**Basic Commands**

* Pretty much self explanatory

plt.title('String Title')

plt.xlabel('x-axis')

plt.ylabel('y-axis')

plt.xlim(0,6)

plt.ylim(0,16)

**Figure and axes**

* This is an object-oriented approach to plotting. We create a ‘figure’ object which is basically like a blank canvas upon which we proceed with our plotting.

fig = plt.figure(figsize=(2,2),dpi=200)

* Once we have this figure, we can have axes on it. Axes are the plots -> Figure is the canvas.
* figsize = (width,height) dpi – clarity of the plot produced

axes1 = fig.add\_axes([0,0,1,1])

axes1.plot(a,b)

* [0,0,1,1] – Here the first two co-ordinates specify where the left corner should be. (0,0) means the axes will start at the left corner . If it was instead (0.5,0.5) the axes would have started midway through the x and y -axis
* 1,1 – This stands for the width and height. These quantities are relative to the canvas size. (1,1) means width and height are same as the figure. (0.5,0.5) would result in an axes with height and width half the size of the figure.

axes1.set\_xlabel('A')

axes1.set\_ylabel('B')

axes1.set\_xlim(0,12)

axes1.set\_ylim(0,10000)

axes1.set\_title('axis1')

**axes.set\_property() :** property – xlabel,ylabel,title,xlim,ylim

fig.savefig('new\_fig.png',bbox\_inches = 'tight' )

* Bbox\_inches parameter is used so that the x and y-axis are saved too

**Subplots using axes and figures**

fig = plt.figure()

plot1 = fig.add\_axes([0,0,0.5,1]) #We are ensuring that the first plot takes exactly half the area

plot1.set\_title('plot1')

plot2 = fig.add\_axes([0.5,0,0.5,1])

plot2.set\_title('plot2') #We are ensuring that the second plot takes up the remaining area

**Subplots**

* Using figures and axes to get subplots can get tedious and hence we go for the subplots function

fig,axes = plt.subplots(nrows =3,ncols=1)

* Here the key thing is the method how we access the subplots. Similar to how we access an element in an n-dimensional array.

#Accessing the plots (similar to accessing the element in an n-dimensional array)

axes[0].plot(x,y)

axes[1].plot(a,b)

axes[2].plot(b,y)

plt.tight\_layout() #Makes sure no axes overlap automatically

axes[0].set\_title('X vs Y')

axes[1].set\_title('A vs B')

axes[2].set\_title('B vs Y')

* **plt.tight\_layout()** makes sure that there is no overlap between the subplots
* In case we had a 2X2 subplot . In that case accessing the elements would be in the following way:

fig,axes = plt.subplots(nrows=2,ncols=2)

axes[0][0].plot(x,y)

axes[0][1].plot(a,b)

axes[1][0].plot(x,y)

axes[1][1].plot(a,b)

**Legends**

fig = plt.figure()

ax = fig.add\_axes([0,0,1,1])

#Adding labels in every plot call and using ax.legend in the end

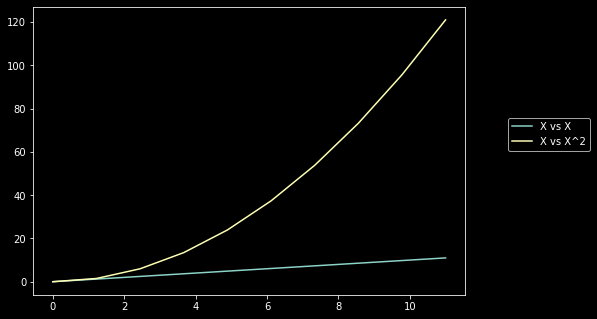
ax.plot(x,x,label= 'X vs X')

ax.plot(x,x\*\*2,label = 'X vs X^2')

#ax.legend(loc='upper left') # We can specify different locations

ax.legend(loc=(1.1,0.5)) #Again the tuple values are relative to the axes dimensions

* We add labels to every plot call and ultimately use the **ax.legend()** function. The location parameter is again relative to the underlying canvas/figure.



**Visual Styling**

* **Colour and Linewidth**

fig = plt.figure()

ax = fig.add\_axes([0,0,1,1])

ax.plot(x,x,color='#65c79b', lw=10,label='X vs X') #RGB HEX code

ax.plot(x,x+1,color='#81bccc',label= 'X vs X+1')

ax.legend()

As seen here, we can provide hex color codes . **lw** is the parameter for linewidth.

