MATPLOTLIB

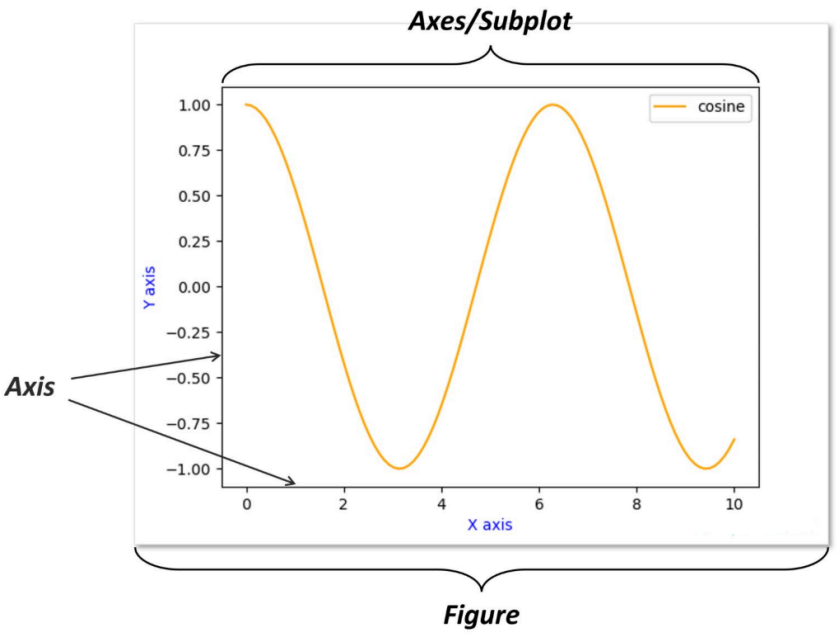
Introduction

Matplotlib is a popular Python library for creating static, interactive, and animated visualizations.
Since it is built on top of **NumPy**, it works efficiently with numerical data and integrates well with **Pandas** for plotting directly from DataFrames.

Anatomy of Matplotlib

A Matplotlib visualization is made up of four main components:

- Figure → The overall window or page where everything is drawn. A figure can contain one or multiple plots.
- Axes → The individual plots inside a figure (where data is actually drawn). Each Axes has its own X and Y axis.
- Axis → The X and Y axis objects inside the Axes, responsible for ticks, limits, and scale.
- Artists → Everything visible on the plot (lines, text, labels, legends, etc.) are called artists.



Pyplot

Pyplot is a submodule of Matplotlib (`matplotlib.pyplot`) that provides a simple, MATLAB-like interface for creating plots.

- It contains functions like plot(), bar(), hist(), scatter(), title(), xlabel(), etc.
- Each function acts on the current figure and axes - so you don't need to create them manually.
- It allows creating visualizations with just a few lines of code.

Importing Matplotlib

Before using Matplotlib, you need to install it , open your command prompt and install Matplotlib by following command

```
pip install matplotlib
```

After installation import the library into your code , you can use any IDE of your choice

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

We also import **NumPy** as a support for Matplotlib, since it helps in generating sample data for plots:

```
pip install numpy
```

```
import numpy as np
```

Let's dive into different types of plot

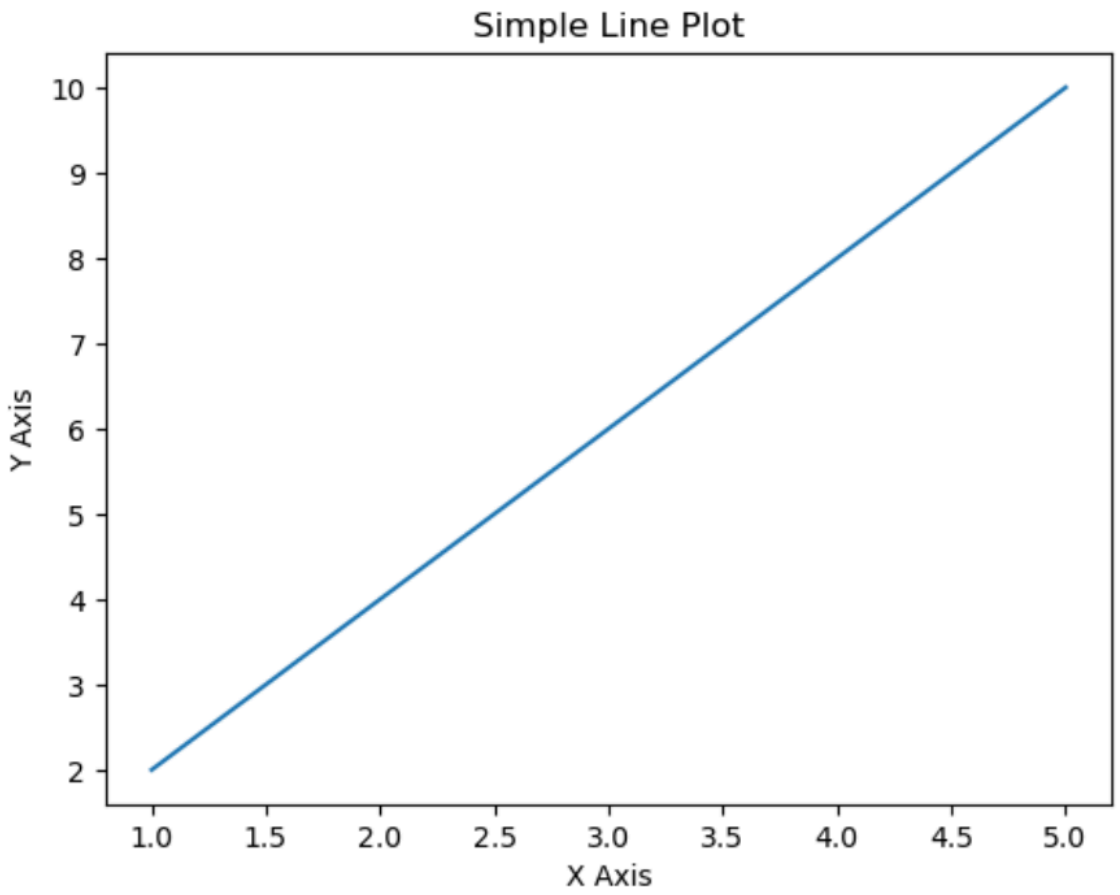
Line Plot

A line plot connects individual data points with straight lines. It is the simplest and most commonly used chart in data visualization.

```
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

plt.plot(x, y)
plt.title("Simple Line Plot")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.show()
```

OUTPUT:



Description:

- np.array() → creates an array of values.
- plot() → plots the line chart connecting data points.
- title() → defines the title of the chart.
- xlabel() and ylabel() → label the X and Y axes.
- show() → displays the final plot.

