Import the libraries

In [1]:
```
import pandas as pd
```

Read the dataset

In [2]:
```
movie = pd.read_csv(r'C:\Users\ADMIN\Downloads\Movie-Rating.csv')
```

In [3]:
```
movie
```

Out[3]:

	Film	Genre	Rotten Tomatoes Ratings %	Audience Ratings %	Budget (million $)	Year of release
0	(500) Days of Summer	Comedy	87	81	8	2009
1	10,000 B.C.	Adventure	9	44	105	2008
2	12 Rounds	Action	30	52	20	2009
3	127 Hours	Adventure	93	84	18	2010
4	17 Again	Comedy	55	70	20	2009
...	...	...	...	...	...	...
554	Your Highness	Comedy	26	36	50	2011
555	Youth in Revolt	Comedy	68	52	18	2009
556	Zodiac	Thriller	89	73	65	2007
557	Zombieland	Action	90	87	24	2009
558	Zookeeper	Comedy	14	42	80	2011

559 rows × 6 columns

Type of the dataset

In [4]:
```
type(movie)
```

Out[4]:
```
pandas.core.frame.DataFrame
```

length of the dataset (no.of rows are there in the dataset)

In [5]:
```
len(movie)
```

Out[5]:
```
559
```

import numpy library and check the versions

In [6]:
```
import numpy as np
np.__version__
```

Out[6]:
```
'1.26.4'
```

In [7]:
```
pd.__version__
```

Out[7]:
```
'2.2.2'
```

find how many columns are there in the given dataset

In [8]:
```
movie.columns
```

Out[8]:
```
Index(['Film', 'Genre', 'Rotten Tomatoes Ratings %', 'Audience Ratings %',
       'Budget (million $)', 'Year of release'],
      dtype='object')
```

.info() ---> Detect missing values,data types, memory size

In [9]:
```
movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 559 entries, 0 to 558
Data columns (total 6 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Film                       559 non-null    object
 1   Genre                      559 non-null    object
 2   Rotten Tomatoes Ratings %  559 non-null    int64
 3   Audience Ratings %         559 non-null    int64
 4   Budget (million $)         559 non-null    int64
 5   Year of release            559 non-null    int64
dtypes: int64(4), object(2)
memory usage: 26.3+ KB
```

.shape --> no of rows and columns

In [10]:
```
movie.shape
```

Out[10]: (559, 6)

.head() ---> by default gives the top 5 records

In [11]:
```
movie.head()
```

Out[11]:

	Film	Genre	Rotten Tomatoes Ratings %	Audience Ratings %	Budget (million $)	Year of release
0	(500) Days of Summer	Comedy	87	81	8	2009
1	10,000 B.C.	Adventure	9	44	105	2008
2	12 Rounds	Action	30	52	20	2009
3	127 Hours	Adventure	93	84	18	2010
4	17 Again	Comedy	55	70	20	2009

.tail() --> by default gives the bottom 5 records

In [12]:
```
movie.tail()
```

Out[12]:

	Film	Genre	Rotten Tomatoes Ratings %	Audience Ratings %	Budget (million $)	Year of release
554	Your Highness	Comedy	26	36	50	2011
555	Youth in Revolt	Comedy	68	52	18	2009
556	Zodiac	Thriller	89	73	65	2007
557	Zombieland	Action	90	87	24	2009
558	Zookeeper	Comedy	14	42	80	2011

In [13]:
```
movie.columns
```

Out[13]:
```
Index(['Film', 'Genre', 'Rotten Tomatoes Ratings %', 'Audience Ratings %',
       'Budget (million $)', 'Year of release'],
      dtype='object')
```

change the column names which is understandable by the organization

In [14]:
```
movie.columns = ['Film', 'Genre', 'CriticRating', 'AudienceRating','BudgetMillion', 'Year']
```

In [15]:
```
movie.head()
```

Out[15]:

	Film	Genre	CriticRating	AudienceRating	BudgetMillion	Year
0	(500) Days of Summer	Comedy	87	81	8	2009
1	10,000 B.C.	Adventure	9	44	105	2008
2	12 Rounds	Action	30	52	20	2009
3	127 Hours	Adventure	93	84	18	2010
4	17 Again	Comedy	55	70	20	2009

.describe() -->distribution of numerical data and descriptive satisticts

In [16]:
```
movie.describe()          # year dont have to
```

Out[16]:

	CriticRating	AudienceRating	BudgetMillion	Year
count	559.000000	559.000000	559.000000	559.000000
mean	47.309481	58.744186	50.236136	2009.152057
std	26.413091	16.826887	48.731817	1.362632
min	0.000000	0.000000	0.000000	2007.000000
25%	25.000000	47.000000	20.000000	2008.000000
50%	46.000000	58.000000	35.000000	2009.000000
75%	70.000000	72.000000	65.000000	2010.000000
max	97.000000	96.000000	300.000000	2011.000000

.describe().transpose() --->flips the rows and columns vise versa

In [17]:
```
movie.describe().transpose()
```

Out[17]:

	count	mean	std	min	25%	50%	75%	max
CriticRating	559.0	47.309481	26.413091	0.0	25.0	46.0	70.0	97.0
AudienceRating	559.0	58.744186	16.826887	0.0	47.0	58.0	72.0	96.0
BudgetMillion	559.0	50.236136	48.731817	0.0	20.0	35.0	65.0	300.0
Year	559.0	2009.152057	1.362632	2007.0	2008.0	2009.0	2010.0	2011.0

In [18]:
```
movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 559 entries, 0 to 558
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Film            559 non-null    object
 1   Genre           559 non-null    object
 2   CriticRating    559 non-null    int64
 3   AudienceRating  559 non-null    int64
 4   BudgetMillion   559 non-null    int64
 5   Year            559 non-null    int64
dtypes: int64(4), object(2)
memory usage: 26.3+ KB
```

.astype() ---> used to convert object datatype to another datatype

In [19]:
```
movie.Film = movie.Film.astype('category')
```

In [20]:
```
movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 559 entries, 0 to 558
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Film            559 non-null    category
 1   Genre           559 non-null    object
 2   CriticRating    559 non-null    int64
 3   AudienceRating  559 non-null    int64
 4   BudgetMillion   559 non-null    int64
 5   Year            559 non-null    int64
dtypes: category(1), int64(4), object(1)
memory usage: 43.6+ KB
```

In [21]:
```
movie.Genre = movie.Genre.astype('category')
```

In [22]:
```
movie.Year = movie.Year.astype('category')
```

In [23]:
```
movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 559 entries, 0 to 558
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Film            559 non-null    category
 1   Genre           559 non-null    category
 2   CriticRating    559 non-null    int64
 3   AudienceRating  559 non-null    int64
 4   BudgetMillion   559 non-null    int64
 5   Year            559 non-null    category
dtypes: category(3), int64(3)
memory usage: 36.5 KB
```

In [24]:
```
movie.describe()
```

Out[24]:

	CriticRating	AudienceRating	BudgetMillion
count	559.000000	559.000000	559.000000
mean	47.309481	58.744186	50.236136
std	26.413091	16.826887	48.731817
min	0.000000	0.000000	0.000000
25%	25.000000	47.000000	20.000000
50%	46.000000	58.000000	35.000000
75%	70.000000	72.000000	65.000000
max	97.000000	96.000000	300.000000

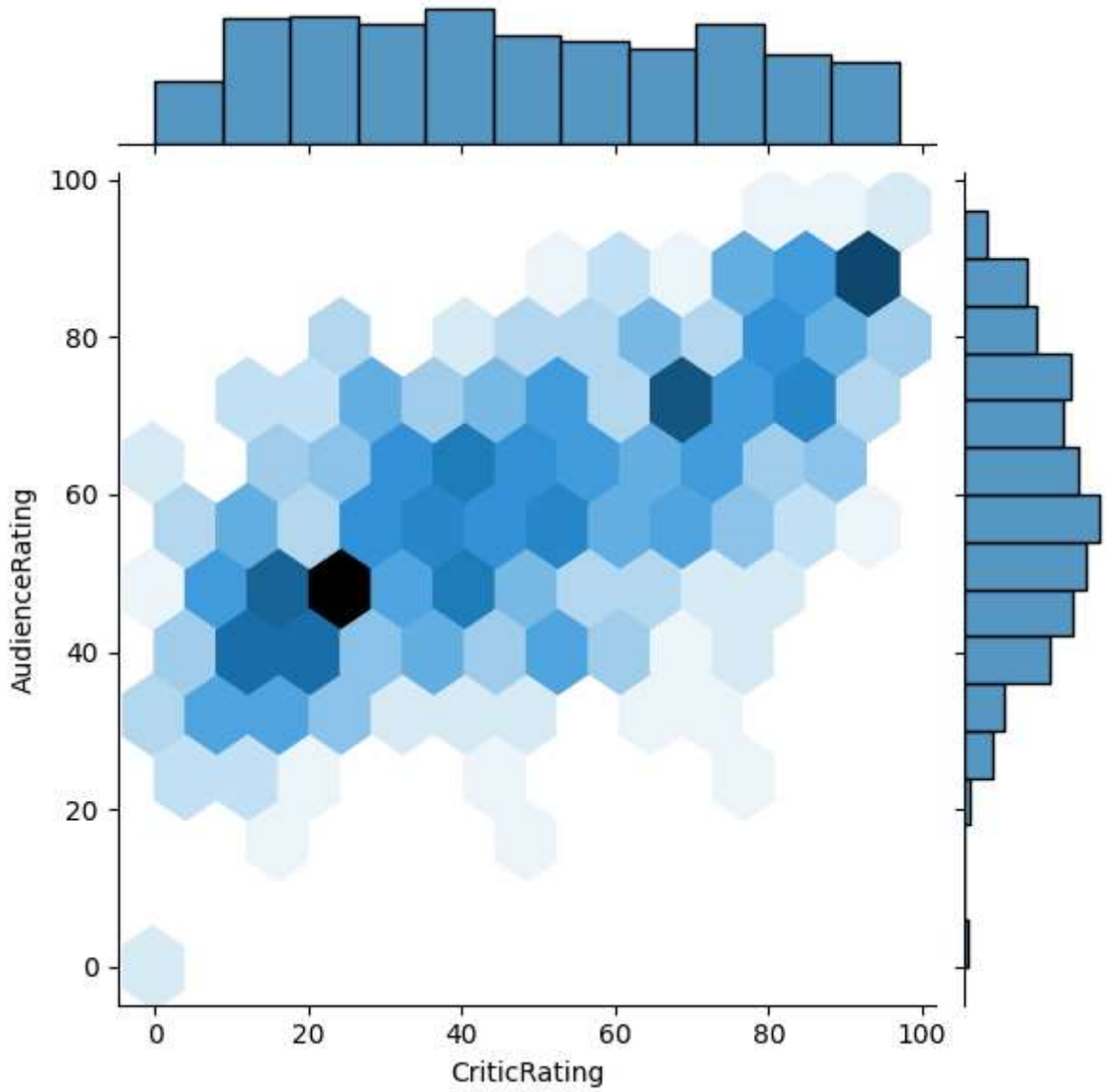
import the visualization libraries(matplotlib , seaborn) , warnings

