Seaborn plots :-

- **Categorical**
  - countplot
  - Barplot
  - Boxplot
  - violinplot
  - stripplot
  - swarmplot
  - Factorplot
- **Distribution**
  - Displot
  - jointplot
  - pairplot
  - Rugplot
  - Histplot
  - KDE plot
  - ECDplot
- **Relational**
- **Regression**
  - simple linear plot
  - lineal plot with additional parametus
  - setting the size and color of plot
  - Displaying multiple plots.
  - size and aspect ratio of the plots
- **matrix**
  - Heatmaps
  - clustermaps
- **multi-plot grids.**

Seaborn is a python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

⇒ using different color palette.

```
sns.countplot(x='product', hue='Gender', data=mart,
              palette='gist-gray')
```

⇒ change style using facecolor, linewidth and edge color

```
sns.countplot(x="", data=mart,
              facecolor=(0,0,0,0),
              linewidth=5,
              edgecolor=sns.color_palette('dark',3))
```

Barplot :-

A barplot represents an aggregate or statistical estimate for a numeric variable with the height of each rectangle and indicates the uncertainty around that estimate using an error bar. Bar plots include 0 in the axis range, and they are a good choice when 0 is a meaningful value for the variable to take

- **Relational plots:**
  this plot is used to understand the relation between two variables.
- **Categorical plots:**
  This plot is deals with categorical variables and how they can be visualized
- **Distribution plots:-**
  This plot is used for examining univariate and bivariate distributions.
- **Regression plots:-**
  The regression plots in seaborn are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analysis.
- **matrix plots:-**
  A matrix plot is an array of scatterplots.
- **multi-plot grids:**
  It is a useful approach to draw multiple instances of the same plot on different subsets of the dataset.

⇒ ~~fig~~

figure (figsize = (x, y) (15, 5)

↑ axis.

It ~~sets~~ the expands the x axis and y axis by seting the figsize (figure size of a plot)

⇒ sns. countplot (x = "product line", data = mart).

categorical data.

X label = product line

Y label = count.

⇒ horizantal ~~bar~~countplot :-

sns. countplot (y = "product line", data = mart)

Xlabel = count
Ylabel = product line } displays horizantal

⇒ Add (hue) to get the count on two categories.

ex product line & gender.

sns. countplot (x = 'product line', hue = 'Gender', data = mar

⇒ Display the seaborn dataset names and load one of them

```
sns.get_dataset_names()
```

⇒
```
a = sns.load_dataset('dataset')
datas  a.head()
```

⇒ **Barplot :-**

```
plt.figure(figsize=(15,5))
sns.barplot(x='product', y='total', data = xxx).
```

hue = categorical data

⇒ to plot a bargraph in order

```
x = mart['product line'].sort_values()
  x.unique()
```

⇒ `sns.barplot(x='', y='', data=xxx, order=['',''],`
`hue_order = ['male','Female'])`

⇒ **Add Cap on the Error bars :-**

`, capsize=0.2`

→ Remove the Errorbar using ci `ci=None`

→ Change bar color using color attribute. `color='...'`

→ Palette --- pattern of colors.

palette = '---'

↳ Saturation --- it changes the color brightness & low or high.

Saturation = 5

→ change default aggregation method using estimator parameter.

estimator = sum
estimator = np.median
estimator = mean

} it changes the y axis values. depends on given estimator.

orient = "v" | "h" | "x" | "y".

orientation of the plot (vertical or horizontal)

fill : bool

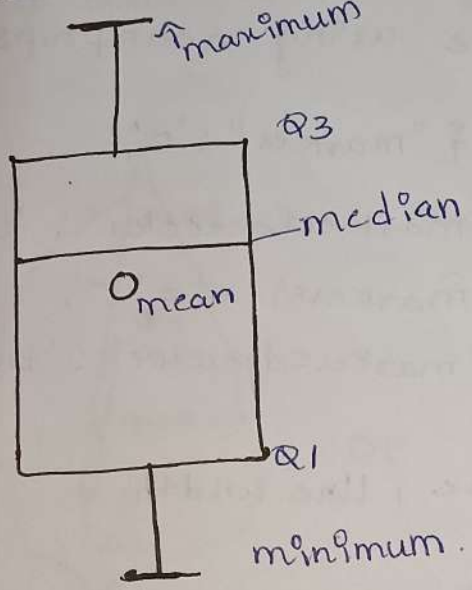
if True, use a solid patch. otherwise draw as line art

