Microsoft Movie Analysis

Author: Jennifer Ha

Overview

Business Problem

- Which genres are the most profitable and yield the highest ROI?
- What budget range in those genres yields the highest ROI?
- Who are the top casts and directors in those genres that Microsoft should work with?

Data

When cleaning the data, we will drop all columns that we don't need for this project. We will clean up duplicates after we join the tables so that we can achieve the desirable output record at once.

- tn_movie_budgets
- imdb_name_basics
- imdb_title_principals
- imdb_title_basics

Prepare Data

In [1]:
```
#import libraries
import datetime as dt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from scipy import stats
```

Clean Up tn_movie_budgets

In [2]:
```
ls zippedData
```

```
bom.movie_gross.csv.gz        imdb.title.ratings.csv.gz
imdb.name.basics.csv.gz       rt.movie_info.tsv.gz
imdb.title.akas.csv.gz        rt.reviews.tsv.gz
imdb.title.basics.csv.gz      tmdb.movies.csv.gz
imdb.title.crew.csv.gz        tn.movie_budgets.csv.gz
imdb.title.principals.csv.gz
```

In [3]:
```
#import the tn_movie_budgets and examine the data
tn_movie_budgets = pd.read_csv("zippedData/tn.movie_budgets.csv.gz")

#get summary of the DataFrame
tn_movie_budgets.info()
```

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

In [4]:
```
#drop id since we don't need column, and domestic_gross column because we w
tn_movie_budgets = tn_movie_budgets.drop(['id','domestic_gross'], axis = 1)
```

In [5]:
```
#convert release_date to datetime and add a release_year column
tn_movie_budgets['release_date'] = pd.to_datetime(tn_movie_budgets['release
tn_movie_budgets['release_year'] = tn_movie_budgets['release_date'].dt.year

#check data
tn_movie_budgets.head()
```

Out[5]:

	release_date	movie	production_budget	worldwide_gross	release_year
0	2009-12-18	Avatar	$425,000,000	$2,776,345,279	2009
1	2011-05-20	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$1,045,663,875	2011
2	2019-06-07	Dark Phoenix	$350,000,000	$149,762,350	2019
3	2015-05-01	Avengers: Age of Ultron	$330,600,000	$1,403,013,963	2015
4	2017-12-15	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$1,316,721,747	2017

In [6]:
```
#check for missing values
tn_movie_budgets.isna().sum()
```

Out[6]:
```
release_date         0
movie                0
production_budget    0
worldwide_gross      0
release_year         0
dtype: int64
```

In [7]:
```
#convert currency with $ to integer
tn_movie_budgets[tn_movie_budgets.columns[2:]] = tn_movie_budgets[tn_movie_

#check data
tn_movie_budgets.head()
```

Out[7]:

	release_date	movie	production_budget	worldwide_gross	release_year
0	2009-12-18	Avatar	425000000	2776345279	2009
1	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	1045663875	2011
2	2019-06-07	Dark Phoenix	350000000	149762350	2019
3	2015-05-01	Avengers: Age of Ultron	330600000	1403013963	2015
4	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000	1316721747	2017

In [8]:
```
#examine the data if there are any rows with 0
print(len(tn_movie_budgets[tn_movie_budgets['production_budget'] > 0]) / le
print(len(tn_movie_budgets[tn_movie_budgets['worldwide_gross'] > 0]) / len(
```

```
1.0
0.936527153234175
```

In [9]:
```
#drop rows with 0 in worldwide_gross column since 93.6% of data have associ
tn_movie_budgets = tn_movie_budgets[tn_movie_budgets['worldwide_gross'] > 0

#check data
tn_movie_budgets.head()
```

Out[9]:

	release_date	movie	production_budget	worldwide_gross	release_year
0	2009-12-18	Avatar	425000000	2776345279	2009
1	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	1045663875	2011
2	2019-06-07	Dark Phoenix	350000000	149762350	2019
3	2015-05-01	Avengers: Age of Ultron	330600000	1403013963	2015
4	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000	1316721747	2017

In [10]:
```
#calculate ROI and add the column
def roi(budget, gross):
    return (gross - budget) / budget * 100
tn_movie_budgets['ROI'] = roi(tn_movie_budgets['production_budget'], tn_mov

#check data
tn_movie_budgets
```

