Microsoft Business Analysis

- Student name: Kimley Kadoche
- Student Pace: Self pace
- Scheduled project review date/time: 05/09/22 at 9:30am EST
- Instructor name: Abhineet Kulkarni
- Blog post URL: Medium Blog post (https://medium.com/@kadoche.k/dont-hesitate-just-go-for-it-and-things-will-fall-into-place-76189f7f7db4)

Business Problem

Microsoft sees its competitors growing their businesses in the movie industry and they want to partake. In order to start producing movies, Microsoft needs a clear understanding of the market and get a key entry point.

Let's explore some key data in order to give Microsoft's stakeholder an accurate understanding of the movie industry and how Microsoft can become relevant in the movie industry.

Data Source and Use

In order to conduct my analysis, I used the following documents:

- TN movies budget/domestic gross returns and international gross returns.
- the gross revenue per blockbuster movie, both domestically and internationally,
- the Rating of the blockbusters (R, PG-13...)
- the Rotten Tomatoes reviews
- Tmdb movies ratings

1. Data Cleaning for the Total Revenue compared to Budgets by Month of Release for Movies.

In [1]:
```
#Let's start by importing the packages needed to perform analysis

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

%matplotlib inline
```

In [2]:
```
#Let's have a look at the material we have for our analysis.
#The first document is a list of the gross returns per blockbusters

df=pd.read_csv('Documents/Flatiron/Phase_1/dsc-phase-1-project-v2-4/Data/tn.movie_bu
df
```

Out[2]:

	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

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

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

In [4]:
```
#We don't need the 'id' column so we might as well just delete it.

df = df.drop('id', axis=1)
df
```

Out[4]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
3	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
4	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747
...	...	...	...	...	...
5777	Dec 31, 2018	Red 11	$7,000	$0	$0
5778	Apr 2, 1999	Following	$6,000	$48,482	$240,495
5779	Jul 13, 2005	Return to the Land of Wonders	$5,000	$1,338	$1,338
5780	Sep 29, 2015	A Plague So Pleasant	$1,400	$0	$0
5781	Aug 5, 2005	My Date With Drew	$1,100	$181,041	$181,041

5782 rows × 5 columns

In [5]:
```
#Change of the 'release_date' column data type to date type.

df["release_date"] = pd.to_datetime(df['release_date'])
df.head()
```

Out[5]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	2009-12-18	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	2011-05-20	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	2019-06-07	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
3	2015-05-01	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
4	2017-12-15	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747

Several steps to go through with that file:

- convert the last 3 rows in int64
- delete rows with a production budget less than 5 million
- add the columns dom + world
- classify the file by release date (by month)
- do the graph that shows the total gross revenue per month and the graph that shows the ratio budget / revenue

```
In [6]: #Convert the last 3 rows in integer

        df['production_budget'] = df['production_budget'].str.replace('$','').str.replace(',
        df['production_budget'] = pd.to_numeric(df['production_budget'])
        df['production_budget']
```

```
Out[6]: 0       425000000
        1       410600000
        2       350000000
        3       330600000
        4       317000000
                  ...
        5777         7000
        5778         6000
        5779         5000
        5780         1400
        5781         1100
        Name: production_budget, Length: 5782, dtype: int64
```

```
In [7]: df['domestic_gross'] = df['domestic_gross'].str.replace('$','').str.replace(',','')
        df['domestic_gross'] = pd.to_numeric(df['domestic_gross'])
        df['domestic_gross']
```

```
Out[7]: 0       760507625
        1       241063875
        2        42762350
        3       459005868
        4       620181382
                  ...
        5777            0
        5778        48482
        5779         1338
        5780            0
        5781       181041
        Name: domestic_gross, Length: 5782, dtype: int64
```

```
In [8]: df['worldwide_gross'] = df['worldwide_gross'].str.replace('$','').str.replace(',',''
        df['worldwide_gross'] = pd.to_numeric(df['worldwide_gross'])
        df['worldwide_gross']
```

```
Out[8]: 0       2776345279
        1       1045663875
        2        149762350
        3       1403013963
        4       1316721747
                   ...
        5777             0
        5778        240495
        5779          1338
        5780             0
        5781        181041
        Name: worldwide_gross, Length: 5782, dtype: int64
```

```
In [9]: #Checking results

        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   release_date       5782 non-null   datetime64[ns]
 1   movie              5782 non-null   object
 2   production_budget  5782 non-null   int64
 3   domestic_gross     5782 non-null   int64
 4   worldwide_gross    5782 non-null   int64
dtypes: datetime64[ns](1), int64(3), object(1)
memory usage: 226.0+ KB
```

```
In [10]: df = df.dropna()
         df.isna().sum()
```

```
Out[10]: release_date         0
         movie                0
         production_budget    0
         domestic_gross       0
         worldwide_gross      0
         dtype: int64
```

```
In [11]: #Checking results.

         df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5782 entries, 0 to 5781
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   release_date       5782 non-null   datetime64[ns]
 1   movie              5782 non-null   object
 2   production_budget  5782 non-null   int64
 3   domestic_gross     5782 non-null   int64
 4   worldwide_gross    5782 non-null   int64
dtypes: datetime64[ns](1), int64(3), object(1)
memory usage: 271.0+ KB
```

In [12]:
```
#Creating a 'total gross revenue' column.

df['total_gross_revenue'] = df['domestic_gross'] + df['worldwide_gross']
df.head()
```

Out[12]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	total_gross_revenue
0	2009-12-18	Avatar	425000000	760507625	2776345279	3536852904
1	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	1286727750
2	2019-06-07	Dark Phoenix	350000000	42762350	149762350	192524700
3	2015-05-01	Avengers: Age of Ultron	330600000	459005868	1403013963	1862019831
4	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	1936903129

In [13]:
```
#Filtering out the revenues equal to 0.

mask = df['total_gross_revenue'] > 0
df[mask].head()
```

Out[13]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	total_gross_revenue
0	2009-12-18	Avatar	425000000	760507625	2776345279	3536852904
1	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	1286727750
2	2019-06-07	Dark Phoenix	350000000	42762350	149762350	192524700
3	2015-05-01	Avengers: Age of Ultron	330600000	459005868	1403013963	1862019831
4	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	1936903129

In [14]:
```
#Filtering out the revenues inferior to $5M.

show = df['total_gross_revenue'] > 5000000
df[show].head()
```

Out[14]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	total_gross_revenue
0	2009-12-18	Avatar	425000000	760507625	2776345279	3536852904
1	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	1286727750
2	2019-06-07	Dark Phoenix	350000000	42762350	149762350	192524700
3	2015-05-01	Avengers: Age of Ultron	330600000	459005868	1403013963	1862019831
4	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	1936903129

Microsoft's decision to enter the movie industry is a great opportunity to add a very promising revenue stream to their business expansion strategy. Considering the size of Microsoft, it makes more sense to look at data that can be of real use for it: a total gross revenue of minimum $5 million is the minimum it should target, at least to start.

