SHADOW FOX

NAME: AKSHAYA RAO

GMAIL:akshayarao1100@gmail.com

**TASK LEVEL (BEGINNER):**

**OBJECTIVE**: Create a comprehensive documentation guide for 2 of the Python visualization Libraries : Matplotlib , Seaborn .

**1)LIBRARY OVERVIEW:**

**i)MATPLOTLIB**

Matplotlib is a popular Python library used for creating static, interactive, and animated visualizations. It is especially useful for plotting data in various chart formats like line graphs, bar charts, histograms, and scatter plots. Matplotlib is highly customizable and is widely used in data analysis and machine learning for visualizing trends and patterns in data.

**Unique Features:**

1. **Wide Range of Plots**: Supports diverse plot types including line plots, bar charts, scatter plots, histograms, pie charts, and more.
2. **Customization**: Every aspect of a plot, from axes to grid lines, colors, and markers, can be customized for precision.
3. **Integration**: Works seamlessly with other libraries like NumPy, Pandas, and Seaborn for enhanced data manipulation and visualization.
4. **Interactive Visualizations**: Offers support for zooming, panning, and tooltips in Jupyter notebooks and other interactive environments.
5. **Publication-Quality Graphics**: Produces highly detailed and attractive visualizations, suitable for research publications or professional reports.

**Typical Use Cases:**

1. **Data Exploration and Analysis**: Visualizing trends, distributions, and relationships in data during exploratory analysis.
2. **Machine Learning**: Plotting training metrics such as loss curves, accuracy trends, and confusion matrices.
3. **Statistical Analysis**: Creating histograms and box plots to understand data distributions and variability.
4. **Business Reporting**: Building static or interactive charts to communicate insights in presentations, reports, or dashboards.
5. **Scientific Research**: Producing figures for academic papers and research publications, where clarity and accuracy are critical.

**ii)SAEBORN**:

Seaborn is a powerful Python library built on top of Matplotlib, designed specifically for making statistical data visualization easier and more appealing. It provides a high-level interface for creating informative and attractive visualizations, making it popular among data scientists for exploring and presenting complex datasets. Seaborn works seamlessly with Pandas data structures and simplifies the creation of beautiful charts.

**Unique Features:**

1. **Statistical Plots**: Specializes in statistical visualizations, including distribution plots, regression plots, and heatmaps.
2. **Built-in Aesthetics**: Offers better default styling and color palettes compared to Matplotlib, producing more attractive visualizations with less effort.
3. **Integration with Pandas**: Directly works with DataFrames, making it easy to visualize data without extensive preprocessing.
4. **Faceting**: Supports "faceted" plots, where multiple subplots are created automatically for different subsets of the data.
5. **Easy Plotting for Complex Data**: Automatically computes and plots complex relationships like pairwise comparisons, kernel density estimates, and categorical data visualizations.

**Typical Use Cases:**

1. **Data Exploration**: Visualizing distributions of variables (e.g., histograms, kernel density plots) and relationships between them (e.g., scatter plots, regression lines).
2. **Correlation Heatmaps**: Plotting heatmaps to show relationships and correlations between variables, often used in feature selection.
3. **Comparative Analysis**: Using facet grids or pair plots to compare multiple subgroups within a dataset simultaneously.
4. **Trend and Regression Analysis**: Visualizing linear relationships between variables with regression plots.
5. **Presentation-Ready Plots**: Creating attractive, publication-quality plots for research papers, reports, and presentations.

**2)GRAPH TYPES**:

**i)Matplotlib:**

In Matplotlib, pairwise plots and statistical distribution graphs are commonly used for visualizing relationships between multiple variables and understanding the distribution of data.

### ****Pairwise Plots:****

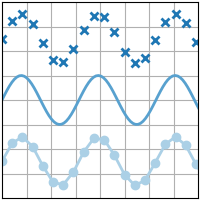
Pairwise plots (also called scatterplot matrices) are useful for exploring relationships between pairs of variables in a dataset. Each scatter plot represents a relationship between two variables, and by visualizing multiple combinations, users can identify correlations, trends, or clusters across multiple dimensions. In Matplotlib, pairwise plots can be created using **subplots** and **scatter** functions, but libraries like Seaborn (built on top of Matplotlib) offer functions like pairplot() for simplified creation of pairwise plots.

**1)Plot(x,y):**

 **Definition**: Creates a simple line plot of x and y data points.

 **Use**: Ideal for visualizing trends over a continuous range, such as time-series data or function plots.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery')  # make data  x = np.linspace(0, 10, 100)  y = 4 + 1 \* np.sin(2 \* x)  x2 = np.linspace(0, 10, 25)  y2 = 4 + 1 \* np.sin(2 \* x2)  # plot  fig, ax = plt.subplots()  ax.plot(x2, y2 + 2.5, 'x', markeredgewidth=2)  ax.plot(x, y, linewidth=2.0)  ax.plot(x2, y2 - 2.5, 'o-', linewidth=2)  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 8), yticks=np.arange(1, 8))  plt.show() |



# 2) scatter(x, y)

A scatter plot of y vs. x with varying marker size and/or color.

 **Definition**: Generates a scatter plot to show individual data points.

 **Use**: Useful for showing relationships and correlations between two variables by plotting points for each observation.

