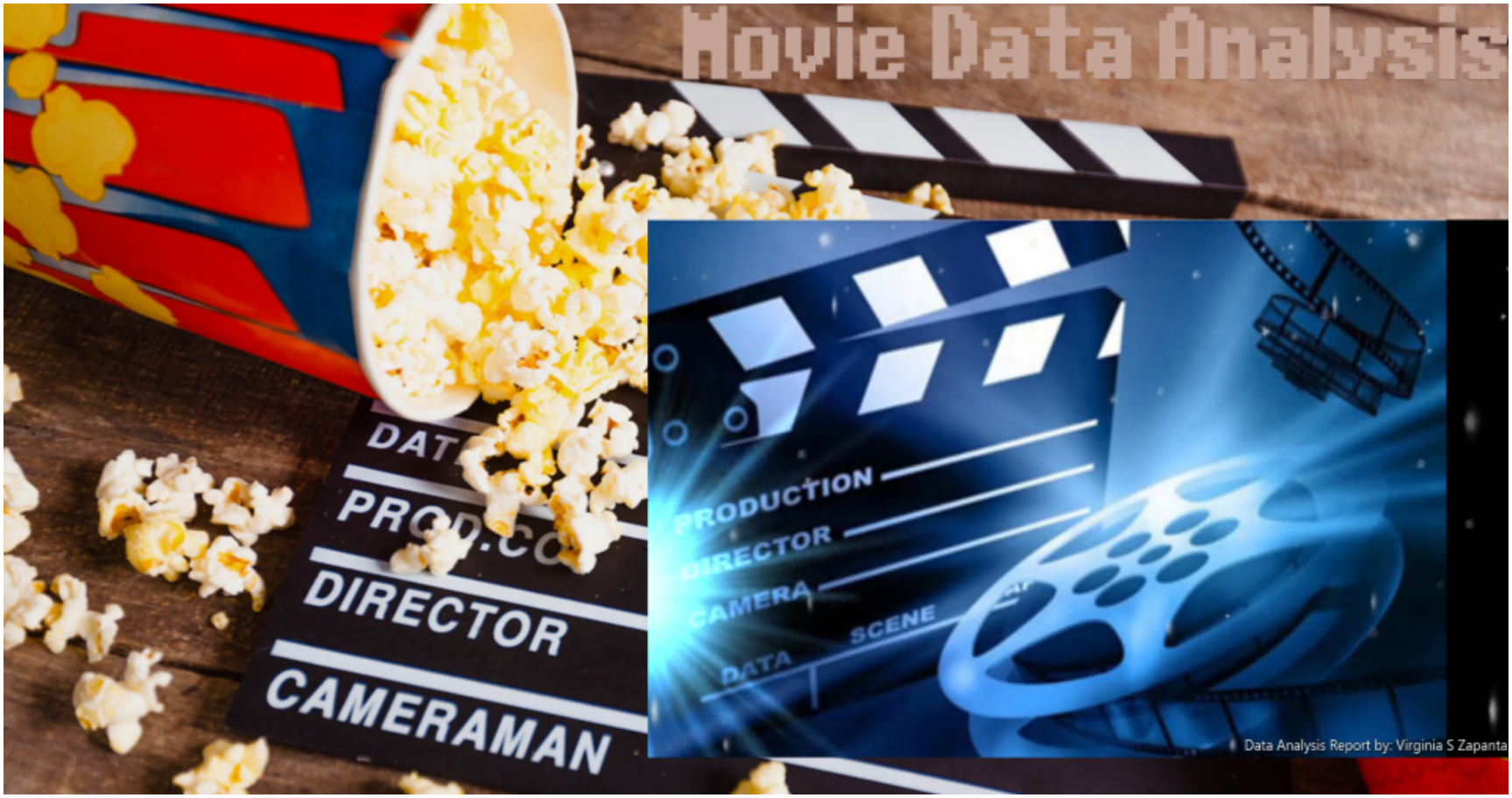
Movie Data Analysis: Assisting Microsoft's Venture Into The Film Industry

by: Virginia S. Zapanta



OVERVIEW

Microsoft recently decided to create original video content and requested my assistance in aims of guiding development of the new movie studio. As an actress and with a history of coordination production producer (my IMDB profile: https://www.imdb.com/name/nm12400698) this is an area that I am quite passionate about.

According to Statista, a leading statistics company on the internet, the movie industry in the United States generated a revenue of more than 25.9 billion dollars, making it the most profitable movie country in the world. It comes with no suprise that many individuals in a variety of facets endeavor to join the multi-billion movie industry.

Indubitably so, the filmmaking business is a risky investment and the purpose of this analysis is to analyze the variables involved that Microsoft should be implementing in order to be lucratively profitable in their newest venture studio business endeavor.

WHAT TOOLS DO I NEED?

To complete this analysis, the following software requirements will be used:

- Python 3
- NumPy
- Pandas - package for data manipulation and analysis in Python.
- SQLite3
- PandaSQL
- Matplotlib
- Seaborn
- Jupyter Notebook
- Anaconda

The DATA

THE FILES

There are six movie dataset files. All six of these files are zipped up. To access these six data files, they need to be unzipped using Python individually.

1. boxoffice.csv.gz

2. rottentomatoes.tsv.gz
3. rottentomatoes.reviews.tsv.gz
4. themoviedatabase.csv.gz
5. thenumbers.csv.gz"
6. im.db.zip

MOVIE DATASETS

In this analysis, randomly selected data for the movies released date years between 1930 through 2022 are provided. I will be analyzing over 4 million records of raw data (4,463,522 records from all data files) provided by Box Office Mojo, Rotten Tomatoes, The Movie Database,The Numbers and the IMDB.

The **Box Office Mojo** csv file contains *five (5)* columns:

- 'title' (e.g., Toy Story 3)
- 'studio'(e.g., BV)
- 'domestic_gross'(e.g., 415000000)
- 'foreign_gross'($ e.g., 652000000)
- 'year'(e.g., 2010)

The **Rotten Tomatoes** tsv file contains *twelve (12)* columns:

- 'id'(e.g.,1890)
- 'synopsis'(e.g.,A misplaced sausage and his savory friends)
- 'rating'(e.g.,R)
- 'genre'(e.g., Comedy)
- 'director'(e.g.,Conrad Vernon)
- 'writer'(e.g., Seth Rogen)
- 'theater_date'(e.g., Aug 12, 2016)
- 'dvd_date'(e.g., Nov 8, 2016)
- 'currency'(e.g., $)
- 'box_office'(e.g., 97,661,826)
- 'runtime'(e.g., 89 minutes)
- 'studio'(e.g., Sony Pictures)

The **Rotten Tomatoes** Reviews tsv file contains *eight (8)* columns:

- 'id'(e.g.,3)
- 'review'(e.g.,For one of the smartest films I've seen)
- 'rating'(e.g.,4/5)
- 'fresh'(fresh or rotten)
- 'critic'(e.g.,Patrick Kolan)
- 'top_critic'(0 or 1)
- 'publisher'(e.g.,Shotgun Cinema)
- 'date'(e.g.,September 26, 2012)

The **Movie Database** csv file contains *nine (9)* columns:

- 'genre_ids'(e.g.,12) -'id'(e.g.,299536)
- 'original_language'(e.g.,en)
- original_title'(e.g.,Avengers: Infinity War)
- 'popularity'(e.g.,80.773)
- 'release_date'(e.g.,2018-04-27)
- 'title'(e.g.,Avengers: Infinity War)
- 'vote_average'(e.g.,8.3)
- 'vote_count'(e.g.,13948)

The **Numbers** csv file contains *six (6)* columns:

- 'id' (e.g.,1)
- 'release_date' (e.g.,Dec 18, 2009)
- 'movie'(e.g.,Avatar)
- 'production_budget' (in dollars, e.g., 425,000,000)
- 'domestic_gross' (in dollars, e.g.,760,507,625)
- 'worldwide_gross' (in dollars, e.g, 2,776,345,279)

The **IMDB** zip file contains *eight(8)* tables:

- directors
- movie_akas

- movie_ratings
- principals
- known_for
- movie_basics
- persons
- writers

> The **movie_basics** table has *six (6)* columns:
>
> - movie_id (e.g.,tt0069049)
> - primary_title (e.g.,The Other Side of the Wind)
> - original_title(e.g.The Other Side of the Wind)
> - start_year(e.g.,2018)
> - runtime_minutes(e.g.,122.0)
> - genres(e.g.,Drama)
>
> The **movie_ratings** table has *three (3)* columns:
>
> - movie_id (e.g.,tt10356526)
> - averagerating(e.g.,8.3)
> - numvotes(e.g.,31)
>
> The **imdb_movie_akas** table has *eight (8)* columns:
>
> - movie_id (e.g.,tt0369610)
> - ordering (e.g.,14)
> - title (e.g.,Jurassic World)
> - region(e.g.,FR)
> - language (e.g.,en)
> - types (e.g.,imdbDisplay)
> - attributes (e.g.,short title)
> - is_original_title (e.g.,1.0)
>
> The **imdb_known_for** table has *two (2)* columns:
>
> - person_id (e.g.,nm0061671)

> - movie_id (e.g.,tt0837562)
>
> The **imdb_principals** table has *six (6)* columns:
>
> - movie_id (e.g.,tt0111414)
> - ordering (e.g.,1)
> - person_id (e.g.,nm0246005)
> - category (e.g.,actor)
> - job (e.g.,producer)
> - characters (e.g.,The Man)
>
> The **imdb_writers** table has *two (2)* columns:
>
> - movie_id (e.g.,tt0285252)
> - person_id (e.g.,nm0899854)
>
> The **imdb_directors** table has *two (2)* columns:
>
> - movie_id (e.g.,tt0878654)
> - person_id (e.g.,nm0089502)
>
> The **imdb_persons** table has *five (5)* columns:
>
> - person_id (e.g.,nm9990381)
> - primary_name (e.g.,Susan Grobes)
> - birth_year (e.g.,)
> - death_year (e.g.,)
> - primary_profession (e.g.,actress)

UNDERSTANDING THE DATA

To explore the data files and to provide recommendations, I will be using Pandas methods and SQL data cleaning methods to better understand the movie datasets.

- What columns are in the datasets?
- Are there any missing values?

- What are the different types of values in each column?

I will be using some useful pandas methods to view the structure and columns.

- df.head()
- df.columns()
- df.info()
- df['column_name'].value_counts()
- df['column_name'].unique()
- df.shape - dataframe has shape:
- df.ndim - dataframe has dimension:
- df.size - dataframe's total # of elements
- df.index - the index of dataframe
- df.values - the data in dataframe

LOADING THE DATA IN A DATAFRAME

I'll start by importing pandas into Python.

In [1]:
```
import pandas as pd

import warnings
warnings.filterwarnings("ignore")
```

Why use pandas? Pandas allows me to load databases of different formats into DataFrame. With Panda DataFrames I can perform the following features for data analysis:

- Easy handling of missing (NaN) values
- Merge different datasets together
- Integrates with Matplotlib which I will be using to create visualizations in Python

- ## BOX OFFICE MOJO DATASET

I will be loading data into Pandas DataFrames from a data file using the pd.read.csv() function.

In [2]:
```
boxoffice=pd.read_csv("DB/boxoffice.csv.gz")
boxoffice
```

Out[2]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
3	Inception	WB	292600000.0	535700000	2010
4	Shrek Forever After	P/DW	238700000.0	513900000	2010
...	...	...	...	...	...
3382	The Quake	Magn.	6200.0	NaN	2018
3383	Edward II (2018 re-release)	FM	4800.0	NaN	2018
3384	El Pacto	Sony	2500.0	NaN	2018
3385	The Swan	Synergetic	2400.0	NaN	2018
3386	An Actor Prepares	Grav.	1700.0	NaN	2018

3387 rows × 5 columns

Now we're really rolling! I've loaded the data set into the most-used pandas data structure, which is called a DataFrame which basically looks like a table.

I have the columns I need and don't need to rename those columns. As you can see, the Pandas indexes the rows of the DataFrame starting from 0

INVESTIGATING OUR DATASET

It's time to explore the dataset.

ATTRIBUTES ABOUT OUR BOX OFFICE MOJO DATASET

THE INFO() METHOD

The info method tells me the number of non-missing values of each column along with the data types of each column.