Use Cases:

- Showing trends over time (e.g., stock prices, temperature changes).
- Visualizing relationships between two continuous variables.

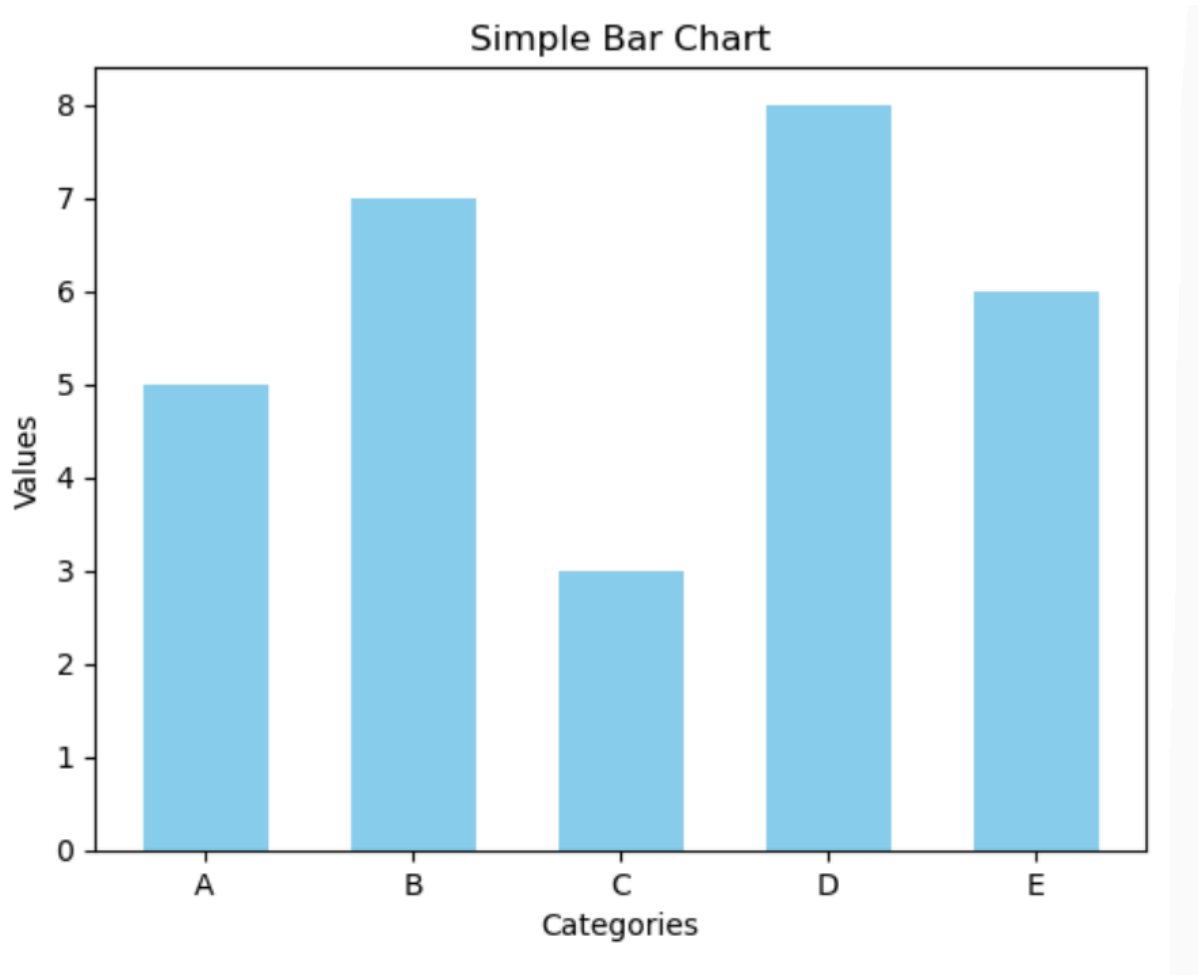
Bar chart

A bar chart represents data using rectangular bars, where the length of each bar is proportional to the value it represents.

```
x = np.array(['A', 'B', 'C', 'D', 'E'])
y = np.array([5, 7, 3, 8, 6])

plt.bar(x, y, color='skyblue', width=0.6)
plt.title("Simple Bar Chart")
plt.xlabel("Categories")
plt.ylabel("Values")
plt.show()
```

Output:



Description:

- np.array() → creates arrays for categories (x) and values (y).
- bar(x, y) → plots the bars with heights equal to the values.
- color → sets the color of bars.
- width → defines the thickness of bars.
- title(), xlabel(), ylabel() → add title and axis labels.
- show() → displays the chart.

Use Case:

- Comparing values across categories (e.g., sales across regions, student scores in subjects).

