Seaborn

```
# For install seabon for first time
 pip install seaborn
```

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

Data

We'll use some generated data from: http://roycekimmons.com/tools/generated_data

```
df= pd.read_csv("D:\\Study\\Programming\\python\\Python course from
udemy\\Udemy - 2022 Python for Machine Learning & Data Science
Masterclass\\01 - Introduction to Course\\1UNZIP-FOR-NOTEBOOKS-FINAL\\
05-Seaborn\\dm_office_sales.csv")
df.head()
```

```
          division  level of education  training level  work
experience  \
0         printers        some college               2
6
1         printers  associate's degree               2
10
2      peripherals         high school               0
9
3  office supplies  associate's degree               2
5
4  office supplies         high school               1
5

   salary   sales
0   91684  372302
1  119679  495660
2   82045  320453
3   92949  377148
4   71280  312802
```

```
df.info
```

```
<bound method DataFrame.info of               division  level of
education  training level  work experience  \
0             printers        some college               2
6
1             printers  associate's degree               2
```

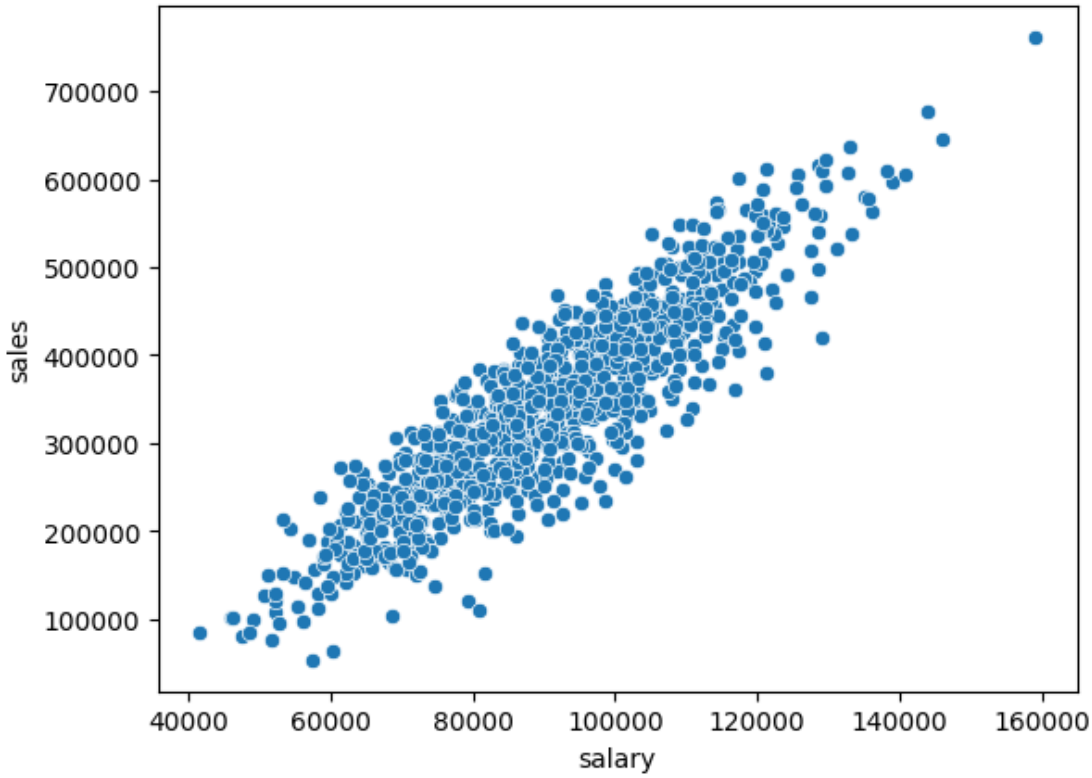
```
10
2          peripherals         high school               0
9
3      office supplies  associate's degree               2
5
4      office supplies         high school               1
5
..                 ...                 ...             ...
...
995  computer hardware  associate's degree               1
1
996  computer software  associate's degree               1
0
997        peripherals  associate's degree               2
8
998        peripherals  associate's degree               2
3
999  computer hardware        some college               0
9

     salary   sales
0     91684  372302
1    119679  495660
2     82045  320453
3     92949  377148
4     71280  312802
..      ...     ...
995   70083  177953
996   68648  103703
997  108354  450011
998   79035  330354
999  108444  364436

[1000 rows x 6 columns]>
```

Scatter Plots

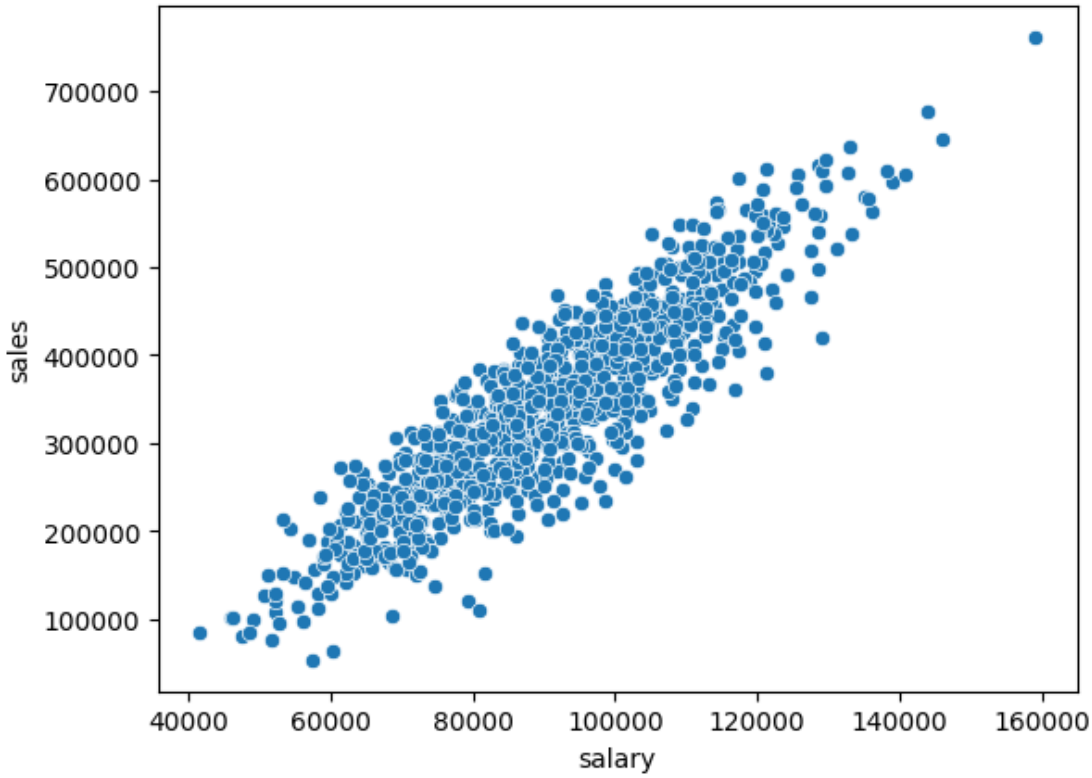
Scatter plots can show how different features are related to one another, the main theme between all relational plot types is they display how features are interconnected to each other. There are many different types of plots that can be used to show this, so let's explore the scatterplot() as well as general seaborn parameters applicable to other plot types.

```
# Here we built a scatterplot between salary and sales
sns.scatterplot(x='salary',y='sales',data=df)
```

```
<AxesSubplot: xlabel='salary', ylabel='sales'>
```



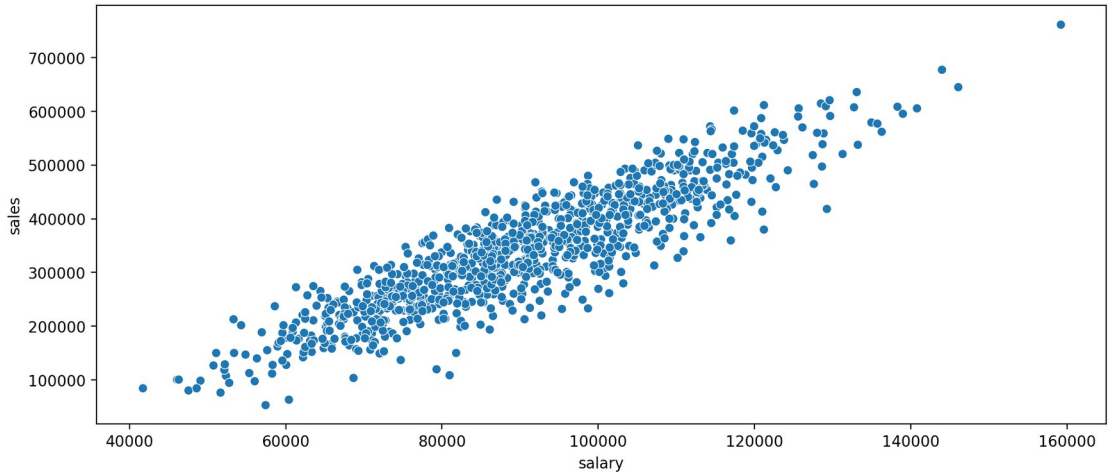

```
# To remove this <AxesSubplot: xlabel='salary', ylabel='sales'> we can
add plt.show() or put ; at the end
sns.scatterplot(x='salary',y='sales',data=df);
#plt.show()
```



Connecting to Figure in Matplotlib

Note how matplotlib is still connected to seaborn underneath (even without importing matplotlib.pyplot), since seaborn itself is directly making a Figure call with matplotlib. We can import matplotlib.pyplot and make calls to directly effect the seaborn figure.

```
from matplotlib import pyplot as plt

plt.figure(figsize=(12,5),dpi=200)
sns.scatterplot(x='salary',y='sales',data=df)
plt.show()
```



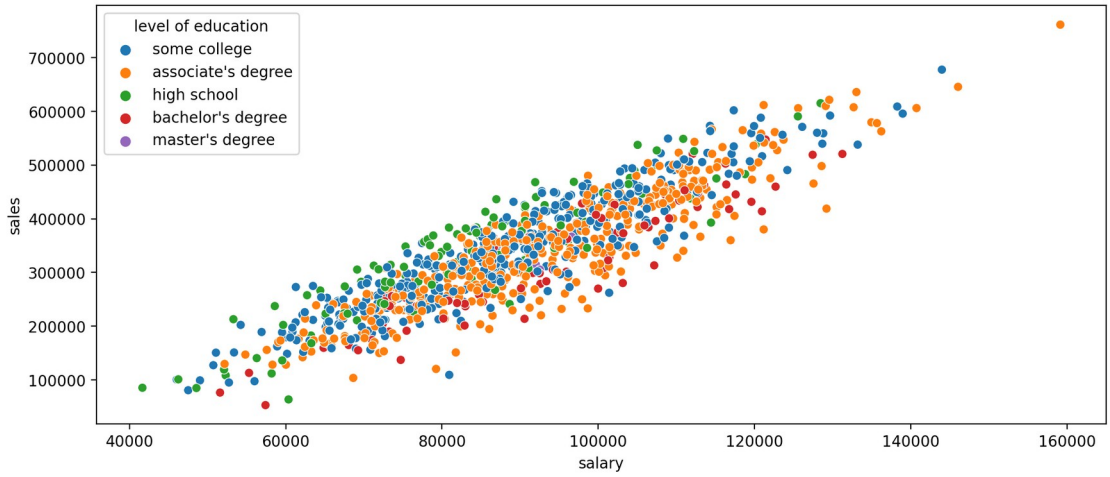
Seaborn Parameters

The hue and palette parameters are commonly available around many plot calls in seaborn.

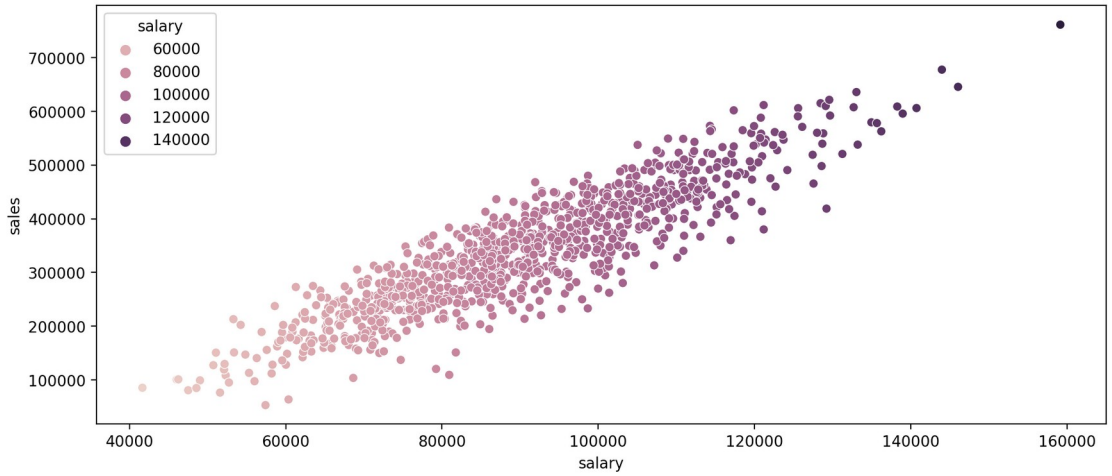
hue

Color points based off a categorical feature in the DataFrame

```
plt.figure(figsize=(12,5),dpi=200)
sns.scatterplot(x='salary',y='sales',data=df,hue='level of education')
plt.show()
```

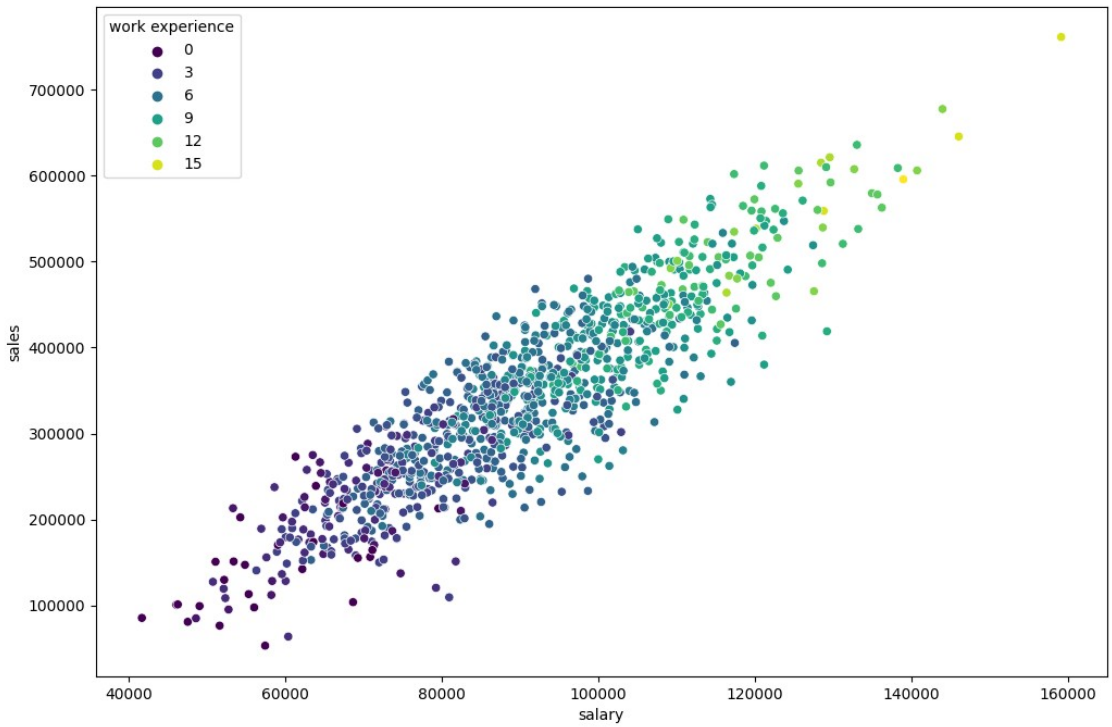



```
# If you provide continous data in hue
plt.figure(figsize=(12,5),dpi=200)
sns.scatterplot(x='salary',y='sales',data=df,hue='salary')
plt.show()
```



Choosing a palette from Matplotlib's cmap:
https://matplotlib.org/tutorials/colors/colormaps.html

```
# Here is palette to change color pattern in hue, for that color code
we can use above link , we will list of all palette
# we give wrong name there we can see name
plt.figure(figsize=(12,8))
sns.scatterplot(x='salary',y='sales',data=df,hue='work
experience',palette='viridis')
plt.show()
```



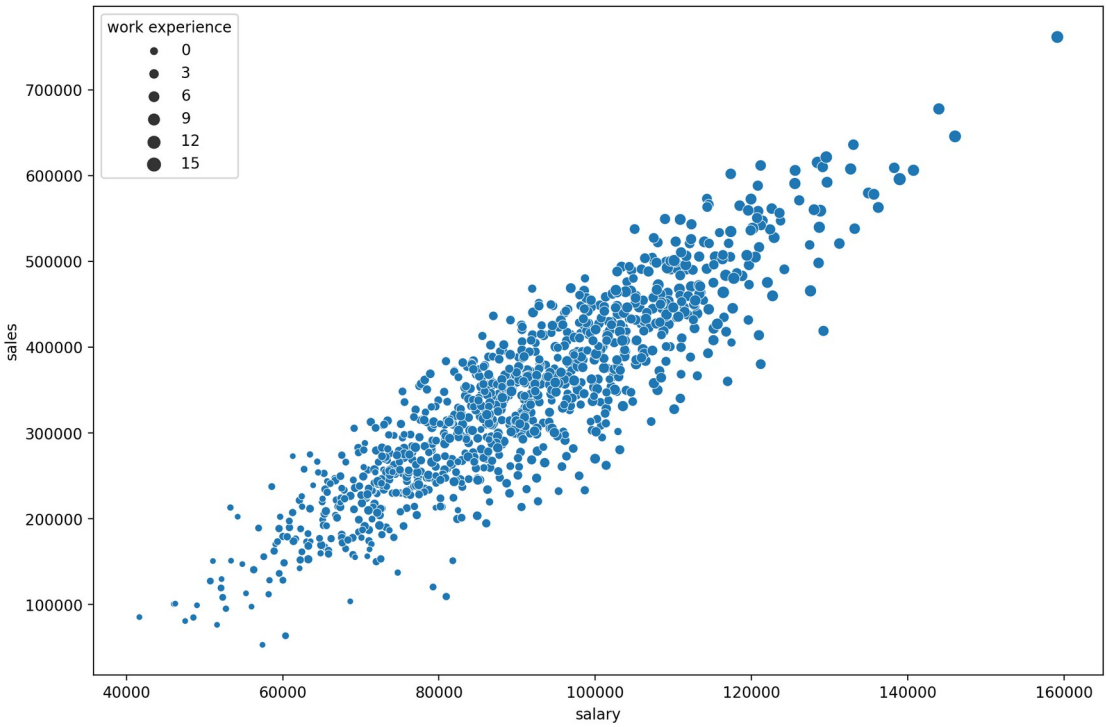
Scatterplot Parameters

These parameters are more specific to the scatterplot() call

size

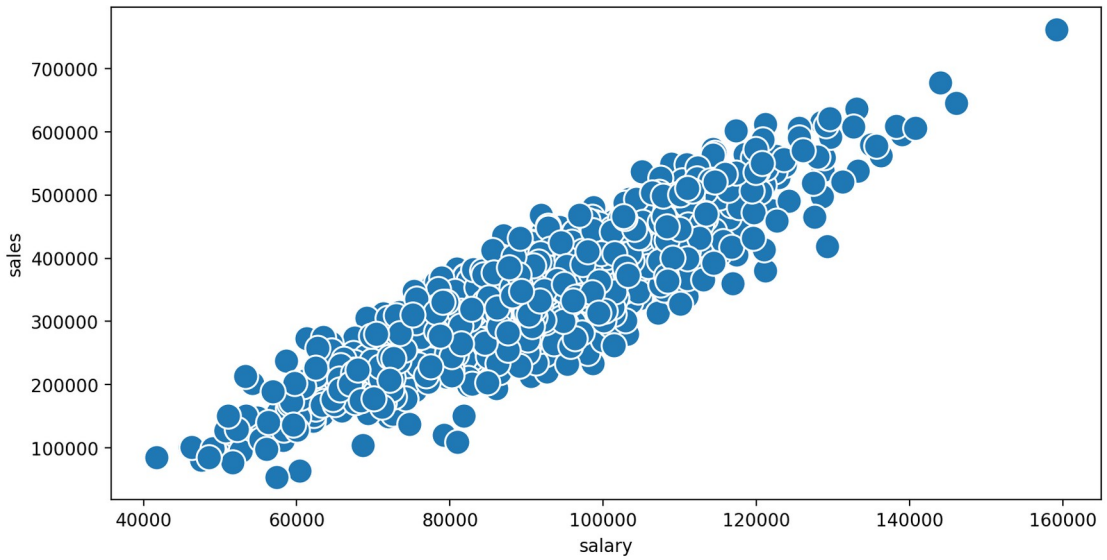
Allows you to size based on another column

```
plt.figure(figsize=(12,8),dpi=200)
sns.scatterplot(x='salary',y='sales',data=df,size='work experience');
```

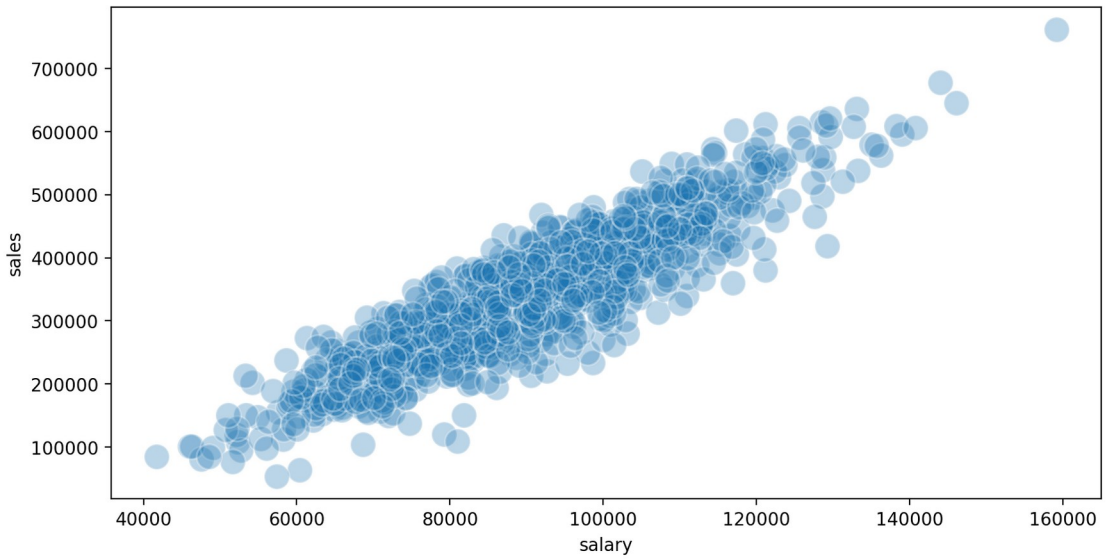


Use *s*= if you want to change the marker size to be some uniform integer value

```
# Here to set the size of marker we can use s=
plt.figure(figsize=(10,5),dpi=200)
sns.scatterplot(x='salary',y='sales',data=df, s=200);
```



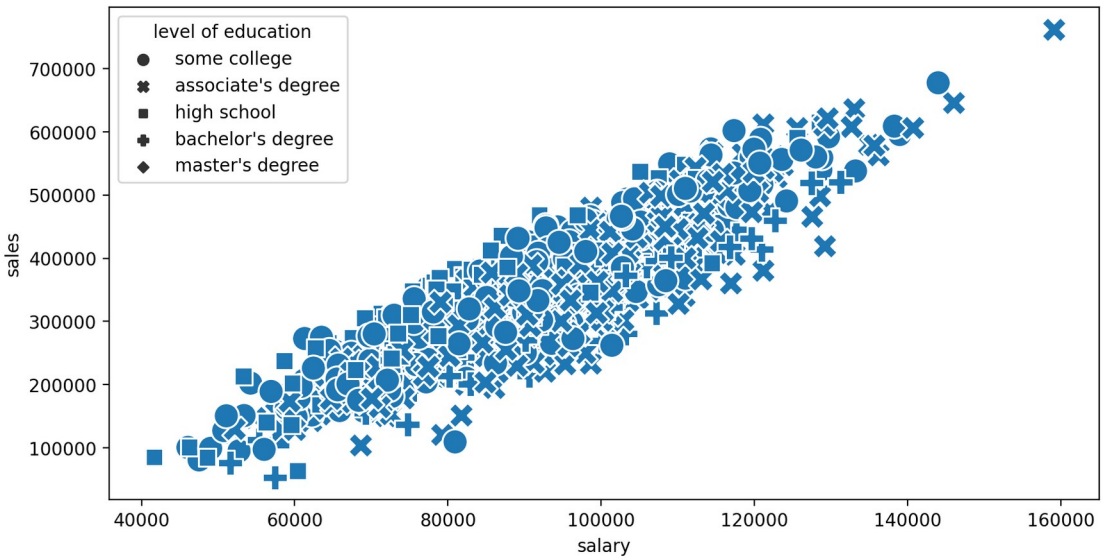

```
# Points we overlap each other we can use alpha= to adjust transpracy
between 0 to 1
# Here to set the size of marker we can use s=
plt.figure(figsize=(10,5),dpi=200)
sns.scatterplot(x='salary',y='sales',data=df, s=200, alpha=0.3);
```



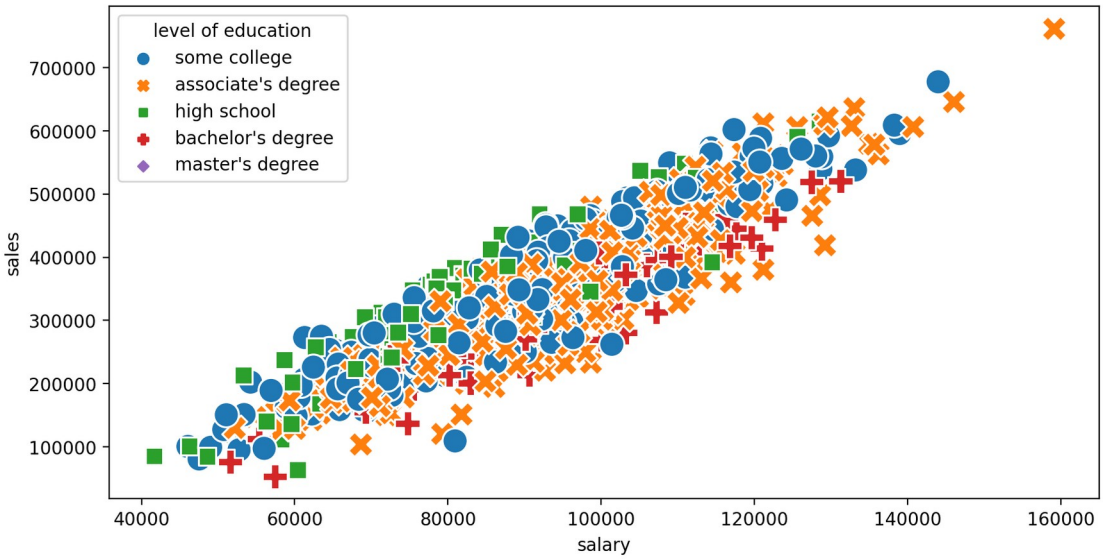
style

Automatically choose styles based on another categorical feature in the dataset. Optionally use the **markers=** parameter to pass a list of marker choices based off matplotlib, for example: ['*','+','o']

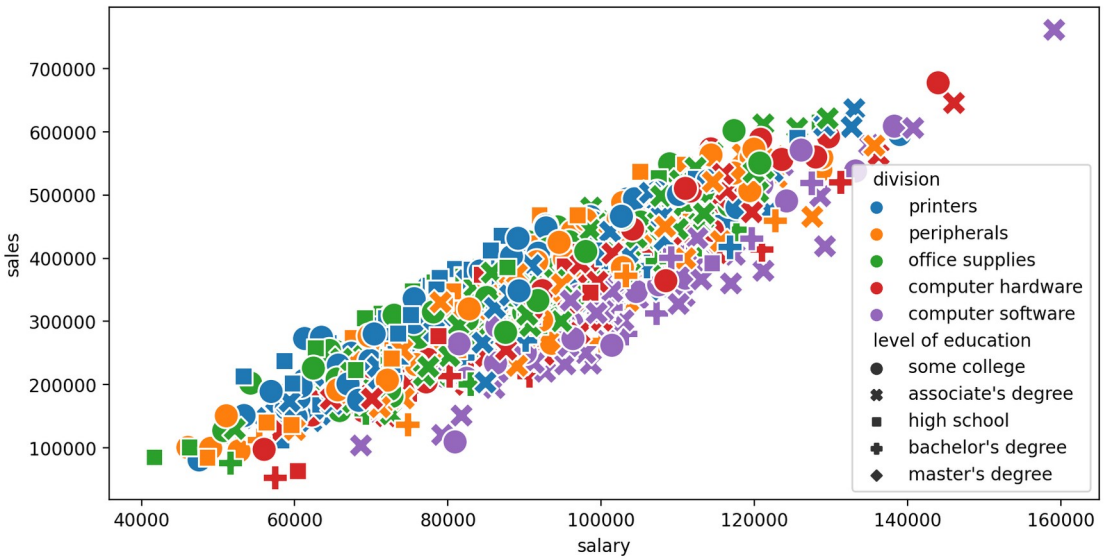
```
# we want to set markers and here s= will change  there size
plt.figure(figsize=(10,5),dpi=200)
sns.scatterplot(x='salary',y='sales',data=df, s=200, style='level of
education');
```




```
# Here we add hue and that showing in different colors
plt.figure(figsize=(10,5),dpi=200)
sns.scatterplot(x='salary',y='sales',data=df, s=200, style='level of
education',hue='level of education');
```

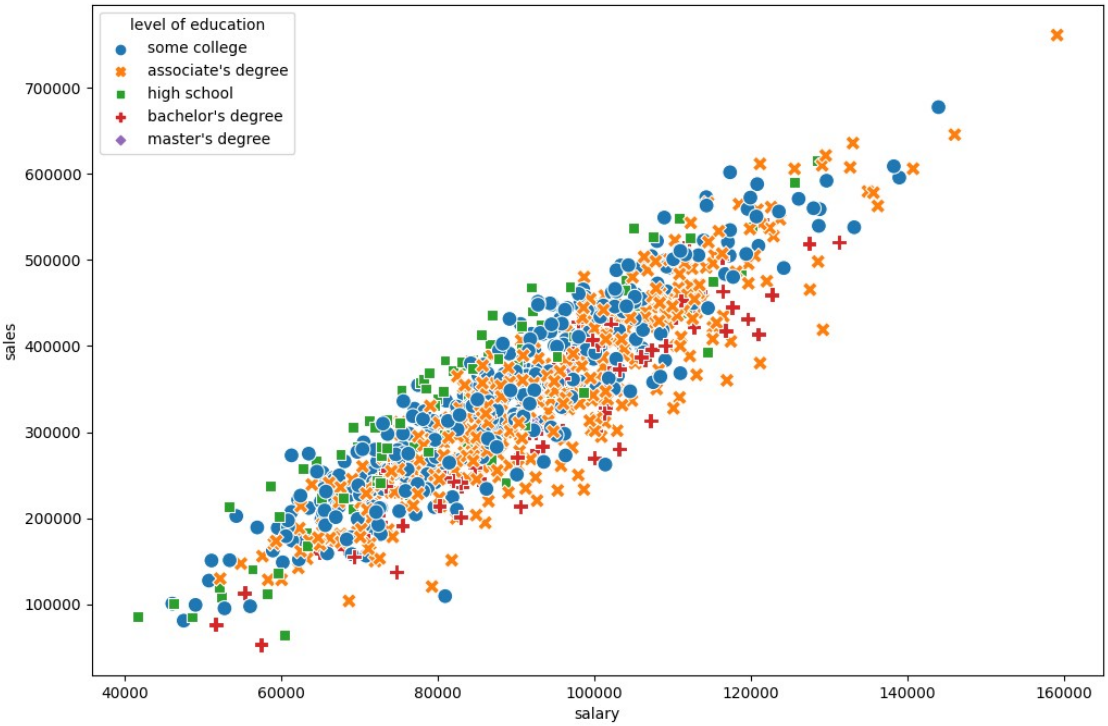



```
# Here we add hue and that showing in different colors , but hue in
different column
plt.figure(figsize=(10,5),dpi=200)
sns.scatterplot(x='salary',y='sales',data=df, s=200, style='level of
education',hue='division');
```



Exporting a Seaborn Figure

```
plt.figure(figsize=(12,8))
sns.scatterplot(x='salary',y='sales',data=df,style='level of education',hue='level of education',s=100)

# Call savefig in the same cell
plt.savefig('example_scatter.jpg')
```



Distributions

There are many ways to display the distributions of a feature. In this notebook we explore 3 related plots for displaying a distribution, the rugplot , the distplot (histogram), and kdeplot.

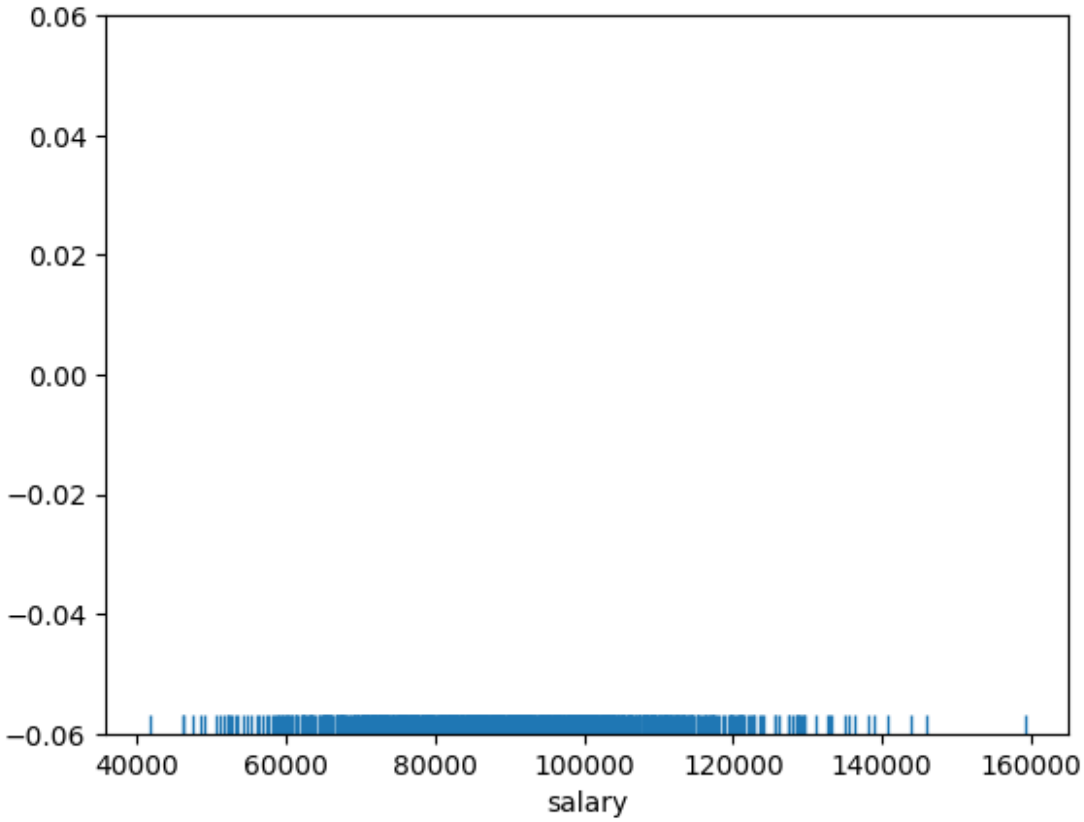
IMPORTANT NOTE!

DO NOT WORRY IF YOUR PLOTS STYLING APPEARS SLIGHTLY DIFFERENT, WE WILL SHOW YOU HOW TO EDIT THE COLOR AND STYLE OF THE PLOTS LATER ON!

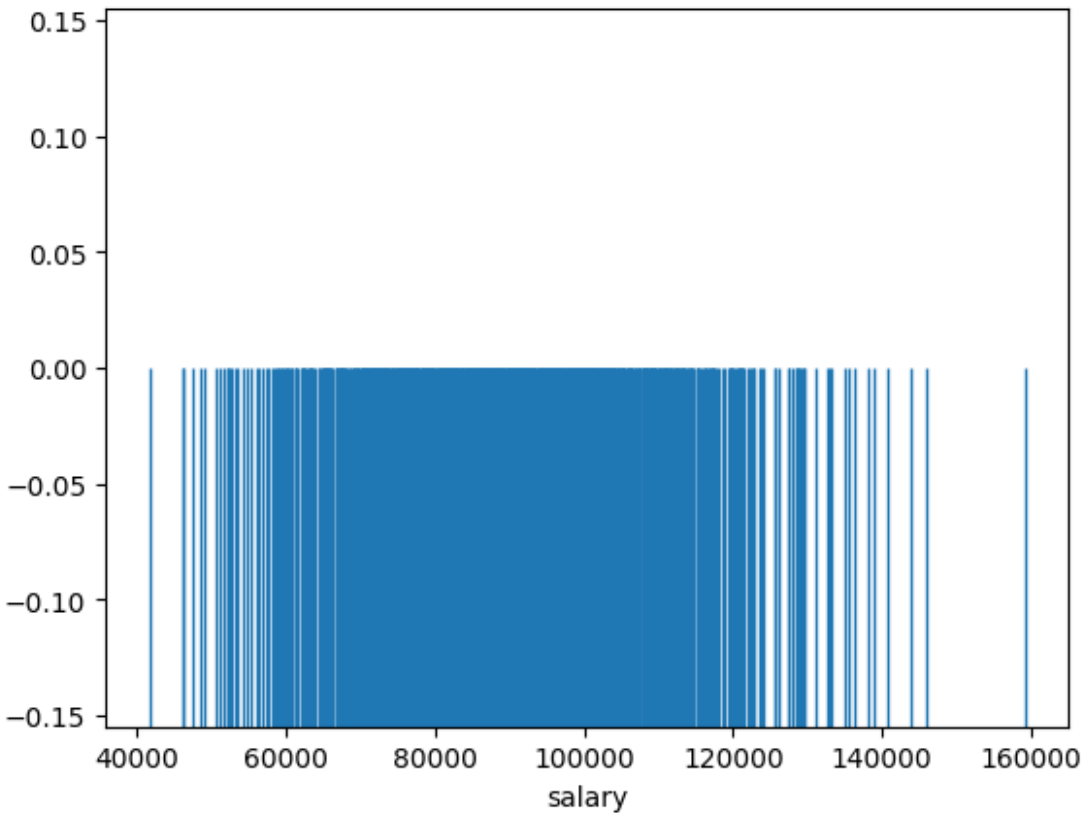
Rugplot

Very simple plot that puts down one mark per data point. This plot needs the single array passed in directly. We won't use it too much since its not very clarifying for large data sets.

```
# The y axis doesn't really represent anything
# X axis is just a stick per data point
sns.rugplot(x='salary',data=df);
```




```
# Here we increase the height
sns.rugplot(x='salary',data=df,height=0.5);
```



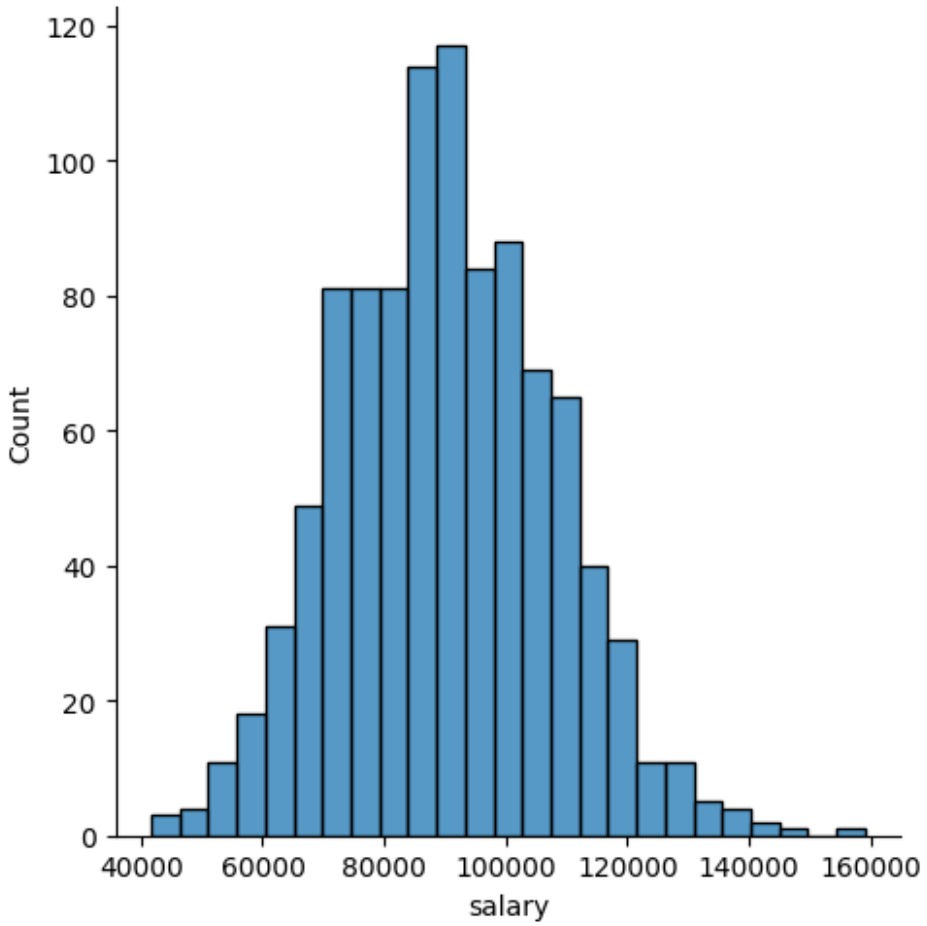
displot() and histplot()

The rugplot itself is not very informative for larger data sets distribution around the mean since so many ticks makes it hard to distinguish one tick from another. Instead we should count the number of tick marks per some segment of the x axis, then construct a histogram from this.