In [15]:
```
print(df['production_budget'].apply(['mean', 'median', 'std']))
```

```
mean      3.158776e+07
median    1.700000e+07
std       4.181208e+07
Name: production_budget, dtype: float64
```

In [16]:
```
print(df['total_gross_revenue'].apply(['mean', 'median', 'std']))
```

```
mean      1.333608e+08
median    4.605855e+07
std       2.399411e+08
Name: total_gross_revenue, dtype: float64
```

In [17]:
```
#Filtering data in the release date column to only keep the month of release

df['month_of_release'] = df['release_date'].dt.month_name()
```

In [18]:
```
df['month_of_release']
```

Out[18]:
```
0        December
1             May
2            June
3             May
4        December
           ...
5777     December
5778        April
5779         July
5780    September
5781       August
Name: month_of_release, Length: 5782, dtype: object
```

```
In [19]: df.head()
```

Out[19]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	total_gross_revenue	month_o
0	2009-12-18	Avatar	425000000	760507625	2776345279	3536852904	D
1	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	1286727750	
2	2019-06-07	Dark Phoenix	350000000	42762350	149762350	192524700	
3	2015-05-01	Avengers: Age of Ultron	330600000	459005868	1403013963	1862019831	
4	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	1936903129	D

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

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

```
In [21]: df_month_of_release = df[['month_of_release', 'total_gross_revenue']].reset_index()
         df_month_of_release.head()
```

Out[21]:

	index	month_of_release	total_gross_revenue
0	0	December	3536852904
1	1	May	1286727750
2	2	June	192524700
3	3	May	1862019831
4	4	December	1936903129

In [22]:
```
#Filtering the file to only keep the month of release and the total gross revenue co

df_month_of_release = df_month_of_release.drop('index', axis=1)
df_month_of_release.head()
```

Out[22]:

	month_of_release	total_gross_revenue
0	December	3536852904
1	May	1286727750
2	June	192524700
3	May	1862019831
4	December	1936903129

In [23]:
```
#Summing total gross revenue per month

df_months_revenue = df.groupby('month_of_release')['total_gross_revenue'].sum()
print(df_months_revenue)
```

```
month_of_release
April           39610890322
August          46200721750
December       110106520078
February        41927998047
January         24468164278
July            88744327892
June            99800102633
March           56026987828
May             93189142692
November        94246480664
October         42337856674
September       34432882048
Name: total_gross_revenue, dtype: int64
```

In [24]:
```
#Dropping index column.

df_budget_revenue = df[['production_budget', 'total_gross_revenue', 'month_of_releas
df_budget_revenue = df_budget_revenue.drop('index', axis=1)
df_budget_revenue.head()
```

Out[24]:

	production_budget	total_gross_revenue	month_of_release
0	425000000	3536852904	December
1	410600000	1286727750	May
2	350000000	192524700	June
3	330600000	1862019831	May
4	317000000	1936903129	December

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   production_budget    5782 non-null   int64 
 1   total_gross_revenue  5782 non-null   int64 
 2   month_of_release     5782 non-null   object
dtypes: int64(2), object(1)
memory usage: 135.6+ KB
```

```
In [26]: #Filtering data by only showing total gross revenue superior to $5,000,000.

         show_2 = df_budget_revenue['total_gross_revenue'] > 5000000
         df_budget_revenue[show_2]
```

Out[26]:

	production_budget	total_gross_revenue	month_of_release
0	425000000	3536852904	December
1	410600000	1286727750	May
2	350000000	192524700	June
3	330600000	1862019831	May
4	317000000	1936903129	December
...	...	...	...
5709	65000	33763176	May
5715	50000	20853012	August
5742	27000	6967668	October
5745	25000	90000000	June
5746	25000	5767322	August

4438 rows × 3 columns

