Python Libraries For Data Science

Last lecture reminder

We learned about:

- Matplotlib subplot() method
- Legends - use and customization
- Plot styling and customization
- Different chart types - definition, usage and Matplotlib creation methods:
    - Line chart
    - Pie chart
    - Bar chart
    - Histogram
    - Scatter chart

seaborn

Introduction to Seaborn Library

Seaborn is a Python data visualization library which is based on Matplotlib (abstraction). It provides a high-level interface for drawing attractive and informative statistical graphics. It is more integrated with the Pandas library and it allows you to have better control over the aesthetics of your plots, providing numerous built-in themes for styling.

Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

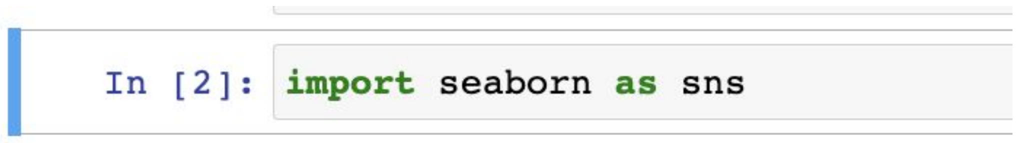
Seaborn commonly usages are:

- Plotting statistical graphs
- Making complex visualizations
- Visualizing distributions of data
- Plotting time series data

Installing & Importing Seaborn

In order to work with the Seaborn library we will first need to install and import it by running those commands:

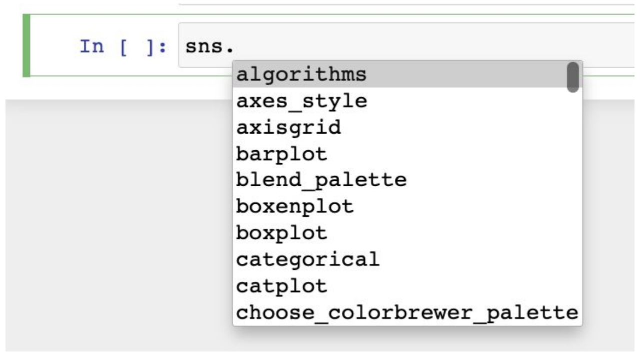
- In the computer CMD → **pip install seaborn**
- In Jupyter Notebook cell → **import seaborn as sns**



```
In [2]: import seaborn as sns
```

What we did was to import all the seaborn functions into a variable called sns.

By pressing sns. + TAB we will be able to see all the different functions that Seaborn has.



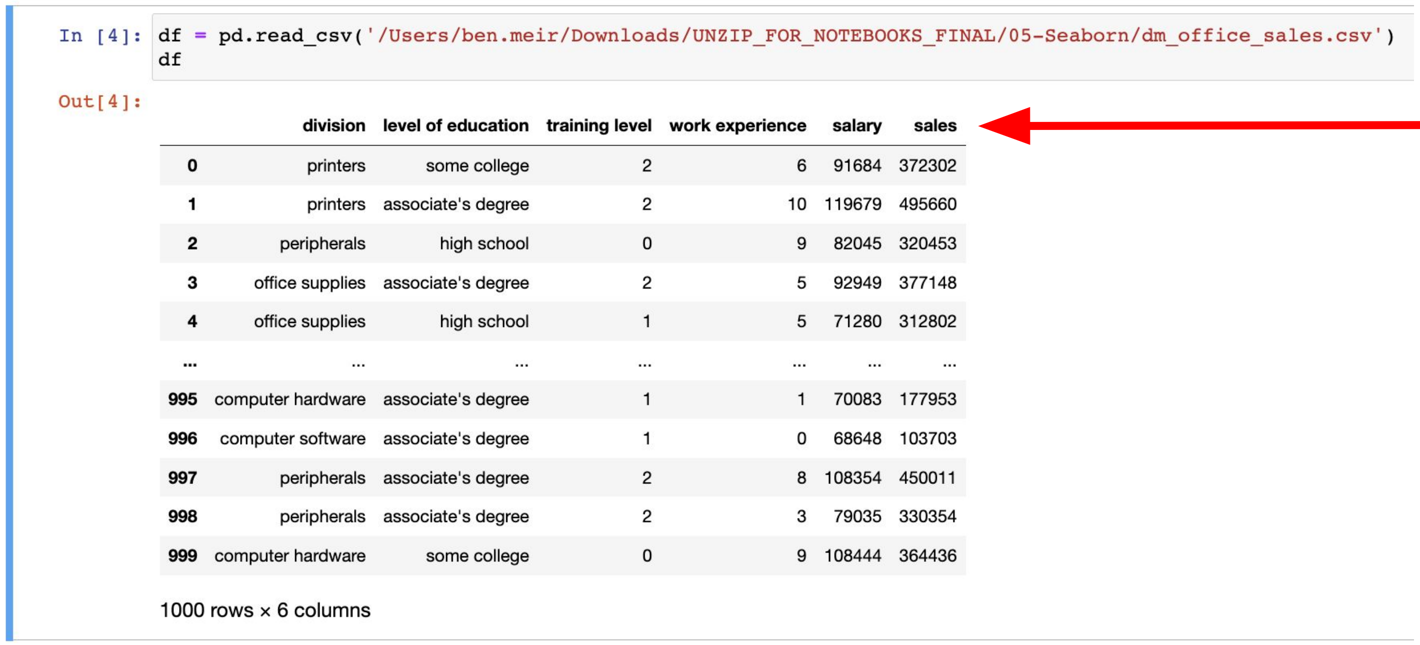
```
In [ ]: sns.
        algorithms
        axes_style
        axisgrid
        barplot
        blend_palette
        boxenplot
        boxplot
        categorical
        catplot
        choose_colorbrewer_palette
```

ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Seaborn - Scatter Plot

sns.scatterplot() → Method that allow us to create scatter plot from providing Dataframe, when using this method we need to specify which Dataframe column is the independence variable (x-axis) and which is the dependent variable (y-axis).

