IMDb Movie Assignment

You have the data for the 100 top-rated movies from the past decade along with various pieces of information about the movie, its actors, and the voters who have rated these movies online. In this assignment, you will try to find some interesting insights into these movies and their voters, using Python.

In [1]:
```
# Filtering out the warnings
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Importing the required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Task 1: Reading the data

- ### Subtask 1.1: Read the Movies Data.

Read the movies data file provided and store it in a dataframe `movies`. The csv file provided by Decodr Technologies stored in the running folder and read.csv executed

In [3]:
```
# Read the csv file using 'read_csv'.
#Subtask 1.1: Read the Movies Data.
#Read the movies data file provided and store it in a dataframe movies.

movies=pd.read_csv('Movie+Assignment+Data.csv')
movies.head()
```

Out[3]:

	Title	title_year	budget	Gross	actor_1_name	actor_2_name	actor_3_name	actor_1_fa
0	La La Land	2016	30000000	151101803	Ryan Gosling	Emma Stone	Amiée Conn	
1	Zootopia	2016	150000000	341268248	Ginnifer Goodwin	Jason Bateman	Idris Elba	
2	Lion	2016	12000000	51738905	Dev Patel	Nicole Kidman	Rooney Mara	
3	Arrival	2016	47000000	100546139	Amy Adams	Jeremy Renner	Forest Whitaker	
4	Manchester by the Sea	2016	9000000	47695371	Casey Affleck	Michelle Williams	Kyle Chandler	

5 rows × 62 columns

- ### Subtask 1.2: Inspect the Dataframe

Inspect the dataframe for dimensions, null-values, and summary of different numeric columns.

In [5]:
```
# Check the number of rows and columns in the dataframe
movies.shape
```

Out[5]:
```
(100, 62)
```

In [6]:
```
movies.columns
```

Out[6]:
```
Index(['Title', 'title_year', 'budget', 'Gross', 'actor_1_name',
       'actor_2_name', 'actor_3_name', 'actor_1_facebook_likes',
       'actor_2_facebook_likes', 'actor_3_facebook_likes', 'IMDb_rating',
       'genre_1', 'genre_2', 'genre_3', 'MetaCritic', 'Runtime', 'CVotes10',
       'CVotes09', 'CVotes08', 'CVotes07', 'CVotes06', 'CVotes05', 'CVotes04',
       'CVotes03', 'CVotes02', 'CVotes01', 'CVotesMale', 'CVotesFemale',
       'CVotesU18', 'CVotesU18M', 'CVotesU18F', 'CVotes1829', 'CVotes1829M',
       'CVotes1829F', 'CVotes3044', 'CVotes3044M', 'CVotes3044F', 'CVotes45A',
       'CVotes45AM', 'CVotes45AF', 'CVotes1000', 'CVotesUS', 'CVotesnUS',
       'VotesM', 'VotesF', 'VotesU18', 'VotesU18M', 'VotesU18F', 'Votes1829',
       'Votes1829M', 'Votes1829F', 'Votes3044', 'Votes3044M', 'Votes3044F',
       'Votes45A', 'Votes45AM', 'Votes45AF', 'Votes1000', 'VotesUS',
       'VotesnUS', 'content_rating', 'Country'],
      dtype='object')
```

In [7]:
```
# Check the column-wise info of the dataframe
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 62 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Title                   100 non-null    object
 1   title_year              100 non-null    int64
 2   budget                  100 non-null    int64
 3   Gross                   100 non-null    int64
 4   actor_1_name            100 non-null    object
 5   actor_2_name            100 non-null    object
 6   actor_3_name            100 non-null    object
 7   actor_1_facebook_likes  100 non-null    int64
 8   actor_2_facebook_likes  99 non-null     float64
 9   actor_3_facebook_likes  98 non-null     float64
 10  IMDb_rating             100 non-null    float64
 11  genre_1                 100 non-null    object
 12  genre_2                 97 non-null     object
 13  genre_3                 74 non-null     object
 14  MetaCritic              95 non-null     float64
 15  Runtime                 100 non-null    int64
 16  CVotes10                100 non-null    int64
 17  CVotes09                100 non-null    int64
 18  CVotes08                100 non-null    int64
 19  CVotes07                100 non-null    int64
 20  CVotes06                100 non-null    int64
 21  CVotes05                100 non-null    int64
 22  CVotes04                100 non-null    int64
 23  CVotes03                100 non-null    int64
 24  CVotes02                100 non-null    int64
```

```
 25  CVotes01                100 non-null    int64
 26  CVotesMale              100 non-null    int64
 27  CVotesFemale            100 non-null    int64
 28  CVotesU18               100 non-null    int64
 29  CVotesU18M              100 non-null    int64
 30  CVotesU18F              100 non-null    int64
 31  CVotes1829              100 non-null    int64
 32  CVotes1829M             100 non-null    int64
 33  CVotes1829F             100 non-null    int64
 34  CVotes3044              100 non-null    int64
 35  CVotes3044M             100 non-null    int64
 36  CVotes3044F             100 non-null    int64
 37  CVotes45A               100 non-null    int64
 38  CVotes45AM              100 non-null    int64
 39  CVotes45AF              100 non-null    int64
 40  CVotes1000              100 non-null    int64
 41  CVotesUS                100 non-null    int64
 42  CVotesnUS               100 non-null    int64
 43  VotesM                  100 non-null    float64
 44  VotesF                  100 non-null    float64
 45  VotesU18                100 non-null    float64
 46  VotesU18M               100 non-null    float64
 47  VotesU18F               100 non-null    float64
 48  Votes1829               100 non-null    float64
 49  Votes1829M              100 non-null    float64
 50  Votes1829F              100 non-null    float64
 51  Votes3044               100 non-null    float64
 52  Votes3044M              100 non-null    float64
 53  Votes3044F              100 non-null    float64
 54  Votes45A                100 non-null    float64
 55  Votes45AM               100 non-null    float64
 56  Votes45AF               100 non-null    float64
 57  Votes1000               100 non-null    float64
 58  VotesUS                 100 non-null    float64
 59  VotesnUS                100 non-null    float64
 60  content_rating          100 non-null    object
 61  Country                 100 non-null    object
dtypes: float64(21), int64(32), object(9)
memory usage: 48.6+ KB
```

In [8]:

```
# Check the summary for the numeric columns
movies.describe()
```

Out[8]:

	title_year	budget	Gross	actor_1_facebook_likes	actor_2_facebook_likes	actor_3_
count	100.000000	1.000000e+02	1.000000e+02	100.000000	99.000000	
mean	2012.820000	7.838400e+07	1.468679e+08	13407.270000	7377.303030	
std	1.919491	7.445295e+07	1.454004e+08	10649.037862	13471.568216	
min	2010.000000	3.000000e+06	2.238380e+05	39.000000	12.000000	
25%	2011.000000	1.575000e+07	4.199752e+07	1000.000000	580.000000	
50%	2013.000000	4.225000e+07	1.070266e+08	13000.000000	1000.000000	
75%	2014.000000	1.500000e+08	2.107548e+08	20000.000000	11000.000000	
max	2016.000000	2.600000e+08	9.366622e+08	35000.000000	96000.000000	

8 rows × 53 columns

Task 2: Data Analysis

Now that we have loaded the dataset and inspected it, we see that most of the data is in place. As of now, no data cleaning is required, so let's start with some data manipulation, analysis, and visualisation to get various insights about the data.

- ### Subtask 2.1: Reduce those Digits!

These numbers in the `budget` and `gross` are too big, compromising its readability. Let's convert the unit of the `budget` and `gross` columns from `$` to `million $` first.

In [4]:
```
# Divide the 'gross' and 'budget' columns by 1000000 to convert '$' to 'million $'
movies["Gross"]=movies["Gross"]/1000000
movies["budget"]=movies["budget"]/1000000
movies.head()
```

Out[4]:

	Title	title_year	budget	Gross	actor_1_name	actor_2_name	actor_3_name	actor_1_faceb
0	La La Land	2016	30.0	151.101803	Ryan Gosling	Emma Stone	Amiée Conn	
1	Zootopia	2016	150.0	341.268248	Ginnifer Goodwin	Jason Bateman	Idris Elba	
2	Lion	2016	12.0	51.738905	Dev Patel	Nicole Kidman	Rooney Mara	
3	Arrival	2016	47.0	100.546139	Amy Adams	Jeremy Renner	Forest Whitaker	
4	Manchester by the Sea	2016	9.0	47.695371	Casey Affleck	Michelle Williams	Kyle Chandler	

5 rows × 62 columns

- ### Subtask 2.2: Let's Talk 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. Extract the top ten profiting movies in descending order and store them in a new dataframe - `top10`.
4. Plot a scatter or a joint plot between the columns `budget` and `profit` and write a few words on what you observed.
5. Extract the movies with a negative profit and store them in a new dataframe - `neg_profit`

In [20]:
```
# Create the new column named 'profit' by subtracting the 'budget' column from the 'gro
movies["profit"]=movies["Gross"]- movies["budget"]
movies.head()
```

Out[20]:

	Title	title_year	budget	Gross	actor_1_name	actor_2_name	actor_3_name	actor_1_faceb
97	Star Wars: Episode VII - The Force Awakens	2015	245.0	936.662225	Doug Walker	Rob Walker	0	
11	The Avengers	2012	220.0	623.279547	Chris Hemsworth	Robert Downey Jr.	Scarlett Johansson	
47	Deadpool	2016	58.0	363.024263	Ryan Reynolds	Ed Skrein	Stefan Kapicic	
32	The Hunger Games: Catching Fire	2013	130.0	424.645577	Jennifer Lawrence	Josh Hutcherson	Sandra Ellis Lafferty	
12	Toy Story 3	2010	200.0	414.984497	Tom Hanks	John Ratzenberger	Don Rickles	

5 rows × 63 columns

In [21]:

```
# Sort the dataframe with the 'profit' column as reference using the 'sort_values' func
#'ascending' to 'False'

movies=movies.sort_values(by="profit",ascending=False)
```

In [22]:

```
# Get the top 10 profitable movies by using position based indexing. Specify the rows t
#Extract the top ten profiting movies in descending order and store them in a new dataf
top10=movies.iloc[:10,:]
top10[['profit','Title']].head(10)
```

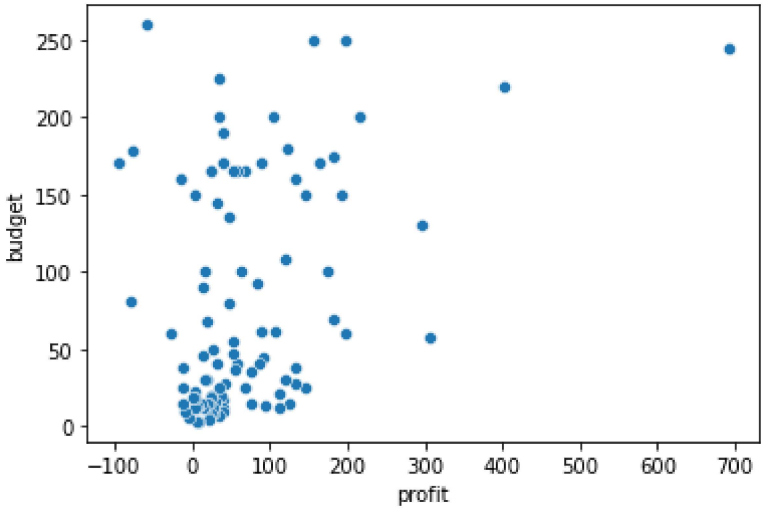
Out[22]:

	profit	Title
97	691.662225	Star Wars: Episode VII - The Force Awakens
11	403.279547	The Avengers
47	305.024263	Deadpool
32	294.645577	The Hunger Games: Catching Fire
12	214.984497	Toy Story 3
8	198.130642	The Dark Knight Rises
45	197.756197	The Lego Movie
1	191.268248	Zootopia
41	182.501645	Despicable Me
18	181.454367	Inside Out

Plot a scatter or a joint plot between the columns budget and profit and write a few words on what

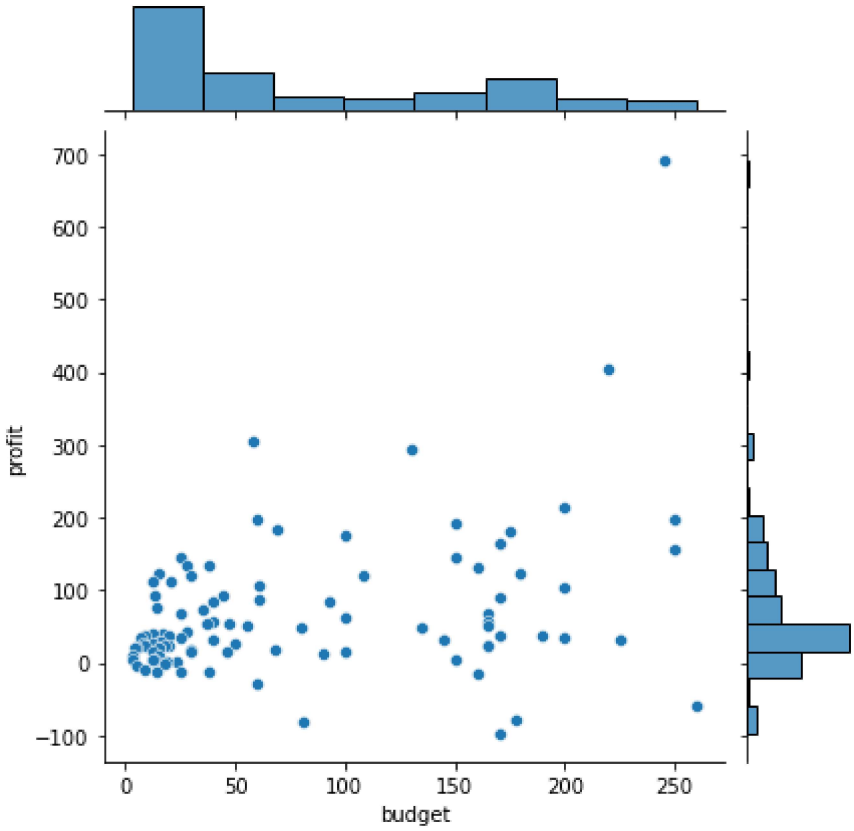
you observed.

In [23]:
```
sns.scatterplot(data=movies,x="profit",y="budget")
plt.show()
```



In [24]:
```
#Plot profit vs budget
plt.figure(figsize=(10,5))
sns.jointplot("budget","profit",movies)
plt.show()
```

```
<Figure size 720x360 with 0 Axes>
```



The dataset contains the 100 best performing movies from the year 2010 to 2016. However scatter plot tells a different story. You can notice that there are some movies with negative profit. Although

good movies do incur losses, but there appear to be quite a few movie with losses. What can be the reason behind this? Lets have a closer look at this by finding the movies with negative profit.

Extract the movies with a negative profit and store them in a new dataframe - neg_profit

In [25]:
```
neg_profit=movies[movies["profit"]<0]
```

In [26]:
```
neg_profit.shape
```

Out[26]: (11, 63)

In [27]:
```
neg_profit=neg_profit.sort_values(by='profit',ascending=False)
neg_profit.head(20)
```

Out[27]:

	Title	title_year	budget	Gross	actor_1_name	actor_2_name	actor_3_name	actor_1_faceb
99	Tucker and Dale vs Evil	2010	5.0	0.223838	Katrina Bowden	Tyler Labine	Chelan Simmons	
89	Amour	2012	8.9	0.225377	Isabelle Huppert	Emmanuelle Riva	Jean-Louis Trintignant	
56	Rush	2013	38.0	26.903709	Chris Hemsworth	Olivia Wilde	Alexandra Maria Lara	
66	Warrior	2011	25.0	13.651662	Tom Hardy	Frank Grillo	Kevin Dunn	
82	Flipped	2010	14.0	1.752214	Madeline Carroll	Rebecca De Mornay	Aidan Quinn	
28	X-Men: First Class	2011	160.0	146.405371	Jennifer Lawrence	Michael Fassbender	Oliver Platt	
46	Scott Pilgrim vs. the World	2010	60.0	31.494270	Anna Kendrick	Kieran Culkin	Ellen Wong	
7	Tangled	2010	260.0	200.807262	Brad Garrett	Donna Murphy	M.C. Gainey	
17	Edge of Tomorrow	2014	178.0	100.189501	Tom Cruise	Lara Pulver	Noah Taylor	
39	The Little Prince	2015	81.2	1.339152	Jeff Bridges	James Franco	Mackenzie Foy	
22	Hugo	2011	170.0	73.820094	ChloÃ« Grace Moretz	Christopher Lee	Ray Winstone	

11 rows × 63 columns

Checkpoint 1: Can you spot the movie `Tangled` in the dataset? You may be aware of the movie 'Tangled'. Although its one of the highest grossing movies of all time, it has negative profit as per this result. If you cross check the gross values of this movie (link:

https://www.imdb.com/title/tt0398286/), you can see that the gross in the dataset accounts only for the domestic gross and not the worldwide gross. This is true for may other movies also in the list.

- ### Subtask 2.3: The General Audience and the Critics

You might have noticed the column `MetaCritic` in this dataset. This is a very popular website where an average score is determined through the scores given by the top-rated critics. Second, you also have another column `IMDb_rating` which tells you the IMDb rating of a movie. This rating is determined by taking the average of hundred-thousands of ratings from the general audience.

In [28]:
```
movies[['MetaCritic','IMDb_rating']].head(10)
```

Out[28]:

	MetaCritic	IMDb_rating
97	81.0	8.1
11	69.0	8.1
47	65.0	8.0
32	76.0	7.6
12	92.0	8.3
8	78.0	8.4
45	83.0	7.8
1	78.0	8.1
41	72.0	7.7
18	94.0	8.2

In [29]:
```
# Change the scale of IMDb_rating
#As a part of this subtask, you are required to find out the highest rated movies which
#liked by critics and audiences alike.
#1. Firstly you will notice that the `MetaCritic` score is on a scale of `100` whereas
#is on a scale of 10. First convert the `IMDb_rating` column to a scale of 100.

