Project 1 Submission

* Student name: Anni Liu
* Instructor name: Hardik Idnani

Microsoft Movie Analysis

Overview

Microsoft sees all the big companies creating original video content and want to get in on the fun. They have decided to create a new movie studio, but have limited knowledge about creating movies. We use exploratory data analysis which will provide them with the types of films that are performing well at the box office. We then translate these findings into actionable insights that the head of Microsoft's new movie studio can use to help decide what types of films to create.

Import libraries

Import all libriaries to enable data import and visualisations.

In [1]:
```
# Import libraries for analysis

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

We answer the following:

- "How did you pick the question(s) that you did?"
- "Why are these questions important from a business perspective?"
- "How did you decide on the data cleaning options you performed?"
- "Why did you choose a given method or library?"
- "Why did you select those visualizations and what did you learn from each of them?"
- "Why did you pick those features as predictors?"
- "How would you interpret the results?"
- "How confident are you in the predictive quality of the results?"
- "What are some of the things that could cause the results to be wrong?"

In [2]:
```
%matplotlib inline
```

In [3]:
```
title_basics_data = pd.read_csv("C:/Users/AnnieLiu/Desktop/Misc/DA - Materials/Proje
```

Understanding the data from "title.basics.csv"

In [4]:
```
# Chose data sets with the most relevant variables/columns
# Review what dataframe looks like
```

In [5]:
```
title_basics_data.head()
```

Out[5]:

	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 [6]:
```
title_basics_data.tail()
```

Out[6]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres
146139	tt9916538	Kuambil Lagi Hatiku	Kuambil Lagi Hatiku	2019	123.0	Drama
146140	tt9916622	Rodolpho Teóphilo - O Legado de um Pioneiro	Rodolpho Teóphilo - O Legado de um Pioneiro	2015	NaN	Documentary
146141	tt9916706	Dankyavar Danka	Dankyavar Danka	2013	NaN	Comedy
146142	tt9916730	6 Gunn	6 Gunn	2017	116.0	NaN
146143	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	NaN	Documentary

In [7]:
```
title_basics_data.shape
```

Out[7]:
```
(146144, 6)
```

In [8]:
```
title_basics_data.columns
```

Out[8]:
```
Index(['tconst', 'primary_title', 'original_title', 'start_year',
       'runtime_minutes', 'genres'],
      dtype='object')
```

Understanding data from "title.ratings.csv"

In [9]:
```
# Review what dataframe looks like
```

In [10]:
```
title_ratings_data = pd.read_csv("C:/Users/AnnieLiu/Desktop/Misc/DA - Materials/Proj
```

In [11]:
```
title_ratings_data.head()
```

Out[11]:

	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 [12]:
```
title_ratings_data.tail()
```

Out[12]:

	tconst	averagerating	numvotes
73851	tt9805820	8.1	25
73852	tt9844256	7.5	24
73853	tt9851050	4.7	14
73854	tt9886934	7.0	5
73855	tt9894098	6.3	128

In [13]:
```
title_ratings_data.shape
```

Out[13]:
```
(73856, 3)
```

In [14]:
```
title_ratings_data.columns
```

Out[14]:
```
Index(['tconst', 'averagerating', 'numvotes'], dtype='object')
```

Understanding the data from "tn.movie_budgets.csv"

In [15]:
```
tn_movie_budgets_data = pd.read_csv("C:/Users/AnnieLiu/Desktop/Misc/DA - Materials/P
```

In [16]:
```
# Review what dataframe looks like
```

In [17]:
```
tn_movie_budgets_data.head()
```

Out[17]:

	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 [18]:
```
tn_movie_budgets_data.tail()
```

Out[18]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
5777	78	Dec 31, 2018	Red 11	$7,000	$0	$0
5778	79	Apr 2, 1999	Following	$6,000	$48,482	$240,495
5779	80	Jul 13, 2005	Return to the Land of Wonders	$5,000	$1,338	$1,338
5780	81	Sep 29, 2015	A Plague So Pleasant	$1,400	$0	$0
5781	82	Aug 5, 2005	My Date With Drew	$1,100	$181,041	$181,041

In [19]:
```
tn_movie_budgets_data.shape
```

Out[19]:
```
(5782, 6)
```

In [20]:
```
tn_movie_budgets_data.columns
```

Out[20]:
```
Index(['id', 'release_date', 'movie', 'production_budget', 'domestic_gross',
       'worldwide_gross'],
      dtype='object')
```

Combining title_basics & title_ratings data

In [21]:
```
# Understand columns
```

In [22]:
```
title_basics_data.columns
```

Out[22]:
```
Index(['tconst', 'primary_title', 'original_title', 'start_year',
       'runtime_minutes', 'genres'],
      dtype='object')
```

In [23]:
```
title_ratings_data.columns
```

Out[23]:
```
Index(['tconst', 'averagerating', 'numvotes'], dtype='object')
```

In [24]:
```
# Merging two data sets
title_basics_ratings = pd.merge(title_basics_data, title_ratings_data)
title_basics_ratings.columns
```

Out[24]:
```
Index(['tconst', 'primary_title', 'original_title', 'start_year',
       'runtime_minutes', 'genres', 'averagerating', 'numvotes'],
      dtype='object')
```

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

Out[25]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres	averagerati
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 [26]:
```
title_basics_ratings.shape
```

Out[26]:
```
(73856, 8)
```

Combine title_basics_ratings & tn_movies_budgets data

In [27]:
```
