seaburn

Seaborn Data Visualization

Unlocking Insights with Python

Python Seaborn

Seaborn is a library mostly used for statistical plotting in Python. It is built on top of Matplotlib and provides beautiful default styles and color palettes to make statistical plots more attractive.
In this tutorial, we will learn about Python Seaborn from basics to advance using a huge dataset of seaborn basics, concepts, and different graphs that can be plotted.

Getting Started
First of all, let us install Seaborn. Seaborn can be installed using the pip. Type the below command in the terminal.

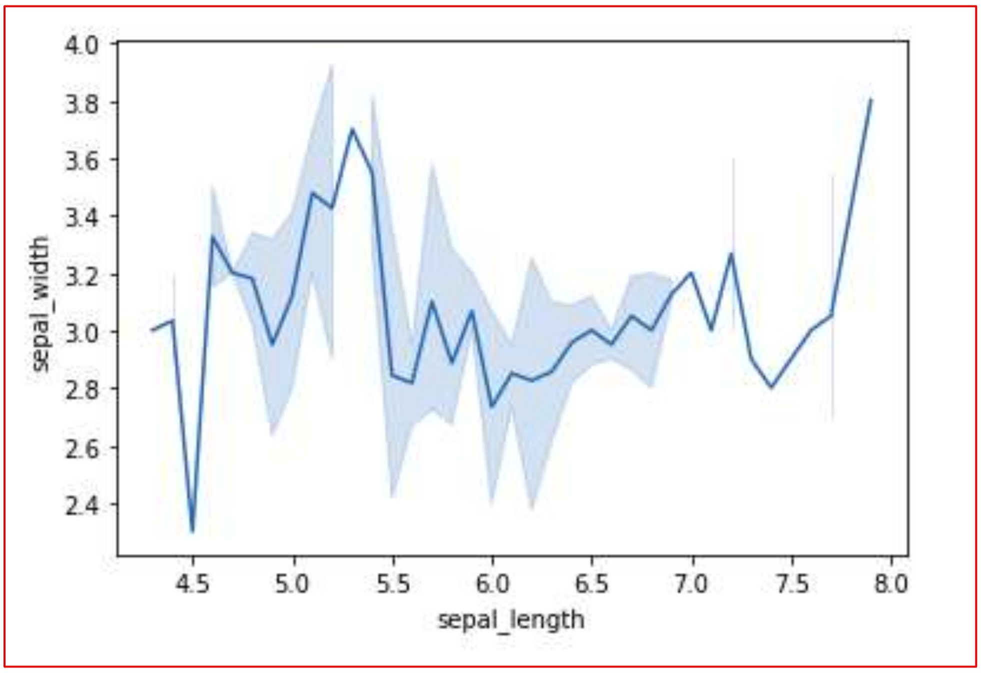
```
pip install seaborn
```

After the installation let us see an example of a simple plot using Seaborn. We will be plotting a simple line plot using the iris dataset. Iris dataset contains five columns such as Petal Length, Petal Width, Sepal Length, Sepal Width and Species Type. Iris is a flowering plant, the researchers have measured various features of the different iris flowers and recorded them digitally.

Example:

```
# importing packages
import seaborn as sns

# loading dataset
data = sns.load_dataset("iris")

# draw lineplot
sns.lineplot(x="sepal_length", y="sepal_width", data=data)
```

Output:
Using Seaborn with Matplotlib

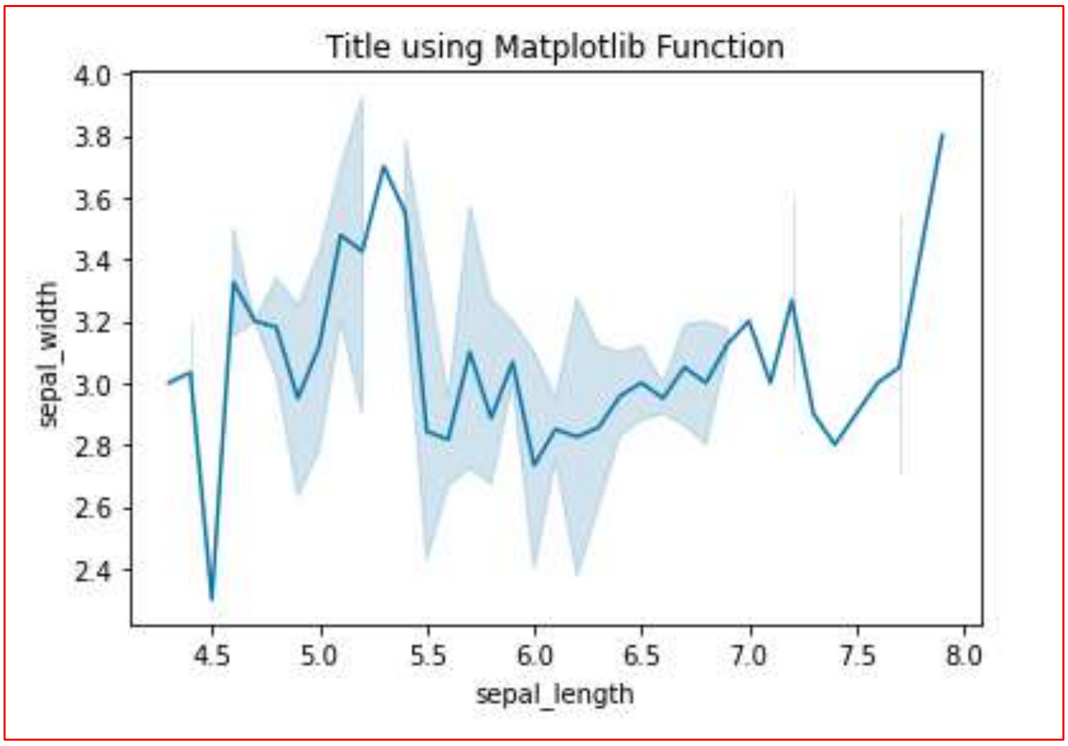


Using both Matplotlib and Seaborn together is a very simple process. We just have to invoke the Seaborn Plotting function as normal, and then we can use Matplotlib's customization function.

Example 1: We will be using the above example and will add the title to the plot using the Matplotlib.

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

# draw lineplot
sns.lineplot(x="sepal_length", y="sepal_width", data=data)

# setting the title using Matplotlib
plt.title('Title using Matplotlib Function')

plt.show()
```

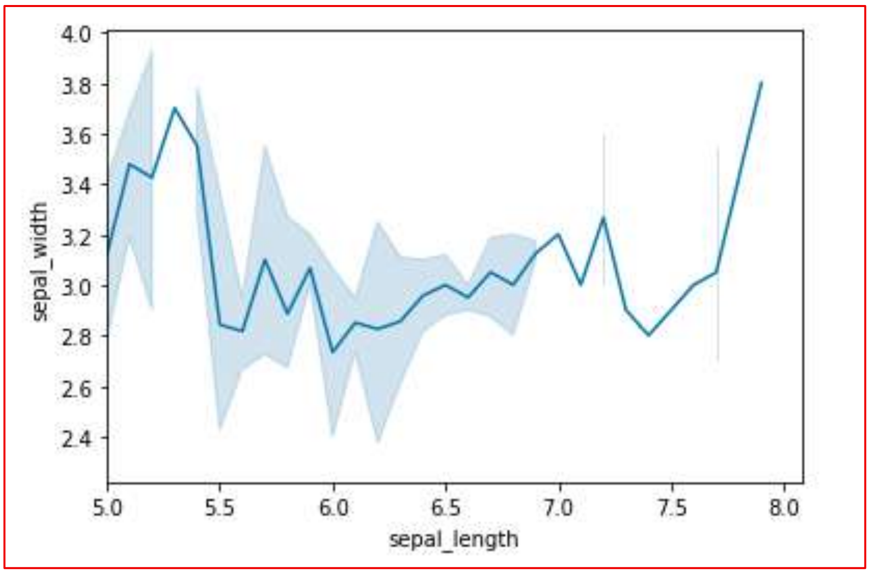
Output:



Example 2: Setting the xlim and ylim

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

# draw lineplot
sns.lineplot(x="sepal_length", y="sepal_width", data=data)

# setting the x limit of the plot
plt.xlim(5)
plt.show()
```

Output:



Customizing Seaborn Plots

Seaborn comes with some customized themes and a high-level interface for customizing the looks of the graphs. Consider the above example where the default of the Seaborn is used. It still looks nice and pretty but we can customize the graph according to our own needs. So let's see the styling of plots in detail.

Changing Figure Aesthetic

set_style() method is used to set the aesthetic of the plot. It means it affects things like the color of the axes, whether the grid is active or not, or other aesthetic elements. There are five themes available in Seaborn.

- darkgrid
- whitegrid
- dark
- white
- ticks

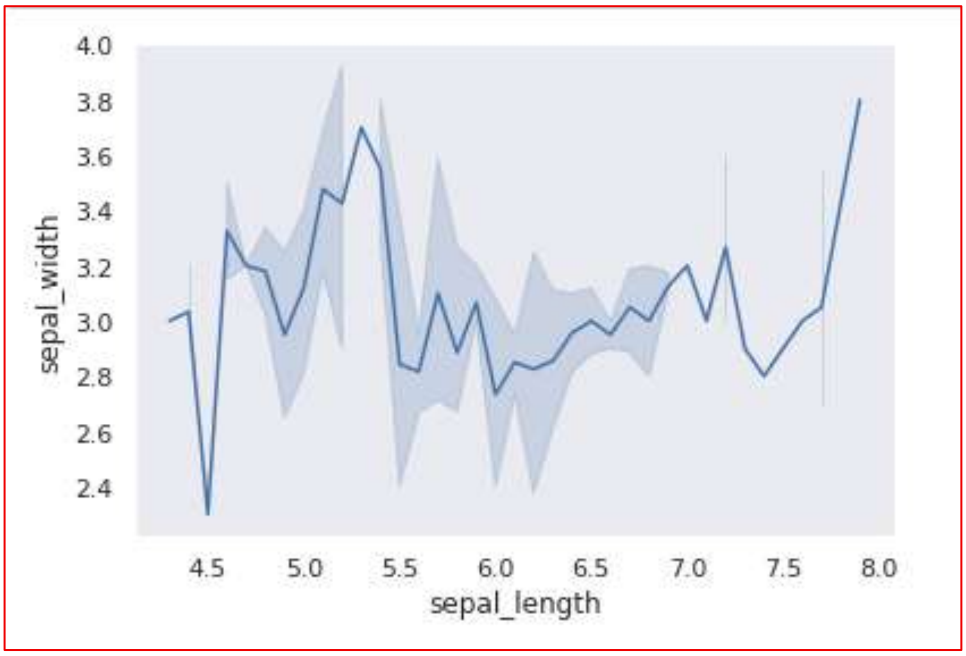
Syntax:

set_style(style=None, rc=None)

Example: Using the dark theme

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

# draw lineplot
sns.lineplot(x="sepal_length", y="sepal_width", data=data)

# changing the theme to dark
sns.set_style("dark")
plt.show()
```

Output:

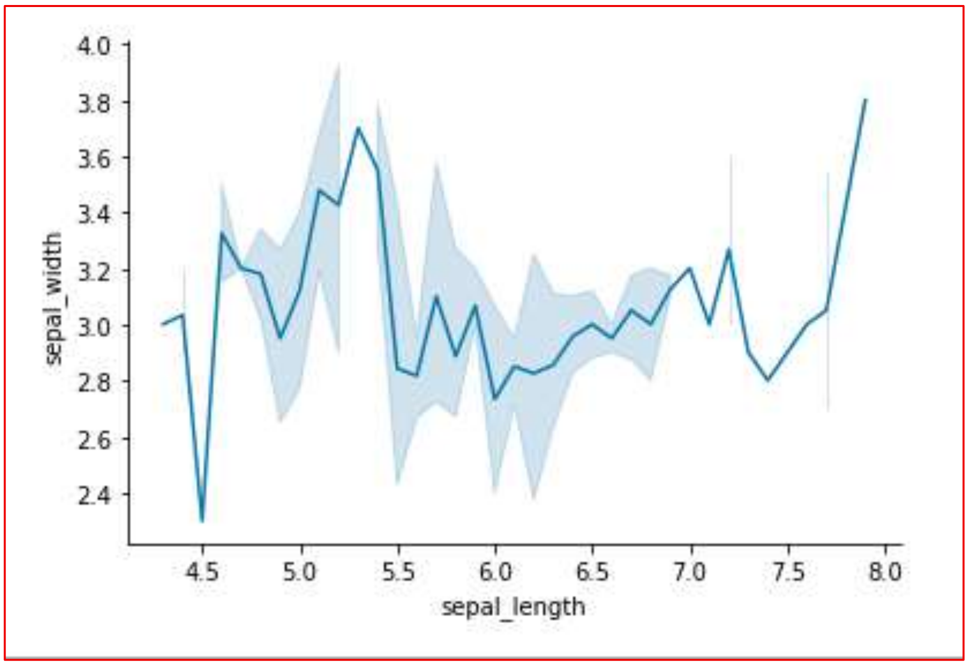


Removal of Spines
Spines are the lines noting the data boundaries and connecting the axis tick marks. It can be removed using the despine() method.

Syntax:
sns.despine(left = True)

Example:
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

# draw lineplot
sns.lineplot(x="sepal_length", y="sepal_width", data=data)

# Removing the spines
sns.despine()
plt.show()
```

Output:

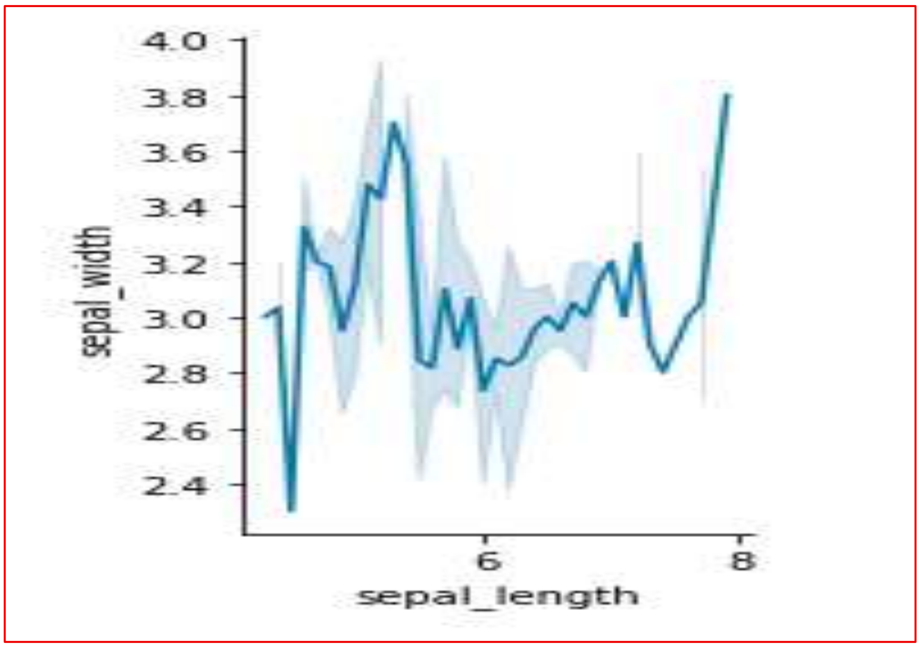


Changing the figure Size
The figure size can be changed using the **figure()** method of Matplotlib. figure() method creates a new figure of the specified size passed in the **figsize** parameter.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

# changing the figure size
plt.figure(figsize = (2, 4))

# draw lineplot
sns.lineplot(x="sepal_length", y="sepal_width", data=data)

# Removing the spines
sns.despine()

plt.show()
```

Output:



Scaling the plots

It can be done using the set_context() method. It allows us to override default parameters. This affects things like the size of the labels, lines, and other elements of the plot, but not the overall style. The base context is "notebook", and the other contexts are "paper", "talk", and "poster". font_scale sets the font size.

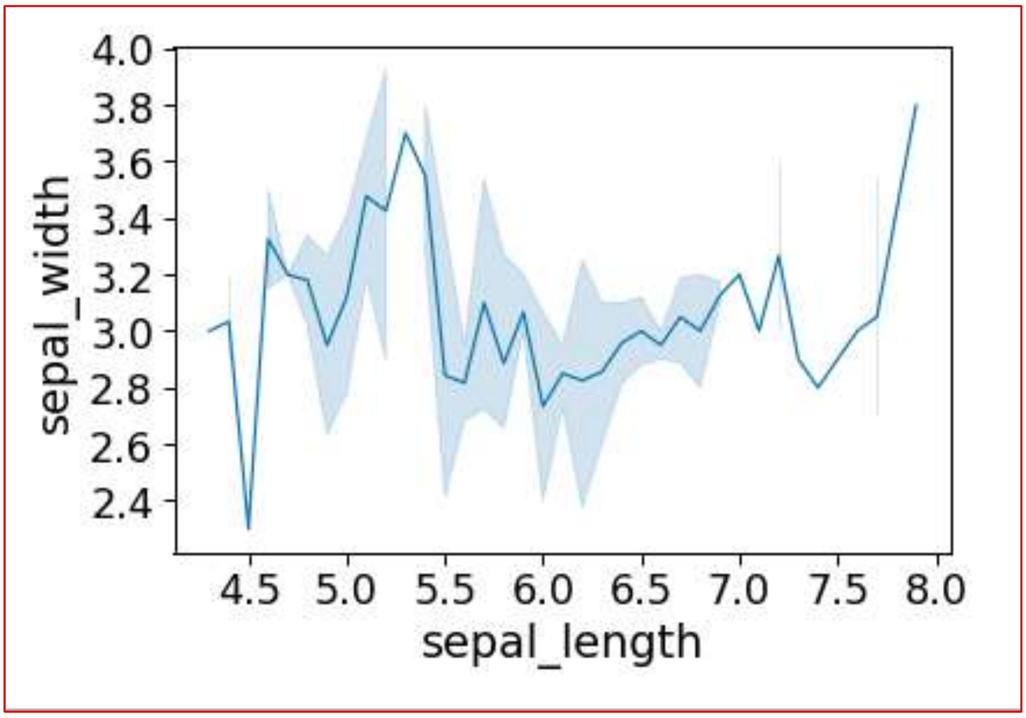
Syntax:

set_context(context=None, font_scale=1, rc=None)

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

# draw lineplot
sns.lineplot(x="sepal_length", y="sepal_width", data=data)

# Setting the scale of the plot
sns.set_context("paper")

plt.show()
```

Output:

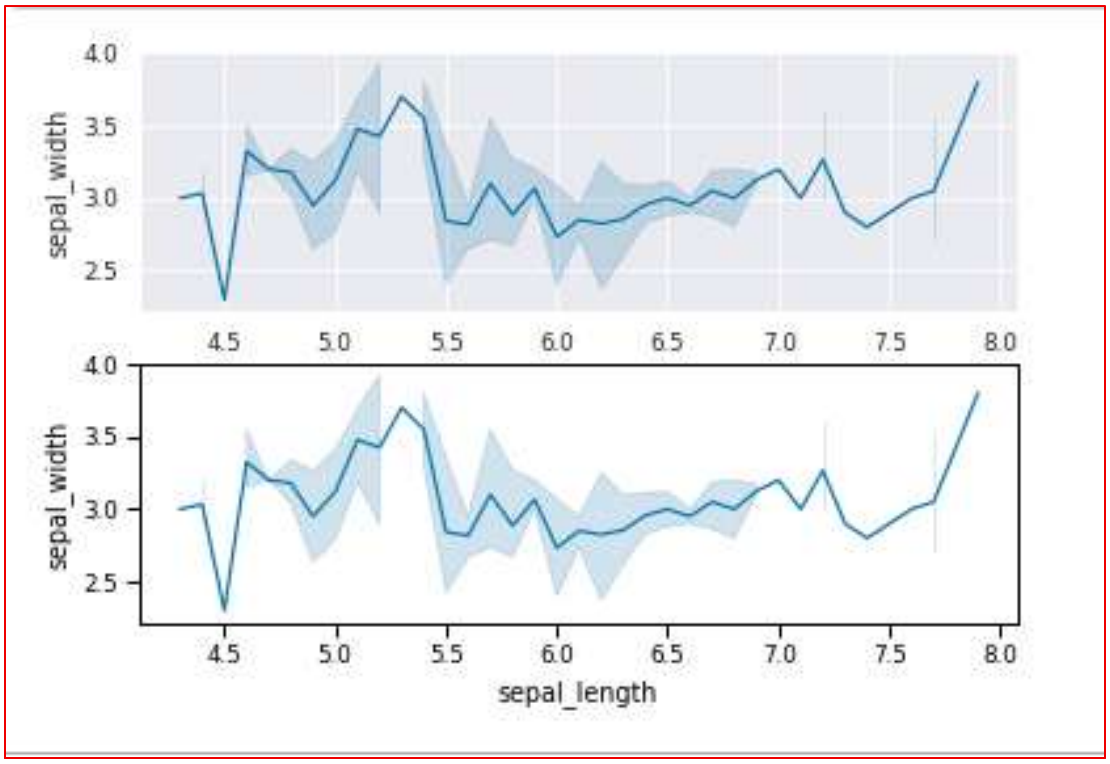


Setting the Style Temporarily
axes_style() method is used to set the style temporarily. It is used along with the **with** statement.

