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

df = pd.read_csv(r"D:\DATA SCIENCE\PRAKASH\EveryDayClassRoom\March\28th\28th -Seaborn movie analy
df.head()
```

Out[2]:

	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

In [3]:
```
movies_df = df.copy()
movies_df.shape
```

Out[3]:
```
(559, 6)
```

In [4]:
```
movies_df.columns
```

Out[4]:
```
Index(['Film', 'Genre', 'Rotten Tomatoes Ratings %', 'Audience Ratings %',
       'Budget (million $)', 'Year of release'],
      dtype='object')
```

In [5]:
```
movies_df.columns = ['Film', 'Genre', 'CriticRatings', 'AudienceRatings',
       'BudgetMillion', 'Year']
movies_df
```

Out[5]:

	Film	Genre	CriticRatings	AudienceRatings	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
...	...	...	...	...	...	...
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

In [6]:
```
movies_df.describe()
```

Out[6]:

	CriticRatings	AudienceRatings	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

In [7]:
```
movies_df.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   CriticRatings    559 non-null    int64 
 3   AudienceRatings  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
```

In [11]:
```
type(movies_df['Year'])
```

Out[11]:
```
pandas.core.series.Series
```

In [12]:
```
type(movies_df.Year)
```

Out[12]:
```
pandas.core.series.Series
```

In [14]:
```
movies_df['Year'] = movies_df.Year.astype("category")
movies_df.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   CriticRatings    559 non-null    int64   
 3   AudienceRatings  559 non-null    int64   
 4   BudgetMillion    559 non-null    int64   
 5   Year             559 non-null    category
dtypes: category(1), int64(3), object(2)
memory usage: 22.7+ KB
```

In [15]:
```
movies_df.Genre = movies_df.Genre.astype('category')
movies_df.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    category
 2   CriticRatings    559 non-null    int64   
 3   AudienceRatings  559 non-null    int64   
 4   BudgetMillion    559 non-null    int64   
 5   Year             559 non-null    category
dtypes: category(2), int64(3), object(1)
memory usage: 19.2+ KB
```

In [17]:
```
movies_df.Genre.cat.categories
```

Out[17]:
```
Index(['Action', 'Adventure', 'Comedy', 'Drama', 'Horror', 'Romance',
       'Thriller'],
      dtype='object')
```

In [18]:
```
movies_df.Genre.unique()
```

Out[18]:
```
['Comedy', 'Adventure', 'Action', 'Horror', 'Drama', 'Romance', 'Thriller']
Categories (7, object): ['Action', 'Adventure', 'Comedy', 'Drama', 'Horror', 'Romance', 'Thrille
r']
```

In [20]:
```
movies_df.Genre.dtypes
```

Out[20]:
```
CategoricalDtype(categories=['Action', 'Adventure', 'Comedy', 'Drama', 'Horror',
                  'Romance', 'Thriller'],
, ordered=False, categories_dtype=object)
```

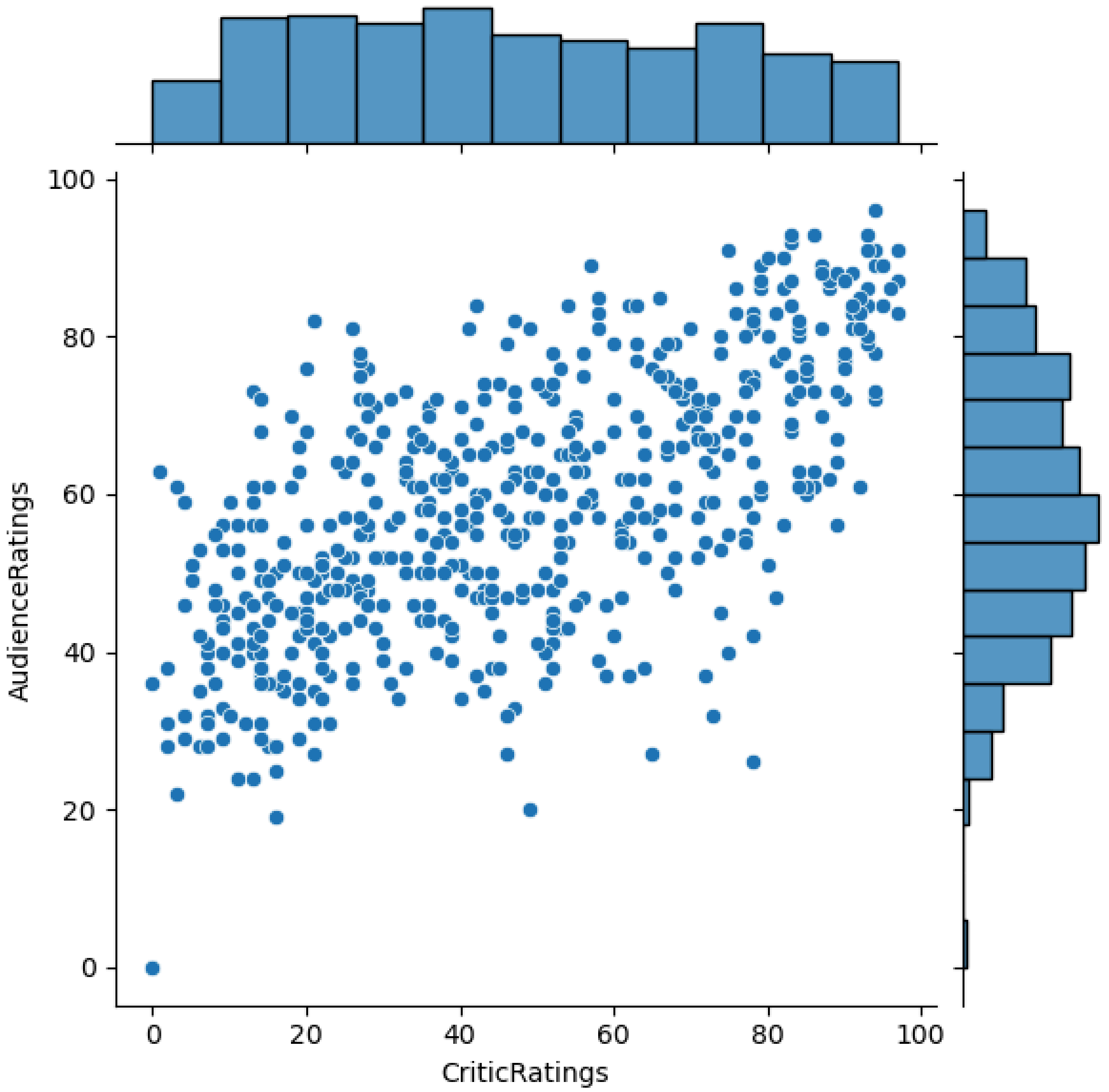
In [21]:
```
movies_df.describe()
```

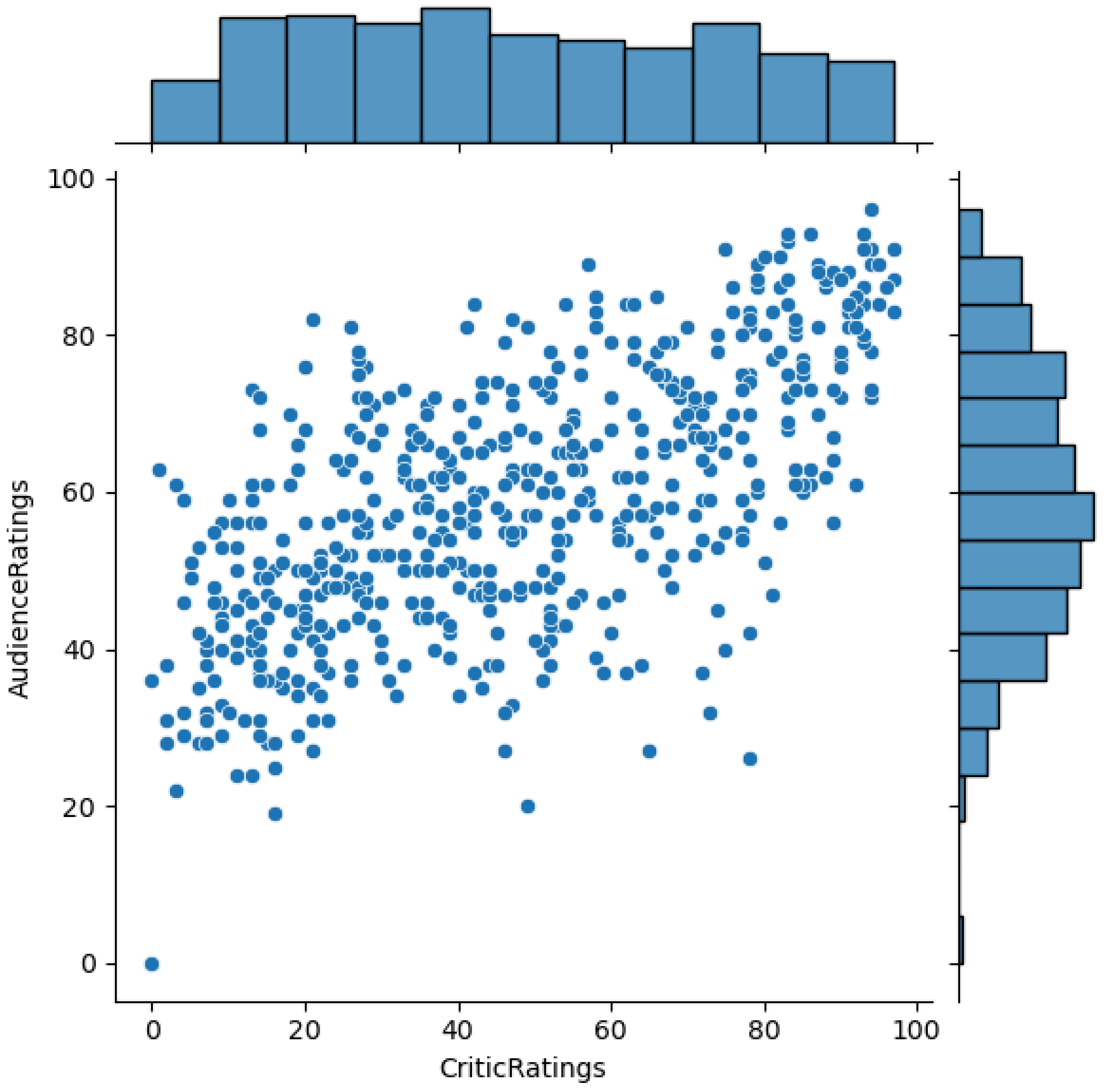
Out[21]:

	CriticRatings	AudienceRatings	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

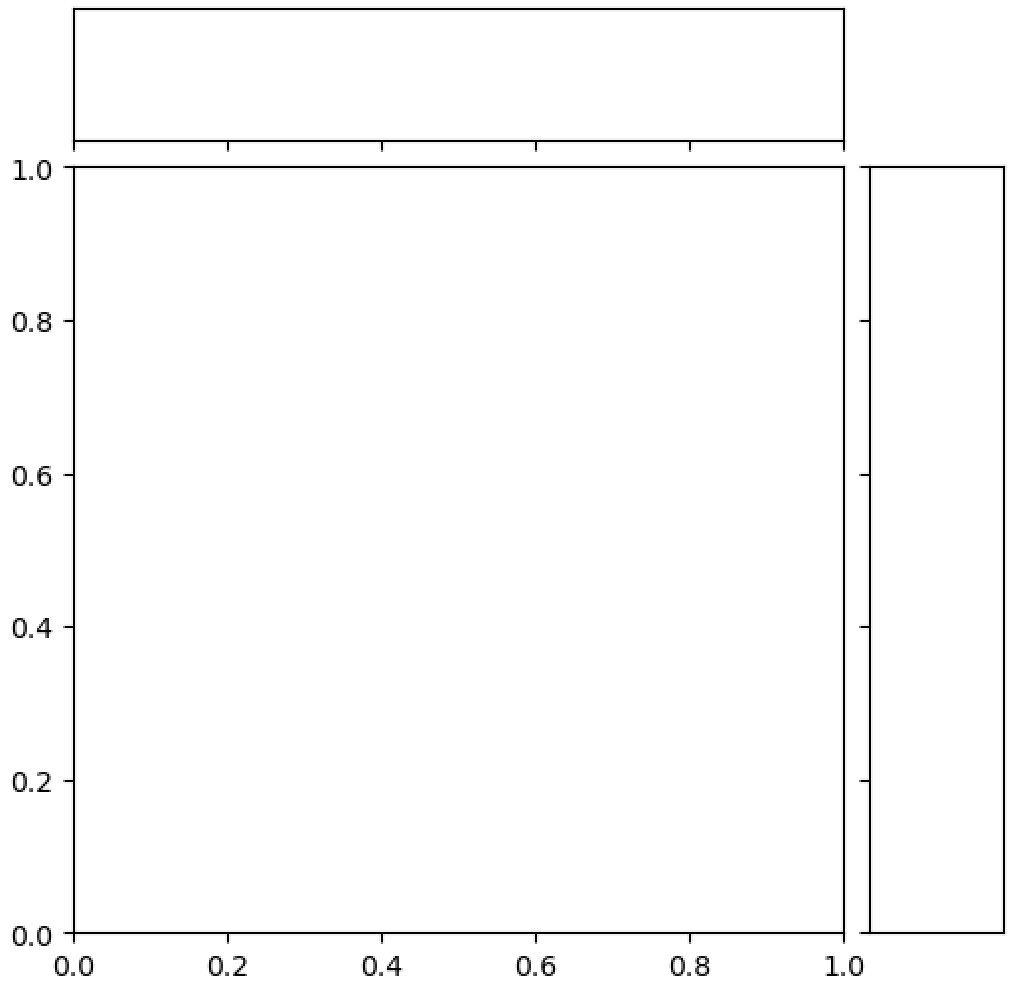
In [22]:
```
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