# The financials are availabile in tn_movies_budgets data so here we merge this with
```

In [28]:
```
title_basics_ratings.columns
```

Out[28]:
```
Index(['tconst', 'primary_title', 'original_title', 'start_year',
       'runtime_minutes', 'genres', 'averagerating', 'numvotes'],
      dtype='object')
```

In [29]:
```
tn_movie_budgets_data.columns
```

Out[29]:
```
Index(['id', 'release_date', 'movie', 'production_budget', 'domestic_gross',
       'worldwide_gross'],
      dtype='object')
```

In [30]:
```
# All 3 data sets are now in one dataframe
basics_ratings_budgets = pd.merge(title_basics_ratings, tn_movie_budgets_data, left_
basics_ratings_budgets
```

Out[30]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres	aver
0	tt0249516	Foodfight!	Foodfight!	2012	91.0	Action,Animation,Comedy	
1	tt0326592	The Overnight	The Overnight	2010	88.0	NaN	
2	tt3844362	The Overnight	The Overnight	2015	79.0	Comedy,Mystery	
3	tt0337692	On the Road	On the Road	2012	124.0	Adventure,Drama,Romance	
4	tt4339118	On the Road	On the Road	2014	89.0	Drama	
...	...	...	...	...	...	...	
2870	tt8680254	Richard III	Richard III	2016	NaN	Drama	
2871	tt8824064	Heroes	Heroes	2019	88.0	Documentary	
2872	tt8976772	Push	Push	2019	92.0	Documentary	
2873	tt9024106	Unplanned	Unplanned	2019	106.0	Biography,Drama	
2874	tt9248762	The Terrorist	The Terrorist	2018	NaN	Thriller	

2875 rows × 14 columns

Data cleaning

In [31]:
```
# Dropping irrelevant columns and columns which represent the same variables. This w
```

In [32]:
```
df_dropped = basics_ratings_budgets.drop(['tconst', 'primary_title', 'runtime_minute
df_dropped
```

Out[32]:

	genres	averagerating	numvotes	release_date	movie	production_budget
0	Action,Animation,Comedy	1.9	8248	Dec 31, 2012	Foodfight!	$45,000,000
1	NaN	7.5	24	Jun 19, 2015	The Overnight	$200,000
2	Comedy,Mystery	6.1	14828	Jun 19, 2015	The Overnight	$200,000
3	Adventure,Drama,Romance	6.1	37886	Mar 22, 2013	On the Road	$25,000,000
4	Drama	6.0	6	Mar 22, 2013	On the Road	$25,000,000
...	...	...	...	...	...	...
2870	Drama	9.1	28	Dec 29, 1995	Richard III	$9,200,000
2871	Documentary	7.3	7	Oct 24, 2008	Heroes	$400,000
2872	Documentary	7.3	33	Feb 6, 2009	Push	$38,000,000
2873	Biography,Drama	6.3	5945	Mar 29, 2019	Unplanned	$6,000,000
2874	Thriller	6.0	6	Jan 14, 2000	The Terrorist	$25,000

2875 rows × 8 columns

In [33]:
```
# Check for any duplicates
```

In [34]:
```
df_dropped.duplicated().sum()
```

Out[34]: 0

In [35]:
```
# Check for duplicates in 'Movie' as we only need one unique movie name
df_dropped['movie'].duplicated().sum()
```

Out[35]: 749

In [36]:
```
# Drop 'movie' duplicates. Create dataframe with attributes against each unique movi
df_movie_duplicate_dropped = df_dropped.drop_duplicates(subset='movie')
df_movie_duplicate_dropped
```

Out[36]:

	genres	averagerating	numvotes	release_date	movie	production_budget
0	Action,Animation,Comedy	1.9	8248	Dec 31, 2012	Foodfight!	$45,000,000
1	NaN	7.5	24	Jun 19, 2015	The Overnight	$200,000
3	Adventure,Drama,Romance	6.1	37886	Mar 22, 2013	On the Road	$25,000,000
6	Adventure,Comedy,Drama	7.3	275300	Dec 25, 2013	The Secret Life of Walter Mitty	$91,000,000
7	Action,Crime,Drama	6.5	105116	Sep 19, 2014	A Walk Among the Tombstones	$28,000,000
...	...	...	...	...	...	...
2870	Drama	9.1	28	Dec 29, 1995	Richard III	$9,200,000
2871	Documentary	7.3	7	Oct 24, 2008	Heroes	$400,000
2872	Documentary	7.3	33	Feb 6, 2009	Push	$38,000,000
2873	Biography,Drama	6.3	5945	Mar 29, 2019	Unplanned	$6,000,000
2874	Thriller	6.0	6	Jan 14, 2000	The Terrorist	$25,000

2126 rows × 8 columns

In [37]:
```
df_movie_duplicate_dropped['movie'].duplicated().sum()
```

Out[37]:
```
0
```

In [38]:
```
# Check for any missing values
df_movie_duplicate_dropped.isnull().sum()
```

Out[38]:
```
genres              2
averagerating       0
numvotes            0
release_date        0
movie               0
production_budget   0
domestic_gross      0
worldwide_gross     0
dtype: int64
```

In [39]:
```
# Only 'genres' has empty data
# Let's see what it look like close up
df_movie_duplicate_dropped['genres'].value_counts()
```

Out[39]:
```
Drama                          159
Comedy                          81
Comedy,Drama                    73
Adventure,Animation,Comedy      68
Comedy,Drama,Romance            66
                              ...
Drama,Family,Fantasy             1
Action,Comedy,Sport              1
Drama,Family,History             1
Documentary,History              1
Crime,Fantasy,Thriller           1
Name: genres, Length: 284, dtype: int64
```

How did you pick the question(s) that you did?

Genres appears like it can be a good starting point to draw genre popularity by count.

Why is this question important from a business perspective?

By asking this question we find out the most commonly (or least) produced movie genres. This provides Microsoft with an indication of ask further questions as to why this might be the case and if pursuing the most popular genre would be most beneficial to them. For e.g. Most popular genre by count may not necessarily return greatest profit.

Genre by count

In [40]:
```
# Fill missing data with the most frequent value from 'genres'

df_movie_duplicate_dropped['genres'].fillna(df_movie_duplicate_dropped['genres'].val
```

