Drive Link
https://drive.google.com/file/d/1PvkOCIqffH3q-UASesNVMsbVIkxCfG_g/view?usp=sharing

WEEK 9 : Task 13 – IMDB Movie Rating

Project Description – In this project, we will collect, transform, and organize data so we can derive meaning conclusions that result in informed decisions. Here, we have provided an 'IMDB Movie Rating' dataset, which includes 28 variables for 5043 movies spanning more than a century.
We would like to know what makes a movie more successful than others, what kind of movies are more successful, which movies have registered highest profit, which movies have top IMDB ratings, which are most successful directors and actors and many more insights.

Approach – To get right insights, we need to follow Data Analytics process i.e. Ask, Prepare, Process, Analyze, Share and Act. To proceed with we first need to clean our data and make it error free. So, that we can get accurate results.

Tech-Stack Used – Jupyter Notebook 6.4.5 which allows us to create and share documents which contains codes, plots, visualizations and project documentations.

Insights – After analyzing data, we find out that Avatar, Jurassic World and Titanic are top 3 movies with highest profit in which two of them are directed by James Cameron, The Shawshank Redemption is movie with highest IMDB score followed by The Godfather, Family + Sci-Fi most popular combo of genres and many more.

Results – This project has helped me gain hands-on experience that has helped me gain a better understanding of the data analytics process - what are the right questions to ask, how to handle rows and columns with missing values, and when to use which plot to get better insights and how to apply theoretical knowledge in practical scenarios.

In [3]:

```
# Supress Warnings

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

In [3]:

```
# Import the numpy and pandas packages

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

Task 1: Reading and Inspection

- ## Subtask 1.1: Import and read

Import and read the movie database. Store it in a variable called `movies` .

In [4]:

```
movies = pd.read_csv('IMDB_Movies.csv') # Write your code for importing the csv file here
movies
```

Out[4]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_fa
0	Color	James Cameron	723.0	178.0	0.0	
1	Color	Gore Verbinski	302.0	169.0	563.0	
2	Color	Sam Mendes	602.0	148.0	0.0	
3	Color	Christopher Nolan	813.0	164.0	22000.0	
4	NaN	Doug Walker	NaN	NaN	131.0	
...	...	...	...	...	...	
5038	Color	Scott Smith	1.0	87.0	2.0	
5039	Color	NaN	43.0	43.0	NaN	
5040	Color	Benjamin Roberds	13.0	76.0	0.0	
5041	Color	Daniel Hsia	14.0	100.0	0.0	
5042	Color	Jon Gunn	43.0	90.0	16.0	

5043 rows × 28 columns

- **Subtask 1.2: Inspect the dataframe**

Inspect the dataframe's columns, shapes, variable types etc.

In [5]:

```
# Write your code for inspection here
movies.columns
```

Out[5]:

```
Index(['color', 'director_name', 'num_critic_for_reviews', 'duration',
       'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
       'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
       'movie_title', 'num_voted_users', 'cast_total_facebook_likes',
       'actor_3_name', 'facenumber_in_poster', 'plot_keywords',
       'movie_imdb_link', 'num_user_for_reviews', 'language', 'country',
       'content_rating', 'budget', 'title_year', 'actor_2_facebook_likes',
       'imdb_score', 'aspect_ratio', 'movie_facebook_likes'],
      dtype='object')
```

In [6]:

```
movies.shape
```

Out[6]:

```
(5043, 28)
```

In [7]:

```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5043 entries, 0 to 5042
Data columns (total 28 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   color                      5024 non-null   object 
 1   director_name              4939 non-null   object 
 2   num_critic_for_reviews     4993 non-null   float64
 3   duration                   5028 non-null   float64
 4   director_facebook_likes    4939 non-null   float64
 5   actor_3_facebook_likes     5020 non-null   float64
 6   actor_2_name               5030 non-null   object 
 7   actor_1_facebook_likes     5036 non-null   float64
 8   gross                      4159 non-null   float64
 9   genres                     5043 non-null   object 
 10  actor_1_name               5036 non-null   object 
 11  movie_title                5043 non-null   object 
 12  num_voted_users            5043 non-null   int64  
 13  cast_total_facebook_likes  5043 non-null   int64  
 14  actor_3_name               5020 non-null   object 
 15  facenumber_in_poster       5030 non-null   float64
 16  plot_keywords              4890 non-null   object 
 17  movie_imdb_link            5043 non-null   object 
 18  num_user_for_reviews       5023 non-null   object 
 19  language                   5031 non-null   object 
 20  country                    5038 non-null   object 
 21  content_rating             4740 non-null   object 
 22  budget                     4551 non-null   float64
 23  title_year                 4935 non-null   float64
 24  actor_2_facebook_likes     5030 non-null   float64
 25  imdb_score                 5043 non-null   float64
 26  aspect_ratio               4714 non-null   float64
 27  movie_facebook_likes       5043 non-null   int64  
dtypes: float64(12), int64(3), object(13)
memory usage: 1.1+ MB
```

Task 2: Cleaning the Data

- ## Subtask 2.1: Inspect Null values

Find out the number of Null values in all the columns and rows. Also, find the percentage of Null values in each column. Round off the percentages upto two decimal places.