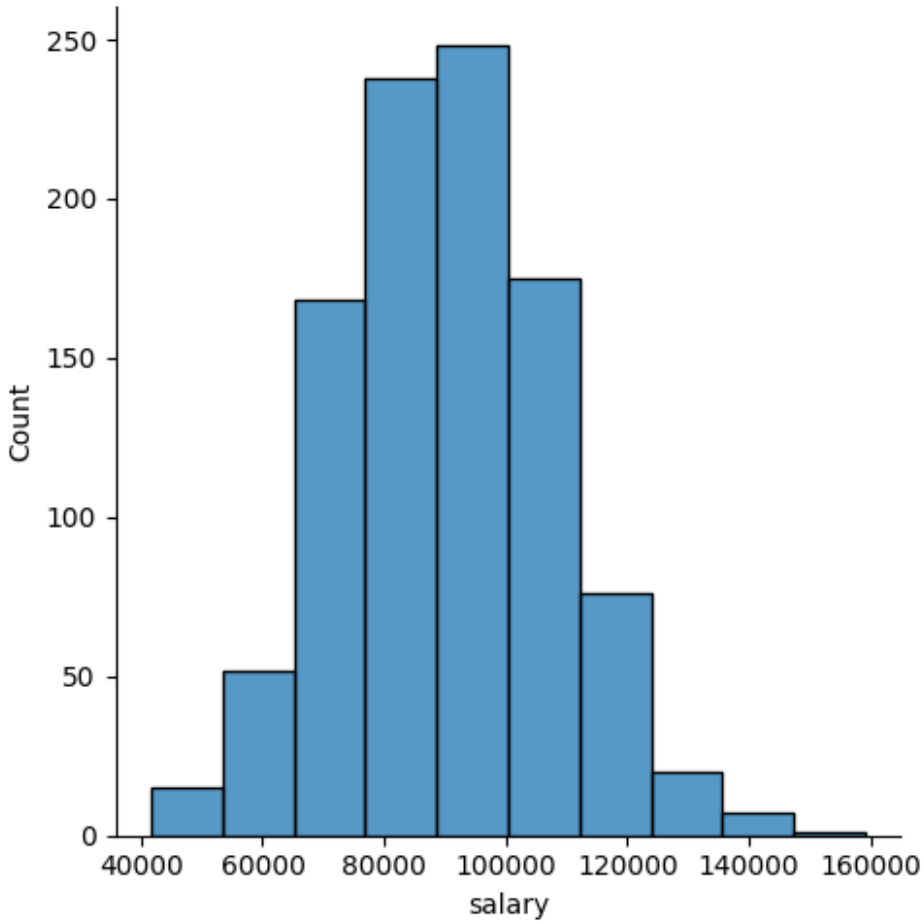
The displot is a plot type that can show you the distribution of a single feature. It is a histogram with the option of adding a "KDE" plot (Kernel Density Estimation) on top of the histogram. Let's explore its use cases and syntax.

Dont use distplot() it going to deprecated in future

```
# Here we created displot, we can pu line also by setting kde=True,
(Kernel Density Estimation (KDE))
sns.displot(x='salary',data=df);
```




```
# Here we add bins too more the number of bin more will be the
precision
sns.displot(x='salary',data=df, bins=10);
```

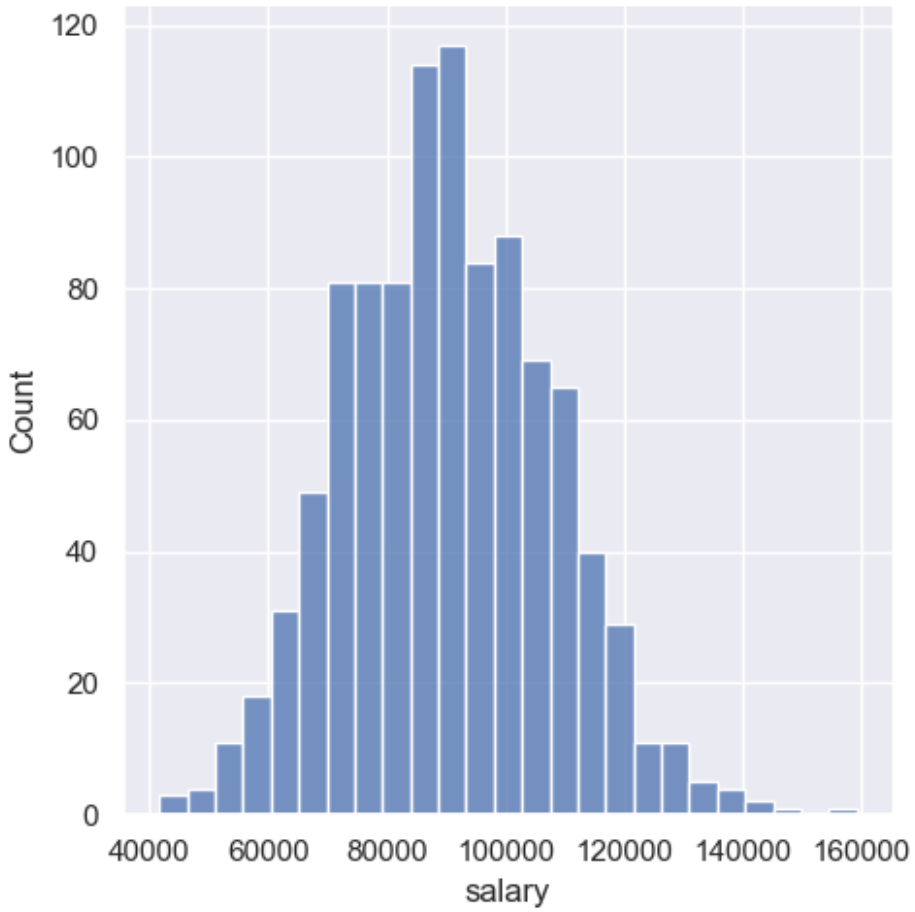


Adding in a grid and styles

You can reset to a different default style: one of {darkgrid, whitegrid, dark, white, ticks}.

In a later notebook and lecture we will cover custom styling in a lot more detail.

```
# Here we can set styles as well
sns.set(style='darkgrid') # we have whitegrid too and default white
and dark, ticks # This set is for upcoming plots
sns.displot(x='salary',data=df);
```

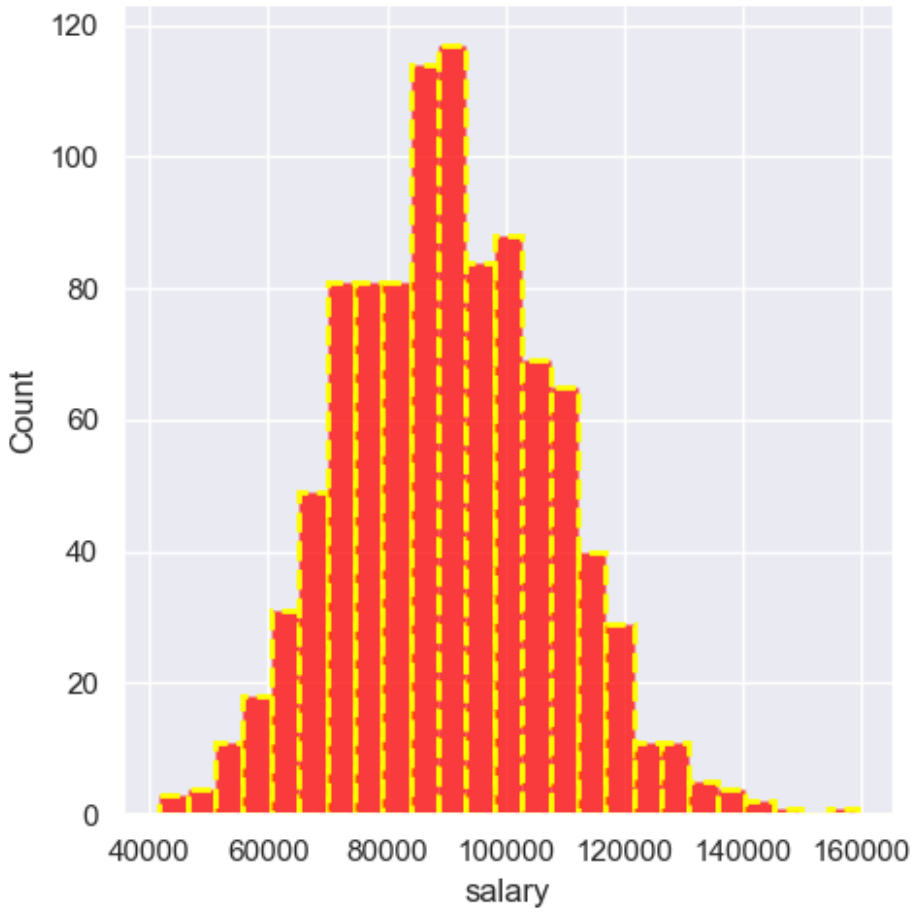


Adding in keywords from matplotlib

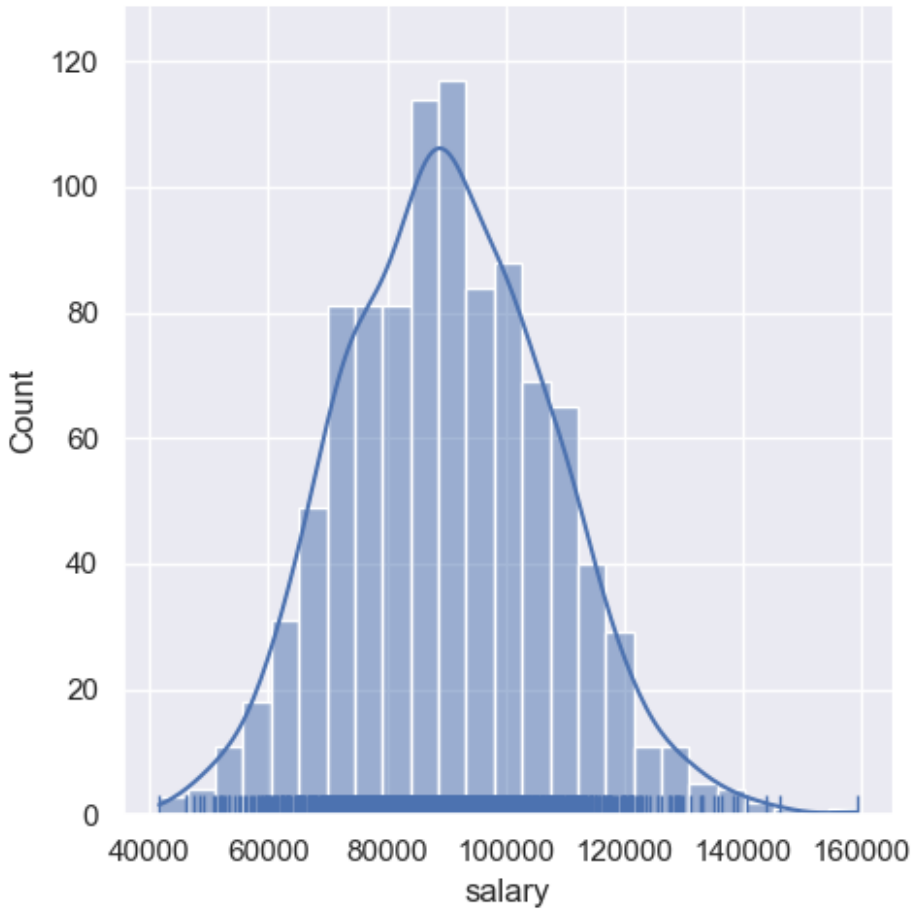
Seaborn plots can accept keyword arguments directly from the matplotlib code that seaborn uses. Keep in mind, not every seaborn plot can accept all matplotlib arguments, but the main styling parameters we've discussed are available.

```
# We set bars color, edgecolor, linewidth and linestyle ls
sns.displot(x='salary',data=df,color='red',edgecolor='yellow',
linewidth=2,ls='--')
```

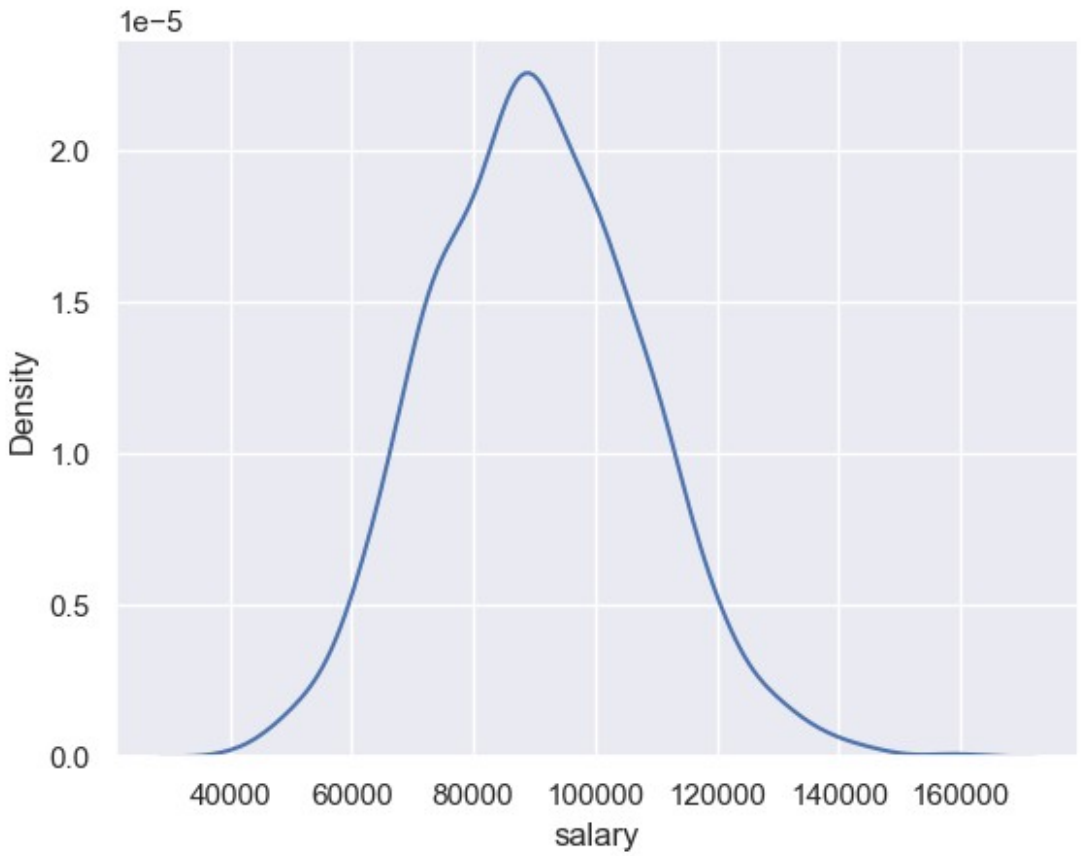
```
<seaborn.axisgrid.FacetGrid at 0x17f961ae5c0>
```




```
# Here we set kde=True and we can see line and we can put rugplot as
well by rug=True
sns.displot(x='salary',data=df,kde=True,rug=True);
```



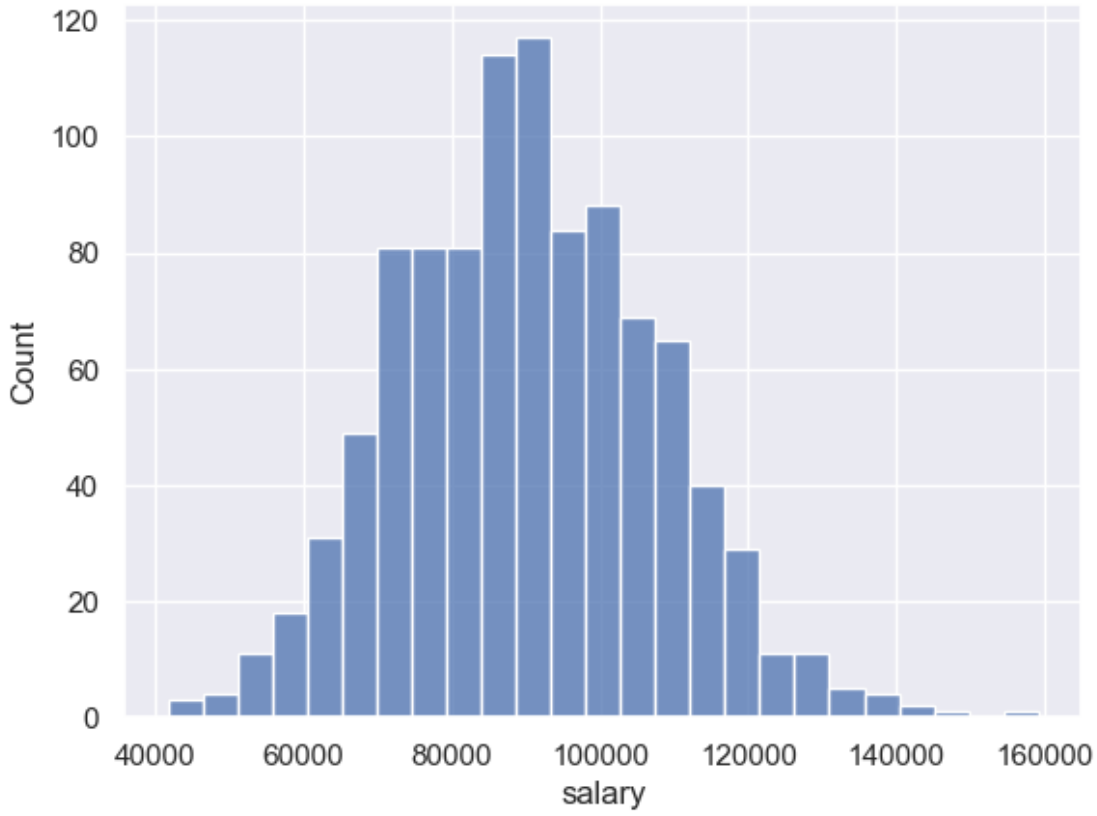

```
# As rugplot rug=True same way there kdeplot kde=True too
sns.kdeplot(x='salary',data=df);
```



Focusing on the Histogram

```
# It same as displot better to use displot
sns.histplot(x='salary',data=df)
```

```
<AxesSubplot: xlabel='salary', ylabel='Count'>
```



The Kernel Density Estimation Plot

Note: Review the video for full detailed explanation.

The KDE plot maps an estimate of a probability *density* function of a random variable. Kernel density estimation is a fundamental data smoothing problem where inferences about the population are made, based on a finite data sample.

Let's build out a simple example:

Task

We have to create random age of 200 sample between 0 to 100 with seed(42) and then built rugplot and displots to see distribution

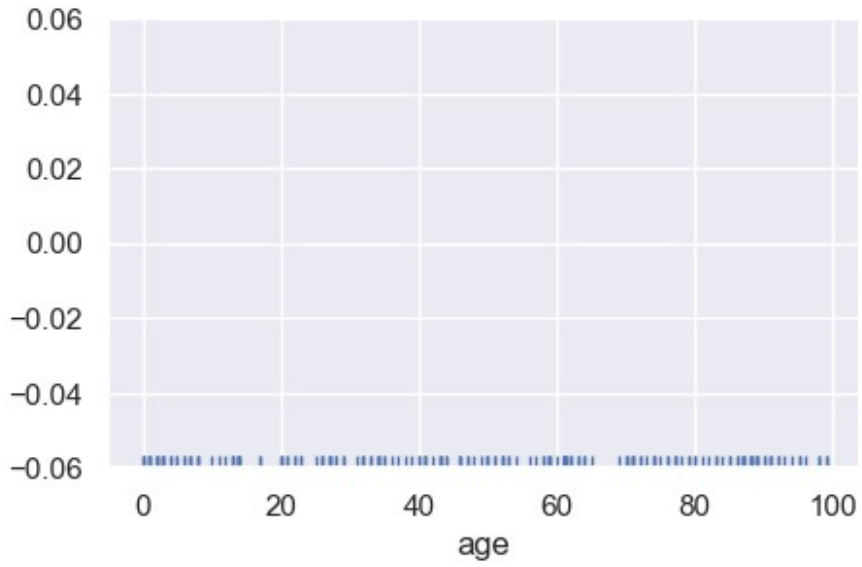
```
# Creting sample
np.random.seed(42)
sample_ages=np.random.randint(0,100,200)
sample_ages

# randint should be uniform, each age has the same chance of being
chosen
# note: in reality ages are almost never uniformally distributed, but
this is just an example
```

```
array([51, 92, 14, 71, 60, 20, 82, 86, 74, 74, 87, 99, 23,  2, 21, 52,
1,
       87, 29, 37,  1, 63, 59, 20, 32, 75, 57, 21, 88, 48, 90, 58, 41,
91,
       59, 79, 14, 61, 61, 46, 61, 50, 54, 63,  2, 50,  6, 20, 72, 38,
17,
        3, 88, 59, 13,  8, 89, 52,  1, 83, 91, 59, 70, 43,  7, 46, 34,
77,
       80, 35, 49,  3,  1,  5, 53,  3, 53, 92, 62, 17, 89, 43, 33, 73,
61,
       99, 13, 94, 47, 14, 71, 77, 86, 61, 39, 84, 79, 81, 52, 23, 25,
88,
       59, 40, 28, 14, 44, 64, 88, 70,  8, 87,  0,  7, 87, 62, 10, 80,
7,
       34, 34, 32,  4, 40, 27,  6, 72, 71, 11, 33, 32, 47, 22, 61, 87,
36,
       98, 43, 85, 90, 34, 64, 98, 46, 77,  2,  0,  4, 89, 13, 26,  8,
78,
       14, 89, 41, 76, 50, 62, 95, 51, 95,  3, 93, 22, 14, 42, 28, 35,
12,
       31, 70, 58, 85, 27, 65, 41, 44, 61, 56,  5, 27, 27, 43, 83, 29,
61,
       74, 91, 88, 61, 96,  0, 26, 61, 76,  2, 69, 71, 26])
```

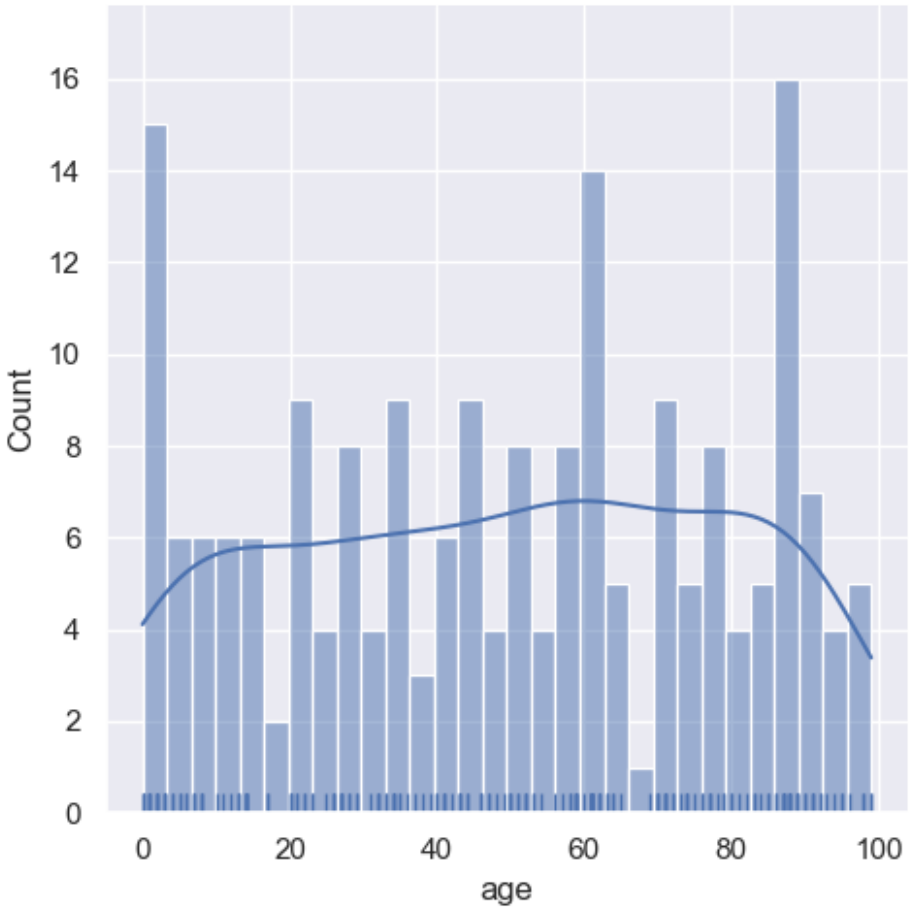
```
sample_ages= pd.DataFrame(sample_ages, columns=['age'])
sample_ages.head()
```

```
   age
0   51
1   92
2   14
3   71
4   60
```

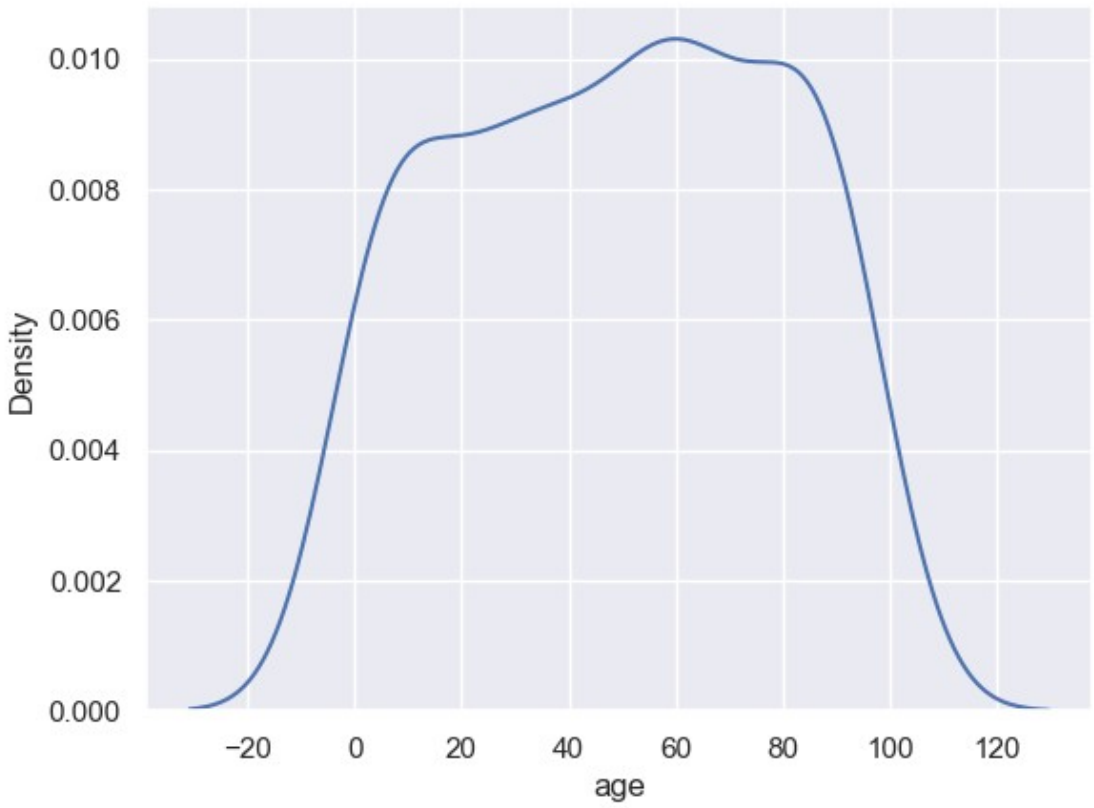
```
plt.figure(figsize=(5,3))
sns.rugplot(x='age',data=sample_ages);
```




```
sns.displot(x='age',data=sample_ages,rug=True,bins=30,kde=True);
```



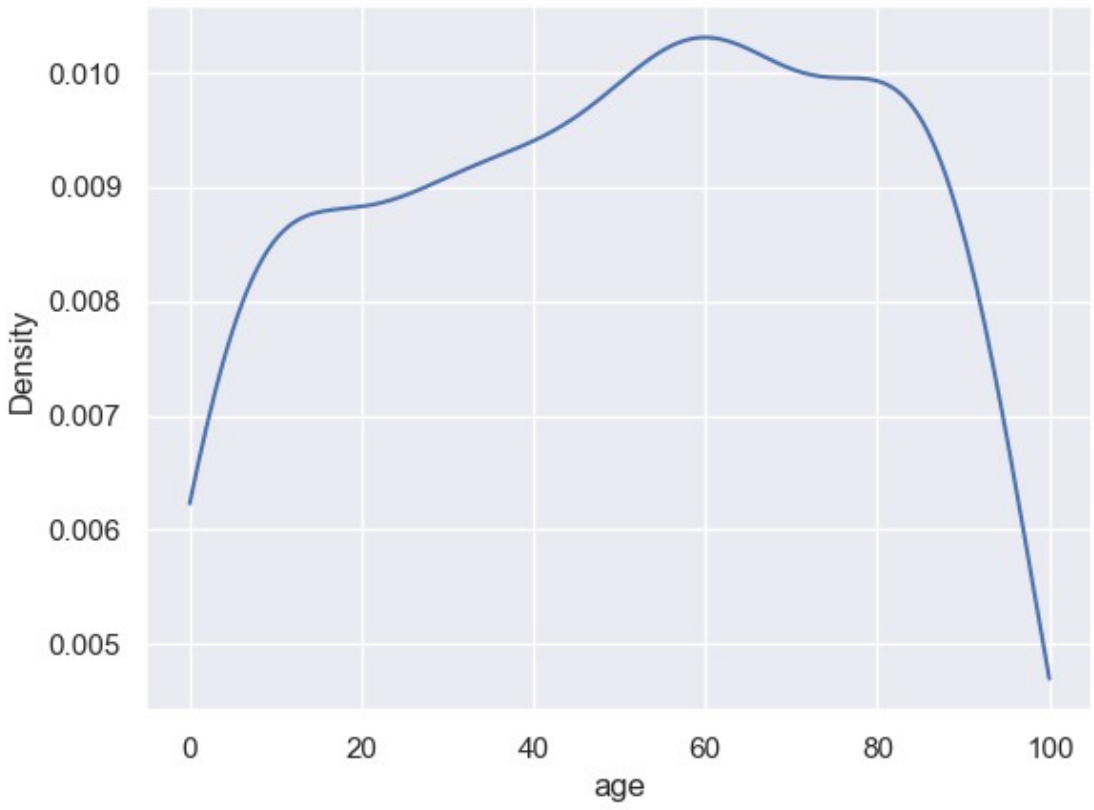

```
sns.kdeplot(x='age',data=sample_ages);
```



Cut Off KDE

We could cut off the KDE if we know our data has hard limits (no one can be a negative age and no one in the population can be older than 100 for some reason)

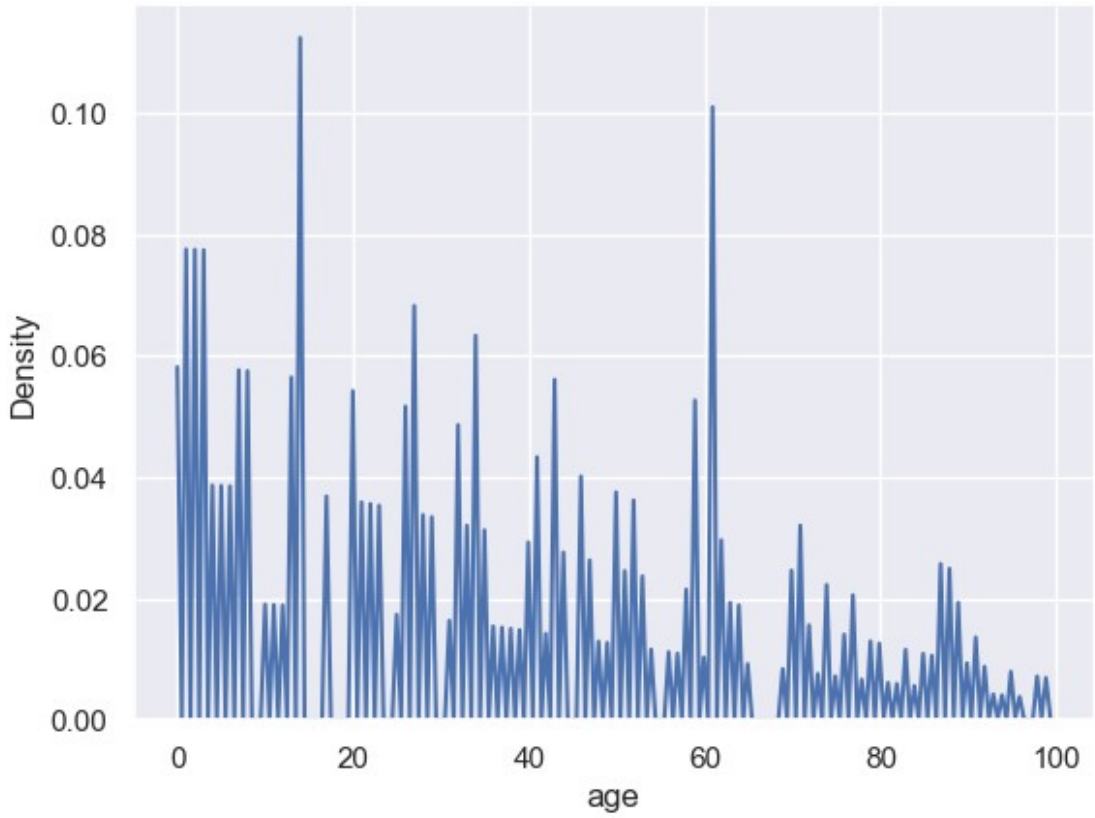
```
sns.kdeplot(x='age',data=sample_ages,clip=[0,100]);
```



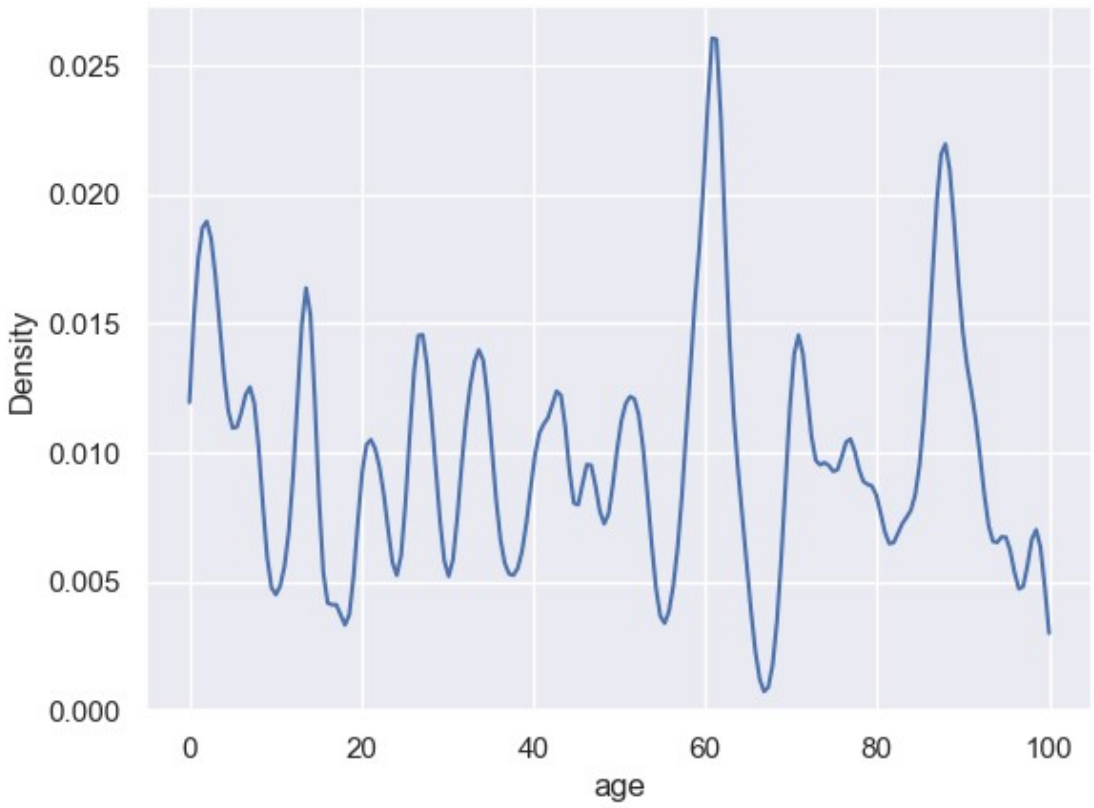
Bandwidth

As explained in the video, the KDE is constructed through the summation of the kernel (most commonly Gaussian), we can effect the bandwith of this kernel to make the KDE more "sensitive" to the data. Notice how with a smaller bandwith, the kernels don't stretch so wide, meaning we don't need the cut-off anymore. This is analagous to increasing the number of bins in a histogram (making the actual bins narrower).