Box plot :-

The box plot provides a summary of supplied data, which includes the information like:

- → mean
- → median
- → minimum
- → maximum
- → 1st Quartile
- → 3rd Quartile
- outliers

⇒ compars b/w variables or across levels of a categorical variable



⇒ boxplot with one variable of numeric data

sns.boxplot(y='---', data=xxx, width=0.2)

→ Boxplot one numeric variable by a CATEGORICAL variable.

x axis = categorical

y axis = Numeri.

⇒ box plot on one numeric variable by Two categorical Variable using hue attribute.

⇒ sns.boxplot(x='', y='', hue='', data=xxx)

~~hue= classification~~.

→ Add MEAN marker in the plot using showmeans attribute and change its style using meanprops.

default showmeans = False, we have to set showmeans = True

⇒ changing style using meanprops.

```
meanprops = {"marker":'o',
             'markerfacecolor': 'white',
             'markersize' = '5',
             'markeredgecolor': 'black'})
```

→ change palette, line width

→ palett = 'bbaeik'

↓ change color inside the bar.

→ linewidth = 5 → it changes the edge width size or outline size of a bar.

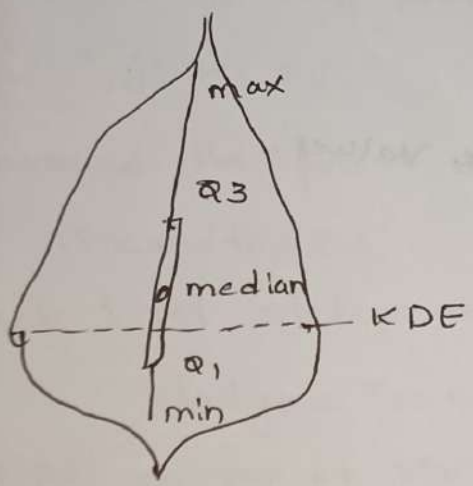
⇒ Create box plot for each of the numeric variable variable in the dataframe.

sns.boxplot(data=xxx)

↓ by default it plot the numeric datatrame

⇒ <u>Violin plot</u> :-

combining boxplot and Kernal density plot together called violin plot.



→ A violin plot plays a similar role as a box and whisker plot. It shows the distribution of data points after grouping by one (or more) variables. unlike a box plot, each violin is drawn using a Kernel density estimate of the underlying distribution

⇒ sns.violinplot(y="", data=xxx)

⇒ a violinplot on two categorical and one nu

sns.violinplot(x="", y="", hue="", data=xxx, split=True

⇒ Change the box in the violinplot to horizantal lines.

inner='quartile'

⇒ line for each observations in a violinplot

inner='stick'

→ smoothing using bw attribute.

bw=1.2

→ cut out the extreme values.

cut=0.

Strip plot :-

a strip plot is a graphical data analysis technique for summarizing a univariate data set

→ A strip plot can be drawn on its own, but it is also a good complement to a box or violin plot in cases where you want to show all observations along with some representation of the underlying distribution.

⇒ sns.stripplot(x='', data=xxx)

→ expand markers in strip plot using jitter

hue="", jitter=5.

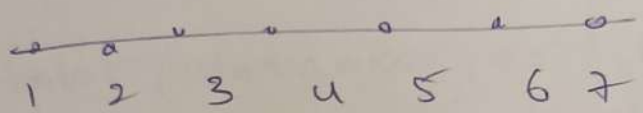
draw line around the points using linewidth

linewidth=0.5

separate each level of hue using dodge

dodge=True

draw the strips on top of violin plot



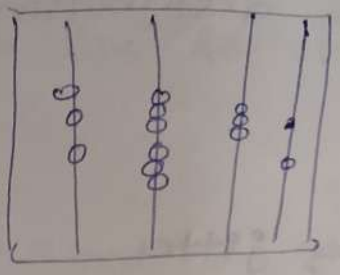
Swarm plot :-

why can't we do that in strip plot ~~then~~

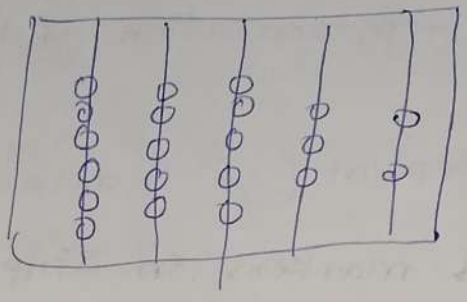
Same data points overlaps in a strip plot.

