upGrad

#LifeKoKaroLift

Pre assignment session
On
IMDb Movies Assignment

Trainer: Dr. Darshan Ingle

upGrad

Course : Pre assignment session on IMDb Movies Assignment

Instructor: Dr. Darshan Ingle

Assignment: Problem Statement

upGrad

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.

This is a compulsory individual assignment wherein you will download a movie dataset, write Python code to explore the data, gain insights into the movies, actors, votes, ratings and collections, and submit the code.

Some tips before starting the assignment

upGrad

1. Go through the data dictionary thoroughly before starting with the assignment. It will give you a good sense of what all the columns represent which is a good practice to follow before proceeding with the analysis.
2. Read through each of the instructions carefully, identify the task to be performed, and only then proceed to write the required code. Don't perform any incorrect analysis or look for information that isn't required for the assignment.
3. In some cases, the variable names have already been assigned, and you just need to write code against them. In other cases, the names to be given are mentioned in the instructions. It is strongly advised that you use the mentioned names only.
4. Always keep inspecting your data frame after you have performed a particular set of operations.

Some tips before starting the assignment

upGrad

5. There might be some subtasks which involve the use of functions which you may not have used before. In such cases, you just need to do a simple search on Google/Stack Overflow to gain an understanding of the function. Please understand that solving this assignment is also a learning process and research is a part of this process. You have also been provided with some links directly in the notebook to make your search easier.

6. Always run the cells of the notebook sequentially/restart the kernel and run all the cells to avoid runtime errors. The number of commented cells provided in the notebook is an estimate based on the steps involved in each subtask. You can always add extra cells to the notebook for any additional steps you wish to perform.

Some tips before starting the assignment

upGrad

7. For plot-related questions, please make sure you are using the appropriate chart size for better readability. Please refer to the links provided in the notebook (can also search for links on your own) in the notebook for better formatting of your charts. If the task involves the comparison of two charts at a time, plot them side by side using subplots to ease the comparison.

8. For the questions that ask you to write your inferences based on a plot, there are no fixed answers for these questions, however, it is expected that you write some good pointers and it carries some marks.

9. There are some checkpoints given in the IPython notebook provided. They're just useful pieces of information you can use to check if the result you have obtained after performing a particular task is correct or not.

df.head()

Some tips before starting the assignment

upGrad

10. Optional subtasks are non-evaluative. However, I encourage you to solve those questions after you are done with the assignment as a good practice.

Lets start

upGrad

In [ ]:
```
# Filtering out the warnings

import warnings

warnings.filterwarnings('ignore')
```

In [ ]:
```
# Importing the required libraries

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

Lets start (Continued...)

upGrad

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.

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`.

In [ ]:
```
# Read the csv file using 'read_csv'. Please write your dataset location here.
```

movies = pd.read_csv("_____")

movies.head()

Lets start (Continued...)

upGrad

	Title	title_year	budget	Gross	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_likes	actor_3_facebool
0	La La Land	2016	30000000	151101803	Ryan Gosling	Emma Stone	Amiée Conn	14000	19000.0	
1	Zootopia	2016	150000000	341268248	Ginnifer Goodwin	Jason Bateman	Idris Elba	2800	28000.0	2
2	Lion	2016	12000000	51738905	Dev Patel	Nicole Kidman	Rooney Mara	33000	96000.0	
3	Arrival	2016	47000000	100546139	Amy Adams	Jeremy Renner	Forest Whitaker	35000	5300.0	
4	Manchester by the Sea	2016	9000000	47695371	Casey Affleck	Michelle Williams	Kyle Chandler	518	71000.0	

Lets start (Continued...)

upGrad

- **Subtask 1.2: Inspect the Dataframe**

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

In [ ]:
```
# Check the number of rows and columns in the dataframe
```

movies.shape

In [ ]:
```
# Check the column-wise info of the dataframe
```

movies.info()

In [ ]:
```
# Check the summary for the numeric columns
```

movies.describe()

Task 2: Data Analysis

No data cleaning

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 [ ]:
```
# Divide the 'gross' and 'budget' columns by 1000000 to convert '$' to 'million $'
```

movies['Gross'] = movies['Gross'] / 1000000

& Similarly for budget.

	Title	title_year	budget	Gross	actor_1_name	actor
0	La La Land	2016	30.0	151.101803	Ryan Gosling	Em
1	Zootopia	2016	150.0	341.268248	Ginnifer Goodwin	
2	Lion	2016	12.0	51.738905	Dev Patel	Nicol
3	Arrival	2016	47.0	100.546139	Amy Adams	
4	Manchester by the Sea	2016	9.0	47.695371	Casey Affleck	

5 rows × 62 columns

Lets start (Continued...)

upGrad

- **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`

(your territory)

In [ ]:
```
# Create the new column named 'profit' by subtracting the 'budget' column from the 'gross' column
```

movies['profit'] = movies['Gross'] - movies['budget']

movies.head()

esUS	VotesnUS	content_rating	Country	Profit
8.3	8.1	PG-13	USA	121.101803
8.0	8.0	PG	USA	191.268248
8.1	8.0	PG-13	Australia	39.738905
8.0	7.9	PG-13	USA	53.546139
7.9	7.8	R	USA	38.695371

Lets start (Continued...)

upGrad

In [ ]:
```
# Sort the dataframe with the 'profit' column as reference using the 'sort_values' function. Make sure to set the argument
#'ascending' to 'False'
```

descending

① movies.sort_values(by='Profit', ascending=False, inplace=True)

OR

② movies = movies.sort_values(by='Profit', ascending=False)

sUS	VotesnUS	content_rating	Country	Profit
8.2	7.9	PG-13	USA	691.662225
8.3	7.9	PG-13	USA	403.279547
8.1	7.9	R	USA	305.024263
7.7	7.4	PG-13	USA	294.645577
8.5	8.3	G	USA	214.984497

Lets start (Continued...)

upGrad