```
C:\Users\AnnieLiu\anaconda3\lib\site-packages\pandas\core\generic.py:6392: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  return self._update_inplace(result)
```

In [41]:
```
# Check there are no missing values in 'genres'
df_movie_duplicate_dropped['genres'].isna().sum()
```

Out[41]:
```
0
```

In [42]:
```
# Check columns types to assess if any types need to be converted for analysis
df_movie_duplicate_dropped.dtypes
```

Out[42]:
```
genres                object
averagerating        float64
numvotes               int64
release_date          object
movie                 object
production_budget     object
domestic_gross        object
worldwide_gross       object
dtype: object
```

In [43]:
```
# Convert worldwide_gross and production_budget being $$ Objects to integers
```

In [44]:
```
df_movie_duplicate_dropped['production_budget'] = df_movie_duplicate_dropped['produc
df_movie_duplicate_dropped['production_budget'] = pd.to_numeric(df_movie_duplicate_d
df_movie_duplicate_dropped['production_budget']
```

```
C:\Users\AnnieLiu\AppData\Local\Temp/ipykernel_31252/494982929.py:1: FutureWarnin
g: The default value of regex will change from True to False in a future version.
In addition, single character regular expressions will *not* be treated as literal
strings when regex=True.
  df_movie_duplicate_dropped['production_budget'] = df_movie_duplicate_dropped['pr
oduction_budget'].str.replace('$','').str.replace(',', '')
C:\Users\AnnieLiu\AppData\Local\Temp/ipykernel_31252/494982929.py:1: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_movie_duplicate_dropped['production_budget'] = df_movie_duplicate_dropped['pr
oduction_budget'].str.replace('$','').str.replace(',', '')
C:\Users\AnnieLiu\AppData\Local\Temp/ipykernel_31252/494982929.py:2: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_movie_duplicate_dropped['production_budget'] = pd.to_numeric(df_movie_duplica
te_dropped['production_budget'])
```

Out[44]:
```
0       45000000
1         200000
3       25000000
6       91000000
7       28000000
           ...
2870     9200000
2871      400000
2872    38000000
2873     6000000
2874       25000
Name: production_budget, Length: 2126, dtype: int64
```

In [45]:
```
df_movie_duplicate_dropped['worldwide_gross'] = df_movie_duplicate_dropped['worldwic
df_movie_duplicate_dropped['worldwide_gross'] = pd.to_numeric(df_movie_duplicate_drc
df_movie_duplicate_dropped['worldwide_gross']
```

```
C:\Users\AnnieLiu\AppData\Local\Temp/ipykernel_31252/1566134900.py:1: FutureWarnin
g: The default value of regex will change from True to False in a future version.
In addition, single character regular expressions will *not* be treated as literal
strings when regex=True.
  df_movie_duplicate_dropped['worldwide_gross'] = df_movie_duplicate_dropped['worl
dwide_gross'].str.replace('$','').str.replace(',', '')
C:\Users\AnnieLiu\AppData\Local\Temp/ipykernel_31252/1566134900.py:1: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_movie_duplicate_dropped['worldwide_gross'] = df_movie_duplicate_dropped['worl
dwide_gross'].str.replace('$','').str.replace(',', '')
C:\Users\AnnieLiu\AppData\Local\Temp/ipykernel_31252/1566134900.py:2: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_movie_duplicate_dropped['worldwide_gross'] = pd.to_numeric(df_movie_duplicate
_dropped['worldwide_gross'])
```

Out[45]:
```
0           73706
1         1165996
3         9313302
6       187861183
7        62108587
           ...
2870      4199334
2871       655538
2872     49678401
2873     18107621
2874       195043
Name: worldwide_gross, Length: 2126, dtype: int64
```

How did you decide on the data cleaning options you performed?

Data looked overwhelmingly large, with duplicates, we firstly drop the unnecessary columns, checked for duplicates such as movie title, removed movie title duplicate as we only need one for our analysis. Then checked for missing data and replaced most frequent value from genre. Focused only on the attributes we needed for each question, some objects with a dollar value had been converted from ojbect to integer for ease of analysis.

In [46]:
```
# Find unique genres so that they are not mixed
```

In [47]:
```
df_movie_duplicate_dropped['genres']
```

Out[47]:
```
0       Action,Animation,Comedy
1                         Drama
3       Adventure,Drama,Romance
6        Adventure,Comedy,Drama
7            Action,Crime,Drama
                 ...
2870                      Drama
2871                Documentary
2872                Documentary
2873             Biography,Drama
2874                    Thriller
Name: genres, Length: 2126, dtype: object
```

In [48]:
```
# Create a list and split the genres in each row
list1 =[]
for value in df_movie_duplicate_dropped['genres']:
    list1.append(value.split(','))
```

In [49]:
```
list1
```