movies['IMDb_rating']=movies['IMDb_rating']*10
```

In [30]:
```
#Now, to find out the movies which have been liked by both critics and audiences alike
#you need to -
#Create a new column `Avg_rating` which will have the average of the `MetaCritic` and `
# Find the average ratings

movies["Avg_rating"]=(movies["IMDb_rating"]+movies["MetaCritic"])/2
```

In [32]:
```
movies[['MetaCritic','IMDb_rating','Avg_rating']].head()
```

Out[32]:

	MetaCritic	IMDb_rating	Avg_rating
97	81.0	81.0	81.0

	MetaCritic	IMDb_rating	Avg_rating
11	69.0	81.0	75.0
47	65.0	80.0	72.5
32	76.0	76.0	76.0
12	92.0	83.0	87.5

In [33]:

```
#Retain only the movies in which the absolute difference(using abs() function) between
#columns is less than 0.5.
#Refer to this link to know how abs() funtion works - https://www.geeksforgeeks.org/abs
mv=movies[["Title","IMDb_rating","MetaCritic","Avg_rating"]]
mv=mv.loc[abs(mv["IMDb_rating"]-mv["MetaCritic"]<0.5)]
mv.head()
```

Out[33]:

	Title	IMDb_rating	MetaCritic	Avg_rating
97	Star Wars: Episode VII - The Force Awakens	81.0	81.0	81.0
32	The Hunger Games: Catching Fire	76.0	76.0	76.0
12	Toy Story 3	83.0	92.0	87.5
45	The Lego Movie	78.0	83.0	80.5
18	Inside Out	82.0	94.0	88.0

In [38]:

```
#Sort in descending order of average rating
# Find the movies with metacritic-IMDb_rating < 0.5 and also with the average rating of
mv=mv.sort_values(by="Avg_rating",ascending=False)
UniversalAcclaim=mv.loc[mv["Avg_rating"]>=80]
UniversalAcclaim.head(30)
```

Out[38]:

	Title	IMDb_rating	MetaCritic	Avg_rating
94	Boyhood	79.0	100.0	89.5
69	12 Years a Slave	81.0	96.0	88.5
18	Inside Out	82.0	94.0	88.0
4	Manchester by the Sea	79.0	96.0	87.5
0	La La Land	82.0	93.0	87.5
12	Toy Story 3	83.0	92.0	87.5
34	Gravity	78.0	96.0	87.0
70	Spotlight	81.0	93.0	87.0
96	Before Midnight	79.0	94.0	86.5
95	Whiplash	85.0	88.0	86.5
89	Amour	79.0	94.0	86.5
53	The Social Network	77.0	95.0	86.0

	Title	IMDb_rating	MetaCritic	Avg_rating
29	Mad Max: Fury Road	81.0	90.0	85.5
67	Her	80.0	90.0	85.0
65	The Grand Budapest Hotel	81.0	88.0	84.5
77	The Artist	79.0	89.0	84.0
76	The King's Speech	80.0	88.0	84.0
72	Birdman or (The Unexpected Virtue of Ignorance)	78.0	88.0	83.0
5	Hell or High Water	77.0	88.0	82.5
93	Dallas Buyers Club	80.0	84.0	82.0
49	Moneyball	76.0	87.0	81.5
51	Argo	77.0	86.0	81.5
87	Nebraska	77.0	86.0	81.5
75	Moonrise Kingdom	78.0	84.0	81.0
97	Star Wars: Episode VII - The Force Awakens	81.0	81.0	81.0
45	The Lego Movie	78.0	83.0	80.5
48	Captain Phillips	78.0	83.0	80.5
3	Arrival	80.0	81.0	80.5
33	The Martian	80.0	80.0	80.0

`Checkpoint 2:` Can you spot a `Star Wars` movie in your final dataset? Yes

- ### Subtask 2.4: Find the Most Popular Trios - I

You're a producer looking to make a blockbuster movie. There will primarily be three lead roles in your movie and you wish to cast the most popular actors for it. Now, since you don't want to take a risk, you will cast a trio which has already acted in together in a movie before. The metric that you've chosen to check the popularity is the Facebook likes of each of these actors.

The dataframe has three columns to help you out for the same, viz. `actor_1_facebook_likes` , `actor_2_facebook_likes` , and `actor_3_facebook_likes` . Your objective is to find the trios which has the most number of Facebook likes combined. That is, the sum of `actor_1_facebook_likes` , `actor_2_facebook_likes` and `actor_3_facebook_likes` should be maximum. Find out the top 5 popular trios, and output their names in a list.

Subtask 2.4: Find the Most Popular Trios - I You're a producer looking to make a blockbuster movie. There will primarily be three lead roles in your movie and you wish to cast the most popular actors for it. Now, since you don't want to take a risk, you will cast a trio which has already acted in together in a movie before. The metric that you've chosen to check the popularity is the Facebook likes of each of these actors. The dataframe has three columns to help you out for the same, viz. actor_1_facebook_likes, actor_2_facebook_likes, and actor_3_facebook_likes. Your objective is to find

the trios which has the most number of Facebook likes combined. That is, the sum of actor_1_facebook_likes, actor_2_facebook_likes and actor_3_facebook_likes should be maximum. Find out the top 5 popular trios, and output their names in a list.

In [39]:
```
# Creating a pivot table of fblikes and actor names and storing in dataset group
group=movies.pivot_table(values=["actor_1_facebook_likes","actor_2_facebook_likes","act
                  aggfunc="sum",index=["actor_1_name","actor_2_name","actor_3_name"])
```

In [40]:
```
#adding a new feature here to sum all the facebook likes of the trio of every movie
group["Total likes"]=group["actor_1_facebook_likes"]+group["actor_2_facebook_likes"]+gr
```

In [41]:
```
#sorting by facebook likes and getting top 5 trio
group.sort_values(by="Total likes",ascending=False,inplace=True)
```

In [42]:
```
group.reset_index(inplace=True)
```

In [43]:
```
group=group.iloc[0:5,:]
```

In [44]:
```
group
```

Out[44]:

	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_likes	actor_3_fa
0	Dev Patel	Nicole Kidman	Rooney Mara	33000	96000.0	
1	Leonardo DiCaprio	Tom Hardy	Joseph Gordon-Levitt	29000	27000.0	
2	Jennifer Lawrence	Peter Dinklage	Hugh Jackman	34000	22000.0	
3	Casey Affleck	Michelle Williams	Kyle Chandler	518	71000.0	
4	Tom Hardy	Christian Bale	Joseph Gordon-Levitt	27000	23000.0	

- ### Subtask 2.5: Find the Most Popular Trios - II

In the previous subtask you found the popular trio based on the total number of facebook likes. Let's add a small condition to it and make sure that all three actors are popular. The condition is **none of the three actors' Facebook likes should be less than half of the other two**. For example, the following is a valid combo:

- actor_1_facebook_likes: 70000
- actor_2_facebook_likes: 40000
- actor_3_facebook_likes: 50000

But the below one is not:

- actor_1_facebook_likes: 70000
- actor_2_facebook_likes: 40000
- actor_3_facebook_likes: 30000

since in this case, `actor_3_facebook_likes` is 30000, which is less than half of `actor_1_facebook_likes`.

Having this condition ensures that you aren't getting any unpopular actor in your trio (since the total likes calculated in the previous question doesn't tell anything about the individual popularities of each actor in the trio.).

You can do a manual inspection of the top 5 popular trios you have found in the previous subtask and check how many of those trios satisfy this condition. Also, which is the most popular trio after applying the condition above?

`Optional:` Even though you are finding this out by a natural inspection of the dataframe, can you also achieve this through some *if-else* statements to incorporate this. You can try this out on your own time after you are done with the assignment.

In [45]:
```
sorted([1,5,2])
```

Out[45]:
```
[1, 2, 5]
```

In [46]:
```
# Your answer here (optional)
j=0
for i in group["Total likes"]:
    temp=sorted([group.loc[j,"actor_1_facebook_likes"],group.loc[j,"actor_2_facebook_li
    if temp[0]>= temp[1]/2 and temp[0]>=temp[2]/2 and temp[1]>=temp[2]/2:
        print(sorted([group.loc[j,"actor_1_name"],group.loc[j,"actor_2_name"],group.loc