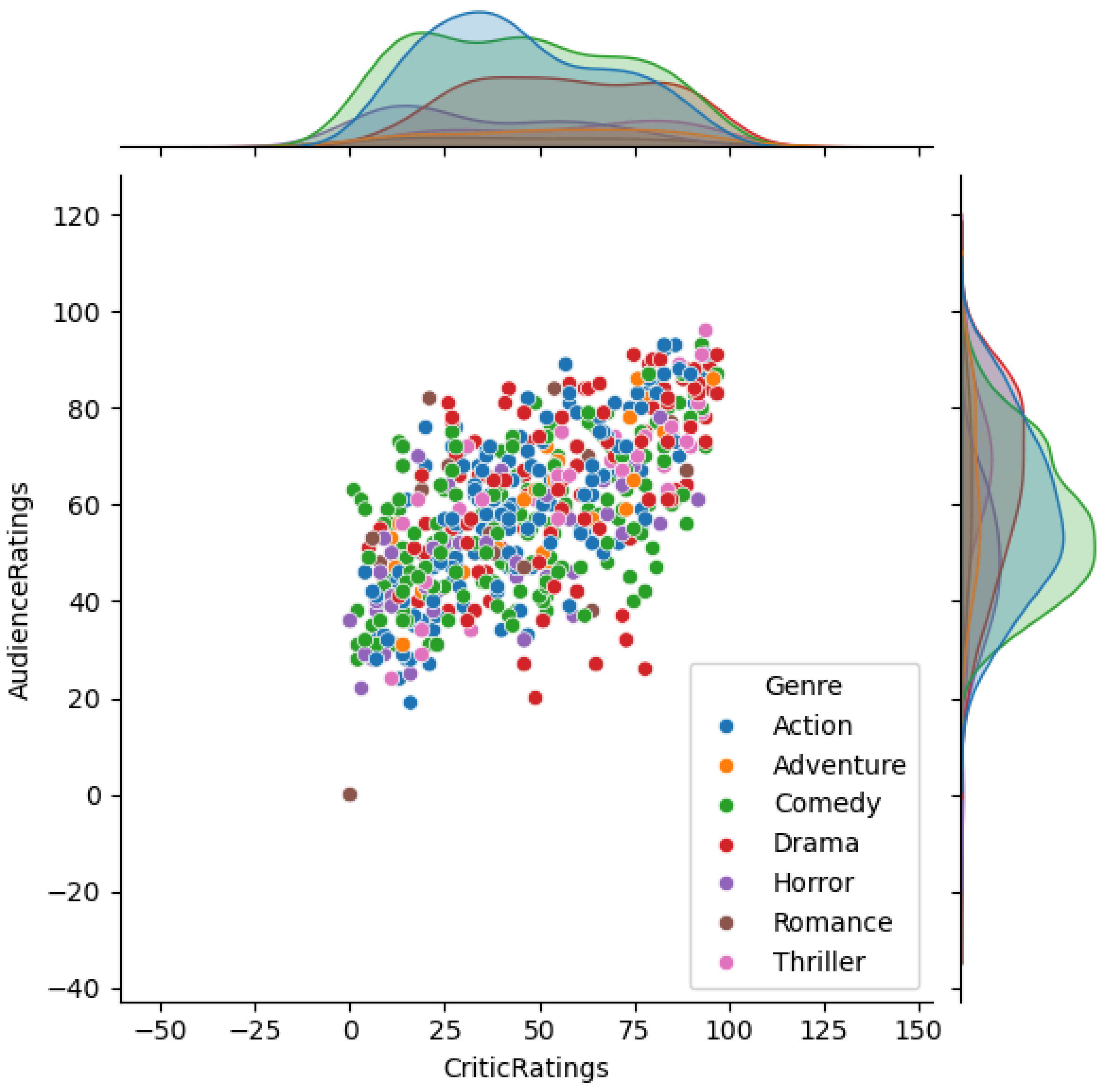
In [24]:
```
sns.jointplot(data=movies_df, x='CriticRatings', y = 'AudienceRatings')
plt.show()
```