In [3]:
```
boxoffice.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object 
 4   year            3387 non-null   int64  
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

SHAPE, NDIM, SIZE, VALUES and INDEX

In [4]:
```
# Print some information about BoxOfficeMojo
print('BoxOfficeMojo has shape:', boxoffice.shape)
print('BoxOfficeMojo has dimension:', boxoffice.ndim)
print('BoxOfficeMojo has a total of', boxoffice.size, 'elements')

# Print the index and data of BoxOfficeMojo
print('The data in BoxOfficeMojo is:', boxoffice.values)
print('The index of BoxOfficeMojo is:', boxoffice.index)
```

```
BoxOfficeMojo has shape: (3387, 5)
BoxOfficeMojo has dimension: 2
BoxOfficeMojo has a total of 16935 elements
The data in BoxOfficeMojo is: [['Toy Story 3' 'BV' 415000000.0 '652000000' 2010]
 ['Alice in Wonderland (2010)' 'BV' 334200000.0 '691300000' 2010]
 ['Harry Potter and the Deathly Hallows Part 1' 'WB' 296000000.0
  '664300000' 2010]
 ...
```

```
 ['El Pacto' 'Sony' 2500.0 nan 2018]
 ['The Swan' 'Synergetic' 2400.0 nan 2018]
 ['An Actor Prepares' 'Grav.' 1700.0 nan 2018]]
The index of BoxOfficeMojo is: RangeIndex(start=0, stop=3387, step=1)
```

DATA TYPES

Another check is to see if the data types (or dtypes) have been correctly interpreted.

In [5]:
```
boxoffice.dtypes
```

Out[5]:
```
title              object
studio             object
domestic_gross    float64
foreign_gross      object
year                int64
dtype: object
```

DATA TYPES ISSUES

It looks like there's something wrong with the 'foreign_gross' column. Its datatype is showing as 'object' and I expect numbers are stored as numerical data types. I expect currency numbers to be a 'float64' like the 'domestic_gross' column as shown above.

This anomaly indicates that it probably contains some noninteger values, so I need to take a look.

In [6]:
```
from pandasql import sqldf
mysql=lambda q: sqldf(q,globals())
```

SQL DATA ANALYSIS

I am going to retrieve the data by using Structured Query Language (SQL).

As you can see, the SQL select statement is highly versatile in filtering operations.

In [7]:
```
q=("""select * from boxoffice group by foreign_gross order by foreign_gross""")
mysql(q)
```

Out[7]:

	title	studio	domestic_gross	foreign_gross	year
0	Flipped	WB	1800000.0	None	2010

	title	studio	domestic_gross	foreign_gross	year
1	The Fate of the Furious	Uni.	226000000.0	1,010.0	2017
2	Jurassic World	Uni.	652300000.0	1,019.4	2015
3	Star Wars: The Force Awakens	BV	936700000.0	1,131.6	2015
4	Furious 7	Uni.	353000000.0	1,163.0	2015
...	...	...	...	...	...
1200	Devil and Angel (E Gun Tian Shi)	CL	131000.0	98500000	2015
1201	Maps to the Stars	FCW	351000.0	988000	2015
1202	Sea Rex 3D: Journey to a Prehistoric World	3D	6100000.0	9900000	2010
1203	White Lion	Scre.	NaN	99600	2010
1204	The East	FoxS	2300000.0	99700	2013

1205 rows × 5 columns

In [8]:
```
q=("""select count(*)  from boxoffice group by foreign_gross order by foreign_gross asc""")
mysql(q)
```

Out[8]:

	count(*)
0	1350
1	1
2	1
3	1
4	1
...	...
1200	2
1201	1
1202	7

	count(*)
1203	1
1204	2

1205 rows × 1 columns

And just as I suspected, some of the values in the 'foreign_gross' column are strings which have been used to indicate missing data 'None'.

DATA CLEANING OUR DATASET

CHANGING THE DATA TYPE

Ad I've discussed above, taking a peek over the data types of each column, I've detected a data anomaly with the 'foreign_gross' column and must be fixed. I want to make sure numbers are stored as numerical data types ike the domestic_gross column.

My goal here is to convert the values under the 'foreign_gross' column from strings to be a float64.

In [9]:
```
boxoffice['foreign_gross'] = pd.to_numeric(boxoffice['foreign_gross'], errors='coerce')
```

You can see in the above figure, I have fixed the anomaly. The data type of the 'foreign_gross' column is now 'float64' that is numeric.

MISSING (NaN) VALUES IN OUR DATAFRAME

Pandas assigns NaN values to missing data and as the number of NaN values is not easily visualized, I will use combination of methods to count the number of NaN values in the data

I'm going to combine .isna() and the sum() methods to count the number of NaN values.

PANDAS DATA ANALYSIS

COUNTING THE TOTAL NaN VALUES

In [10]:
```
boxoffice.isna()
```

Out[10]:

	title	studio	domestic_gross	foreign_gross	year
0	False	False	False	False	False

	title	studio	domestic_gross	foreign_gross	year
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
3382	False	False	False	True	False
3383	False	False	False	True	False
3384	False	False	False	True	False
3385	False	False	False	True	False
3386	False	False	False	True	False

3387 rows × 5 columns

As you can see, Pandas returned boolean True or False. for each element if it's a NaN. True values have numerical value 1 and False values have numerical value 0.

To ensure I have all boolean columns, I can call the dtype as this guarantees me that every single value in the entire new dataframe is either True of False.

In [11]:
```
boxoffice.isna().dtypes
```

Out[11]:
```
title             bool
studio            bool
domestic_gross    bool
foreign_gross     bool
year              bool
dtype: object
```

SUMMING A BOOLEAN DATAFRAME

Now, I can count the total number of NaN values by counting the number of True values.

In [12]:
```
boxoffice.isna().sum()
```

Out[12]:
```
title                0
studio               5
domestic_gross      28
foreign_gross     1355
year                 0
dtype: int64
```

I am interested in counting the total NaN values, so I will be using the .sum method twice to add up all the missing values.

In [13]:
```
# Count the number of NaN values
bom_missing =  boxoffice.isna().sum().sum()

# Print the number of missing values
print('Number of NaN values in our Box Office Mojo DataFrame:', bom_missing)
```

```
Number of NaN values in our Box Office Mojo DataFrame: 1388
```

HANDLING NaN VALUES

Now that I know that the Box Office Mojo dataset has missing values, my next step is to decide what to do with the missing values.

PERCENTAGE OF MISSING VALUES

What's the pecentage of missing values in our dataframe? Let's find out.

In [14]:
```
boxoffice.isna().mean().round(4) * 100
```

Out[14]:
```
title              0.00
studio             0.15
domestic_gross     0.83
foreign_gross     40.01
year               0.00
dtype: float64
```

So if we were to delete all rows with NaN values, we would lose this much percentage of our data.

FOREIGN_GROSS COLUMN

In [15]:
```
q=("""select * from boxoffice where foreign_gross is null""")
mysql(q)
```

Out[15]:

	title	studio	domestic_gross	foreign_gross	year

	title	studio	domestic_gross	foreign_gross	year
0	Flipped	WB	1800000.0	None	2010
1	The Polar Express (IMAX re-issue 2010)	WB	673000.0	None	2010
2	Tiny Furniture	IFC	392000.0	None	2010
3	Grease (Sing-a-Long re-issue)	Par.	366000.0	None	2010
4	Last Train Home	Zeit.	288000.0	None	2010
...	...	...	...	...	...
1350	The Quake	Magn.	6200.0	None	2018
1351	Edward II (2018 re-release)	FM	4800.0	None	2018
1352	El Pacto	Sony	2500.0	None	2018
1353	The Swan	Synergetic	2400.0	None	2018
1354	An Actor Prepares	Grav.	1700.0	None	2018

1355 rows × 5 columns

DOMESTIC_GROSS COLUMN

In [16]:
```
q=("""select * from boxoffice where domestic_gross is null order by foreign_gross desc""")
mysql(q)
```

Out[16]:

	title	studio	domestic_gross	foreign_gross	year
0	Secret Superstar	None	None	122000000.0	2017
1	Finding Mr. Right 2	CL	None	114700000.0	2016
2	Surprise - Journey To The West	AR	None	49600000.0	2015
3	Solace	LGP	None	22400000.0	2016
4	22 Bullets	Cdgm.	None	21300000.0	2013
5	Solomon Kane	RTWC	None	19600000.0	2012
6	Jessabelle	LGF	None	7000000.0	2014

	title	studio	domestic_gross	foreign_gross	year
7	Matru Ki Bijlee Ka Mandola	FIP	None	6000000.0	2013
8	The Tall Man	Imag.	None	5200000.0	2012
9	Force	FoxS	None	4800000.0	2011
10	Keith Lemon: The Film	None	None	4000000.0	2012
11	Lula, Son of Brazil	NYer	None	3800000.0	2012
12	14 Blades	RTWC	None	3800000.0	2014
13	Jack and the Cuckoo-Clock Heart	Shout!	None	3400000.0	2014
14	The Snitch Cartel	PI	None	2100000.0	2013
15	All the Boys Love Mandy Lane	RTWC	None	1900000.0	2013
16	The Cup (2012)	Myr.	None	1800000.0	2012
17	It's a Wonderful Afterlife	UTV	None	1300000.0	2010
18	Lila Lila	Crnth	None	1100000.0	2014
19	6 Souls	RTWC	None	852000.0	2013
20	Viral	W/Dim.	None	552000.0	2016
21	Dark Tide	WHE	None	432000.0	2012
22	Celine: Through the Eyes of the World	Sony	None	119000.0	2010
23	White Lion	Scre.	None	99600.0	2010
24	The Green Wave	RF	None	70100.0	2012
25	Badmaash Company	Yash	None	64400.0	2010
26	Empire of Silver	NeoC	None	19000.0	2011
27	Aashayein (Wishes)	Relbig.	None	3800.0	2010

In [17]:
```
q=("""select * from boxoffice where foreign_gross='122000000.0'""")
mysql(q)
```

Out[17]:

	title	studio	domestic_gross	foreign_gross	year

	title	studio	domestic_gross	foreign_gross	year
0	Secret Superstar	None	None	122000000.0	2017

STUDIO COLUMN

In [18]:
```
q=("""select * from boxoffice where studio is null""")
mysql(q)
```

Out[18]:

	title	studio	domestic_gross	foreign_gross	year
0	Outside the Law (Hors-la-loi)	None	96900.0	3300000.0	2010
1	Fireflies in the Garden	None	70600.0	3300000.0	2011
2	Keith Lemon: The Film	None	NaN	4000000.0	2012
3	Plot for Peace	None	7100.0	NaN	2014
4	Secret Superstar	None	NaN	122000000.0	2017

WHAT'S THE BEST WAY TO HANDLE NaN VALUES?

Now that we know the percentage of the missing values in the dataframe, to erase the missing values, we would lose valuable information.

Although there is no equal distribution of domestic and foreign films, there is a preference over domestic films compared to foreign ones. There are other reasons like remaking foreign films may require more than translation, or some foreign films can't make it to the United States.

So, instead of eliminating or dropping the rows or columns with missing values, the suitable option is to replace them with suitable values, that is to replace all NaN values with the value of 0.

FILLING MISSING VALUES

In [19]:
```
boxoffice.describe()
```

Out[19]:

	domestic_gross	foreign_gross	year
count	3.359000e+03	2.032000e+03	3387.000000

	domestic_gross	foreign_gross	year
mean	2.874585e+07	7.505704e+07	2013.958075
std	6.698250e+07	1.375294e+08	2.478141
min	1.000000e+02	6.000000e+02	2010.000000
25%	1.200000e+05	3.775000e+06	2012.000000
50%	1.400000e+06	1.890000e+07	2014.000000
75%	2.790000e+07	7.505000e+07	2016.000000
max	9.367000e+08	9.605000e+08	2018.000000

In [20]:
```
boxoffice=boxoffice.fillna(0)
```

In [21]:
```
boxoffice.isna().mean().round(4) * 100
```

Out[21]:
```
title             0.0
studio            0.0
domestic_gross    0.0
foreign_gross     0.0
year              0.0
dtype: float64
```

In [22]:
```
boxoffice.describe()
```

Out[22]:

	domestic_gross	foreign_gross	year
count	3.387000e+03	3.387000e+03	3387.000000
mean	2.850821e+07	4.502979e+07	2013.958075
std	6.675575e+07	1.126843e+08	2.478141
min	0.000000e+00	0.000000e+00	2010.000000
25%	1.115000e+05	0.000000e+00	2012.000000
50%	1.300000e+06	1.500000e+06	2014.000000
75%	2.750000e+07	2.915000e+07	2016.000000

	domestic_gross	foreign_gross	year
max	9.367000e+08	9.605000e+08	2018.000000

In [23]:
```
q=("""select * from boxoffice group by foreign_gross order by foreign_gross""")
mysql(q)
```

Out[23]:

	title	studio	domestic_gross	foreign_gross	year
0	Flipped	WB	1800000.0	0.0	2010
1	Chasing Mavericks	Fox	6000000.0	600.0	2012
2	To Die Like a Man	Strand	4000.0	900.0	2011
3	The Red Baron	Mont.	37200.0	3100.0	2010
4	Client 9: The Rise and Fall of Eliot Spitzer	Magn.	189000.0	3500.0	2010
...	...	...	...	...	...
1195	Frozen	BV	400700000.0	875700000.0	2013
1196	Jurassic World: Fallen Kingdom	Uni.	417700000.0	891800000.0	2018
1197	Marvel's The Avengers	BV	623400000.0	895500000.0	2012
1198	Avengers: Age of Ultron	BV	459000000.0	946400000.0	2015
1199	Harry Potter and the Deathly Hallows Part 2	WB	381000000.0	960500000.0	2011

1200 rows × 5 columns

In [24]:
```
boxoffice.head(15)
```

Out[24]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000.0	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000.0	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000.0	2010
3	Inception	WB	292600000.0	535700000.0	2010

	title	studio	domestic_gross	foreign_gross	year
4	Shrek Forever After	P/DW	238700000.0	513900000.0	2010
5	The Twilight Saga: Eclipse	Sum.	300500000.0	398000000.0	2010
6	Iron Man 2	Par.	312400000.0	311500000.0	2010
7	Tangled	BV	200800000.0	391000000.0	2010
8	Despicable Me	Uni.	251500000.0	291600000.0	2010
9	How to Train Your Dragon	P/DW	217600000.0	277300000.0	2010
10	Clash of the Titans (2010)	WB	163200000.0	330000000.0	2010
11	The Chronicles of Narnia: The Voyage of the Da...	Fox	104400000.0	311300000.0	2010
12	The King's Speech	Wein.	135500000.0	275400000.0	2010
13	Tron Legacy	BV	172100000.0	228000000.0	2010
14	The Karate Kid	Sony	176600000.0	182500000.0	2010

In [25]:
```
boxoffice.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   title           3387 non-null   object
 1   studio          3387 non-null   object
 2   domestic_gross  3387 non-null   float64
 3   foreign_gross   3387 non-null   float64
 4   year            3387 non-null   int64
dtypes: float64(2), int64(1), object(2)
memory usage: 132.4+ KB
```

In [26]:
```
q=("""select min(year), max(year) from boxoffice
""")
mysql(q)
```

Out[26]:

	min(year)	max(year)

	min(year)	max(year)
0	2010	2018

In [27]:
```
studio_counts = boxoffice.studio.value_counts()
top_studios = studio_counts.head(10)
studio_counts.head(10)
```

Out[27]:
```
IFC      166
Uni.     147
WB       140
Fox      136
Magn.    136
SPC      123
Sony     110
BV       106
LGF      103
Par.     101
Name: studio, dtype: int64
```

In [28]:
```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [29]:
```
q=("""select studio,count(studio) as NumMovies, sum(domestic_gross+foreign_gross) as TotalGross
from boxoffice group by studio order by TotalGross desc limit 10
""")
top5studio_numfilms = mysql(q)
top5studio_numfilms
```