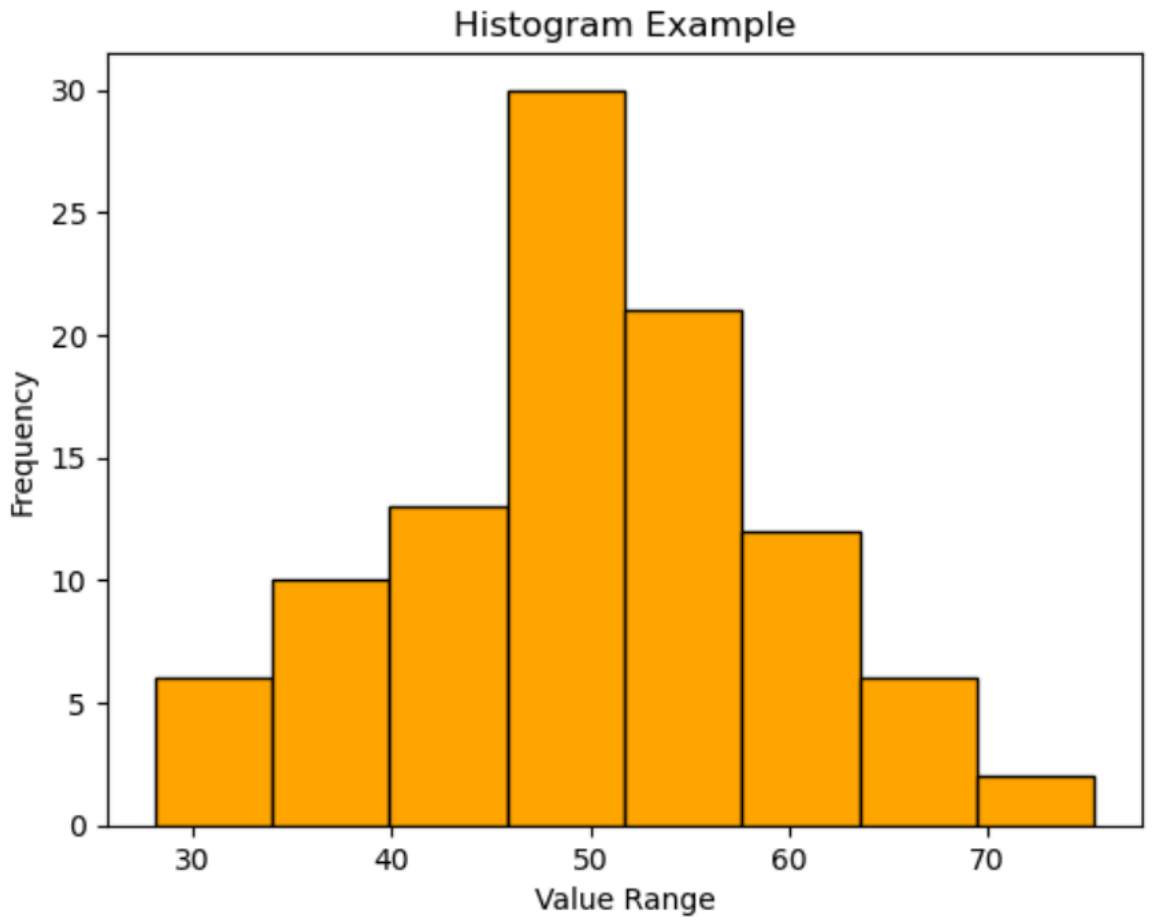
Histogram

A histogram shows the distribution of a dataset by grouping data into bins (intervals) and counting how many values fall into each bin.

```
data = np.random.normal(50, 10, 100)

plt.hist(data, bins=8, color='orange', edgecolor='black')
plt.title("Histogram Example")
plt.xlabel("Value Range")
plt.ylabel("Frequency")
plt.show()
```

OUTPUT:



Description:

- np.random.normal(50, 10, 100) → generates 100 random values with mean=50 and standard deviation=10.
- hist(data, bins=8) → plots the histogram by dividing data into 8 bins.
- color → sets the fill color of the bars.
- edgecolor → adds borders to bars for clarity.
- title(), xlabel(), ylabel() → add title and axis labels.
- show() → displays the chart.

Use Case:

- Understanding data distribution (e.g., exam scores, age distribution).
- Detecting skewness, spread, or outliers in datasets.