```
In [ ]: # Get the top 10 profitable movies by using position based indexing. Specify the rows till 10 (0-9)
```

top10= movies.iloc[:10,]

top10

	Title	title_year	budget	Gross	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_l
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	26
47	Deadpool	2016	58.0	363.024263	Ryan Reynolds	Ed Skrein	Stefan Kapicic	16
32	The Hunger Games: Catching Fire	2013	130.0	424.645577	Jennifer Lawrence	Josh Hutcherson	Sandra Ellis Lafferty	34
12	Toy Story 3	2010	200.0	414.984497	Tom Hanks	John Ratzenberger	Don Rickles	15
8	The Dark Knight Rises	2012	250.0	448.130642	Tom Hardy	Christian Bale	Joseph Gordon-Levitt	27
45	The Lego	2014	60.0	257.756197	Morgan	Will Ferrell	Alison Brie	11

Lets start (Continued...)

upGrad

```
In [ ]: #Plot profit vs budget
```

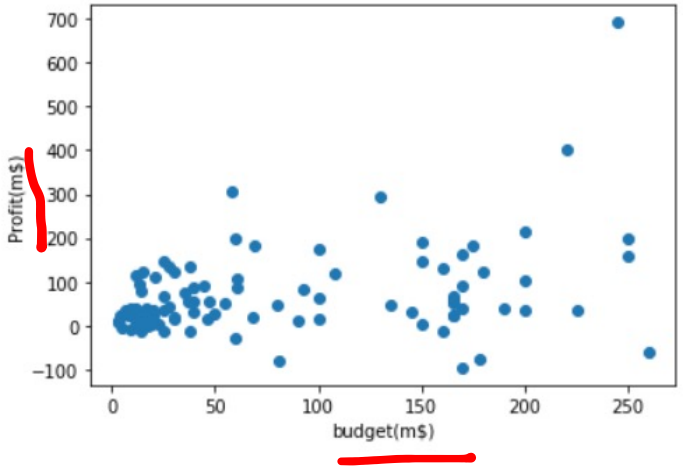
plt.scatter(movies['budget'], movies['profit'])

also give appropriate

{ xlabel()
  ylabel()

plt.xlabel("——")

plt.ylabel("——")



Lets start (Continued...)

upGrad

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.

In [ ]:
```
#Find the movies with negative profit
```

neg_profit = movies [movies['profit'] < 0]

000	VotesUS	VotesnUS	content_rating	Country	Profit
7.1	7.7	7.5	R	Canada	-4.776162
7.2	7.9	7.8	PG-13	France	-8.674623
7.1	7.9	8.1	R	UK	-11.096291
7.1	8.2	8.1	PG-13	USA	-11.348338
6.4	7.5	7.7	PG	USA	-12.247786
7.3	7.8	7.7	PG-13	USA	-13.594629

Lets start (Continued...)

upGrad

- **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.

As a part of this subtask, you are required to find out the highest rated movies which have been liked by critics and audiences alike.

1. Firstly you will notice that the `MetaCritic` score is on a scale of `100` whereas the `IMDb_rating` is on a scale of 10. First convert the `MetaCritic` column to a scale of 10.
2. Now, to find out the movies which have been liked by both critics and audiences alike and also have a high rating overall, you need to -
   - Create a new column `Avg_rating` which will have the average of the `MetaCritic` and `Rating` columns
   - Retain only the movies in which the absolute difference(using abs() function) between the `IMDb_rating` and `Metacritic` columns is less than 0.5. Refer to this link to know how abs() funtion works - https://www.geeksforgeeks.org/abs-in-python/ .
   - Sort these values in a descending order of `Avg_rating` and retain only the movies with a rating equal to higher than `8` and store these movies in a new dataframe `UniversalAcclaim` .