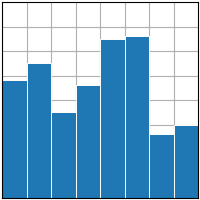
|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery')  # make the data  np.random.seed(3)  x = 4 + np.random.normal(0, 2, 24)  y = 4 + np.random.normal(0, 2, len(x))  # size and color:  sizes = np.random.uniform(15, 80, len(x))  colors = np.random.uniform(15, 80, len(x))  # plot  fig, ax = plt.subplots()  ax.scatter(x, y, s=sizes, c=colors, vmin=0, vmax=100)  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 8), yticks=np.arange(1, 8))  plt.show() |

**3) bar(x, height):**

 **Definition**: Creates a bar chart, where each bar's height corresponds to a value.

 **Use**: Used to compare quantities among different categories or show frequency counts.

|  |
| --- |
| **import matplotlib.pyplot as plt**  **import numpy as np**  **plt.style.use('\_mpl-gallery')**  **# make data:**  **x = 0.5 + np.arange(8)**  **y = [4.8, 5.5, 3.5, 4.6, 6.5, 6.6, 2.6, 3.0]**  **# plot**  **fig, ax = plt.subplots()**  **ax.bar(x, y, width=1, edgecolor="white", linewidth=0.7)**  **ax.set(xlim=(0, 8), xticks=np.arange(1, 8),**  **ylim=(0, 8), yticks=np.arange(1, 8))**  **plt.show()** |

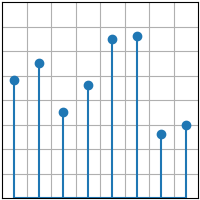
****

# 4) stem(x, y):

 **Definition**: Generates a stem plot, where lines (stems) extend from a baseline to points (markers).

 **Use**: Suitable for highlighting individual data points and their exact values in discrete data sets.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery')  # make data  x = 0.5 + np.arange(8)  y = [4.8, 5.5, 3.5, 4.6, 6.5, 6.6, 2.6, 3.0]  # plot  fig, ax = plt.subplots()  ax.stem(x, y)  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 8), yticks=np.arange(1, 8))  plt.show() |



# 5)fill\_between(x, y1, y2)

*  **Definition**: Fills the area between two horizontal curves or between a curve and the x-axis.
* **Use**: Helpful for showing areas under curves, error bands, or ranges of values over an interval.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery')  # make data  np.random.seed(1)  x = np.linspace(0, 8, 16)  y1 = 3 + 4\*x/8 + np.random.uniform(0.0, 0.5, len(x))  y2 = 1 + 2\*x/8 + np.random.uniform(0.0, 0.5, len(x))  # plot  fig, ax = plt.subplots()  ax.fill\_between(x, y1, y2, alpha=.5, linewidth=0)  ax.plot(x, (y1 + y2)/2, linewidth=2)  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 8), yticks=np.arange(1, 8))  plt.show() |

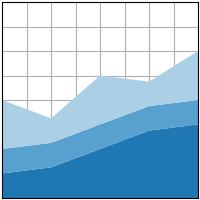


# 6) stackplot(x, y)

 **Definition**: Creates a stacked plot, showing different layers stacked on top of each other.

 **Use**: Ideal for visualizing part-to-whole relationships over time or comparing contributions from different categories.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery')  # make data  x = np.arange(0, 10, 2)  ay = [1, 1.25, 2, 2.75, 3]  by = [1, 1, 1, 1, 1]  cy = [2, 1, 2, 1, 2]  y = np.vstack([ay, by, cy])  # plot  fig, ax = plt.subplots()  ax.stackplot(x, y)  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 8), yticks=np.arange(1, 8))  plt.show() |

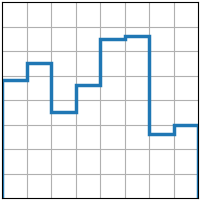


# 7) stairs(values)

 **Definition**: Produces a step plot, where data points are connected by horizontal and vertical lines, resembling stairs.

 **Use**: Useful for plotting discrete events or values that change at specific intervals, such as digital signals.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery')  # make data  y = [4.8, 5.5, 3.5, 4.6, 6.5, 6.6, 2.6, 3.0]  # plot  fig, ax = plt.subplots()  ax.stairs(y, linewidth=2.5)  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 8), yticks=np.arange(1, 8))  plt.show() |



**Statistical Distribution Graphs:**

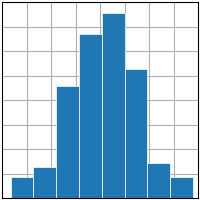
Statistical distribution graphs help in visualizing the distribution of data for a single variable or multiple variables. Some common types of distribution plots in Matplotlib include:

* **Histograms**: Used to visualize the frequency distribution of a single variable.
* **Box Plots**: Summarize data distribution by showing the median, quartiles, and potential outliers.
* **Violin Plots**: Combine box plots with kernel density estimates to show both the summary statistics and the distribution shape.
* **KDE (Kernel Density Estimation) Plots**: Provide a smoothed estimate of the distribution of a variable, making it easier to see the overall data trend.

# hist(x)

 **Definition**: Creates a histogram to show the frequency distribution of a dataset.

 **Use**: Commonly used to visualize the distribution of data, highlighting how often values occur in specified intervals (bins).