In [25]:
```
#
import matplotlib
from matplotlib import pyplot as plt
import seaborn as sns
```

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

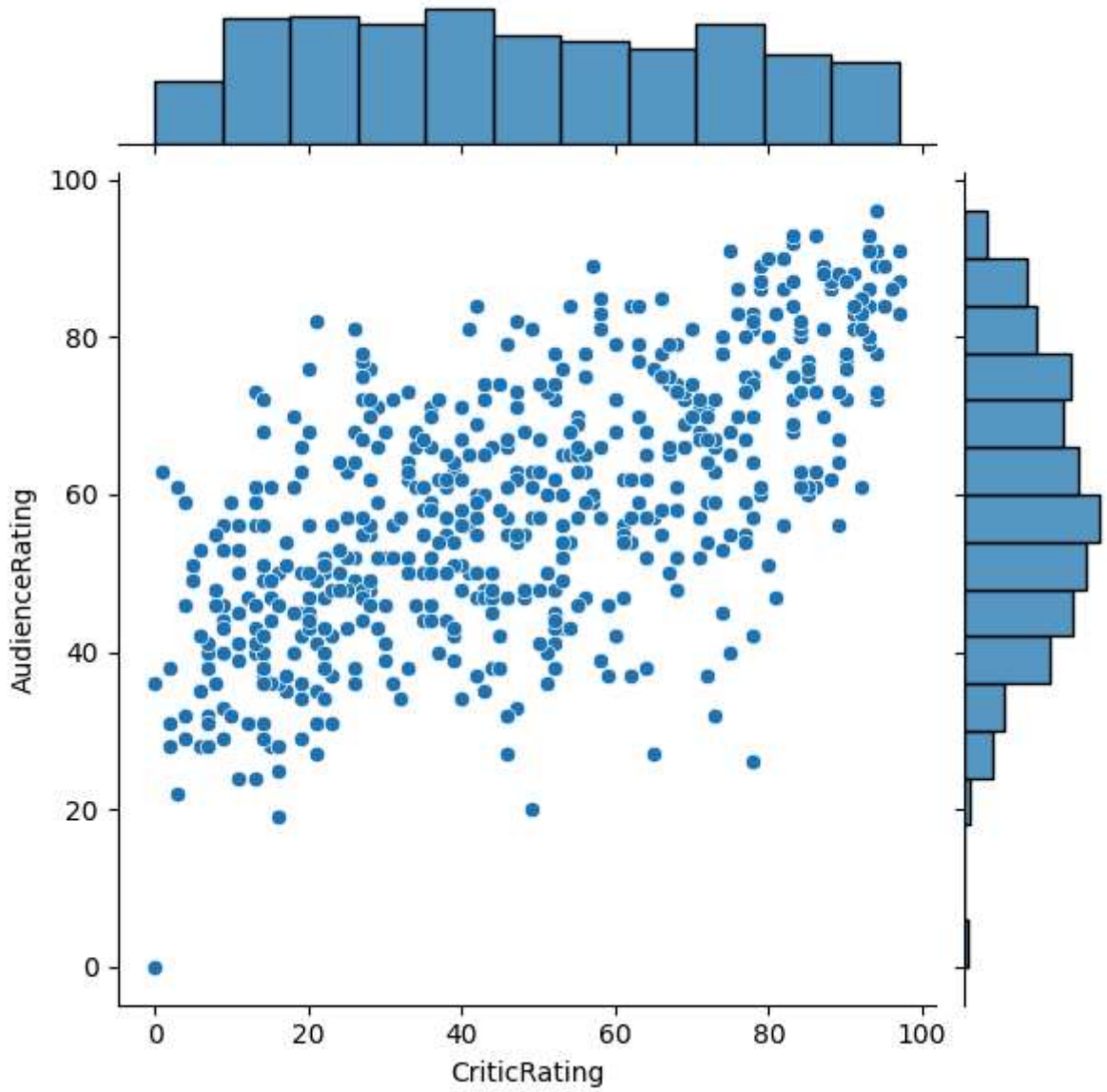
A joint plot shows the relationship between two variables along with their individual distributions. where kind = 'hex' [bivarient analaysis]

In [26]:
```
j = sns.jointplot(data = movie , x='CriticRating',y='AudienceRating',kind = 'hex' )
```



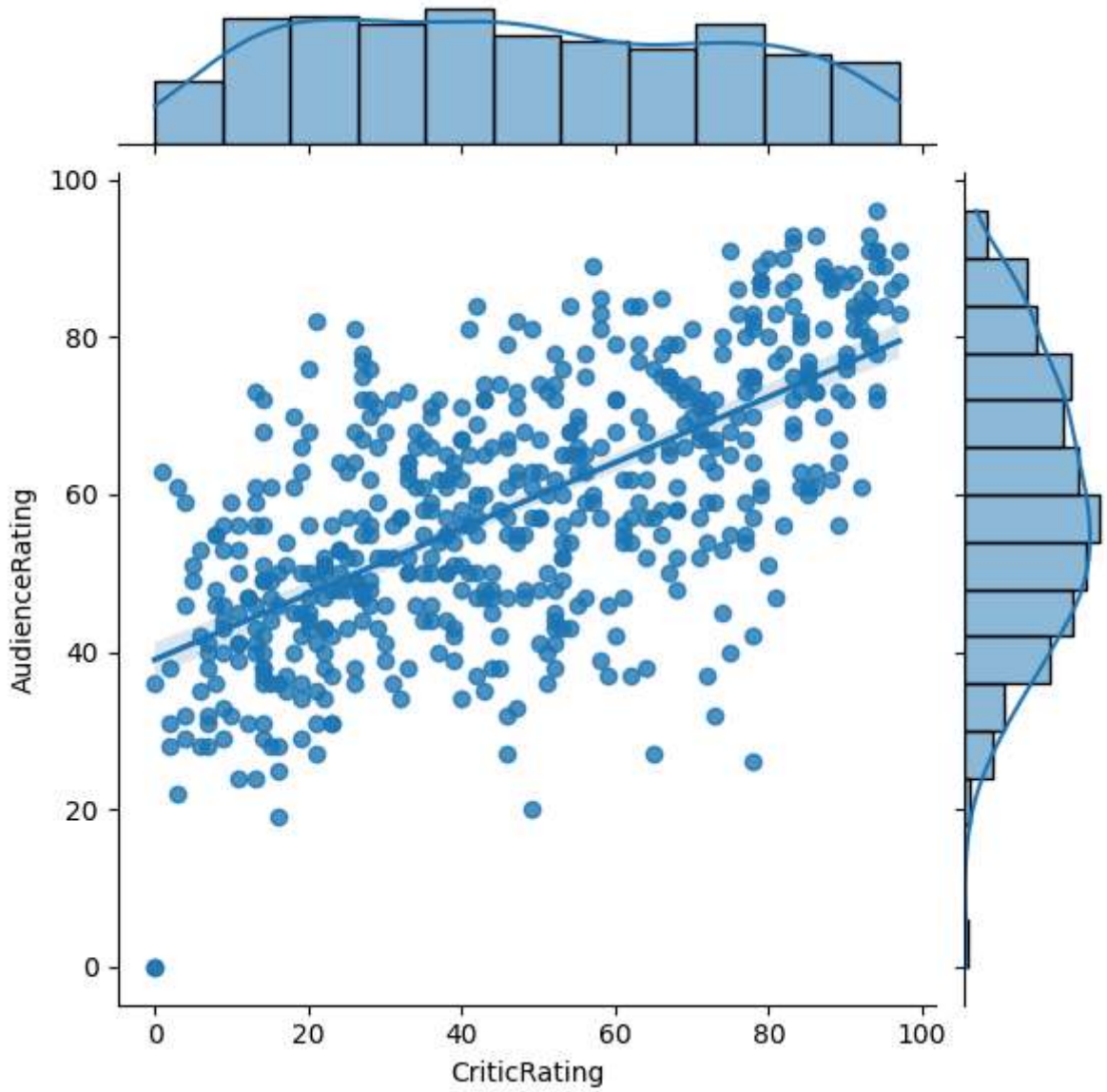
ploting a jointplot where kind = 'scatter' [bivarient analayis]

In [27]:
```
j = sns.jointplot(data = movie , x='CriticRating',y='AudienceRating',kind = 'scatter' )
```



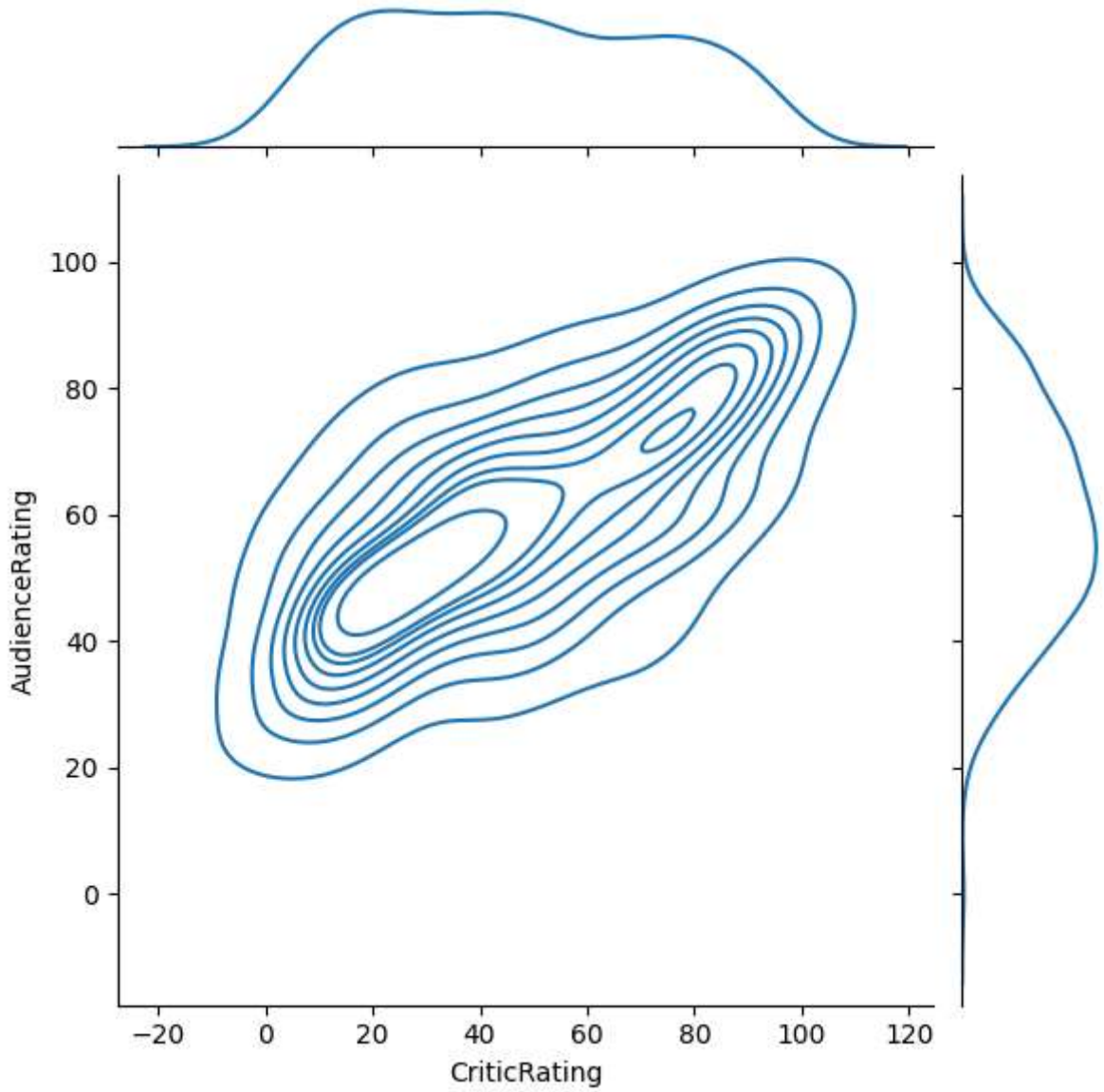
ploting a jointplot where x='CriticRating',y='AudienceRating',kind = 'reg' [bivarient analysis]