```
In [ ]: # Change the scale of MetaCritic

In [ ]: # Find the average ratings

In [ ]: #Sort in descending order of average rating

In [ ]: # Find the movies with metacritic-rating < 0.5 and also with the average rating of >8
```

Lets start (Continued...)

upGrad

- **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 [ ]:
```
# Extract Cols
```

	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_likes	actor_3_facebook_likes
94	Adam Goldberg	Lorelei Linklater	Libby Villari	230	193.0	127.0
69	QuvenzhanÃ© Wallis	Scoot McNairy	Taran Killam	2000	660.0	500.0
18	Amy Poehler	Mindy Kaling	Phyllis Smith	1000	767.0	384.0
12	Tom Hanks	John Ratzenberger	Don Rickles	15000	1000.0	721.0
4	Casey Affleck	Michelle Williams	Kyle Chandler	518	71000.0	3300.0

Lets start (Continued...)

upGrad

In [ ]:
```
# Find sums
```

actor['total_likes'] = actor['act_1_fblikes'] + actor['act_2_fblikes'] + actor['act_3_fblikes']

actor.head()

	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_likes	actor_3_facebook_likes	total_likes
94	Ellar Coltrane	Lorelei Linklater	Libby Villari	230	193.0	127.0	550.0
69	QuvenzhanÃ© Wallis	Scoot McNairy	Taran Killam	2000	660.0	500.0	3160.0
18	Amy Poehler	Mindy Kaling	Phyllis Smith	1000	767.0	384.0	2151.0
12	Tom Hanks	John Ratzenberger	Don Rickles	15000	1000.0	721.0	16721.0
4	Casey Affleck	Michelle Williams	Kyle Chandler	518	71000.0	3300.0	74818.0

Lets start (Continued...)

upGrad

```
In [ ]: # Sort the cols based on 'total_likes' in descending order and extract top 5 Actor trios
```

top5 = actor.sort_values(by='totalLikes', asc=F).head(5)

top5

	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_likes	actor_3_facebook_likes	total_likes
2	Dev Patel	Nicole Kidman	Rooney Mara	33000	96000.0	9800.0	138800.0
27	Leonardo DiCaprio	Tom Hardy	Joseph Gordon-Levitt	29000	27000.0	23000.0	79000.0
14	Jennifer Lawrence	Peter Dinklage	Hugh Jackman	34000	22000.0	20000.0	76000.0
4	Casey Affleck	Michelle Williams	Kyle Chandler	518	71000.0	3300.0	74818.0
8	Tom Hardy	Christian Bale	Joseph Gordon-Levitt	27000	23000.0	23000.0	73000.0

Lets start (Continued...)

upGrad

```
In [ ]: # Having found the top 5 popular trios, lets output their names in a list.
```

```
[['Dev Patel', 'Nicole Kidman', 'Rooney Mara'],
 ['Leonardo DiCaprio', 'Tom Hardy', 'Joseph Gordon-Levitt'],
 ['Jennifer Lawrence', 'Peter Dinklage', 'Hugh Jackman'],
 ['Casey Affleck', 'Michelle Williams ', 'Kyle Chandler'],
 ['Tom Hardy', 'Christian Bale', 'Joseph Gordon-Levitt']]
```

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

No Coding reqd.

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

1 | 2 | 3
70000 | 40000 | 50000
20000 | 35000 | 35000
25000 | 25000 | 20000

But the below one is not:

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

1 | 2 | 3
70000 | 40000 | 30000
20000 | 35000 | 35000 ×
15000 | 15000

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?

Write your answers below.

- `No. of trios that satisfy the above condition:`
- `Most popular trio after applying the condition:`

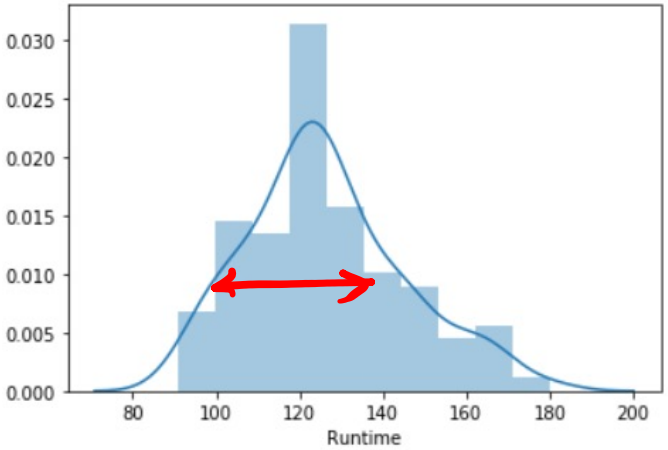
Lets start (Continued...)

upGrad

- **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 [ ]:
```
# Runtime histogram/density plot
```



- **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 [ ]: # Extract R rated movies that have been voted by the under-18 age group
```

r_movies = movies[movies['content_rating'] == 'R']

	Title	title_year	budget	Gross	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_l
57	The Town	2010	37.0	92.173235	Jeremy Renner	Jon Hamm	Owen Burke	1(
36	The Wolf of Wall Street	2013	100.0	116.866727	Leonardo DiCaprio	Matthew McConaughey	Jon Favreau	29
31	The Revenant	2015	135.0	183.635922	Leonardo DiCaprio	Tom Hardy	Lukas Haas	29
69	12 Years a Slave	2013	20.0	56.667870	QuvenzhanÃ© Wallis	Scoot McNairy	Taran Killam	2
47	Deadpool	2016	58.0	363.024263	Ryan Reynolds	Ed Skrein	Stefan Kapicic	16

