Microsoft Wants In

Overview

This project analyzes the resource needs of Microsoft, the world's largest software maker founded by Bill Gates and Paul Allen, to extend their already extensive repertoire in technology to include movie studios. Analysis shows that some specifics make more of an impact on ratings and gross earnings than others. Microsoft can use this analysis to break into the movies industry.

Business Problem

Microsoft, while very knowledgeable in the tech world, is new to the idea of making movies. Using information data collected from Box Office Mojo, IMDB, Rotten Tomatoes and The Movie DB, I describe patterns in movie genres, time of year movies are released and suggestions for the best writers and directors that create the best movies.

Data Understanding

The Datasets Used

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

In [2]:
```
! unzip -n zippedData/im.db.zip
```

```
Archive:  zippedData/im.db.zip
```

In [3]:
```
import sqlite3
```

In [4]:
```
conn = sqlite3.connect("im.db")
```

In [5]:
```
#Looking at tables in sql file to see what will be useful
schema_imdb = pd.read_sql("""SELECT *FROM sqlite_master""", conn)
schema_imdb
```

Out[5]:

	type	name	tbl_name	rootpage	sql
0	table	movie_basics	movie_basics	2	CREATE TABLE "movie_basics" (\n"movie_id" TEXT...
1	table	directors	directors	3	CREATE TABLE "directors" (\n"movie_id" TEXT,\n...
2	table	known_for	known_for	4	CREATE TABLE "known_for" (\n"person_id" TEXT,\...
3	table	movie_akas	movie_akas	5	CREATE TABLE "movie_akas" (\n"movie_id" TEXT,\...
4	table	movie_ratings	movie_ratings	6	CREATE TABLE "movie_ratings" (\n"movie_id" TEX...
5	table	persons	persons	7	CREATE TABLE "persons" (\n"person_id" TEXT,\n ...
6	table	principals	principals	8	CREATE TABLE "principals" (\n"movie_id" TEXT,\...
7	table	writers	writers	9	CREATE TABLE "writers" (\n"movie_id" TEXT,\n ...

In [6]:
```
tmdb_movie_info = pd.read_csv("zippedData/tmdb.movies.csv.gz",encoding="latir
tmdb_movie_info.head()
```

Out[6]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	H P and De Hall P
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	Ho Dra
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	S
4	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Ince

In [7]:
```
budget_info = pd.read_csv("zippedData/tn.movie_budgets.csv.gz",encoding="lati
budget_info.head()
```

Out[7]:

	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 [8]:
```
movie_basics = pd.read_sql("SELECT * FROM movie_basics;", conn)
movie_basics.head()
```

Out[8]:

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

In [9]:
```
movie_ratings = pd.read_sql("SELECT * FROM movie_ratings;", conn)
movie_ratings.head()
```

Out[9]:

	movie_id	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21

In [10]:
```
persons = pd.read_sql("SELECT * FROM persons;", conn)
persons.head()
```

Out[10]:

	person_id	primary_name	birth_year	death_year	primary_prof
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,pr
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_depa
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,actc
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_depa
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_de

In [11]:
```
directors = pd.read_sql("SELECT * FROM directors;", conn)
directors.head()
```

Out[11]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0462036	nm1940585
2	tt0835418	nm0151540
3	tt0835418	nm0151540
4	tt0878654	nm0089502

In [12]:
```
writers = pd.read_sql("SELECT * FROM writers;", conn)
writers.head()
```

Out[12]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0438973	nm0175726
2	tt0438973	nm1802864
3	tt0462036	nm1940585
4	tt0835418	nm0310087

Cleaning Data

I first made copies of all the files I used.

In [13]:
```
budget_info_clean1=budget_info.copy()
budget_info_clean1.head()
```

Out[13]:

	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 [14]:
```
writers_copy=writers.copy()
writers_copy.head()
```

Out[14]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0438973	nm0175726
2	tt0438973	nm1802864
3	tt0462036	nm1940585
4	tt0835418	nm0310087

In [15]:
```
persons_copy=persons.copy()
persons_copy.head()
```

Out[15]:

	person_id	primary_name	birth_year	death_year	primary_prof
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,pr
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_depa
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,acto
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_depa
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_de

In [16]:
```
movie_basics_copy=movie_basics.copy()
movie_basics_copy.head()
```

Out[16]:

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

In [17]:
```
movie_ratings_copy=movie_ratings.copy()
movie_ratings.head()
```

Out[17]:

	movie_id	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21

In [18]:
```
directors_copy = directors.copy()
directors_copy.head()
```

Out[18]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0462036	nm1940585
2	tt0835418	nm0151540
3	tt0835418	nm0151540
4	tt0878654	nm0089502

In [19]:

```
tmdb_movie_info_clean= tmdb_movie_info.copy()
tmdb_movie_info_clean.head()
```

Out[19]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	H P anc De Hall P
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	Ho - Dra
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	S
4	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Ince

Movie Genre

I dropped some columns and rearranged the remaining

In [94]:
```
clist=['id','title','popularity','vote_average','vote_count','release_date','
tmdb_movie_info_clean=tmdb_movie_info_clean[clist]
tmdb_movie_info_clean.head()
```

Out[94]:

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

Seperated and created a column for genres.

In [21]:

```
#create column for seperated genres
tmdb_movie_info_clean['genre_id']=tmdb_movie_info_clean['genre_ids']
tmdb_movie_info_clean['genre_lst'] = tmdb_movie_info_clean['genre_ids'].str.s
df_explode = tmdb_movie_info_clean.explode('genre_lst')
df_explode.head()
```

Out[21]:

	id	title	popularity	vote_average	vote_count	release_date	genre_ids	genre_id	g
0	12444	Harry Potter and the Deathly Hallows: Part 1	33.533	7.7	10788	2010-11-19	[12, 14, 10751]	[12, 14, 10751]	
0	12444	Harry Potter and the Deathly Hallows: Part 1	33.533	7.7	10788	2010-11-19	[12, 14, 10751]	[12, 14, 10751]	
0	12444	Harry Potter and the Deathly Hallows: Part 1	33.533	7.7	10788	2010-11-19	[12, 14, 10751]	[12, 14, 10751]	
1	10191	How to Train Your Dragon	28.734	7.7	7610	2010-03-26	[14, 12, 16, 10751]	[14, 12, 16, 10751]	
1	10191	How to Train Your Dragon	28.734	7.7	7610	2010-03-26	[14, 12, 16, 10751]	[14, 12, 16, 10751]	