In [28]:
```
j = sns.jointplot(data = movie , x='CriticRating',y='AudienceRating',kind = 'reg' )
```



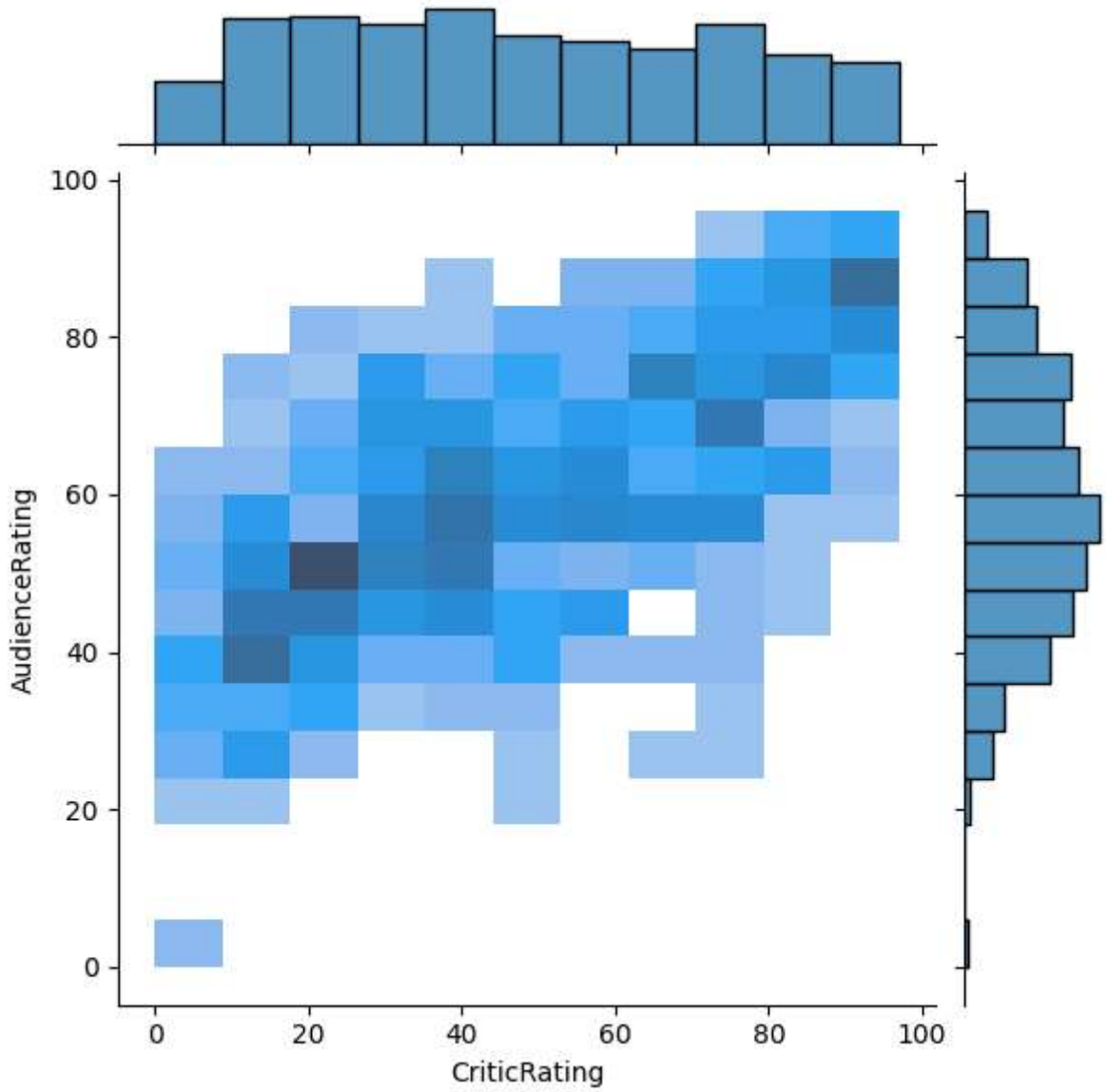
ploting jointplot where x='CriticRating',y='AudienceRating',kind = 'kde'

In [29]:
```
j = sns.jointplot(data = movie , x='CriticRating',y='AudienceRating',kind = 'kde' ) # kernel density
```



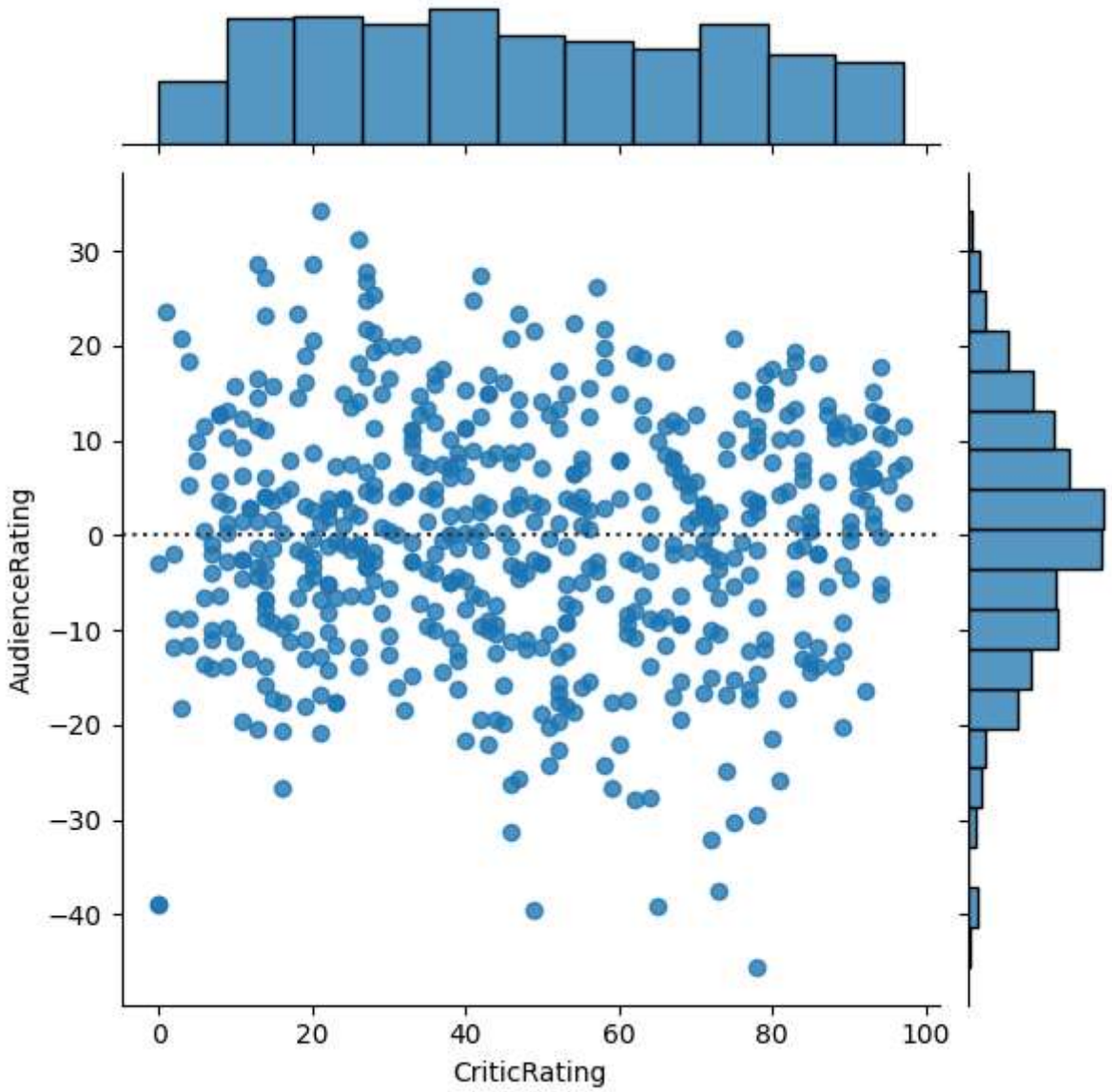
ploting jointplot where x='CriticRating',y='AudienceRating',kind = 'hist'

In [30]:
```
j = sns.jointplot(data = movie , x='CriticRating',y='AudienceRating',kind = 'hist' )
```



