```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

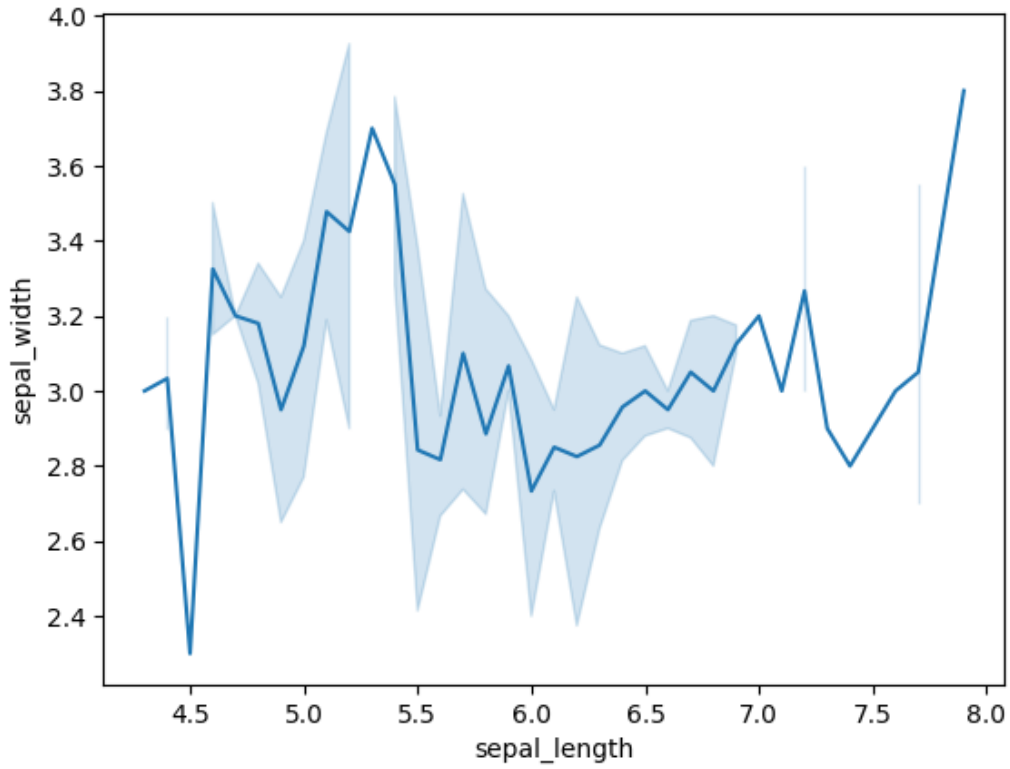
Plotting Chart Using seaborn Library

> **Line plot:**

- The seaborn line plot is one of the most basic plots presents in the seaborn library.
- We use the seaborn line plot mainly to visualize the given data in some time-series form, i.e., in a continuous manner with respect to time.

```
# 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)
```

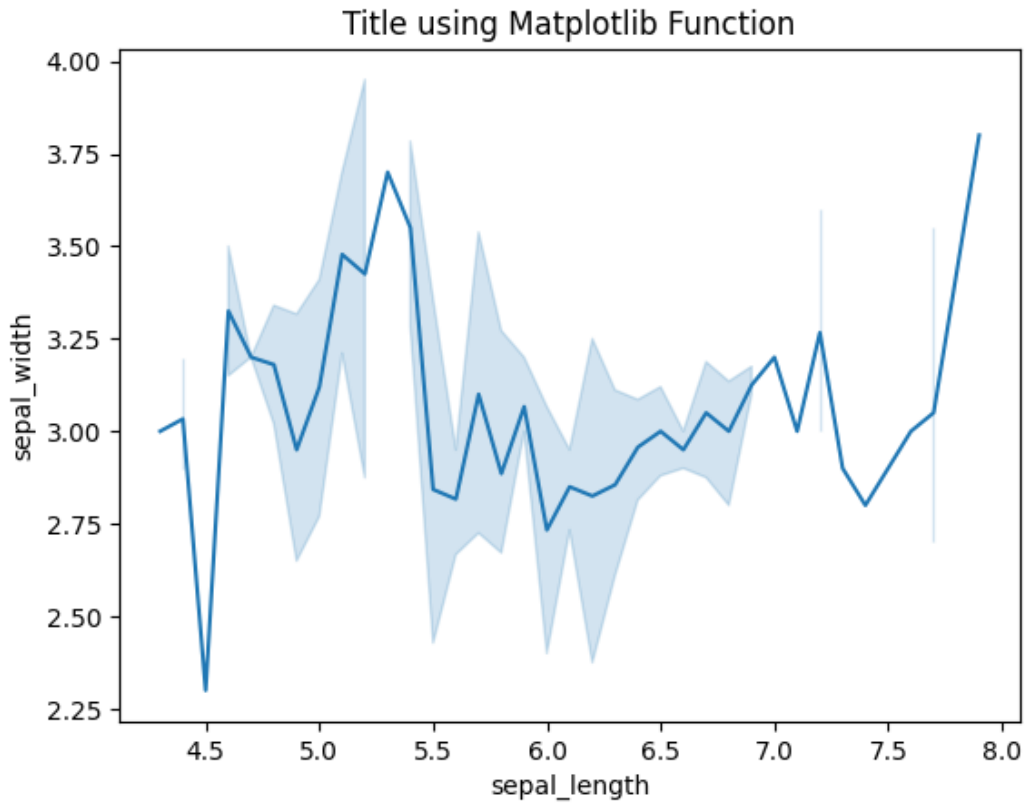
```
<Axes: xlabel='sepal_length', ylabel='sepal_width'>
```



> **Using Seaborn with Matplotlib:**

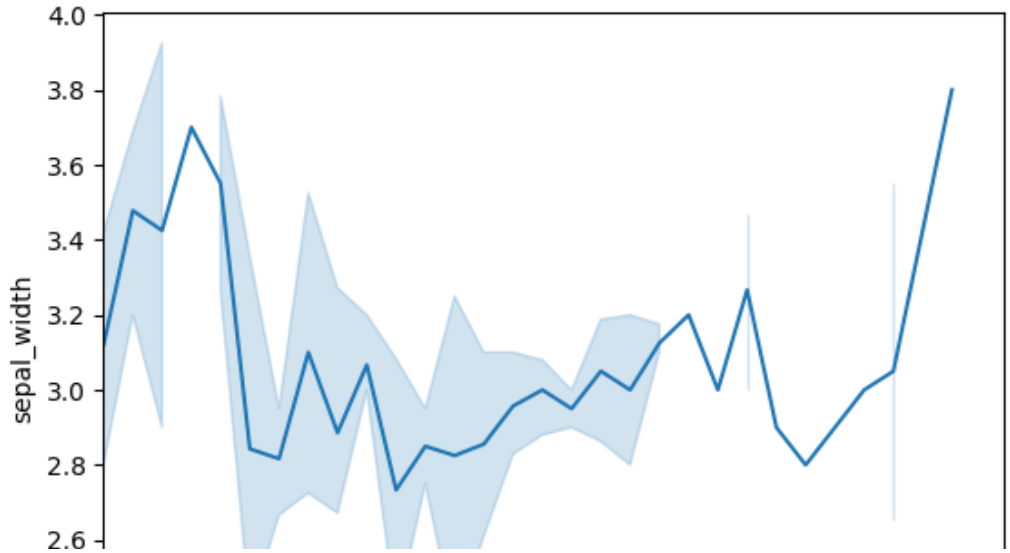
- Just invoke the Seaborn Plotting function as normal, and then use Matplotlib's customization function.