Syntax:
axes_style(style=None, rc=None)

Example:
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")


def plot():
    sns.lineplot(x="sepal_length", y="sepal_width", data=data)

with sns.axes_style('darkgrid'):

    # Adding the subplot
    plt.subplot(211)
    plot()

plt.subplot(212)
plot()
```

Output:



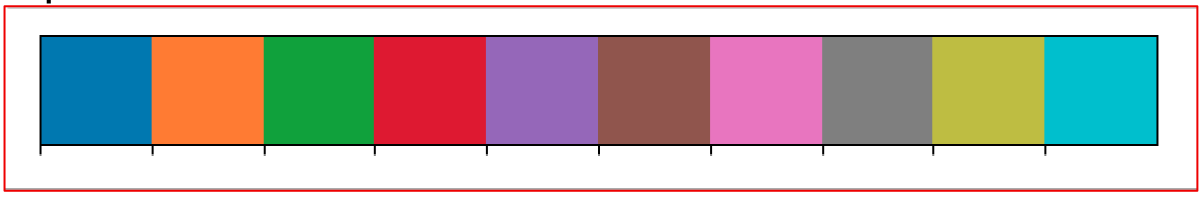
Color Palette

Colormaps are used to visualize plots effectively and easily. One might use different sorts of colormaps for different kinds of plots. **color_palette()** method is used to give colors to the plot. Another function **palplot()** is used to deal with the color palettes and plots the color palette as a horizontal array.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# current colot palette
palette = sns.color_palette()

# plots the color palette as a
# horizontal array
sns.palplot(palette)

plt.show()
```

Output:



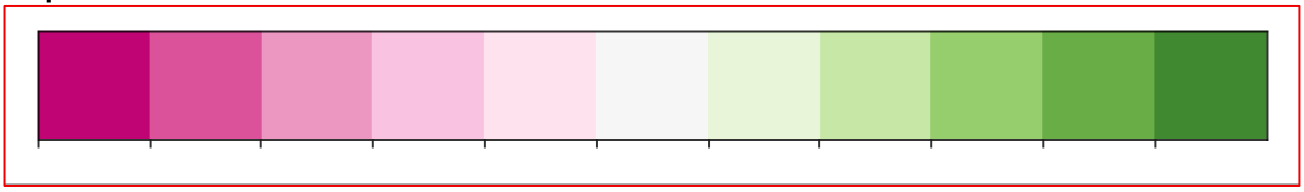
Diverging Color Palette

This type of color palette uses two different colors where each color depicts different points ranging from a common point in either direction. Consider a range of -10 to 10 so the value from -10 to 0 takes one color and values from 0 to 10 take another.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# current colot palette
palette = sns.color_palette('PiYG', 11)

# diverging color palette
sns.palplot(palette)

plt.show()
```

Output:



In the above example, we have used an in-built diverging color palette which shows 11 different points of color. The color on the left shows pink color and color on the right shows green color.

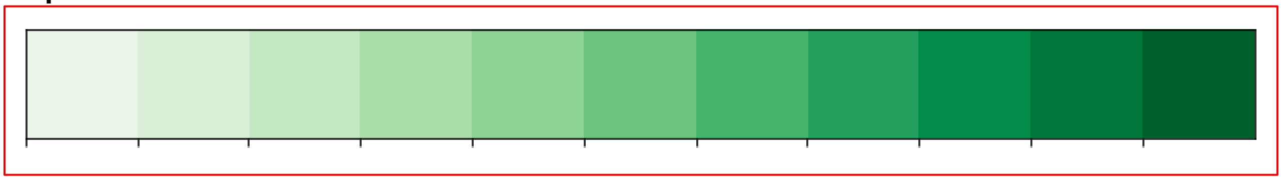
Sequential Color Palette

A sequential palette is used where the distribution ranges from a lower value to a higher value. To do this add the character 's' to the color passed in the color palette.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# current colot palette
palette = sns.color_palette('Greens', 11)

# sequential color palette
sns.palplot(palette)

plt.show()
```

Output:

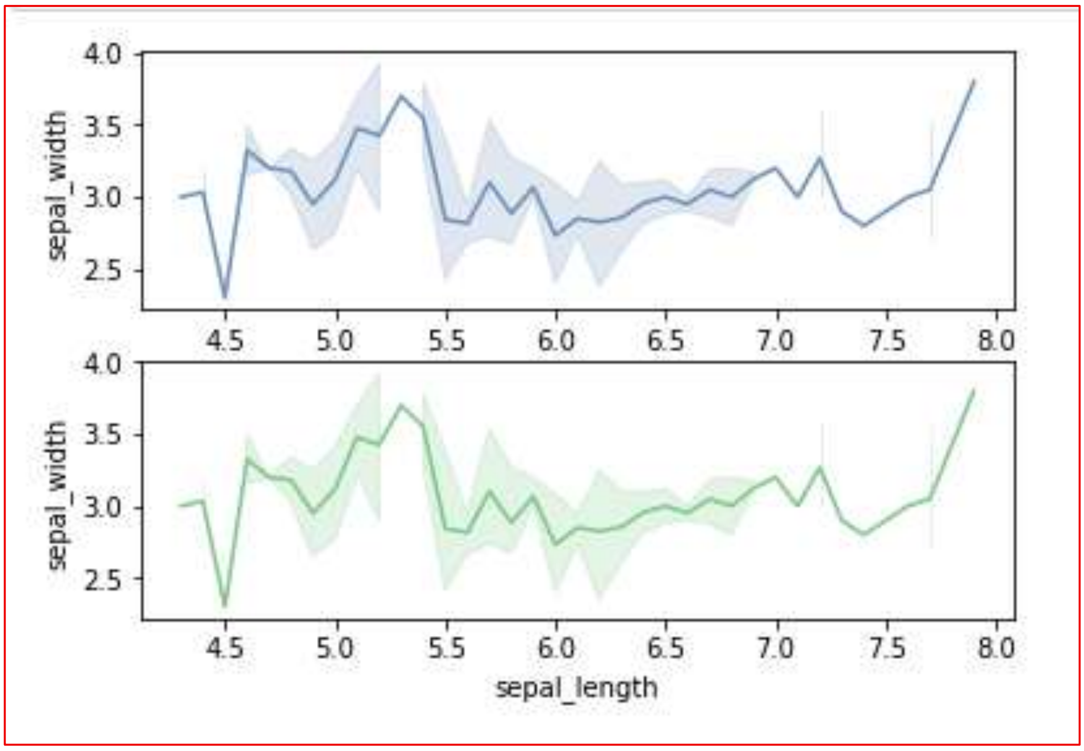


Setting the default Color Palette

set_palette() method is used to set the default color palette for all the plots. The arguments for both color_palette() and set_palette() is same. set_palette() changes the default matplotlib parameters.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

def plot():
    sns.lineplot(x="sepal_length", y="sepal_width", data=data)

# setting the default color palette
sns.set_palette('vlag')
plt.subplot(211)

# plotting with the color palette
# as vlag
plot()

# setting another default color palette
sns.set_palette('Accent')
plt.subplot(212)
plot()

plt.show()
```
Output:



Multiple plots with Seaborn

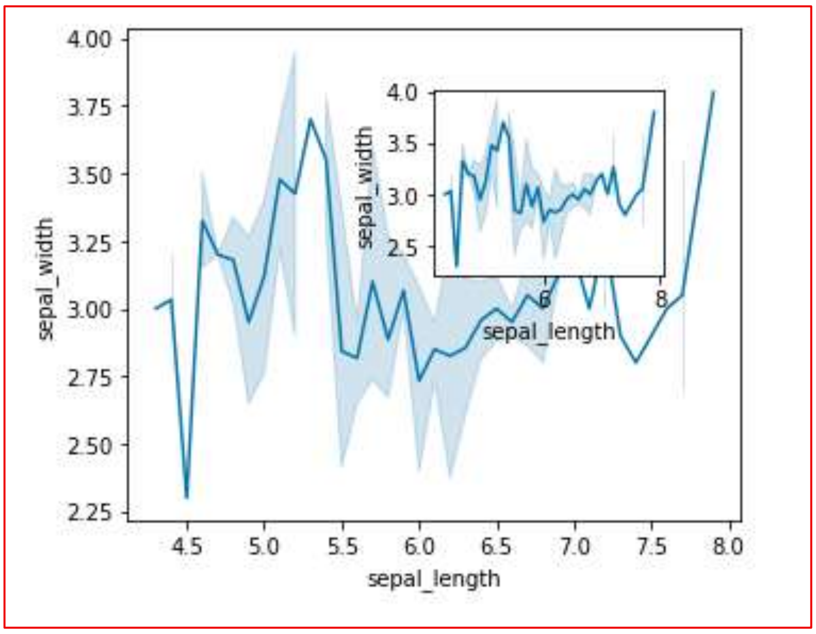
Using Matplotlib

Matplotlib provides various functions for plotting subplots. Some of them are add_axes(), subplot(), and subplot2grid(). Let's see an example of each function for better understanding.

Example 1: Using add_axes() method
```
# importing packages
```

```
import seaborn as sns
import matplotlib.pyplot as plt


# loading dataset
data = sns.load_dataset("iris")

def graph():
    sns.lineplot(x="sepal_length", y="sepal_width", data=data)

# Creating a new figure with width = 5 inches
# and height = 4 inches
fig = plt.figure(figsize =(5, 4))

# Creating first axes for the figure
ax1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])

# plotting the graph
graph()

# Creating second axes for the figure
ax2 = fig.add_axes([0.5, 0.5, 0.3, 0.3])

# plotting the graph
graph()

plt.show()
```

Output:

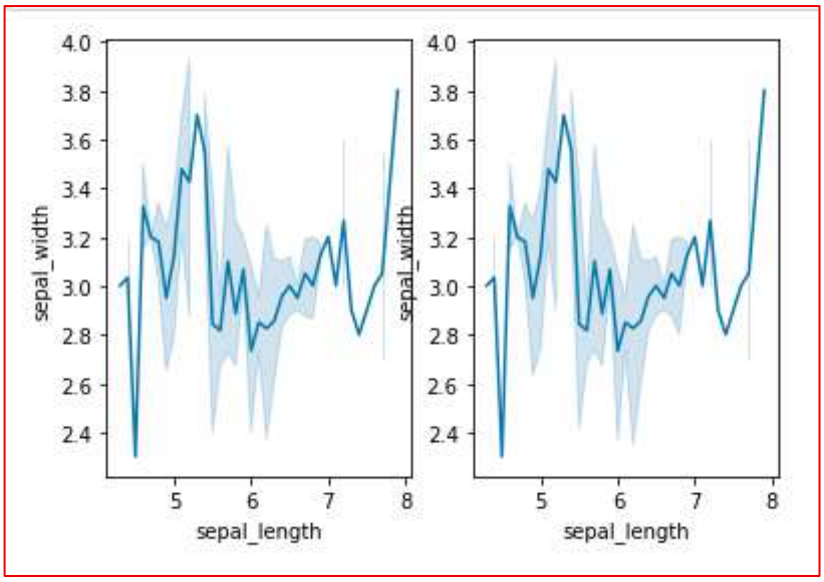


Example 2: Using subplot() method

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# loading dataset
data = sns.load_dataset("iris")

def graph():
    sns.lineplot(x="sepal_length", y="sepal_width", data=data)

# Adding the subplot at the specified
# grid position
plt.subplot(121)
graph()

# Adding the subplot at the specified
# grid position
plt.subplot(122)
graph()

plt.show()
```

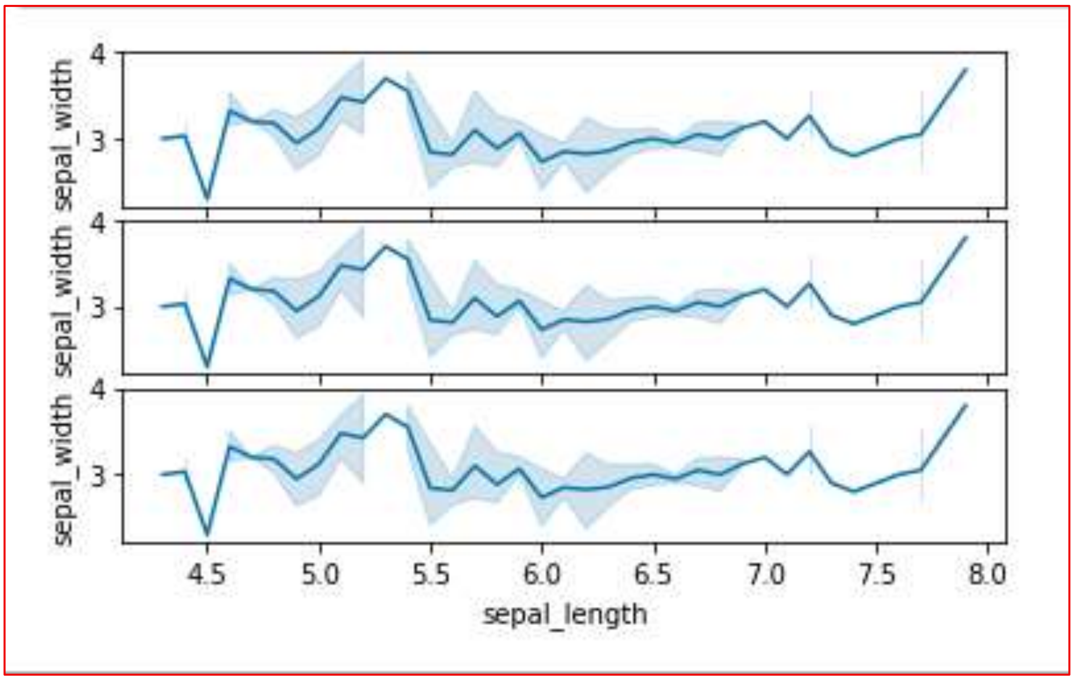
Output:



Example 3: Using subplot2grid() method

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

def graph():
    sns.lineplot(x="sepal_length", y="sepal_width", data=data)

# adding the subplots
axes1 = plt.subplot2grid (
  (7, 1), (0, 0), rowspan = 2,  colspan = 1)
graph()

axes2 = plt.subplot2grid (
  (7, 1), (2, 0), rowspan = 2, colspan = 1)
graph()

axes3 = plt.subplot2grid (
  (7, 1), (4, 0), rowspan = 2, colspan = 1)
graph()
```

Output:



Using Seaborn

Seaborn also provides some functions for plotting multiple plots.

Method 1: Using FacetGrid() method

- FacetGrid class helps in visualizing distribution of one variable as well as the relationship between multiple variables separately within subsets of your dataset using multiple panels.
- A FacetGrid can be drawn with up to three dimensions ? row, col, and hue. The first two have obvious correspondence with the resulting array of axes; think of the hue variable as a third dimension along a depth axis, where different levels are plotted with different colors.
- FacetGrid object takes a dataframe as input and the names of the variables that will form the row, column, or hue dimensions of the grid. The variables should be categorical and the data at each level of the variable will be used for a facet along that axis.

Syntax:

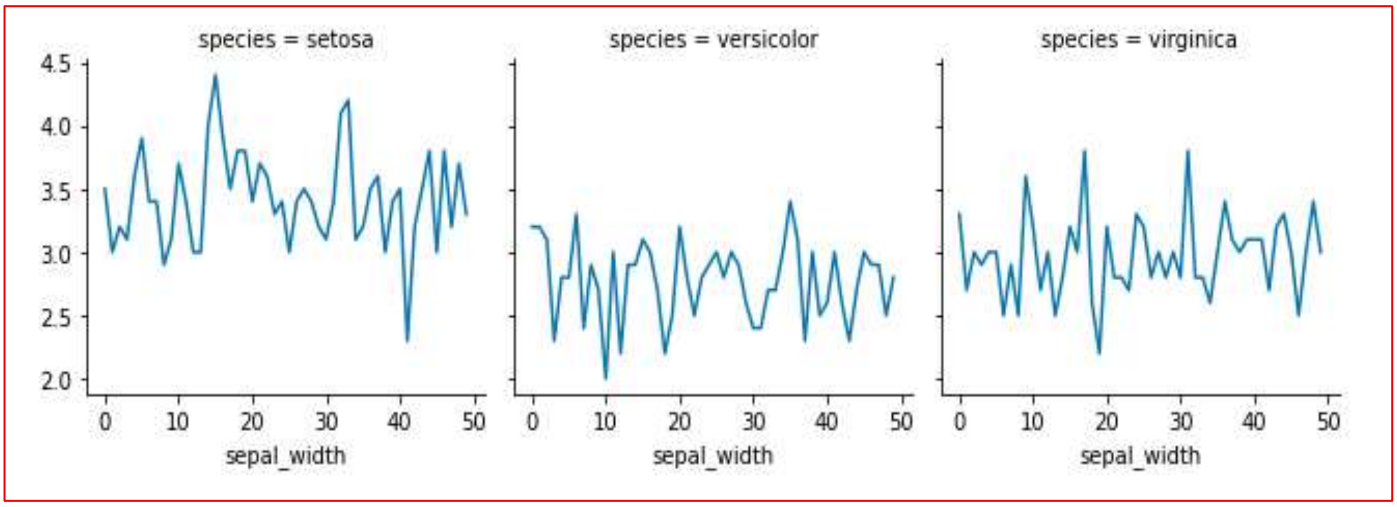
*seaborn.FacetGrid(data, **kwargs)*

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

plot = sns.FacetGrid(data, col="species")
plot.map(plt.plot, "sepal_width")
```

```
plt.show()
```

Output:



Method 2: Using PairGrid() method

- Subplot grid for plotting pairwise relationships in a dataset.
- This class maps each variable in a dataset onto a column and row in a grid of multiple axes. Different axes-level plotting functions can be used to draw bivariate plots in the upper and lower triangles, and the marginal distribution of each variable can be shown on the diagonal.
- It can also represent an additional level of conventionalization with the hue parameter, which plots different subsets of data in different colors. This uses color to resolve elements on a third dimension, but only draws subsets on top of each other and will not tailor the hue parameter for the specific visualization the way that axes-level functions that accept hue will.

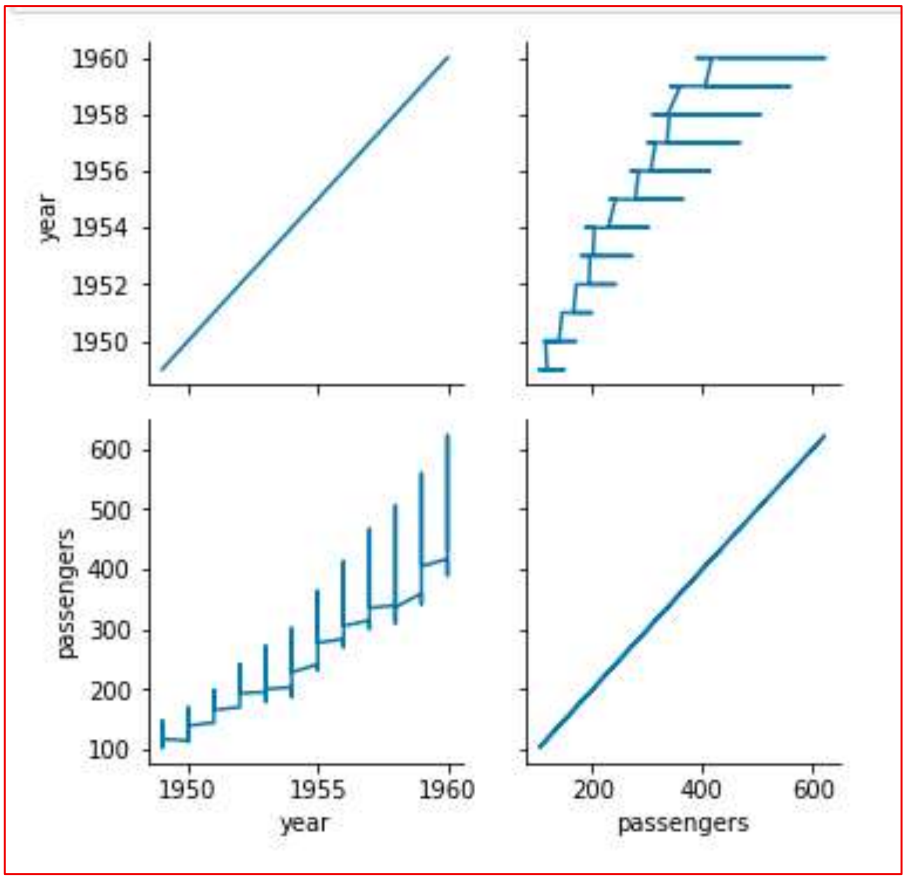
Syntax:

*seaborn.PairGrid(data, **kwargs)*

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("flights")

plot = sns.PairGrid(data)
plot.map(plt.plot)

plt.show()
```

Output:



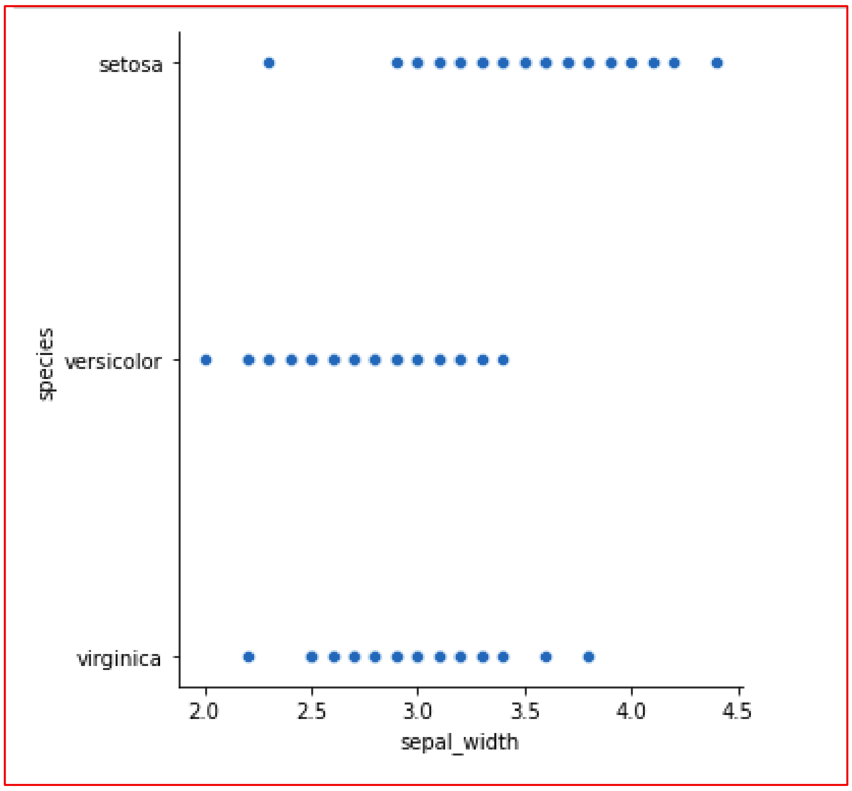
Creating Different Types of Plots
Relational Plots
Relational plots are used for visualizing the statistical relationship between the data points. Visualization is necessary because it allows the human to see trends and patterns in the data. The process of understanding how the variables in the dataset relate each other and their relationships are termed as Statistical analysis.

Relplot()
This function provides us the access to some other different axes-level functions which shows the relationships between two variables with semantic mappings of subsets. It is plotted using the **relplot()** method.

Syntax:
*seaborn.relplot(x=None, y=None, data=None, **kwargs)*

Example:
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

# creating the relplot
sns.relplot(x='sepal_width', y='species', data=data)

plt.show()
```

Output:



Scatter Plot

The **scatter plot** is a mainstay of statistical visualization. It depicts the joint distribution of two variables using a cloud of points, where each point represents an observation in the dataset. This depiction allows the eye to infer a substantial amount of information about whether there is any meaningful relationship between them. It is plotted using the **scatterplot()** method.