In [8]:

```
# Write your code for column-wise null count here
movies.isnull().sum()
```

Out[8]:

```
color                         19
director_name                104
num_critic_for_reviews        50
duration                      15
director_facebook_likes      104
actor_3_facebook_likes        23
actor_2_name                  13
actor_1_facebook_likes         7
gross                        884
genres                         0
actor_1_name                   7
movie_title                    0
num_voted_users                0
cast_total_facebook_likes      0
actor_3_name                  23
facenumber_in_poster          13
plot_keywords                153
movie_imdb_link                0
num_user_for_reviews          20
language                      12
country                        5
content_rating               303
budget                       492
title_year                   108
actor_2_facebook_likes        13
imdb_score                     0
aspect_ratio                 329
movie_facebook_likes           0
dtype: int64
```

In [9]:

```
# Write your code for row-wise null count here
movies.isnull().sum(axis=1)
```

Out[9]:

```
0        0
1        0
2        0
3        0
4       13
        ..
5038     4
5039     5
5040     4
5041     2
5042     0
Length: 5043, dtype: int64
```

In [10]:

```
# Write your code for column-wise null percentages here
round(movies.isnull().sum()/len(movies)*100,2)
```

Out[10]:

```
color                         0.38
director_name                 2.06
num_critic_for_reviews        0.99
duration                      0.30
director_facebook_likes       2.06
actor_3_facebook_likes        0.46
actor_2_name                  0.26
actor_1_facebook_likes        0.14
gross                        17.53
genres                        0.00
actor_1_name                  0.14
movie_title                   0.00
num_voted_users               0.00
cast_total_facebook_likes     0.00
actor_3_name                  0.46
facenumber_in_poster          0.26
plot_keywords                 3.03
movie_imdb_link               0.00
num_user_for_reviews          0.40
language                      0.24
country                       0.10
content_rating                6.01
budget                        9.76
title_year                    2.14
actor_2_facebook_likes        0.26
imdb_score                    0.00
aspect_ratio                  6.52
movie_facebook_likes          0.00
dtype: float64
```

- ### Subtask 2.2: Drop unecessary columns

For this assignment, you will mostly be analyzing the movies with respect to the ratings, gross collection, popularity of movies, etc. So many of the columns in this dataframe are not required. So it is advised to drop the following columns.

- color
- director_facebook_likes
- actor_1_facebook_likes
- actor_2_facebook_likes
- actor_3_facebook_likes
- actor_2_name
- cast_total_facebook_likes
- actor_3_name
- duration
- facenumber_in_poster
- content_rating
- country
- movie_imdb_link
- aspect_ratio

- plot_keywords

In [11]:

```
# Write your code for dropping the columns here. It is advised to keep inspecting the dataf
movies= movies.drop(['color','director_facebook_likes','actor_1_facebook_likes','actor_2_fa
'actor_2_name','cast_total_facebook_likes','actor_3_name','duration','facenumber_in_poster'
'movie_imdb_link','aspect_ratio','plot_keywords'],axis=1)
movies
```

Out[11]:

	director_name	num_critic_for_reviews	gross	genres	actor_1_r
0	James Cameron	723.0	760505847.0	Action|Adventure|Fantasy|Sci-Fi	CCH Pou
1	Gore Verbinski	302.0	309404152.0	Action|Adventure|Fantasy	Johnny I
2	Sam Mendes	602.0	200074175.0	Action|Adventure|Thriller	Chris \
3	Christopher Nolan	813.0	448130642.0	Action|Thriller	Tom H
4	Doug Walker	NaN	NaN	Documentary	Doug W
...	...	...	...	...	
5038	Scott Smith	1.0	NaN	Comedy|Drama	Eric Ma
5039	NaN	43.0	NaN	Crime|Drama|Mystery|Thriller	Natalie
5040	Benjamin Roberds	13.0	NaN	Drama|Horror|Thriller	Eva Boe
5041	Daniel Hsia	14.0	10443.0	Comedy|Drama|Romance	Alan
5042	Jon Gunn	43.0	85222.0	Documentary	John Au

5043 rows × 13 columns

- ### Subtask 2.3: Drop unecessary rows using columns with high Null percentages

Now, on inspection you might notice that some columns have large percentage (greater than 5%) of Null values. Drop all the rows which have Null values for such columns.