```
# 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()
```



> **Setting the xlim and ylim:**

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

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

plt.show()
```



Customizing Seaborn Plots

5.0 5.5 6.0 6.5 7.0 7.5 8.0

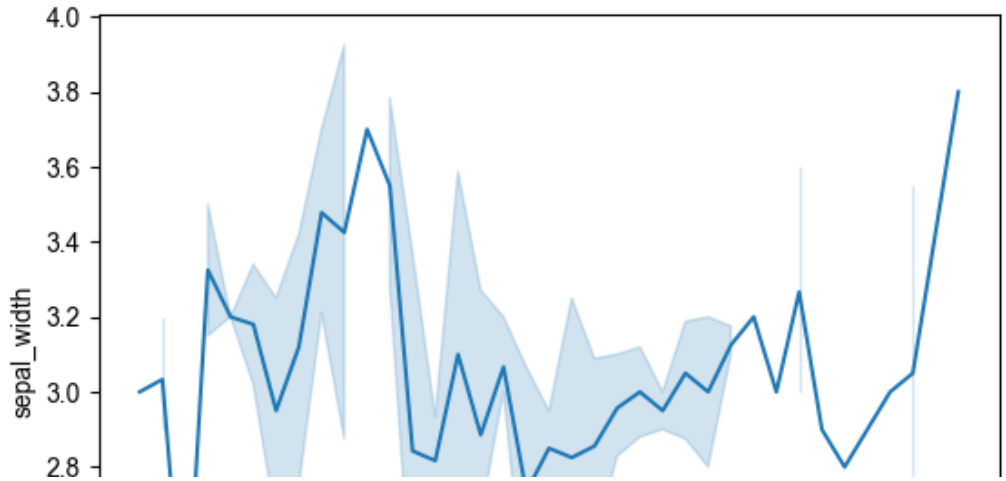
> **Changing Figure Aesthetic:**

- **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)
```

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

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



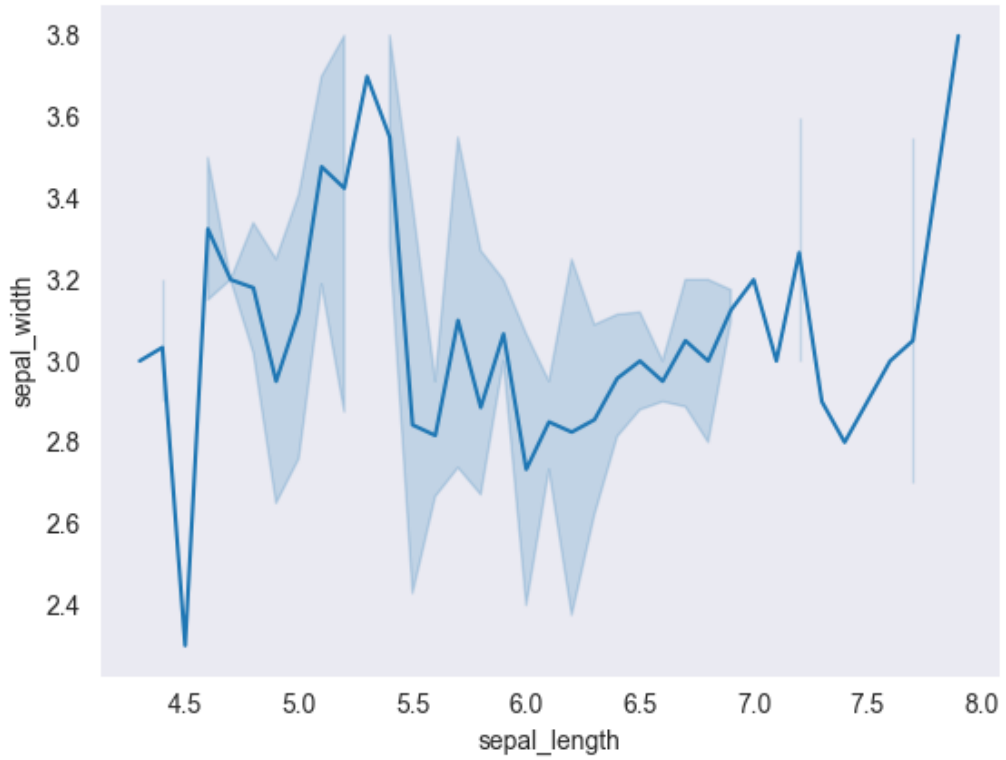
Removal of Spines:

- 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)
```

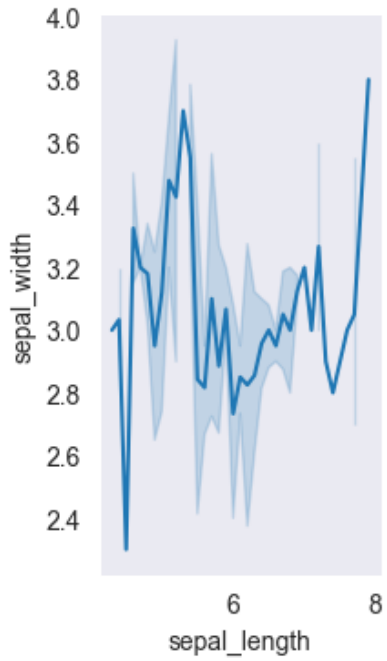
```
# draw lineplot
sns.lineplot(x="sepal_length", y="sepal_width", data=data)

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



> **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.