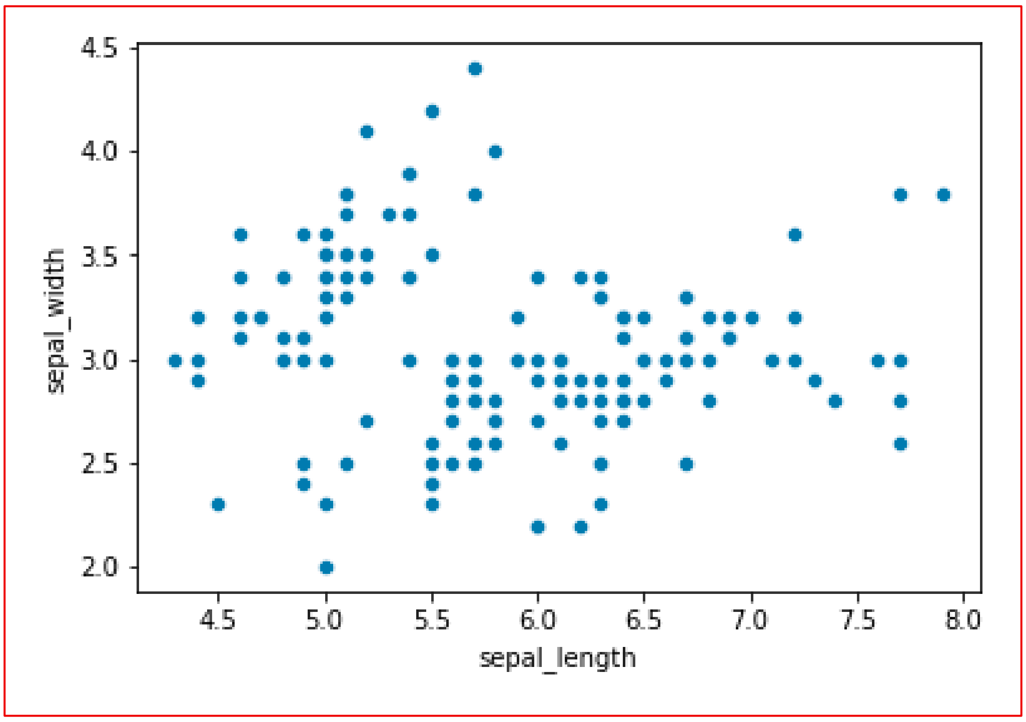
Syntax:

*seaborn.scatterplot(x=None, y=None, data=None, **kwargs)*

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.scatterplot(x='sepal_length', y='sepal_width', data=data)
plt.show()
```

Output:



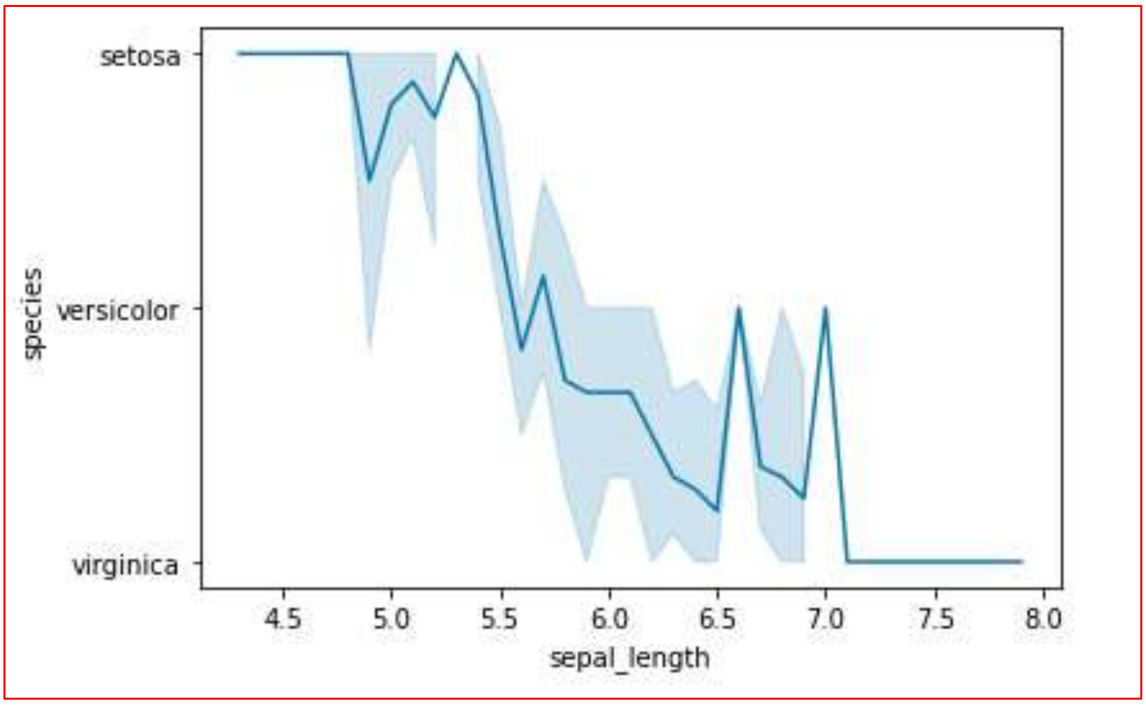
Line Plot
For certain datasets, you may want to consider changes as a function of time in one variable, or as a similarly continuous variable. In this case, drawing a line-plot is a better option. It is plotted using the **lineplot()** method.

Syntax:
*seaborn.lineplot(x=None, y=None, data=None, **kwargs)*

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.lineplot(x='sepal_length', y='species', data=data)
plt.show()
```

Output:



Categorical Plots

<u>**Categorical Plots**</u> are used where we have to visualize relationship between two numerical values. A more specialized approach can be used if one of the main variable is **categorical**which means such variables that take on a fixed and limited number of possible values.

Bar Plot

A **barplot** is basically used to aggregate the categorical data according to some methods and by default its the mean. It can also be understood as a visualization of the group by action. To use this plot we choose a categorical column for the x axis and a numerical column for the y axis and we see that it creates a plot taking a mean per categorical column. It can be created using the **barplot()** method.

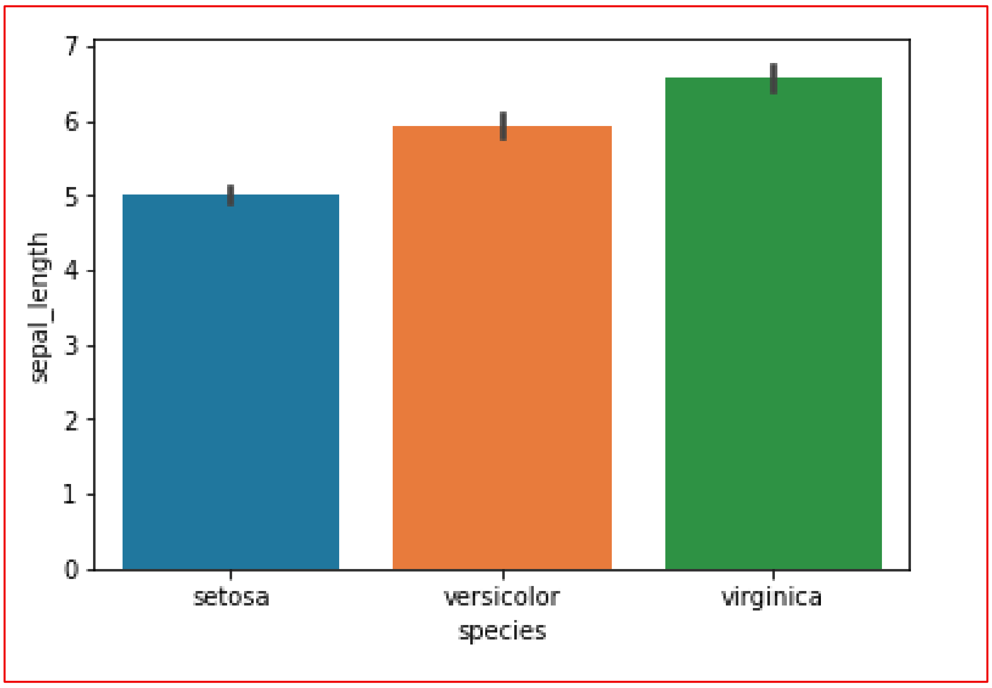
Syntax:

barplot([x, y, hue, data, order, hue_order, …])

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.barplot(x='species', y='sepal_length', data=data)
plt.show()
```

Output:

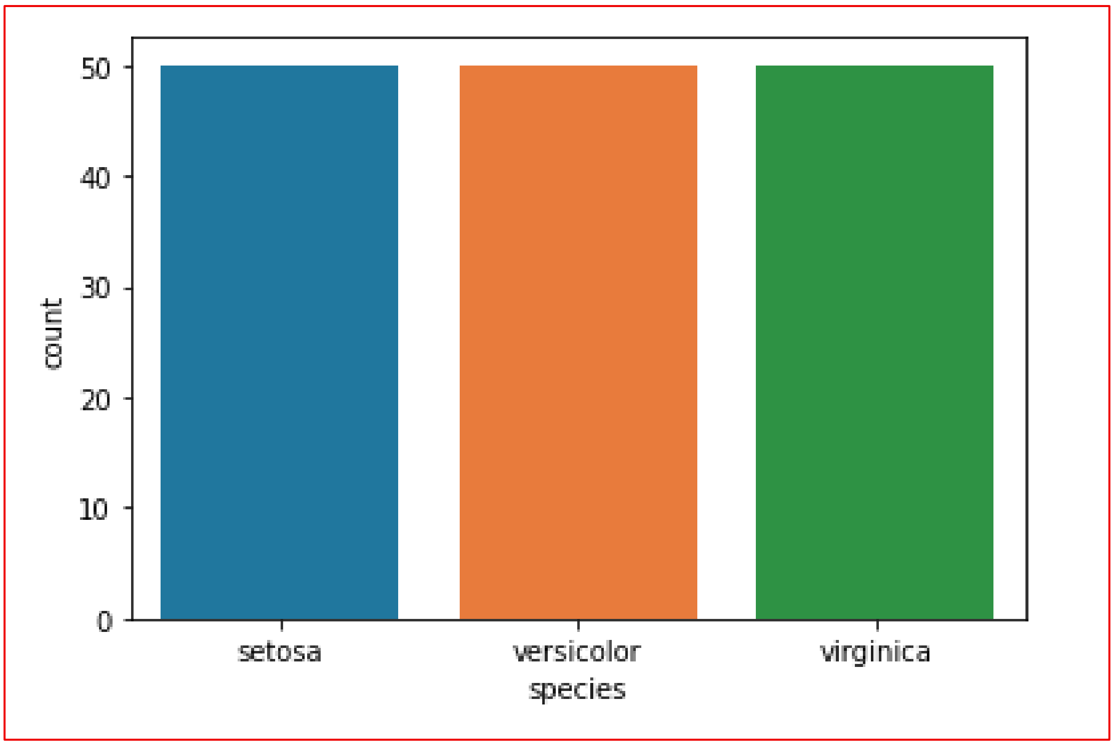


Count Plot
A **countplot** basically counts the categories and returns a count of their occurrences. It is one of the most simple plots provided by the seaborn library. It can be created using the **countplot()** method.

Syntax:
countplot([x, y, hue, data, order, ...])

Example:
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.countplot(x='species', data=data)
plt.show()
```

Output:

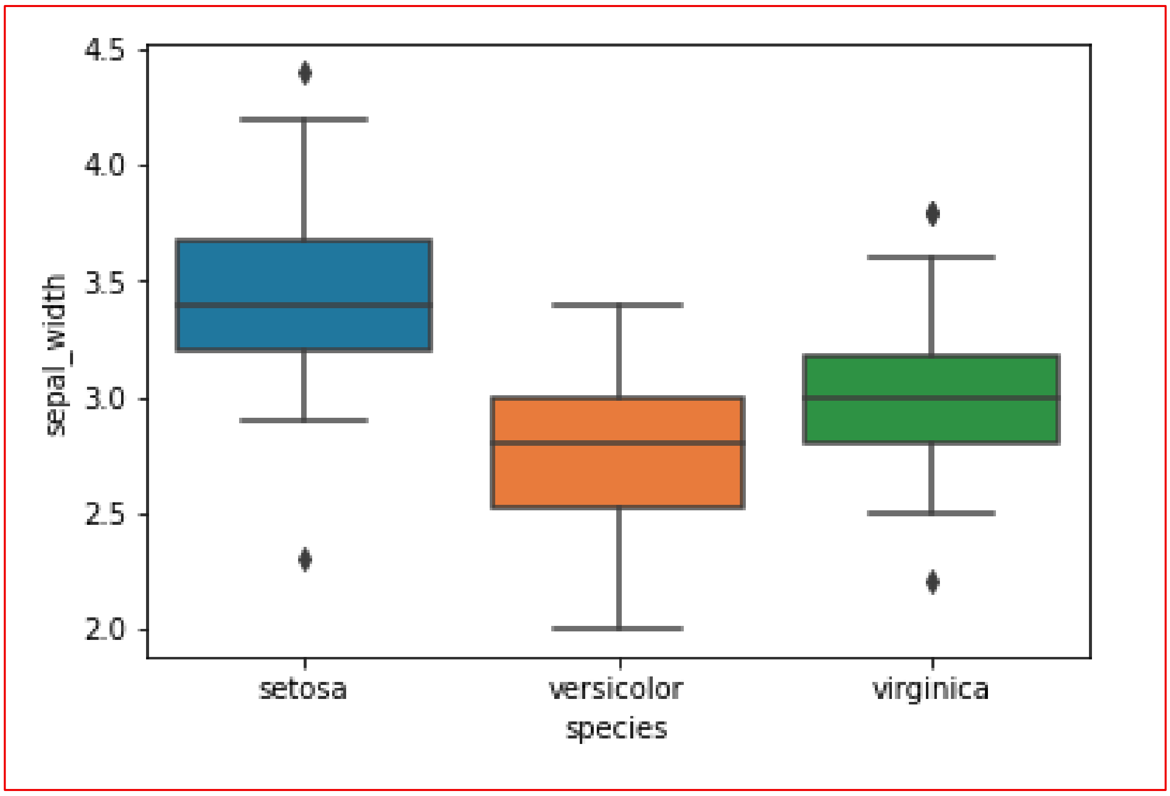


Box Plot
A **boxplot** is sometimes known as the box and whisker plot.It shows the distribution of the quantitative data that represents the comparisons between variables. boxplot shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution i.e. the dots indicating the presence of outliers. It is created using the **boxplot()** method.

Syntax:
boxplot([x, y, hue, data, order, hue_order, …])

Example:
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.boxplot(x='species', y='sepal_width', data=data)
plt.show()
```

Output:



Violinplot

It is similar to the boxplot except that it provides a higher, more advanced visualization and uses the kernel density estimation to give a better description about the data distribution. It is created using the **violinplot()** method.

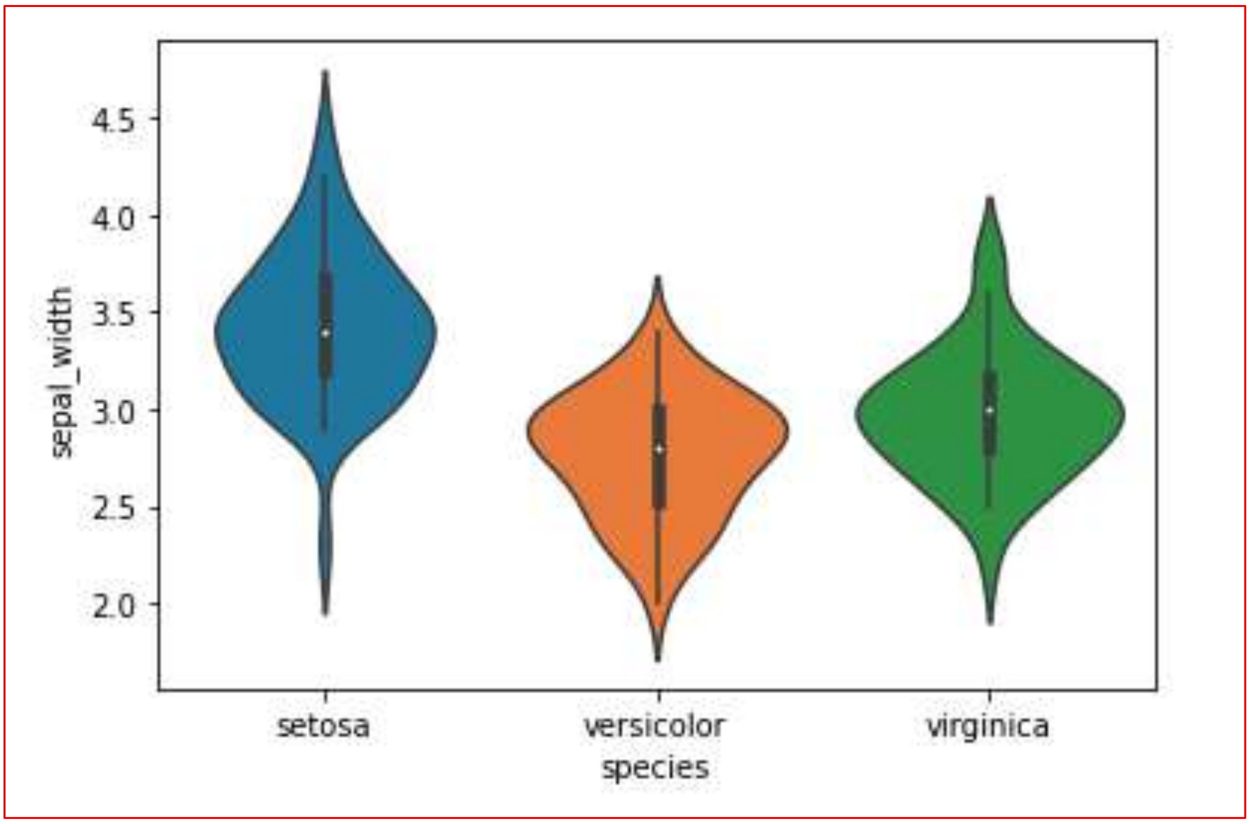
Syntax:

violinplot([x, y, hue, data, order, …]

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.violinplot(x='species', y='sepal_width', data=data)
plt.show()
```

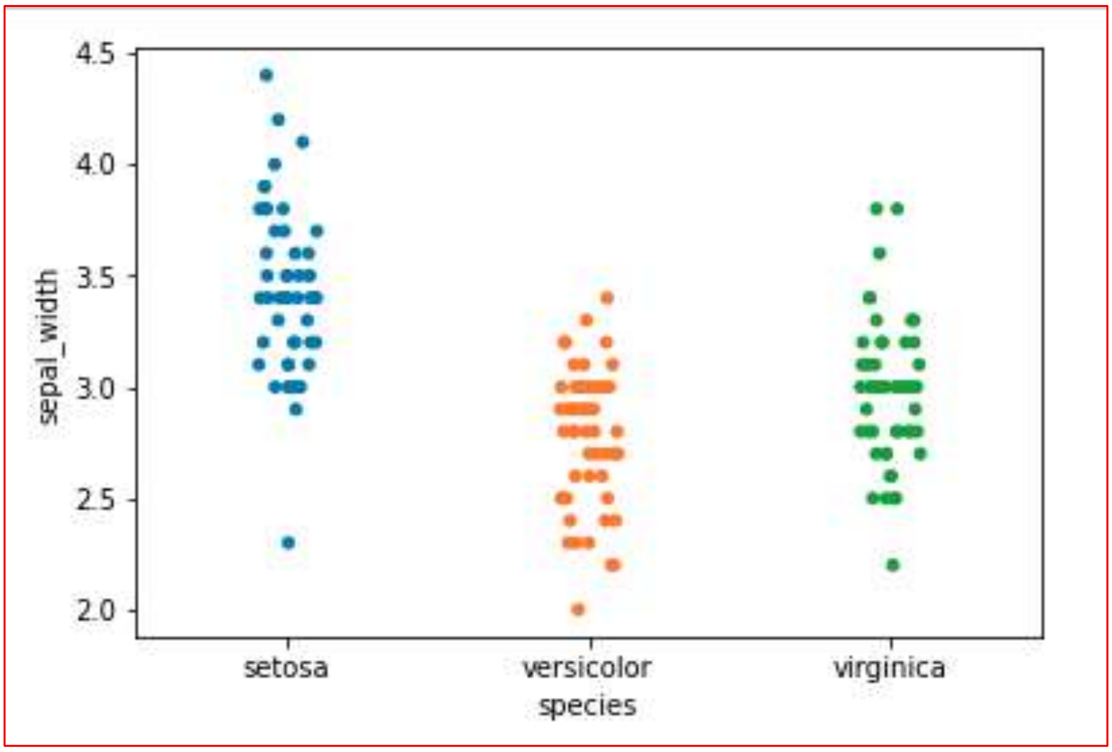
Output:



Stripplot
It basically creates a scatter plot based on the category. It is created using the **stripplot()** method.

