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.

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

```
data = sns.load_dataset("iris")
```

```
data.head()
```

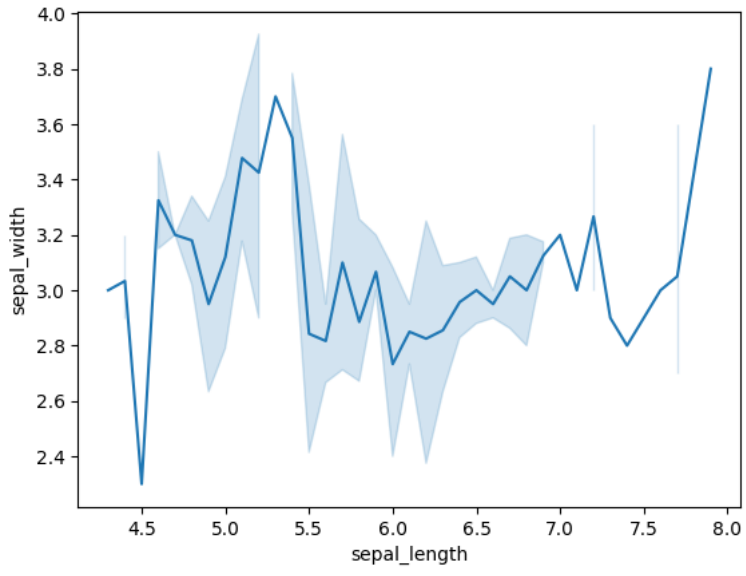
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
# Datasets come with Seaborn
sns.get_dataset_names()
```

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

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

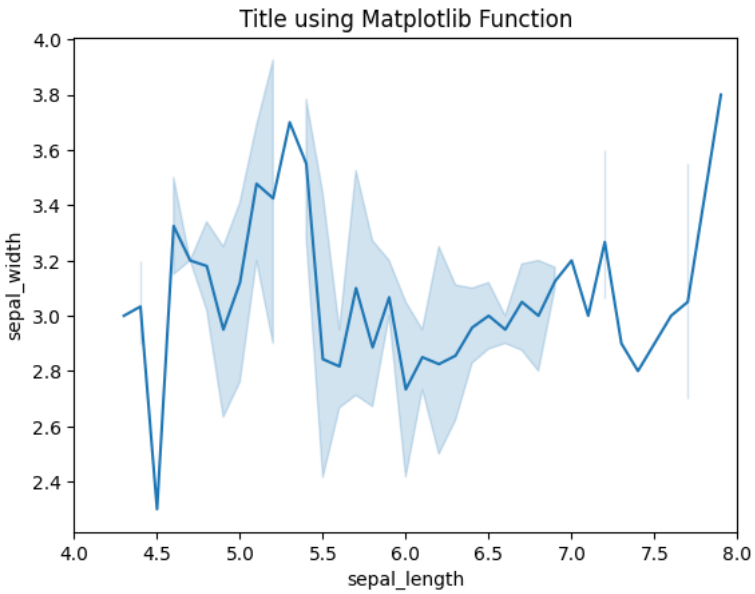
```
<Axes: xlabel='sepal_length', ylabel='sepal_width'>
```



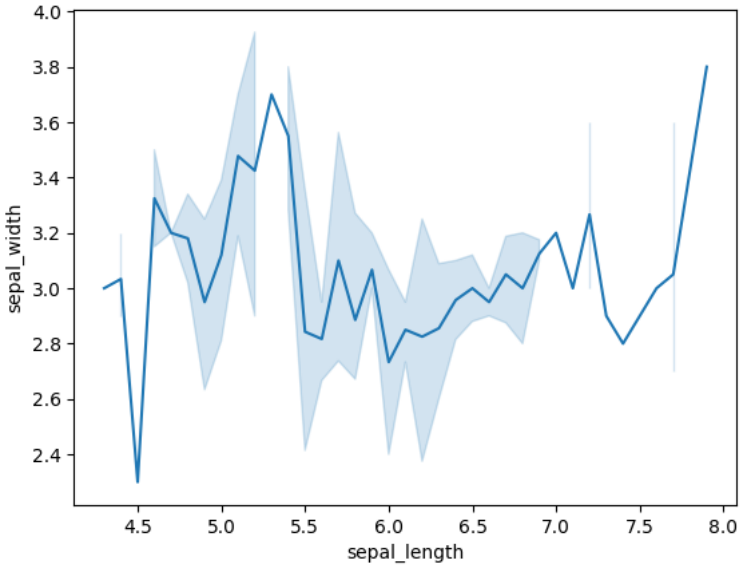
Using Seaborn with Matplotlib

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.

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

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

plt.xlim(4, 8)

plt.show()
```