```
# 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()
```



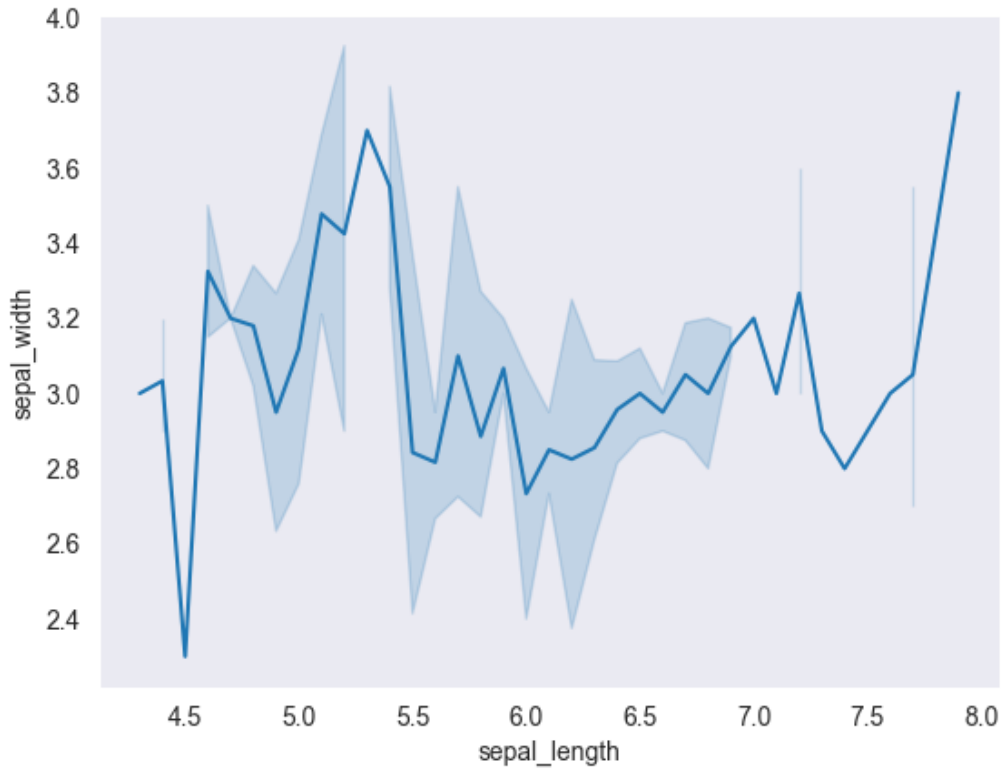
> **Scaling the plots:**

- 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)
```

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

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

plt.show()
```



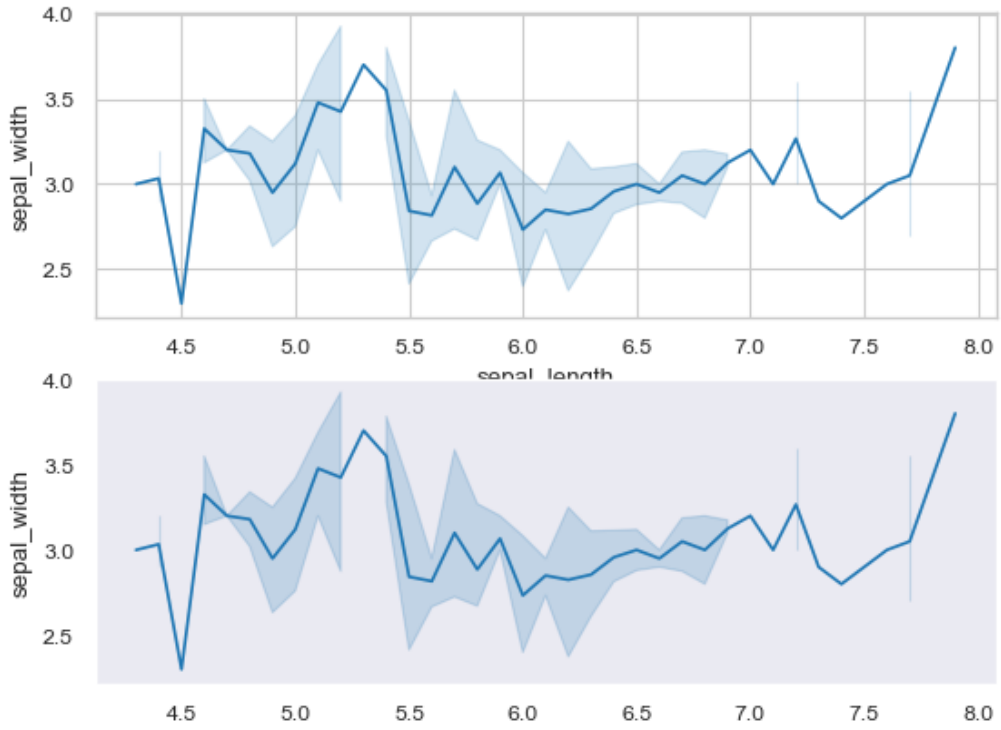
> **Setting the Style Temporarily:**

- **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)
```

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

with sns.axes_style('whitegrid'):

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

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



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.

```
# current colot palette
palette = sns.color_palette()

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

plt.show()
```

> **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.

```
# current colot palette
palette = sns.color_palette('PiYG', 15)

# diverging color palette
sns.palplot(palette)

plt.show()
```

