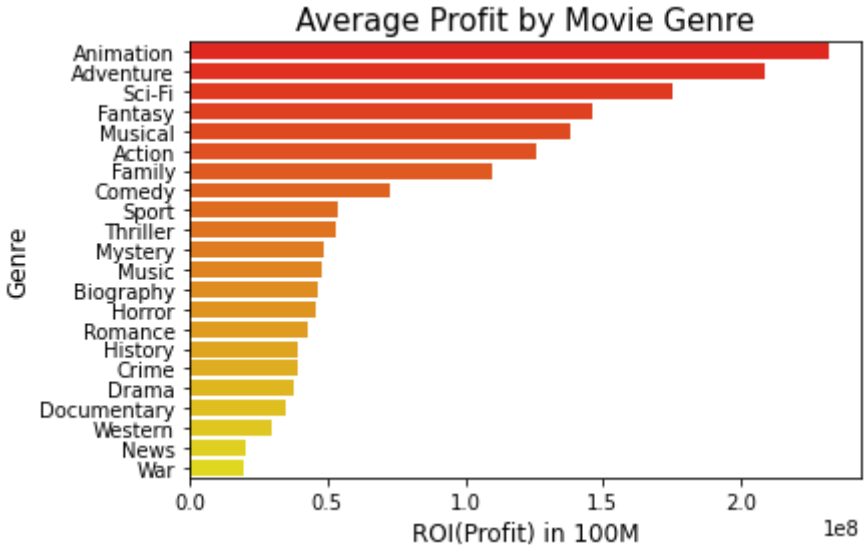
In [268]:
```
#plot out our next question, what is the Average ROI(Return on Investment)
q2_bar = q2.sort_values('ROI', ascending=False).reset_index().plot.bar(x='g
plt.title("Average ROI by Genre")
plt.xlabel("Genres")
plt.ylabel("ROI-(Profit)")
```

Out[268]:
```
Text(0, 0.5, 'ROI-(Profit)')
```



In [412]:
```
#Again very informative! But we can do better
sns.barplot(y='genre_split', x='ROI', data=final_data, palette='autumn',
            order=['Animation', 'Adventure', 'Sci-Fi', 'Fantasy', 'Musical'
                   'Comedy', 'Sport', 'Thriller', 'Mystery', 'Music',
                    'Biography', 'Horror', 'Romance', 'History', 'Crime', '
                  'Western', 'News', 'War'], ci=None)
plt.title('Average Profit by Movie Genre', size= 15)
plt.ylabel('Genre', size = 12)
plt.xlabel('ROI(Profit) in 100M', size = 12)
```

Out[412]:
```
Text(0.5, 0, 'ROI(Profit) in 100M')
```



In [243]:
```
#Check the data type
aggregate_data_split_genre['release_date'] = pd.to_datetime(aggregate_data[
type(aggregate_data_split_genre['release_date'])
```

Out[243]:
```
pandas.core.series.Series
```

Reformat DateTime Data

In [244]:
```
#Start grouping our data by month
aggregate_data_split_genre['month'] = aggregate_data_split_genre['release_d
```

In [245]:
```
aggregate_data_split_genre.head()
```

Out[245]:

budget	domestic_gross	worldwide_gross	tconst	original_title	start_year	runtime_minutes	averag
000000	$760,507,625	2776345279	tt1775309	Abatâ	2011	93.0	
600000	$241,063,875	1045663875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	136.0	
600000	$241,063,875	1045663875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	136.0	
600000	$241,063,875	1045663875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	136.0	
000000	$42,762,350	149762350	tt6565702	Dark Phoenix	2019	113.0	

In [235]:
```
type(aggregate_data_split_genre['month'])
```

Out[235]:
```
pandas.core.series.Series
```

In [246]:
```
#Use a lambda function to reassign Month number to the actual month names
aggregate_data_split_genre['month'] = aggregate_data_split_genre['month'].a
```

In [247]:
```
#lets see if it worked
aggregate_data_split_genre.head()
```

Out[247]:

budget	domestic_gross	worldwide_gross	tconst	original_title	start_year	runtime_minutes	averag
000000	$760,507,625	2776345279	tt1775309	Abatâ	2011	93.0	
600000	$241,063,875	1045663875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	136.0	
600000	$241,063,875	1045663875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	136.0	
600000	$241,063,875	1045663875	tt1298650	Pirates of the Caribbean: On Stranger Tides	2011	136.0	
000000	$42,762,350	149762350	tt6565702	Dark Phoenix	2019	113.0	

In [248]:
```
#Organize our data by the month of the films' release date, with the averag
q3= aggregate_data_split_genre.groupby('month').mean('ROI').astype(int)
q3
```