```
In [27]: df_budget_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   production_budget    5782 non-null   int64 
 1   total_gross_revenue  5782 non-null   int64 
 2   month_of_release     5782 non-null   object
dtypes: int64(2), object(1)
memory usage: 135.6+ KB
```

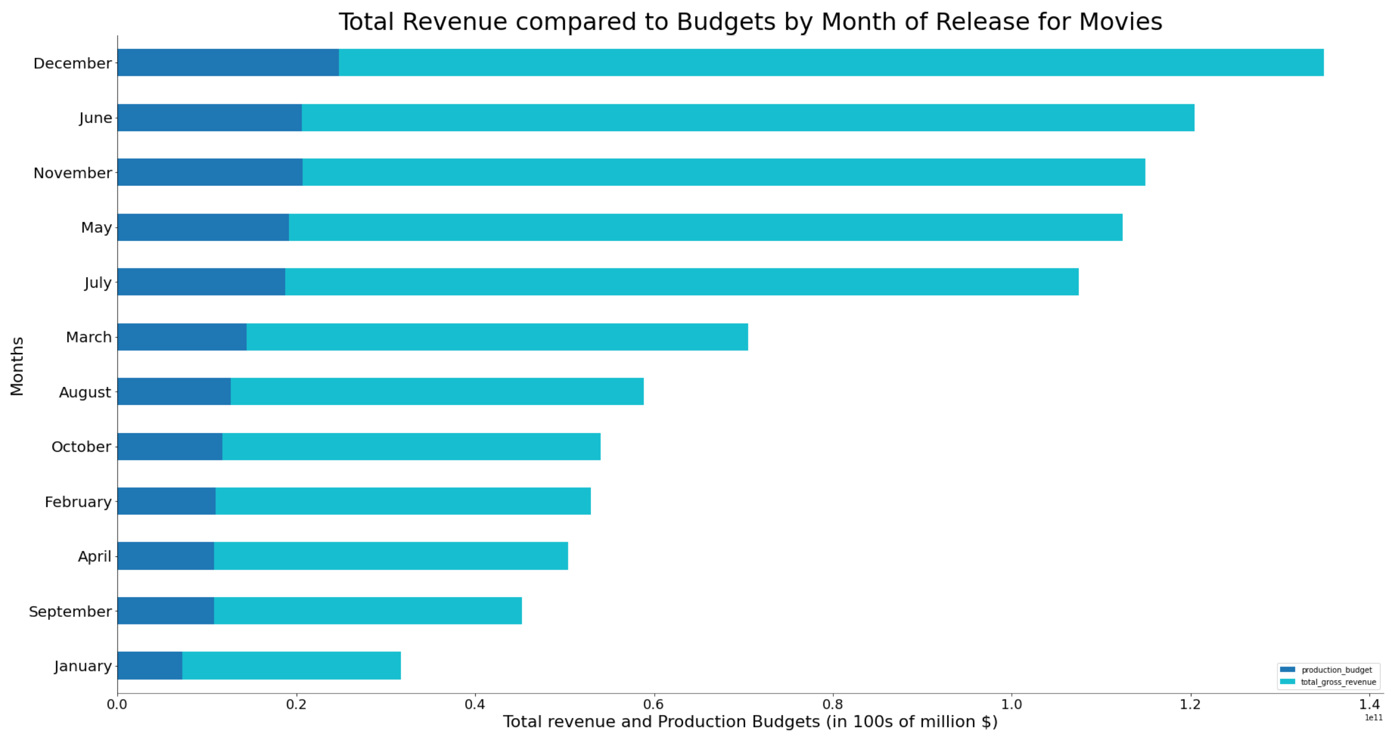
Let's take the average of production budget per month and then the average ratio budget/revenue.

```
In [28]: #Groupby data by month of release.

         df_months_budget = df.groupby('month_of_release')['production_budget', 'total_gross_
         print(df_months_budget)
```

```
                    production_budget  total_gross_revenue
month_of_release
April                     10806485000          39610890322
August                    12675822719          46200721750
December                  24772446000         110106520078
February                  10994196247          41927998047
January                    7232691000          24468164278
July                      18720308775          88744327892
June                      20644478311          99800102633
March                     14467577021          56026987828
May                       19184024596          93189142692
November                  20703628016          94246480664
October                   11684993000          42337856674
September                 10753760847          34432882048


<ipython-input-28-752849681b87>:3: FutureWarning: Indexing with multiple keys (imp
licitly converted to a tuple of keys) will be deprecated, use a list instead.
  df_months_budget = df.groupby('month_of_release')['production_budget', 'total_gr
oss_revenue'].sum()
```

2. Data visualization: Total revenue compared to budgets by month of release of movies.

In [29]:
```
#Modified horizontal bar graph
barh = df_months_budget.sort_values('total_gross_revenue').plot(kind='barh',figsize=
barh

plt.yticks(fontsize = 20)
plt.xticks(fontsize = 18)
plt.xlabel('Total revenue and Production Budgets (in 100s of million $)', fontsize =
plt.ylabel('Months', fontsize = 22)
plt.title('Total Revenue compared to Budgets by Month of Release for Movies', fontsi
barh.spines['top'].set_visible(False)
barh.spines['right'].set_visible(False)
barh.spines['bottom'].set_linewidth(0.5)
barh.spines['left'].set_visible(True);

plt.show()
```



As we can see, the correlation is quite high between the revenues and the budgets allocated to the production of movies. The strength of the correlation production budget/total revenue is a very relevant factor to be taken into consideration by the Microsoft movie production team. Still need to give key number (return on investment of budget over revenue).

3. Data cleaning for the Average runtime.

In [30]:
```
df2=pd.read_csv('Documents/Flatiron/Phase_1/dsc-phase-1-project-v2-4/Data/rt.movie_i
df2.head()
```

Out[30]:

	id	synopsis	rating	genre	director	writer	theater_date	dvd_date	curren
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971	Sep 25, 2001	N
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo	Aug 17, 2012	Jan 1, 2013	
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996	Apr 18, 2000	N
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton	Dec 9, 1994	Aug 27, 1997	N
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	NaN	NaN	N

In [31]:
```
#Dropping useless columns.

df2 = df2.drop(['writer','dvd_date','currency','box_office'], axis = 1)
df2.head()
```

Out[31]:

	id	synopsis	rating	genre	director	theater_date	runtime	studio
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Oct 9, 1971	104 minutes	NaN
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	Aug 17, 2012	108 minutes	Entertainment One
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Sep 13, 1996	116 minutes	NaN
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Dec 9, 1994	128 minutes	NaN
4	7	NaN	NR	Drama|Romance	Rodney Bennett	NaN	200 minutes	NaN

In [32]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1560 entries, 0 to 1559
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            1560 non-null   int64 
 1   synopsis      1498 non-null   object
 2   rating        1557 non-null   object
 3   genre         1552 non-null   object
 4   director      1361 non-null   object
 5   theater_date  1201 non-null   object
 6   runtime       1530 non-null   object
 7   studio        494 non-null    object
dtypes: int64(1), object(7)
memory usage: 97.6+ KB
```

In [33]:
```
#Removing the word 'minutes' from the runtime column.

df2['runtime'] = df2["runtime"].str.replace("minutes","")
df2['runtime'] = pd.to_numeric(df2['runtime'])
df2['runtime']
```

Out[33]:
```
0       104.0
1       108.0
2       116.0
3       128.0
4       200.0
        ...  
1555    106.0
1556     88.0
1557    111.0
1558    101.0
1559     94.0
Name: runtime, Length: 1560, dtype: float64
```

