Exploratory Data Analysis of IMDB Movies Dataset using Python

In [1]:
```python
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [12]:
```python
# Read the IMDB movies data

movies = pd.read_csv("Movie Assignment Data.csv")
pd.set_option('display.max_columns',70)
movies.head()
```

Out[12]:

	Title	title_year	budget	Gross	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_lik
0	La La Land	2016	30000000	151101803	Ryan Gosling	Emma Stone	Amiée Conn	14000	1900
1	Zootopia	2016	150000000	341268248	Ginnifer Goodwin	Jason Bateman	Idris Elba	2800	2800
2	Lion	2016	12000000	51738905	Dev Patel	Nicole Kidman	Rooney Mara	33000	9600
3	Arrival	2016	47000000	100546139	Amy Adams	Jeremy Renner	Forest Whitaker	35000	530
4	Manchester by the Sea	2016	9000000	47695371	Casey Affleck	Michelle Williams	Kyle Chandler	518	7100

In [4]:
```python
#understanding the structure of the data

movies.shape
```

Out[4]: (100, 62)

In [5]:
```python
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]:
```python
#summary statistics of numerical columns

movies.describe(include="all")
```

Out[8]:

	Title	title_year	budget	Gross	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_face
count	100	100.000000	1.000000e+02	1.000000e+02	100	100	100	100.000000	
unique	100	NaN	NaN	NaN	75	94	96	NaN	
top	La La Land	NaN	NaN	NaN	Leonardo DiCaprio	Tom Hardy	Mackenzie Foy	NaN	
freq	1	NaN	NaN	NaN	5	2	2	NaN	
mean	NaN	2012.820000	7.838400e+07	1.468679e+08	NaN	NaN	NaN	13407.270000	7
std	NaN	1.919491	7.445295e+07	1.454004e+08	NaN	NaN	NaN	10649.037862	13
min	NaN	2010.000000	3.000000e+06	2.238380e+05	NaN	NaN	NaN	39.000000	
25%	NaN	2011.000000	1.575000e+07	4.199752e+07	NaN	NaN	NaN	1000.000000	
50%	NaN	2013.000000	4.225000e+07	1.070266e+08	NaN	NaN	NaN	13000.000000	1
75%	NaN	2014.000000	1.500000e+08	2.107548e+08	NaN	NaN	NaN	20000.000000	11
max	NaN	2016.000000	2.600000e+08	9.366622e+08	NaN	NaN	NaN	35000.000000	96

In [13]:
```python
#converting budget and gross columns from $ to million $ for better readability
movies['budget'] = movies['budget'].floordiv(1000000)
movies['Gross'] = movies['Gross'].floordiv(1000000)
```

In [14]:
```python
movies.head()
```

Out[14]:

	Title	title_year	budget	Gross	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_likes
0	La La Land	2016	30	151	Ryan Gosling	Emma Stone	Amiée Conn	14000	19000.0
1	Zootopia	2016	150	341	Ginnifer Goodwin	Jason Bateman	Idris Elba	2800	28000.0
2	Lion	2016	12	51	Dev Patel	Nicole Kidman	Rooney Mara	33000	96000.0
3	Arrival	2016	47	100	Amy Adams	Jeremy Renner	Forest Whitaker	35000	5300.0
4	Manchester by the Sea	2016	9	47	Casey Affleck	Michelle Williams	Kyle Chandler	518	71000.0

Profit

- create a new column Profit = Gross - budget
- sort the data according to profit and finding the top 10 profitable movies

In [15]:
```python
movies.insert(4, column = 'Profit', value = movies['Gross'] - movies['budget'])
```

In [16]:
```python
movies.head()
```

Out[16]:

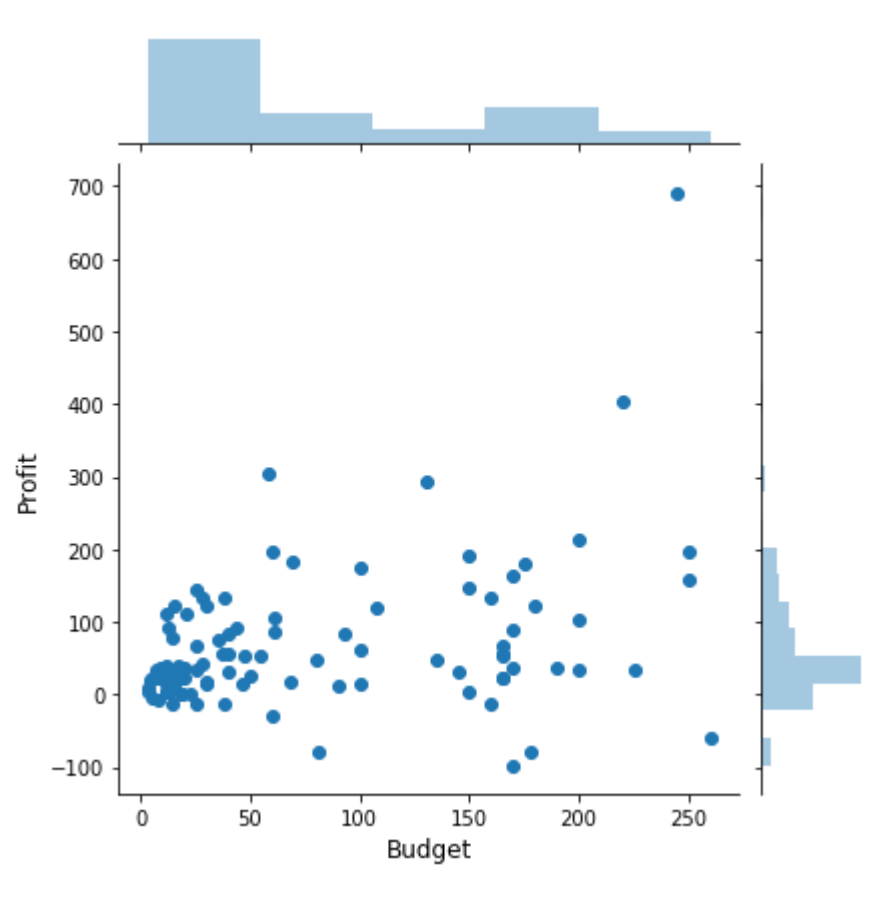
	Title	title_year	budget	Gross	Profit	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_lik
0	La La Land	2016	30	151	121	Ryan Gosling	Emma Stone	Amiée Conn	14000	19000
1	Zootopia	2016	150	341	191	Ginnifer Goodwin	Jason Bateman	Idris Elba	2800	28000
2	Lion	2016	12	51	39	Dev Patel	Nicole Kidman	Rooney Mara	33000	96000
3	Arrival	2016	47	100	53	Amy Adams	Jeremy Renner	Forest Whitaker	35000	5300
4	Manchester by the Sea	2016	9	47	38	Casey Affleck	Michelle Williams	Kyle Chandler	518	71000

In [17]:
```python
movies.sort_values(by=['Profit'], ascending=False, inplace = True, ignore_index = True)
top10_movies = movies.loc[0:9,'Title']
top10_movies
```

Out[17]:
```
0    Star Wars: Episode VII - The Force Awakens
1                                  The Avengers
2                                      Deadpool
3                 The Hunger Games: Catching Fire
4                                    Toy Story 3
5                         The Dark Knight Rises
6                                The Lego Movie
7                                      Zootopia
8                                 Despicable Me
9                                    Inside Out
Name: Title, dtype: object
```

In [20]:
```python
plt.figure(figsize=[8,6])
h = sns.jointplot(x='budget',y='Profit', data = movies)
h.set_axis_labels('Budget','Profit',fontsize=12)
plt.show()
```

<Figure size 576x432 with 0 Axes>



Inference

- Most of the movies are profitable, there is a slight positive trend between 'budget' and 'profit'
- There are some big budget movies with negative profits.
- between a budget of 0 - 100 million, there are a lot of profitable movies with upto 300 million profit.