```
['Crime', 'Drama', 'Horror'],
['Action', 'Adventure', 'Drama'],
['Comedy', 'Crime', 'Drama'],
['Crime', 'Drama', 'Horror'],
['Action', 'Adventure', 'Family'],
['Biography', 'Drama', 'Sport'],
['Crime', 'Drama', 'History'],
['Adventure', 'Drama', 'Family'],
['Comedy', 'Romance'],
['Action', 'Adventure', 'Fantasy'],
['Biography', 'Drama', 'Thriller'],
['Action', 'Adventure', 'Fantasy'],
['Comedy'],
['Adventure', 'Family', 'Fantasy'],
['Action', 'Adventure', 'Animation'],
['Action', 'Crime', 'Thriller'],
['Drama', 'Romance', 'War'],
['Horror', 'Thriller'],
['Action', 'Comedy', 'Crime'],
```

In [50]:
```
# # Turn multi dimensional (nested) list into a single list that contains all the el
list2= []
for item in list1:
    for item1 in item:
       list2.append(item1)
```

In [51]:
```
list2
```

Out[51]:
```
['Action',
 'Animation',
 'Comedy',
 'Drama',
 'Adventure',
 'Drama',
 'Romance',
 'Adventure',
 'Comedy',
 'Drama',
 'Action',
 'Crime',
 'Drama',
 'Action',
 'Adventure',
 'Sci-Fi',
 'Comedy',
 'Drama',
 'Comedy',
```

In [52]:
```
# Only show unique elements
un_list = []
for item in list2:
    if item not in un_list:
        un_list.append(item)
```

In [53]:
```
un_list
```

Out[53]:
```
['Action',
 'Animation',
 'Comedy',
 'Drama',
 'Adventure',
 'Romance',
 'Crime',
 'Sci-Fi',
 'Family',
 'Thriller',
 'Horror',
 'Mystery',
 'Biography',
 'History',
 'War',
 'Fantasy',
 'Sport',
 'Music',
 'Documentary',
 'Western',
 'Musical',
 'News']
```

In [54]:
```
# Identify the count of each of the elements in the container
from collections import Counter
```

In [55]:
```
Counter(list2)
```

Out[55]:
```
Counter({'Action': 522,
         'Animation': 118,
         'Comedy': 650,
         'Drama': 1076,
         'Adventure': 395,
         'Romance': 256,
         'Crime': 301,
         'Sci-Fi': 170,
         'Family': 115,
         'Thriller': 365,
         'Horror': 266,
         'Mystery': 166,
         'Biography': 165,
         'History': 55,
         'War': 27,
         'Fantasy': 145,
         'Sport': 46,
         'Music': 60,
         'Documentary': 118,
         'Western': 14,
         'Musical': 12,
         'News': 1})
```

In [56]:

```
# Transfer counter data into a dataframe
from collections import Counter
d = Counter({'Action': 522,'Animation': 118,'Comedy': 650,'Drama': 1076,'Adventure':
genres_produced = pd.DataFrame.from_dict(d, orient='index').reset_index()
genres_produced
```

Out[56]:

	index	0
0	Action	522
1	Animation	118
2	Comedy	650
3	Drama	1076
4	Adventure	395
5	Romance	256
6	Crime	301
7	Sci-Fi	170
8	Family	115
9	Thriller	365
10	Horror	266
11	Mystery	166
12	Biography	165
13	History	55
14	War	27
15	Fantasy	145
16	Sport	46
17	Music	60
18	Documentary	118
19	Western	14
20	Musical	12
21	News	1

In [57]:
```
# Replace index name with Genre
genres_produced = genres_produced.rename(columns={'index':'Genre', 0:'Count'})
genres_produced
```

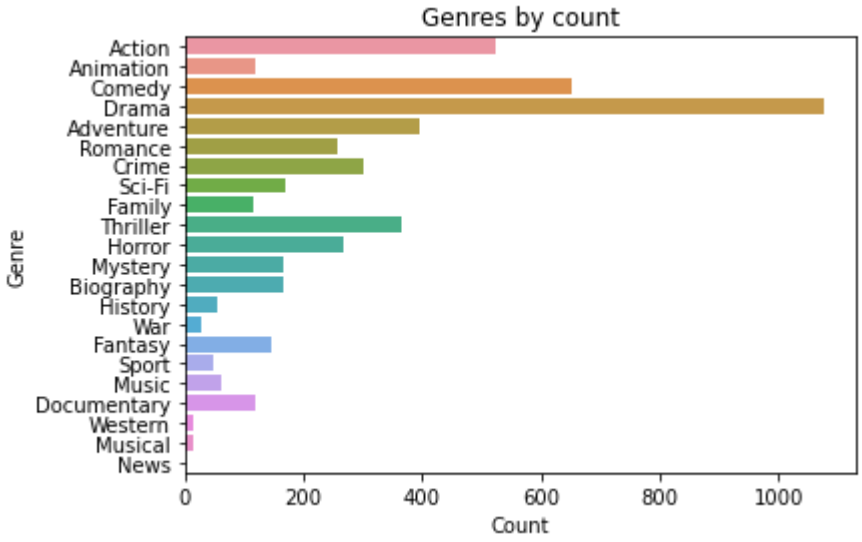
Out[57]:

	Genre	Count
0	Action	522
1	Animation	118
2	Comedy	650
3	Drama	1076
4	Adventure	395
5	Romance	256
6	Crime	301
7	Sci-Fi	170
8	Family	115
9	Thriller	365
10	Horror	266
11	Mystery	166
12	Biography	165
13	History	55
14	War	27
15	Fantasy	145
16	Sport	46
17	Music	60
18	Documentary	118
19	Western	14
20	Musical	12
21	News	1

Plot Genres by Count for visualisation

In [58]:
```
# Bar plotting Genre by Count
sns.barplot(x=genres_produced['Count'],y=genres_produced['Genre'],data=genres_produc
plt.title("Genres by count")
```

Out[58]: Text(0.5, 1.0, 'Genres by count')



Why did you select those visualizations and what did you learn from each of them?

Bar chart chosen to display relationship between numeric and categoric variable. Produced easy to read results.

How would you interpret the results?

Result indicates drama films were most popular by counts of movies produced, followed by comedy and action.

Profitability

How did you pick the question(s) that you did?

There are readily availabile financial data and movie release dates. We use revenue gross and production budget to assess profitability throughout the year to see if there are any trends.

Why is this question important from a business perspective?

Understanding the bottom line commercially, Microsoft will be in a better position to align with strategic goals. Higher budget may not necessarily generate greater profit for ROI. We look for trends such as the time of year to assess if there may be particular periods which generate greater profit. Understanding this will allow Microsoft to appropriate forecast their release dates.

In [59]:
```
# To work out profitability we deduct budget from gross revenue
# Create a new column for profit
for ind, row in df_movie_duplicate_dropped.iterrows():
    df_movie_duplicate_dropped.loc[ind, "Profit"] = row['worldwide_gross'] - row['pr
```