[1,1,1,1,1,1,1,2,2,2,2,3,3,3,3,3,5,5,6,6,6,6,6]

Strip plot



Swarm plot :-



→ catplot , kind , row = 'custom type',

or

~~F~~actor plot

Distribution

1) Histplot() :-

distribution which can done in between

0 - 200

201 - 400

401 - 600

- - -

- - -

→ A histogram is a classic visualization tool that represents the distribution of one or more variables by counting the numbers of observations that fall within discrete bins

→ normalize the statistic computed within each bin to estimate frequency, density or probability mass,

⇒ binwidth = used to increse the width of bar plots.

→ bins — increase the no of bars in the interval.

→ sns.histplot(data=xxx, x='', bins = np.arange(0,1100,50))
plt.xticks(np.arange(0,1100,50))

→ use a categorical variable in hue and stack it using multiple argument.

sns.histplot(data=xxx, x="", hue='', multiple='stack')

⇒ make it a step/poly plot using Element arguement and change the fill.

sns.histplot (data=xxx, x="", element='step', fill=False

element: Total fill the color.

fill: gives outer line

element = 'poly'

⇒ Stat = probability ; ---

⇒ shrink = 0.4 - --> it increases or decreas the gap b/w the bars.

KDE plot - Kernel density Estimation plot.

⇒ Adjust the smoothing using bw_adjust.

. bw_adjust = 0.2

⇒ group the KDE on a category variable.

(hue):

⇒ Stack KDE on a category using multiple argument

hue, multiple = 'stack'

→ change styling of hued KDE using line width, palette, alpha etc.

⇒ linewidth — increase or decrease the width of plot.
↘ say 0.5, 0.7 ---

→ palette = 'black'

⇒ alpha = it change the total frame brightness.
say 0.6, 0.7 ---

→ create a bivariate KDE.

x = '', y = ''

→ group the bivariate KDE on a categorical variable and show the contours.

x = '', y = '', hue = ''

fill = True → fill the plot color.

levels = 5 + -- it shows the contours.

(1) Rug plot :-

Plot marginal distributions by drawing ticks along the x and y axis.

→ this function is intended to complement other plots by showing the location of individual observations in an unobstrusive way.

⇒ Rug plot for two variable :-

x=" " , y=" "

→ group it by a categorical variable using hue :-

x=" ", y=" " , hue =" "

⇒ height = float.

→ combine KDE to rug plot.

kdeplot (x=, y=, fill = True.

scatterplot.

→ show rugs outside of the Axis.

height = -0.04, clip_on = False.

ECDF ⇒ empirical cumulative distribution functions.

⇒ Represents the proportion or count of observations falling below each unique value in a dataset.

income — below 10, below 20, below 50 } cumulative distribution

⇒ ~~sns.ecdf(data=~~

→ sns.ecdfplot(data=xxx, x="")
hue="", stat='count'

(13) displot ⇒ distribution plot :-

⇒ sns.displot(data=mart, x='Total', kde=True, rug=True)

⇒ rug_kws={'height':0.5}, kde_kws={'bw_adjust':.9})
hue='Gender') → if we have more values
multiple='stack' ↙

→ if we want to separate the two columns we have to use

col='Gender' → the values which are present in this col, shows in different graphs.

⇒ 1 row for each col.

row = "sex".

⇒ kind = 'ecdf'.

⇒ multiple only work on bar & histplots.

---

jointplot :-

Draw a plot of two variables with bivariate and univariate graphs.

⇒ iris data.