For example → We will use the **‘dm_office_sales.csv’** file and will explore the correlation between the salary and sales of each salesman:



```
In [4]: df = pd.read_csv('/Users/ben.meir/Downloads/UNZIP_FOR_NOTEBOOKS_FINAL/05-Seaborn/dm_office_sales.csv')
        df
```

Out[4]:

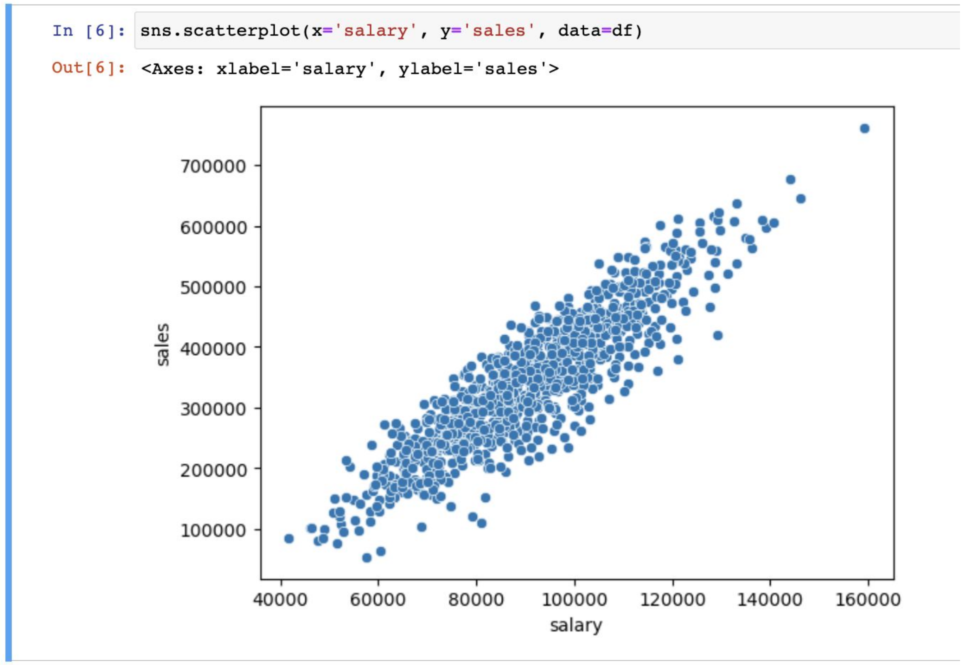
	division	level of education	training level	work experience	salary	sales
0	printers	some college	2	6	91684	372302
1	printers	associate's degree	2	10	119679	495660
2	peripherals	high school	0	9	82045	320453
3	office supplies	associate's degree	2	5	92949	377148
4	office supplies	high school	1	5	71280	312802
...	...	...	...	...	...	...
995	computer hardware	associate's degree	1	1	70083	177953
996	computer software	associate's degree	1	0	68648	103703
997	peripherals	associate's degree	2	8	108354	450011
998	peripherals	associate's degree	2	3	79035	330354
999	computer hardware	some college	0	9	108444	364436

1000 rows × 6 columns

We want to use the ‘sales’ and ‘salary’ Dataframe columns

Seaborn - Scatter Plot

Now let’s create a simple scatter plot using Seaborn scatterplot() method:



```
In [6]: sns.scatterplot(x='salary', y='sales', data=df)
Out[6]: <Axes: xlabel='salary', ylabel='sales'>
```

What can we learn from this plot is that indeed there is a correlation between the salary of the salesman's and the amount of sales they successfully getting. The more salary they get the more money they are making.

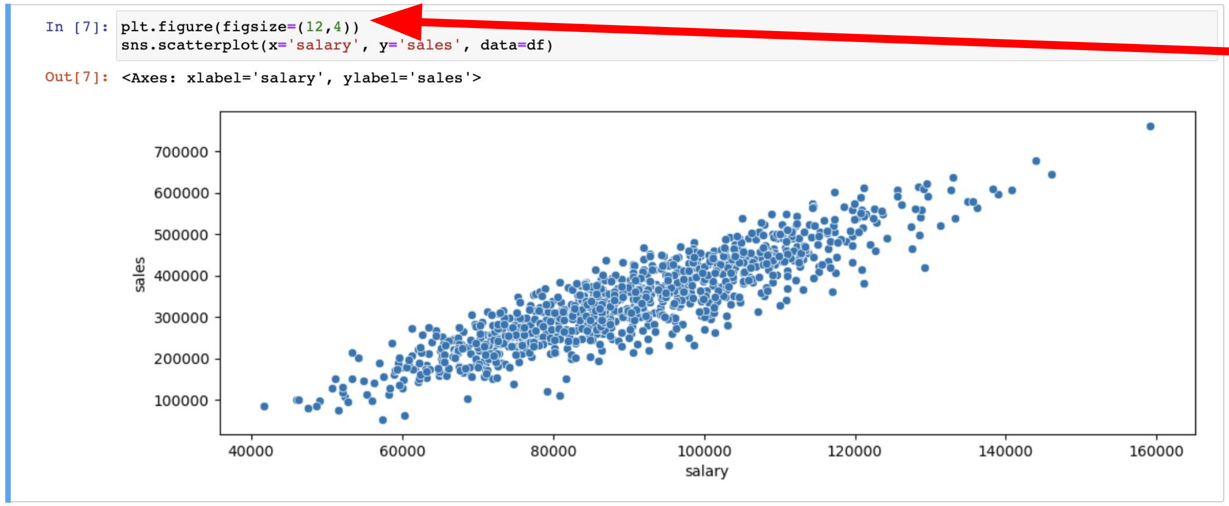
ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Seaborn - Scatter Plot

As mentioned, Seaborn as an abstraction of the Matplotlib library that allowing us to use the Pandas Dataframes themselves without any change necessary.

What happening in the background is that Seaborn is creating a Matplotlib figure and using Matplotlib methods to create the chart we want. Meaning we can also use Matplotlib figure methods to allow some customization on the plot.