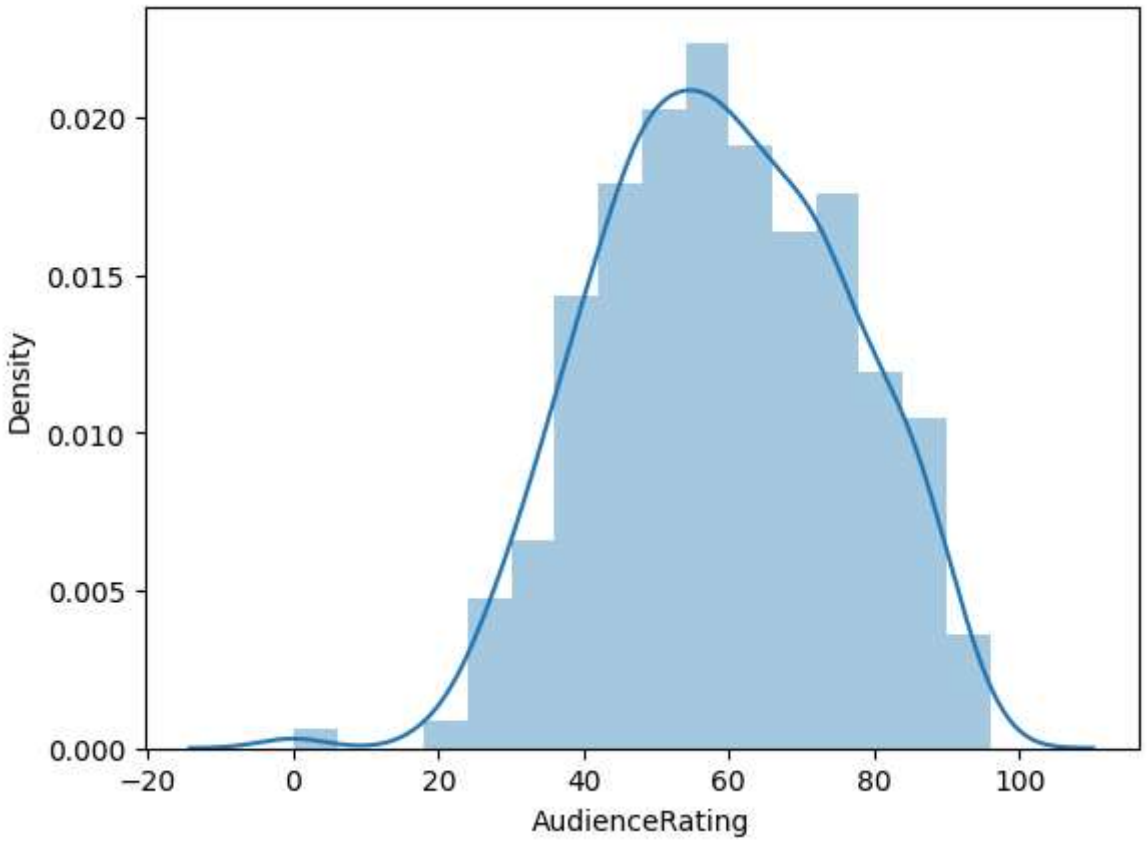
ploting jointplot where x='CriticRating',y='AudienceRating',kind = 'resid'

In [31]:
```
j = sns.jointplot(data = movie , x='CriticRating',y='AudienceRating',kind = 'resid' )
```



distplot() --> used for visualizing the univariate distribution of a dataset.

In [32]:
```
m1 = sns.distplot(movie.AudienceRating)
```

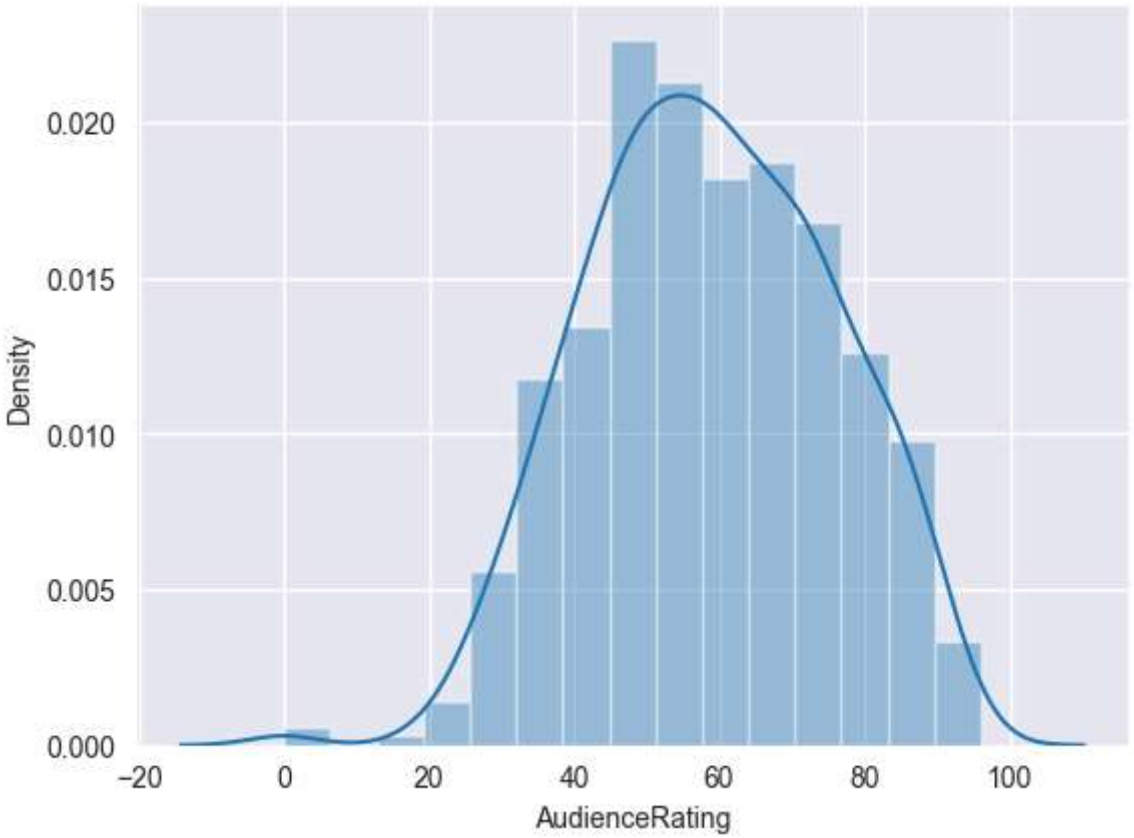


set_style() -->set the background and gridlines.