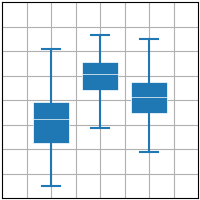


|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery')  # make data  np.random.seed(1)  x = 4 + np.random.normal(0, 1.5, 200)  # plot:  fig, ax = plt.subplots()  ax.hist(x, bins=8, linewidth=0.5, edgecolor="white")  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 56), yticks=np.linspace(0, 56, 9))  plt.show() |

# 2)boxplot(X)

 **Definition**: Generates a box plot (also known as a whisker plot) that summarizes data through its quartiles.

 **Use**: Used to visualize the spread of data, showing the median, quartiles, and potential outliers.

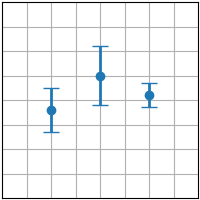


|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery')  # make data:  np.random.seed(10)  D = np.random.normal((3, 5, 4), (1.25, 1.00, 1.25), (100, 3))  # plot  fig, ax = plt.subplots()  VP = ax.boxplot(D, positions=[2, 4, 6], widths=1.5, patch\_artist=True,  showmeans=False, showfliers=False,  medianprops={"color": "white", "linewidth": 0.5},  boxprops={"facecolor": "C0", "edgecolor": "white",  "linewidth": 0.5},  whiskerprops={"color": "C0", "linewidth": 1.5},  capprops={"color": "C0", "linewidth": 1.5})  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 8), yticks=np.arange(1, 8))  plt.show() |

# 3)errorbar(x, y, yerr, xerr)

*  **Definition**: Plots data points with vertical and/or horizontal error bars.
* **Use**: Useful for representing the uncertainty or variability in data points, commonly used in experimental data to indicate measurement error.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery')  # make data:  np.random.seed(1)  x = [2, 4, 6]  y = [3.6, 5, 4.2]  yerr = [0.9, 1.2, 0.5]  # plot:  fig, ax = plt.subplots()  ax.errorbar(x, y, yerr, fmt='o', linewidth=2, capsize=6)  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 8), yticks=np.arange(1, 8))  plt.show() |

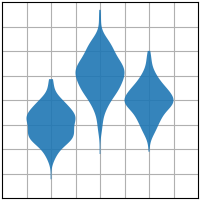


# 4) violinplot(D)

 **Definition**: Combines box plots with kernel density estimates, showing the distribution and probability density of data.

 **Use**: Ideal for visualizing both the summary statistics and the full distribution shape, especially in comparing multiple groups.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery')  # make data:  np.random.seed(10)  D = np.random.normal((3, 5, 4), (0.75, 1.00, 0.75), (200, 3))  # plot:  fig, ax = plt.subplots()  vp = ax.violinplot(D, [2, 4, 6], widths=2,  showmeans=False, showmedians=False, showextrema=False)  # styling:  for body in vp['bodies']:  body.set\_alpha(0.9)  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 8), yticks=np.arange(1, 8))  plt.show() |

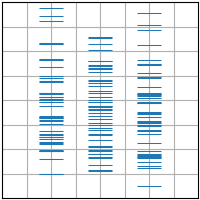


# 5)eventplot(D):

 **Definition**: Visualizes events occurring along a continuous axis, with lines drawn at each event point.

 **Use**: Used for visualizing occurrences of events, like spike trains in neuroscience or timelines in event-based data.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery')  # make data:  np.random.seed(1)  x = [2, 4, 6]  D = np.random.gamma(4, size=(3, 50))  # plot:  fig, ax = plt.subplots()  ax.eventplot(D, orientation="vertical", lineoffsets=x, linewidth=0.75)  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 8), yticks=np.arange(1, 8))  plt.show() |



# 6) pie(x)

 **Definition**: Creates a pie chart where slices represent proportions of a whole.

 **Use**: Commonly used to show relative sizes of different categories or percentages in a dataset.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  plt.style.use('\_mpl-gallery-nogrid')  # make data  x = [1, 2, 3, 4]  colors = plt.get\_cmap('Blues')(np.linspace(0.2, 0.7, len(x)))  # plot  fig, ax = plt.subplots()  ax.pie(x, colors=colors, radius=3, center=(4, 4),  wedgeprops={"linewidth": 1, "edgecolor": "white"}, frame=True)  ax.set(xlim=(0, 8), xticks=np.arange(1, 8),  ylim=(0, 8), yticks=np.arange(1, 8))  plt.show() |