For example → Let's say we want to change the figure size, we can use the **figsize** parameter before creating the scatter plot



```
In [7]:  plt.figure(figsize=(12,4))
         sns.scatterplot(x='salary', y='sales', data=df)

Out[7]:  <Axes: xlabel='salary', ylabel='sales'>
```

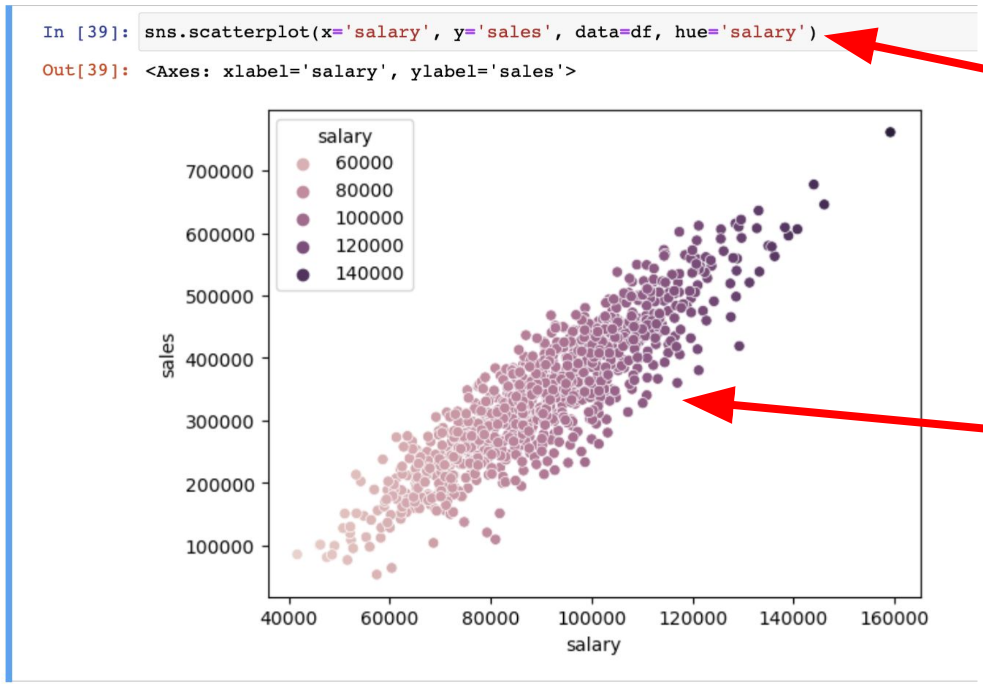
We customize the Matplotlib figure size before creating the scatter plot

Seaborn - Scatter Plot Parameters

'hue' parameter → Allow us to add additional segmentation to the scatter plot, we can think about it as <u>adding groups to the chart</u> so it will provide more information or will be better understandable.

We can also use one of the axis data and provide it as hue.

For example →



```
In [39]: sns.scatterplot(x='salary', y='sales', data=df, hue='salary')
Out[39]: <Axes: xlabel='salary', ylabel='sales'>
```

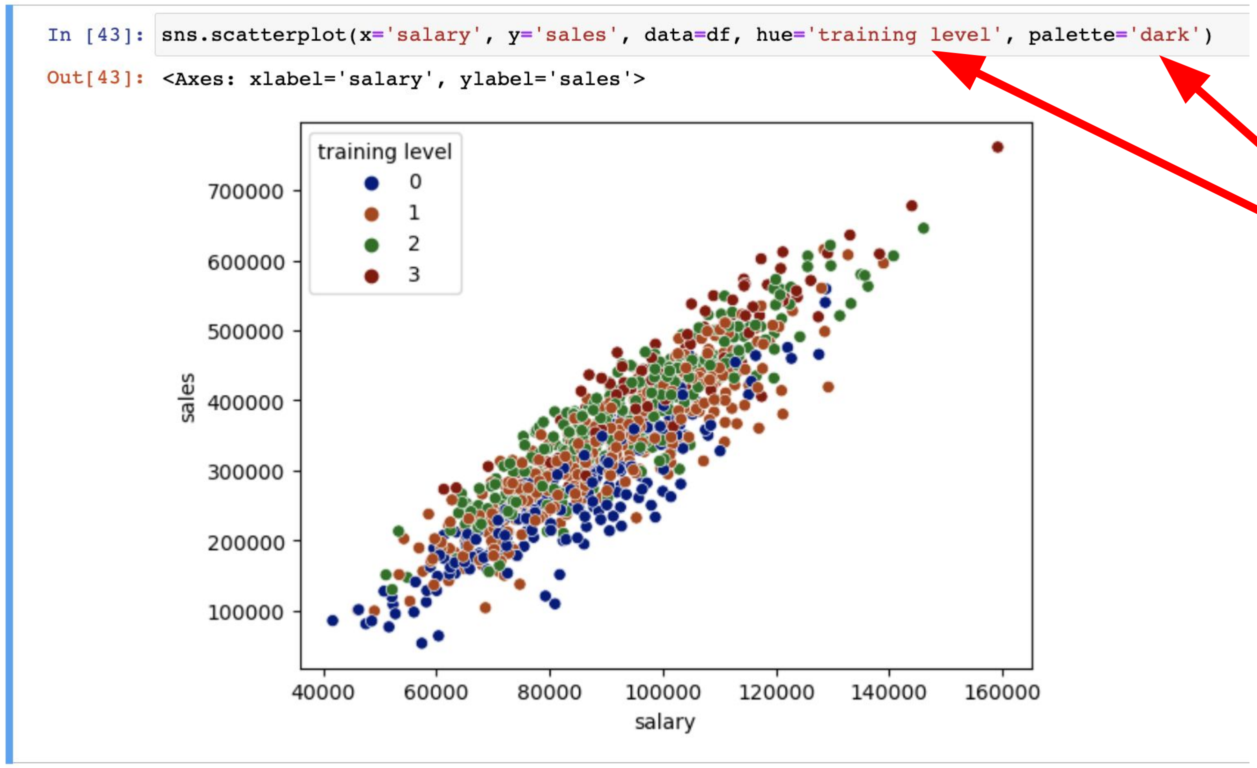
We are using the 'salary' column as for the x-axis and for the hue.

The scatter plot added additional segmentation according to the salary amount, providing us more visibility about the different salary groups

Seaborn - Scatter Plot Parameters

‘palette’ parameter → We can determine the colors family the hue will be using by providing the **‘palette’** parameter. In Seaborn documentation we can find the different types of colors we can provide to the palette parameter.

For example →



```
In [43]: sns.scatterplot(x='salary', y='sales', data=df, hue='training level', palette='dark')
Out[43]: <Axes: xlabel='salary', ylabel='sales'>
```

Now We are using the ‘training level’ column as for the x-axis and for the hue, and choosing ‘dark’ colors

Seaborn - Scatter Plot Parameters

‘size’ parameter → Determine the size of each dot in the scatter plot according to column value. The bigger the chosen column value the bigger the dot will be in the plot.

‘s’ parameter → Determine the size of each dot according to fix value.

‘alpha’ parameter → Determine the amount of transparency of each dot in the scatter plot. The alpha parameter takes values between 0 - 1 while 0 meaning highest transparency and 1 meaning no transparency. Using transparency can be useful when we have multiple dots at the same place so we could see them all and not just one of them.