In [22]:
```
df_explode['genre_lst'].str.strip(' ')
df_explode.head()
```

Out[22]:

	id	title	popularity	vote_average	vote_count	release_date	genre_ids	genre_id	g
0	12444	Harry Potter and the Deathly Hallows: Part 1	33.533	7.7	10788	2010-11-19	[12, 14, 10751]	[12, 14, 10751]	
0	12444	Harry Potter and the Deathly Hallows: Part 1	33.533	7.7	10788	2010-11-19	[12, 14, 10751]	[12, 14, 10751]	
0	12444	Harry Potter and the Deathly Hallows: Part 1	33.533	7.7	10788	2010-11-19	[12, 14, 10751]	[12, 14, 10751]	
1	10191	How to Train Your Dragon	28.734	7.7	7610	2010-03-26	[14, 12, 16, 10751]	[14, 12, 16, 10751]	
1	10191	How to Train Your Dragon	28.734	7.7	7610	2010-03-26	[14, 12, 16, 10751]	[14, 12, 16, 10751]	

Checking that there are no null values.

In [23]:
```
df_explode.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 47834 entries, 0 to 26516
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   id            47834 non-null  int64  
 1   title         47834 non-null  object 
 2   popularity    47834 non-null  float64
 3   vote_average  47834 non-null  float64
 4   vote_count    47834 non-null  int64  
 5   release_date  47834 non-null  object 
 6   genre_ids     47834 non-null  object 
 7   genre_id      47834 non-null  object 
 8   genre_lst     47834 non-null  object 
dtypes: float64(2), int64(2), object(5)
memory usage: 3.6+ MB
```

Comparing Genre to Rating

I wanted to change the genre id numbers to the actual name of each genre, but could not, so I created a table to use as a key.

In [24]:
```
genre_labels={'genre_num':['28','12','16','35','80','99','18','10751','14','3
                                                                  '9648','10749','878
                           '53','10753','37'],'genre_titles': ['action','adver
                                                                     'documentary','c
                                                                     'horror','music'
                                                                     'tv movie','thri
genre_labels=pd.DataFrame(genre_labels)
genre_labels
```

Out[24]:

	genre_num	genre_titles
0	28	action
1	12	adventure
2	16	animation
3	35	comedy
4	80	crime
5	99	documentary
6	18	drama
7	10751	family
8	14	fantasy
9	36	history
10	27	horror
11	10402	music
12	9648	mystery
13	10749	romance
14	878	science fiction
15	10770	tv movie
16	53	thriller
17	10753	war
18	37	western

Change vote_average type from float64 to object

In [25]:
```
df_explode['vote_average'].astype('object')
```

Out[25]:
```
0        7.7
0        7.7
0        7.7
1        7.7
1        7.7
        ...
26515    0.0
26515    0.0
26515    0.0
26516    0.0
26516    0.0
Name: vote_average, Length: 47834, dtype: object
```

In [26]:
```
genre_rating=df_explode.groupby('genre_lst').mean(['vote_average'])
genre_rating.head()
```

Out[26]:

genre_lst	id	popularity	vote_average	vote_count
	319733.985075	0.759605	6.059863	2.013715
10402	308250.127334	2.904005	6.924109	131.269949
10749	249634.170252	4.456935	6.019115	302.744315
10751	250468.954792	5.464077	6.089512	517.461121
10752	258463.533040	5.741441	6.318943	440.259912

Dropped columns I didn't need

In [27]:
```
clist=['vote_average','popularity','genre_lst']
genre_rating=df_explode[clist]
genre_rating.head()
```

Out[27]:

	vote_average	popularity	genre_lst
0	7.7	33.533	12
0	7.7	33.533	14
0	7.7	33.533	10751
1	7.7	28.734	14
1	7.7	28.734	12

Sorted table by vote avg from highest to lowest

In [28]:
```
tmdb_movie_vote_avg=genre_rating.sort_values('vote_average', ascending=False)
tmdb_movie_vote_avg.head()
```

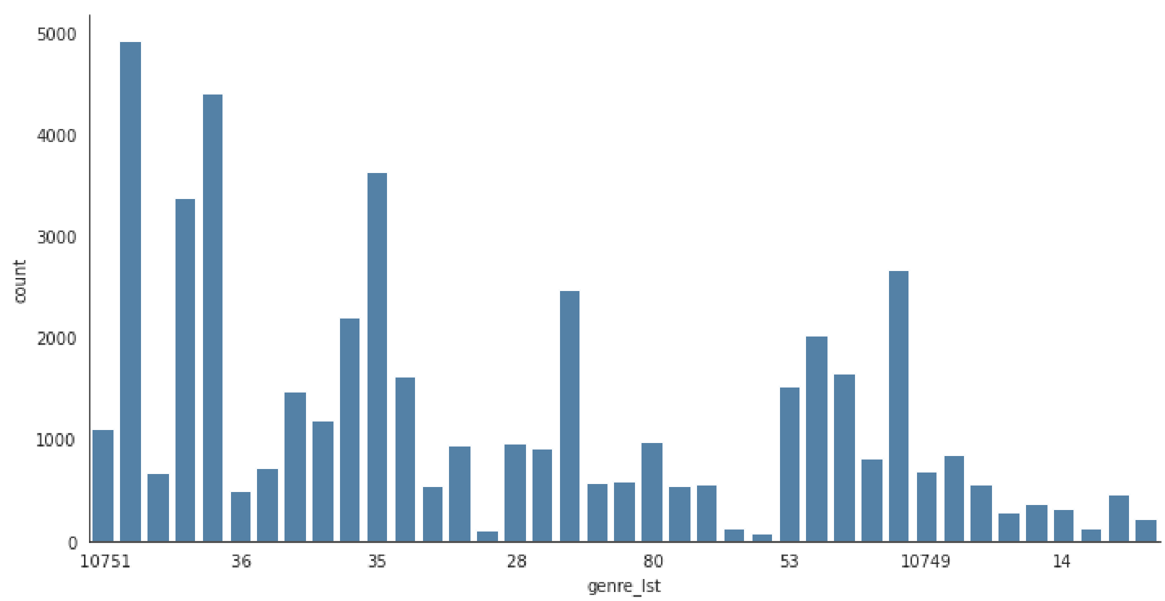
Out[28]:

	vote_average	popularity	genre_lst
9198	10.0	1.40	10751
23022	10.0	0.64	18
23023	10.0	0.64	10402
23023	10.0	0.64	18
23024	10.0	0.64	10402

