In [1]:

```
# Supress Warnings

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

In [2]:

```
# Import the numpy and pandas packages

import numpy as np
import pandas as pd
```

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 [3]:

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

Out[3]:

	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 [4]:

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

Out[4]:

```
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 [5]:

```
movies.shape
```

Out[5]:

```
(5043, 28)
```

In [6]:

```
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 [7]:

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

Out[7]:

```
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 [8]:

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

Out[8]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_fa
0	False	False	False	False	False	
1	False	False	False	False	False	
2	False	False	False	False	False	
3	False	False	False	False	False	
4	True	False	True	True	False	
...	...	...	...	...	...	
5038	False	False	False	False	False	
5039	False	True	False	False	True	
5040	False	False	False	False	False	
5041	False	False	False	False	False	
5042	False	False	False	False	False	

5043 rows × 28 columns

In [9]:

```
# Write your code for column-wise null percentages here
missing_percent=round((movies.isnull().sum()*100)/len(movies),3)
missing_percent
```

Out[9]:

```
color                        0.377
director_name                2.062
num_critic_for_reviews       0.991
duration                     0.297
director_facebook_likes      2.062
actor_3_facebook_likes       0.456
actor_2_name                 0.258
actor_1_facebook_likes       0.139
gross                       17.529
genres                       0.000
actor_1_name                 0.139
movie_title                  0.000
num_voted_users              0.000
cast_total_facebook_likes    0.000
actor_3_name                 0.456
facenumber_in_poster         0.258
plot_keywords                3.034
movie_imdb_link              0.000
num_user_for_reviews         0.397
language                     0.238
country                      0.099
content_rating               6.008
budget                       9.756
title_year                   2.142
actor_2_facebook_likes       0.258
imdb_score                   0.000
aspect_ratio                 6.524
movie_facebook_likes         0.000
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 [10]:

```
# 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_
                      'actor_2_name', 'cast_total_facebook_likes', 'actor_3_name', 'duration',
                      'content_rating', 'country', 'movie_imdb_link', 'aspect_ratio', 'plot_ke
movies.shape
```

Out[10]:

```
(5043, 13)
```

- **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 [11]:

```
# Write your code for dropping the rows here
missing_percent=round((movies.isnull().sum()*100)/len(movies),3)
missing_percent
```

Out[11]:

```
director_name              2.062
num_critic_for_reviews     0.991
gross                     17.529
genres                     0.000
actor_1_name               0.139
movie_title                0.000
num_voted_users            0.000
num_user_for_reviews       0.397
language                   0.238
budget                     9.756
title_year                 2.142
imdb_score                 0.000
movie_facebook_likes       0.000
dtype: float64
```

In [12]:

```
## Removing NaN values from 'Gross' columns
a=[movies['gross'][4]]
movies=movies[~movies['gross'].isin(a)]
movies
```

Out[12]:

	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
...	...	...	...	...	
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	
5041	Daniel Hsia	14.0	10443.0	Comedy|Drama|Romance	
5042	Jon Gunn	43.0	85222.0	Documentary	

4159 rows × 13 columns

In [13]:

```
## Removing NaN values from 'Gross' columns
b=[movies['budget'][5041]]
movies=movies[~movies['budget'].isin(b)]
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	Sh
5034	Neill Dela Llana	35.0	70071.0	Thriller	Ia
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 2.4: 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 [14]:

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

Out[14]:

```
array(['English', 'Mandarin', 'Aboriginal', 'Spanish', 'French',
       'Filipino', 'Maya', 'Kazakh', 'Telugu', 'Cantonese', 'Japanese',
       'Aramaic', 'Italian', 'Dutch', 'Dari', 'German', 'Mongolian',
       'Thai', 'Bosnian', 'Korean', 'Hungarian', 'Hindi', 'Icelandic',
       'Danish', 'Portuguese', 'Norwegian', 'Czech', 'Russian', 'None',
       'Zulu', 'Hebrew', 'Dzongkha', 'Arabic', 'Vietnamese', 'Indonesian',
       'Romanian', 'Persian', 'Swedish'], dtype=object)
```