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

# changing the theme to dark (darkgrid/whitegrid/dark/white/ticks)
sns.set_style("darkgrid")
plt.show()
```




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

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


```
# Changing the figure Size
# changing the figure size
plt.figure(figsize = (10, 6))

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

# Removing the spines
sns.despine()

plt.show()


# Scaling the plots (paper/talk/poster)
# draw lineplot
sns.lineplot(x="sepal_length", y="sepal_width", data=data)

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

plt.show()
```

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.

Components of Colors

Because of the way our eyes work, a particular color can be defined using three components. We usually program colors in a computer by specifying their RGB values, which set the intensity of the red, green, and blue channels in a display. But for analyzing the perceptual attributes of a color, it's better to think in terms of hue, saturation, and luminance channels.

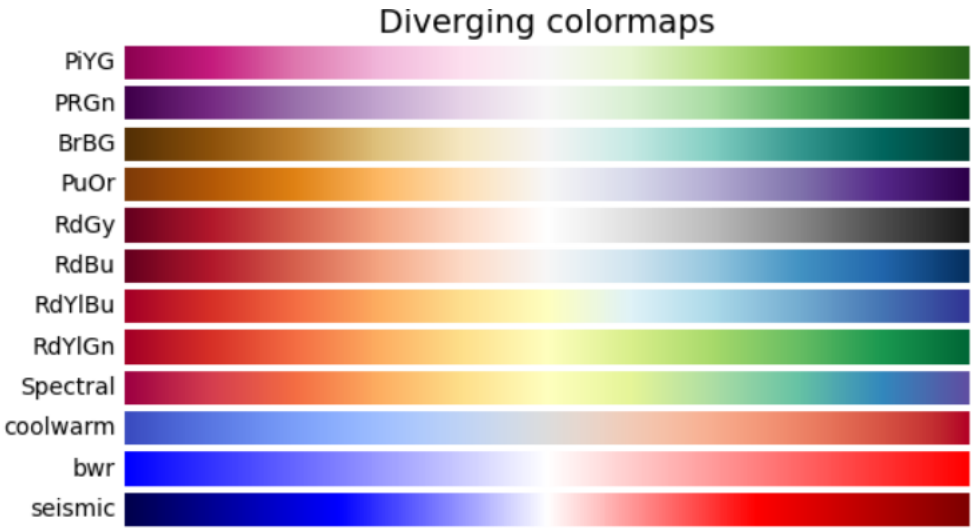
Hue is the component that distinguishes "different colors" in a non-technical sense. It's property of color that leads to first-order names like "red" and "blue":

Saturation (or chroma) is the *colorfulness*. Two colors with different hues will look more distinct when they have more saturation:

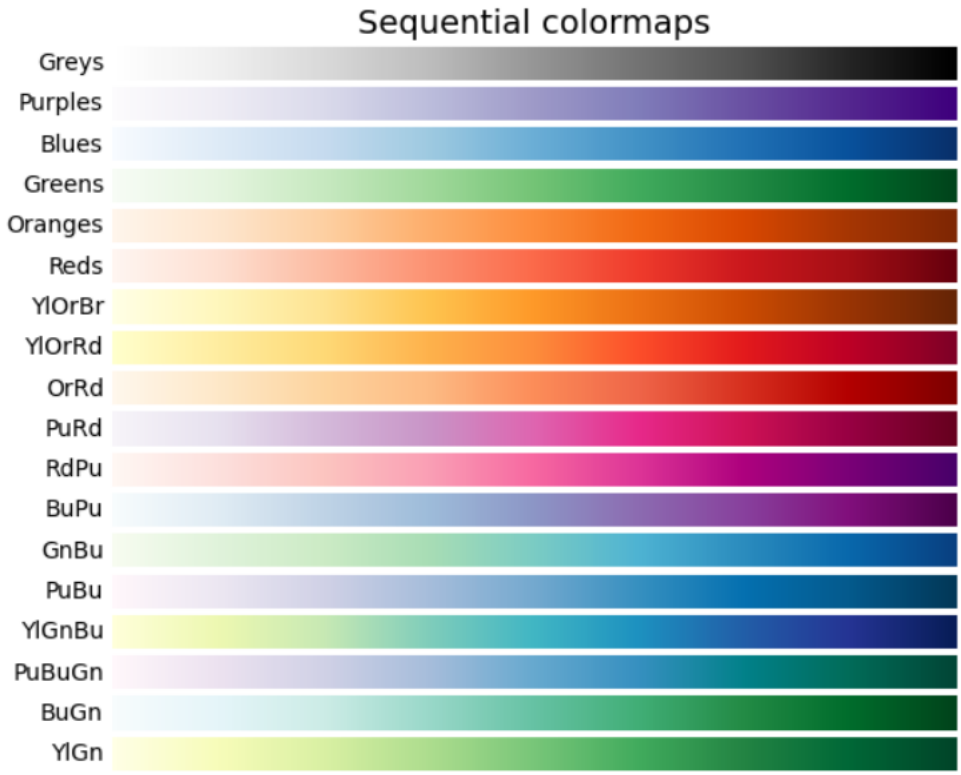
And lightness corresponds to how much light is emitted (or reflected, for printed colors), ranging from black to white:

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

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

plt.show()
```

