Visualization Library Documentation

(Matplotlib and Seaborn)

Data visualization is a key part of data analysis as it helps in identifying patterns, trends, and insights from data. Python provides several libraries for creating visual representations. In this documentation, I have explained two popular Python visualization libraries, Matplotlib and Seaborn, along with the different types of graphs they support, their use cases, and a comparison between them.

Let us dive deeper into matplotlib and seaborn.

MATPLOTLIB

Matplotlib is a Python library used for creating static and customizable visualizations. It gives full control over plot elements such as axes, labels, and colors. Matplotlib is commonly used when detailed customization and precise control over graphs are required. A plot of matplotlib contains:

1 Figure

2 Axes

3 Axis

4 Artists

Figure

Figure is the container in which plots are present. It can have a single plot or multiple plots.

Axes

Axes are the plots, and they are artist attached to the figure. A plot can have 2 or 3 axes. set_xlabel(), set_ylabel() are the functions used to set the labels of x and y respectively.

Axis

It is responsible for generating ticks or the limits on the axes.

Artists

The whatever content visible in a figure are the artists.

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:

```
PS C:\Users\harip\Downloads\shadowfox_beginner> pip install matplotlib seaborn numpy pandas
>>
```

Some of the plots in matplotlib:

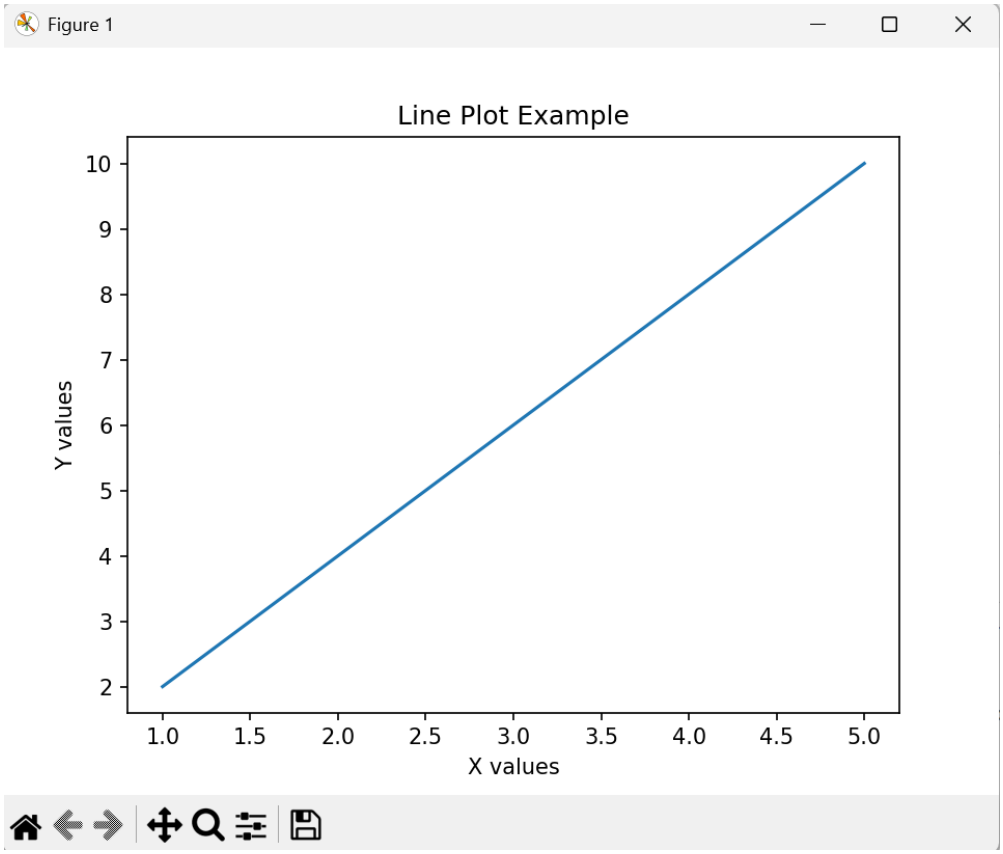
LINE CHART

A line plot represents data points connected by straight lines and is mainly used to show trends or changes in values over a period of time.

Code snippet:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

plt.plot(x, y)
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Line Plot Example")
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

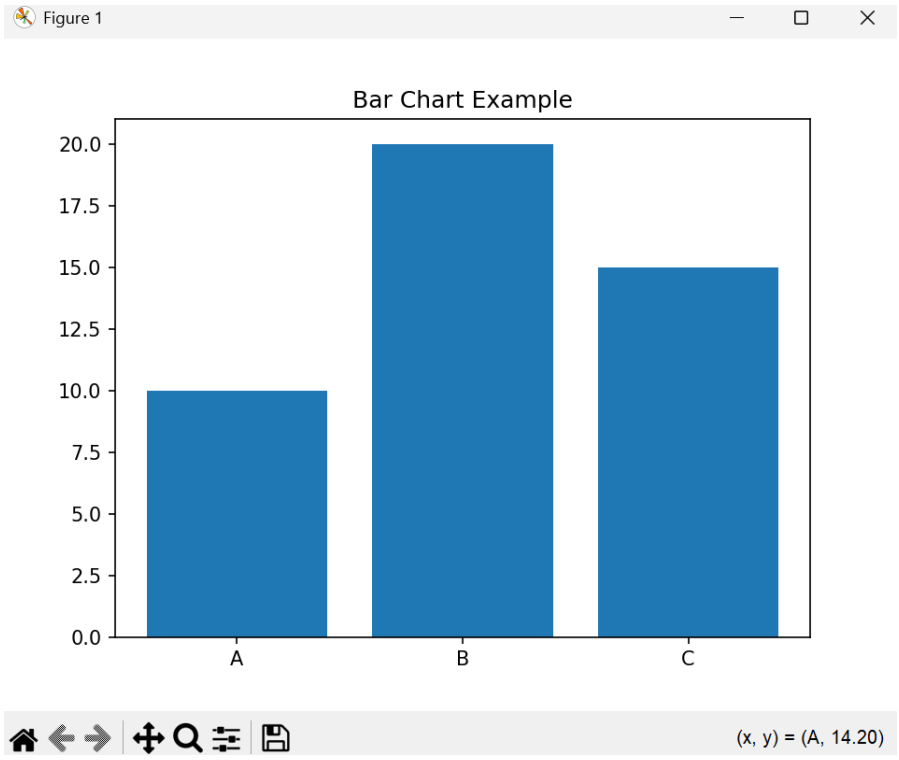
A bar chart displays data using rectangular bars where the height of each bar represents the value of a category, making it easy to compare different groups.

Code snippet:

plots.py > ...

```
1   import matplotlib.pyplot as plt
2   import seaborn as sns
3   import numpy as np
4   import pandas as pd
5   x = np.array([1, 2, 3, 4, 5])
6   y = np.array([2, 4, 6, 8, 10])
7
8   plt.bar(categories, values)
9   plt.title("Bar Chart Example")
10  plt.show()
11  data = np.random.randn(100)
12  plt.hist(data, bins=10)
13  plt.title("Histogram Example")
14  plt.show()
15  x = [1, 2, 3, 4, 5]
16  y = [5, 7, 8, 7, 6]
17
```

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.