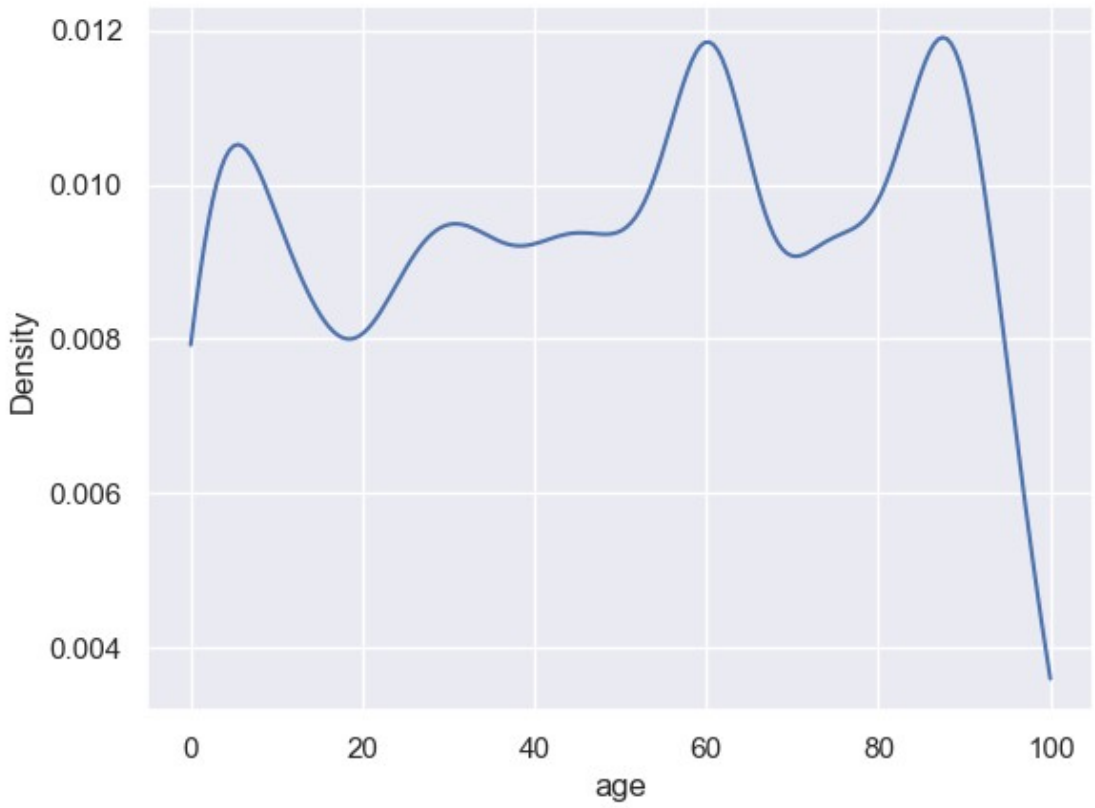
```
# Here we adjust bandwidth but that is too low
sns.kdeplot(x='age',data=sample_ages,clip=[0,100],bw_adjust=0.01);
```



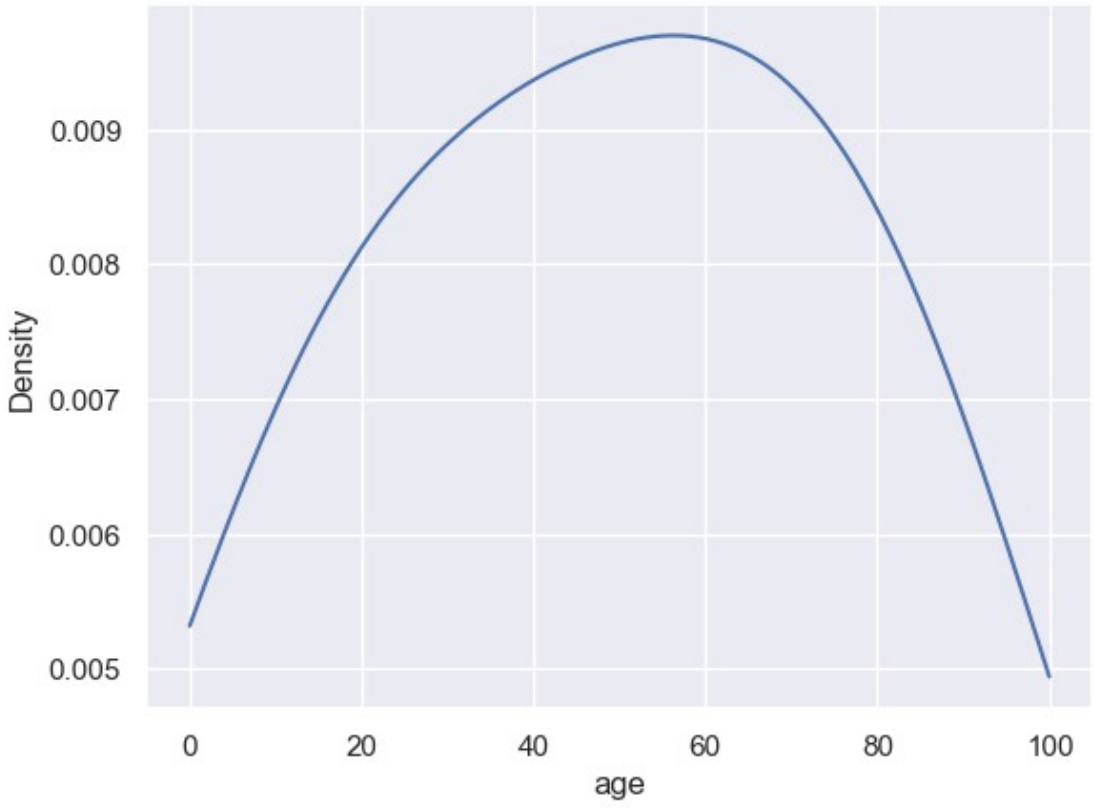

```
# Here we adjust bandwidth
sns.kdeplot(x='age',data=sample_ages,clip=[0,100],bw_adjust=0.1);
```




```
# Here we adjust bandwidth
sns.kdeplot(x='age',data=sample_ages,clip=[0,100],bw_adjust=0.5);
```




```
# Here we adjust bandwidth too high it will not caught data
sns.kdeplot(x='age',data=sample_ages,clip=[0,100],bw_adjust=2);
```



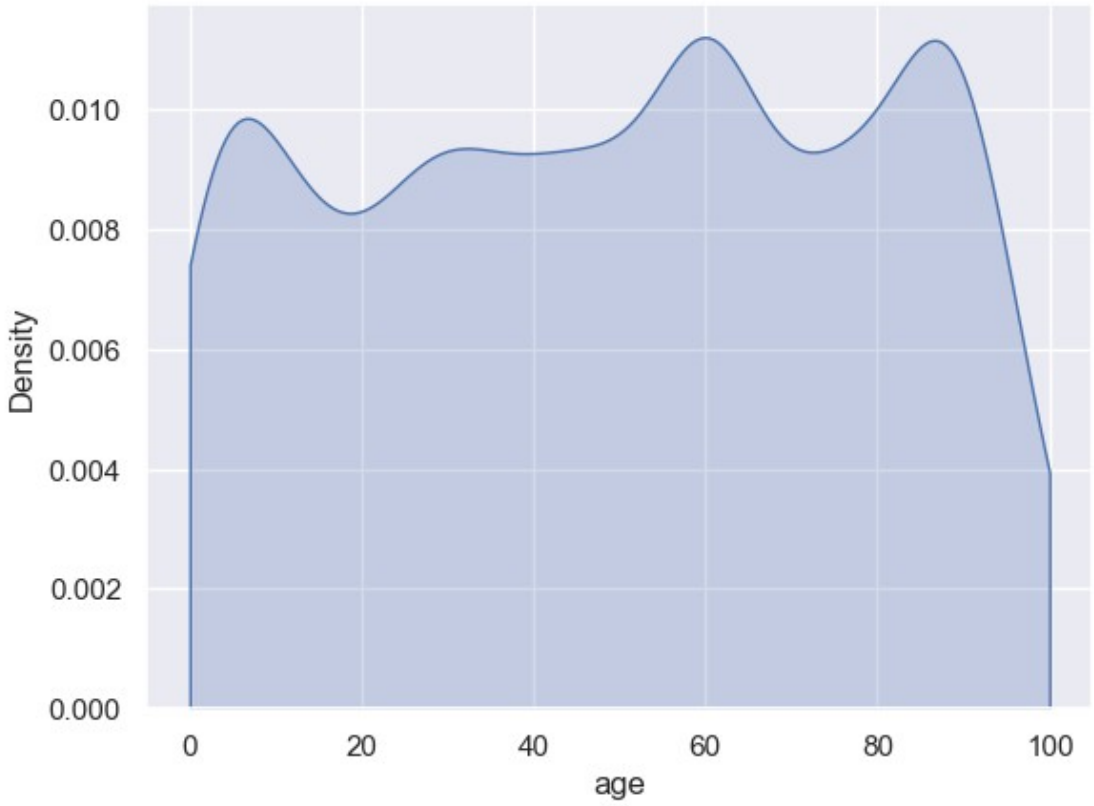
Basic Styling

There are a few basic styling calls directly availble in a KDE.

```
# We can shade and color in kde plot too
sns.kdeplot(x='age',data=sample_ages,clip=[0,100],bw_adjust=.6,shade=T
rue);
```

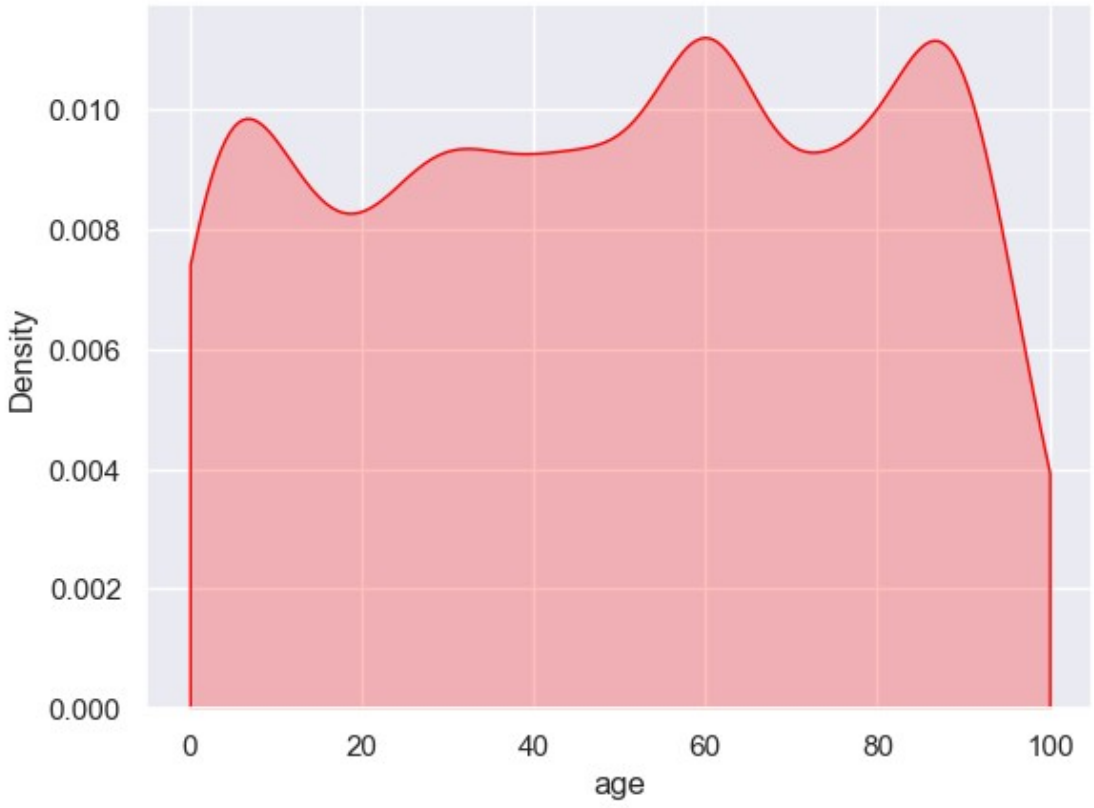
```
C:\Users\Chromsy\AppData\Local\Temp\ipykernel_9544\2271456020.py:2:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.
```

```
sns.kdeplot(x='age',data=sample_ages,clip=[0,100],bw_adjust=.6,shade=T
rue);
```




```
# We can shade and color in kde plot too
sns.kdeplot(x='age',data=sample_ages,clip=[0,100],bw_adjust=.6,shade=T
rue,color='red');
```

```
C:\Users\Chromsy\AppData\Local\Temp\ipykernel_9544\61414404.py:2:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.


sns.kdeplot(x='age',data=sample_ages,clip=[0,100],bw_adjust=.6,shade=T
rue,color='red');
```



2 Dimensional KDE Plots

We will cover these in more detail later, but just keep in mind you could compare two continuous features and create a 2d KDE plot showing there distributions with the same kdeplot() call. Don't worry about this now, since we will cover it in more detail later when we talk about comparing 2 features against each other in a plot call.

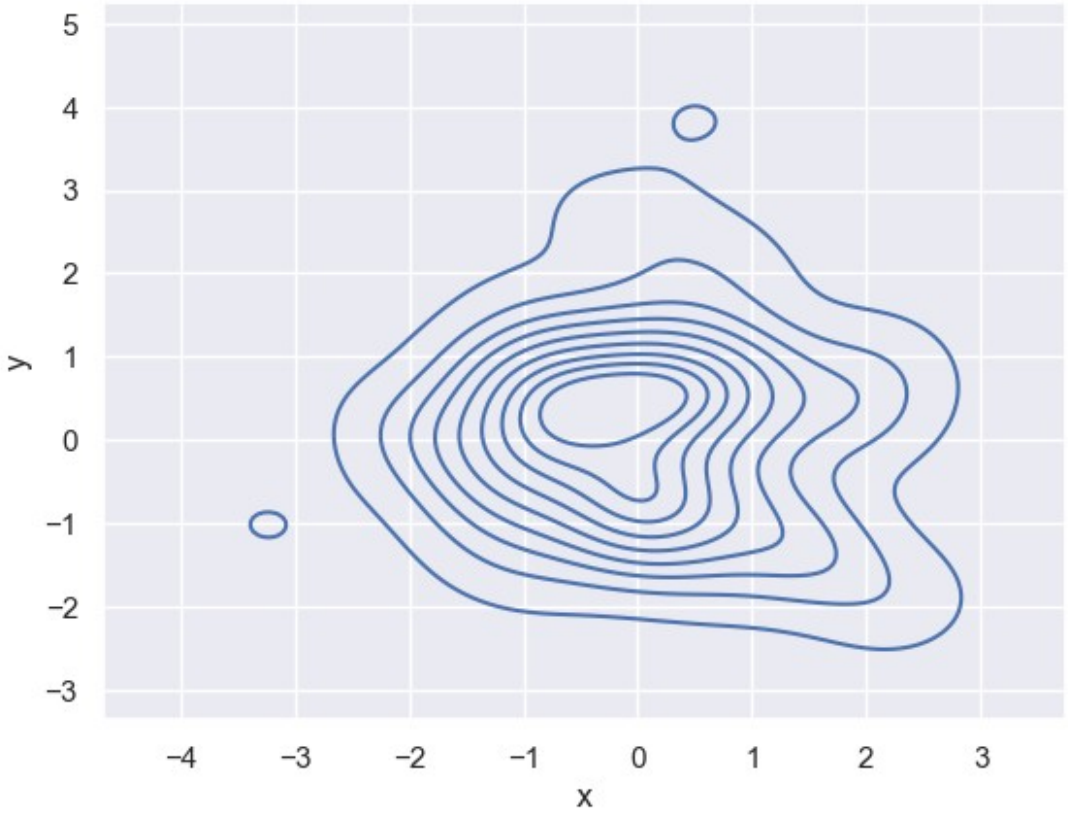
```
random_data =
pd.DataFrame(np.random.normal(0,1,size=(100,2)),columns=['x','y'])
```

```
random_data
```

```
           x         y
0  -1.415371 -0.420645
1  -0.342715 -0.802277
2  -0.161286  0.404051
3   1.886186  0.174578
4   0.257550 -0.074446
..       ...       ...
95 -0.208122 -0.493001
96 -0.589365  0.849602
97  0.357015 -0.692910
98  0.899600  0.307300
99  0.812862  0.629629
```

```
[100 rows x 2 columns]
```

```
sns.kdeplot(data=random_data,x='x',y='y');
```



Bonus Code for Visualizations from Video Lecture

Below is the code used to create the visualizations shown in the video lecture for an explanation of a KDE plot. We will not cover this code in further detail, since it was only used for the creation of the slides shown in the video.

```
pip install scipy
```

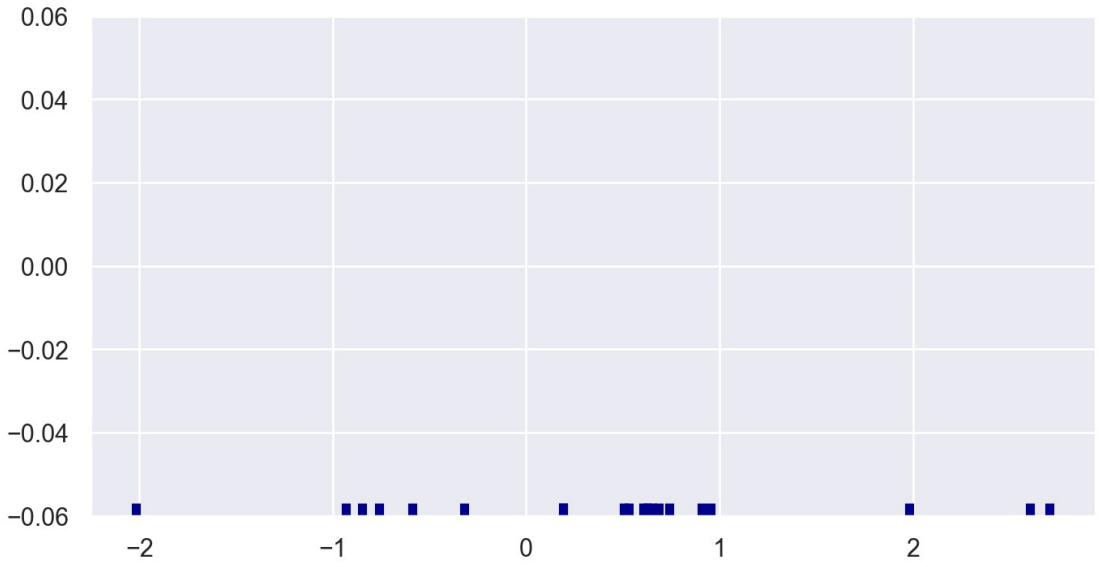
```
from scipy import stats
```

Data

```
np.random.seed(101)
x = np.random.normal(0, 1, size=20)
```

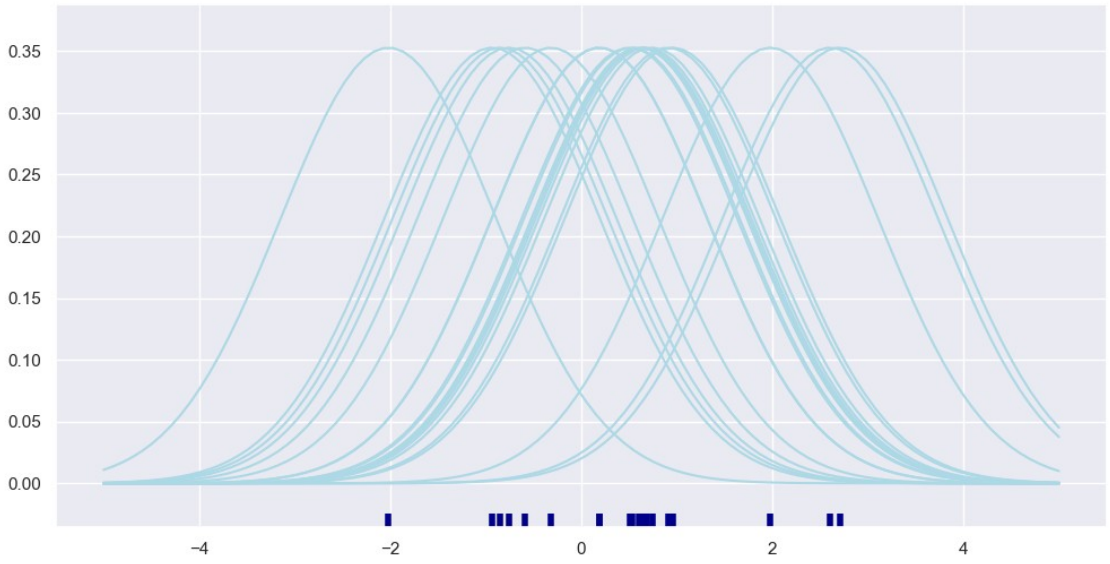
```
plt.figure(figsize=(8,4),dpi=200)
sns.rugplot(x, color="darkblue", linewidth=4)
```

```
<AxesSubplot: >
```



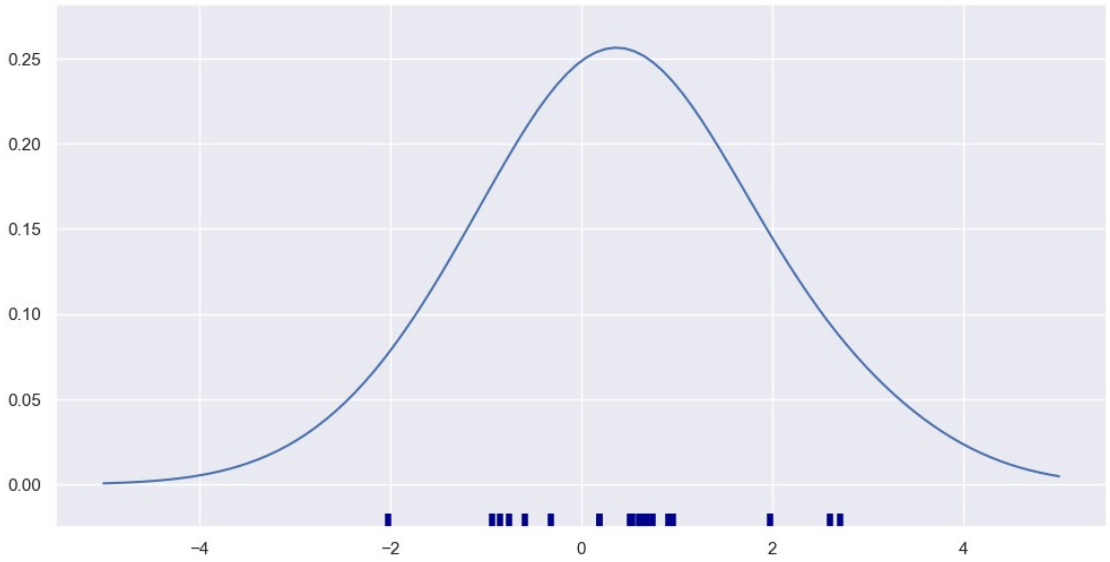

```
plt.figure(figsize=(8,4),dpi=200)
bandwidth = x.std() * x.size ** (-0.001)
support = np.linspace(-5, 5, 100)

kernels = []

plt.figure(figsize=(12,6))

for x_i in x:

    kernel = stats.norm(x_i, bandwidth).pdf(support)
    kernels.append(kernel)
    plt.plot(support, kernel, color="lightblue")

sns.rugplot(x, color="darkblue", linewidth=4);
```

```
<Figure size 1600x800 with 0 Axes>
```




```
plt.figure(figsize=(8,4),dpi=200)
from scipy.integrate import trapz
plt.figure(figsize=(12,6))
density = np.sum(kernels, axis=0)
density /= trapz(density, support)
plt.plot(support, density);
sns.rugplot(x, color="darkblue", linewidth=4);
```

```
<Figure size 1600x800 with 0 Axes>
```



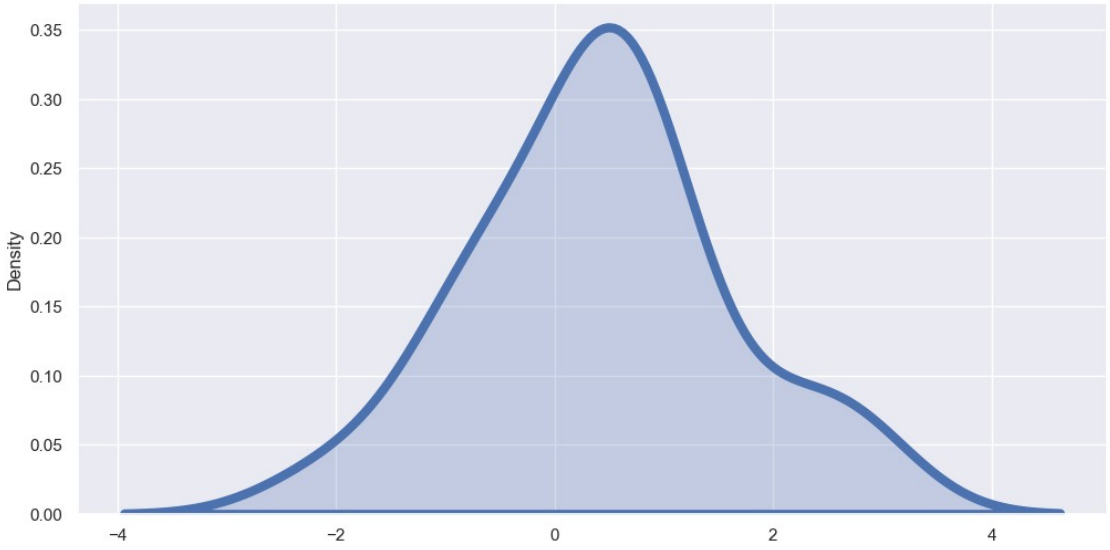

```
plt.figure(figsize=(8,4),dpi=200)
bandwidth = x.std() * x.size ** (-0.001)
support = np.linspace(-5, 5, 100)

kernels = []
```

```
plt.figure(figsize=(12,6))

for x_i in x:

    kernel = stats.norm(x_i, bandwidth).pdf(support)
    kernels.append(kernel)
#     plt.plot(support, kernel, color="lightblue")

# sns.rugplot(x, color="darkblue", linewidth=4);
sns.kdeplot(x,linewidth=6,shade=True)
```

```
C:\Users\Chromsy\AppData\Local\Temp\ipykernel_9544\921692335.py:16:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(x,linewidth=6,shade=True)
```

```
<AxesSubplot: ylabel='Density'>
```

```
<Figure size 1600x800 with 0 Axes>
```



Categorical Plots - Statistical Estimation within Categories

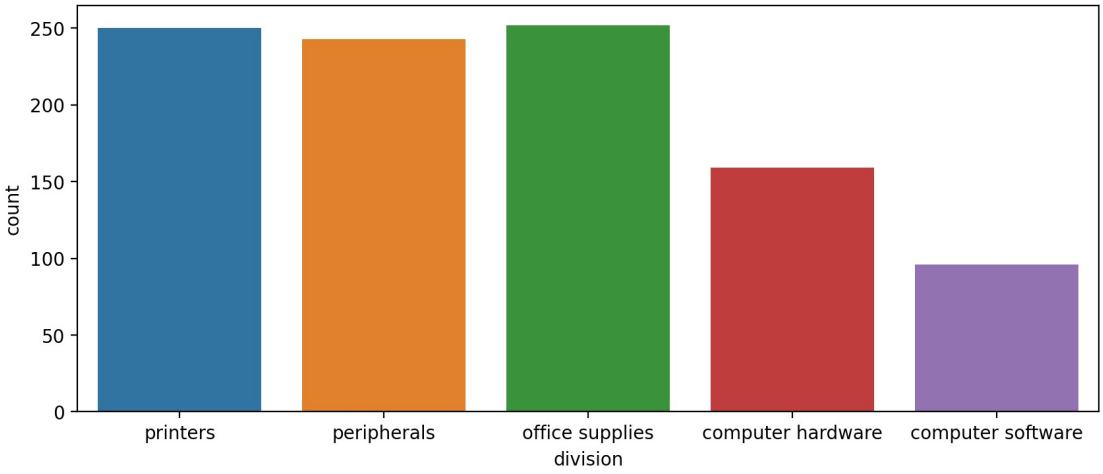
Often we have **categorical** data, meaning the data is in distinct groupings, such as Countries or Companies. There is no country value "between" USA and France and there is no company value "between" Google and Apple, unlike continuous data where we know values can exist between data points, such as age or price.

To begin with categorical plots, we'll focus on statistical estimation within categories. Basically this means we will visually report back some statistic (such as mean or count) in a plot. We already know how to get this data with pandas, but often its easier to understand the data if we plot this.

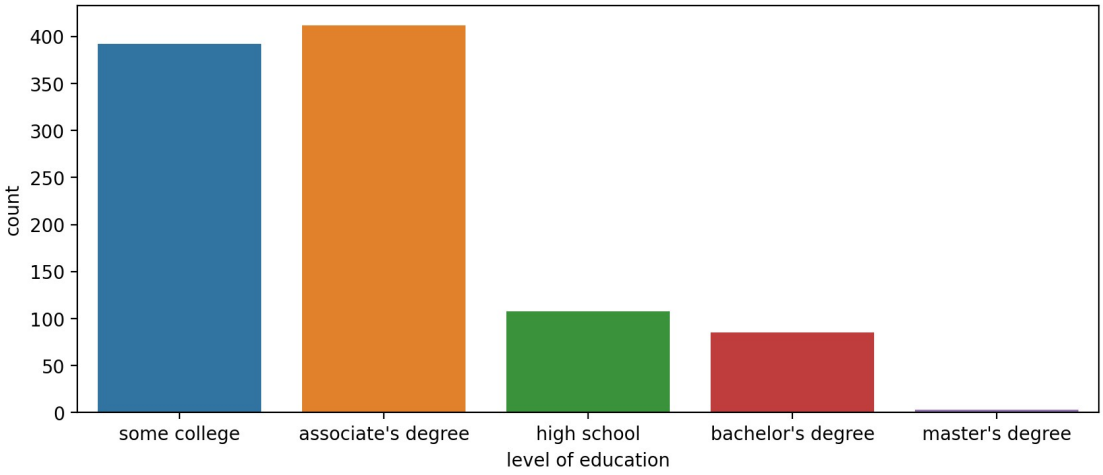
Countplot()

A simple plot, it merely shows the total count of rows per category.

```
plt.figure(figsize=(10,4),dpi=200)
sns.countplot(x='division',data=df);
```

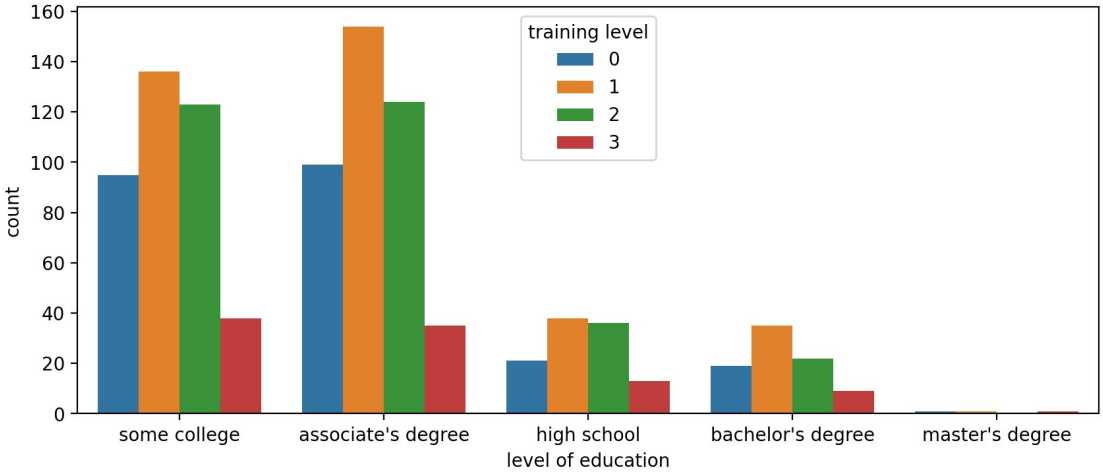



```
plt.figure(figsize=(10,4),dpi=200)
sns.countplot(x='level of education',data=df);
```



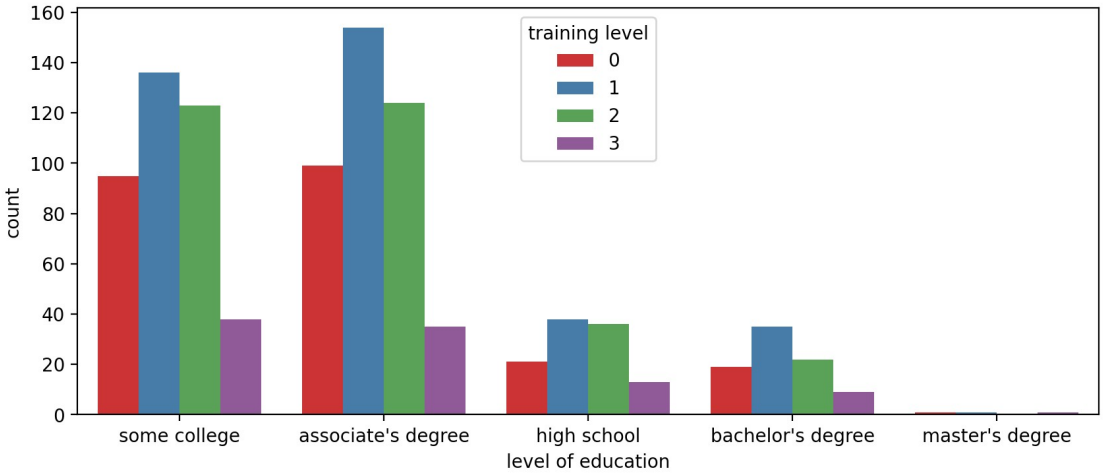
Breakdown within another category with 'hue'

```
plt.figure(figsize=(10,4),dpi=200)
sns.countplot(x='level of education',data=df,hue='training level');
```



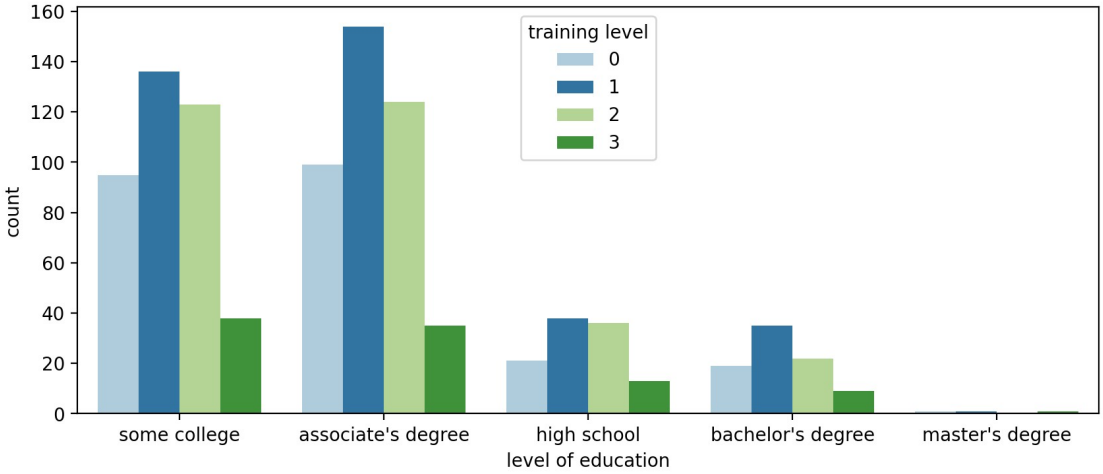
NOTE: You can always edit the palette to your liking to any matplotlib colormap

```
plt.figure(figsize=(10,4),dpi=200)
sns.countplot(x='level of education',data=df,hue='training
level',palette='Set1');
```




```
plt.figure(figsize=(10,4),dpi=200)
# Paired would be a good choice if there was a distinct jump from 0
and 1 to 2 and 3
sns.countplot(x='level of education',data=df,hue='training
level',palette='Paired')
```

```
<AxesSubplot:xlabel='level of education', ylabel='count'>
```

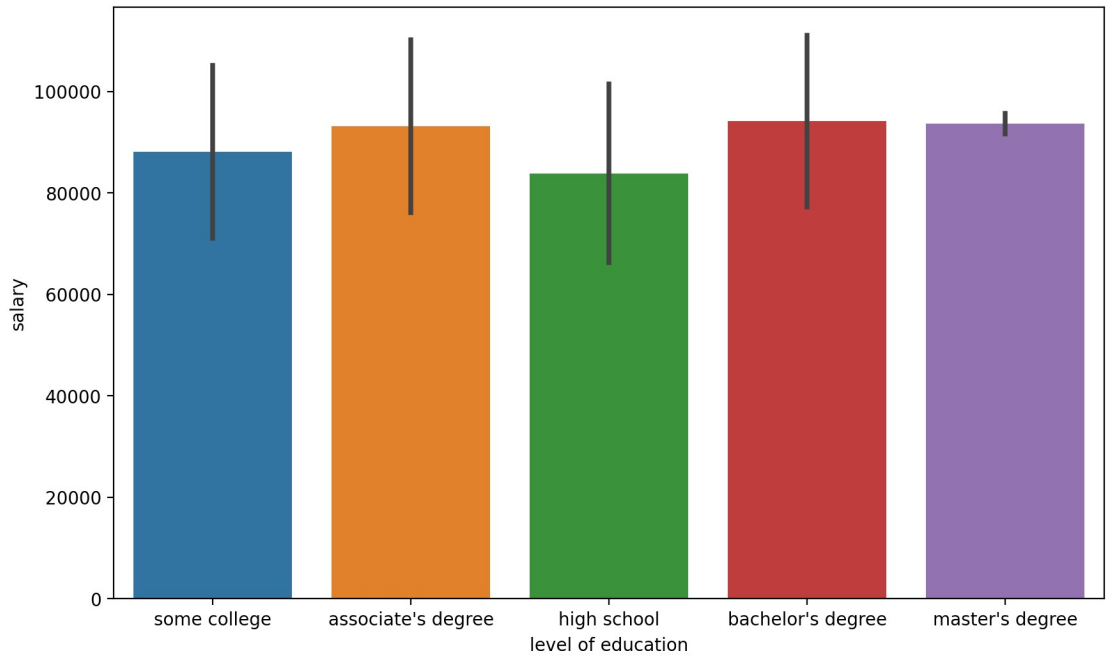


barplot()

So far we've seen the y axis default to a count (similar to a .groupby(x_axis).count() call in pandas). We can expand our visualizations by specifying a specific continuous feature for the y-axis. Keep in mind, you should be careful with these plots, as they may imply a relationship continuity along the y axis where there is none.