Out[10]:

	release_date	movie	production_budget	worldwide_gross	release_year	ROI
0	2009-12-18	Avatar	425000000	2776345279	2009	553.257713
1	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	1045663875	2011	154.667286
2	2019-06-07	Dark Phoenix	350000000	149762350	2019	-57.210757
3	2015-05-01	Avengers: Age of Ultron	330600000	1403013963	2015	324.384139
4	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000	1316721747	2017	315.369636
...	...	...	...	...	...	...
5775	2006-05-26	Cavite	7000	71644	2006	923.485714
5776	2004-12-31	The Mongol King	7000	900	2004	-87.142857
5778	1999-04-02	Following	6000	240495	1999	3908.250000
5779	2005-07-13	Return to the Land of Wonders	5000	1338	2005	-73.240000
5781	2005-08-05	My Date With Drew	1100	181041	2005	16358.272727

5415 rows × 6 columns

Clean Up imdb_title_basics

In [11]:
```
#import the file and examine the data
imdb_title_basics = pd.read_csv("zippedData/imdb.title.basics.csv.gz")

imdb_title_basics.head()
```

Out[11]:

	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 [12]:
```
#set tconst column as the index since it will be used as primary key/foreig
imdb_title_basics = imdb_title_basics.set_index('tconst')

#check data
imdb_title_basics.head()
```

Out[12]:

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

In [13]:
```
#drop original_title and runtime_minutes columns as we don't need this data
imdb_title_basics = imdb_title_basics.drop(['original_title','runtime_minut

#check data
imdb_title_basics.head()
```

Out[13]:

tconst	primary_title	start_year	genres
tt0063540	Sunghursh	2013	Action,Crime,Drama
tt0066787	One Day Before the Rainy Season	2019	Biography,Drama
tt0069049	The Other Side of the Wind	2018	Drama
tt0069204	Sabse Bada Sukh	2018	Comedy,Drama
tt0100275	The Wandering Soap Opera	2017	Comedy,Drama,Fantasy

In [14]:
```
#check for missing values
imdb_title_basics.isna().sum()
```

Out[14]:
```
primary_title        0
start_year           0
genres            5408
dtype: int64
```

In [15]:
```
#examine genre column to determine whether we shoup drop N/As or replace th
imdb_title_basics['genres'].isna().sum()/len(imdb_title_basics)
```

Out[15]:
```
0.037004598204510616
```

In [16]:
```
#drop rows with N/A in genre column since only 3.7% of data shows N/A
imdb_title_basics = imdb_title_basics.dropna(subset = ['genres'])
```

In [17]:
```
#convert genre to list
imdb_title_basics['genres'] = imdb_title_basics['genres'].map(lambda x: x.s

#check data
imdb_title_basics.head()
```

Out[17]:

tconst	primary_title	start_year	genres
tt0063540	Sunghursh	2013	[Action, Crime, Drama]
tt0066787	One Day Before the Rainy Season	2019	[Biography, Drama]
tt0069049	The Other Side of the Wind	2018	[Drama]
tt0069204	Sabse Bada Sukh	2018	[Comedy, Drama]
tt0100275	The Wandering Soap Opera	2017	[Comedy, Drama, Fantasy]

Clean Up imdb_title_principals

In [18]:
```
#import the file and examine the data
imdb_title_principals = pd.read_csv("zippedData/imdb.title.principals.csv.g

#examine data
imdb_title_principals.head()
```

Out[18]:

	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 [19]:
```
#set tconst column as the index since it will be used as primary key/foreig
imdb_title_principals = imdb_title_principals.set_index('tconst')

#check data
imdb_title_principals.head()
```

Out[19]:

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

In [20]:
```
#drop ordering, job, and characters columns as we don't need this data for
imdb_title_principals = imdb_title_principals.drop(['ordering','job','chara

#check data
imdb_title_principals.head()
```

Out[20]:

	nconst	category
tconst		
tt0111414	nm0246005	actor
tt0111414	nm0398271	director
tt0111414	nm3739909	producer
tt0323808	nm0059247	editor
tt0323808	nm3579312	actress

In [21]:
```
#check for missing values
imdb_title_principals.isna().sum()
```

Out[21]:
```
nconst      0
category    0
dtype: int64
```

In [22]:
```
#examine the job titles and value counts in category column
imdb_title_principals.category.value_counts()
```

Out[22]:
```
actor                  256718
director               146393
actress                146208
producer               113724
cinematographer         80091
composer                77063
writer                  74357
self                    65424
editor                  55512
production_designer      9373
archive_footage          3307
archive_sound              16
Name: category, dtype: int64
```

In [23]:
```
#replace 'actress' with 'actors'
imdb_title_principals['category'] = imdb_title_principals.category.str.repl

#check data
imdb_title_principals.head()
```

Out[23]:

	nconst	category
tconst		
tt0111414	nm0246005	actor
tt0111414	nm0398271	director
tt0111414	nm3739909	producer
tt0323808	nm0059247	editor
tt0323808	nm3579312	actor