AudienceRatings
CriticRatings
0
20
40
60
80
100

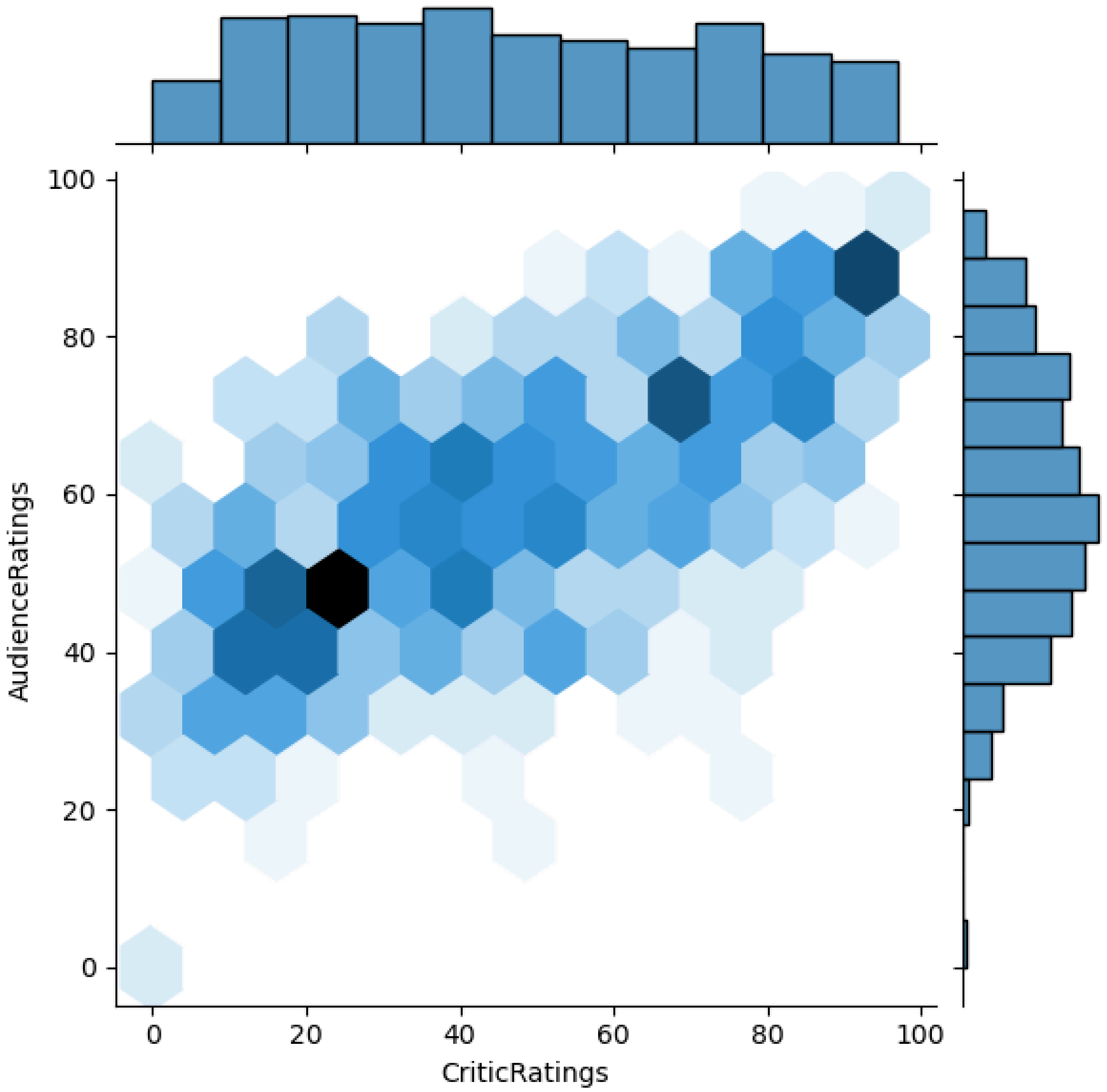


```
In [26]: sns.jointplot(data=movies_df, x='CriticRatings', y = 'AudienceRatings', hue='Genre')
         plt.show()
```