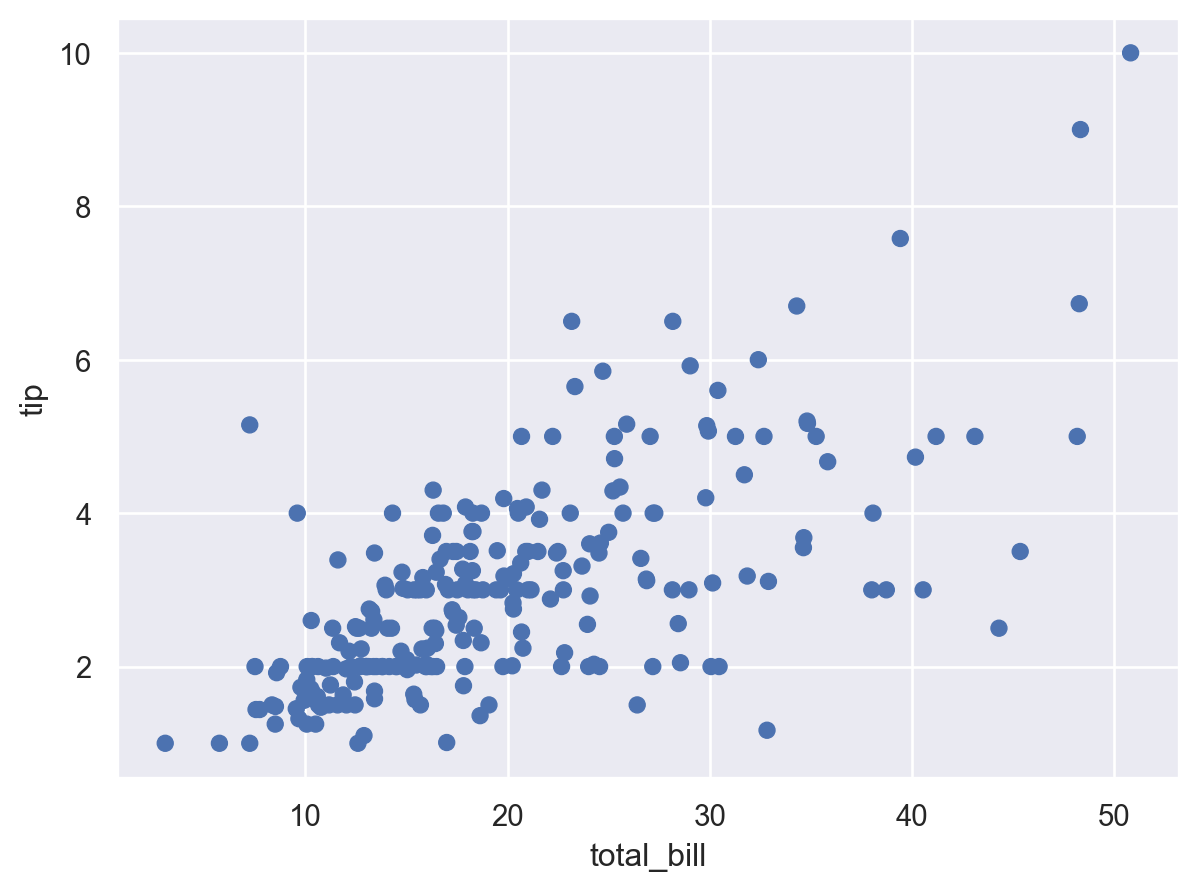
**ii)SEABORN:**

**1)Scatter Plot:**

 **Definition**: A scatter plot displays points for two continuous variables (x and y) to visualize their relationship.

 **Use Case**: Ideal for exploring potential correlations or relationships between two variables. For example, in the tips dataset, you can use a scatter plot to show how total\_bill correlates with tip.

|  |
| --- |
| p = so.Plot(tips, "total\_bill", "tip").add(so.Dot())  p |

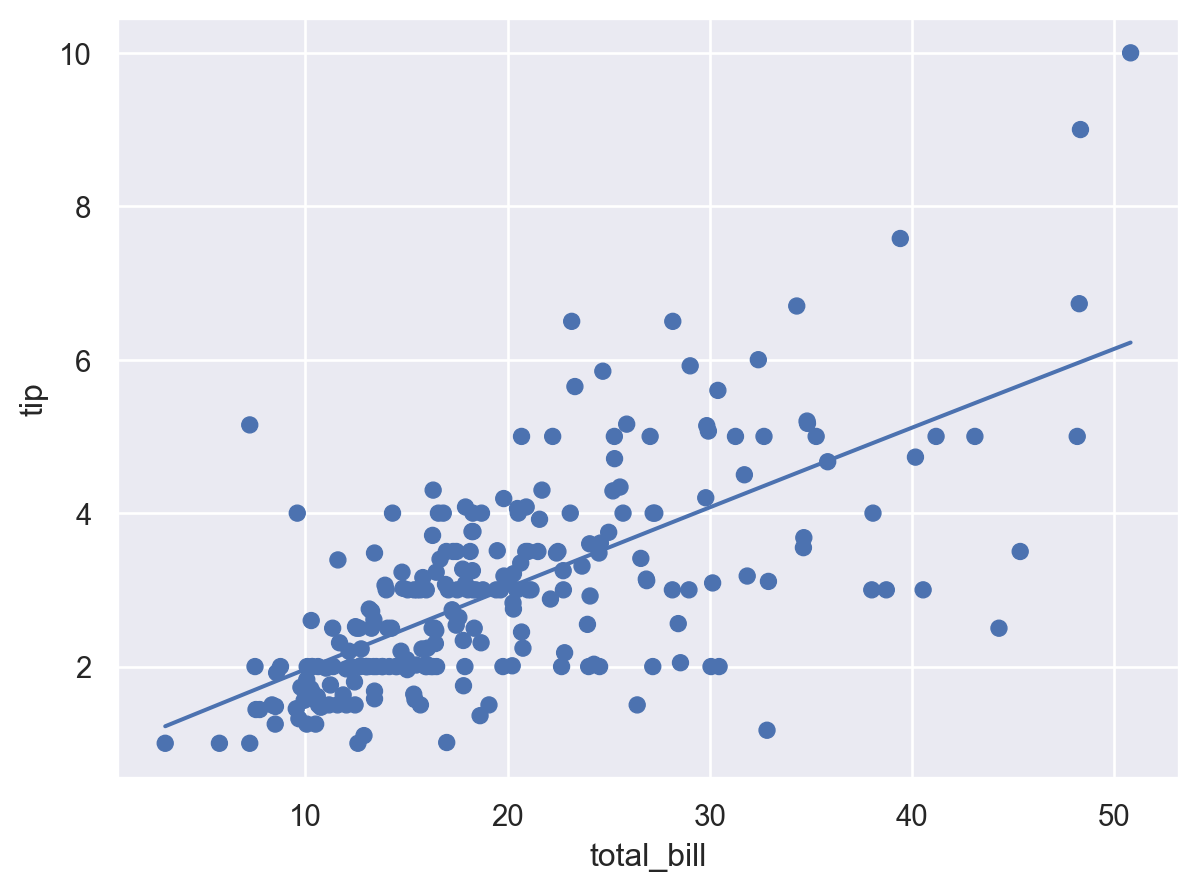
****

**2)The Combined Plot:**

 **Definition**: A combined plot layers multiple plot types on the same graph, such as combining scatter and line plots or scatter with a regression line.