In [34]:
```
df2['runtime'].dropna()
```

Out[34]:
```
0       104.0
1       108.0
2       116.0
3       128.0
4       200.0
        ...  
1555    106.0
1556     88.0
1557    111.0
1558    101.0
1559     94.0
Name: runtime, Length: 1530, dtype: float64
```

```
In [35]: df2.head()
```

Out[35]:

	id	synopsis	rating	genre	director	theater_date	runtime	studio
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Oct 9, 1971	104.0	NaN
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	Aug 17, 2012	108.0	Entertainment One
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Sep 13, 1996	116.0	NaN
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Dec 9, 1994	128.0	NaN
4	7	NaN	NR	Drama|Romance	Rodney Bennett	NaN	200.0	NaN

4. Average runtime result.

```
In [36]: #Average runtime of a movie.

df2['runtime'].mean()
print('The average runtime of a movie is:', df2['runtime'].mean())
```

```
The average runtime of a movie is: 103.96797385620916
```

5. Data cleaning for the Average rating by genres.

```
In [37]: import sqlite3
conn = sqlite3.connect('Documents/Flatiron/Phase_1/dsc-phase-1-project-v2-4/Data/im.
!pip install pandasql
from pandasql import sqldf
```

```
Requirement already satisfied: pandasql in ./opt/anaconda3/envs/learn-env/lib/pyth
on3.8/site-packages (0.7.3)
Requirement already satisfied: pandas in ./opt/anaconda3/envs/learn-env/lib/python
3.8/site-packages (from pandasql) (1.1.3)
Requirement already satisfied: numpy in ./opt/anaconda3/envs/learn-env/lib/python
3.8/site-packages (from pandasql) (1.18.5)
Requirement already satisfied: sqlalchemy in ./opt/anaconda3/envs/learn-env/lib/py
thon3.8/site-packages (from pandasql) (1.3.20)
Requirement already satisfied: python-dateutil>=2.7.3 in ./opt/anaconda3/envs/lear
n-env/lib/python3.8/site-packages (from pandas->pandasql) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in ./opt/anaconda3/envs/learn-env/lib/
python3.8/site-packages (from pandas->pandasql) (2020.1)
Requirement already satisfied: six>=1.5 in ./opt/anaconda3/envs/learn-env/lib/pyth
on3.8/site-packages (from python-dateutil>=2.7.3->pandas->pandasql) (1.15.0)
```

In [38]:
```
#Pull data from the table movie_basics.

query = """
SELECT *
FROM movie_basics
;
"""
```

In [39]:
```
movie_name = pd.read_sql(query, conn)
```

In [40]:
```
movie_name.head()
```

Out[40]:

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

In [41]:
```
movie_name.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         146144 non-null  object 
 1   primary_title    146144 non-null  object 
 2   original_title   146123 non-null  object 
 3   start_year       146144 non-null  int64  
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
```

In [42]:
```
#Pull data from the table movie_ratings.

query = """
SELECT *
FROM movie_ratings
WHERE numvotes > 200
ORDER BY numvotes
;
"""
```

In [43]:
```
movie_ratings = pd.read_sql(query, conn)
```

In [44]:
```
movie_ratings.head()
```
Out[44]:

	movie_id	averagerating	numvotes
0	tt9204352	6.4	201
1	tt4190256	6.0	201
2	tt5145662	5.7	201
3	tt5987042	3.6	201
4	tt6275296	7.2	201

In [45]:
```
movie_ratings.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21644 entries, 0 to 21643
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   movie_id       21644 non-null  object 
 1   averagerating  21644 non-null  float64
 2   numvotes       21644 non-null  int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 507.4+ KB
```

In [46]:
```
#Joining the tables using the movie_id column present in both tables.

query = """
SELECT movie_basics.movie_id,
movie_basics.primary_title,
movie_basics.genres
FROM movie_basics
INNER JOIN movie_ratings ON movie_basics.movie_id=movie_ratings.movie_id
;
"""
```

In [47]:
```
movie_infos = pd.read_sql(query, conn)
```

In [48]:
```
movie_infos.head()
```
Out[48]:

	movie_id	primary_title	genres
0	tt0063540	Sunghursh	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Biography,Drama
2	tt0069049	The Other Side of the Wind	Drama
3	tt0069204	Sabse Bada Sukh	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	Comedy,Drama,Fantasy

In [49]:
```
movie_infos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   movie_id       73856 non-null  object
 1   primary_title  73856 non-null  object
 2   genres         73052 non-null  object
dtypes: object(3)
memory usage: 1.7+ MB
```

In [50]:
```
movie_infos.dropna()
```

Out[50]:

	movie_id	primary_title	genres
0	tt0063540	Sunghursh	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Biography,Drama
2	tt0069049	The Other Side of the Wind	Drama
3	tt0069204	Sabse Bada Sukh	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	Comedy,Drama,Fantasy
...	...	...	...
73850	tt9913056	Swarm Season	Documentary
73851	tt9913084	Diabolik sono io	Documentary
73852	tt9914286	Sokagin Çocuklari	Drama,Family
73853	tt9914642	Albatross	Documentary
73855	tt9916160	Drømmeland	Documentary

73052 rows × 3 columns

```
In [51]: #Cleaning of the 'genres' column.

         movie_infos['genres'] = movie_infos['genres'].str.split(',')
         movie_infos
```

```
Out[51]:
             movie_id                   primary_title                      genres
         0   tt0063540                      Sunghursh     [Action, Crime, Drama]
         1   tt0066787  One Day Before the Rainy Season       [Biography, Drama]
         2   tt0069049            The Other Side of the Wind             [Drama]
         3   tt0069204                  Sabse Bada Sukh          [Comedy, Drama]
         4   tt0100275          The Wandering Soap Opera  [Comedy, Drama, Fantasy]
       ...         ...                              ...                      ...
     73851   tt9913084                Diabolik sono io            [Documentary]
     73852   tt9914286                Sokagin Çocuklari          [Drama, Family]
     73853   tt9914642                        Albatross            [Documentary]
     73854   tt9914942        La vida sense la Sara Amat                   None
     73855   tt9916160                       Drømmeland            [Documentary]

         73856 rows × 3 columns
```

```
In [52]: movie_infos.explode('genres')
```

```
Out[52]:
             movie_id                   primary_title        genres
         0   tt0063540                        Sunghursh        Action
         0   tt0063540                        Sunghursh         Crime
         0   tt0063540                        Sunghursh         Drama
         1   tt0066787  One Day Before the Rainy Season     Biography
         1   tt0066787  One Day Before the Rainy Season         Drama
       ...         ...                              ...           ...
     73852   tt9914286                Sokagin Çocuklari         Drama
     73852   tt9914286                Sokagin Çocuklari        Family
     73853   tt9914642                        Albatross   Documentary
     73854   tt9914942       La vida sense la Sara Amat          None
     73855   tt9916160                       Drømmeland   Documentary

         129294 rows × 3 columns
```