* **Linestyles**

fig = plt.figure()

ax = fig.add\_axes([0,0,1,1])

ax.plot(x,x,color='#65c79b',linewidth=4,linestyle= '-.')

We can either use the predefined linestyles or feed our customized styles.

lines = ax.plot(x,x,color='#65c79b',linewidth=4,linestyle= '-.')

lines[0].set\_dashes([5,2,5,2])

* **Markers**

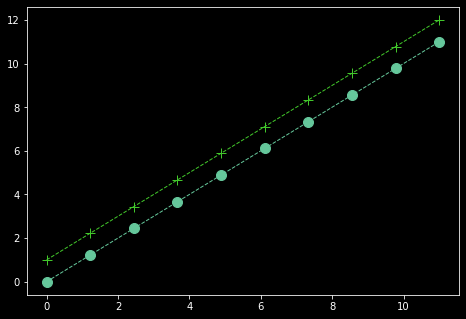
These are used to highlight the individual data points which might otherwise be masked in the graph. We can choose different markers by feeding in different string characters. Marker width is controlled by the parameter **ms.**

fig = plt.figure()

ax = fig.add\_axes([0,0,1,1])

ax.plot(x,x,color='#65c79b',linewidth=1,marker='o',ls='--',ms=10)

ax.plot(x,x+1,color='#42c72b',linewidth=1,marker='+',ls='--',ms=10)



**Seaborn**

**Scatterplots**

* Used for continuous data.

plt.figure(figsize=(5,5),dpi=200)

sns.scatterplot(x='salary',y='sales',data=df,hue='level of education',palette='Dark2')

* **‘hue’** parameter is used to colour the datapoints based on the value of the column that is fed to this parameter. If the column has a continuous feature then the resulting colouring will be a gradient and if it is categorical we will have a legend with a distinct colour for each feature
* **Palette** parameter is used to set colormaps. This is basically the colour schemes that will be adapted when the hue parameter is used.
* Similar to hue, we can vary the size of datapoints based on certain column values.

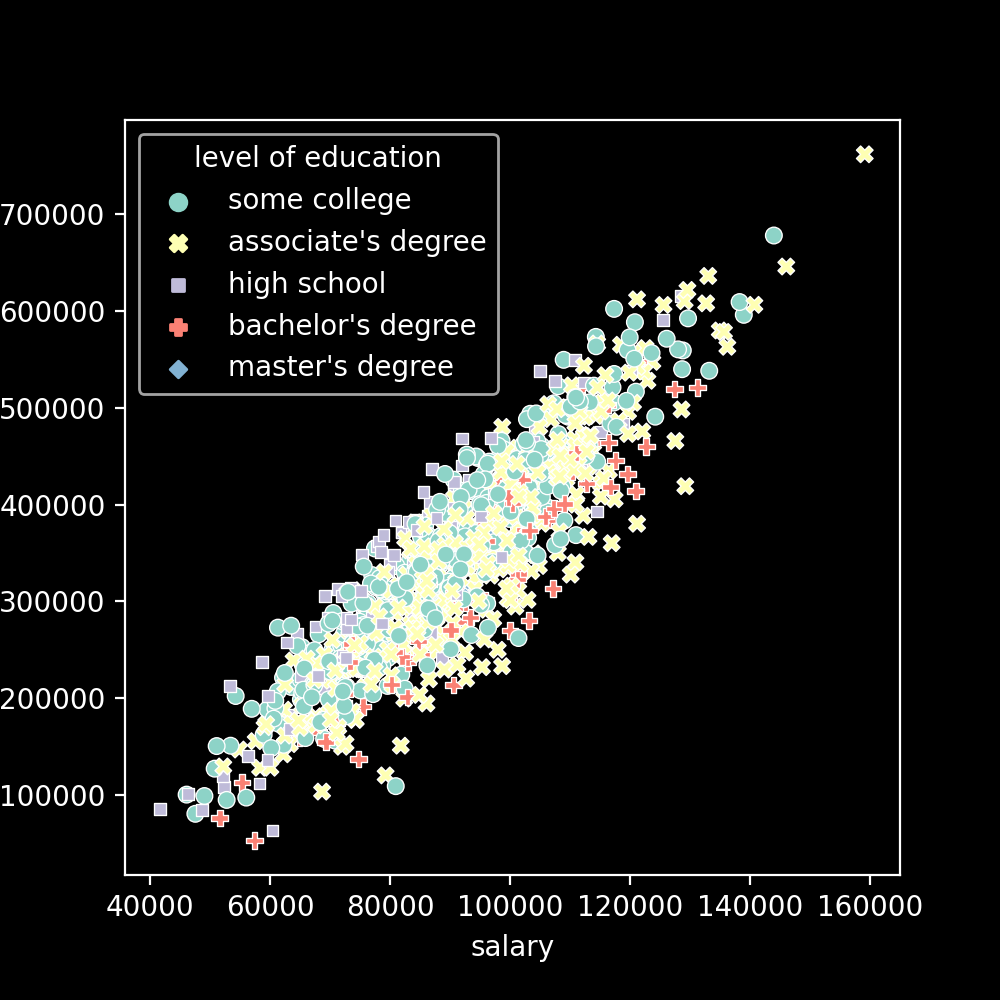
plt.figure(figsize=(5,5),dpi=200)