Clean Up imdb_name_basics

In [24]:
```
#import the file and examine the data
imdb_name_basics = pd.read_csv("zippedData/imdb.name.basics.csv.gz")

#examine data
imdb_name_basics.head()
```

Out[24]:

	nconst	primary_name	birth_year	death_year	primary_professio
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,produce
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_departmen
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,actor,write
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_departmen
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_decorato

In [25]:
```
#set nconst column as the index since it will be used as primary key/foreig
imdb_name_basics = imdb_name_basics.set_index('nconst')

#check data
imdb_name_basics.head()
```

Out[25]:

	primary_name	birth_year	death_year	primary_profession
nconst				
nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,producer
nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_department
nm0062070	Bruce Baum	NaN	NaN	miscellaneous,actor,writer
nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_department
nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_decorator

In [26]:
```
#drop birth_year, death_year, primary_profession, and 'known_for_titles' as
imdb_name_basics = imdb_name_basics.drop(['birth_year','death_year','primar

#check data
imdb_name_basics.head()
```

Out[26]:

	primary_name
nconst	
nm0061671	Mary Ellen Bauder
nm0061865	Joseph Bauer
nm0062070	Bruce Baum
nm0062195	Axel Baumann
nm0062798	Pete Baxter

Join Datasets

Merge movie_budgets and title_basics as df

Combine movie_budget and title_basics DataFrame to create a new DataFrame called df. Since the column names are different, we are going to explicitly mention both the column names using 'left_on' and 'right_on' arguments. Then, we will remove the duplicate columns as a result of the merge, and reset the index to 'tconst' so that we can easily use the index as the primary key when joining with other tables for additional analyes.

In [27]:
```
#merge tn_movie_budgets and imdb_title_basics as df
df = pd.merge(imdb_title_basics.reset_index(), tn_movie_budgets,
              left_on=['primary_title', 'start_year'],
              right_on=['movie', 'release_year'])

#drop duplicate columns (primary_title, start_year)
df = df.drop(['primary_title','start_year'], axis = 1)

#reset index to tconst
df = df.set_index('tconst')

#check data
df.head()
```

Out[27]:

	genres	release_date	movie	production_budget	worldwide_gross	release_year
tconst						
tt0249516	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	73706	2012
tt0359950	[Adventure, Comedy, Drama]	2013-12-25	The Secret Life of Walter Mitty	91000000	187861183	2013
tt0365907	[Action, Crime, Drama]	2014-09-19	A Walk Among the Tombstones	28000000	62108587	2014
tt0369610	[Action, Adventure, Sci-Fi]	2015-06-12	Jurassic World	215000000	1648854864	2015
tt0376136	[Comedy, Drama]	2011-10-28	The Rum Diary	45000000	21544732	2011

In [28]:
```
#check for duplicates since df DataFrame will serve as our primary table fo
df[df.duplicated(
        subset = ['movie', 'release_year'],
        keep = False)].shape
```

Out[28]: (94, 7)

For higher accuracy and simplicity of this analysis project, we will remove duplicates from this DataFrame.

In [29]:
```
#drop duplicates
df = df[~df.duplicated(
        subset = ['movie', 'release_year'],
        keep=False)]

#check data
df.head()
```

Out[29]:

tconst	genres	release_date	movie	production_budget	worldwide_gross	release_year
tt0249516	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	73706	2012
tt0359950	[Adventure, Comedy, Drama]	2013-12-25	The Secret Life of Walter Mitty	91000000	187861183	2013
tt0365907	[Action, Crime, Drama]	2014-09-19	A Walk Among the Tombstones	28000000	62108587	2014
tt0369610	[Action, Adventure, Sci-Fi]	2015-06-12	Jurassic World	215000000	1648854864	2015
tt0376136	[Comedy, Drama]	2011-10-28	The Rum Diary	45000000	21544732	2011

Join df with imdb_title_principals

Update the df DataFrame by joining imdb_title_principals using an inner join, and the output will automatically drop N/A values. There is no need to specify where the join will occur since we have already reset indices for this purpose when cleaning the data.