Worst Performing Movies (Negative Profit)

In [21]:
```python
negative_profit = movies[movies['Profit']<0]
negative_profit.reset_index(drop = True)
#Worst performing 5 movies
negative_profit.sort_values(by='Profit')[0:5]
```

Out[21]:

	Title	title_year	budget	Gross	Profit	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_like
99	Hugo	2011	170	73	-97	ChloÃ« Grace Moretz	Christopher Lee	Ray Winstone	17000	16000
98	The Little Prince	2015	81	1	-80	Jeff Bridges	James Franco	Mackenzie Foy	12000	11000
97	Edge of Tomorrow	2014	178	100	-78	Tom Cruise	Lara Pulver	Noah Taylor	10000	854
96	Tangled	2010	260	200	-60	Brad Garrett	Donna Murphy	M.C. Gainey	799	553
95	Scott Pilgrim vs. the World	2010	60	31	-29	Anna Kendrick	Kieran Culkin	Ellen Wong	10000	1000

Movie Ratings Analysis

- MetaCritic (Critics opinion) rating and IMDB rating (public opinion)
- Average overall rating of a movie will be the average of both the ratings

In [22]:
```python
# MetaCritic rating is out of 100, so to make it into the same scale as IMDB rating
movies['MetaCritic'] = movies['MetaCritic']/10
```

In [23]:
```python
movies.insert(18,column='Avg Rating', value = (movies['MetaCritic']+movies['IMDb_rating'])/2)
```

In [24]:
```python
movies.head()
```

Out[24]:

	Title	title_year	budget	Gross	Profit	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_likes
0	Star Wars: Episode VII - The Force Awakens	2015	245	936	691	Doug Walker	Rob Walker	0	131	12.0
1	The Avengers	2012	220	623	403	Chris Hemsworth	Robert Downey Jr.	Scarlett Johansson	26000	21000.0
2	Deadpool	2016	58	363	305	Ryan Reynolds	Ed Skrein	Stefan Kapicic	16000	805.0
3	The Hunger Games: Catching Fire	2013	130	424	294	Jennifer Lawrence	Josh Hutcherson	Sandra Ellis Lafferty	34000	14000.0
4	Toy Story 3	2010	200	414	214	Tom Hanks	John Ratzenberger	Don Rickles	15000	1000.0

In [25]:
```python
#sorting movies by their avg rating (descending)
movies.sort_values(by='Avg Rating', ascending=False, inplace=True)
movies.head()
```

Out[25]:

	Title	title_year	budget	Gross	Profit	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_li
69	Boyhood	2014	4	25	21	Ellar Coltrane	Lorelei Linklater	Libby Villari	230	19
51	12 Years a Slave	2013	20	56	36	QuvenzhanÃ© Wallis	Scoot McNairy	Taran Killam	2000	66
9	Inside Out	2015	175	356	181	Amy Poehler	Mindy Kaling	Phyllis Smith	1000	76
4	Toy Story 3	2010	200	414	214	Tom Hanks	John Ratzenberger	Don Rickles	15000	100
49	Manchester by the Sea	2016	9	47	38	Casey Affleck	Michelle Williams	Kyle Chandler	518	7100

Most Popular Actor Trios

- popoularity based on facebook likes of lead actors in the movie

In [26]:
```python
movies.insert(11,column='Popularity', value=movies.loc[:,['actor_1_facebook_likes','actor_2_facebook_likes'
                                                          ,'actor_3_facebook_likes']].sum(axis=1))
```

In [27]:
```python
#top 5
most_popular_actors = movies.sort_values(by='Popularity',ascending=False, ignore_index=True).loc[0:4,:]
most_popular_actors
```

Out[27]:

	Title	title_year	budget	Gross	Profit	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_lik
0	Lion	2016	12	51	39	Dev Patel	Nicole Kidman	Rooney Mara	33000	96000
1	Inception	2010	160	292	132	Leonardo DiCaprio	Tom Hardy	Joseph Gordon-Levitt	29000	27000
2	X-Men: Days of Future Past	2014	200	233	33	Jennifer Lawrence	Peter Dinklage	Hugh Jackman	34000	22000
3	Manchester by the Sea	2016	9	47	38	Casey Affleck	Michelle Williams	Kyle Chandler	518	71000
4	The Dark Knight Rises	2012	250	448	198	Tom Hardy	Christian Bale	Joseph Gordon-Levitt	27000	23000