- ### 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 [15]:

```
# Write your code for checking number of retained rows here
missing_percent=round((movies.isnull().sum()*100)/len(movies),3)
missing_percent
```

Out[15]:

```
director_name              0.000
num_critic_for_reviews     0.026
gross                      0.000
genres                     0.000
actor_1_name               0.077
movie_title                0.000
num_voted_users            0.000
num_user_for_reviews       0.000
language                   0.000
budget                     0.000
title_year                 0.000
imdb_score                 0.000
movie_facebook_likes       0.000
dtype: float64
```

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 [16]:

```
# Write your code for unit conversion here
for x in movies['gross'].index:
    movies['gross'][x]=movies['gross'][x]/1000000
```

In [17]:

```
for x in movies['budget'].index:
    movies['budget'][x]=movies['budget'][x]/1000000
```

In [18]:

```
movies
```

Out[18]:

	director_name	num_critic_for_reviews	gross	genres	acto
0	James Cameron	723.0	760.505847	Action|Adventure|Fantasy|Sci-Fi	CC
1	Gore Verbinski	302.0	309.404152	Action|Adventure|Fantasy	Jo
2	Sam Mendes	602.0	200.074175	Action|Adventure|Thriller	
3	Christopher Nolan	813.0	448.130642	Action|Thriller	
5	Andrew Stanton	462.0	73.058679	Action|Adventure|Sci-Fi	D
...	...	...	...	...	
5033	Shane Carruth	143.0	0.424760	Drama|Sci-Fi|Thriller	Sha
5034	Neill Dela Llana	35.0	0.070071	Thriller	Ian
5035	Robert Rodriguez	56.0	2.040920	Action|Crime|Drama|Romance|Thriller	
5037	Edward Burns	14.0	0.004584	Comedy|Drama	I
5042	Jon Gunn	43.0	0.085222	Documentary	J

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 [19]:

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

Out[19]:

	director_name	num_critic_for_reviews	gross	genres	acto
0	James Cameron	723.0	760.505847	Action|Adventure|Fantasy|Sci-Fi	CC
1	Gore Verbinski	302.0	309.404152	Action|Adventure|Fantasy	Jo
2	Sam Mendes	602.0	200.074175	Action|Adventure|Thriller	
3	Christopher Nolan	813.0	448.130642	Action|Thriller	
5	Andrew Stanton	462.0	73.058679	Action|Adventure|Sci-Fi	Da
...	...	...	...	...	
5033	Shane Carruth	143.0	0.424760	Drama|Sci-Fi|Thriller	Sha
5034	Neill Dela Llana	35.0	0.070071	Thriller	Ian
5035	Robert Rodriguez	56.0	2.040920	Action|Crime|Drama|Romance|Thriller	
5037	Edward Burns	14.0	0.004584	Comedy|Drama	I
5042	Jon Gunn	43.0	0.085222	Documentary	J

3891 rows × 14 columns

In [20]:

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

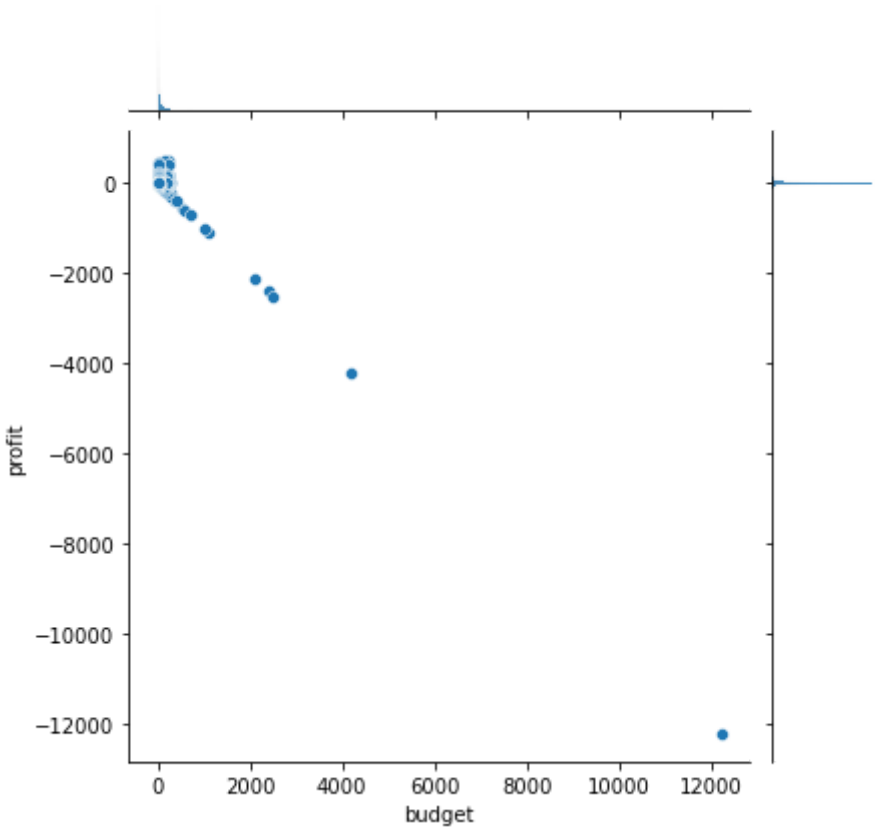
Out[20]:

	director_name	num_critic_for_reviews	gross	genres	ac
0	James Cameron	723.0	760.505847	Action|Adventure|Fantasy|Sci-Fi	C
29	Colin Trevorrow	644.0	652.177271	Action|Adventure|Sci-Fi|Thriller	
26	James Cameron	315.0	658.672302	Drama|Romance	
3024	George Lucas	282.0	460.935665	Action|Adventure|Fantasy|Sci-Fi	H
3080	Steven Spielberg	215.0	434.949459	Family|Sci-Fi	H
...	...	...	...	...	
2334	Katsuhiro Ôtomo	105.0	0.410388	Action|Adventure|Animation|Family|Sci-Fi|Thriller	
2323	Hayao Miyazaki	174.0	2.298191	Adventure|Animation|Fantasy	I
3005	Lajos Koltai	73.0	0.195888	Drama|Romance|War	I
3859	Chan-wook Park	202.0	0.211667	Crime|Drama	
2988	Joon-ho Bong	363.0	2.201412	Comedy|Drama|Horror|Sci-Fi	

3891 rows × 14 columns

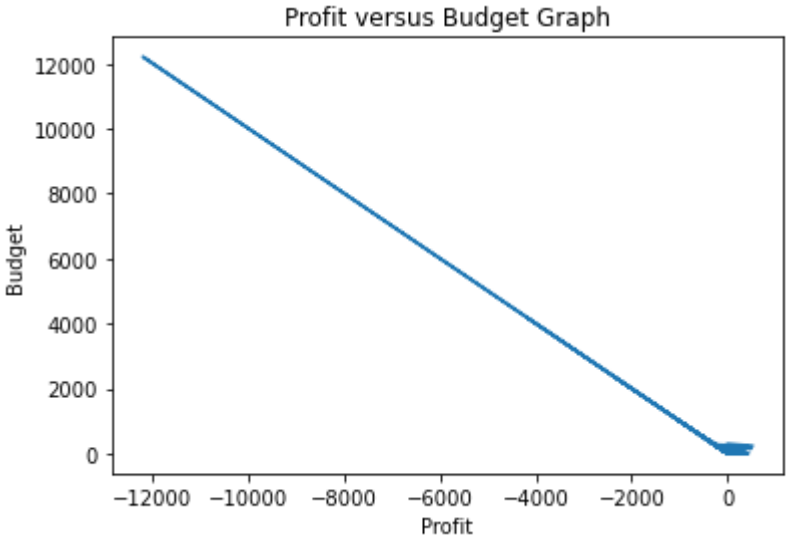
In [21]:

```
# Write code for profit vs budget plot here
import seaborn as sns
fig = sns.jointplot(x='budget', y='profit', data=movies)
```