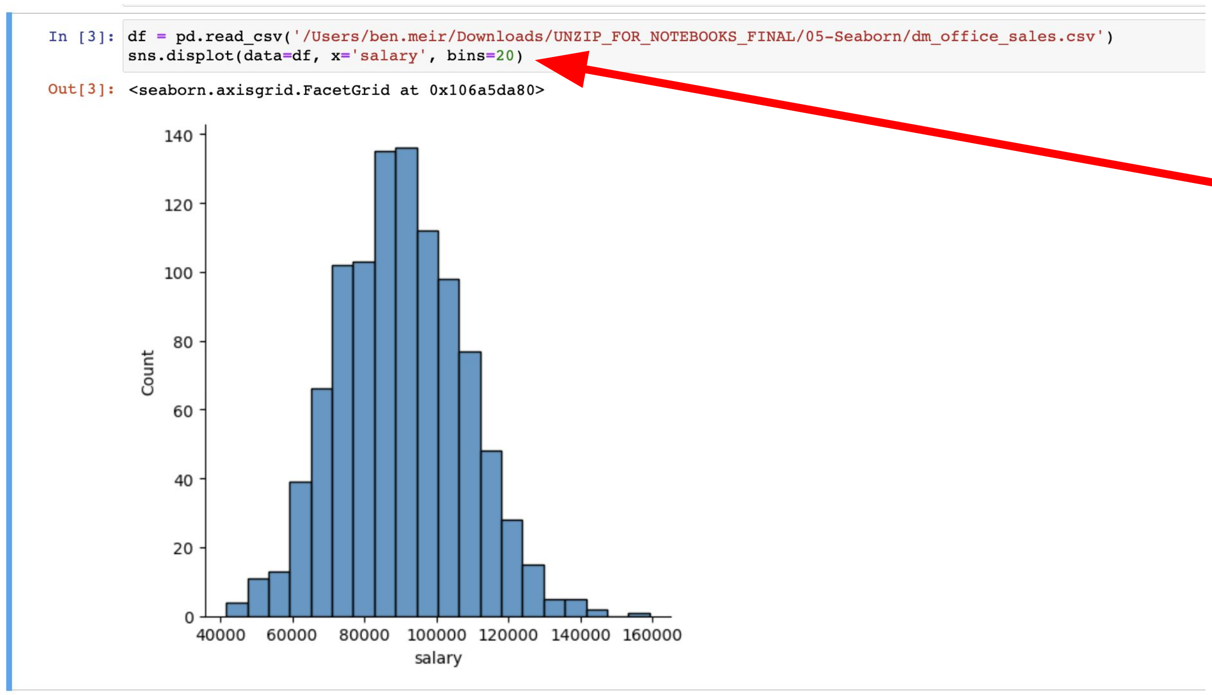
‘style’ parameter → Determine the style of the dots in the scatter plot, same as with the ‘size’ parameter we need to pass to the style parameter a column value and according to this value the scatter will change the style of the dot. This allow us to add another segmentation style to our scatter plot.

Note: Saving a Seaborn plot will be the same as with Matplotlib (using **plt.savefig()** method)

Seaborn - Histogram

sns.displot() → Same as we saw with the Matplotlib library, we can also easily create Histogram chart using the displot() method. When creating a histogram chart we need to provide the data for the x-axis and the number of bins. While selecting more bins will create more accurate histogram it can also make it more noisy and less understandable.

For example → Let's explore the salary distribution between the workers



```
In [3]: df = pd.read_csv('/Users/ben.meir/Downloads/UNZIP_FOR_NOTEBOOKS_FINAL/05-Seaborn/dm_office_sales.csv')
        sns.displot(data=df, x='salary', bins=20)

Out[3]: <seaborn.axisgrid.FacetGrid at 0x106a5da80>
```

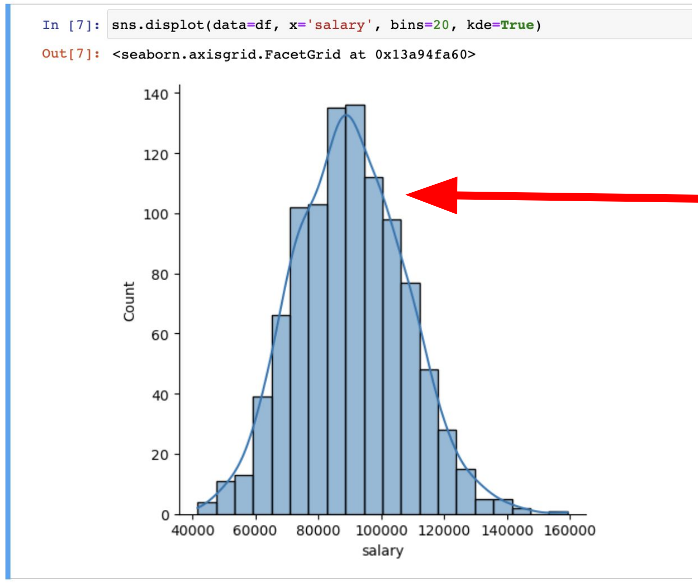
We created a histogram of the salaries between workers and selected 20 bins as the number of bins in the distribution

Seaborn - Histogram

‘kde’ parameter → The **‘kde’** parameter is a boolean parameter that will add additional KDE plot on top of the histogram itself.

If we will want to generate only the KDE chart without the histogram itself we can use **sns.kdeplot().**

KDE plot → KDE stand for kernel density estimate and used for visualizing the distribution of observations in a dataset, analogous to a histogram. By using KDE we can estimate probability distributions based on the simple intuition that the more data points in a sample that occur around a location, the higher the likelihood of an observation occurring at that location.



```
In [7]: sns.displot(data=df, x='salary', bins=20, kde=True)
Out[7]: <seaborn.axisgrid.FacetGrid at 0x13a94fa60>
```