In [12]:

```
# Write your code for dropping the rows here
round(100*(movies.isnull().sum()/len(movies.index)), 2)>5
movies = movies[~pd.isnull(movies['gross'])]
movies = movies[~pd.isnull(movies['budget'])]
round(100*(movies.isnull().sum(axis=0)/len(movies.index)), 2)
```

Out[12]:

```
director_name             0.00
num_critic_for_reviews    0.03
gross                     0.00
genres                    0.00
actor_1_name              0.08
movie_title               0.00
num_voted_users           0.00
num_user_for_reviews      0.00
language                  0.08
budget                    0.00
title_year                0.00
imdb_score                0.00
movie_facebook_likes      0.00
dtype: float64
```

- **Subtask 2.4: Fill NaN values**

You might notice that the `language` column has some NaN values. Here, on inspection, you will see that it is safe to replace all the missing values with `'English'` .

In [13]:

```
# Write your code for filling the NaN values in the 'language' column here
movies['language'].fillna('English',inplace=True) #fillna() method returns a new DataFrame
movies
```

Out[13]:

	director_name	num_critic_for_reviews	gross	genres	ac
0	James Cameron	723.0	760505847.0	Action|Adventure|Fantasy|Sci-Fi	C
1	Gore Verbinski	302.0	309404152.0	Action|Adventure|Fantasy	.
2	Sam Mendes	602.0	200074175.0	Action|Adventure|Thriller	
3	Christopher Nolan	813.0	448130642.0	Action|Thriller	
5	Andrew Stanton	462.0	73058679.0	Action|Adventure|Sci-Fi	I
...	...	...	...	...	
5033	Shane Carruth	143.0	424760.0	Drama|Sci-Fi|Thriller	Sł
5034	Neill Dela Llana	35.0	70071.0	Thriller	I
5035	Robert Rodriguez	56.0	2040920.0	Action|Crime|Drama|Romance|Thriller	
5037	Edward Burns	14.0	4584.0	Comedy|Drama	
5042	Jon Gunn	43.0	85222.0	Documentary	

3891 rows × 13 columns

- ## Subtask 2.5: Check the number of retained rows

You might notice that two of the columns viz. `num_critic_for_reviews` and `actor_1_name` have small percentages of NaN values left. You can let these columns as it is for now. Check the number and percentage of the rows retained after completing all the tasks above.

In [14]:

```
# Write your code for checking number of retained rows here
round(100*(movies.shape[0]/5043),2) #5043 obtained from subtasking 1.2
```

Out[14]:

```
77.16
```

Checkpoint 1: You might have noticed that we still have around `77%` of the rows!

Task 3: Data Analysis

- ## Subtask 3.1: Change the unit of columns

Convert the unit of the `budget` and `gross` columns from `$` to `million $`.

In [15]:

```
# Write your code for unit conversion here
movies['budget']=(movies['budget']/1000000).round(2)
movies['gross']=(movies['gross']/1000000).round(2)
movies
```

Out[15]:

	director_name	num_critic_for_reviews	gross	genres	actor_1_
0	James Cameron	723.0	760.51	Action|Adventure|Fantasy|Sci-Fi	CCH Pc
1	Gore Verbinski	302.0	309.40	Action|Adventure|Fantasy	Johnny
2	Sam Mendes	602.0	200.07	Action|Adventure|Thriller	Chr
3	Christopher Nolan	813.0	448.13	Action|Thriller	Tom
5	Andrew Stanton	462.0	73.06	Action|Adventure|Sci-Fi	Daryl S
...	...	...	...	...	
5033	Shane Carruth	143.0	0.42	Drama|Sci-Fi|Thriller	Shane C
5034	Neill Dela Llana	35.0	0.07	Thriller	Ian Gar
5035	Robert Rodriguez	56.0	2.04	Action|Crime|Drama|Romance|Thriller	Ga
5037	Edward Burns	14.0	0.00	Comedy|Drama	Kerry
5042	Jon Gunn	43.0	0.09	Documentary	John /

3891 rows × 13 columns

- ## Subtask 3.2: Find the movies with highest profit

1. Create a new column called `profit` which contains the difference of the two columns: `gross` and `budget`.
2. Sort the dataframe using the `profit` column as reference.
3. Plot `profit` (y-axis) vs `budget` (x- axis) and observe the outliers using the appropriate chart type.

4. Extract the top ten profiting movies in descending order and store them in a new dataframe - `top10`

In [16]:

```
# Write your code for creating the profit column here
movies['profit']= movies['gross']-movies['budget']
movies
```

Out[16]:

	director_name	num_critic_for_reviews	gross	genres	actor_1_
0	James Cameron	723.0	760.51	Action|Adventure|Fantasy|Sci-Fi	CCH Pc
1	Gore Verbinski	302.0	309.40	Action|Adventure|Fantasy	Johnny
2	Sam Mendes	602.0	200.07	Action|Adventure|Thriller	Chı
3	Christopher Nolan	813.0	448.13	Action|Thriller	Tom
5	Andrew Stanton	462.0	73.06	Action|Adventure|Sci-Fi	Daryl S
...	...	...	...	...	
5033	Shane Carruth	143.0	0.42	Drama|Sci-Fi|Thriller	Shane C
5034	Neill Dela Llana	35.0	0.07	Thriller	Ian Gar
5035	Robert Rodriguez	56.0	2.04	Action|Crime|Drama|Romance|Thriller	Ga
5037	Edward Burns	14.0	0.00	Comedy|Drama	Kerry
5042	Jon Gunn	43.0	0.09	Documentary	John A

