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

Seaborn

- 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.
- Seaborn is also closely integrated with the Panda's data structures.

> **Installation of Seaborn:**

- First, we should ensure that, **Python** and **PIP** are installed in the system, then can install seaborn using the following pip command:

```
pip install seaborn
```

> **Import Seaborn:**

- After successful installation of **seaborn**, we can import it using the following import module statement:

```
import seaborn
```

```
import seaborn
```

> **Checking Seaborn Version**

- The version string is stored under **__version__** attribute.

```
import seaborn

print(seaborn.__version__)
```

```
0.12.2
```

Categories of Plots in Python's seaborn library

- **Distribution plots:** This type of plot is used for examining both types of distributions, i.e., univariate and bivariate distribution.
- **Relational plots:** This type of plot is used to understand the relation between the two given variables.
- **Regression plots:** Regression plots in the seaborn library are primarily intended to add an additional visual guide that will help to emphasize dataset patterns during the analysis of exploratory data.
- **Categorical plots:** The categorical plots are used to deals with categories of variables and how we can visualize them.
- **Multi-plot grids:** The multi-plot grids are also a type of plot that is a useful approach is to draw multiple instances for the same plot with different subsets of a single dataset.
- **Matrix plots:** The matrix plots are a type of arrays of the scatterplots.

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:**

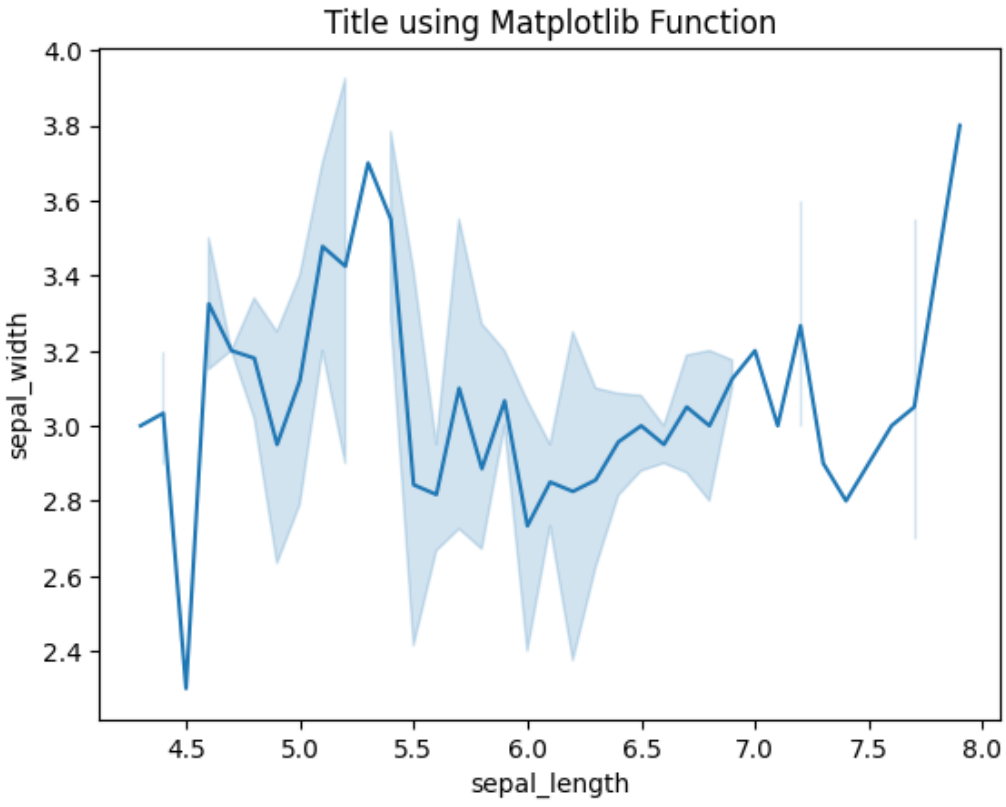
- Just invoke the Seaborn Plotting function as normal, and then use Matplotlib's customization function.

```
# 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()
```