sns.scatterplot(x='salary',y='sales',data=df,size='salary',palette='Dark2',alpha=0.5)

* alpha parameter is for the transparency of the data points.
* Finally the style(different symbols for different column values) can also be used. Particularly useful when black and white plots are being used.

plt.figure(figsize=(5,5),dpi=200)

sns.scatterplot(x='salary',y='sales',data=df,style='level of education',hue='level of education',alpha=1)



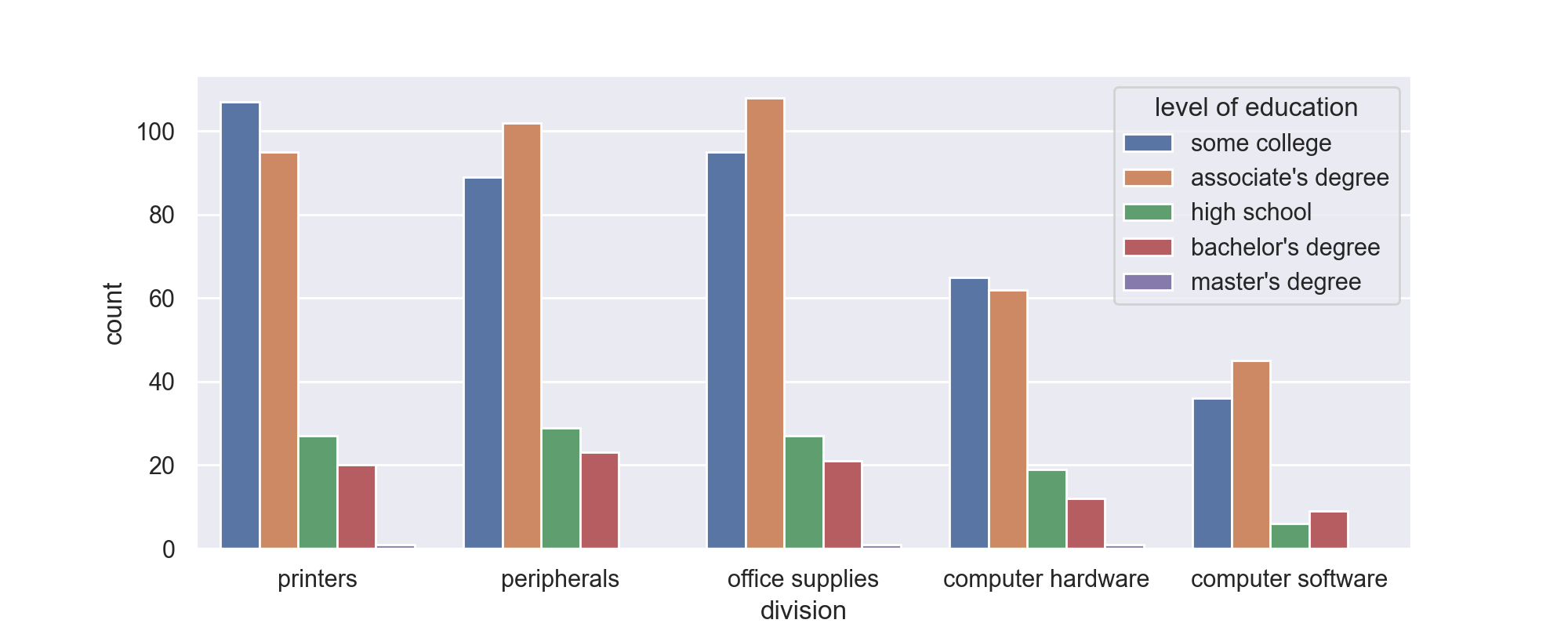
**Categorical Plots**

**Countplot vs bar plot**

* Countplot is used to represent the frequency of occurrence for categorical variables.
* Bar plot is a bit more general. We can represent values like average values of a category as well as their standard deviation.