Syntax:
stripplot([x, y, hue, data, order, …])

Example:
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.stripplot(x='species', y='sepal_width', data=data)
plt.show()
```
Output:



Swarmplot

Swarmplot is very similar to the stripplot except the fact that the points are adjusted so that they do not overlap.Some people also like combining the idea of a violin plot and a stripplot to form this plot. One drawback to using swarmplot is that sometimes they dont scale well to really large numbers and takes a lot of computation to arrange them. So in case we want to visualize a swarmplot properly we can plot it on top of a violinplot. It is plotted using the **swarmplot()** method.

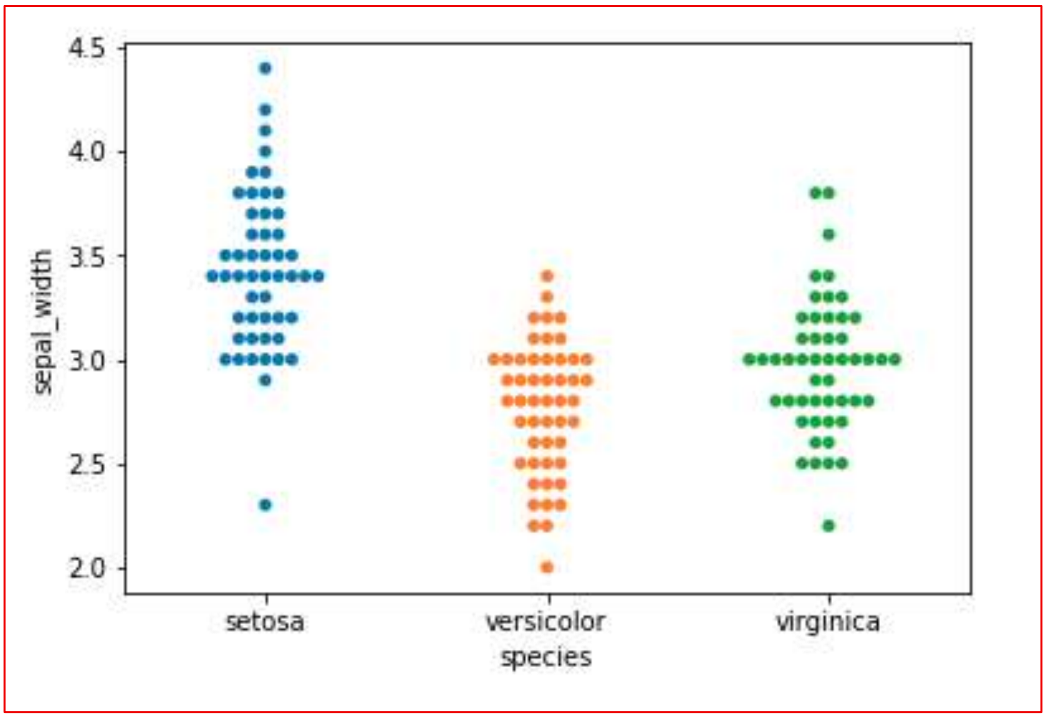
Syntax:

swarmplot([x, y, hue, data, order, …])

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.swarmplot(x='species', y='sepal_width', data=data)
plt.show()
```

Output:



Factorplot

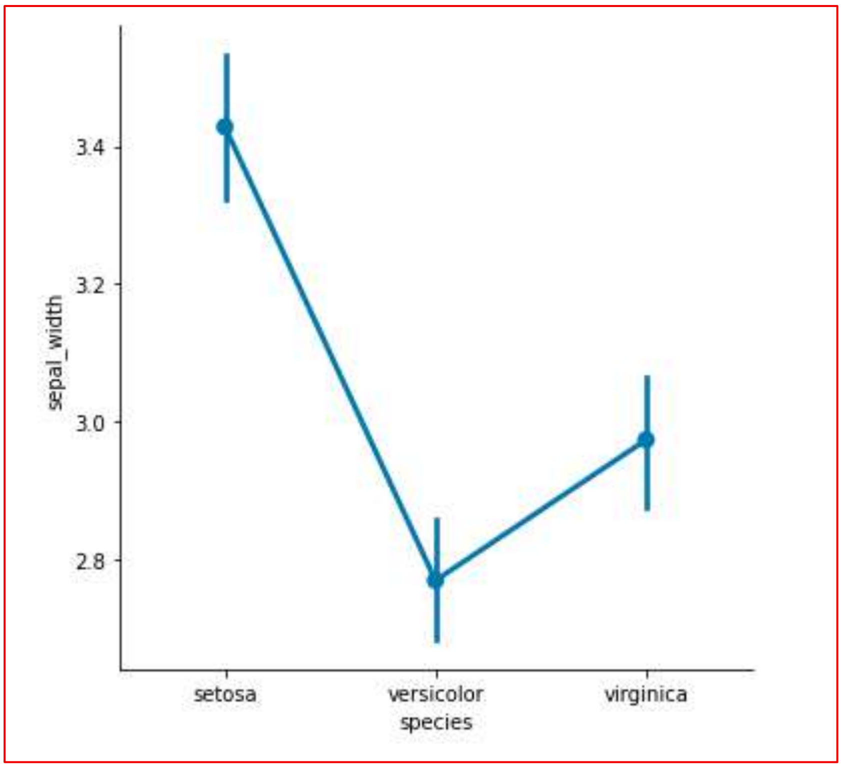
Factorplot is the most general of all these plots and provides a parameter called kind to choose the kind of plot we want thus saving us from the trouble of writing these plots separately. The kind parameter can be bar, violin, swarm etc. It is plotted using the **factorplot()** method.

Syntax:
sns.factorplot([x, y, hue, data, row, col, ...])

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.factorplot(x='species', y='sepal_width', data=data)
plt.show()
```

Output:

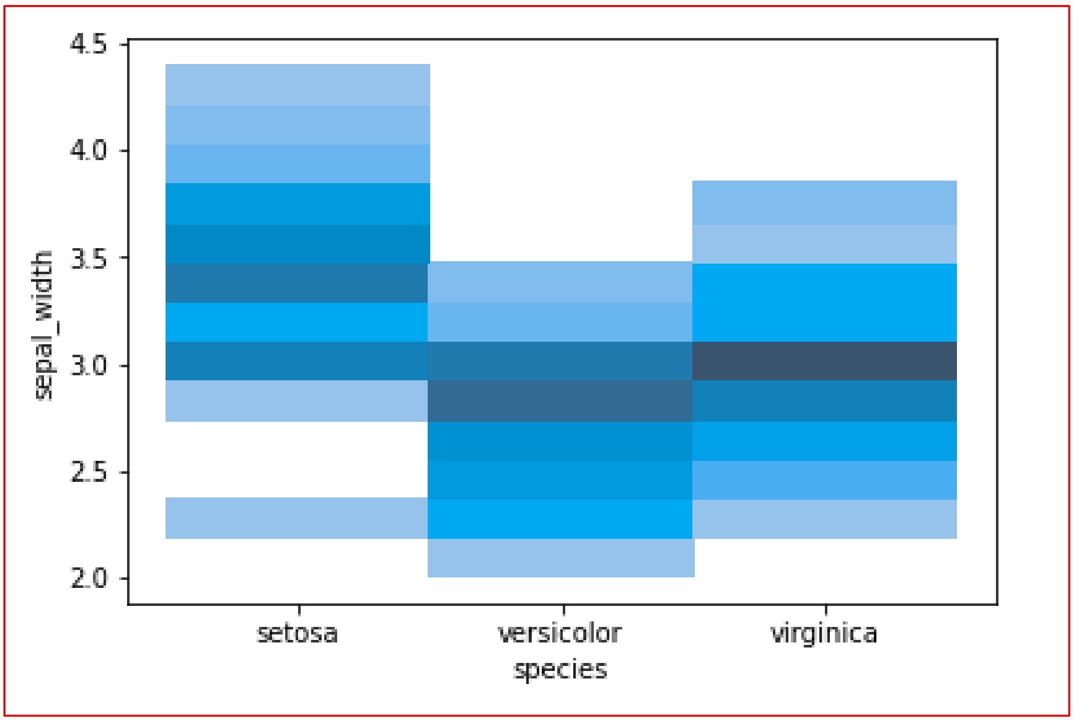


Distribution Plots
Distribution Plots are used for examining univariate and bivariate distributions meaning such distributions that involve one variable or two discrete variables.
Histogram
A histogram is basically used to represent data provided in a form of some groups.It is accurate method for the graphical representation of numerical data distribution. It can be plotted using the **histplot()** function.

Syntax:
*histplot(data=None, *, x=None, y=None, hue=None, **kwargs)*

Example:
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.histplot(x='species', y='sepal_width', data=data)
plt.show()
```

Output:



Distplot

Distplot is used basically for univariant set of observations and visualizes it through a histogram i.e. only one observation and hence we choose one particular column of the dataset. It is potted using the **distplot()** method.

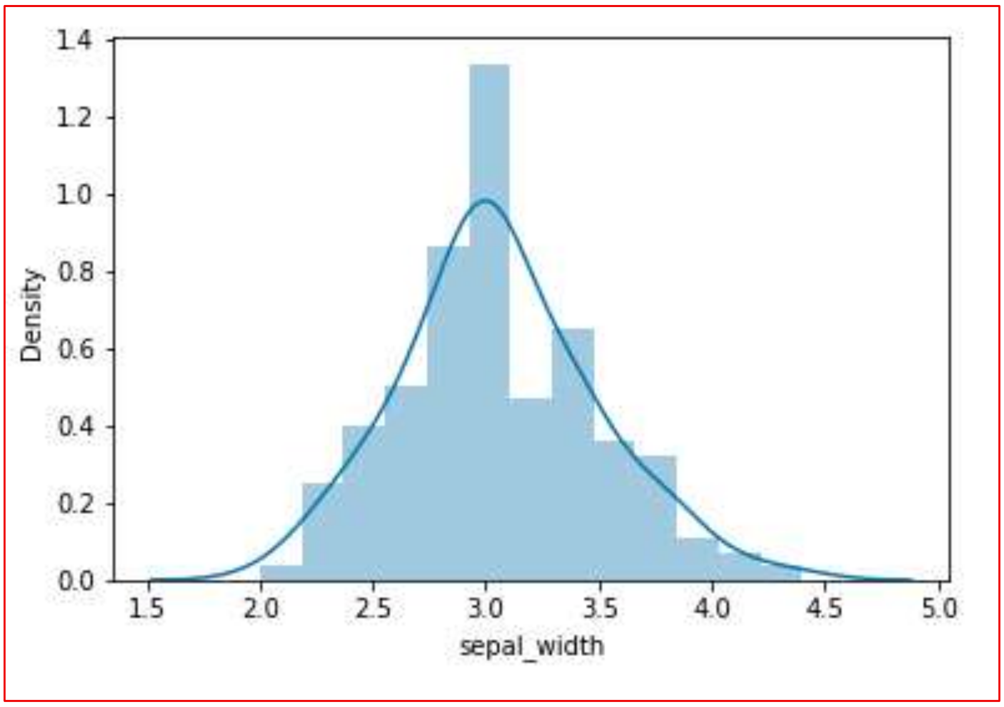
Syntax:

distplot(a[, bins, hist, kde, rug, fit, ...])

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.distplot(data['sepal_width'])
plt.show()
```

Output:



Jointplot

Jointplot is used to draw a plot of two variables with bivariate and univariate graphs. It basically combines two different plots. It is plotted using the **jointplot()** method.

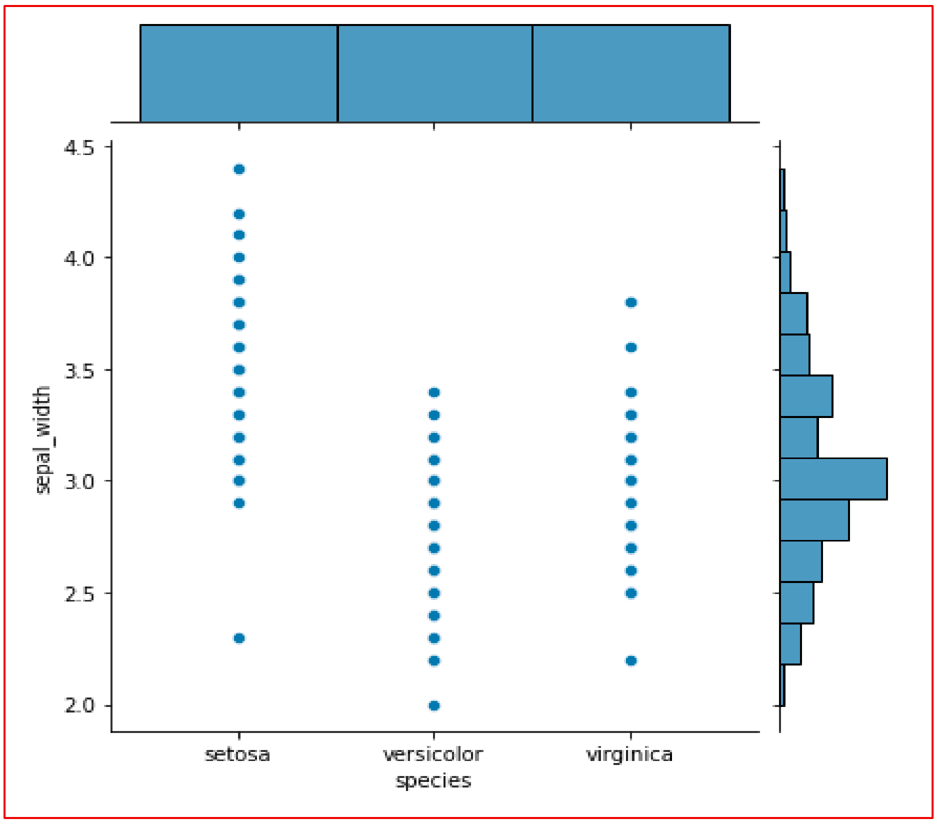
Syntax:

jointplot(x, y[, data, kind, stat_func, ...])

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.jointplot(x='species', y='sepal_width', data=data)
plt.show()
```

Output:



Pairplot

Pairplot represents pairwise relation across the entire dataframe and supports an additional argument called hue for categorical separation. What it does basically is create a jointplot between every possible numerical column and takes a while if the dataframe is really huge. It is plotted using the **pairplot()** method.

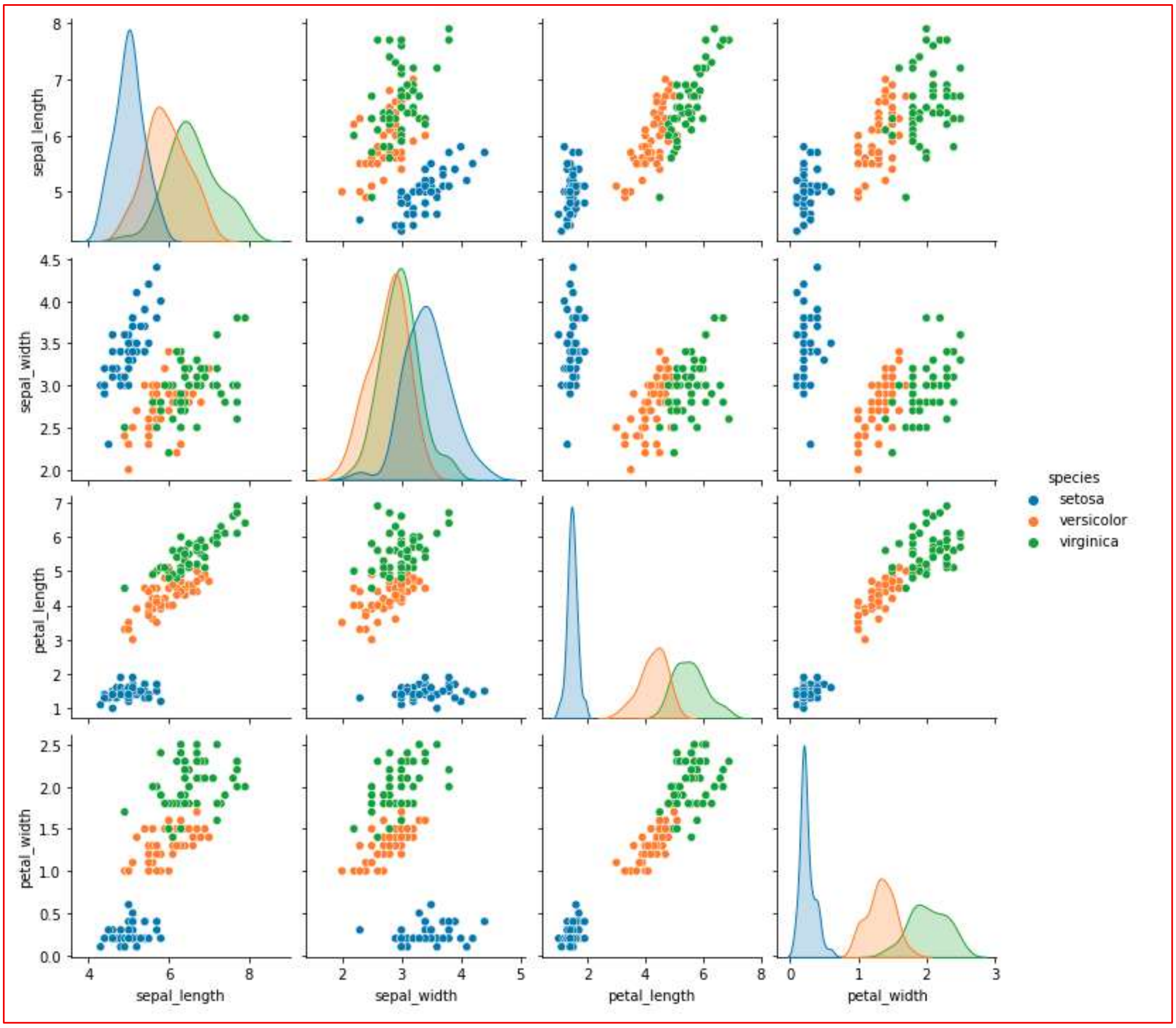
Syntax:

pairplot(data[, hue, hue_order, palette, …])

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.pairplot(data=data, hue='species')
plt.show()
```

Output:

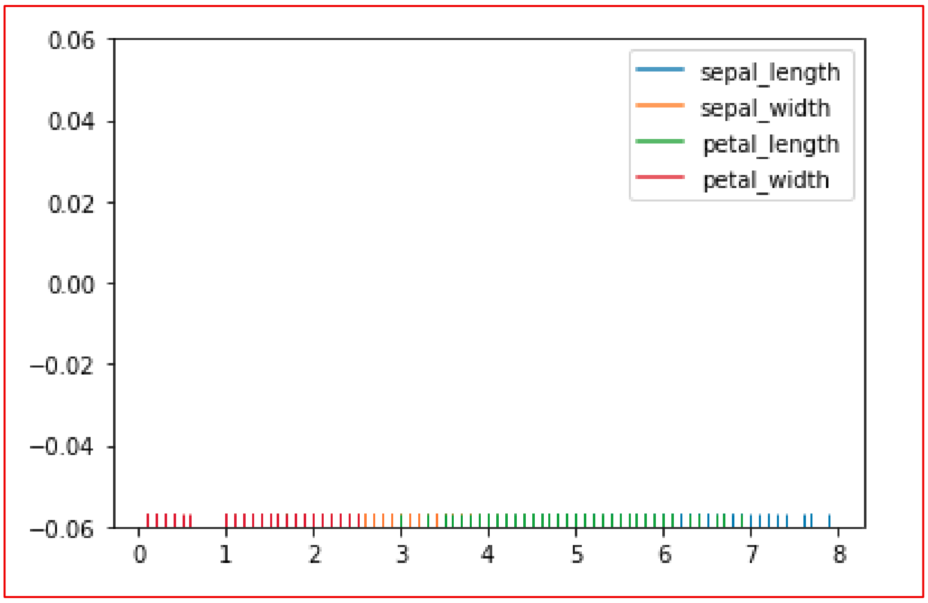


Rugplot
Rugplot plots datapoints in an array as sticks on an axis.Just like a distplot it takes a single column. Instead of drawing a histogram it creates dashes all across the plot. If you compare it with the joinplot you can see that what a jointplot does is that it counts the dashes and shows it as bins. It is plotted using the **rugplot()** method.

Syntax:
rugplot(a[, height, axis, ax])

Example:
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.rugplot(data=data)
plt.show()
```

Output:



KDE Plot

KDE Plot described as **Kernel Density Estimate** is used for visualizing the Probability Density of a continuous variable. It depicts the probability density at different values in a continuous variable. We can also plot a single graph for multiple samples which helps in more efficient data visualization.

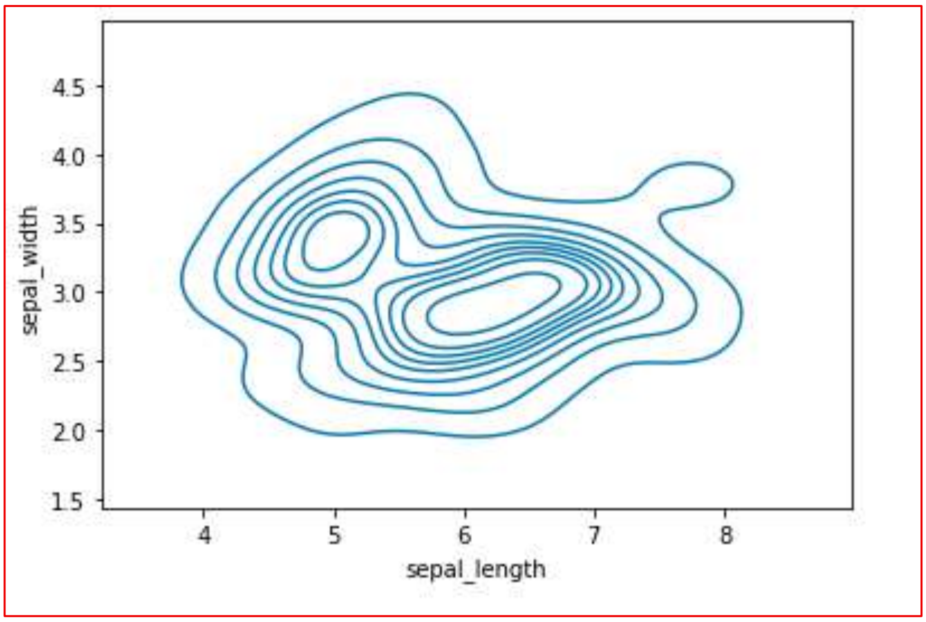
Syntax:

*seaborn.kdeplot(x=None, *, y=None, vertical=False, palette=None, **kwargs)*

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("iris")

sns.kdeplot(x='sepal_length', y='sepal_width', data=data)
plt.show()
```

Output:



Regression Plots

The **regression plots** are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analyses. Regression plots as the name suggests creates a regression line between two parameters and helps to visualize their linear relationships.

there are two main functions that are used to draw linear regression models. These functions are lmplot(), and regplot(), are closely related to each other. They even share their core functionality.

lmplot

lmplot() method can be understood as a function that basically creates a linear model plot. It creates a scatter plot with a linear fit on top of it.

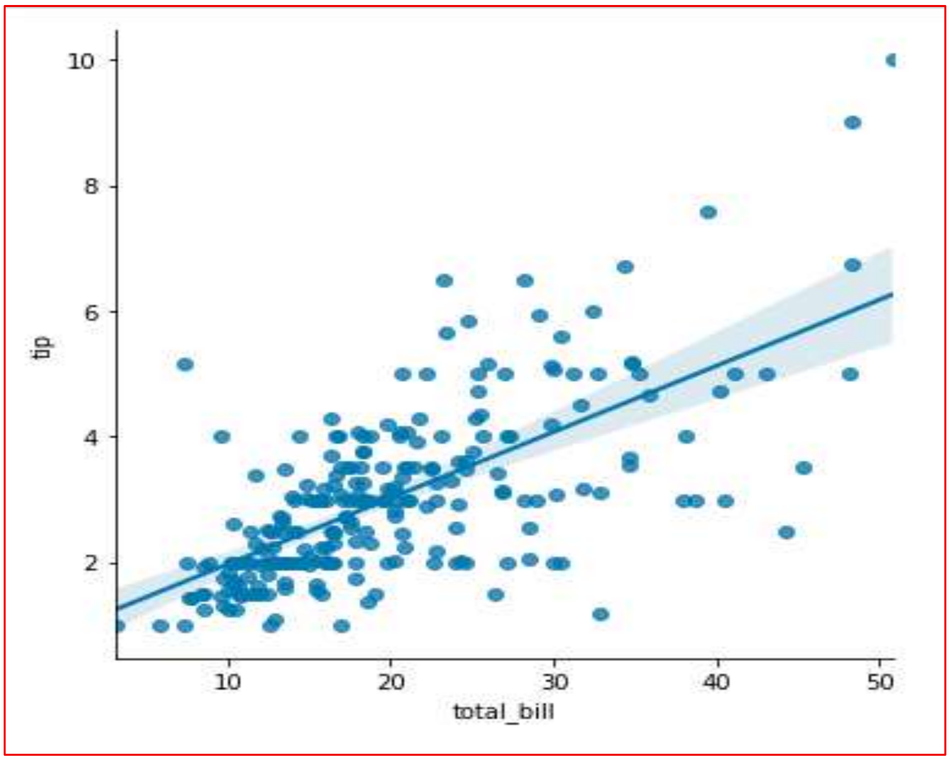
Syntax:

*seaborn.lmplot(x, y, data, hue=None, col=None, row=None, **kwargs)*

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("tips")

sns.lmplot(x='total_bill', y='tip', data=data)
plt.show()
```