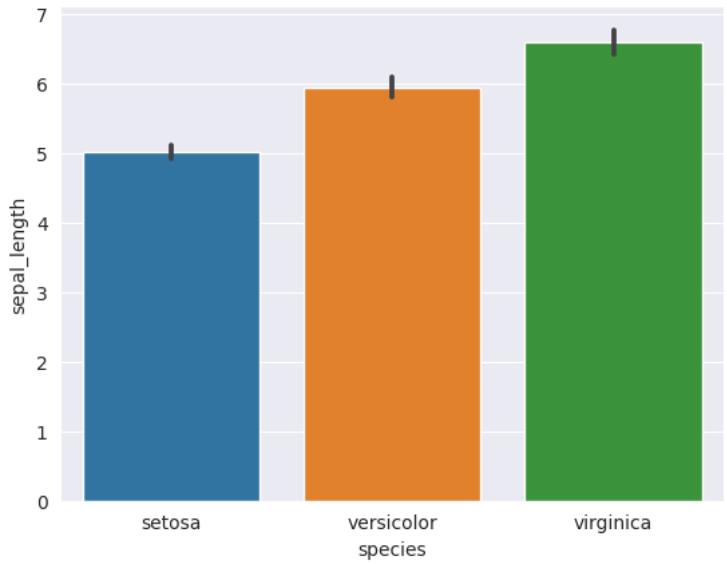



```
# Diverging Color Palette

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

# diverging color palette
sns.palplot(palette)

plt.show()
```



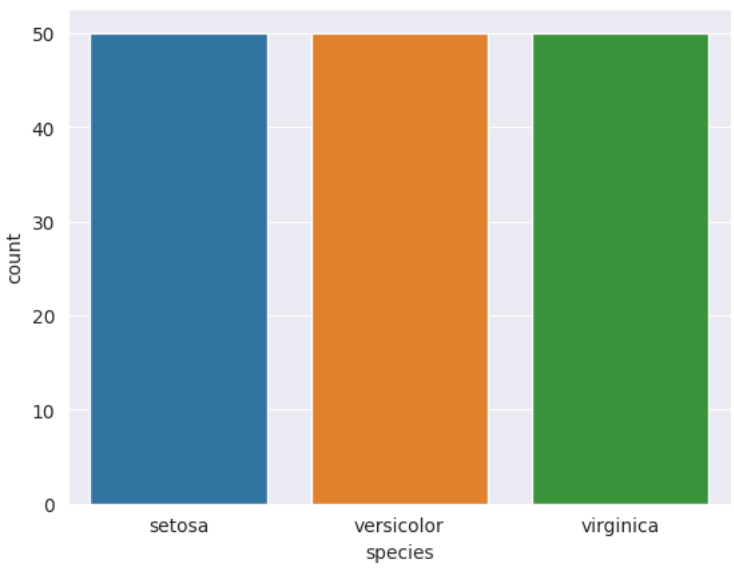

```
# Sequential Color Palette
# current colot palette
palette = sns.color_palette('YlGn', 15)

# sequential color palette
sns.palplot(palette)

plt.show()
```

```
# Bar Plot

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

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



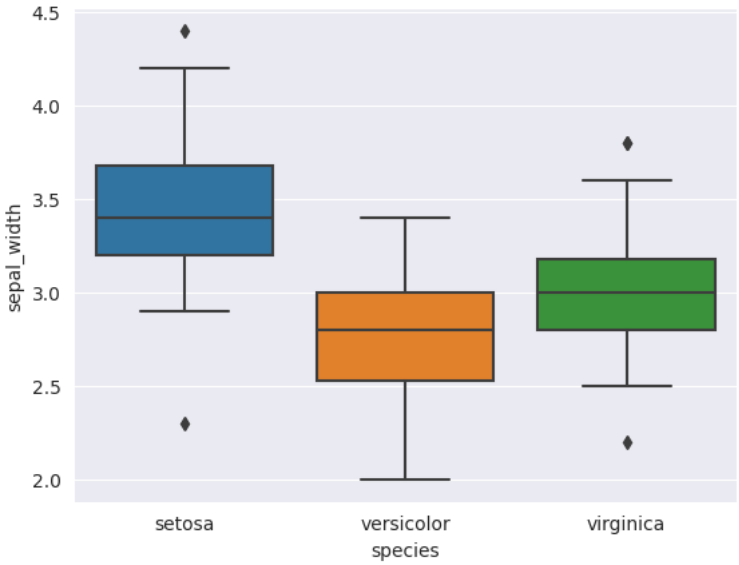