plt.figure(figsize=(10,4),dpi=200)

sns.countplot(data=df,x='division',hue='level of education')



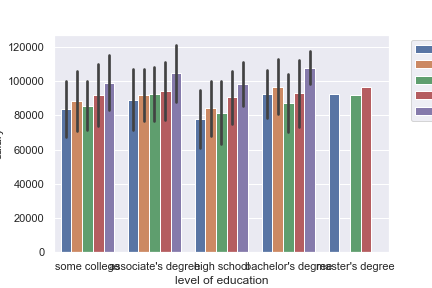
Barplot

#Barplot

sns.barplot(data=df,x='level of education',y='salary',estimator=np.mean,ci='sd',hue='division')

plt.legend(bbox\_to\_anchor=(1.05,1))

plt.savefig('barplot.png')



* As mentioned above, barplot can be used not just to count the occurrence of a categorical variable but also other parameters. Here **estimator** parameter is used to achieve that and the mean is being displayed.
* **‘ci’** parameter stands for confidence interval(the line in each bar). In this case it is being used to depict the standard deviation of the data.
* **bbox\_to\_anchor** is used to position the legend in the plot. Again, the coordinates are relative to the underlying canvas.

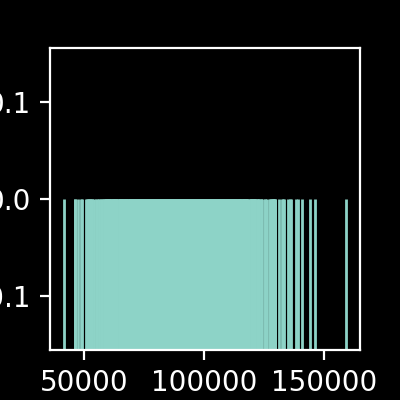
**Distribution plots**

**Rug plot**

* One tick per data point on y axis. Y-axis is meaningless.
* Helps identify outliers

plt.figure(figsize=(2,2),dpi=200)

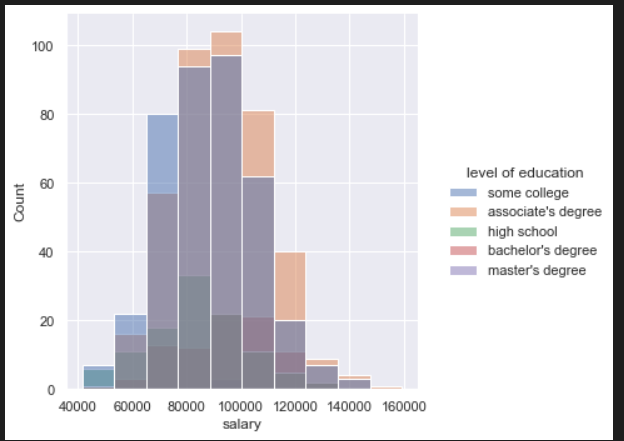
sns.rugplot(x='salary',data=df,height=0.5)



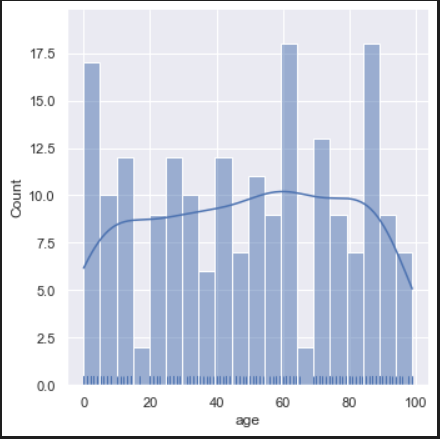
**KDE plot(kernel density estimation plot)**

* Estimating a continuous distribution for a given datapoint.
* For histograms, although there is a **sns.histplot()** available we go for the more general **sns.displot().**

sns.displot(data=df,x='salary',bins=10,hue='level of education')