Created a bar plot that shows popularity of each genre

In [29]:
```
with sns.axes_style('white'):
    g = sns.factorplot("genre_lst", data=tmdb_movie_vote_avg, aspect=2,
                       kind="count", color='steelblue')
    g.set_xticklabels(step=5)
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/categorical.py:3714: UserWarning: The `factorplot` function has been renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
/opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Comparing Genre to Return on Investment

Dropping columns not needed

In [30]:
```
#dropping unwanted columns
clist=['genre_lst','title']
genre_lst=df_explode[clist]
genre_lst.head()
```

Out[30]:

	genre_lst	title
0	12	Harry Potter and the Deathly Hallows: Part 1
0	14	Harry Potter and the Deathly Hallows: Part 1
0	10751	Harry Potter and the Deathly Hallows: Part 1
1	14	How to Train Your Dragon
1	12	How to Train Your Dragon

Checking for null values and dtypes.

In [31]:
```
budget_info_clean1.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
```

Changed worldwide_gross and production_budget from object to float in order to do some mathmatical operations.

In [32]:
```
budget_info_clean1['worldwide_gross']=budget_info_clean1['worldwide_gross'].a
budget_info_clean1.head()
```

Out[32]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	2.776345e+09
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	1.045664e+09
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	1.497624e+08
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	1.403014e+09
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	1.316722e+09

In [33]:
```
budget_info_clean1['production_budget']=budget_info_clean1['production_budget
budget_info_clean1.head()
```

Out[33]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	425000000.0	$760,507,625	2.776345e+09
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000.0	$241,063,875	1.045664e+09
2	3	Jun 7, 2019	Dark Phoenix	350000000.0	$42,762,350	1.497624e+08
3	4	May 1, 2015	Avengers: Age of Ultron	330600000.0	$459,005,868	1.403014e+09
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000.0	$620,181,382	1.316722e+09

Calculated the return on investment (roi) by subtracting the production budget from worldwide gross.

In [34]:
```
#calculate roi
roi=budget_info_clean1['worldwide_gross']-budget_info_clean1['production_budg
roi.head()
```

Out[34]:
```
0    2.351345e+09
1    6.350639e+08
2   -2.002376e+08
3    1.072414e+09
4    9.997217e+08
dtype: float64
```

Convert roi number into millions and added column to table.

In [35]:
```
budget_info_clean1["roi_in_mils"]=(budget_info_clean1['worldwide_gross']-budg
budget_info_clean1.head()
```

Out[35]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	roi_in_
0	1	Dec 18, 2009	Avatar	425000000.0	$760,507,625	2.776345e+09	2351.345
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000.0	$241,063,875	1.045664e+09	635.063
2	3	Jun 7, 2019	Dark Phoenix	350000000.0	$42,762,350	1.497624e+08	-200.237
3	4	May 1, 2015	Avengers: Age of Ultron	330600000.0	$459,005,868	1.403014e+09	1072.413
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000.0	$620,181,382	1.316722e+09	999.721

In [36]:
```
clist=['id','movie','roi_in_mils',]
budget_roi=budget_info_clean1[clist]
budget_roi.head()
```

Out[36]:

	id	movie	roi_in_mils
0	1	Avatar	2351.345279
1	2	Pirates of the Caribbean: On Stranger Tides	635.063875
2	3	Dark Phoenix	-200.237650
3	4	Avengers: Age of Ultron	1072.413963
4	5	Star Wars Ep. VIII: The Last Jedi	999.721747

Combining tables to have roi and genres on same table

In [37]:
```
tmbd_roi_merge=pd.merge(genre_lst,budget_roi,left_on=['title'],right_on=['mov
tmbd_roi_merge.head()
```

Out[37]:

	genre_lst	title	id	movie	roi_in_mils
0	14	How to Train Your Dragon	30	How to Train Your Dragon	329.870992
1	12	How to Train Your Dragon	30	How to Train Your Dragon	329.870992
2	16	How to Train Your Dragon	30	How to Train Your Dragon	329.870992
3	10751	How to Train Your Dragon	30	How to Train Your Dragon	329.870992
4	12	Iron Man 2	15	Iron Man 2	451.156389

Calculate average roi of each genre

In [93]:
```
genre_roi=tmbd_roi_merge.groupby('genre_lst').mean(['roi_in_mils'])
genre_roi.head()
```

Out[93]:

	id	roi_in_mils
genre_lst		
	52.461538	32.617812
10402	56.620000	60.610604
10749	52.819820	69.821652
10751	47.188776	207.414240
10752	44.428571	58.882295

In [39]:
```
clist=['genre_lst','roi_in_mils']
genre_roi=tmbd_roi_merge[clist]
genre_roi.head()
```

Out[39]:

	genre_lst	roi_in_mils
0	14	329.870992
1	12	329.870992
2	16	329.870992
3	10751	329.870992
4	12	451.156389

In [40]:
```
genre_roi_ordered=genre_roi.sort_values(['roi_in_mils'],ascending=False)
genre_roi_ordered.head()