In [30]:
```
#join df with imdb_title_principals
df = imdb_title_principals.join(df, how='inner')

df
```

Out[30]:

	nconst	category	genres	release_date	movie	production_budget	worldwide
tconst							
tt0249516	nm0257258	producer	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	
tt0249516	nm0240381	actor	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	
tt0249516	nm0240380	actor	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	
tt0249516	nm0000221	actor	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	
tt0249516	nm0519456	actor	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	
...	...	...	...	...	...	...	
tt9024106	nm0465484	director	[Biography, Drama]	2019-03-29	Unplanned	6000000	18
tt9024106	nm0813301	director	[Biography, Drama]	2019-03-29	Unplanned	6000000	18
tt9024106	nm2445956	producer	[Biography, Drama]	2019-03-29	Unplanned	6000000	18
tt9024106	nm7839151	producer	[Biography, Drama]	2019-03-29	Unplanned	6000000	18
tt9024106	nm0498920	producer	[Biography, Drama]	2019-03-29	Unplanned	6000000	18

13144 rows × 9 columns

In [31]:
```
#Check for people who appears more than once in a movie
df.reset_index().duplicated(subset=['tconst', 'nconst']).sum()
```

Out[31]: 0

In [32]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 13144 entries, tt0249516 to tt9024106
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   nconst             13144 non-null  object
 1   category           13144 non-null  object
 2   genres             13144 non-null  object
 3   release_date       13144 non-null  datetime64[ns]
 4   movie              13144 non-null  object
 5   production_budget  13144 non-null  int64
 6   worldwide_gross    13144 non-null  int64
 7   release_year       13144 non-null  int64
 8   ROI                13144 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(3), object(4)
memory usage: 1.0+ MB
```

Join df with imbd_name_basics

In [33]:
```
df = pd.merge(df.reset_index(),imdb_name_basics,how = 'inner', on = 'nconst

df
```

Out[33]:

	tconst	nconst	category	genres	release_date	movie	production_budget	wc
0	tt0249516	nm0257258	producer	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	
1	tt0249516	nm0240381	actor	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	
2	tt0249516	nm0240380	actor	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	
3	tt0249516	nm0000221	actor	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	
4	tt0249516	nm0519456	actor	[Action, Animation, Comedy]	2012-12-31	Foodfight!	45000000	
...	...	...	...	...	...	...	...	
13139	tt9024106	nm0478750	actor	[Biography, Drama]	2019-03-29	Unplanned	6000000	
13140	tt9024106	nm5237225	actor	[Biography, Drama]	2019-03-29	Unplanned	6000000	
13141	tt9024106	nm2445956	producer	[Biography, Drama]	2019-03-29	Unplanned	6000000	
13142	tt9024106	nm7839151	producer	[Biography, Drama]	2019-03-29	Unplanned	6000000	
13143	tt9024106	nm0498920	producer	[Biography, Drama]	2019-03-29	Unplanned	6000000	

13144 rows × 11 columns

Split Genres to Separate Rows

Use `explode` function to transform each element in a list in 'genres' column. This will allow us to easily run analysis by **genres**.

In [34]:
```
df = df.explode('genres')
df.head()
```

Out[34]:

	tconst	nconst	category	genres	release_date	movie	production_budget	worldwid
0	tt0249516	nm0257258	producer	Action	2012-12-31	Foodfight!	45000000	
0	tt0249516	nm0257258	producer	Animation	2012-12-31	Foodfight!	45000000	
0	tt0249516	nm0257258	producer	Comedy	2012-12-31	Foodfight!	45000000	
1	tt0249516	nm0240381	actor	Action	2012-12-31	Foodfight!	45000000	
1	tt0249516	nm0240381	actor	Animation	2012-12-31	Foodfight!	45000000	

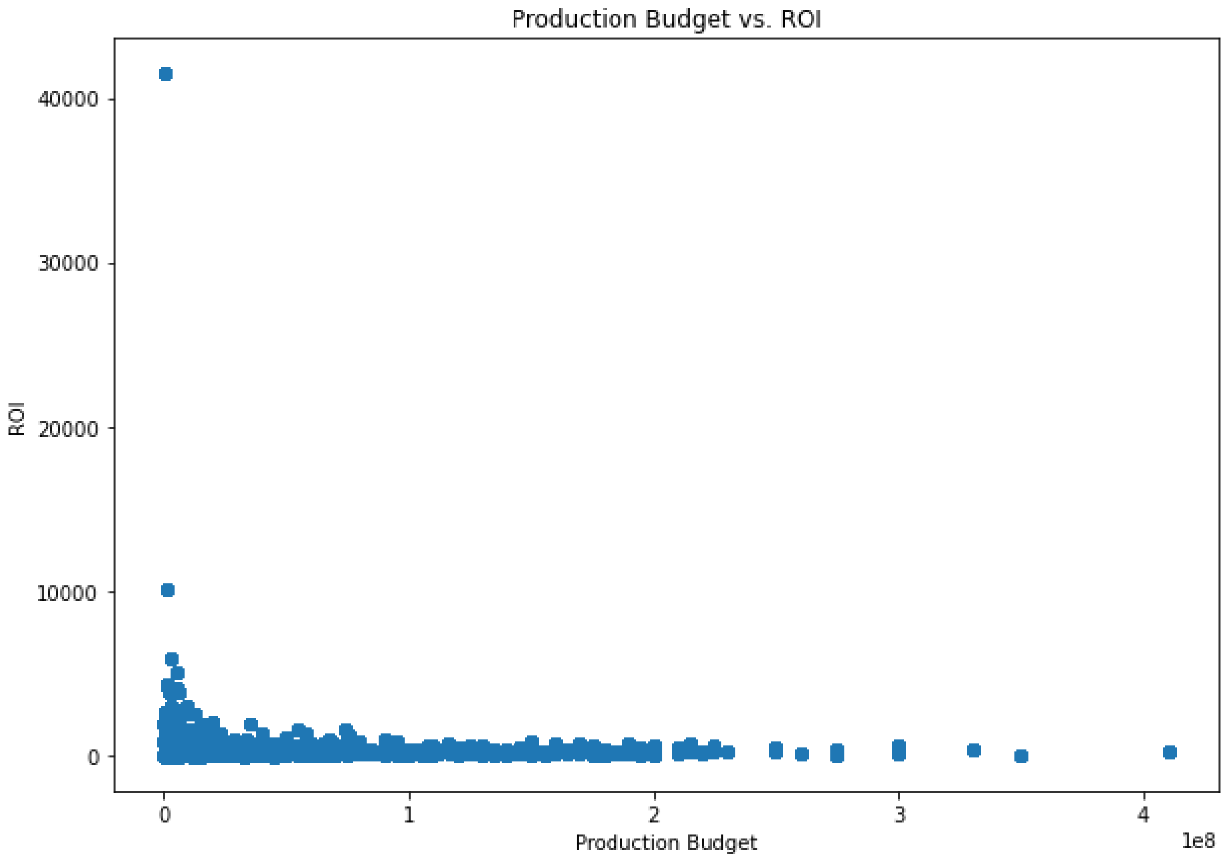
Identify and Remove Outliers

We have now completed cleaning up and joining the DataFrames that are required for this project. However, we need to further examine the data before we start data exploration to check if there are any outliers that should be removed.

First, we will examine the overall production_budget and ROI, and will create a fuction so that we can re-visualize the data after removing the outliers.

In [35]:
```
#define a function to visualize scatter plots with outliers
def scatter_plot(df, x ='production_budget', y = 'ROI'):