> **Sequential Color Palette:**

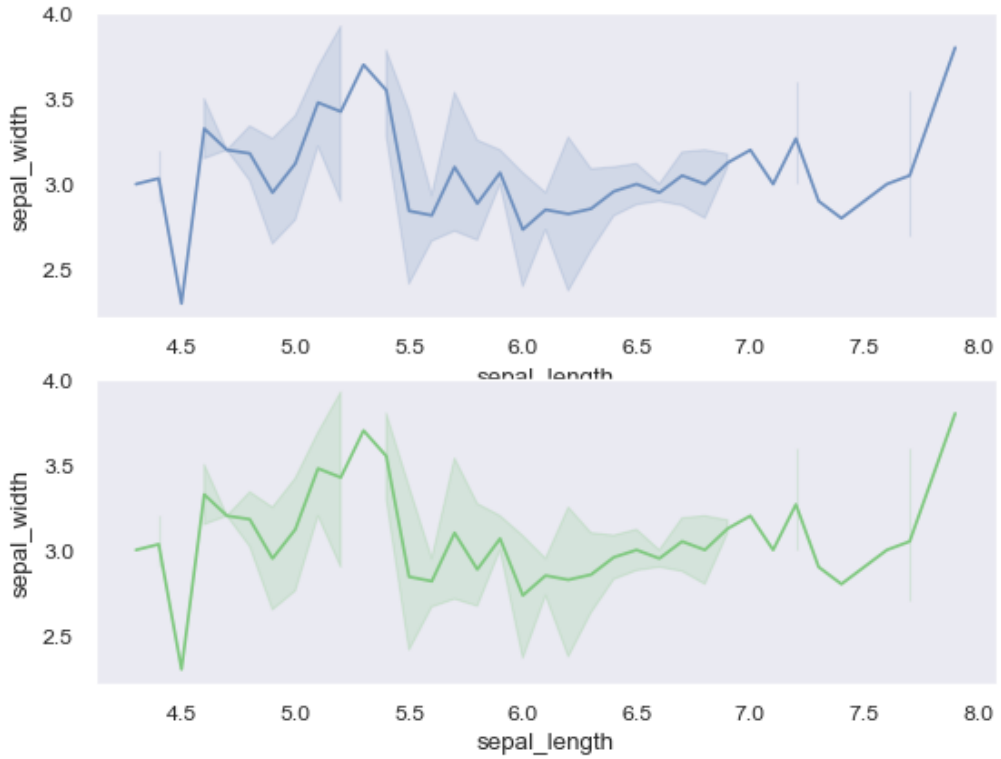
- A sequential palette is used where the distribution ranges from a lower value to a higher value.
- To do this add the character 's' to the color passed in the color palette.

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

# sequential color palette
sns.palplot(palette)

plt.show()
```

> **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.

```
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()
```

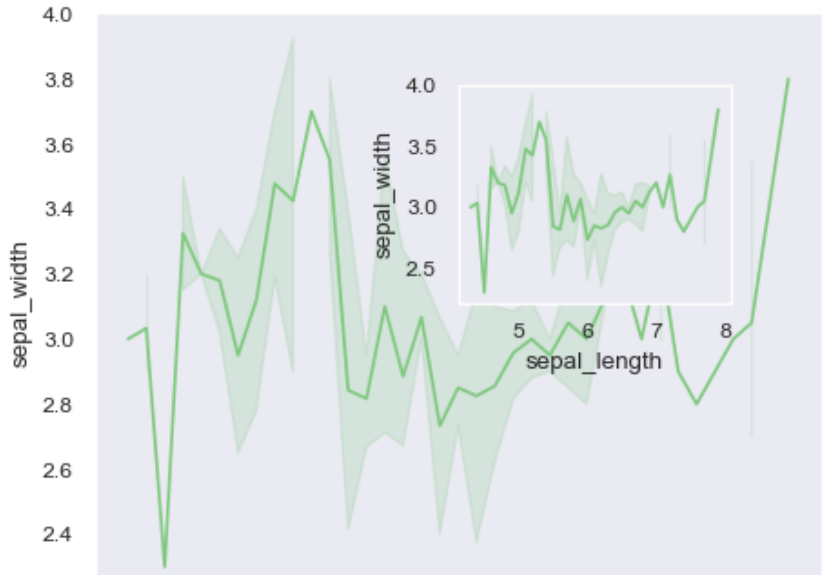


Multiple plots with Seaborn

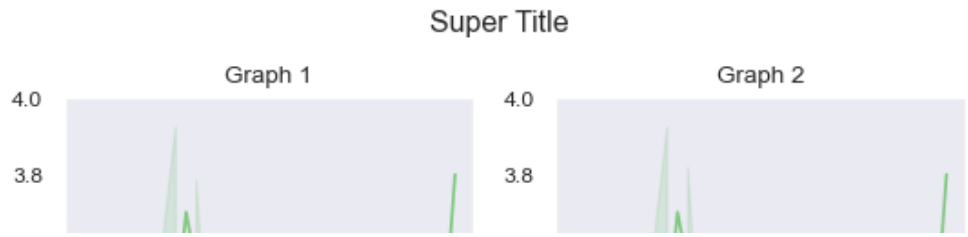
> **Using Matplotlib:**

- Matplotlib provides various functions for plotting subplots. Some of them are **add_axes()**, **subplot()** and **subplot2grid()**.

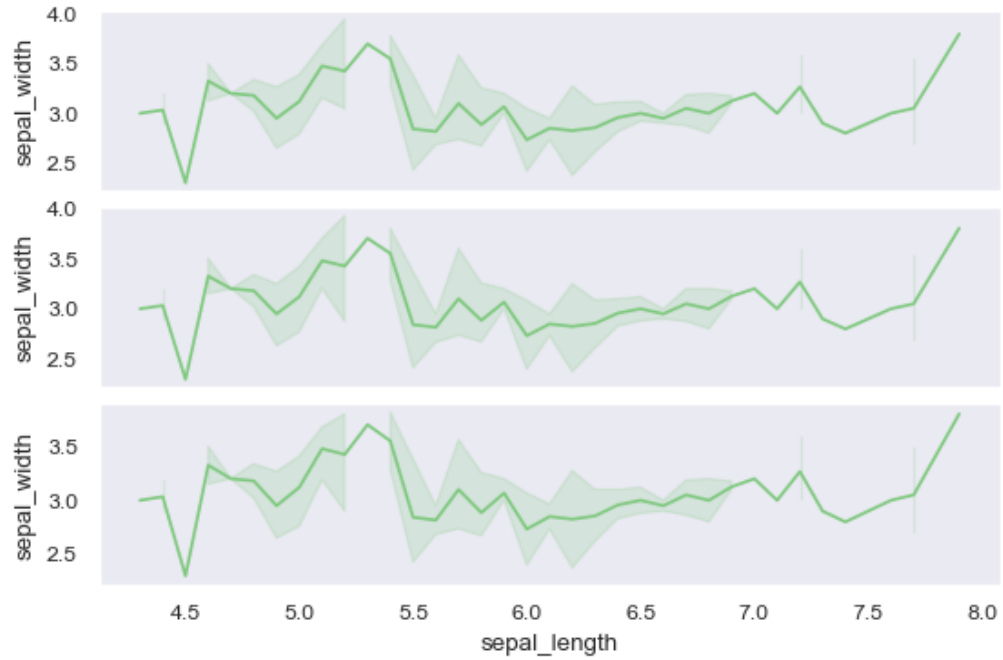
```
# 1: Using add_axes() method