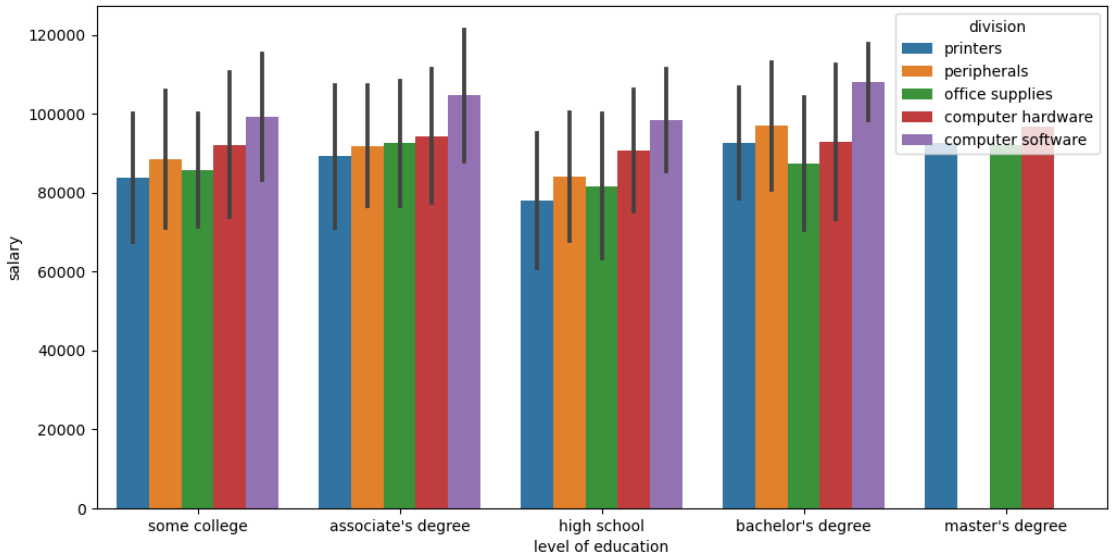
```
plt.figure(figsize=(10,6),dpi=200)
# By default barplot() will show the mean
# Information on the black bar:
https://stackoverflow.com/questions/58362473/what-does-black-lines-on-a-seaborn-barplot-mean
sns.barplot(x='level of
education',y='salary',data=df,estimator=np.mean,ci='sd')
```

```
<AxesSubplot:xlabel='level of education', ylabel='salary'>
```




```
plt.figure(figsize=(12,6))
sns.barplot(x='level of
education',y='salary',data=df,estimator=np.mean,ci='sd',hue='division'
)
```

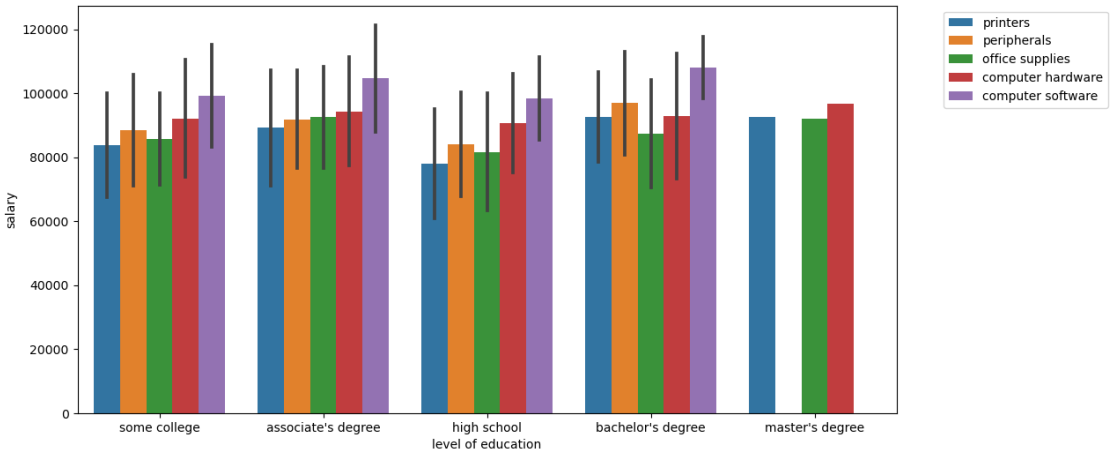
```
<AxesSubplot:xlabel='level of education', ylabel='salary'>
```




```
plt.figure(figsize=(12,6),dpi=100)

# https://stackoverflow.com/questions/30490740/move-legend-outside-
figure-in-seaborn-tsplot
sns.barplot(x='level of
education',y='salary',data=df,estimator=np.mean,ci='sd',hue='division'
```

```
)

plt.legend(bbox_to_anchor=(1.05, 1))
```

```
<matplotlib.legend.Legend at 0x203fe8127c0>
```



Data

Loading New data for upcoming plots

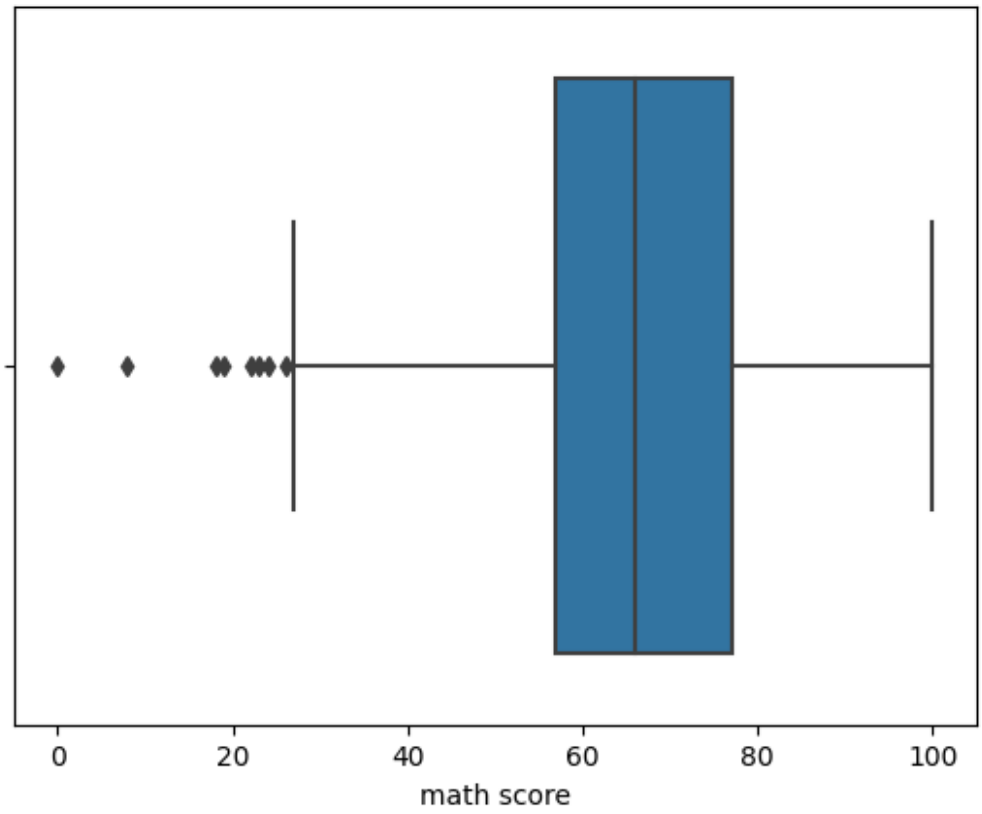
```
dff=pd.read_csv("D:\\Study\\Programming\\python\\Python course from udemy\\Udemy - 2022 Python for Machine Learning & Data Science Masterclass\\01 - Introduction to Course\\1UNZIP-FOR-NOTEBOOKS-FINAL\\05-Seaborn\\StudentsPerformance.csv")
dff.head()
```

```
   gender race/ethnicity parental level of education         lunch  \
0  female        group B           bachelor's degree      standard
1  female        group C                some college      standard
2  female        group B             master's degree      standard
3    male        group A          associate's degree  free/reduced
4    male        group C                some college      standard

  test preparation course  math score  reading score  writing score
0                    none          72             72             74
1               completed          69             90             88
2                    none          90             95             93
3                    none          47             57             44
4                    none          76             78             75
```

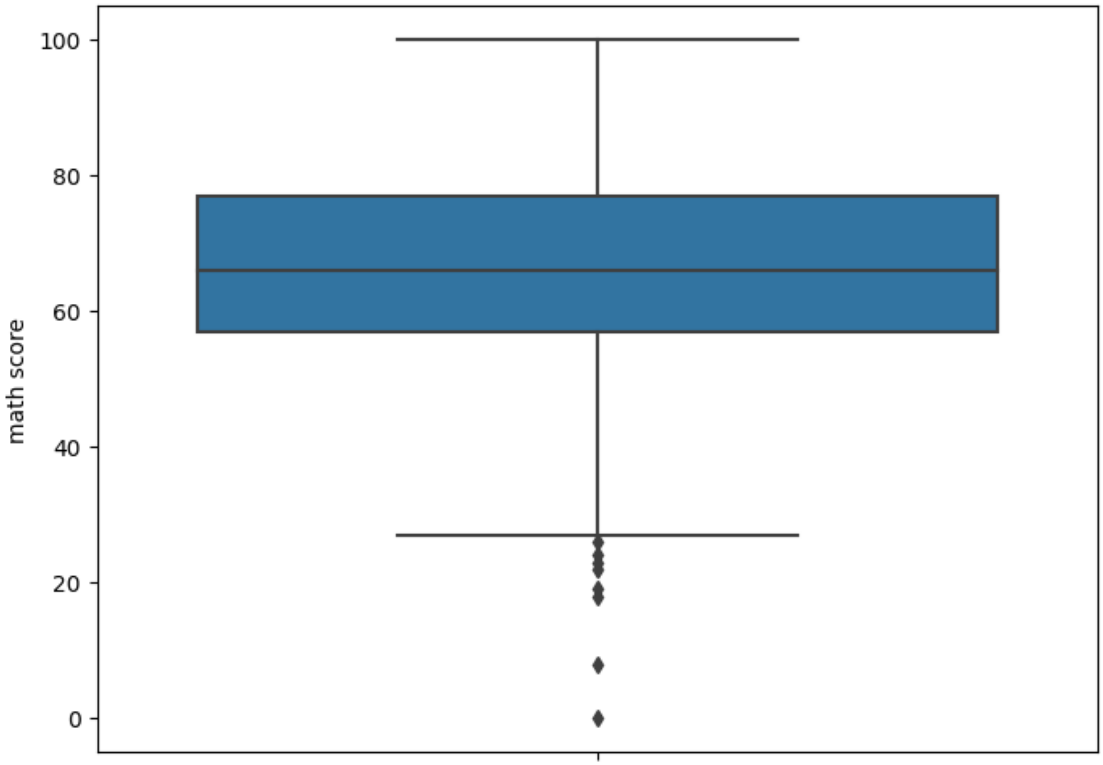
Boxplot

As described in the video, a boxplot display distribution through the use of quartiles and an IQR for outliers.

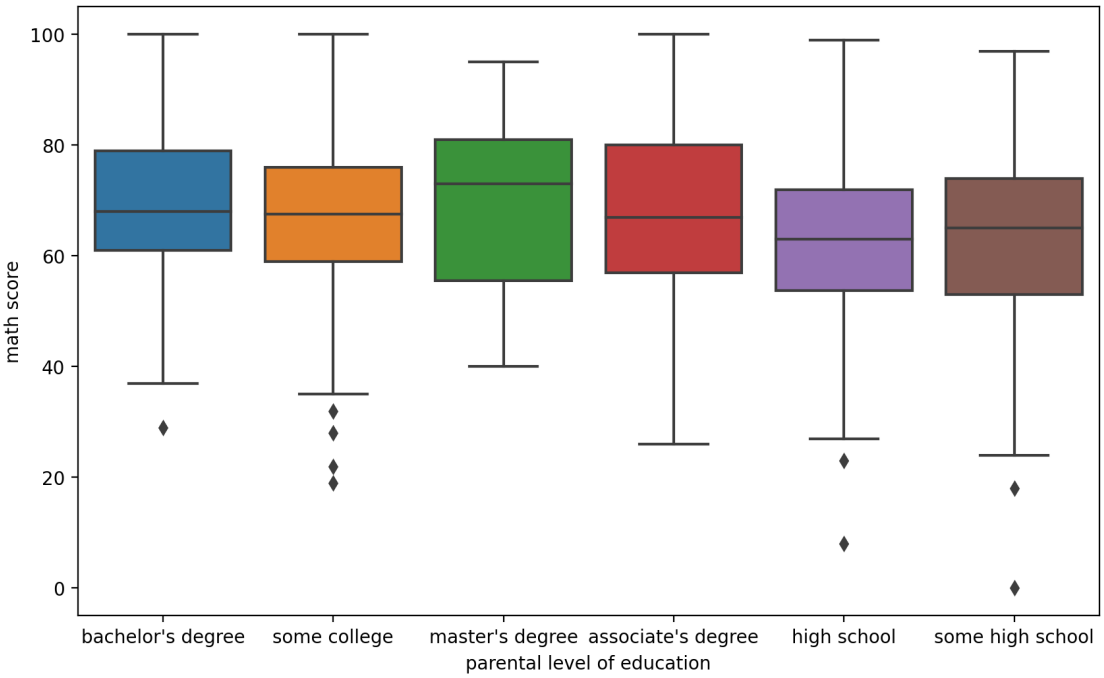
```
# Here is just maths score so we can approx 65
sns.boxplot(x='math score',data=dff);
```



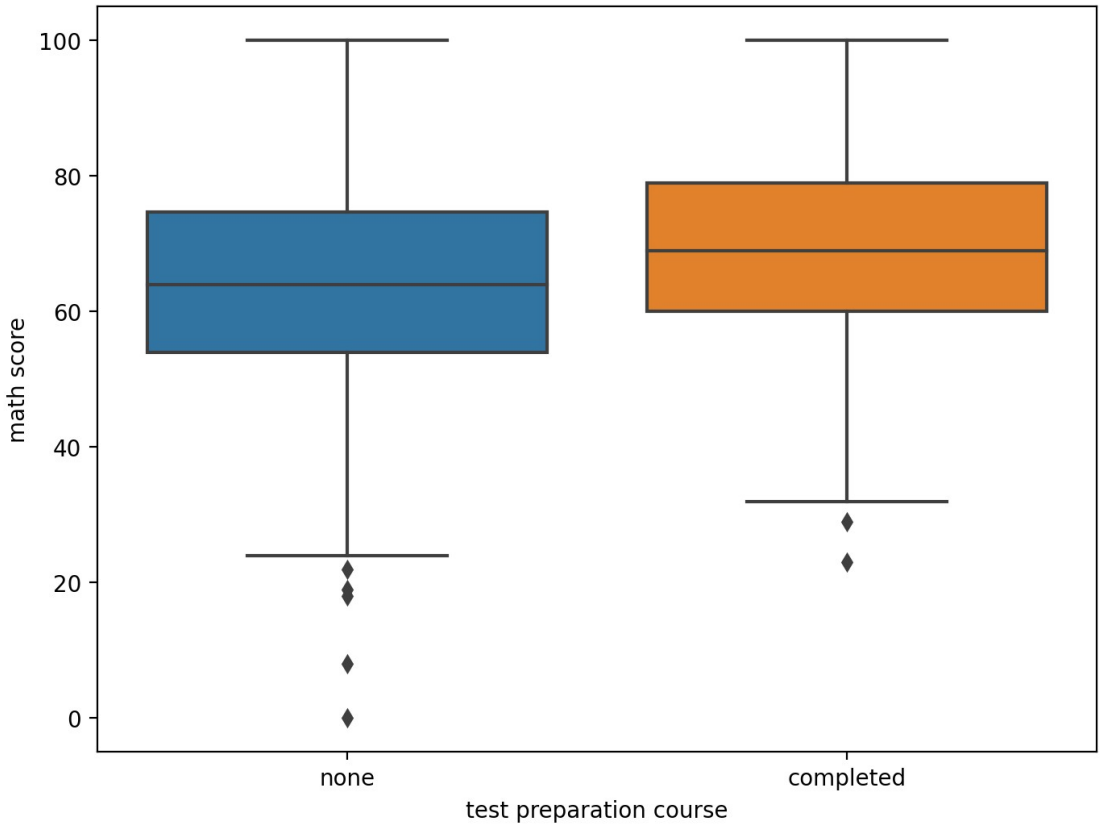

```
# To change horizontal to vertial we can math score on y
plt.figure(figsize=(8,6))
sns.boxplot(y='math score', data= dff);
```




```
# Here we add on the basis of parental level of education
plt.figure(figsize=(10,6),dpi=200)
sns.boxplot(x='parental level of education',y='math score',data=dff);
```



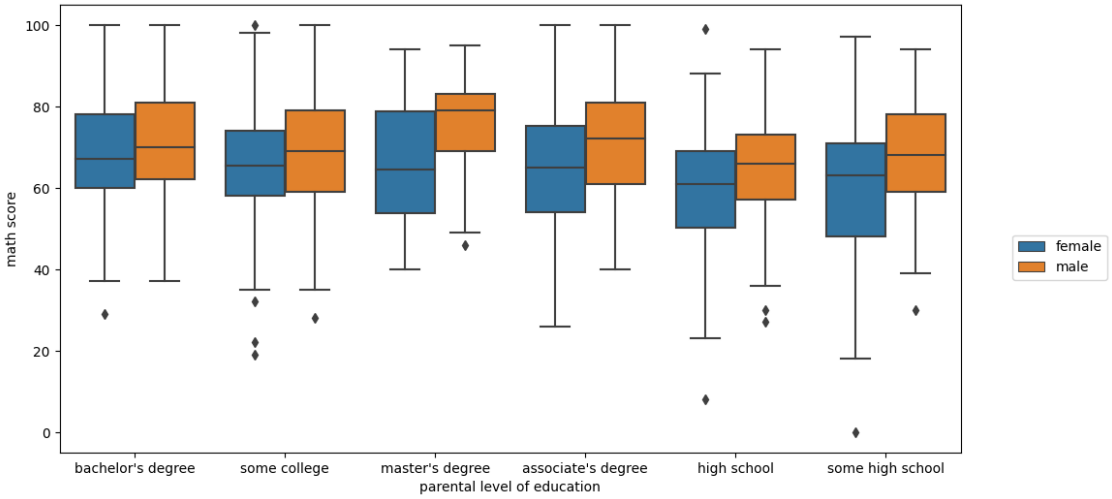
```
# Here we add on the basis of test preparation course
plt.figure(figsize=(8,6),dpi=200)
sns.boxplot(x='test preparation course',y='math score',data=dff);
```



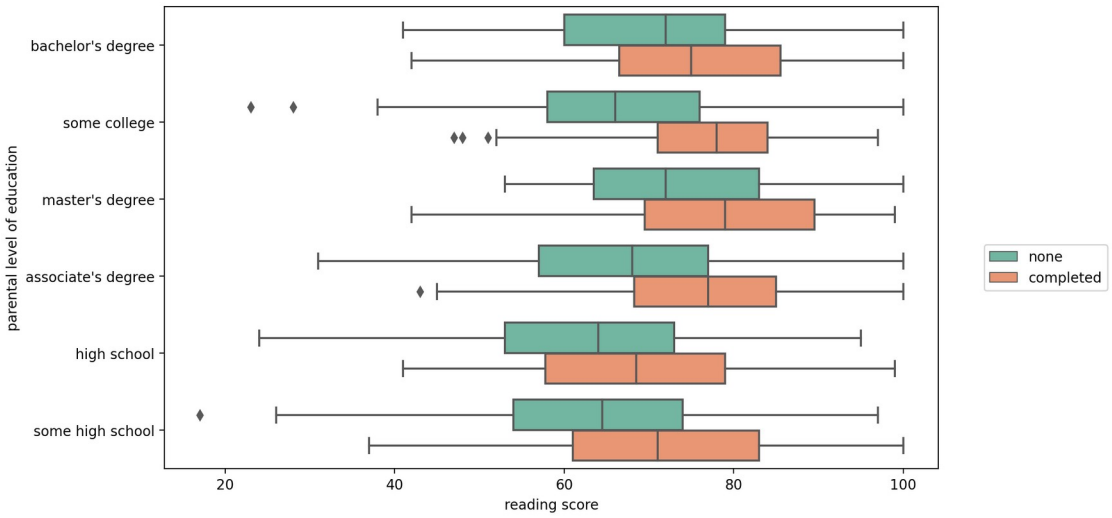
Adding hue for further segmentation

```
plt.figure(figsize=(12,6))
sns.boxplot(x='parental level of education',y='math
score',data=dff,hue='gender')

# Optional move the legend outside
plt.legend(bbox_to_anchor=(1.05, .5))
```

```
<matplotlib.legend.Legend at 0x24833deebf0>
```



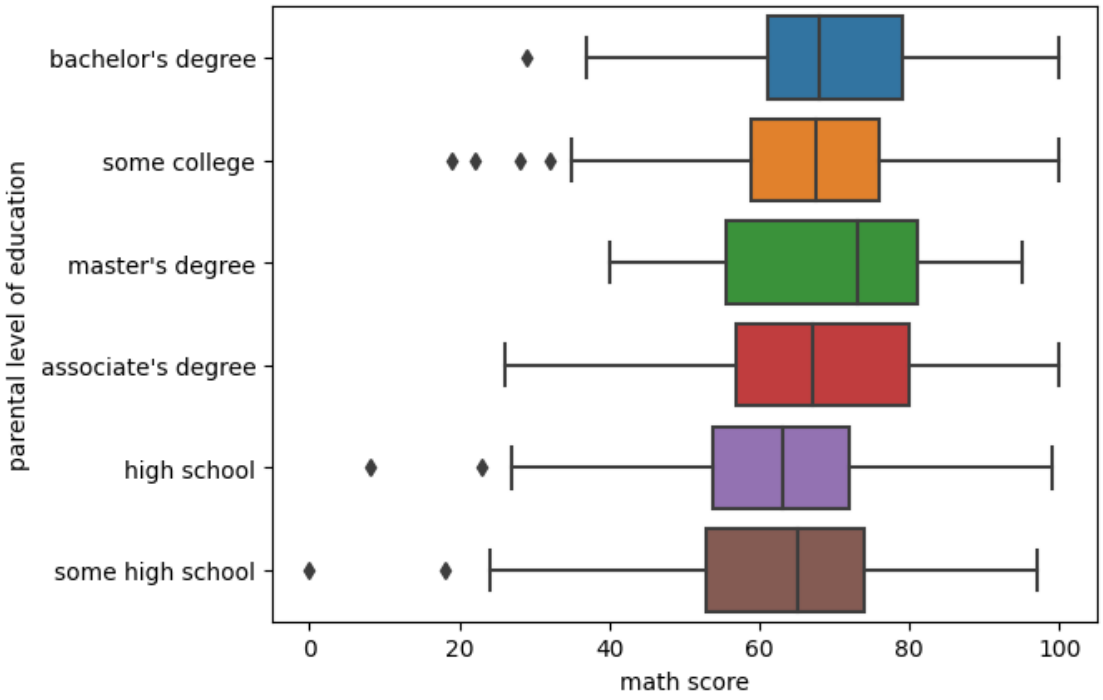

```
# Now reading score on the bases of parental level of education and
test preparation course and we can set palette too
plt.figure(figsize=(10,6),dpi=200)
sns.boxplot(x='reading score', y='parental level of education',
hue='test preparation course',data=dff, palette='Set2' )
plt.legend(bbox_to_anchor=(1.05,.5));
```



Boxplot Styling Parameters

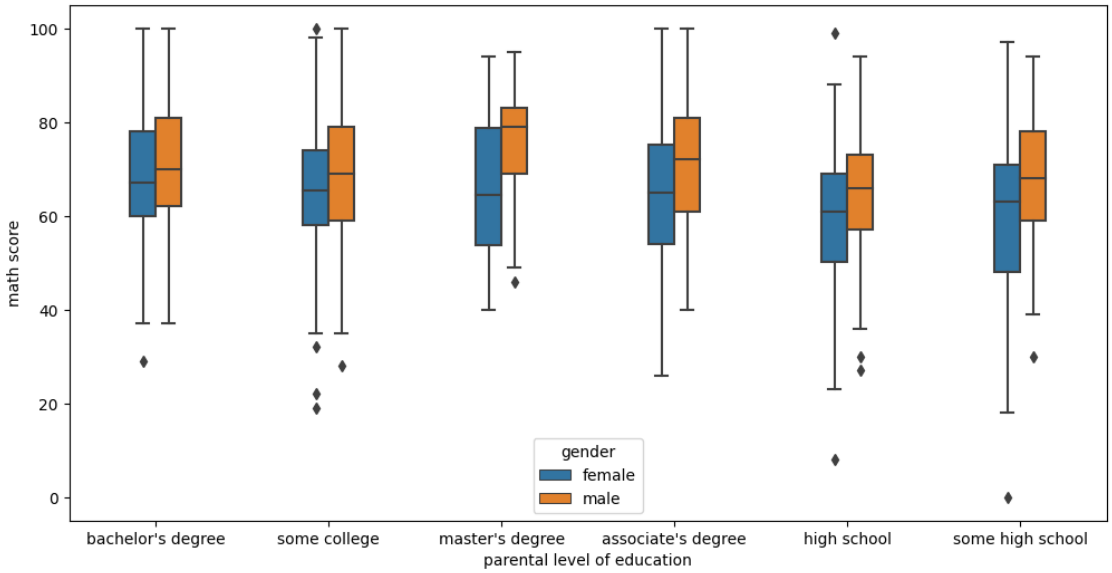
Orientation

```
# NOTICE HOW WE HAVE TO SWITCH X AND Y FOR THE ORIENTATION TO MAKE
SENSE!
sns.boxplot(x='math score',y='parental level of
education',data=dff,orient='h');
```



Width

```
plt.figure(figsize=(12,6))
sns.boxplot(x='parental level of education',y='math
score',data=dff,hue='gender',width=0.3);
```



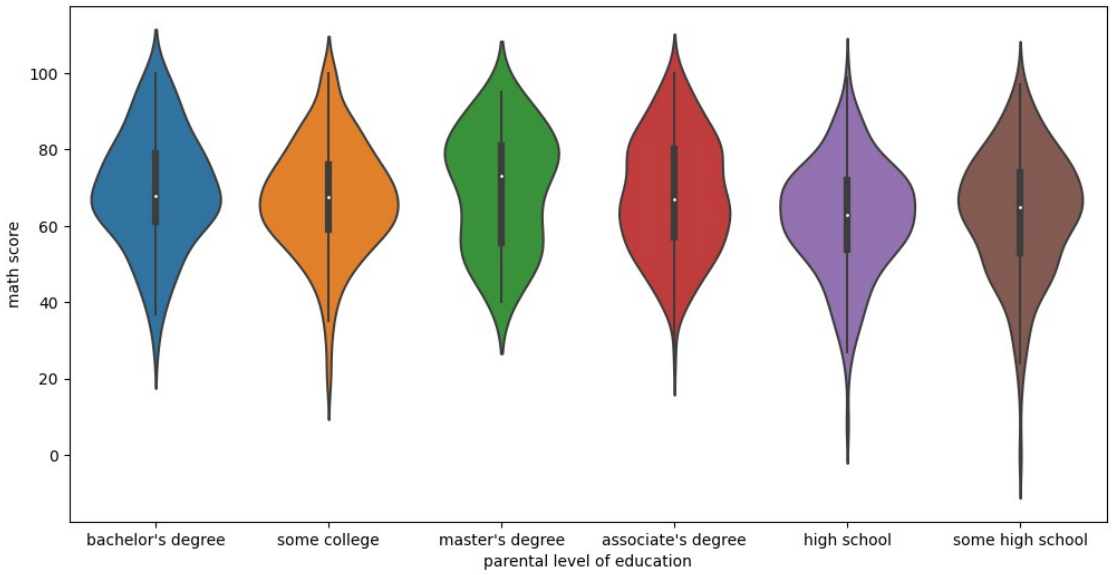
Violinplot

A violin plot plays a similar role as a box and whisker plot. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared. Unlike a box plot, in which all of the plot components

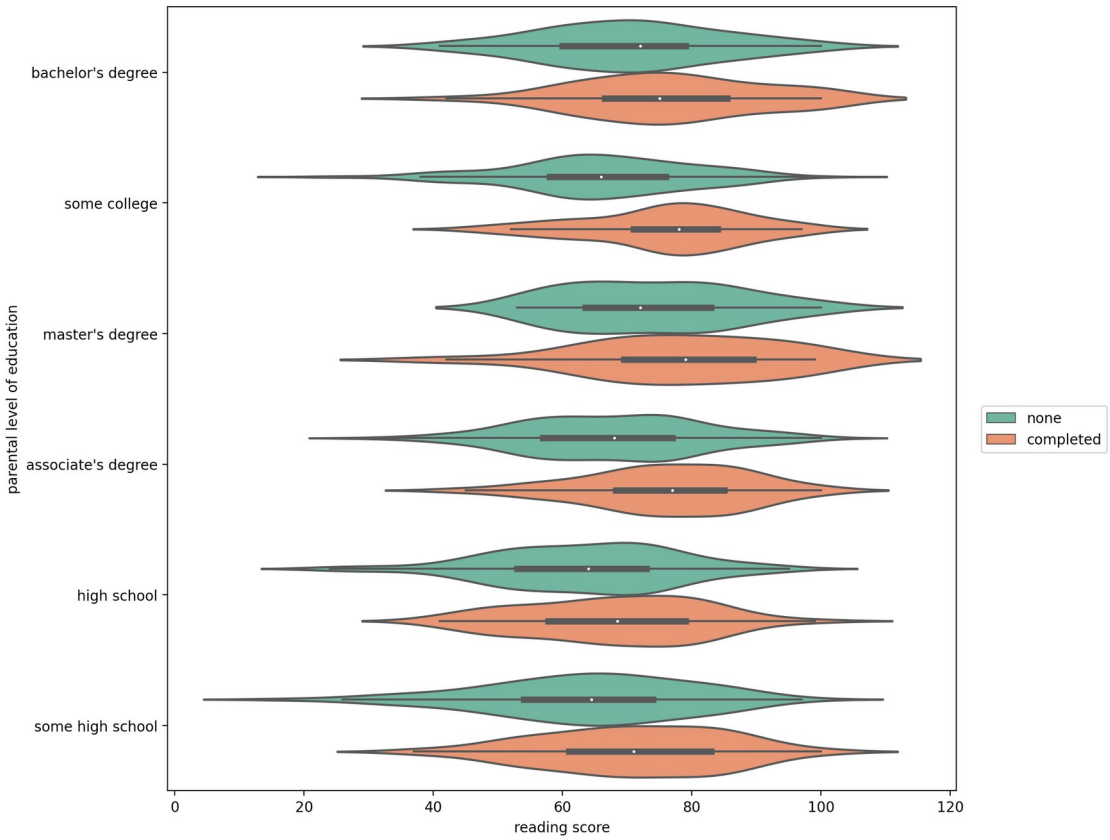
correspond to actual datapoints, the violin plot features a kernel density estimation of the underlying distribution.

```
plt.figure(figsize=(12,6))
sns.violinplot(x='parental level of education',y='math
score',data=dff)
```

```
<AxesSubplot: xlabel='parental level of education', ylabel='math
score'>
```




```
plt.figure(figsize=(10,10),dpi=200)
sns.violinplot(x='reading score',y='parental level of
education',data=dff,hue='test preparation course',palette='Set2')
plt.legend(bbox_to_anchor=(1.2,.5));
```

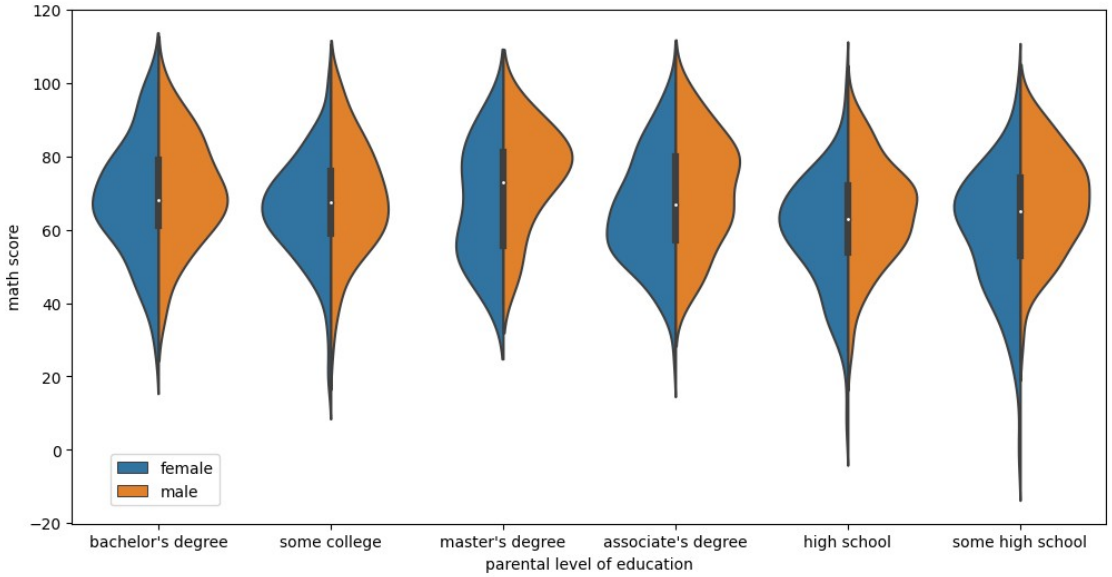


Violinplot Parameters

split

When using hue nesting with a variable that takes two levels, setting split to True will draw half of a violin for each level. This can make it easier to directly compare the distributions.

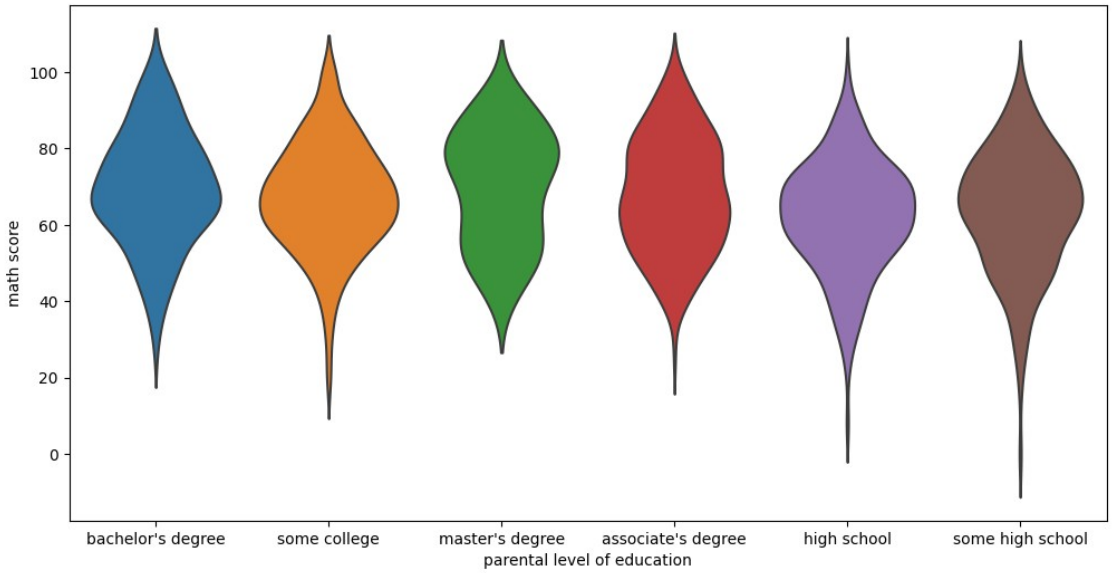
```
plt.figure(figsize=(12,6))
sns.violinplot(x='parental level of education',y='math
score',data=dff,hue='gender',split=True);
plt.legend(bbox_to_anchor=(0.15,0.15));
```



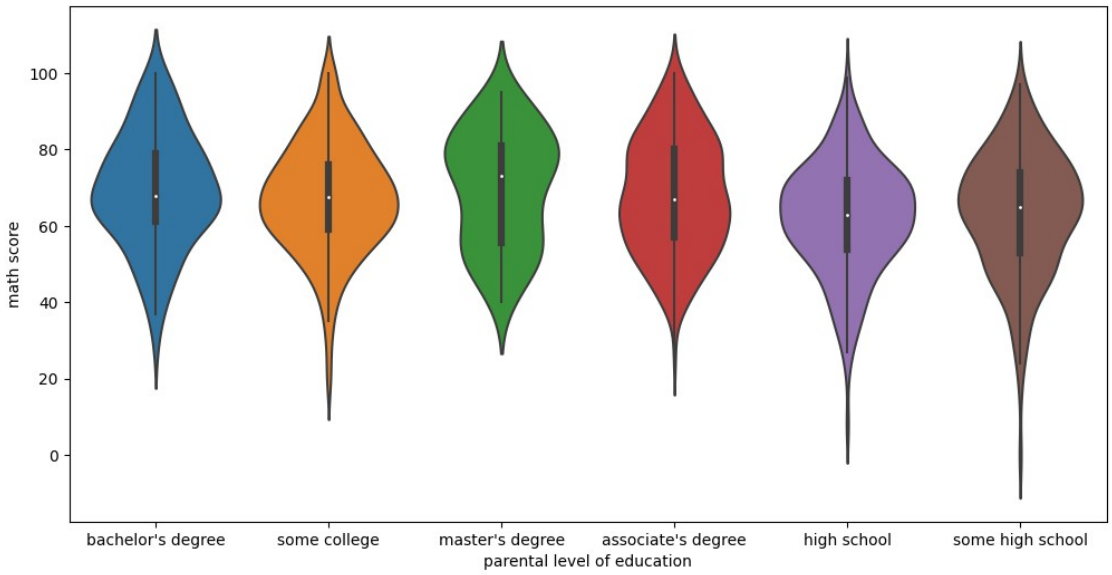
inner

Representation of the datapoints in the violin interior. If box, draw a miniature boxplot. If quartiles, draw the quartiles of the distribution. If point or stick, show each underlying datapoint. Using None will draw unadorned violins.