```
# Count Plot
```

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

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



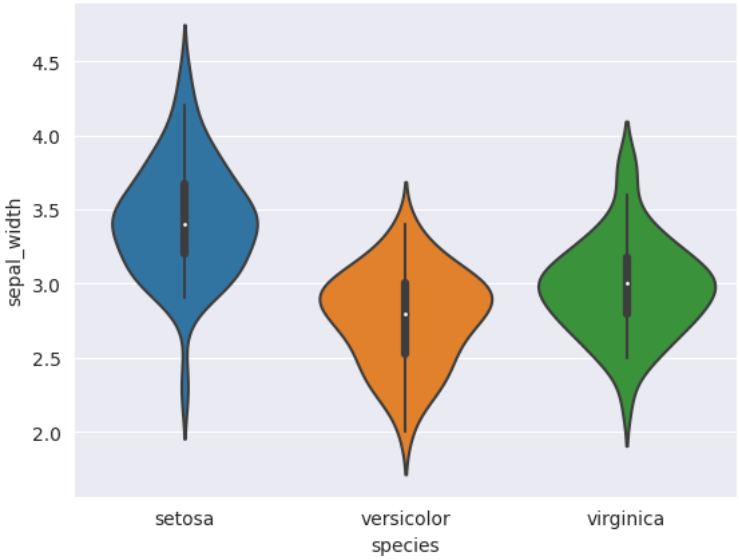
A **box plot, also known as a box-and-whisker** plot, is a statistical data visualization that provides a concise summary of the distribution of a continuous dataset. It displays key statistics such as the median, quartiles, and potential outliers in a clear and intuitive manner.

```
# Box Plot

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

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



A **violin plot** is a type of data visualization that combines features of a box plot and a kernel density plot to provide insights into the distribution of a dataset. It is particularly useful for comparing the distribution of multiple datasets or groups. The name "violin plot" is derived from the shape of the plot, which resembles a violin.