 **Use Case**: Useful for visualizing both raw data points and underlying trends or patterns (e.g., adding a regression line to a scatter plot).

|  |
| --- |
| **p.add(so.Line(), so.PolyFit())** |

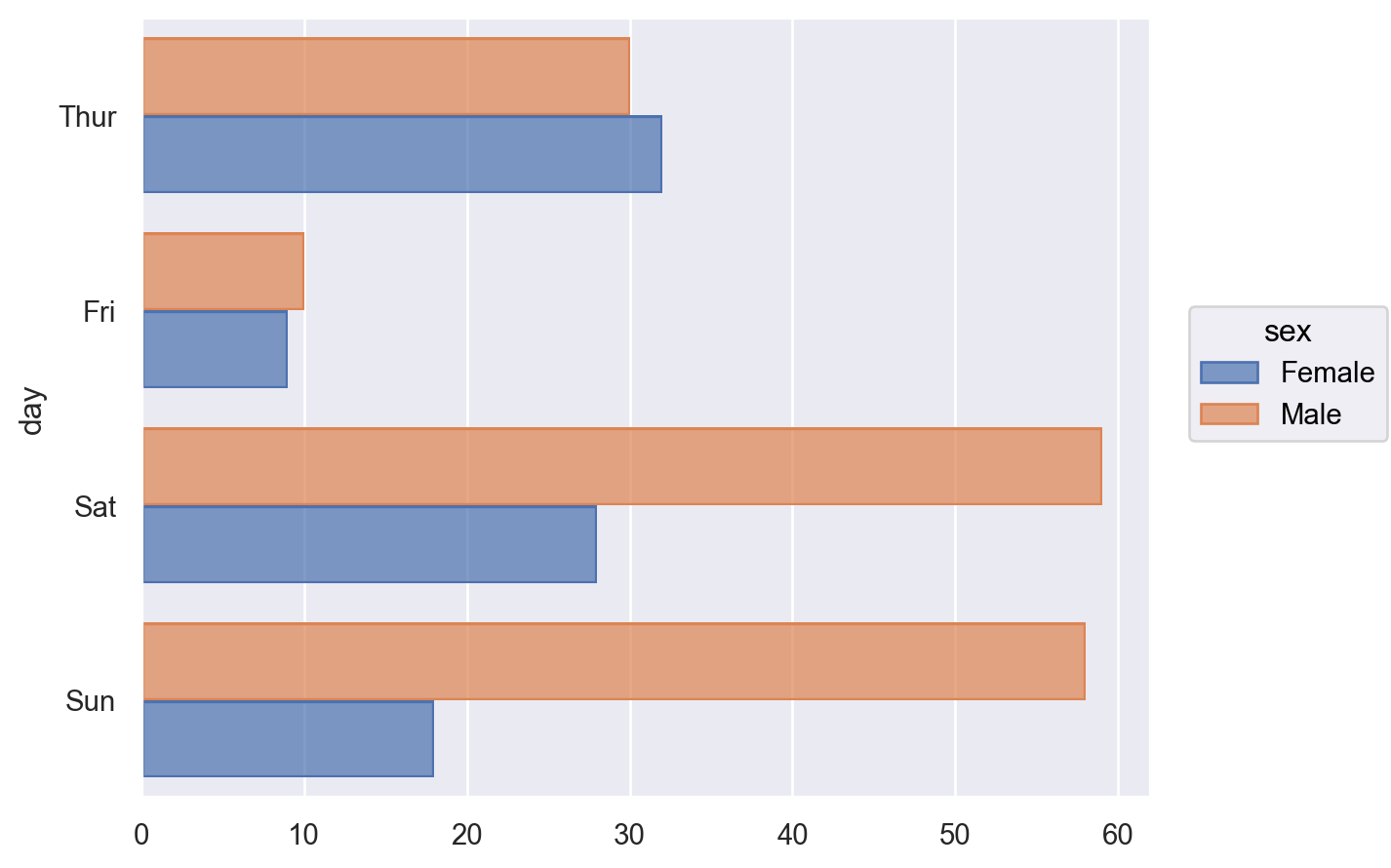


**3)dodged bar plot:**

 **Definition**: A bar plot that groups data based on a categorical variable, and positions (or "dodges") bars side-by-side for comparison.