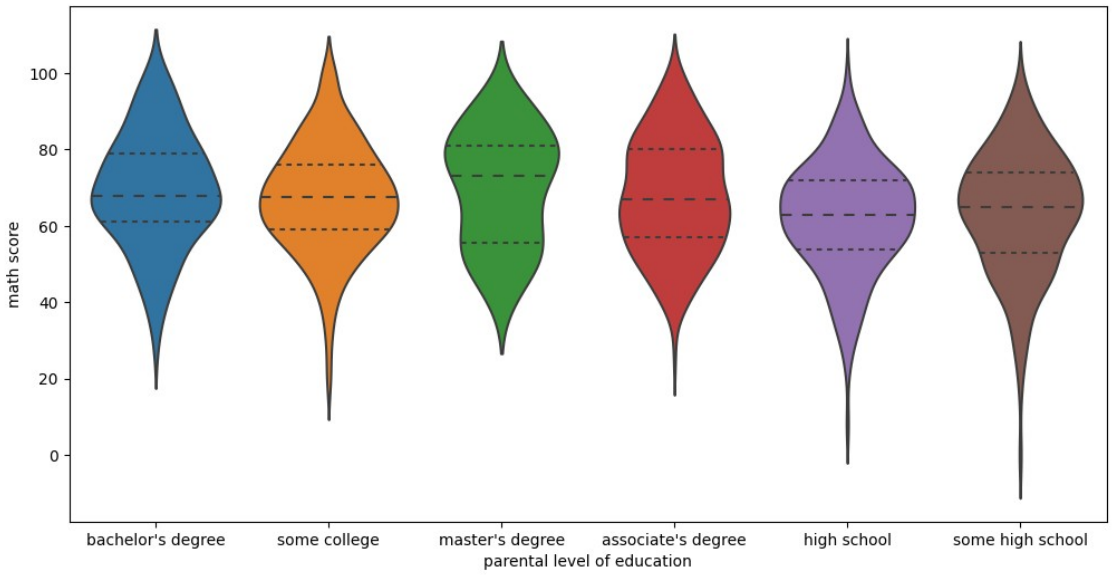
```
plt.figure(figsize=(12,6))
sns.violinplot(x='parental level of education',y='math
score',data=dff,inner=None);
```



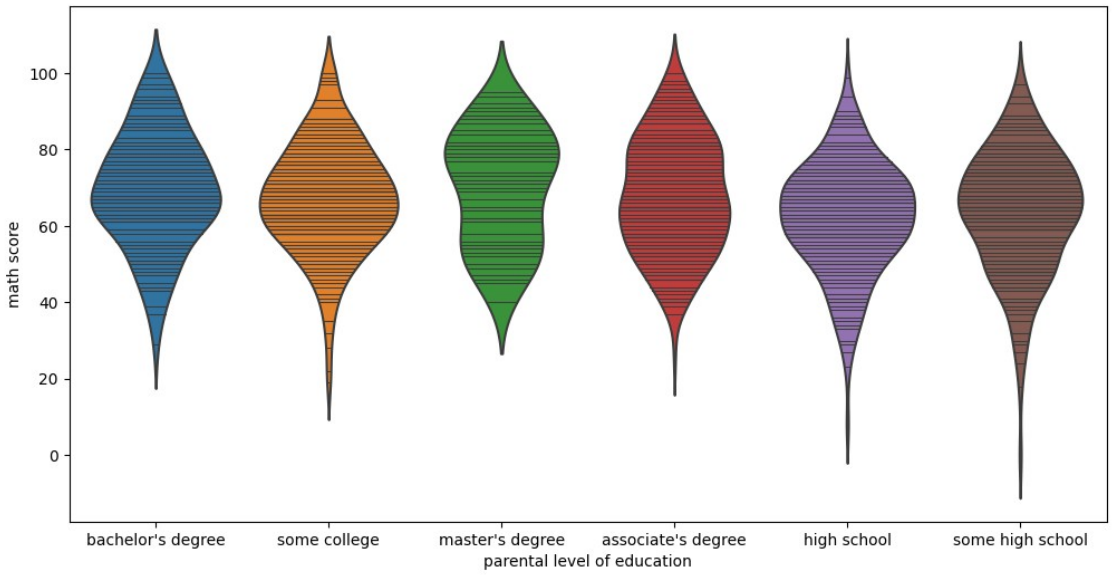

```
# we can put box plot inside
plt.figure(figsize=(12,6))
sns.violinplot(x='parental level of education',y='math
score',data=dff,inner='box');
```




```
# or we an put quartile inside
plt.figure(figsize=(12,6))
sns.violinplot(x='parental level of education',y='math
score',data=dff,inner='quartile');
```



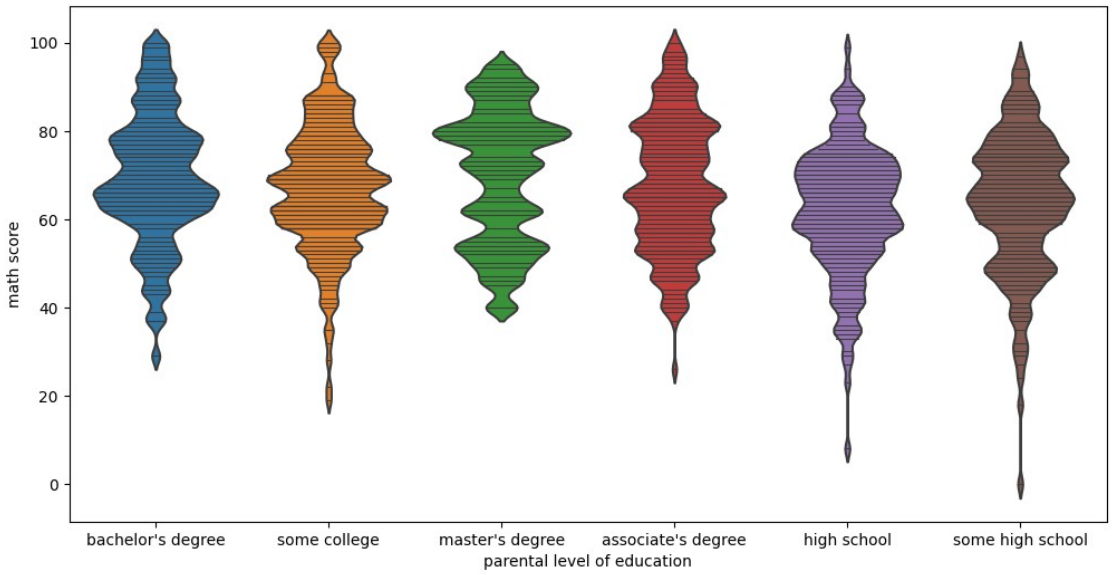

```
# or we an put stick inside
plt.figure(figsize=(12,6))
sns.violinplot(x='parental level of education',y='math
score',data=dff,inner='stick');
```



bandwidth

Similar to bandwidth argument for kdeplot

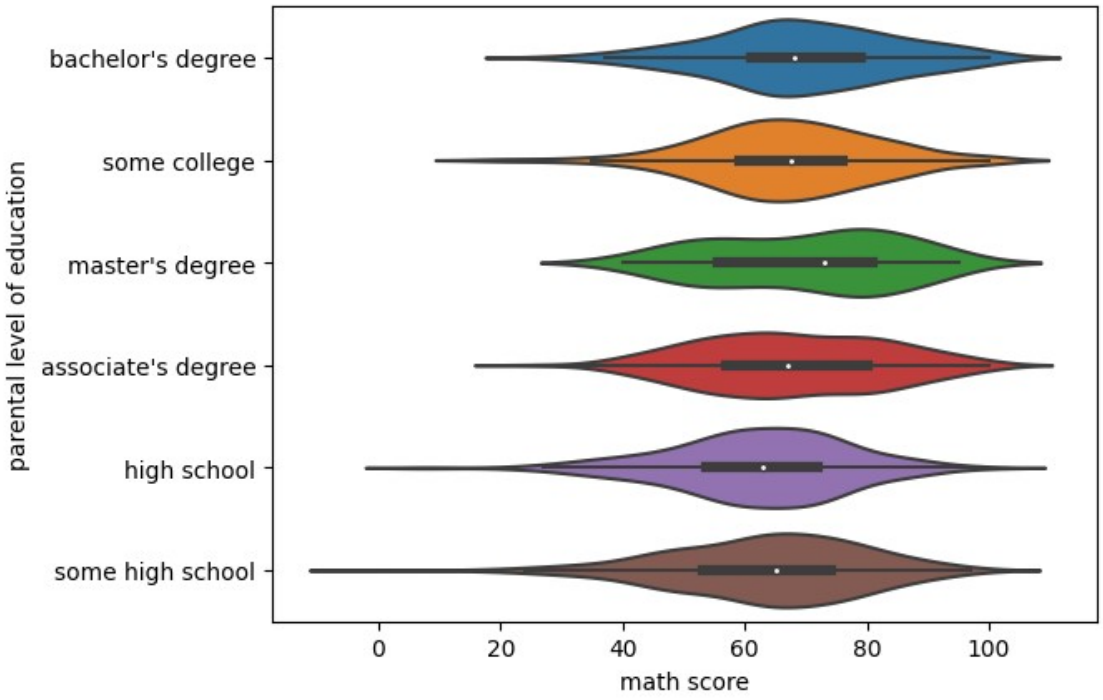
```
# Here we add bandwidth so lesser bandwidth will caught more noise
plt.figure(figsize=(12,6))
sns.violinplot(x='parental level of education',y='math
score',data=dff,inner='stick',bw=0.1);
```



orientation

```
# Simply switch the continuous variable to y and the categorical to x
sns.violinplot(x='math score',y='parental level of
education',data=dff,)
```

```
<AxesSubplot: xlabel='math score', ylabel='parental level of
education'>
```

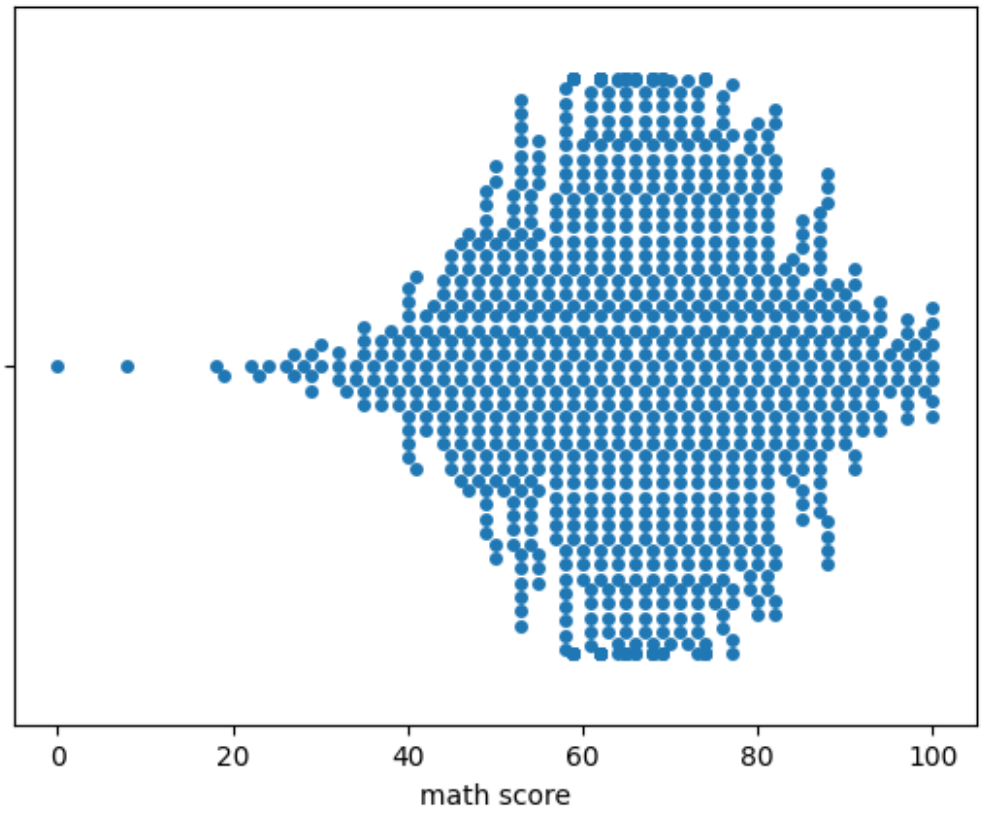


Advanced Plots

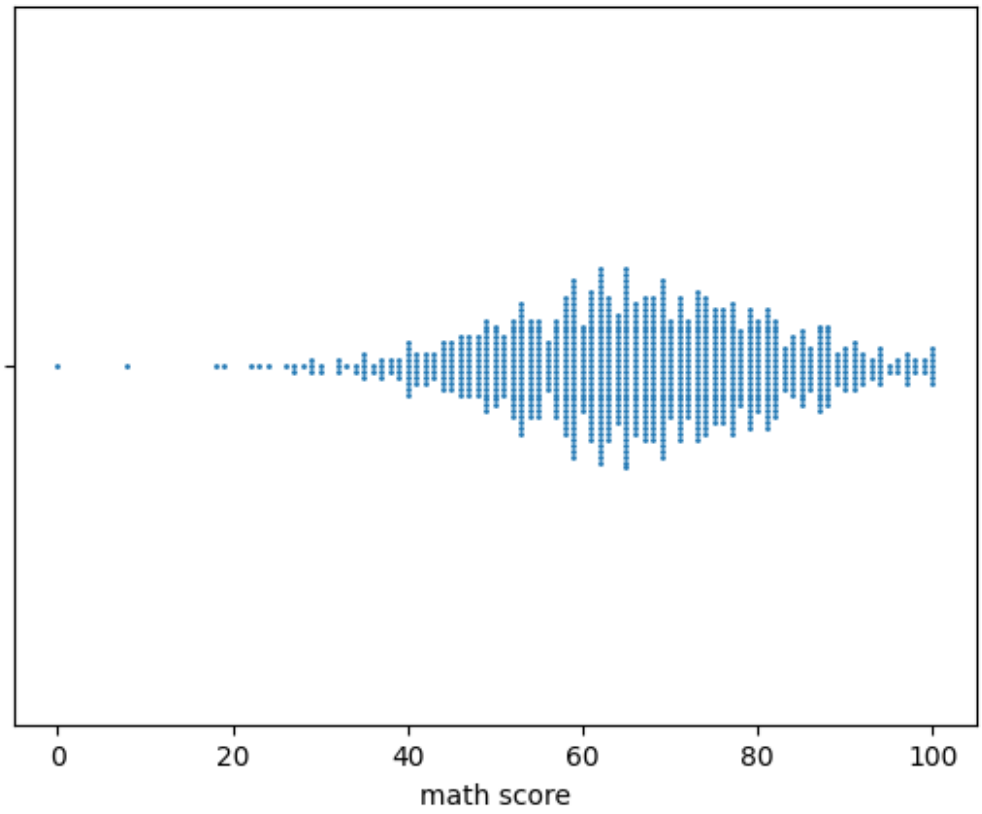
We can use a boxenplot and swarmplot to achieve the same effect as the boxplot and violinplot, but with slightly more information included. Be careful when using these plots, as they often require you to educate the viewer with how the plot is actually constructed. Only use these if you are sure your audience will understand the visualization.

```
sns.swarmplot(x='math score',data=dff);
```

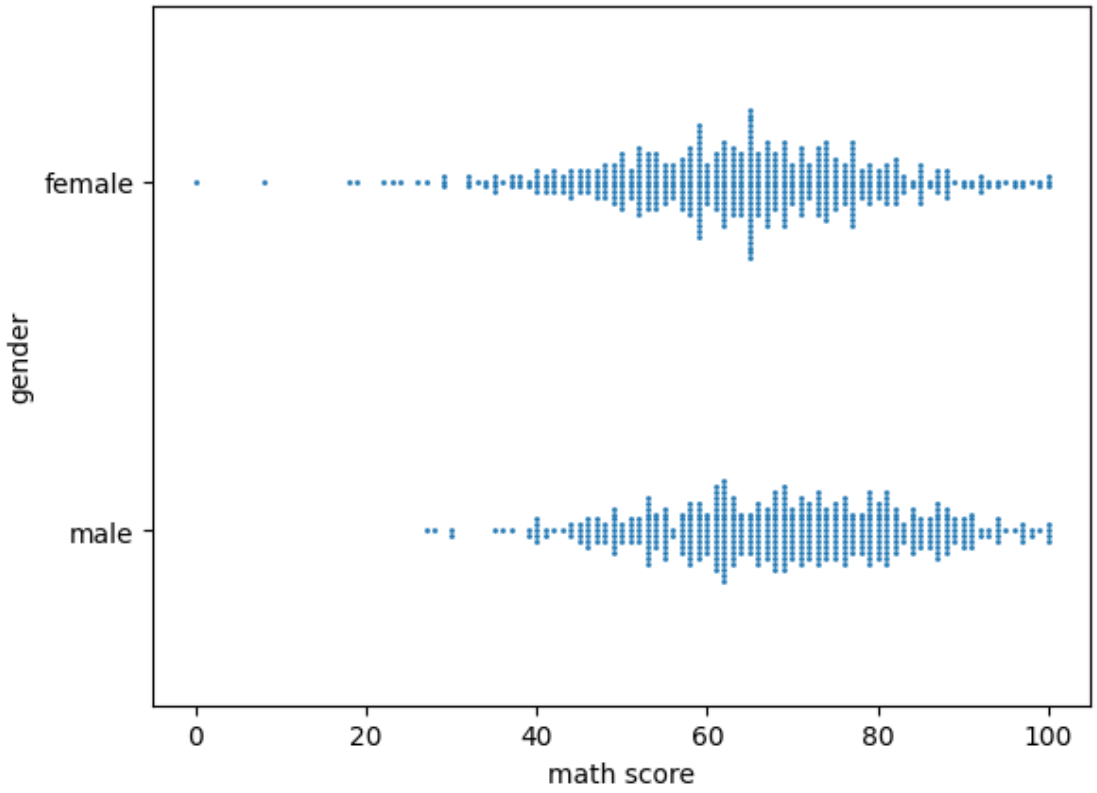
```
D:\Installed Software\Anaconda\envs\requirements\lib\site-packages\
seaborn\categorical.py:3543: UserWarning: 7.3% of the points cannot be
placed; you may want to decrease the size of the markers or use
stripplot.
  warnings.warn(msg, UserWarning)
```




```
# To remove that error here we decrease size
sns.swarmplot(x='math score',data=dff,size=2);
```

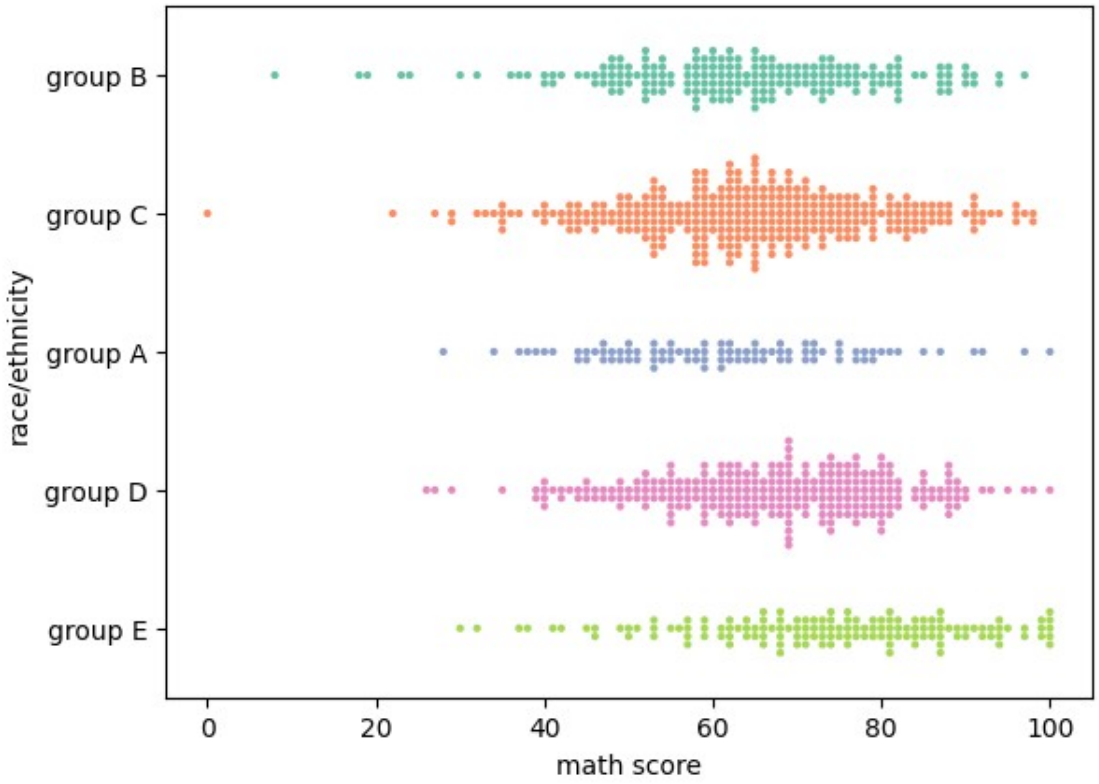



```
sns.swarmplot(x='math score',y='gender',data=dff,size=2);
```

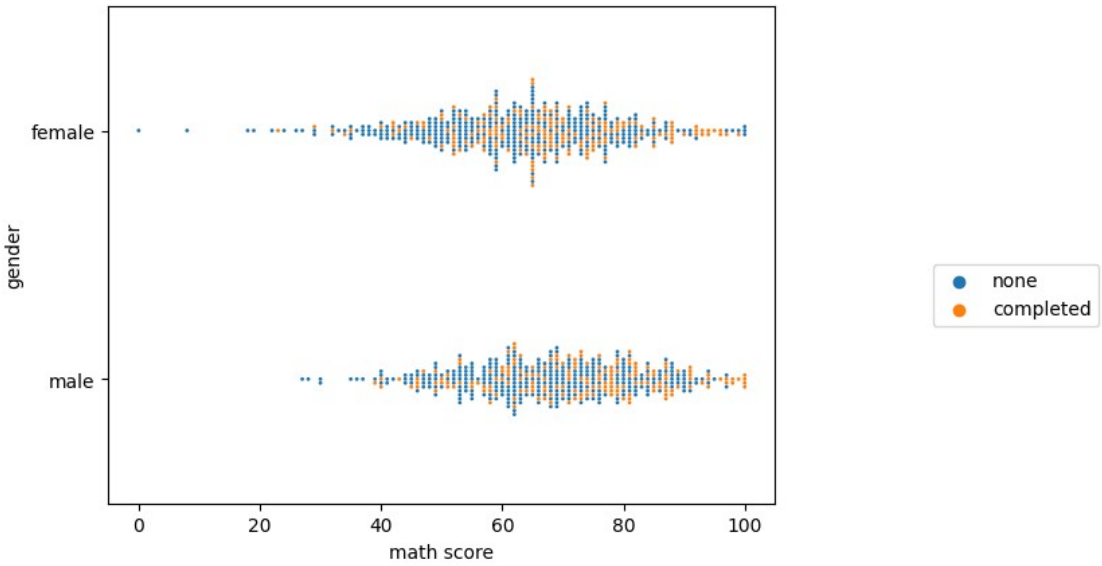



```
sns.swarmplot(x='math
score',y='race/ethnicity',data=dff,size=3,palette='Set2');
```

```
C:\Users\Chromsy\AppData\Local\Temp\ipykernel_4180\809034065.py:1:
FutureWarning: Passing `palette` without assigning `hue` is
deprecated.
  sns.swarmplot(x='math
score',y='race/ethnicity',data=dff,size=3,palette='Set2');
```



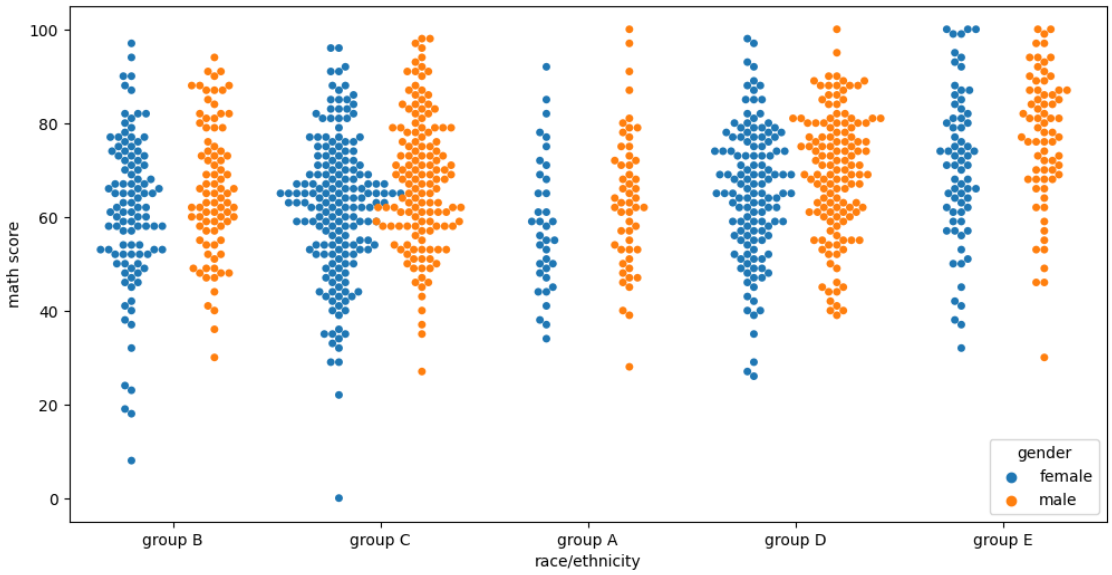

```
# Adding hue
sns.swarmplot(x='math score',y='gender',data=dff,size=2,hue='test
preparation course');
plt.legend(bbox_to_anchor=(1.5,.5));
```




```
# By add dodge it seprate hue data in different different swarms
plt.figure(figsize=(12,6))
```

```
sns.swarmplot(x='race/ethnicity',y='math
score',data=dff,hue='gender',dodge=True)
```

```
<AxesSubplot: xlabel='race/ethnicity', ylabel='math score'>
```

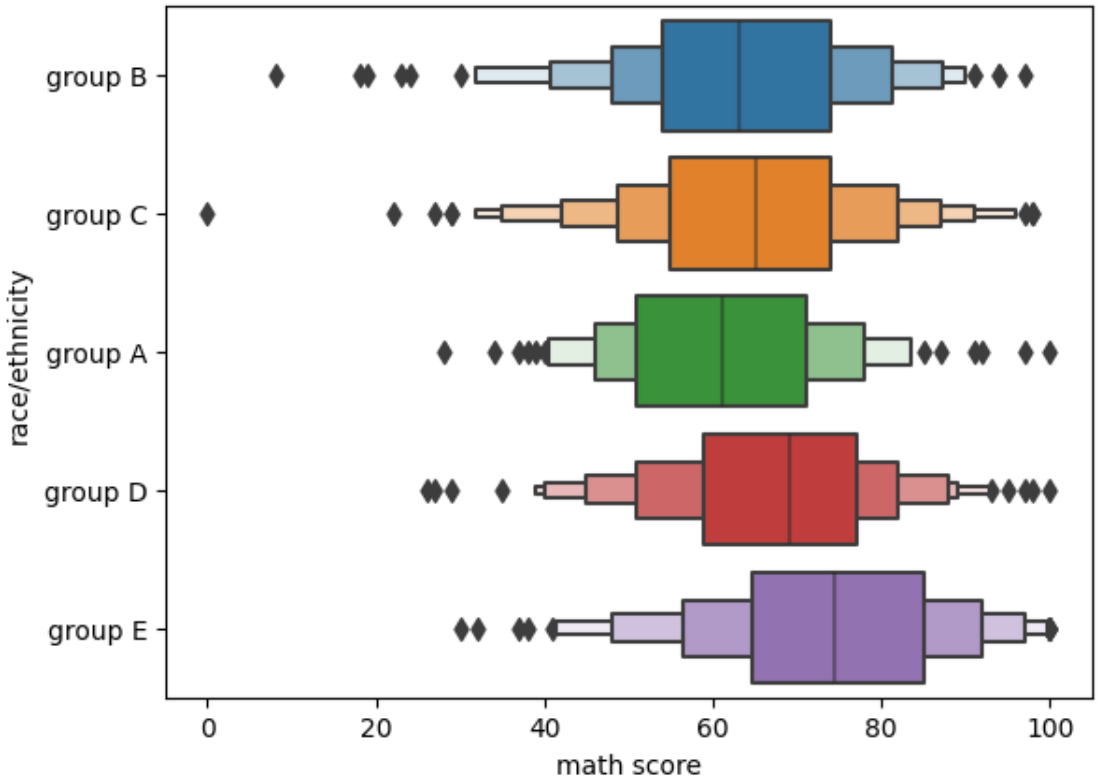


boxenplot (letter-value plot)

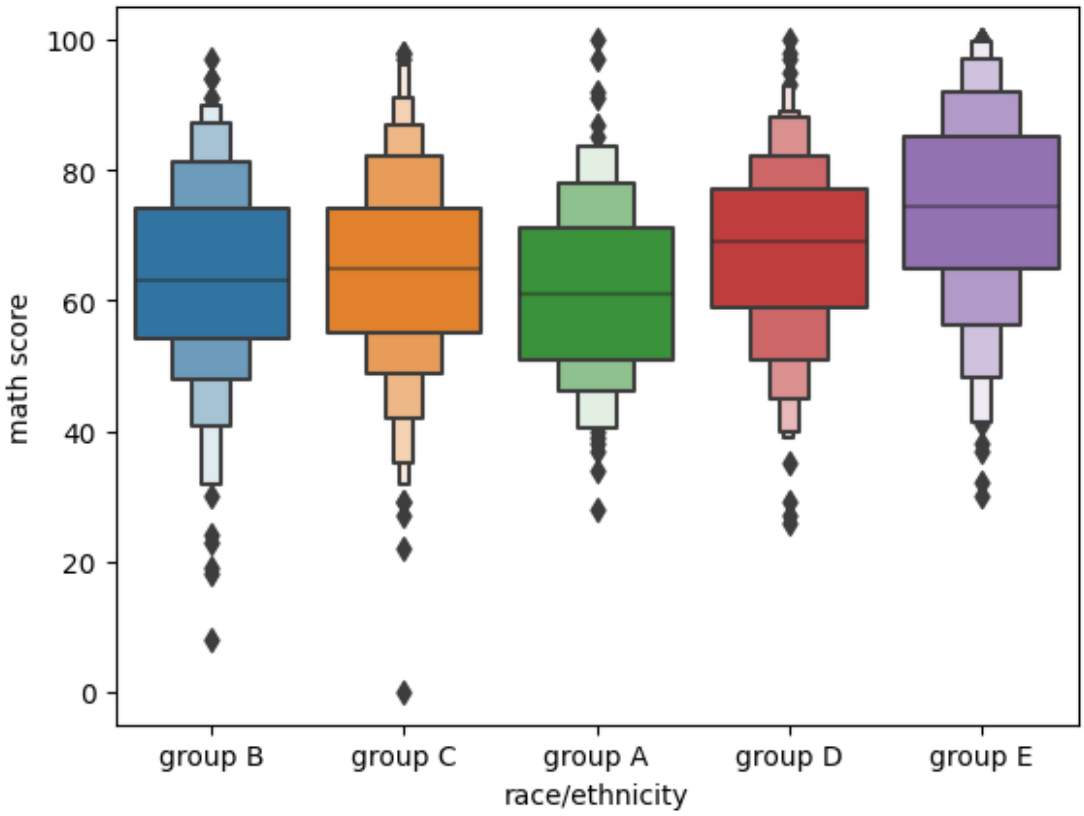
Official Paper on this plot: https://vita.had.co.nz/papers/letter-value-plot.html

This style of plot was originally named a “letter value” plot because it shows a large number of quantiles that are defined as “letter values”. It is similar to a box plot in plotting a nonparametric representation of a distribution in which all features correspond to actual observations. By plotting more quantiles, it provides more information about the shape of the distribution, particularly in the tails.

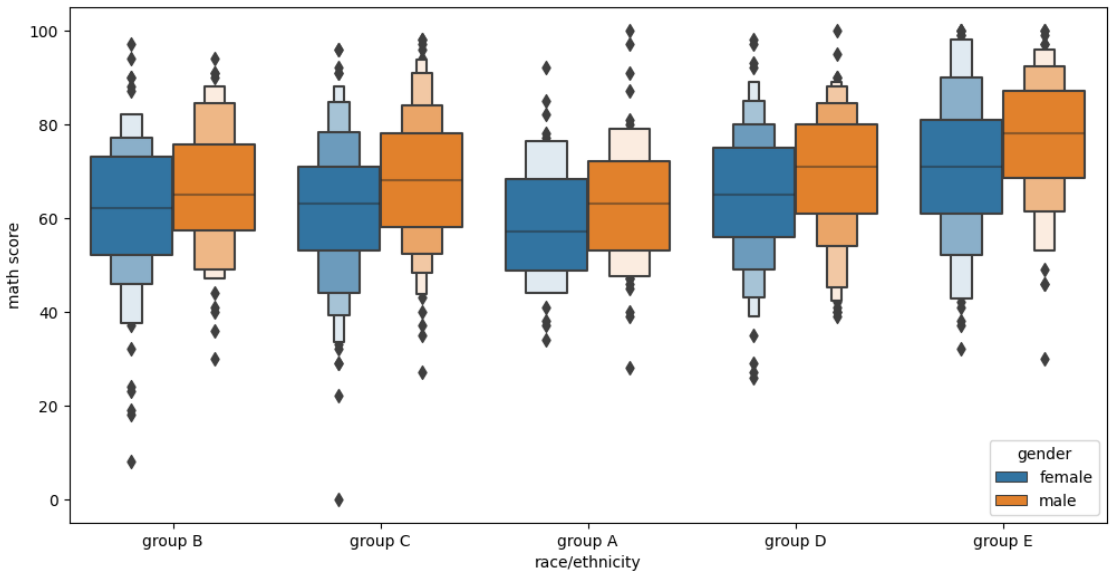
```
sns.boxenplot(x='math score',y='race/ethnicity',data=dff);
```




```
sns.boxenplot(x='race/ethnicity',y='math score',data=dff);
```




```
plt.figure(figsize=(12,6))
sns.boxenplot(x='race/ethnicity',y='math
score',data=dff,hue='gender');
```

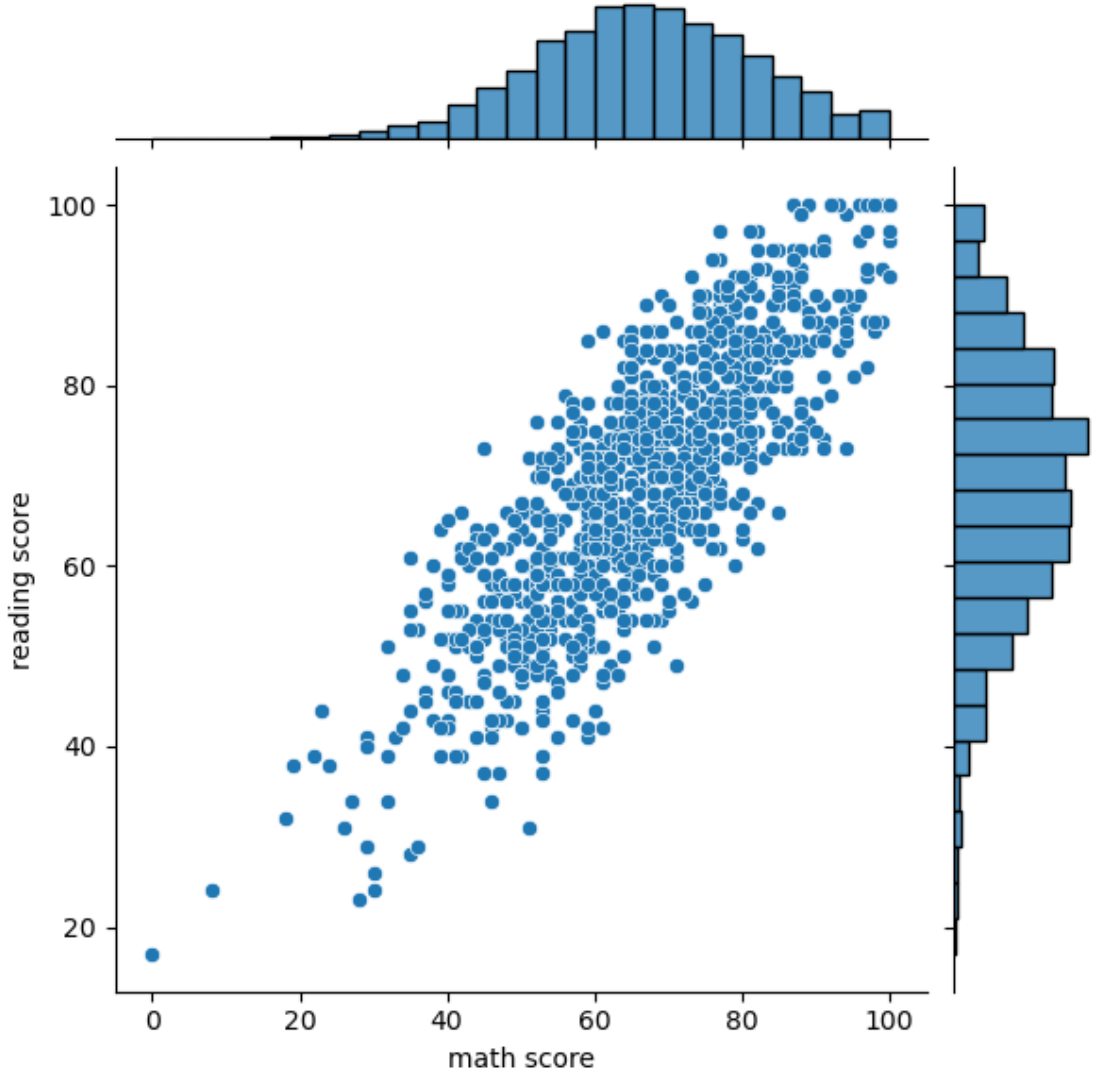


Categorical Plots - Distribution within Categories

So far we've seen how to apply a statistical estimation (like mean or count) to categories and compare them to one another. Let's now explore how to visualize the distribution within categories. We already know about distplot() which allows to view the distribution of a single feature, now we will break down that same distribution per category.

