In [1]:
```
# MOVIE RATING ANALYTICS (ADVANCED VISULIZATION)

import pandas as pd
import os
```

In [2]:
```
os.getcwd() # if you want to change the working directory
```

Out[2]:
```
'C:\\Users\\Sai\\A in Acodes\\data science\\projects\\Basic\\MOVIE RATINGS _ ADVANCE VISUALIZATION _ EDA 1'
```

In [5]:
```
movies = pd.read_csv(r"C:\Users\Sai\A in Acodes\data science\projects\Basic\MOVI
```

In [16]:
```
movies
```

Out[16]:

	Film	Genre	CriticRating	AudienceRating	BudgetMillions	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 [17]:
```
len(movies)
```

Out[17]:
```
559
```

In [18]:
```
movies.head()
```

Out[18]:

	Film	Genre	CriticRating	AudienceRating	BudgetMillions	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 [19]:
```
movies.tail()
```

Out[19]:

	Film	Genre	CriticRating	AudienceRating	BudgetMillions	Year
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 [20]:
```
movies.columns
```

Out[20]:
```
Index(['Film', 'Genre', 'CriticRating', 'AudienceRating', 'BudgetMillions',
       'Year'],
      dtype='object')
```

In [21]:
```
movies.columns = ['Film', 'Genre', 'CriticRating', 'AudienceRating','BudgetMilli
```

In [22]:
```
movies.head() # Removed spaces & % removed noise characters
```

Out[22]:

	Film	Genre	CriticRating	AudienceRating	BudgetMillions	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 [23]:
```
movies.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   BudgetMillions  559 non-null    int64 
 5   Year            559 non-null    int64 
dtypes: int64(4), object(2)
memory usage: 26.3+ KB
```

In [24]:
```
movies.describe()
# if you look at the year the data type is int but when you look at the mean val
# we have to change to categroy type
# also from object datatype we will convert to category datatypes
#
```

Out[24]:

	CriticRating	AudienceRating	BudgetMillions	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 [25]:
```
movies['Film']
#movies['Audience Ratings %']
```

Out[25]:
```
0      (500) Days of Summer
1                10,000 B.C.
2                 12 Rounds
3                 127 Hours
4                 17 Again
               ...
554           Your Highness
555         Youth in Revolt
556                  Zodiac
557            Zombieland
558              Zookeeper
Name: Film, Length: 559, dtype: object
```

In [26]:
```
movies.Film
```

Out[26]:
```
0      (500) Days of Summer
1                10,000 B.C.
2                 12 Rounds
3                  127 Hours
4                  17 Again
               ...
554            Your Highness
555          Youth in Revolt
556                   Zodiac
557              Zombieland
558                Zookeeper
Name: Film, Length: 559, dtype: object
```

In [27]:
```
movies.Film = movies.Film.astype('category')
```

In [28]:
```
movies.Film
```

Out[28]:
```
0      (500) Days of Summer
1                10,000 B.C.
2                 12 Rounds
3                  127 Hours
4                  17 Again
               ...
554            Your Highness
555          Youth in Revolt
556                   Zodiac
557              Zombieland
558                Zookeeper
Name: Film, Length: 559, dtype: category
Categories (559, object): ['(500) Days of Summer ', '10,000 B.C.', '12 Rounds
', '127 Hours', ..., 'Youth in Revolt', 'Zodiac', 'Zombieland ', 'Zookeeper']
```

In [29]:
```
movies.head()
```

Out[29]:

	Film	Genre	CriticRating	AudienceRating	BudgetMillions	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 [30]:
```
movies.info()

# now the same thing we will change genra to category & year to category
```

```
<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   BudgetMillions  559 non-null    int64   
 5   Year            559 non-null    int64   
dtypes: category(1), int64(4), object(1)
memory usage: 43.6+ KB
```

In [31]:
```
movies.Genre = movies.Genre.astype('category')
movies.Year = movies.Year.astype('category')
```

In [32]:
```
movies.Genre
```

Out[32]:
```
0         Comedy
1      Adventure
2         Action
3      Adventure
4         Comedy
          ...   
554       Comedy
555       Comedy
556     Thriller
557       Action
558       Comedy
Name: Genre, Length: 559, dtype: category
Categories (7, object): ['Action', 'Adventure', 'Comedy', 'Drama', 'Horror', 'R
omance', 'Thriller']
```

In [33]:
```
movies.Year # is it real no. year you can take average,min,max but out come have
```

Out[33]:
```
0      2009
1      2008
2      2009
3      2010
4      2009
       ...
554    2011
555    2009
556    2007
557    2009
558    2011
Name: Year, Length: 559, dtype: category
Categories (5, int64): [2007, 2008, 2009, 2010, 2011]
```

In [34]:
```
movies.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   BudgetMillions  559 non-null    int64   
 5   Year            559 non-null    category
dtypes: category(3), int64(3)
memory usage: 36.5 KB
```

In [35]:
```
movies.Genre.cat.categories
```

Out[35]:
```
Index(['Action', 'Adventure', 'Comedy', 'Drama', 'Horror', 'Romance',
       'Thriller'],
      dtype='object')
```

In [36]:
```
movies.describe()
#now when you see the describt you will get only integer value mean, standard de
```

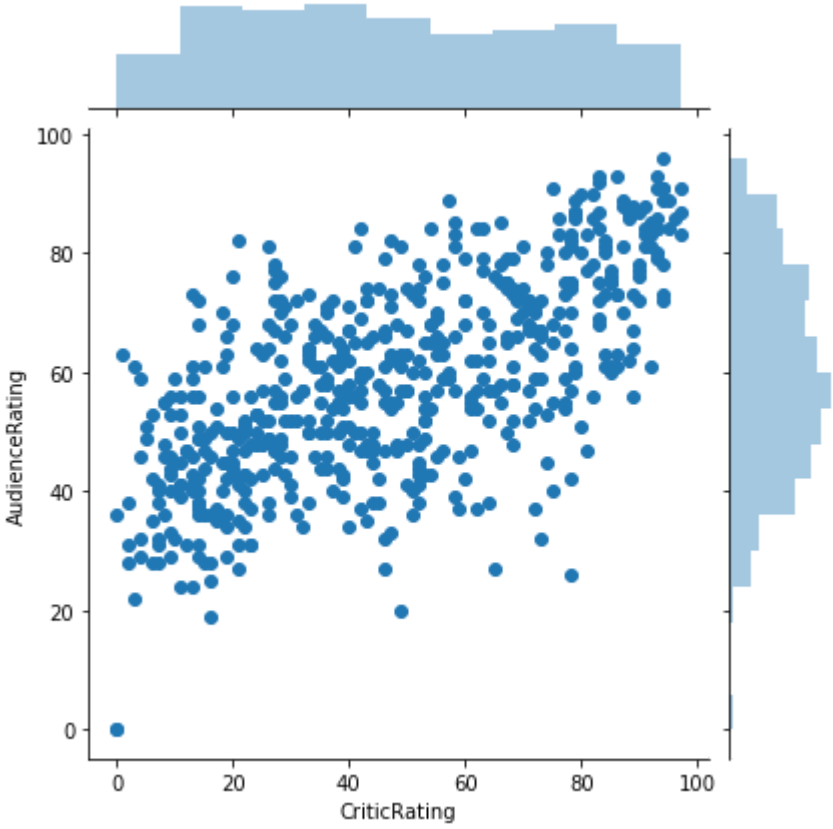
Out[36]:

	CriticRating	AudienceRating	BudgetMillions
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

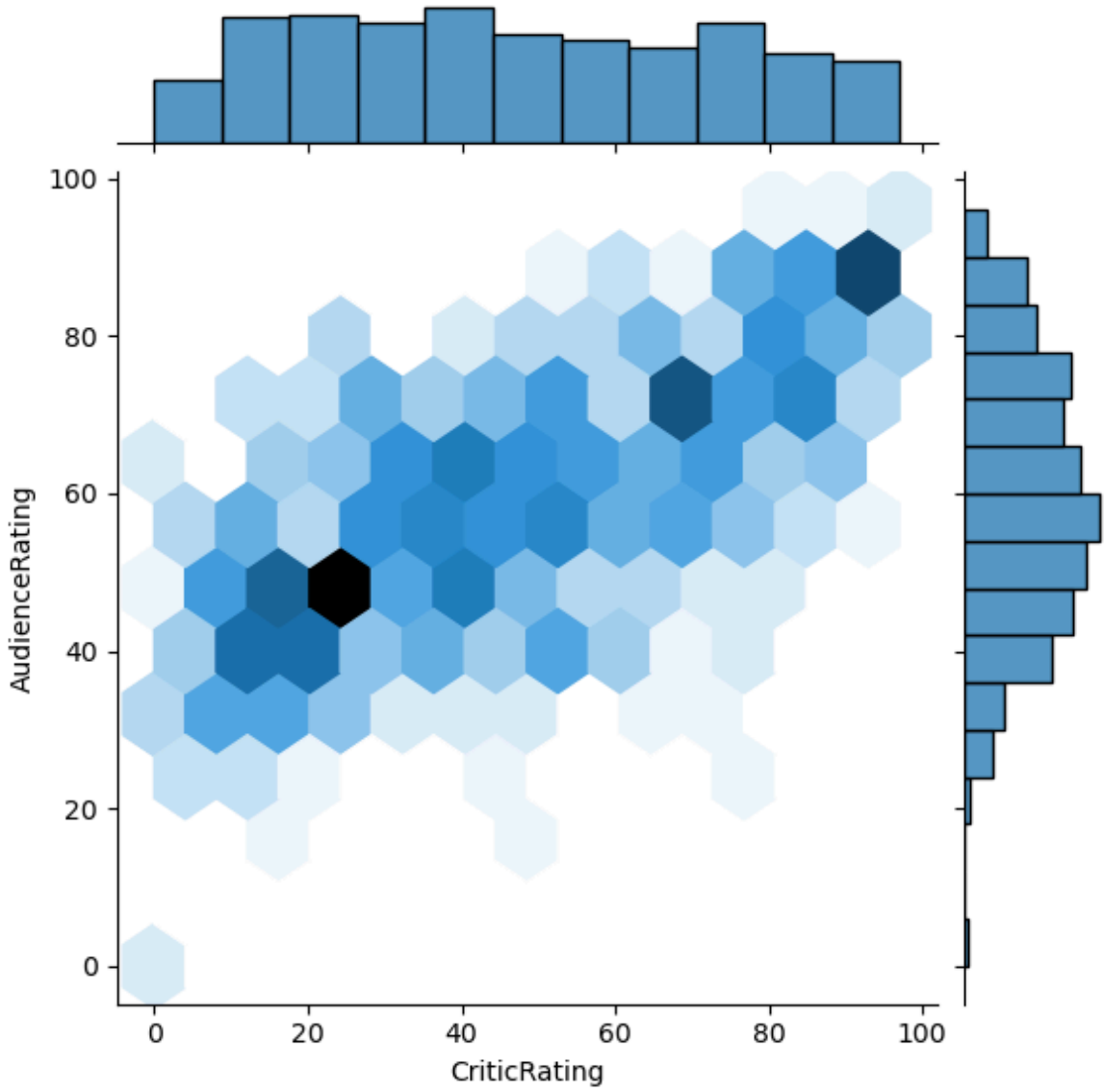
In [37]:
```
# How to working with joint plots

from matplotlib import pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

- basically joint plot is a scatter plot & it find the relation b/w audiene & critics
- also if you look up you can find the uniform distribution (critics)and normal distriution (audience)

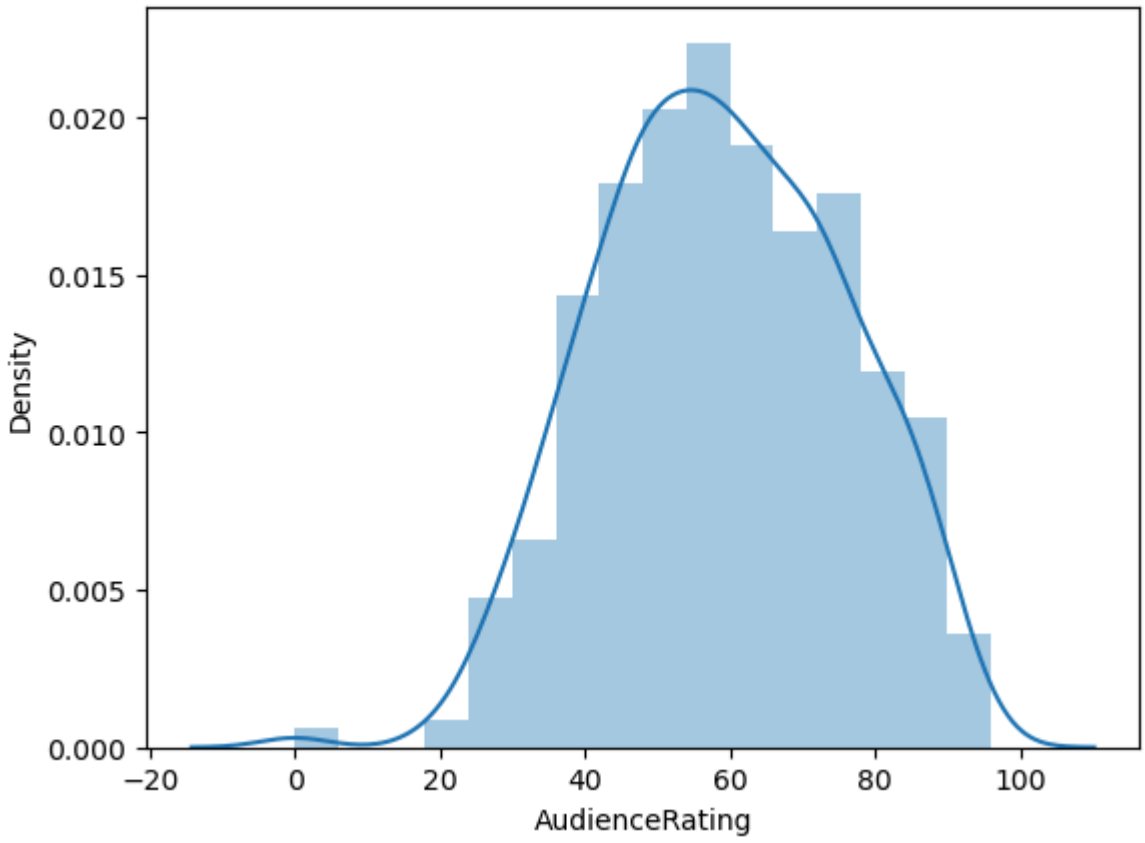
In [26]:
```
j = sns.jointplot( data = movies, x = 'CriticRating', y = 'AudienceRating')
# Audience rating is more dominant then critics rating
# Based on this we find out as most people are most liklihood to watch audience
# let me explain the excel - if you filter audience rating & critic rating. crit
```



In [40]:
```
j = sns.jointplot( data = movies, x = 'CriticRating', y = 'AudienceRating', kind

#j = sns.jointplot( data = movies, x = 'CriticRating', y = 'AudienceRating', kin
```

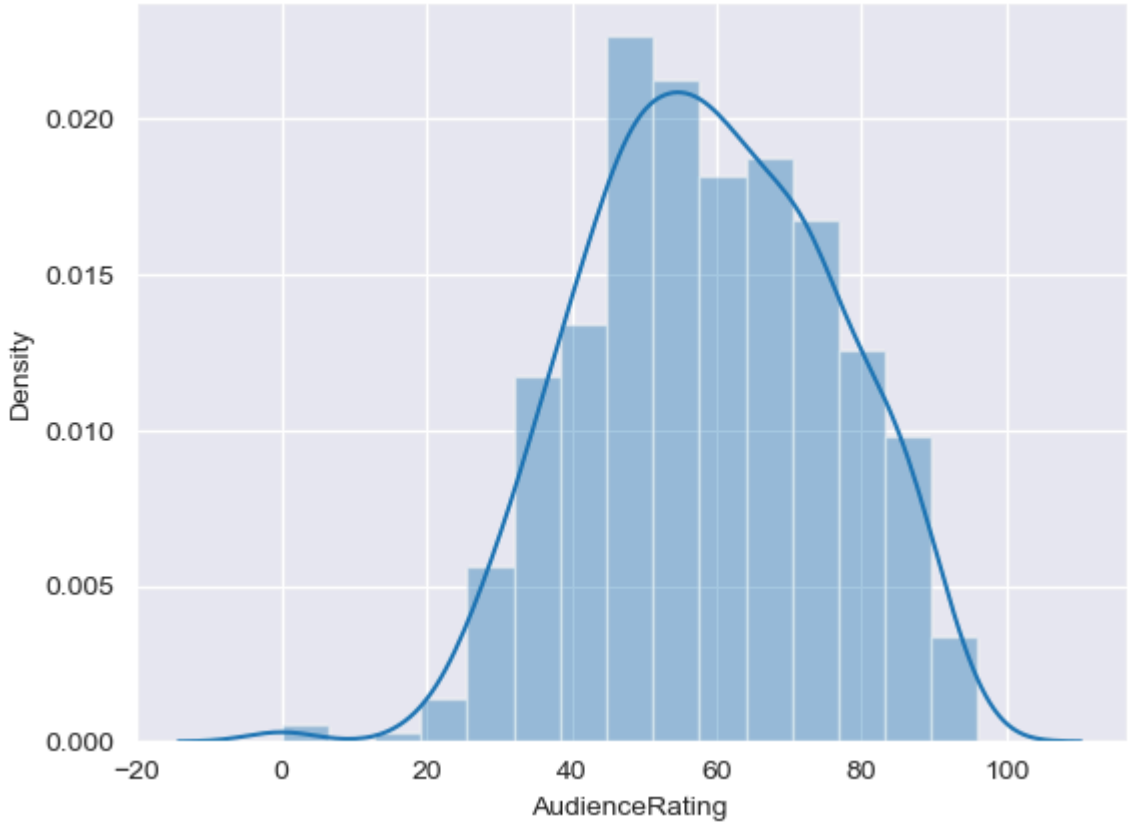


In [41]:
```
#Histograms

# <<< chat1

m1 = sns.distplot(movies.AudienceRating)

#y - axis generated by seaborn automatically that is the powefull of seaborn gal
```

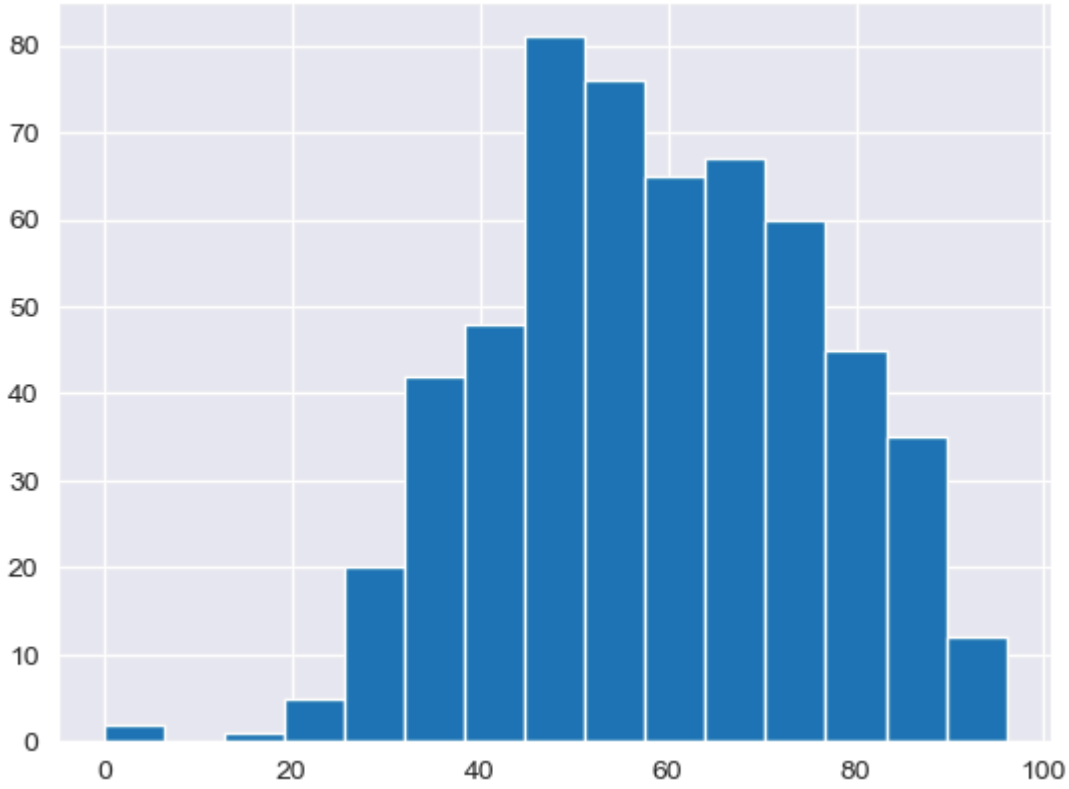


In [42]:
```
sns.set_style('darkgrid')
```

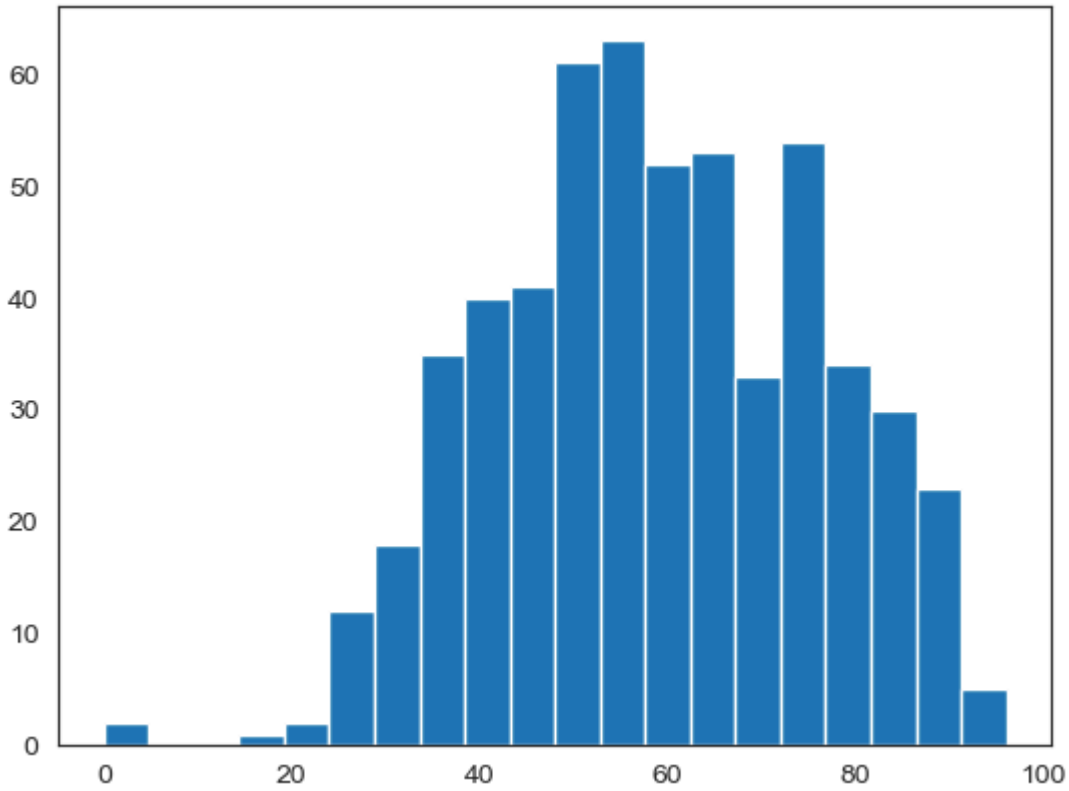
In [43]:
```
m2 = sns.distplot(movies.AudienceRating, bins = 15)
```



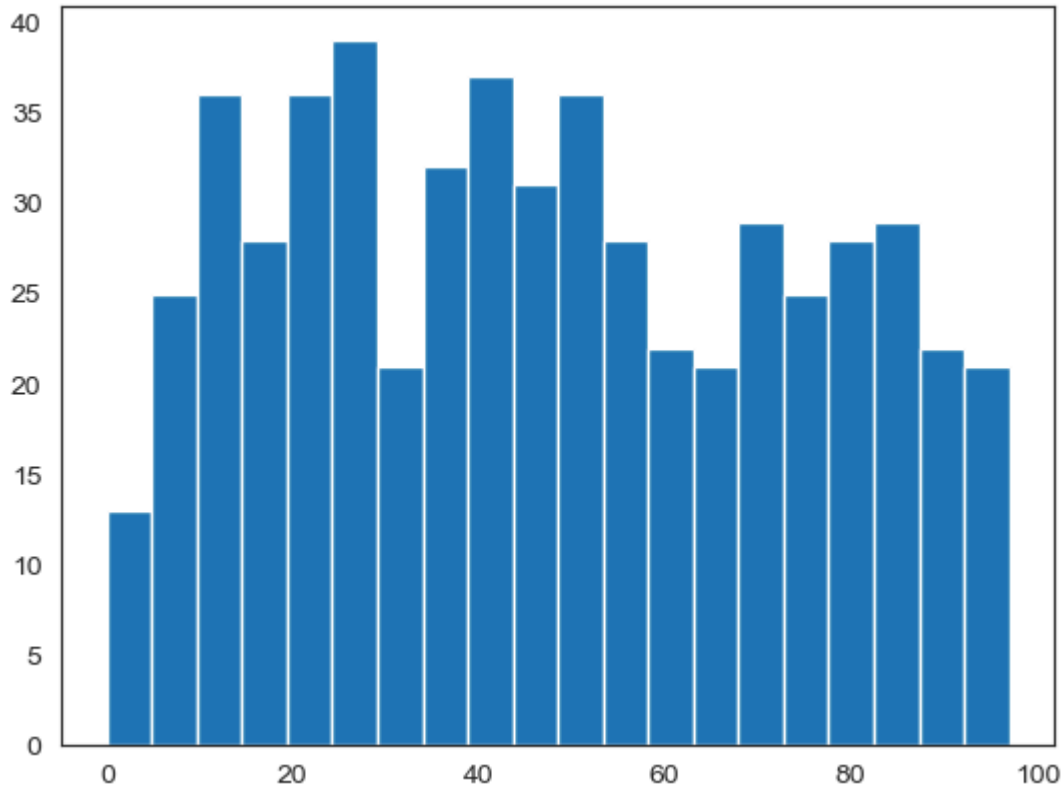
In [44]:
```
#sns.set_style('darkgrid')
n1 = plt.hist(movies.AudienceRating, bins=15)
```



In [45]:
```
sns.set_style('white') #normal distribution & called as bell curve
n1 = plt.hist(movies.AudienceRating, bins=20)
```

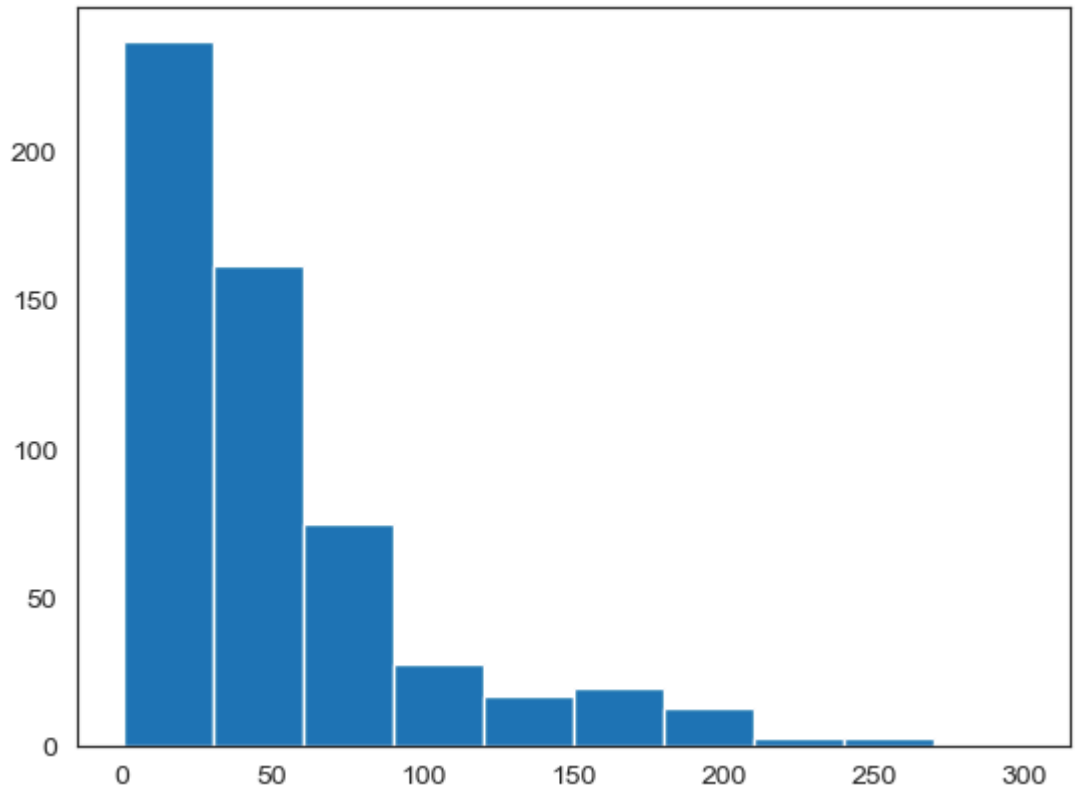


In [46]:
```
n1 = plt.hist(movies.CriticRating, bins=20) #uniform distribution
```

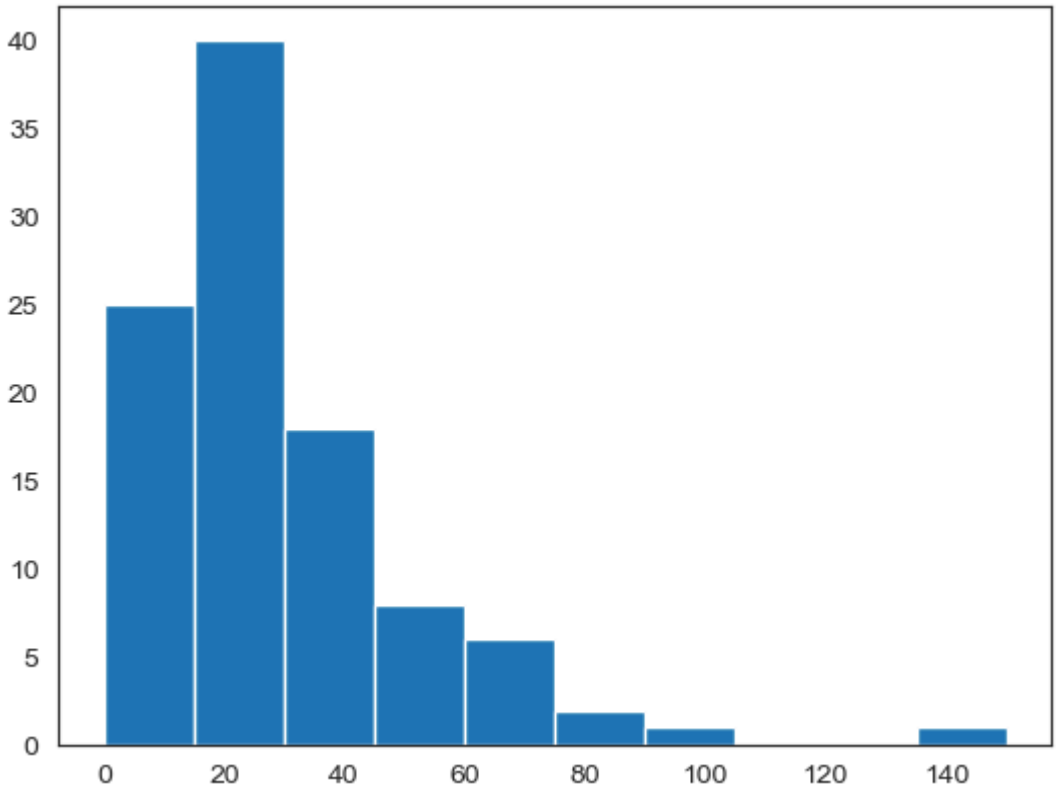


In [47]:
```
# <<< chat - 2

# Creating stacked histograms & this is bit tough to understand
```

In [48]:
```
#h1 = plt.hist(movies.BudgetMillions)

plt.hist(movies.BudgetMillions)
plt.show()
```



In [49]:
```
plt.hist(movies[movies.Genre == 'Drama'].BudgetMillions)
plt.show()
```



In [50]:
```
movies.head()
```

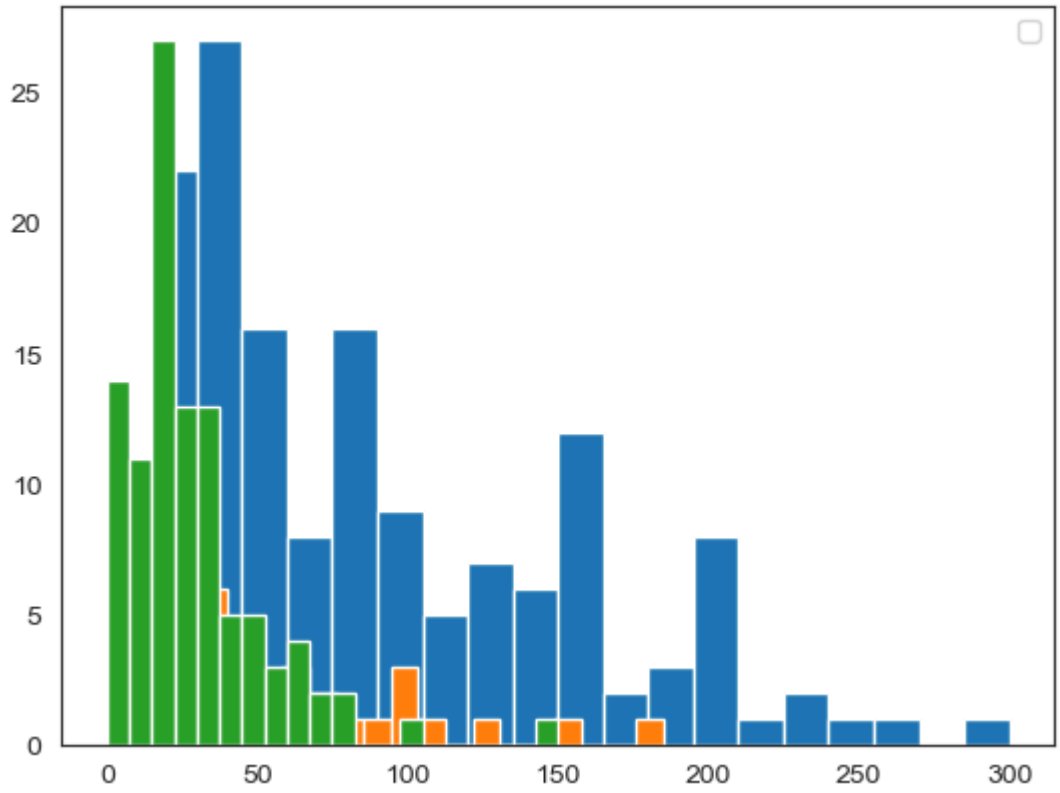
Out[50]:

	Film	Genre	CriticRating	AudienceRating	BudgetMillions	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

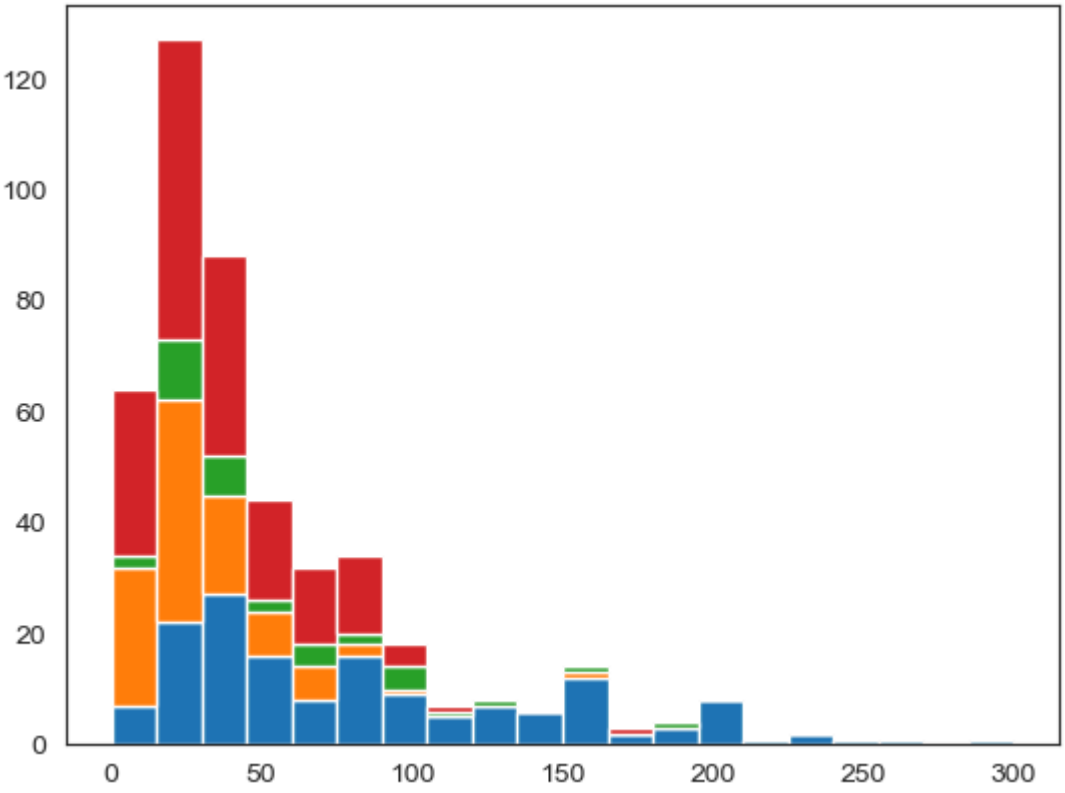
In [51]:
```
#movies.Genre.unique()
```

In [52]:
```
# Below plots are stacked histogram becuase overlaped

plt.hist(movies[movies.Genre == 'Action'].BudgetMillions, bins = 20)
plt.hist(movies[movies.Genre == 'Thriller'].BudgetMillions, bins = 20)
plt.hist(movies[movies.Genre == 'Drama'].BudgetMillions, bins = 20)
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label sta
rt with an underscore are ignored when legend() is called with no argument.
```



In [53]:
```
plt.hist([movies[movies.Genre == 'Action'].BudgetMillions,\
          movies[movies.Genre == 'Drama'].BudgetMillions, \
          movies[movies.Genre == 'Thriller'].BudgetMillions, \
          movies[movies.Genre == 'Comedy'].BudgetMillions],
         bins = 20, stacked = True)
plt.show()
```

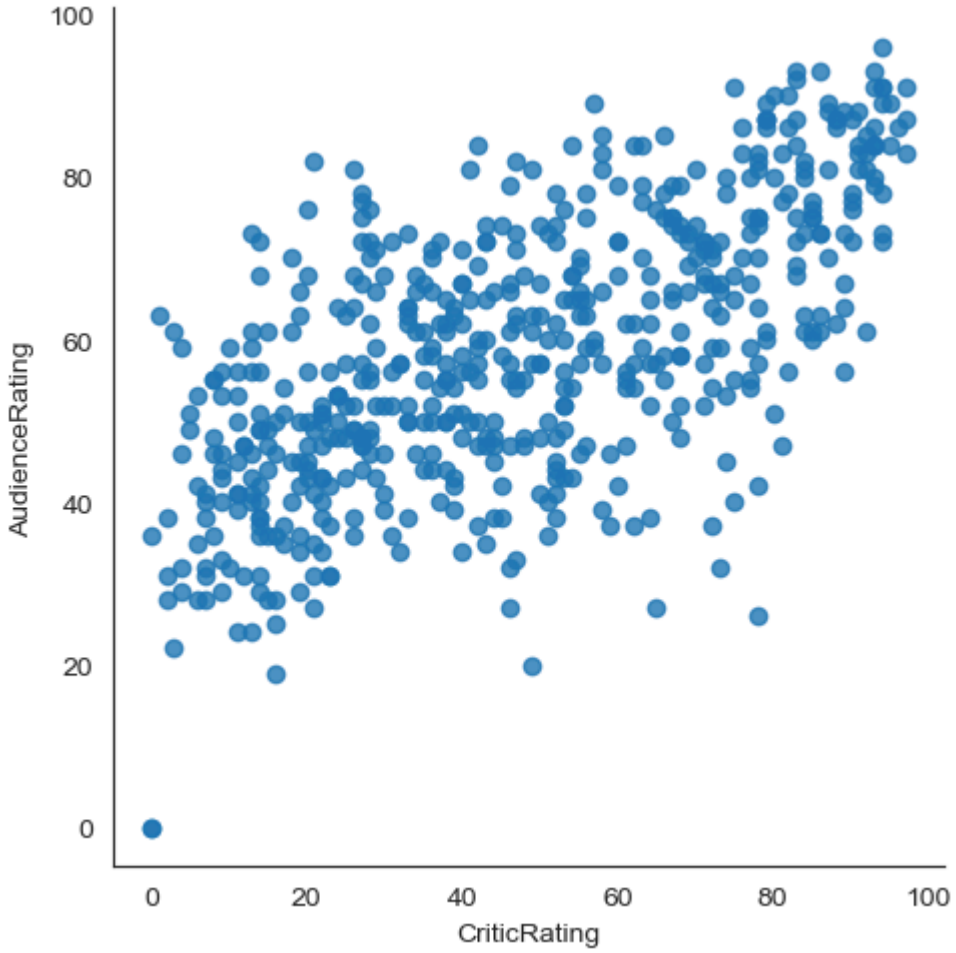


In [54]:
```
# if you have 100 categories you cannot copy & paste all the things
```

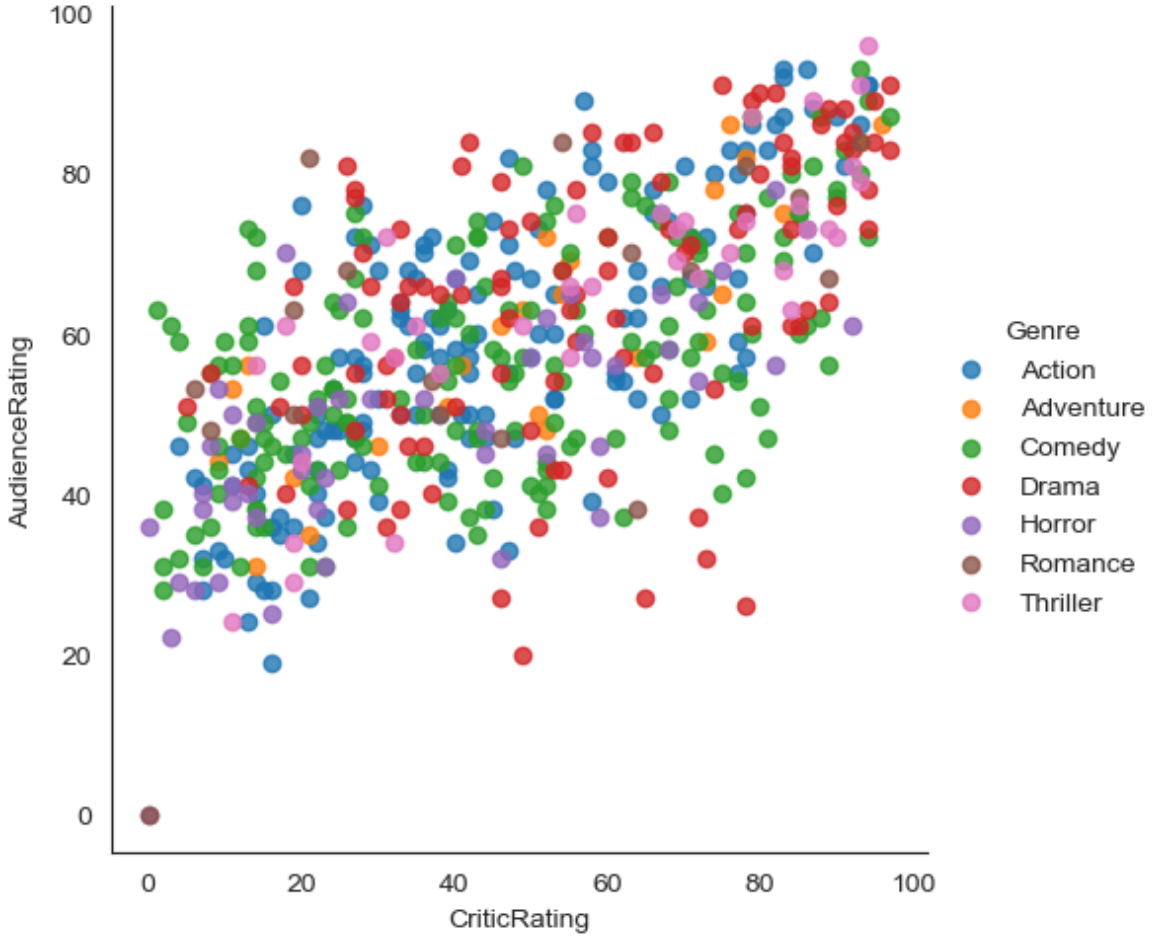
```
for gen in movies.Genre.cat.categories:
    print(gen)
```

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

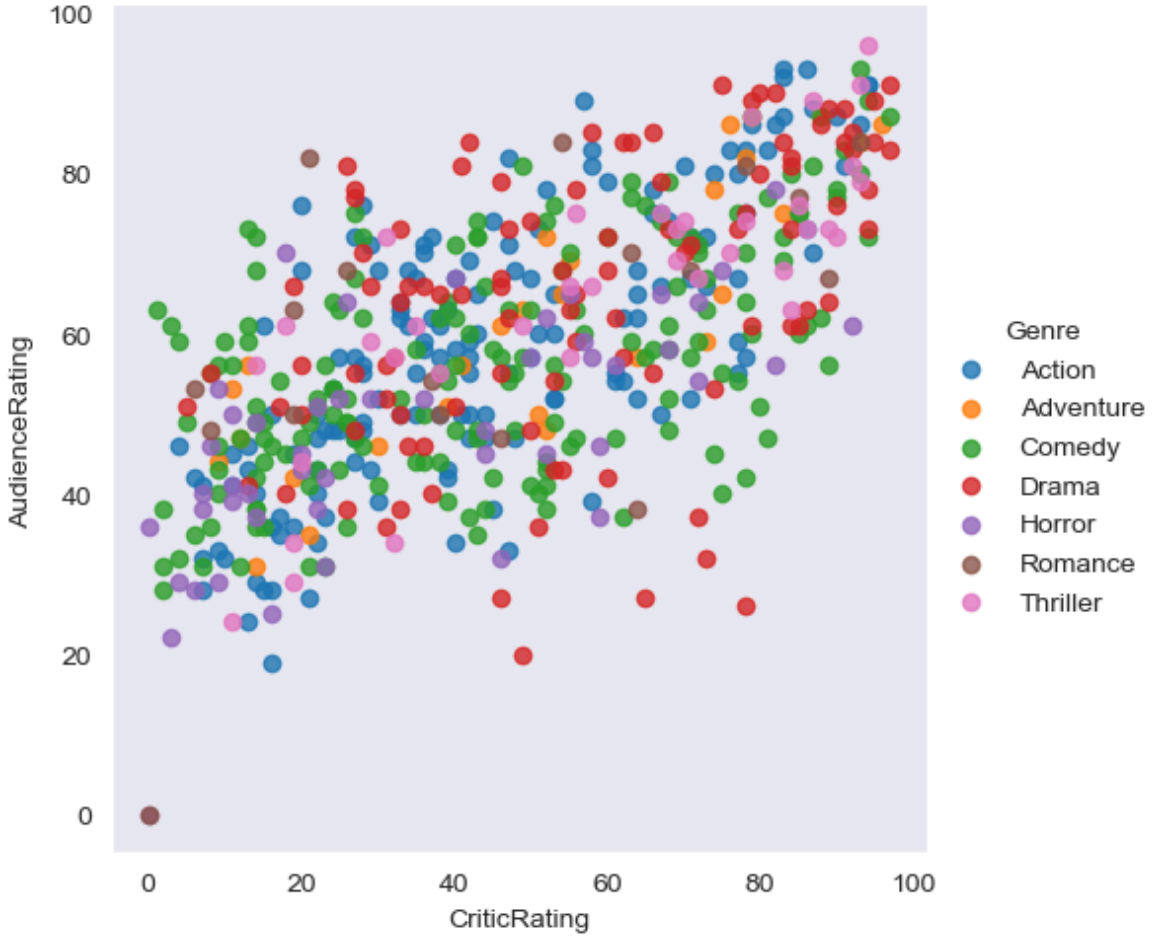
In [55]:
```
vis1 = sns.lmplot(data=movies, x='CriticRating', y='AudienceRating',\
                  fit_reg=False)
```



In [56]:
```
vis1 = sns.lmplot(data=movies, x='CriticRating', y='AudienceRating',\
                  fit_reg=False, hue = 'Genre')
```

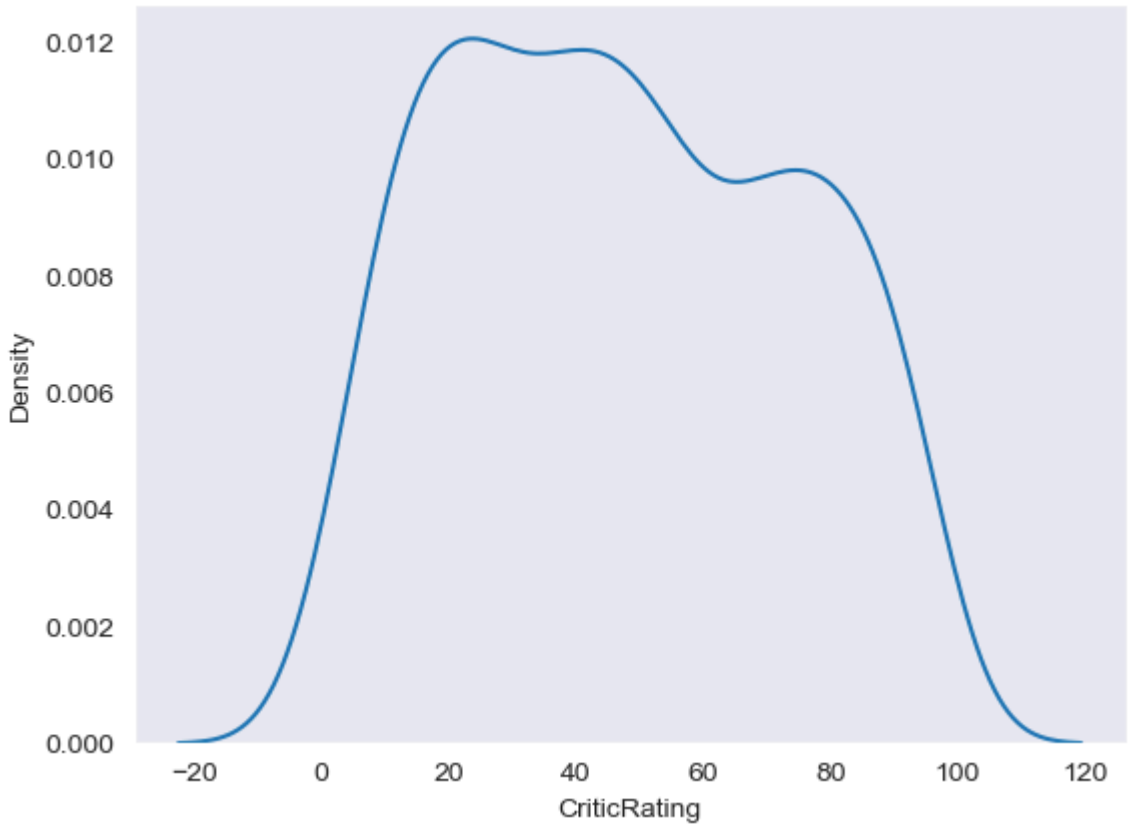


In [75]:
```
vis1 = sns.lmplot(data=movies, x='CriticRating', y='AudienceRating',\
                  fit_reg=False, hue = 'Genre',aspect=1)
```

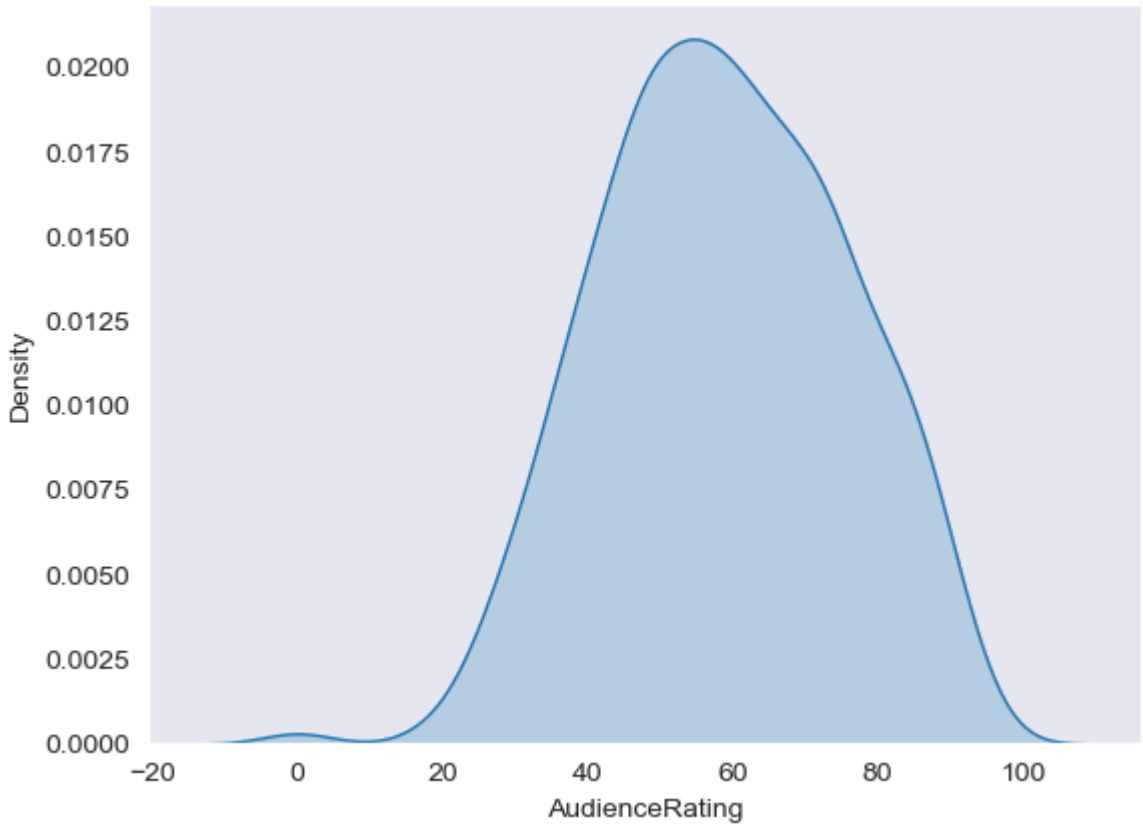


In [76]:
```
# Kernal Density Estimate plot ( KDE PLOT)
# how can i visulize audience rating & critics rating . using scatterplot
```

In [79]:
```
k1 = sns.kdeplot(movies.CriticRating)

# where do u find more density and how density is distibuted across from the the
# center point is kernal this is calld KDE & insteade of dots it visualize like
# we can able to clearly see the spread at the audience ratings
```



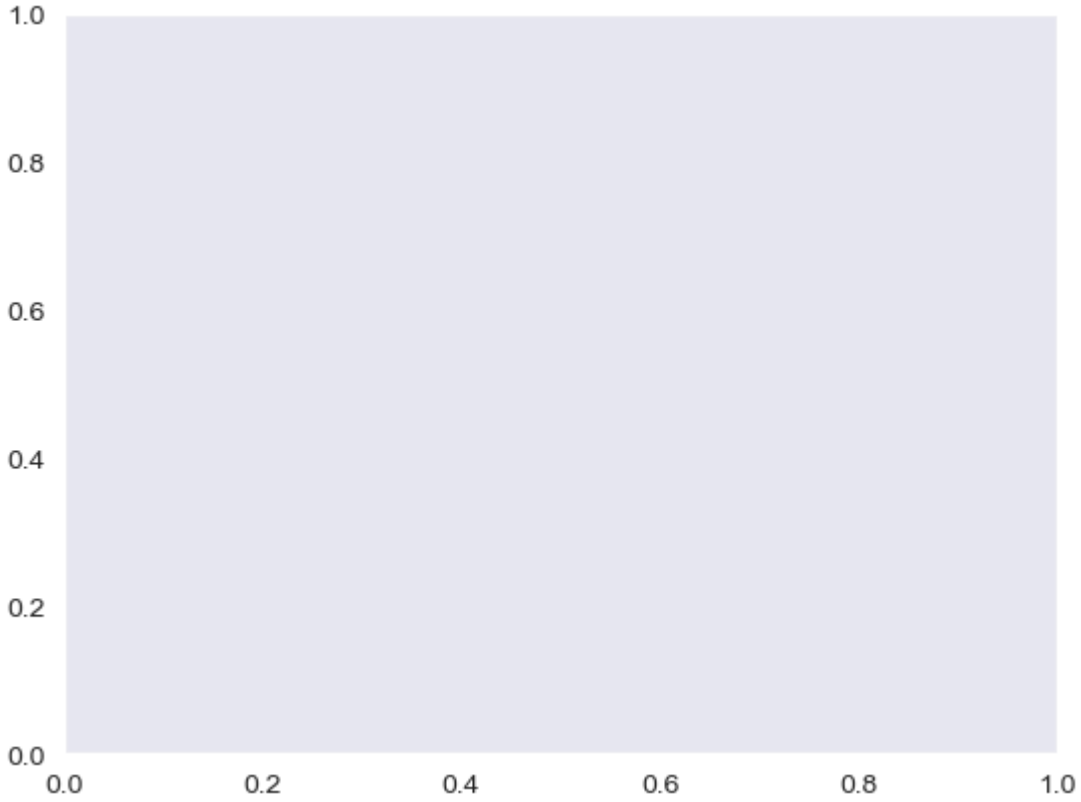
In [81]:
```
k1 = sns.kdeplot(movies.AudienceRating,shade = True,shade_lowest=False,cmap='Red
```



In [82]:
```
k2 = sns.kdeplot(movies.CriticRating,shade_lowest=False,cmap='Greens_r')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[82], line 1
----> 1 k2 = sns.kdeplot(movies.CriticRating,shade_lowest=False,cmap=          )

File ~\anaconda3\Lib\site-packages\seaborn\distributions.py:1701, in kdeplot(dat
a, x, y, hue, weights, palette, hue_order, hue_norm, color, fill, multiple, commo
n_norm, common_grid, cumulative, bw_method, bw_adjust, warn_singular, log_scale,
levels, thresh, gridsize, cut, clip, legend, cbar, cbar_ax, cbar_kws, ax, **kwarg
s)
   1697 if p.univariate:
   1699     plot_kws = kwargs.copy()
-> 1701     p.plot_univariate_density(
   1702         multiple=multiple,
   1703         common_norm=common_norm,
   1704         common_grid=common_grid,
   1705         fill=fill,
   1706         color=color,
   1707         legend=legend,
   1708         warn_singular=warn_singular,
   1709         estimate_kws=estimate_kws,
   1710         **plot_kws,
   1711     )
   1713 else:
   1715     p.plot_bivariate_density(
   1716         common_norm=common_norm,
   1717         fill=fill,
   (...)   1727         **kwargs,
   1728     )

File ~\anaconda3\Lib\site-packages\seaborn\distributions.py:991, in _Distribution
Plotter.plot_univariate_density(self, multiple, common_norm, common_grid, warn_si
ngular, fill, color, legend, estimate_kws, **plot_kws)
    988     artist = ax.fill_between(support, fill_from, density, **artist_kws)
    990 else:
--> 991     artist, = ax.plot(support, density, **artist_kws)
    993 artist.sticky_edges.x[:] = sticky_support
    994 artist.sticky_edges.y[:] = sticky_density

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_axes.py:1724, in Axes.plot(se
lf, scalex, scaley, data, *args, **kwargs)
   1481 """
   1482 Plot y versus x as lines and/or markers.
   1483
   (...)   1721 (``'green'``) or hex strings (``'#008000'``).
   1722 """
   1723 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1724 lines = [*self._get_lines(self, *args, data=data, **kwargs)]
   1725 for line in lines:
   1726     self.add_line(line)

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:303, in _process_plot
_var_args.__call__(self, axes, data, *args, **kwargs)
    301     this += args[0],
    302     args = args[1:]
--> 303 yield from self._plot_args(
    304     axes, this, kwargs, ambiguous_fmt_datakey=ambiguous_fmt_datakey)

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:539, in _process_plot
_var_args._plot_args(self, axes, tup, kwargs, return_kwargs, ambiguous_fmt_datake
```

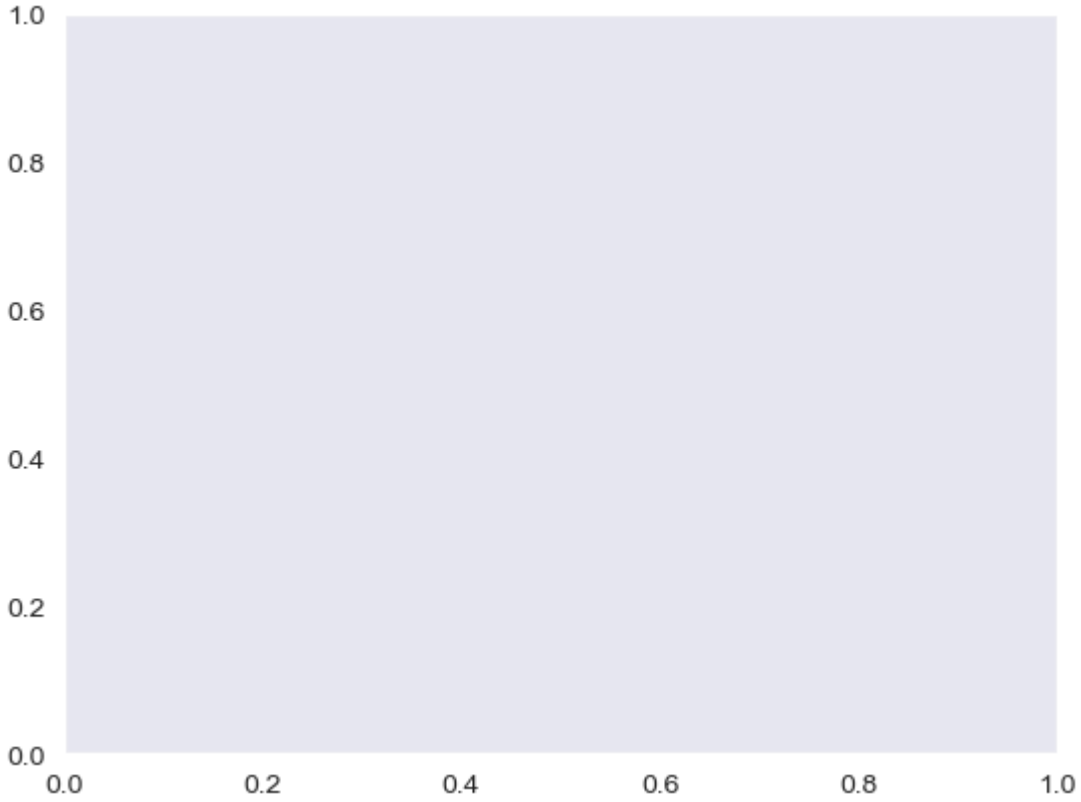
```
y)
    537     return list(result)
    538 else:
--> 539     return [l[0] for l in result]

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:532, in <genexpr>(.0)
    529 else:
    530     labels = [label] * n_datasets
--> 532 result = (make_artist(axes, x[:, j % ncx], y[:, j % ncy], kw,
    533                       {**kwargs,        : label})
    534             for j, label in enumerate(labels))
    536 if return_kwargs:
    537     return list(result)

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:346, in _process_plot
_var_args._makeline(self, axes, x, y, kw, kwargs)
    344 default_dict = self._getdefaults(set(), kw)
    345 self._setdefaults(default_dict, kw)
--> 346 seg = mlines.Line2D(x, y, **kw)
    347 return seg, kw

File ~\anaconda3\Lib\site-packages\matplotlib\lines.py:407, in Line2D.__init__(se
lf, xdata, ydata, linewidth, linestyle, color, gapcolor, marker, markersize, mark
eredgewidth, markeredgecolor, markerfacecolor, markerfacecoloralt, fillstyle, ant
ialiased, dash_capstyle, solid_capstyle, dash_joinstyle, solid_joinstyle, pickrad
ius, drawstyle, markevery, **kwargs)
    403 self.set_markeredgewidth(markeredgewidth)
    405 # update kwargs before updating data to give the caller a
    406 # chance to init axes (and hence unit support)
--> 407 self._internal_update(kwargs)
    408 self.pickradius = pickradius
    409 self.ind_offset = 0

File ~\anaconda3\Lib\site-packages\matplotlib\artist.py:1219, in Artist._internal
_update(self, kwargs)
   1212 def _internal_update(self, kwargs):
   1213     """
   1214     Update artist properties without prenormalizing them, but generating
   1215     errors as if calling `set`.
   1216
   1217     The lack of prenormalization is to maintain backcompatibility.
   1218     """
-> 1219     return self._update_props(
   1220         kwargs,  {cls.__name__}

   1221         {prop_name!r} )

File ~\anaconda3\Lib\site-packages\matplotlib\artist.py:1193, in Artist._update_p
rops(self, props, errfmt)
   1191             func = getattr(self, f"set_{k}", None)
   1192             if not callable(func):
-> 1193                 raise AttributeError(
   1194                     errfmt.format(cls=type(self), prop_name=k))
   1195             ret.append(func(v))
   1196 if ret:

AttributeError: Line2D.set() got an unexpected keyword argument 'cmap'
```



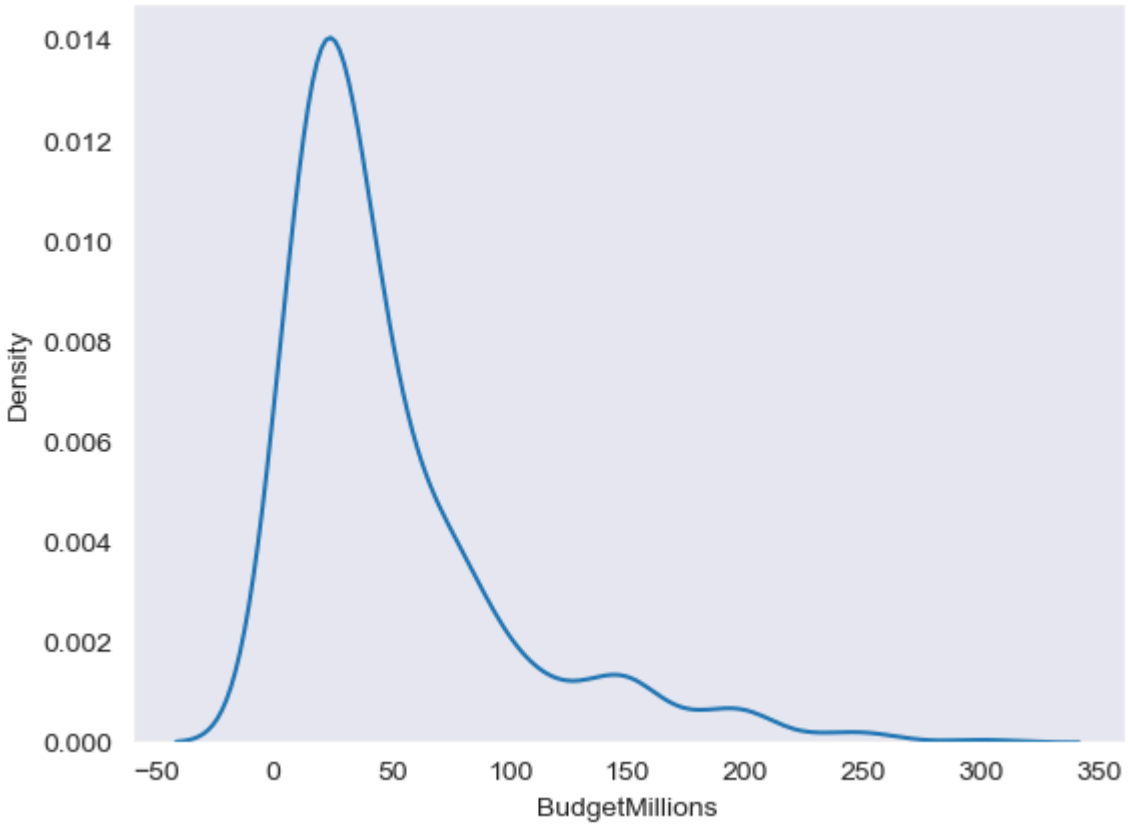
In [84]:
```
sns.set_style('dark')
k1 = sns.kdeplot(movies.BudgetMillions,shade_lowest=False,cmap='Greens_r')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[84], line 2
      1 sns.set_style('dark')
----> 2 k1 = sns.kdeplot(movies.BudgetMillions,shade_lowest=False,cmap=
  )

File ~\anaconda3\Lib\site-packages\seaborn\distributions.py:1701, in kdeplot(dat
a, x, y, hue, weights, palette, hue_order, hue_norm, color, fill, multiple, commo
n_norm, common_grid, cumulative, bw_method, bw_adjust, warn_singular, log_scale,
levels, thresh, gridsize, cut, clip, legend, cbar, cbar_ax, cbar_kws, ax, **kwarg
s)
   1697 if p.univariate:
   1699     plot_kws = kwargs.copy()
-> 1701     p.plot_univariate_density(
   1702         multiple=multiple,
   1703         common_norm=common_norm,
   1704         common_grid=common_grid,
   1705         fill=fill,
   1706         color=color,
   1707         legend=legend,
   1708         warn_singular=warn_singular,
   1709         estimate_kws=estimate_kws,
   1710         **plot_kws,
   1711     )
   1713 else:
   1715     p.plot_bivariate_density(
   1716         common_norm=common_norm,
   1717         fill=fill,
   (...)   1727         **kwargs,
   1728     )

File ~\anaconda3\Lib\site-packages\seaborn\distributions.py:991, in _Distribution
Plotter.plot_univariate_density(self, multiple, common_norm, common_grid, warn_si
ngular, fill, color, legend, estimate_kws, **plot_kws)
    988     artist = ax.fill_between(support, fill_from, density, **artist_kws)
    990 else:
--> 991     artist, = ax.plot(support, density, **artist_kws)
    993 artist.sticky_edges.x[:] = sticky_support
    994 artist.sticky_edges.y[:] = sticky_density

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_axes.py:1724, in Axes.plot(se
lf, scalex, scaley, data, *args, **kwargs)
   1481 """
   1482 Plot y versus x as lines and/or markers.
   1483
   (...)   1721 (``'green'``) or hex strings (``'#008000'``).
   1722 """
   1723 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1724 lines = [*self._get_lines(self, *args, data=data, **kwargs)]
   1725 for line in lines:
   1726     self.add_line(line)

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:303, in _process_plot
_var_args.__call__(self, axes, data, *args, **kwargs)
    301     this += args[0],
    302     args = args[1:]
--> 303 yield from self._plot_args(
    304     axes, this, kwargs, ambiguous_fmt_datakey=ambiguous_fmt_datakey)
```

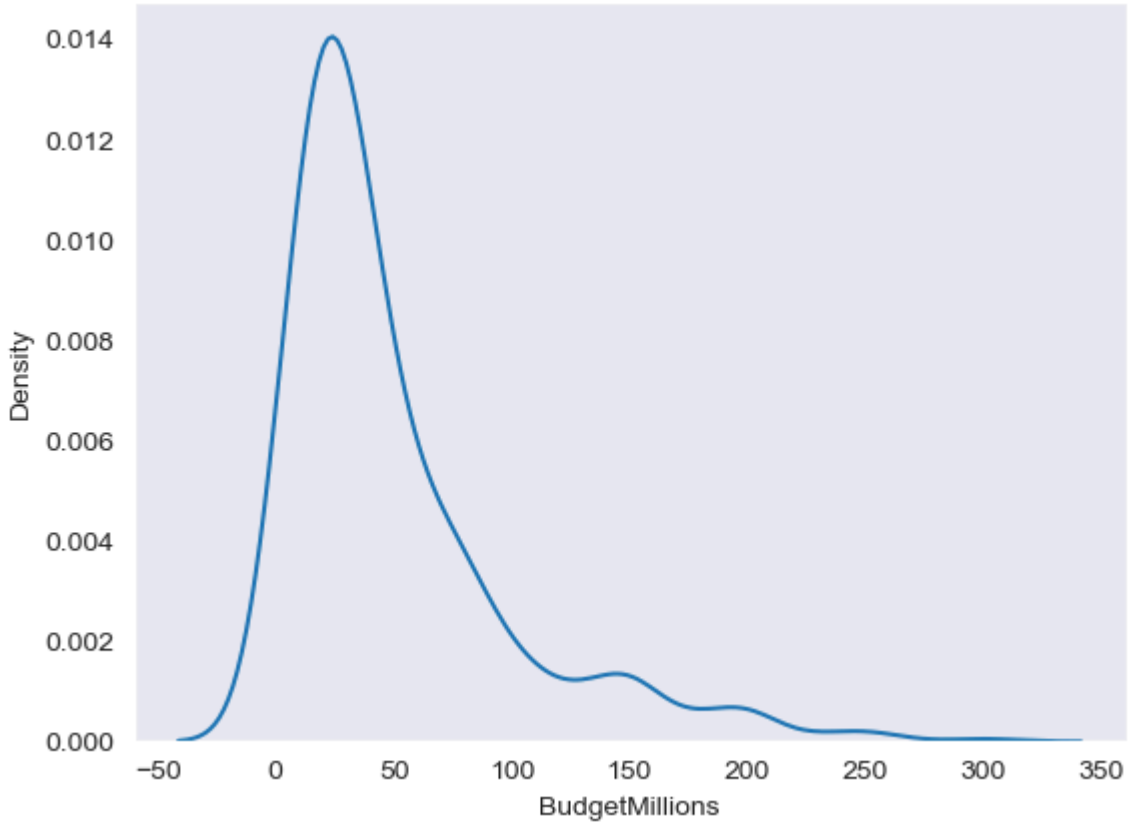
```
File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:539, in _process_plot
_var_args._plot_args(self, axes, tup, kwargs, return_kwargs, ambiguous_fmt_datake
y)
    537     return list(result)
    538 else:
--> 539     return [l[0] for l in result]

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:532, in <genexpr>(.0)
    529 else:
    530     labels = [label] * n_datasets
--> 532 result = (make_artist(axes, x[:, j % ncx], y[:, j % ncy], kw,
    533                       {**kwargs,        : label})
    534             for j, label in enumerate(labels))
    536 if return_kwargs:
    537     return list(result)

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:346, in _process_plot
_var_args._makeline(self, axes, x, y, kw, kwargs)
    344 default_dict = self._getdefaults(set(), kw)
    345 self._setdefaults(default_dict, kw)
--> 346 seg = mlines.Line2D(x, y, **kw)
    347 return seg, kw

File ~\anaconda3\Lib\site-packages\matplotlib\lines.py:407, in Line2D.__init__(se
lf, xdata, ydata, linewidth, linestyle, color, gapcolor, marker, markersize, mark
eredgewidth, markeredgecolor, markerfacecolor, markerfacecoloralt, fillstyle, ant
ialiased, dash_capstyle, solid_capstyle, dash_joinstyle, solid_joinstyle, pickrad
ius, drawstyle, markevery, **kwargs)
    403 self.set_markeredgewidth(markeredgewidth)
    405 # update kwargs before updating data to give the caller a
    406 # chance to init axes (and hence unit support)
--> 407 self._internal_update(kwargs)
    408 self.pickradius = pickradius
    409 self.ind_offset = 0

File ~\anaconda3\Lib\site-packages\matplotlib\artist.py:1219, in Artist._internal
_update(self, kwargs)
   1212 def _internal_update(self, kwargs):
   1213     """
   1214     Update artist properties without prenormalizing them, but generating
   1215     errors as if calling `set`.
   1216
   1217     The lack of prenormalization is to maintain backcompatibility.
   1218     """
-> 1219     return self._update_props(
   1220         kwargs,  {cls.__name__}

   1221         {prop_name!r} )

File ~\anaconda3\Lib\site-packages\matplotlib\artist.py:1193, in Artist._update_p
rops(self, props, errfmt)
   1191             func = getattr(self, f"set_{k}", None)
   1192             if not callable(func):
-> 1193                 raise AttributeError(
   1194                     errfmt.format(cls=type(self), prop_name=k))
   1195             ret.append(func(v))
   1196 if ret:

AttributeError: Line2D.set() got an unexpected keyword argument 'cmap'
```



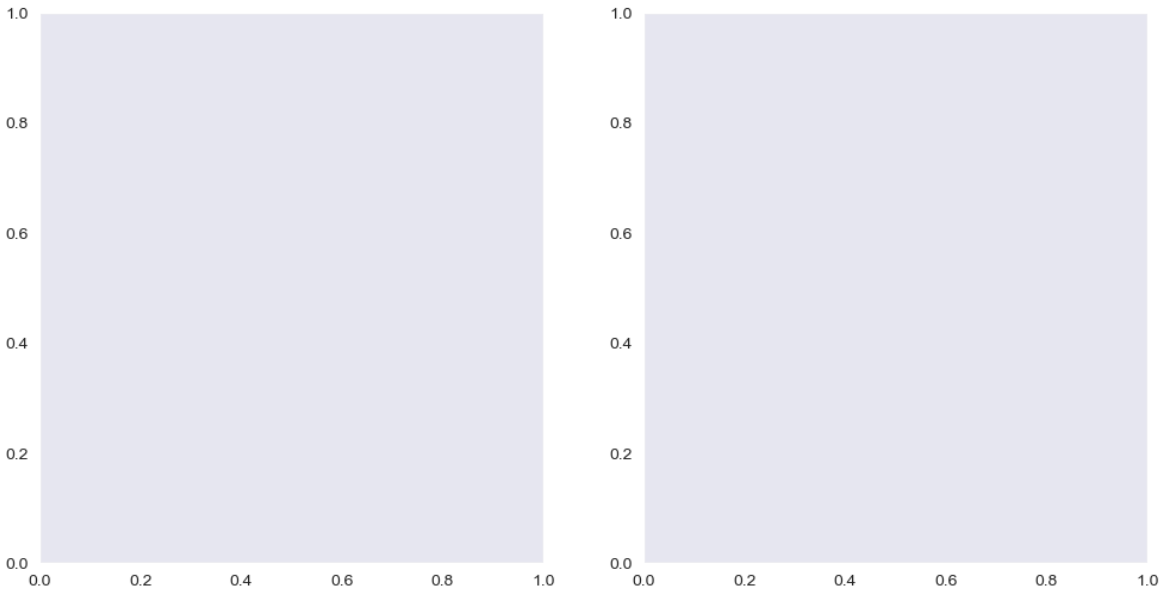
In [86]:
```
sns.set_style('dark')
k1 = sns.kdeplot(movies.BudgetMillions)
```



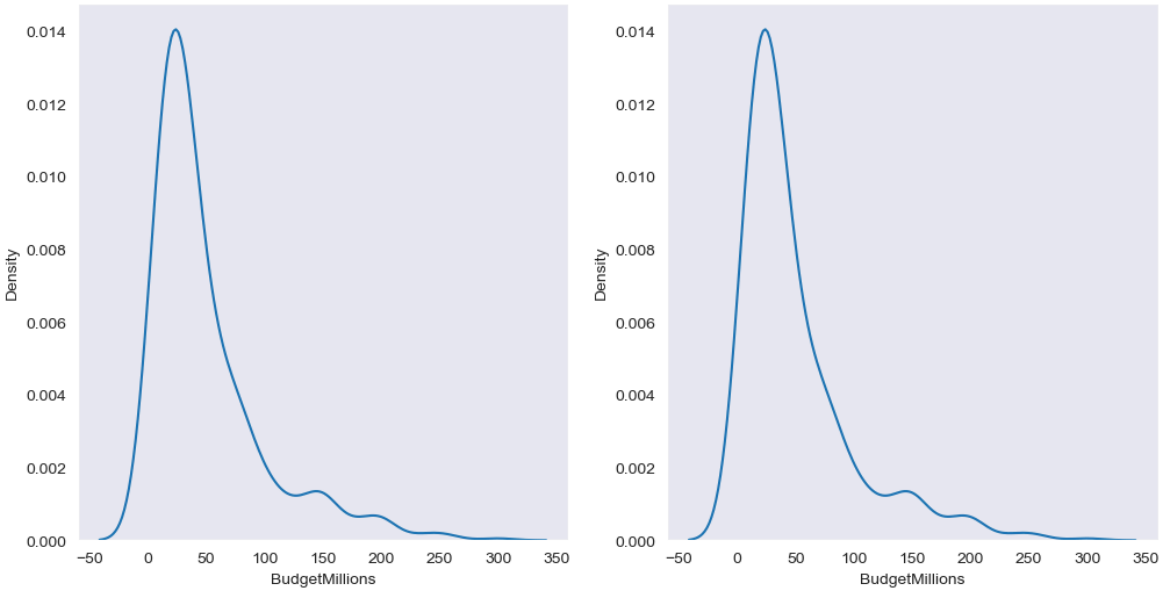
In [87]:
```
k2 = sns.kdeplot(movies.BudgetMillions)
```



In [88]:
```
#subplots

f, ax = plt.subplots(1,2, figsize =(12,6))
#f, ax = plt.subplots(3,3, figsize =(12,6))
```



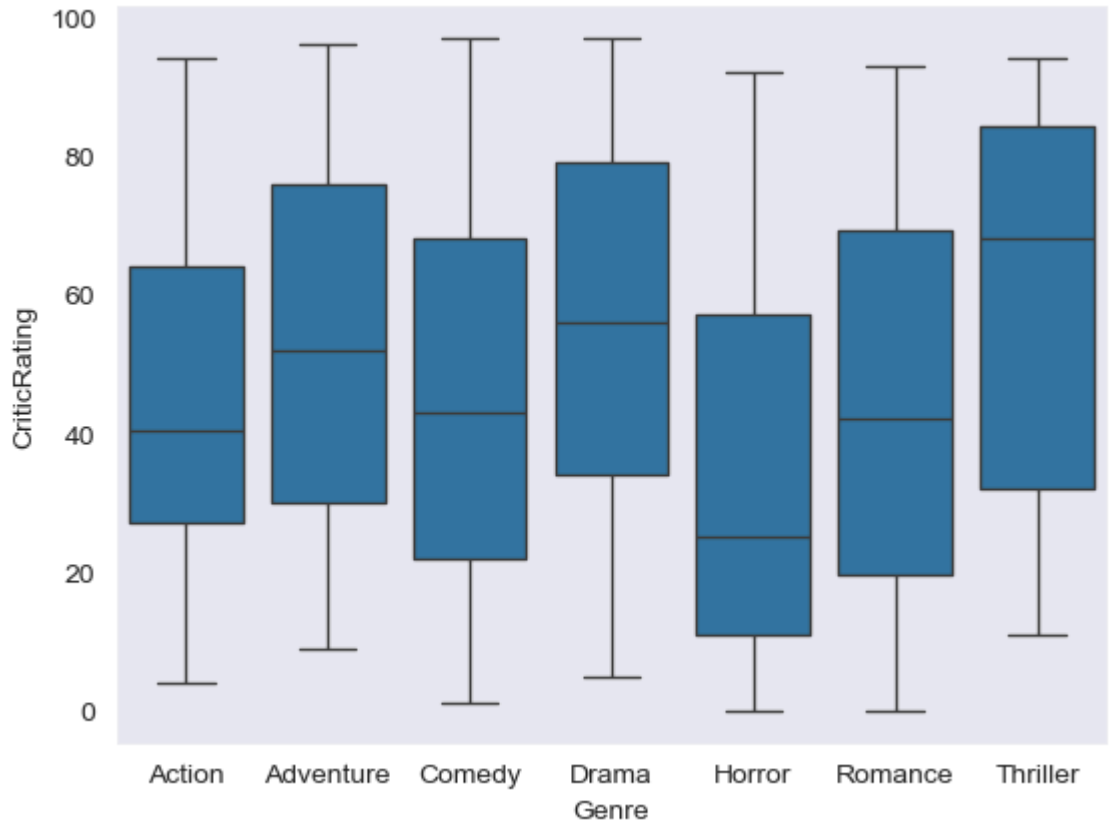
In [89]:
```
f, axes = plt.subplots(1,2, figsize =(12,6))

k1 = sns.kdeplot(movies.BudgetMillions,ax=axes[0])
k2 = sns.kdeplot(movies.BudgetMillions,ax = axes[1])
```



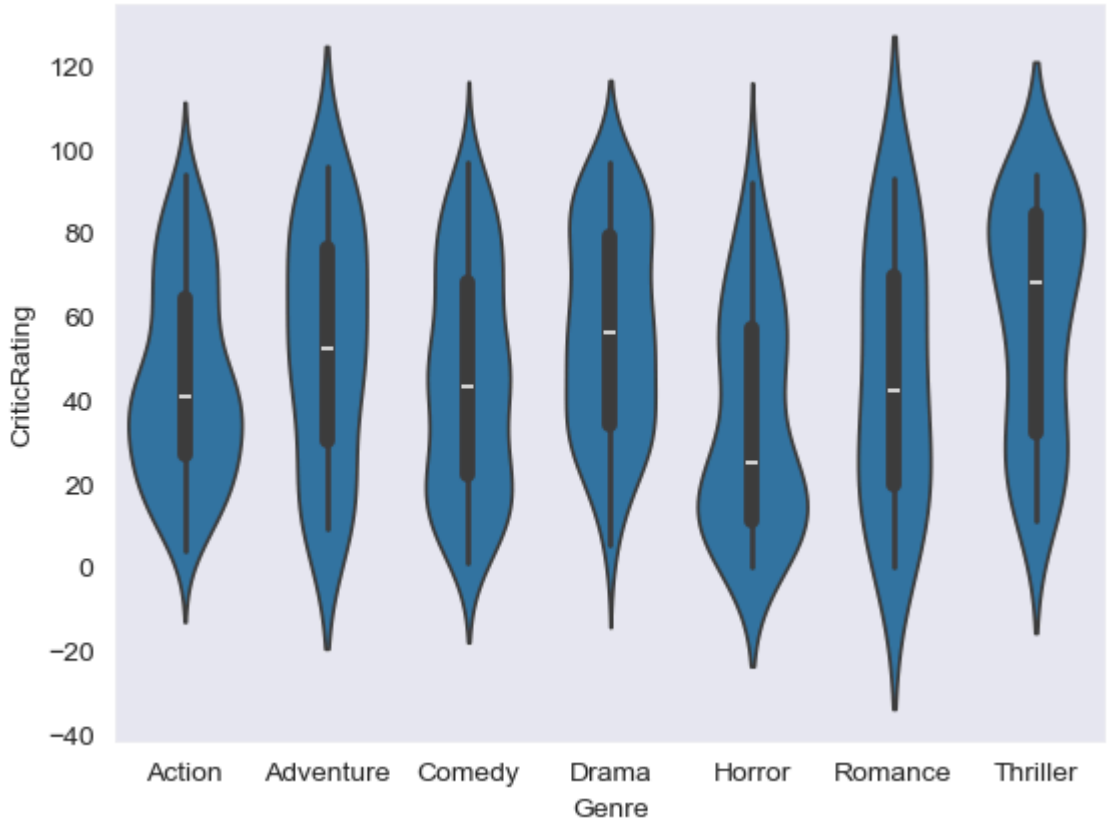
In [90]:
```
axes
```

Out[90]:
```
array([<Axes: xlabel='BudgetMillions', ylabel='Density'>,
       <Axes: xlabel='BudgetMillions', ylabel='Density'>], dtype=object)
```

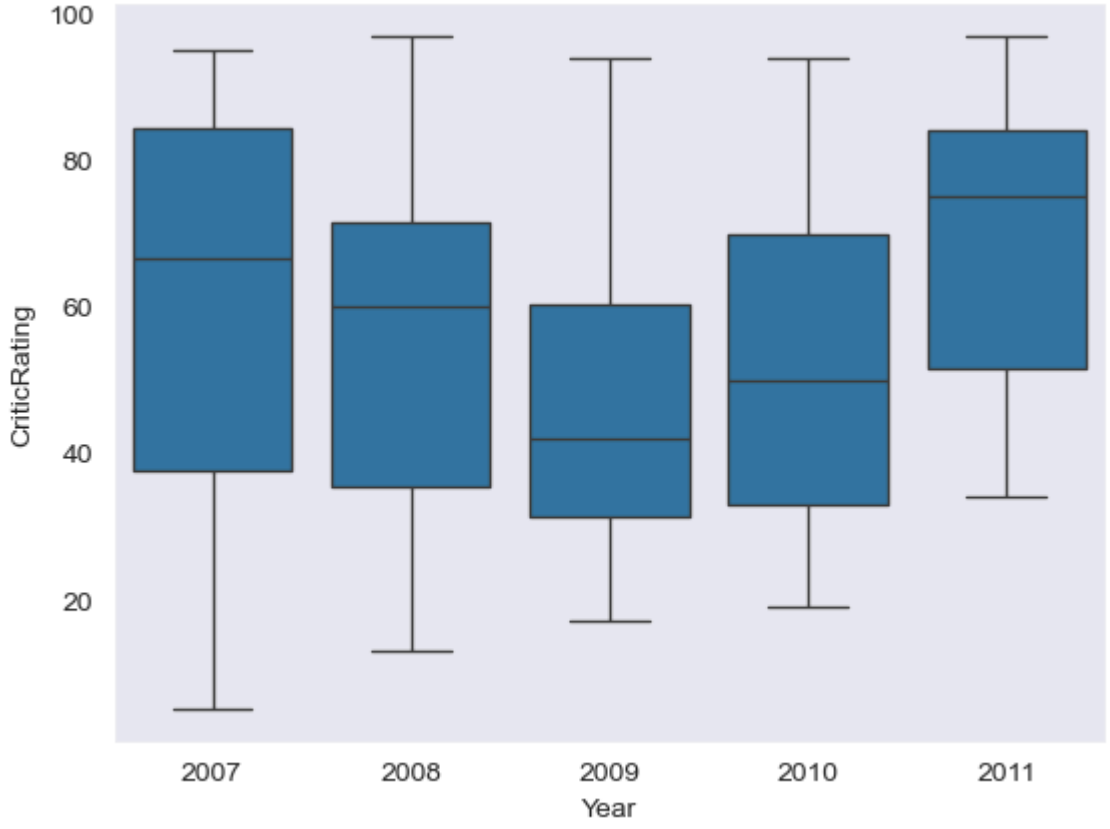
In [91]:
```
#Box plots -

w = sns.boxplot(data=movies, x='Genre', y = 'CriticRating')
```



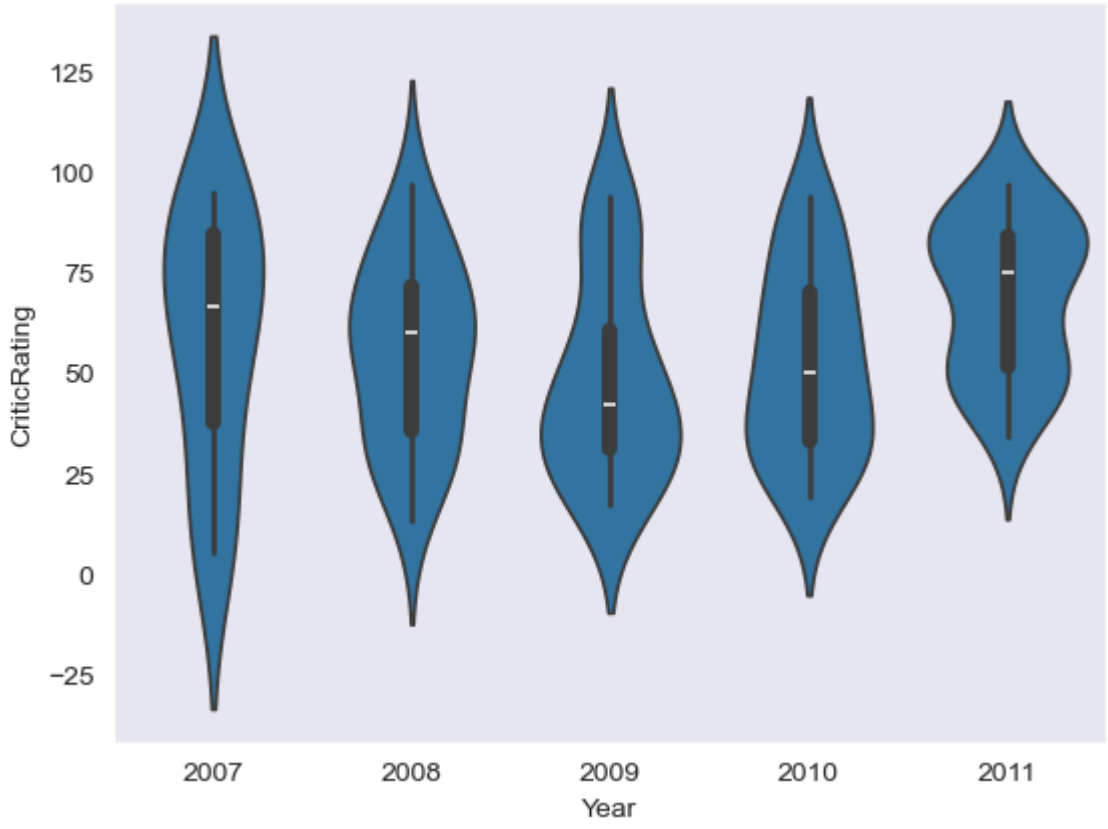
In [92]:
```
#violin plot

z = sns.violinplot(data=movies, x='Genre', y = 'CriticRating')
```



In [93]:
```
w1 = sns.boxplot(data=movies[movies.Genre == 'Drama'], x='Year', y = 'CriticRati
```



In [94]:
```
z = sns.violinplot(data=movies[movies.Genre == 'Drama'], x='Year', y = 'CriticRa
```

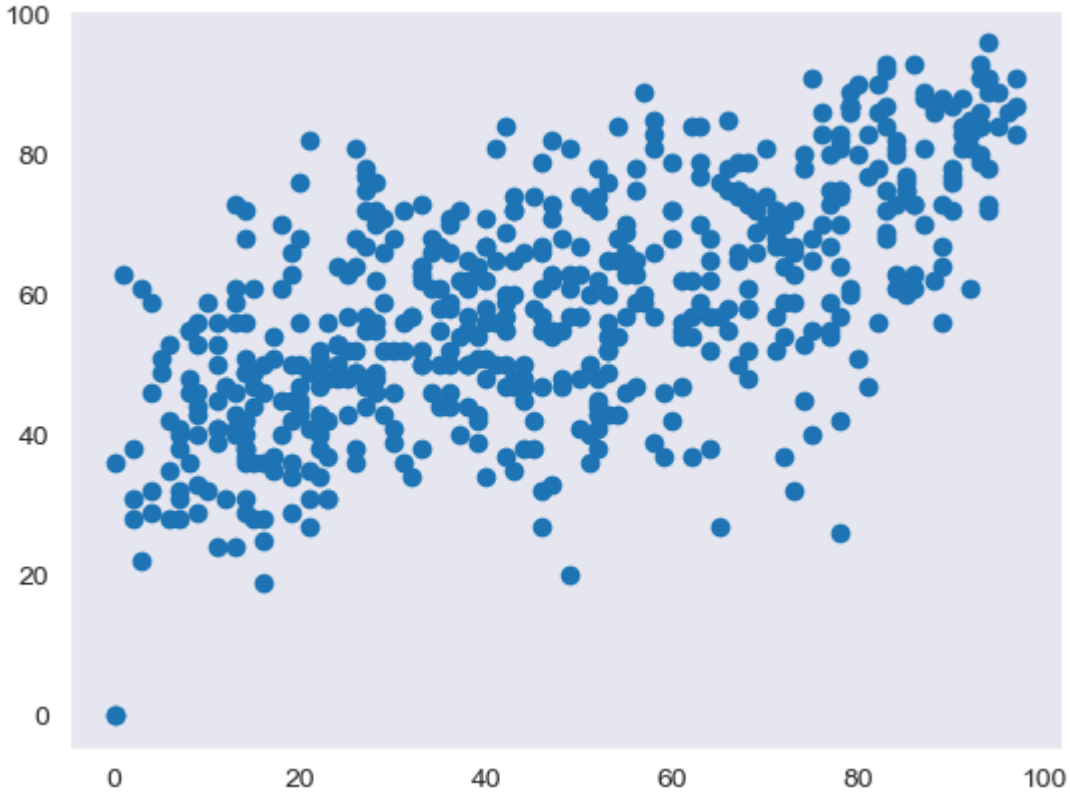


In [96]:
```
# Createing a Facet grid
```

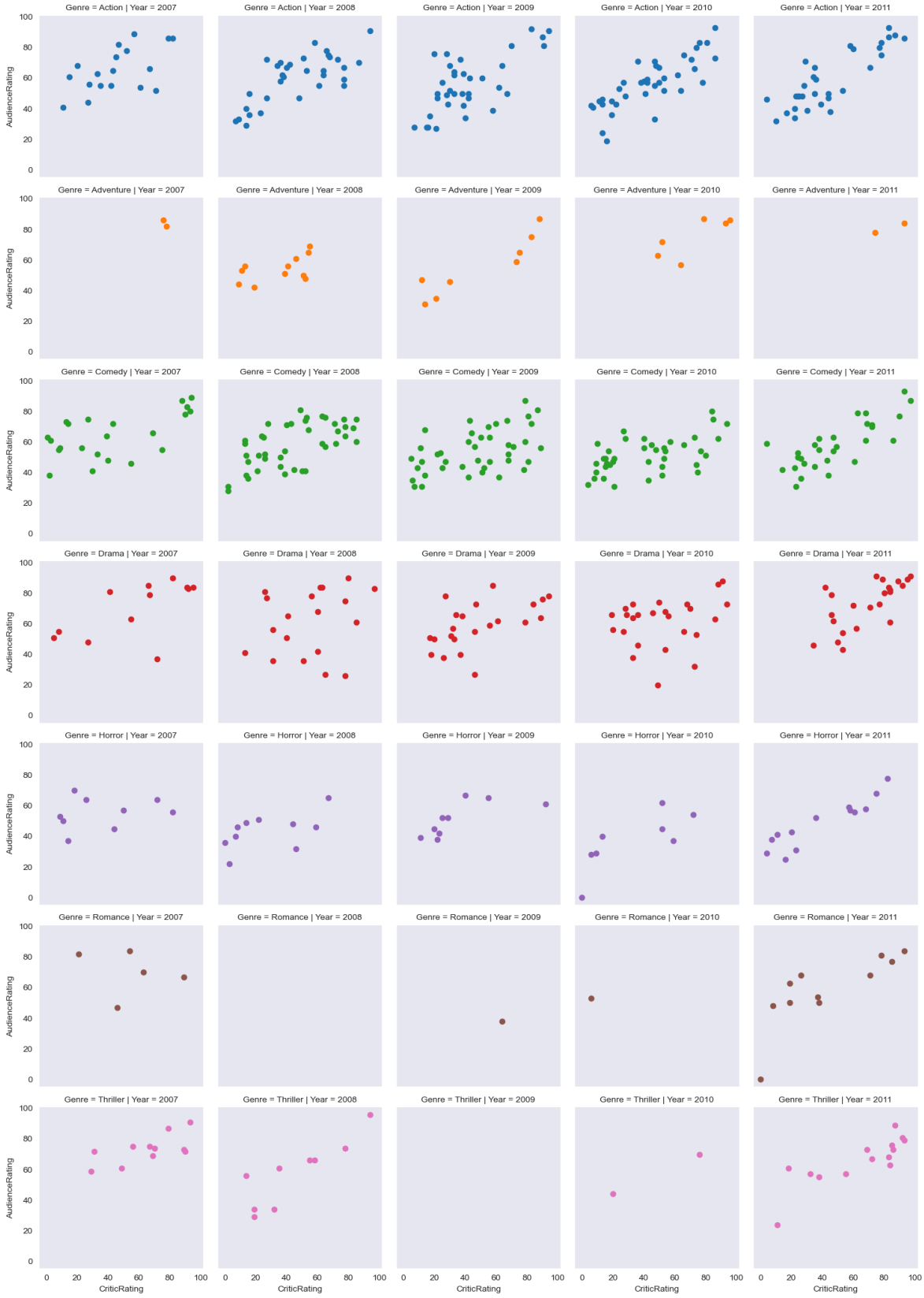
In [97]:
```
g =sns.FacetGrid (movies, row = 'Genre', col = 'Year', hue = 'Genre') #kind of s
```

In [98]:
```
plt.scatter(movies.CriticRating,movies.AudienceRating)
```

Out[98]:
```
<matplotlib.collections.PathCollection at 0x1fe48b8f1d0>
```



In [99]:
```
g =sns.FacetGrid (movies, row = 'Genre', col = 'Year', hue = 'Genre')
g = g.map(plt.scatter, 'CriticRating', 'AudienceRating' ) #scatterplots are mapp
```

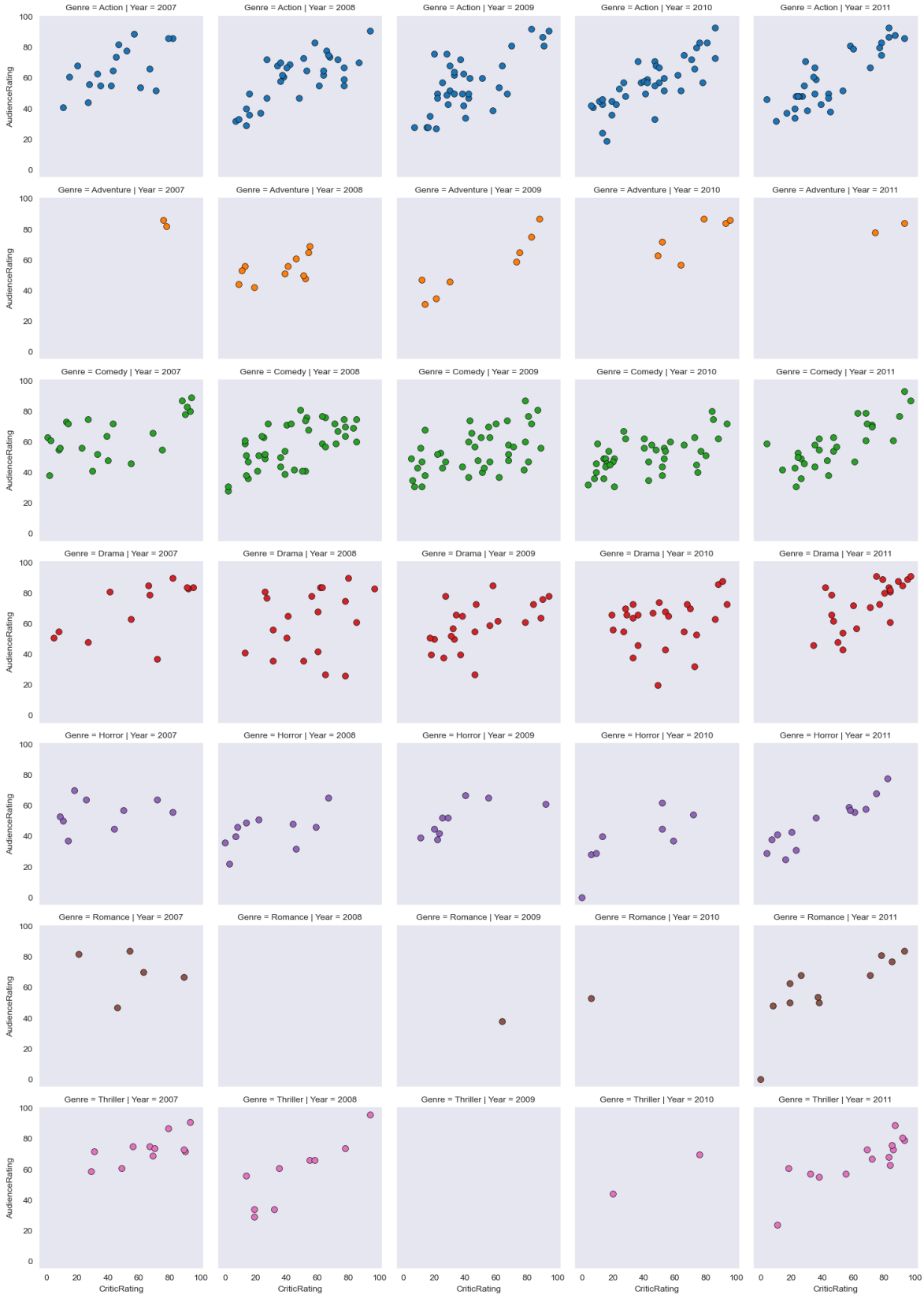


In [100…

```
# you can populated any type of chat.

g =sns.FacetGrid (movies, row = 'Genre', col = 'Year', hue = 'Genre')
g = g.map(plt.hist, 'BudgetMillions') #scatterplots are mapped in facetgrid
```

In [101…

```
#
g =sns.FacetGrid (movies, row = 'Genre', col = 'Year', hue = 'Genre')
kws = dict(s=50, linewidth=0.5,edgecolor='black')
g = g.map(plt.scatter, 'CriticRating', 'AudienceRating',**kws ) #scatterplots ar
```

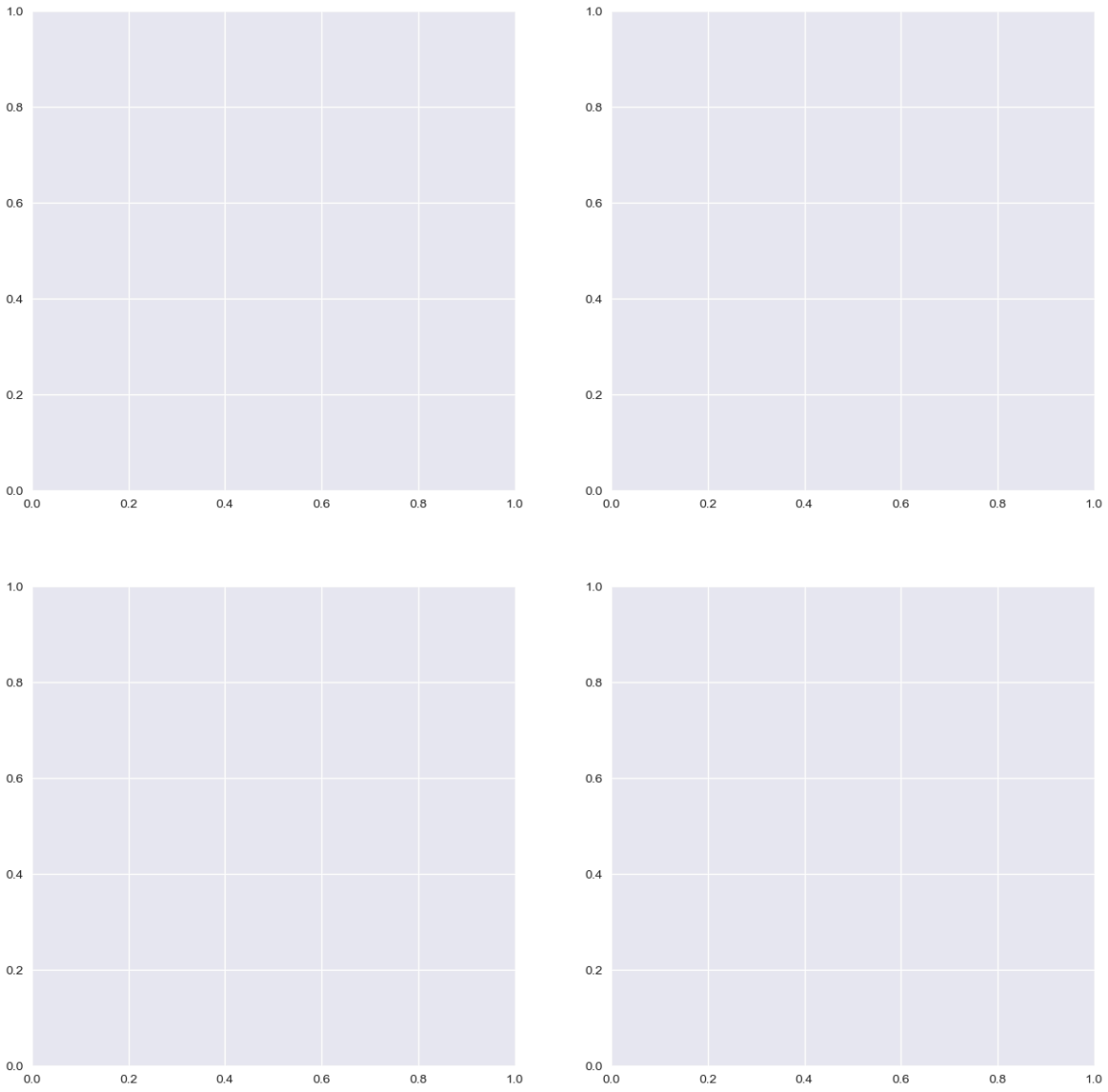


In [102…

```
# python is not vectorize programming language
# Building dashboards (dashboard - combination of chats)

sns.set_style('darkgrid')
f, axes = plt.subplots (2,2, figsize = (15,15))

k1 = sns.kdeplot(movies.BudgetMillions,movies.AudienceRating,ax=axes[0,0])
k2 = sns.kdeplot(movies.BudgetMillions,movies.CriticRating,ax = axes[0,1])

k1.set(xlim=(-20,160))
```

```
k2.set(xlim=(-20,160))

z = sns.violinplot(data=movies[movies.Genre=='Drama'], x='Year', y = 'CriticRati

k4 = sns.kdeplot(movies.CriticRating,movies.AudienceRating,shade = True,shade_lo

k4b = sns.kdeplot(movies.CriticRating, movies.AudienceRating,cmap='Reds',ax = ax

plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[102], line 7
      4 sns.set_style('darkgrid')
      5 f, axes = plt.subplots (2,2, figsize = (15,15))
----> 7 k1 = sns.kdeplot(movies.BudgetMillions,movies.AudienceRating,ax=axes[0,
0])
      8 k2 = sns.kdeplot(movies.BudgetMillions,movies.CriticRating,ax = axes[0,
1])
     10 k1.set(xlim=(-20,160))

TypeError: kdeplot() takes from 0 to 1 positional arguments but 2 positional argu
ments (and 1 keyword-only argument) were given
```



In [ ]:
```
# How can you style your dashboard  using different color map
```

```
# python is not vectorize programming language
# Building dashboards (dashboard - combination of chats)

sns.set_style('dark',{'axes.facecolor':'black'})
f, axes = plt.subplots (2,2, figsize = (15,15))

#plot [0,0]
k1 = sns.kdeplot(movies.BudgetMillions,movies.AudienceRating, \
                 shade = True, shade_lowest=True,cmp = 'inferno', \
                 ax = axes[0,0])
k1b = sns.kdeplot(movies.BudgetMillions, movies.AudienceRating, \
                 cmap = 'cool',ax = axes[0,0])

#plot [0,1]
k2 = sns.kdeplot(movies.BudgetMillions,movies.CriticRating,\
                 shade=True, shade_lowest=True, cmap='inferno',\
                 ax = axes[0,1])
k2b = sns.kdeplot(movies.BudgetMillions,movies.CriticRating,\
                  cmap = 'cool', ax = axes[0,1])

#plot[1,0]
z = sns.violinplot(data=movies[movies.Genre=='Drama'], \
                   x='Year', y = 'CriticRating', ax=axes[1,0])

#plot[1,1]
k4 = sns.kdeplot(movies.CriticRating,movies.AudienceRating, \
                 shade = True,shade_lowest=False,cmap='Blues_r', \
                 ax=axes[1,1])

k4b = sns.kdeplot(movies.CriticRating, movies.AudienceRating, \
                  cmap='gist_gray_r',ax = axes[1,1])


k1.set(xlim=(-20,160))
k2.set(xlim=(-20,160))

plt.show()
```

Final discussion what we learn so far - 1> category datatype in python 2> jointplots 3> histogram 4> stacked histograms 5> Kde plot 6> subplot 7> violin plots 8> Factet grid 9> Building dashboards

In [ ]:
```
# eda is completed
```