In [53]:
```
#Let's count the values for the genres column.

movie_infos['genres'].dropna()
```

Out[53]:
```
0             [Action, Crime, Drama]
1                [Biography, Drama]
2                           [Drama]
3                   [Comedy, Drama]
4          [Comedy, Drama, Fantasy]
                     ...
73850                [Documentary]
73851                [Documentary]
73852              [Drama, Family]
73853                [Documentary]
73855                [Documentary]
Name: genres, Length: 73052, dtype: object
```

In [54]:
```
movie_ratings.head()
```

Out[54]:

	movie_id	averagerating	numvotes
0	tt9204352	6.4	201
1	tt4190256	6.0	201
2	tt5145662	5.7	201
3	tt5987042	3.6	201
4	tt6275296	7.2	201

In [55]:
```
x_df = pd.merge(movie_ratings, movie_infos, on='movie_id')
x_df
```

Out[55]:

	movie_id	averagerating	numvotes	primary_title	genres
0	tt9204352	6.4	201	Porndemic	[Documentary, Drama, Mystery]
1	tt4190256	6.0	201	The Orphanage	[Action, Biography, Drama]
2	tt5145662	5.7	201	Monsters at Large	[Family]
3	tt5987042	3.6	201	The Devil's Well	[Horror, Mystery, Thriller]
4	tt6275296	7.2	201	The Rules for Everything	[Comedy, Drama]
...	...	...	...	...	...
21639	tt0848228	8.1	1183655	The Avengers	[Action, Adventure, Sci-Fi]
21640	tt1853728	8.4	1211405	Django Unchained	[Drama, Western]
21641	tt0816692	8.6	1299334	Interstellar	[Adventure, Drama, Sci-Fi]
21642	tt1345836	8.4	1387769	The Dark Knight Rises	[Action, Thriller]
21643	tt1375666	8.8	1841066	Inception	[Action, Adventure, Sci-Fi]

21644 rows × 5 columns

In [56]:
```
#Let's use the function .explode() for the column 'genres'.

x_df = x_df.explode('genres')
x_df
```

Out[56]:

	movie_id	averagerating	numvotes	primary_title	genres
0	tt9204352	6.4	201	Porndemic	Documentary
0	tt9204352	6.4	201	Porndemic	Drama
0	tt9204352	6.4	201	Porndemic	Mystery
1	tt4190256	6.0	201	The Orphanage	Action
1	tt4190256	6.0	201	The Orphanage	Biography
...	...	...	...	...	...
21642	tt1345836	8.4	1387769	The Dark Knight Rises	Action
21642	tt1345836	8.4	1387769	The Dark Knight Rises	Thriller
21643	tt1375666	8.8	1841066	Inception	Action
21643	tt1375666	8.8	1841066	Inception	Adventure
21643	tt1375666	8.8	1841066	Inception	Sci-Fi

44143 rows × 5 columns

In [57]:
```
#Dropping the genres that have less than 200 votes.

x_df.drop(index=x_df[x_df['genres'] == 'News'].index, inplace=True)
x_df.drop(index=x_df[x_df['genres'] == 'Game-Show'].index, inplace=True)
x_df.drop(index=x_df[x_df['genres'] == 'Musical'].index, inplace=True)
x_df.drop(index=x_df[x_df['genres'] == 'Western'].index, inplace=True)
```

In [58]:
```
x_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43300 entries, 0 to 21643
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   movie_id       43300 non-null  object 
 1   averagerating  43300 non-null  float64
 2   numvotes       43300 non-null  int64  
 3   primary_title  43300 non-null  object 
 4   genres         43292 non-null  object 
dtypes: float64(1), int64(1), object(3)
memory usage: 2.0+ MB
```

```
In [59]: x_df.isna().sum()
```

```
Out[59]: movie_id         0
         averagerating    0
         numvotes         0
         primary_title    0
         genres           8
         dtype: int64
```

```
In [60]: x_df.dropna().isna().sum()
```

```
Out[60]: movie_id         0
         averagerating    0
         numvotes         0
         primary_title    0
         genres           0
         dtype: int64
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43300 entries, 0 to 21643
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   movie_id       43300 non-null  object 
 1   averagerating  43300 non-null  float64
 2   numvotes       43300 non-null  int64  
 3   primary_title  43300 non-null  object 
 4   genres         43292 non-null  object 
dtypes: float64(1), int64(1), object(3)
memory usage: 2.0+ MB
```

```
In [62]: #Filtering the genres by ratings.

         ratings_by_genre = x_df[['genres', 'averagerating']].reset_index()
         ratings_by_genre
```

Out[62]:

	index	genres	averagerating
0	0	Documentary	6.4
1	0	Drama	6.4
2	0	Mystery	6.4
3	1	Action	6.0
4	1	Biography	6.0
...	...	...	...
43295	21642	Action	8.4
43296	21642	Thriller	8.4
43297	21643	Action	8.8
43298	21643	Adventure	8.8
43299	21643	Sci-Fi	8.8