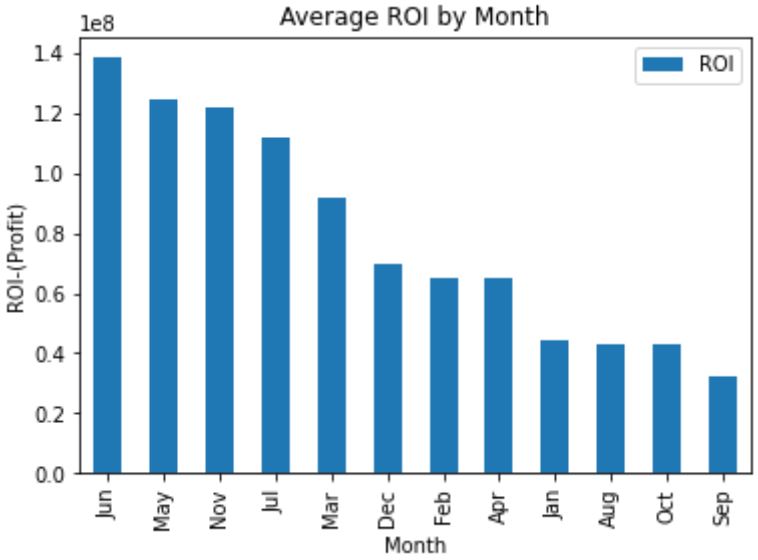
Out[248]:

month	id	production_budget	worldwide_gross	start_year	runtime_minutes	averagerating	numvot
Apr	49	29926636	94917175	2013	103	6	596
Aug	55	28258886	71411685	2013	100	6	691
Dec	52	32861622	102428458	2013	107	6	668
Feb	53	35027414	100204775	2014	102	6	705
Jan	55	26458305	70851434	2014	105	6	458
Jul	51	46059838	158140742	2013	104	6	1035
Jun	52	55945349	194518235	2013	105	6	990
Mar	51	48521344	140210199	2014	102	6	845
May	47	62428616	187308685	2013	107	6	1187
Nov	47	49298334	171117770	2014	108	6	1137
Oct	50	25156466	67858893	2013	105	6	756
Sep	49	25259576	57745979	2013	102	6	677

Group Profit Averages by Release Month

In [414]:
```
#lets plot it out!
q3_bar = q3.sort_values('ROI', ascending=False).reset_index().plot.bar(x='m
plt.title("Average ROI by Month")
plt.xlabel("Month")
plt.ylabel("ROI-(Profit)")
```

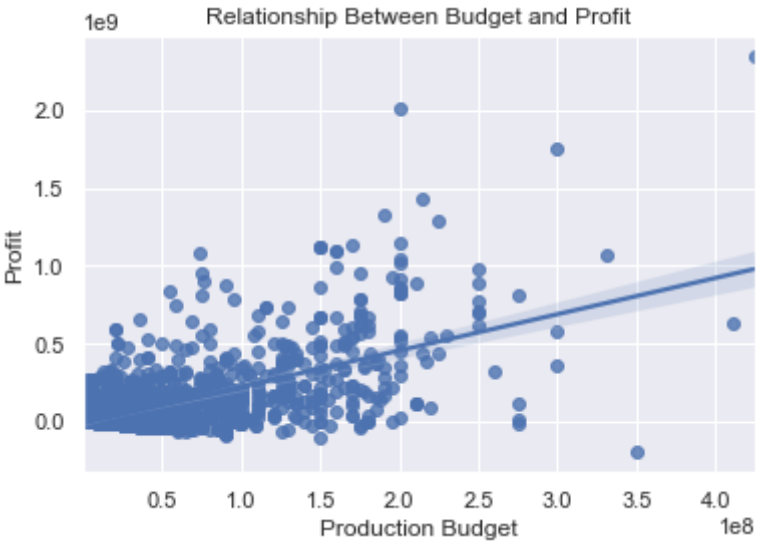
Out[414]:
```
Text(0, 0.5, 'ROI-(Profit)')
```



Use a Scatterplot to Display the Relationship between Budget and Profit

In [434]:
```
#Finally, let's explore the relationship between production investments and
budget_review = sns.regplot(x=aggregate_data['production_budget'], y=aggreg
sns.set_theme()
plt.title('Relationship Between Budget and Profit')
plt.xlabel('Production Budget')
plt.ylabel('Profit')
```

Out[434]:
```
Text(0, 0.5, 'Profit')
```



Conclusions

This Analysis leads to three key conclusions regarding Microsoft Studios' new project:

Animation, Adventure, and Musicals are he most reliably successful genres through both rating and profit. Pick a film that falls under any combination of the three. Investing between 100-200 million on a production budget protects a studio's profits while also minimizing the risk of any losses, so I would recommend choosing or financing a film with a budget in that range. Release dates for the film should prioritize the months of June, May, and November, in respective order of profit increase.

Next Steps

Deeper Analysis could further assist the Studios decisions:

1.Investigating ROI by percentage returned rather than gross value

2.Analyze monthly Profits by genre to find the best potential matches for release dates and specific movies

3.Lastly finding successful partners who have proven to have comparable impacts as the data solutions available so far, you know, which directors, which actors, which studios if you were to consider a joint venture.

In [435]:
```
#Neat! Looks like that about does it!
```