    #create the plot
    f, ax = plt.subplots(figsize = (10,7))

    #Generate a scatter plot
    ax.scatter(data = df,
           x = 'production_budget',
           y= 'ROI')

    #Add labels for x and y axes
    ax.set_xlabel('Production Budget')
    ax.set_ylabel('ROI')

    #Add a title for the plot
    ax.set_title('Production Budget vs. ROI')
    ;
```

In [36]:
```
scatter_plot(df)
```



Z Score Method

Using a [z-score method (https://www.ctspedia.org/do/view/CTSpedia/OutLier#:~:text=Any%20z%2Dscore%20greater%20th](https://www.ctspedia.org/do/view/CTSpedia/OutLier#:~:text=Any%20z%2Dscore%20greater%20th) we will clean up the data by removing data points with z-score greater than 3. For this project, we can ignore checking data points with z-score less than 3 because ROI cannot be less than -100%.

In [37]:
```
#calculate z-score on ROI and create a new column to show the z-score
df['ROI_zscore'] = stats.zscore(df['ROI'])

#remove rows with z-score greater than 3
df = df[df['ROI_zscore'] < 3]

df
```

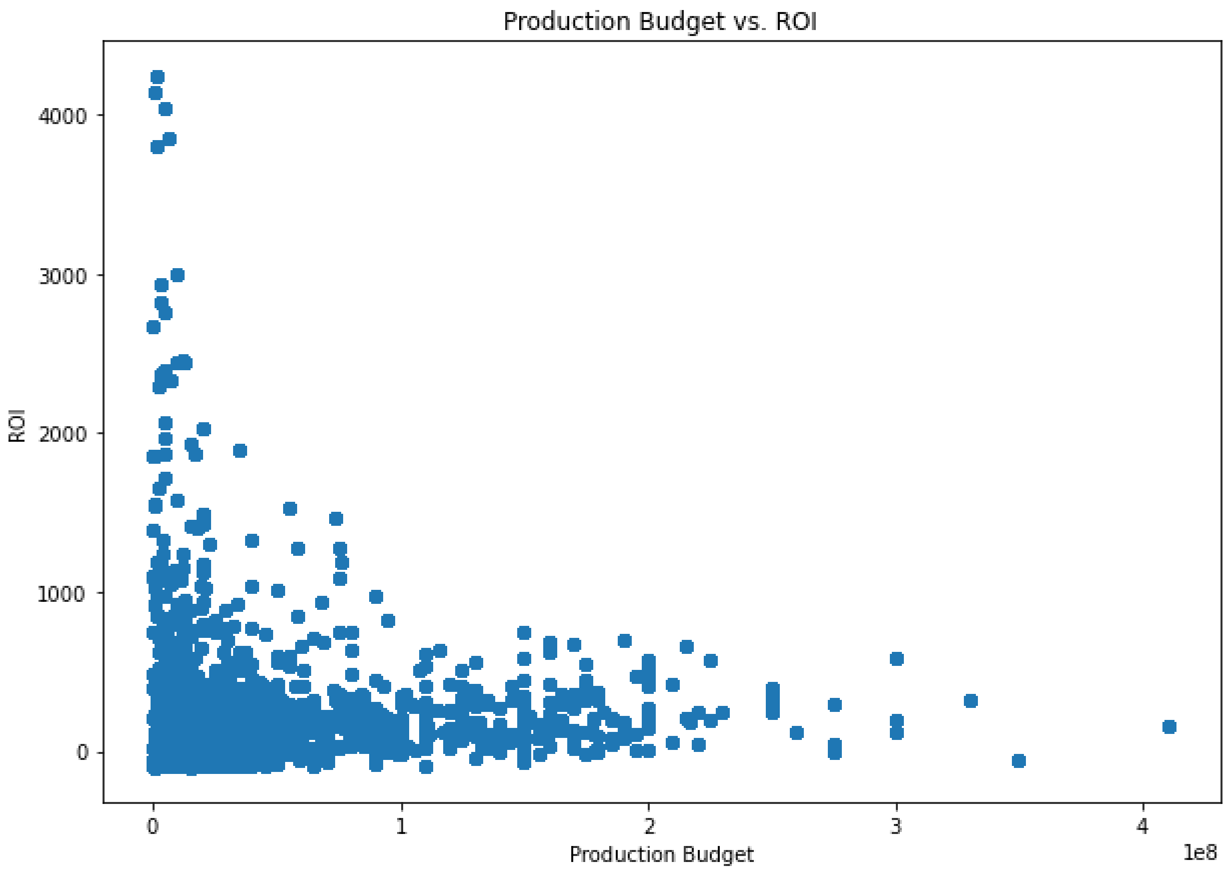
Out[37]:

	tconst	nconst	category	genres	release_date	movie	production_budget	wor
0	tt0249516	nm0257258	producer	Action	2012-12-31	Foodfight!	45000000	