⇒ sns.jointplot(data=iris, x='sepal_length', y='petal_le')

⇒ change its kind to 'scatter' | 'kde' | 'hist' | 'hex' | 'reg' | 'resid'

kind = 'scatter', 'kde' ---

'reg' → regression

⇒ group the categorical data.

hue = 'species'

⇒ color = 'Red', palette = 'BOPU',

→ joint_kws = dict(marker = '+'), → now graph in scatter so we have to use scatter arguments

color = 'Red'

for margin distribution we have to use

⇒ margin_kws = dict(color='green', Kde=True) → this is in histogram so we have to use histplot argument

→ for changing the format of individual plots in a joint plot.
for scatter & hist plot

→ Adjust height, ratio, space and show/hide the marginal_ticks.

sns.jointplot(data=xxx, x="", y="",
height = 9, (change total height of the graph)
ratio = 2 (change the size (or) ratio of plots)
space = 5, (change the marginal plot & relational plot
marginal_ticks = bool (True, show the count/density axis of the marginal plots.

⇒ plot KDE and RUG on top of joint plot:-

1 = sns.jointplot(. . . color='Red', joint_kws=
1.plot_joint(sns.kdeplot
.. .. rugplot.

(50)

Pairplot :-

By default, this function will create a grid of axes such that each numeric variable in data will by shared across the y-axis across a single row and the x-axis across a single column. the diagonal plots are treated differently; a univariate distribution plot is drawn to show the marginal distribution of the data in each column.

⇒ changing the diagonal plot kind to KDE, hist or None.

⇒ sns.pairplot(data=iris, diag_kind='kde')

⇒ changing the non diagonal plot kind to scatter, kde, hist or reg.

sns.pairplot(data=iris, kind="reg")

⇒ hue to pairplot. hue = categorical.

⇒ Creating pair plot for specific list of variables (diag_kind=None)

sns.pairplot(data=iris, x_vars=[' ',' ',' '], y_vars='sepal length', diag_kind=None)

⇒ showing only the lower triangle using CORNOR argument :-

```
sns.pairplot(data=iris, corner=True)
```

⇒ making changes specific to diagonal and non diagonal plots separately :-

```
diag_kws = dict(color='red', kde=True)
```

Nondiagonal ← `plot_kws = dict(color='red', marker=10, s=100)`

⇒ creating KDE plot on top of the pairplot using map_lower or upper.

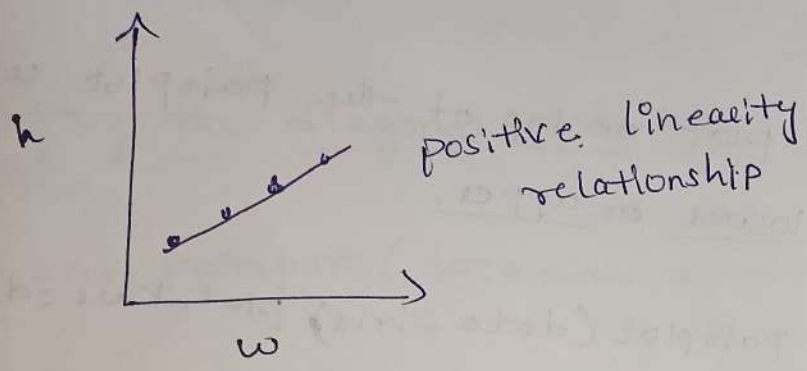
```
I= sns.pairplot(data=iris, plot_kws=dict(color
I.map_upper(sns.kdeplot)
I.map_lower(sns.kdeplot)
```

Relational

- Scatteplot
- Lineplot
- Relplot

⇒ Scatteplot :-

A diagram which shows the relationship between two variables by plotting the point / dots.



⇒ sns.scatteplot(data=xxx, x="", y="" ; have 3 values high, small, medium
style='outlet_size', →
markers={"high":"^","small":"v","medium":"o"}

⇒ grouping basis on a categorical variable using HUE & STYLE both together :-

style="outlet_size", hue='outlet_size'
markers={"high":"^","small":"v","medium":"o"}

⇒ grouping basics on numeric variables using HUE and use palette, size:

hue = " " , palette = " " , size = "coloum"

markers = {" ":"^", " ":"v",

⇒ sizes = (20, 200) → range of plots size.