    j=j+1
```

```
['Joseph Gordon-Levitt', 'Leonardo DiCaprio', 'Tom Hardy']
['Hugh Jackman', 'Jennifer Lawrence', 'Peter Dinklage']
['Christian Bale', 'Joseph Gordon-Levitt', 'Tom Hardy']
```

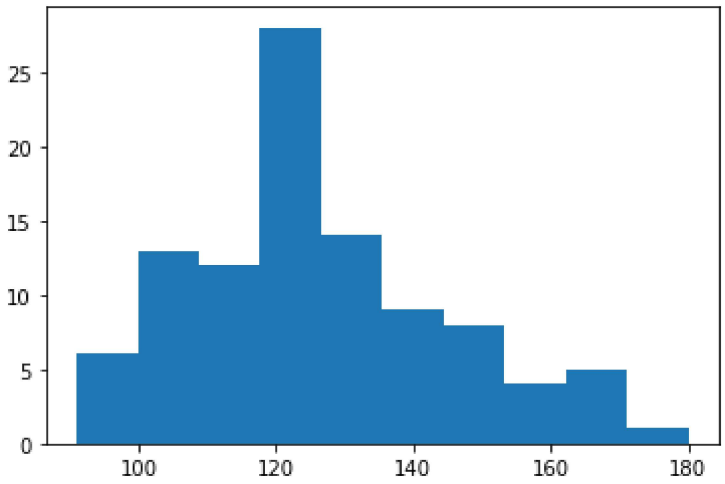
Write your answers below.

- **`No. of trios that satisfy the above condition:`** 3
- **`Most popular trio after applying the condition:`** Leonardo DiCaprio Tom Hardy Joseph Gordon-Levitt

- ### Subtask 2.6: Runtime Analysis

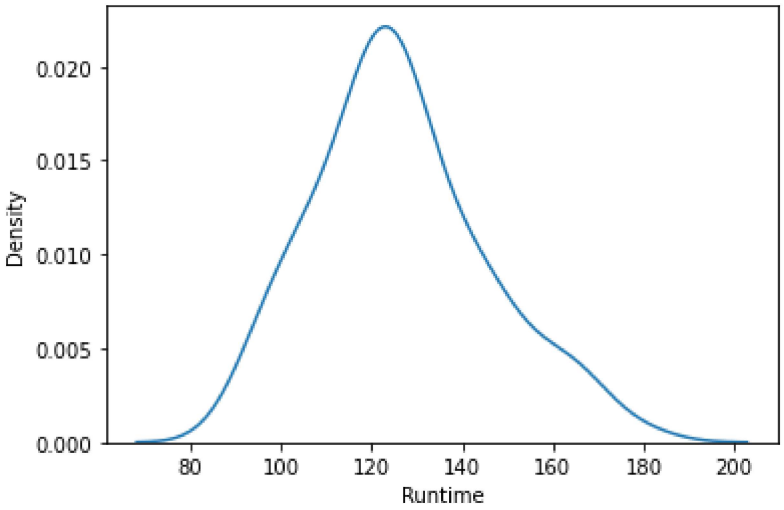
There is a column named `Runtime` in the dataframe which primarily shows the length of the movie. It might be intersting to see how this variable this distributed. Plot a `histogram` or `distplot` of seaborn to find the `Runtime` range most of the movies fall into.

In [47]:
```
# Runtime histogram/dist plot
plt.hist(movies["Runtime"])
plt.show()
```



In [48]:
```
sns.distplot(movies["Runtime"],hist=False)
```

Out[48]:
```
<AxesSubplot:xlabel='Runtime', ylabel='Density'>
```



Checkpoint 3: Most of the movies appear to be sharply 2 hour-long.

- ### Subtask 2.7: R-Rated Movies

Although R rated movies are restricted movies for the under 18 age group, still there are vote counts from that age group. Among all the R rated movies that have been voted by the under-18 age group, find the top 10 movies that have the highest number of votes i.e. `CVotesU18` from the `movies` dataframe. Store these in a dataframe named `PopularR`.

In [49]:
```
# Write your code here

PopularR=movies.loc[movies["content_rating"]=="R"].sort_values(by="CVotesU18",ascending
PopularR
```

Out[49]:

	Title	CVotesU18
47	Deadpool	4598
36	The Wolf of Wall Street	3622
35	Django Unchained	3250
29	Mad Max: Fury Road	3159
95	Whiplash	2878
31	The Revenant	2619
40	Shutter Island	2321
43	Gone Girl	2286
65	The Grand Budapest Hotel	2083
72	Birdman or (The Unexpected Virtue of Ignorance)	1891

`Checkpoint 4:` Are these kids watching `Deadpool` a lot? Yes atleast in terms of votes i.e 4598

Task 3 : Demographic analysis

If you take a look at the last columns in the dataframe, most of these are related to demographics of the voters (in the last subtask, i.e., 2.8, you made use one of these columns - CVotesU18). We also have three genre columns indicating the genres of a particular movie. We will extensively use these columns for the third and the final stage of our assignment wherein we will analyse the voters across all demographics and also see how these vary across various genres. So without further ado, let's get started with `demographic analysis .`

- ### Subtask 3.1 Combine the Dataframe by Genres

There are 3 columns in the dataframe - `genre_1` , `genre_2` , and `genre_3` . As a part of this subtask, you need to aggregate a few values over these 3 columns.

1. First create a new dataframe `df_by_genre` that contains `genre_1` , `genre_2` , and `genre_3` and all the columns related to **CVotes/Votes** from the `movies` data frame. There are 47 columns to be extracted in total.
2. Now, Add a column called `cnt` to the dataframe `df_by_genre` and initialize it to one. You will realise the use of this column by the end of this subtask.
3. First group the dataframe `df_by_genre` by `genre_1` and find the sum of all the numeric columns such as `cnt` , columns related to CVotes and Votes columns and store it in a dataframe `df_by_g1` .
4. Perform the same operation for `genre_2` and `genre_3` and store it dataframes `df_by_g2` and `df_by_g3` respectively.
5. Now that you have 3 dataframes performed by grouping over `genre_1` , `genre_2` , and `genre_3` separately, it's time to combine them. For this, add the three dataframes and store it in a new dataframe `df_add` , so that the corresponding values of Votes/CVotes get added for

each genre.There is a function called `add()` in pandas which lets you do this. You can refer to this link to see how this function works. https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.DataFrame.add.html

6. The column `cnt` on aggregation has basically kept the track of the number of occurences of each genre.Subset the genres that have atleast 10 movies into a new dataframe `genre_top10` based on the `cnt` column value.
7. Now, take the mean of all the numeric columns by dividing them with the column value `cnt` and store it back to the same dataframe. We will be using this dataframe for further analysis in this task unless it is explicitly mentioned to use the dataframe `movies`.
8. Since the number of votes can't be a fraction, type cast all the CVotes related columns to integers. Also, round off all the Votes related columns upto two digits after the decimal point.

In [50]:
```
# Create the dataframe df_by_genre
df_by_genre = movies.iloc[0:,11:14].join(movies.iloc[0:,16:60])
```

In [51]:
```
df_by_genre.shape
```

Out[51]: (100, 47)

In [52]:
```
# Create a column cnt and initialize it to 1
df_by_genre["cnt"]=1
```

In [53]:
```
df_by_genre[["genre_1","genre_2","genre_3"]]
```

Out[53]:

	genre_1	genre_2	genre_3
97	Action	Adventure	Fantasy
11	Action	Sci-Fi	NaN
47	Action	Adventure	Comedy
32	Action	Adventure	Mystery
12	Animation	Adventure	Comedy
...	...	...	...
46	Action	Comedy	Romance
7	Animation	Adventure	Comedy
17	Action	Adventure	Sci-Fi
39	Animation	Adventure	Drama
22	Adventure	Drama	Family

100 rows × 3 columns

In [54]:
```
# Group the movies by individual genres
df_by_g1 = df_by_genre.groupby("genre_1").sum()
```