0	tt0249516	nm0257258	producer	Animation	2012-12-31	Foodfight!	45000000	
0	tt0249516	nm0257258	producer	Comedy	2012-12-31	Foodfight!	45000000	
1	tt0249516	nm0240381	actor	Action	2012-12-31	Foodfight!	45000000	
1	tt0249516	nm0240381	actor	Animation	2012-12-31	Foodfight!	45000000	
...	...	...	...	...	...	...	...	
13141	tt9024106	nm2445956	producer	Drama	2019-03-29	Unplanned	6000000	
13142	tt9024106	nm7839151	producer	Biography	2019-03-29	Unplanned	6000000	
13142	tt9024106	nm7839151	producer	Drama	2019-03-29	Unplanned	6000000	
13143	tt9024106	nm0498920	producer	Biography	2019-03-29	Unplanned	6000000	
13143	tt9024106	nm0498920	producer	Drama	2019-03-29	Unplanned	6000000	

34149 rows × 12 columns

Now, vusualize the updated DataFrame after removing rows with z-score greater 3.

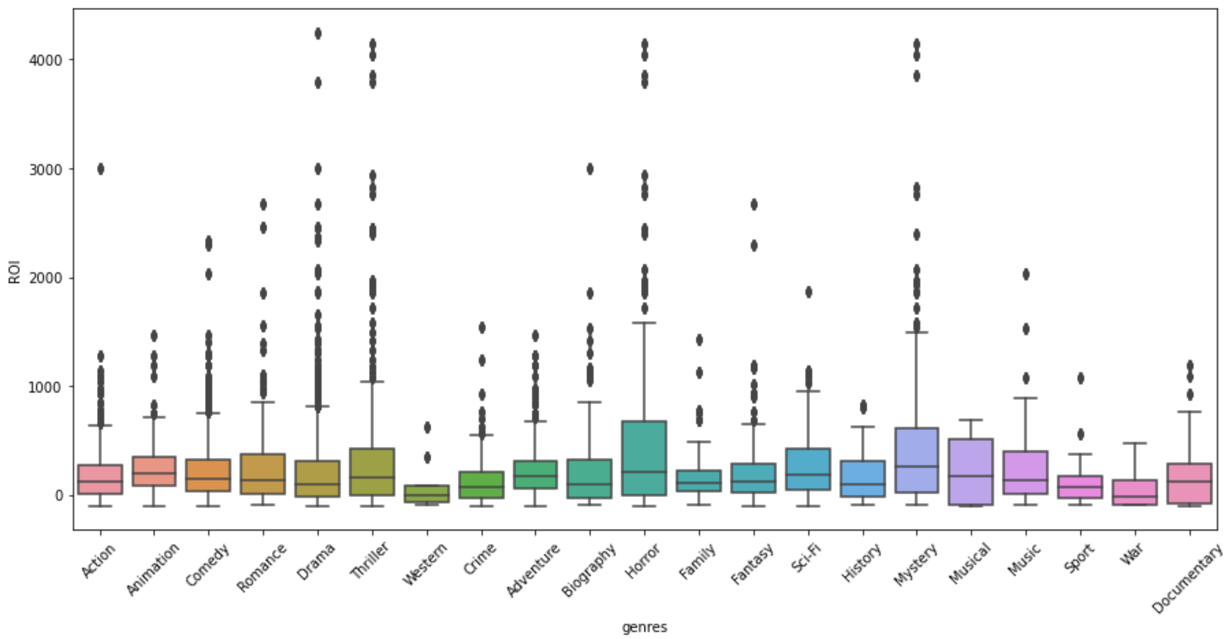
In [38]:
```
scatter_plot(df)
```



The scatter plot looks so much better! Now that we have cleaned up the overall DataFrame with z-score calculation on ROI, we want to examine the data more thoroughly. Let's take a look at ROI by genre - create a box plot with outliers.

In [39]:
```
#define a function to visualize box plots with outliers
def box_plot(df, x ='genres', y = 'ROI'):