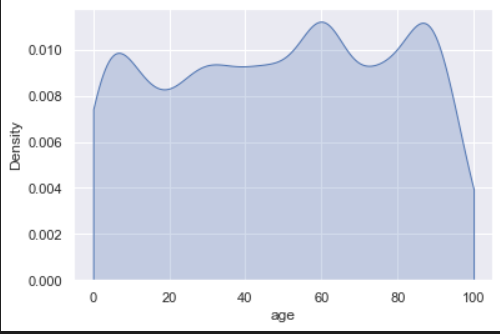
sns.displot(data=sample\_ages,x='age',bins=20,rug=True,kde=True)



* **rug=True and kde=True** parameters can be used to have the kde estimation and Rug plot included within the histogram. Can vary bin size to obtain varying plots.

sns.kdeplot(data=sample\_ages,x='age',clip=[0,100],bw\_adjust=0.6,shade=True)

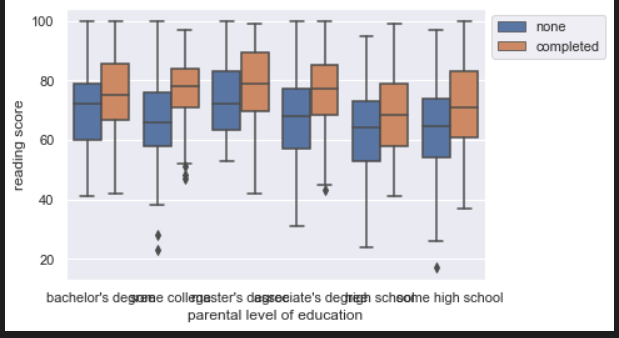
* **Clipping** off values is similar to xlim and ylim. It is to eliminate impossible values like negative salary values.
* Lower the value of **bw\_adjust**, more the variance that is picked up. A high value would mean a high degrees of underfitting
* **shade** fills up the area under the curve



**Distribution Within categories**

**Box plots**

sns.boxplot(data=df,y='reading score',x='parental level of education',hue='test preparation course')



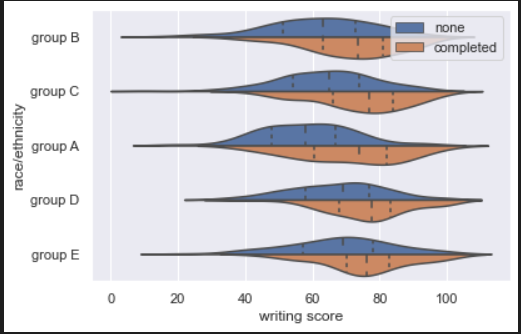
* This can be flipped too depending on which axis the continuous variable is located on. Setting hue to a column value, we are obtaining separate box plots for each column value.

**Violin Plot**

* This is a KDE plot which is mirrored and attached at the bottom.

sns.violinplot(data=df,x='writing score',y='race/ethnicity',hue='test preparation course',split=True,inner='quartile')

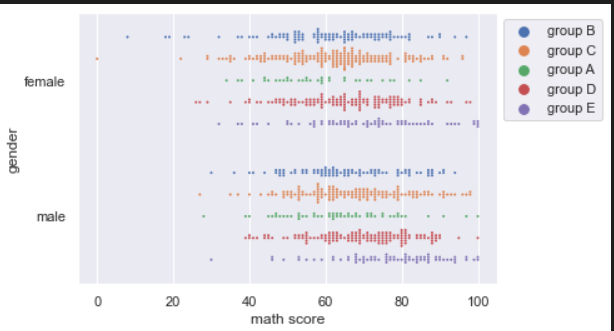
* By having **split=True** one half of violin plot is one category other half is another. This prevents separate violin plots for column plots like in the box plot displayed above.
* Inner = quartile draws lines at each quartile



**Swarm Plot**

#Swarm plot

sns.swarmplot(data=df,x='math score',y='gender',hue='race/ethnicity',size=2,dodge=True)



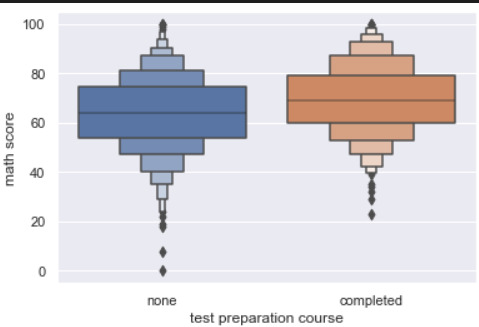
* Hue paramater will be difficult to interpret since we are considering individual data points