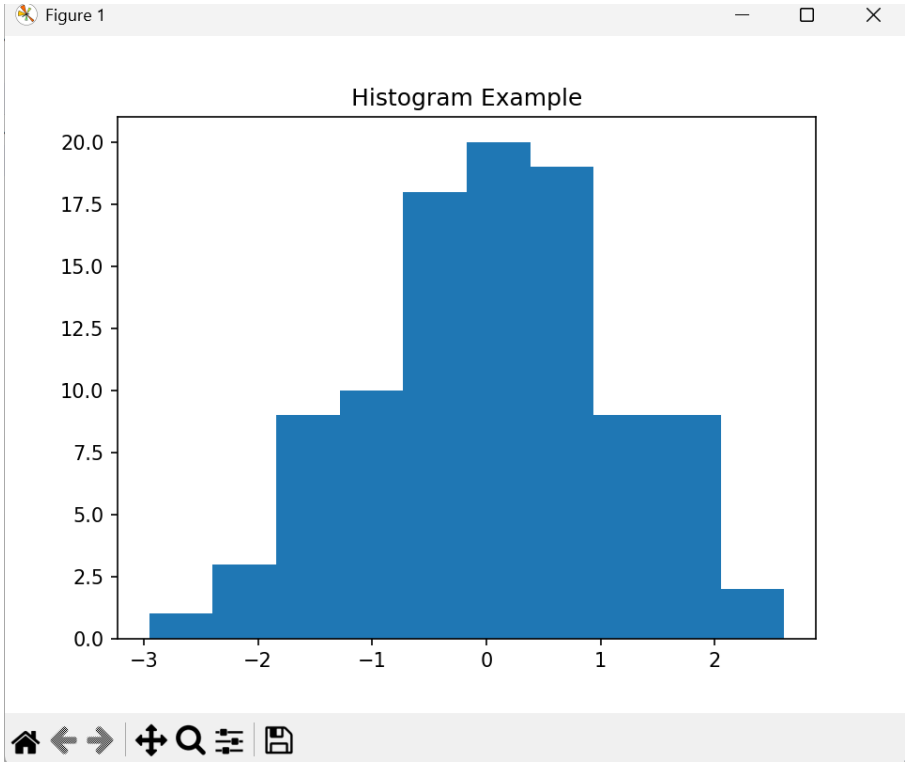
HISTOGRAM

A histogram shows the distribution of numerical data by grouping values into ranges, helping to understand the frequency and spread of the data.

Code snippet:

```
Welcome (preview)
1   import matplotlib.pyplot as plt
2   import seaborn as sns
3   import numpy as np
4   import pandas as pd
5   x = np.array([1, 2, 3, 4, 5])
6   y = np.array([2, 4, 6, 8, 10])
7   plt.title("Histogram Example")
8   plt.show()
9   x = [1, 2, 3, 4, 5]
10  y = [5, 7, 8, 7, 6]
```

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.

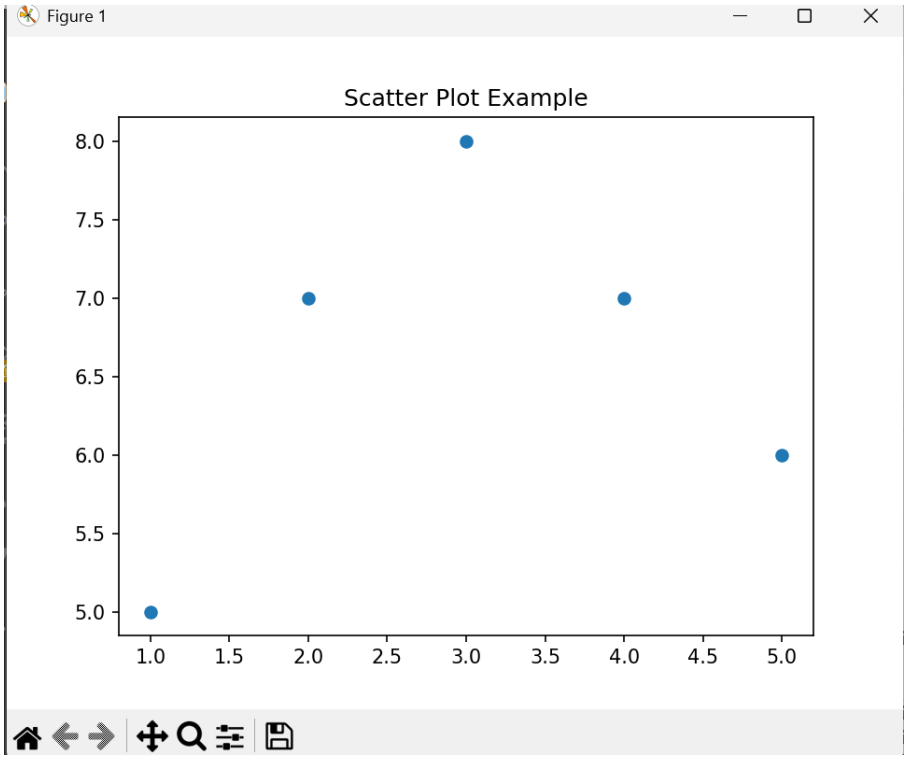
SCATTER PLOT

A scatter plot displays individual data points on a two-dimensional plane and is used to observe relationships or correlations between two variables.

Code snippet:

```
plots.py > ...
1  import matplotlib.pyplot as plt
2  import seaborn as sns
3  import numpy as np
4  import pandas as pd
5  x = np.array([1, 2, 3, 4, 5])
6  y = np.array([2, 4, 6, 8, 10])
7  plt.scatter(x, y)
8  plt.title("Scatter Plot Example")
9  plt.show()
10 sizes = [40, 30, 20, 10]
11 labels = ['A', 'B', 'C', 'D']
```

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.