```
df_by_g2 = df_by_genre.groupby("genre_2").sum()
df_by_g3 = df_by_genre.groupby("genre_3").sum()
```

In [55]:
```
df_by_g1
```

Out[55]:

genre_1	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes
Action	2928407	3261919	4247693	2662020	986774	364234	156150	89483	61
Adventure	1058779	1179818	1560541	966275	365486	136985	58559	33174	22
Animation	681562	798227	1153214	722782	251076	83069	30718	15733	10
Biography	666831	1088430	1654704	962977	306247	100005	38874	21536	15
Comedy	371217	496905	770395	518566	205434	81933	35788	20965	15
Crime	383290	690221	1083469	627593	206756	71460	30336	17190	11
Drama	1080725	1494053	1827363	1078966	417205	163874	75525	45846	32
Mystery	150405	230844	278844	132349	45167	15615	7061	3780	2

8 rows × 45 columns

In [56]:
```
df_by_g2
```

Out[56]:

genre_2	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes
Action	238060	285510	430062	260106	88580	29250	10820	5521	3
Adventure	2297820	2548864	3271725	2055600	758009	272735	113691	64623	44
Biography	185172	313178	576374	370003	119348	38643	14844	7974	5
Comedy	428995	624720	854162	512668	193916	76752	35193	20995	14
Crime	19576	40247	85359	64633	24920	8548	3261	1669	
Drama	1923492	2761237	4112363	2492241	853434	300100	124511	70205	49
Family	68937	54947	102488	80465	31205	11792	4808	2454	1
Fantasy	270616	290831	447307	291071	120920	47215	19848	10871	6
History	15757	32840	83322	63800	19183	5178	1657	735	
Horror	16572	19818	44460	35863	13456	4588	1684	855	
Music	110404	161864	132656	56007	16577	6031	2937	1859	1
Musical	54268	47750	63323	45160	22393	10744	5551	3484	2
Mystery	85313	152619	233474	148176	55575	20135	8863	5034	3
Romance	230425	270702	353326	215355	87701	35226	16298	9448	6

genre_2	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes
Sci-Fi	350243	361819	433983	284879	125143	53350	24462	15069	10
Sport	21364	28964	58237	45563	16432	5118	1655	848	
Thriller	624792	709424	734770	365134	135892	52714	25734	15558	10
War	15911	17607	32570	24461	10274	3848	1387	726	
Western	234824	339329	286911	121445	38251	14227	6469	4149	3

19 rows × 45 columns

In [57]:
```
df_by_g3
```

Out[57]:

genre_3	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes
Adventure	238060	285510	430062	260106	88580	29250	10820	5521	3
Comedy	583404	653362	882294	559835	200937	68167	26488	14258	9
Crime	171660	236650	250667	129164	46715	18682	8674	5854	4
Drama	400221	680085	1167327	748493	258717	88338	35439	19075	12
Family	29228	40728	77893	62936	27932	11179	4664	2674	1
Fantasy	301836	311392	442460	308676	120911	46269	19555	11362	7
History	135504	227547	311209	159262	48678	16055	6307	3649	2
Music	74245	71191	64640	38831	17377	8044	3998	2839	2
Mystery	274446	443661	654167	375087	128131	44818	18755	10578	7
Romance	319534	418790	715954	497486	193588	75675	32615	18250	12
Sci-Fi	1975041	2169036	2569011	1517219	546668	200825	87463	50835	35
Sport	95717	140282	214806	138600	44470	13821	5155	2509	1
Thriller	456909	756067	1258608	810665	280154	97239	39547	22382	15
War	36753	54703	111271	82505	30231	10553	4303	2388	1
Western	21094	40901	91825	67175	23055	7191	2678	1305	

15 rows × 45 columns

In [58]:
```
# Add the grouped data frames and store it in a new data frame
df_add = df_by_g1.add(df_by_g2,fill_value=0).add(df_by_g3,fill_value=0)
df_add
```

Out[58]:

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVot
Action	3166467.0	3547429.0	4677755.0	2922126.0	1075354.0	393484.0	166970.0	95004.0	65
Adventure	3594659.0	4014192.0	5262328.0	3281981.0	1212075.0	438970.0	183070.0	103318.0	69
Animation	681562.0	798227.0	1153214.0	722782.0	251076.0	83069.0	30718.0	15733.0	10
Biography	852003.0	1401608.0	2231078.0	1332980.0	425595.0	138648.0	53718.0	29510.0	20
Comedy	1383616.0	1774987.0	2506851.0	1591069.0	600287.0	226852.0	97469.0	56218.0	39
Crime	574526.0	967118.0	1419495.0	821390.0	278391.0	98690.0	42271.0	24713.0	16
Drama	3404438.0	4935375.0	7107053.0	4319700.0	1529356.0	552312.0	235475.0	135126.0	94
Family	98165.0	95675.0	180381.0	143401.0	59137.0	22971.0	9472.0	5128.0	3
Fantasy	572452.0	602223.0	889767.0	599747.0	241831.0	93484.0	39403.0	22233.0	14
History	151261.0	260387.0	394531.0	223062.0	67861.0	21233.0	7964.0	4384.0	3
Horror	16572.0	19818.0	44460.0	35863.0	13456.0	4588.0	1684.0	855.0	
Music	184649.0	233055.0	197296.0	94838.0	33954.0	14075.0	6935.0	4698.0	3
Musical	54268.0	47750.0	63323.0	45160.0	22393.0	10744.0	5551.0	3484.0	2
Mystery	510164.0	827124.0	1166485.0	655612.0	228873.0	80568.0	34679.0	19392.0	13
Romance	549959.0	689492.0	1069280.0	712841.0	281289.0	110901.0	48913.0	27698.0	19
Sci-Fi	2325284.0	2530855.0	3002994.0	1802098.0	671811.0	254175.0	111925.0	65904.0	46
Sport	117081.0	169246.0	273043.0	184163.0	60902.0	18939.0	6810.0	3357.0	2
Thriller	1081701.0	1465491.0	1993378.0	1175799.0	416046.0	149953.0	65281.0	37940.0	25
War	52664.0	72310.0	143841.0	106966.0	40505.0	14401.0	5690.0	3114.0	1
Western	255918.0	380230.0	378736.0	188620.0	61306.0	21418.0	9147.0	5454.0	3

20 rows × 45 columns

In [59]:
```
df_add["cnt"]
```

Out[59]:
```
Action       31.0
Adventure    38.0
Animation    11.0
Biography    18.0
Comedy       23.0
Crime        11.0
Drama        65.0
Family        2.0
Fantasy       7.0
History       4.0
Horror        1.0
Music         2.0
Musical       1.0
Mystery       7.0
Romance      13.0
Sci-Fi       17.0
Sport         3.0
```