In [33]:
```
sns.set_style('darkgrid')
```

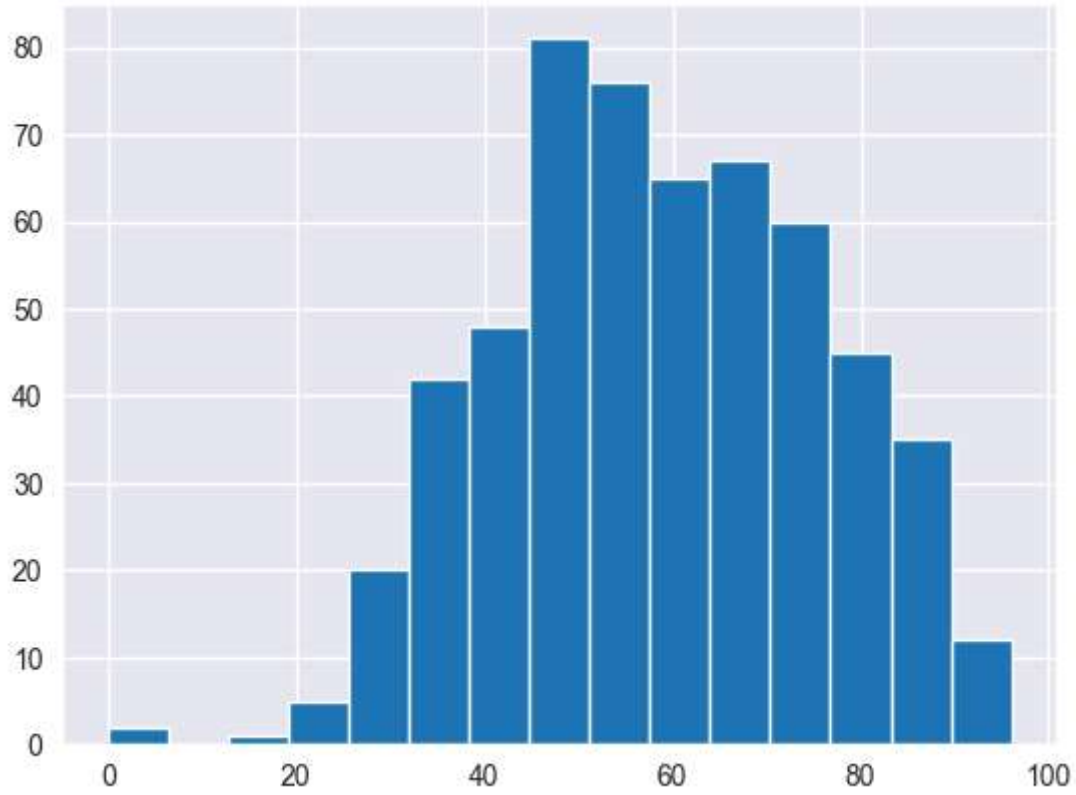
ploting the distplot with bins = 15

In [34]:
```
m2 = sns.distplot(movie.AudienceRating , bins = 15)
```

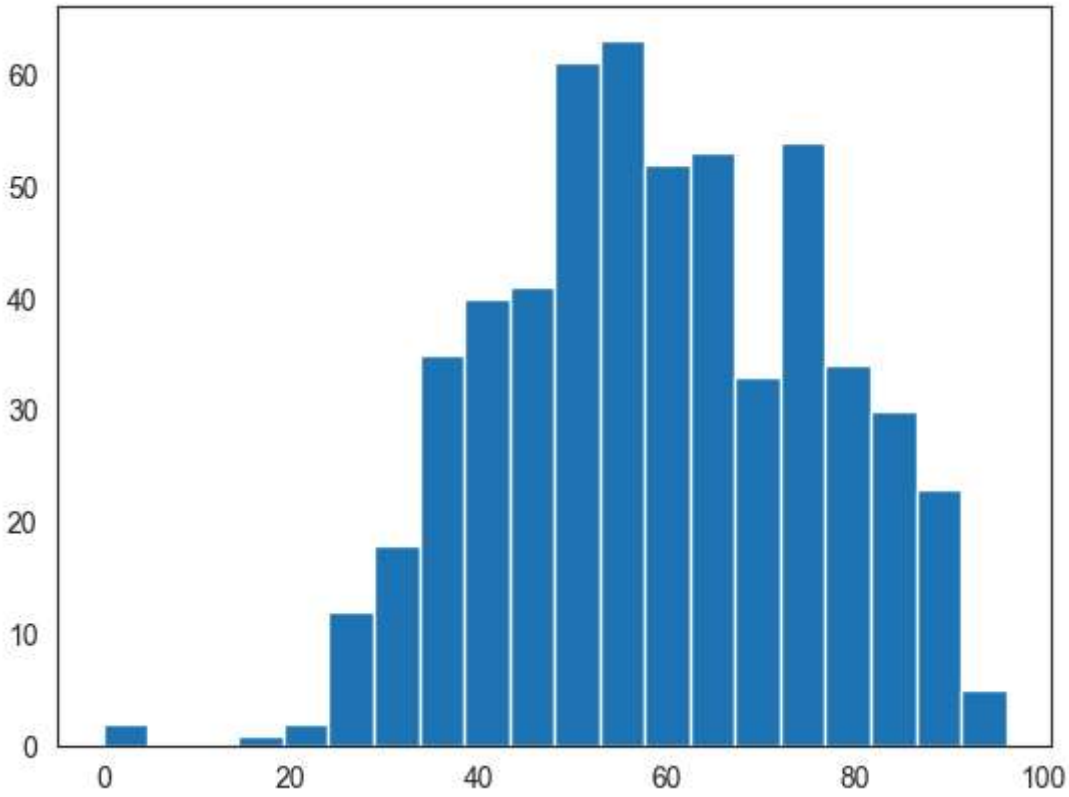


ploting the hist plot where bins = 15

In [35]:
```
n1 = plt.hist(movie.AudienceRating , bins = 15)
```

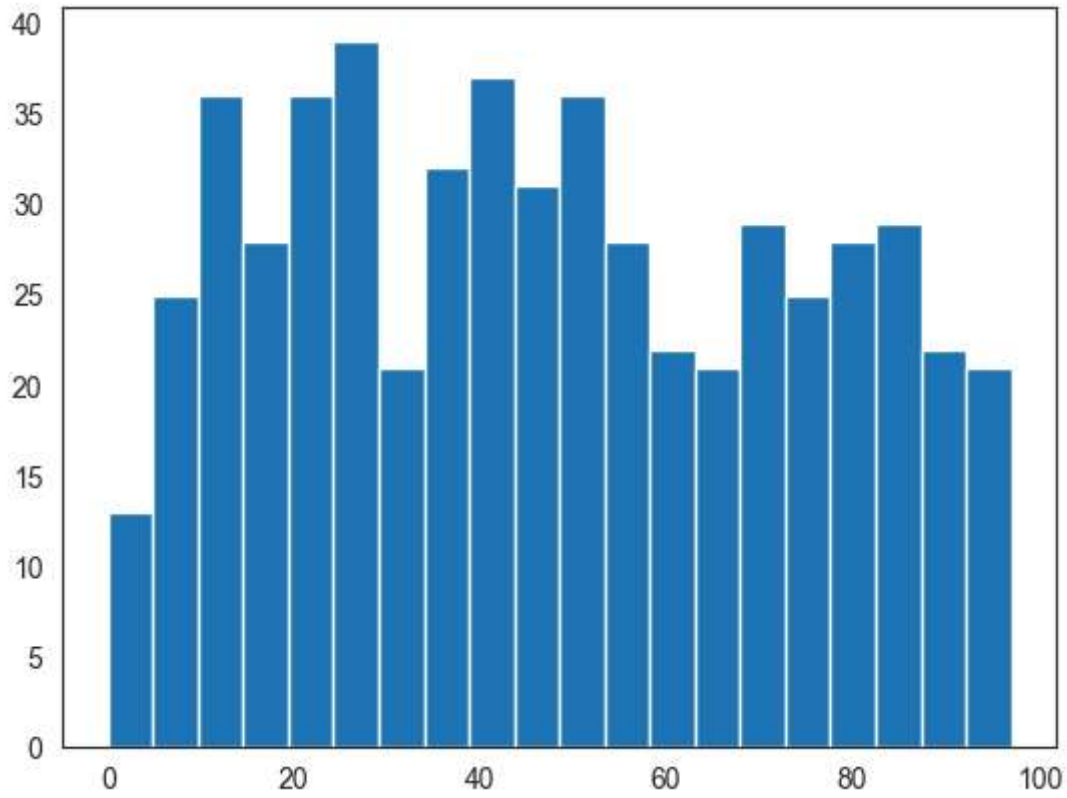


In [36]:
```
sns.set_style('white')  #normal distribution & called as bell curve
n1 = plt.hist(movie.AudienceRating , bins = 20)
```



ploting the hist plot with CriticRating , bins = 20

In [37]:
```
n1 = plt.hist(movie.CriticRating , bins = 20)
```

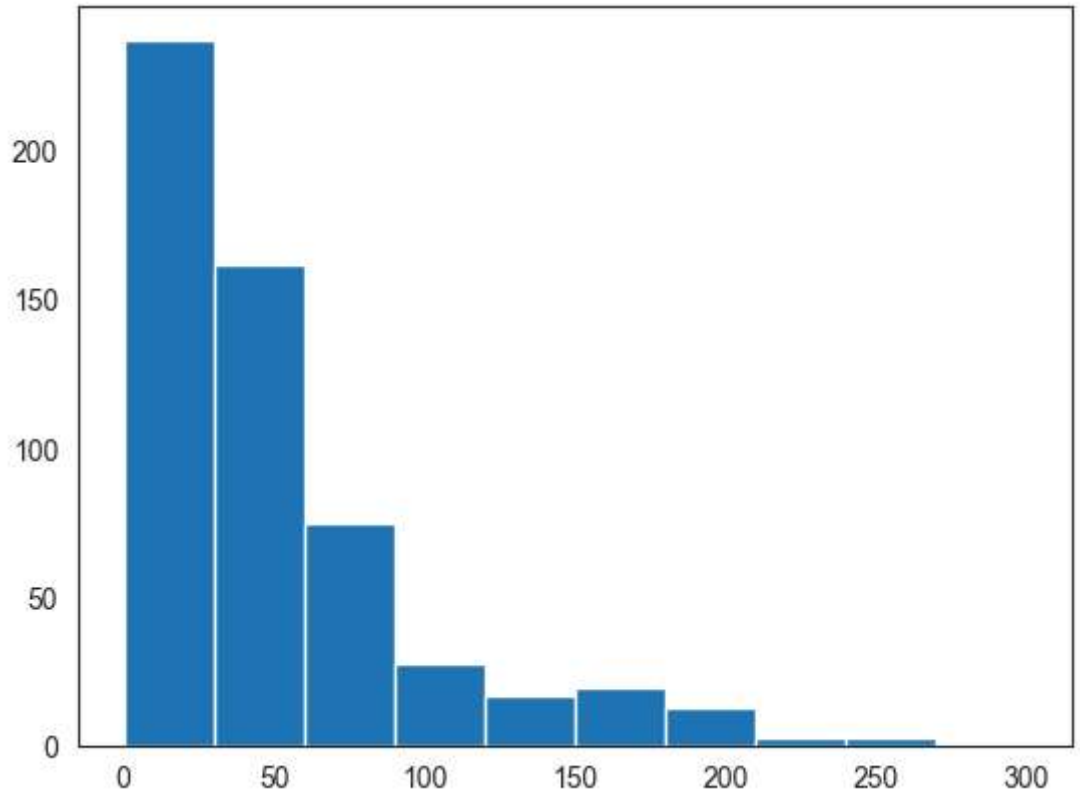


In [38]:
```
movie.columns
```

Out[38]:
```
Index(['Film', 'Genre', 'CriticRating', 'AudienceRating', 'BudgetMillion',
       'Year'],
      dtype='object')
```

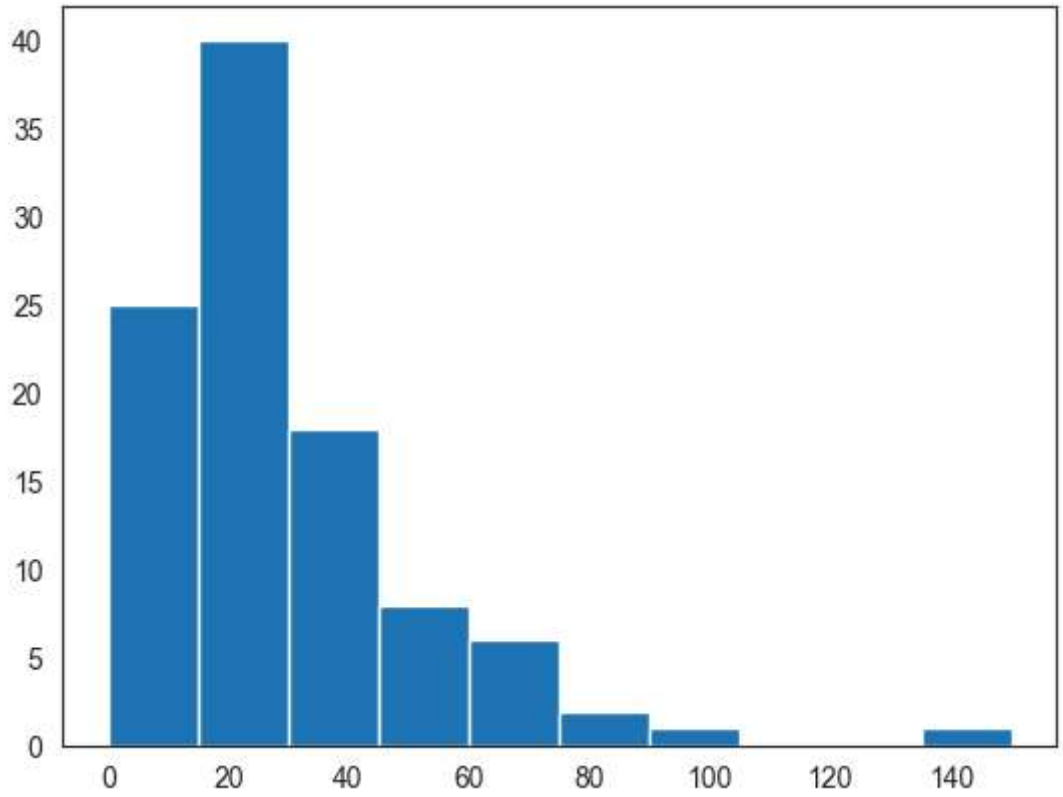
ploting the hist plot with BudgetMillion

In [39]:
```
plt.hist(movie.BudgetMillion)
plt.show()
```



ploting the hist plot where Genre == 'Drama' [how many Drama movies fall into different budget ranges]

In [40]:
```
plt.hist(movie[movie.Genre == 'Drama'].BudgetMillion)
plt.show()
```