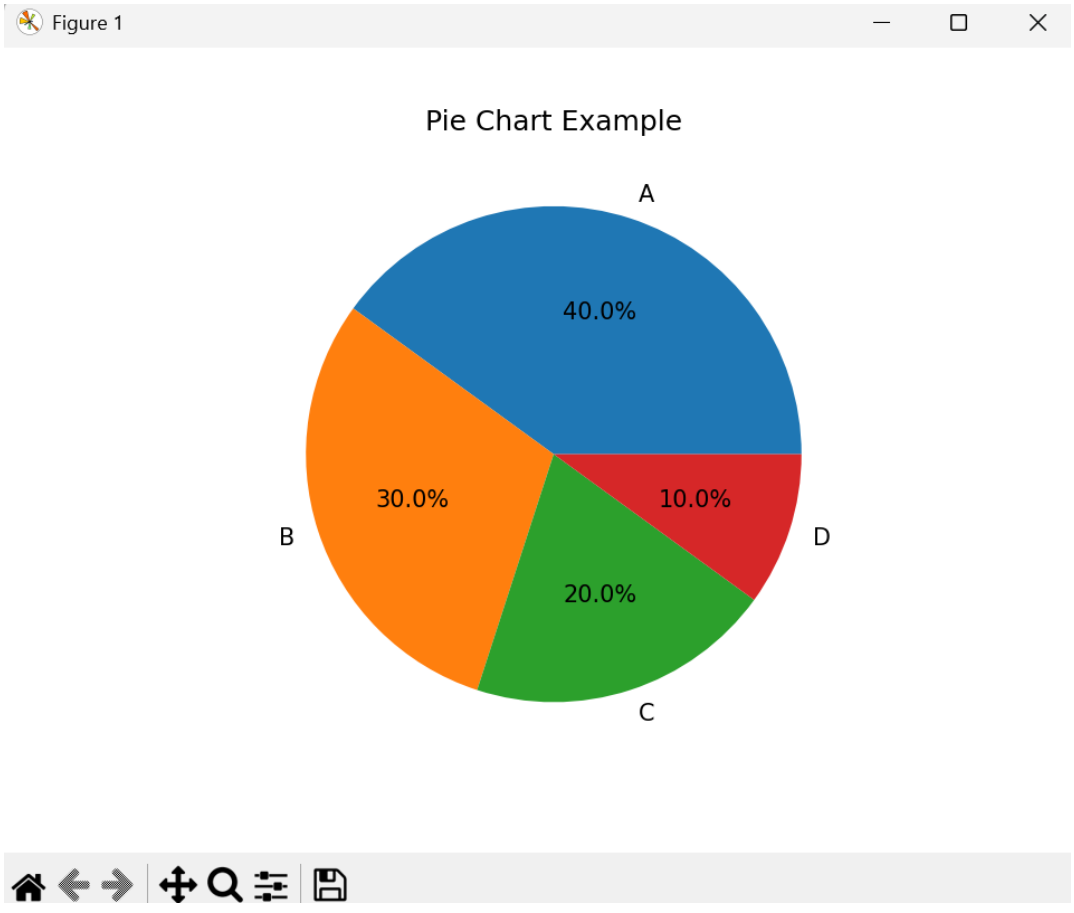
PIECHART

A pie chart represents data as slices of a circle, where each slice shows the proportion or percentage of a category relative to the whole.

Code snippet:

```
plots.py > ...
1   import matplotlib.pyplot as plt
2   import seaborn as sns
3   import numpy as np
4   import pandas as pd
5   x = np.array([1, 2, 3, 4, 5])
6   y = np.array([2, 4, 6, 8, 10])
7   plt.pie(sizes, labels=labels, autopct='%1.1f%%')
8   plt.title("Pie Chart Example")
9   plt.show()
10  tips = sns.load_dataset("tips")
```

Output:



Description:

Here in the code, time is the x variable and labels are defined using labels keyword.

wedgeprops attribute is used to define the linewidth and the color to separate the elements in the pie chart.

color array contains the colors to each element.

AREA CHART

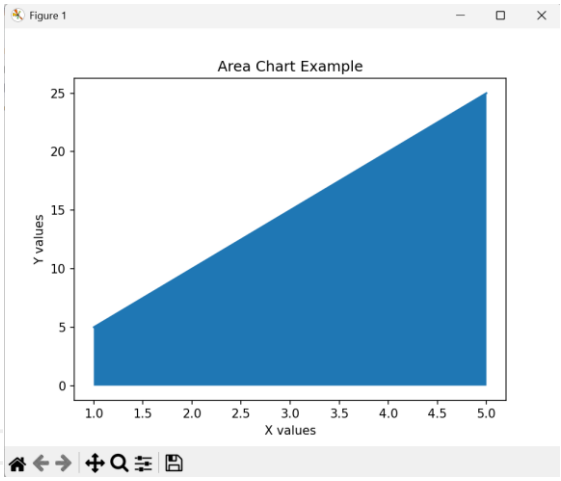
An area chart is similar to a line plot but with the area below the line filled, making it useful for visualizing cumulative values over time.

Code snippet:

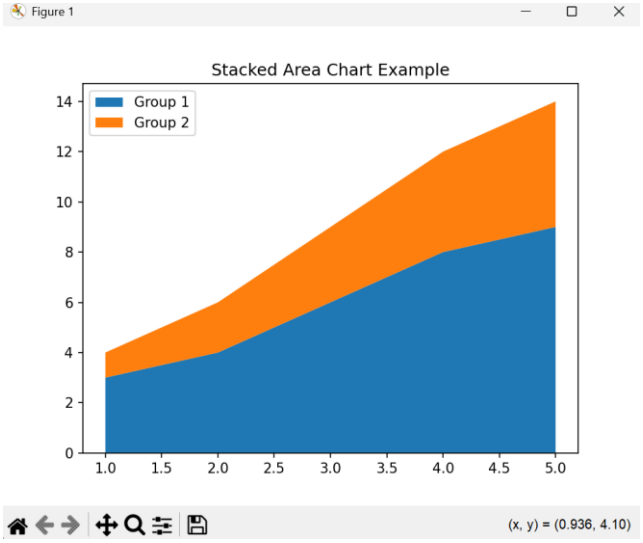
Using fill_between() function

area_chart.py > ...

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

x = np.array([1, 2, 3, 4, 5])
y = np.array([5, 10, 15, 20, 25])

plt.plot(x, y)
plt.fill_between(x, y)
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Area Chart Example")
plt.show()

```



Using stackplot() function



area_chart.py > ...

```python
import matplotlib.pyplot as plt
import numpy as np
x = [1, 2, 3, 4, 5]
y1 = [3, 4, 6, 8, 9]
y2 = [1, 2, 3, 4, 5]