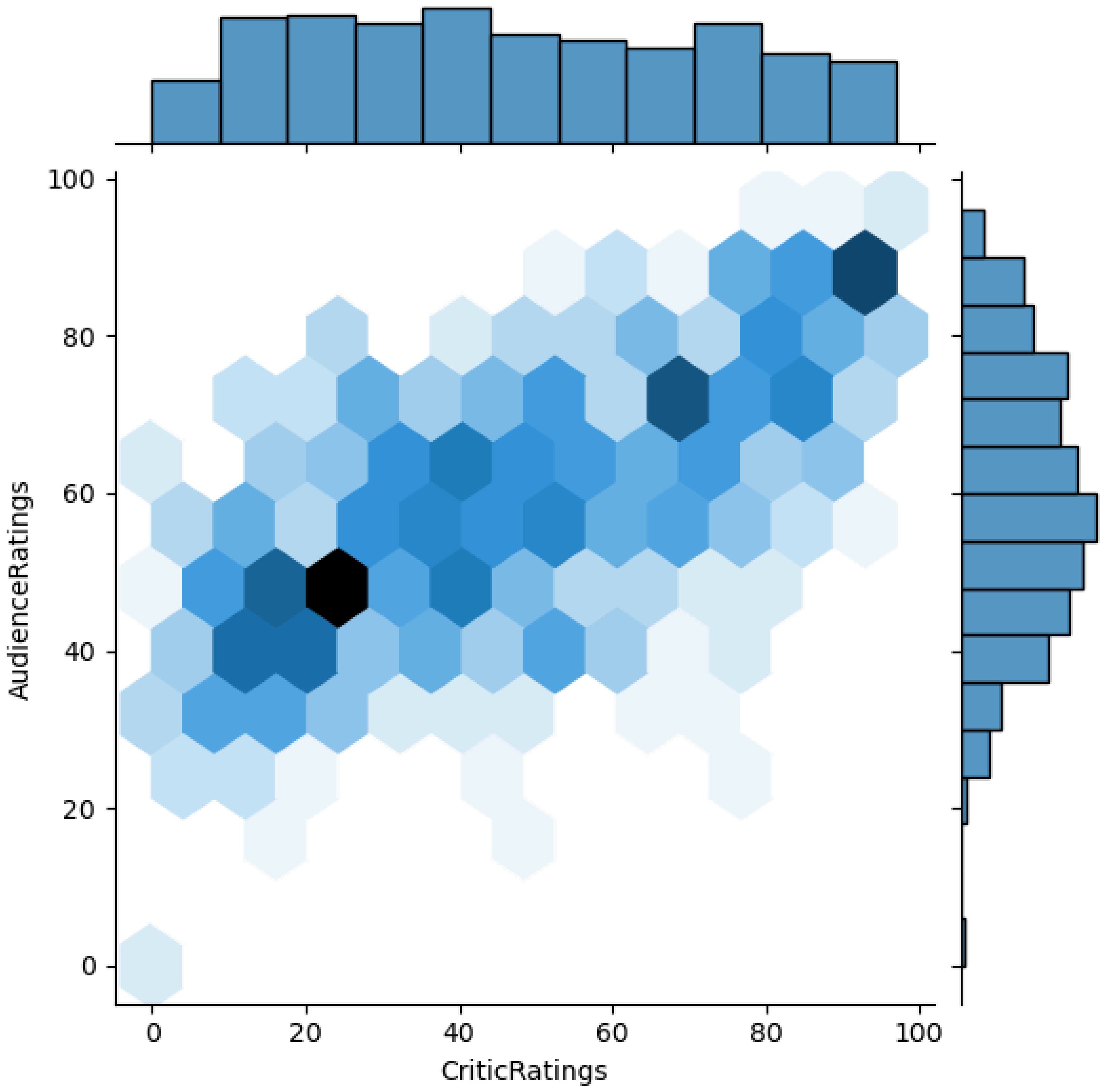

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



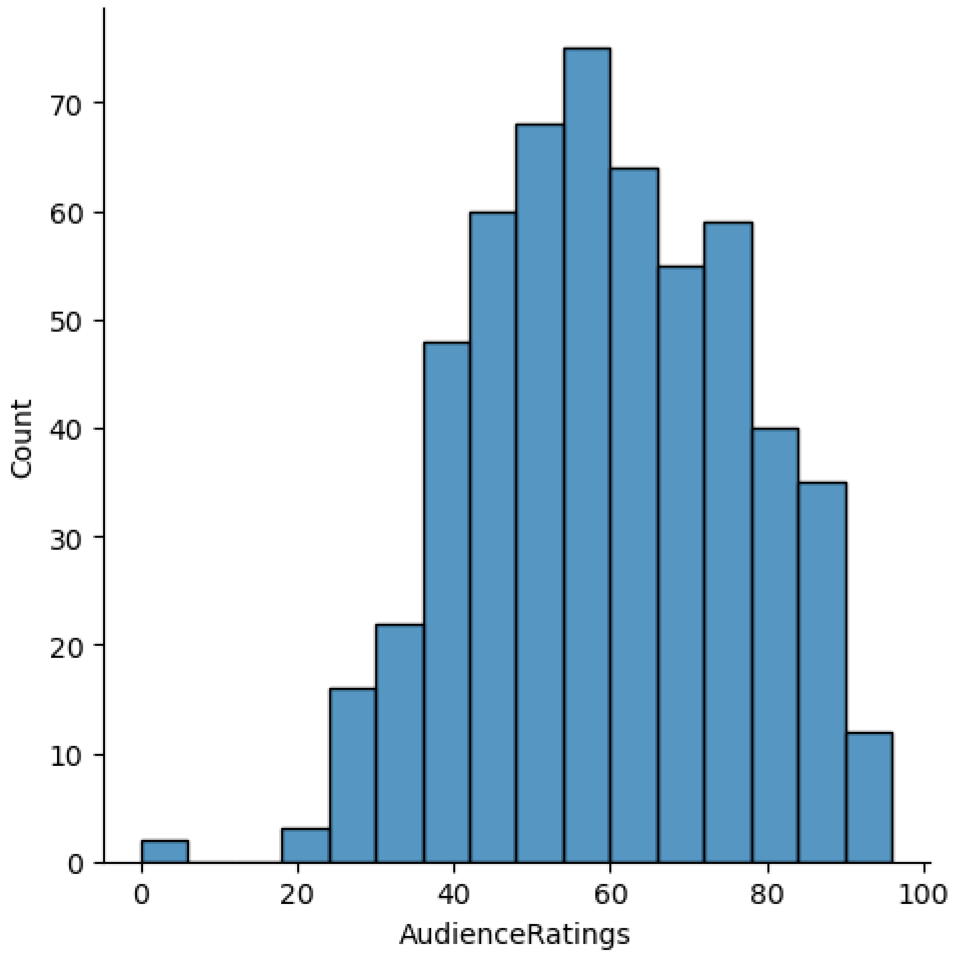
```
In [28]: sns.jointplot(data=movies_df,x='CriticRatings', y = 'AudienceRatings', kind='hex')
         plt.show()
```



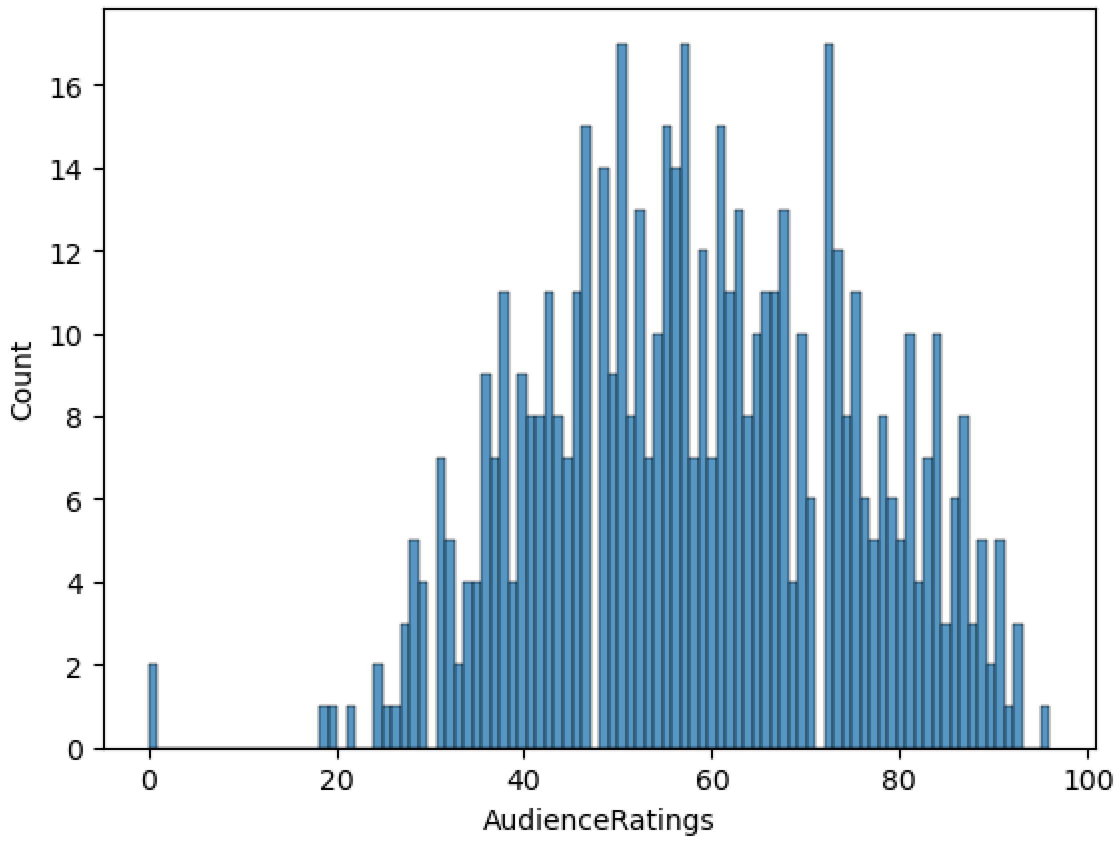
100
80
60
40
20
0
AudienceRatings
0
20
40
60
80
100
CriticRatings



```
In [29]: sns.displot(movies_df.AudienceRatings)
         plt.show()
```

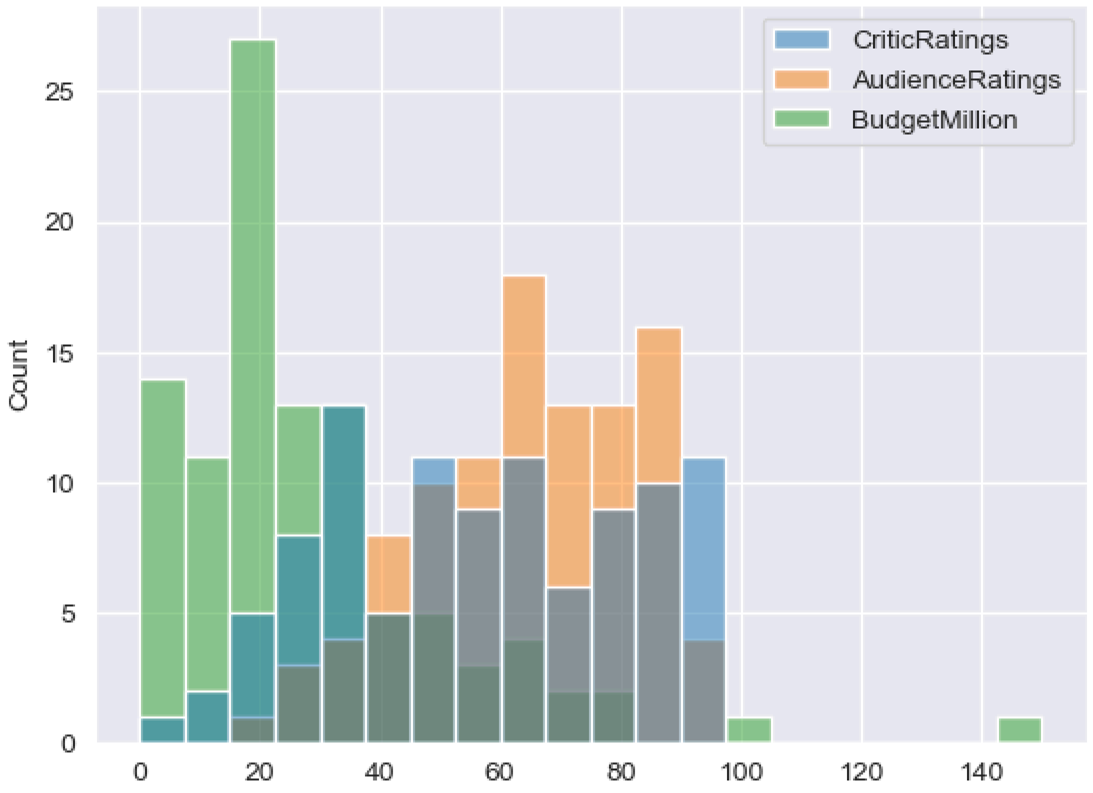


```
In [32]: sns.histplot(movies_df.AudienceRatings,bins= 100)
         plt.show()
```



```
In [42]: sns.set_style('darkgrid')
         sns.histplot(movies_df[movies_df.Genre == 'Drama'], bins = 20)
```

```
plt.show()
```