Output:

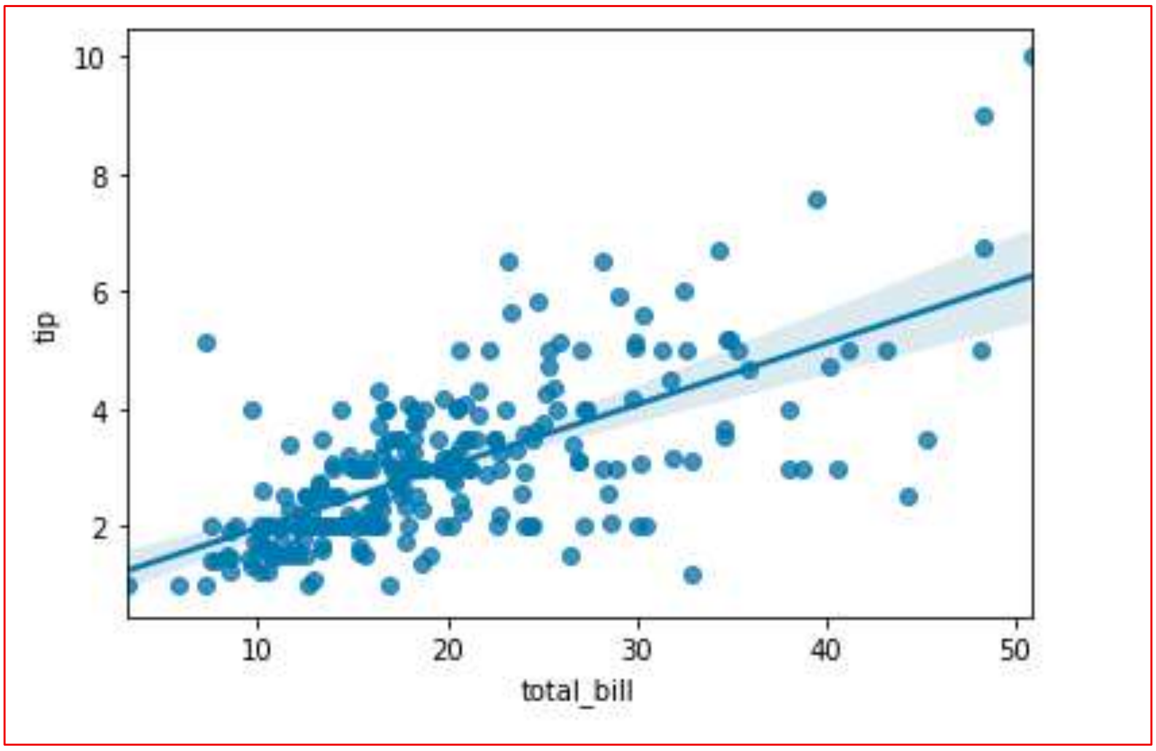


Regplot
regplot() method is also similar to lmplot which creates linear regression model.

Syntax:
*seaborn.regplot(x, y, data=None, x_estimator=None, **kwargs)*

Example:
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("tips")

sns.regplot(x='total_bill', y='tip', data=data)
plt.show()
```

Output:



Matrix Plots

A **matrix plot** means plotting matrix data where color coded diagrams shows rows data, column data and values. It can shown using the heatmap and clustermap.

Heatmap

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. it can be plotted using the **heatmap()** function.

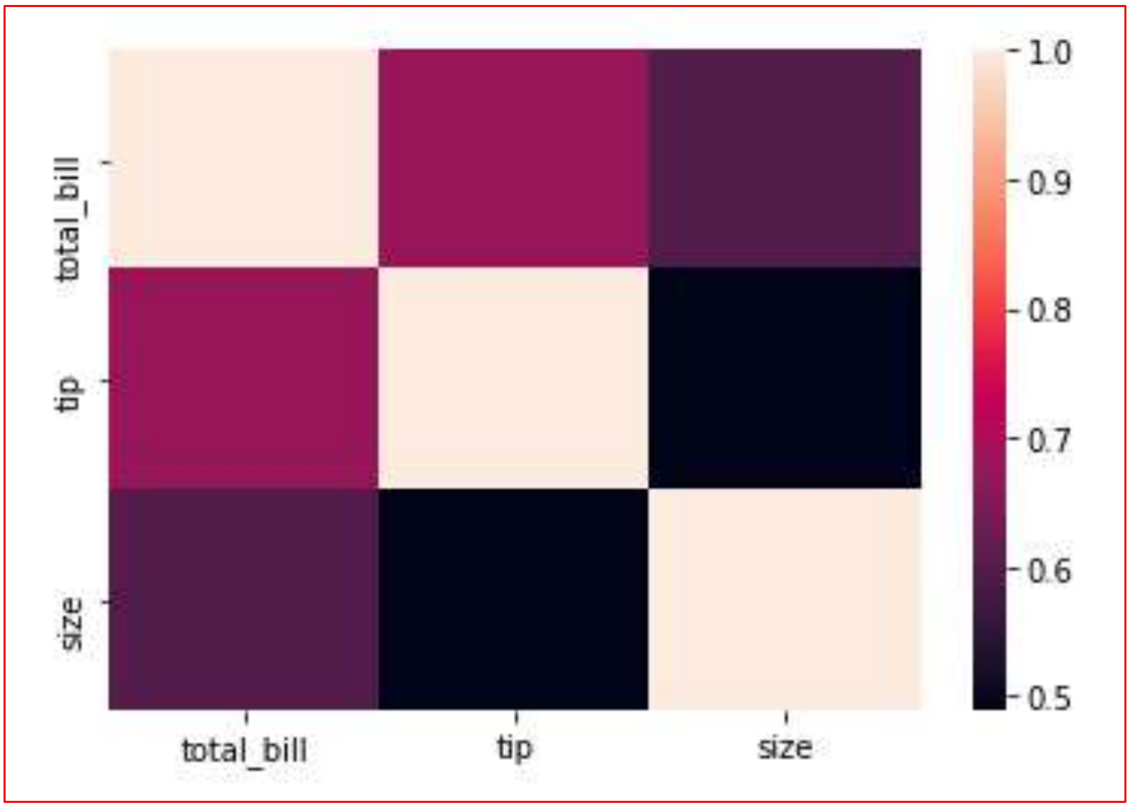
Syntax:

*seaborn.heatmap(data, *, vmin=None, vmax=None, cmap=None, center=None, annot_kws=None, linewidths=0, linecolor='white', cbar=True, **kwargs)*

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("tips")

# correlation between the different parameters
tc = data.corr()

sns.heatmap(tc)
plt.show()
```

Output:



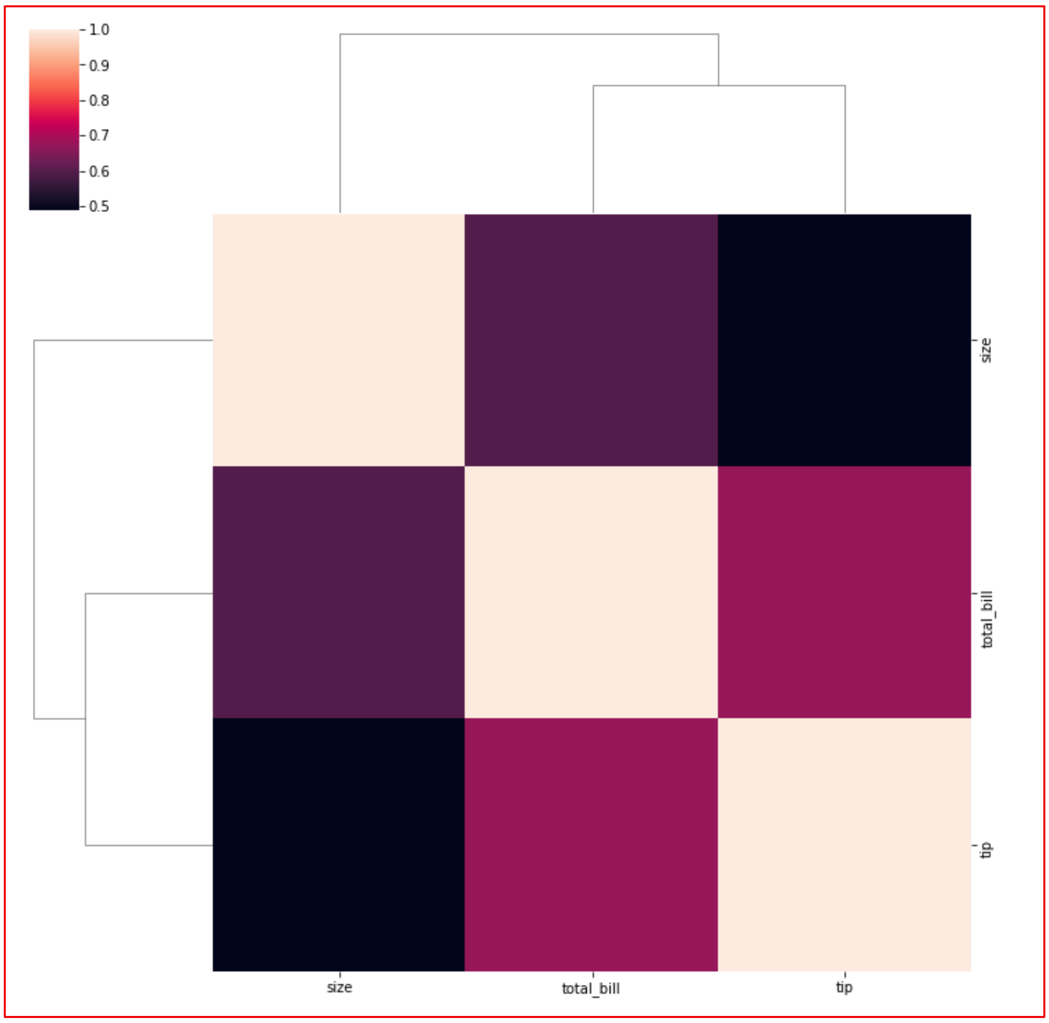
Clustermap
The clustermap() function of seaborn plots the hierarchically-clustered heatmap of the given matrix dataset. Clustering simply means grouping data based on relationship among the variables in the data.

Syntax:
*clustermap(data, *, pivot_kws=None, **kwargs)*

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("tips")

# correlation between the different parameters
tc = data.corr()

sns.clustermap(tc)
plt.show()
```

Output:

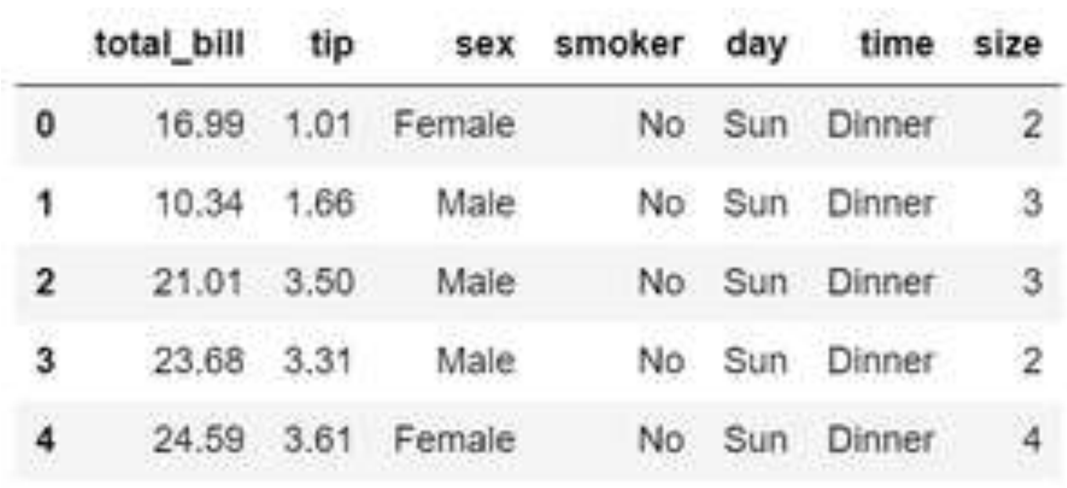


Change Axis Labels, Set Title and Figure Size to Plots with Seaborn

Seaborn is Python's visualization library built as an extension to Matplotlib. Seaborn has **Axes-level functions** (scatterplot, regplot, boxplot, kdeplot, etc.) as well as **Figure-level functions** (lmplot, factorplot, jointplot, relplot etc.). Axes-level functions return Matplotlib axes objects with the plot drawn on them while figure-level functions include axes that are always organized in a meaningful way. The basic customization that a graph needs to make it understandable is setting the title, setting the axis labels, and adjusting the figure size. Any customization made is on the axes object for axes-level functions and the figure object for figure-level functions.

```
# Import required libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Load data set
tips = sns.load_dataset( "tips" )
tips.head()
```

Output:

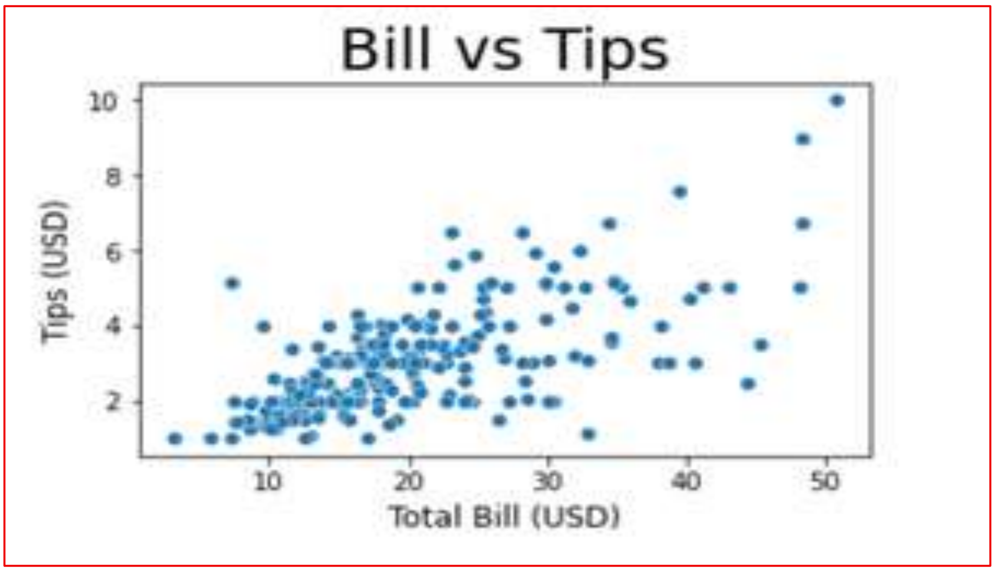


	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Example 1: Customizing plot with axes object
For axes-level functions, pass the figsize argument to the plt.subplots() function to set the figure size. The function plt.subplots() returns Figure and Axes objects. These objects are created ahead of time and later the plots are drawn on it. We make use of the set_title(), set_xlabel(), and set_ylabel() functions to change axis labels and set the title for a plot. We can set the size of the text with size attribute. Make sure to assign the axes-level object while creating the plot. This object is then used for setting the title and labels as shown below.

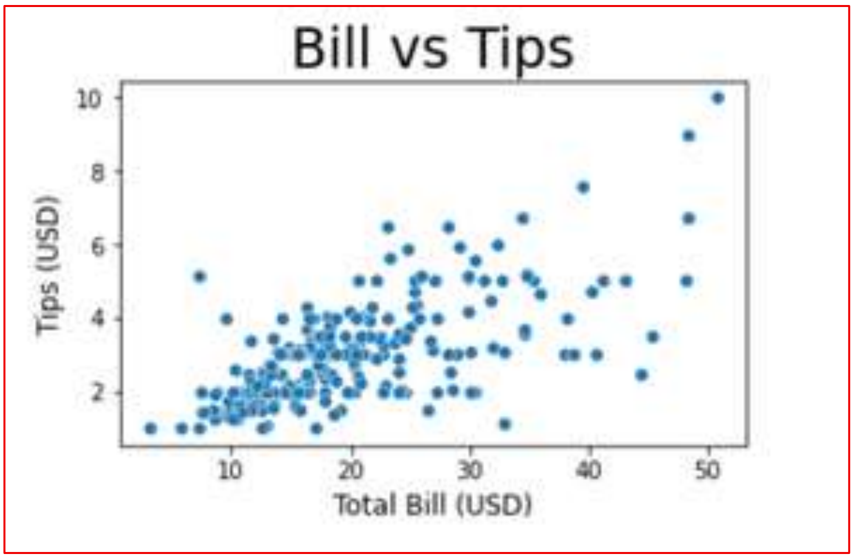
```
# Set figure size (width, height) in inches
fig, ax = plt.subplots(figsize = ( 5 , 3 ))

# Plot the scatterplot
sns.scatterplot( ax = ax , x = "total_bill" , y = "tip" , data = tips )

# Set label for x-axis
ax.set_xlabel( "Total Bill (USD)" , size = 12 )

# Set label for y-axis
ax.set_ylabel( "Tips (USD)" , size = 12 )

# Set title for plot
ax.set_title( "Bill vs Tips" , size = 24 )

# Display figure
plt.show()
```

Output:



Example 2: Customizing scatter plot with pyplot object
We can also change the axis labels and set the plot title with the matplotlib.pyplot object using xlabel(), ylabel() and title() functions. Similar to the above example, we can set the size of the text with the size attribute. The function plt.figure() creates a Figure instance and the figsize argument allows to set the figure size.

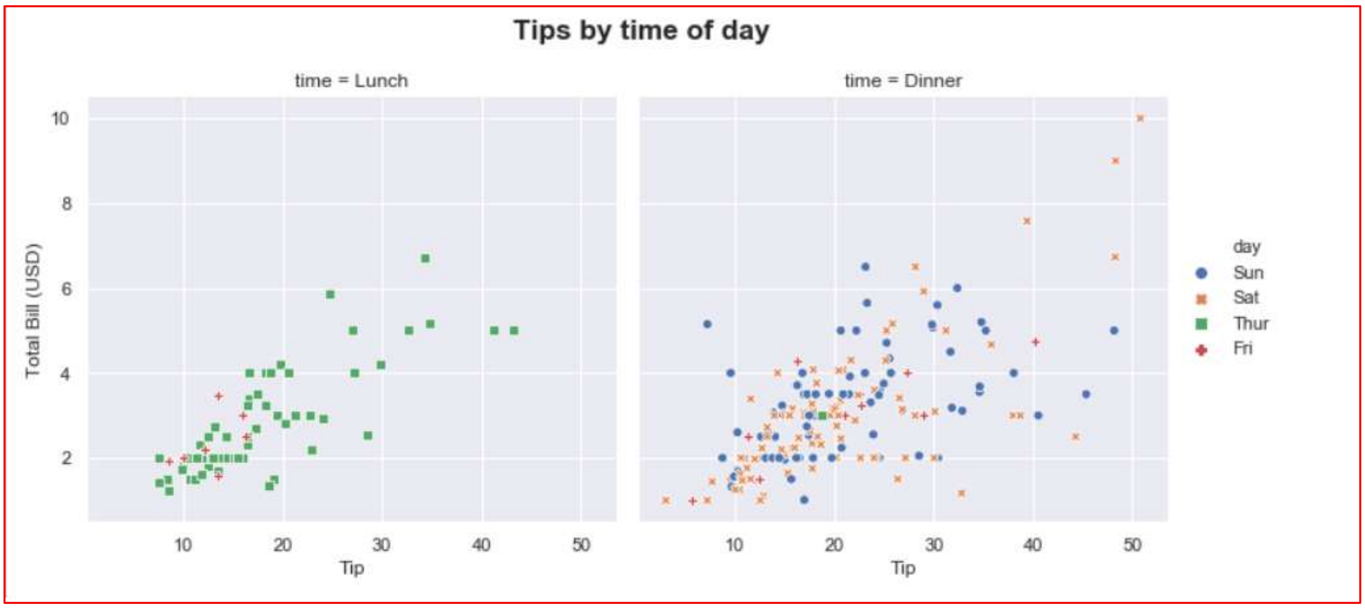
```
# Set figure size (width, height) in inches
plt.figure(figsize = ( 5 , 3 ))

# Plot scatterplot
sns.scatterplot( x = "total_bill" , y = "tip" , data = tips )

# Set label for x-axis
plt.xlabel( "Total Bill (USD)" , size = 12 )

# Set label for y-axis
plt.ylabel( "Tips (USD)" , size = 12 )

# Set title for figure
plt.title( "Bill vs Tips" , size = 24 )

# Display figure
plt.show()
```
Output:



Example 3: Customizing multiple plots in the same figure

Seaborn's relplot function returns a FacetGrid object which is a figure-level object. This object allows the convenient management of subplots. To give a title to the complete figure containing multiple subplots, we use the suptitle() method. The subplots_adjust() method is used to avoid overlapping of subplot titles and the figure title by specifying the top, bottom, left, and right edge positions of the subplots. To set the figure size, pass a dictionary with the key 'figure.figsize' in the set() method. The set() method allows to set multiple theme parameters in a single step.