plt.stackplot(x, y1, y2, labels=["Group 1", "Group 2"])
plt.legend(loc="upper left")
plt.title("Stacked Area Chart Example")
plt.show()

```

Description:

fill_between() function have parameters of x and y.

stackplot() function parameters are x, y1,y1,y3 where y1, y2, y3 are the different groups.

Legends are added by mentioning the labels attribute in stackplot() function and giving the position to the legend using plt.legend() function with attribute loc.

SEABORN

Seaborn is a visualization library in python which is built on top of matplotlib. It is advanced than matplotlib and have different features which are default like style, color palette. Seaborn has different categories of plots like relational plot, categorical plot, distribution plot, regression plot, matrix plot.

Let us see the different plots in each category.

1 Relational Plots:

- scatterplot()
- lineplot()
- relplot()

2 Categorical Plots:

- barplot()
- countplot()
- boxplot()
- violinplot()
- swarmplot()
- pointplot()
- catplot()

3 Distribution Plots:

- histplot()
- kdeplot()
- rugplot()
- distplot()

4 Relational Plots:

- regplot()
- lmplot()

5 Matrix Plots:

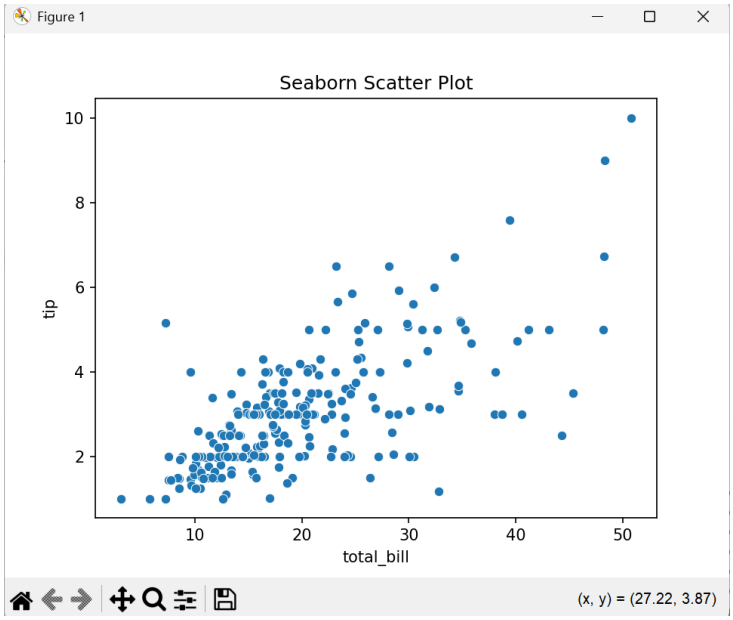
- heatmap()
- clustermap()

Here are some sample codes for some of the graphs.

Scatter plot:

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

# Load sample dataset
tips = sns.load_dataset("tips")
#scatterplot
sns.scatterplot(x="total_bill", y="tip", data=tips)
plt.title("Seaborn Scatter Plot")
plt.savefig("seaborn_scatter.png")
plt.show()
```

Output:



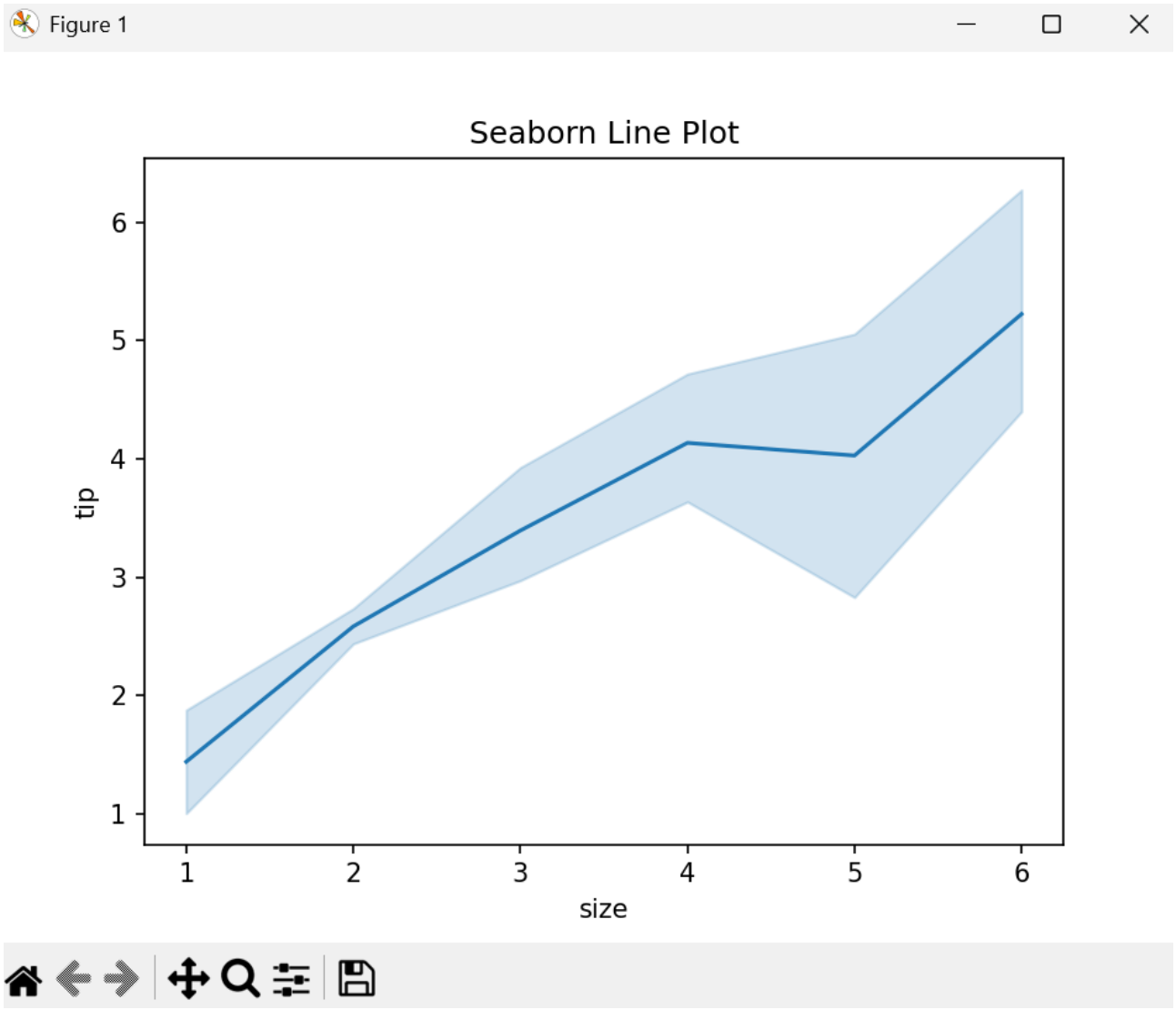
Description:

A Seaborn scatter plot visualizes the relationship between two variables and allows additional grouping using colors for better comparison.