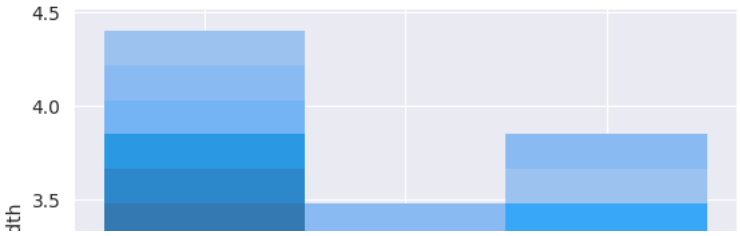
```
# Violinplot

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

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



A **strip plot** is a type of data visualization used to display individual data points along a single axis. It is particularly useful for visualizing the distribution of data within a categorical variable or comparing the distribution of a continuous variable across different categories.

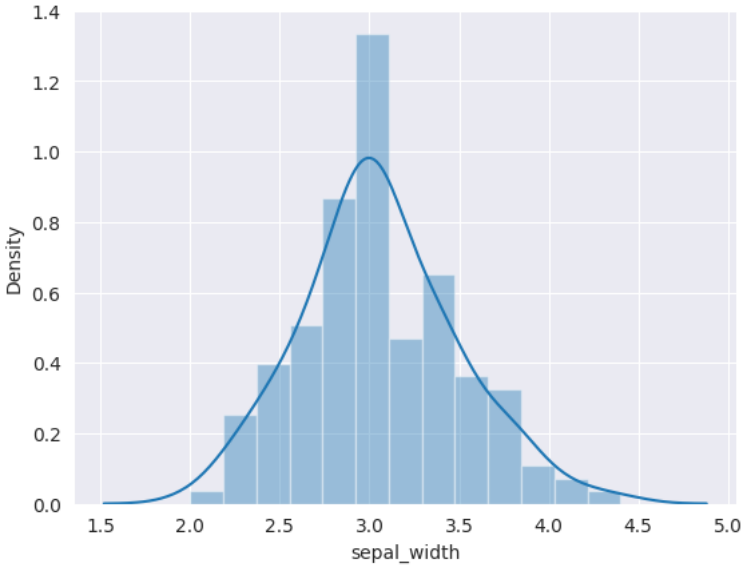
```
# Stripplot

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

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


```
# Histogram

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

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



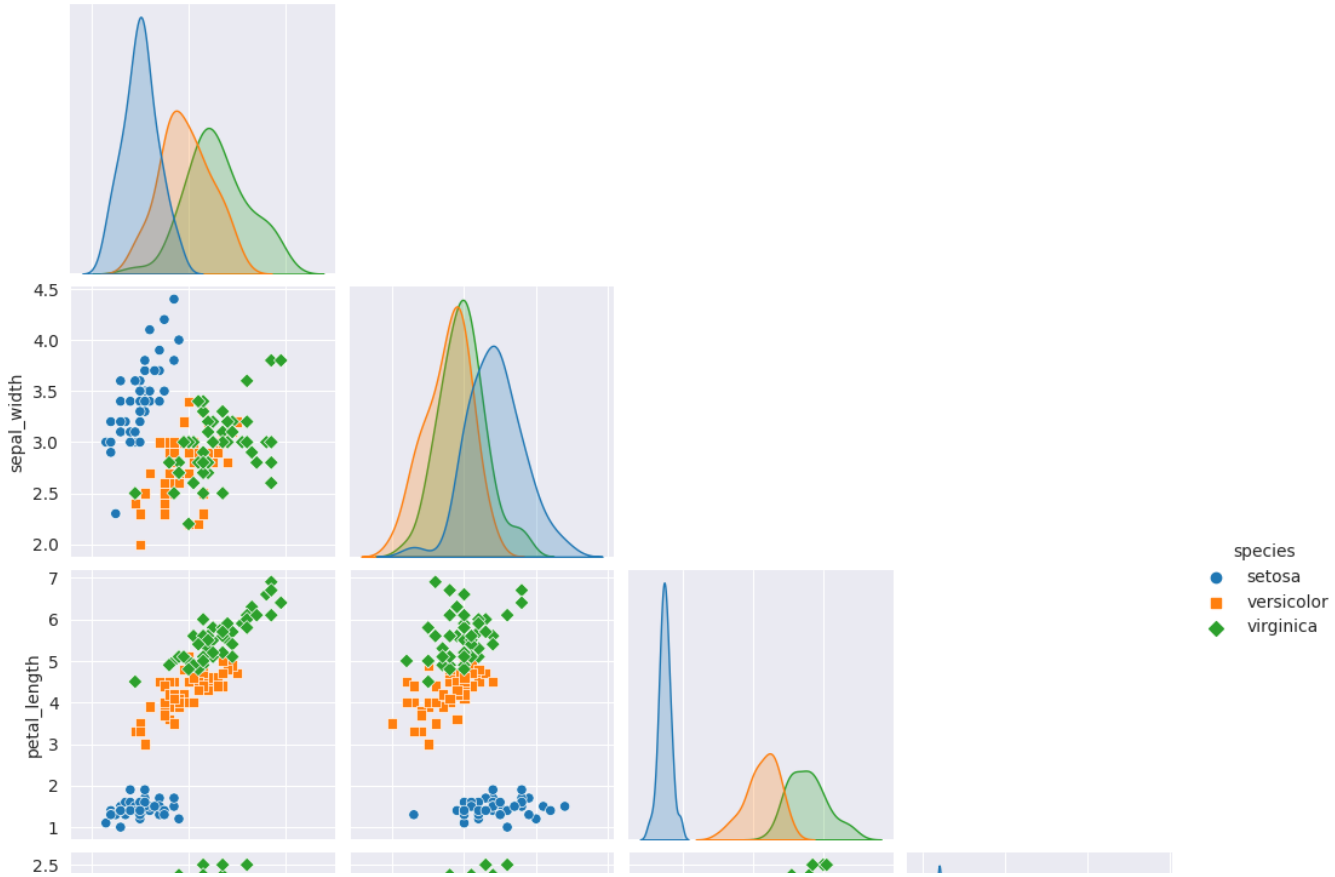