5 rows × 64 columns

(42x64)

- **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 [ ]:
```
# 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.
```

PopularR = r_movies.sort_values(by='CVotesU18',ascending=False).head(10)

PopularR

	Title	title_year	budget	Gross	actor_1_name	actor_2_name	actor_3_n
47	Deadpool	2016	58.0	363.024263	Ryan Reynolds	Ed Skrein	Stefan Ka
36	The Wolf of Wall Street	2013	100.0	116.866727	Leonardo DiCaprio	Matthew McConaughey	Jon Fav
35	Django Unchained	2012	100.0	162.804648	Leonardo DiCaprio	Christoph Waltz	Ato Essar
29	Mad Max: Fury Road	2015	150.0	153.629485	Tom Hardy	Charlize Theron	ZoÃ« Kra
95	Whiplash	2014	3.3	13.092000	J.K. Simmons	Melissa Benoist	Chris Mu

5 rows × 64 columns

Lets start (Continued...)

upGrad

- **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.

Lets start (Continued...)

upGrad

In [ ]:
```
# 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.
# Create the dataframe df_by_genre
```

	genre_1	genre_2	genre_3	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	...	Votes1829F	Votes3044	Votes3044I
94	Drama	NaN	NaN	49673	62055	76838	52238	23789	10431	4906	...	7.8	7.8	7.
69	Biography	Drama	History	75556	126223	161460	83070	27231	9603	4021	...	8.2	8.0	7.
18	Animation	Adventure	Comedy	87509	113244	119801	67153	24210	8542	3349	...	8.3	8.1	8.
12	Animation	Adventure	Comedy	139773	149992	158704	88289	31291	11850	4859	...	8.4	8.2	8.
4	Drama	NaN	NaN	18191	33532	46596	29626	11879	4539	1976	...	7.8	7.7	7.

5 rows × 47 columns

Lets start (Continued...)

upGrad

In [ ]:
```
# Create a column cnt and initialize it to 1
```

M	Votes3044F	Votes45A	Votes45AM	Votes45AF	Votes1000	VotesUS	VotesnUS	cnt
.8	7.6	7.7	7.7	7.7	7.2	8.0	7.9	1
.9	8.0	7.8	7.8	8.1	7.7	8.3	8.0	1
.1	8.1	7.9	7.9	7.9	7.6	8.2	8.1	1
.2	8.3	8.1	8.1	8.1	8.1	8.5	8.3	1
.7	7.7	7.6	7.6	7.6	7.1	7.9	7.8	1

Lets start (Continued...)

upGrad

```
In [ ]: # Group the movies by individual genres
```

genre_3	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes02	CVotes01	...	Votes3044	Votes3044
Adventure	238060	285510	430062	260106	88580	29250	10820	5521	3598	8821	...	30.9	30
Comedy	583404	653362	882294	559835	200937	68167	26488	14258	9307	24617	...	54.8	54
Crime	171660	236650	250667	129164	46715	18682	8674	5854	4258	9689	...	8.0	8
Drama	400221	680085	1167327	748493	258717	88338	35439	19075	12475	26948	...	91.8	91
Family	29228	40728	77893	62936	27932	11179	4664	2674	1700	3023	...	7.4	7
Fantasy	301836	311392	442460	308676	120911	46269	19555	11362	7808	24139	...	30.4	30
History	135504	227547	311209	159262	48678	16055	6307	3649	2729	8413	...	23.7	23
Music	74245	71191	64640	38831	17377	8044	3998	2839	2407	6802	...	7.9	7
Mystery	274446	443661	654167	375087	128131	44818	18755	10578	7149	17825	...	30.8	30
Romance	319534	418790	715954	497486	193588	75675	32615	18250	12492	25186	...	60.2	60
Sci-Fi	1975041	2169036	2569011	1517219	546668	200825	87463	50835	35424	86434	...	117.8	117
Sport	95717	140282	214806	138600	44470	13821	5155	2509	1828	4083	...	15.5	15
Thriller	456909	756067	1258608	810665	280154	97239	39547	22382	15051	32213	...	76.5	76
War	36753	54703	111271	82505	30231	10553	4303	2388	1629	3246	...	7.4	7
Western	21094	40901	91825	67175	23055	7191	2678	1305	779	1672	...	7.6	7

15 rows × 45 columns

Lets start (Continued...)

upGrad

In [ ]:
```
# Add the grouped data frames and store it in a new data frame
```

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes02	CVotes01	...	Votes3044	Votes3044M	Votes304
Action	3166467.0	3547429.0	4677755.0	2922126.0	1075354.0	393484.0	166970.0	95004.0	65573.0	171247.0	...	240.0	239.5	24
Adventure	3594659.0	4014192.0	5262328.0	3281981.0	1212075.0	438970.0	183070.0	103318.0	69737.0	173858.0	...	294.6	293.7	29
Animation	681562.0	798227.0	1153214.0	722782.0	251076.0	83069.0	30718.0	15733.0	10026.0	25193.0	...	85.4	84.9	8
Biography	852003.0	1401608.0	2231078.0	1332980.0	425595.0	138648.0	53718.0	29510.0	20613.0	51297.0	...	139.1	138.9	13
Comedy	1383616.0	1774987.0	2506851.0	1591069.0	600287.0	226852.0	97469.0	56218.0	39391.0	88367.0	...	177.4	177.4	17