LINE PLOT

```
sns.lineplot(x="size", y="tip", data=tips)
plt.title("Seaborn Line Plot")
plt.savefig("seaborn_line.png")
plt.show()
```

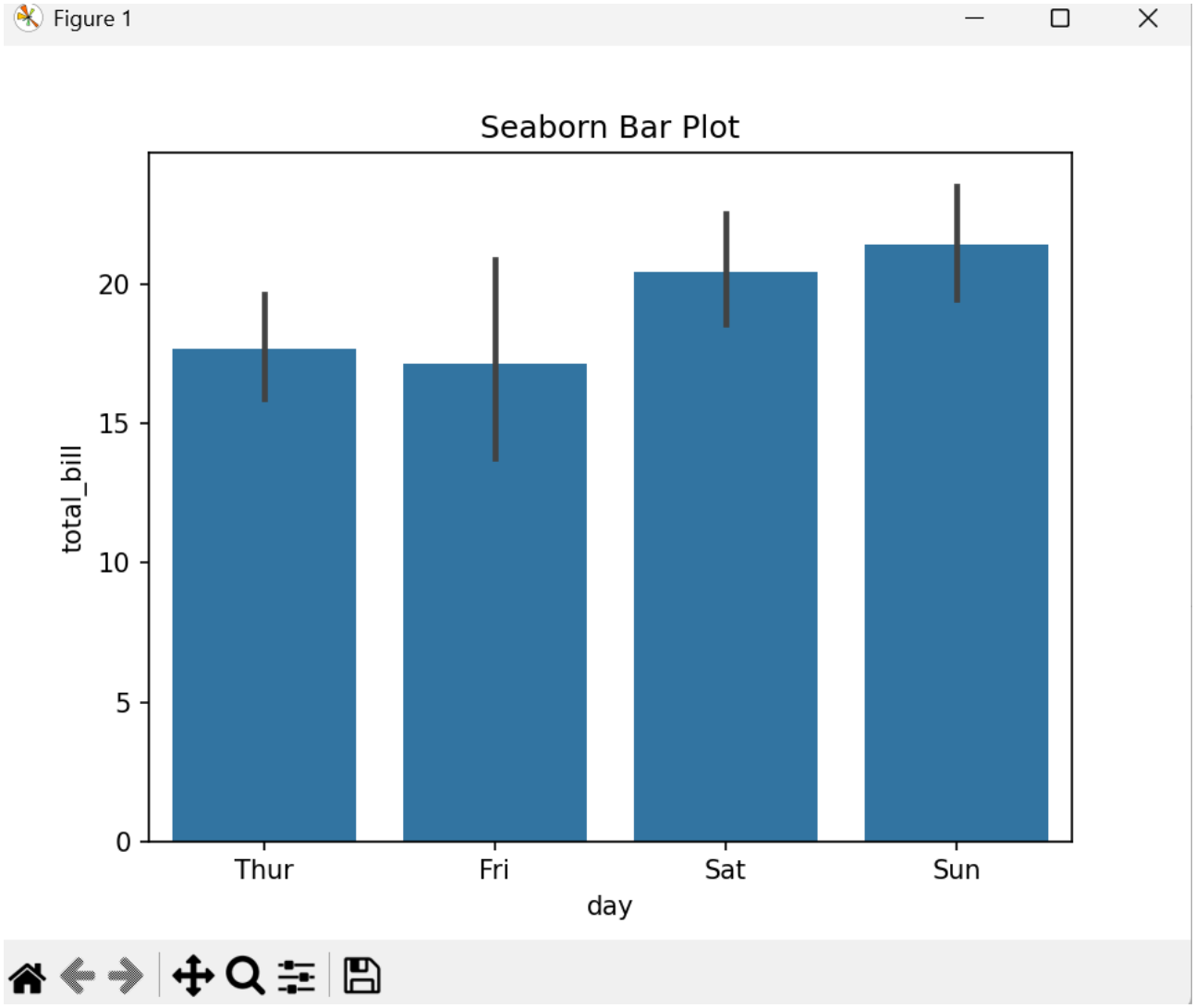
Output:



BAR PLOT

```
sns.barplot(x="day", y="total_bill", data=tips)
plt.title("Seaborn Bar Plot")
plt.savefig("seaborn_bar.png")
plt.show()
```

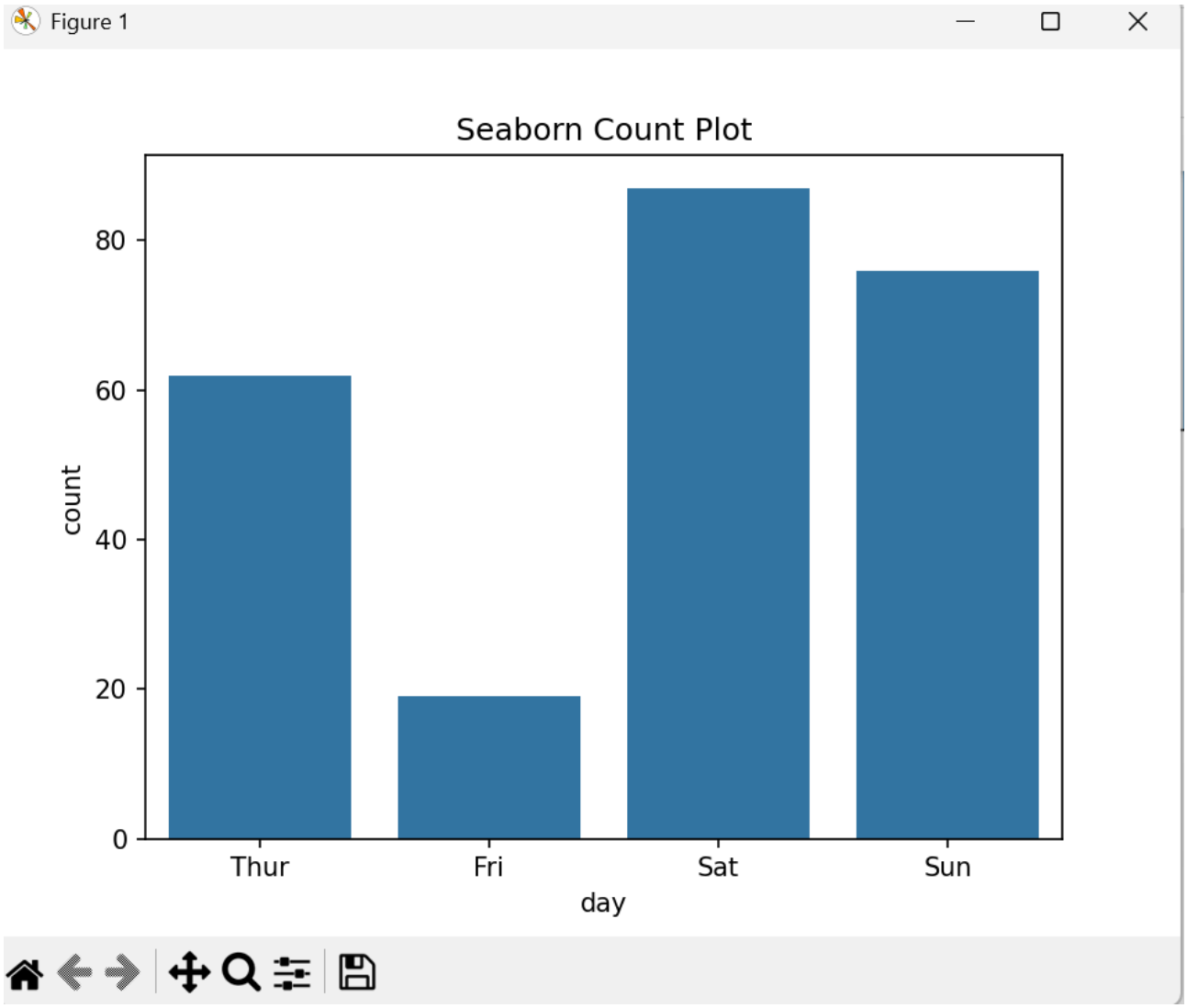
Output:



COUNT PLOT

```
sns.countplot(x="day", data=tips)
plt.title("Seaborn Count Plot")
plt.savefig("seaborn_count.png")
plt.show()
```

Output:



Description:

The count plots give the count of the data passed.