3891 rows × 14 columns

In [17]:

```
# Write your code for sorting the dataframe here
movies=movies.sort_values(by='profit',ascending=False)
movies
```

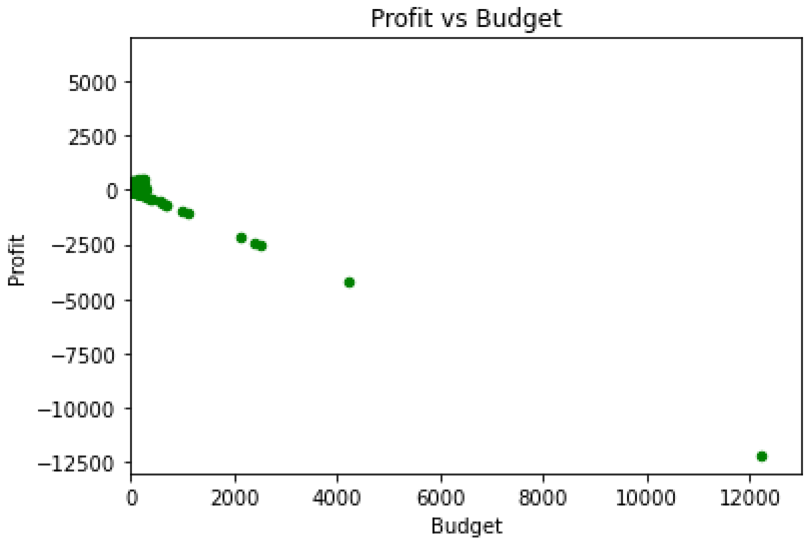
Out[17]:

	director_name	num_critic_for_reviews	gross	genres	actor_1
0	James Cameron	723.0	760.51	Action|Adventure|Fantasy|Sci-Fi	CCH F
29	Colin Trevorrow	644.0	652.18	Action|Adventure|Sci-Fi|Thriller	Bryc
26	James Cameron	315.0	658.67	Drama|Romance	Le D
3024	George Lucas	282.0	460.94	Action|Adventure|Fantasy|Sci-Fi	Harris
3080	Steven Spielberg	215.0	434.95	Family|Sci-Fi	Henry
...	...	...	...	...	
2334	Katsuhiro Ôtomo	105.0	0.41	Action|Adventure|Animation|Family|Sci-Fi|Thriller	H
2323	Hayao Miyazaki	174.0	2.30	Adventure|Animation|Fantasy	Minni
3005	Lajos Koltai	73.0	0.20	Drama|Romance|War	Marc
3859	Chan-wook Park	202.0	0.21	Crime|Drama	Min-
2988	Joon-ho Bong	363.0	2.20	Comedy|Drama|Horror|Sci-Fi	Doo

3891 rows × 14 columns

In [18]:

```
# Write code for profit vs budget plot here
movies.plot(kind='scatter',x='budget',y='profit',color='green')
plt.xlim([0,13000])
plt.ylim([-13000,7000])
plt.title("Profit vs Budget")
plt.xlabel("Budget")
plt.ylabel("Profit")
plt.show()
```



In [19]:

```
# Write your code to get the top 10 profiting movies here
top10 = movies.sort_values(by='profit',ascending=False)
top10.head(10)
```

Out[19]:

	director_name	num_critic_for_reviews	gross	genres	act
0	James Cameron	723.0	760.51	Action|Adventure|Fantasy|Sci-Fi	C
29	Colin Trevorrow	644.0	652.18	Action|Adventure|Sci-Fi|Thriller	E
26	James Cameron	315.0	658.67	Drama|Romance	
3024	George Lucas	282.0	460.94	Action|Adventure|Fantasy|Sci-Fi	H
3080	Steven Spielberg	215.0	434.95	Family|Sci-Fi	He
794	Joss Whedon	703.0	623.28	Action|Adventure|Sci-Fi	
17	Joss Whedon	703.0	623.28	Action|Adventure|Sci-Fi	
509	Roger Allers	186.0	422.78	Adventure|Animation|Drama|Family|Musical	
240	George Lucas	320.0	474.54	Action|Adventure|Fantasy|Sci-Fi	
66	Christopher Nolan	645.0	533.32	Action|Crime|Drama|Thriller	Cl

- ## Subtask 3.3: Drop duplicate values

After you found out the top 10 profiting movies, you might have noticed a duplicate value. So, it seems like the dataframe has duplicate values as well. Drop the duplicate values from the dataframe and repeat `Subtask 3.2`. Note that the same `movie_title` can be there in different languages.