5 rows × 45 columns

Lets start (Continued...)

upGrad

```
In [ ]: # Extract genres with atleast 10 occurences
```

45A	Votes45AM	Votes45AF	Votes1000	VotesUS	VotesnUS	cnt
37.0	236.4	240.4	226.2	247.6	240.6	31.0
91.7	290.4	298.0	280.6	303.5	296.2	38.0
84.5	84.1	86.7	80.0	87.6	86.1	11.0
38.5	137.9	141.7	130.1	142.7	139.9	18.0
75.0	174.7	177.1	165.4	182.6	178.9	23.0
83.9	83.8	84.5	81.3	87.8	85.8	11.0
96.8	495.3	503.2	469.5	515.9	506.0	65.0
97.8	97.5	98.9	89.9	101.8	100.1	13.0
31.1	130.8	131.5	127.9	137.5	134.0	17.0
99.6	99.3	100.7	96.2	103.1	101.5	13.0

Lets start (Continued...)

upGrad

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.

In [ ]:
```
# Take the mean for every column by dividing with cnt
```

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes02	CVotes01	...
Action	102144.096774	114433.193548	150895.322581	94262.129032	34688.838710	12693.032258	5386.129032	3064.645161	2115.258065	5524.096774	...
Adventure	94596.289474	105636.631579	138482.315789	86367.921053	31896.710526	11551.842105	4817.631579	2718.894737	1835.184211	4575.210526	...
Animation	61960.181818	72566.090909	104837.636364	65707.454545	22825.090909	7551.727273	2792.545455	1430.272727	911.454545	2290.272727	...
Biography	47333.500000	77867.111111	123948.777778	74054.444444	23644.166667	7702.666667	2984.333333	1639.444444	1145.166667	2849.833333	...
Comedy	60157.217391	77173.347826	108993.521739	69176.913043	26099.434783	9863.130435	4237.782609	2444.260870	1712.652174	3842.043478	...
Crime	52229.636364	87919.818182	129045.000000	74671.818182	25308.272727	8971.818182	3842.818182	2246.636364	1544.090909	3383.363636	...
Drama	52375.969231	75928.846154	109339.276923	66456.923077	23528.553846	8497.107692	3622.692308	2078.861538	1449.000000	3250.892308	...
Romance	42304.538462	53037.846154	82252.307692	54833.923077	21637.615385	8530.846154	3762.538462	2130.615385	1476.923077	3082.692308	...
Sci-Fi	136781.411765	148873.823529	176646.705882	106005.764706	39518.294118	14951.470588	6583.823529	3876.705882	2715.941176	6731.470588	...
Thriller	83207.769231	112730.076923	153336.769231	90446.076923	32003.538462	11534.846154	5021.615385	2918.461538	1982.076923	4433.076923	...

10 rows × 45 columns

Lets start (Continued...)

upGrad

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 [ ]:
```
# Converting CVotes to int type
```

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes02	CVotes01	...	Votes3044	Votes3044M	Votes
Action	102144	114433	150895	94262	34688	12693	5386	3064	2115	5524	...	7.741935	7.725806	7.8
Adventure	94596	105636	138482	86367	31896	11551	4817	2718	1835	4575	...	7.752632	7.728947	7.8
Animation	61960	72566	104837	65707	22825	7551	2792	1430	911	2290	...	7.763636	7.718182	7.9
Biography	47333	77867	123948	74054	23644	7702	2984	1639	1145	2849	...	7.727778	7.716667	7.7
Comedy	60157	77173	108993	69176	26099	9863	4237	2444	1712	3842	...	7.713043	7.713043	7.7
Crime	52229	87919	129045	74671	25308	8971	3842	2246	1544	3383	...	7.718182	7.763636	7.6
Drama	52375	75928	109339	66456	23528	8497	3622	2078	1449	3250	...	7.712308	7.709231	7.7
Romance	42304	53037	82252	54833	21637	8530	3762	2130	1476	3082	...	7.607692	7.607692	7.6
Sci-Fi	136781	148873	176646	106005	39518	14081	6583	3876	2715	6731	...	7.858824	7.852941	7.8
Thriller	83207	112730	153336	90446	32003	11534	5021	2918	1982	4433	...	7.738462	7.746154	7.7

10 rows × 45 columns

Lets start (Continued...)