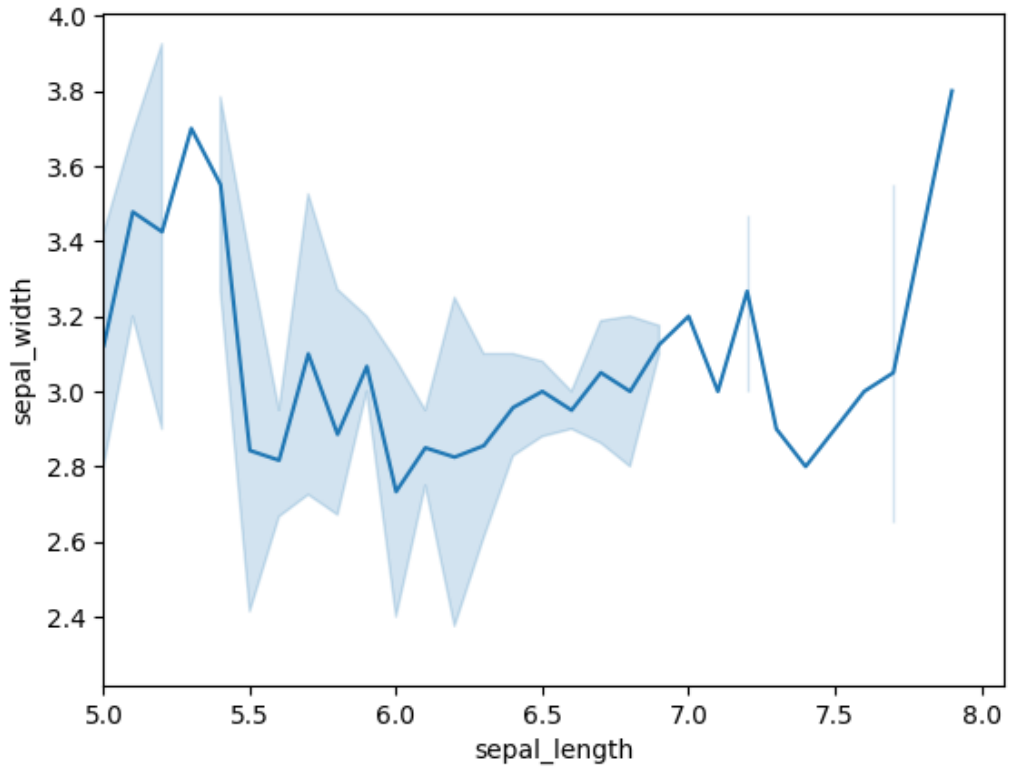


> **Setting the xlim and ylim:**

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

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

plt.show()
```



Creating Different Types of Plots

- **Relational Plots:**
    1. Relational Plot
    2. Scatter Plot
    3. Line Plot
- **Categorical Plots:**
    1. Bar Plot
    2. Count Plot
    3. Box Plot
    4. Violinplot
    5. Stripplot
    6. Swarmplot
    7. Factorplot
- **Distribution Plot:**
    1. Histogram
    2. Distplot

3. Jointplot
4. Pairplot
5. Rugplot
6. KDE Plot

- **Regression Plots:**

1. lmplot
2. Regplot
3. Matrix Plots
4. Heatmap
5. Clustermap

Categorical Plots

- Categorical Plots 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.

There are various types of categorical plots:

1. Bar Plot
2. Count Plot
3. Box Plot
4. Violinplot
5. Stripplot
6. Swarmplot
7. Factorplot

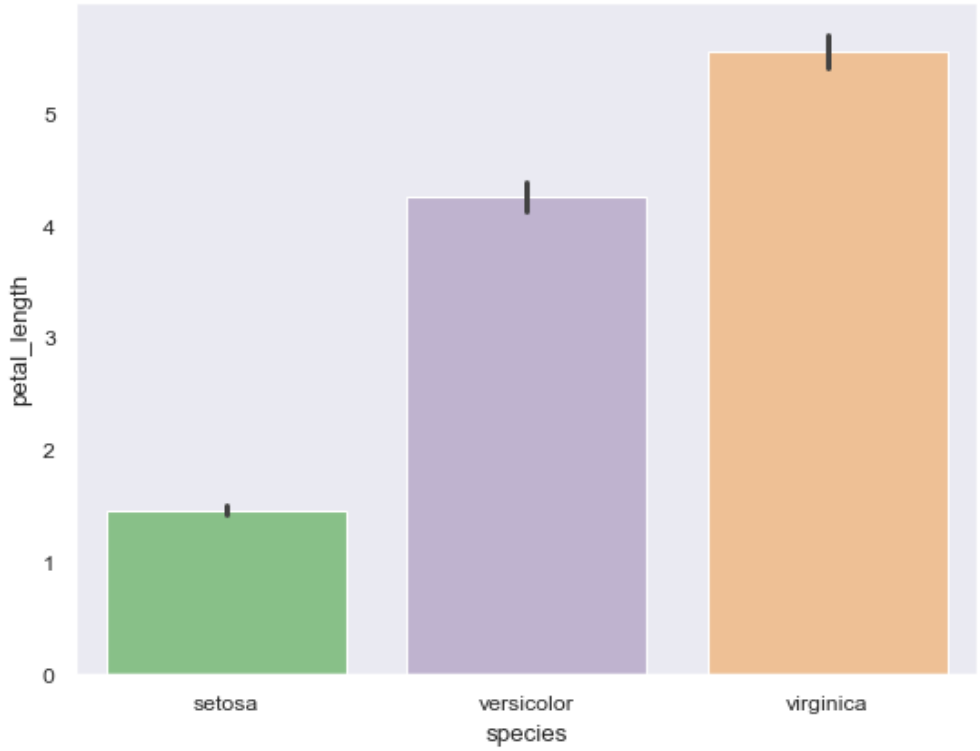
> **1. Bar Plot:**

- 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, …])
```