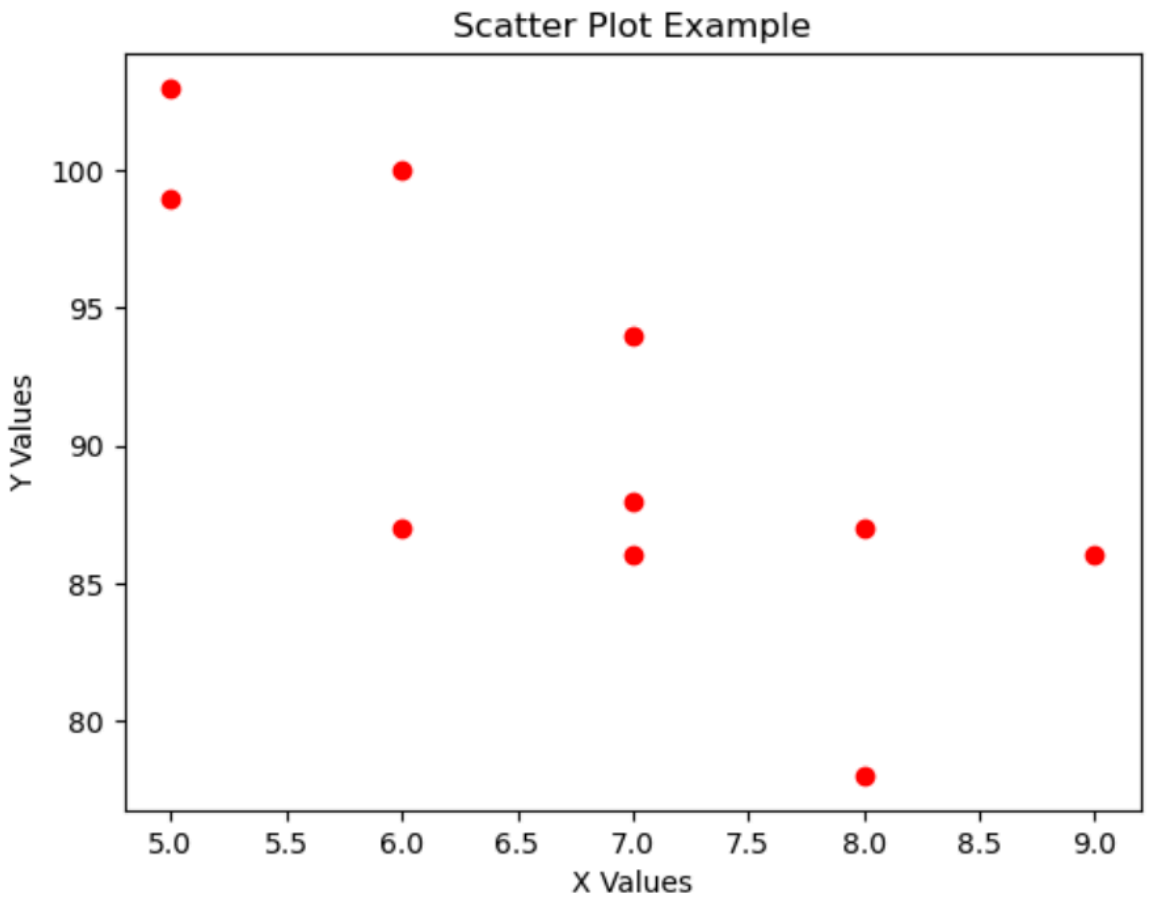
Scatter Plot

A scatter plot displays data points as dots on a two-dimensional plane, showing the relationship between two variables.

```
x = np.array([5, 7, 8, 7, 6, 9, 5, 6, 7, 8])
y = np.array([99, 86, 87, 88, 100, 86, 103, 87, 94, 78])

plt.scatter(x, y, color='red', marker='o')
plt.title("Scatter Plot Example")
plt.xlabel("X Values")
plt.ylabel("Y Values")
plt.show()
```

Output:



Description:

- np.random.normal(50, 10, 100) → generates 100 random values with mean=50 and standard deviation=10.
- hist(data, bins=8) → plots the histogram by dividing data into 8 bins.
- color → sets the fill color of the bars.
- edgecolor → adds borders to bars for clarity.
- title(), xlabel(), ylabel() → add title and axis labels.
- show() → displays the chart.

Use Case:

- Understanding data distribution (e.g., exam scores, age distribution).
- Detecting skewness, spread, or outliers in datasets.

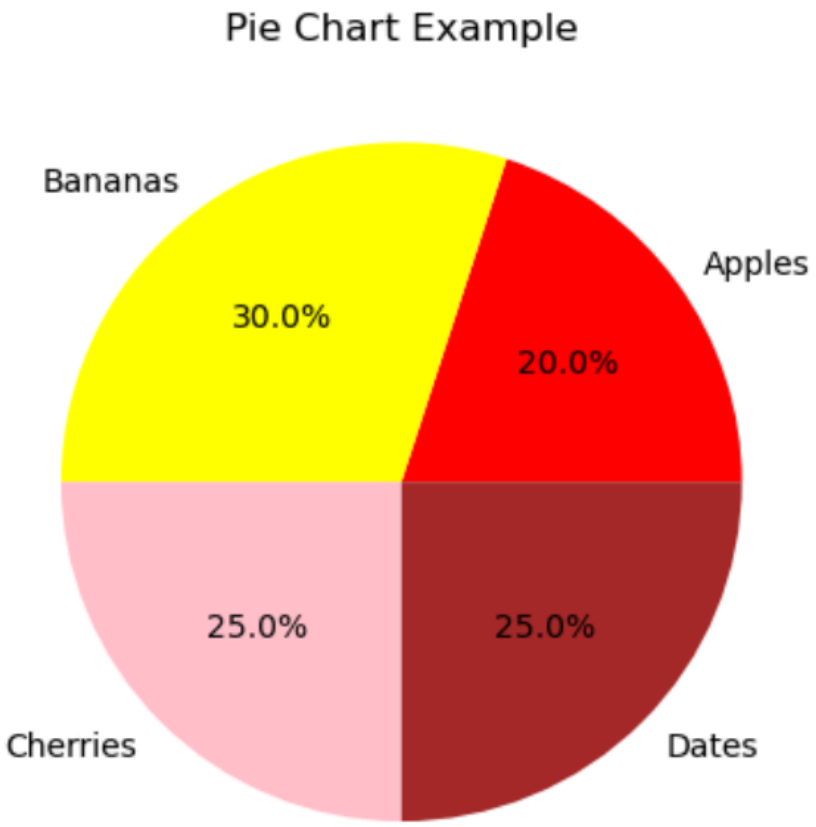
Pie Chart

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

```
sizes = np.array([20, 30, 25, 25])
labels = ["Apples", "Bananas", "Cherries", "Dates"]
colors = ["red", "yellow", "pink", "brown"]

plt.pie(sizes, labels=labels, colors=colors, autopct="%1.1f%%")
plt.title("Pie Chart Example")
plt.show()
```

Output:



Description:

- np.array() → creates an array of values representing each category's size.
- labels → defines names for each slice.
- colors → assigns colors to slices.
- pie() → creates the pie chart.
    - autopct → shows percentages on slices.
- title() → adds a chart title.
- show() → displays the chart.

Use Case:

- Showing proportion or percentage distribution (e.g., market share, budget allocation, survey results).