```
# Distplot
# loading dataset
data = sns.load_dataset("iris")

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

```
<ipython-input-19-fc27cf1511da>:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

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

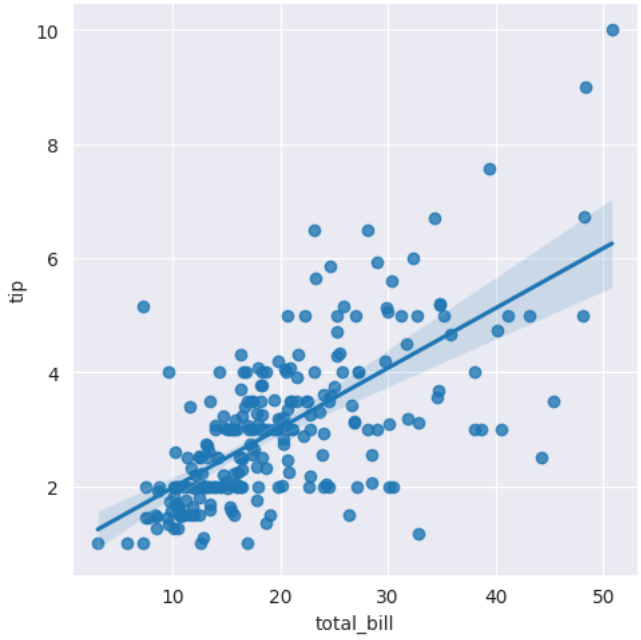


A **pair plot**, also known as a scatterplot matrix, is a data visualization that provides a grid of scatter plots, each showing the relationships between pairs of variables in a dataset. Pair plots are particularly useful for identifying patterns, correlations, and potential insights when dealing with multiple variables simultaneously.

```
# Pairplot - distribution of a single variable and relationship between two varaibles
# loading dataset
data = sns.load_dataset("iris")

sns.pairplot(data=data, hue="species", markers=["o", "s", "D"], corner=True )
plt.show()
```




```
# Regression Plots

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

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

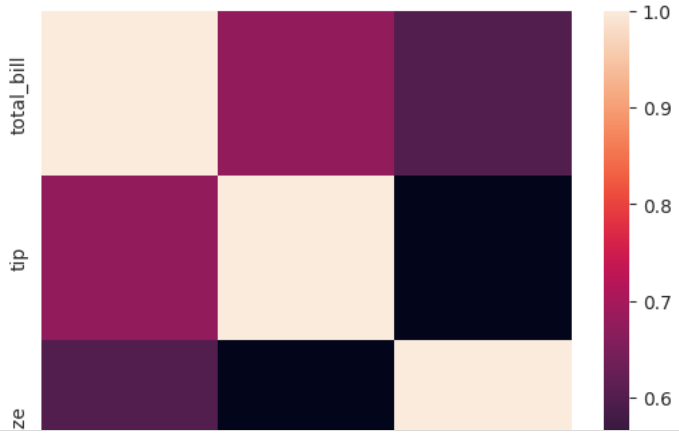



```
# loading dataset
data = sns.load_dataset("tips")

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

A **heatmap** is a graphical representation of data where values are displayed using a color gradient. Heatmaps are particularly useful for visualizing large and complex datasets, especially when the data is organized in a matrix-like structure. They provide an intuitive way to identify patterns, trends, and relationships in the data by using colors to represent different levels of values.

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

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

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

```
<ipython-input-22-aab631024fa0>:6: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ve
  tc = data.corr()
```



✓ 0s completed at 10:19 AM