In [22]:

```
import matplotlib.pyplot as plt
x=movies['profit']
y=movies['budget']
plt.plot(x,y)
plt.xlabel('Profit')
plt.ylabel('Budget')
plt.title('Profit versus Budget Graph')
plt.show()
```



In [23]:

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

Out[23]:

	director_name	num_critic_for_reviews	gross	genres
0	James Cameron	723.0	760.505847	Action|Adventure|Fantasy|Sci-Fi
29	Colin Trevorrow	644.0	652.177271	Action|Adventure|Sci-Fi|Thriller
26	James Cameron	315.0	658.672302	Drama|Romance
3024	George Lucas	282.0	460.935665	Action|Adventure|Fantasy|Sci-Fi
3080	Steven Spielberg	215.0	434.949459	Family|Sci-Fi
794	Joss Whedon	703.0	623.279547	Action|Adventure|Sci-Fi
17	Joss Whedon	703.0	623.279547	Action|Adventure|Sci-Fi
509	Roger Allers	186.0	422.783777	Adventure|Animation|Drama|Family|Musical
240	George Lucas	320.0	474.544677	Action|Adventure|Fantasy|Sci-Fi
66	Christopher Nolan	645.0	533.316061	Action|Crime|Drama|Thriller

- ## Subtask 3.3: Drop duplicate values

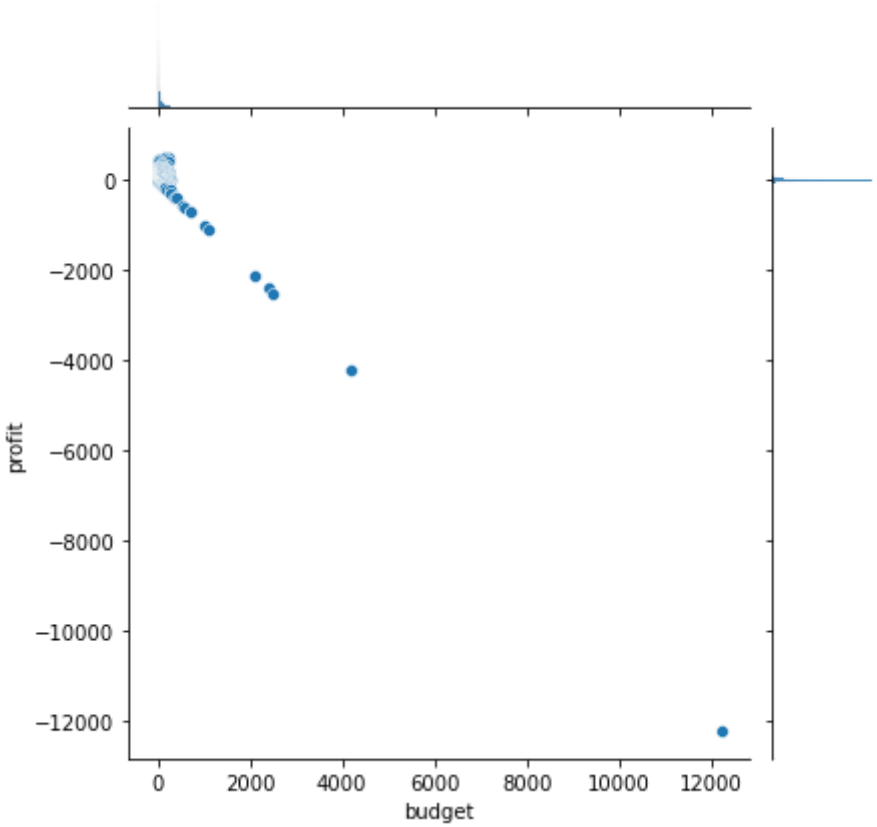
After you found out the top 10 profiting movies, you might have noticed a duplicate value. So, it seems like the dataframe has duplicate values as well. Drop the duplicate values from the dataframe and repeat `Subtask 3.2`. Note that the same `movie_title` can be there in different languages.