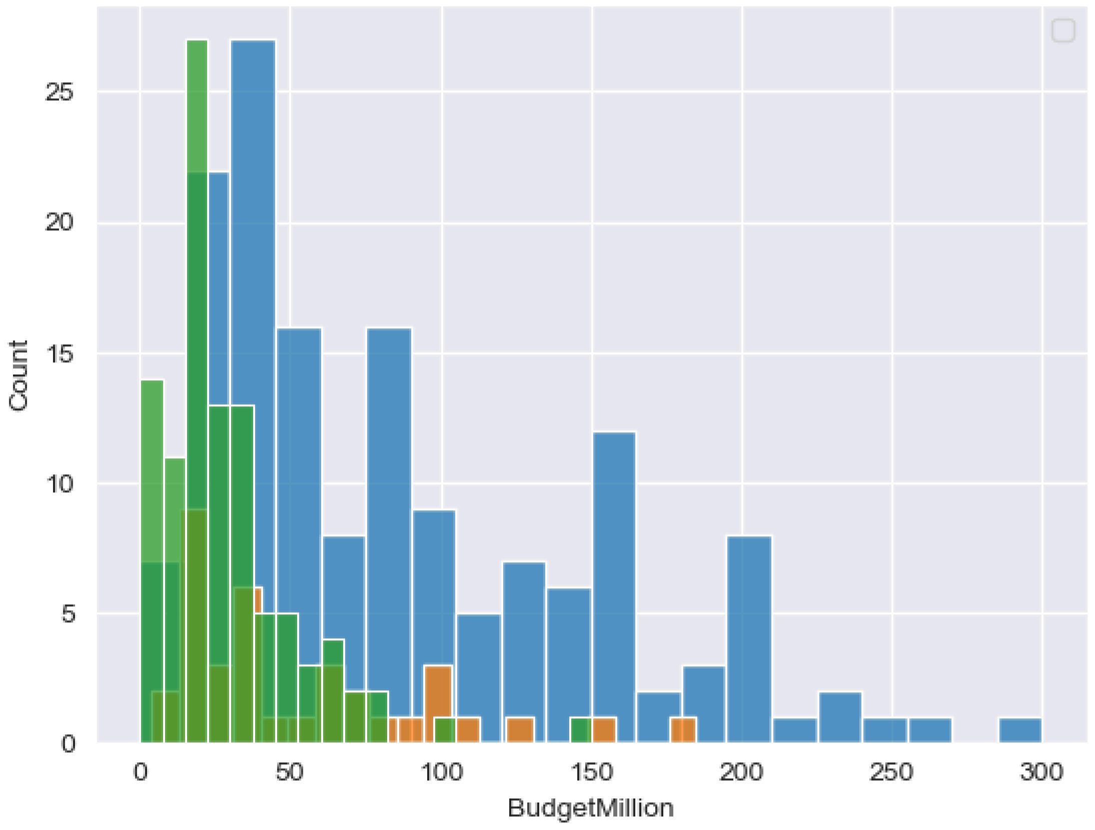
In [36]:
```
movies_df[movies_df.Genre == 'Drama']
```

Out[36]:

	Film	Genre	CriticRatings	AudienceRatings	BudgetMillion	Year
10	88 Minutes	Drama	5	51	30	2007
11	A Dangerous Method	Drama	79	89	20	2011
13	A Serious Man	Drama	89	64	7	2009
18	Albert Nobbs	Drama	53	43	8	2011
23	All Good Things	Drama	33	64	20	2010
...	...	...	...	...	...	...
529	War Horse	Drama	77	73	66	2011
532	Water For Elephants	Drama	60	72	38	2011
534	We Own the Night	Drama	55	63	21	2007
541	Whip It	Drama	84	73	15	2009
545	Winter's Bone	Drama	94	73	2	2010

101 rows × 6 columns

In [43]:
```
sns.histplot(movies_df[movies_df.Genre == 'Action'].BudgetMillion, bins = 20)
sns.histplot(movies_df[movies_df.Genre == 'Thriller'].BudgetMillion, bins = 20)
sns.histplot(movies_df[movies_df.Genre == 'Drama'].BudgetMillion, bins = 20)
plt.legend()
plt.show()
```

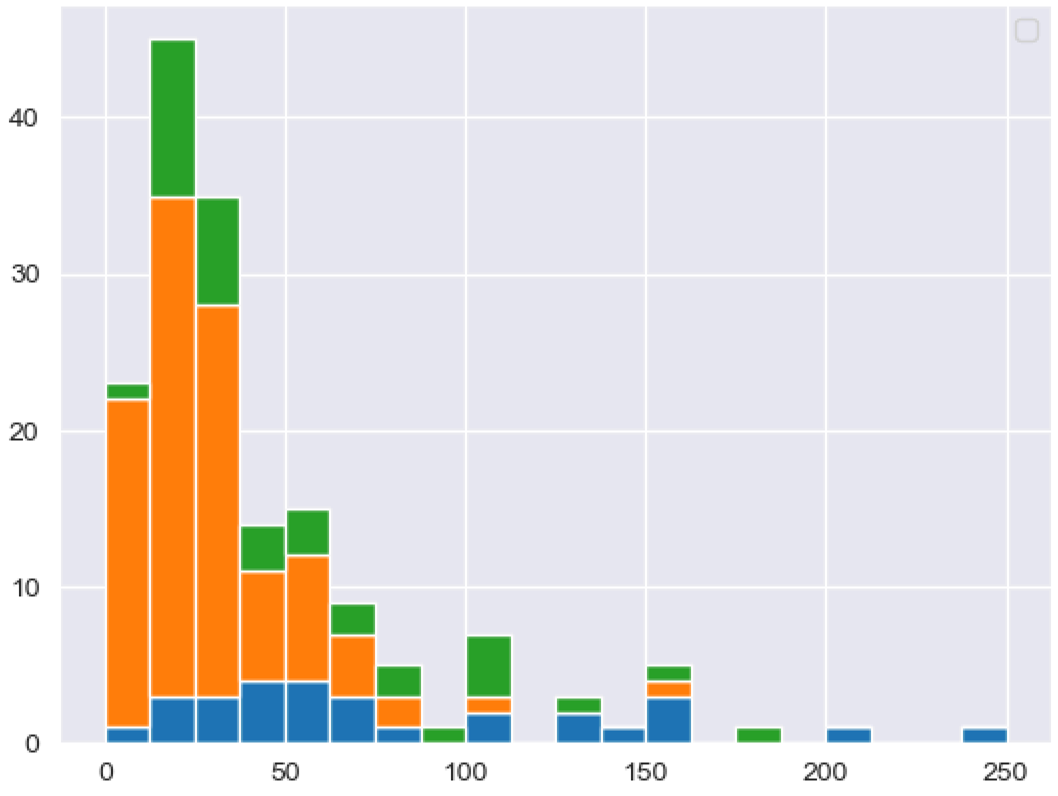


```
In [44]: movies_df.head()
```

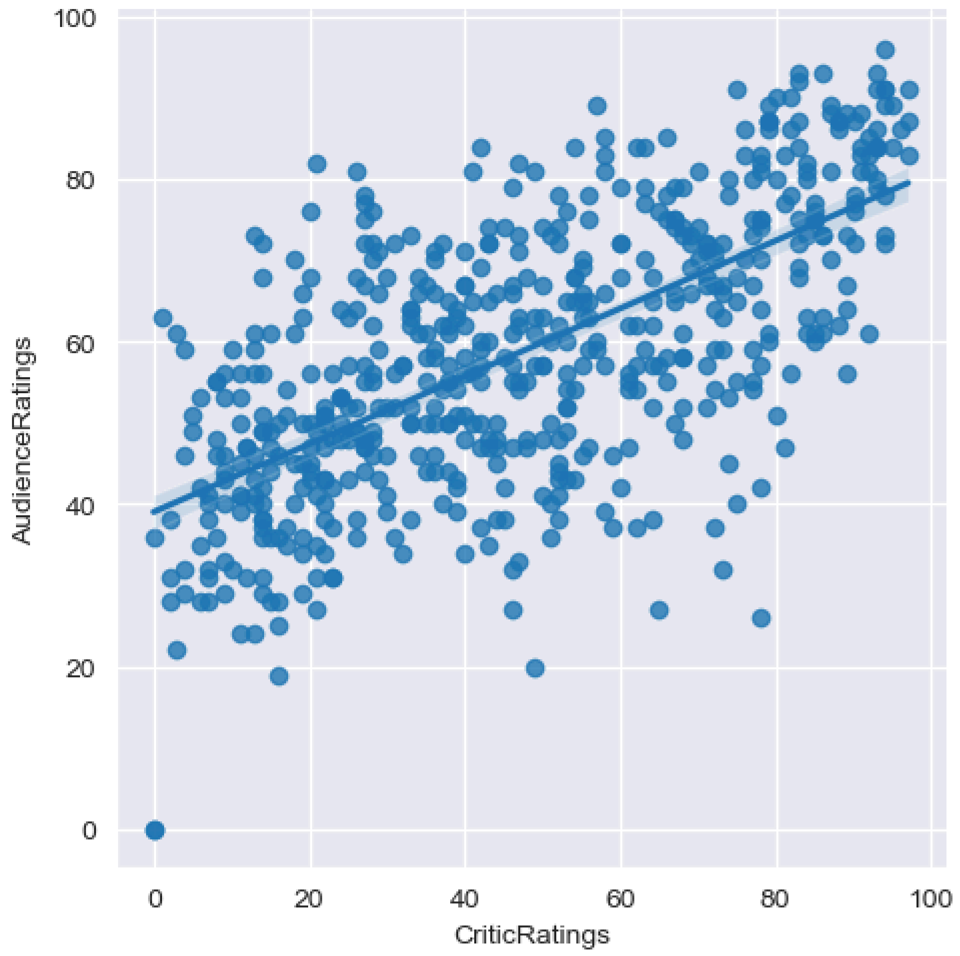
Out[44]:

	Film	Genre	CriticRatings	AudienceRatings	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

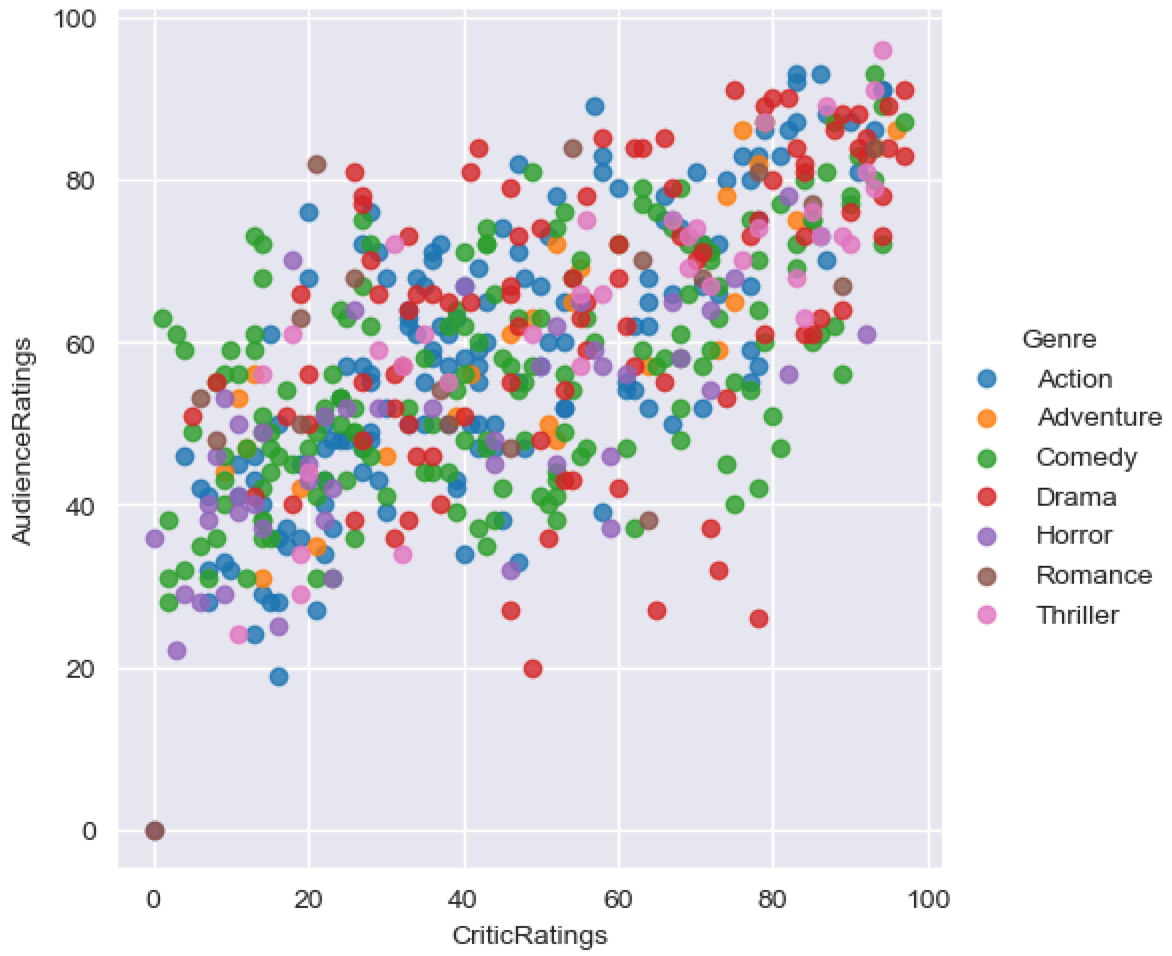
```
In [50]: plt.hist([movies_df[movies_df.Genre == 'Adventure'].BudgetMillion, \
                  movies_df[movies_df.Genre == 'Drama'].BudgetMillion, \
                  movies_df[movies_df.Genre == 'Thriller'].BudgetMillion], bins= 20, stacked=True)
         plt.legend()
         plt.show()
```



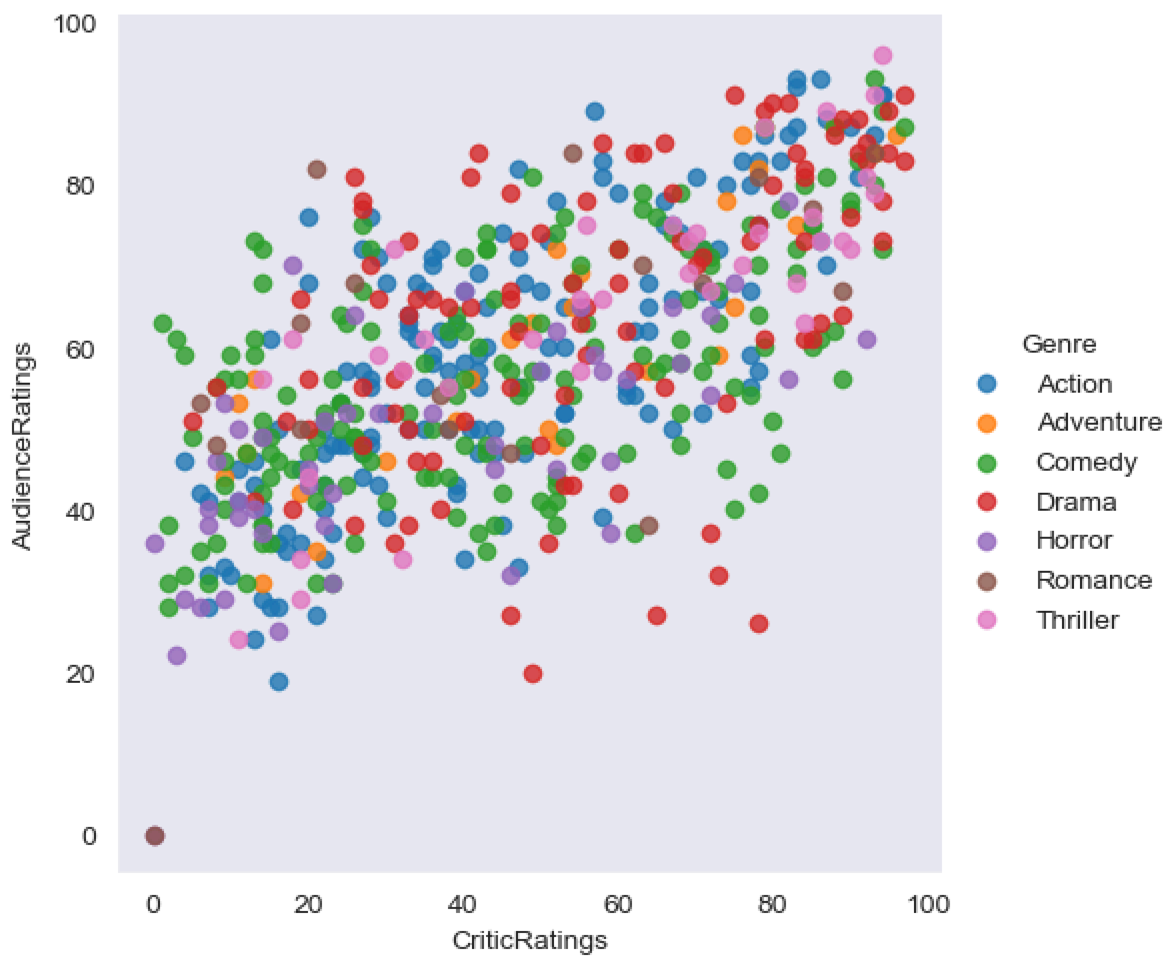
```
In [51]: sns.lmplot(data=movies_df,x='CriticRatings',y='AudienceRatings')
         plt.show()
```



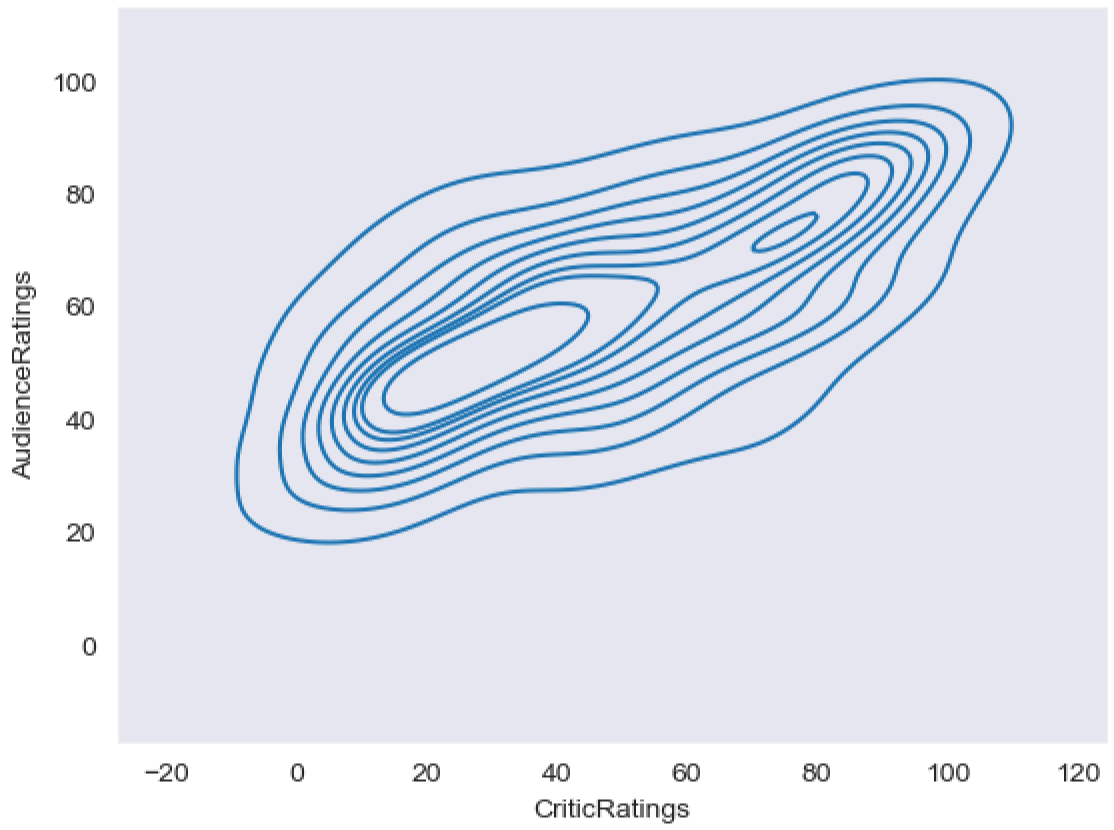
```
In [54]: sns.lmplot(data=movies_df,x='CriticRatings',y='AudienceRatings', fit_reg=False, hue='Genre')
         plt.show()
```



In [58]:
```
sns.set_style('dark')
vis = sns.lmplot(data=movies_df,x='CriticRatings',y='AudienceRatings', fit_reg=False, hue='Genre
plt.show()
```

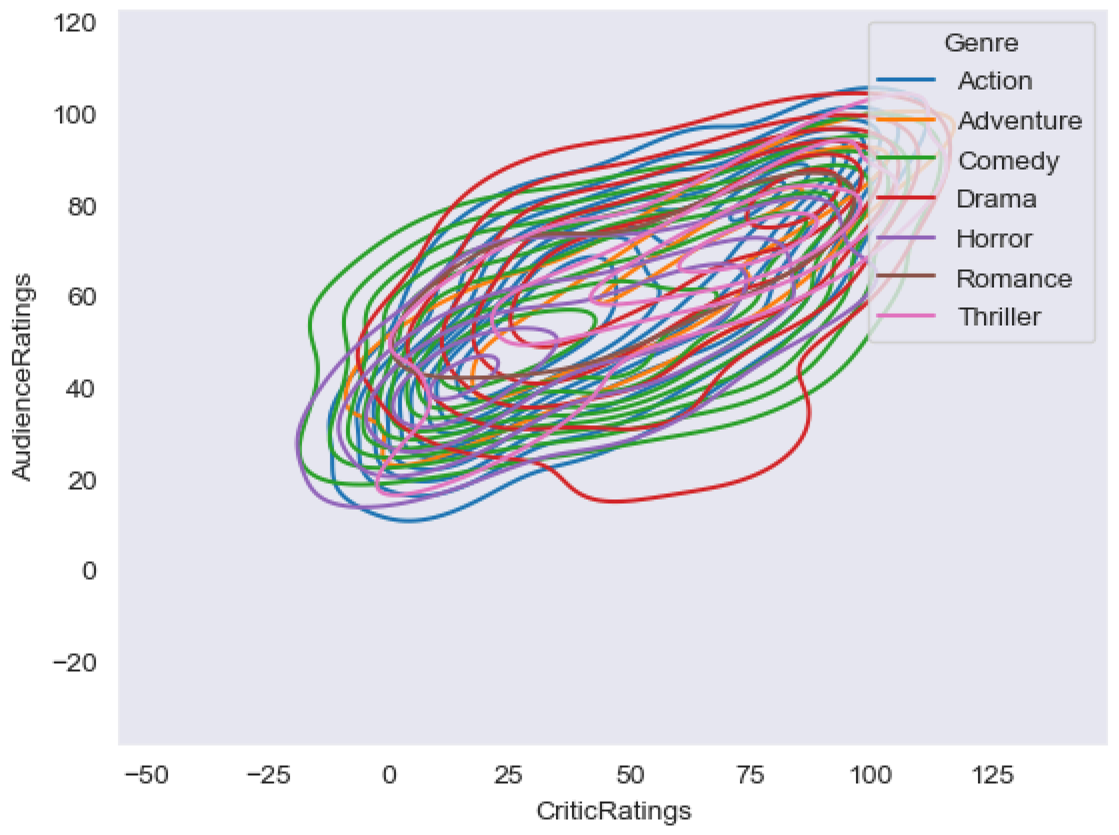


In [62]:
```
sns.kdeplot(data=movies_df, x='CriticRatings', y='AudienceRatings')
plt.show()
```