Out[29]:

	studio	NumMovies	TotalGross
0	BV	106	4.421288e+10
1	Fox	136	3.100537e+10
2	WB	140	3.083595e+10
3	Uni.	147	2.975716e+10
4	Sony	110	2.240504e+10

	studio	NumMovies	TotalGross
5	Par.	101	1.954926e+10
6	WB (NL)	45	1.033470e+10
7	LGF	103	8.601583e+09
8	LG/S	41	5.431924e+09
9	P/DW	10	5.076500e+09

In [30]:
```
type(top5studio_numfilms)
```

Out[30]:
```
pandas.core.frame.DataFrame
```

In [31]:
```
q=("""select title, domestic_gross
from boxoffice  order by domestic_gross desc limit 10
""")
topmovies_domestic = mysql(q)
topmovies_domestic
```

Out[31]:

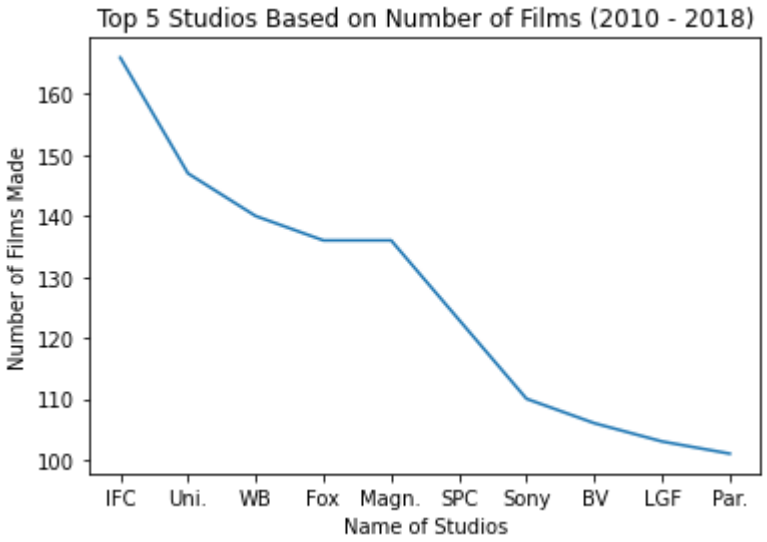
	title	domestic_gross
0	Star Wars: The Force Awakens	936700000.0
1	Black Panther	700100000.0
2	Avengers: Infinity War	678800000.0
3	Jurassic World	652300000.0
4	Marvel's The Avengers	623400000.0
5	Star Wars: The Last Jedi	620200000.0
6	Incredibles 2	608600000.0
7	Rogue One: A Star Wars Story	532200000.0
8	Beauty and the Beast (2017)	504000000.0
9	Finding Dory	486300000.0

VISUALIZATIONS

TOP 10 STUDIOS BASED ON NUMBER OF FILMS

In [32]:

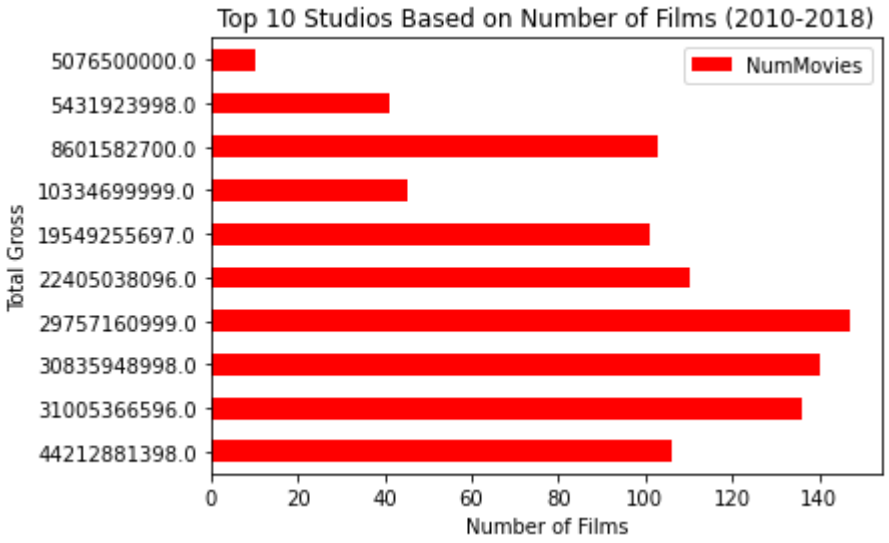
```
ax = sns.lineplot(top_studios.index, top_studios.values)
ax.set_title('Top 5 Studios Based on Number of Films (2010 - 2018)')
ax.set_xlabel('Name of Studios')
ax.set_ylabel('Number of Films Made');
```



TOP HIGHEST GROSS BASED ON NUMBER OF FILMS

In [33]:

```
# Plotting  Bar Chart
top5studio_numfilms.plot.barh(x='TotalGross', y='NumMovies', title='Top 10 Studios Based on Number of Films (2010-2018)',

# set the labels
plt.xlabel("Number of Films" )
plt.ylabel("Total Gross" )
# display the plotted  Bar Chart
plt.show()
```



TOP 10 FILMS WITH THE HIGHEST DOMESTIC GROSS

In [34]:
```
q=("""select title,studio,domestic_gross from boxoffice order by domestic_gross desc limit 10
""")
top10domesticfilms=mysql(q)
top10domesticfilms
```

Out[34]:

	title	studio	domestic_gross
0	Star Wars: The Force Awakens	BV	936700000.0
1	Black Panther	BV	700100000.0
2	Avengers: Infinity War	BV	678800000.0
3	Jurassic World	Uni.	652300000.0
4	Marvel's The Avengers	BV	623400000.0
5	Star Wars: The Last Jedi	BV	620200000.0
6	Incredibles 2	BV	608600000.0
7	Rogue One: A Star Wars Story	BV	532200000.0
8	Beauty and the Beast (2017)	BV	504000000.0
9	Finding Dory	BV	486300000.0

In [35]:
```
q=("""select studio, domestic_gross,count(*) as NumFilms from boxoffice group by studio order by domestic_gross desc limi
""")
bom_highestgross=mysql(q)
bom_highestgross
```

Out[35]:

	studio	domestic_gross	NumFilms
0	BV	415000000.0	106
1	Par.	312400000.0	101
2	Sum.	300500000.0	15
3	WB	296000000.0	140
4	LG/S	292300000.0	41
5	Uni.	251500000.0	147
6	P/DW	238700000.0	10
7	Sony	176600000.0	110
8	Wein.	135500000.0	77
9	FoxS	107000000.0	67

In [36]:
```
bom_highestgross.describe()
```

Out[36]:

	domestic_gross	NumFilms
count	1.000000e+01	10.000000
mean	2.525500e+08	81.400000
std	9.224727e+07	48.201429
min	1.070000e+08	10.000000
25%	1.921250e+08	47.500000
50%	2.719000e+08	89.000000
75%	2.993750e+08	109.000000

	domestic_gross	NumFilms
max	4.150000e+08	147.000000

RECOMMENDATION ANALYSIS #1:

- Based on the data charts, the best studio to work with is Walt Disney Studios Motion Pictures, formerly known as Buena Vista Pictures (BV). The second option is to work with Summit Entertainment (Sum.)
- Nine (9) out of top (10) films with the highest domestic gross were produced by BV (Walt Disney).
- There is no high correlation between the total number of films made and gross sales

Studio domestic_gross NumFilms BV 415000000.0 106 Sum. 300500000.0 15

ROTTEN TOMATOES MOVIES DATASETS

rottentomatoesmovies=pd.read_csv("DB/rottentomatoes.tsv.gz", delimiter="\t") rottentomatoesmovies

In [37]:
```
rottenreviews=pd.read_csv("DB/rottentomatoes.reviews.tsv.gz",  delimiter="\t", encoding='latin1')
rottenreviews
```

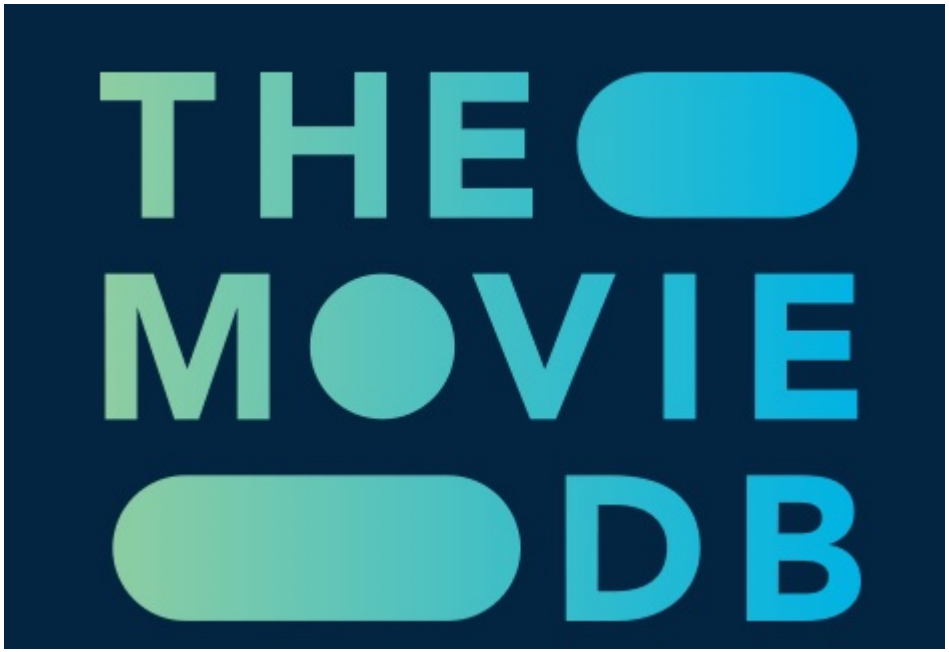
Out[37]:

	id	review	rating	fresh	critic	top_critic	publisher	date

	id	review	rating	fresh	critic	top_critic	publisher	date
0	3	A distinctly gallows take on contemporary fina...	3/5	fresh	PJ Nabarro	0	Patrick Nabarro	November 10, 2018
1	3	It's an allegory in search of a meaning that n...	NaN	rotten	Annalee Newitz	0	io9.com	May 23, 2018
2	3	... life lived in a bubble in financial dealin...	NaN	fresh	Sean Axmaker	0	Stream on Demand	January 4, 2018
3	3	Continuing along a line introduced in last yea...	NaN	fresh	Daniel Kasman	0	MUBI	November 16, 2017
4	3	... a perverse twist on neorealism...	NaN	fresh	NaN	0	Cinema Scope	October 12, 2017
...	...	...	...	...	...	...	...	...
54427	2000	The real charm of this trifle is the deadpan c...	NaN	fresh	Laura Sinagra	1	Village Voice	September 24, 2002
54428	2000	NaN	1/5	rotten	Michael Szymanski	0	Zap2it.com	September 21, 2005
54429	2000	NaN	2/5	rotten	Emanuel Levy	0	EmanuelLevy.Com	July 17, 2005
54430	2000	NaN	2.5/5	rotten	Christopher Null	0	Filmcritic.com	September 7, 2003
54431	2000	NaN	3/5	fresh	Nicolas Lacroix	0	Showbizz.net	November 12, 2002