 **Use Case**: Useful for comparing groups within a categorical variable across categories. For example, comparing the average tip amount between male and female customers for each day of the week.

|  |
| --- |
| **(**  **so.Plot(tips, y="day", color="sex")**  **.add(so.Bar(), so.Hist(), so.Dodge())**  **)** |

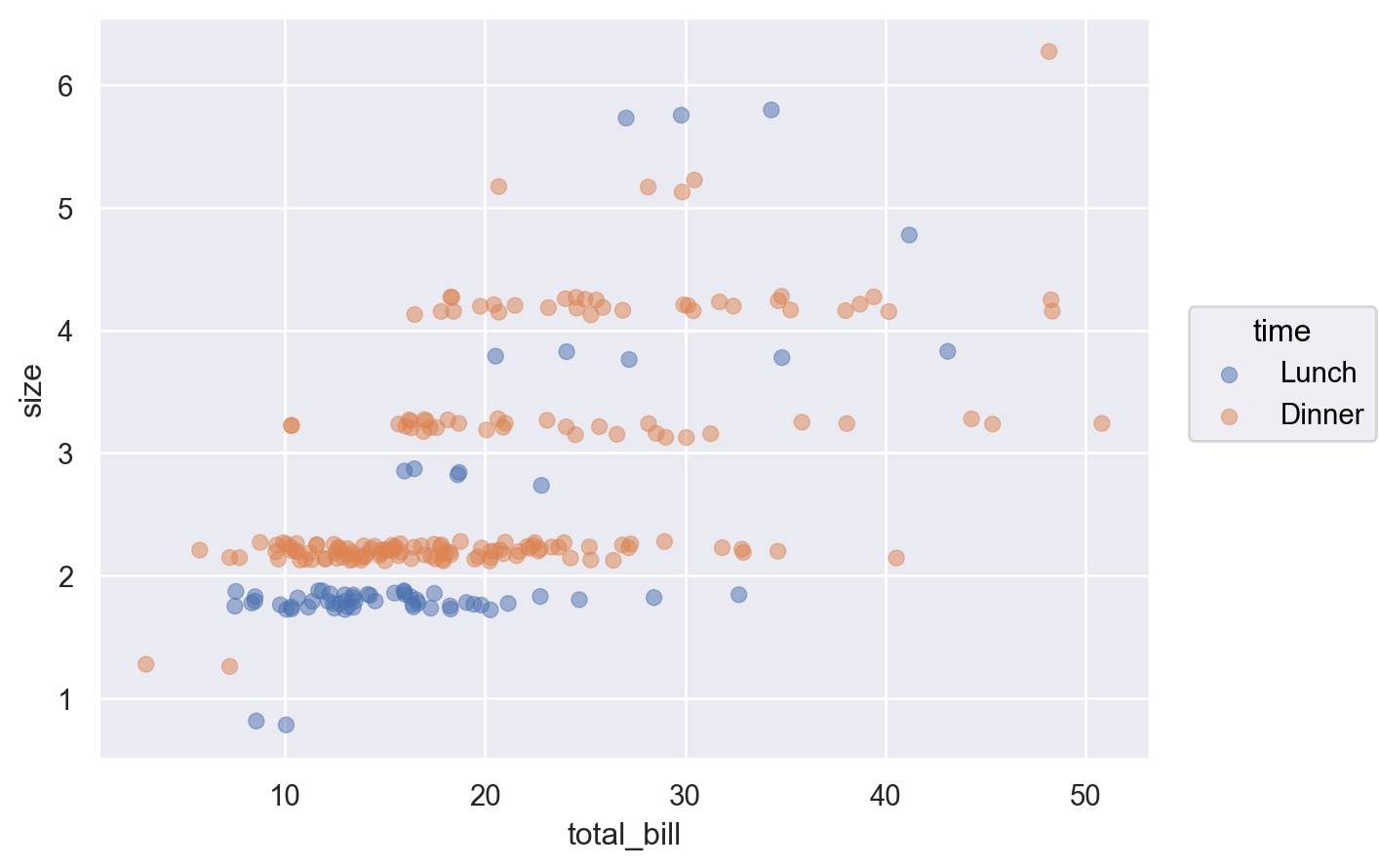


**4)dodge-jittered scatter plot:**

 **Definition**: A scatter plot that adds both dodging (for categorical separation) and jittering (random noise to reduce overplotting) to help visualize points that might overlap.

 **Use Case**: Used when plotting points for a continuous variable with respect to a categorical variable to better distinguish data points that might otherwise stack up, such as visualizing the distribution of size by day in the tips dataset.

|  |
| --- |
| **(**  **so.PPlot(tips, x="total\_bill", y="size", color="time")**  **.add(so.Dot(alpha=.5), so.Dodge(), so.Jitter(.4), orient="y")**  **)** |