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

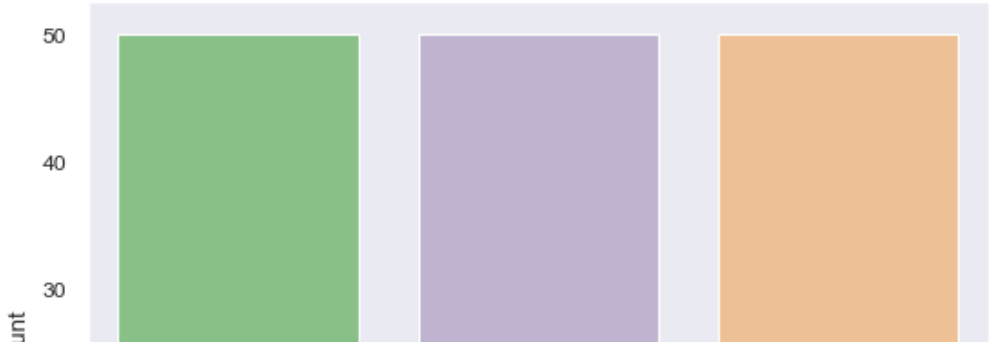


> **2. Count Plot:**

- 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, …])
```

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



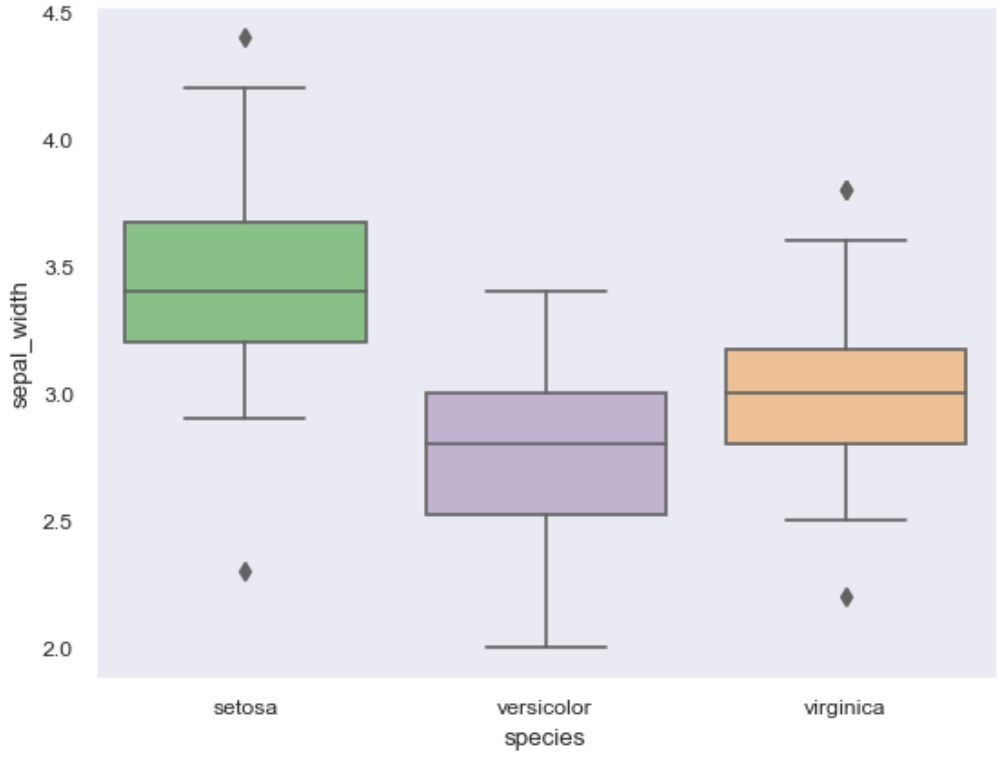
3. Box Plot:

- 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, …])
```

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

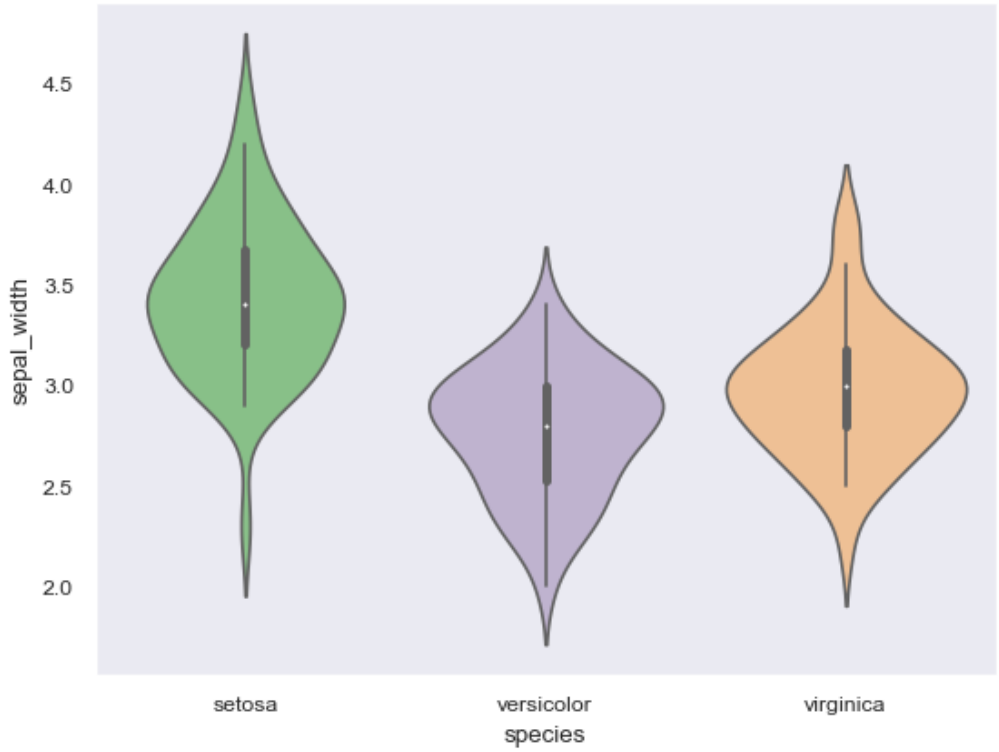


4. Violinplot:

- 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, …])
```

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



5. 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, …])
```

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

6. 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, …])
```

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

> **7. 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, …])
```

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

Distribution Plots

- Distribution Plots are used for examining univariate and bivariate distributions meaning such distributions that involve one variable or two discrete variables.