54432 rows × 8 columns

THE MOVIE DATABASE DATASET



In [38]:
```
themoviedb=pd.read_csv("DB/themoviedatabase.csv.gz")
themoviedb
```

Out[38]:

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

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average	vote_count
26512	26512	[27, 18]	488143	en	Laboratory Conditions	0.600	2018-10-13	Laboratory Conditions	0.0	1
26513	26513	[18, 53]	485975	en	_EXHIBIT_84xxx_	0.600	2018-05-01	_EXHIBIT_84xxx_	0.0	1
26514	26514	[14, 28, 12]	381231	en	The Last One	0.600	2018-10-01	The Last One	0.0	1
26515	26515	[10751, 12, 28]	366854	en	Trailer Made	0.600	2018-06-22	Trailer Made	0.0	1
26516	26516	[53, 27]	309885	en	The Church	0.600	2018-10-05	The Church	0.0	1

26517 rows × 10 columns

Dropping 'Unnamed:0' column

In [39]:
```
# # Drop 'unnamed: 0' column
themoviedb.drop('Unnamed: 0', axis=1, inplace=True)
```

In [40]:
```
themoviedb
```

Out[40]:

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

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average	vote_count
...	...	...	...	...	...	...	...	...	...
26512	[27, 18]	488143	en	Laboratory Conditions	0.600	2018-10-13	Laboratory Conditions	0.0	1
26513	[18, 53]	485975	en	_EXHIBIT_84xxx_	0.600	2018-05-01	_EXHIBIT_84xxx_	0.0	1
26514	[14, 28, 12]	381231	en	The Last One	0.600	2018-10-01	The Last One	0.0	1
26515	[10751, 12, 28]	366854	en	Trailer Made	0.600	2018-06-22	Trailer Made	0.0	1
26516	[53, 27]	309885	en	The Church	0.600	2018-10-05	The Church	0.0	1

26517 rows × 9 columns

THE NUMBERS DATASET

In [41]:
```
thenumbers=pd.read_csv("DB/thenumbers.csv.gz")
thenumbers
```

Out[41]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross

	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
...	...	...	...	...	...	...
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

5782 rows × 6 columns

Missing Values

In [42]:
```
thenumbers.isna().sum()
```

Out[42]:
```
id                 0
release_date       0
movie              0
production_budget  0
domestic_gross     0
worldwide_gross    0
dtype: int64
```

Duplicated Values

In [43]:
```
thenumbers.duplicated().sum()
```

Out[43]:
```
0
```

There are no duplicated rows and there are no missing values. Let's check the data types.

Stripping Characters In the Numbers Column

In [44]:
```
thenumbers.head()
```

Out[44]:

	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

I am going to remove the following characters from the numbers columns: $ and , (comma)

In [45]:
```
# production_budget column
thenumbers['production_budget'] = thenumbers['production_budget'].str.replace(',', '')
thenumbers['production_budget'] = thenumbers['production_budget'].str.replace('$', '')

# domestic_gross column
thenumbers['domestic_gross'] = thenumbers['domestic_gross'].str.replace(',', '')
thenumbers['domestic_gross'] = thenumbers['domestic_gross'].str.replace('$', '')

# worldwide_gross column
thenumbers['worldwide_gross'] = thenumbers['worldwide_gross'].str.replace(',', '')
thenumbers['worldwide_gross'] = thenumbers['worldwide_gross'].str.replace('$', '')
```

In [46]:
```
thenumbers.head()
```

Out[46]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
2	3	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350
3	4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747

Investigating Data Types

In [47]:
```
thenumbers.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
```

Changing Data Types

Numeric Columns Converted to Float Type

In [48]:
```
thenumbers['production_budget'] = thenumbers['production_budget'].astype(float)

thenumbers['domestic_gross'] = thenumbers['domestic_gross'].astype(float)

thenumbers['worldwide_gross'] = thenumbers['worldwide_gross'].astype(float)
```

Date Columns Converted to Datetime Type

In [49]:
```
thenumbers["release_date"] = pd.to_datetime(thenumbers["release_date"])
```

In [50]:
```
thenumbers.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   datetime64[ns]
 2   movie              5782 non-null   object        
 3   production_budget  5782 non-null   float64       
 4   domestic_gross     5782 non-null   float64       
 5   worldwide_gross    5782 non-null   float64       
dtypes: datetime64[ns](1), float64(3), int64(1), object(1)
memory usage: 271.2+ KB
```

SQL Filtering

As you can see, we have fixed the datatypes issues.

THE IMDb DATASETS

In [51]:

```
import sqlite3
```

In [52]:
```
connect = sqlite3.connect("imdb.db")
```

In [53]:
```
%%script sqlite3 imdb.db
.table
```

```
RTmoviestable  known_for      movie_basics   persons        writers
directors      movie_akas     movie_ratings  principals
```

IMDb MOVIE_BASICS TABLE

In [54]:
```
imdb_movie_basics =pd.read_sql("select * from movie_basics;",connect)
imdb_movie_basics
```

Out[54]:

	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
...	...	...	...	...	...	...
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	None
146143	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	NaN	Documentary

146144 rows × 6 columns

SPLITTING GENRES

In [55]:
```
imdb_movie_basics_expand = imdb_movie_basics.genres.str.split(",", expand=True)
```

In [56]:
```
imdb_movie_basics['genres_1']=imdb_movie_basics_expand[0]
imdb_movie_basics['genres_2']=imdb_movie_basics_expand[1]
imdb_movie_basics['genres_3']=imdb_movie_basics_expand[2]
#imdb_movie_basics.drop('genres', axis=1, inplace=True)
imdb_movie_basics.set_index('movie_id', inplace=True)
```

In [57]:
```
imdb_movie_basics
```

Out[57]:

movie_id	primary_title	original_title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3
tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama	Action	Crime	Drama
tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama	Biography	Drama	None
tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama	Drama	None	None
tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama	Comedy	Drama	None
tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy	Comedy	Drama	Fantasy
...	...	...	...	...	...	...	...	...
tt9916538	Kuambil Lagi Hatiku	Kuambil Lagi Hatiku	2019	123.0	Drama	Drama	None	None
tt9916622	Rodolpho Teóphilo - O Legado de um Pioneiro	Rodolpho Teóphilo - O Legado de um Pioneiro	2015	NaN	Documentary	Documentary	None	None
tt9916706	Dankyavar Danka	Dankyavar Danka	2013	NaN	Comedy	Comedy	None	None
tt9916730	6 Gunn	6 Gunn	2017	116.0	None	None	None	None
tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	NaN	Documentary	Documentary	None	None

146144 rows × 8 columns

As you can see, I split the 'genres' column into three columns:

- genres_1
- genres_2
- genres_3

In [58]:
```
imdb_movie_basics.isna().sum()
```

Out[58]:
```
primary_title           0
original_title         21
start_year              0
runtime_minutes     31739
genres               5408
genres_1             5408
genres_2            86766
genres_3           116708
dtype: int64
```

DROPPED DUPLICATES

In [59]:
```
imdb_movie_basics.duplicated().sum()
```

Out[59]:
```
122
```

In [60]:
```
imdb_movie_basics.drop_duplicates(inplace=True)
```

In [61]:
```
imdb_movie_basics.duplicated().sum()
```

Out[61]:
```
0
```

In [62]:
```
imdb_movie_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 146022 entries, tt0063540 to tt9916754
Data columns (total 8 columns):
```

```
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   primary_title    146022 non-null  object
 1   original_title   146001 non-null  object
 2   start_year       146022 non-null  int64
 3   runtime_minutes  114342 non-null  float64
 4   genres           140627 non-null  object
 5   genres_1         140627 non-null  object
 6   genres_2         59359 non-null   object
 7   genres_3         29433 non-null   object
dtypes: float64(1), int64(1), object(6)
memory usage: 10.0+ MB
```

In [63]:
```
imdb_movie_basics
```

Out[63]:

movie_id	primary_title	original_title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3
tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama	Action	Crime	Drama
tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama	Biography	Drama	None
tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama	Drama	None	None
tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama	Comedy	Drama	None
tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy	Comedy	Drama	Fantasy
...	...	...	...	...	...	...	...	...
tt9916538	Kuambil Lagi Hatiku	Kuambil Lagi Hatiku	2019	123.0	Drama	Drama	None	None
tt9916622	Rodolpho Teóphilo - O Legado de um Pioneiro	Rodolpho Teóphilo - O Legado de um Pioneiro	2015	NaN	Documentary	Documentary	None	None
tt9916706	Dankyavar Danka	Dankyavar Danka	2013	NaN	Comedy	Comedy	None	None
tt9916730	6 Gunn	6 Gunn	2017	116.0	None	None	None	None
tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	NaN	Documentary	Documentary	None	None

146022 rows × 8 columns

DROP REDUNDANT COLUMN

Since both primary_title and original_title columns have identical values, it is redundant to keep the two columns in our dataframe. I am going to keep the 'primary_title' column which contains more non-null values.

Column Non-Null Count Dtype ------ -------------- ----- primary_title 146144 non-null object original_title 146123 non-null object

In [64]:
```
imdb_movie_basics = imdb_movie_basics.drop(['original_title'],
axis=1)
```

RENAME COLUMN

I am going to shorten the 'primary_title' column and rename it to 'title' column.

In [65]:
```
imdb_movie_basics.rename(columns = {'primary_title':'title'}, inplace = True)
```

CLEANED IMDB MOVIE BASICS TABLE

All duplicates and redundancy have been fixed.

In [66]:
```
imdb_movie_basics
```

Out[66]:

movie_id	title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3
tt0063540	Sunghursh	2013	175.0	Action,Crime,Drama	Action	Crime	Drama
tt0066787	One Day Before the Rainy Season	2019	114.0	Biography,Drama	Biography	Drama	None
tt0069049	The Other Side of the Wind	2018	122.0	Drama	Drama	None	None
tt0069204	Sabse Bada Sukh	2018	NaN	Comedy,Drama	Comedy	Drama	None
tt0100275	The Wandering Soap Opera	2017	80.0	Comedy,Drama,Fantasy	Comedy	Drama	Fantasy
...	...	...	...	...	...	...	...

movie_id	title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3
tt9916538	Kuambil Lagi Hatiku	2019	123.0	Drama	Drama	None	None
tt9916622	Rodolpho Teóphilo - O Legado de um Pioneiro	2015	NaN	Documentary	Documentary	None	None
tt9916706	Dankyavar Danka	2013	NaN	Comedy	Comedy	None	None
tt9916730	6 Gunn	2017	116.0	None	None	None	None
tt9916754	Chico Albuquerque - Revelações	2013	NaN	Documentary	Documentary	None	None

146022 rows × 7 columns

IMDb MOVIE_RATINGS TABLE

In [67]:
```
imdb_movie_ratings =pd.read_sql("select * from movie_ratings;",connect)
imdb_movie_ratings
```

Out[67]:

	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
...	...	...	...
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

73856 rows × 3 columns

INVESTIGATING DUPLICATES, MISSING VALUES, DATA TYPES

In [68]:
```
imdb_movie_ratings.duplicated().sum()
```

Out[68]:
```
0
```

In [69]:
```
imdb_movie_ratings.isna().sum()
```

Out[69]:
```
movie_id         0
averagerating    0
numvotes         0
dtype: int64
```

In [70]:
```
imdb_movie_ratings.set_index('movie_id', inplace=True)
```

In [71]:
```
imdb_movie_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73856 entries, tt10356526 to tt9894098
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   averagerating  73856 non-null  float64
 1   numvotes       73856 non-null  int64  
dtypes: float64(1), int64(1)
memory usage: 1.7+ MB
```

In [72]:
```
imdb_movie_ratings
```

Out[72]:

movie_id	averagerating	numvotes
tt10356526	8.3	31
tt10384606	8.9	559

movie_id	averagerating	numvotes
tt1042974	6.4	20
tt1043726	4.2	50352
tt1060240	6.5	21
...	...	...
tt9805820	8.1	25
tt9844256	7.5	24
tt9851050	4.7	14
tt9886934	7.0	5
tt9894098	6.3	128

73856 rows × 2 columns

Analysis

There are no duplicates, no missing values and all data types are correct.