upGrad

In [ ]:
```
# Rounding off the columns of Votes to two decimals
```

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes02	CVotes01	...	Votes3044	Votes3044M	Votes3044F
Action	102144	114433	150895	94262	34688	12693	5386	3064	2115	5524	...	7.74	7.73	7.80
Adventure	94596	105636	138482	86367	31896	11551	4817	2718	1835	4575	...	7.75	7.73	7.87
Animation	61960	72566	104837	65707	22825	7551	2792	1430	911	2290	...	7.76	7.72	7.98
Biography	47333	77867	123948	74054	23644	7702	2984	1639	1145	2849	...	7.73	7.72	7.77
Comedy	60157	77173	108993	69176	26099	9863	4237	2444	1712	3842	...	7.71	7.71	7.75
Crime	52229	87919	129045	74671	25308	8971	3842	2246	1544	3383	...	7.72	7.76	7.61
Drama	52375	75928	109339	66456	23528	8497	3622	2078	1449	3250	...	7.71	7.71	7.72
Romance	42304	53037	82252	54833	21637	8530	3762	2130	1476	3082	...	7.61	7.61	7.66
Sci-Fi	136781	148873	176646	106005	39518	14951	6583	3876	2715	6731	...	7.86	7.85	7.84
Thriller	83207	112730	153336	90446	32003	11534	5021	2918	1982	4433	...	7.74	7.75	7.70

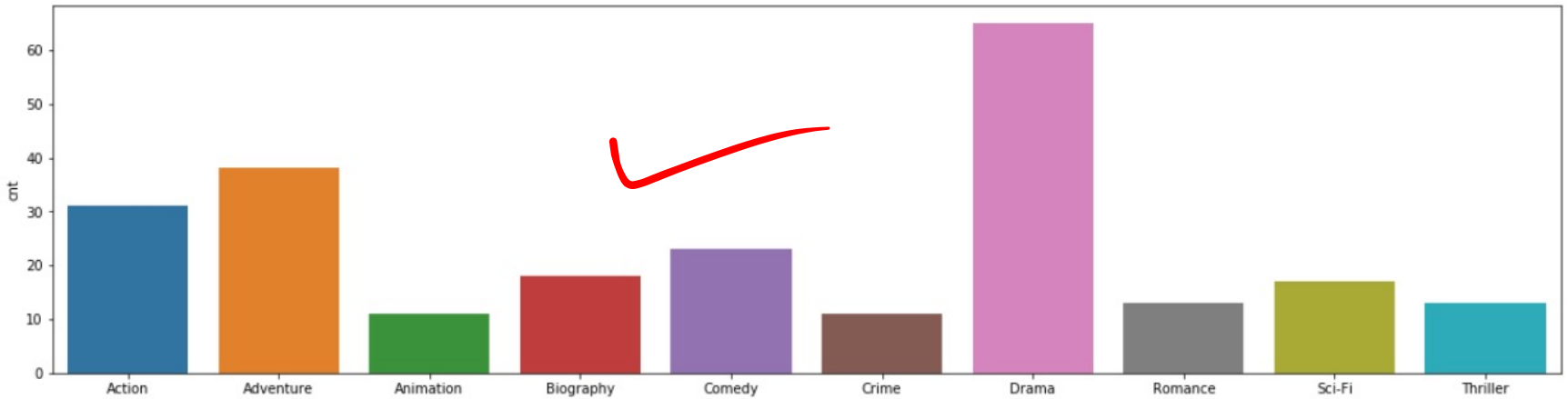
10 rows × 45 columns

- **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 [ ]:
```
# Countplot for genres
```

plt.figure(figsize=(20,5))
sns.barplot(x=genre_top10.index, y=genre_top10['cnt']);



Lets start (Continued...)

upGrad

- **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.

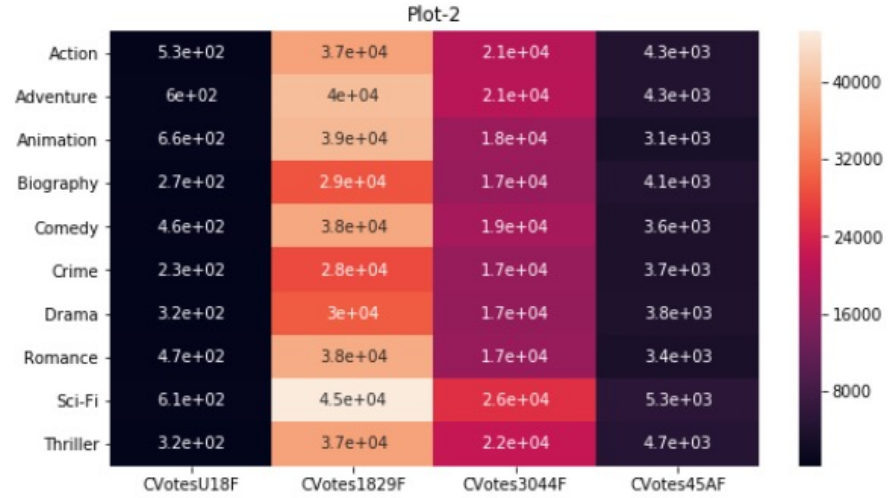
You might need the below link for formatting your heatmap. https://stackoverflow.com/questions/56942670/matplotlib-seaborn-first-and-last-row-cut-in-half-of-heatmap-plot

- Note : Use `genre_top10` dataframe for this subtask

Lets start (Continued...)