The KDE plot line on top of the histogram itself

Seaborn - Count Plot

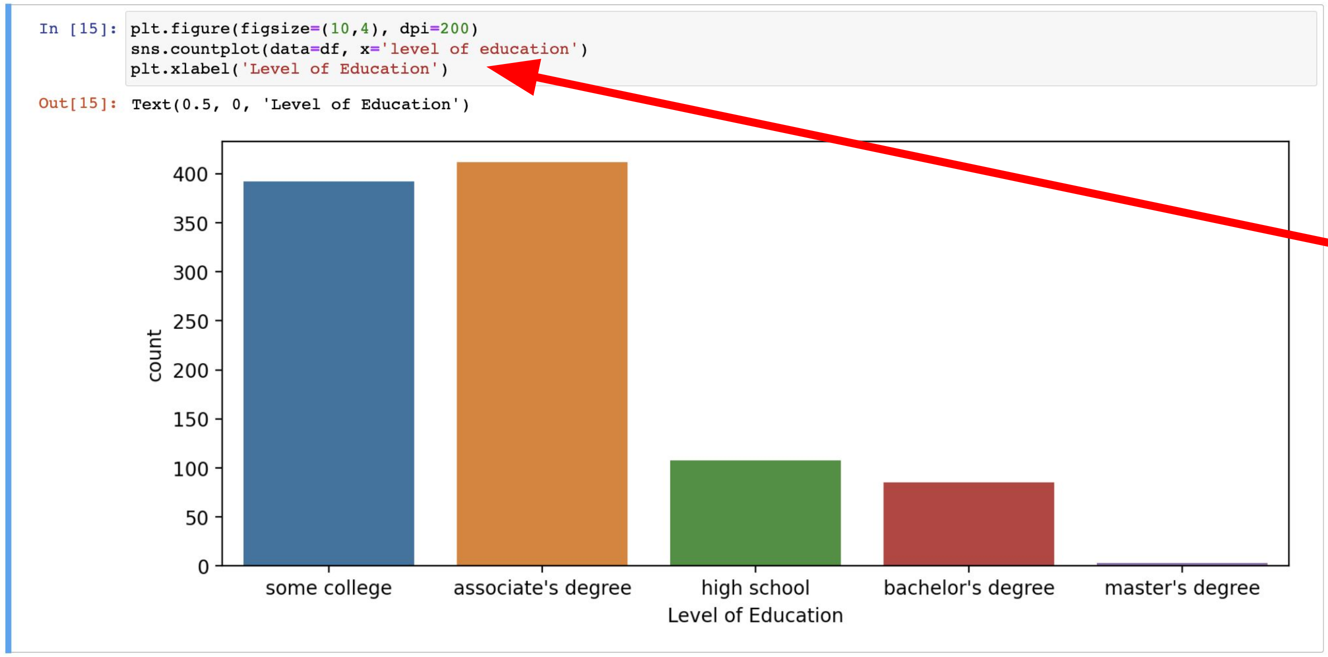
When dealing with categorical data, seaborn allow us to select from 2 different types of plots

The most basic plot in this group is **countplot**, which is a simple plot that allow us to get the count of rows for each category.

sns.countplot() → Method that will generate a simple countplot based on the category data provided.

In count plot the y-axis will always be the count metric for each category.

For example → Let’s make a simple countplot according to the employees level of education categorical data



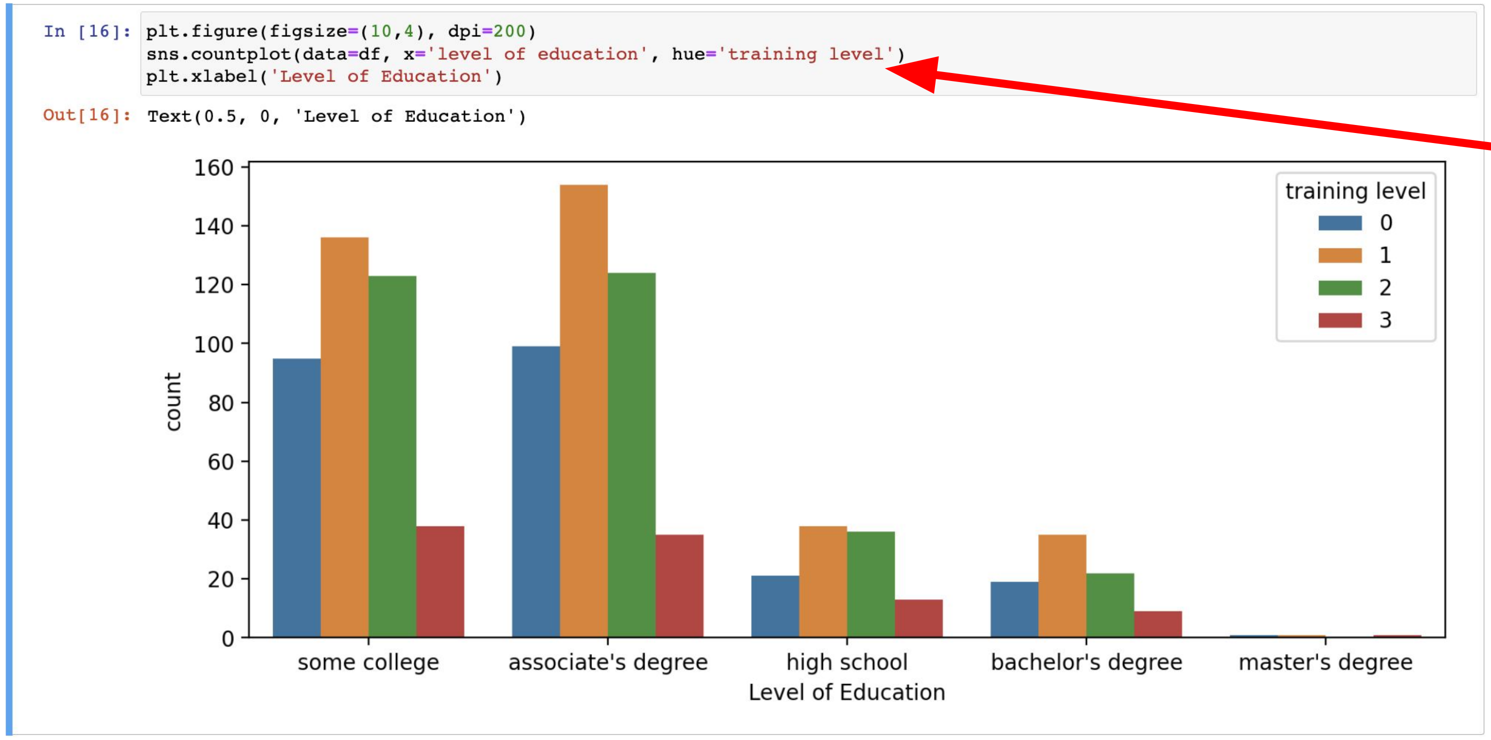
```
In [15]: plt.figure(figsize=(10,4), dpi=200)
         sns.countplot(data=df, x='level of education')
         plt.xlabel('Level of Education')

Out[15]: Text(0.5, 0, 'Level of Education')
```

We created a count plot of the level of education categories. The plot provide how many workers we have in each category.