```
Thriller     13.0
War           2.0
Western       2.0
Name: cnt, dtype: float64
```

In [60]:
```
# Extract genres with atleast 10 occurences
genre_top10 = df_add[df_add["cnt"] > 10].sort_values(by="cnt",ascending=False)
```

In [61]:
```
genre_top10
```

Out[61]:

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVot
Drama	3404438.0	4935375.0	7107053.0	4319700.0	1529356.0	552312.0	235475.0	135126.0	94
Adventure	3594659.0	4014192.0	5262328.0	3281981.0	1212075.0	438970.0	183070.0	103318.0	69
Action	3166467.0	3547429.0	4677755.0	2922126.0	1075354.0	393484.0	166970.0	95004.0	65
Comedy	1383616.0	1774987.0	2506851.0	1591069.0	600287.0	226852.0	97469.0	56218.0	39
Biography	852003.0	1401608.0	2231078.0	1332980.0	425595.0	138648.0	53718.0	29510.0	20
Sci-Fi	2325284.0	2530855.0	3002994.0	1802098.0	671811.0	254175.0	111925.0	65904.0	46
Romance	549959.0	689492.0	1069280.0	712841.0	281289.0	110901.0	48913.0	27698.0	19
Thriller	1081701.0	1465491.0	1993378.0	1175799.0	416046.0	149953.0	65281.0	37940.0	25
Animation	681562.0	798227.0	1153214.0	722782.0	251076.0	83069.0	30718.0	15733.0	10
Crime	574526.0	967118.0	1419495.0	821390.0	278391.0	98690.0	42271.0	24713.0	16

10 rows × 45 columns

In [62]:
```
# Take the mean for every column by dividing with cnt
genre_top10.iloc[:,0:-1]=genre_top10.iloc[:,0:-1].divide(genre_top10["cnt"],axis=0)
```

In [63]:
```
genre_top10
```

Out[63]:

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	
Drama	52375.969231	75928.846154	109339.276923	66456.923077	23528.553846	8497.107692	36
Adventure	94596.289474	105636.631579	138482.315789	86367.921053	31896.710526	11551.842105	48
Action	102144.096774	114433.193548	150895.322581	94262.129032	34688.838710	12693.032258	53
Comedy	60157.217391	77173.347826	108993.521739	69176.913043	26099.434783	9863.130435	42
Biography	47333.500000	77867.111111	123948.777778	74054.444444	23644.166667	7702.666667	29
Sci-Fi	136781.411765	148873.823529	176646.705882	106005.764706	39518.294118	14951.470588	65
Romance	42304.538462	53037.846154	82252.307692	54833.923077	21637.615385	8530.846154	37
Thriller	83207.769231	112730.076923	153336.769231	90446.076923	32003.538462	11534.846154	50
Animation	61960.181818	72566.090909	104837.636364	65707.454545	22825.090909	7551.727273	27

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	
Crime	52229.636364	87919.818182	129045.000000	74671.818182	25308.272727	8971.818182	38

10 rows × 45 columns

In [64]:
```
# Rounding off the columns of Votes to two decimals
genre_top10.loc[:,"VotesM":"VotesnUS"]=round(genre_top10.loc[:,"VotesM":"VotesnUS"],2)
```

In [65]:
```
# Converting CVotes to int type
genre_top10[genre_top10.loc[:,"CVotes10":"CVotesnUS"].columns]=genre_top10[genre_top10.
```

In [66]:
```
genre_top10
```

Out[66]:

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes
Drama	52375	75928	109339	66456	23528	8497	3622	2078	14
Adventure	94596	105636	138482	86367	31896	11551	4817	2718	18
Action	102144	114433	150895	94262	34688	12693	5386	3064	2
Comedy	60157	77173	108993	69176	26099	9863	4237	2444	1
Biography	47333	77867	123948	74054	23644	7702	2984	1639	1
Sci-Fi	136781	148873	176646	106005	39518	14951	6583	3876	2
Romance	42304	53037	82252	54833	21637	8530	3762	2130	14
Thriller	83207	112730	153336	90446	32003	11534	5021	2918	19
Animation	61960	72566	104837	65707	22825	7551	2792	1430	9
Crime	52229	87919	129045	74671	25308	8971	3842	2246	15

10 rows × 45 columns

Subtask 3.1 Combine the Dataframe by Genres There are 3 columns in the dataframe - genre_1, genre_2, and genre_3. As a part of this subtask, you need to aggregate a few values over these 3 columns. First create a new dataframe df_by_genre that contains genre_1, genre_2, and genre_3 and all the columns related to CVotes/Votes from the movies data frame. There are 47 columns to be extracted in total. Now, Add a column called cnt to the dataframe df_by_genre and initialize it to one. You will realise the use of this column by the end of this subtask.

First group the dataframe df_by_genre by genre_1 and find the sum of all the numeric columns such as cnt, columns related to CVotes and Votes columns and store it in a dataframe df_by_g1.

Perform the same operation for genre_2 and genre_3 and store it dataframes df_by_g2 and df_by_g3 respectively.

Now that you have 3 dataframes performed by grouping over genre_1, genre_2, and genre_3 separately, it's time to combine them. For this, add the three dataframes and store it in a new dataframe df_add, so that the corresponding values of Votes/CVotes get added for each genre. There is a function called add() in pandas which lets you do this. You can refer to this link to see how this function works. https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandasDataFrame.add.html

The column cnt on aggregation has basically kept the track of the number of occurences of each genre.

Subset the genres that have atleast 10 movies into a new dataframe genre_top10 based on the cnt column value.

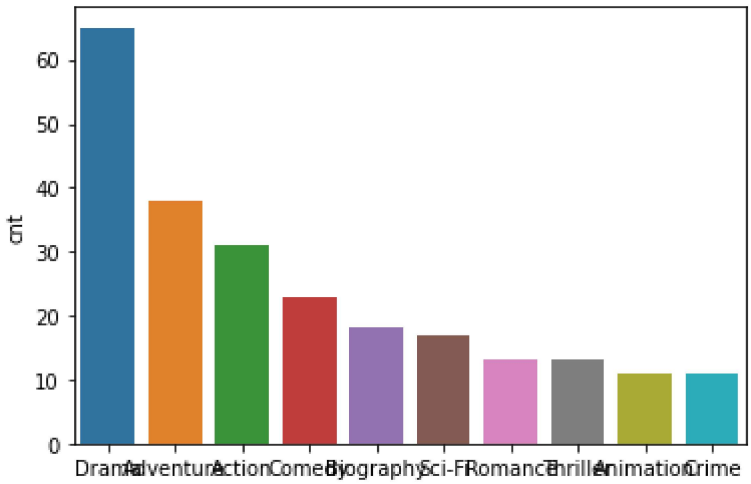
Now, take the mean of all the numeric columns by dividing them with the column value cnt and store it back to the same dataframe. We will be using this dataframe for further analysis in this task unless it is explicitly mentioned to use the dataframe movies.

Since the number of votes can't be a fraction, type cast all the CVotes related columns to integers. Also, round off all the Votes related columns upto two digits after the decimal point.

- ### Subtask 3.2: Genre Counts!

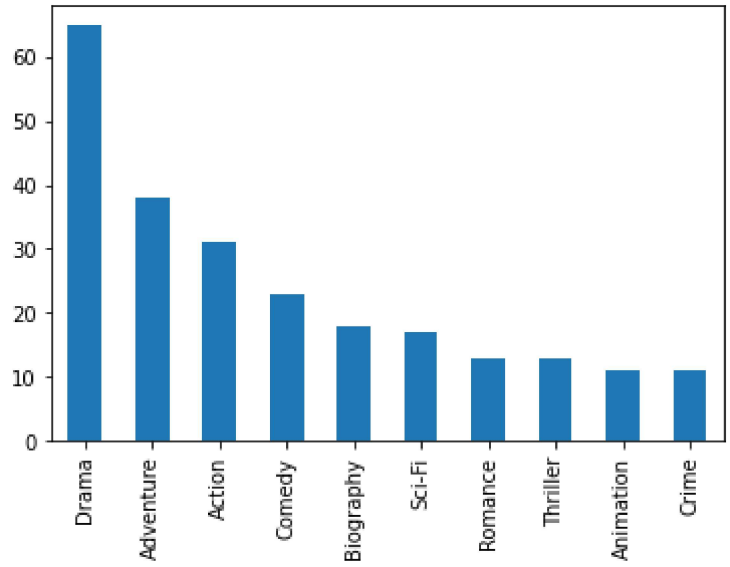
Now let's derive some insights from this data frame. Make a bar chart plotting different genres vs cnt using seaborn.

In [67]:
```
# Countplot for genres
sns.barplot(x=genre_top10.index,y=genre_top10["cnt"])
plt.show()
```



In [68]:
```
# Barplot for genres
genre_top10.cnt.plot.bar()
```

Out[68]: <AxesSubplot:>



`Checkpoint 5:` Is the bar for `Drama` the tallest? Yes with the count above 60

- ### Subtask 3.3: Gender and Genre

If you have closely looked at the Votes- and CVotes-related columns, you might have noticed the suffixes `F` and `M` indicating Female and Male. Since we have the vote counts for both males and females, across various age groups, let's now see how the popularity of genres vary between the two genders in the dataframe.

1. Make the first heatmap to see how the average number of votes of males is varying across the genres. Use seaborn heatmap for this analysis. The X-axis should contain the four age-groups for males, i.e., `CVotesU18M`, `CVotes1829M`, `CVotes3044M`, and `CVotes45AM`. The Y-axis will have the genres and the annotation in the heatmap tell the average number of votes for that age-male group.

2. Make the second heatmap to see how the average number of votes of females is varying across the genres. Use seaborn heatmap for this analysis. The X-axis should contain the four age-groups for females, i.e., `CVotesU18F`, `CVotes1829F`, `CVotes3044F`, and `CVotes45AF`. The Y-axis will have the genres and the annotation in the heatmap tell the average number of votes for that age-female group.

3. Make sure that you plot these heatmaps side by side using `subplots` so that you can easily compare the two genders and derive insights.

4. Write your any three inferences from this plot. You can make use of the previous bar plot also here for better insights. Refer to this link- https://seaborn.pydata.org/generated/seaborn.heatmap.html. You might have to plot something similar to the fifth chart in this page (You have to plot two such heatmaps side by side).

5. Repeat subtasks 1 to 4, but now instead of taking the CVotes-related columns, you need to do the same process for the Votes-related columns. These heatmaps will show you how the two

genders have rated movies across various genres.

You might need the below link for formatting your heatmap.
https://stackoverflow.com/questions/56942670/matplotlib-seaborn-first-and-last-row-cut-in-half-of-heatmap-plot

- Note : Use `genre_top10` dataframe for this subtask

If you take a look at the final dataframe that you have gotten, you will see that you now have the complete information about all the demographic (Votes- and CVotes-related) columns across the top 10 genres. We can use this dataset to extract exciting insights about the voters!

In [69]:
```
genre_top10.head()
```

Out[69]:

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes
Drama	52375	75928	109339	66456	23528	8497	3622	2078	1
Adventure	94596	105636	138482	86367	31896	11551	4817	2718	1
Action	102144	114433	150895	94262	34688	12693	5386	3064	2
Comedy	60157	77173	108993	69176	26099	9863	4237	2444	1
Biography	47333	77867	123948	74054	23644	7702	2984	1639	1

5 rows × 45 columns

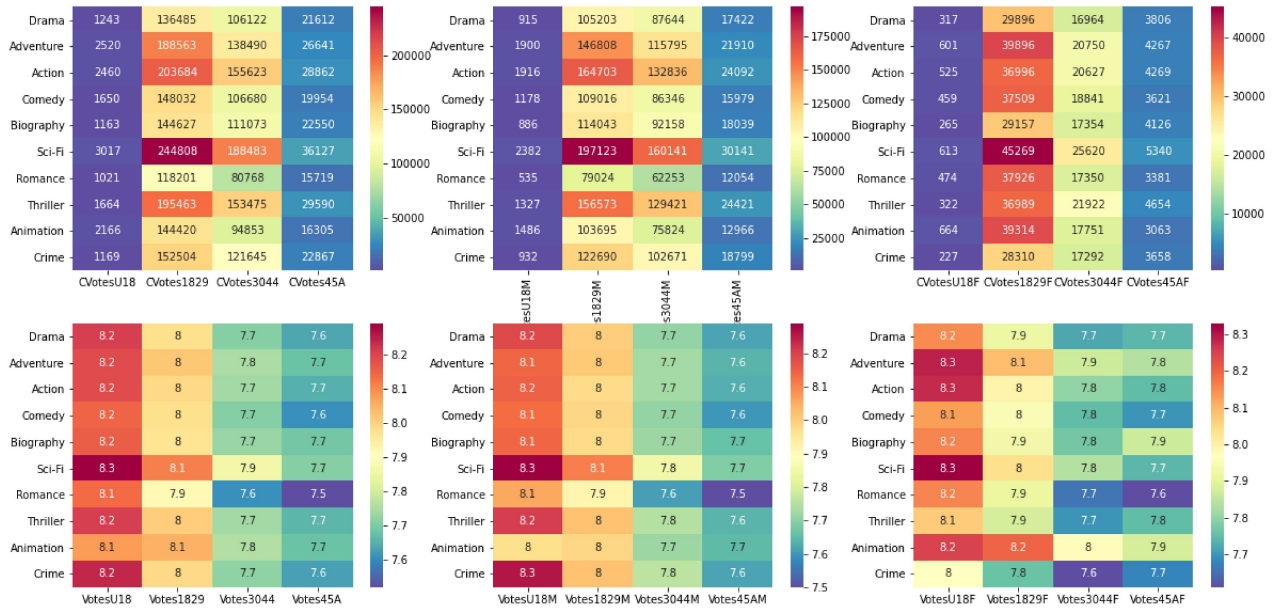
In [70]:
```
genre_top10.columns
```

Out[70]:
```
Index(['CVotes10', 'CVotes09', 'CVotes08', 'CVotes07', 'CVotes06', 'CVotes05',
       'CVotes04', 'CVotes03', 'CVotes02', 'CVotes01', 'CVotesMale',
       'CVotesFemale', 'CVotesU18', 'CVotesU18M', 'CVotesU18F', 'CVotes1829',
       'CVotes1829M', 'CVotes1829F', 'CVotes3044', 'CVotes3044M',
       'CVotes3044F', 'CVotes45A', 'CVotes45AM', 'CVotes45AF', 'CVotes1000',
       'CVotesUS', 'CVotesnUS', 'VotesM', 'VotesF', 'VotesU18', 'VotesU18M',
       'VotesU18F', 'Votes1829', 'Votes1829M', 'Votes1829F', 'Votes3044',
       'Votes3044M', 'Votes3044F', 'Votes45A', 'Votes45AM', 'Votes45AF',
       'Votes1000', 'VotesUS', 'VotesnUS', 'cnt'],
      dtype='object')
```

Subtask 3.3: Gender and Genre If you have closely looked at the Votes- and CVotes-related columns, you might have noticed the suffixes F and M indicating Female and Male. Since we have the vote counts for both males and females, across various age groups, let's now see how the popularity of genres vary between the two genders in the dataframe. Make the first heatmap to see how the average number of votes of males is varying across the genres. Use seaborn heatmap for this analysis. The X-axis should contain the four age-groups for males, i.e., CVotesU18M,CVotes1829M, CVotes3044M, and CVotes45AM. The Y-axis will have the genres and the annotation in the heatmap tell the average number of votes for that age-male group. Make the second heatmap to see how the average number of votes of females is varying across the genres. Use seaborn heatmap for this analysis. The X-axis should contain the four age-groups for females, i.e., CVotesU18F,CVotes1829F, CVotes3044F, and CVotes45AF. The Y-axis will have the genres and the annotation in the heatmap tell the average number of votes for that age-female group. Make sure that you plot these heatmaps side by side using subplots so that you can easily compare the two genders and derive insights. Write your any three inferences from this plot. You can make use of the previous bar plot also here for better insights. Refer to this link- https://seaborn.pydata.org/generated/seaborn.heatmap.html.

You might have to plot something similar to the fifth chart in this page (You have to plot two such heatmaps side by side). Repeat subtasks 1 to 4, but now instead of taking the CVotes-related columns, you need to do the same process for the Votes-related columns. These heatmaps will show you how the two genders have rated movies across various genres. You might need the below link for formatting your heatmap.
https://stackoverflow.com/questions/56942670/matplotlib-seaborn-first-and-last-row-cut-in-half-of-heatmap-plot
Note : Use genre_top10 dataframe for this subtask

In [71]:
```
plt.figure(figsize=(20,10))
plt.subplot(2,3,1)
ax=sns.heatmap(genre_top10[['CVotesU18','CVotes1829','CVotes3044','CVotes45A']],annot=T
plt.subplot(2,3,2)
ax=sns.heatmap(genre_top10[["CVotesU18M","CVotes1829M","CVotes3044M","CVotes45AM"]],ann
plt.subplot(2,3,3)
ax=sns.heatmap(genre_top10[["CVotesU18F","CVotes1829F","CVotes3044F","CVotes45AF"]],ann
plt.subplot(2,3,4)
ax=sns.heatmap(genre_top10[["VotesU18","Votes1829","Votes3044","Votes45A"]],annot=True,
plt.subplot(2,3,5)
ax=sns.heatmap(genre_top10[["VotesU18M","Votes1829M","Votes3044M","Votes45AM"]],annot=T
plt.subplot(2,3,6)
ax=sns.heatmap(genre_top10[["VotesU18F","Votes1829F","Votes3044F","Votes45AF"]],annot=T
plt.show()
```



1. Sci-Fi appears to be the highest rated genre in males of all age groups. 2. Except in under 18 category, Animation appears to be the highest rated genre in females of all age groups 3 Sci-Fi appears to be the highest rated genre in the age group of U18 across genders. 4. For males, animated movies are the least rated for age group under 18 whereas romantic movies become the least rated for all other age groups 5. Crime movies are generally the least favourite among females of all ages except when they turn above 45 when romance genre takes their place 6. In general, people, irrespective of age and gender like movies more when they are younger and that liking and the tendency to give higher rating decreases over time, thereby decreasing the average rating of age groups if we go from under 18 to 45+ 7. Genre romance has in general got the least number of votes among males 8. Under 18 age group seems to be most actively voting for any genre irrespective of gender 9. Males of Under18 would rather watch any other genre of movie other than animation whereas the females under 18 would rather watch any other genre of movie other than crime 10.Agegroup 18-29 males like romantic movies the least, females like crime movies the least. 11.There is a progressive dip in male votes as age progresses, in all genres , however in females the dip in seen in all other genres except for crime and thriller. 12.Males have voted more than females. Across genders Age group under 18 Scifi Romance and animation Age group 18-29 Animation & scifi Romance Age group 30-44 Scifi Romance Age group 45 and above distributed genres Romance Males Age group under 18 Scifi & Crime Animation Age group 18-29 Scifi Romance Age group 30-44 Scifi,Crime,thriller Romance Age group 45 and above

Scifi,animation,biography Romance Females Age group under 18 Scifi,Action Adventure Crime Age group 18-29 Animation Crime Age group 30-44 Animation Crime. Age group 45 and above Animation,biography Romance

- ### Subtask 3.4: US vs non-US Cross Analysis

The dataset contains both the US and non-US movies. Let's analyse how both the US and the non-US voters have responded to the US and the non-US movies.

1. Create a column `IFUS` in the dataframe `movies`. The column `IFUS` should contain the value "USA" if the `Country` of the movie is "USA". For all other countries other than the USA, `IFUS` should contain the value `non-USA`.

1. Now make a boxplot that shows how the number of votes from the US people i.e. `CVotesUS` is varying for the US and non-US movies. Make use of the column `IFUS` to make this plot. Similarly, make another subplot that shows how non US voters have voted for the US and non-US movies by plotting `CVotesnUS` for both the US and non-US movies. Write any of your two inferences/observations from these plots.

1. Again do a similar analysis but with the ratings. Make a boxplot that shows how the ratings from the US people i.e. `VotesUS` is varying for the US and non-US movies. Similarly, make another subplot that shows how `VotesnUS` is varying for the US and non-US movies. Write any of your two inferences/observations from these plots.

Note : Use `movies` dataframe for this subtask. Make use of this documention to format your boxplot - https://seaborn.pydata.org/generated/seaborn.boxplot.html

In [73]:
```
movies['IFUS']='USA'
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100 entries, 97 to 22
Data columns (total 65 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Title                   100 non-null    object
 1   title_year              100 non-null    int64
 2   budget                  100 non-null    float64
 3   Gross                   100 non-null    float64
 4   actor_1_name            100 non-null    object
 5   actor_2_name            100 non-null    object
 6   actor_3_name            100 non-null    object
 7   actor_1_facebook_likes  100 non-null    int64
 8   actor_2_facebook_likes  99 non-null     float64
 9   actor_3_facebook_likes  98 non-null     float64
 10  IMDb_rating             100 non-null    float64
 11  genre_1                 100 non-null    object
 12  genre_2                 97 non-null     object
 13  genre_3                 74 non-null     object
 14  MetaCritic              95 non-null     float64
 15  Runtime                 100 non-null    int64
 16  CVotes10                100 non-null    int64
 17  CVotes09                100 non-null    int64
 18  CVotes08                100 non-null    int64
```