upGrad

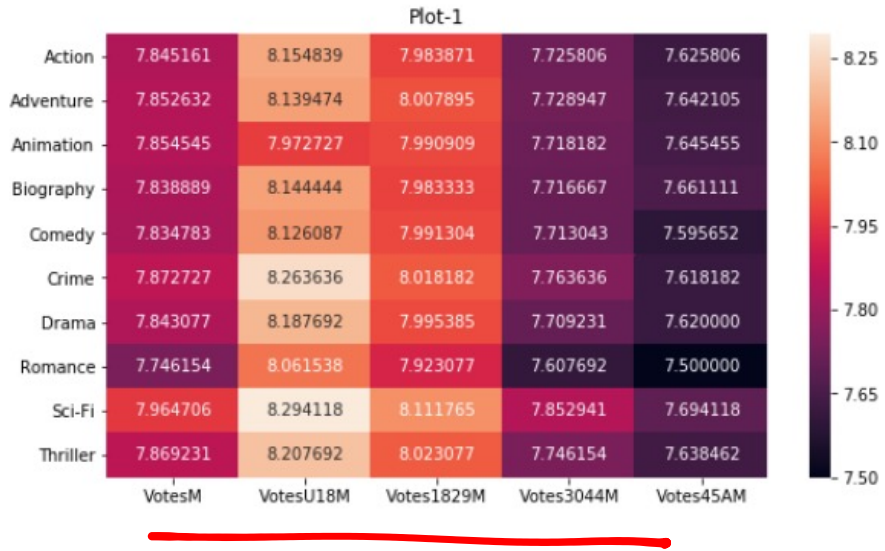
```
In [ ]: # 1st set of heat maps for CVotes-related columns
```

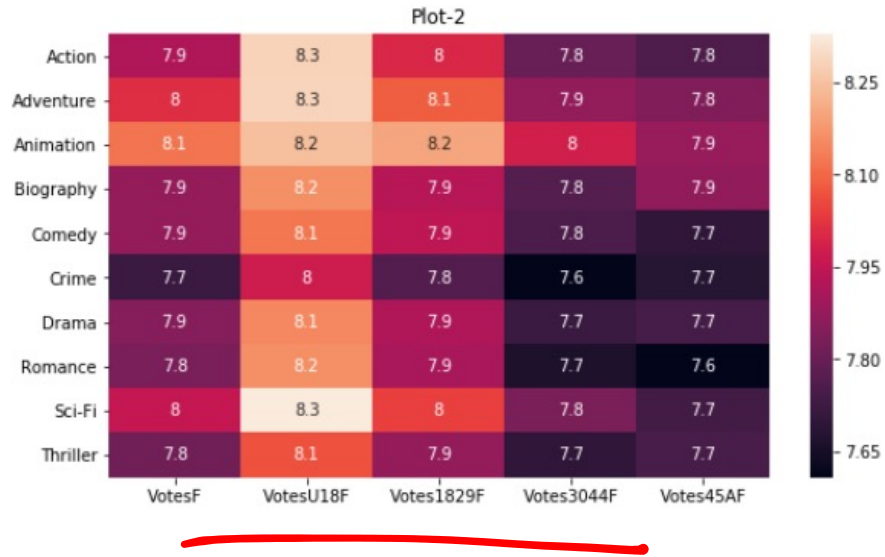


Lets start (Continued...)

upGrad

```
In [ ]: # 2nd set of heat maps for Votes-related columns
```





Lets start (Continued...)

upGrad

- **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`.
2. 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.
3. 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

Lets start (Continued...)

upGrad

```
In [ ]: # Creating IFUS column
```

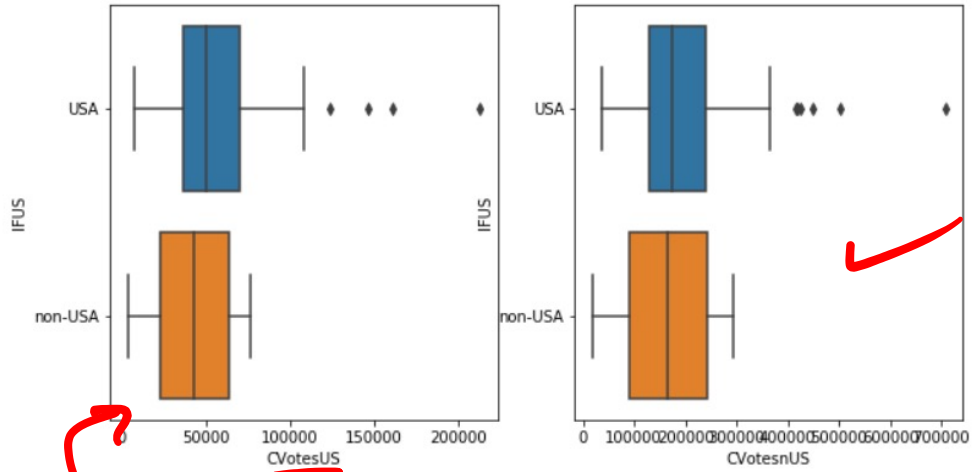
movies['IFUS'] = movies['Country'].apply(lambda x: "USA" if x == 'USA' else "nonUSA")

s45AF	Votes1000	VotesUS	VotesnUS	content_rating	Country	Profit	avg_rating	IFUS
7.8	7.6	8.1	7.9	R	USA	48.635922	7.80	USA
8.1	7.2	7.9	7.8	PG-13	France	-8.674623	8.65	non-USA
7.4	7.3	8.0	7.6	PG-13	USA	56.917897	8.60	USA
7.5	7.2	7.9	7.6	PG	USA	24.412677	7.45	USA
7.9	7.3	7.8	7.7	R	USA	5.613460	8.15	USA