. Hence we go for **dodge=true** to split the plot based on the hue parameter

**Boxen Plot**

#Boxen plot: Said to be more informative than normal box plot

sns.boxenplot(data=df,x='test preparation course',y='math score')



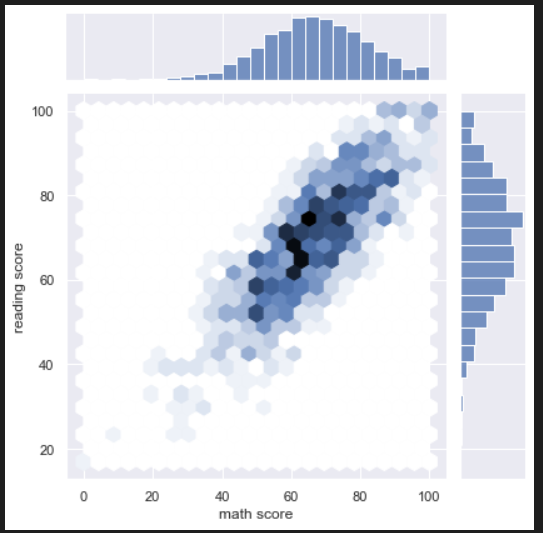
**Comparison plots**

**Joint Plot**

* In this plot when we consider two features, we plot the relation between the two categories(hex/scatter plot) along with the underlying distributions of those features(kde /histogram plot)

sns.jointplot(x='math score',y="reading score",data=df,kind='hex')

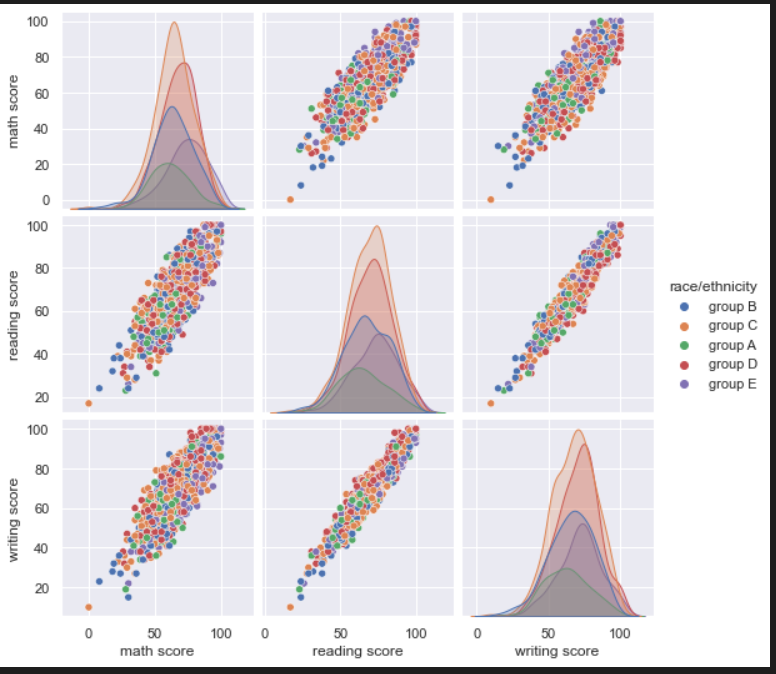
* the kind parameter can be used to specify between plot types be it for the relation plots(hex/scatter) or the underlying distributions(histogram,kde)



**Pairplots**

* It is CPU intensive
* Plots out histograms as well as scatter plots for every possible feature combination
* With pairplots, it is advisable to feed in only the columns we are interested in.