```

Out[40]:

	genre_lst	roi_in_mils
21	14	2351.345279
22	878	2351.345279
20	12	2351.345279
19	28	2351.345279
5328	12	1748.134200

Dropped duplicate rows that may skew my numbers.

In [41]:
```
df2 = genre_roi_ordered.drop_duplicates(keep='first')
df2.head()
```

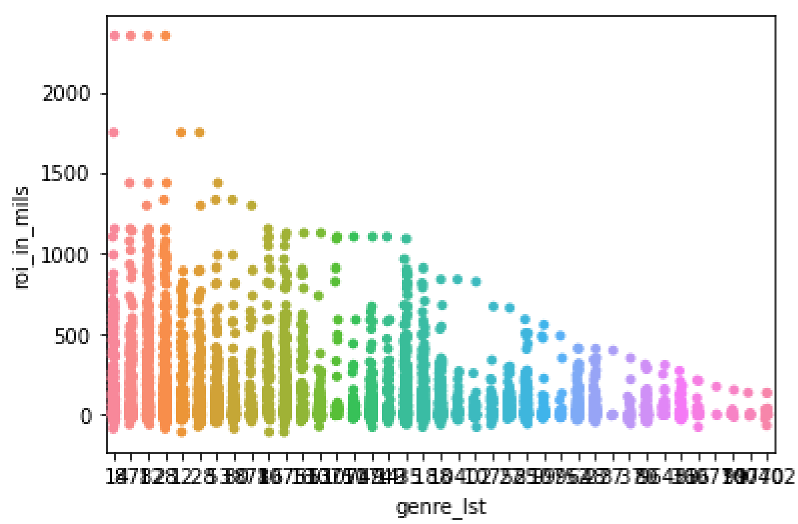
Out[41]:

	genre_lst	roi_in_mils
21	14	2351.345279
22	878	2351.345279
20	12	2351.345279
19	28	2351.345279
5328	12	1748.134200

Created a scatter plot showing the roi for each genre.

In [42]:
```
sns.stripplot(x='genre_lst', y='roi_in_mils', data=df2)
```

Out[42]: <AxesSubplot:xlabel='genre_lst', ylabel='roi_in_mils'>



Analysis

My data shows that even though fantasy, adventure and science fiction are generally the most expensive genres to produce, they are the ones that make the most money. Where family, comedy and history were shown to be the most popular.

Month of Movie Release

Comparing Return on Investment to the Release Month

Converting realease_date to datetime.

In [43]:
```
budget_info_clean1['rel_date']=pd.to_datetime(budget_info_clean1['release_dat
budget_info_clean1.head()
```

Out[43]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	roi_in_
0	1	Dec 18, 2009	Avatar	425000000.0	$760,507,625	2.776345e+09	2351.345
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000.0	$241,063,875	1.045664e+09	635.063
2	3	Jun 7, 2019	Dark Phoenix	350000000.0	$42,762,350	1.497624e+08	-200.237
3	4	May 1, 2015	Avengers: Age of Ultron	330600000.0	$459,005,868	1.403014e+09	1072.413
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000.0	$620,181,382	1.316722e+09	999.721

Breaking down the date to extact month.

In [44]:
```
budget_info_clean1['release_month']=budget_info_clean1['rel_date'].dt.month
budget_info_clean1.head()
```

Out[44]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	roi_in_
0	1	Dec 18, 2009	Avatar	425000000.0	$760,507,625	2.776345e+09	2351.345
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000.0	$241,063,875	1.045664e+09	635.063
2	3	Jun 7, 2019	Dark Phoenix	350000000.0	$42,762,350	1.497624e+08	-200.237
3	4	May 1, 2015	Avengers: Age of Ultron	330600000.0	$459,005,868	1.403014e+09	1072.413
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000.0	$620,181,382	1.316722e+09	999.721

Creating tables with desired columns.

In [45]:
```
clist=['movie','roi_in_mils']
movie_roi=budget_info_clean1[clist]
movie_roi.head()
```

Out[45]:

	movie	roi_in_mils
0	Avatar	2351.345279
1	Pirates of the Caribbean: On Stranger Tides	635.063875
2	Dark Phoenix	-200.237650
3	Avengers: Age of Ultron	1072.413963
4	Star Wars Ep. VIII: The Last Jedi	999.721747

In [46]:
```
clist=['release_month','roi_in_mils']
month_roi=budget_info_clean1[clist]
month_roi.head(12)
```

Out[46]:

	release_month	roi_in_mils
0	12	2351.345279
1	5	635.063875
2	6	-200.237650
3	5	1072.413963
4	12	999.721747
5	12	1747.311220
6	4	1748.134200
7	5	663.420425
8	11	355.945209
9	11	579.620923
10	7	809.439099
11	5	118.151347

Combining rows and getting total income for each month.

In [92]:
```
df_new = month_roi.groupby(month_roi['release_month']).aggregate({'roi_in_mil
df_new.head()
```

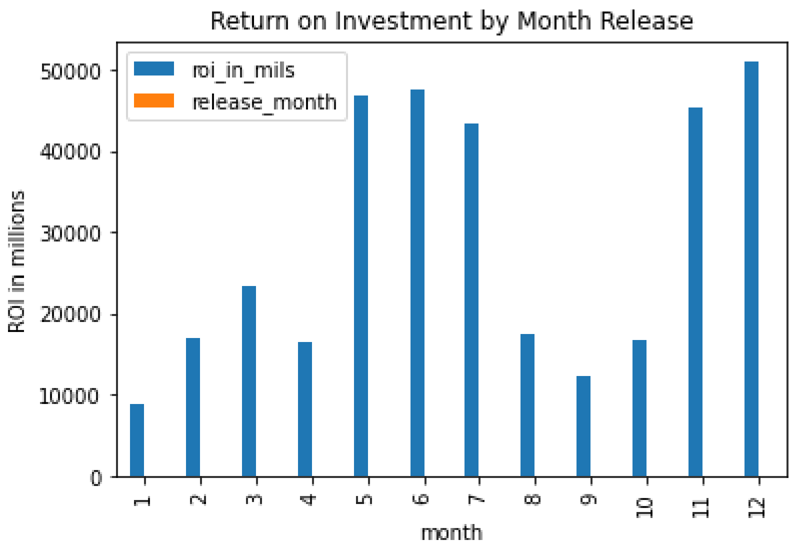
Out[92]:

	roi_in_mils	release_month
release_month		
1	8924.955936	1
2	17051.257874	2
3	23430.107410	3
4	16397.312390	4
5	46859.053019	5

Created a bar plot to show the return of investment for each month

In [48]:
```
df_new.plot(kind="bar")
plt.title("Return on Investment by Month Release")
plt.xlabel("month")
plt.ylabel("ROI in millions")
```

Out[48]: Text(0, 0.5, 'ROI in millions')



Comparing the Release Month to Ratings

In [49]:
```
clist=['release_month','movie']
month_movie=budget_info_clean1[clist]
month_movie.head(12)
```

Out[49]:

	release_month	movie
0	12	Avatar
1	5	Pirates of the Caribbean: On Stranger Tides
2	6	Dark Phoenix
3	5	Avengers: Age of Ultron
4	12	Star Wars Ep. VIII: The Last Jedi
5	12	Star Wars Ep. VII: The Force Awakens
6	4	Avengers: Infinity War
7	5	Pirates of the Caribbean: At WorldÃ¢Â Â s End
8	11	Justice League
9	11	Spectre
10	7	The Dark Knight Rises
11	5	Solo: A Star Wars Story

In [50]:
```
movie_basics_copy['movie']=movie_basics_copy['primary_title']
clist=['movie','movie_id']
movie_basics_clean=movie_basics_copy[clist]
movie_basics_clean.head()
```

Out[50]:

	movie	movie_id
0	Sunghursh	tt0063540
1	One Day Before the Rainy Season	tt0066787
2	The Other Side of the Wind	tt0069049
3	Sabse Bada Sukh	tt0069204
4	The Wandering Soap Opera	tt0100275

In [51]:
```
movie_ratings_copy.rename(columns = {'averagerating':'average_rating'}, inpla
movie_ratings_copy.columns
```

Out[51]:
```
Index(['movie_id', 'average_rating', 'numvotes'], dtype='object')
```

In [52]:

```
clist=['average_rating','movie_id']
movie_ratings_clean=movie_ratings_copy[clist]
movie_ratings_clean.head()
```

Out[52]:

	average_rating	movie_id
0	8.3	tt10356526
1	8.9	tt10384606
2	6.4	tt1042974
3	4.2	tt1043726
4	6.5	tt1060240

Combine the 3 previous tables to get ratings with month release.

In [53]:

```
movie_ratings_month_merge=pd.merge(pd.merge(movie_ratings_clean,movie_basics_
movie_ratings_month_merge.head()
```

Out[53]:

	average_rating	movie_id	movie	release_month
0	4.2	tt1043726	The Legend of Hercules	1
1	7.0	tt1094666	The Hammer	3
2	6.5	tt3096900	The Hammer	3
3	5.1	tt1171222	Baggage Claim	9
4	7.6	tt1210166	Moneyball	9

In [54]:

```
clist=['average_rating','release_month']
month_rating=movie_ratings_month_merge[clist]
month_rating.head()
```

Out[54]:

	average_rating	release_month
0	4.2	1
1	7.0	3
2	6.5	3
3	5.1	9
4	7.6	9

In [55]:
```
month_rating_ordered=month_rating.sort_values(['average_rating'],ascending=Fa
month_rating_ordered.head()
```

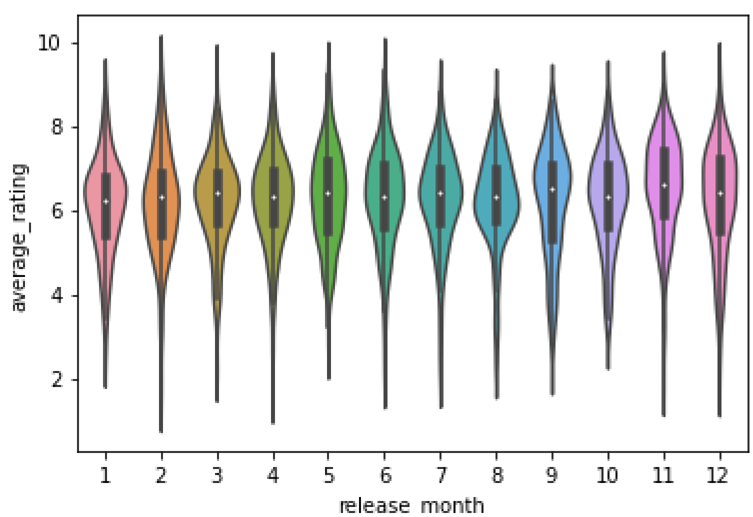
Out[55]:

	average_rating	release_month
329	9.3	2
330	9.3	6
211	9.2	3
1118	9.2	5
1712	9.2	12

Created a violin plot to show the average movie ratings for each month

In [56]:
```
sns.violinplot(x='release_month', y='average_rating', data=month_rating)
```

Out[56]: <AxesSubplot:xlabel='release_month', ylabel='average_rating'>



Analysis

The top 3 release months to earn the most money and had the best ratings are: May, June, and December.

Most Lucrative Writers and Directors

WRITERS & DIRECTORS
WORLDWIDE

Comparing Return on Investment to Writers and Directors

In [57]:

```
persons_copy['primary_professions']=persons_copy['primary_profession']
persons_copy['primary_professions'] =persons_copy['primary_profession'].str.s
df_explode =persons_copy.explode('primary_professions')
df_explode.head()
```

Out[57]:

	person_id	primary_name	birth_year	death_year	primary_professic
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,produc
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,produc
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,produc
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_departme
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_departme

In [58]:
```
df_explode.dropna()
```

Out[58]:

	person_id	primary_name	birth_year	death_year	primary_profession	primary_profes
32	nm0071116	Valérie Benguigui	1961.0	2013.0	actress,soundtrack	a
32	nm0071116	Valérie Benguigui	1961.0	2013.0	actress,soundtrack	sounc
38	nm0073426	Laxmikant Berde	1954.0	2004.0	actor	
62	nm0083767	Fernando Birri	1925.0	2017.0	director,actor,writer	di
62	nm0083767	Fernando Birri	1925.0	2017.0	director,actor,writer	
...	...	...	...	...	...	
600210	nm9211845	Jan C. Gabriel	1940.0	2010.0	director,writer,editor	
602878	nm7455311	Joost van der Westhuizen	1971.0	2017.0	producer	pro
603895	nm8201131	Lewis Lucky Carrillo III	1968.0	2017.0	actor,producer	
603895	nm8201131	Lewis Lucky Carrillo III	1968.0	2017.0	actor,producer	pro
604364	nm8659676	Zygmunt Bauman	1925.0	2017.0	writer	

11868 rows × 6 columns

In [59]:
```
df_explode.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1140331 entries, 0 to 606647
Data columns (total 6 columns):
 #   Column                Non-Null Count    Dtype  
---  ------                --------------    -----  
 0   person_id             1140331 non-null  object 
 1   primary_name          1140331 non-null  object 
 2   birth_year            184940 non-null   float64
 3   death_year            13538 non-null    float64
 4   primary_profession    1088991 non-null  object 
 5   primary_professions   1088991 non-null  object 
dtypes: float64(2), object(4)
memory usage: 60.9+ MB
```

Fill nan spaces in primary_professions in order to pull specific titles.

In [60]:
```
df_explode.primary_professions = df_explode.primary_professions.fillna('unknc
```

In [61]:
```
df_explode.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1140331 entries, 0 to 606647
Data columns (total 6 columns):
 #   Column               Non-Null Count    Dtype  
---  ------               --------------    -----  
 0   person_id            1140331 non-null  object 
 1   primary_name         1140331 non-null  object 
 2   birth_year           184940 non-null   float64
 3   death_year           13538 non-null    float64
 4   primary_profession   1088991 non-null  object 
 5   primary_professions  1140331 non-null  object 
dtypes: float64(2), object(4)
memory usage: 60.9+ MB
```

Dropped columns I didn't need

In [62]:
```
clist=['person_id','primary_name','primary_professions']
names_id=df_explode[clist]
names_id.head()
```

Out[62]:

	person_id	primary_name	primary_professions
0	nm0061671	Mary Ellen Bauder	miscellaneous
0	nm0061671	Mary Ellen Bauder	production_manager
0	nm0061671	Mary Ellen Bauder	producer
1	nm0061865	Joseph Bauer	composer
1	nm0061865	Joseph Bauer	music_department

Pulled out the writers and directors names

In [90]:
```
writer_names=names_id[names_id['primary_professions'].str.contains("writer")]
writer_names.head()
```

Out[90]:

	person_id	primary_name	primary_professions
2	nm0062070	Bruce Baum	writer
10	nm0064023	Bryan Beasley	writer
12	nm0065847	Michael Frost Beckner	writer
15	nm0066163	Arnaud Bedouët	writer
18	nm0067234	Hans Beimler	writer

In [91]:
```
director_names=names_id[names_id['primary_professions'].str.contains("directo
director_names.head()
```

Out[91]:

	person_id	primary_name	primary_professions
5	nm0062879	Ruel S. Bayani	director
10	nm0064023	Bryan Beasley	director
15	nm0066163	Arnaud Bedouët	director
16	nm0066268	Steve Mitchell Beebe	director
21	nm0068170	Dylan Bell	director

Merged 3 tables to join writer and director names with movie titles.

In [65]:
```
writer_movies_merge=pd.merge(pd.merge(writer_names,writers_copy,on='person_ic
writer_movies_merge.head()
```

Out[65]:

	person_id	primary_name	primary_professions	movie_id	primary_title	original_title	sta
0	nm0064023	Bryan Beasley	writer	tt3501180	The Quiet Philanthropist: The Edith Gaylord Story	The Quiet Philanthropist: The Edith Gaylord Story	
1	nm0065847	Michael Frost Beckner	writer	tt6349302	Sniper: Ultimate Kill	Sniper: Ultimate Kill	
2	nm0508052	Crash Leyland	writer	tt6349302	Sniper: Ultimate Kill	Sniper: Ultimate Kill	
3	nm0369675	Chris Hauty	writer	tt6349302	Sniper: Ultimate Kill	Sniper: Ultimate Kill	
4	nm0068874	Hava Kohav Beller	writer	tt7701650	In the Land of Pomegranates	In the Land of Pomegranates	

In [66]:

```
director_movies_merge=pd.merge(pd.merge(director_names,directors_copy,on='per
director_movies_merge.head()
```

Out[66]:

	person_id	primary_name	primary_professions	movie_id	primary_title	original_title	start_
0	nm0062879	Ruel S. Bayani	director	tt1592569	Paano na kaya	Paano na kaya	
1	nm0062879	Ruel S. Bayani	director	tt1592569	Paano na kaya	Paano na kaya	
2	nm0062879	Ruel S. Bayani	director	tt1592569	Paano na kaya	Paano na kaya	
3	nm0062879	Ruel S. Bayani	director	tt1592569	Paano na kaya	Paano na kaya	
4	nm0062879	Ruel S. Bayani	director	tt8421806	Kasal	Kasal	

Dropped colunms not needed

In [67]:

```
clist=['primary_name','movie']
writer_movie=writer_movies_merge[clist]
writer_movie.head()
```

Out[67]:

	primary_name	movie
0	Bryan Beasley	The Quiet Philanthropist: The Edith Gaylord Story
1	Michael Frost Beckner	Sniper: Ultimate Kill
2	Crash Leyland	Sniper: Ultimate Kill
3	Chris Hauty	Sniper: Ultimate Kill
4	Hava Kohav Beller	In the Land of Pomegranates

In [68]:

```
clist=['primary_name','movie']
director_movie=director_movies_merge[clist]
director_movie.head()
```

Out[68]:

	primary_name	movie
0	Ruel S. Bayani	Paano na kaya
1	Ruel S. Bayani	Paano na kaya
2	Ruel S. Bayani	Paano na kaya
3	Ruel S. Bayani	Paano na kaya
4	Ruel S. Bayani	Kasal

Dropped duplicates

In [69]:

```
writer_movie.drop_duplicates(keep="first")
```

Out[69]:

	primary_name	movie
0	Bryan Beasley	The Quiet Philanthropist: The Edith Gaylord Story
1	Michael Frost Beckner	Sniper: Ultimate Kill
2	Crash Leyland	Sniper: Ultimate Kill
3	Chris Hauty	Sniper: Ultimate Kill
4	Hava Kohav Beller	In the Land of Pomegranates
...	...	...
220808	Andrew Whaley	The Envelope
220809	Subrata Samanta Roy	PREM PARINOTI
220810	Rich Allen	Home Cookin: 5.17.18
220811	Elina Gakou Gomba	Le choc du futur
220812	Samir Eshra	The Shadow Lawyers

156052 rows × 2 columns

In [70]:
```
director_movie.drop_duplicates(keep='first')
```

Out[70]:

	primary_name	movie
0	Ruel S. Bayani	Paano na kaya
4	Ruel S. Bayani	Kasal
6	Ruel S. Bayani	No Other Woman
9	Ruel S. Bayani	One More Try
10	Bryan Beasley	Not Such a Bad Guy: Conversations with Dabney ...
...	...	...
280552	Rich Allen	Home Cookin: 5.17.18
280553	Zheng Wei	The Old Road
280554	Rama Narayanan	Chain Jayapal
280556	Rama Narayanan	Arya Suriya
280557	Samir Eshra	The Shadow Lawyers

150254 rows × 2 columns

In [71]:
```
writer_movie_merge=pd.merge(writer_movie,budget_roi,on=['movie'],how='inner')
writer_movie_merge.head()
```

Out[71]:

	primary_name	movie	id	roi_in_mils
0	David Bowers	Diary of a Wimpy Kid: The Long Haul	27	13.609577
1	Jeff Kinney	Diary of a Wimpy Kid: The Long Haul	27	13.609577
2	Francesco Bruni	Slam	53	0.087521
3	Nick Hornby	Slam	53	0.087521
4	Ludovica Rampoldi	Slam	53	0.087521

In [72]:
```
director_movie_merge=pd.merge(director_movie,budget_roi,on=['movie'],how='inr
director_movie_merge.head()
```

Out[72]:

	primary_name	movie	id	roi_in_mils
0	David Bowers	Diary of a Wimpy Kid: Rodrick Rules	80	55.695194
1	David Bowers	Diary of a Wimpy Kid: Rodrick Rules	80	55.695194
2	David Bowers	Diary of a Wimpy Kid: Rodrick Rules	80	55.695194
3	David Bowers	Diary of a Wimpy Kid: The Long Haul	27	13.609577
4	David Bowers	Diary of a Wimpy Kid: The Long Haul	27	13.609577

In [73]:
```
writer_movie_merge.drop_duplicates(keep="first")
writer_movie_merge.head()
```

Out[73]:

	primary_name	movie	id	roi_in_mils
0	David Bowers	Diary of a Wimpy Kid: The Long Haul	27	13.609577
1	Jeff Kinney	Diary of a Wimpy Kid: The Long Haul	27	13.609577
2	Francesco Bruni	Slam	53	0.087521
3	Nick Hornby	Slam	53	0.087521
4	Ludovica Rampoldi	Slam	53	0.087521

Merged tables to get names with roi

In [74]:
```
director_movie_merge.drop_duplicates()
director_movie_merge.head()
```

Out[74]:

	primary_name	movie	id	roi_in_mils
0	David Bowers	Diary of a Wimpy Kid: Rodrick Rules	80	55.695194
1	David Bowers	Diary of a Wimpy Kid: Rodrick Rules	80	55.695194
2	David Bowers	Diary of a Wimpy Kid: Rodrick Rules	80	55.695194
3	David Bowers	Diary of a Wimpy Kid: The Long Haul	27	13.609577
4	David Bowers	Diary of a Wimpy Kid: The Long Haul	27	13.609577

In [75]:
```
writer_roi= writer_movie_merge.groupby(writer_movie_merge['primary_name']).ag
writer_roi.head()
```

Out[75]:

	roi_in_mils
primary_name	
A. Jaye Williams	16.393939
A. Scott Berg	-9.734717
A. Sreedhar	17.226218
A.A. Milne	54.265324
A.C. Mughil	13.257000

Sorted to get top 5 most lucrative writers and directors

In [76]:
```
director_roi= director_movie_merge.groupby(director_movie_merge['primary_name
director_roi.head()
```

Out[76]:

	roi_in_mils
primary_name	
Aaron Agrasanchez	22.897191
Aaron Alon	-0.718176
Aaron Hann	-1.989976
Aaron Schnobrich	0.350641
Aaron Seltzer	61.424988

In [77]:
```
writer_roi_top=writer_roi.sort_values(['roi_in_mils'],ascending='false')
writer_roi_top.tail(5)
```

Out[77]:

	roi_in_mils
primary_name	
Keith Giffen	1365.711972
Steve Gan	1365.711972
Ravi Punj	2008.208395
Kevin Lincoln	2008.208395
Teruo Noguchi	2351.345279

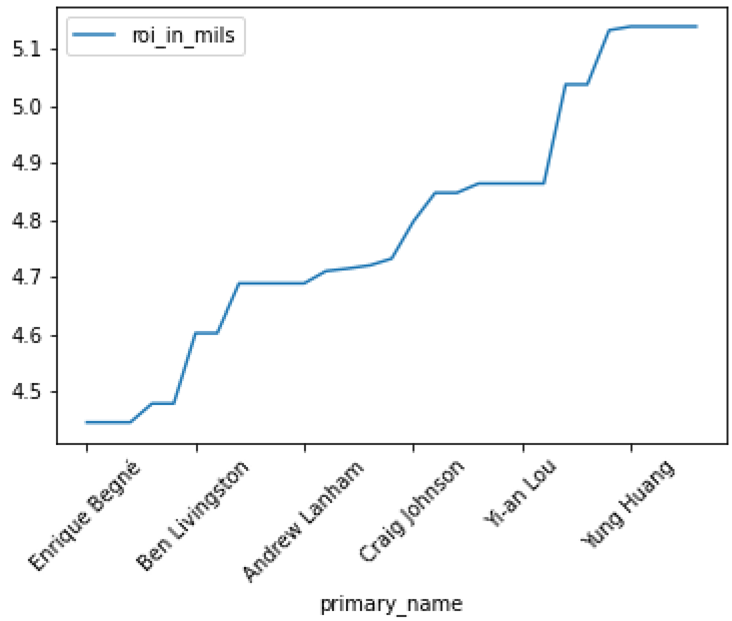
In [78]:
```
director_roi_top=director_roi.sort_values(['roi_in_mils'],ascending='False')
director_roi_top.tail(5)
```

Out[78]:

	roi_in_mils
primary_name	
Anthony Russo	1205.153604
Joe Russo	1205.153604
Ravi Punj	2008.208395
Kevin Lincoln	2008.208395
Atsushi Wada	2351.345279

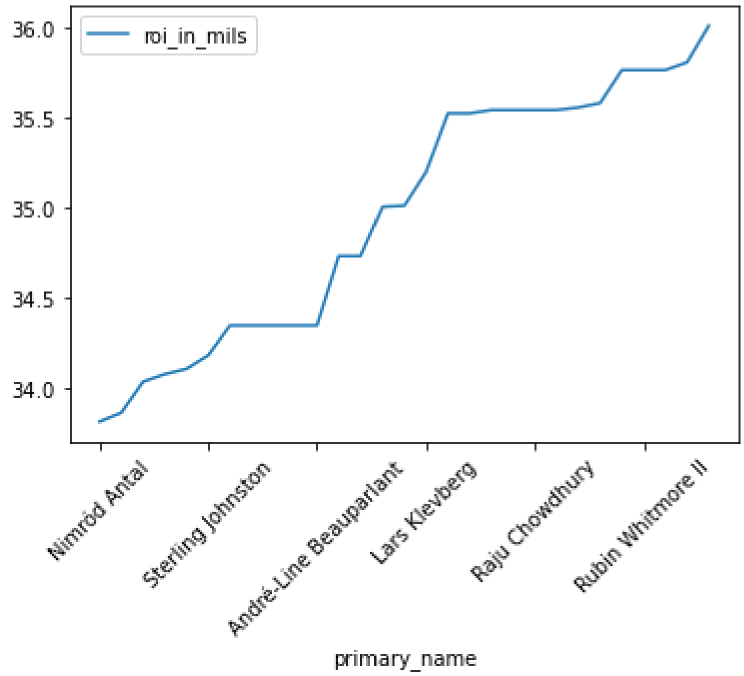
In [79]:
```
writer_roi_top.iloc[2060:2089].plot(y='roi_in_mils'),plt.xticks(rotation = 45
```

Out[79]:
```
(<AxesSubplot:xlabel='primary_name'>,
 (array([-5.,  0.,  5., 10., 15., 20., 25., 30.]),
  [Text(-5.0, 0, 'Xavier Giannoli'),
   Text(0.0, 0, 'Enrique Begné'),
   Text(5.0, 0, 'Ben Livingston'),
   Text(10.0, 0, 'Andrew Lanham'),
   Text(15.0, 0, 'Craig Johnson'),
   Text(20.0, 0, 'Yi-an Lou'),
   Text(25.0, 0, 'Yung Huang'),
   Text(30.0, 0, '')]))
```



In [80]:
```
director_roi_top.iloc[2060:2089].plot(y='roi_in_mils'),plt.xticks(rotation =
```

Out[80]:
```
(<AxesSubplot:xlabel='primary_name'>,
 (array([-5.,  0.,  5., 10., 15., 20., 25., 30.]),
  [Text(-5.0, 0, 'Alex Zalban'),
   Text(0.0, 0, 'Nimród Antal'),
   Text(5.0, 0, 'Sterling Johnston'),
   Text(10.0, 0, 'André-Line Beauparlant'),
   Text(15.0, 0, 'Lars Klevberg'),
   Text(20.0, 0, 'Raju Chowdhury'),
   Text(25.0, 0, 'Rubin Whitmore II'),
   Text(30.0, 0, '')]))
```



Comparing Ratings to Writers and Directors

In [81]:
```
writer_movies_r_merge=pd.merge(pd.merge(writer_names,writers_copy,on='person_
writer_movies_r_merge.head()
```

Out[81]:

	person_id	primary_name	primary_professions	movie_id	average_rating	numvotes
0	nm0065847	Michael Frost Beckner	writer	tt6349302	5.6	1809
1	nm0508052	Crash Leyland	writer	tt6349302	5.6	1809
2	nm0369675	Chris Hauty	writer	tt6349302	5.6	1809
3	nm0068874	Hava Kohav Beller	writer	tt7701650	5.4	11
4	nm0072476	Doug Benson	writer	tt1975283	6.3	474

In [82]:
```
director_movies_r_merge=pd.merge(pd.merge(director_names,directors_copy,on='p
director_movies_r_merge.head()
```

Out[82]:

	person_id	primary_name	primary_professions	movie_id	average_rating	numvotes
0	nm0062879	Ruel S. Bayani	director	tt1592569	6.4	77
1	nm0062879	Ruel S. Bayani	director	tt1592569	6.4	77
2	nm0062879	Ruel S. Bayani	director	tt1592569	6.4	77
3	nm0062879	Ruel S. Bayani	director	tt1592569	6.4	77
4	nm0062879	Ruel S. Bayani	director	tt8421806	7.9	54

In [83]:
```
clist=['primary_name','average_rating']
director_rating=director_movies_r_merge[clist]
director_rating.head()
```

Out[83]:

	primary_name	average_rating
0	Ruel S. Bayani	6.4
1	Ruel S. Bayani	6.4
2	Ruel S. Bayani	6.4
3	Ruel S. Bayani	6.4
4	Ruel S. Bayani	7.9

In [84]:
```
clist=['primary_name','average_rating']
writer_rating=writer_movies_r_merge[clist]
writer_rating.head()
```

Out[84]:

	primary_name	average_rating
0	Michael Frost Beckner	5.6
1	Crash Leyland	5.6
2	Chris Hauty	5.6
3	Hava Kohav Beller	5.4
4	Doug Benson	6.3

In [85]:
```
writer_ratings= writer_rating.groupby(writer_rating['primary_name']).aggregat
writer_ratings.head()
```

Out[85]:

primary_name	average_rating
'A.J.' Marriot	7.3
'Om' Rakesh Chaturvedi	5.6
A Normale Jef	7.2
A Shawn Austin	8.8
A Type Machine	4.5

In [86]:
```
director_ratings= director_rating.groupby(director_rating['primary_name']).ag
director_ratings.head()
```

Out[86]:

primary_name	average_rating
A Normale Jef	7.2
A. Blaine Miller	7.0
A. Cengiz Mert	3.2
A. Fishman	7.8
A. Haluk Unal	8.8

In [87]:

```
writer_ratings_top=writer_ratings.sort_values(['average_rating'],ascending='F
writer_ratings_top.tail(10)
```

Out[87]:

primary_name	average_rating
Javi Larrauri	9.8
Rok Andres	9.8
Fujisaki Ryuta	9.8
Dante Tanikie-Montagnani	9.8
Cristina Duarte	10.0
Heather Augustyn	10.0
Emre Oran	10.0
Ivana Diniz	10.0
Brian Baucum	10.0
Daniel Alexander	10.0

In [88]:

```
director_ratings_top=director_ratings.sort_values(['average_rating'],ascendir
director_ratings_top.tail(10)
```

Out[88]:

primary_name	average_rating
Raphael Sbarge	9.9
Amoghavarsha	9.9
Nagaraja Uppunda	9.9
Emre Oran	10.0
Ivana Diniz	10.0
Lindsay Thompson	10.0
Chad Carpenter	10.0
Masahiro Hayakawa	10.0
Michiel Brongers	10.0
Loreto Di Cesare	10.0

Analysis

As far as the most money made by thier movies, the top 3 earning writers are: Ravi Punj, Kevin Lincoln and Teruo Noguchi. The Top 3 earning directors are: Ravi Punj, Kevin Lincoln and Atsushi Wada. The directors with the highest average rated movies are: . Emre Oran, Ivana Diniz, Lindsay

Thompson, Chad Carpenter, Masahiro Hayakawa, Michiel Brongers and Loreto Di Cesare. The the writers with the highest average rating are: Heather Augustyn, Emre Oran, Ivana Diniz, Brian Baucum and Daniel Alexande. All averaging a 10 rating.

Conclusions

My analysis leads to three recommendations for Microsoft to be successful in the movie industy:

- Focus on creating movies in the fantasy, science fiction, family, comedy, history and adventure genres. They prove to be the most popular income earning.
- Have your movies be released in the months of May, June and December. Movies released during these months made the most money and had the highest ratings. May be due to the fact that they coincide with times when people are out and about due to holidays or vacations.
- Consider having Ravi Punj, Kevin Lincoln on your staff. They were the top earning writers and directors.

In [89]:

```
conn.close()
```