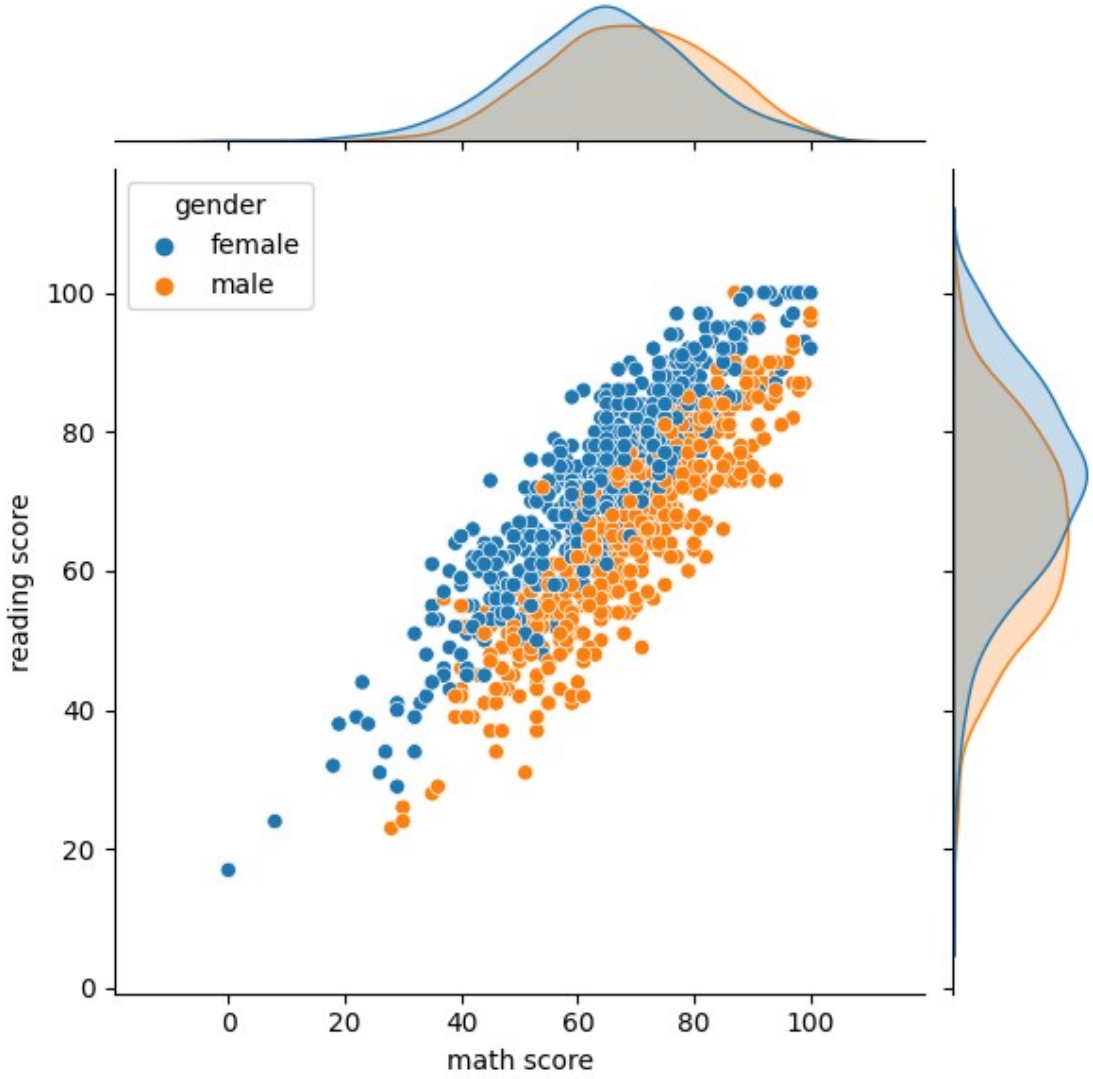
jointplot

```
sns.jointplot(data=dff,x='math score',y='reading score')
```



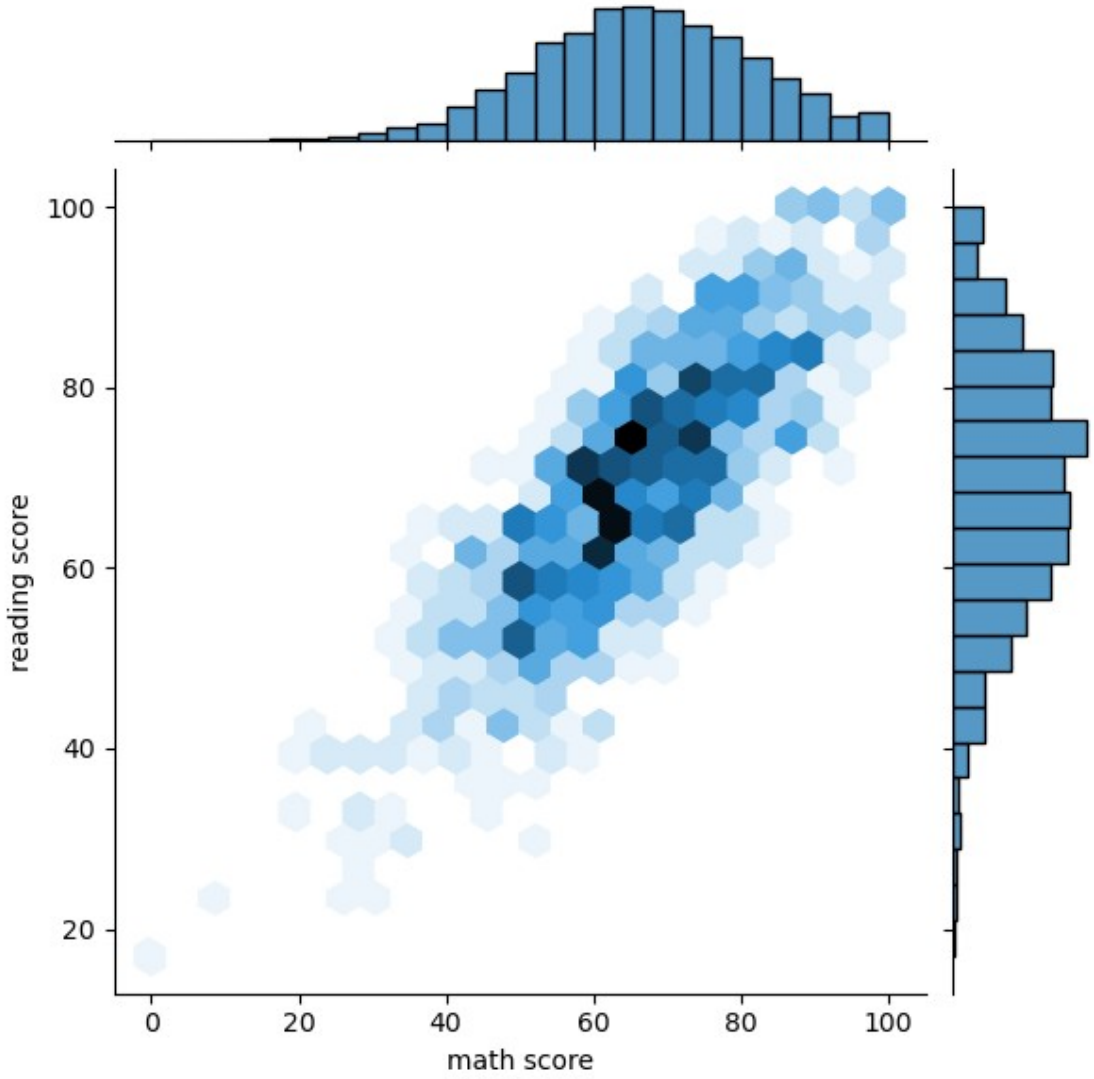

```
# We can use hue also like here it will create 2 kde color lines
sns.jointplot(data=dff,x='math score',y='reading score',hue='gender')
```

```
<seaborn.axisgrid.JointGrid at 0x1ec365514b0>
```

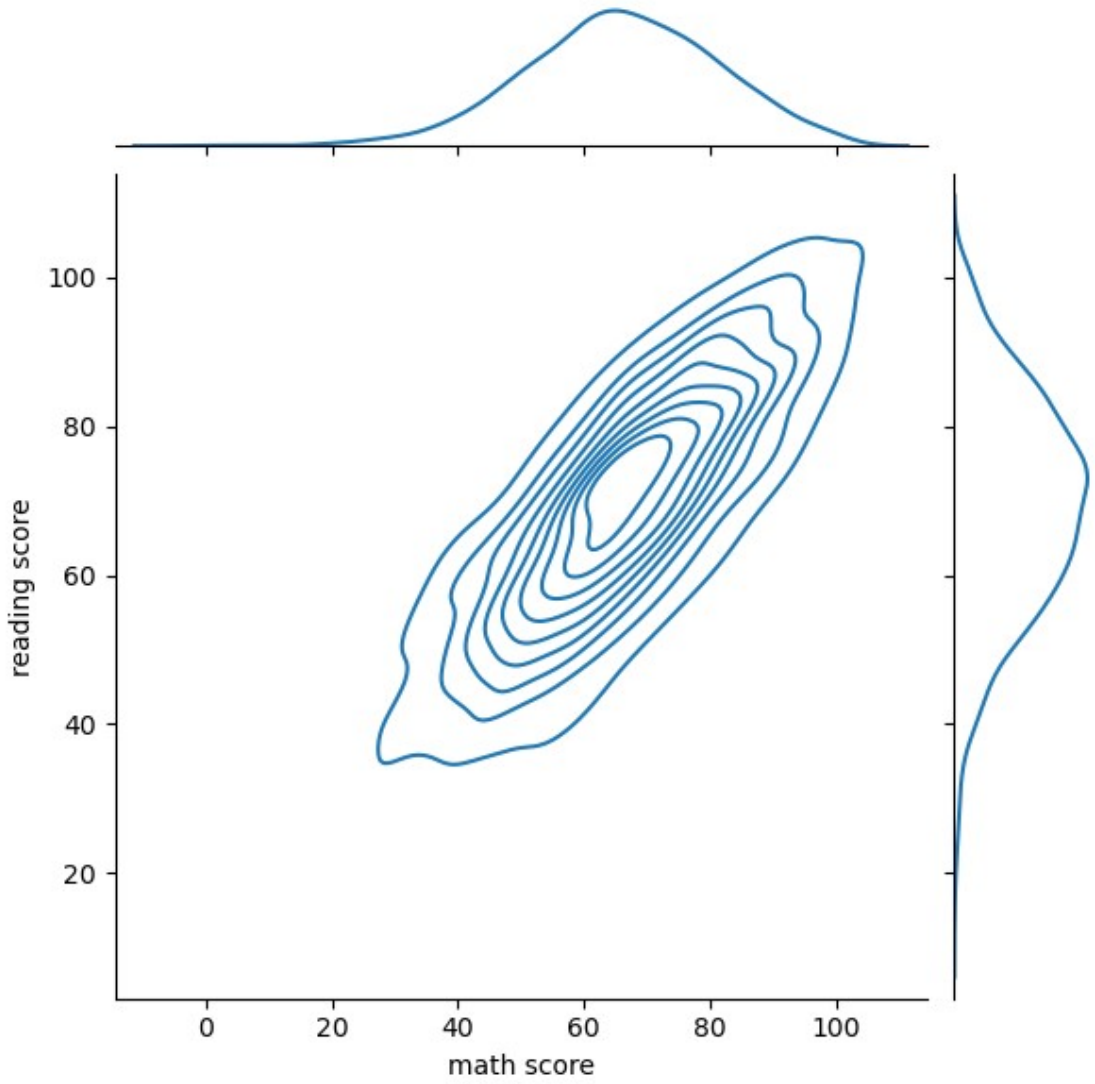



```
# Here we cant say how many points are over-laping so we use
kind='hex' for hexagone ,
# `kind` must be one of ['scatter', 'hist', 'hex', 'kde', 'reg',
'resid'], but circle was passed.`
sns.jointplot(x='math score',y='reading score',data=dff,kind='hex')
```

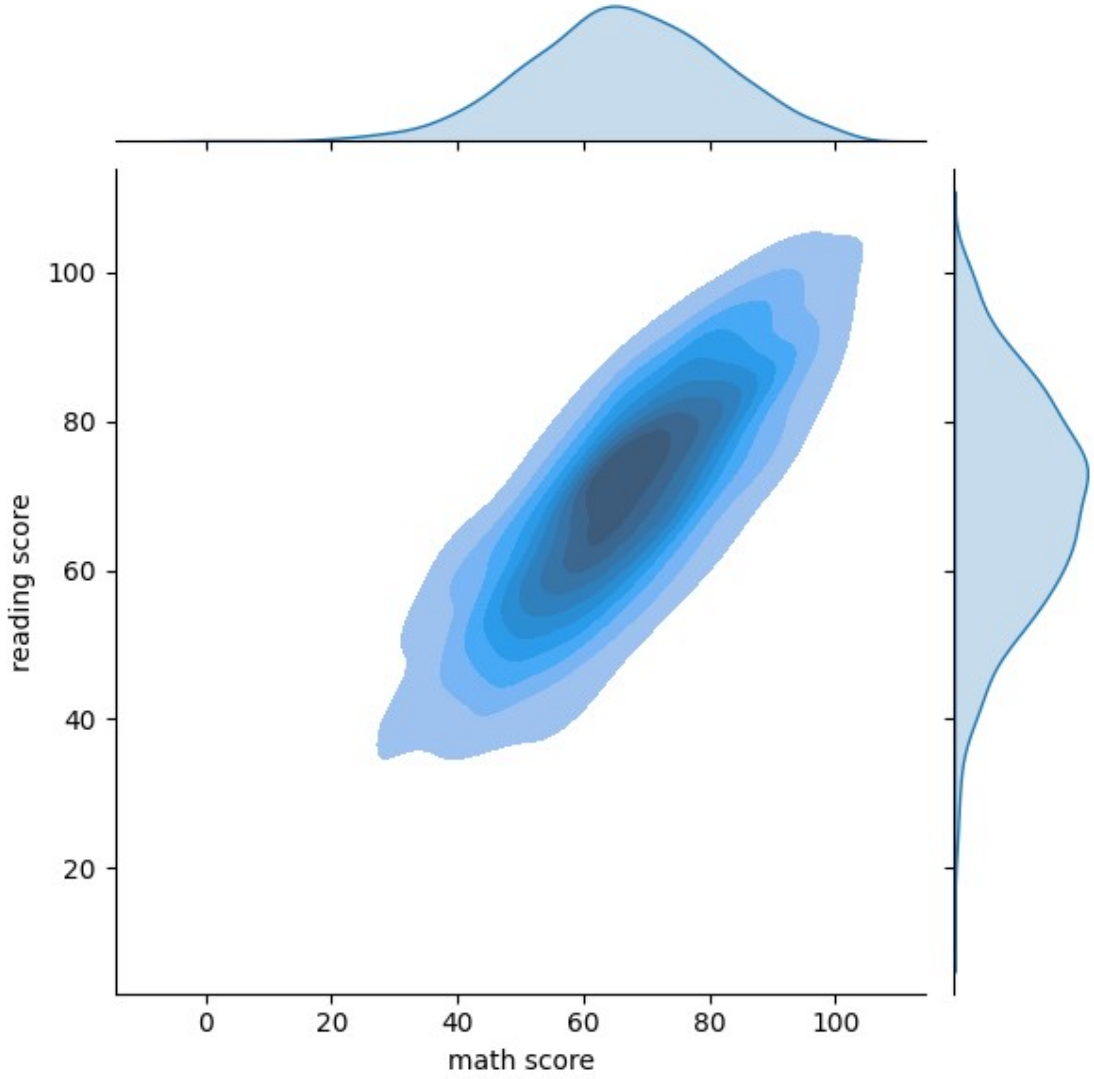
```
<seaborn.axisgrid.JointGrid at 0x1fa143fad90>
```



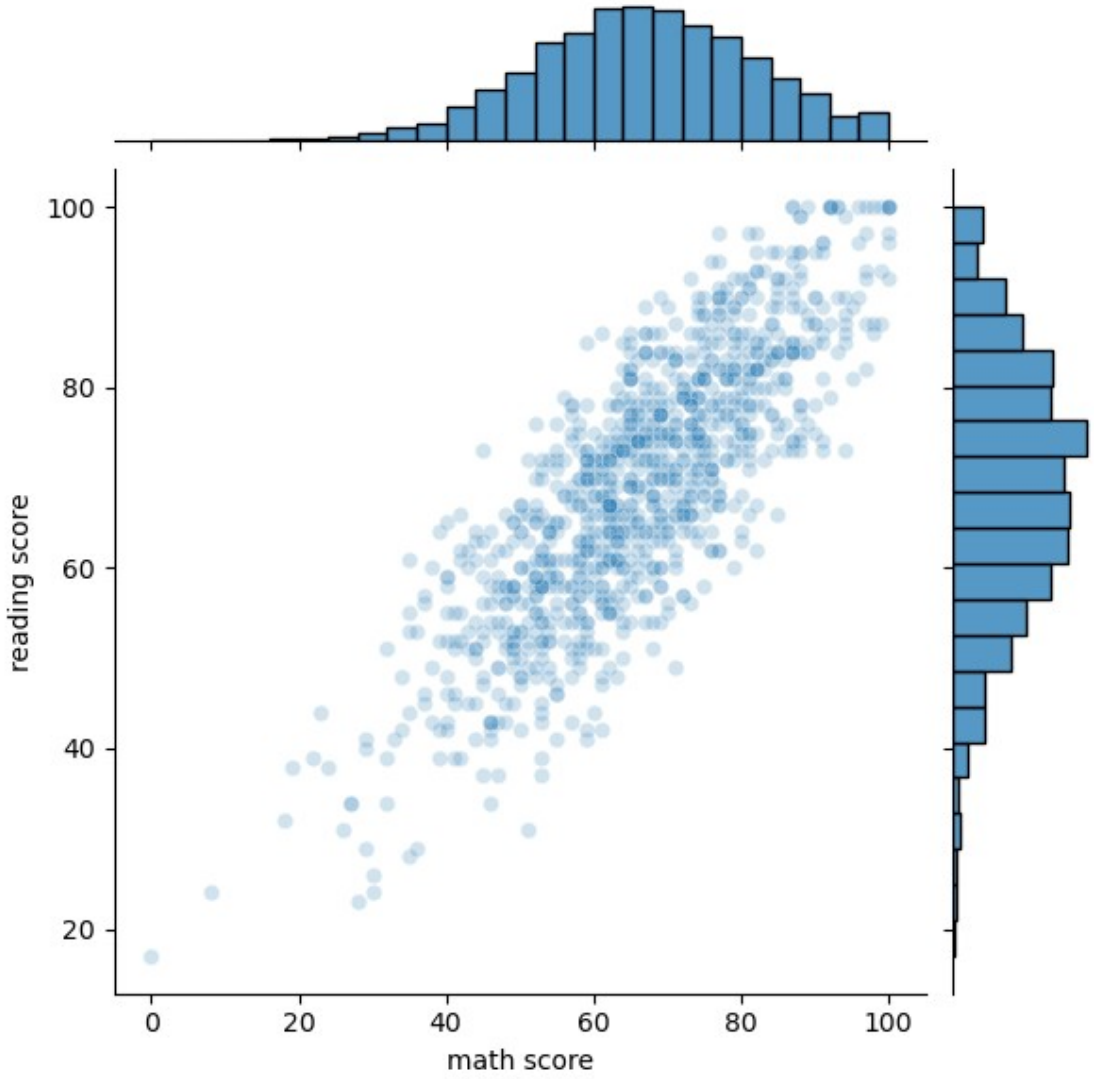

```
# Here we put kind ='kde'
sns.jointplot(x='math score',y='reading score',data=dff,kind='kde');
```




```
# Here we put kind ='kde' and here we can put shade = True but it give
warring so we use fill=True
# `shade` is now deprecated in favor of `fill`; setting `fill=True`.
sns.jointplot(x='math score',y='reading score',data=dff,kind='kde',
fill=True);
```



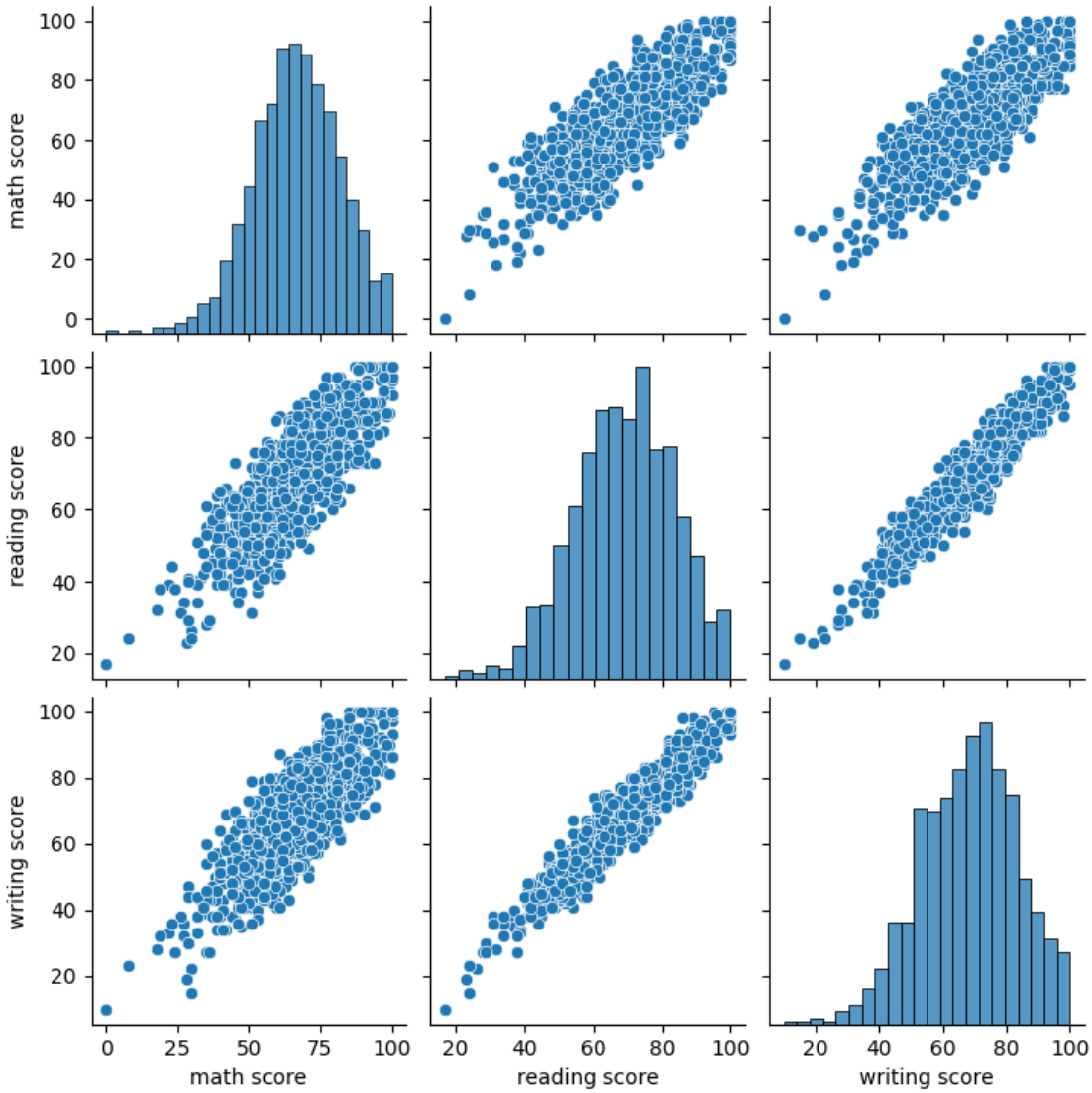

```
# Here we put kind = 'scatter' alon with alpha , 1 is darkest and 0
means low
sns.jointplot(x='math score',y='reading
score',data=dff,kind='scatter',alpha=.2);
```



pairplot

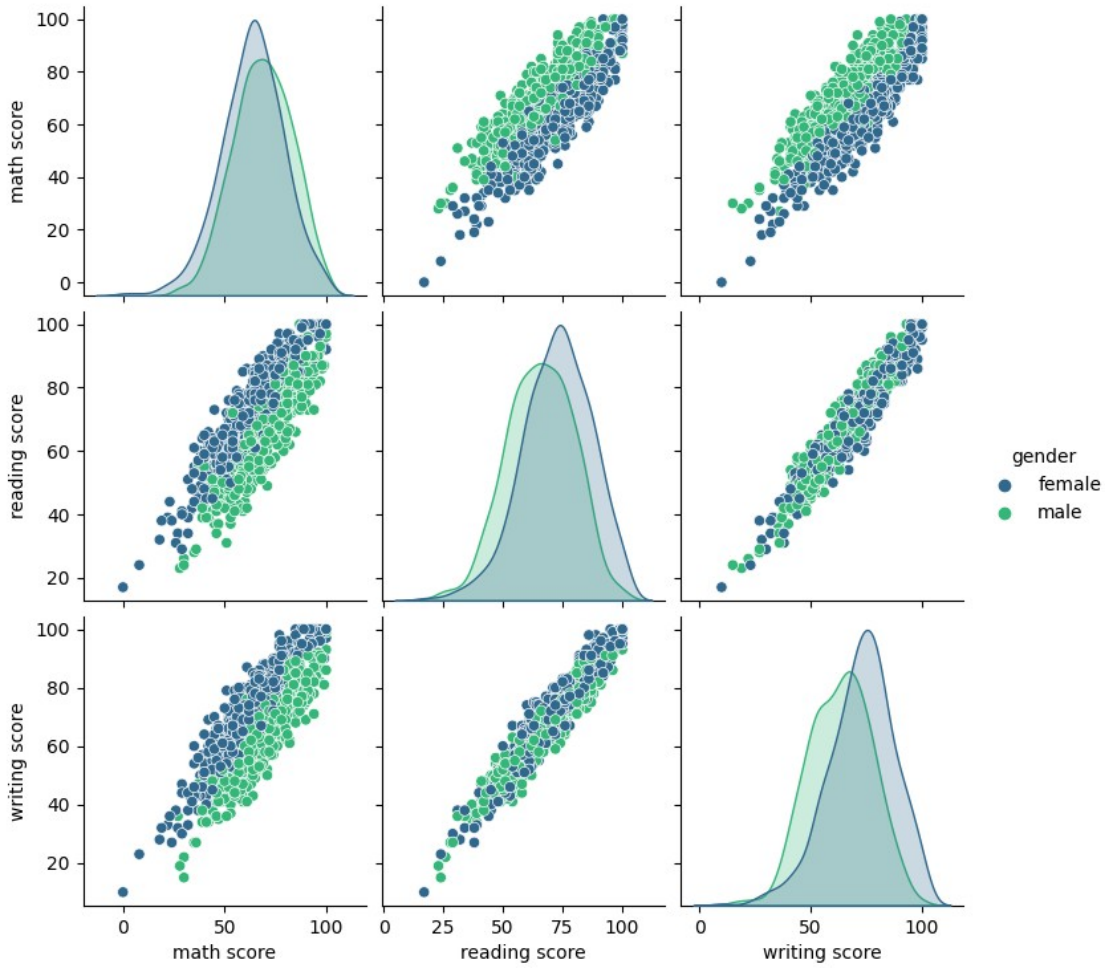
It just required data only after that it creates plots by own but plots are mirror place diagonally

```
sns.pairplot(data=dff);
```

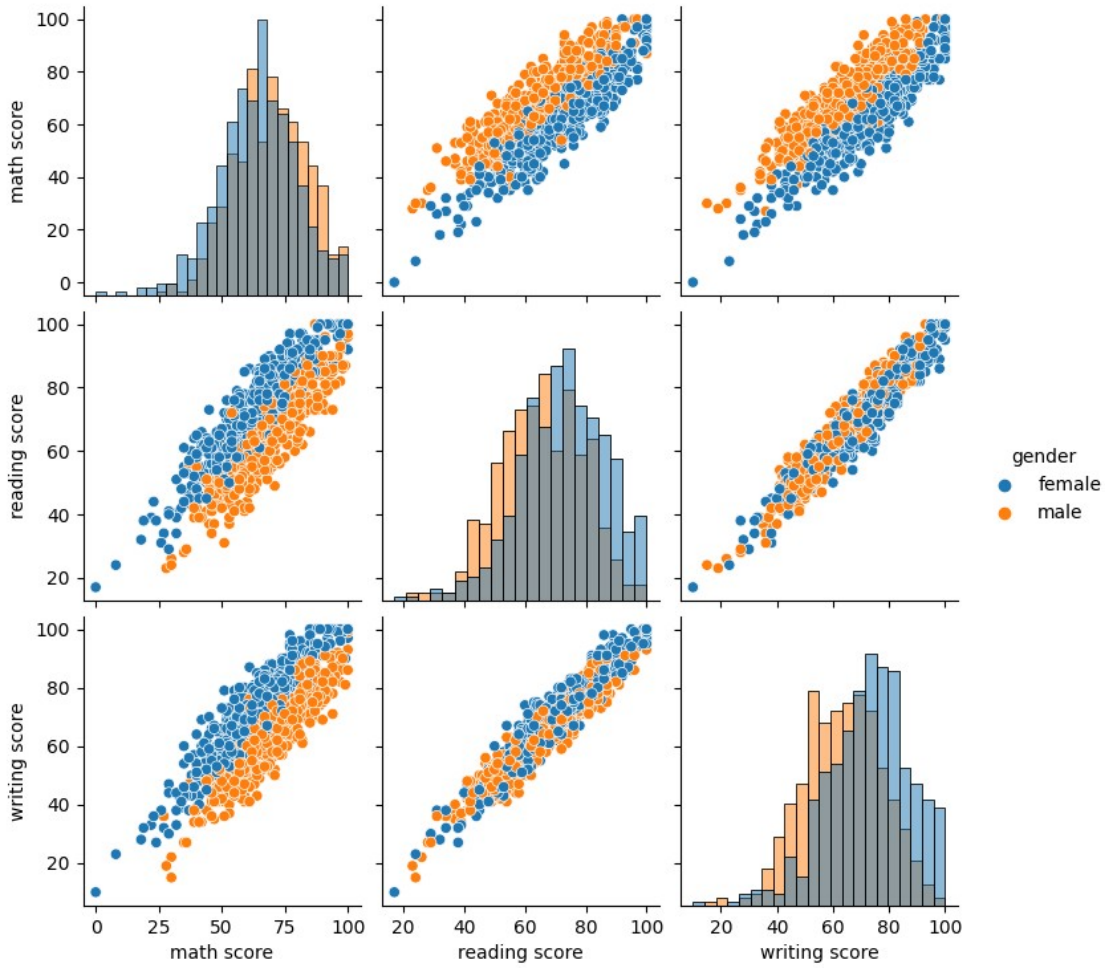



```
# We can also add hue in this and we use palette to change color
sns.pairplot(dff,hue='gender',palette='viridis')
```

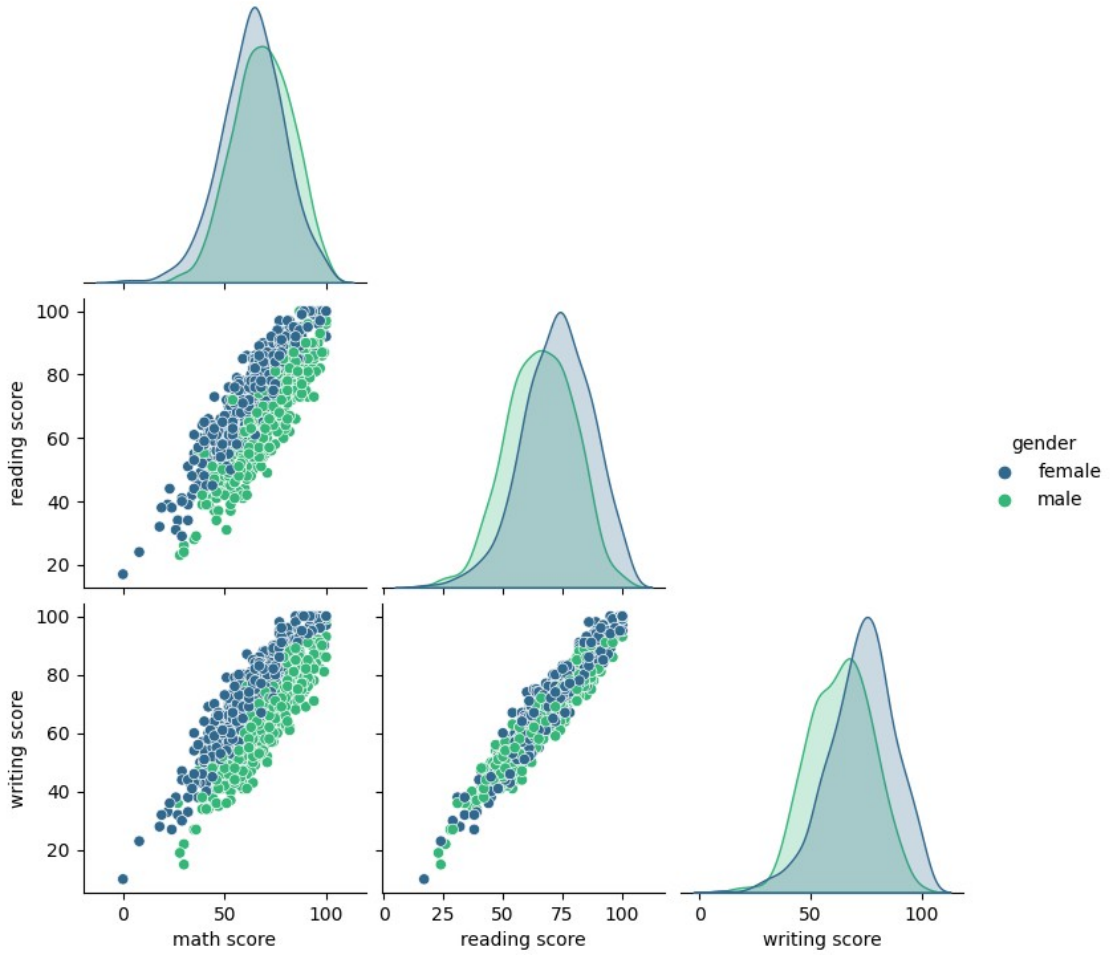
```
<seaborn.axisgrid.PairGrid at 0x1ec36423a00>
```



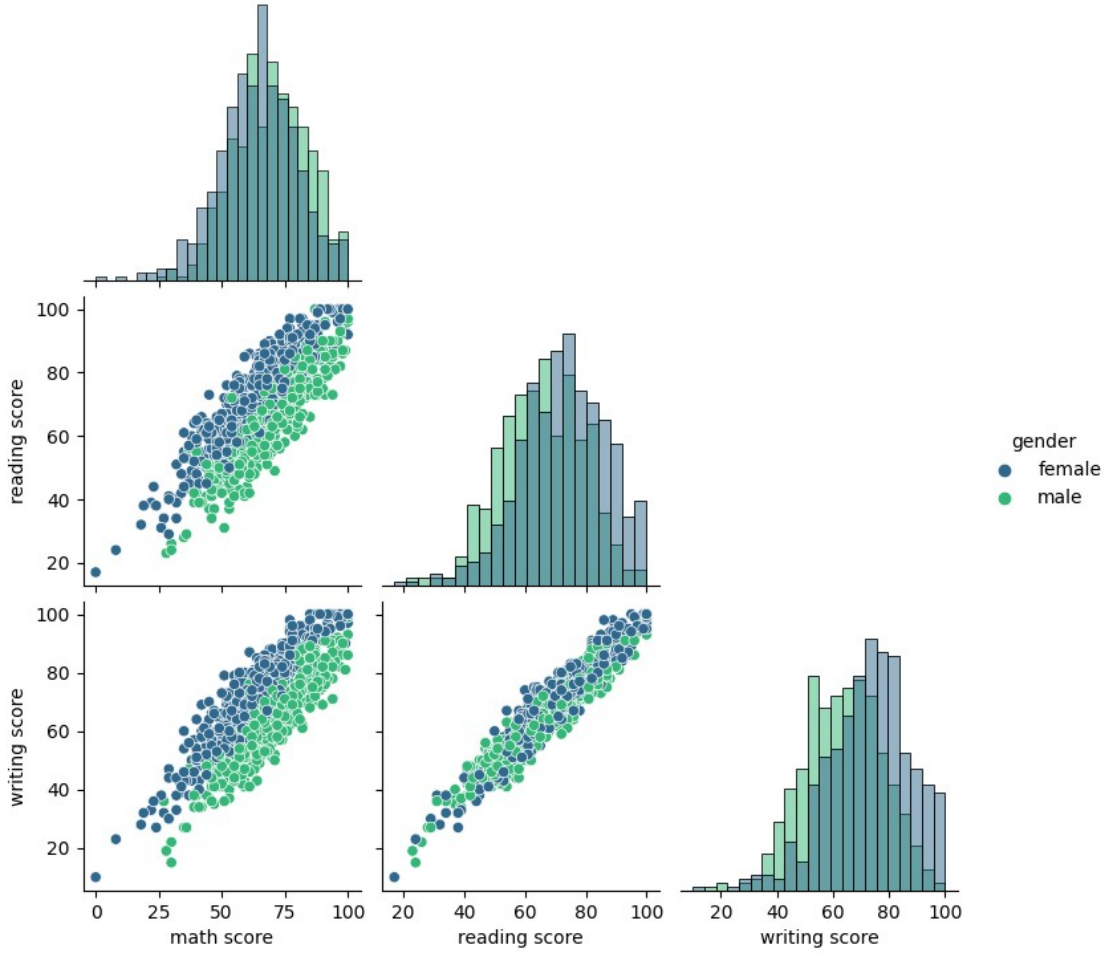

```
# If we want to change this diagonal kde plot to histograph we can use diag_kind='hist'
sns.pairplot(data=dff,hue='gender',diag_kind='hist');
```




```
# IF we want repeated plots we can use corners=True
sns.pairplot(data=dff,hue='gender',palette='viridis',corner=True);
```



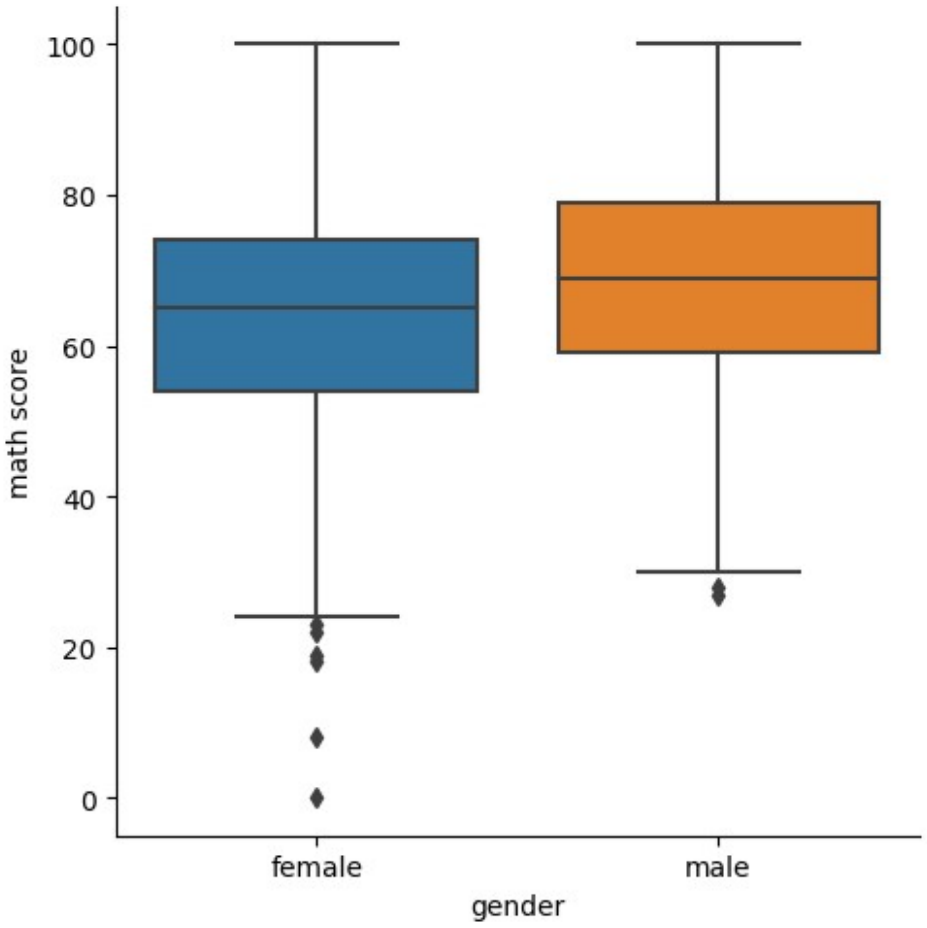

```
sns.pairplot(dff,hue='gender',palette='viridis',diag_kind='hist',
corner=True);
```



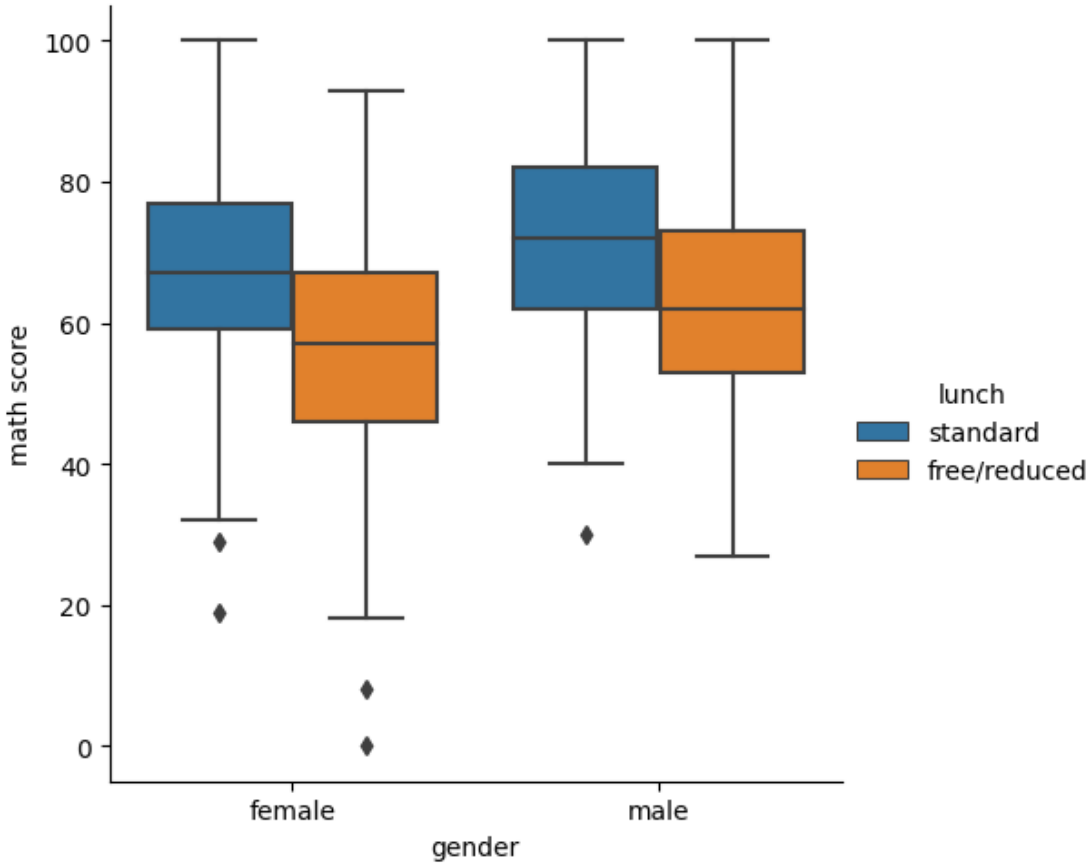
Grids

catplot()