Here x is the species, hue is sex and palette as Set1.

Palette sets the color and it is Set1 is the default palette in seaborn.

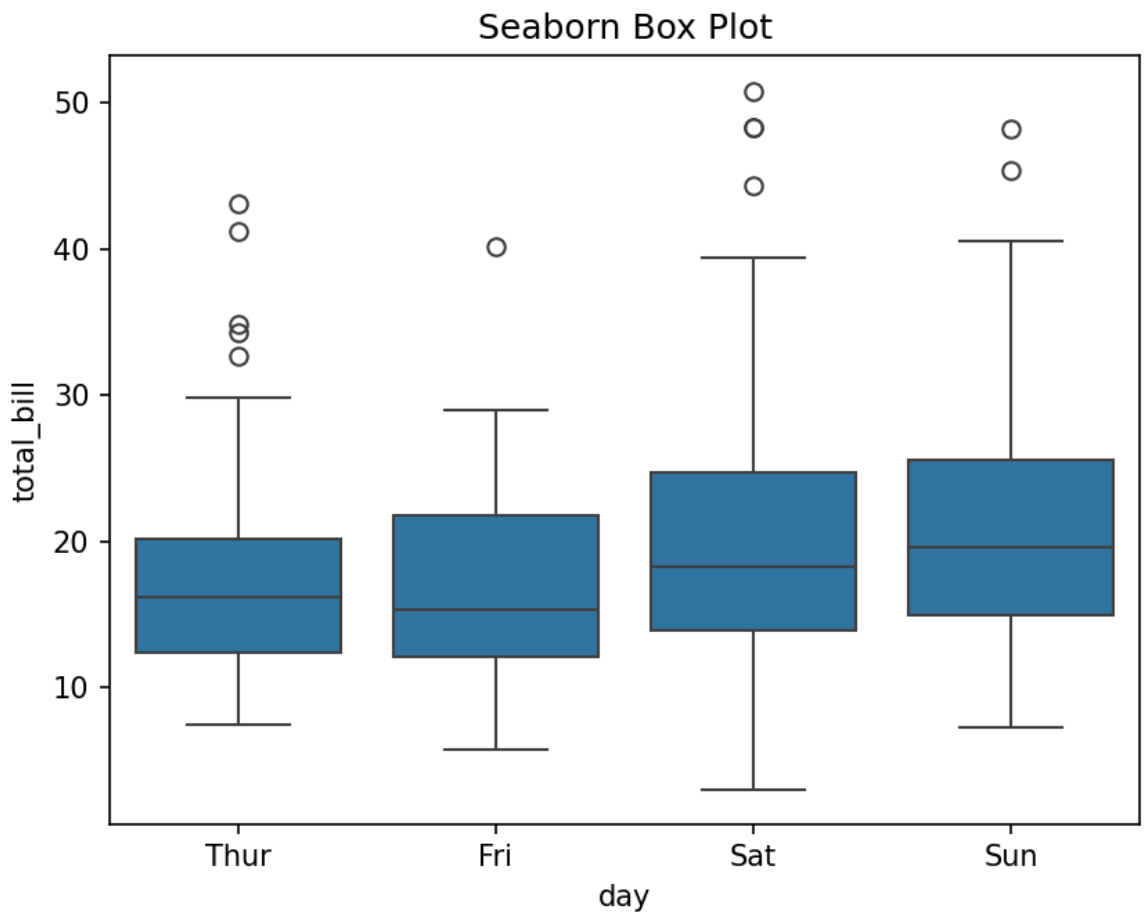
BOX PLOT

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:

```
sns.boxplot(x="day", y="total_bill", data=tips)
plt.title("Seaborn Box Plot")
plt.savefig("seaborn_box.png")
plt.show()
```

Output:



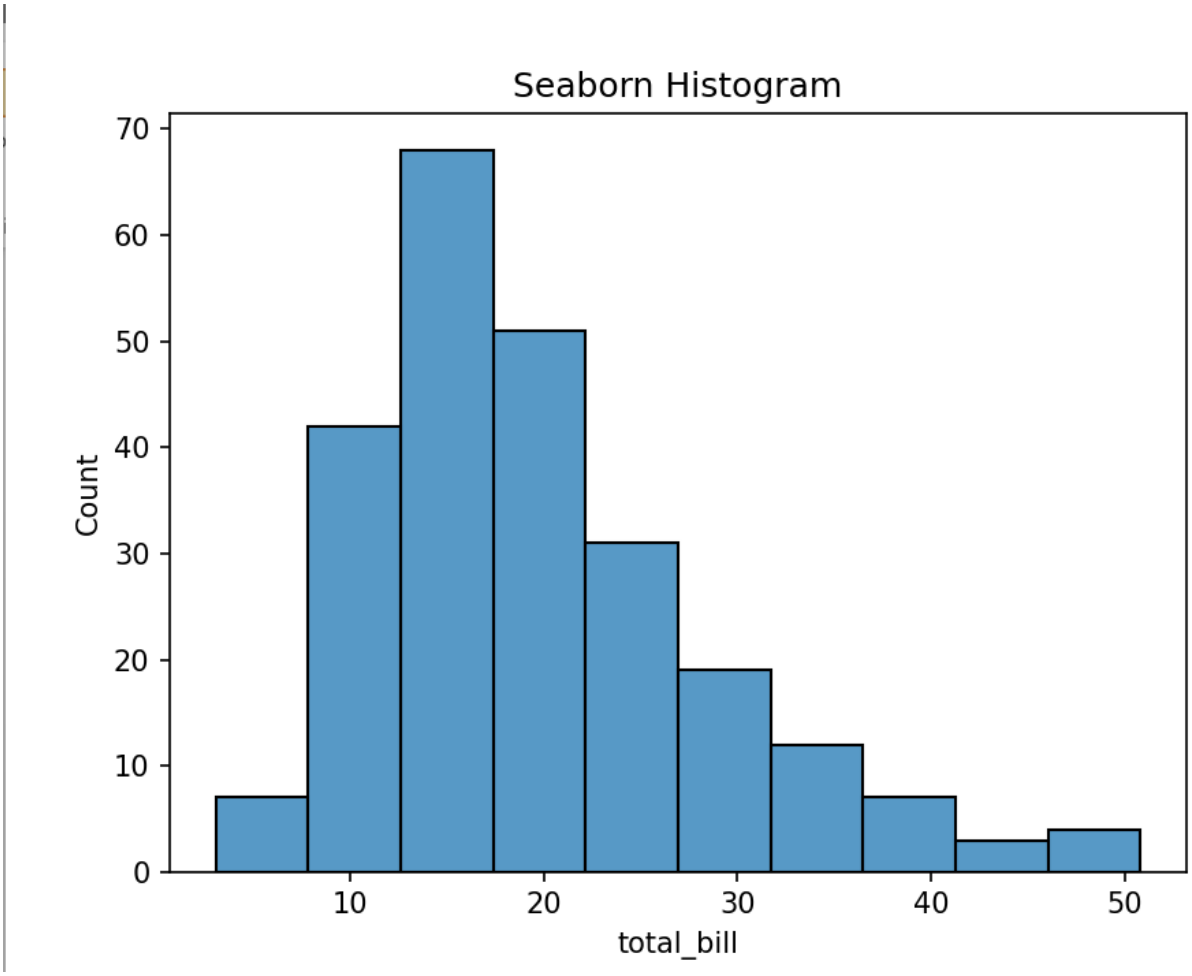
Description:

The parameter x should be numeric as the plot shows the difference between categories.