In [20]:

```
# Write your code for dropping duplicate values here
movies=movies.drop_duplicates()
```

In [21]:

```
# Write code for repeating subtask 2 here
top10 = movies.sort_values(by='profit',ascending=False)
top10.head(10)
```

Out[21]:

	director_name	num_critic_for_reviews	gross	genres	act
0	James Cameron	723.0	760.51	Action|Adventure|Fantasy|Sci-Fi	C
29	Colin Trevorrow	644.0	652.18	Action|Adventure|Sci-Fi|Thriller	E
26	James Cameron	315.0	658.67	Drama|Romance	
3024	George Lucas	282.0	460.94	Action|Adventure|Fantasy|Sci-Fi	H
3080	Steven Spielberg	215.0	434.95	Family|Sci-Fi	He
794	Joss Whedon	703.0	623.28	Action|Adventure|Sci-Fi	
509	Roger Allers	186.0	422.78	Adventure|Animation|Drama|Family|Musical	
240	George Lucas	320.0	474.54	Action|Adventure|Fantasy|Sci-Fi	
66	Christopher Nolan	645.0	533.32	Action|Crime|Drama|Thriller	Cl
439	Gary Ross	673.0	408.00	Adventure|Drama|Sci-Fi|Thriller	

Checkpoint 2: You might spot two movies directed by `James Cameron` in the list.

- ## Subtask 3.4: Find IMDb Top 250

1. Create a new dataframe `IMDb_Top_250` and store the top 250 movies with the highest IMDb Rating (corresponding to the column: `imdb_score`). Also make sure that for all of these movies, the `num_voted_users` is greater than 25,000. Also add a `Rank` column containing the values 1 to 250 indicating the ranks of the corresponding films.
2. Extract all the movies in the `IMDb_Top_250` dataframe which are not in the English language and store them in a new dataframe named `Top_Foreign_Lang_Film` .

In [23]:

```
# Write your code for extracting the top 250 movies as per the IMDb score here. Make sure t
# and name that dataframe as 'IMDb_Top_250'
IMDb_Top_250 = movies[(movies.num_voted_users>25000)].sort_values(['imdb_score'], ascending
IMDb_Top_250['Rank']=np.array([i for i in range(1,251,1)])
IMDb_Top_250.set_index('Rank',inplace=True)
IMDb_Top_250
```

Out[23]:

	director_name	num_critic_for_reviews	gross	genres	actor_1_nan
Rank					
1	Frank Darabont	199.0	28.34	Crime|Drama	Morg Freem
2	Francis Ford Coppola	208.0	134.82	Crime|Drama	Al Pacii
3	Francis Ford Coppola	149.0	57.30	Crime|Drama	Robert De Ni
4	Christopher Nolan	645.0	533.32	Action|Crime|Drama|Thriller	Christian Ba
5	Peter Jackson	328.0	377.02	Action|Adventure|Drama|Fantasy	Orlando Bloo
...	...	...	...	...	
246	Cristian Mungiu	233.0	1.19	Drama	Anamai Marin
247	John Carpenter	318.0	47.00	Horror|Thriller	Jamie Le Cur
248	John Carpenter	318.0	47.00	Horror|Thriller	Jamie Le Cur
249	David O. Russell	410.0	93.57	Biography|Drama|Sport	Christian Ba
250	Michael Mann	209.0	28.97	Biography|Drama|Thriller	Al Pacii

250 rows × 14 columns

In [24]:

```
# Write your code to extract top foreign language films from 'IMDb_Top_250' here
Top_Foreign_Lang_Film = IMDb_Top_250[(IMDb_Top_250.language!='English')]
Top_Foreign_Lang_Film
```

Out[24]:

Rank	director_name	num_critic_for_reviews	gross	g
7	Sergio Leone	181.0	6.10	We
15	Fernando Meirelles	214.0	7.56	Crime|D
17	Akira Kurosawa	153.0	0.27	Action|Adventure|D
26	Hayao Miyazaki	246.0	10.05	Adventure|Animation|Family|Fa
43	Majid Majidi	46.0	0.93	Drama|F
46	Florian Henckel von Donnersmarck	215.0	11.28	Drama|T
47	S.S. Rajamouli	44.0	6.50	Action|Adventure|Drama|Fantas
49	Asghar Farhadi	354.0	7.10	Drama|M
52	Jean-Pierre Jeunet	242.0	33.20	Comedy|Ron
57	Chan-wook Park	305.0	2.18	Drama|Mystery|T
58	Hayao Miyazaki	174.0	2.30	Adventure|Animation|Fa
60	Wolfgang Petersen	96.0	11.43	Adventure|Drama|Thrille
68	Fritz Lang	260.0	0.03	Drama|
70	Thomas Vinterberg	349.0	0.61	D
74	Oliver Hirschbiegel	192.0	5.50	Biography|Drama|Histor
88	Denis Villeneuve	226.0	6.86	Drama|Myster
96	Juan José Campanella	262.0	20.17	Drama|Mystery|T
104	Guillermo del Toro	406.0	37.62	Drama|Fantas
107	Hayao Miyazaki	212.0	4.71	Adventure|Animation|Family|Fa
109	José Padilha	142.0	0.01	Action|Crime|Drama|T