```
# Set figure size
sns.set( rc = {'figure.figsize' : ( 20, 20 ),
              'axes.labelsize' : 12 })

# Plot scatter plot
g = sns.relplot(data = tips , x = "total_bill" ,
               y = "tip" , col = "time" ,
               hue = "day" , style = "day" ,
               kind = "scatter" )

# Title for the complete figure
g.fig.suptitle("Tips by time of day" ,
              fontsize = 'x-large' ,
              fontweight = 'bold' )

# Adjust subplots so that titles don't overlap
g.fig.subplots_adjust( top = 0.85 )

# Set x-axis and y-axis labels
g.set_axis_labels( "Tip" , "Total Bill (USD)" )

# Display the figure
plt.show()
```
Output:



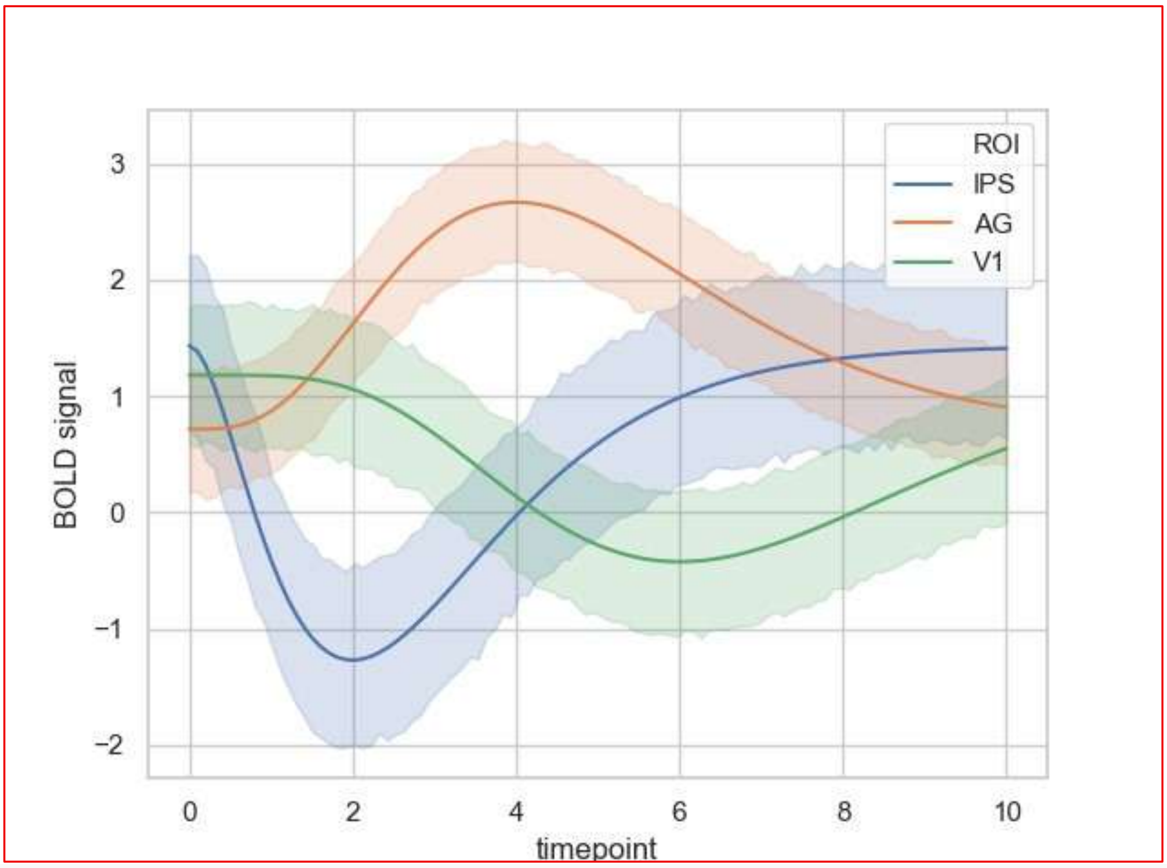
How To Place Legend Outside the Plot with Seaborn in Python
Seaborn is a Python **data visualization library** based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Basically, it helps us stylize our basic plot made using matplotlib. Moreover, it also provides us different plotting techniques to ease our **Exploratory Data Analysis(EDA)**. With these plots, it also becomes important to provide legends for a particular plot.
In this following article, we are going to see how can we place our Legend on our plot, and later in this article, we will also see how can we place the legend outside the plot using **Seaborn**.

```
import seaborn as sns
import matplotlib.pyplot as plt
```

We will be using Seaborn for not only plotting the data but also importing our dataset. Here we will be using the **Gamma dataset** by seaborn.

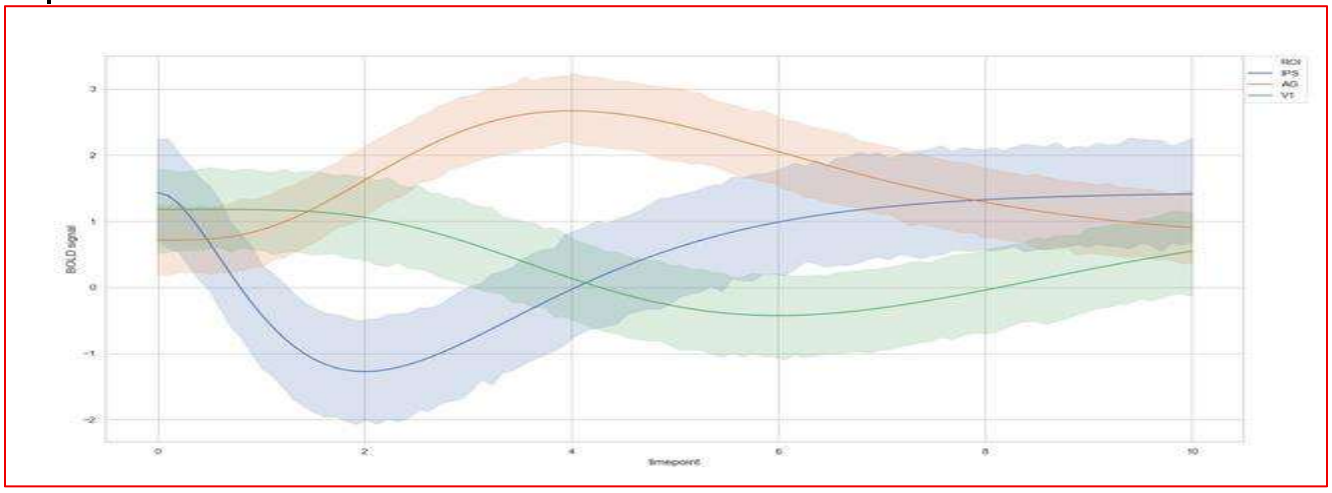
```
# set our graph style to whitegrid
sns.set(style="whitegrid")

# load the gammas dataset
ds = sns.load_dataset("gammas")

# use seaborn's lineplot to plot our timeplot
# and BOLD signal columns
sns.lineplot(data=ds, x="timepoint", y="BOLD signal", hue = "ROI")

plt.show()
```

Output:



```
plt.legend(bbox_to_anchor=(1, 1), loc=2)
```

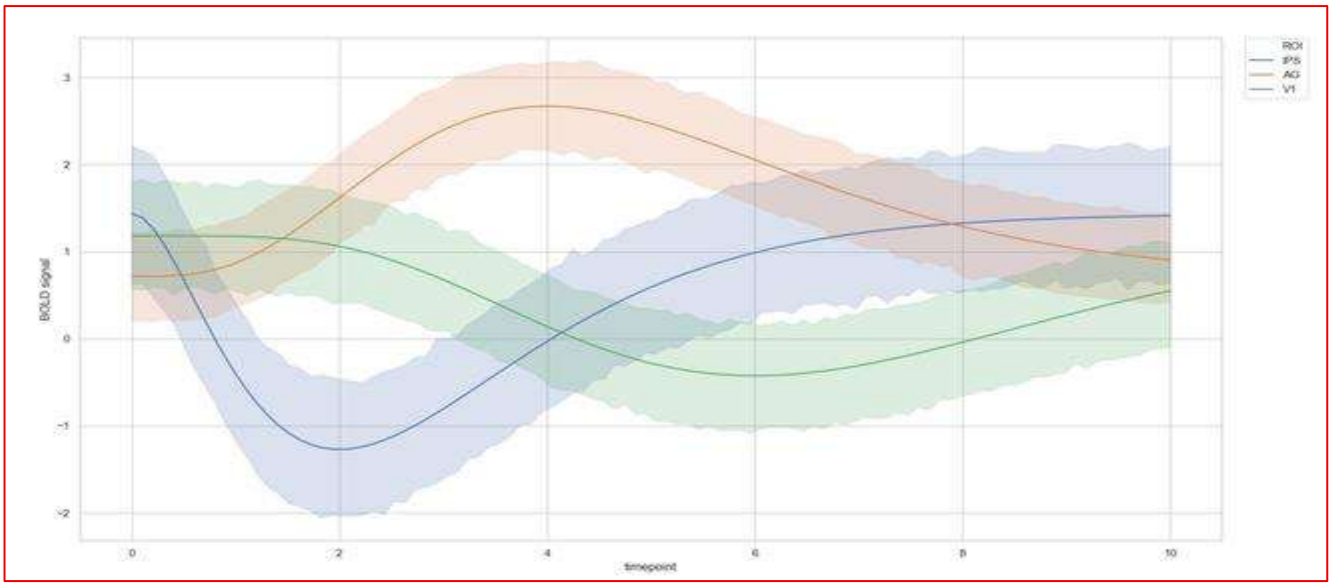
Output:



We can also tune our parameters according to our necessities.

```
plt.legend(bbox_to_anchor=(1.02, 1), loc=2)
```

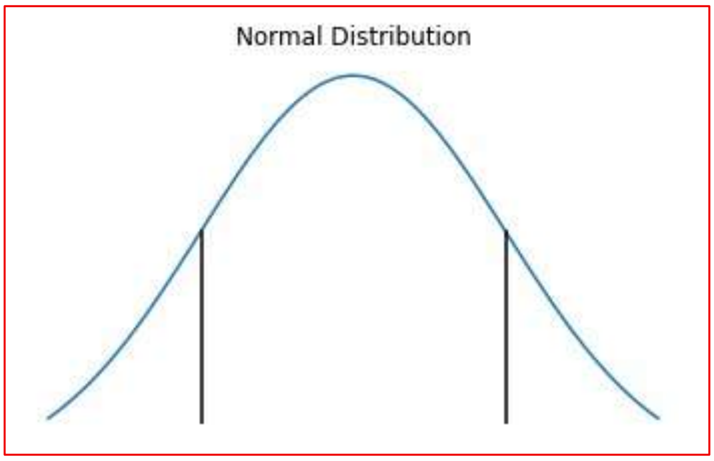
Output:



How to Plot a Confidence Interval in Python

Confidence Interval is a type of estimate computed from the statistics of the observed data which gives a range of values that's likely to contain a population parameter with a particular level of confidence.

A 95% confidence interval, will tell me that if we take an infinite number of samples from my population, calculate the interval each time, then in 95% of those intervals, the interval will contain the true population mean. So, with one sample we can calculate the sample mean, and from there get an interval around it, that most likely will contain the true population mean.



Confidence Interval as a concept was put forth by Jerzy Neyman in a paper published in 1937. There are various types of the confidence interval, some of the most commonly used ones are: CI for mean, CI for the median, CI for the difference between means, CI for a proportion and CI for the difference in proportions.

Computing C.I given the underlying distribution using lineplot()
The lineplot() function which is available in *Seaborn,* a data visualization library for Python is best to show trends over a period of time however it also helps in plotting the confidence interval.

Syntax:
sns.lineplot(x=None, y=None, hue=None, size=None, style=None, data=None, palette=None, hue_order=None, hue_norm=None, sizes=None, size_order=None, size_norm=None, dashes=True, markers=None,

*style_order=None, units=None, estimator='mean', ci=95, n_boot=1000, sort=True, err_style='band', err_kws=None, legend='brief', ax=None, **kwargs,)*

Parameters:

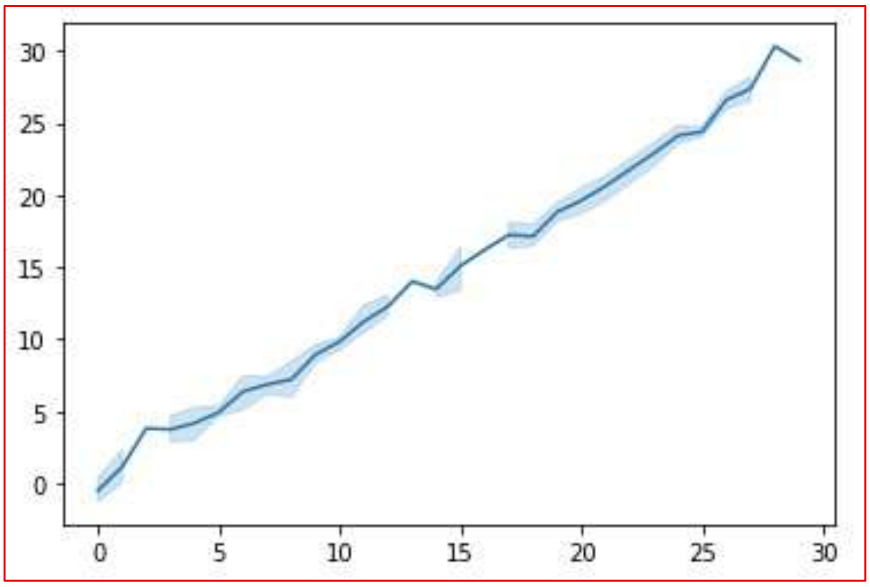
- ***x, y:*** *Input data variables; must be numeric. Can pass data directly or reference columns in data.*
- ***hue:*** *Grouping variable that will produce lines with different colors. Can be either categorical or numeric, although color mapping will behave differently in latter case.*
- ***style:*** *Grouping variable that will produce lines with different dashes and/or markers. Can have a numeric dtype but will always be treated as categorical.*
- ***data:*** *Tidy ("long-form") dataframe where each column is a variable and each row is an observation.*
- ***markers:*** *Object determining how to draw the markers for different levels of the style variable.*
- ***legend:*** *How to draw the legend. If "brief", numeric ``hue`` and ``size`` variables will be represented with a sample of evenly spaced values.*

Return: *The Axes object containing the plot.*

By default, the plot aggregates over multiple y values at each value of x and shows an estimate of the central tendency and a confidence interval for that estimate.

Example:

```
# import libraries
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# generate random data
np.random.seed(0)
x = np.random.randint(0, 30, 100)
y = x+np.random.normal(0, 1, 100)

# create lineplot
ax = sns.lineplot(x, y)
```



The light blue shade indicates the confidence level around that point if it has higher confidence the shaded line will be thicker.

Computing C.I. given the underlying distribution using regplot()

The seaborn.regplot() helps to plot data and a linear regression model fit. This function also allows plotting the confidence interval.

Syntax:
seaborn.regplot(x, y, data=None, x_estimator=None, x_bins=None, x_ci='ci', scatter=True, fit_reg=True, ci=95, n_boot=1000, units=None, order=1, logistic=False, lowess=False, robust=False, logx=False, x_partial=None, y_partial=None, truncate=False, dropna=True, x_jitter=None, y_jitter=None, label=None, color=None, marker='o',   scatter_kws=None, line_kws=None, ax=None)

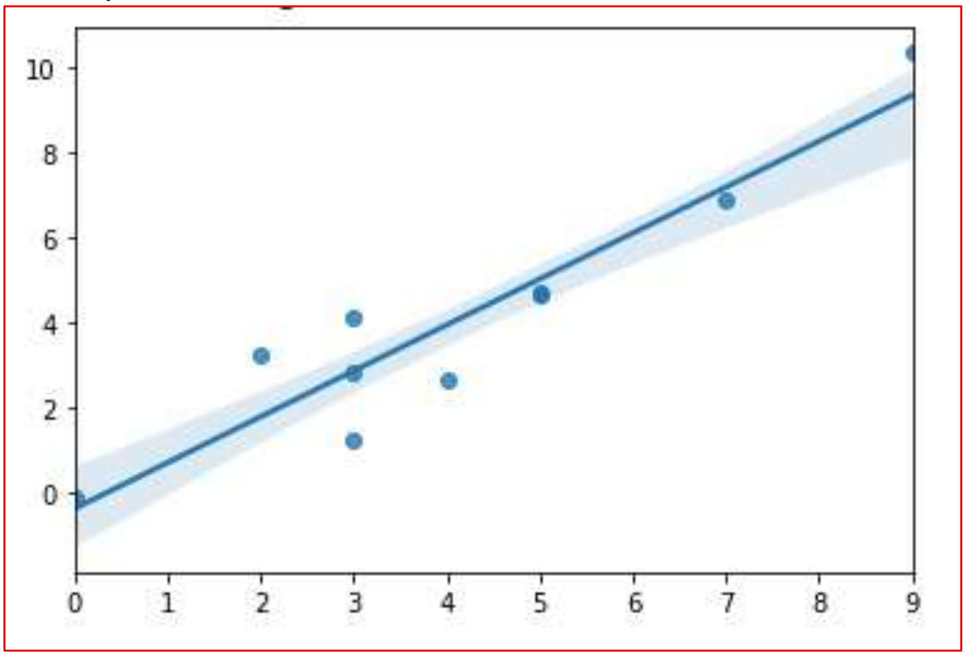
Parameters: *The description of some main parameters are given below:*

- ***x, y:*** *These are Input variables. If strings, these should correspond with column names in "data". When pandas objects are used, axes will be labeled with the series name.*
- ***data:*** *This is dataframe where each column is a variable and each row is an observation.*
- ***lowess:*** *(optional) This parameter take boolean value. If "True", use "statsmodels" to estimate a nonparametric lowess model (locally weighted linear regression).*
- ***color:*** *(optional) Color to apply to all plot elements.*
- ***marker:*** *(optional) Marker to use for the scatterplot glyphs.*

Return: *The Axes object containing the plot.*
Basically, it includes a regression line in the *scatterplot* and helps in seeing any linear relationship between two variables. Below example will show how it can be used to plot confidence interval as well.
Example:

```
# import libraries
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# create random data
np.random.seed(0)
x = np.random.randint(0, 10, 10)
y = x+np.random.normal(0, 1, 10)

# create regression plot
ax = sns.regplot(x, y, ci=80)
```

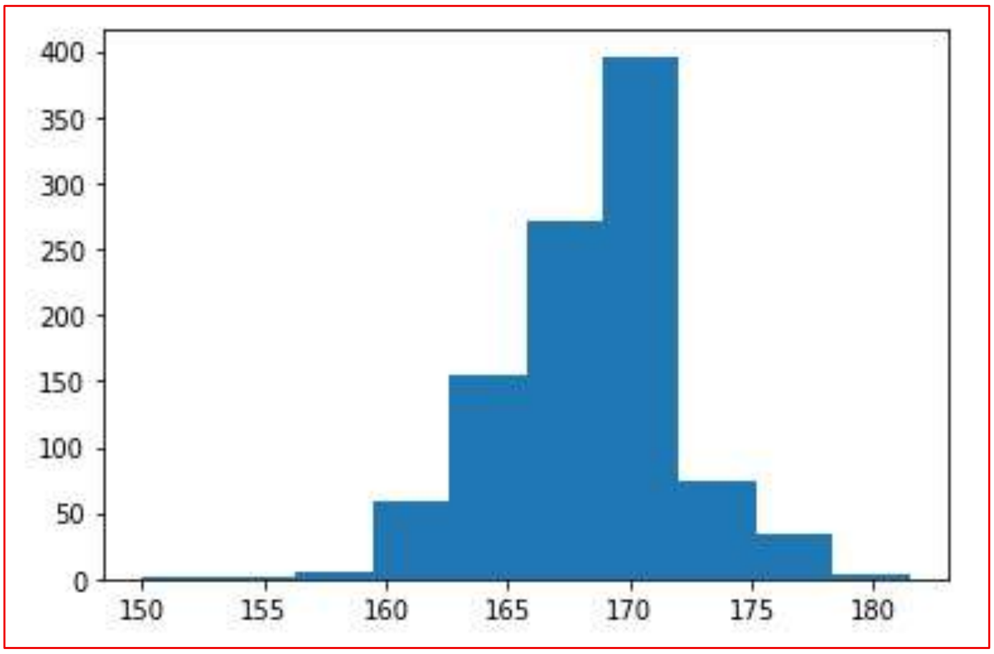
The *regplot()* function works in the same manner as the *lineplot()* with a 95% confidence interval by default. Confidence interval can easily be changed by changing the value of the parameter 'ci' which lies in the range of [0, 100]. Here I have passed ci=80 which means instead of the default 95% confidence interval, an 80% confidence interval is plotted.



The width of light blue color shade indicates the confidence level around the regression line.

Computing C.I. using Bootstrapping
Bootstrapping is a test/metric that uses random sampling with replacement. It gives the measure of accuracy (bias, variance, confidence intervals, prediction error, etc.) to sample estimates. It allows the estimation of the sampling distribution for most of the statistics using random sampling methods. It may also be used for constructing hypothesis tests.

Example:
```
# import libraries
import pandas
import numpy
from sklearn.utils import resample
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt

# load dataset
x = numpy.array([180,162,158,172,168,150,171,183,165,176])

# configure bootstrap
n_iterations = 1000 # here k=no. of bootstrapped samples
n_size = int(len(x))

# run bootstrap
medians = list()
for i in range(n_iterations):
   s = resample(x, n_samples=n_size);
   m = numpy.median(s);
   medians.append(m)

# plot scores
plt.hist(medians)
plt.show()

# confidence intervals
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower =  numpy.percentile(medians, p)
p = (alpha+((1.0-alpha)/2.0)) * 100
upper =  numpy.percentile(medians, p)

print(f"\n{alpha*100} confidence interval {lower} and {upper}")
```



Creating A Time Series Plot With Seaborn And Pandas

- **Seaborn** is a tremendous visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to form statistical plots more attractive. It's built on the highest of matplotlib library and also closely integrated to the info structures from pandas.
- A **timeplot** (sometimes called a statistic graph) displays values against the clock. They're almost like x-y graphs, but while an x-y graph can plot a spread of "x" variables (for example, height, weight, age), timeplots can only display time on the x-axis. Unlike the pie charts and bar charts, these plots don't have categories. Timeplots are good for showing how data changes over time. For instance, this sort of chart would work well if you were sampling data randomly times.

Steps Needed

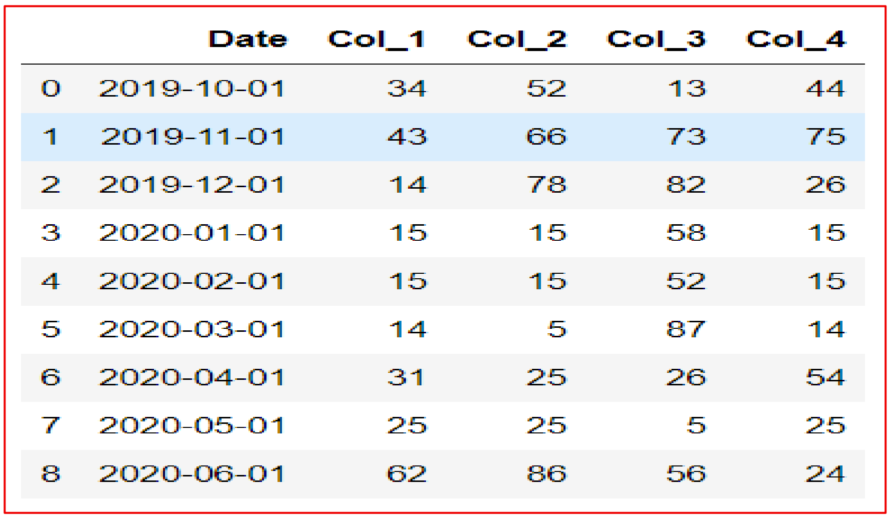
1. Import packages
2. Import / Load / Create data.
3. Plot the time series plot over data using lineplot (as tsplot was replaced with lineplot since Sep 2020).

Examples

# importing packages

```
import pandas as pd

# creating data
df = pd.DataFrame({'Date': ['2019-10-01', '2019-11-01',
                    '2019-12-01','2020-01-01',
                    '2020-02-01', '2020-03-01',
                    '2020-04-01', '2020-05-01',
                    '2020-06-01'],
           'Col_1': [34, 43, 14, 15,
                     15, 14, 31, 25, 62],

           'Col_2': [52, 66, 78, 15, 15,
                     5, 25, 25, 86],

           'Col_3': [13, 73, 82, 58, 52,
                     87, 26, 5, 56],

           'Col_4': [44, 75, 26, 15, 15,
                     14, 54, 25, 24]})

# view dataset
display(df)
```

Output:

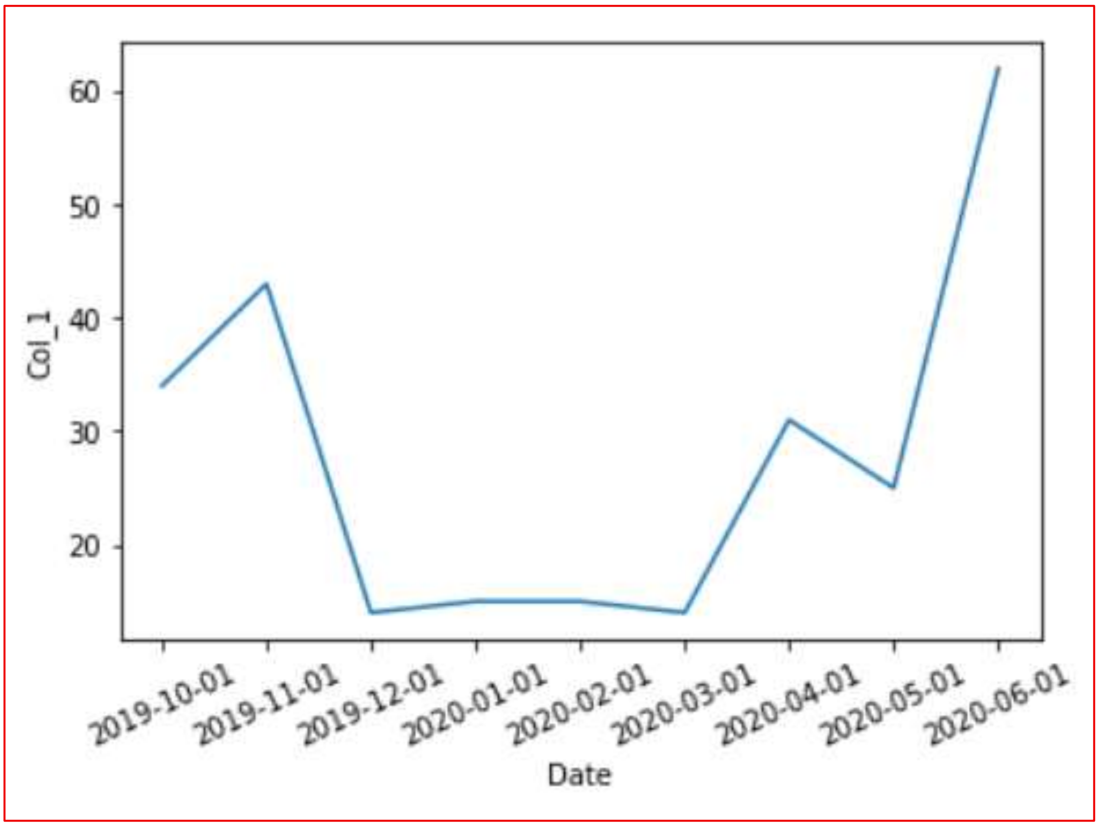


	Date	Col_1	Col_2	Col_3	Col_4
0	2019-10-01	34	52	13	44
1	2019-11-01	43	66	73	75
2	2019-12-01	14	78	82	26
3	2020-01-01	15	15	58	15
4	2020-02-01	15	15	52	15
5	2020-03-01	14	5	87	14
6	2020-04-01	31	25	26	54
7	2020-05-01	25	25	5	25
8	2020-06-01	62	86	56	24

Example 1: Simple time series plot with single column using lineplot

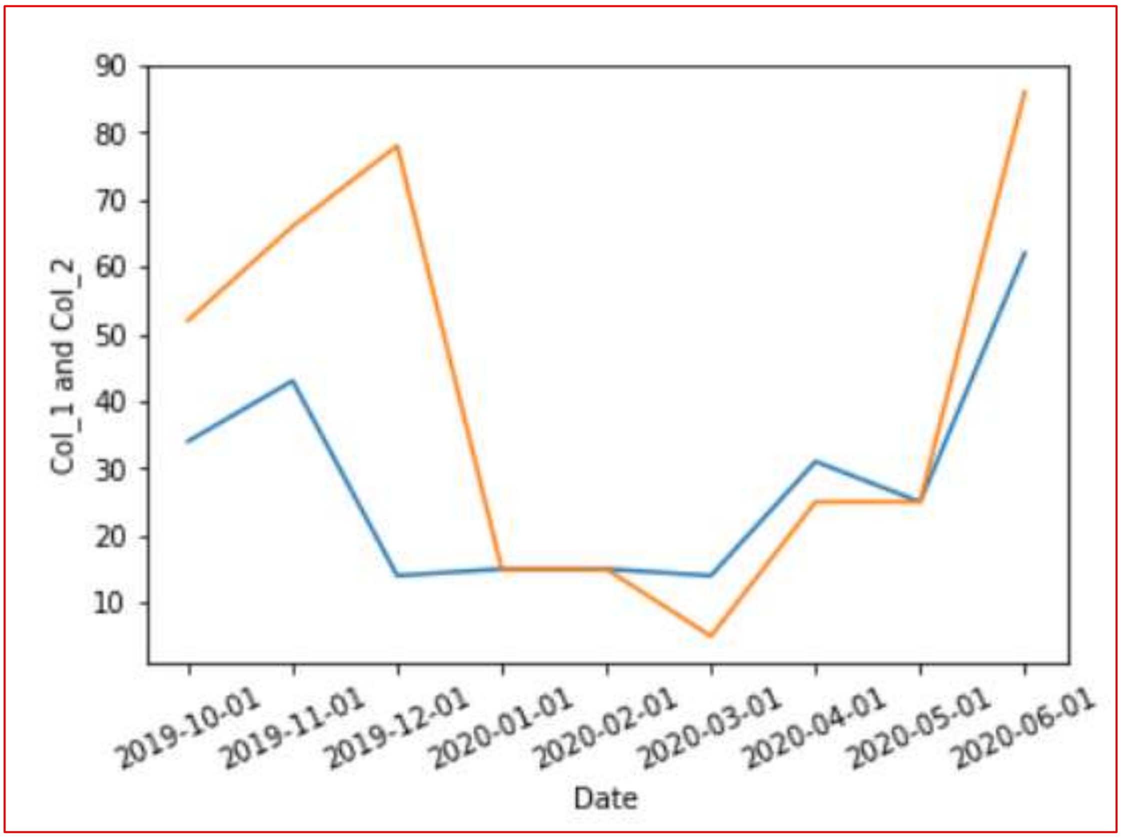
```
# importing packages
import seaborn as sns
import pandas as pd

# creating data
df = pd.DataFrame({'Date': ['2019-10-01', '2019-11-01',
                  '2019-12-01','2020-01-01',
                  '2020-02-01', '2020-03-01',
                  '2020-04-01', '2020-05-01',
                  '2020-06-01'],

           'Col_1': [34, 43, 14, 15, 15,
                     14, 31, 25, 62],

           'Col_2': [52, 66, 78, 15, 15,
                     5, 25, 25, 86],

           'Col_3': [13, 73, 82, 58, 52,
                     87, 26, 5, 56],
           'Col_4': [44, 75, 26, 15, 15,
                     14, 54, 25, 24]})
```

```
# create the time series plot
sns.lineplot(x = "Date", y = "Col_1",
            data = df)

plt.xticks(rotation = 25)
```
Output :



Example 2: (Simple time series plot with multiple columns using line plot)
```
# importing packages
import seaborn as sns
import pandas as pd

# creating data
df = pd.DataFrame({'Date': ['2019-10-01', '2019-11-01',
                    '2019-12-01','2020-01-01',
                    '2020-02-01', '2020-03-01',
                    '2020-04-01', '2020-05-01',
                    '2020-06-01'],

               'Col_1': [34, 43, 14, 15, 15,
                        14, 31, 25, 62],

               'Col_2': [52, 66, 78, 15, 15,
                        5, 25, 25, 86],

               'Col_3': [13, 73, 82, 58, 52,
                        87, 26, 5, 56],
               'Col_4': [44, 75, 26, 15, 15,
                        14, 54, 25, 24]})

# create the time series plot
sns.lineplot(x = "Date", y = "Col_1", data = df)
sns.lineplot(x = "Date", y = "Col_2", data = df)
plt.ylabel("Col_1 and Col_2")
plt.xticks(rotation = 25)
```

Output :

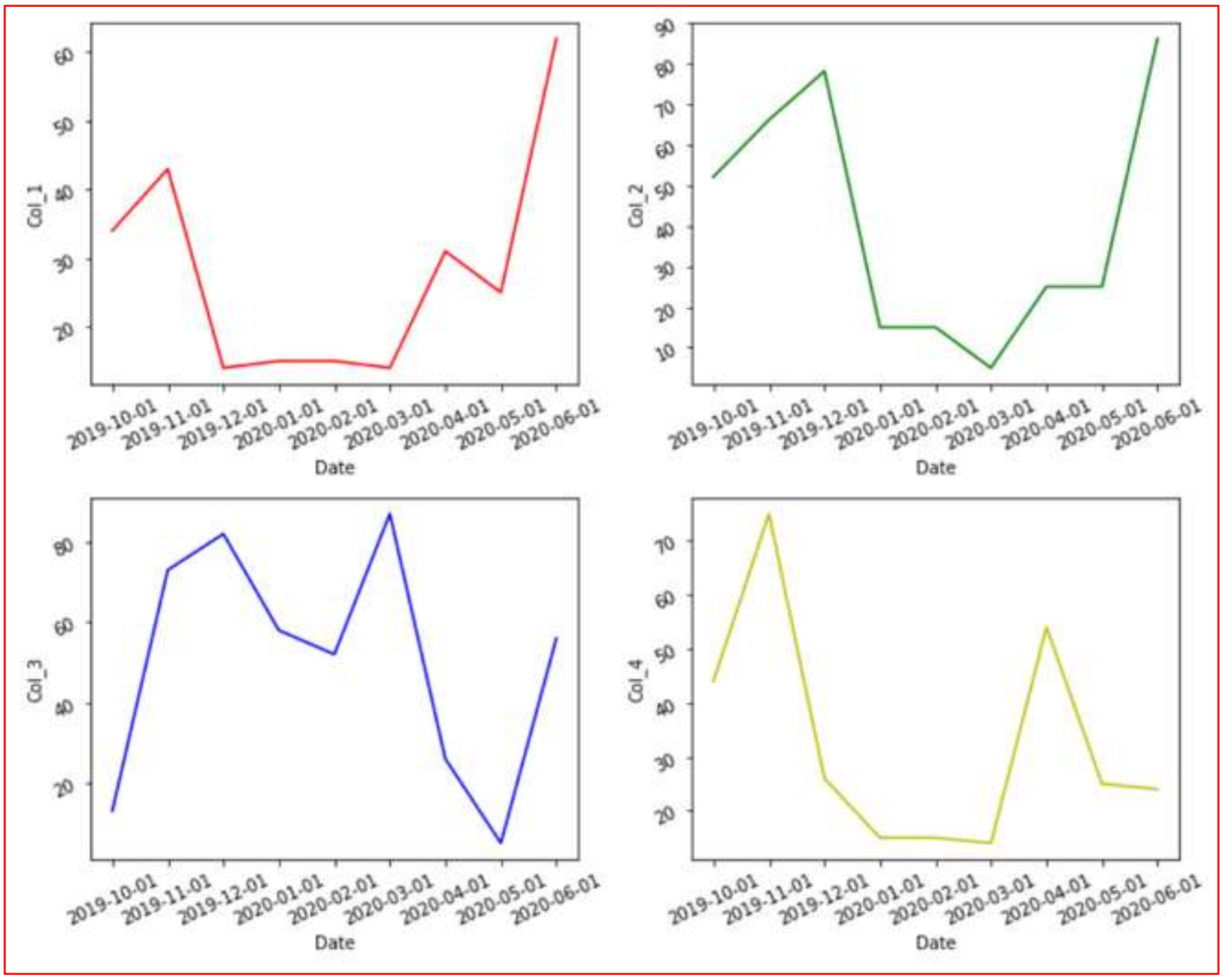


Example 3: Multiple time series plot with multiple columns

```
# importing packages
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# creating data
df = pd.DataFrame({'Date': ['2019-10-01', '2019-11-01',
                            '2019-12-01','2020-01-01',
                            '2020-02-01', '2020-03-01',
                            '2020-04-01', '2020-05-01',
                            '2020-06-01'],

                  'Col_1': [34, 43, 14, 15, 15,
                            14, 31, 25, 62],

                  'Col_2': [52, 66, 78, 15, 15,
                            5, 25, 25, 86],

                  'Col_3': [13, 73, 82, 58, 52,
                            87, 26, 5, 56],
                  'Col_4': [44, 75, 26, 15, 15,
                            14, 54, 25, 24]})
# create the time series subplots
fig,ax =  plt.subplots( 2, 2,
                        figsize = ( 10, 8))
```

```
sns.lineplot( x = "Date", y = "Col_1",
            color = 'r', data = df,
            ax = ax[0][0])

ax[0][0].tick_params(labelrotation = 25)
sns.lineplot( x = "Date", y = "Col_2",
            color = 'g', data = df,
            ax = ax[0][1])

ax[0][1].tick_params(labelrotation = 25)
sns.lineplot(x = "Date", y = "Col_3",
            color = 'b', data = df,
            ax = ax[1][0])

ax[1][0].tick_params(labelrotation = 25)

sns.lineplot(x = "Date", y = "Col_4",
            color = 'y', data = df,
            ax = ax[1][1])

ax[1][1].tick_params(labelrotation = 25)
fig.tight_layout(pad = 1.2)
```

Output :



Seaborn - Bubble Plot

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on the top of matplotlib library and also closely integrated to the data structures from pandas.
Scatter plots are used to observe relationship between variables and uses dots to represent the relationship between them. **Bubble plots** are scatter plots with bubbles (color filled circles) rather than information focuses. Bubbles have various sizes dependent on another variable in the data. Likewise, Bubbles can be of various color dependent on another variable in the dataset.
Let us load the required module and the simplified Iris data as a Pandas Data frame:

```
# import all important libraries
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# load dataset
data=
"https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv"

# convert to dataframe
df = pd.read_csv(data)

# display top most rows
df.head()
```

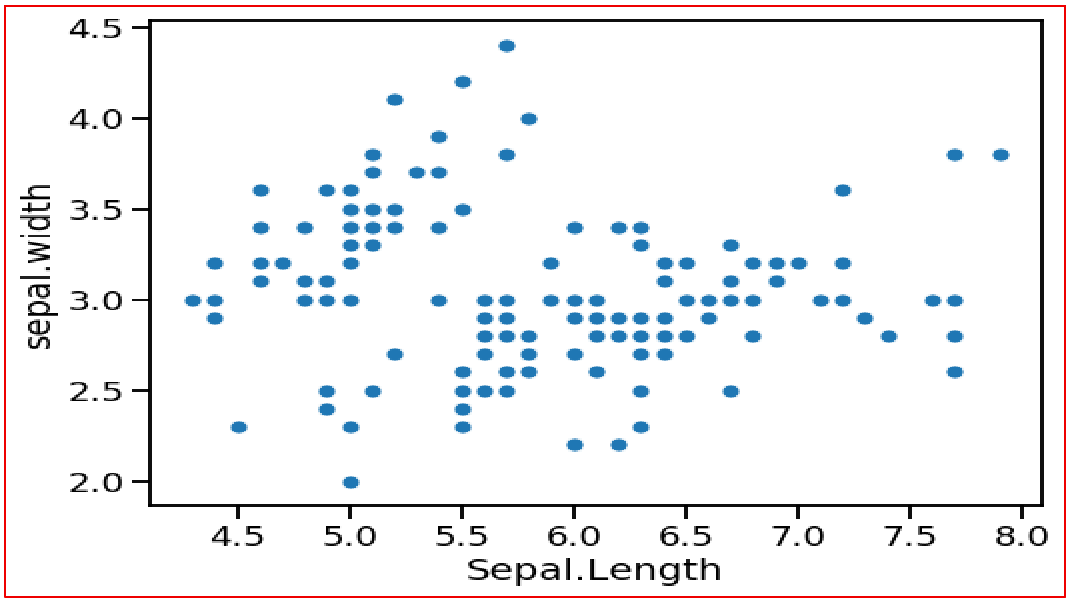
Scatter plot with Seaborn:

As stated earlier than, bubble is a unique form of scatter plot with bubbles as opposed to easy facts points in scatter plot. Let us first make a simple scatter plot the usage of *Seaborn*'s *scatterplot()* function.

```
# import all important libraries
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# load dataset
data =
"https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv"

# convert to dataframe
df = pd.read_csv(data)

# display top most rows
df.head()

# depict scatterplot illustration
sns.set_context("talk", font_scale=1.1)
plt.figure(figsize=(8, 6))
sns.scatterplot(x="sepal.length",
            y="sepal.width",
            data=df)
```

```
# assign labels
plt.xlabel("Sepal.Length")
plt.ylabel("sepal.width")
```

Output:

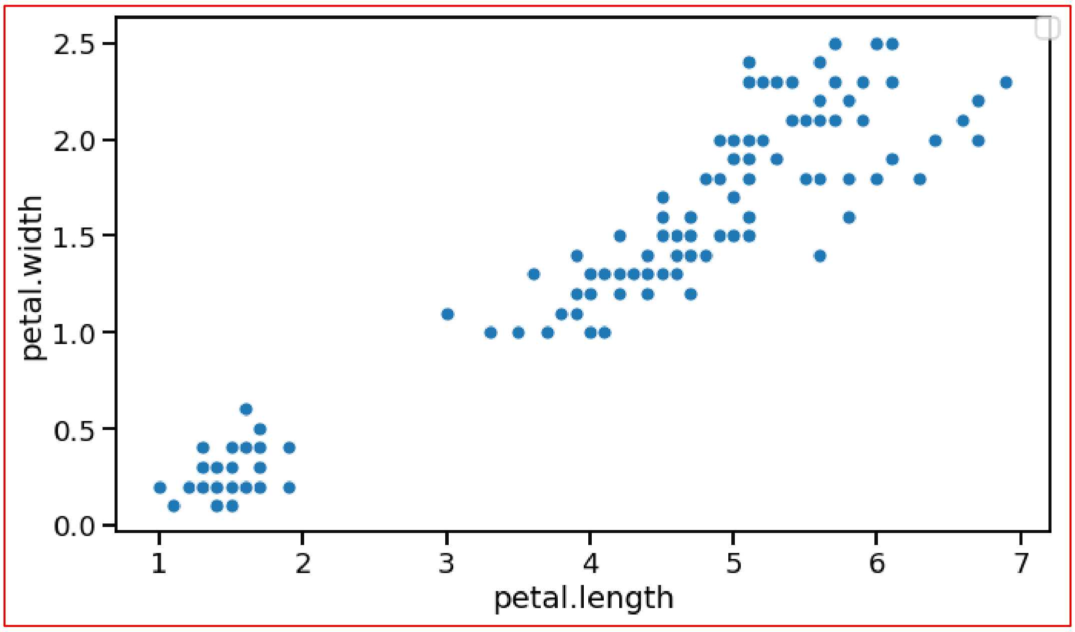


Bubble plot with Seaborn scatterplot():

To make bubble plot in *Seaborn*, we are able to use *scatterplot()* function in *Seaborn* with a variable specifying *size* argument in addition to x and y-axis variables for scatter plot.
In this bubble plot instance, we have *length= "body_mass_g"*.

```
# import all important libraries
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# load dataset
data =
"https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv"

# convert to dataframe
df = pd.read_csv(data)

# display top most rows
df.head()

# depict scatter plot illustration
sns.set_context("talk", font_scale=1.1)
plt.figure(figsize=(10, 6))
sns.scatterplot(x="petal.length",
                y="petal.width",
                data=df)
# Put the legend out of the figure
plt.legend(bbox_to_anchor=(1.01, 1), borderaxespad=0)
plt.xlabel("petal.length")
plt.ylabel("petal.width")
plt.tight_layout()
plt.savefig("Bubble_plot_Seaborn_scatterplot.png",
          format='png', dpi=150)
```

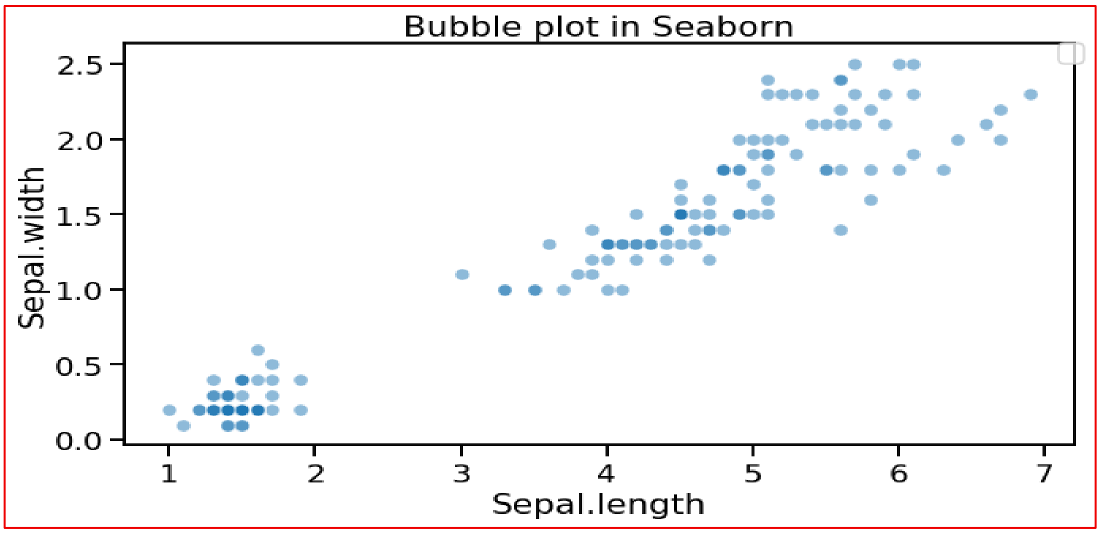
Output:



The below example depicts a bubble plot having colored bubbles:

```
# import all important libraries
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# load dataset
data=
"https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv"

# convert to dataframe
df = pd.read_csv(data)

# display top most rows
df.head()

# depict bubble plot illustration
sns.set_context("talk", font_scale=1.2)
plt.figure(figsize=(10,6))
sns.scatterplot(x='petal.length',
               y='petal.width',
               sizes=(20,500),
               alpha=0.5,
               data= df)
# Put the legend out of the figure
plt.legend(bbox_to_anchor=(1.01, 1),borderaxespad=0)

# assign labels
plt.xlabel("Sepal.length")
plt.ylabel("Sepal.width")
```

```
# assign title
plt.title("Bubble plot in Seaborn")
# adjust layout
plt.tight_layout()
```
Output:

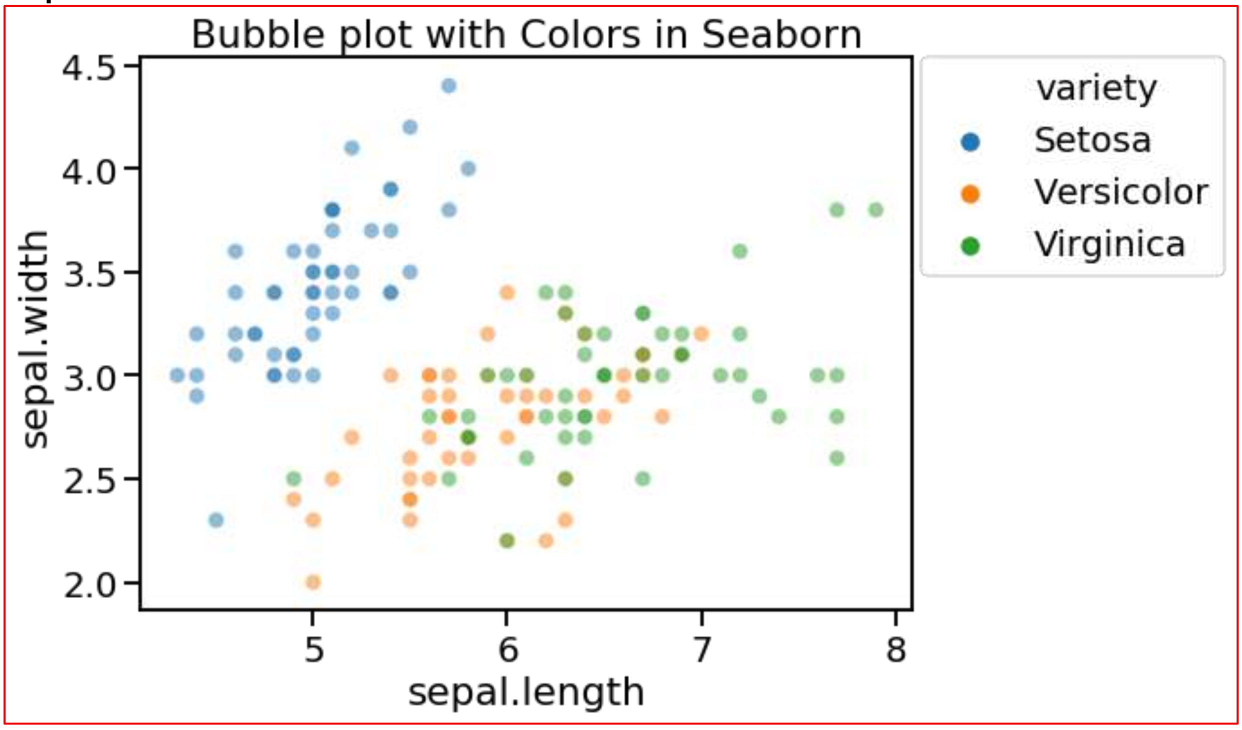


Bubble plot with explicit size ranges Seaborn scatterplot()

We can alter the air bubble plot made with *Seaborn* without any problem. Something that we notice from the bubble plot above is that the bubble size range is by all accounts little. It will be extraordinary in the event that we could differ the littlest and biggest bubble sizes.
With the contention *sizes* in *Seaborn's scatterplot()* work, we can indicate ranges for the bubble sizes. In this air pocket plot model underneath, we utilized *sizes=(20,500).*

```
# import all important libraries
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# load dataset
data =
"https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv"

# convert to dataframe
df = pd.read_csv(data)

# display top most rows
df.head()

# depict bubble plot illustration
sns.set_context("talk", font_scale=1.2)
plt.figure(figsize=(10, 6))
sns.scatterplot(x='sepal.length',
            y='sepal.width',
            # size="body_mass_g",
            sizes=(20, 500),
            alpha=0.5,
            hue='variety',
            data=df)

# Put the legend out of the figure
plt.legend(bbox_to_anchor=(1.01, 1), borderaxespad=0)

# Put the legend out of the figure
plt.xlabel("sepal.length")
plt.ylabel("sepal.width")
plt.title("Bubble plot with Colors in Seaborn")
plt.tight_layout()
```

Output:



Python Seaborn - Catplot
Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn helps resolve the two major problems faced by Matplotlib; the problems are?

- Default Matplotlib parameters
- Working with data frames

As Seaborn compliments and extends Matplotlib, the learning curve is quite gradual. If you know Matplotlib, you are already half-way through Seaborn. Seaborn library offers many advantages over other plotting libraries:

- It is very easy to use and requires less code syntax
- Works really well with `pandas` data structures, which is just what you need as a data scientist.
- It is built on top of Matplotlib, another vast and deep data visualization library.

Syntax: *seaborn.catplot(*, x=None, y=None, hue=None, data=None, row=None, col=None, kind='strip', color=None, palette=None, **kwargs)*

Parameters

- ***x, y, hue:*** *names of variables in data*
  Inputs for plotting long-form data. See examples for interpretation.

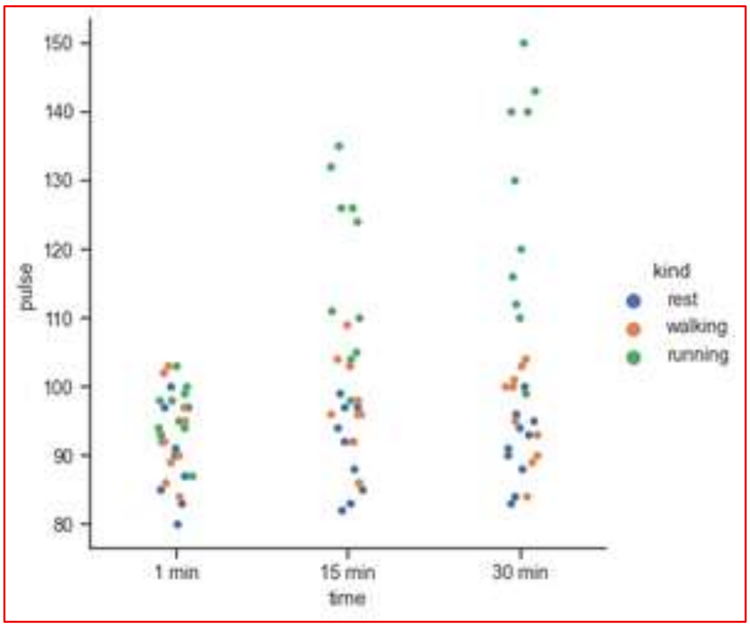
- ***data:*** *DataFrame*
  Long-form (tidy) dataset for plotting. Each column should correspond to a variable, and each row should correspond to an observation.
- ***row, col:*** *names of variables in data, optional*
  Categorical variables that will determine the faceting of the grid.
- ***kind:*** *str, optional*
  The kind of plot to draw, corresponds to the name of a categorical axes-level plotting function. Options are: "strip", "swarm", "box", "violin", "boxen", "point", "bar", or "count".
- ***color:*** *matplotlib color, optional*
  Color for all of the elements, or seed for a gradient palette.
- ***palette:*** *palette name, list, or dict*
  Colors to use for the different levels of the hue variable. Should be something that can be interpreted by color_palette(), or a dictionary mapping hue levels to matplotlib colors.
- ***kwargs:*** *key, value pairings*
  Other keyword arguments are passed through to the underlying plotting function.

Examples:

If you are working with data that involves any categorical variables like survey responses, your best tools to visualize and compare different features of your data would be categorical plots. Plotting categorical plots it is very easy in seaborn. In this example x,y and hue take the names of the features in your data. Hue parameters encode the points with different colors with respect to the target variable.

```
import seaborn as sns

exercise = sns.load_dataset("exercise")
g = sns.catplot(x="time", y="pulse",
                hue="kind",
                data=exercise)
```

Output:

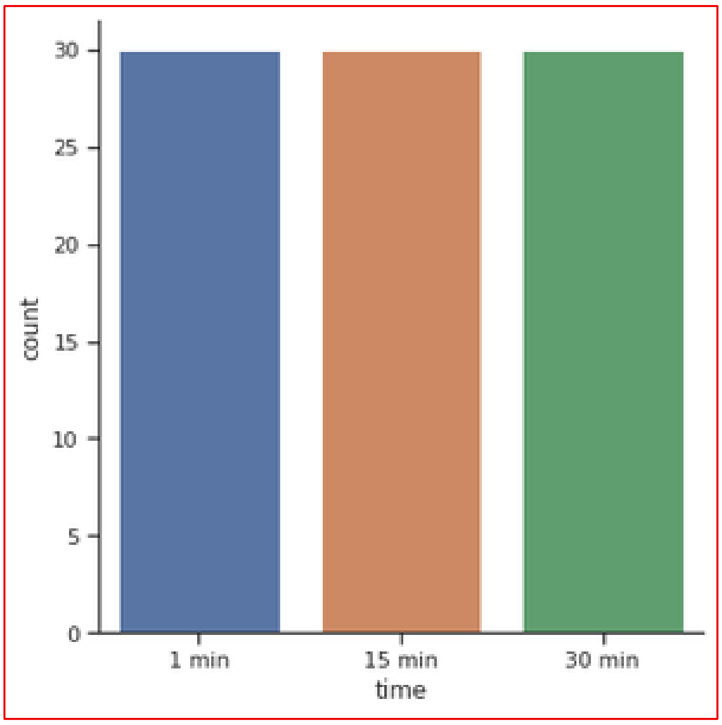


For the count plot, we set a kind parameter to count and feed in the data using data parameters. Let's start by exploring the time feature. We start off with catplot() function and use x argument to specify the axis we want to show the categories.

```
import seaborn as sns

sns.set_theme(style="ticks")
exercise = sns.load_dataset("exercise")
```

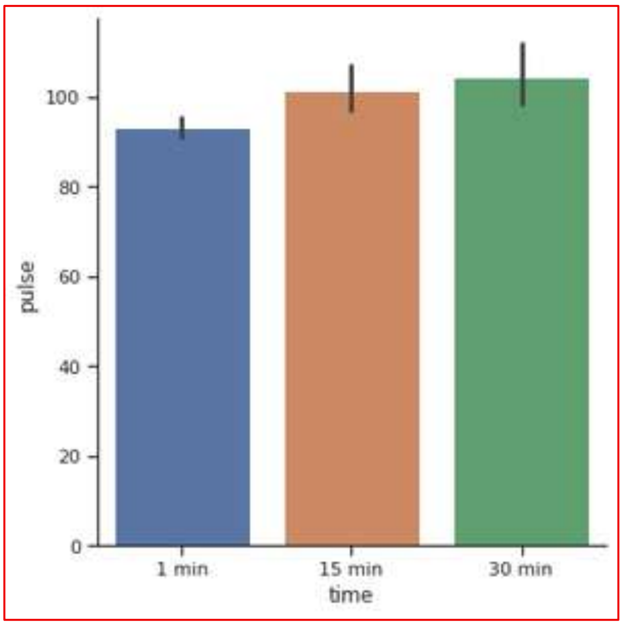
```
g = sns.catplot(x="time",
            kind="count",
            data=exercise)
```

Output:



```
import seaborn as sns

exercise = sns.load_dataset("exercise")
g = sns.catplot(x="time",
            y="pulse",
            kind="bar",
            data=exercise)
```

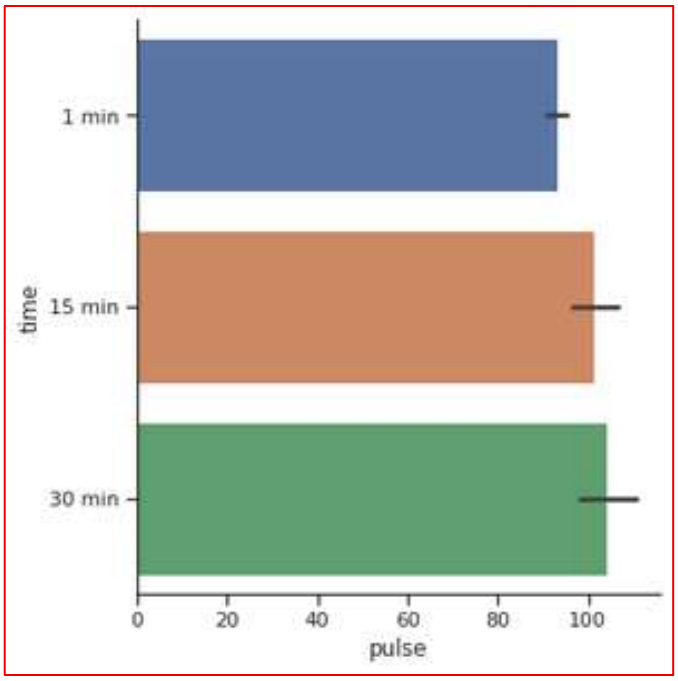
Output:



For creating the horizontal bar plot we have to change the x and y features. When you have lots of categories or long category names it's a good idea to change the orientation.

```
import seaborn as sns

exercise = sns.load_dataset("exercise")
g = sns.catplot(x="pulse",
            y="time",
            kind="bar",
            data=exercise)
```

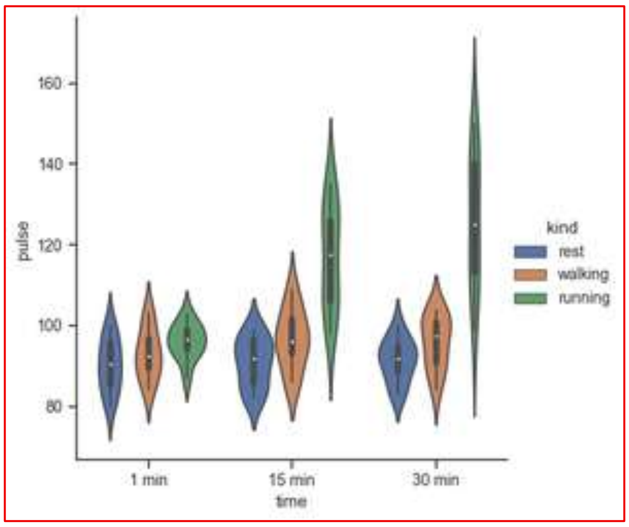
Output:



Use a different plot kind to visualize the same data:

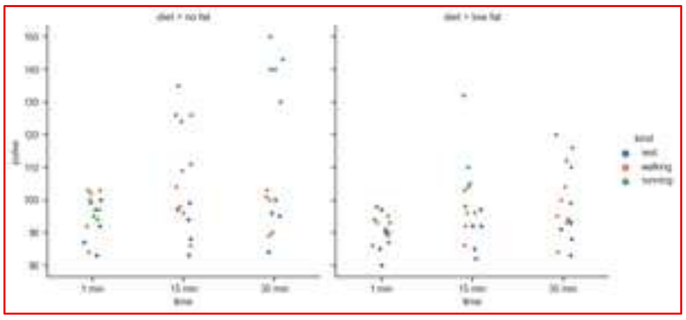
```
import seaborn as sns


exercise = sns.load_dataset("exercise")

g = sns.catplot(x="time",
            y="pulse",
            hue="kind",
            data=exercise,
            kind="violin")
```

Output:



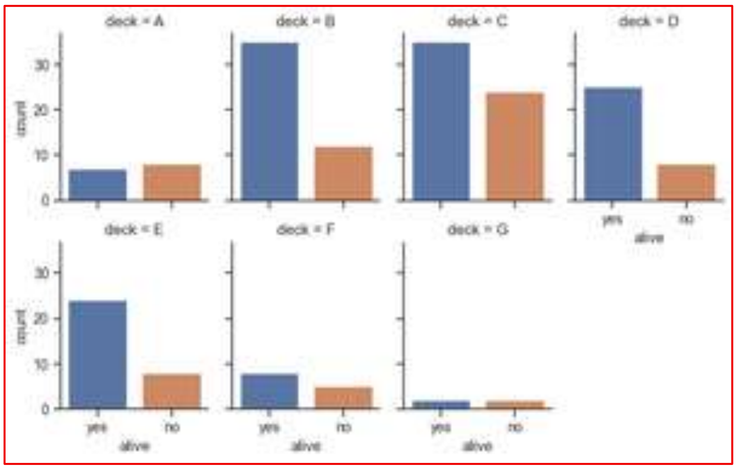
```
import seaborn as sns

exercise = sns.load_dataset("exercise")

g = sns.catplot(x="time",
            y="pulse",
            hue="kind",
            col="diet",
            data=exercise)
```

Output:



```
titanic = sns.load_dataset("titanic")
g = sns.catplot(x="alive", col="deck", col_wrap=4,
            data=titanic[titanic.deck.notnull()],
            kind="count", height=2.5, aspect=.8)
```

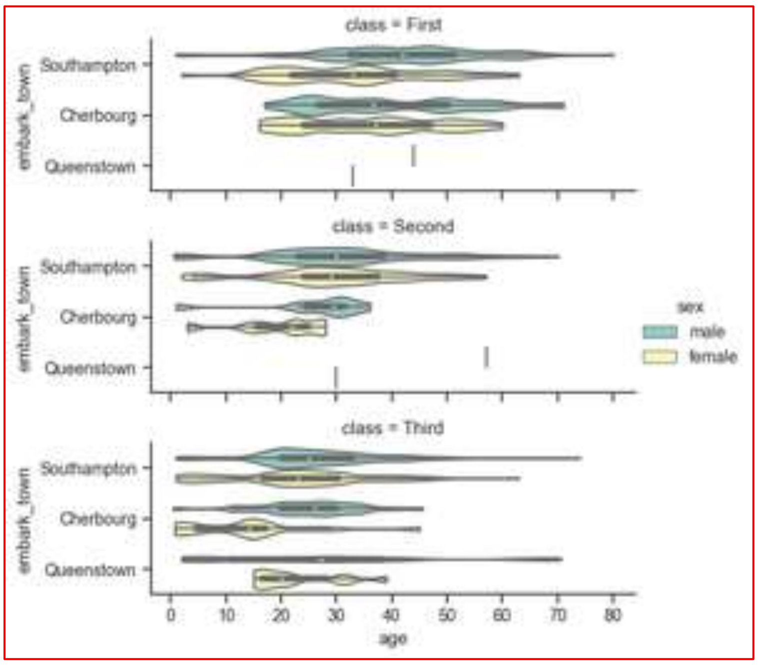
Output:



Plot horizontally and pass other keyword arguments to the plot function:

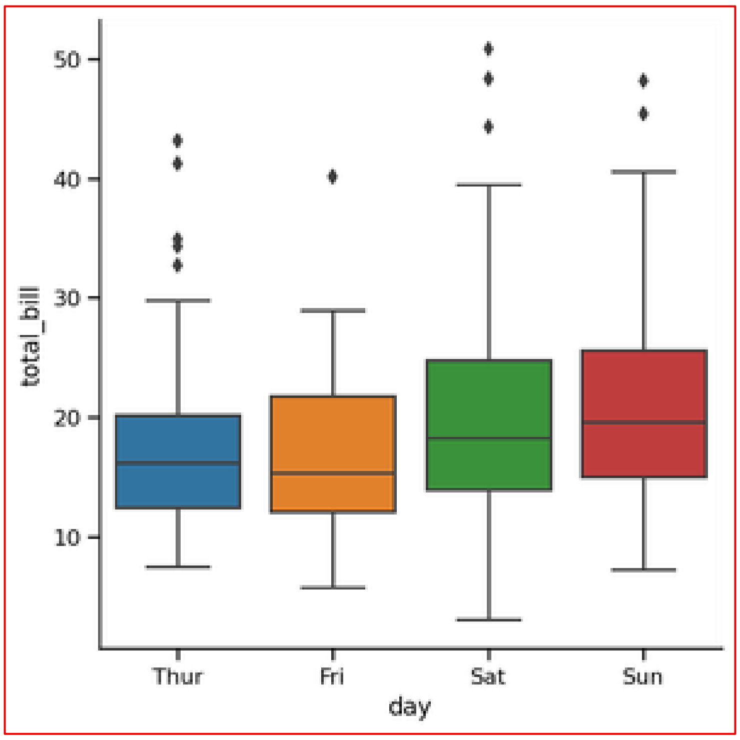
```
g = sns.catplot(x="age", y="embark_town",
            hue="sex", row="class",
            data=titanic[titanic.embark_town.notnull()],
            orient="h", height=2, aspect=3, palette="Set3",
            kind="violin", dodge=True, cut=0, bw=.2)
```

Output:



```
tips = sns.load_dataset('tips')
sns.catplot(x='day',
          y='total_bill',
          data=tips,
          kind='box');
```

Output:


total_bill
Thur
Fri
Sat
Sun
day