HISTOPLOT

```
sns.histplot(tips["total_bill"], bins=10)
plt.title("Seaborn Histogram")
plt.savefig("seaborn_histogram.png")
plt.show()
```

Output:



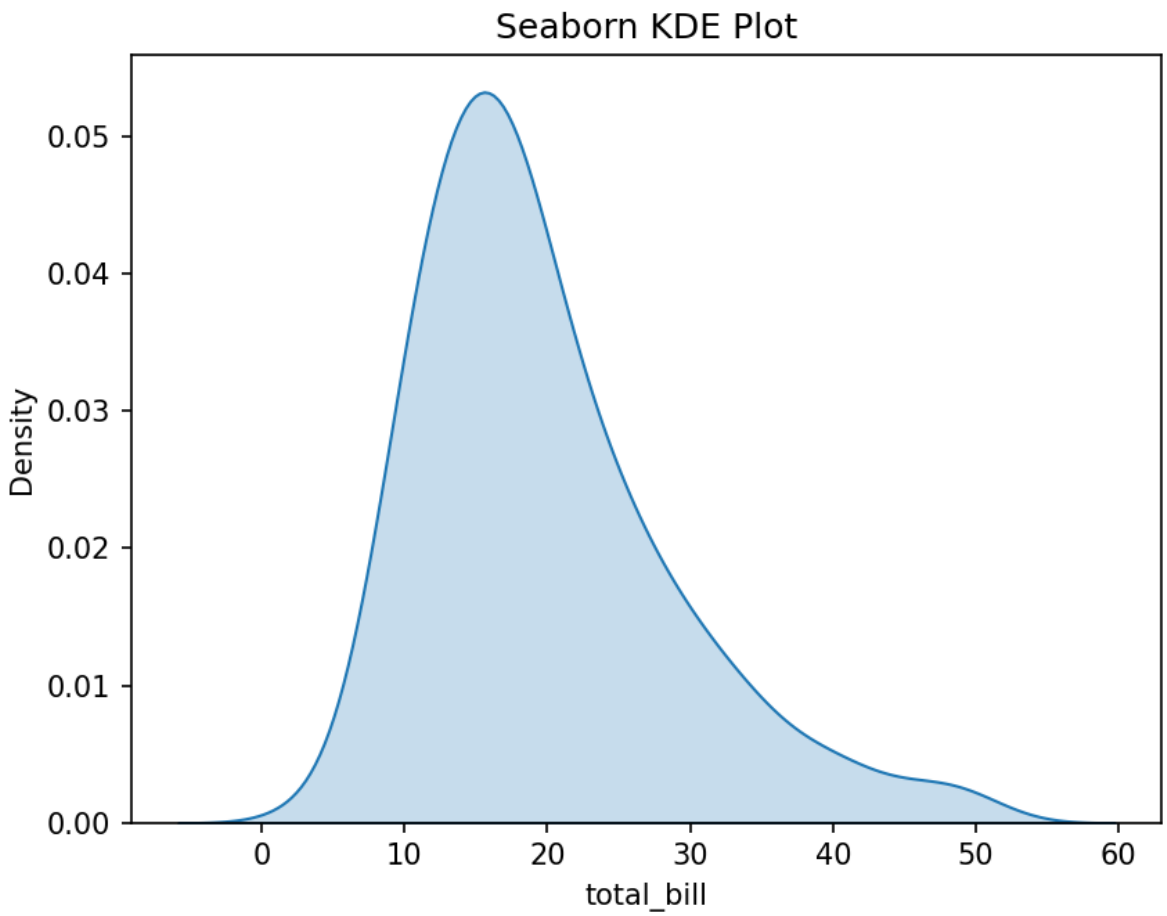
KDEPLOT

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

Code snippet:

```
sns.kdeplot(tips["total_bill"], fill=True)
plt.title("Seaborn KDE Plot")
plt.savefig("seaborn_kde.png")
plt.show()
```

Output:



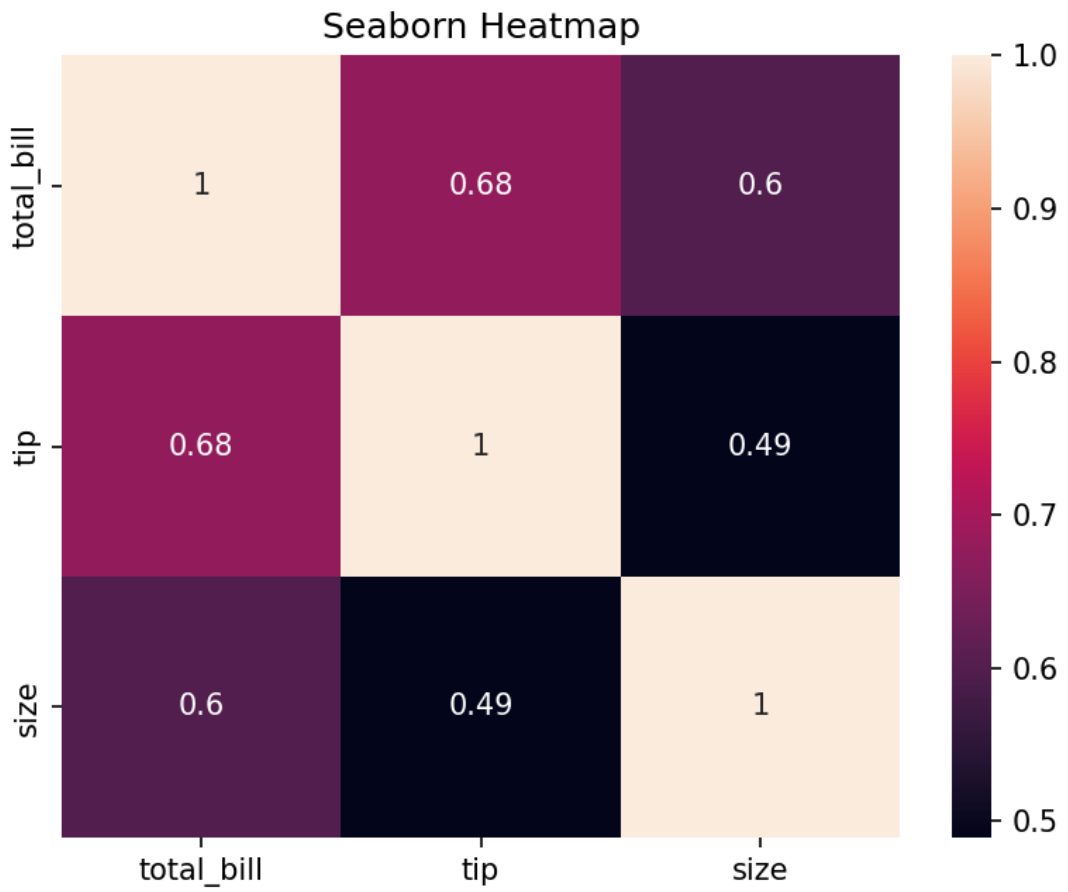
HEATMAP

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

Code snippet:

```
corr = tips.corr(numeric_only=True)
sns.heatmap(corr, annot=True)
plt.title("Seaborn Heatmap")
plt.savefig("seaborn_heatmap.png")
plt.show()
```

Output:



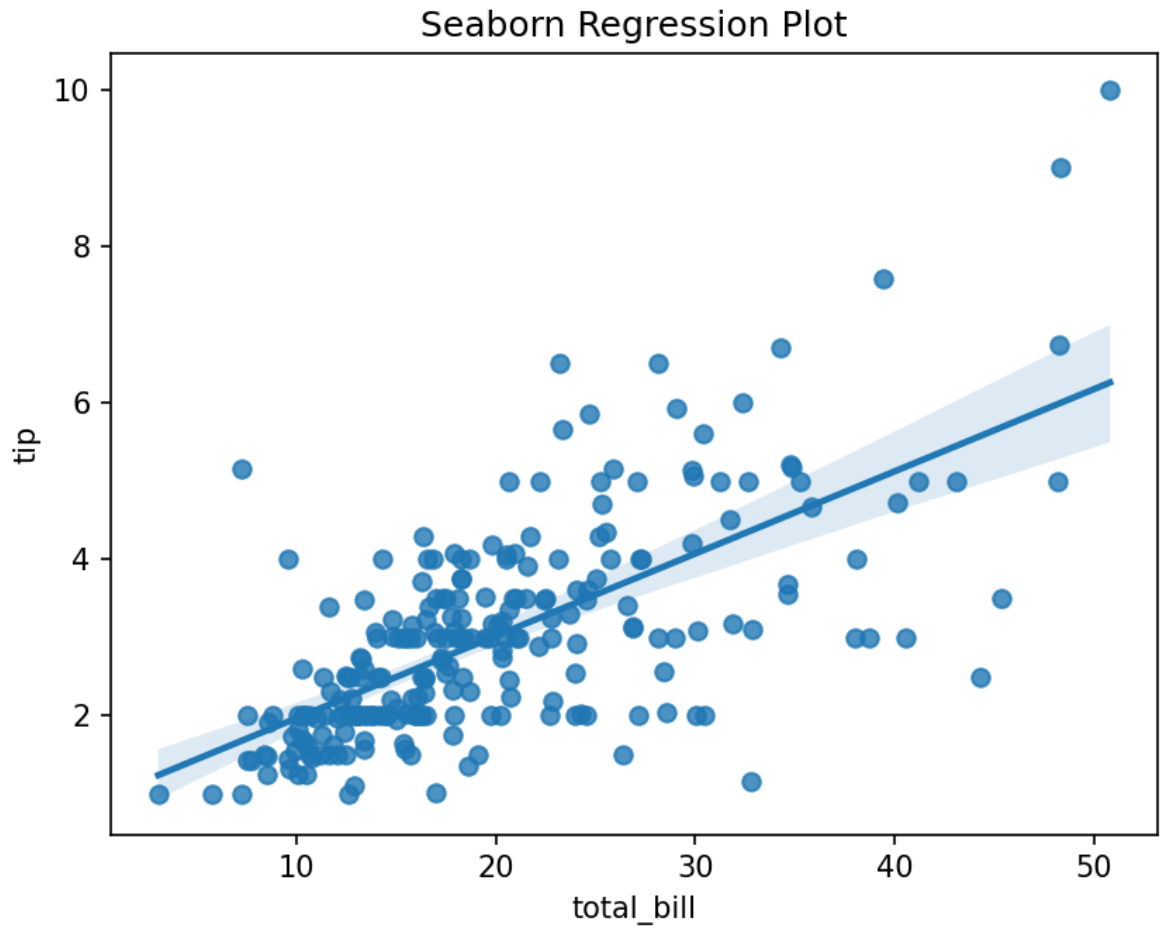
REGPLOT

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

Code snippet:

```
sns.regplot(x="total_bill", y="tip", data=tips)
plt.title("Seaborn Regression Plot")
plt.savefig("seaborn_regplot.png")
plt.show()
```

Output:



COMPARISON OF MATPLOTLIB AND SEABORN

Matplotlib:

- Core Library: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It provides a low-level interface for creating plots with fine-grained control over every aspect of the figure.
- Flexibility: Matplotlib allows users to create any type of plot imaginable, from basic line plots and scatter plots to complex 3D plots and geographical maps.
- Mature: Matplotlib has been around for a long time and is a well-established library in the Python ecosystem. Many other libraries, including Seaborn, are built on top of Matplotlib.
- Customization: Matplotlib offers extensive customization options, allowing users to customize every aspect of a plot, including colors, line styles, markers, fonts, annotations, and more.
- Integration: Matplotlib can be easily integrated with other libraries and frameworks, such as NumPy, Pandas, SciPy, which makes it a versatile choice for data visualization in various contexts.

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.

Seaborn:

- High-Level Interface: Seaborn is built on top of Matplotlib and provides a high-level interface for creating attractive and informative statistical graphics. It simplifies the process of creating complex visualizations by providing default aesthetics and convenient functions for common tasks.
- Statistical Plotting: Seaborn specializes in statistical plotting and offers a variety of plot types specifically designed for visualizing relationships in data, such as scatter plots, bar plots, box plots, violin plots, pair plots, and more.
- Aesthetics: Seaborn comes with built-in themes and color palettes that make it easy to create visually appealing plots with minimal effort. It also provides tools for fine-tuning plot aesthetics, such as controlling the size of plot elements and adjusting the color palette.
- Integration with Pandas: Seaborn seamlessly integrates with Pandas dataframes, allowing users to easily plot data stored in Pandas data structures without the need for extensive data manipulation.

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.

Overall, the choice between Matplotlib and Seaborn depends on the specific requirements of data visualization task. Matplotlib is preferred for its flexibility and fine-grained control over plots, while Seaborn is preferred for its ease of use, statistical plotting capabilities, and attractive default aesthetics. Many users combine both libraries, leveraging the strengths of each as needed.