```
C:\Users\AnnieLiu\anaconda3\lib\site-packages\pandas\core\indexing.py:1684: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  self.obj[key] = infer_fill_value(value)
C:\Users\AnnieLiu\anaconda3\lib\site-packages\pandas\core\indexing.py:1817: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  self._setitem_single_column(loc, value, pi)
```

In [60]:
```
df_movie_duplicate_dropped.head()
```

Out[60]:

	genres	averagerating	numvotes	release_date	movie	production_budget	do
0	Action,Animation,Comedy	1.9	8248	Dec 31, 2012	Foodfight!	45000000	
1	Drama	7.5	24	Jun 19, 2015	The Overnight	200000	
3	Adventure,Drama,Romance	6.1	37886	Mar 22, 2013	On the Road	25000000	
6	Adventure,Comedy,Drama	7.3	275300	Dec 25, 2013	The Secret Life of Walter Mitty	91000000	
7	Action,Crime,Drama	6.5	105116	Sep 19, 2014	A Walk Among the Tombstones	28000000	

In [61]:
```
df_movie_duplicate_dropped.shape
```

Out[61]:
```
(2126, 9)
```

In [62]:
```
# Narrowing down to the columns we need analysis
df_movie_duplicate_dropped['Profit'] = df_movie_duplicate_dropped['worldwide_gross']
df_movie_duplicate_dropped[['release_date', 'worldwide_gross', 'production_budget',
```

```
C:\Users\AnnieLiu\AppData\Local\Temp/ipykernel_31252/3639130179.py:2: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_movie_duplicate_dropped['Profit'] = df_movie_duplicate_dropped['worldwide_gro
ss'] - df_movie_duplicate_dropped['production_budget']
```

Out[62]:

	release_date	worldwide_gross	production_budget	Profit
0	Dec 31, 2012	73706	45000000	-44926294
1	Jun 19, 2015	1165996	200000	965996
3	Mar 22, 2013	9313302	25000000	-15686698
6	Dec 25, 2013	187861183	91000000	96861183
7	Sep 19, 2014	62108587	28000000	34108587
...	...	...	...	...
2870	Dec 29, 1995	4199334	9200000	-5000666
2871	Oct 24, 2008	655538	400000	255538
2872	Feb 6, 2009	49678401	38000000	11678401
2873	Mar 29, 2019	18107621	6000000	12107621
2874	Jan 14, 2000	195043	25000	170043

2126 rows × 4 columns

In [63]:
```
# Create new 'release_month' column, determining the month from 'release_date'
df_movie_duplicate_dropped['release_month'] = pd.DatetimeIndex(df_movie_duplicate_dr
```

