Seaborn :-

Overview of plotting fns :-

1) Similar fns for similar tasks :- Plots are categorized to different categories depending upon the tasks they perform.

There are three different categories :-

1) Relational :- Scatter, Line.

2) Distribution :- Histogram, kernel density plots.

3) Categorical :- Box plot, violen plot, Bar chart.

2) Figure level vs Axis level fns :-

1) Responsible for plotting in a figure.

1) Responsible for plotting in a single axis.

- Can call different axis level fns.
- Each seaborn module has a single figure level fn.
- Can create figures with multiple subplots.

relplot (relational)	displot (distributions)	catplot (Categorical)
1) Scatter	1) histplot	1) Stripplot
2) Line	2) kdeplot	2) Swarmplot
	3) ecdfplot	3) boxplot
	4) rugplot	4) violinplot
		5) Pointplot
		6) Barplot

→ Axis level fns are replacement to matplotlib fns :-

- They only act on a specific axis.
- Legend is always inside the axis.
- They can be integrated to a figure & still work.

→ Figure level fn own their figure →

- Returns an object that could be used to customize the figure.

→ unlike matplotlib → figure level fn adjust figure size & not axis size.

height, aspect → Input parameters.
width = height × aspect → Calculated automatically.

→ Most plots are composed using figure level fns but for custom & nested plots axes

1) Call a figure in matplotlib.
2) Call seaborn axis fns on the figure.

Data Structures accepted by Seaborn →

→ Traditional data

Both long form & wide form data are accepted by Seaborn.

→ Pivot table.

Long Form	Wide Form
1) Has many advantages & is preferred.	- only three variables at play so representation options are limited.
2) Different columns can be assigned to different variables in the plot. → making it more customizable.	

→ Tidy data always preferred over messy or repeated data.

→ Python lists, dicts or np.arrays().

→ Vectors or collection of vectors are accepted but they lose name information. → This is used for naming purposes (axis labels.)

Seaborn's Object interface → Built to provide end to end plot specification & customization without dropping to matplotlib.
- Although dropping to matplotlib is also possible.

```
import seaborn.objects as so
// Specifying a plot & mapping
(
    so.Plot( df,  x=, y=, color, .... )
    .add( so.Dot( mark properties )
)
```

a) Setting properties → properties are set with the direct property name instead of function api names.
ex → color vs hue

b) Mapping properties → data is mapped when passed to the plot fn & is treated as a direct # when passed to Dot fn.

ex →
```
( so.Plot(Penguins, x, y, color = 'sex')
)
```
// Color is mapped to sex column.

vs

```
( so.Plot( penguins, x, y )
  .add( so.Dot( color = 'green' )
)
```
// color has a direct value of green.

so.Dot() → mark object that represents a single data point.

more instances → so.Bar(), .Line()
- Different objects for different chart types.

- Each mark object can be given a statistical transformation to be displayed by the mark layer.
ex → aggregation, regression line ....

3) Building & Displaying the plot

1) Adding multiple layers - multiple mark objects can be added on the same plot to display different information.

```
(so.Plot(df, x, y, color)
 .add(so.Dot())
 .add(so.Line(), so.PolyFit())
)
```

agg operation.

adds a line on top of Dots.

Variable mappings used in the Plot() constructor will be used by all layers.

2) Layer Specific mappings - There are three ways to specify mapping in layers

1) Provide the mapping in Plot object.

```
so.Plot(df, x, y, color)
```

2) Provide a mapping in layer

```
so.Plot(df, x, y, color='sex')
 .add(so.Dot(), color='sex'/None)
```

To set the mapping from Plot constructor.

Visualizing Stastical Relationship →

relplot() → (figure level) → Plots the relationship b/w two variables.

1) Scatter plot → Plots joint distribution b/w two variables with each point representing a data point. (default option for rel plot).

Diamention of the plot can be increased by adding additional variables to the plot with mappings

1) hue → Color → Different for different values
2) style → Shape → of the third variable.
3) size

Unlike matplotlib, sizes are calculated using a normalized range.
- A custom range can be provided using a tuple.

relplot(... , size = 'var', sizes = (20, 2000) ...)

2) Line plot → use when one variable is continuous.
ex → price of stock vs time.

relplot(df, x, y, kind = 'line', hue/style/size, → mappings for additional vars.

errbar , estimate = 0/1) → Turn on/off aggregation.
→ by default prints 95 % confidence interval.
→ = 'sd' → standard deviation instead of confidence interval.
→ = None → No intervals.

Computing estimates can be time consuming.

dashes = False → disables dashes.
markers = True → Prints markers at datapoint.
units = 'var' → Plot the distribution of each group.
→ categorical variable.

Visualizing Distribution of Data :-

1) Histplot :- Represents the freq of occurance of each category of a discrete variable as bars height.

1) binwidth , bins :- adjust these parameters to have a good representation of the distribution.

- To get accurate dist of data.

2) Conditioning :- (hue) :- Color the bars according to a 2nd discrete variable.

3) Statistic :- (stat) :- To change from freq. to a different stat like probability/density.

- Different bars height for different statistics.

Q:- Multiple bars overlapping because of conditional rendering :-

1) element = 'step' :- avoids overlap.

2) multiple = stack / dodge :- Places opaque bars side by side

↳ Stacks opaque bars on top of each-other.

2) Kernel Density Estimate :- Displays data's prob. dist.

- Assumes that the data is smooth.

1) bandwidth :- bin size

bandwidth	smoothness
high	high
Sweet spot in the middle.	
low	rugged.

2) Conditioning on other variables is easier than histogram.

a) multiple = stack :- Stacks KDEs on top of each-other.

b) fill = True

↳ Prints KDEs with different opacities.

3) Drawbacks :- 1) Not suitable for discrete variables.

2) Confusing for extreme observations.