	director_name	num_critic_for_reviews	gross	g
Rank				
112	Katsuhiro Ôtomo	150.0	0.44	Action|Animation|
123	Je-kyu Kang	86.0	1.11	Action|Dram
124	Thomas Vinterberg	98.0	1.65	[
138	Alejandro Amenábar	157.0	2.09	Biography|Drama|Ron
148	Alejandro G. Iñárritu	157.0	5.38	Drama|T
156	Ari Folman	231.0	2.28	Animation|Biography|Documentary|Drama|Histor
163	Vincent Paronnaud	242.0	4.44	Animation|Biography|Dram
166	Karan Johar	210.0	4.02	Adventure|Drama|T
182	Sergio Leone	122.0	3.50	Action|Drama|W
200	Walter Salles	71.0	5.60	[
206	Michael Haneke	447.0	0.23	Drama|Ron
208	Clint Eastwood	251.0	13.75	Drama|Histor
211	Ang Lee	287.0	128.07	Action|Drama|Ron
227	Yash Chopra	29.0	2.92	Drama|Musical|Ron
238	Fabián Bielinsky	94.0	1.22	Crime|Drama|T
241	Christophe Barratier	112.0	3.63	Drama|
242	Yimou Zhang	283.0	0.08	Action|Adventure|H
246	Cristian Mungiu	233.0	1.19	[

Checkpoint 3: Can you spot `Veer-Zaara` in the dataframe?

- ## Subtask 3.5: Find the best directors

1. Group the dataframe using the `director_name` column.
2. Find out the top 10 directors for whom the mean of `imdb_score` is the highest and store them in a new dataframe `top10director`. Incase of a tie in IMDb score between two directors, sort them alphabetically.

In [137]:

```
# Write your code for extracting the top 10 directors here
grp_by_director=movies.groupby('director_name')
top10director=round(grp_by_director.imdb_score.mean().sort_values(ascending = False)[0:10],
top10director
```

Out[137]:

```
director_name
Charles Chaplin         8.60
Tony Kaye               8.60
Alfred Hitchcock        8.50
Ron Fricke              8.50
Damien Chazelle         8.50
Majid Majidi            8.50
Sergio Leone            8.43
Christopher Nolan       8.43
S.S. Rajamouli          8.40
Marius A. Markevicius   8.40
Name: imdb_score, dtype: float64
```

Checkpoint 4: No surprises that `Damien Chazelle` (director of Whiplash and La La Land) is in this list.

- ## Subtask 3.6: Find popular genres

You might have noticed the `genres` column in the dataframe with all the genres of the movies seperated by a pipe (|). Out of all the movie genres, the first two are most significant for any film.

1. Extract the first two genres from the `genres` column and store them in two new columns: `genre_1` and `genre_2`. Some of the movies might have only one genre. In such cases, extract the single genre into both the columns, i.e. for such movies the `genre_2` will be the same as `genre_1`.
2. Group the dataframe using `genre_1` as the primary column and `genre_2` as the secondary column.
3. Find out the 5 most popular combo of genres by finding the mean of the gross values using the `gross` column and store them in a new dataframe named `PopGenre`.

In [144]:

```
# Write your code for extracting the first two genres of each movie here
movies["genre_1"] = movies["genres"].str.split("|").str.get(0)
movies["genre_2"] = movies["genres"].str.split("|").str.get(1)
movies["genre_2"] = movies["genre_2"].fillna(movies["genre_1"])
movies.head()
```

Out[144]:

	director_name	num_critic_for_reviews	gross	genres	actor_1_nam
1937	Frank Darabont	199.0	28.34	Crime|Drama	Morga Freema
3466	Francis Ford Coppola	208.0	134.82	Crime|Drama	Al Pacir
2837	Francis Ford Coppola	149.0	57.30	Crime|Drama	Robert De Nir
66	Christopher Nolan	645.0	533.32	Action|Crime|Drama|Thriller	Christian Ba
339	Peter Jackson	328.0	377.02	Action|Adventure|Drama|Fantasy	Orlando Bloor

In [145]:

```
# Write your code for grouping the dataframe here
grp_by_genre = movies.groupby(['genre_1','genre_2'])
```

In [146]:

```
# Write your code for getting the 5 most popular combo of genres here
PopGenre = grp_by_genre['gross'].mean().sort_values(ascending=False)[0:5]
PopGenre
```

Out[146]:

```
genre_1    genre_2
Family     Sci-Fi       434.950000
Adventure  Sci-Fi       228.628750
           Family       118.918824
           Animation    116.998462
Action     Adventure    109.595510
Name: gross, dtype: float64
```

Checkpoint 5: Well, as it turns out. `Family + Sci-Fi` is the most popular combo of genres out there!