Area Chart

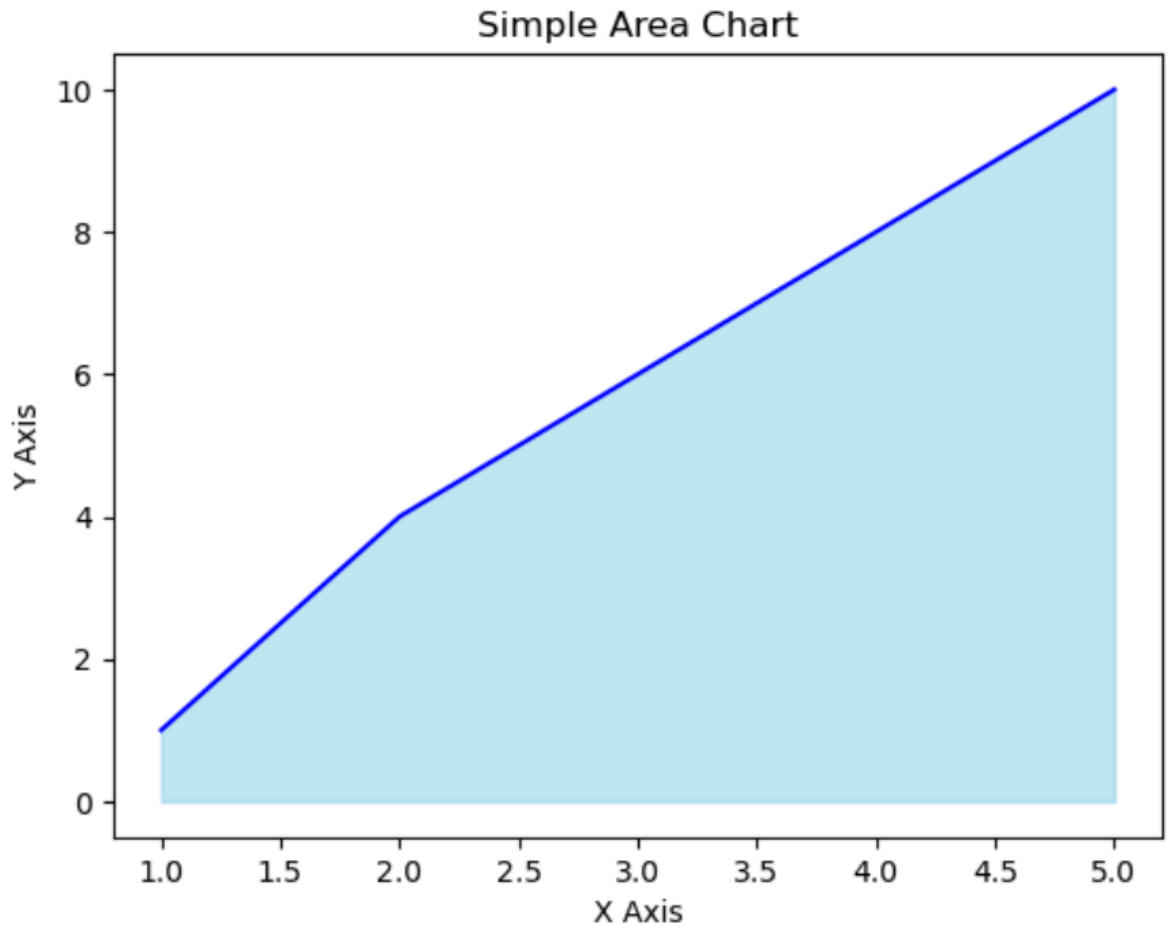
An area chart is similar to a line chart, but the space under the line is filled with color. It emphasizes the magnitude of values. When multiple datasets are stacked, it becomes a stacked area chart, which shows how different groups contribute to a total.

Simple Area Chart (using fill_between())

```
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 4, 6, 8, 10])

plt.fill_between(x, y, color="skyblue", alpha=0.5)
plt.plot(x, y, color="blue")
plt.title("Simple Area Chart")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.show()
```

Output:



Description:

- fill_between(x, y) → fills the area between the curve and X-axis.
- color → sets fill color.
- alpha → controls transparency.
- plot(x, y) → overlays the line.
- title(), xlabel(), ylabel() → add chart title and axis labels.
- show() → displays the plot.

Use Case:

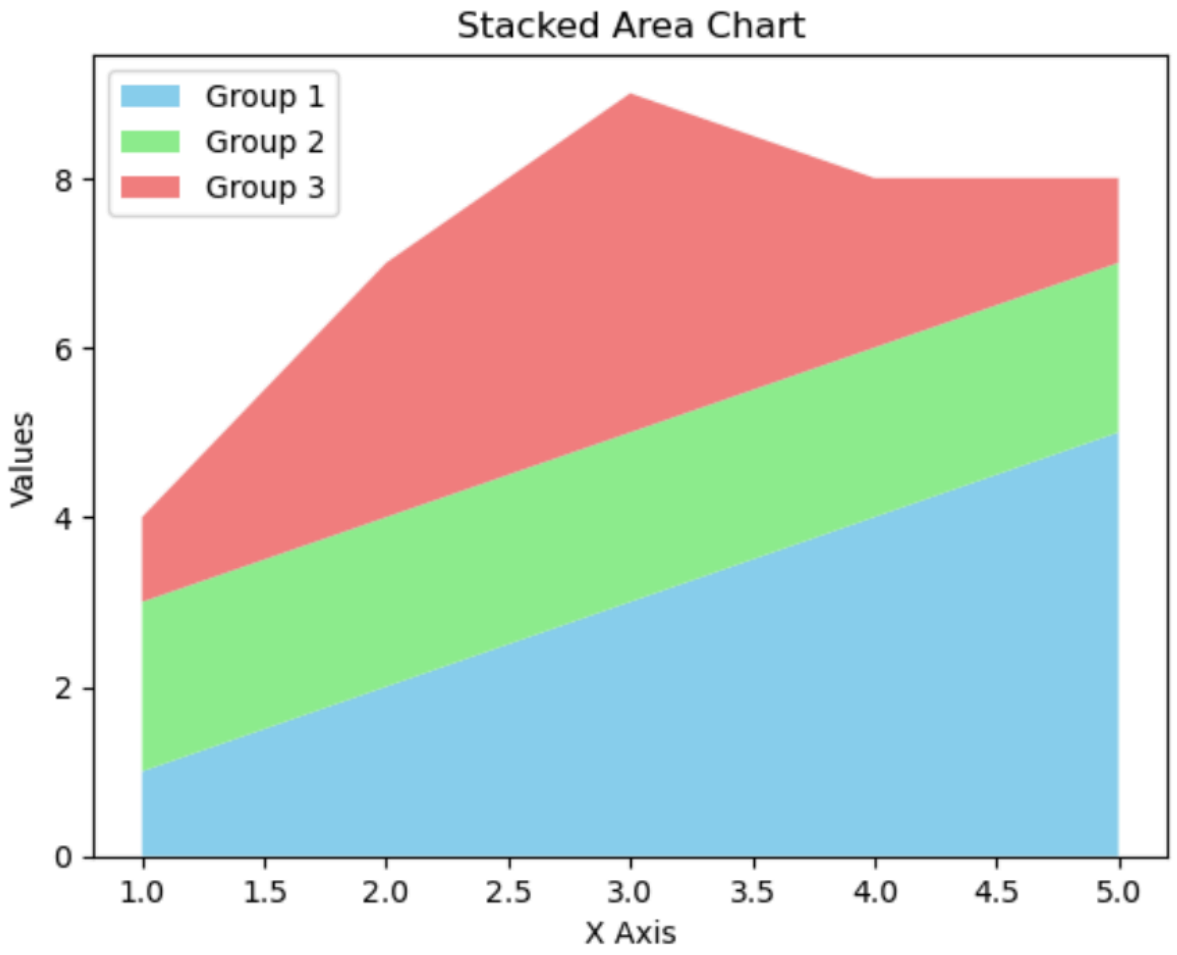
- Showing cumulative growth (e.g., revenue increase, population trend).