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()
```




```
# 2: Using subplot() method

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

# Adding the subplot at the specified
# grid position
plt.subplot(121)
plt.title('Graph 1')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
graph()

# Adding the subplot at the specified
# grid position
plt.subplot(122)
plt.title('Graph 2')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
graph()

plt.suptitle('Super Title')
plt.show()
```




```
# 3: Using subplot2grid() method

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()
```



> **Using Seaborn :**

- **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 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)
```

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

plt.show()
```

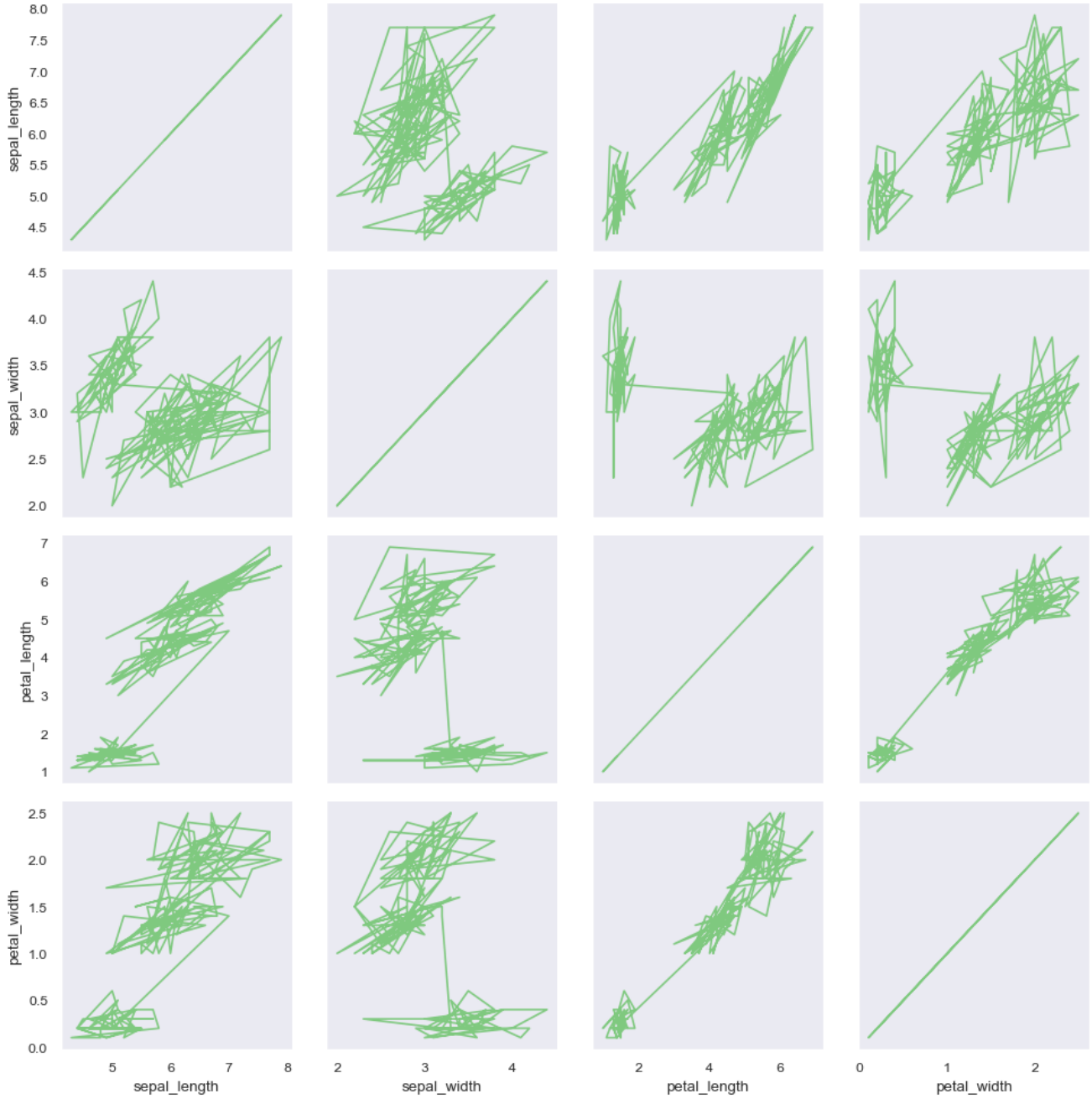
```
c:\Users\ABC\Desktop\Notes\Online Notes PDF\Python\pandas\virtual_env\Lib\site-packages\seabor
  self._figure.tight_layout(*args, **kwargs)
```

- **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)
```