Seaborn - Count Plot

‘hue’ parameter → Same as with other Seaborn plots we can also provide **‘hue’** parameter to create additional division inside our count plot.

For example → Let’s add additional division of ‘training level’ for each ‘level of education’ category



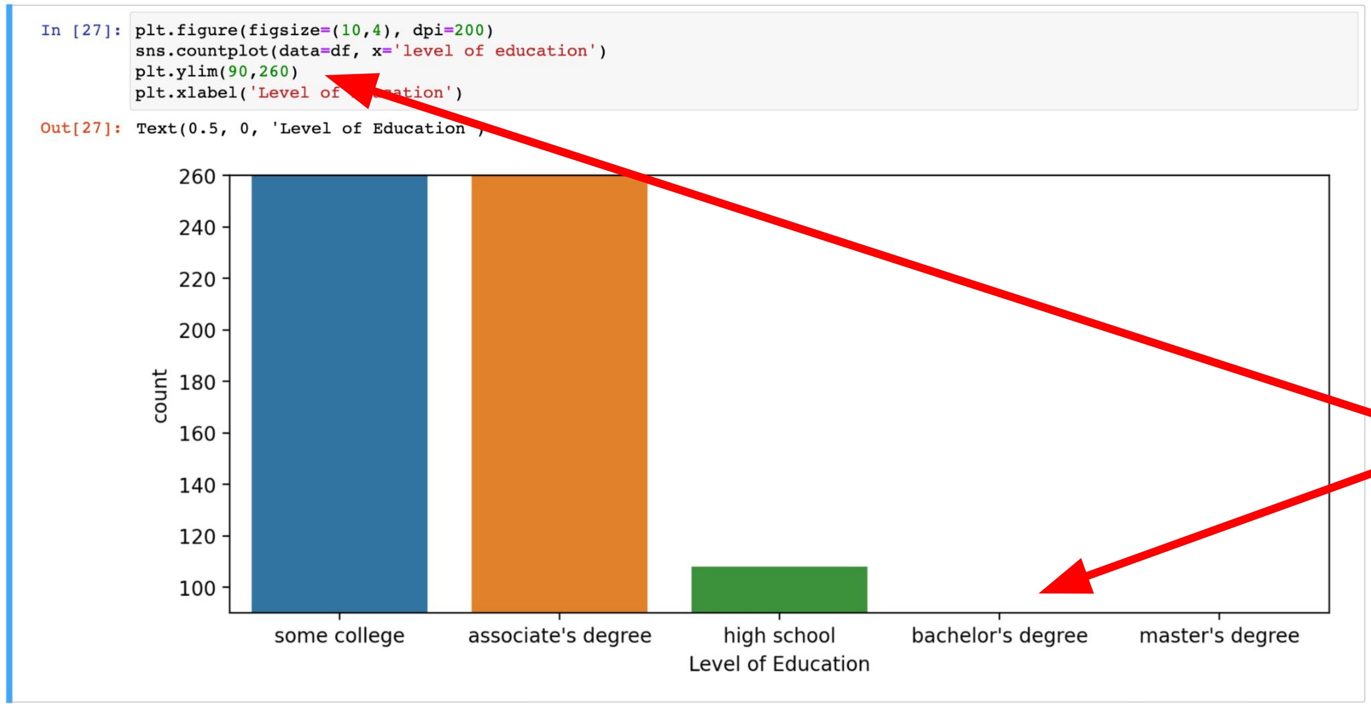
```
In [16]: plt.figure(figsize=(10,4), dpi=200)
         sns.countplot(data=df, x='level of education', hue='training level')
         plt.xlabel('Level of Education')

Out[16]: Text(0.5, 0, 'Level of Education')
```

We added additional deviation of ‘training level’ for each category using the ‘hue’ parameter

Seaborn - Count Plot

Count plots can be very deceiving if we are not using them right because intuitively we can think about the bar height compare to the other bars as a comparison indicator.

For example → let’s take a look at the previous count plot we created and change the ylim so it will start from 90 instead of 0.



```
In [27]: plt.figure(figsize=(10,4), dpi=200)
         sns.countplot(data=df, x='level of education')
         plt.ylim(90,260)
         plt.xlabel('Level of Education')

Out[27]: Text(0.5, 0, 'Level of Education')
```

We can see that by changing the ylim range the same data will look differently so we could think that we don’t have bachelors degree workers at all which is of course not the case.

Seaborn - Bar Plot

Another type of categorical plot Seaborn provide is **bar plot**. Bar plot enable us to create any type of statistic metric calculated for each category (Those statistic metrics could be count, mean, sum, std, ect..). We can think of count plot as one implementation of the bar plot which the statistic metric is the count value.

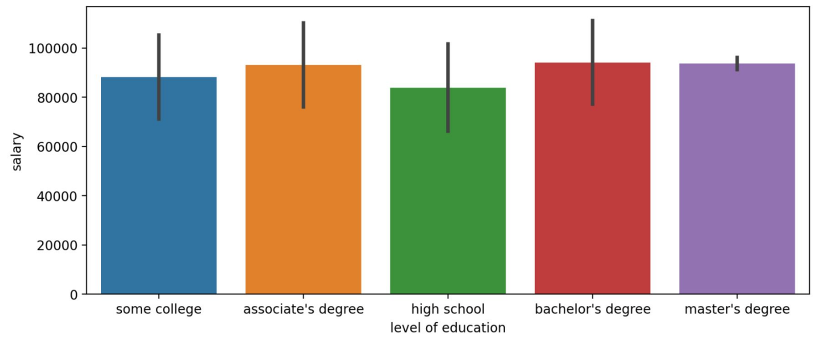
sns.barplot() → Allow us to generate a bar plot by specifying the categorical data and the statistic metric we want to use. This metric should be passed as the value of the **'estimator' parameter.**
Unlike the count plot, when using bar plot method we also need to specify the y-axis data.
'errorbar' parameter → We can add to the bar plot the **'errorbar'** parameter specified with **'sd'** value This will add additional line on top of each bar that should show the standard deviation.
'hue' parameter → Same as with other plots in Seaborn we can add additional division by using the **'hue'** parameter.