Stacked Area Chart

```
x = np.array([1, 2, 3, 4, 5])
y1 = np.array([1, 2, 3, 4, 5])
y2 = np.array([2, 2, 2, 2, 2])
y3 = np.array([1, 3, 4, 2, 1])

plt.stackplot(x, y1, y2, y3, labels=['Group 1', 'Group 2', 'Group 3'],
              colors=['skyblue', 'lightgreen', 'lightcoral'])
plt.title("Stacked Area Chart")
plt.xlabel("X Axis")
plt.ylabel("Values")
plt.legend()
plt.show()
```

Output:



Description:

- stackplot(x, y1, y2, y3) → creates stacked areas.
- labels → names each dataset.
- colors → assigns colors to each group.
- legend() → adds legend to differentiate groups.
- title(), xlabel(), ylabel() → add chart details.
- show() → displays the figure.

Use Case:

- Showing category-wise contribution to a total (e.g., expense breakdown, energy sources, market shares).

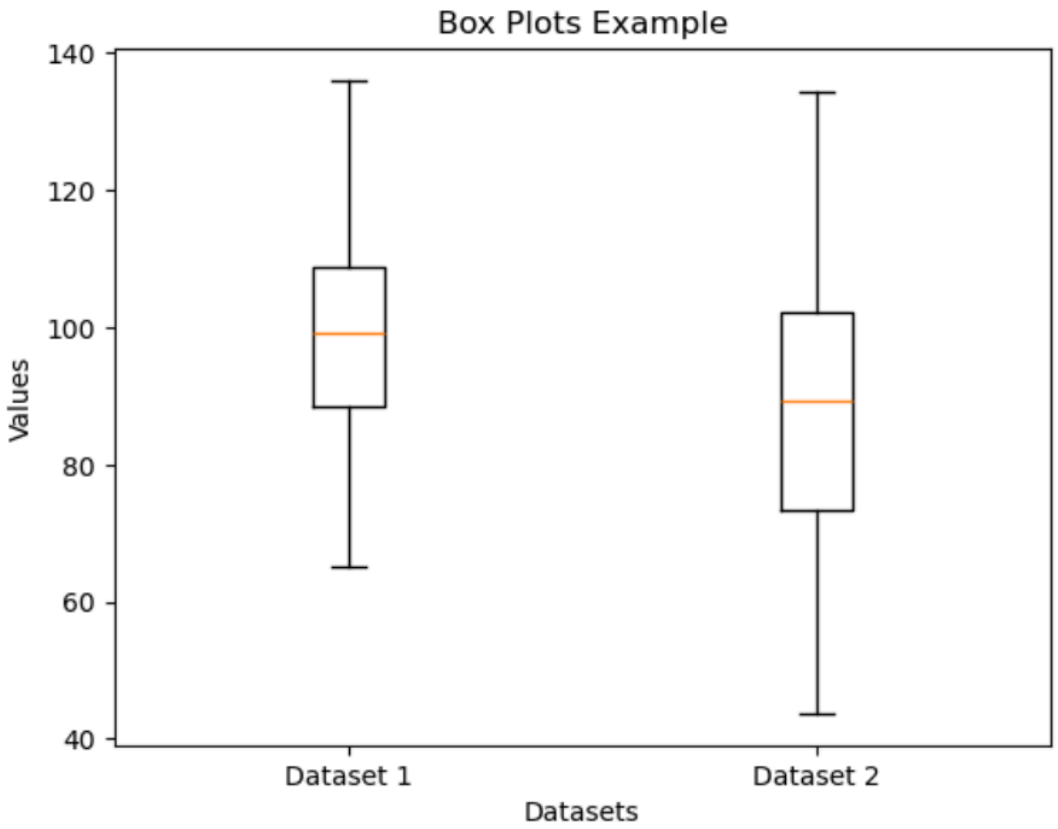
Box Plot

A box plot (also called a whisker plot) summarizes the distribution of a dataset by showing its minimum, first quartile (Q1), median, third quartile (Q3), and maximum. It also highlights outliers.

```
data1 = np.random.normal(100, 15, 200)
data2 = np.random.normal(90, 20, 200)

plt.boxplot([data1, data2], tick_labels=["Dataset 1", "Dataset 2"])
plt.title("Box Plots Example")
plt.xlabel("Datasets")
plt.ylabel("Values")
plt.show()
```

Output:



Description:

- np.random.normal() → generates two datasets with different means and spreads.
- boxplot([data1, data2], tick_labels=[...]) → draws two box plots side by side for comparison.
- tick_labels → assigns names under each box
- Each box shows:
    - Middle line = median
    - Box edges = Q1 and Q3
    - Whiskers = min and max (within 1.5×IQR)
    - Points beyond whiskers = outliers
- title(), xlabel(), ylabel() → add chart details.
- show() → displays the chart.