In [29]:
```python
most_popular_actor_trios = movies.sort_values(by='Popularity',ascending=False, ignore_index=True).loc[0
:4,['actor_1_name','actor_2_name','actor_3_name']].values.tolist()
most_popular_actor_trios
```

Out[29]:
```
[['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']]
```

In [30]:
```python
#none of the three actors’ Facebook likes should be less than half of the other two

individual_popular_trio = movies[~((((movies['actor_1_facebook_likes'] < movies['actor_2_facebook_like
s']/2)|
                                     (movies['actor_1_facebook_likes'] < movies['actor_3_facebook_like
s']/2))==True)
                                   |(((movies['actor_2_facebook_likes'] < movies['actor_1_facebook_like
s']/2)|
                                     (movies['actor_2_facebook_likes'] < movies['actor_3_facebook_like
s']/2))==True)
                                   |(((movies['actor_3_facebook_likes'] < movies['actor_1_facebook_lik
es']/2)|
                                     (movies['actor_3_facebook_likes'] < movies['actor_2_facebook_like
s']/2))==True))]
```

In [31]:
```python
#common in the two dfs

common_actors  = most_popular_actors[most_popular_actors['Title'].isin(individual_popular_trio['Title'
])]
common_actors.loc[:,['actor_1_name','actor_2_name','actor_3_name']].values.tolist()
```

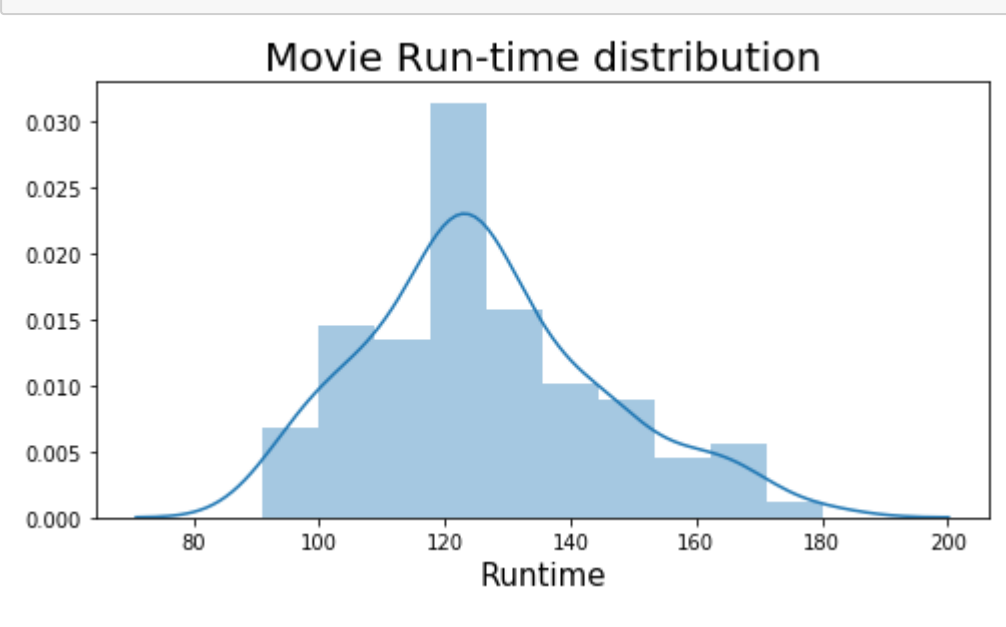
Out[31]:
```
[['Leonardo DiCaprio', 'Tom Hardy', 'Joseph Gordon-Levitt'],
 ['Jennifer Lawrence', 'Peter Dinklage', 'Hugh Jackman'],
 ['Tom Hardy', 'Christian Bale', 'Joseph Gordon-Levitt']]
```