Note: In a lot of cases we will want to show statistical data in a table rather than in a plot like bar plot.

Seaborn - Bar Plot

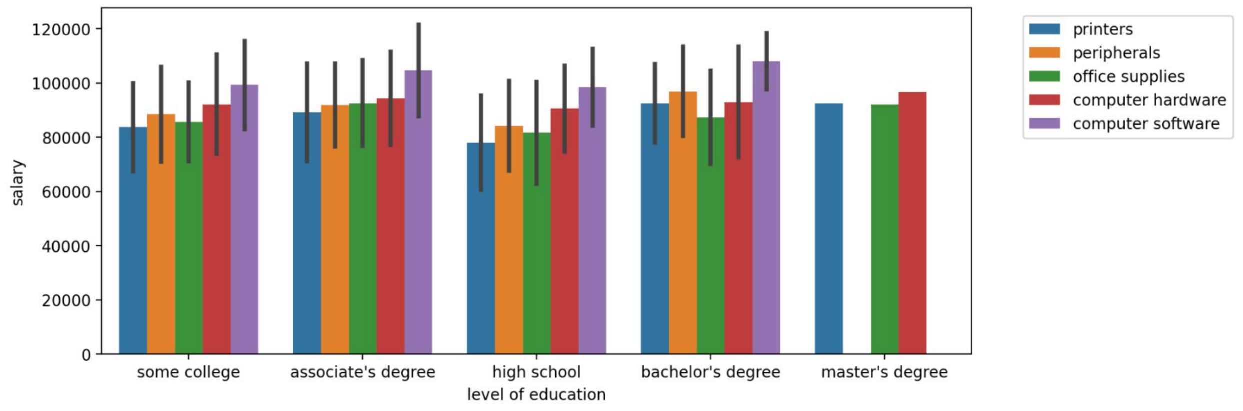
For example → Let’s show the mean of all the salaries according to each ‘level of education’ category

```
In [22]: plt.figure(figsize=(10,4), dpi=200)
         sns.barplot(data=df, x='level of education', y='salary', estimator=np.mean, errorbar = 'sd')

Out[22]: <Axes: xlabel='level of education', ylabel='salary'>
```



Now, let’s use the ‘hue parameter to add additional data according to the worker ‘division’ in each category

```
In [25]: plt.figure(figsize=(10,4), dpi=200)
         sns.barplot(data=df, x='level of education', y='salary', estimator=np.mean,
                     errorbar='sd', hue='division')
         plt.legend(bbox_to_anchor=(1.05,1))

Out[25]: <matplotlib.legend.Legend at 0x175642fb0>
```



Seaborn - Boxplot

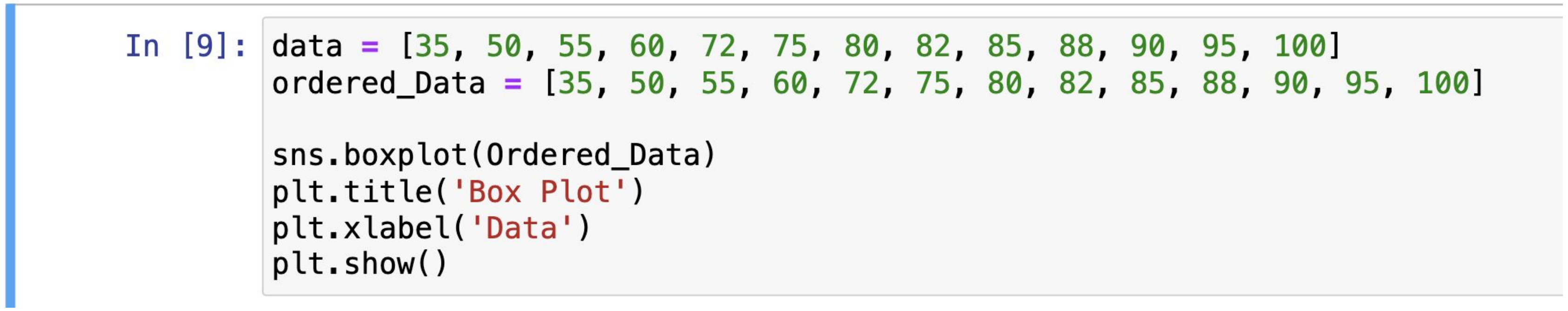
Until now, we only saw plots that can show the values distribution of a specific data set (histogram) or some basic metric statistic for each category.
But what if we will want to create a plot that show more than just one statistic value?
For that we have the **Box Plot**.

Box Plot → A box plot is a graphical representation of statistical data based on a five-number summary: the minimum, the maximum, the median (middle), and the first and third quartiles.

IQR (Interquartile Range) → In a box plot, IQR is a measure of statistical dispersion, or in simple terms, the spread of the middle 50% of the data values. It is the range between the first quartile (25th percentile, also represented by Q1) and the third quartile (75th percentile, also represented by Q3).
IQR = Q3 - Q1