- ### Subtask 3.7: Find the critic-favorite and audience-favorite actors

1. Create three new dataframes namely, `Meryl_Streep`, `Leo_Caprio`, and `Brad_Pitt` which contain the movies in which the actors: 'Meryl Streep', 'Leonardo DiCaprio', and 'Brad Pitt' are the lead actors. Use only the `actor_1_name` column for extraction. Also, make sure that you use the names 'Meryl Streep', 'Leonardo DiCaprio', and 'Brad Pitt' for the said extraction.
2. Append the rows of all these dataframes and store them in a new dataframe named `Combined`.
3. Group the combined dataframe using the `actor_1_name` column.
4. Find the mean of the `num_critic_for_reviews` and `num_users_for_review` and identify the actors which have the highest mean.
5. Observe the change in number of voted users over decades using a bar chart. Create a column called `decade` which represents the decade to which every movie belongs to. For example, the `title_year` year 1923, 1925 should be stored as 1920s. Sort the dataframe based on the column `decade`, group it by `decade` and find the sum of users voted in each decade. Store this in a new data frame called `df_by_decade`.

In [148]:

```
# Write your code for creating three new dataframes here
Meryl_Streep = movies[(movies.actor_1_name == 'Meryl Streep')]# Include all movies in which
Meryl_Streep
```

Out[148]:

	director_name	num_critic_for_reviews	gross	genres	actor_1_name
1925	Stephen Daldry	174.0	41.60	Drama|Romance	Meryl Streep
1575	Sydney Pollack	66.0	87.10	Biography|Drama|Romance	Meryl Streep
1674	Carl Franklin	64.0	23.21	Drama	Meryl Streep
1204	Nora Ephron	252.0	94.13	Biography|Drama|Romance	Meryl Streep
1408	David Frankel	208.0	124.73	Comedy|Drama|Romance	Meryl Streep
3135	Robert Altman	211.0	20.34	Comedy|Drama|Music	Meryl Streep
410	Nancy Meyers	187.0	112.70	Comedy|Drama|Romance	Meryl Streep
2781	Phyllida Lloyd	331.0	29.96	Biography|Drama|History	Meryl Streep
1618	David Frankel	234.0	63.54	Comedy|Drama|Romance	Meryl Streep
1106	Curtis Hanson	42.0	46.82	Action|Adventure|Crime|Thriller	Meryl Streep
1483	Robert Redford	227.0	15.00	Drama|Thriller|War	Meryl Streep

In [149]:

```
Leo_Caprio = movies[(movies.actor_1_name == 'Leonardo DiCaprio')] # Include all movies in w
Leo_Caprio
```

Out[149]:

	director_name	num_critic_for_reviews	gross	genres	actor_1_nar
97	Christopher Nolan	642.0	292.57	Action|Adventure|Sci-Fi|Thriller	Leonar DiCap
361	Martin Scorsese	352.0	132.37	Crime|Drama|Thriller	Leonar DiCap
296	Quentin Tarantino	765.0	162.80	Drama|Western	Leonar DiCap
308	Martin Scorsese	606.0	116.87	Biography|Comedy|Crime|Drama	Leonar DiCap
179	Alejandro G. Iñárritu	556.0	183.64	Adventure|Drama|Thriller|Western	Leonar DiCap
452	Martin Scorsese	490.0	127.97	Mystery|Thriller	Leonar DiCap
911	Steven Spielberg	194.0	164.44	Biography|Crime|Drama	Leonar DiCap
307	Edward Zwick	166.0	57.37	Adventure|Drama|Thriller	Leonar DiCap
26	James Cameron	315.0	658.67	Drama|Romance	Leonar DiCap
326	Martin Scorsese	233.0	77.68	Crime|Drama	Leonar DiCap
257	Martin Scorsese	267.0	102.61	Biography|Drama	Leonar DiCap
1114	Sam Mendes	323.0	22.88	Drama|Romance	Leonar DiCap
3476	Baz Luhrmann	490.0	144.81	Drama|Romance	Leonar DiCap
50	Baz Luhrmann	490.0	144.81	Drama|Romance	Leonar DiCap
641	Ridley Scott	238.0	39.38	Action|Drama|Thriller	Leonar DiCap
2757	Baz Luhrmann	106.0	46.34	Drama|Romance	Leonar DiCap
2067	Jerry Zaks	45.0	12.78	Drama	Leonar DiCap
990	Danny Boyle	118.0	39.78	Adventure|Drama|Thriller	Leonar DiCap
1453	Clint Eastwood	392.0	37.30	Biography|Crime|Drama	Leonar DiCap
1560	Sam Raimi	63.0	18.64	Action|Thriller|Western	Leonar DiCap
1422	Randall Wallace	83.0	56.88	Action|Adventure	Leonar DiCap