There are various types of distribution plots:

1. Histogram
2. Distplot
3. Jointplot
4. Pairplot
5. Rugplot
6. KDE Plot

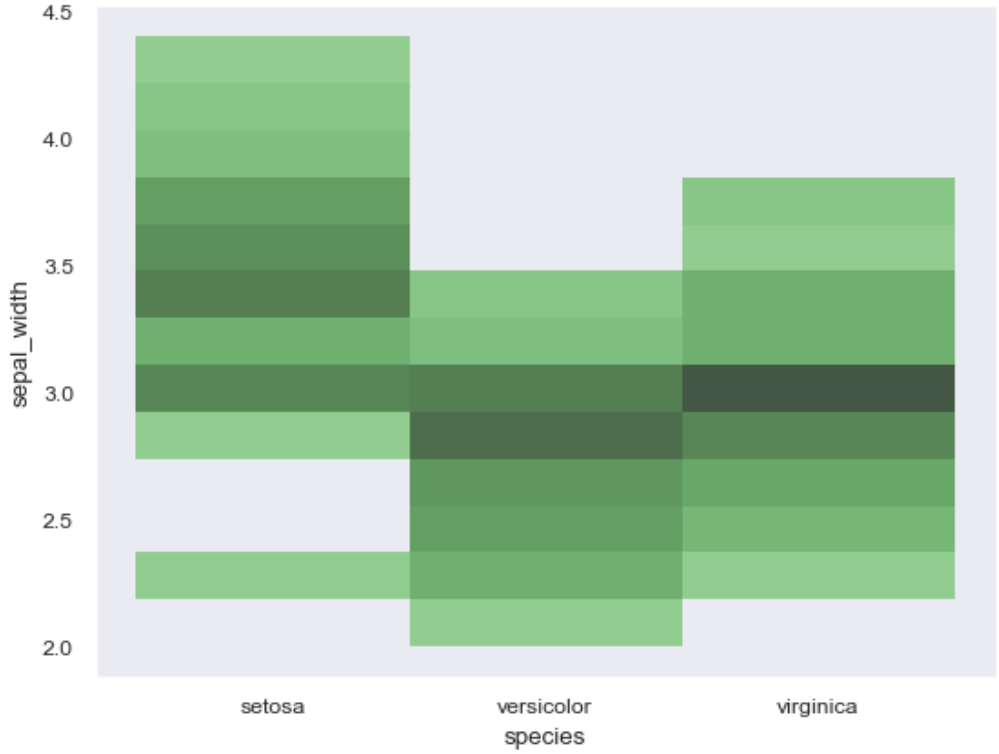
> **1. Histogram:**

- 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)
```

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



2. 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, …])
```

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

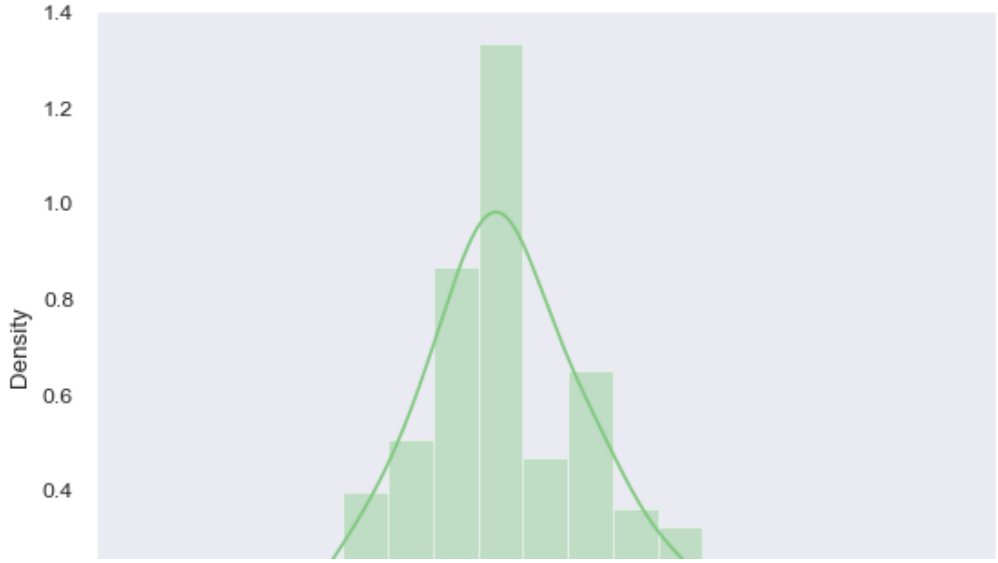
```
C:\Users\ABC\AppData\Local\Temp\ipykernel_11140\327049665.py:1: UserWarning:

`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'])
```