```
C:\Users\AnnieLiu\AppData\Local\Temp/ipykernel_31252/1144128276.py:2: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.o
rg/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_movie_duplicate_dropped['release_month'] = pd.DatetimeIndex(df_movie_duplicat
e_dropped['release_date']).month
```

In [64]:
```
# New dataframe with release month added
df_movie_duplicate_dropped[df_movie_duplicate_dropped.duplicated()]
```

Out[64]:

	genres	averagerating	numvotes	release_date	movie	production_budget	domestic_gross	worldwid

In [65]:
```
df_movie_duplicate_dropped
```

Out[65]:

	genres	averagerating	numvotes	release_date	movie	production_budget
0	Action,Animation,Comedy	1.9	8248	Dec 31, 2012	Foodfight!	45000000
1	Drama	7.5	24	Jun 19, 2015	The Overnight	200000
3	Adventure,Drama,Romance	6.1	37886	Mar 22, 2013	On the Road	25000000
6	Adventure,Comedy,Drama	7.3	275300	Dec 25, 2013	The Secret Life of Walter Mitty	91000000
7	Action,Crime,Drama	6.5	105116	Sep 19, 2014	A Walk Among the Tombstones	28000000
...	...	...	...	...	...	...
2870	Drama	9.1	28	Dec 29, 1995	Richard III	9200000
2871	Documentary	7.3	7	Oct 24, 2008	Heroes	400000
2872	Documentary	7.3	33	Feb 6, 2009	Push	38000000
2873	Biography,Drama	6.3	5945	Mar 29, 2019	Unplanned	6000000
2874	Thriller	6.0	6	Jan 14, 2000	The Terrorist	25000

2126 rows × 10 columns

In [66]:
```
# Find the earliest release date to determine how far back the data goes
df_movie_duplicate_dropped['release_date'].min()
```

Out[66]:
```
'Apr 1, 2010'
```

In [67]:
```
df_movie_duplicate_dropped.shape
```

Out[67]:
```
(2126, 10)
```

In [68]:
```
# Find the average of the 'release_month' to determine unique months
df_movie_duplicate_dropped.groupby('release_month').mean()
```

Out[68]:

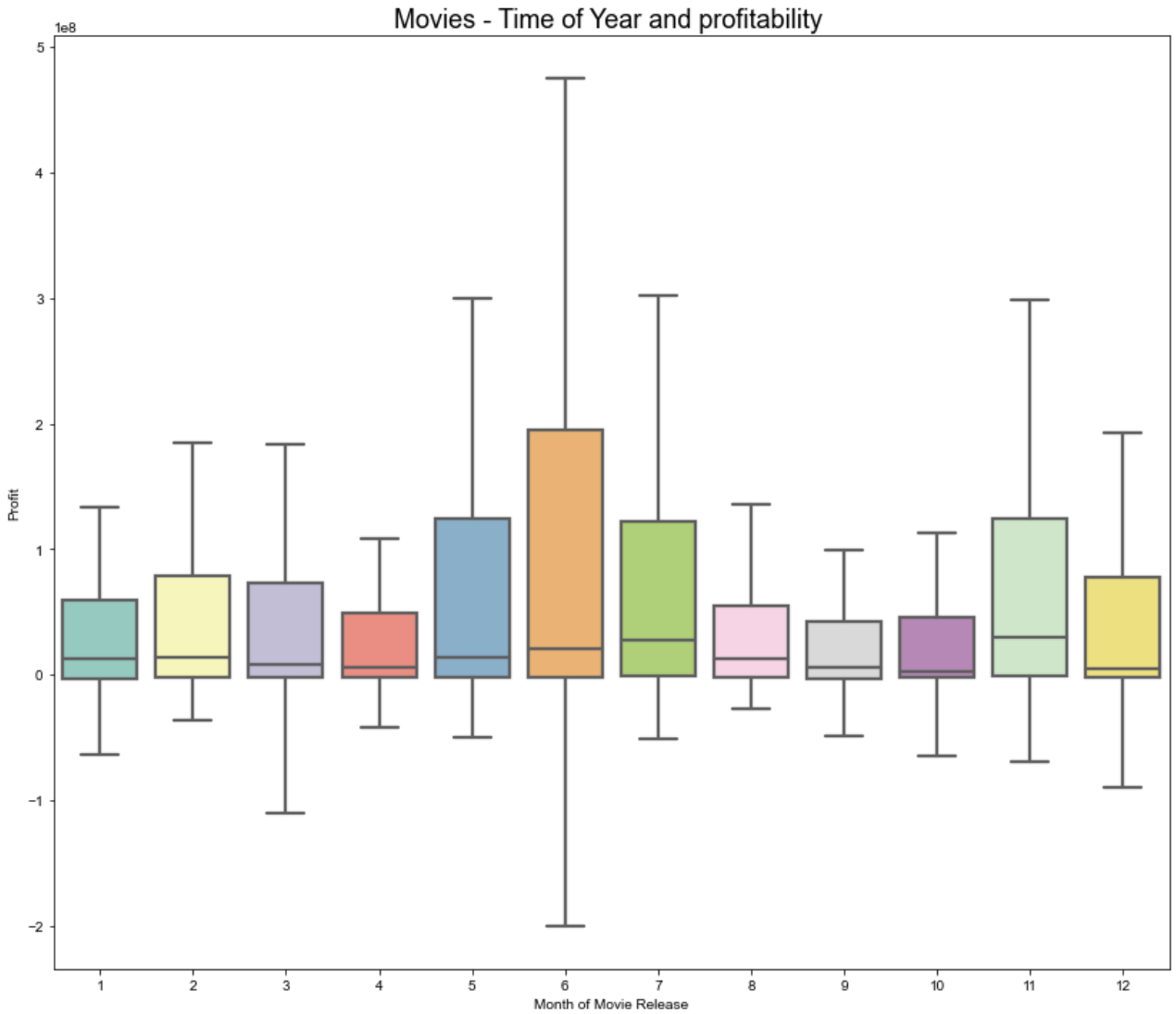
release_month	averagerating	numvotes	production_budget	worldwide_gross	Profit
1	5.982963	47666.992593	2.640623e+07	6.398189e+07	3.757566e+07
2	6.165132	75535.151316	3.471961e+07	9.977082e+07	6.505122e+07
3	6.140110	79390.868132	4.302652e+07	1.172696e+08	7.424311e+07
4	6.260112	58971.269663	3.005708e+07	9.439885e+07	6.434177e+07
5	6.258451	121914.077465	5.971279e+07	1.835001e+08	1.237873e+08
6	6.322892	99163.277108	5.467102e+07	1.910767e+08	1.364057e+08
7	6.268263	105282.724551	4.518946e+07	1.548137e+08	1.096243e+08
8	6.116949	66833.666667	2.834536e+07	7.351027e+07	4.516491e+07
9	6.295628	69529.945355	2.469013e+07	5.994329e+07	3.525316e+07
10	6.256522	78599.908213	2.359413e+07	6.461829e+07	4.102415e+07
11	6.636313	116944.335196	4.777460e+07	1.546475e+08	1.068729e+08
12	6.077907	72193.426357	3.455381e+07	1.179025e+08	8.334867e+07

In [69]:

```
# Create a boxplot using release month and profit

x = df_movie_duplicate_dropped['release_month']
y = df_movie_duplicate_dropped['Profit']
f, ax = plt.subplots(figsize=(14,12))
sns.set_style('darkgrid')
sns.set_context('talk')
sns.boxplot(x, y, palette='Set3', showfliers=False)
plt.title('Movies - Time of Year and profitability ')
plt.ylabel('Profit')
plt.xlabel('Month of Movie Release')
plt.show()
```

```
C:\Users\AnnieLiu\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWar
ning: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Why did you select those visualizations and what did you learn from each of them?

We chose a box plot to compare the distributions because the centre, spread and skewness are immediately apparent. We learn that whilst there are trends in that some months are higher and others lower in profit, there is a positive skew and the median is closely comparable across all months.

How would you interpret the results?

There seems to be quite a lot of distribution/dispersion of data as it captures a very large data set. Therefore we look at the IQR to give us a better guide on profit. We conclude that there is higher profitability during mid-year months and November.

Budget vs Rating

How did you pick the question(s) that you did?

Interestingly the average ratings got us thinking if production budget would have any impact on it?

In [70]:
```
# Create dataframe with relevant columns
df_av_ratings = df_movie_duplicate_dropped[['averagerating', 'production_budget']]
df_av_ratings
```

Out[70]:

	averagerating	production_budget
0	1.9	45000000
1	7.5	200000
3	6.1	25000000
6	7.3	91000000
7	6.5	28000000
...	...	...
2870	9.1	9200000
2871	7.3	400000
2872	7.3	38000000
2873	6.3	6000000
2874	6.0	25000

2126 rows × 2 columns

In [71]:
```
#Sort rating by lowest to highest
df_av_ratings.sort_values(by=['averagerating'])
```

Out[71]:

	averagerating	production_budget
1129	1.6	13000000
0	1.9	45000000
2189	2.1	30000000
1761	2.2	1000000
2144	2.3	3000000
...	...	...
1804	9.0	20000000
1306	9.0	45000000
1921	9.0	11000000
2870	9.1	9200000
684	9.2	3000000

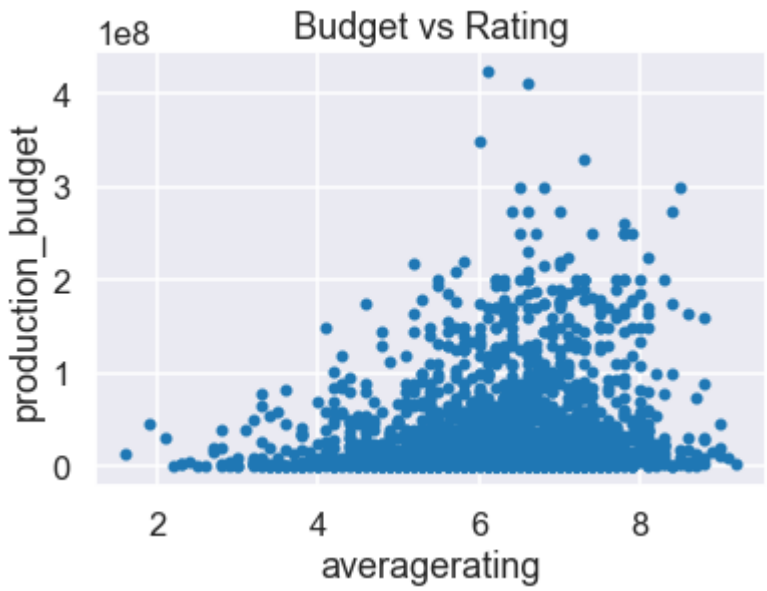
2126 rows × 2 columns

In [72]:
```
# Create scatter plot to determine if any correlation
x = 'averagerating'
y = 'production_budget'

subset_data = df_av_ratings.sample(2126)[[x,y]]
subset_data.sort_values(['averagerating'], ascending=True, inplace=True)
subset_data.dropna(inplace=True)

subset_data.plot(x=x, y=y, kind='scatter', title='Budget vs Rating')
```

Out[72]:
```
<AxesSubplot:title={'center':'Budget vs Rating'}, xlabel='averagerating', ylabel
='production_budget'>
```



In [73]:
```
# As visualisation is too dense, find mean for ratings
df_rating_mean = df_av_ratings.groupby('averagerating').mean()
df_rating_mean
```

Out[73]:

averagerating	production_budget
1.6	1.300000e+07
1.9	4.500000e+07
2.1	3.000000e+07
2.2	1.000000e+06
2.3	3.000000e+06
...	...
8.8	5.316667e+07
8.9	1.500000e+07
9.0	2.533333e+07
9.1	9.200000e+06
9.2	3.000000e+06

74 rows × 1 columns

In [74]:
```
# Create new column for rating with whole integers from 'averagerating'
df_rating_mean['Rating'] = df_rating_mean.index.astype(int)
```

In [75]:
```
df_rating_mean
```

Out[75]:

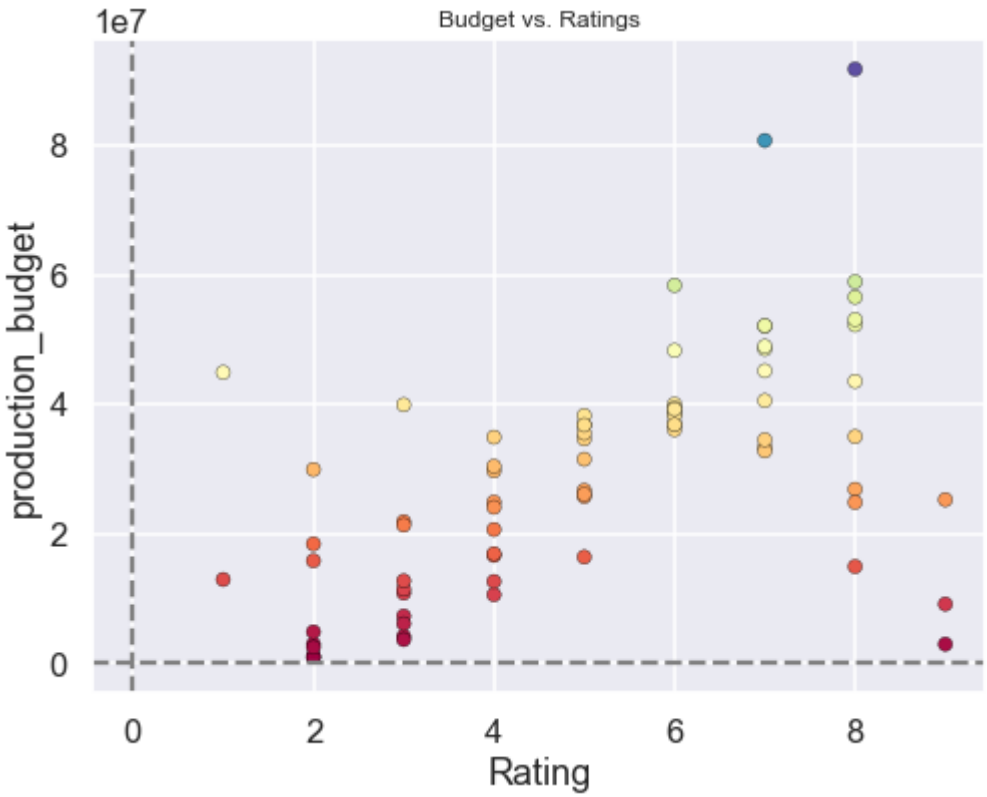
averagerating	production_budget	Rating
1.6	1.300000e+07	1
1.9	4.500000e+07	1
2.1	3.000000e+07	2
2.2	1.000000e+06	2
2.3	3.000000e+06	2
...	...	...
8.8	5.316667e+07	8
8.9	1.500000e+07	8
9.0	2.533333e+07	9
9.1	9.200000e+06	9
9.2	3.000000e+06	9

74 rows × 2 columns

In [76]:
```
# Create new scatter plot with colours and outline to distinguish placement of each
from matplotlib import rcParams
rcParams['figure.figsize'] = 8,6
plt.axvline(0, c=(.5, .5, .5), ls='--')
plt.axhline(0, c=(.5, .5, .5), ls='--')
plt.style.use('seaborn')
plt.scatter(df_rating_mean.Rating, df_rating_mean.production_budget, c=df_rating_mea


plt.xlabel("Rating")
plt.ylabel("production_budget")
plt.title('Budget vs. Ratings')
```

Out[76]: Text(0.5, 1.0, 'Budget vs. Ratings')



Why did you select those visualizations and what did you learn from each of them?

Using a scatter plot (with applied color) allows us to better distinguish the placement of data. Our first scatter plot had greater density, so we reduced the sample size (via descriptive analysis). We learn that whils the correlation isn't perfect, there is certain some positive relationships.

How would you interpret the results?

As we can see from this scatter plot, movies with higher budgets, generally have higher average ratings.

Why did you choose a given method or library?

We started by choosing the dataset with the most amount of valuable data that we can draw insights from. From there we noticed we needed at least 3 datasets and merged these to create a broader picture for analysis. The data was messy with duplicates and missing values. Data cleaned. Applied descriptive statistics and visualizations to emphasize data fluctations. This method was taken to aid in analyzing common themes or correlations so that Microsoft can make more informed business decisions.

Why did you pick those features as predictors?

By way of reducing the number of features and steps required for our first analysis, we adopted features which would present topline indications or relationships in our results.

How confident are you in the predictive quality of the results?

The business problem is very broad at this stage. To determine what movies to produce, we need to determine what Microsoft defines as movies they would want to produce. In this analysis we made broad some initial assumptions to help define our questions and find our answers. Whilst we can generalize that drama seems to be the most popular genre produced, there are many other factors which can contribute to the data. For example, why are drama movies most commonly produced? We are uncertain at this stage, therefore we cannot generalize for example that drama movies would then produce good return for Microsoft. Perhaps drama films might have influence on number of votes and therefore creates more hype or publicity with more viewers engaging with these types of movies. These are hyperthetical assumptions, we would require further analysis.

What are some of the things that could cause the results to be wrong?

There could be other variables that Microsoft would be interested in understanding to better establish the types of movies to produce. For instance, analysing from a local perspective, whilst these results are worldwide data, local data might different results.
Microsoft might also want to determine the types of studios and Directors/team that they would want on their production. These are all factors that can influence our current data. Microsoft should take into account that this is was an international analysis (as Microsoft is an international company). Having a bigger picture of the landscape, we can drill down further. It would also be interesting to assess if there is any correlation between genre and num votes or what drives number of votes?