43300 rows × 3 columns

```
In [63]: #Counting the genres recurrence.

         counts = ratings_by_genre.value_counts('genres')
         counts
```

```
Out[63]: genres
         Drama          10426
         Comedy          6618
         Thriller        3569
         Action          3345
         Horror          2786
         Romance         2703
         Documentary     2218
         Crime           2183
         Adventure       1610
         Mystery         1294
         Biography       1180
         Sci-Fi           958
         Family           843
         Fantasy          833
         History          749
         Animation        681
         Music            562
         Sport            407
         War              327
         dtype: int64
```

```
In [64]: #Calcuating the average rating mean per genre.

         ratings_by_genre = ratings_by_genre.groupby('genres').mean('averagerating').sort_val
         ratings_by_genre.head(3)
```

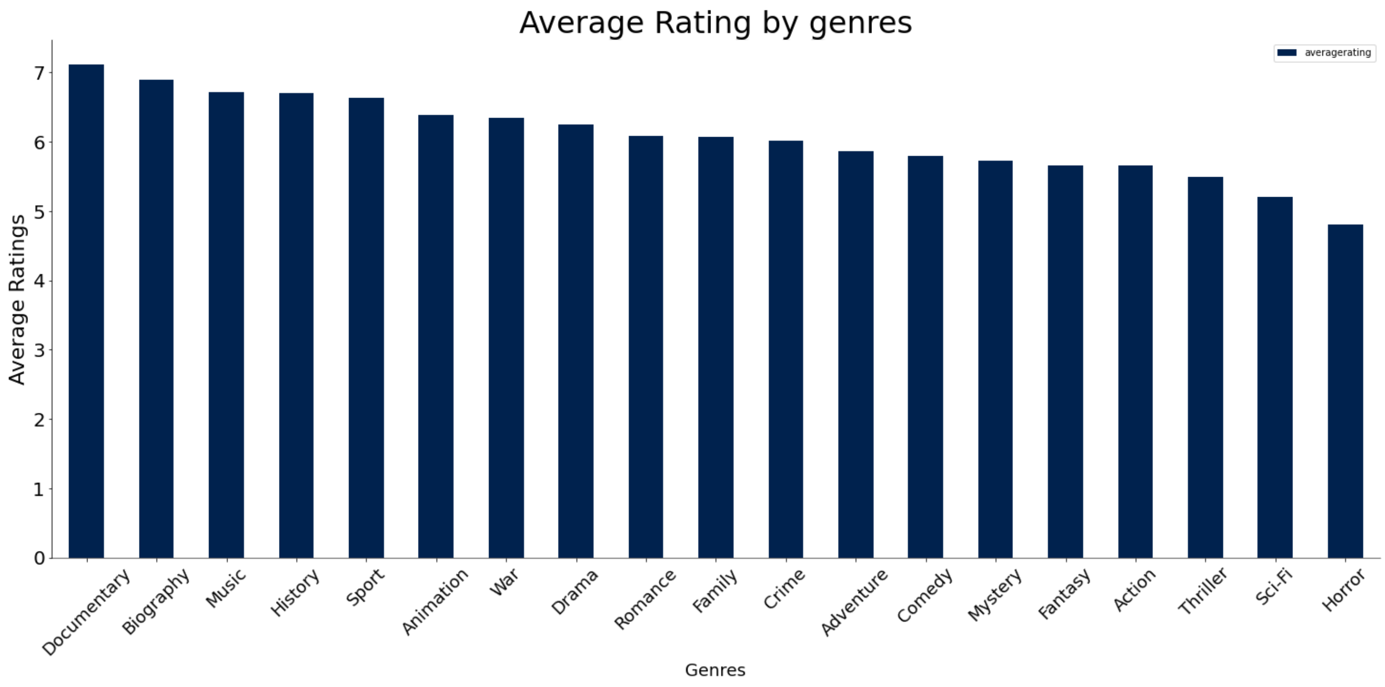
```
Out[64]:
                    index      averagerating
         genres
         Documentary   8878.543282    7.117223
         Biography    12747.827966    6.888475
         Music        11533.754448    6.720107
```

```
In [65]: #Calcuating the average rating mean per genre and dropping the index column.

         ratings_by_genre2 = ratings_by_genre.drop('index', axis=1)
         ratings_by_genre2.head(5)
```

```
Out[65]:
                    averagerating
         genres
         Documentary      7.117223
         Biography        6.888475
         Music            6.720107
         History          6.706008
         Sport            6.630467
```

6. Average rating by genres data visualization.

```
In [66]: #Plotting a graph showing the correlation between genres and average votes.

         bar = ratings_by_genre2.plot(kind='bar',figsize=(25, 10), colormap='cividis', legend
         bar

         plt.yticks(fontsize = 20)
         plt.xticks(fontsize = 18)
         plt.xticks(rotation = 45)
         plt.xlabel('Genres', fontsize = 18)
         plt.ylabel('Average Ratings', fontsize = 22)
         plt.title('Average Rating by genres', fontsize=32)
         bar.spines['top'].set_visible(False)
         bar.spines['right'].set_visible(False)
         bar.spines['bottom'].set_linewidth(0.5)
         bar.spines['left'].set_visible(True);

         plt.show()
```



7. Data cleaning for the Average Budget per genre.

For that, we need to merge the csv file movie_budgets and the im.db database.

In [67]:
```
df.head()
```

Out[67]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	total_gross_revenue	month_of
0	2009-12-18	Avatar	425000000	760507625	2776345279	3536852904	D
1	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	1286727750	
2	2019-06-07	Dark Phoenix	350000000	42762350	149762350	192524700	
3	2015-05-01	Avengers: Age of Ultron	330600000	459005868	1403013963	1862019831	
4	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	1936903129	D

In [68]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5782 entries, 0 to 5781
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   release_date         5782 non-null   datetime64[ns]
 1   movie                5782 non-null   object
 2   production_budget    5782 non-null   int64
 3   domestic_gross       5782 non-null   int64
 4   worldwide_gross      5782 non-null   int64
 5   total_gross_revenue  5782 non-null   int64
 6   month_of_release     5782 non-null   object
dtypes: datetime64[ns](1), int64(4), object(2)
memory usage: 361.4+ KB
```

In [69]:
```
movie_name.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         146144 non-null  object 
 1   primary_title    146144 non-null  object 
 2   original_title   146123 non-null  object 
 3   start_year       146144 non-null  int64  
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
```

In [70]:
```
#We need to merge the imdb database with the csv file on the movie name in order.
#to see which genre has the higher production budget.