In [32]:
```python
top_popular = individual_popular_trio.sort_values(by='Popularity',ascending=False)[0:5].reset_index(dro
p=True)
top_popular.loc[:,['actor_1_name','actor_2_name','actor_3_name']].values.tolist()
```

Out[32]:
```
[['Leonardo DiCaprio', 'Tom Hardy', 'Joseph Gordon-Levitt'],
 ['Jennifer Lawrence', 'Peter Dinklage', 'Hugh Jackman'],
 ['Tom Hardy', 'Christian Bale', 'Joseph Gordon-Levitt'],
 ['Chris Hemsworth', 'Robert Downey Jr.', 'Scarlett Johansson'],
 ['Philip Seymour Hoffman', 'Robin Wright', 'Brad Pitt']]
```

Movie Run-time Analysis

In [33]:
```python
#movie run time histogram density plot

plt.figure(figsize=[8,4])
runtime = sns.distplot(movies['Runtime'])
runtime.axes.set_title("Movie Run-time distribution",fontsize=20)
runtime.set_xlabel("Runtime",fontsize=15)
plt.show()
```



- Most movies have a run-time between 120 to 130 minutes

Movie Genre Analysis

In [35]:
```python
#creating dataframe by genre
columns = []

for i in movies.columns:
    if i.startswith('CV')|i.startswith('V')|i.startswith("ge"):
        columns.append(i)

df_genre = movies.loc[:,columns]

df_genre['cnt'] = 1
#grouping movies by individual genre
df_g1 = df_genre.groupby('genre_1')
df_g2 = df_genre.groupby('genre_2')
df_g3 = df_genre.groupby('genre_3')

#finding the sum for genre1
df_g1 = pd.DataFrame(df_g1.sum())
df_g2 = pd.DataFrame(df_g2.sum())
df_g3 = pd.DataFrame(df_g3.sum())
```

In [36]:
```python
add_df1_df2 = df_g1.add(df_g2,fill_value=0)
df_add=add_df1_df2.add(df_g3,fill_value=0)

genre_top10 = df_add[df_add['cnt'] >=10]
genre_top10.head()
```

Out[36]:

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes02	CVotes01	CVotesMale	CVot
Action	3166467.0	3547429.0	4677755.0	2922126.0	1075354.0	393484.0	166970.0	95004.0	65573.0	171247.0	10837034.0	
Adventure	3594659.0	4014192.0	5262328.0	3281981.0	1212075.0	438970.0	183070.0	103318.0	69737.0	173858.0	11759815.0	
Animation	681562.0	798227.0	1153214.0	722782.0	251076.0	83069.0	30718.0	15733.0	10026.0	25193.0	2282985.0	
Biography	852003.0	1401608.0	2231078.0	1332980.0	425595.0	138648.0	53718.0	29510.0	20613.0	51297.0	4329471.0	
Comedy	1383616.0	1774987.0	2506851.0	1591069.0	600287.0	226852.0	97469.0	56218.0	39391.0	88367.0	5223033.0	

In [37]:
```python
#extracting count column
count_column= genre_top10['cnt']
#taking the mean of every column
genre_top10=genre_top10.div(genre_top10['cnt'],axis='index')

genre_top10=genre_top10.apply(lambda x:round(x,2),axis=0)
genre_top10.drop(columns='cnt',inplace=True)
```