In [24]:

```
# Write your code for dropping duplicate values here
movies = movies.drop_duplicates(subset=['movie_title','language'], keep='first')
movies['profit']= movies['gross']-movies['budget']
movies=movies.sort_values(by=['profit'], ascending=False)
```

In [25]:

```
# Write code for repeating subtask 2 here
fig = sns.jointplot(x='budget', y='profit', data=movies)
```



In [26]:

```
top10=movies.head(10)
top10
```

Out[26]:

	director_name	num_critic_for_reviews	gross	genres
0	James Cameron	723.0	760.505847	Action|Adventure|Fantasy|Sci-Fi
29	Colin Trevorrow	644.0	652.177271	Action|Adventure|Sci-Fi|Thriller
26	James Cameron	315.0	658.672302	Drama|Romance
3024	George Lucas	282.0	460.935665	Action|Adventure|Fantasy|Sci-Fi
3080	Steven Spielberg	215.0	434.949459	Family|Sci-Fi
17	Joss Whedon	703.0	623.279547	Action|Adventure|Sci-Fi
509	Roger Allers	186.0	422.783777	Adventure|Animation|Drama|Family|Musical
240	George Lucas	320.0	474.544677	Action|Adventure|Fantasy|Sci-Fi
66	Christopher Nolan	645.0	533.316061	Action|Crime|Drama|Thriller
439	Gary Ross	673.0	407.999255	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 [27]:

```
# 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(by=['imdb_score'], ascen
IMDb_Top_250['Rank'] = range(1, 1+len(IMDb_Top_250))
IMDb_Top_250
```

Out[27]:

	director_name	num_critic_for_reviews	gross	genres	actor_1_nam
1937	Frank Darabont	199.0	28.341469	Crime|Drama	Morga Freema
3466	Francis Ford Coppola	208.0	134.821952	Crime|Drama	Al Pacin
2837	Francis Ford Coppola	149.0	57.300000	Crime|Drama	Robert De Nir
66	Christopher Nolan	645.0	533.316061	Action|Crime|Drama|Thriller	Christian Bal
4498	Sergio Leone	181.0	6.100000	Western	Clint Eastwoo
...	...	...	...	...	.
845	Paul Greengrass	491.0	107.100855	Biography|Drama|Thriller	Tom Hank
3680	Spike Lee	103.0	27.545445	Drama	Ruby De
3080	Steven Spielberg	215.0	434.949459	Family|Sci-Fi	Henry Thoma
4082	Richard Linklater	405.0	8.114507	Drama|Romance	Seamu Davey Fitzpatric
23	Peter Jackson	509.0	258.355354	Adventure|Fantasy	Aidan Turne

250 rows × 15 columns

In [28]:

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

Out[28]:

	director_name	num_critic_for_reviews	gross	
4498	Sergio Leone	181.0	6.100000	
4747	Akira Kurosawa	153.0	0.269061	Action|Adventu
4029	Fernando Meirelles	214.0	7.563397	Crim
2373	Hayao Miyazaki	246.0	10.049886	Adventure|Animation|Family
4921	Majid Majidi	46.0	0.925402	Dram
4259	Florian Henckel von Donnersmarck	215.0	11.284657	Dram
2323	Hayao Miyazaki	174.0	2.298191	Adventure|Animatior
4105	Chan-wook Park	305.0	2.181290	Drama|Myster
2970	Wolfgang Petersen	96.0	11.433134	Adventure|Drama|Th
4659	Asghar Farhadi	354.0	7.098492	Drama
1298	Jean-Pierre Jeunet	242.0	33.201661	Comedy|F
1329	S.S. Rajamouli	44.0	6.498000	Action|Adventure|Drama|Fan
2829	Oliver Hirschbiegel	192.0	5.501940	Biography|Drama|His
4033	Thomas Vinterberg	349.0	0.610968	
2734	Fritz Lang	260.0	0.026435	Dra
2047	Hayao Miyazaki	212.0	4.710455	Adventure|Animation|Family
3550	Denis Villeneuve	226.0	6.857096	Drama|Mys
2551	Guillermo del Toro	406.0	37.623143	Drama|Fan
4000	Juan José Campanella	262.0	20.167424	Drama|Myster
3553	José Padilha	142.0	0.008060	Action|Crime|Dram
3423	Katsuhiro Ôtomo	150.0	0.439162	Action|Animati