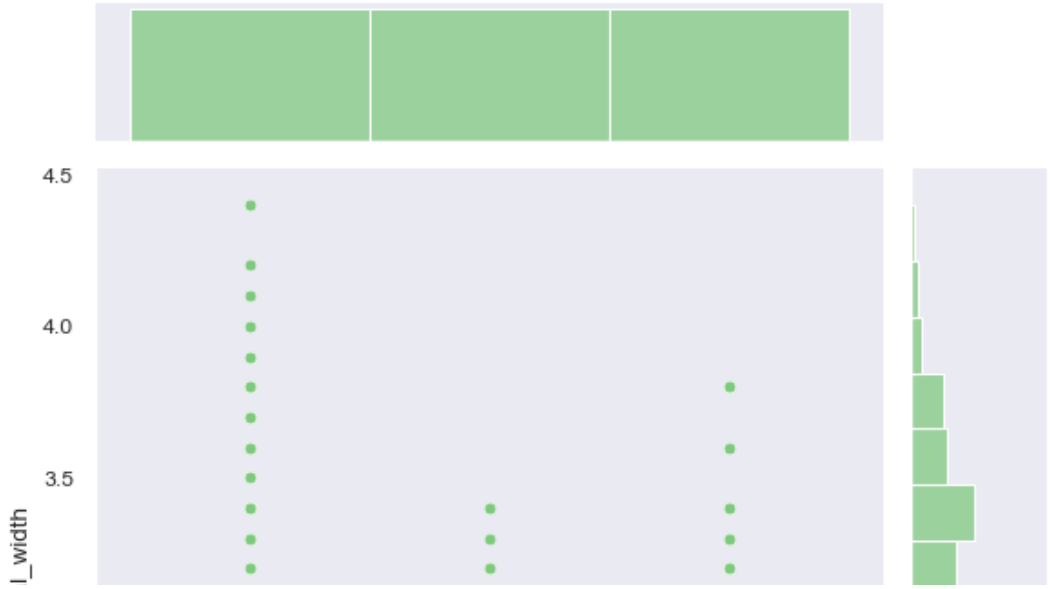
3. Jointplot:

- 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, …])
```

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



4. Pairplot:

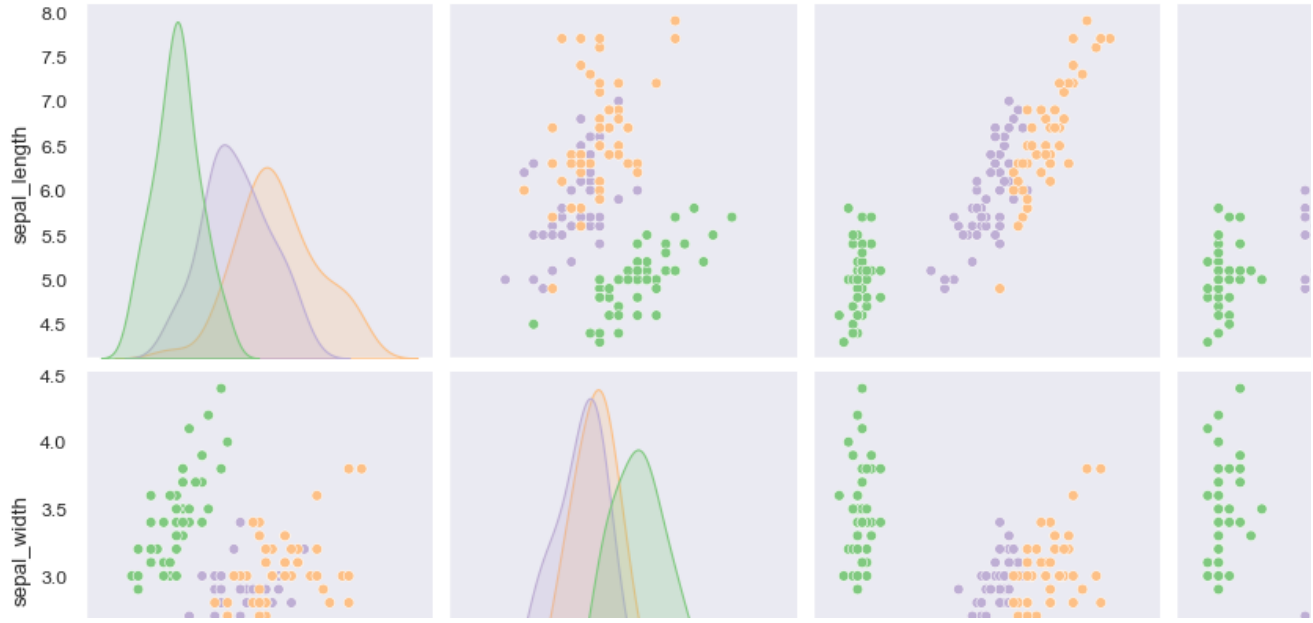
- **Pairplot** represents pairwise relation across the entire dataframe and supports an additional argument called **hue** for categorical separation.
- What it does basically, is create a jointplot between every possible numerical column and takes a while if the dataframe is really huge.
- It is plotted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

- **Syntax:**

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

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

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



5. 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])
```

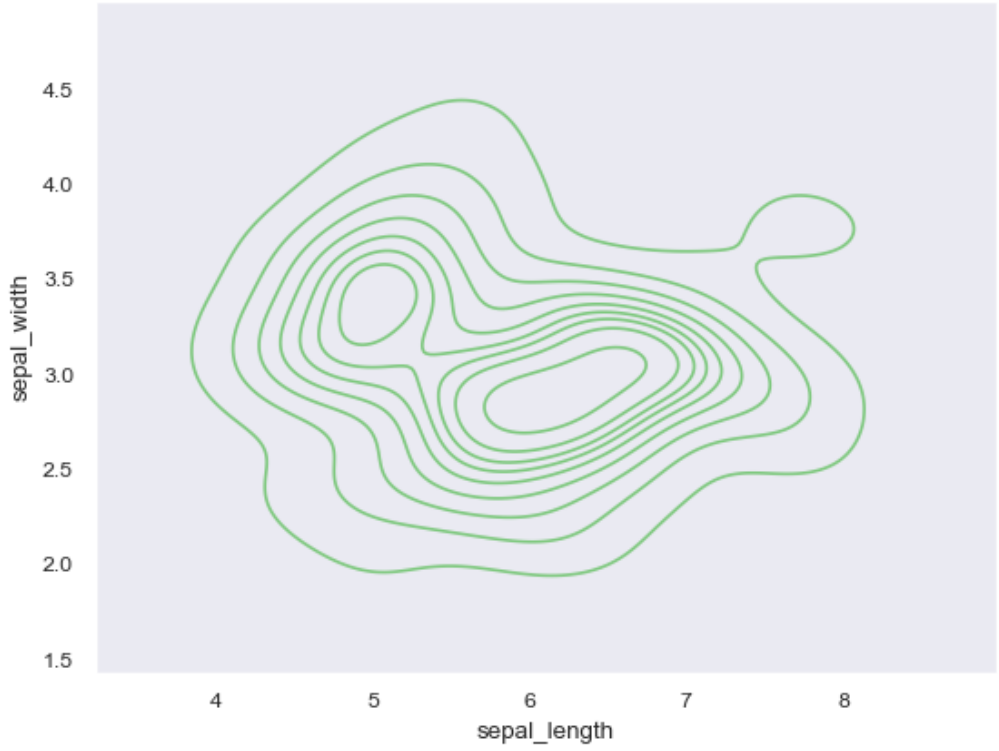
```
sns.rugplot(data=data)
plt.show()
```

6. 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)
```

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



✓ 0s completed at 18:50