⇒ legend = "full"

⇒ change the marker size with (s) argument :-

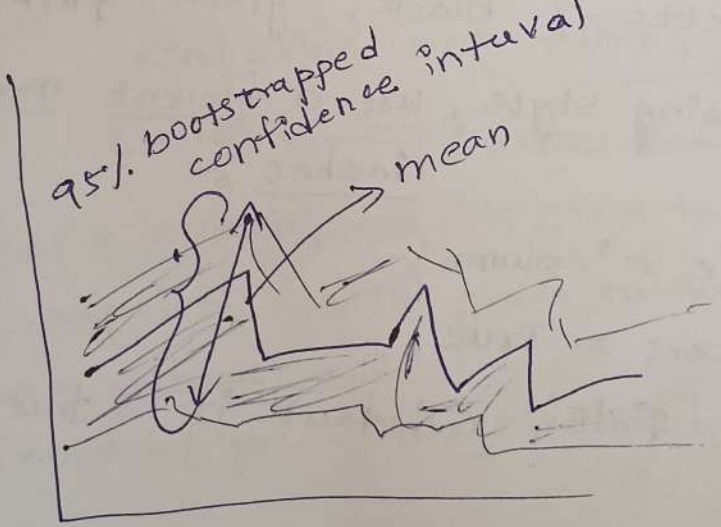
sns.scatterplot(data=xxx, x=" ", y=" ", s=500)

color = 'red', edge.color = 'black')

Line plot :-

→ shows the relation between two variables
→ majorly used in time series ananalysis

⇒ showcases how the value of variable changes over the time.



⇒ create a basic line plot and try different estimators :-

sns.lineplot (data = xxx, x = " ", y = " "
ci = None (for shadow of graph)
estimator = sum, None

⇒ Test with different confidence intervals and with different number of Bootstraping :-

sns.lineplot (data = xxx, x = " ", y = " ",
ci = 'sd', None, (in % 20, 30 - - -)

⇒ n_boot = 50 (change the no. of bootstraps

⇒ Grouping using hue and use different palettes

ci = None, hue = "column",
Palette = 'black'

⇒ In hue we have 3 values so,
palette = ['black', 'green', 'pink']

⇒ grouping using style, use different markes and dashes :-

Style = 'column',
markers = True,
dash = False (default it is true).

→ grouping using size :-

size = "column" (thickness of line in plot.

sizes = (.5, 5) → adjustment of size of lines in range.

→ use different semantic styling parameters on same or different variables :-

hue = "column1",
style = "column1",
markers = True,
size = "column1"

Rel plot (relational plot)

By default gives a scatterplot at same time it gives option to switch to lineplot or scatter plot

⇒ col_wap

---

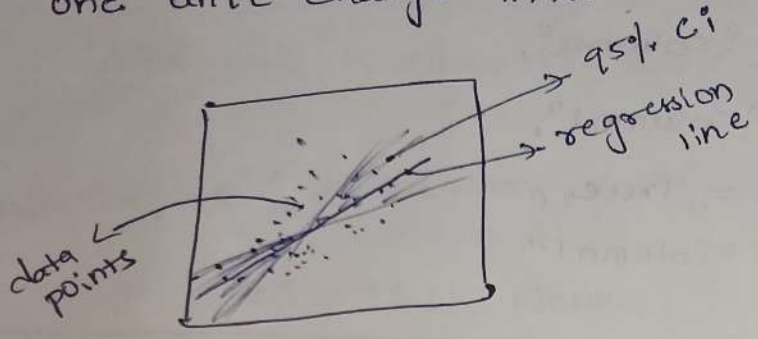
sns.relplot(data = xxx, x = " ", y = " ",
kind = 'line', ci = None, → confidence interval.
s = 100, ——— (increase the size of the plots)
col = 'column1', → it gives to list of graphs.
col_wrap = 5, → it gives to the no. of coloums.
row = 'column2'
palette = ['green', 'black', 'pink']

Regression

reg plot (regression plot)

plot data and a linear regression model fit

⇒ finds the best fit linear regression line which helps to predict the amount of change in y on one unit change in x.



⇒ sns.regplot(data=xxx, x="", y="")

→ change the styling of regplot, only line, only scatter, show/hide the ci, change the ci value, change n_boot :-

```
sns.regplot(data=xxx, x="", y="",
    color='red',
    marker='+',
    line_kws=dict(color='red', linestyle='--'),   (changing the reg line)
    scatter_kws=dict(s=100, color='green',
                     alpha=0.5)
ci=None (or) ci=60, 90, ---,
n_boot=500, ---
```

⇒ Discuss about the statistical models in seaborn.