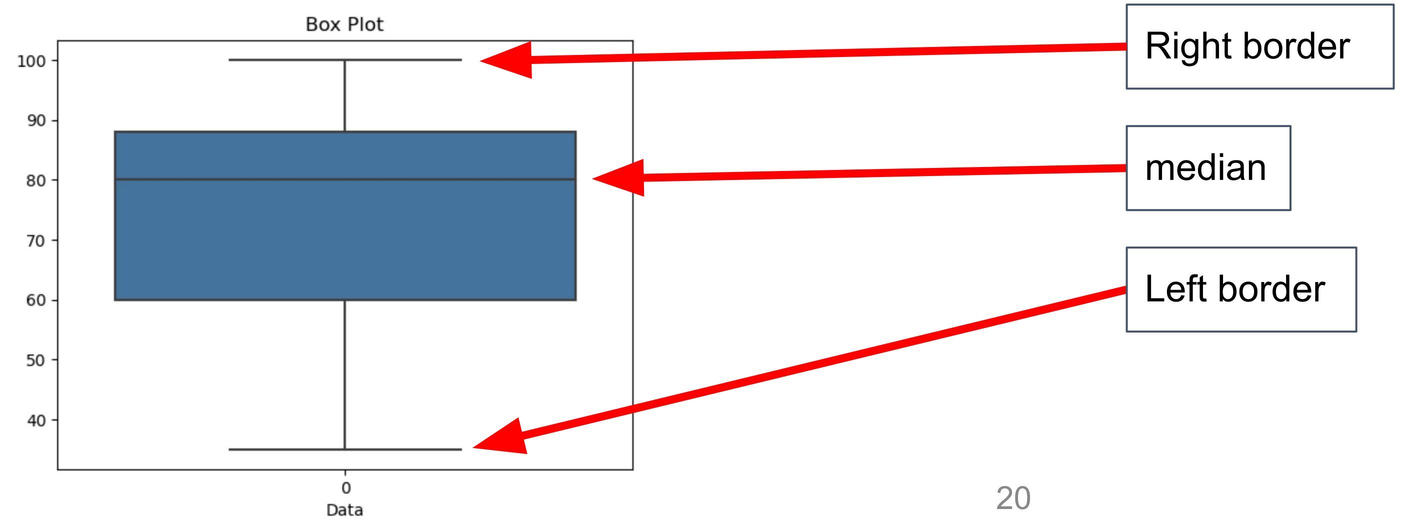
Seaborn - Boxplot

Let’s take a simple box plot example to better understand what it’s providing us:



```
In [9]: data = [35, 50, 55, 60, 72, 75, 80, 82, 85, 88, 90, 95, 100]
        ordered_Data = [35, 50, 55, 60, 72, 75, 80, 82, 85, 88, 90, 95, 100]

        sns.boxplot(Ordered_Data)
        plt.title('Box Plot')
        plt.xlabel('Data')
        plt.show()
```

When generating a simple boxplot using the given data we will get the following chart:



Seaborn - Boxplot

We get the following quartiles:

- Q1 (1st Quartile or 25th percentile) = 55 (value at 1/4th of the data)
- Q2 (2nd Quartile or 50th percentile), which is also the median = 75 (value in the middle)
- Q3 (3rd Quartile or 75th percentile) = 88 (value at 3/4th of the data)

The Interquartile Range (IQR) = Q3 - Q1 = 88 - 55 = 33.

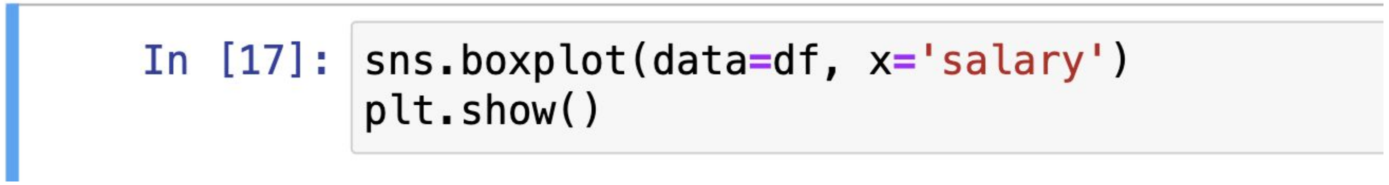
The box would start at Q1 (55) and end at Q3 (88), making the height of the box equal to the IQR (33).

The line inside the box represents the median or Q2 (75).

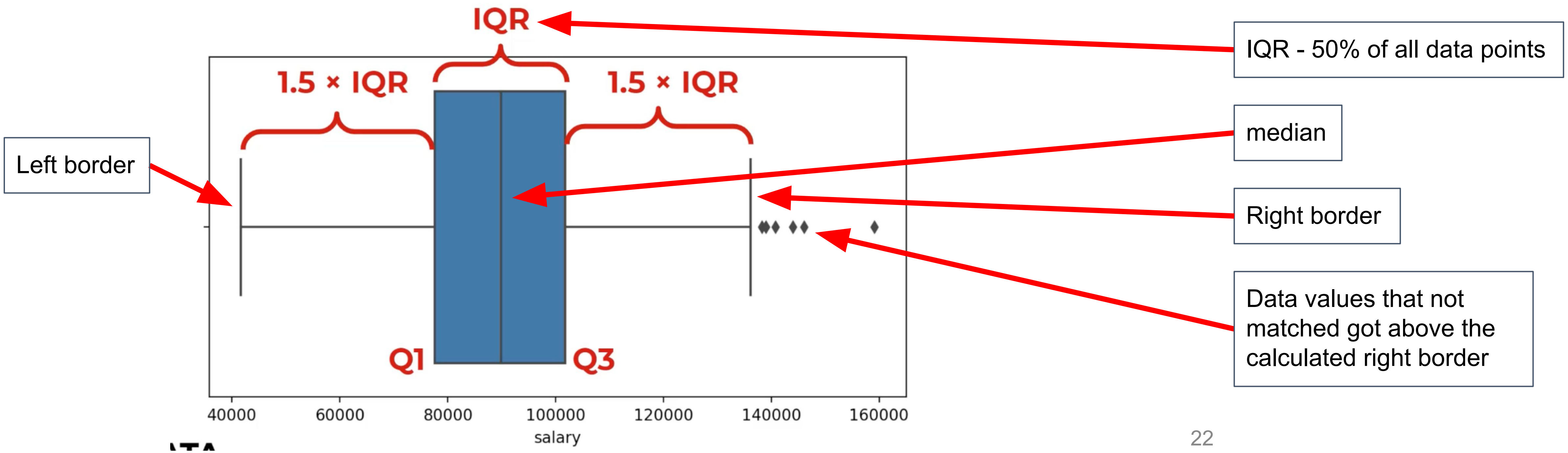
The 'whiskers' of the box plot (the lines extending from the box) typically represent the minimum data point (35) within Q1 - 1.5 * IQR and the maximum data point (100) within Q3 + 1.5 * IQR

Seaborn - Boxplot

Let’s generate the salary distribution from our previous csv example:



```
In [17]: sns.boxplot(data=df, x='salary')
         plt.show()
```



Class Exercise - Seaborn Charts

Instructions:

For this exercise use the **tips.csv** file, load the file into a proper Dataframe.

Create Matplotlib charts that implement the following:

- Create a scatter plot to show the correlation between total_bill and tip.
  Add additional information about the gender distribution.
- Evaluate if smokers tend to pay more for their meals as compared to non-smokers, use a bar chart to visualize the comparison.
- Create a count plot for the 'day' column to examine which day has the most orders and which day has the most less orders.
- Plot a histogram for the 'total_bill' column, visualize the ‘total_bill’ distribution.
- Make a box plot to display the statistical distribution of 'total_bill' for each 'day'.

Class Exercise Solution - Matplotlib Charts