Genre.unique() --> checks how many unique records in the Genre column

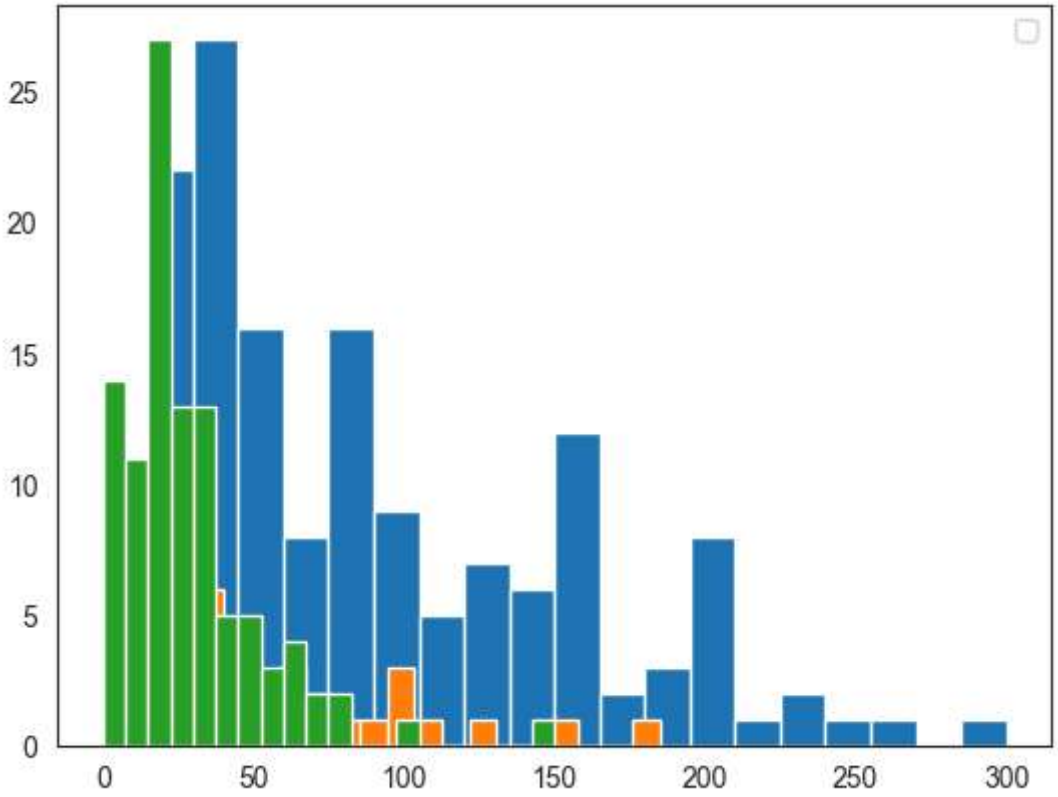
In [41]:
```
movie.Genre.unique()
```

Out[41]:
```
['Comedy', 'Adventure', 'Action', 'Horror', 'Drama', 'Romance', 'Thriller']
Categories (7, object): ['Action', 'Adventure', 'Comedy', 'Drama', 'Horror', 'Romance', 'Thriller']
```

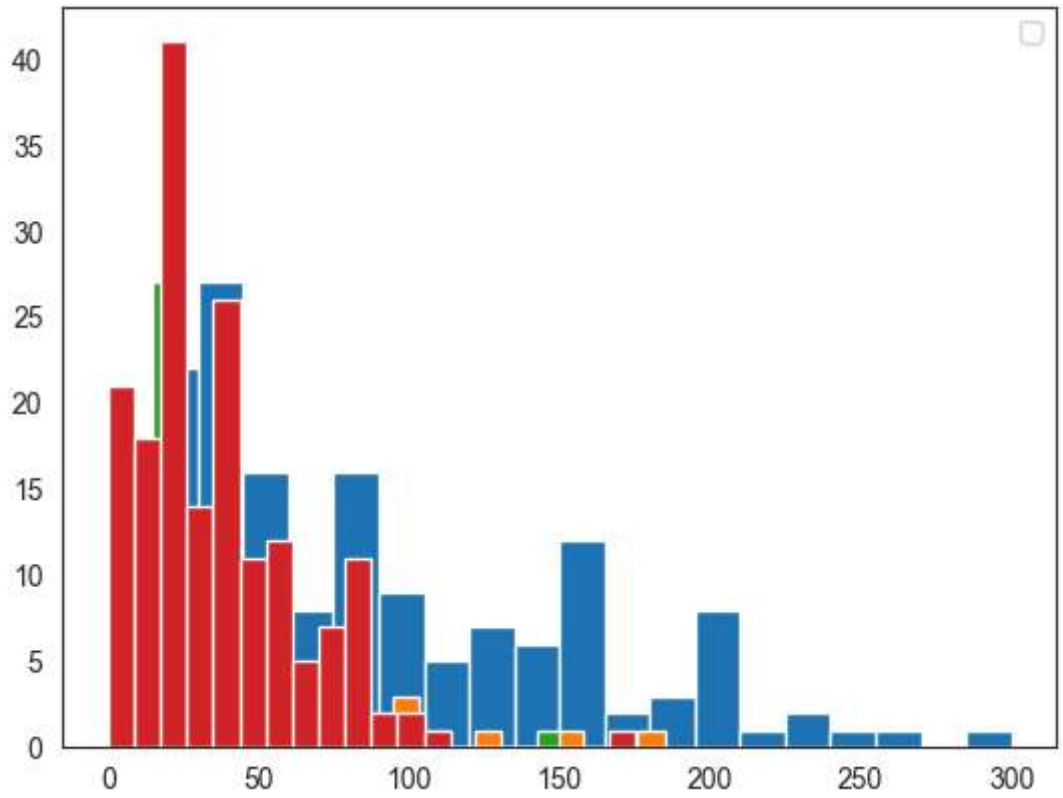
ploting different Genre == 'Action' ,'Thriller' ,'Drama' with budget ranges

In [42]:
```
plt.hist(movie[movie.Genre == 'Action'].BudgetMillion, bins = 20)
plt.hist(movie[movie.Genre == 'Thriller'].BudgetMillion, bins = 20)
plt.hist(movie[movie.Genre == 'Drama'].BudgetMillion, bins = 20)
```

```
plt.legend()
plt.show()
```



In [43]:
```
plt.hist(movie[movie.Genre == 'Action'].BudgetMillion, bins = 20)
plt.hist(movie[movie.Genre == 'Thriller'].BudgetMillion, bins = 20)
plt.hist(movie[movie.Genre == 'Drama'].BudgetMillion, bins = 20)
plt.hist(movie[movie.Genre == 'Comedy'].BudgetMillion, bins = 20)
plt.legend()
plt.show()
```

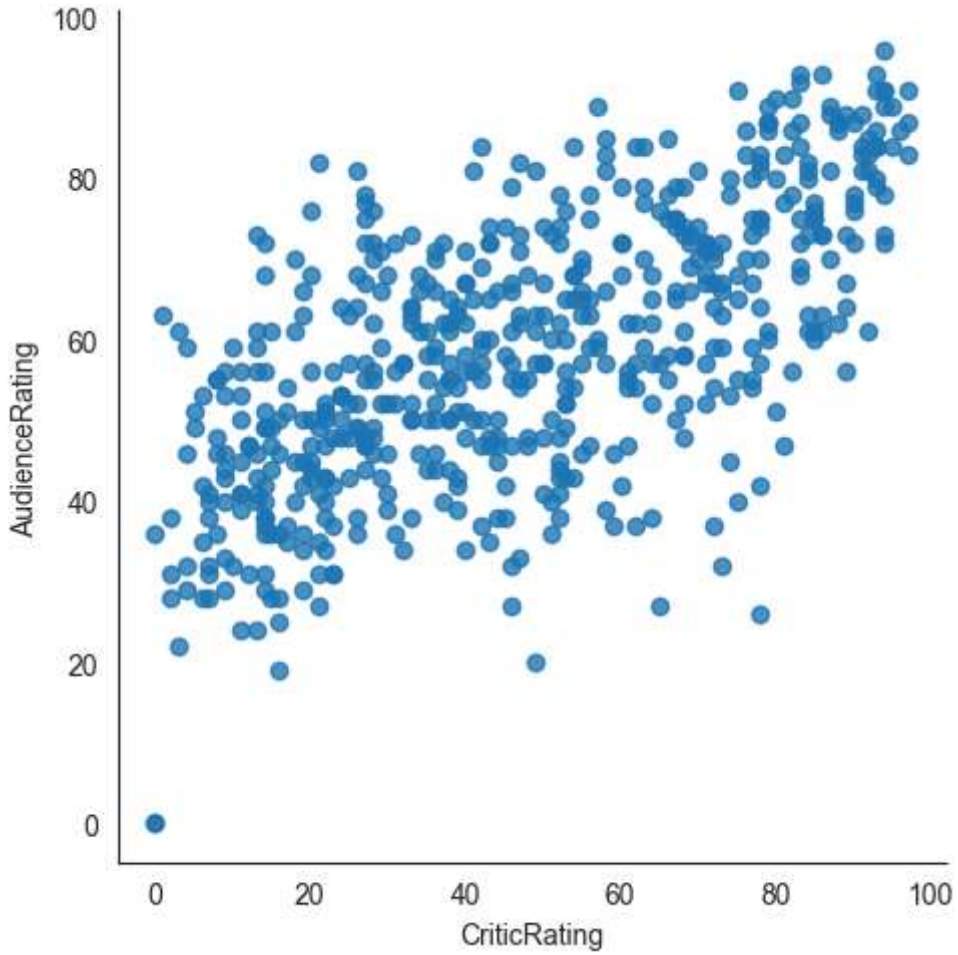


In [44]:
```
for gen in movie.Genre.cat.categories:
    print(gen)
```

```
Action
Adventure
Comedy
Drama
Horror
Romance
Thriller
```

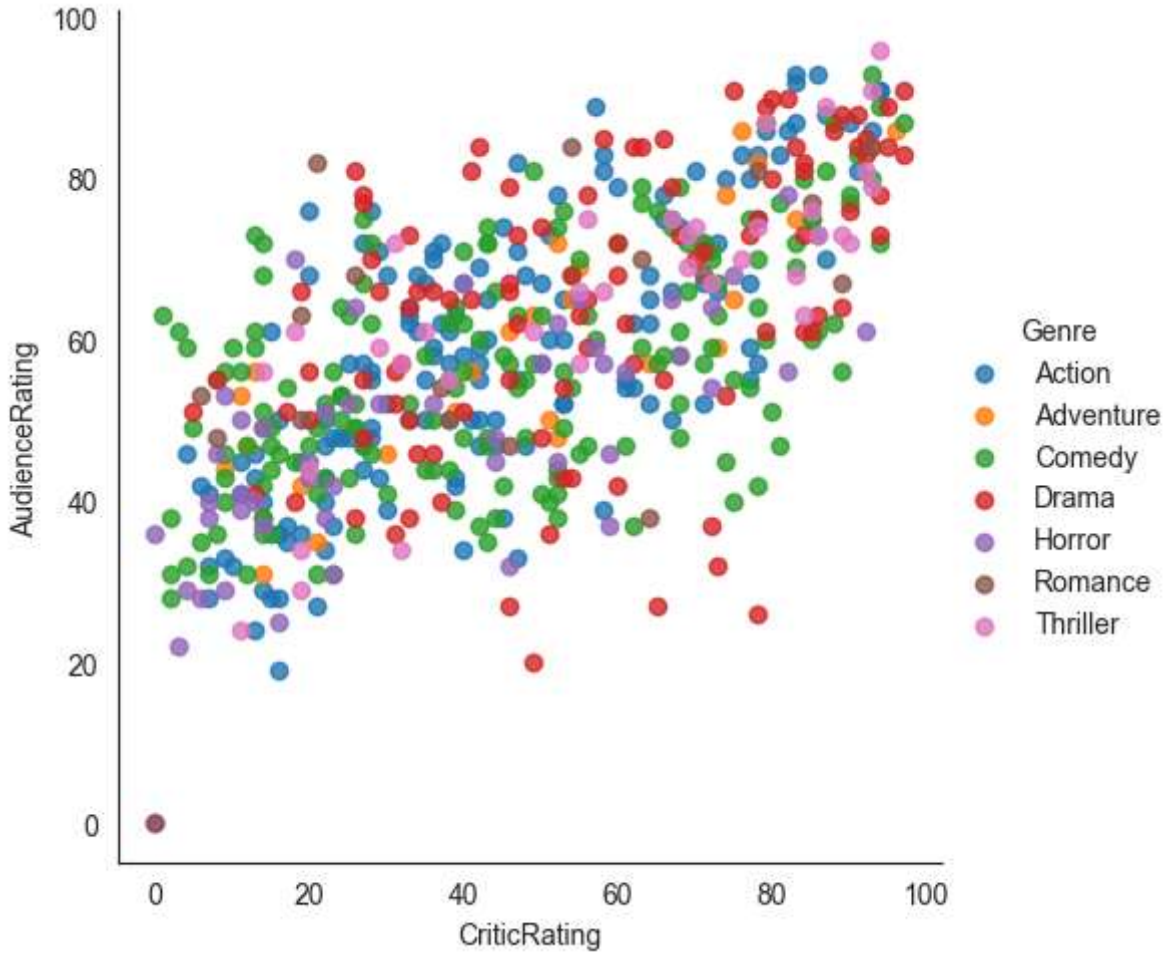
lmplot() --> used to plot linear model graphs [linear regression graphs]

In [45]:
```
vis1 = sns.lmplot(data = movie , x= 'CriticRating' , y = 'AudienceRating',fit_reg=False)
```



hue --> functions to add color coding based on a categorical variable.