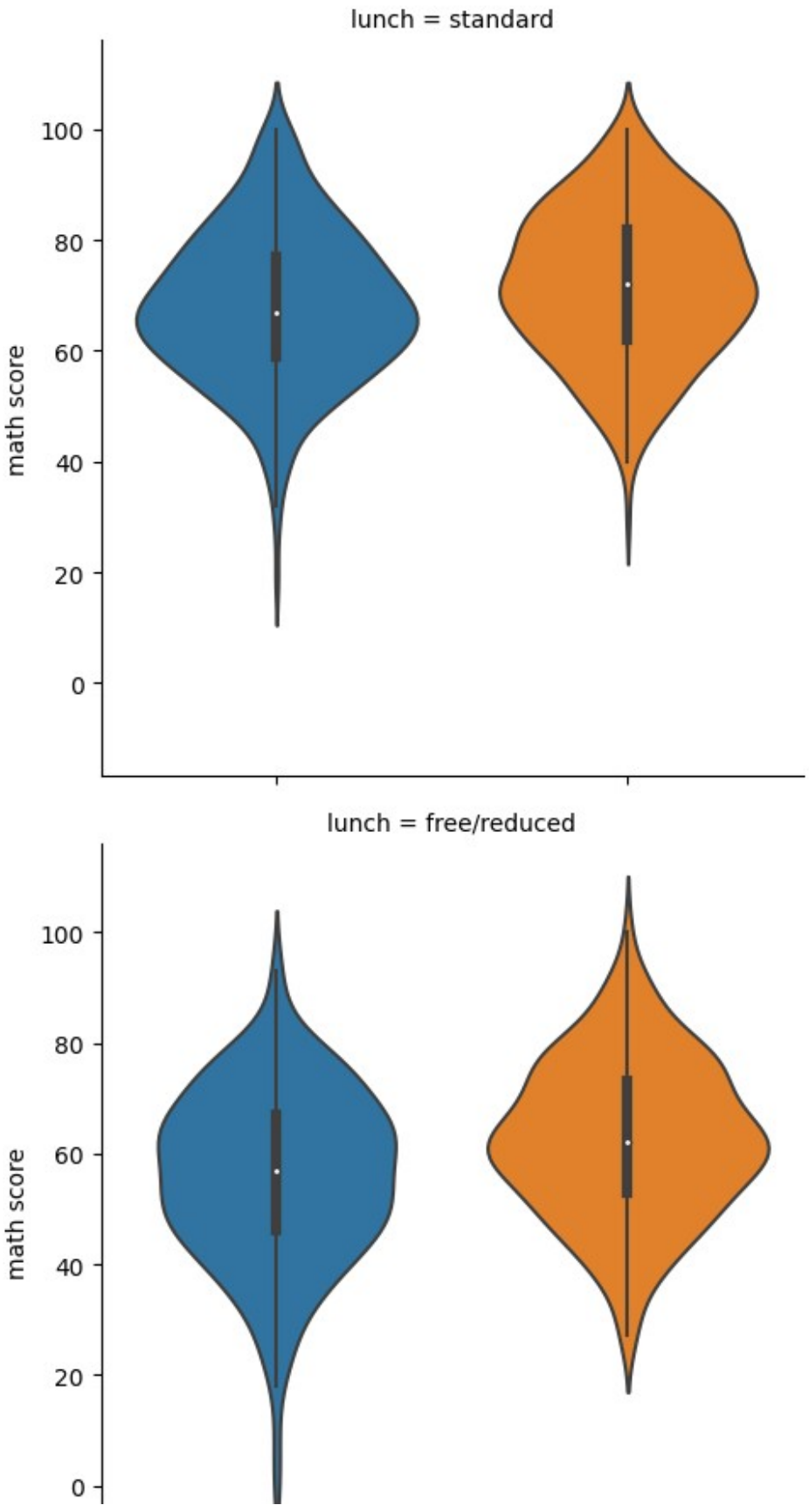
```
# Kind Options are: “point”, “bar”, “strip”, “swarm”, “box”, “violin”, or “boxen”
sns.catplot(x='gender',y='math score',data=dff,kind='box');
```



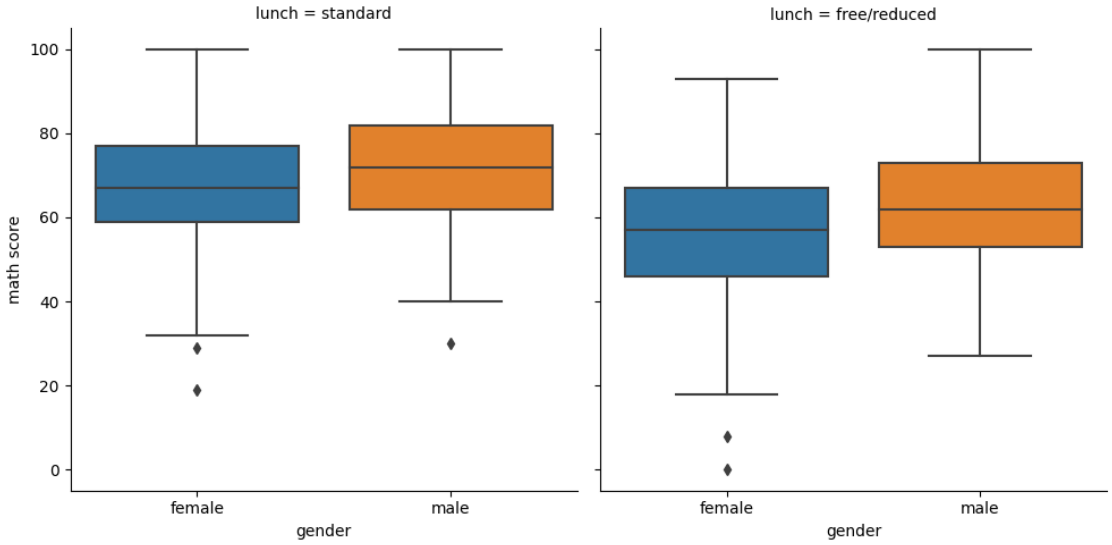

```
# Using hue
sns.catplot(x='gender',y='math
score',data=dff,kind='box',hue='lunch');
```



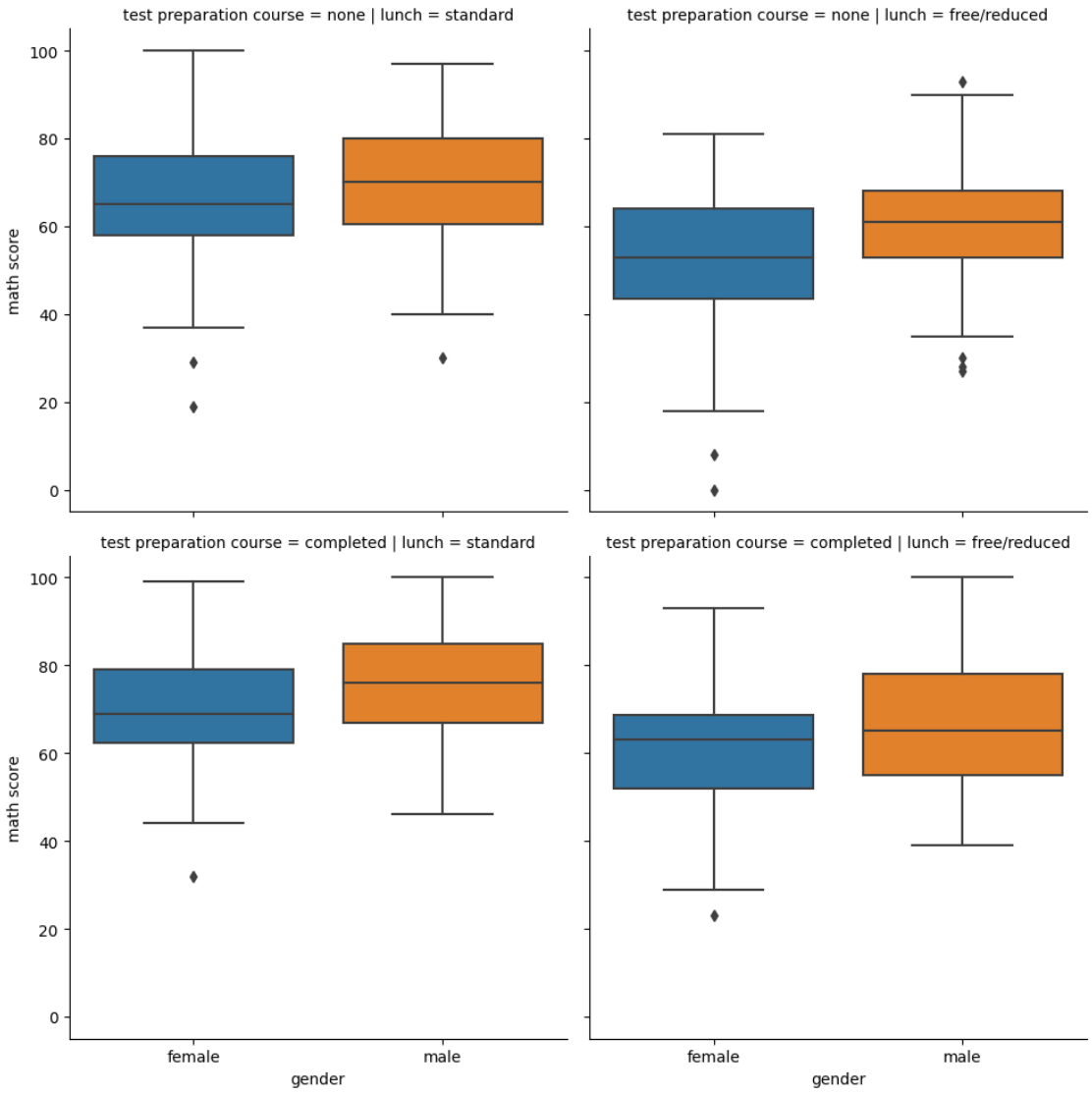

```
# Like we have hue but hue show 2 plot at one but if we wanna see in
differnt plots
                                                    # we can do that in
pairplot by row and colums
sns.catplot(data=dff,x='gender' ,y='math score', kind='violin', row
='lunch');
```


lunch = standard
math score
100
80
60
40
20
0
lunch = free/reduced
math score
100
80
60
40
20
0

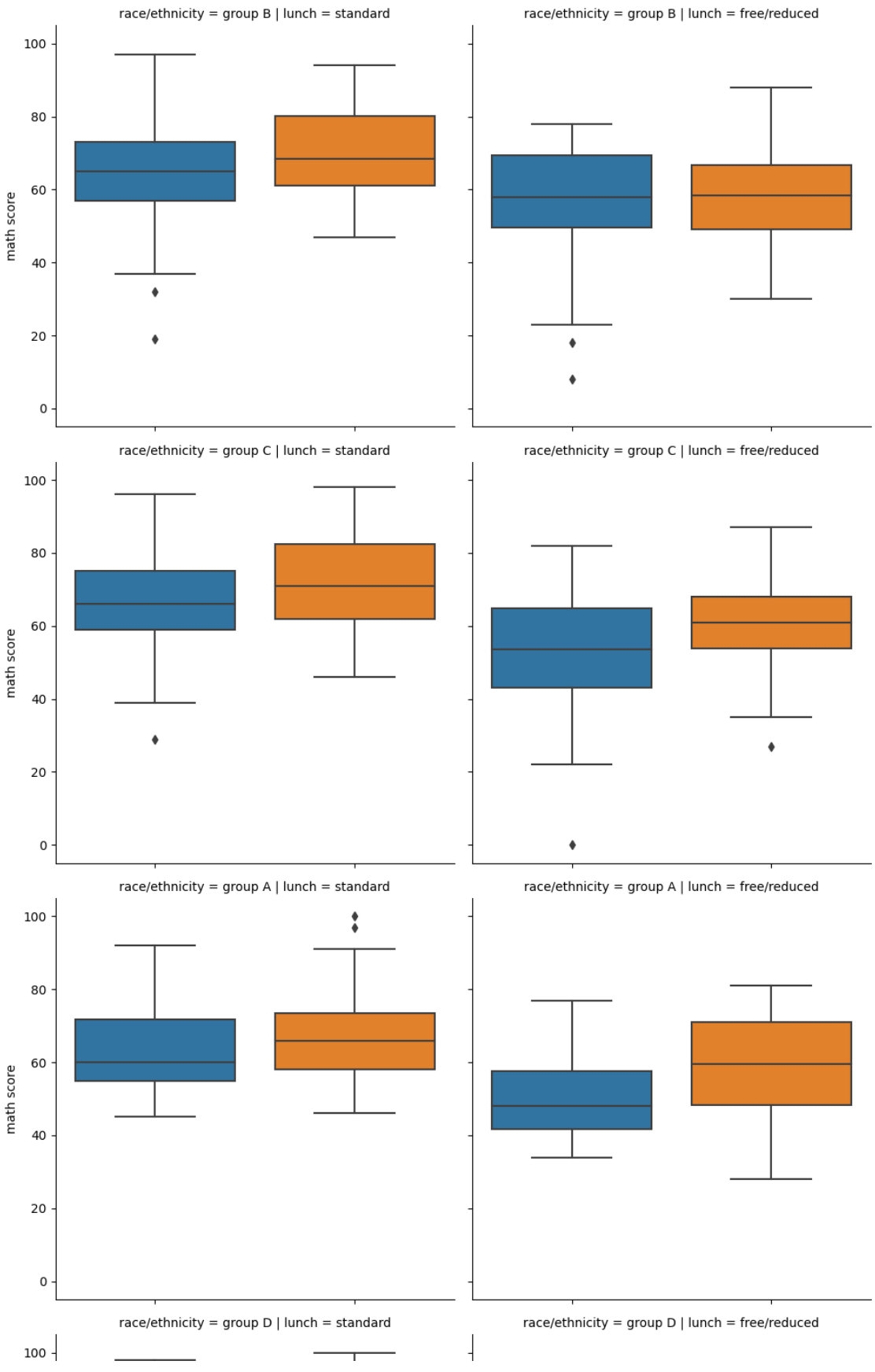
```
# Same we can arrange column wise
sns.catplot(data=dff,x='gender' ,y='math score', kind='box', col
='lunch');
```




```
# Same we us both row and Col both at once
sns.catplot(data=dff,x='gender' ,y='math score', kind='box', col
='lunch', row='test preparation course');
```




```
# we can take row and column having more than 2 values that will
create more rows or columns
# Here we take race/ethnicity which have 5 entries so we will have 5
rows as well
sns.catplot(data=dff,x='gender' ,y='math score', kind='box', col
='lunch', row='race/ethnicity');
```


race/ethnicity = group B | lunch = standard
race/ethnicity = group B | lunch = free/reduced
race/ethnicity = group C | lunch = standard
race/ethnicity = group C | lunch = free/reduced
race/ethnicity = group A | lunch = standard
race/ethnicity = group A | lunch = free/reduced
race/ethnicity = group D | lunch = standard
race/ethnicity = group D | lunch = free/reduced
math score
0
20
40
60
80
100

```
dff
```

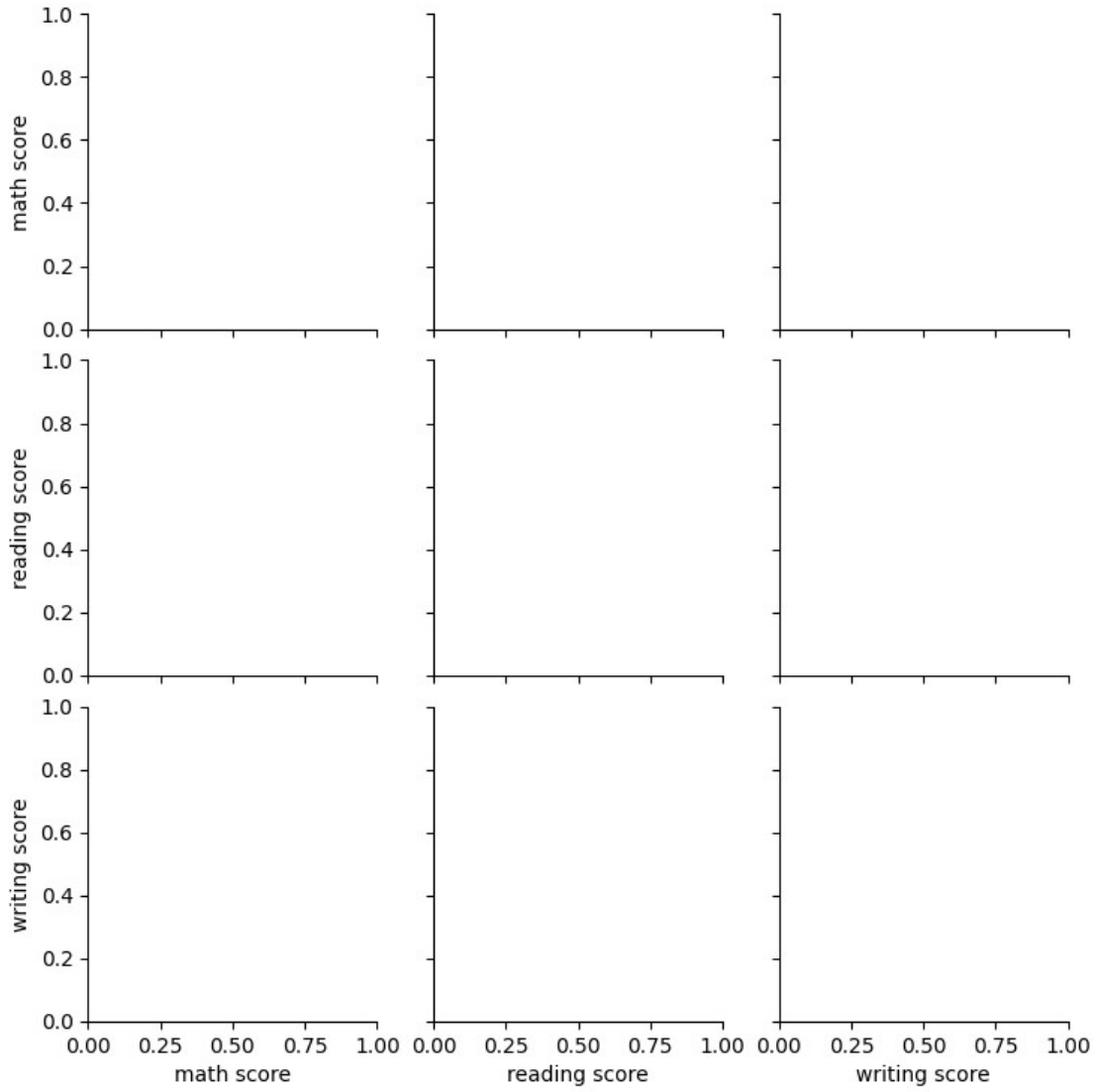
	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75
..	...	...	...	...	...	...	...	...
995	female	group E	master's degree	standard	completed	88	99	95
996	male	group C	high school	free/reduced	none	62	55	55
997	female	group C	high school	free/reduced	completed	59	71	65
998	female	group D	some college	standard	completed	68	78	77
999	female	group D	some college	free/reduced	none	77	86	86

```
999                  none          77          86          86

[1000 rows x 8 columns]
```

Pairgrid

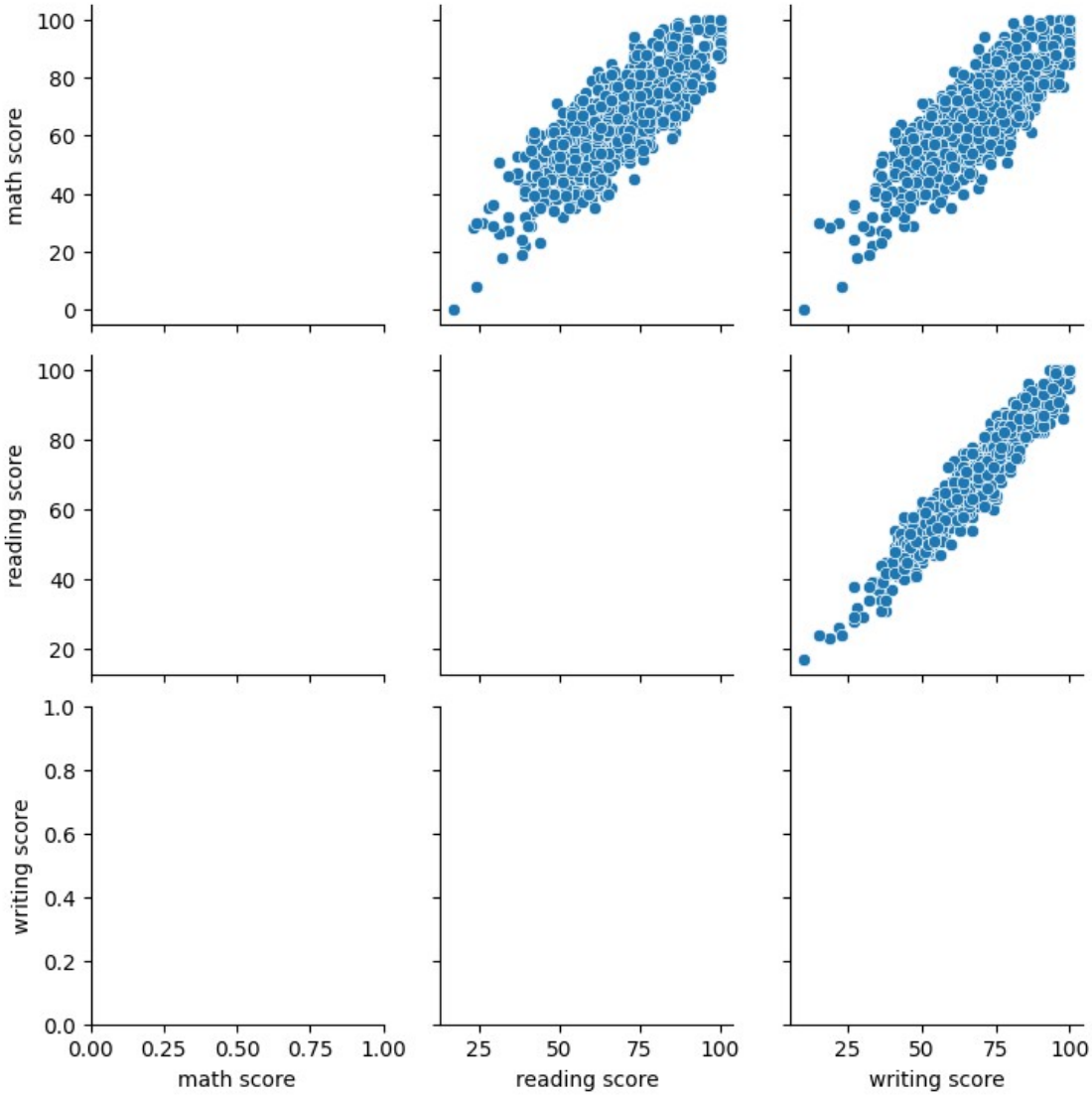
Grid that pairplot is built on top of, allows for heavy customization of the pairplot seen earlier.

```
# Pairgrid allow us to get pairplot as per our will, here we get blank
plot we have to put values
g= sns.PairGrid(data=dff)
```

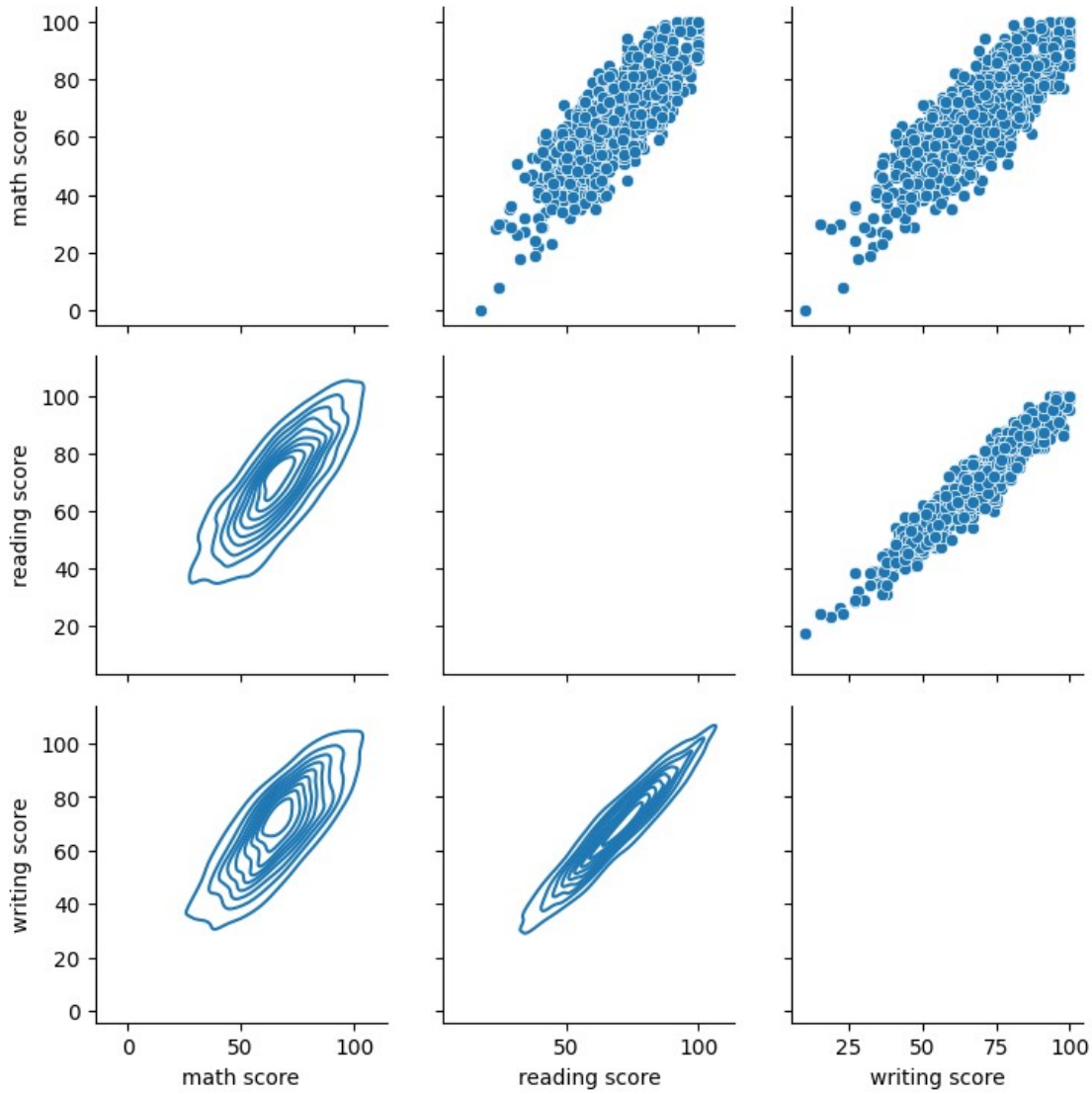



```
# Adding scatterplot to upper diagonal
g= sns.PairGrid(data=dff)
```

```
g=g.map_upper(sns.scatterplot)
```



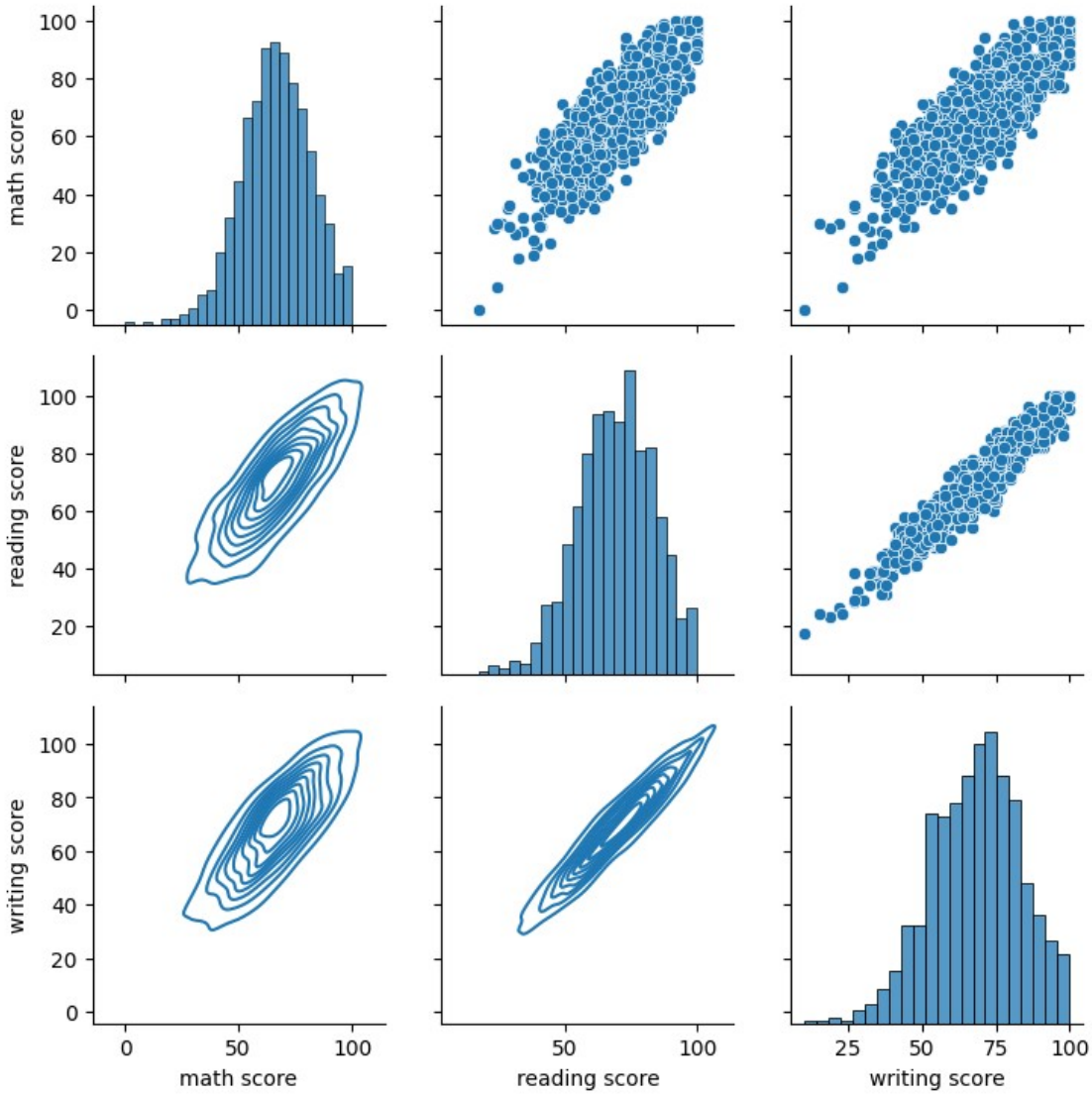

```
# Adding kdeplot to lower diagonal

g= sns.PairGrid(data=dff)

g=g.map_upper(sns.scatterplot)

g=g.map_lower(sns.kdeplot)
```




```
# Adding histplot on diagonal
g= sns.PairGrid(data=dff)

g=g.map_upper(sns.scatterplot)

g=g.map_lower(sns.kdeplot)

g=g.map_diag(sns.histplot)
```

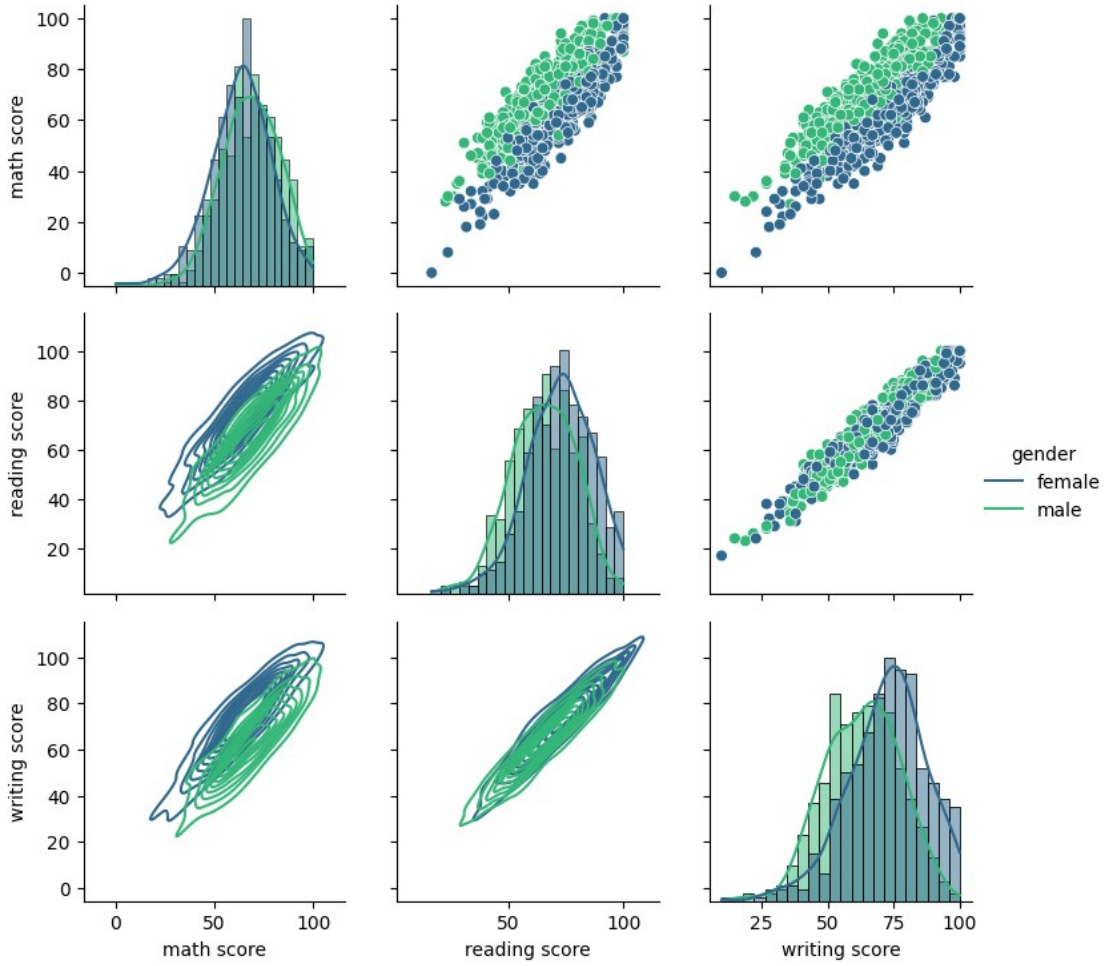
```
D:\Installed Software\Anaconda\envs\requirements\lib\site-packages\
seaborn\distributions.py:1185: UserWarning: The following kwargs were
not used by contour: 'lw'
  cset = contour_func(
D:\Installed Software\Anaconda\envs\requirements\lib\site-packages\
seaborn\distributions.py:1185: UserWarning: The following kwargs were
not used by contour: 'lw'
```

```
  cset = contour_func(
D:\Installed Software\Anaconda\envs\requirements\lib\site-packages\
seaborn\distributions.py:1185: UserWarning: The following kwargs were
not used by contour: 'lw'
  cset = contour_func(
```



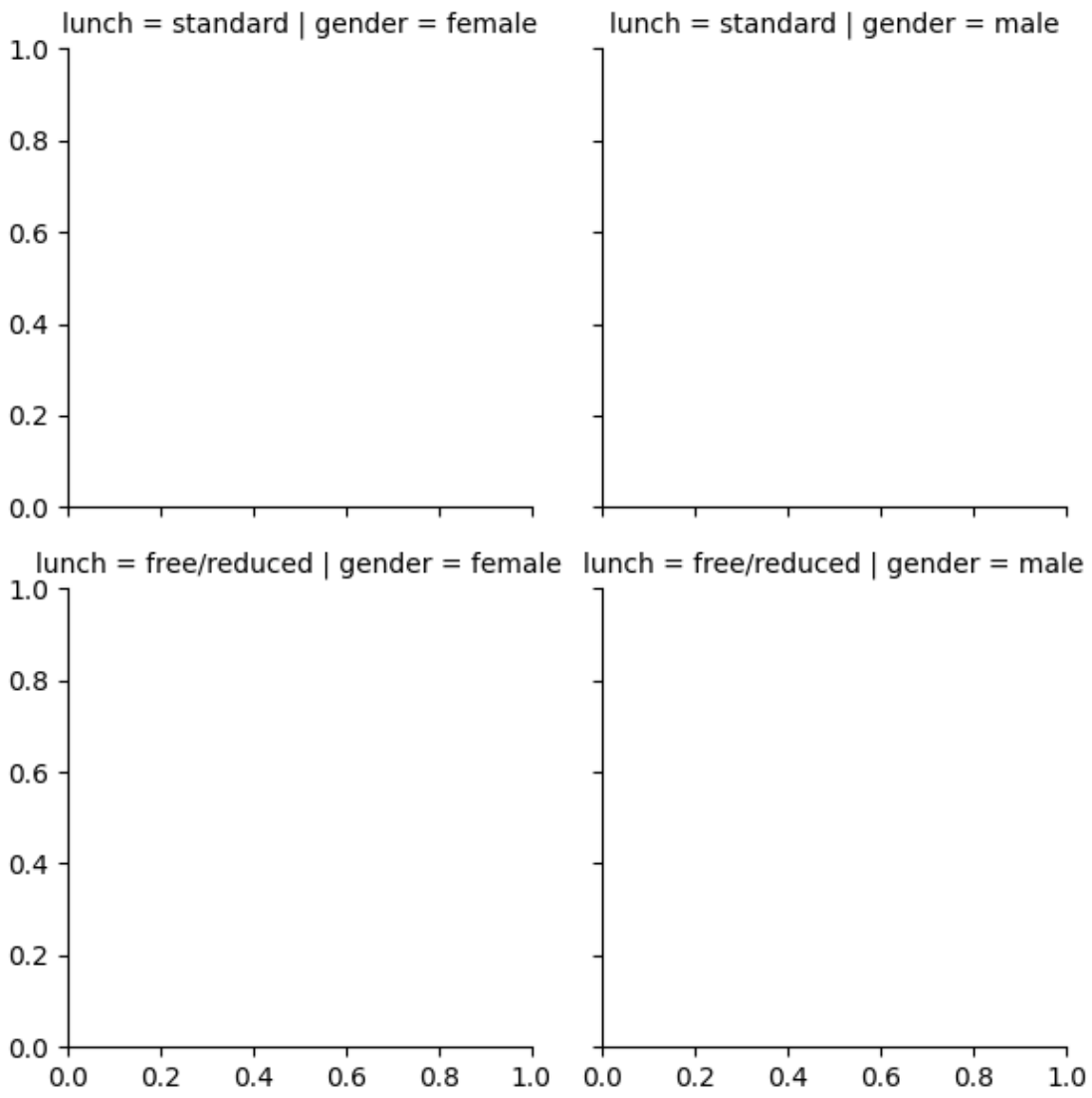

```
# Adding hue as well and adding other feature
g= sns.PairGrid(data=dff,  hue="gender",
palette="viridis",hue_kws={"marker": ["o", "+"]})

g=g.map_upper(sns.scatterplot ,linewidths=1, edgecolor="w", s=40)

g=g.map_lower(sns.kdeplot)

g=g.map_diag(sns.histplot,kde=True)
```

```
g = g.add_legend();

# Safely ignore the warning, its telling you it didn't use the marker
for kde plot
```