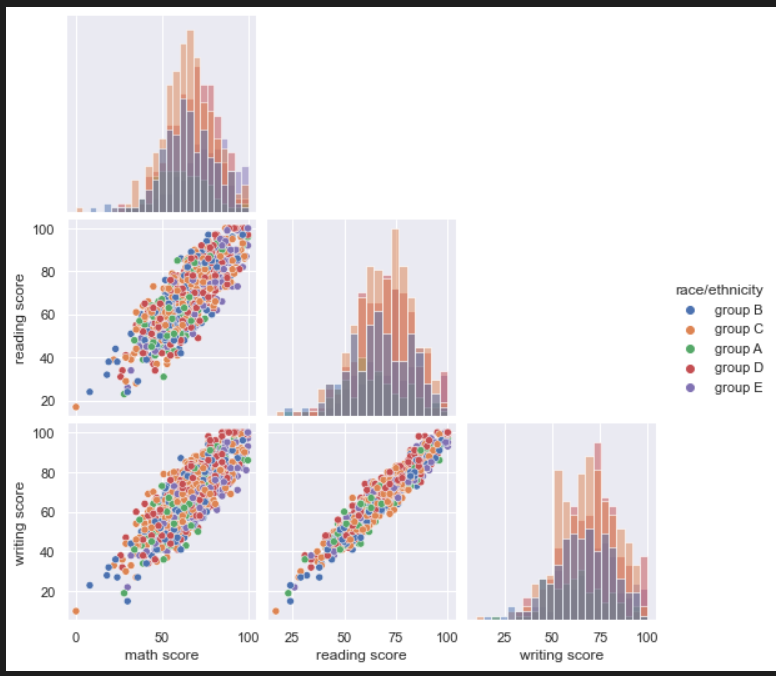
sns.pairplot(data=df,hue='race/ethnicity')



* It can be noticed that the diagonal elements are underlying distributions while the non diagonal elements are relation plots. It is also noticed that the plot is symmetric along the diagonal.
* We have parameters for both of this. That is to get rid of duplicates as well as choose what the underlying distribution type must be (histogram/ kde)

sns.pairplot(data=df,hue='race/ethnicity',diag\_kind='hist',corner=True)

* **corner=True** eliminates the duplicates
* **diag\_kind** parameter is used to ensure that the diagonal elements are histograms only



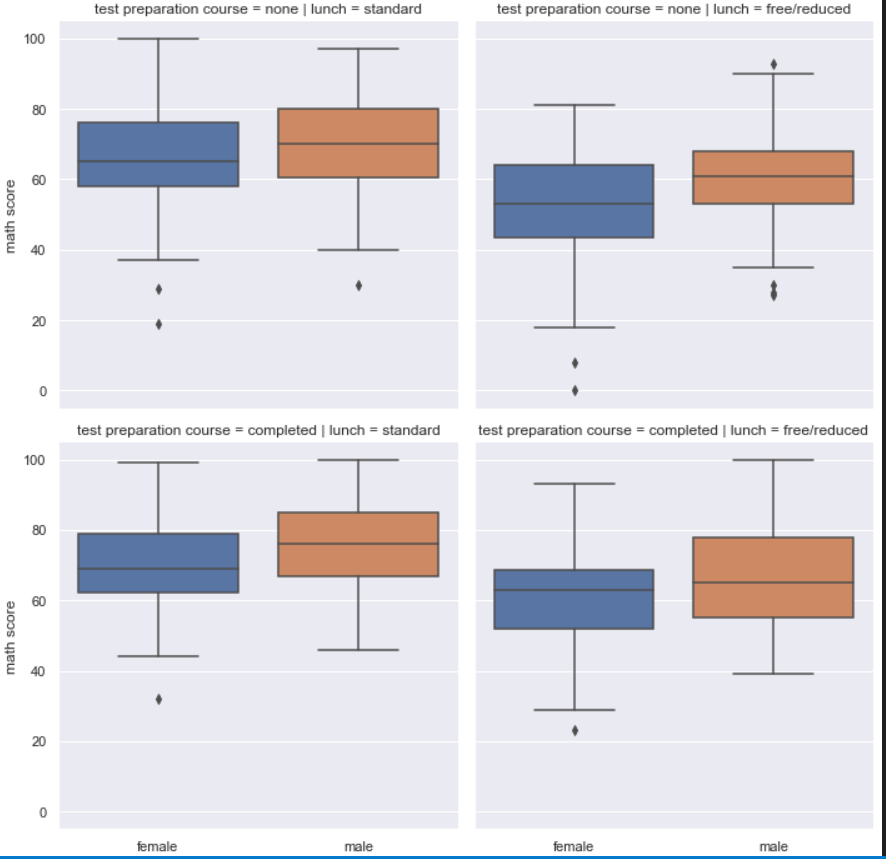
**Grid plots**

* These are equivalent to using the subplot functionality in matplotlib. There are two methods, catplot and pairgrid.