	director_name	num_critic_for_reviews	gross	
4267	Alejandro G. Iñárritu	157.0	5.383834	Dram
2830	Alejandro Amenábar	157.0	2.086345	Biography|Drama|F
2914	Je-kyu Kang	86.0	1.110186	Action|Dr
4461	Thomas Vinterberg	98.0	1.647780	
3344	Karan Johar	210.0	4.018695	Adventure|Dram
3456	Vincent Paronnaud	242.0	4.443403	Animation|Biography|Dr
4897	Sergio Leone	122.0	3.500000	Action|Drama
4284	Ari Folman	231.0	2.283276	Animation|Biography|Documentary|Drama|His
4144	Walter Salles	71.0	5.595428	
3264	Michael Haneke	447.0	0.225377	Drama|F
2863	Clint Eastwood	251.0	13.753931	Drama|His
3677	Christophe Barratier	112.0	3.629758	Dran
4640	Cristian Mungiu	233.0	1.185783	
1171	Yimou Zhang	283.0	0.084961	Action|Adventur
4415	Fabián Bielinsky	94.0	1.221261	Crime|Dram

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 [29]:

```
# Write your code for extracting the top 10 directors here
top10director=movies.groupby('director_name',as_index=False)["imdb_score"].mean().sort_valu
top10director
```

Out[29]:

	director_name	imdb_score
216	Charles Chaplin	8.600000
1675	Tony Kaye	8.600000
45	Alfred Hitchcock	8.500000
302	Damien Chazelle	8.500000
1017	Majid Majidi	8.500000
1440	Ron Fricke	8.500000
1498	Sergio Leone	8.433333
260	Christopher Nolan	8.425000
103	Asghar Farhadi	8.400000
1035	Marius A. Markevicius	8.400000

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 [30]:

```
# Write your code for extracting the first two genres of each movie here
col2 = movies['genres'].apply(lambda x: pd.Series(x.split('|'))).rename(columns={0:'genre1'
col2=col2.drop(['2','3','4','5','6','7'], axis=1)
movies.insert(3,'genre_1',col2.genre1)
movies.insert(4,'genre_2',col2.genre2)
movies=movies.drop('genres', axis=1)
movies
```

Out[30]:

	director_name	num_critic_for_reviews	gross	genre_1	genre_2	actor_1_name	m
0	James Cameron	723.0	760.505847	Action	Adventure	CCH Pounder	
29	Colin Trevorrow	644.0	652.177271	Action	Adventure	Bryce Dallas Howard	
26	James Cameron	315.0	658.672302	Drama	Romance	Leonardo DiCaprio	
3024	George Lucas	282.0	460.935665	Action	Adventure	Harrison Ford	
3080	Steven Spielberg	215.0	434.949459	Family	Sci-Fi	Henry Thomas	
...	...	...	...	...	...	...	
2334	Katsuhiro Ôtomo	105.0	0.410388	Action	Adventure	William Hootkins	
2323	Hayao Miyazaki	174.0	2.298191	Adventure	Animation	Minnie Driver	
3005	Lajos Koltai	73.0	0.195888	Drama	Romance	Marcell Nagy	
3859	Chan-wook Park	202.0	0.211667	Crime	Drama	Min-sik Choi	
2988	Joon-ho Bong	363.0	2.201412	Comedy	Drama	Doona Bae	