In [46]:
```
vis2 = sns.lmplot(data = movie , x= 'CriticRating' , y = 'AudienceRating',fit_reg=False , hue = 'Genre')
```



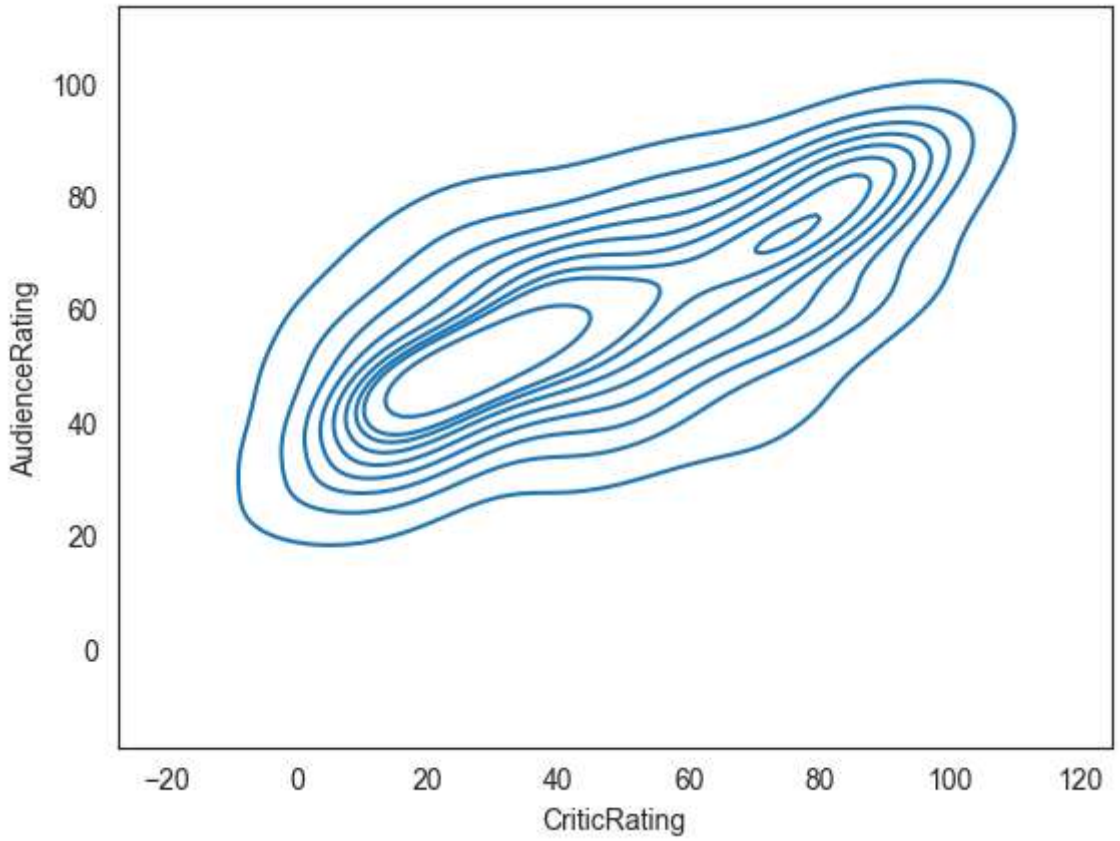
In [47]:
```
sns.set_style=('Dark')
```

In [48]:
```
vis3 = sns.lmplot(data = movie , x= 'CriticRating' , y = 'AudienceRating',fit_reg=False , hue = 'Genre' , height = 1
```

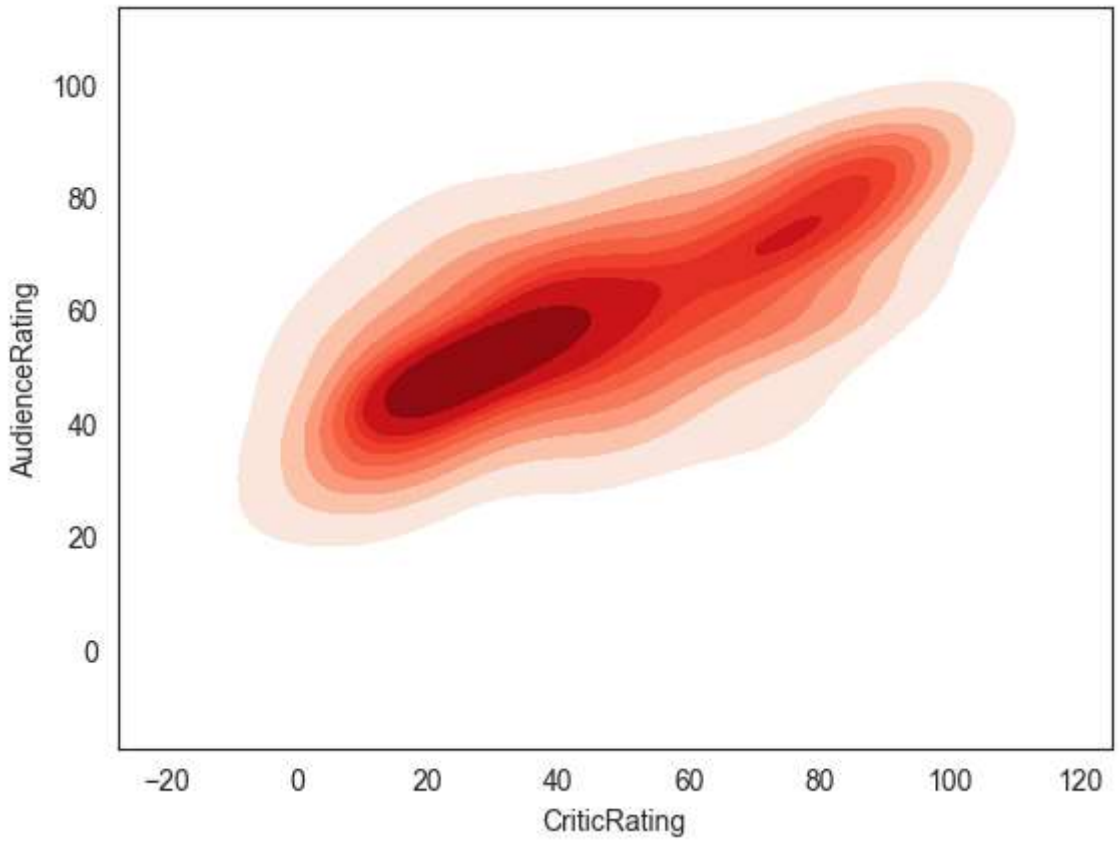

100
80
60
40
20
0
AudienceRating
Genre
Action
Adventure
Comedy
Drama
Horror
Romance
Thriller

Kernal Density Estimate plot (KDE PLOT)

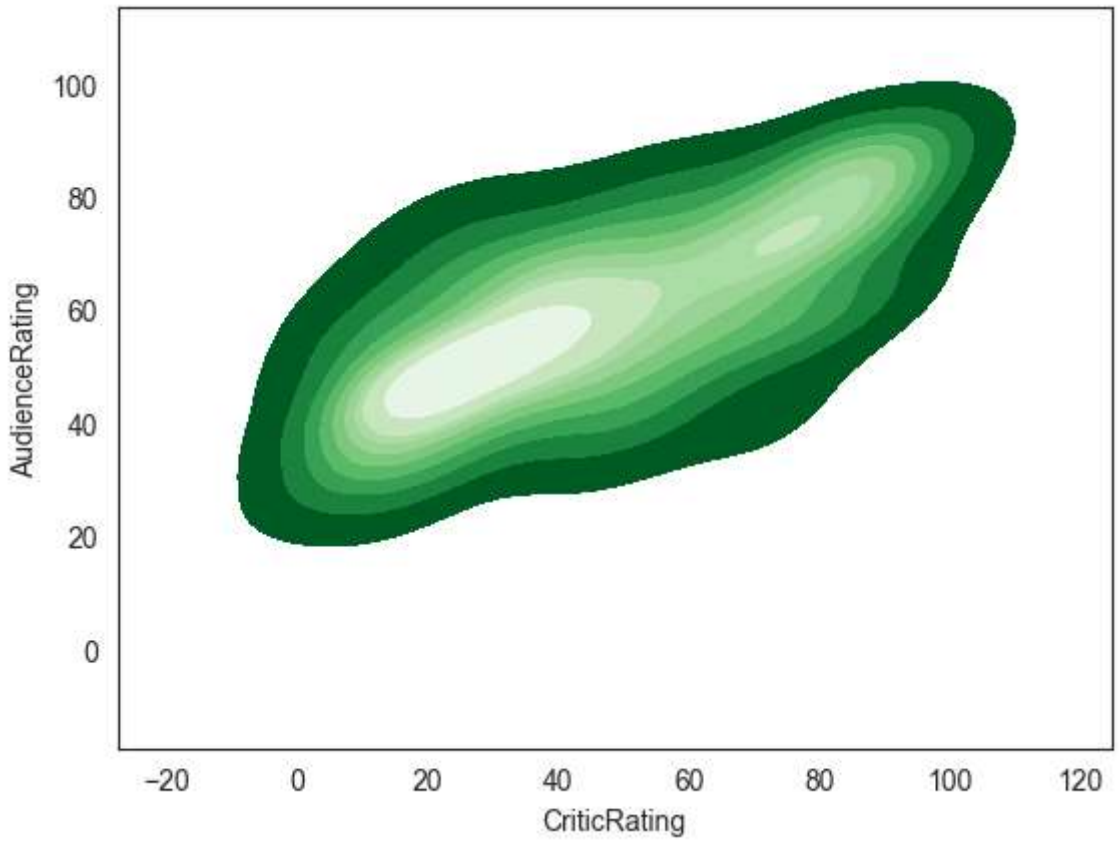
In [49]:
```
k1 = sns.kdeplot(data=movie, x = 'CriticRating',y = 'AudienceRating')
```