⇒ sns.regplot (data=xxx, x="", y=""

order =,
logistic
lowess
robust

---

matrix

- Heatmaps.
- Cluster maps.

1) Heatmaps :-

A heatmaps is a graphical representation of data where values are depicted by color

⇒ by using pivot and correlation.

⇒ x = data.pivot_table (index ="", columns ="",
values ="")
print (x)

⇒ ~~sns.heatmap (x)~~

⇒ sns.set_style('white')
plt = figure (figsize = (5,5))
sns.heatmap (x, annot = True, → (to visible of numbers in sq box
fmt = '.0f',
annot_kws = dict (size = 15, weight = 'bold'))
linewidth = 0.5, linecolor = 'black',
(palatte) → cmap = 'OrRd_r')

⇒ using correlation matrix data to plot the heatmap

⇒ data.corr()

```
sns.heatmap(data.corr(),
    vmin = -1,
    vmax = 1
    center = 0,
    cmap = 'OrRd_r',
    annot = True,
    fmt = '.1f', annot_kws = dict(size = 15, weight = 'bold',
        linecolor = 'black',
        linewidth = 0.5))
```

change the values of the bar

Cluster map :-

Plot a matrix dataset as a hierarchically - clustered heatmap. (maps the columns with identically

⇒ data.columns = data.columns.str.lower()

Pivot_table :-

```
x = data.pivot_table(index = " ", columns = " ", values = " ")
sns.clustermap(x, col_cluster = False, (row clustering)
    row_cluster = False, (column cluster)
    annot = True,
    fmt = '.0f', z_score = 1, (0 or 1)
    standard_scale = 1, linewidth = 0.5,
```

(22)

multi-plot grids :-

⇒ FacetGrid plot or

Creating structure & supplying the chart/plot

⇒ It helps in visualizing distribution of one variable as well as the relationship between multiple variables separately within subsets of your dataset using multiple panels.

⇒ multiplot grid for plotting conditional relationships.

⇒ data.column.unique()

⇒ Supplying desired plot type in the facetgrid :-

```
x = sns.FacetGrid(mart, col = "colname")
x.map_dataframe(sns.histplot, x = " ",
                lineplot → 2 axis.
hue = "colname", marker = '+',
alpha = 0.5, color = 'red',
S = 100
```

⇒ Applying stylings specific to the facetGrid :-

set limits on Y axis.

all y axis 0-1000

```
x = sns.FacetGrid(data, col = '', row = '', sharey = True,
x.map_dataframe(sns.histplot, x = '', ylim = (0, 1000))
```

⇒ setting the axis lables :-

```
x.set_axis_labels('sales', 'count')
```

⇒ styling the titles (x or y titles)

```
x.set_titles(col_template = '{col_name}', size'
             row_template = '{row_name}', Type'))
```

⇒ count the values — countplot

→ Average amount → Barplot

⇒ Statistical like median, min, max — Boxplot.

→ density along with above statistics — Violin plot

→ display data points ⟶ Stripplot.

→ same display data points ⟶ Swarm plot.

`(sns.stripplot(data = x.sample(200)))` — jitter = 0.5 ...

→ flexibility to change the plots with catplot.

using col → divides the no. of colums.

⇒ transactions by on a interval (histplot)

bins = np.arange(0, 11000, 2000))

→ Distribution using KDE plot

→ Rugplot introduction. (combines with histplot).

→ check out the proportional of transactions (in %) using ECDF

→ distribution by combining histogram, KDE and Rugplot.

using distribution or Displot

↳ `sns.displot(data = xx, x = "", kde = True, rug = True)`

→ investigate how one column is related to another column using — Jointplot.

→ investigate how each of the numeric variable in the Dataframe is related to each other using Pairplot

→ investigat relationship b/w two column variables
scatterplot.

→ column trend by outlet → lineplot.

→ relplot : kind = 'line'

→ predict ... regression plot.

→ heatmap. → annot = True

→ clustering using clustermapplot.