    #create the plot
    f, ax = plt.subplots(figsize = (15,7))

    #Generate a box plot
    sns.boxplot(data = df,
           x = 'genres',
           y= 'ROI',
           showfliers = True)

    plt.xticks(rotation = 45)
```

In [40]:
```
box_plot(df)
```



1.5xIQR Rule

Here, we are going to use [1.5xIQR rule (https://www.khanacademy.org/math/statistics-probability/summarizing-quantitative-data/box-whisker-plots/a/identifying-outliers-iqr-rule#:~:text=A%20commonly%20used%20rule%20says,or%20below%20the%20first%20quartile.)](https://www.khanacademy.org/math/statistics-probability/summarizing-quantitative-data/box-whisker-plots/a/identifying-outliers-iqr-rule#:~:text=A%20commonly%20used%20rule%20says,or%20below%20the%20first%20quartile.), which defines a data point as an outlier if it is more than '1.5 * IQR' above the thrid quartile or less than '1.5 * IQR' below the first quartile. We can ignore outliers less than '1.5 * IQR' because ROI cannot be less than -100%.

Let's create a new DataFrame called iqr to calculate the upper threshold that will be used to identify the high outliers.

In [41]:
```
#create an empty DataFrame
iqr = pd.DataFrame()

#calculate Quartile 1
iqr['Q1'] = df.groupby('genres').ROI.quantile(.25)

#calculate Quartile 3
iqr['Q3'] = df.groupby('genres').ROI.quantile(.75)

#calculate interquartile range
iqr['IQR'] = iqr['Q3'] - iqr['Q1']

#calculate the upper threshold
iqr['upper_threshold'] = iqr['Q3'] + 1.5*iqr['IQR']