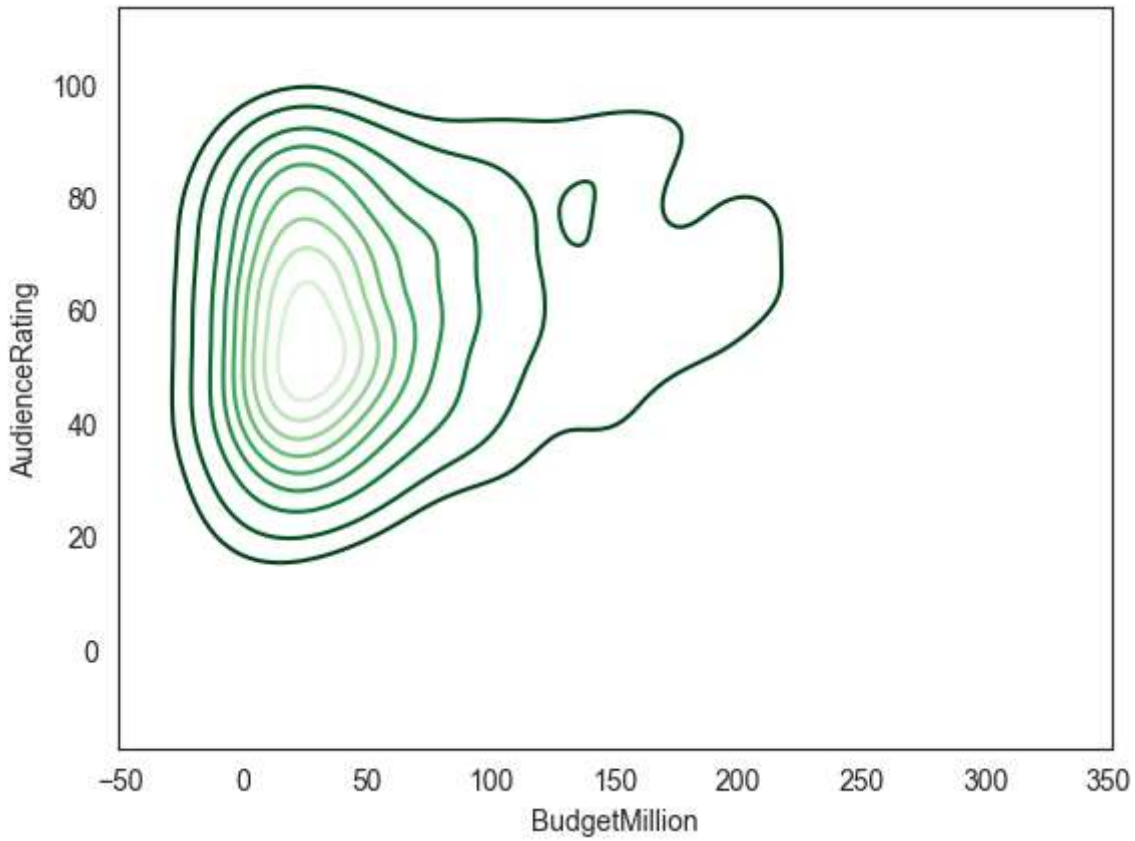
In [50]:
```
k2 = sns.kdeplot(data=movie, x = 'CriticRating',y = 'AudienceRating', shade = True , shade_lowest = False , cmap= 'R
```



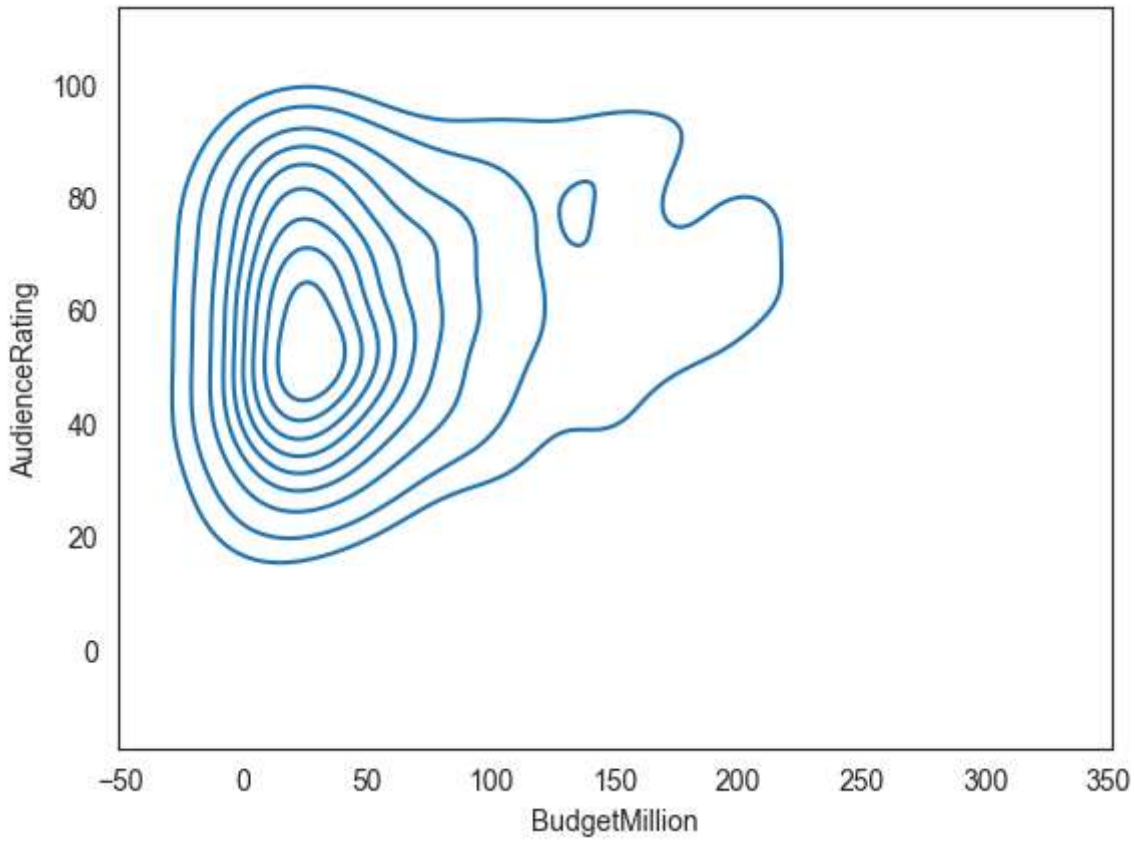
In [51]:
```
k3 = sns.kdeplot(data=movie, x = 'CriticRating',y = 'AudienceRating', shade = True , shade_lowest = False , cmap= 'Gr
```



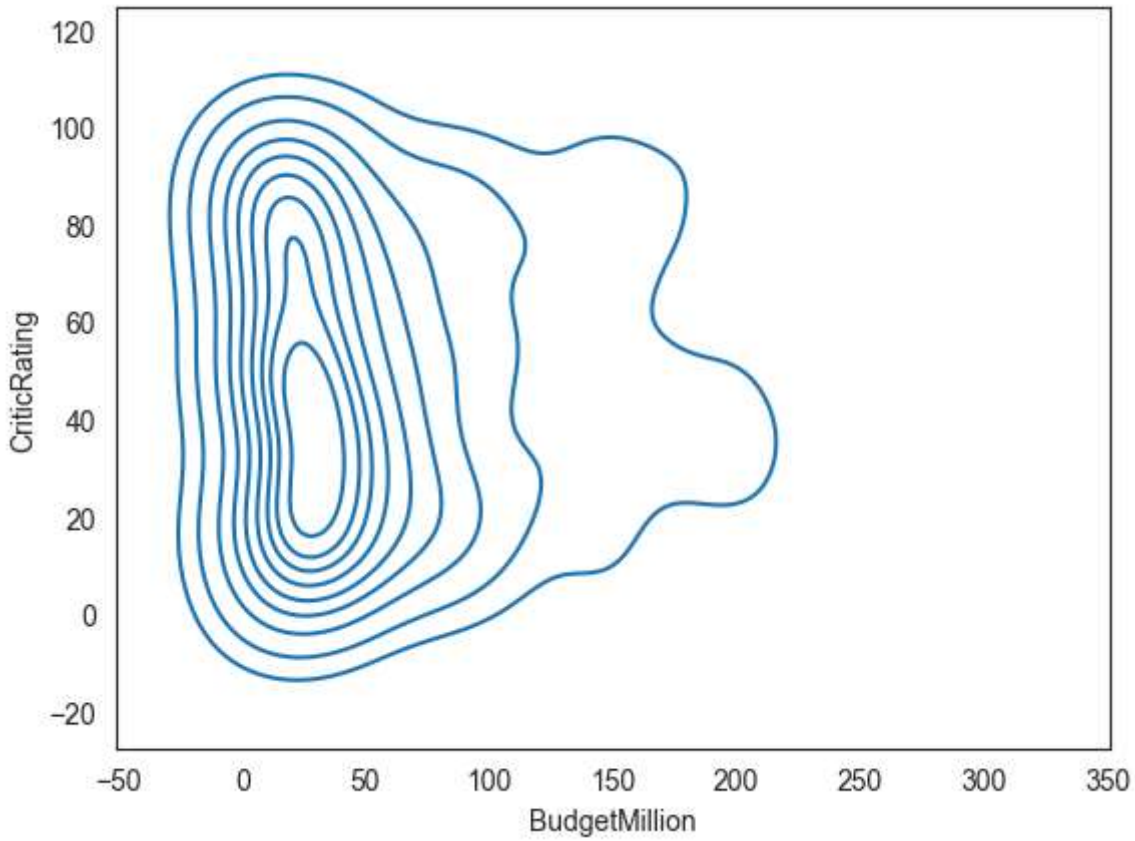
In [52]:
```
k4 = sns.kdeplot(data = movie , x = 'BudgetMillion' , y = 'AudienceRating' , shade_lowest = False , cmap = 'Greens_r
```



In [53]:
```
k5 = sns.kdeplot(data = movie , x = 'BudgetMillion' , y = 'AudienceRating')
```



In [54]:
```
k6 = sns.kdeplot(data = movie , x = 'BudgetMillion' , y = 'CriticRating')
```



In [55]:
```
with open("multiple_models.pkl", "wb") as file:
    pickle.dump(models, file)
```

In [ ]: