VISUALIZATION

Data visualization is a crucial part of data analysis and machine learning. It helps in understanding patterns, trends, and relationships in data by representing it graphically. Python provides several powerful libraries for data visualization, including **Matplotlib, Seaborn, Plotly, and Pandas visualization**.

Let us dive deeper into matplotlib and seaborn.

MATPLOTLIB

Matplotlib, introduced in 2002 by John Hunter,is a widely used Python library for data visualization. It allows users to create static, animated, and interactive plots in Python. It is particularly useful for creating graphs, charts, and other visual representations of data.

Key Features:

- **2D and 3D plotting**: Supports line plots, scatter plots, bar charts, histograms, pie charts, and even 3D visualizations.
- **Highly customizable**: Users can modify every aspect of a plot, including colors, labels, and styles.
- **Integration with NumPy and Pandas**: Works seamlessly with other scientific libraries.
- **Supports multiple backends**: Can generate plots in different formats (PNG, PDF, SVG, etc.).
- **Interactivity**: Enables zooming, panning, and interactive visualization when used with Jupyter Notebook.

PYPLOT

Pyplot is a sub library of matplotlib where all the utilities lie under. It has different types of plots including bar graphs, scatter plots, pie charts, histograms, area charts.

We import pyplot from matplotlib as following:

```
1 import matplotlib.pyplot as plt
```

We also import numpy as a support for the matplotlib :

```
import numpy as np
```

Some of the plots in matplotlib:

LINE CHART

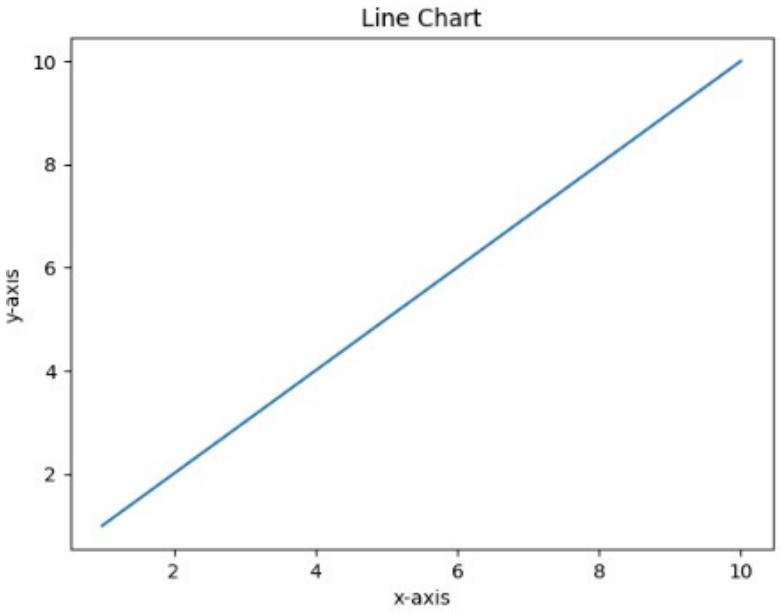
Line charts are the basic charts where dot consecutive points are connected by a continuous line.

The default plot() is used for line charts.

Code snippet:

```
import matplotlib.pyplot as plt
import numpy as np

x=np.array([1,10])
y=np.array([1,10])

plt.plot(x,y)
plt.title("Line Chart")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.show()
```

Output:



Description:

np.array() will generate an array in the specified range.

plot() for the plotting the line chart.

title() for defining the title, ylabel() and xlabel () for labelling y and x axis respectively.

show() displays the graph.

BAR GRAPH

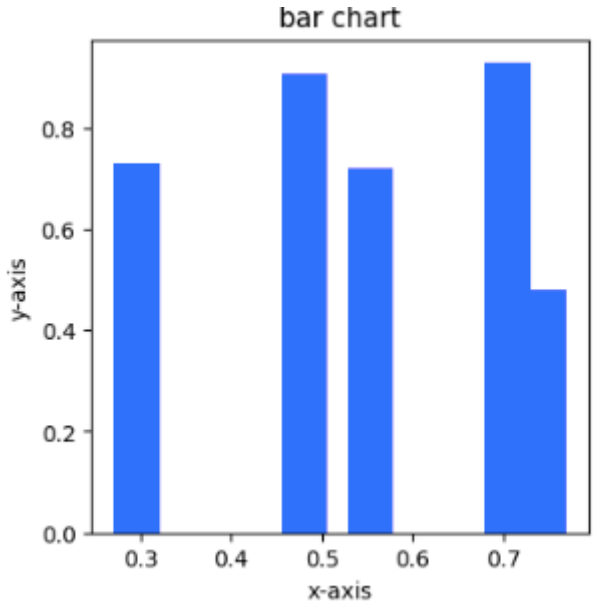
Rectangular bars are used in bar chart as the height represent the frequency of a particular element. bar() is used for plotting bar graphs, it has parameters like x, y, width, color.

bar(x, y, color, width)

Code snippet:

```
import matplotlib.pyplot as plt
import numpy as np

x=np.random.rand(5)
y=np.random.rand(5)
print(x)

fig=plt.figure(figsize=(4,4))
plt.bar(x,y,width=0.05,color="blue")
plt.title("bar chart")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.show()
```

Output:



Description:
random.rand(5) generates a random array.

Here bar() is used along with the parameters, width which denoted the width of the rectangular bars and color for the bars.

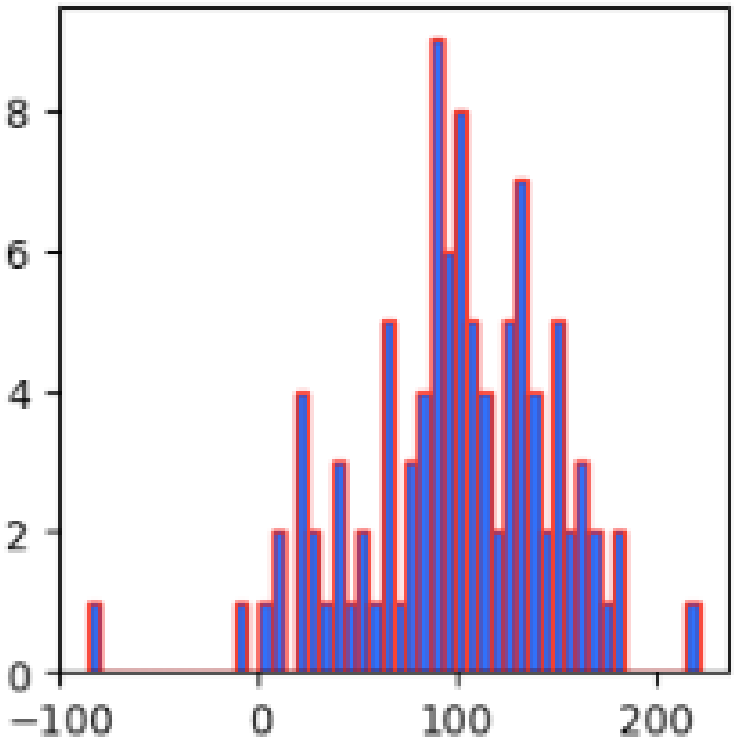
HISTOGRAM

Histogram is a type of bar graph where the graph is represented in groups. hist() is used for plotting histogram with parameters x, bins, color, edgecolor.

Code snippet:

```
import matplotlib.pyplot as plt
import numpy as np

x=np.random.normal(100,50,100)
fig=plt.figure(figsize=(3,3))
plt.hist(x,bins=50,color="blue",edgecolor="red")
plt.show()
```

Output:



Description:

hist() have parameters x which is a random generated array around one hundred with standard deviation 50 of 100 values.

bins indicate the number of sections on x axis.

color indicates the color of rectangular bars.

edge color that divides the rectangular bars.

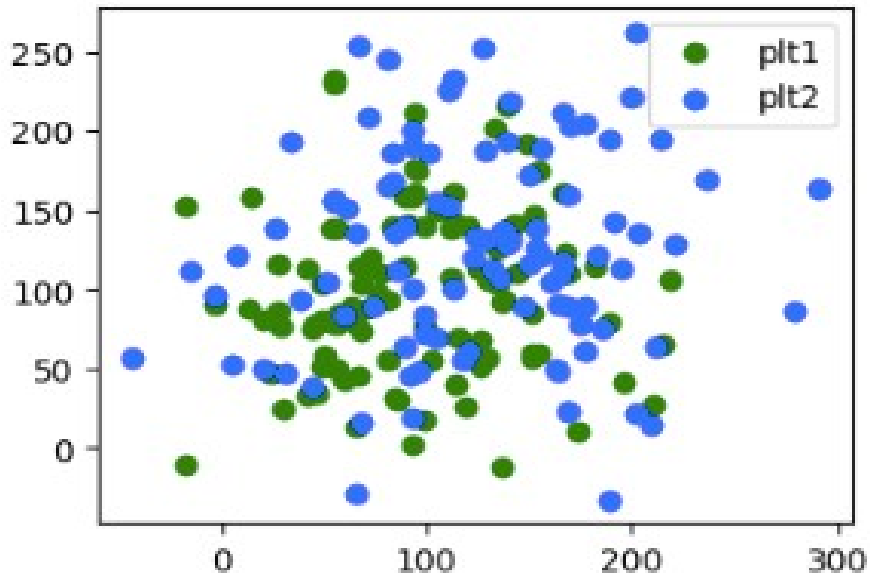
SCATTER PLOT

Scatter plot uses dots to depict the relationship between the data. scatter() is used for plotting scatter plot and scatter plot can be done for multiple datasets at the same time by differentiating it with colors. Legends help to achieve this difference.

Code snippet:

```
import matplotlib.pyplot as plt
import numpy as np

x1=np.random.normal(100,50,100)
y1=np.random.normal(100,50,100)
x2=np.random.normal(120,60,100)
y2=np.random.normal(120,60,100)
fig=plt.figure(figsize=(4,3))
plt.scatter(x1,y1,c="g")
plt.scatter(x2,y2,c="b")
plt.legend(["plt1","plt2"])

plt.show()
```

Output:



Description:

x1, y1 belong to one dataset and x2,y2 belong to another dataset.

Here the scatter() is used for the scatter plot and the parameters x, y and c which represents color.

Legend adds the label which helps to differentiate the plots.

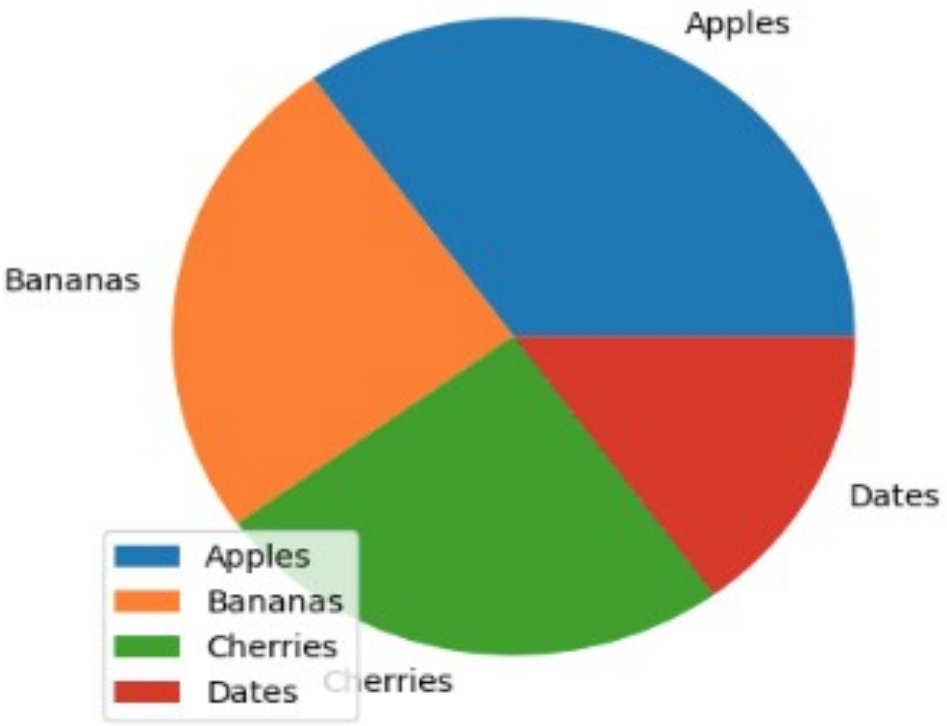
PIECHART

Pie charts are used to plot data of same kind which means the same series of data where the different elements are divide based on their percentage.

Code snippet:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend()
plt.show()
```

Output



Description:

Here,

The **np.array()** method initializes an array representing data values.

- The **plt.pie()** method generates the pie chart with labeled sections.
- The **plt.legend()** method adds a legend for better readability.

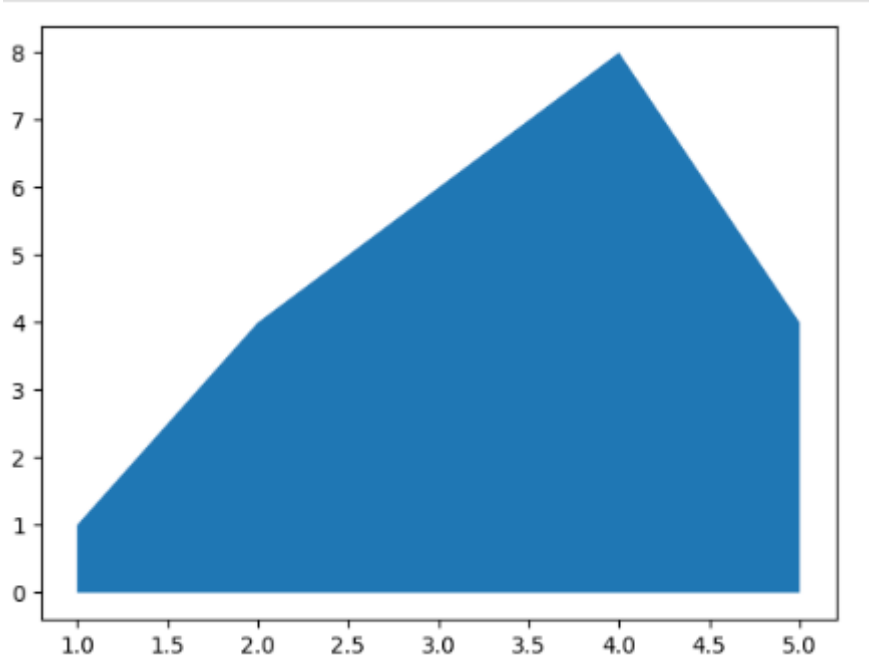
AREA CHART

In area chart, the area under the line to x axis is filled or shaded. It can be done by using fill_between() function or stackplot() function.

Code snippet:

```
import numpy as np
import matplotlib.pyplot as plt

x=range(1,6)
y=[1,4,6,8,4]

plt.fill_between(x, y)
plt.show()
```

Output:



Description:

The **range(1,6)** defines the x-axis values.

The **plt.fill_between(x, y)** method fills the area between the x-axis and the given y-values, creating a shaded region.

SEABORN

Seaborn is a powerful **data visualization** library built on top of Matplotlib for **Python**. It provides a **high-level interface** for creating attractive and informative statistical graphics.

Key Features of Seaborn:

1.Built-In Themes:

- Seaborn provides various themes like darkgrid, whitegrid, dark, white, and ticks for styling plots.

2.Statistical Visualization:

Seaborn integrates well with Pandas and supports statistical functions like regression plotting, KDE (Kernel Density Estimation), and histograms.

3.Works with Pandas DataFrames:

You can directly pass Pandas DataFrames to Seaborn functions, making data visualization seamless.

4.Advanced Categorical Plots:

Provides functions like boxplot(), violinplot(), and stripplot() to visualize categorical data.

5.Beautiful Color Palettes:

Seaborn has built-in color palettes like deep, muted, bright, and can use color codes.

6.Pairwise and Correlation Plots:

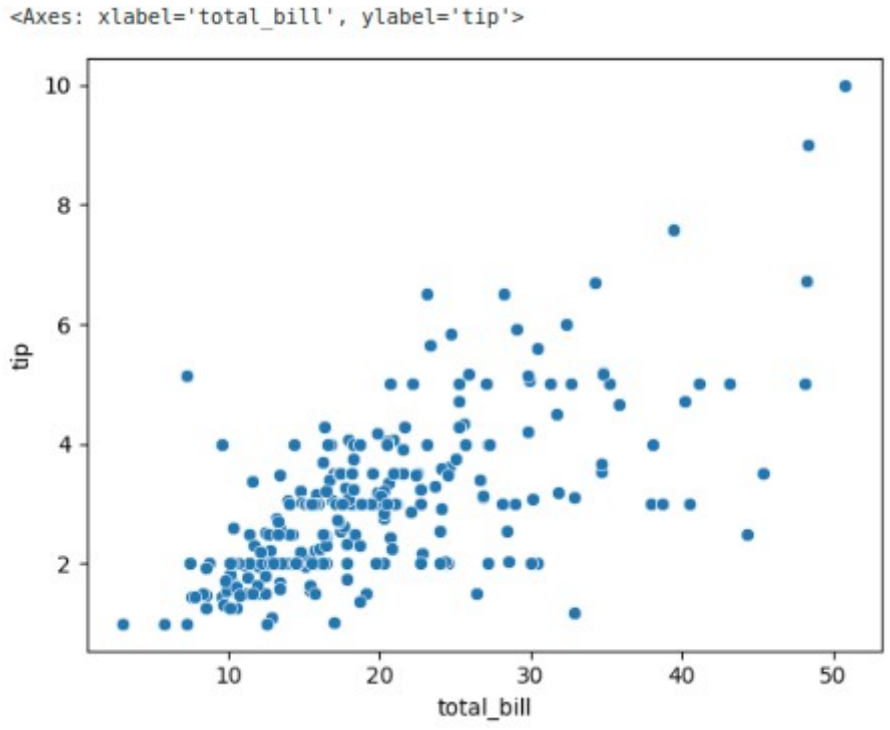
Functions like pairplot() and heatmap() make it easy to explore relationships between multiple variables.

Here are some sample codes for some of the graphs:-

Scatter plot:

```
import seaborn as sns

tips = sns.load_dataset("tips")

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

Output:



Description:

In this Code-

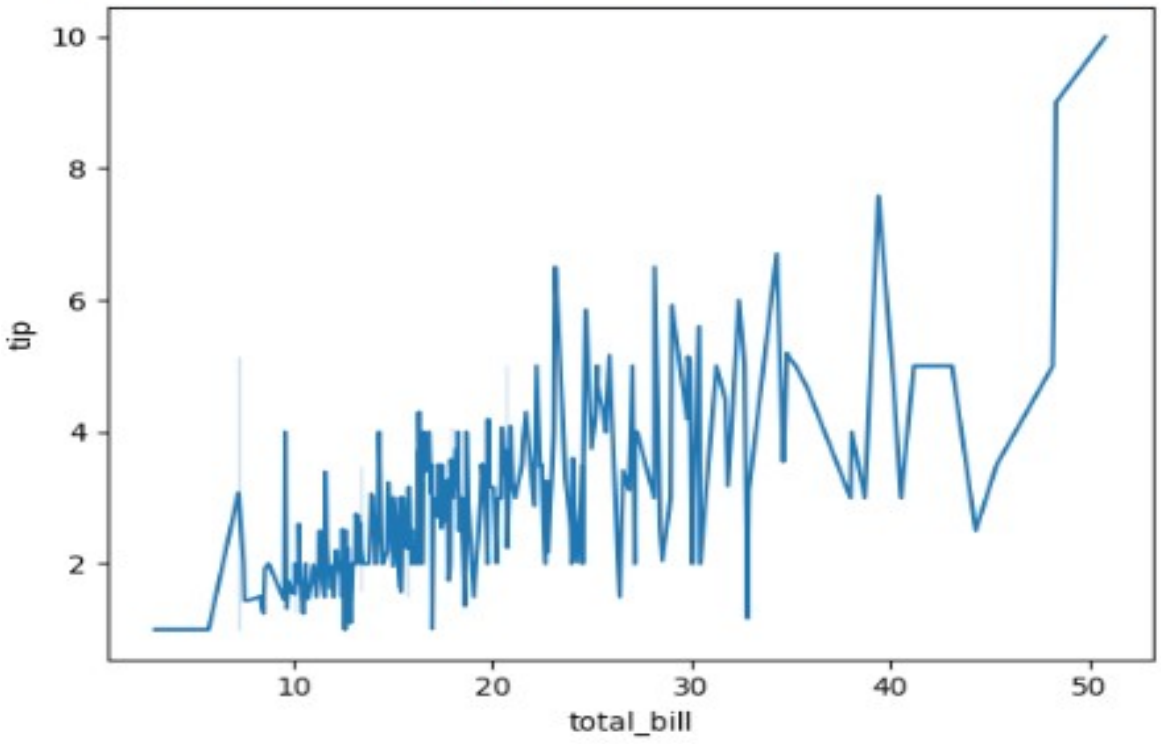
- The **sns.load_dataset("tips")** method loads the built-in "tips" dataset.
- The **sns.scatterplot()** method plots points where the x-axis represents total_bill and the y-axis represents tip.

LINE PLOT

```
import seaborn as sns

tips = sns.load_dataset("tips")

sns.lineplot(x="total_bill", y="tip", data=tips)
```

Output:

```
<Axes: xlabel='total_bill', ylabel='tip'>
```



Description:

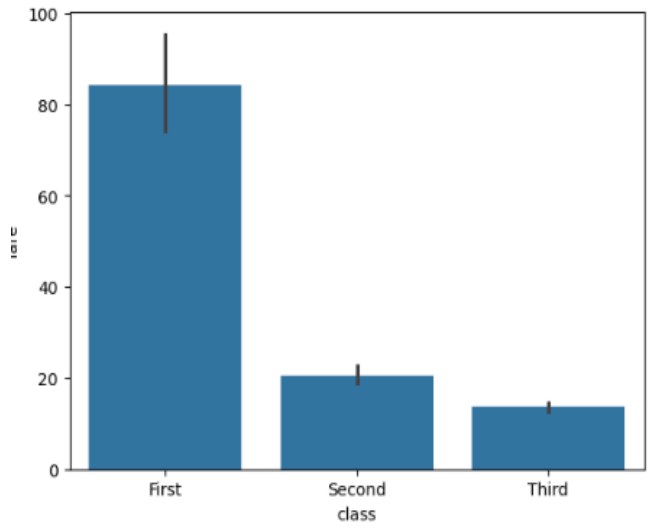
The **sns.load_dataset("tips")** method loads the built-in "tips" dataset.

The **sns.lineplot()** method plots a line connecting data points where total_bill is on the x-axis and tip is on the y-axis.

BAR PLOT

```
import seaborn as sns

titanic = sns.load_dataset("titanic")

sns.barplot(x="class", y="fare", data=titanic)
```

Output:



Description:

The **sns.load_dataset("titanic")** method loads the built-in "titanic" dataset.

The **sns.barplot()** method plots bars where the x-axis represents passenger **class**, and the y-axis represents the **fare** paid.

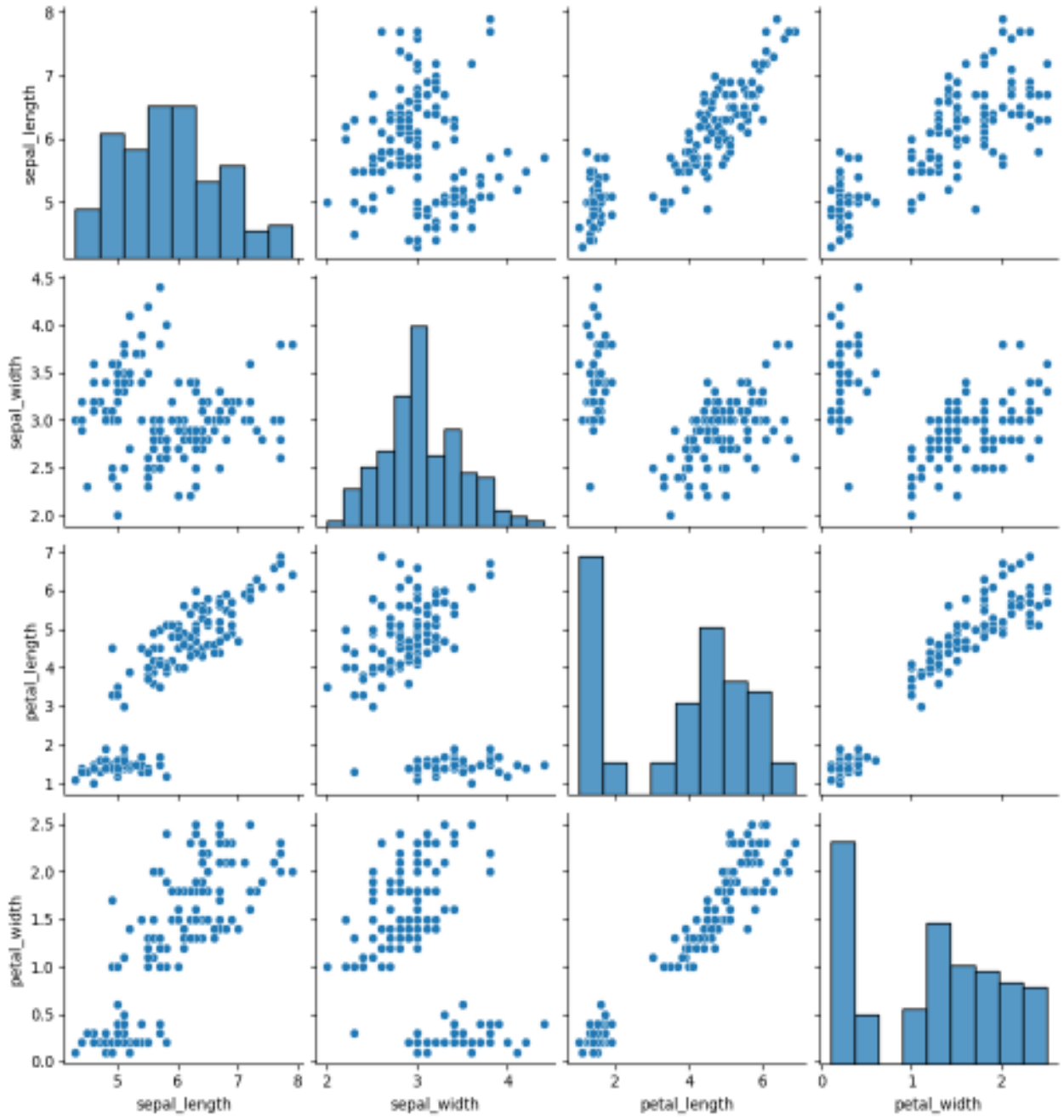
PAIR PLOT

Pairplot is a data visualization tool that creates a matrix of scatterplots, showing pairwise relationships between variables in a dataset

Code Snippet:

```
import seaborn as sns

iris = sns.load_dataset("iris")

sns.pairplot(data=iris)

plt.show()
```

Output:



Description:

The **sns.load_dataset("iris")** method loads the built-in **Iris** dataset, which contains data on different flower species.

The **sns.pairplot()** method creates scatter plots for each pair of numerical columns, allowing for pattern and correlation analysis.

plt.show() displays the plot.

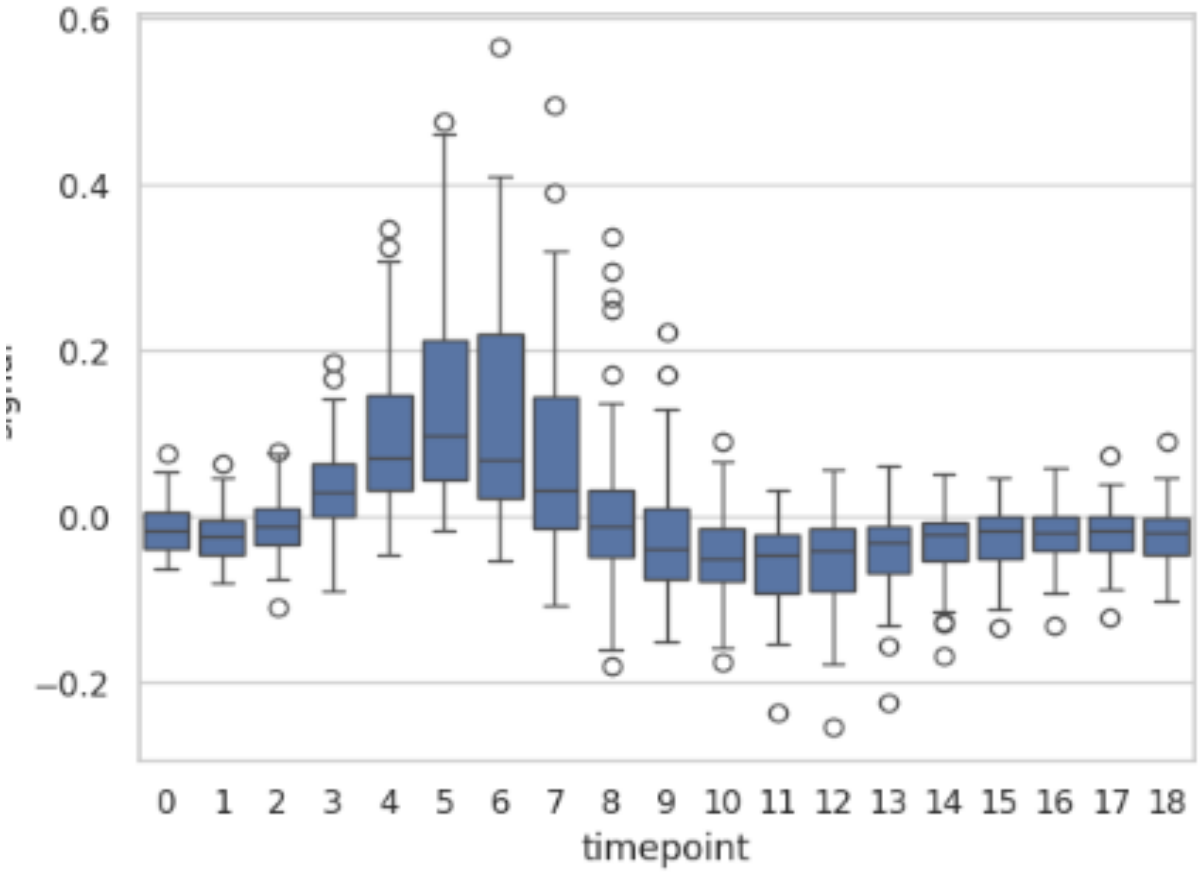
BOX PLOT

Box plot shows the quartiles where the data lies in interquartile range will be inside the box. The points which are away from the whiskers are outliers.

Code snippet:

```
import seaborn

seaborn.set(style='whitegrid')
fmri = seaborn.load_dataset("fmri")

seaborn.boxplot(x="timepoint",
                y="signal",
                data=fmri)
```

Output:



Description:

seaborn.set(style='whitegrid') applies a clean, grid-based background for better readability.

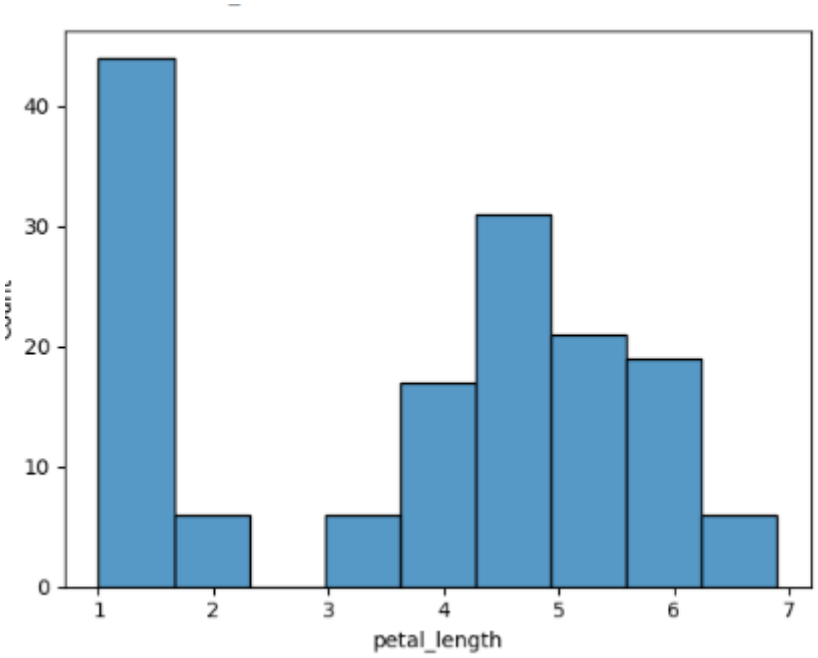
- **seaborn.load_dataset("fmri")** loads the built-in **fmri** dataset, which contains brain activity data.
- **seaborn.boxplot()** creates a box plot where:

- **x-axis:** Represents **timepoints** (independent variable).
- **y-axis:** Represents **signal values** (dependent variable).

HISTPLOT

```
import seaborn as sns

iris = sns.load_dataset("iris")

sns.histplot(x="petal_length", data=iris)
```

Output:



Description:

sns.load_dataset("iris") loads the built-in **Iris** dataset, which contains measurements of different flower species.

- **sns.histplot()** creates a histogram where:
- **x-axis:** Represents **petal length** values.
- **y-axis:** Represents the **frequency** of occurrences in each bin.

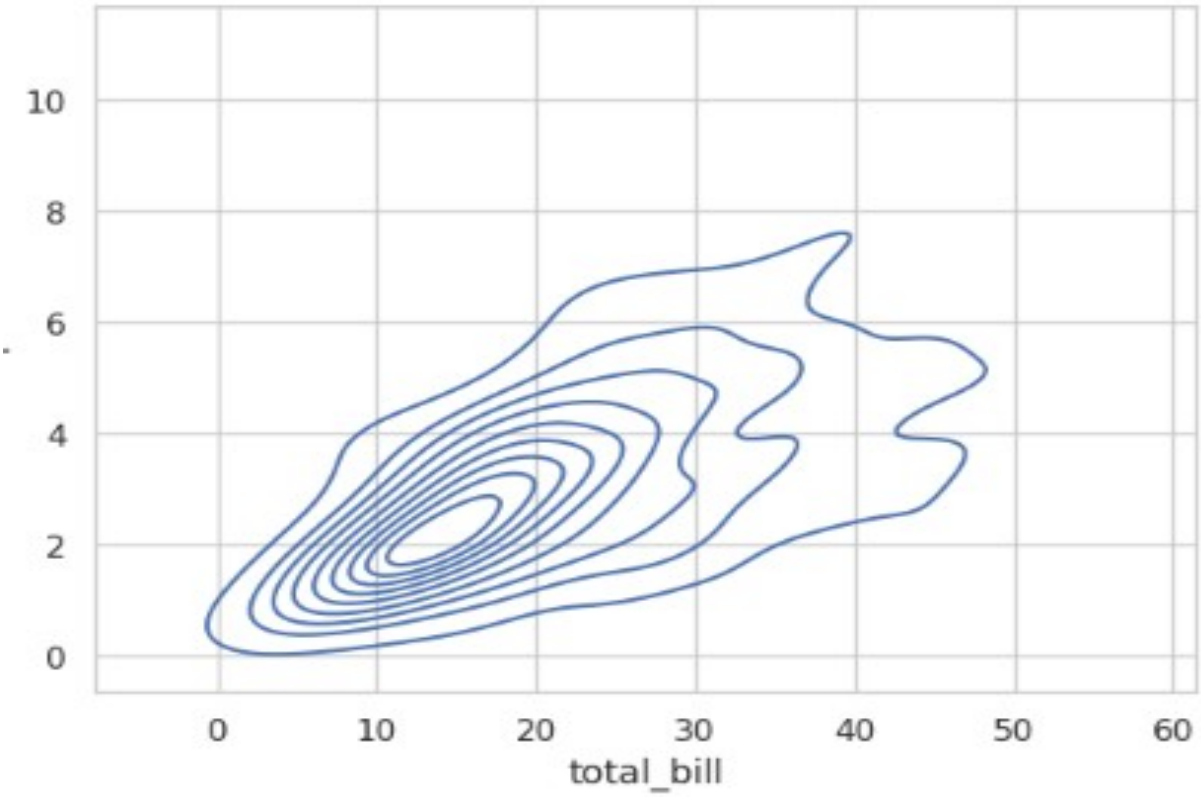
KDEPLOT

KDE stands for kernel density estimate. It creates a curve as based on probability. It creates single graph for multiple data samples.

Code snippet:

```
import seaborn as sns

tips = sns.load_dataset("tips")

sns.kdeplot(x="total_bill", y="tip", data=tips)
```

Output:



Description:

sns.load_dataset("tips") loads the built-in dataset containing restaurant bill and tip data.

- **sns.kdeplot()** creates a KDE plot where:
- **x-axis:** Represents **total_bill** values.
- **y-axis:** Represents **tip** values.

- The plot shows density contours, indicating where data points are concentrated.

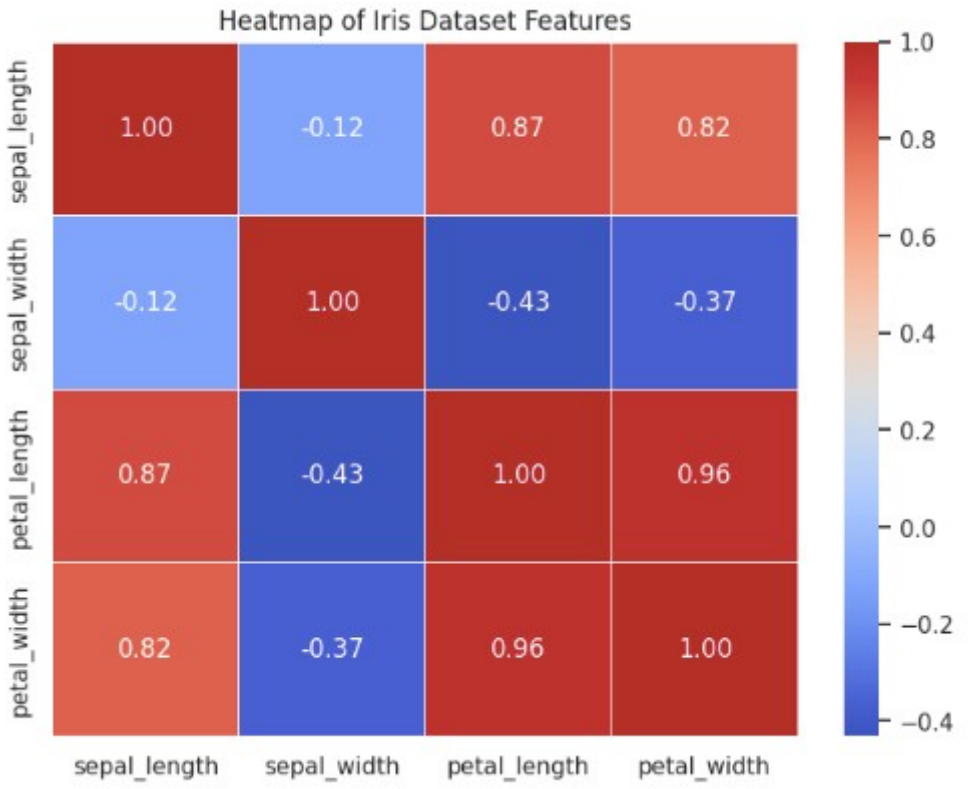
HEATMAP

Heatmaps uses correlation matrix and visualizes data. The datapoints where the higher values get brighter colors and lower values get darker colors.

Code snippet:

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

iris = sns.load_dataset("iris")

corr_matrix = iris.drop(columns=["species"]).corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

plt.title("Heatmap of Iris Dataset Features")
plt.show()
```

Output:



Description:

This Python code generates a **heatmap** to visualize the correlation between numerical features in the **Iris** dataset.

- **sns.load_dataset("iris")** loads the built-in **Iris** dataset.
- **corr_matrix = iris.drop(columns=["species"]).corr()** computes the correlation matrix, excluding the categorical **species** column.
- **sns.heatmap()** creates a heatmap where:
  - **annot=True** displays correlation values in each cell.
  - **cmap="coolwarm"** applies a color gradient for better visualization.
  - **linewidths=0.5** adds grid lines for clarity.

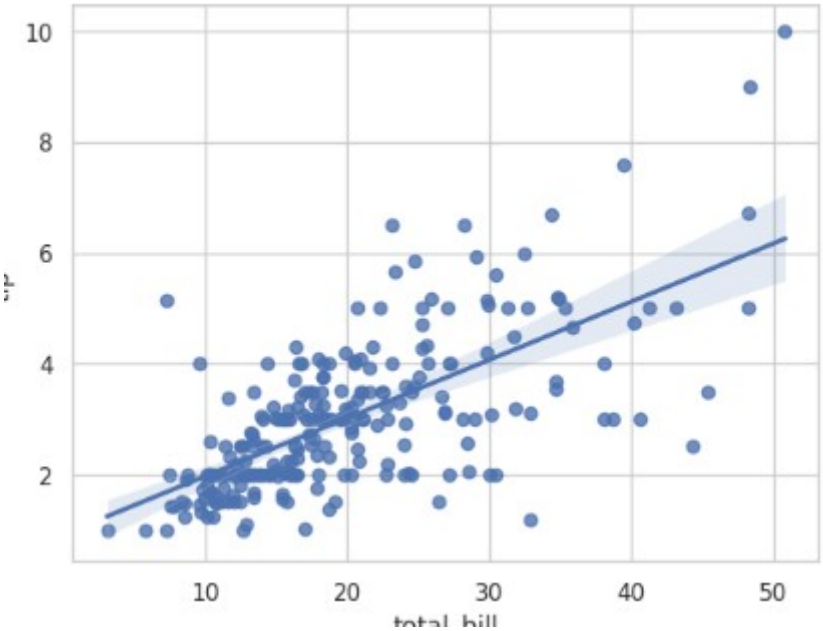
REGPLOT

Regression plots shows the relationship between variables along with the regression line.

Code snippet:

```
import seaborn as sns

tips = sns.load_dataset("tips")

sns.regplot(x="total_bill", y="tip", data=tips)
```

Output:



Description:

This Python code generates a **regression plot** using **Seaborn** to visualize the relationship between **total bill** and **tip** amounts in the **Tips** dataset.

- **sns.load_dataset("tips")** loads the built-in dataset containing restaurant bill and tip data.
- **sns.regplot()** creates a scatter plot with a **regression line**, where:
  - **x-axis:** Represents **total_bill** values.
  - **y-axis:** Represents **tip** values.
  - The regression line shows the **trend** and **correlation** between the two variables.

COMPARISON OF MATPLOTLIB AND SEABORN

Matplotlib and Seaborn are two of the most widely used Python libraries for data visualization. While both serve similar purposes, they have distinct features and use cases. Here's a detailed comparison:

1. Overview

- **Matplotlib**: A low-level, highly customizable library for creating static, animated, and interactive visualizations.
- **Seaborn**: Built on top of Matplotlib, it simplifies statistical visualization and enhances aesthetics.

2. Ease of Use

- **Matplotlib**: Requires more code to generate plots, but provides complete control over visual elements.
- **Seaborn**: Uses high-level functions that make it easier to create complex visualizations with fewer lines of code.

3. Functionality

- **Matplotlib**:
  - Supports a wide variety of plots (line, bar, scatter, histogram, etc.).
  - Allows detailed customization of every aspect of a plot.
  - Can create interactive plots using pyplot.
- **Seaborn**:
  - Specializes in statistical visualizations.

- Provides functions for visualizing relationships (sns.scatterplot(), sns.lineplot()).
- Has built-in support for categorical plots and heatmaps.

4.Integration with Pandas

- **Matplotlib**: Requires explicit handling of Pandas DataFrames (e.g., extracting columns manually).
- **Seaborn**: Works seamlessly with Pandas DataFrames, making it easier to visualize structured data.

5.Performance

- **Matplotlib**: Faster for simple plots and when handling large datasets.
- **Seaborn**: May be slightly slower due to additional processing for enhanced visuals and statistical computations.

6.Customization

- **Matplotlib**: Offers deep customization, but requires more code.
- **Seaborn**: Limited customization but provides aesthetically pleasing defaults.

7.Dependencies

- **Matplotlib**: A standalone library.
- **Seaborn**: Built on top of Matplotlib and depends on Pandas and NumPy.

Advantages of Matplotlib:

- High degree of customization and control over plots.
- Wide range of plot types and styles.
- Strong community support and extensive documentation.
- Well-suited for creating publication-quality figures and graphics.

Advantages of Seaborn:

- Simplified syntax and high-level functions for creating complex plots.
- Built-in support for statistical plotting and data exploration.
- Attractive default aesthetics and color palettes.
- Seamless integration with Pandas dataframes for data visualization.
- Ideal for exploratory data analysis and quick visualization of relationships in data.

Conclusion:

- If you need **full control and customization**, use **Matplotlib**.
- If you want **quick and attractive statistical visualizations**, use **Seaborn**.