```
 19  CVotes07                100 non-null    int64
 20  CVotes06                100 non-null    int64
 21  CVotes05                100 non-null    int64
 22  CVotes04                100 non-null    int64
 23  CVotes03                100 non-null    int64
 24  CVotes02                100 non-null    int64
 25  CVotes01                100 non-null    int64
 26  CVotesMale              100 non-null    int64
 27  CVotesFemale            100 non-null    int64
 28  CVotesU18               100 non-null    int64
 29  CVotesU18M              100 non-null    int64
 30  CVotesU18F              100 non-null    int64
 31  CVotes1829              100 non-null    int64
 32  CVotes1829M             100 non-null    int64
 33  CVotes1829F             100 non-null    int64
 34  CVotes3044              100 non-null    int64
 35  CVotes3044M             100 non-null    int64
 36  CVotes3044F             100 non-null    int64
 37  CVotes45A               100 non-null    int64
 38  CVotes45AM              100 non-null    int64
 39  CVotes45AF              100 non-null    int64
 40  CVotes1000              100 non-null    int64
 41  CVotesUS                100 non-null    int64
 42  CVotesnUS               100 non-null    int64
 43  VotesM                  100 non-null    float64
 44  VotesF                  100 non-null    float64
 45  VotesU18                100 non-null    float64
 46  VotesU18M               100 non-null    float64
 47  VotesU18F               100 non-null    float64
 48  Votes1829               100 non-null    float64
 49  Votes1829M              100 non-null    float64
 50  Votes1829F              100 non-null    float64
 51  Votes3044               100 non-null    float64
 52  Votes3044M              100 non-null    float64
 53  Votes3044F              100 non-null    float64
 54  Votes45A                100 non-null    float64
 55  Votes45AM               100 non-null    float64
 56  Votes45AF               100 non-null    float64
 57  Votes1000               100 non-null    float64
 58  VotesUS                 100 non-null    float64
 59  VotesnUS                100 non-null    float64
 60  content_rating          100 non-null    object
 61  Country                 100 non-null    object
 62  profit                  100 non-null    float64
 63  Avg_rating              95 non-null     float64
 64  IFUS                    100 non-null    object
dtypes: float64(25), int64(30), object(10)
memory usage: 51.6+ KB
```

In [75]:
```
movies['IFUS'].unique()
```

Out[75]:
```
array(['USA'], dtype=object)
```

In [76]:
```
movies['Country'].unique()
```

Out[76]:
```
array(['USA', 'Australia', 'UK', 'Spain', 'France', 'Canada'],
      dtype=object)
```

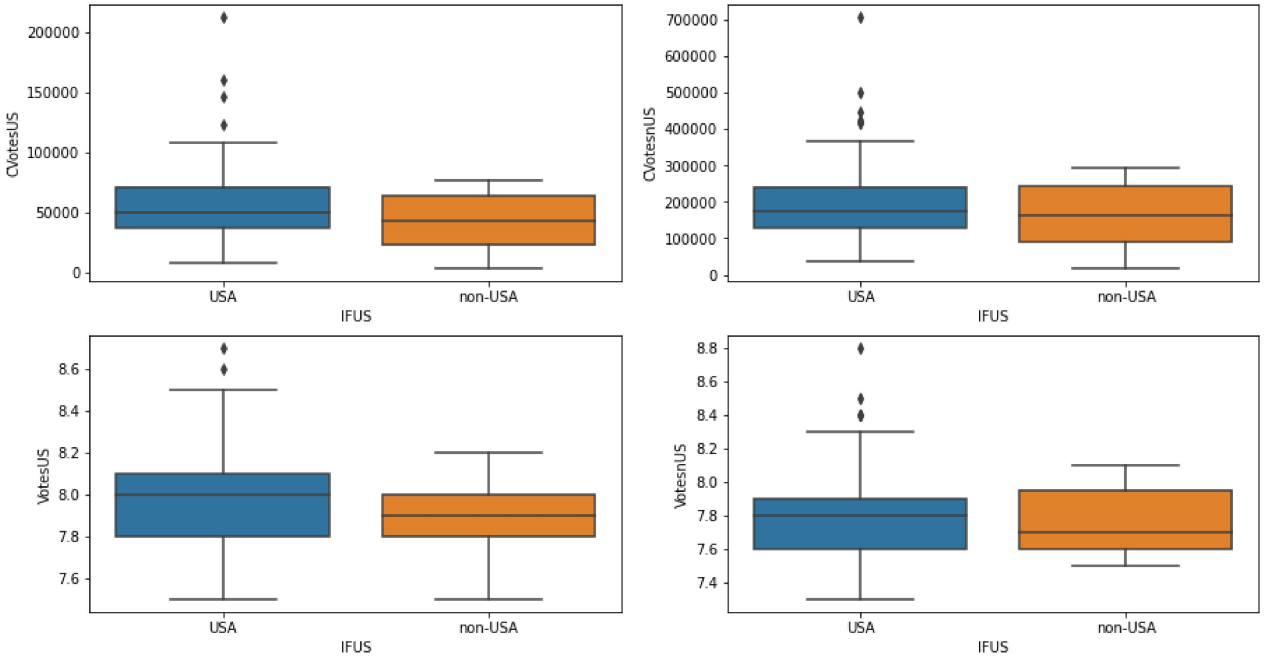
In [77]:
```
movies.loc[movies["Country"] != "USA","IFUS"] = "non-USA"
movies["Country"].unique()
```

Out[77]:
```
array(['USA', 'Australia', 'UK', 'Spain', 'France', 'Canada'],
      dtype=object)
```

In [81]:
```
movies['IFUS'].unique()
```

Out[81]:
```
array(['USA', 'non-USA'], dtype=object)
```

In [82]:
```
plt.figure(figsize=(15,8))
plt.subplot(2,2,1)
sns.boxplot(x=movies["IFUS"],y=movies["CVotesUS"])
plt.subplot(2,2,2)
sns.boxplot(x=movies["IFUS"],y=movies["CVotesnUS"])
plt.subplot(2,2,3)
sns.boxplot(x=movies["IFUS"],y=movies["VotesUS"])
plt.subplot(2,2,4)
sns.boxplot(x=movies["IFUS"],y=movies["VotesnUS"])
plt.show()
```



Inferences: Write your two inferences/observations below: a: CVotesUS(y) vs IFUS(x) b: VotesUS(y) vs IFUS(x)

- Inference 1: a) From both plots, we can see that non-USA plot's IQR is slightly larger thant USA people plot
  b) From both plots, we can see that there are some USA movies that have got exceptionally high rating from USA and non-USA people (outliers in USA plot)
- Inference 2: a) From both plots, there seem to be some outliers in USA plot, suggesting that some USA movies got exceptionally high votes from USA and non-USA people
  b) From both plots there seems to be trend here that states that USA people will rate USA movies higher and non-USA people will rate non-USA movies higher.

- ### Subtask 3.5: Top 1000 Voters Vs Genres

You might have also observed the column `CVotes1000`. This column represents the top 1000 voters on IMDb and gives the count for the number of these voters who have voted for a particular movie. Let's see how these top 1000 voters have voted across the genres.

1. Sort the dataframe genre_top10 based on the value of `CVotes1000` in a descending order.
2. Make a seaborn barplot for `genre` vs `CVotes1000`.
3. Write your inferences. You can also try to relate it with the heatmaps you did in the previous subtasks.

In [83]:
```
# Sorting by CVotes1000