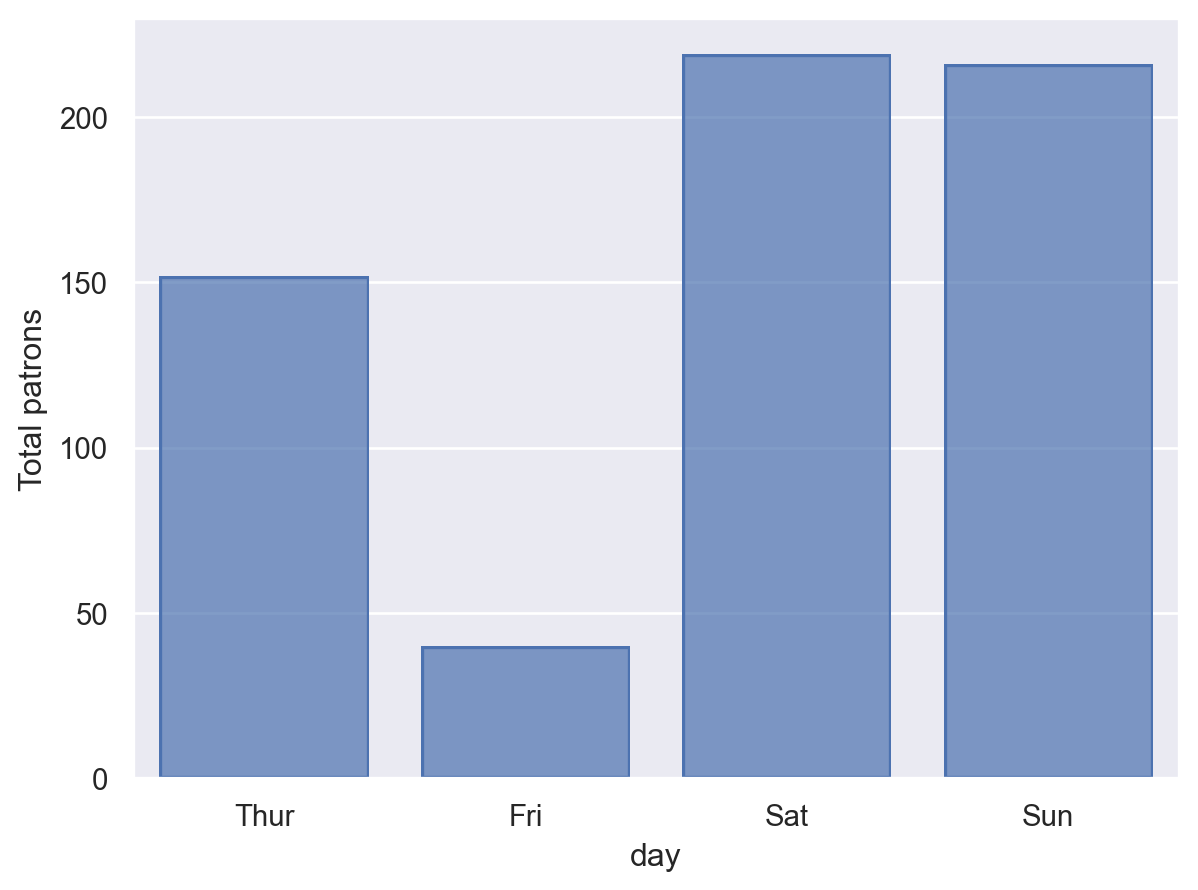


5)**Bar plot:**

 **Definition**: A bar plot shows the relationship between a categorical variable and a continuous variable using bars to represent the average or count of values.

 **Use Case**: Commonly used to compare values across different categories. For example, you can compare the average total bill for each day of the week.

|  |
| --- |
| (  so.Plot(tips, "day")  .add(so.Bar(), so.Hist(), weight="size")  .label(y="Total patrons")  ) |