Lets start (Continued...)

upGrad

```
In [ ]: # Box plot - 1: CVotesUS(y) vs IFUS(x)
```

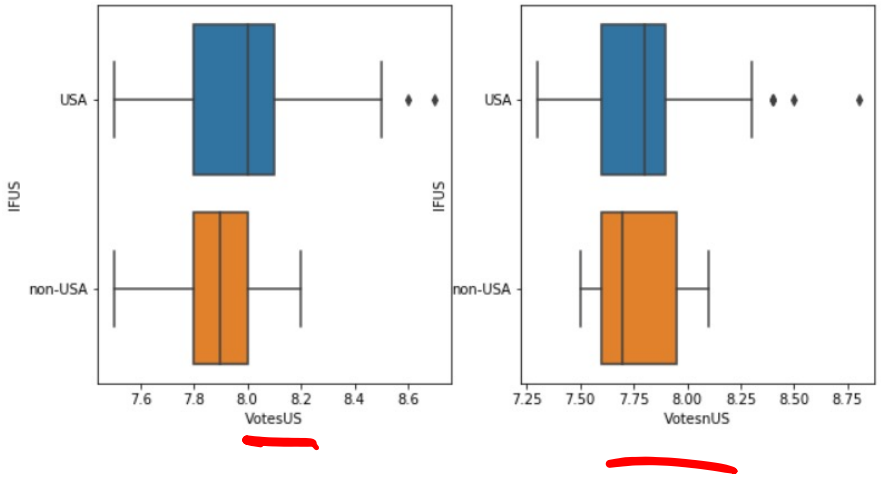


sns.boxplot(x='CVotesUS', y='IFUS', data=merged)

Lets start (Continued...)

upGrad

In [ ]:
```
# Box plot - 2: VotesUS(y) vs IFUS(x)
```



Lets start (Continued...)

upGrad

- **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 [ ]:
```
# Sorting by CVotes1000
```

genre_top10.sort_values(by='CVotes1000', asc=False, inplace=True)

```
genre_top10['CVotes1000']
```

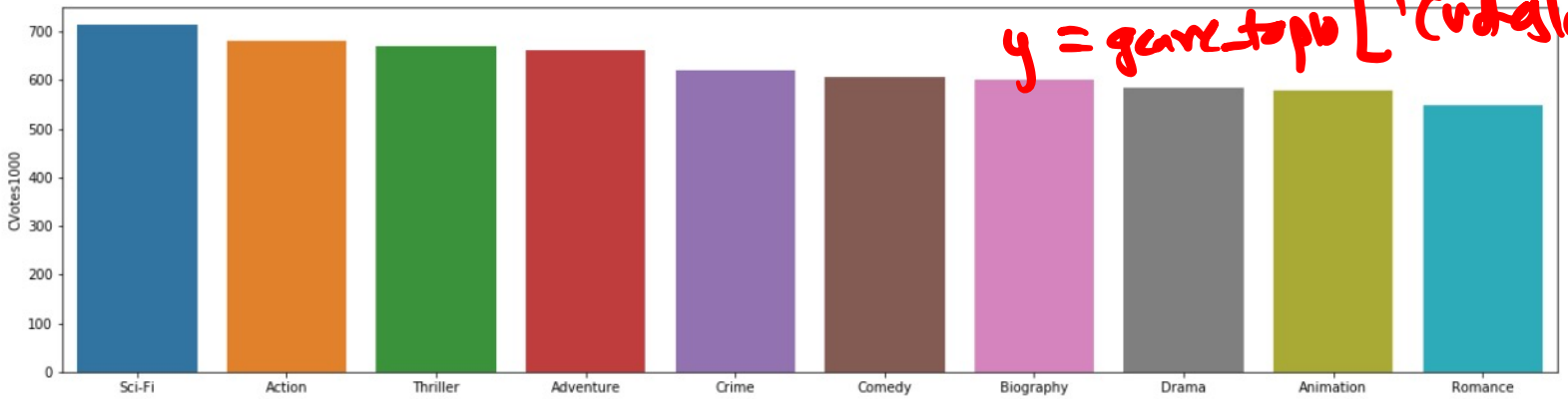
```
Sci-Fi        714
Action        681
Thriller      669
Adventure     662
Crime         620
Comedy        605
Biography     600
Drama         585
Animation     578
Romance       548
```

- **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 [ ]:
```
# Bar plot
```



upGrad

Congratulations you are on the right way to commence your assignment

Trainer: Dr. Darshan Ingle

upGrad

#LifeKoKaroLift

Thank You!

Connect me: https://www.linkedin.com/in/dr-darshan-ingle-corporate-trainer/