genre_top10.sort_values(by='CVotes1000',ascending=False)
```

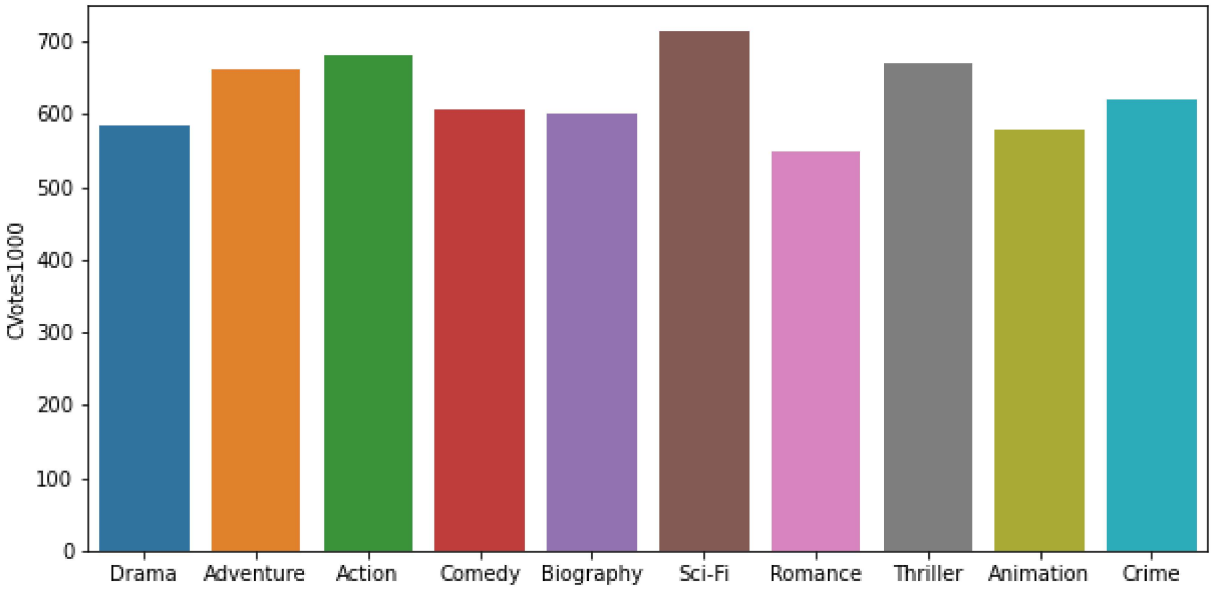
Out[83]:

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes
Sci-Fi	136781	148873	176646	106005	39518	14951	6583	3876	2
Action	102144	114433	150895	94262	34688	12693	5386	3064	2
Thriller	83207	112730	153336	90446	32003	11534	5021	2918	1
Adventure	94596	105636	138482	86367	31896	11551	4817	2718	1
Crime	52229	87919	129045	74671	25308	8971	3842	2246	1
Comedy	60157	77173	108993	69176	26099	9863	4237	2444	1
Biography	47333	77867	123948	74054	23644	7702	2984	1639	1
Drama	52375	75928	109339	66456	23528	8497	3622	2078	1
Animation	61960	72566	104837	65707	22825	7551	2792	1430	
Romance	42304	53037	82252	54833	21637	8530	3762	2130	1

10 rows × 45 columns

In [84]:
```
# Bar plot
plt.figure(figsize=(10,5))
sns.barplot(x=genre_top10.index,y=genre_top10["CVotes1000"])
```

Out[84]: <AxesSubplot:ylabel='CVotes1000'>



Inferences/observations here. 1 Sci-Fi seems to be the highest voted category here as well, as was the case in case of heatmap 2.Adventure, action and thriller being the next voted genres. 3.The genre Romance seems to be most unpopular among the top 1000 voters.