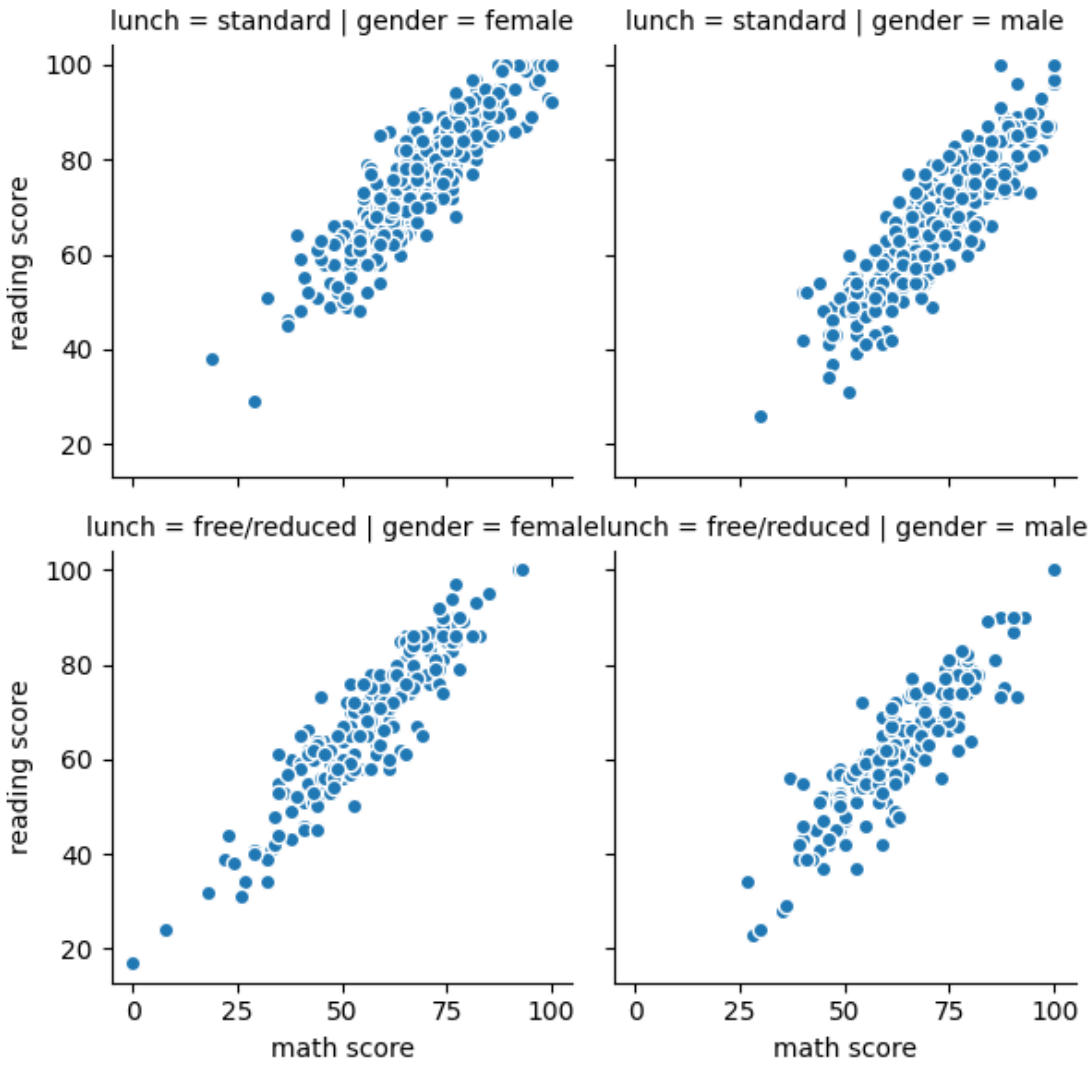
FacetGrid

```
sns.FacetGrid(data=dff,col='gender',row='lunch');
```



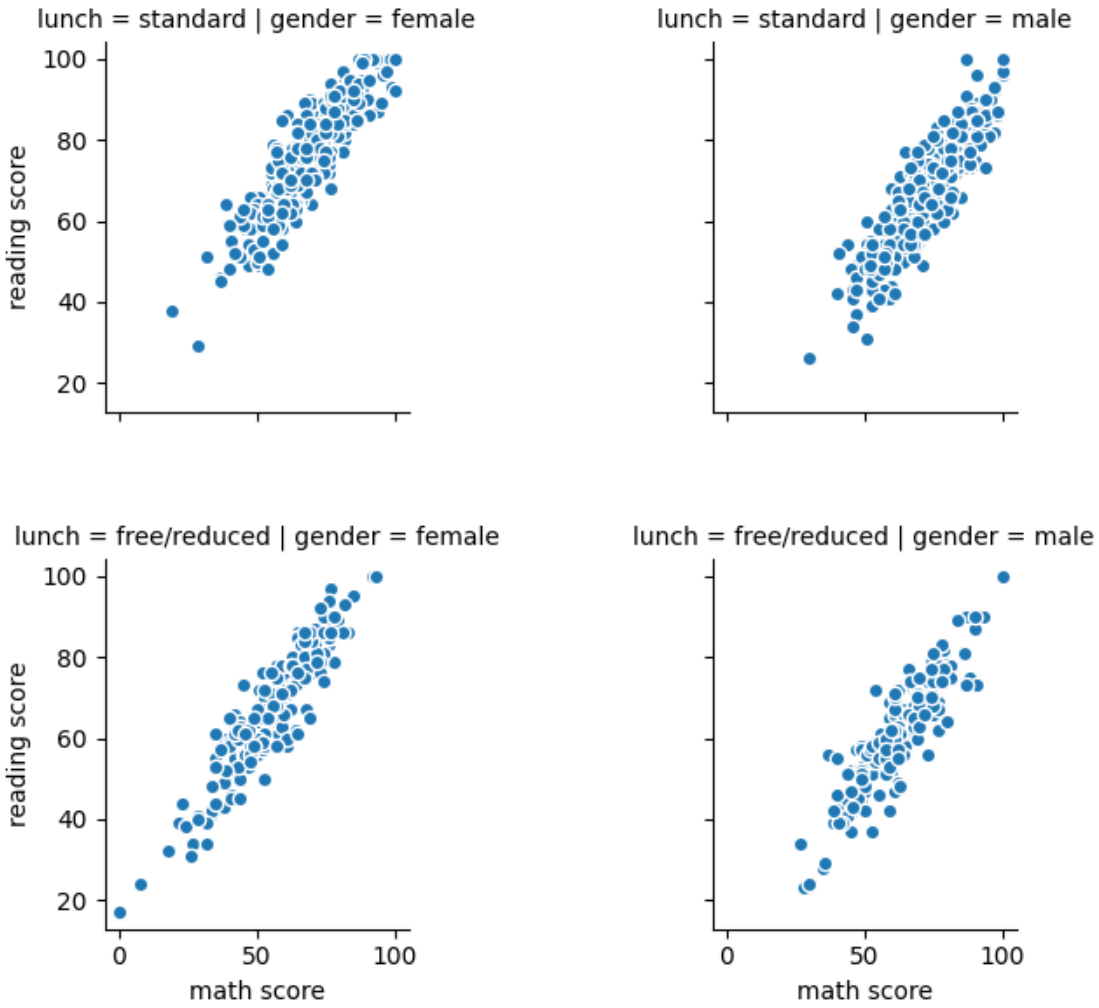

```
g = sns.FacetGrid(data=dff,col='gender',row='lunch')
g = g.map(plt.scatter, "math score", "reading score", edgecolor="w")
g.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x1ec43c9b6a0>
```



https://stackoverflow.com/questions/43669229/increase-space-between-rows-on-facetgrid-plot

```
g = sns.FacetGrid(data=dff,col='gender',row='lunch')
g = g.map(plt.scatter, "math score", "reading score", edgecolor="w")
g.add_legend()

plt.subplots_adjust(hspace=0.4, wspace=1)
```



Matrix Plots

NOTE: Make sure to watch the video lecture, not all datasets are well suited for a heatmap or clustermap.

The Data

World Population Prospects publishes United Nations population estimates for all world countries and every year from 1950 to 2020, as well as projections for different scenarios (low, middle and high variants) from 2020 to 2100. The figures presented here correspond to middle variant projections for the given year.

https://www.ined.fr/en/everything_about_population/data/all-countries/?lst_continent=900&lst_pays=926

Source : Estimates for the current year based on data from the World Population Prospects. United Nations.

```
# 2020 Projections
```

```
sd=pd.read_csv("D:\\Study\\Programming\\python\\Python course from
udemy\\Udemy - 2022 Python for Machine Learning & Data Science
Masterclass\\01 - Introduction to Course\\1UNZIP-FOR-NOTEBOOKS-FINAL\\
05-Seaborn\\country_table.csv")
sd.head()
```

```
                         Countries  Birth rate  Mortality rate  \
0                           AFRICA      32.577           7.837
1                             ASIA      15.796           7.030
2                           EUROPE      10.118          11.163
3  LATIN AMERICA AND THE CARIBBEAN      15.886           6.444
4                  NORTHERN AMERICA      11.780           8.833

   Life expectancy  Infant mortality rate  Growth rate
0           63.472                 44.215        24.40
1           73.787                 23.185         8.44
2           78.740                  3.750         0.38
3           75.649                 14.570         8.89
4           79.269                  5.563         6.11
```

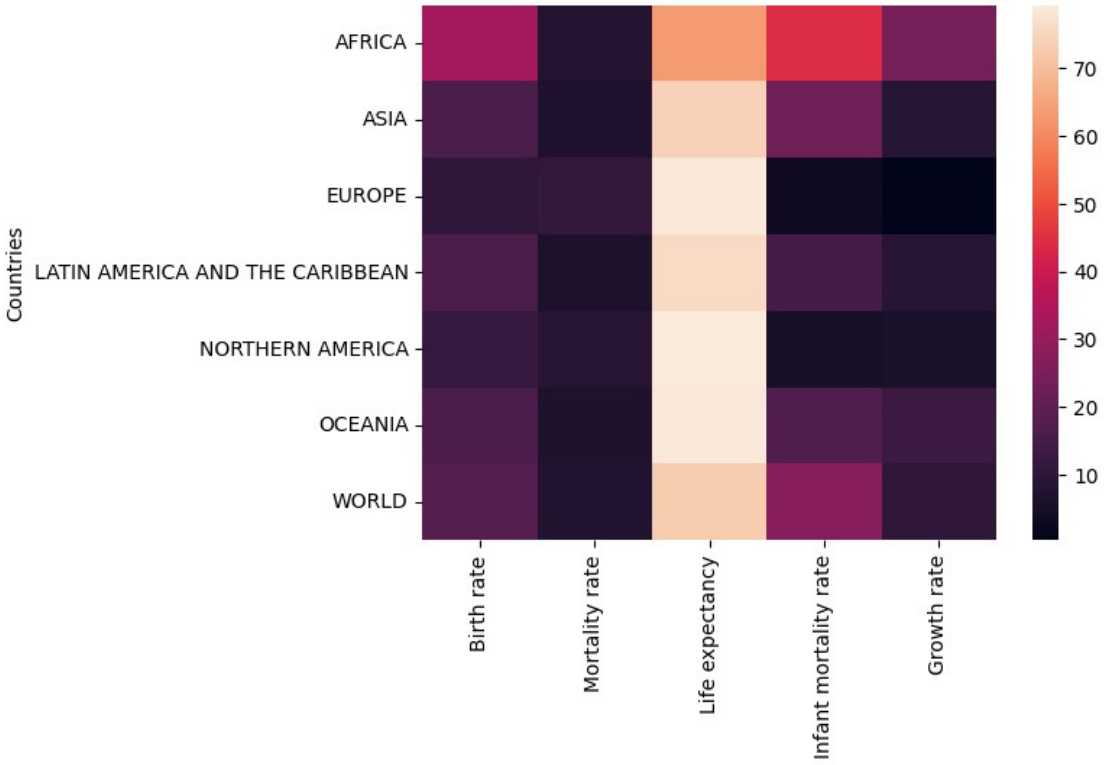
Heatmap

```
sd=sd.set_index('Countries')
```

```
sd.head()
```

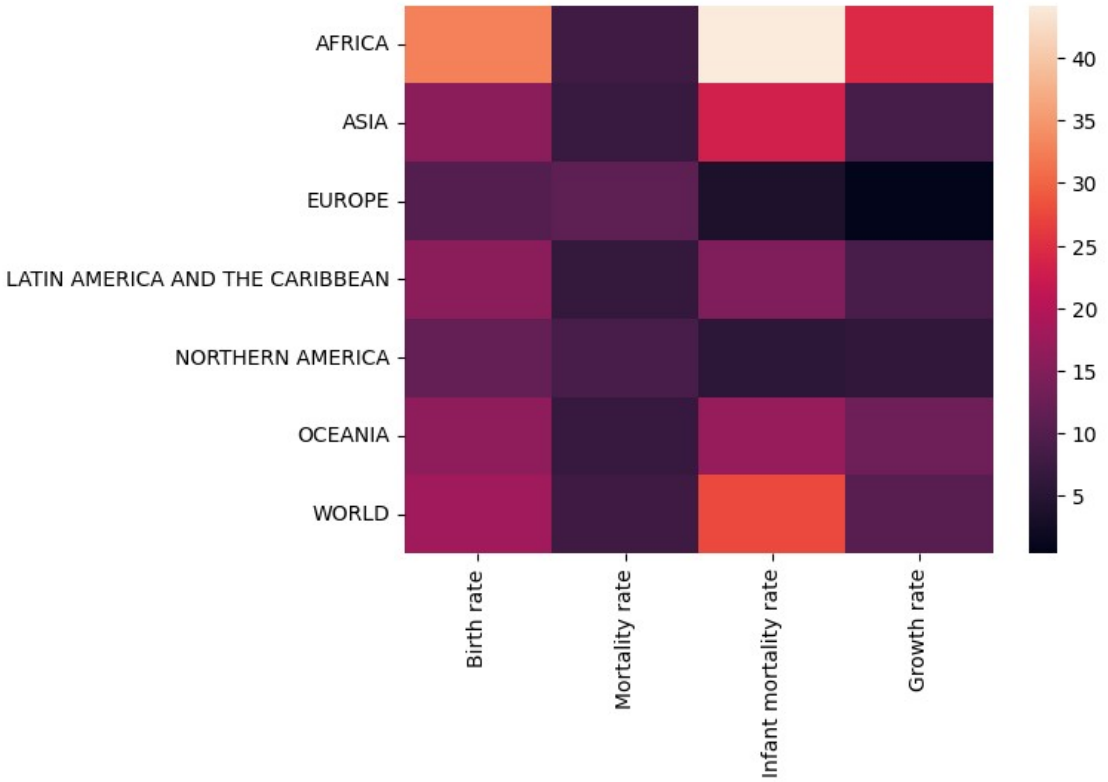
```
                                Birth rate  Mortality rate  Life
expectancy  \
Countries

AFRICA                             32.577           7.837
63.472
ASIA                               15.796           7.030
73.787
EUROPE                             10.118          11.163
78.740
LATIN AMERICA AND THE CARIBBEAN    15.886           6.444
75.649
NORTHERN AMERICA                   11.780           8.833
79.269

                                Infant mortality rate  Growth rate
Countries
AFRICA                                         44.215        24.40
ASIA                                           23.185         8.44
EUROPE                                          3.750         0.38
LATIN AMERICA AND THE CARIBBEAN                14.570         8.89
NORTHERN AMERICA                                5.563         6.11
```

```
# Clearly shows life expectancy in different units like it in 70,80
etc so we need to drop that to manage our scale
sns.heatmap(data=sd);
```



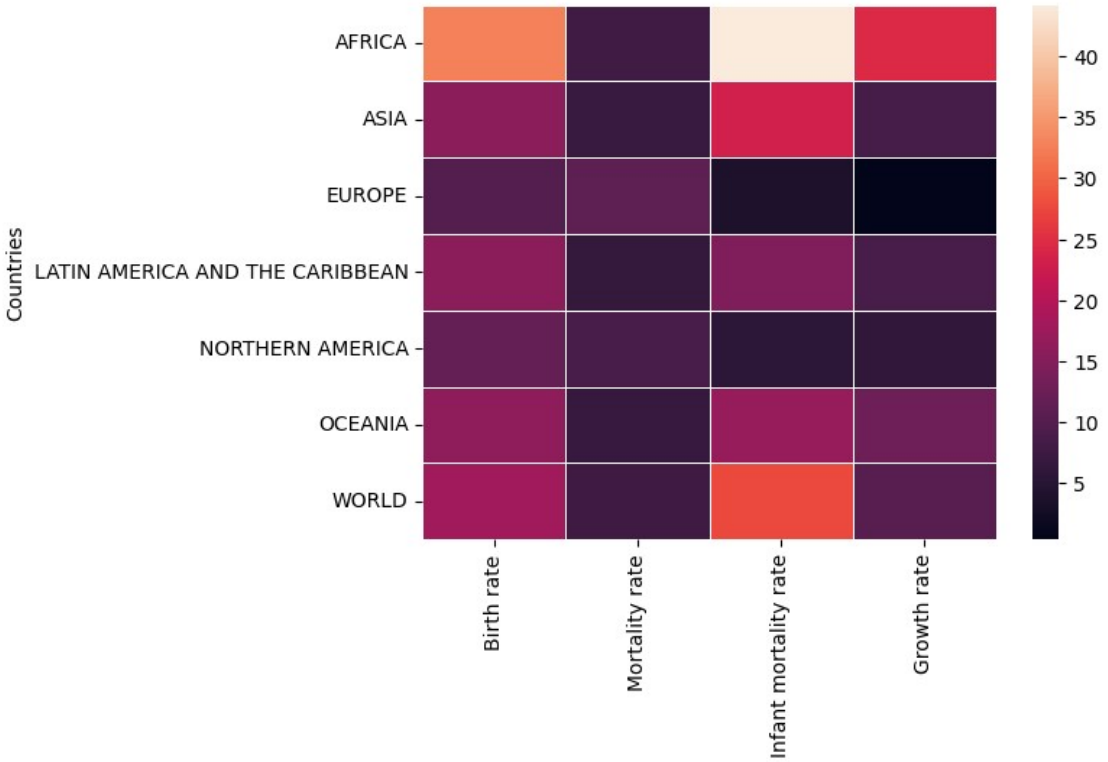

```
rates = sd.drop('Life expectancy',axis=1)

sns.heatmap(rates) # Now scale is much improved

# OR

#sns.headmap(df.drop('life expectancy', axis =1))
```

```
<AxesSubplot: >
```



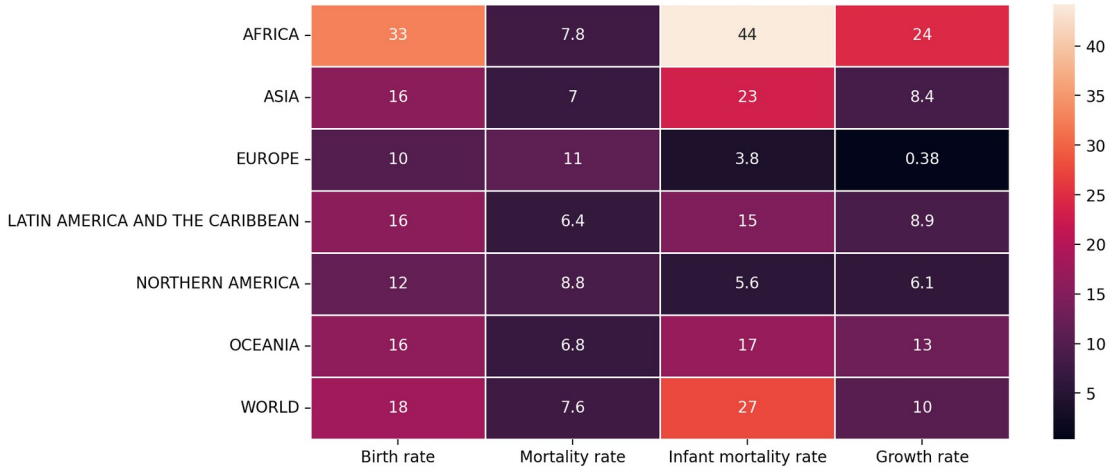

```
sns.heatmap(rates,linewidth=0.5)
```

```
<AxesSubplot: ylabel='Countries'>
```



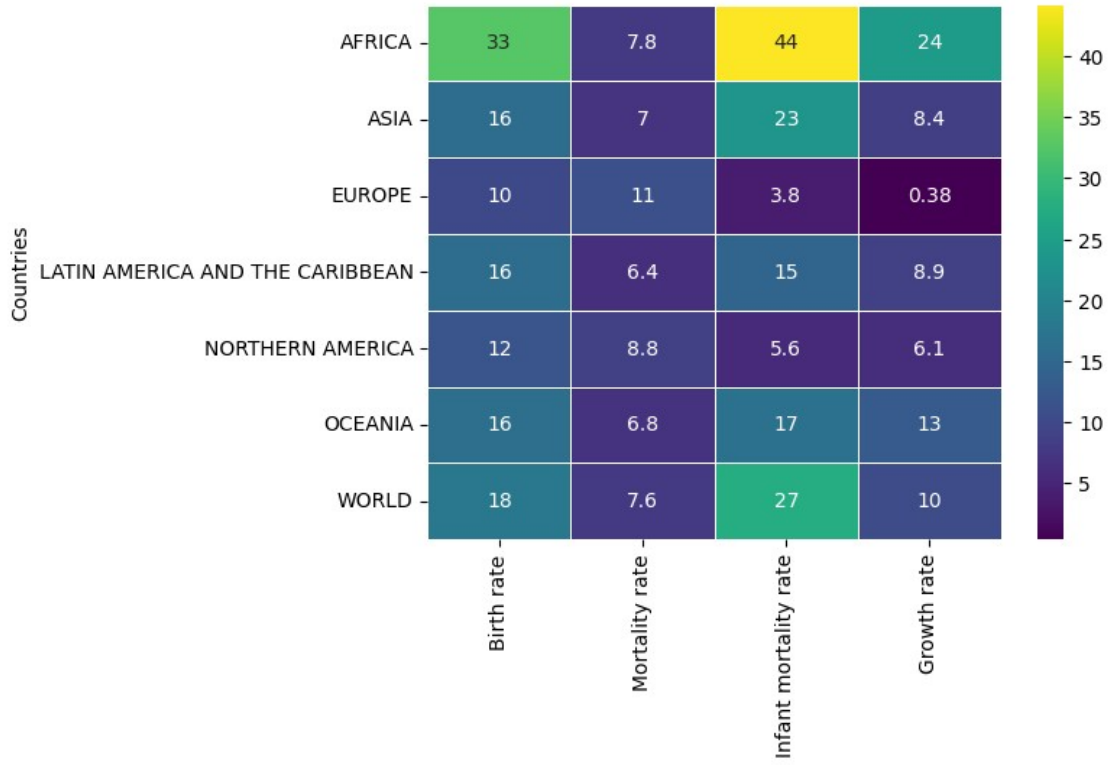
```
# Here annotation are those number that showing in mid of cell
plt.figure(figsize=(10,5),dpi=200)
sns.heatmap(rates,linewidth=0.5,annot=True)
```

```
<AxesSubplot: >
```



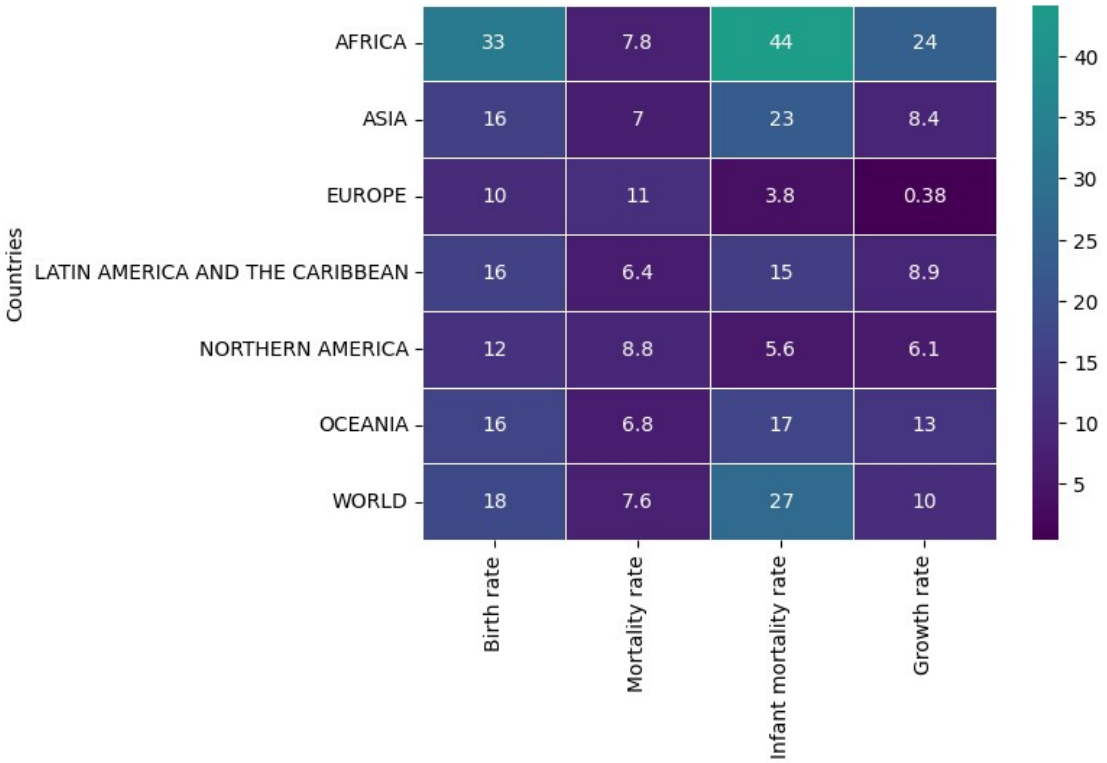

```
# Note how its not palette here, its cmap
sns.heatmap(rates,linewidth=0.5,annot=True,cmap='viridis')
```

```
<AxesSubplot: ylabel='Countries'>
```




```
# Set colorbar based on value from dataset
sns.heatmap(rates,linewidth=0.5,annot=True,cmap='viridis',center=40)
```

```
<AxesSubplot: ylabel='Countries'>
```




```
# Set colorbar based on value from dataset
sns.heatmap(rates,linewidth=0.5,annot=True,cmap='viridis',center=1)
```

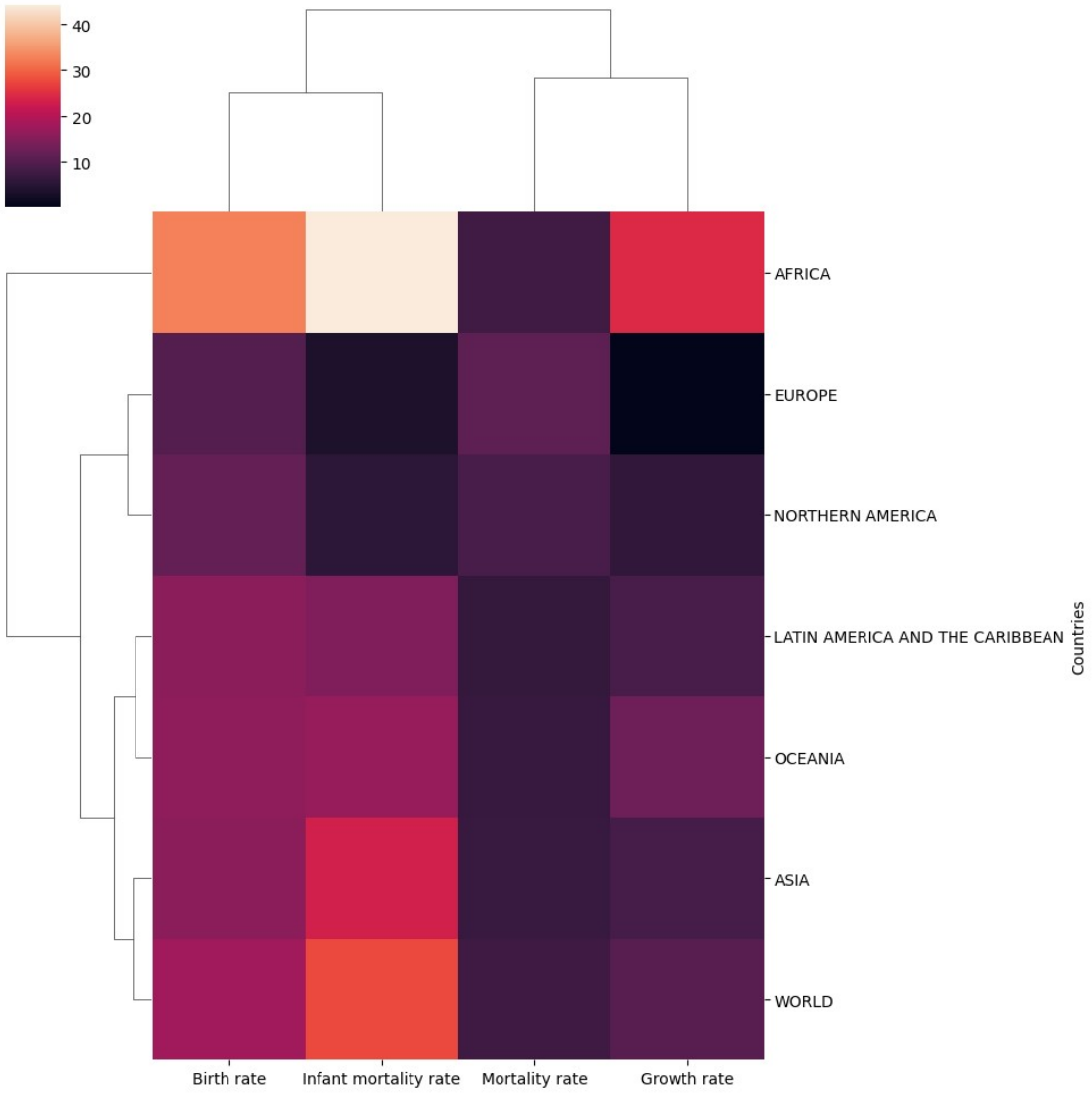
```
<AxesSubplot: ylabel='Countries'>
```

Clustermap

Plot a matrix dataset as a hierarchically-clustered heatmap.

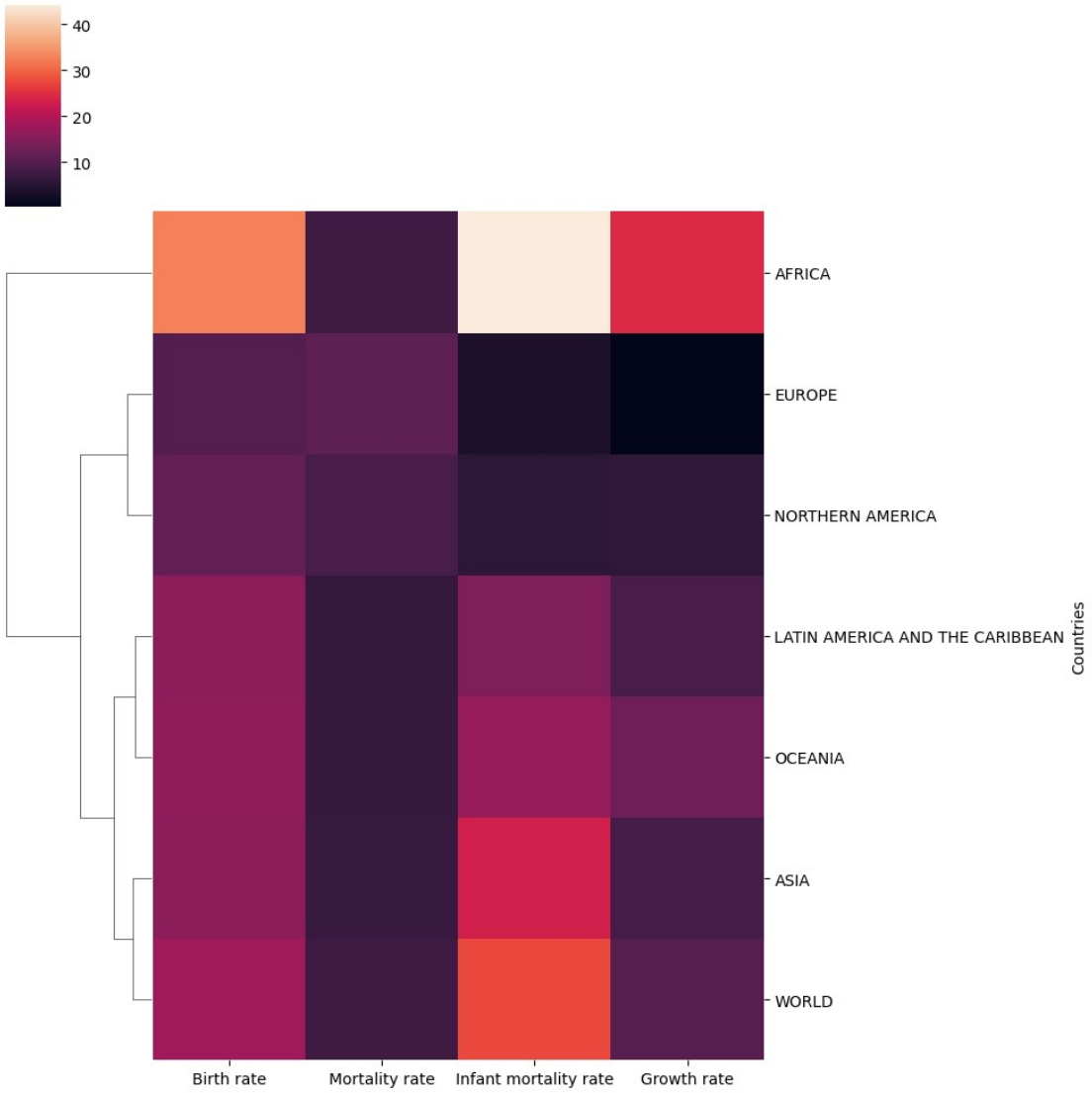
```
sns.clustermap(rates)
```

```
<seaborn.matrix.ClusterGrid at 0x1ec426d65f0>
```



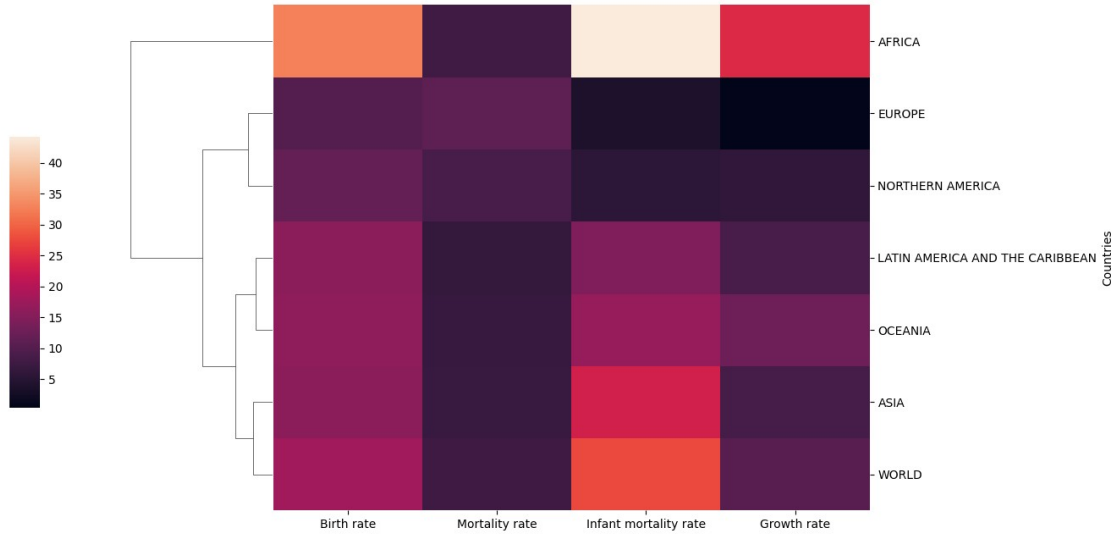

```
sns.clustermap(rates,col_cluster=False)
```

```
<seaborn.matrix.ClusterGrid at 0x1ec451bcaf0>
```




```
sns.clustermap(rates,col_cluster=False,figsize=(12,8),cbar_pos=(-0.1,
.2, .03, .4))
```

```
<seaborn.matrix.ClusterGrid at 0x1ec3fa2e680>
```



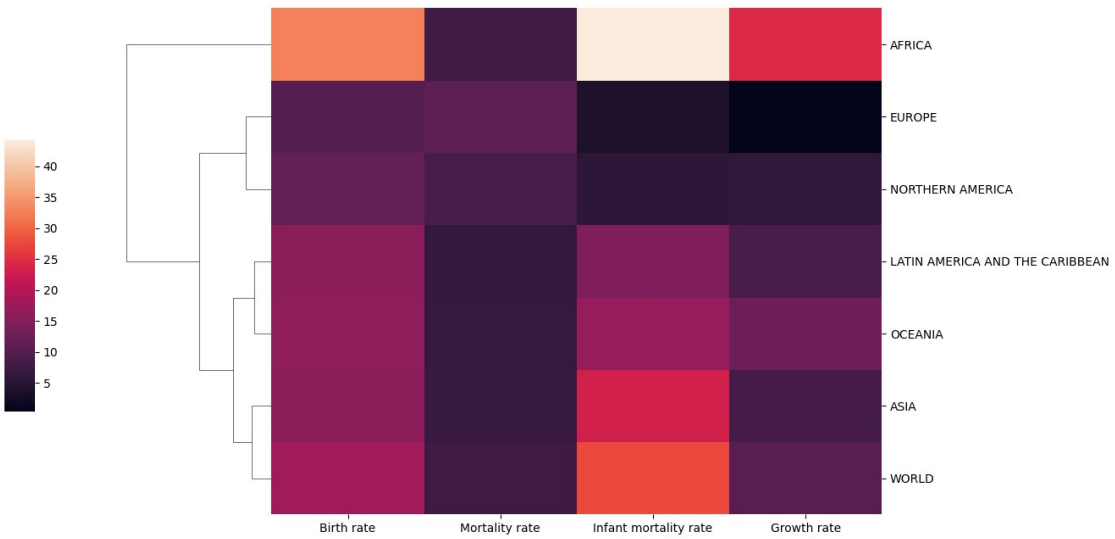

```
rates.index.set_names('',inplace=True)
```

```
rates
```

```
                                Birth rate  Mortality rate  \

AFRICA                              32.577           7.837
ASIA                                15.796           7.030
EUROPE                              10.118          11.163
LATIN AMERICA AND THE CARIBBEAN     15.886           6.444
NORTHERN AMERICA                    11.780           8.833
OCEANIA                             16.235           6.788
WORLD                               17.963           7.601

                                Infant mortality rate  Growth rate
AFRICA                                         44.215        24.40
ASIA                                           23.185         8.44
EUROPE                                          3.750         0.38
LATIN AMERICA AND THE CARIBBEAN                14.570         8.89
NORTHERN AMERICA                                5.563         6.11
OCEANIA                                        16.939        12.79
WORLD                                          27.492        10.36
```

```
# Recall you can always edit the DF before seaborn
sns.clustermap(rates,col_cluster=False,figsize=(12,8),cbar_pos=(-0.1,
.2, .03, .4))
```

```
<seaborn.matrix.ClusterGrid at 0x1ec459a83d0>
```


40
35
30
25
20
15
10
5
AFRICA
EUROPE
NORTHERN AMERICA
LATIN AMERICA AND THE CARIBBEAN
OCEANIA
ASIA
WORLD
Birth rate
Mortality rate
Infant mortality rate
Growth rate