IMDb MOVIE_AKAS TABLE

In [73]:
```
imdb_movie_akas =pd.read_sql("select * from movie_akas;",connect)
imdb_movie_akas
```

Out[73]:

	movie_id	ordering	title	region	language	types	attributes	is_original_title
0	tt0369610	10	Джурасик свят	BG	bg	None	None	0.0
1	tt0369610	11	Jurashikku warudo	JP	None	imdbDisplay	None	0.0
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	None	imdbDisplay	None	0.0
3	tt0369610	13	O Mundo dos Dinossauros	BR	None	None	short title	0.0
4	tt0369610	14	Jurassic World	FR	None	imdbDisplay	None	0.0

	movie_id	ordering	title	region	language	types	attributes	is_original_title
...	...	...	...	...	...	...	...	...
331698	tt9827784	2	Sayonara kuchibiru	None	None	original	None	1.0
331699	tt9827784	3	Farewell Song	XWW	en	imdbDisplay	None	0.0
331700	tt9880178	1	La atención	None	None	original	None	1.0
331701	tt9880178	2	La atención	ES	None	None	None	0.0
331702	tt9880178	3	The Attention	XWW	en	imdbDisplay	None	0.0

331703 rows × 8 columns

INVESTIGATING MISSING VALUES AND DUPLICATES

In [74]:
```
imdb_movie_akas.isna().sum()
```

Out[74]:
```
movie_id                0
ordering                0
title                   0
region              53293
language           289988
types              163256
attributes         316778
is_original_title      25
dtype: int64
```

In [75]:
```
imdb_movie_akas.duplicated().sum()
```

Out[75]:
```
0
```

DROPPED UNNECESSARY COLUMNS

In [76]:
```
# drop unnecessary columns.
imdb_movie_akas = imdb_movie_akas.drop(['language',
'types','attributes','region','is_original_title'],
axis=1)
```

In [77]:
```
imdb_movie_akas
```

Out[77]:

	movie_id	ordering	title
0	tt0369610	10	Джурасик свят
1	tt0369610	11	Jurashikku warudo
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros
3	tt0369610	13	O Mundo dos Dinossauros
4	tt0369610	14	Jurassic World
...	...	...	...
331698	tt9827784	2	Sayonara kuchibiru
331699	tt9827784	3	Farewell Song
331700	tt9880178	1	La atención
331701	tt9880178	2	La atención
331702	tt9880178	3	The Attention

331703 rows × 3 columns

MISSING VALUES RESOLVED

In [78]:
```
imdb_movie_akas.isna().sum()
```

Out[78]:
```
movie_id    0
ordering    0
title       0
dtype: int64
```

IMDb IMDB_KNOWN_FOR TABLE

In [79]:
```
imdb_known_for =pd.read_sql("select * from known_for ;",connect)
imdb_known_for
```

Out[79]:

	person_id	movie_id
0	nm0061671	tt0837562
1	nm0061671	tt2398241
2	nm0061671	tt0844471
3	nm0061671	tt0118553
4	nm0061865	tt0896534
...	...	...
1638255	nm9990690	tt9090932
1638256	nm9990690	tt8737130
1638257	nm9991320	tt8734436
1638258	nm9991320	tt9615610
1638259	nm9993380	tt8743182

1638260 rows × 2 columns

In [80]:
```
imdb_known_for.duplicated().sum()
```

Out[80]:
```
0
```

In [81]:
```
imdb_known_for.isna().sum()
```

Out[81]:
```
person_id    0
movie_id     0
dtype: int64
```

In [82]:
```
imdb_known_for.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1638260 entries, 0 to 1638259
Data columns (total 2 columns):
 #   Column     Non-Null Count    Dtype 
---  ------     --------------    ----- 
 0   person_id  1638260 non-null  object
```

```
 1   movie_id   1638260 non-null  object
dtypes: object(2)
memory usage: 25.0+ MB
```

IMDb IMDB_PRINCIPALS TABLE

In [83]:
```
imdb_principals=pd.read_sql("select * from principals;",connect)
imdb_principals
```

Out[83]:

	movie_id	ordering	person_id	category	job	characters
0	tt0111414	1	nm0246005	actor	None	["The Man"]
1	tt0111414	2	nm0398271	director	None	None
2	tt0111414	3	nm3739909	producer	producer	None
3	tt0323808	10	nm0059247	editor	None	None
4	tt0323808	1	nm3579312	actress	None	["Beth Boothby"]
...	...	...	...	...	...	...
1028181	tt9692684	1	nm0186469	actor	None	["Ebenezer Scrooge"]
1028182	tt9692684	2	nm4929530	self	None	["Herself","Regan"]
1028183	tt9692684	3	nm10441594	director	None	None
1028184	tt9692684	4	nm6009913	writer	writer	None
1028185	tt9692684	5	nm10441595	producer	producer	None

1028186 rows × 6 columns

In [84]:
```
imdb_principals.isna().sum()
```

Out[84]:
```
movie_id           0
ordering           0
person_id          0
category           0
job           850502
```

```
characters    634826
dtype: int64
```

In [85]:
```
imdb_principals.duplicated().sum()
```

Out[85]:
```
0
```

DROP REDUNDANT COLUMN

In [86]:
```
# drop unnecessary columns. Both category and job columns are the same. Drop job column and rename
# category column to profession1
imdb_principals = imdb_principals.drop(['job'],
axis=1)
```

In [87]:
```
imdb_principals
```

Out[87]:

	movie_id	ordering	person_id	category	characters
0	tt0111414	1	nm0246005	actor	["The Man"]
1	tt0111414	2	nm0398271	director	None
2	tt0111414	3	nm3739909	producer	None
3	tt0323808	10	nm0059247	editor	None
4	tt0323808	1	nm3579312	actress	["Beth Boothby"]
...	...	...	...	...	...
1028181	tt9692684	1	nm0186469	actor	["Ebenezer Scrooge"]
1028182	tt9692684	2	nm4929530	self	["Herself","Regan"]
1028183	tt9692684	3	nm10441594	director	None
1028184	tt9692684	4	nm6009913	writer	None
1028185	tt9692684	5	nm10441595	producer	None

1028186 rows × 5 columns

In [88]:
```
imdb_principals.isna().sum()
```

Out[88]:
```
movie_id           0
ordering           0
person_id          0
category           0
characters    634826
dtype: int64
```

RENAME COLUMN

In [89]:
```
imdb_principals.rename(columns = {'category':'profession_1'}, inplace = True)
```

In [90]:
```
imdb_principals
```

Out[90]:

	movie_id	ordering	person_id	profession_1	characters
0	tt0111414	1	nm0246005	actor	["The Man"]
1	tt0111414	2	nm0398271	director	None
2	tt0111414	3	nm3739909	producer	None
3	tt0323808	10	nm0059247	editor	None
4	tt0323808	1	nm3579312	actress	["Beth Boothby"]
...	...	...	...	...	...
1028181	tt9692684	1	nm0186469	actor	["Ebenezer Scrooge"]
1028182	tt9692684	2	nm4929530	self	["Herself","Regan"]
1028183	tt9692684	3	nm10441594	director	None
1028184	tt9692684	4	nm6009913	writer	None
1028185	tt9692684	5	nm10441595	producer	None

1028186 rows × 5 columns

In [91]:
```
imdb_principals.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028186 entries, 0 to 1028185
Data columns (total 5 columns):
 #   Column        Non-Null Count    Dtype 
---  ------        --------------    ----- 
 0   movie_id      1028186 non-null  object
 1   ordering      1028186 non-null  int64 
 2   person_id     1028186 non-null  object
 3   profession_1  1028186 non-null  object
 4   characters    393360 non-null   object
dtypes: int64(1), object(4)
memory usage: 39.2+ MB
```

IMDb IMDB_WRITERS TABLE

In [92]:

```
imdb_writers=pd.read_sql("select * from writers;",connect)
imdb_writers
```

Out[92]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0438973	nm0175726
2	tt0438973	nm1802864
3	tt0462036	nm1940585
4	tt0835418	nm0310087
...	...	...
255868	tt8999892	nm10122246
255869	tt8999974	nm10122357
255870	tt9001390	nm6711477
255871	tt9004986	nm4993825
255872	tt9010172	nm8352242

255873 rows × 2 columns

RESOLVED DUPLICATES

In [93]:
```
imdb_writers.duplicated().sum()
```

Out[93]:
```
77521
```

In [94]:
```
imdb_writers.drop_duplicates(inplace=True)
```

In [95]:
```
imdb_writers.duplicated().sum()
```

Out[95]:
```
0
```

MISSING VALUES

In [96]:
```
imdb_writers.isna().sum()
```

Out[96]:
```
movie_id     0
person_id    0
dtype: int64
```

DATATYPES

In [97]:
```
imdb_writers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 178352 entries, 0 to 255872
Data columns (total 2 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   movie_id   178352 non-null  object
 1   person_id  178352 non-null  object
dtypes: object(2)
memory usage: 4.1+ MB
```

In [98]:
```
imdb_writers
```

Out[98]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0438973	nm0175726
2	tt0438973	nm1802864
3	tt0462036	nm1940585
4	tt0835418	nm0310087
...	...	...
255868	tt8999892	nm10122246
255869	tt8999974	nm10122357
255870	tt9001390	nm6711477
255871	tt9004986	nm4993825
255872	tt9010172	nm8352242

178352 rows × 2 columns

IMDb_DIRECTORS TABLE

In [99]:

```
imdb_directors=pd.read_sql("select * from directors;",connect)
imdb_directors
```

Out[99]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0462036	nm1940585
2	tt0835418	nm0151540
3	tt0835418	nm0151540
4	tt0878654	nm0089502
...	...	...
291169	tt8999974	nm10122357

	movie_id	person_id
291170	tt9001390	nm6711477
291171	tt9001494	nm10123242
291172	tt9001494	nm10123248
291173	tt9004986	nm4993825

291174 rows × 2 columns

RESOLVED DUPLICATES

In [100…
```
imdb_directors.duplicated().sum()
```

Out[100…
```
127639
```

In [101…
```
imdb_directors.drop_duplicates(inplace=True)
```

In [102…
```
imdb_directors.duplicated().sum()
```

Out[102…
```
0
```

MISSING VALUES

In [103…
```
imdb_directors.isna().sum()
```

Out[103…
```
movie_id     0
person_id    0
dtype: int64
```

DATATYPES

In [104…
```
imdb_directors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 163535 entries, 0 to 291173
Data columns (total 2 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   movie_id   163535 non-null  object
 1   person_id  163535 non-null  object
dtypes: object(2)
memory usage: 3.7+ MB
```

In [105…

```
imdb_directors
```

Out[105…

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0462036	nm1940585
2	tt0835418	nm0151540
4	tt0878654	nm0089502
5	tt0878654	nm2291498
...	...	...
291169	tt8999974	nm10122357
291170	tt9001390	nm6711477
291171	tt9001494	nm10123242
291172	tt9001494	nm10123248
291173	tt9004986	nm4993825

163535 rows × 2 columns

IMDb_PERSONS TABLE

In [106…

```
imdb_persons=pd.read_sql("select * from persons;",connect)
imdb_persons
```

Out[106…

	person_id	primary_name	birth_year	death_year	primary_profession
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,producer
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_department
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,actor,writer
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_department
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_decorator
...	...	...	...	...	...
606643	nm9990381	Susan Grobes	NaN	NaN	actress
606644	nm9990690	Joo Yeon So	NaN	NaN	actress
606645	nm9991320	Madeline Smith	NaN	NaN	actress
606646	nm9991786	Michelle Modigliani	NaN	NaN	producer
606647	nm9993380	Pegasus Envoyé	NaN	NaN	director,actor,writer

606648 rows × 5 columns

CHECKING FOR DUPLICATES

In [107…
```
imdb_persons.duplicated().sum()
```

Out[107…
```
0
```

DROPPING UNNECESSARY COLUMNS

In [108…
```
# drop unnecessary columns. Only interested in name,  profession
imdb_persons = imdb_persons.drop(['birth_year',
'death_year'],
axis=1)
```

In [109…
```
imdb_persons
```

Out[109…

	person_id	primary_name	primary_profession
0	nm0061671	Mary Ellen Bauder	miscellaneous,production_manager,producer
1	nm0061865	Joseph Bauer	composer,music_department,sound_department
2	nm0062070	Bruce Baum	miscellaneous,actor,writer
3	nm0062195	Axel Baumann	camera_department,cinematographer,art_department
4	nm0062798	Pete Baxter	production_designer,art_department,set_decorator
...	...	...	...
606643	nm9990381	Susan Grobes	actress
606644	nm9990690	Joo Yeon So	actress
606645	nm9991320	Madeline Smith	actress
606646	nm9991786	Michelle Modigliani	producer
606647	nm9993380	Pegasus Envoyé	director,actor,writer

606648 rows × 3 columns

SPLITTING COLUMNS

In [110…

```
imdb_persons_expand = imdb_persons.primary_profession.str.split(",", expand=True)
```

In [111…

```
imdb_persons_expand
```

Out[111…

	0	1	2
0	miscellaneous	production_manager	producer
1	composer	music_department	sound_department
2	miscellaneous	actor	writer
3	camera_department	cinematographer	art_department
4	production_designer	art_department	set_decorator

	0	1	2
...	...	...	...
606643	actress	None	None
606644	actress	None	None
606645	actress	None	None
606646	producer	None	None
606647	director	actor	writer

606648 rows × 3 columns

In [112...
```
imdb_persons['profession_1']=imdb_persons_expand[0]
imdb_persons['profession_2']=imdb_persons_expand[1]
imdb_persons['profession_3']=imdb_persons_expand[2]
#imdb_title_basics.drop('genres', axis=1, inplace=True)
```

In [113...
```
imdb_persons.set_index('person_id', inplace=True)
```

NEW SEPARATED COLUMNS ADDED

In [114...
```
imdb_persons
```

Out[114...

person_id	primary_name	primary_profession	profession_1	profession_2	profession_3
nm0061671	Mary Ellen Bauder	miscellaneous,production_manager,producer	miscellaneous	production_manager	producer
nm0061865	Joseph Bauer	composer,music_department,sound_department	composer	music_department	sound_department
nm0062070	Bruce Baum	miscellaneous,actor,writer	miscellaneous	actor	writer
nm0062195	Axel Baumann	camera_department,cinematographer,art_department	camera_department	cinematographer	art_department
nm0062798	Pete Baxter	production_designer,art_department,set_decorator	production_designer	art_department	set_decorator
...	...	...	...	...	...

person_id	primary_name	primary_profession	profession_1	profession_2	profession_3
nm9990381	Susan Grobes	actress	actress	None	None
nm9990690	Joo Yeon So	actress	actress	None	None
nm9991320	Madeline Smith	actress	actress	None	None
nm9991786	Michelle Modigliani	producer	producer	None	None
nm9993380	Pegasus Envoyé	director,actor,writer	director	actor	writer

606648 rows × 5 columns

As you can see, I split the 'primary_profession' column into three columns:

- profession_1
- profession_2
- profession_3

DATATYPES

In [115…

```
imdb_persons.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 606648 entries, nm0061671 to nm9993380
Data columns (total 5 columns):
 #   Column              Non-Null Count   Dtype 
---  ------              --------------   ----- 
 0   primary_name        606648 non-null  object
 1   primary_profession  555308 non-null  object
 2   profession_1        555308 non-null  object
 3   profession_2        313677 non-null  object
 4   profession_3        220006 non-null  object
dtypes: object(5)
memory usage: 27.8+ MB
```

LISTING ALL THE COLUMNS OF CLEANED TABLES

In [116…

```
print()
print("IMDB CLEANED TABLES AND CLEANED NEW COLUMNS:")
print()
print("IMDB MOVIE BASICS:", imdb_movie_basics.columns)
print("IMDB MOVIE RATINGS:", imdb_movie_ratings.columns)
print("IMDB MOVIE AKAS:", imdb_movie_akas.columns)
print("IMDB DIRECTOR:", imdb_directors.columns)
print("IMDB WRITERS:", imdb_writers.columns)
print("IMDB PRINCIPALS:", imdb_principals.columns)
print("IMDB KNOWN-FOR:", imdb_known_for.columns)
print("IMDB PERSONS:", imdb_persons.columns)
print()
```

```
IMDB CLEANED TABLES AND CLEANED NEW COLUMNS:

IMDB MOVIE BASICS: Index(['title', 'start_year', 'runtime_minutes', 'genres', 'genres_1',
       'genres_2', 'genres_3'],
      dtype='object')
IMDB MOVIE RATINGS: Index(['averagerating', 'numvotes'], dtype='object')
IMDB MOVIE AKAS: Index(['movie_id', 'ordering', 'title'], dtype='object')
IMDB DIRECTOR: Index(['movie_id', 'person_id'], dtype='object')
IMDB WRITERS: Index(['movie_id', 'person_id'], dtype='object')
IMDB PRINCIPALS: Index(['movie_id', 'ordering', 'person_id', 'profession_1', 'characters'], dtype='object')
IMDB KNOWN-FOR: Index(['person_id', 'movie_id'], dtype='object')
IMDB PERSONS: Index(['primary_name', 'primary_profession', 'profession_1', 'profession_2',
       'profession_3'],
      dtype='object')
```

In [117…

```
imdb_movie_basics
```

Out[117…

movie_id	title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3
tt0063540	Sunghursh	2013	175.0	Action,Crime,Drama	Action	Crime	Drama
tt0066787	One Day Before the Rainy Season	2019	114.0	Biography,Drama	Biography	Drama	None
tt0069049	The Other Side of the Wind	2018	122.0	Drama	Drama	None	None
tt0069204	Sabse Bada Sukh	2018	NaN	Comedy,Drama	Comedy	Drama	None
tt0100275	The Wandering Soap Opera	2017	80.0	Comedy,Drama,Fantasy	Comedy	Drama	Fantasy

movie_id	title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3
...	...	...	...	...	...	...	...
tt9916538	Kuambil Lagi Hatiku	2019	123.0	Drama	Drama	None	None
tt9916622	Rodolpho Teóphilo - O Legado de um Pioneiro	2015	NaN	Documentary	Documentary	None	None
tt9916706	Dankyavar Danka	2013	NaN	Comedy	Comedy	None	None
tt9916730	6 Gunn	2017	116.0	None	None	None	None
tt9916754	Chico Albuquerque - Revelações	2013	NaN	Documentary	Documentary	None	None

146022 rows × 7 columns

ANALYSIS ON MOVIE GENRES

JOINING TABLES via SQL: MOVIE BASICS WITH MOVIE RATINGS|

In [118...

```
q=("""select * from imdb_movie_basics
join imdb_movie_ratings
using (movie_id)
""")
joined_movies_with_ratings=mysql(q)
joined_movies_with_ratings
```

Out[118...

	movie_id	title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3	averagerating	numvot
0	tt10356526	Laiye Je Yaarian	2019	117.0	Romance	Romance	None	None	8.3	3
1	tt10384606	Borderless	2019	87.0	Documentary	Documentary	None	None	8.9	5
2	tt1042974	Just Inès	2010	90.0	Drama	Drama	None	None	6.4	2
3	tt1043726	The Legend of Hercules	2014	99.0	Action,Adventure,Fantasy	Action	Adventure	Fantasy	4.2	503
4	tt1060240	Até Onde?	2011	73.0	Mystery,Thriller	Mystery	Thriller	None	6.5	2
...	...	...	...	...	...	...	...	...	...	

	movie_id	title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3	averagerating	numvot
73846	tt9805820	Caisa	2018	84.0	Documentary	Documentary	None	None	8.1	
73847	tt9844256	Code Geass: Lelouch of the Rebellion - Glorifi...	2018	120.0	Action,Animation,Sci-Fi	Action	Animation	Sci-Fi	7.5	
73848	tt9851050	Sisters	2019	NaN	Action,Drama	Action	Drama	None	4.7	
73849	tt9886934	The Projectionist	2019	81.0	Documentary	Documentary	None	None	7.0	
73850	tt9894098	Sathru	2019	129.0	Thriller	Thriller	None	None	6.3	1

73851 rows × 10 columns

TOP 5 MOVIE GENRES

With Highest Rating

In [119…

```
q=("""select genres_1 as genre,count(genres_1) as count,averagerating,numvotes from joined_movies_with_ratings group by g
order by averagerating desc limit 5
""")
bygenre_highestrating=mysql(q)
bygenre_highestrating
```

Out[119…

	genre	count	averagerating	numvotes
0	Musical	153	9.3	50
1	Game-Show	1	9.0	7
2	Documentary	13960	8.9	559
3	Sport	89	8.3	13
4	Romance	786	8.3	31

With Lowest Rating

In [120…

```
q=("""select genres_1 as genre,count(genres_1) as count,averagerating,numvotes from joined_movies_with_ratings group by g
order by averagerating asc limit 5
""")
bygenre_lowestgerating=mysql(q)
bygenre_lowestgerating
```

Out[120…

	genre	count	averagerating	numvotes
0	Adult	1	2.0	128
1	Family	604	3.4	212
2	Horror	4489	3.4	387
3	Reality-TV	5	3.4	38
4	Animation	962	3.8	9

RUNTIME MINUTES

With Perfect Rating

In [121…

```
q=("""select runtime_minutes,* from joined_movies_with_ratings
where averagerating='10.0'
order by runtime_minutes desc limit 10
""")
byruntime_perfectrating=mysql(q)
byruntime_perfectrating
```

Out[121…

	runtime_minutes	movie_id	title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3	averageratin
0	129.0	tt10378660	The Dark Knight: The Ballad of the N Word	2018	129.0	Comedy,Drama	Comedy	Drama	None	10
1	100.0	tt1770682	Freeing Bernie Baran	2010	100.0	Crime,Documentary	Crime	Documentary	None	10

	runtime_minutes	movie_id	title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3	averageratin
2	99.0	tt8730716	Pick It Up! - Ska in the '90s	2019	99.0	Documentary	Documentary	None	None	10
3	93.0	tt6991826	A Dedicated Life: Phoebe Brand Beyond the Group	2015	93.0	Documentary	Documentary	None	None	10
4	77.0	tt7259300	Calamity Kevin	2019	77.0	Adventure,Comedy	Adventure	Comedy	None	10
5	72.0	tt2632430	Hercule contre Hermès	2012	72.0	Documentary	Documentary	None	None	10
6	70.0	tt4960818	Revolution Food	2015	70.0	Documentary	Documentary	None	None	10
7	70.0	tt7227500	Ellis Island: The Making of a Master Race in A...	2018	70.0	Documentary,History	Documentary	History	None	10
8	65.0	tt5089804	Fly High: Story of the Disc Dog	2019	65.0	Documentary	Documentary	None	None	10
9	59.0	tt5390098	The Paternal Bond: Barbary Macaques	2015	59.0	Documentary	Documentary	None	None	10

With Lowest Rating

In [122…

```
q=("""select runtime_minutes,* from joined_movies_with_ratings
```

```
where averagerating='10.0' and runtime_minutes not null
order by runtime_minutes asc limit 10
""")
byruntime_lowestrating=mysql(q)
byruntime_lowestrating
```

Out[122...

	runtime_minutes	movie_id	title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3	averagerating
0	31.0	tt4109192	I Was Born Yesterday!	2015	31.0	Documentary	Documentary	None	None	10.0
1	48.0	tt6295832	Requiem voor een Boom	2016	48.0	Documentary	Documentary	None	None	10.0
2	52.0	tt10176328	Exteriores: Mulheres Brasileiras na Diplomacia	2018	52.0	Documentary	Documentary	None	None	10.0
3	59.0	tt5390098	The Paternal Bond: Barbary Macaques	2015	59.0	Documentary	Documentary	None	None	10.0
4	65.0	tt5089804	Fly High: Story of the Disc Dog	2019	65.0	Documentary	Documentary	None	None	10.0
5	70.0	tt4960818	Revolution Food	2015	70.0	Documentary	Documentary	None	None	10.0
6	70.0	tt7227500	Ellis Island: The Making of a Master Race in A...	2018	70.0	Documentary,History	Documentary	History	None	10.0
7	72.0	tt2632430	Hercule contre Hermès	2012	72.0	Documentary	Documentary	None	None	10.0
8	77.0	tt7259300	Calamity Kevin	2019	77.0	Adventure,Comedy	Adventure	Comedy	None	10.0

	runtime_minutes	movie_id	title	start_year	runtime_minutes	genres	genres_1	genres_2	genres_3	averagerating
9	93.0	tt6991826	A Dedicated Life: Phoebe Brand Beyond the Group	2015	93.0	Documentary	Documentary	None	None	10.0

In [123…

```
bygenre_highestrating
```

Out[123…

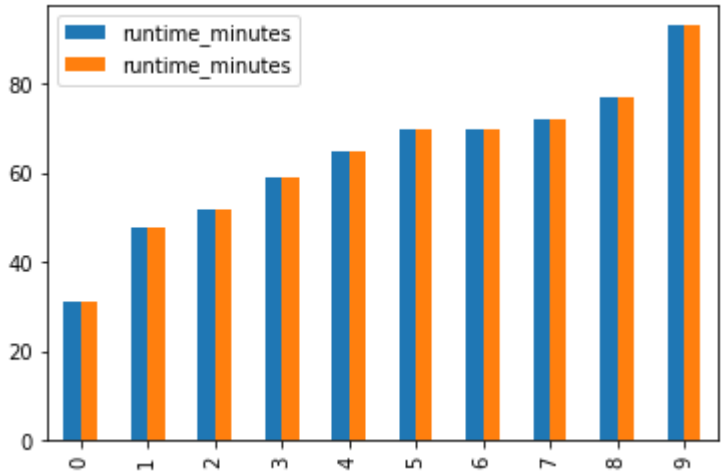
	genre	count	averagerating	numvotes
0	Musical	153	9.3	50
1	Game-Show	1	9.0	7
2	Documentary	13960	8.9	559
3	Sport	89	8.3	13
4	Romance	786	8.3	31

In [124…

```
byruntime_lowestrating.plot(kind='bar', y='runtime_minutes')
```

Out[124…

```
<AxesSubplot:>
```

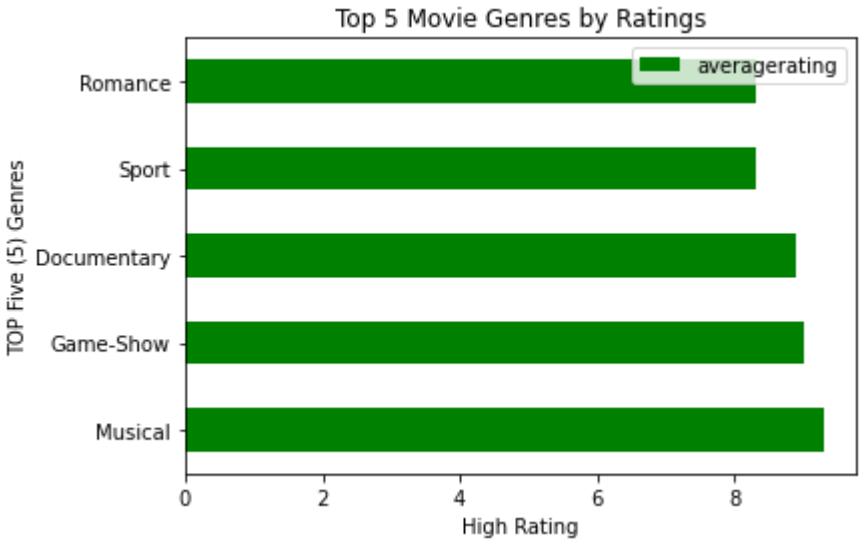


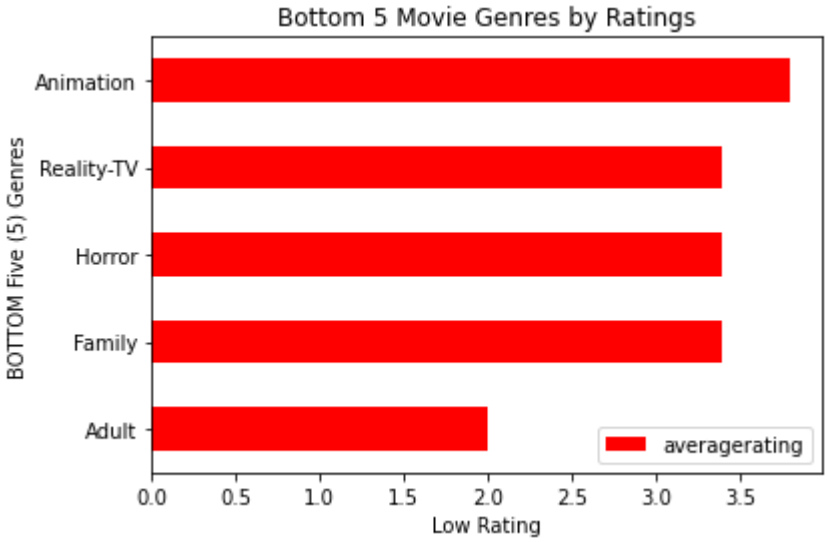
VISUALIZATIONS

Genre by Ratings

In [125…

```
# Plot Movie Genre and Average Rating

bygenre_highestrating.plot(kind = 'barh',
        x = 'genre',
        y = 'averagerating',
        color = 'green')
# Title
plt.title('Top 5 Movie Genres by Ratings')
plt.xlabel("High Rating")
plt.ylabel("TOP Five (5) Genres")
# Plot Movie Genre and Average Rating
bygenre_lowestgerating.plot(kind = 'barh',
        x = 'genre',
        y = 'averagerating',
        color = 'red')
# Title
plt.title('Bottom 5 Movie Genres by Ratings')
plt.xlabel("Low Rating")
plt.ylabel("BOTTOM Five (5) Genres")
# Display
plt.show()
```



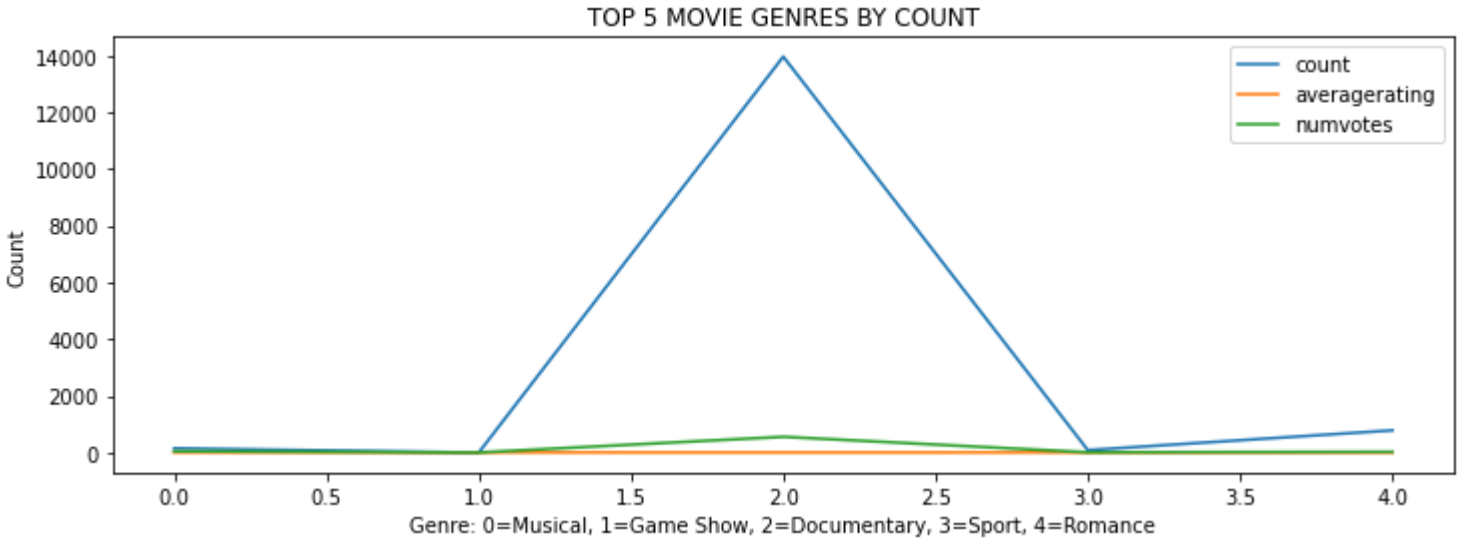


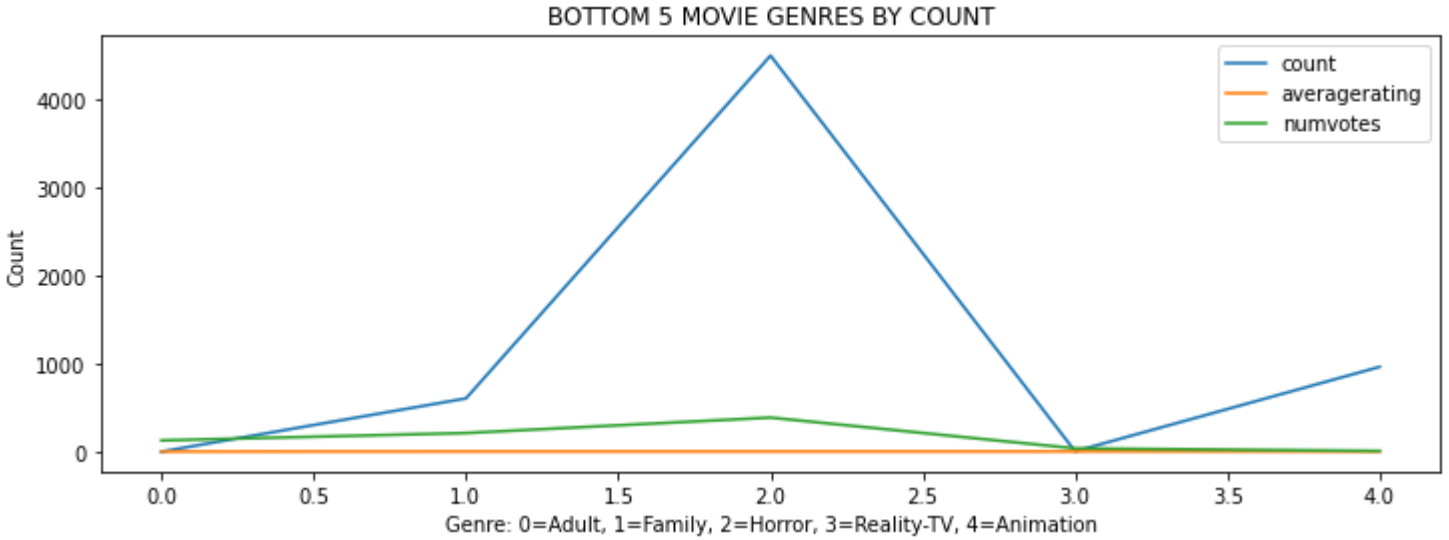
Genre by Count

In [126…

```
fig, axs = plt.subplots(figsize=(12, 4))
bygenre_highestrating.plot.line(ax=axs)
axs.set_ylabel("Count")
axs.set_xlabel("Genre: 0=Musical, 1=Game Show, 2=Documentary, 3=Sport, 4=Romance")
axs.set_title('TOP 5 MOVIE GENRES BY COUNT');
```

```
fig, axs = plt.subplots(figsize=(12, 4))
bygenre_lowestgerating.plot.line(ax=axs)
axs.set_ylabel("Count")
axs.set_xlabel("Genre: 0=Adult, 1=Family, 2=Horror, 3=Reality-TV, 4=Animation")
axs.set_title('BOTTOM 5 MOVIE GENRES BY COUNT');
```

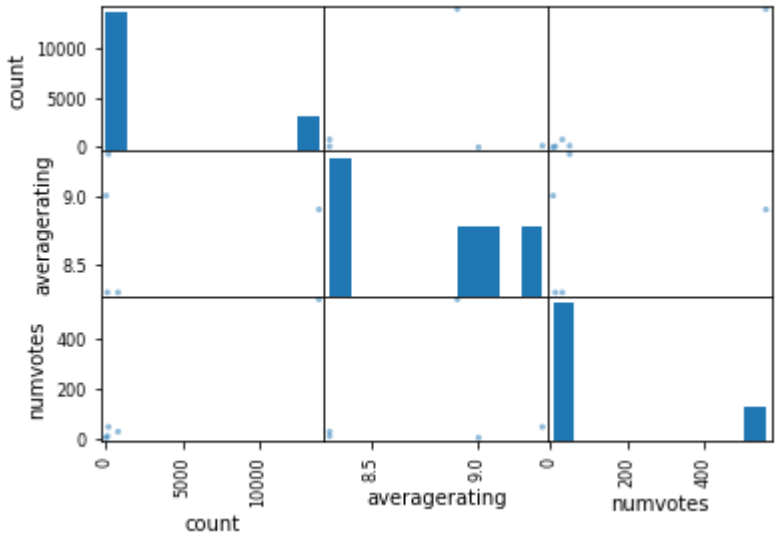


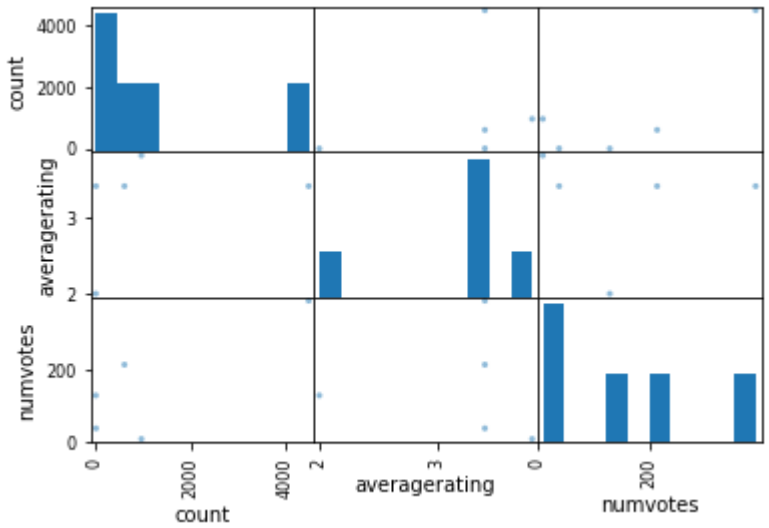


Matrix of Plots

In [127…

```
from pandas.plotting import scatter_matrix
scatter_matrix(bygenre_highestrating);
scatter_matrix(bygenre_lowestgerating);
```