**Catplot**

sns.catplot(data=df,x='gender',y='math score',kind='box',col='lunch',row='test preparation course')

* Advantage to matplotlib is that we don’t need to specify the number of rows and columns. Instead the row and col parameters take in values and the fed features will vary along the rows and columns respectively. The **kind** parameter again is used to specify what kind of plots we need.



**Pairgrid**

* When we use pairplots , internally the pairgrid functionality is called and used.

grid = sns.PairGrid(df,hue='gender')

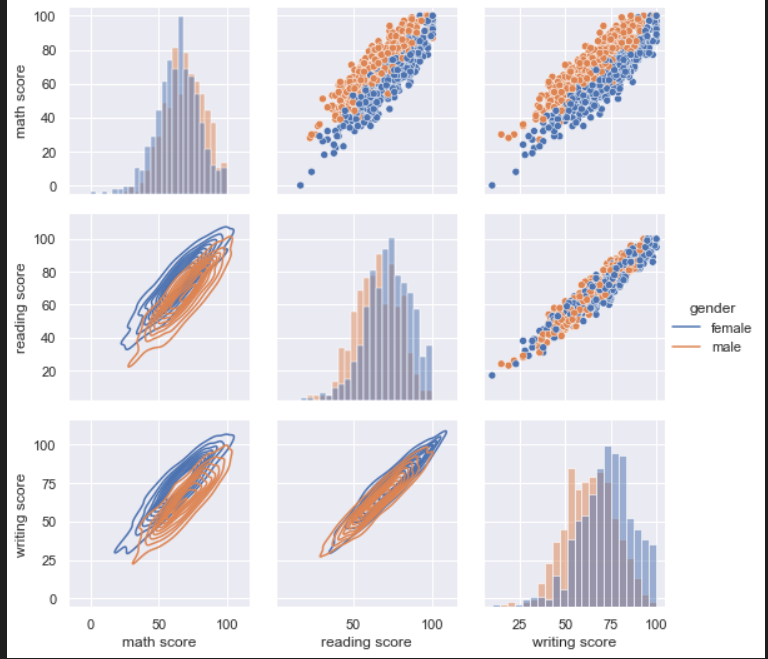
grid=grid.map\_upper(sns.scatterplot)

grid= grid.map\_lower(sns.kdeplot)

grid = grid.map\_diag(sns.histplot)

grid=grid.add\_legend()

* The listed functions are used to customize the plots above, below and along the diagonal respectively. And since grids aren’t added automatically , we need to use the command mentioned in the end.



**Matrix Plots**

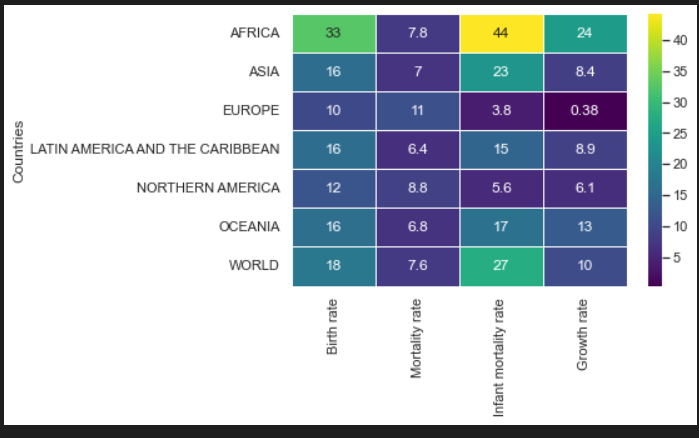
* The purpose of heatmap or cluster-map is to better visualize the volume of locations/events within a dataset and assist in directing viewers towards areas on data visualizations that matter most.
* Ideally all the values should be in the same unit
* Helps identify regions/people with similar features.
* Seaborn can innately group together similar groups

df=df.set\_index("Countries")

* This is a crucial step because it allows us to obtain a more meaningful graphical representation.

sns.heatmap(df.drop('Life expectancy',axis=1),linewidth=0.5,annot=True,cmap='viridis')

* The life expectancy column in being dropped because it’s units/magnitude is not on scale with the other features.(Others were all rates).
* **Annot= true** to have the value displayed and **linewidth** to better segregate the categories.



**Clustermap**

* This helps us identify the categories with similar values.

sns.clustermap(df.drop('Life expectancy',axis=1),linewidth=0.5,annot=True,cmap='viridis',col\_cluster=False)