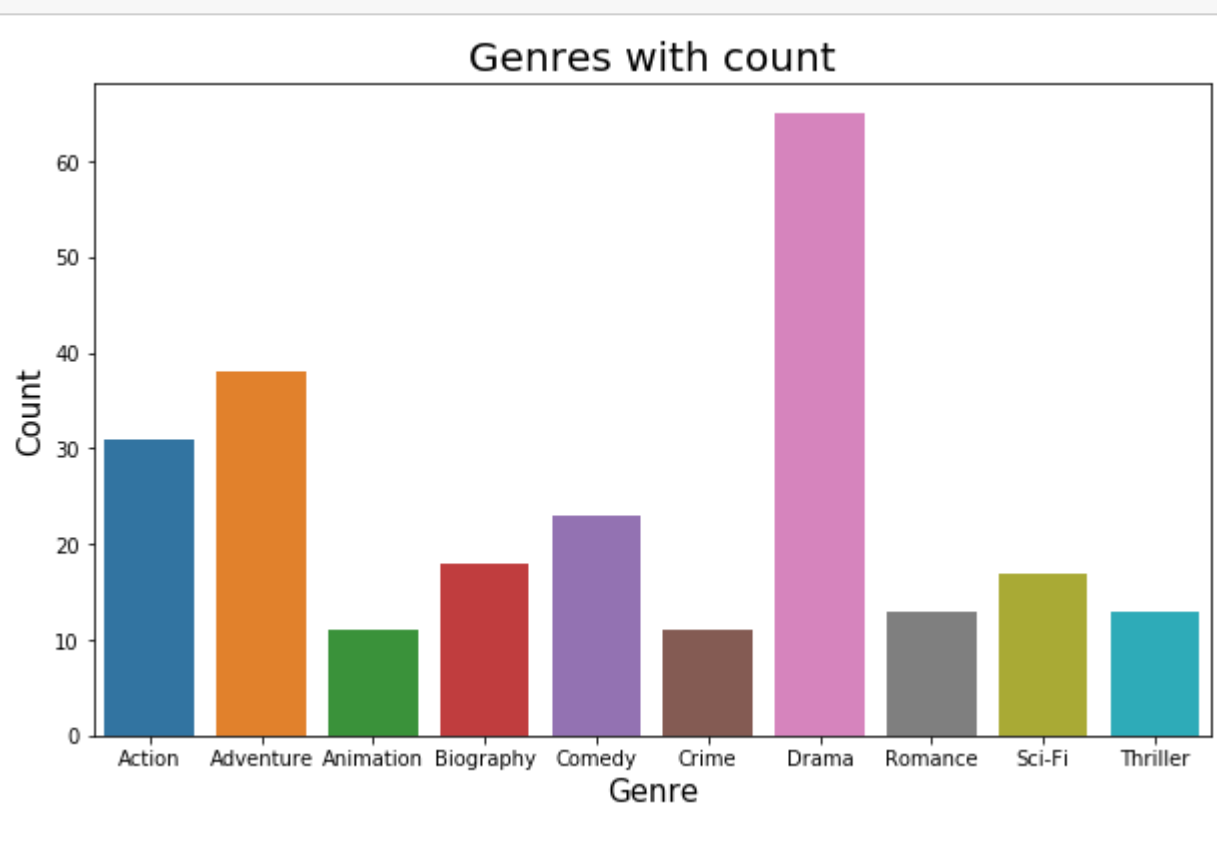
In [38]:
```python
#convert CVotes into int

col_cvotes=[]
for i in genre_top10.columns:
    if i.startswith('CVotes'):
        col_cvotes.append(i)

genre_top10[col_cvotes]=genre_top10[col_cvotes].astype('int')
genre_top10.head()
```

Out[38]:

	CVotes10	CVotes09	CVotes08	CVotes07	CVotes06	CVotes05	CVotes04	CVotes03	CVotes02	CVotes01	CVotesMale	CVotes
Action	102144	114433	150895	94262	34688	12693	5386	3064	2115	5524	349581	
Adventure	94596	105636	138482	86367	31896	11551	4817	2718	1835	4575	309468	
Animation	61960	72566	104837	65707	22825	7551	2792	1430	911	2290	207544	
Biography	47333	77867	123948	74054	23644	7702	2984	1639	1145	2849	240526	
Comedy	60157	77173	108993	69176	26099	9863	4237	2444	1712	3842	227088	