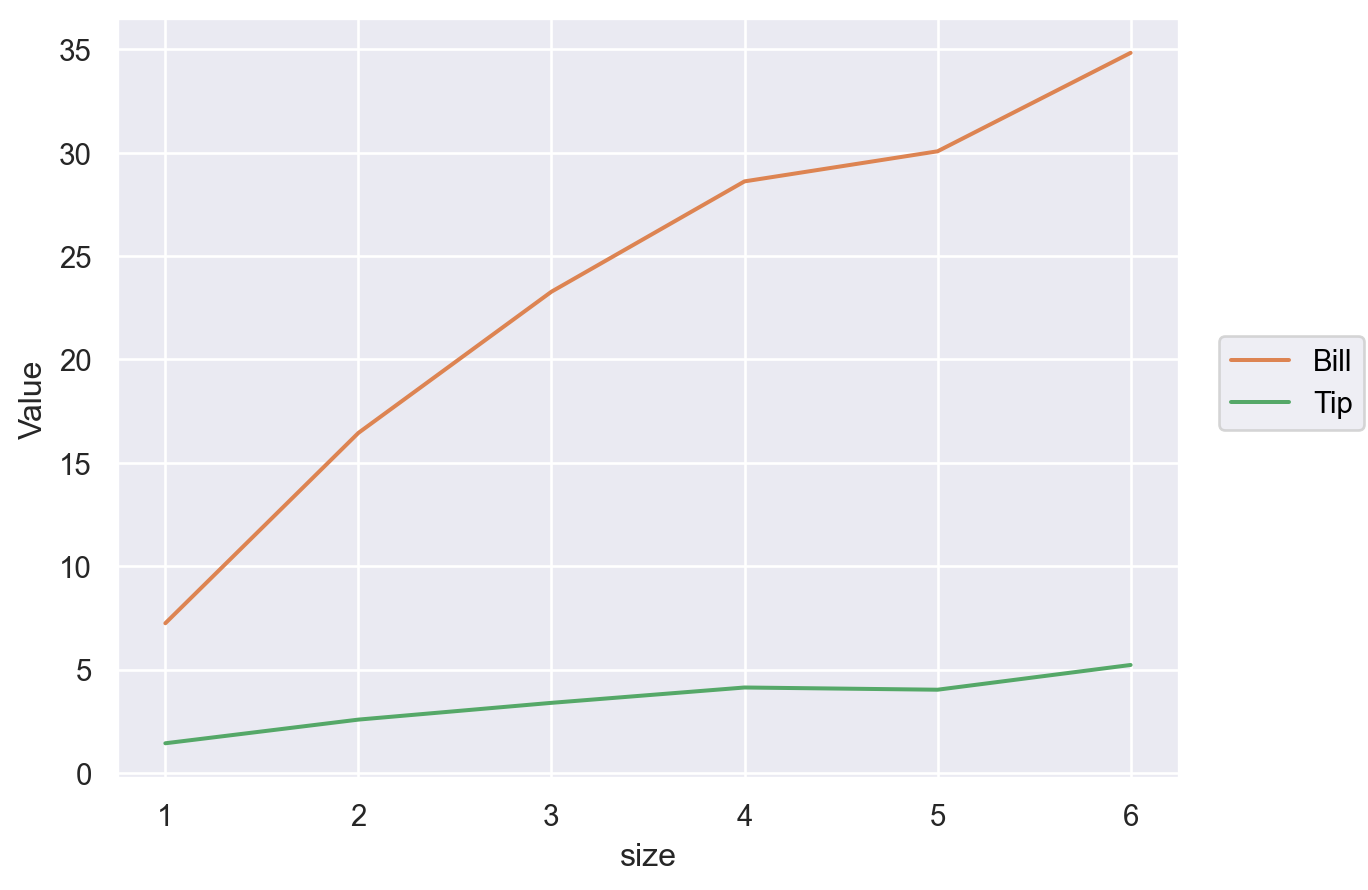


6)**Line plot:**

 **Definition**: A line plot connects data points with a continuous line, typically used to visualize trends over time or another continuous variable.

 **Use Case**: Suitable for visualizing time-series data or identifying trends. For example, showing how temperature changes over time.

|  |
| --- |
| (  so.Plot(tips, x="size")  .add(so.Line(color="C1"), so.Agg(), y="total\_bill", label="Bill")  .add(so.Line(color="C2"), so.Agg(), y="tip", label="Tip")  .label(y="Value")  ) |

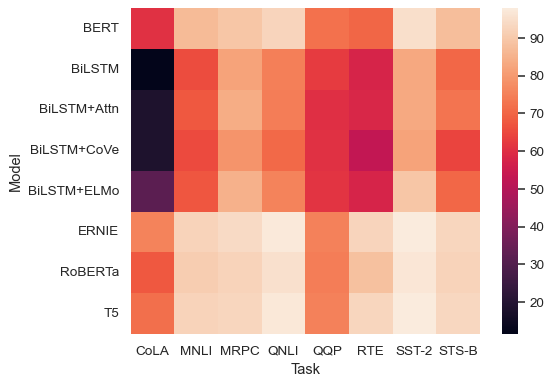


7)**heatmap:**

 **Definition**: A heatmap uses color to represent the values of a matrix or 2D dataset, often used to visualize the correlation between variables or to display the values of a matrix.

 **Use Case**: Best for visualizing correlations or any matrix-like data where the intensity of color represents magnitude. For example, visualizing the correlation between variables in a dataset.

|  |
| --- |
| glue = sns.load\_dataset("glue").pivot(index="Model", columns="Task", values="Score")  sns.heatmap(glue) |

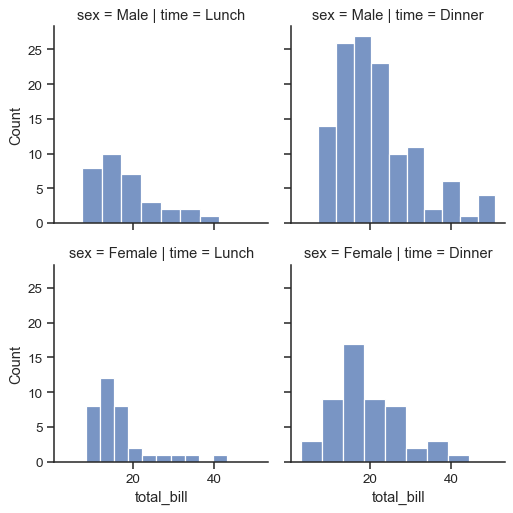


8**)Facet Grid:**

 **Definition**: A facet grid splits the data into subplots based on one or more categorical variables, allowing for easy comparison across subsets of data.

 **Use Case**: Facet grids are excellent for visualizing how variables behave across different subgroups. For example, comparing the distribution of total\_bill for different days and times.

|  |
| --- |
| g = sns.FacetGrid(tips, col="time", row="sex")  g.map\_dataframe(sns.histplot, x="total\_bill") |



3)**COMPARISON:**

A comparison of **Matplotlib** and **Seaborn** in table format:

|  |  |  |
| --- | --- | --- |
| Features | Matplotlib | Seaborn |
| Ease of Use | Requires more lines of code for basic plots, making it less beginner-friendly. | High-level API that simplifies creating complex visualizations with fewer lines of code. |
| Customization | Highly customizable, allows fine control over every aspect of the plot (axes, labels, lines, colors). | Limited customization compared to Matplotlib, but offers high-level customization options. |
| Aesthetic Appeal | Basic default aesthetics. Requires effort to make plots look polished and attractive. | Beautiful default styles with pleasing color palettes and grid lines. |
| Plot Variety | Supports a broad range of plots (e.g., 3D plots, complex subplots, time-series, animations). | Focuses on statistical plots like bar plots, scatter plots, histograms, and violin plots. |
| Plot Complexity | Excellent for creating highly complex, multi-layered plots. | Simplified plotting for quick visualizations but less suited for highly complex visualizations. |
| Statistical Visualizations | Requires more effort and code for statistical plots. | Excels in statistical visualizations (e.g., box plots, heatmaps, pair plots) and comes with built-in statistical features. |
| Interactivity | Supports static, animated, and interactive plots. | Primarily static plots but can work with interactive plots through Matplotlib. |
| Use Case | Ideal for highly customized and detailed visualizations, complex scientific plots. | Best for quick, beautiful statistical visualizations and exploratory data analysis (EDA). |
| Dependency | A standalone library. | Built on top of Matplotlib, can rely on it for more complex customization. |