In [150]:

```
Brad_Pitt = movies[(movies.actor_1_name == 'Brad Pitt')] # Include all movies in which Brad
Brad_Pitt
```

Out[150]:

	director_name	num_critic_for_reviews	gross	ge
683	David Fincher	315.0	37.02	Dı
2898	Tony Scott	122.0	12.28	Action|Crime|Drama|Romance|Tł
101	David Fincher	362.0	127.49	Drama|Fantasy|Rom
400	Steven Soderbergh	186.0	183.41	Crime|Tł
470	David Ayer	406.0	85.71	Action|Drama
940	Neil Jordan	120.0	105.26	Drama|Fantasy|H
2204	Alejandro G. Iñárritu	285.0	34.30	Dı
1722	Andrew Dominik	273.0	3.90	Biography|Crime|Drama|History|We
147	Wolfgang Petersen	220.0	133.23	Adveı
382	Tony Scott	142.0	0.03	Action|Crime|Tł
611	Jean-Jacques Annaud	76.0	37.90	Adventure|Biography|Drama|History
1490	Terrence Malick	584.0	13.30	Drama|Far
792	Patrick Gilmore	98.0	26.29	Adventure|Animation|Comedy|Drama|Family|Fan
255	Doug Liman	233.0	186.34	Action|Comedy|Crime|Romance|Tł
254	Steven Soderbergh	198.0	125.53	Crime|Tł
2682	Andrew Dominik	414.0	14.94	Crime|Tł
2333	Angelina Jolie Pitt	131.0	0.53	Drama|Rom

In [151]:

```
# Write your code for combining the three dataframes here
Combined = Meryl_Streep.append(Leo_Caprio).append(Brad_Pitt)
Combined
```

Out[151]:

	director_name	num_critic_for_reviews	gross	genres	actor_1_
1925	Stephen Daldry	174.0	41.60	Drama|Romance	Meryl
1575	Sydney Pollack	66.0	87.10	Biography|Drama|Romance	Meryl
1674	Carl Franklin	64.0	23.21	Drama	Meryl
1204	Nora Ephron	252.0	94.13	Biography|Drama|Romance	Meryl
1408	David Frankel	208.0	124.73	Comedy|Drama|Romance	Meryl
3135	Robert Altman	211.0	20.34	Comedy|Drama|Music	Meryl
410	Nancy Meyers	187.0	112.70	Comedy|Drama|Romance	Meryl

In [156]:

```
# Write your code for grouping the combined dataframe here
grp_combined= Combined.groupby('actor_1_name')
```

In [162]:

```
# Write the code for finding the mean of critic reviews and audience reviews here
grp_combined.num_critic_for_reviews.mean().sort_values(ascending=False)
```

Out[162]:

```
actor_1_name
Leonardo DiCaprio    330.190476
Brad Pitt            245.000000
Meryl Streep         181.454545
Name: num_critic_for_reviews, dtype: float64
```

Checkpoint 6: `Leonardo` has aced both the lists!

In [181]:

```
# Write the code for calculating decade here
movies['decade'] = ((movies['title_year']//10)*10).astype(np.int64)
movies['decade']=movies['decade'].astype(str)+'s'
movies
```

Out[181]:

	director_name	num_critic_for_reviews	gross	genre
4958	Harry F. Millarde	1.0	3.00	Crime|Dram
4812	Harry Beaumont	36.0	2.81	Musical|Romanc
2734	Fritz Lang	260.0	0.03	Drama|Sci-F
4157	Victor Fleming	213.0	22.20	Adventure|Family|Fantasy|Musica
3970	Victor Fleming	157.0	198.66	Drama|History|Romance|Wa
...	...	...	...	.
3010	Tim Johnson	165.0	177.34	Adventure|Animation|Comedy|Family|Fantasy|Sc F
2194	Tom McCarthy	474.0	44.99	Biography|Crime|Drama|Histor
4499	Anna Muylaert	111.0	0.38	Comedy|Dram
2677	Derek Cianfrance	417.0	21.38	Crime|Drama|Thrille
1367	J Blakeson	194.0	34.91	Action|Adventure|Sci-Fi|Thrille

3856 rows × 17 columns

In [182]:

```
# Write your code for creating the data frame df_by_decade here
df_by_decade=movies.groupby(['decade'])['num_voted_users'].sum()
df_by_decade
```

Out[182]:

```
decade
1920s       116392
1930s       804839
1940s       230838
1950s       678336
1960s      2983442
1970s      8524102
1980s     19987476
1990s     69735679
2000s    170908676
2010s    120640994
Name: num_voted_users, dtype: int64
```

In [231]:

```
# Write your code for plotting number of voted users vs decade
df_by_decade.plot.bar()
plt.yscale('log') # to convert axes to logarithmic scale
plt.xlabel("Decade")
plt.ylabel("Number of voted users")
```

Out[231]:

```
Text(0, 0.5, 'Number of voted users')
```