3790 rows × 15 columns

In [31]:

```
movies_by_segment =movies.groupby(['genre_1','genre_2'],as_index=False)["gross"].mean()
movies_by_segment # Write your code for grouping the dataframe here
```

Out[31]:

	genre_1	genre_2	gross
0	Action	Action	59.520907
1	Action	Adventure	107.880651
2	Action	Animation	92.680515
3	Action	Biography	44.355422
4	Action	Comedy	52.678671
...	...	...	...
98	Romance	Sci-Fi	62.453315
99	Sci-Fi	Sci-Fi	0.018195
100	Sci-Fi	Thriller	29.793790
101	Thriller	Thriller	0.040513
102	Western	Western	15.914589

103 rows × 3 columns

In [32]:

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

Out[32]:

	genre_1	genre_2	gross
83	Family	Sci-Fi	434.949459
28	Adventure	Sci-Fi	228.627758
18	Adventure	Animation	115.949069
24	Adventure	Family	110.942298
1	Action	Adventure	107.880651

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 [33]:

```
# Write your code for creating three new dataframes here

Meryl_Streep =movies.loc[movies['actor_1_name'] == "Meryl Streep",:] # Include all movies i
```

In [34]:

```
Leo_Caprio =movies.loc[movies['actor_1_name'] == "Leonardo DiCaprio",:] # Include all movie
```

In [35]:

```
Brad_Pitt =movies.loc[movies['actor_1_name'] == "Brad Pitt",:] # Include all movies in whic
```

In [36]:

```
# Write your code for combining the three dataframes here
Combined=pd.concat([Meryl_Streep, Leo_Caprio,Brad_Pitt], axis = 0)
```

In [37]:

```
# Write your code for grouping the combined dataframe here
Combined.groupby('actor_1_name').apply(lambda x:x)
```

Out[37]:

	director_name	num_critic_for_reviews	gross	genre_1	genre_2	actor_1_name	movie_title
1408	David Frankel	208.0	124.732962	Comedy	Drama	Meryl Streep	The Devil Wears Prada
1575	Sydney Pollack	66.0	87.100000	Biography	Drama	Meryl Streep	Out of Africa
1204	Nora Ephron	252.0	94.125426	Biography	Drama	Meryl Streep	Julie & Julia
1618	David Frankel	234.0	63.536011	Comedy	Drama	Meryl Streep	Hope Springs
410	Nancy Meyers	187.0	112.703470	Comedy	Drama	Meryl Streep	It's Complicated
2781	Phyllida Lloyd	331.0	29.959436	Biography	Drama	Meryl Streep	The Iron Lady
1925	Stephen Daldry	174.0	41.597830	Drama	Romance	Meryl Streep	The Hours
3135	Robert Altman	211.0	20.338609	Comedy	Drama	Meryl Streep	A Prairie Home

In [38]:

```
# Write the code for finding the mean of critic reviews and audience reviews here
Combined.groupby('actor_1_name').agg({"num_critic_for_reviews":"mean","num_user_for_reviews
```

Out[38]:

actor_1_name	num_critic_for_reviews	num_user_for_reviews
Brad Pitt	245.000000	4.975285e+49
Leonardo DiCaprio	322.200000	1.264140e+64
Meryl Streep	181.454545	5.738184e+30

Checkpoint 6: Leonardo has aced both the lists!

In [39]:

```
# Write the code for calculating decade here
df_by_decade=movies.copy(deep=True)
df_by_decade['decade']=df_by_decade['title_year'].apply(lambda x:10*(int(x/10)))
```

In [40]:

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

In [41]:

```
# Write your code for plotting number of voted users vs decade
ax=sns.barplot(x='decade',y='num_voted_users',data=df_by_decade)
plt.yscale('log')
plt.ylabel("num_voted_users")
plt.xlabel("decade")
plt.show()
```