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

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

plt.show()
```



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**.

There are different types of Relational Plots:

1. relplot()

2. Scatter Plot

3. Line Plot

> **1. Relplot():**

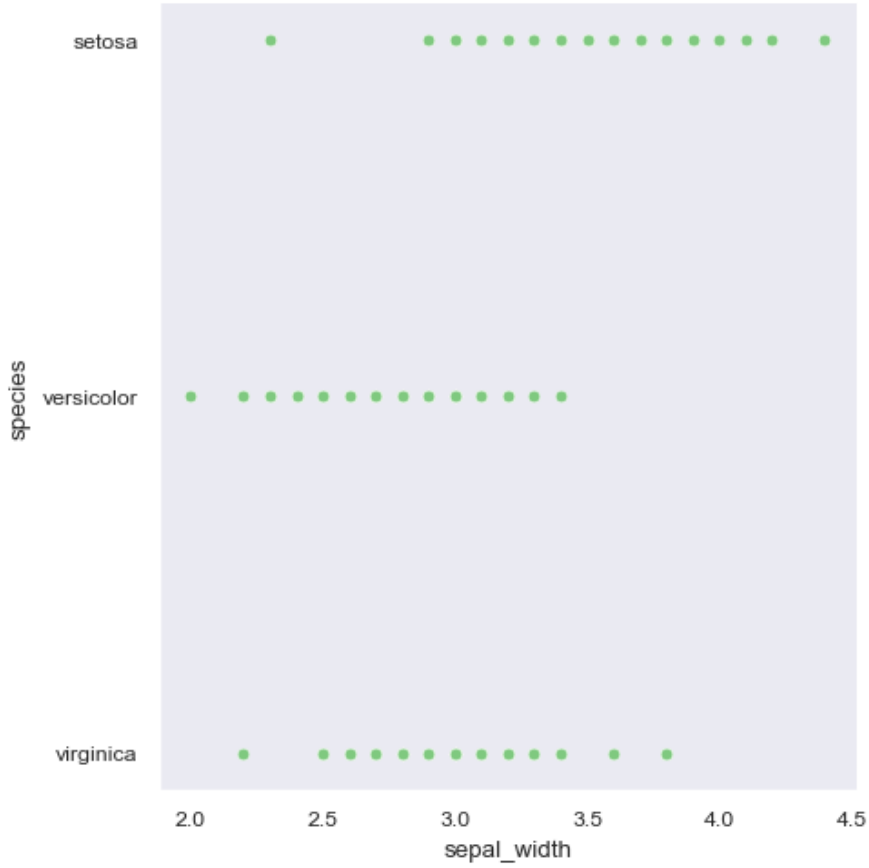
- This function provides us the access to some other different axes-level functions which shows the relationships between two variables with semantic mappings of subsets.

- **Syntax:**

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

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

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

plt.show()
```

```
c:\Users\ABC\Desktop\Notes\Online Notes PDF\Python\pandas\virtual_env\Lib\site-packages\seabor
  self._figure.tight_layout(*args, **kwargs)
```



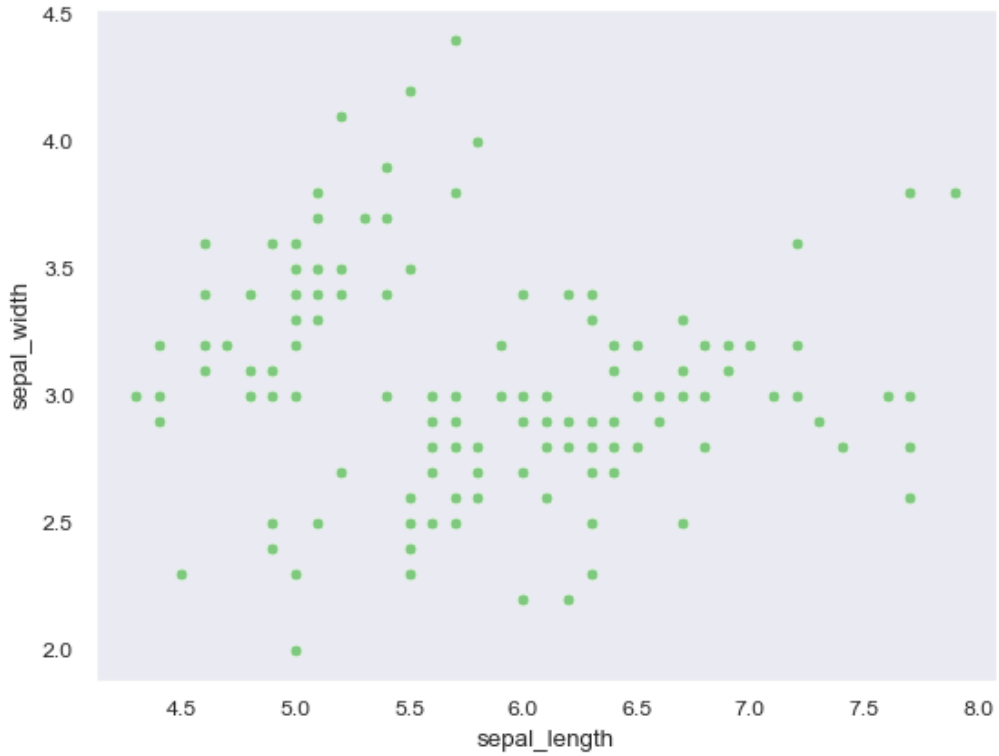
> **2. Scatter Plot:**

- 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)
```

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

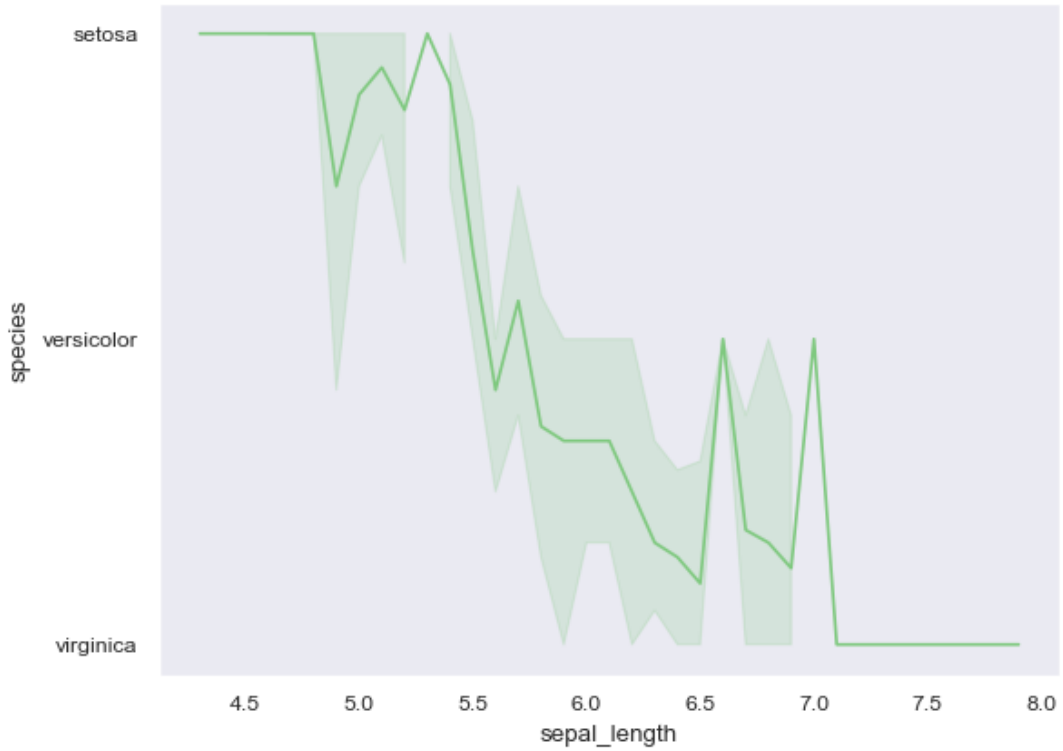


3. Line Plot:

- 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)
```

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



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:
    1. lmplot()
    2. regplot()
- They are closely related to each other.
- They even share their core functionality.

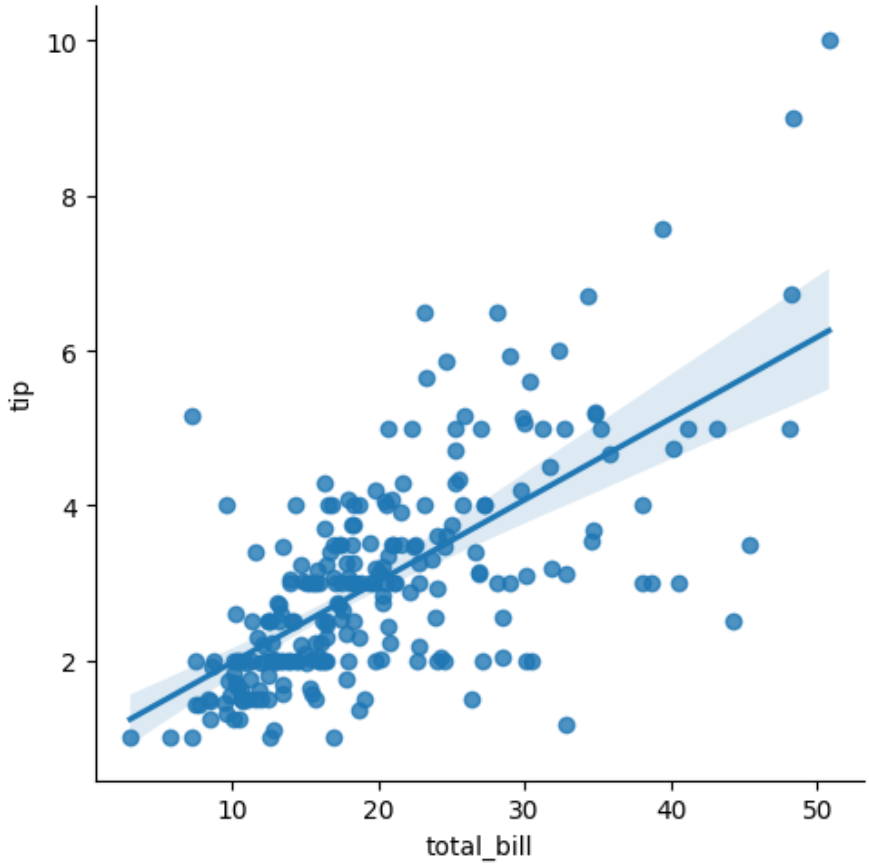
> **1. lmplot:**

- **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)
```

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

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

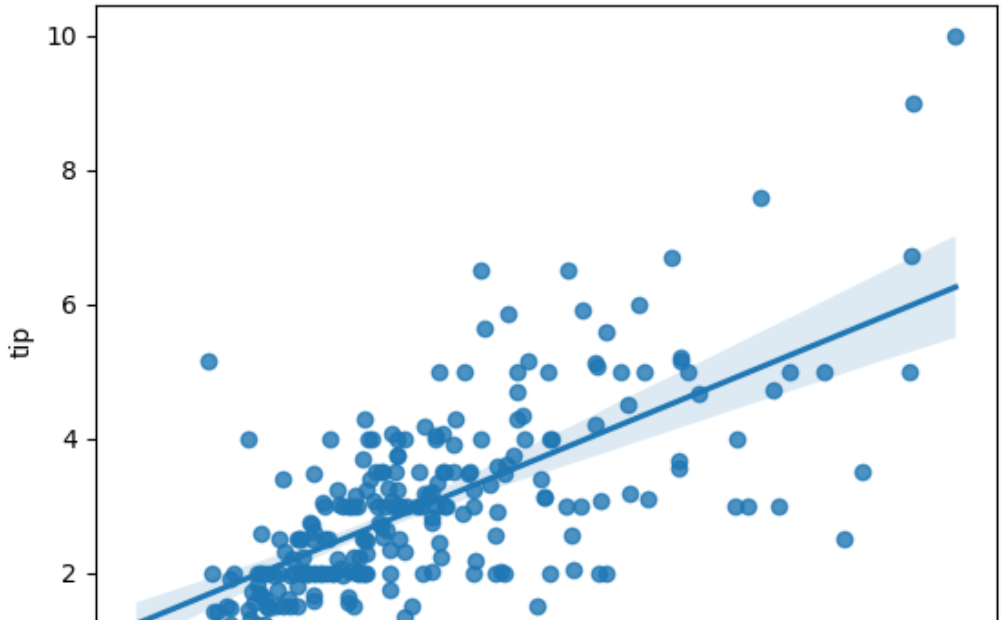


> **2. Regplot:**

- **regplot()** method is also similar to lmplot which creates **linear regression** model.
- **Syntax:**

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

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

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



Note: The difference between both the function is that **regplot** accepts the x, y variables in different format including NumPy arrays, Pandas objects, whereas, the **lmplot** only accepts the value as strings.

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**.

> **1. 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.

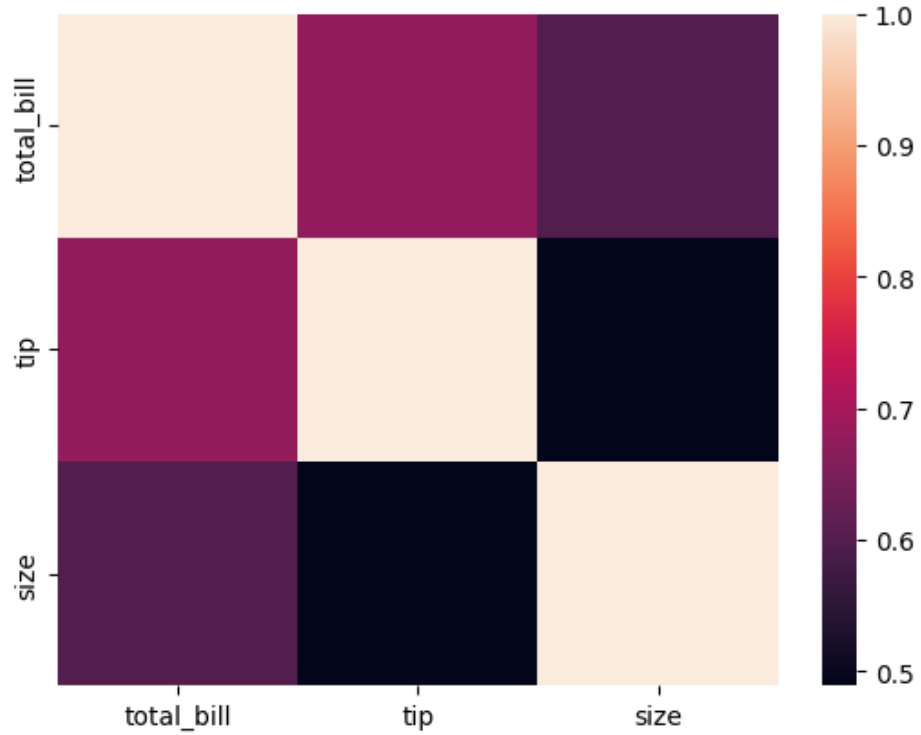
- **Syntax:**

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

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

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

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

```
<ipython-input-4-52e849d3dd14>:5: FutureWarning: The default value of numeric_only in DataFram
  tc = data.corr()
```



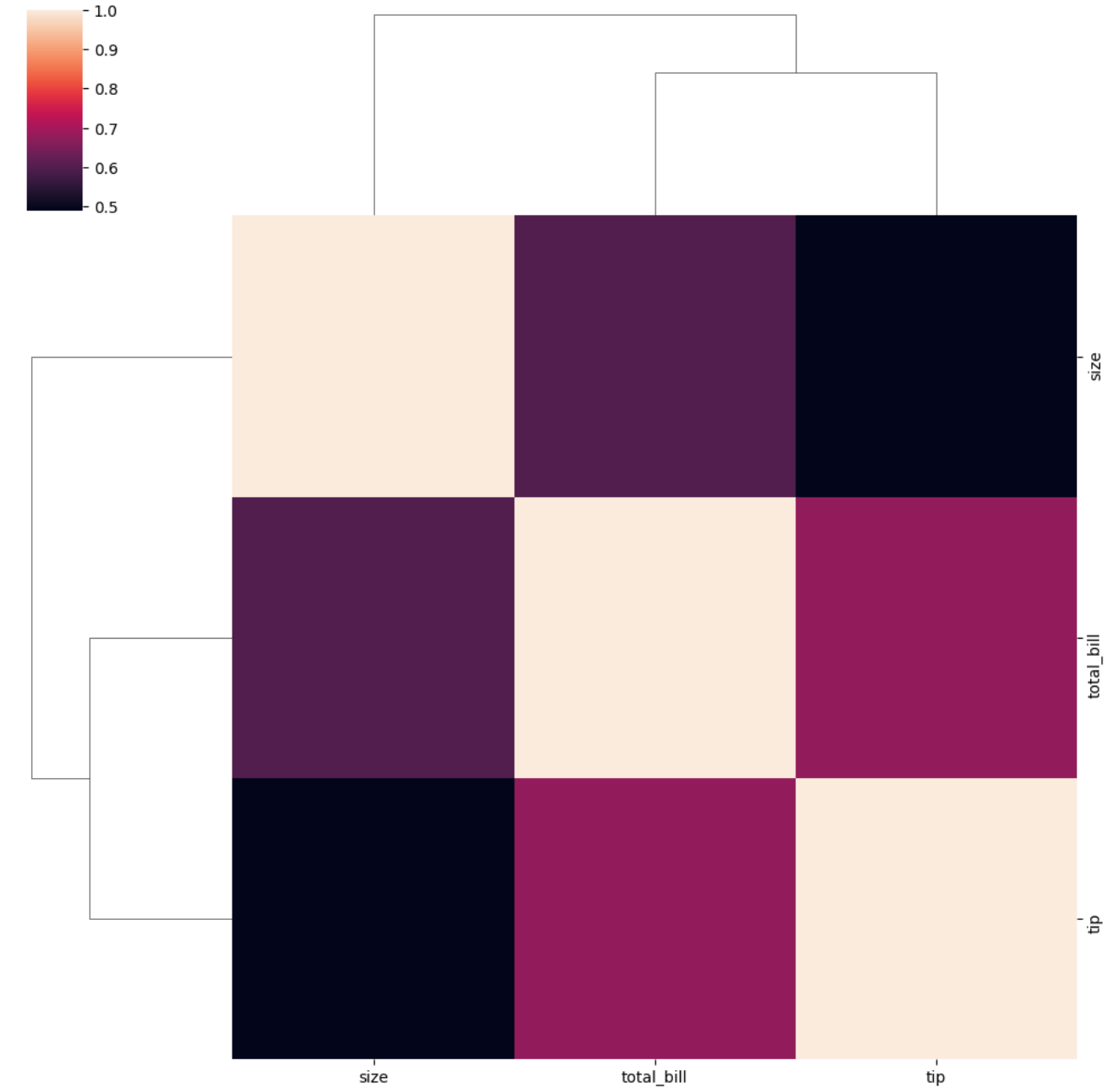
2. Clustermap:

- 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)
```

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

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

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

```
<ipython-input-7-d94e1cee0d71>:6: FutureWarning: The default value of numeric_only in DataFram
  tc = data.corr()
```



✓ 1s completed at 20:30