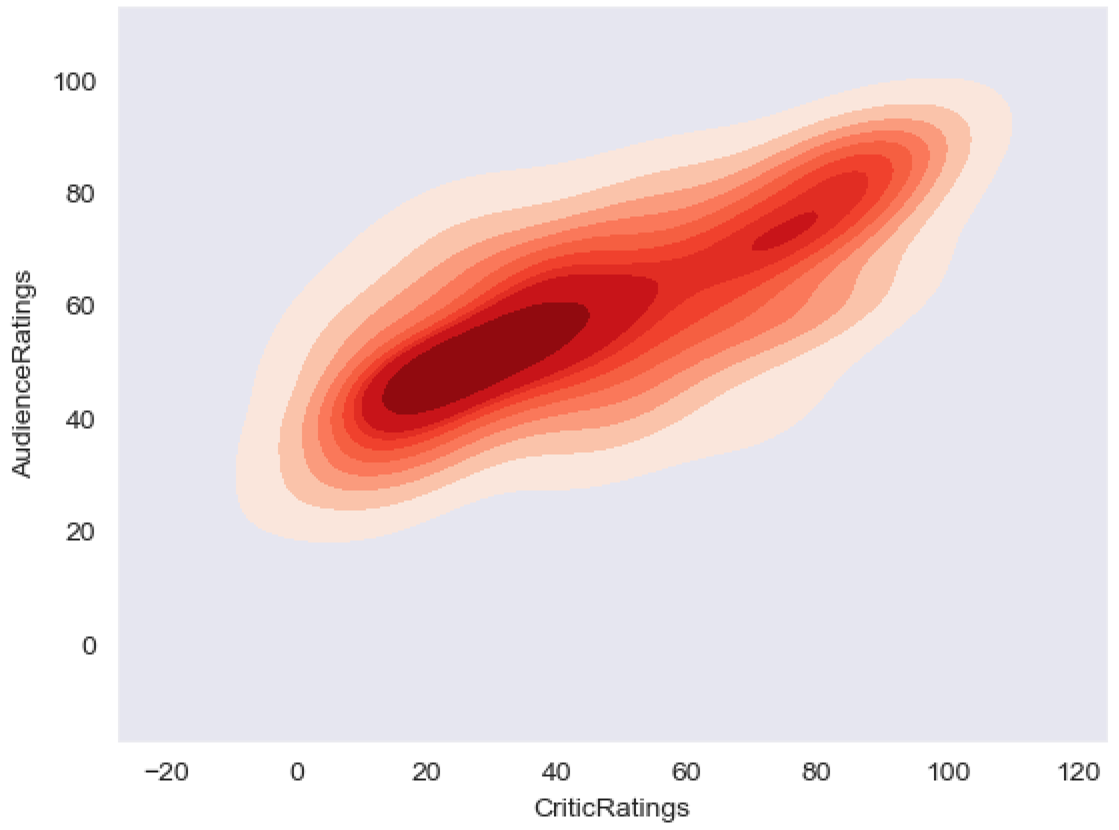


In [63]:
```
sns.kdeplot(data=movies_df, x='CriticRatings', y='AudienceRatings', hue='Genre')
```

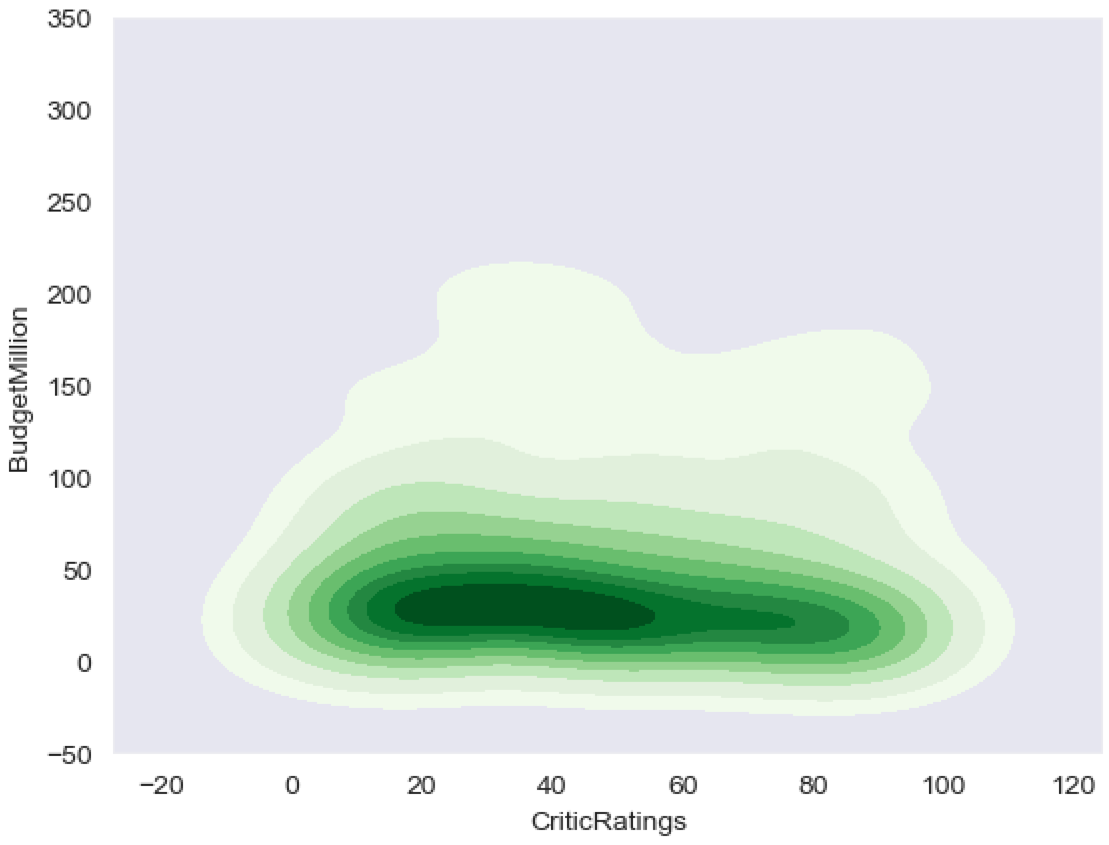
```
plt.show()
```



In [64]:
```
sns.kdeplot(data=movies_df, x='CriticRatings', y='AudienceRatings', fill=True, cmap="Reds")
plt.show()
```



In [67]:
```
sns.kdeplot(data=movies_df, x='CriticRatings', y='BudgetMillion', fill=True, cmap="Greens")
plt.show()
```



```
In [68]: sns.boxplot(data = movies_df, x='Genre', y = 'CriticRating')
         plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[68], line 1
----> 1 sns.boxplot(data = movies_df, x='Genre', y = 'CriticRating')
      2 plt.show()

File C:\ProgramData\anaconda3\Lib\site-packages\seaborn\categorical.py:1597, in boxplot(data, x,
y, hue, order, hue_order, orient, color, palette, saturation, fill, dodge, width, gap, whis, line
color, linewidth, fliersize, hue_norm, native_scale, log_scale, formatter, legend, ax, **kwargs)
   1589 def boxplot(
   1590     data=None, *, x=None, y=None, hue=None, order=None, hue_order=None,
   1591     orient=None, color=None, palette=None, saturation=.75, fill=True,
   (...)
   1594     legend="auto", ax=None, **kwargs
   1595 ):
-> 1597     p = _CategoricalPlotter(
   1598         data=data,
   1599         variables=dict(x=x, y=y, hue=hue),
   1600         order=order,
   1601         orient=orient,
   1602         color=color,
   1603         legend=legend,
   1604     )
   1606     if ax is None:
   1607         ax = plt.gca()

File C:\ProgramData\anaconda3\Lib\site-packages\seaborn\categorical.py:67, in _CategoricalPlotte
r.__init__(self, data, variables, order, orient, require_numeric, color, legend)
     56 def __init__(
     57     self,
     58     data=None,
   (...)
     64     legend="auto",
     65 ):
---> 67     super().__init__(data=data, variables=variables)
     69     # This method takes care of some bookkeeping that is necessary because the
     70     # original categorical plots (prior to the 2021 refactor) had some rules that
     71     # don't fit exactly into VectorPlotter logic. It may be wise to have a second
   (...)
     76     # default VectorPlotter rules. If we do decide to make orient part of the
     77     # _base variable assignment, we'll want to figure out how to express that.
     78     if self.input_format == "wide" and orient in ["h", "y"]:

File C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_base.py:634, in VectorPlotter.__init__(s
elf, data, variables)
    629 # var_ordered is relevant only for categorical axis variables, and may
    630 # be better handled by an internal axis information object that tracks
    631 # such information and is set up by the scale_* methods. The analogous
    632 # information for numeric axes would be information about log scales.
    633 self._var_ordered = {"x": False, "y": False}  # alt., used DefaultDict
--> 634 self.assign_variables(data, variables)
    636 # TODO Lots of tests assume that these are called to initialize the
    637 # mappings to default values on class initialization. I'd prefer to
    638 # move away from that and only have a mapping when explicitly called.
    639 for var in ["hue", "size", "style"]:

File C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_base.py:679, in VectorPlotter.assign_var
iables(self, data, variables)
    674 else:
    675     # When dealing with long-form input, use the newer PlotData
    676     # object (internal but introduced for the objects interface)
    677     # to centralize / standardize data consumption logic.
    678     self.input_format = "long"
--> 679     plot_data = PlotData(data, variables)
    680     frame = plot_data.frame
    681     names = plot_data.names

File C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_core\data.py:58, in PlotData.__init__(se
lf, data, variables)
     51 def __init__(
     52     self,
     53     data: DataSource,
     54     variables: dict[str, VariableSpec],
     55 ):
     57     data = handle_data_source(data)
---> 58     frame, names, ids = self._assign_variables(data, variables)
     60     self.frame = frame
     61     self.names = names

File C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_core\data.py:232, in PlotData._assign_va
riables(self, data, variables)
    230     else:
    231         err += "An entry with this name does not appear in `data`."
--> 232     raise ValueError(err)
    234 else:
    235 
    236     # Otherwise, assume the value somehow represents data
    237 
    238     # Ignore empty data structures
    239     if isinstance(val, Sized) and len(val) == 0:

ValueError: Could not interpret value `CriticRating` for `y`. An entry with this name does not ap
pear in `data`.
```

```
In [ ]:
```