df_genre_gross = movie_name.merge(df, left_on = "original_title", right_on = "movie"
df_genre_gross.head()
```

Out[70]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	release_date	
0	tt0249516	Foodfight!	Foodfight!	2012	91.0	Action,Animation,Comedy	2012-12-31	Fo
1	tt0293429	Mortal Kombat	Mortal Kombat	2021	NaN	Action,Adventure,Fantasy	1995-08-18	
2	tt0326592	The Overnight	The Overnight	2010	88.0	None	2015-06-19	O
3	tt3844362	The Overnight	The Overnight	2015	79.0	Comedy,Mystery	2015-06-19	O
4	tt0337692	On the Road	On the Road	2012	124.0	Adventure,Drama,Romance	2013-03-22	

In [71]:
```
#Let's check for duplicates.

df_genre_gross.drop_duplicates(inplace=True)
df_genre_gross.head()
```

Out[71]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	release_date	
0	tt0249516	Foodfight!	Foodfight!	2012	91.0	Action,Animation,Comedy	2012-12-31	Fo
1	tt0293429	Mortal Kombat	Mortal Kombat	2021	NaN	Action,Adventure,Fantasy	1995-08-18	
2	tt0326592	The Overnight	The Overnight	2010	88.0	None	2015-06-19	O
3	tt3844362	The Overnight	The Overnight	2015	79.0	Comedy,Mystery	2015-06-19	O
4	tt0337692	On the Road	On the Road	2012	124.0	Adventure,Drama,Romance	2013-03-22	

```
In [72]: #Null values

         df_genre_gross.isna().sum()
```

```
Out[72]: movie_id              0
         primary_title         0
         original_title        0
         start_year            0
         runtime_minutes     467
         genres               64
         release_date          0
         movie                 0
         production_budget     0
         domestic_gross        0
         worldwide_gross       0
         total_gross_revenue   0
         month_of_release      0
         dtype: int64
```

```
         df_genre_gross.head()
```

```
In [73]: #Dropping some columns to ease the reading of the table.

         df_genre_gross = df_genre_gross.drop(['runtime_minutes', 'release_date', 'domestic_g
         df_genre_gross.head()
```

Out[73]:

	movie_id	primary_title	original_title	start_year	genres	movie	production_budget	mo
0	tt0249516	Foodfight!	Foodfight!	2012	Action,Animation,Comedy	Foodfight!	45000000	
1	tt0293429	Mortal Kombat	Mortal Kombat	2021	Action,Adventure,Fantasy	Mortal Kombat	20000000	
2	tt0326592	The Overnight	The Overnight	2010	None	The Overnight	200000	
3	tt3844362	The Overnight	The Overnight	2015	Comedy,Mystery	The Overnight	200000	
4	tt0337692	On the Road	On the Road	2012	Adventure,Drama,Romance	On the Road	25000000	

```
In [74]: df_genre_gross.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3537 entries, 0 to 3536
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   movie_id           3537 non-null   object
 1   primary_title      3537 non-null   object
 2   original_title     3537 non-null   object
 3   start_year         3537 non-null   int64 
 4   genres             3473 non-null   object
 5   movie              3537 non-null   object
 6   production_budget  3537 non-null   int64 
 7   month_of_release   3537 non-null   object
dtypes: int64(2), object(6)
memory usage: 248.7+ KB
```

```
In [75]: #Separating the 'genres' column.

         df_genre_gross['genres_split'] = df_genre_gross['genres'].str.split(',')
         df_genre_gross['genres_split']
```

```
Out[75]: 0          [Action, Animation, Comedy]
         1         [Action, Adventure, Fantasy]
         2                                 None
         3                    [Comedy, Mystery]
         4          [Adventure, Drama, Romance]
                              ...
         3532                           [Drama]
         3533              [Documentary, Sport]
         3534                           [Crime]
         3535          [Action, Drama, Romance]
         3536                     [Documentary]
         Name: genres_split, Length: 3537, dtype: object
```

```
In [76]: df_genre_gross = df_genre_gross.explode('genres_split')
```

```
In [77]: df_genre_gross
```

Out[77]:

	movie_id	primary_title	original_title	start_year	genres	movie	production_budget	n
0	tt0249516	Foodfight!	Foodfight!	2012	Action,Animation,Comedy	Foodfight!	45000000	
0	tt0249516	Foodfight!	Foodfight!	2012	Action,Animation,Comedy	Foodfight!	45000000	
0	tt0249516	Foodfight!	Foodfight!	2012	Action,Animation,Comedy	Foodfight!	45000000	
1	tt0293429	Mortal Kombat	Mortal Kombat	2021	Action,Adventure,Fantasy	Mortal Kombat	20000000	
1	tt0293429	Mortal Kombat	Mortal Kombat	2021	Action,Adventure,Fantasy	Mortal Kombat	20000000	
...	...	...	...	...	...	...	...	
3534	tt9729206	Diner	Diner	2019	Crime	Diner	5000000	
3535	tt9805168	Traitor	Traitor	2015	Action,Drama,Romance	Traitor	22000000	
3535	tt9805168	Traitor	Traitor	2015	Action,Drama,Romance	Traitor	22000000	
3535	tt9805168	Traitor	Traitor	2015	Action,Drama,Romance	Traitor	22000000	
3536	tt9893078	Sublime	Sublime	2019	Documentary	Sublime	1800000	

7404 rows × 9 columns

```
In [78]: #Renaming of columns.

         df_genre_gross.rename(columns = {'genres_split':'Genres', 'production_budget':'Produ
         df_genre_gross.head()
```

```
Out[78]:
```

	movie_id	primary_title	original_title	start_year	grouped_genres	movie	Production_Budget	mon
0	tt0249516	Foodfight!	Foodfight!	2012	Action,Animation,Comedy	Foodfight!	45000000	
0	tt0249516	Foodfight!	Foodfight!	2012	Action,Animation,Comedy	Foodfight!	45000000	
0	tt0249516	Foodfight!	Foodfight!	2012	Action,Animation,Comedy	Foodfight!	45000000	
1	tt0293429	Mortal Kombat	Mortal Kombat	2021	Action,Adventure,Fantasy	Mortal Kombat	20000000	
1	tt0293429	Mortal Kombat	Mortal Kombat	2021	Action,Adventure,Fantasy	Mortal Kombat	20000000	

```
In [79]: #putting the 2 relevant columns together before plotting
         avg_per_genre = df_genre_gross[['Genres','Production_Budget']]

         #calculating the average budget per genre
         avg_per_genre = df_genre_gross.groupby('Genres').mean('Production_Budget').reset_ind

         #input
         avg_per_genre
```

Out[79]:

	Genres	start_year	Production_Budget
0	Action	2014.316932	6.201080e+07
1	Adventure	2014.447205	8.689506e+07
2	Animation	2014.398601	8.139277e+07
3	Biography	2014.407258	2.521123e+07
4	Comedy	2013.840304	3.336855e+07
5	Crime	2013.921466	2.716669e+07
6	Documentary	2014.554839	2.463367e+07
7	Drama	2014.142070	2.347606e+07
8	Family	2014.000000	4.713292e+07
9	Fantasy	2014.341969	6.604272e+07
10	History	2014.704545	3.358591e+07
11	Horror	2014.202439	1.792089e+07
12	Music	2013.795181	1.605687e+07
13	Musical	2015.333333	3.759664e+07
14	Mystery	2014.190476	2.289054e+07
15	News	2013.666667	3.880000e+07
16	Reality-TV	2016.000000	1.000000e+06
17	Romance	2013.525526	1.973998e+07
18	Sci-Fi	2014.625000	6.431965e+07
19	Sport	2014.273973	2.523664e+07
20	Thriller	2014.126316	2.621675e+07
21	War	2013.468085	2.281702e+07
22	Western	2014.807692	3.627500e+07

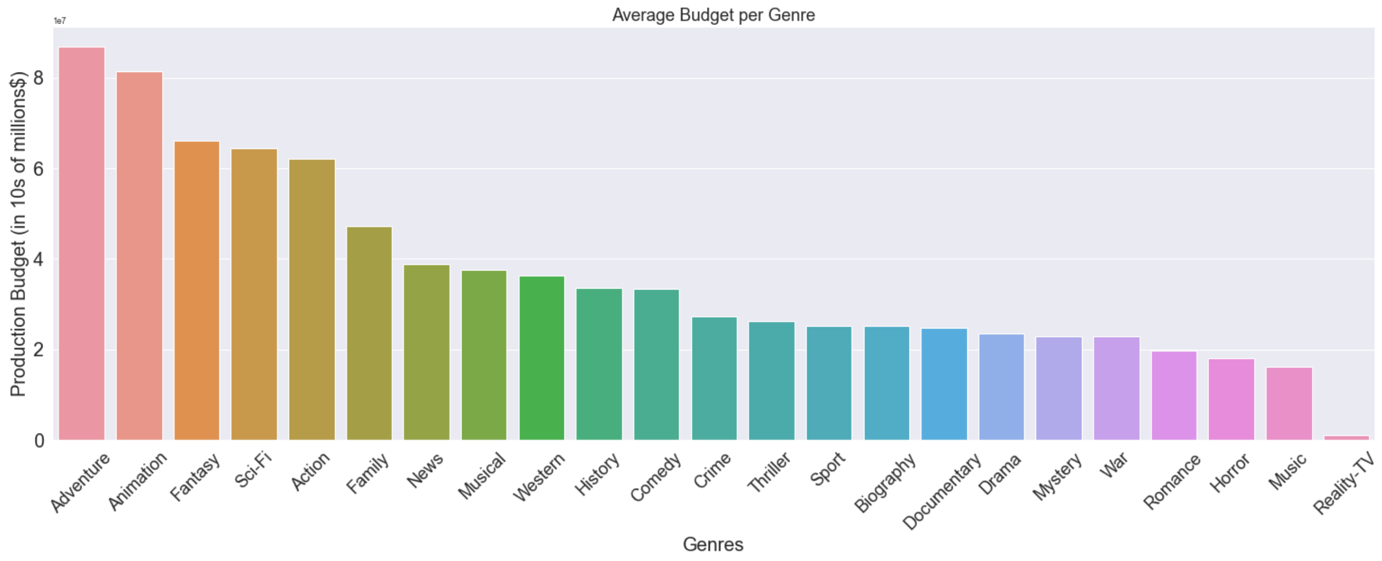
In [80]:
```
#Dropping of the 'start_year" column'.

avg_per_genre = avg_per_genre.drop(['start_year'], axis=1)
avg_per_genre.head()
```

Out[80]:

	Genres	Production_Budget
0	Action	6.201080e+07
1	Adventure	8.689506e+07
2	Animation	8.139277e+07
3	Biography	2.521123e+07
4	Comedy	3.336855e+07

8. Average Budget per genre data visualization.

In [81]:
```
#Size of the barplot
plt.figure(figsize=(25, 8))
sns.set(font_scale=0.8)
sns.set_palette("pastel")

#Setting the dataset and x and y axis and ordering in ascending order
sns.barplot(x = 'Genres',
            y = 'Production_Budget',
            data = avg_per_genre,
            order=avg_per_genre.sort_values('Production_Budget', ascending=False).Ge

plt.yticks(fontsize = 20)
plt.xticks(fontsize = 18)
plt.xticks(rotation = 45)
plt.xlabel("Genres", size=20)
plt.ylabel("Production Budget (in 10s of millions$)", size=20)
plt.title("Average Budget per Genre", size=18);

plt.show();
```



Conclusion

The datasets used to perform the analysis lead to 3 recommendations given to Microsoft in order for them to consider entering the movie industry:

- I. The release period is of prime importance.
- II. Microsoft should consider the highest rated movie genres.
- III. An accurate analysis of budget is needed in order to avoid overspending (it is easy to overspend and it is not necessarily worth it!).

A step further

Based on the data provided, we could perform a deeper analysis in order to generate more insights:

- I. More details on budget/ratings ratios (for more accurate budget projections).
- II. More details on genres combinations.
- III. Research on directors, actors, ... that would help get a higher ROI.