RECOMMENDATION ANALYSIS #2:

Based on the data charts, these five (5) movie genres received at least 8.0 rating (between 8.3 to 9.3). My recommendation is that Microsoft should be producing these type of movies in order to be highly profitable:

- Musical
- Game Show
- Documentary
- Sport
- Romance

I do not recommend the following genres as they received the lowest ratings (between 2.0 to 3.8):

- Adult
- Family
- Horror
- Reality-TV
- Animation

JOINING TABLES via SQL

JOIN - RATINGS BY DIRECTORS

In [129…

```
q=("""select * from imdb_directors
join imdb_movie_ratings
using (movie_id)
join imdb_persons
using (person_id)
join imdb_known_for
using (person_id)
join imdb_movie_basics
using (movie_id)
""")
directors_ratings=mysql(q)
directors_ratings
```

Out[129…

	movie_id	person_id	averagerating	numvotes	primary_name	primary_profession	profession_1	profession_2
0	tt0285252	nm0899854	3.9	219	Tony Vitale	producer,director,writer	producer	director
1	tt0285252	nm0899854	3.9	219	Tony Vitale	producer,director,writer	producer	director
2	tt0285252	nm0899854	3.9	219	Tony Vitale	producer,director,writer	producer	director

	movie_id	person_id	averagerating	numvotes	primary_name	primary_profession	profession_1	profession_2
3	tt0285252	nm0899854	3.9	219	Tony Vitale	producer,director,writer	producer	director
4	tt0462036	nm1940585	5.5	18	Bill Haley	director,writer,producer	director	writer
...	...	...	...	...	...	...	...	...
305755	tt8991416	nm7731173	6.7	13	Richard Squires	director,animation_department,editor	director	animation_department
305756	tt9004986	nm4993825	8.4	7	Fredrik Horn Akselsen	director,writer,assistant_director	director	writer
305757	tt9004986	nm4993825	8.4	7	Fredrik Horn Akselsen	director,writer,assistant_director	director	writer
305758	tt9004986	nm4993825	8.4	7	Fredrik Horn Akselsen	director,writer,assistant_director	director	writer
305759	tt9004986	nm4993825	8.4	7	Fredrik Horn Akselsen	director,writer,assistant_director	director	writer

305760 rows × 17 columns

In [130…
```
directors_ratings.duplicated().sum()
```

Out[130…
```
0
```

TOP 5 RATED DIRECTORS

In [131…
```
q=("""select distinct person_id,primary_name,genres_1,averagerating,numvotes from directors_ratings
where averagerating='10.0'
```

```
order by numvotes asc limit 5
""")
top_perfectrated_directors=mysql(q)
top_perfectrated_directors
```

Out[131...

	person_id	primary_name	genres_1	averagerating	numvotes
0	nm3704168	Michiel Brongers	Documentary	10.0	5
1	nm4166962	Daniel Alexander	Crime	10.0	5
2	nm4637768	Masahiro Hayakawa	Documentary	10.0	5
3	nm5472684	Michael J. Sanderson	Documentary	10.0	5
4	nm4568586	Taylor Morden	Documentary	10.0	5

JOIN - RATINGS BY WRITERS

In [138...

```
q=("""select * from imdb_writers
join imdb_movie_ratings
using (movie_id)
join imdb_persons
using (person_id)
join imdb_known_for
using (person_id)
join imdb_movie_basics
using (movie_id)
order by averagerating desc
""")
writers_ratings=mysql(q)
writers_ratings
```

Out[138...

	movie_id	person_id	averagerating	numvotes	primary_name	primary_profession	profession_1	profess
0	tt1770682	nm4166961	10.0	5	Daniel Alexander	writer,editor,producer	writer	
1	tt4960818	nm6680574	10.0	8	Brian Baucum	camera_department,writer,editorial_department	camera_department	
2	tt4960818	nm6680574	10.0	8	Brian Baucum	camera_department,writer,editorial_department	camera_department	

	movie_id	person_id	averagerating	numvotes	primary_name	primary_profession	profession_1	profess
3	tt4960818	nm6680574	10.0	8	Brian Baucum	camera_department,writer,editorial_department	camera_department	
4	tt4960818	nm6680574	10.0	8	Brian Baucum	camera_department,writer,editorial_department	camera_department	
...	...	...	...	...	...	...	...	
385107	tt6792126	nm6008960	1.0	5	Eva Toulová	director,writer,producer	director	
385108	tt8313262	nm8400242	1.0	20	Nikita Derney	actor,editor,writer	actor	
385109	tt8313262	nm8400242	1.0	20	Nikita Derney	actor,editor,writer	actor	
385110	tt8313262	nm8400242	1.0	20	Nikita Derney	actor,editor,writer	actor	
385111	tt8313262	nm8400242	1.0	20	Nikita Derney	actor,editor,writer	actor	

385112 rows × 17 columns

TOP 5 RATED WRITERS

In [140…

```
q=("""select distinct person_id,primary_name,genres_1,averagerating,numvotes from writers_ratings
where averagerating='10.0'
order by numvotes asc limit 5
""")
top_5_perfectedwriters=mysql(q)
top_5_perfectedwriters
```

Out[140…

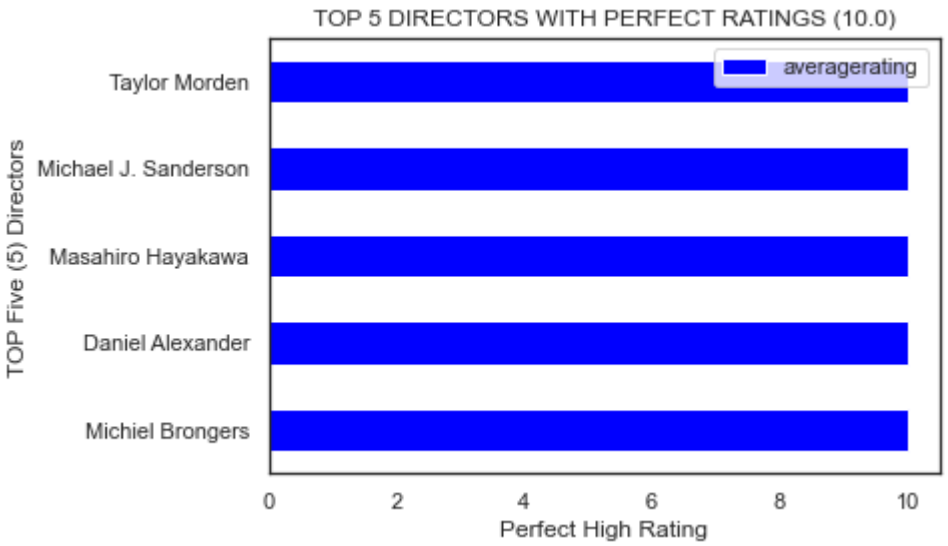
person_id	primary_name	genres_1	averagerating	numvotes

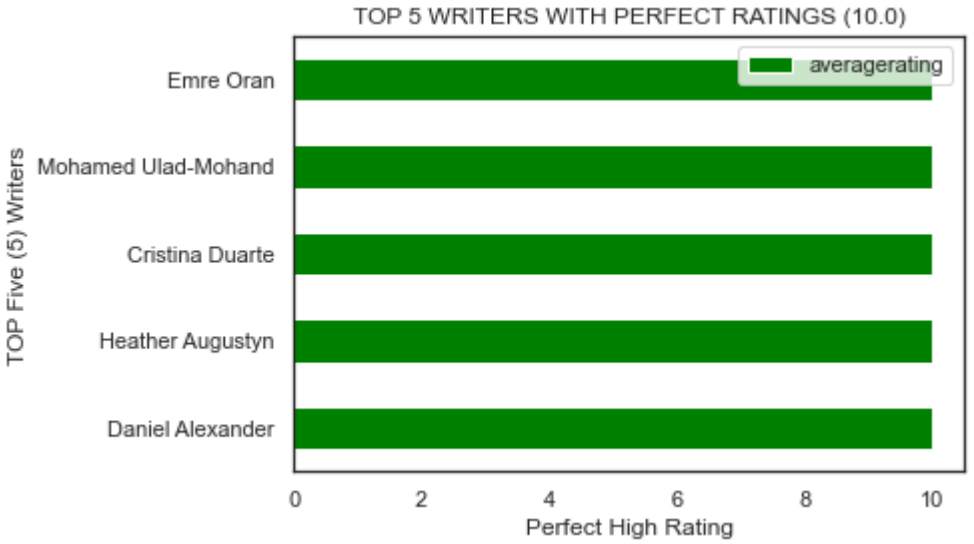
	person_id	primary_name	genres_1	averagerating	numvotes
0	nm4166961	Daniel Alexander	Crime	10.0	5
1	nm9500109	Heather Augustyn	Documentary	10.0	5
2	nm2751775	Cristina Duarte	Documentary	10.0	5
3	nm0880350	Mohamed Ulad-Mohand	Documentary	10.0	5
4	nm8791543	Emre Oran	Adventure	10.0	6

VISUALIZATIONS

In [164…

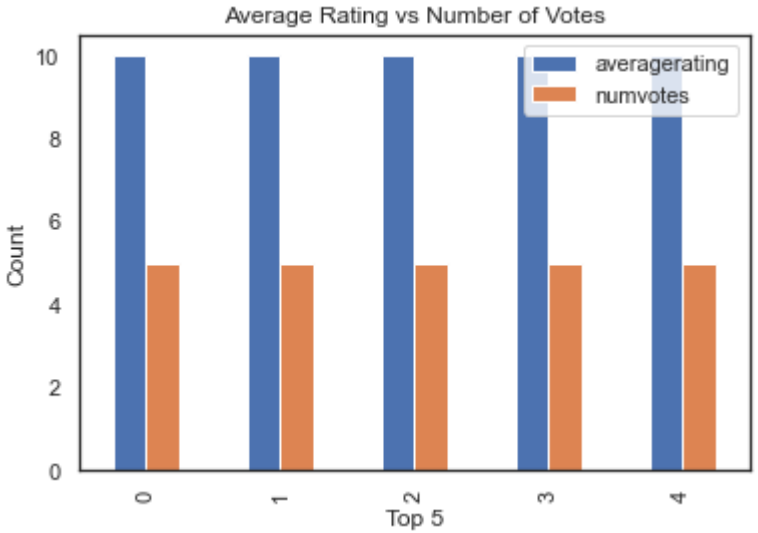
```
# Plot Movie Genre and Average Rating

top_perfectrated_directors.plot(kind = 'barh',
        x = 'primary_name',
        y = 'averagerating',
        color = 'blue')
# Title
plt.title('TOP 5 DIRECTORS WITH PERFECT RATINGS (10.0)')
plt.xlabel("Perfect High Rating")
plt.ylabel("TOP Five (5) Directors")
# Plot Movie Genre and Average Rating
top_5_perfectedwriters.plot(kind = 'barh',
        x = 'primary_name',
        y = 'averagerating',
        color = 'green')
# Title
plt.title('TOP 5 WRITERS WITH PERFECT RATINGS (10.0)')
plt.xlabel("Perfect High Rating")
plt.ylabel("TOP Five (5) Writers")
# Display
plt.show()
```





In [168…

```
top_perfectrated_directors.plot(kind='bar',xlabel='Top 5', ylabel='Count', title='Average Rating vs Number of Votes');
```



RECOMMENDATION ANALYSIS #3:

Based on the data charts, My recommendation is that Microsoft should be producing movies with these top rated directors and top rated writers in order to be highly profitable.

Directors:

- Michiel Brongers
- Daniel Alexander
- Masahiro Hayakawa
- Michael J. Sanderson
- Taylor Morden

Writers:

- Daniel Alexander
- Heather Augusty
- Cristina Duarte
- Mohamed Ulad-Mohand
- Emre Oran

The following attributes do not guarantee high ratings (profitable)

- High popularity (votes)
- Time Duration (runningminutes)