#check data
iqr
```

Out[41]:

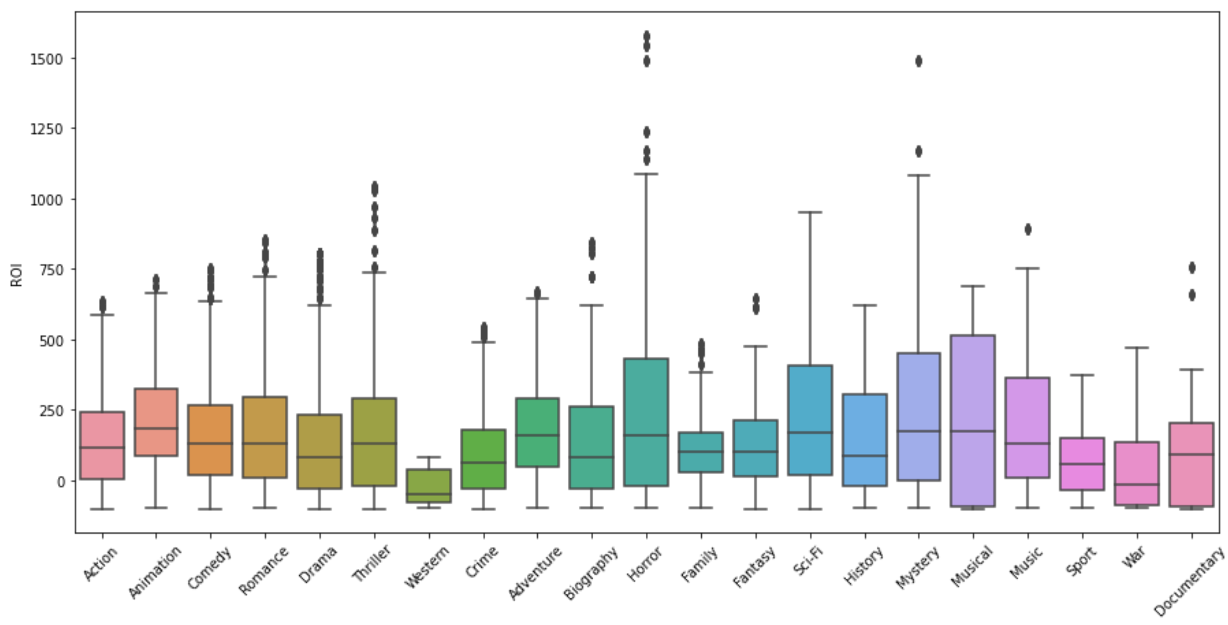
genres	Q1	Q3	IQR	upper_threshold
Action	12.515718	269.713673	257.197955	655.510605
Adventure	55.974192	306.801427	250.827235	683.042280
Animation	86.577085	345.300955	258.723870	733.386761
Biography	-26.218484	322.261705	348.480189	844.981989
Comedy	27.430332	319.246700	291.816368	756.971252
Crime	-26.138270	207.666667	233.804937	558.374072
Documentary	-79.944000	289.962850	369.906850	844.823125
Drama	-19.907396	311.539470	331.446865	808.709768
Family	32.972215	226.110043	193.137828	515.816785
Fantasy	16.348439	284.146742	267.798303	685.844196
History	-16.447875	306.539022	322.986897	791.019368
Horror	-11.086850	679.593740	690.680590	1715.614626
Music	13.301505	401.673606	388.372101	984.231757
Musical	-90.503464	512.379108	602.882573	1416.702968
Mystery	23.503274	613.105500	589.602226	1497.508839
Romance	12.515718	367.887600	355.371882	900.945423
Sci-Fi	42.489871	425.732893	383.243022	1000.597426
Sport	-32.963460	168.870141	201.833601	471.620543
Thriller	-2.476980	421.434029	423.911009	1057.300541
War	-89.894700	135.857985	225.752685	474.487013
Western	-72.948733	80.583507	153.532240	310.881867

In [42]:
```
#create a new column in df to map the upper threshold
df['iqr_upper_threshold'] = df['genres'].map(iqr['upper_threshold'])

#remove rows with the upper outliers
df = df[df['ROI'] < df['iqr_upper_threshold']]

#check data
df.head()
```

Out[42]:

	tconst	nconst	category	genres	release_date	movie	production_budget	worldwid
0	tt0249516	nm0257258	producer	Action	2012-12-31	Foodfight!	45000000	
0	tt0249516	nm0257258	producer	Animation	2012-12-31	Foodfight!	45000000	
0	tt0249516	nm0257258	producer	Comedy	2012-12-31	Foodfight!	45000000	
1	tt0249516	nm0240381	actor	Action	2012-12-31	Foodfight!	45000000	
1	tt0249516	nm0240381	actor	Animation	2012-12-31	Foodfight!	45000000	

In [43]:
```
box_plot(df)
```



There are still some outliers but this box plot looks much better after we cleaned up the DataFrame using 1.5xIQR rule.

Questions

Question 1: Which genres are the most profitable and yield the highest ROI?

Genre 1, Genre2, Genre3 win!

Question 2: What is the ideal budget range to yield the highest ROI?

Question 3: Who are the top casts and directors in those genres that Microsoft should work with?

Conclusions

Further Analysis

For More Information

In [6]:
```
ls zippedData/
```

```
bom.movie_gross.csv.gz          imdb.title.ratings.csv.gz
imdb.name.basics.csv.gz         rt.movie_info.tsv.gz
imdb.title.akas.csv.gz          rt.reviews.tsv.gz
imdb.title.basics.csv.gz        tmdb.movies.csv.gz
imdb.title.crew.csv.gz          tn.movie_budgets.csv.gz
imdb.title.principals.csv.gz
```