In [39]:
```python
cnt = count_column.tolist()
genre_top10['cnt'] = cnt
```

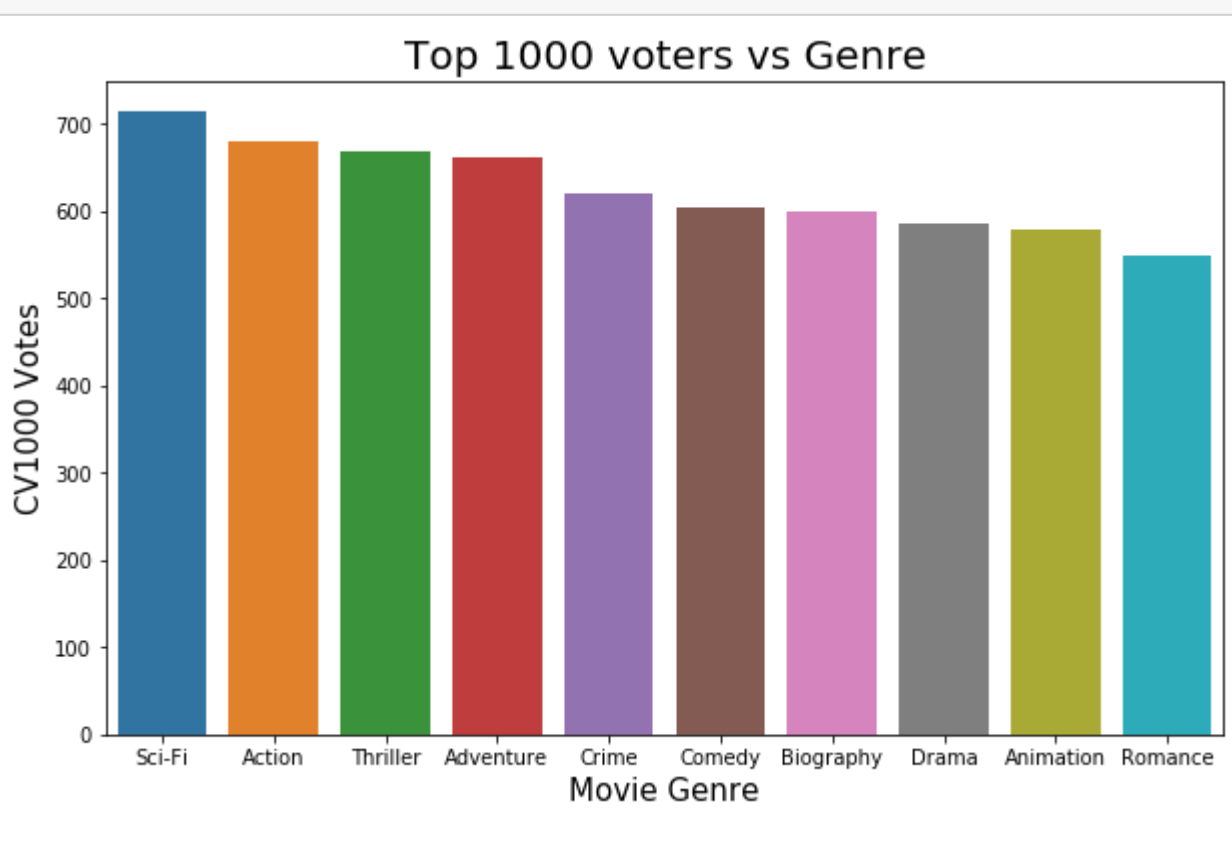
In [41]:
```python
plt.figure(figsize=[10,6])
sns.barplot(x=genre_top10.index, y = genre_top10.cnt)
plt.title('Genres with count',fontsize = 20)
plt.xlabel('Genre',fontsize=15)
plt.ylabel('Count',fontsize=15)
plt.show()
```



Top 1000 Voters in IMDB vs Genre

In [42]:
```python
genre_top10_cvotes = genre_top10.sort_values(by='CVotes1000',ascending = False)
```

In [43]:
```python
plt.figure(figsize=[10,6])
sns.barplot(x=genre_top10_cvotes.index, y = genre_top10_cvotes.CVotes1000)
plt.title('Top 1000 voters vs Genre',fontsize = 20)
plt.xlabel('Movie Genre',fontsize=15)
plt.ylabel('CV1000 Votes',fontsize=15)
plt.show()
```



Inferences

- Romance genre was voted the least by top 1000 voters
- Sci-Fi is the most popular
- Action, Thriller and Adventure are similar, as they are genres that goes hand in hand