Use Case:

- Comparing distributions between groups (e.g., exam scores of two classes, revenue of two companies, performance of two algorithms).

PLOTLY

Introduction

Plotly is a Python library used for creating interactive graphs and charts. It allows users to make visualizations where you can move your mouse to see values, zoom in, and explore the data easily. Plotly supports many types of charts such as line charts, bar charts, scatter plots, pie charts, and even 3D charts. It is often used when we want our graphs to be more engaging and easy to understand.

Importing Plotly

pip install plotly

In Python, Plotly is usually imported in two ways:

```
import plotly.express as px
import plotly.graph_objects as go
```

- **Plotly Express (px)** → High-level interface for simple and quick charts. Minimal code, beginner-friendly.
- **Graph Objects (go)** → Low-level interface giving more control and customization. Requires more lines of code.

In this documentation, we will use **Plotly Express (px)** for all examples, as it is easier to learn and understand for beginners.

Let's dive into different types of plot

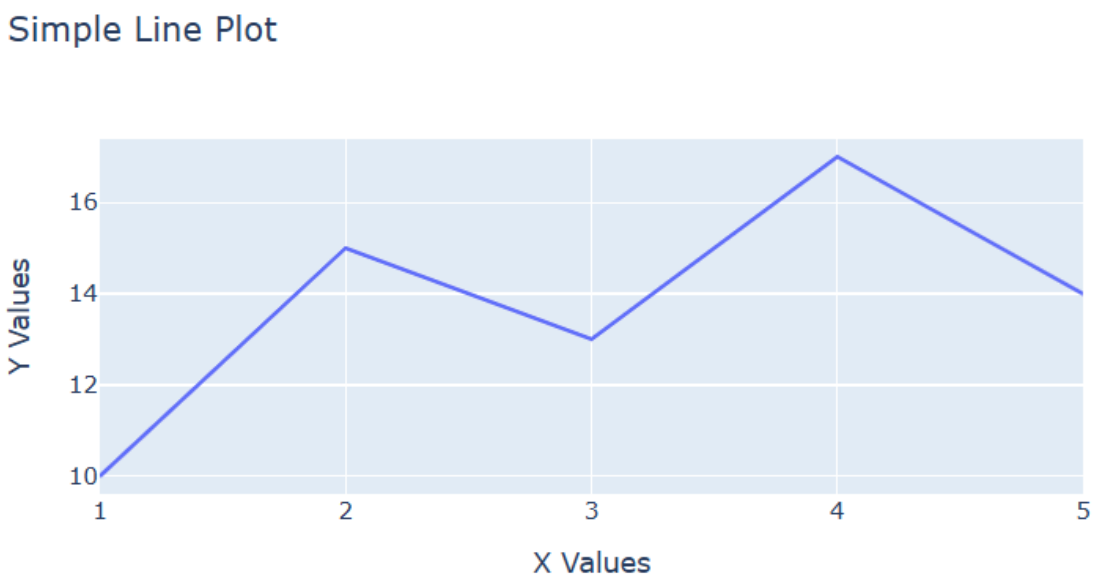
Line plot

Code snippet:

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 13, 17, 14]

fig = px.line(x=x, y=y, title="Simple Line Plot", labels={'x':'X Values', 'y':'Y Values'})
fig.show()
```

Output:



Description:

- x → list of values for the X-axis.
- y → list of values for the Y-axis.
- px.line(x=x, y=y) → creates a line plot connecting the points.
- title → sets the chart title.
- labels → customizes axis labels ({'x':'X Values','y':'Y Values'}).
- fig.show() → displays the interactive chart.

Use Case:

- Tracking changes over time (e.g., stock prices, temperature trends).
- Visualizing continuous data or trends.

Bar Chart

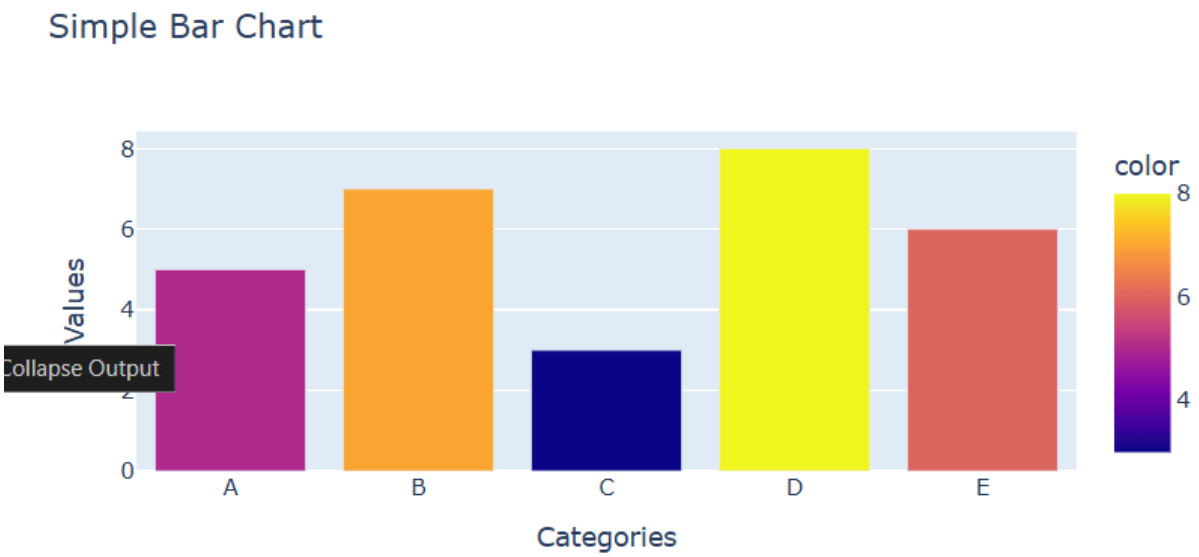
Code snippet:

```
x = ['A', 'B', 'C', 'D', 'E']
y = [5, 7, 3, 8, 6]

fig = px.bar(x=x, y=y, title="Simple Bar Chart",
             labels={'x':'Categories', 'y':'Values'}, color=y)

fig.show()
```

Output:



Description:

- x → list of categories.
- y → list of values.
- px.bar(x=x, y=y) → creates a bar chart.
- title → sets the chart title.
- labels → customizes axis labels.
- color → sets the color of bars (here using values for gradient).
- fig.show() → displays the interactive chart.

Use Case:

- Comparing values across categories (e.g., sales across regions, student scores).

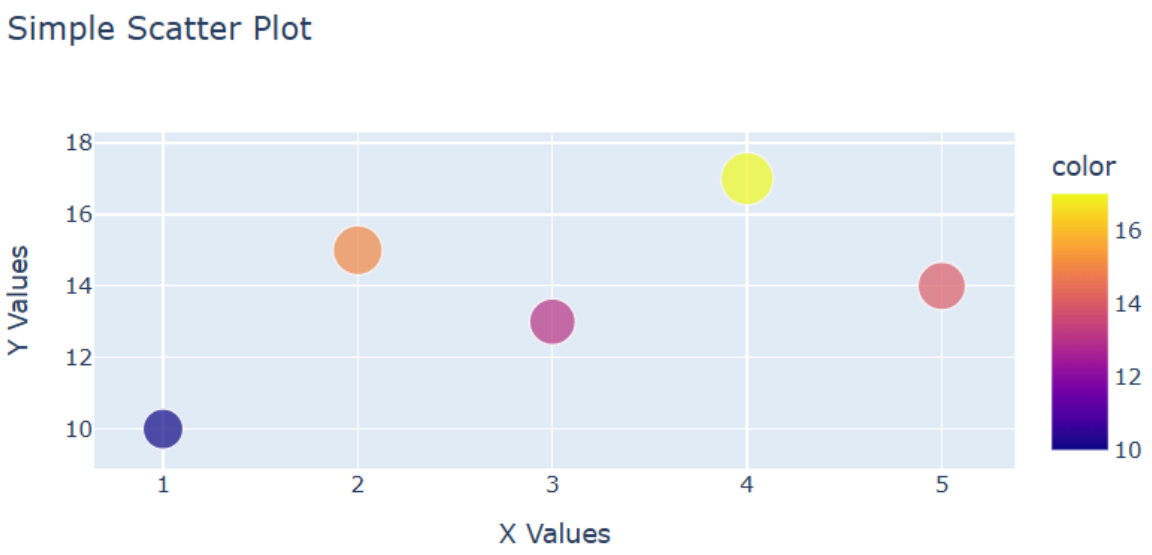
Scatter Plot

Code snippet:

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 13, 17, 14]

fig = px.scatter(x=x, y=y, title="Simple Scatter Plot",
                 labels={'x':'X Values', 'y':'Y Values'}, size=y, color=y)
fig.show()
```

Output:



Description:

- x → list of X-axis values.
- y → list of Y-axis values.
- px.scatter(x=x, y=y) → creates a scatter plot with points at (x, y).
- title → sets the chart title.
- labels → customizes axis labels.

- size → sets the size of each marker (optional).
- color → sets the color of each marker (optional).
- fig.show() → displays the interactive chart.

Use Case:

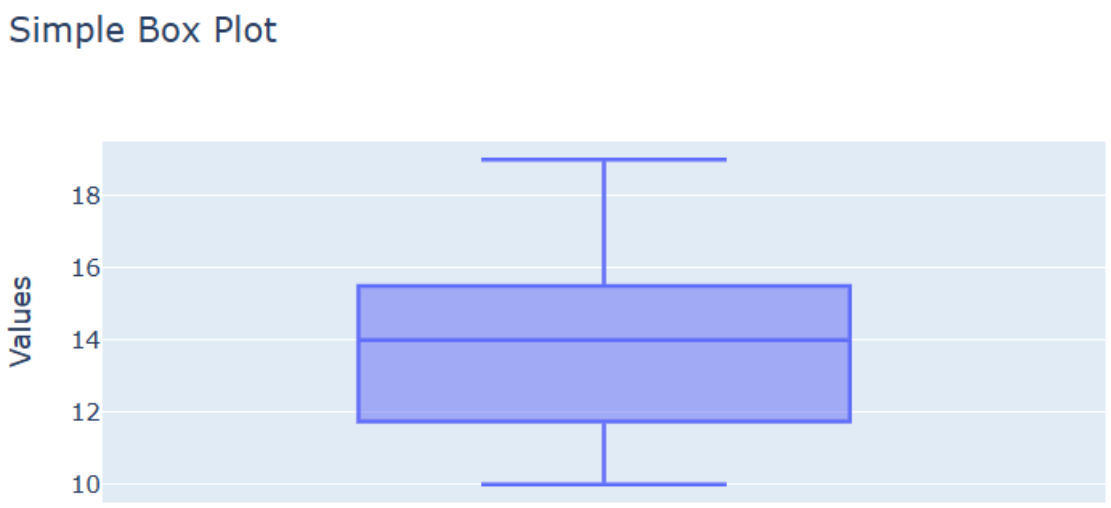
- Showing relationships between two variables.
- Identifying patterns, trends, or clusters in data.

Box plot

Code snippet:

```
y = [10, 15, 13, 17, 14, 19, 12, 15, 11]
fig = px.box(y=y, title="Simple Box Plot", labels={'y':'Values'})
fig.show()
```

Output:



Description:

- x → list of X-axis values.
- y → list of Y-axis values.
- px.scatter(x=x, y=y) → creates a scatter plot with points at (x, y).

- title → sets the chart title.
- labels → customizes axis labels.
- size → sets the size of each marker (optional).
- color → sets the color of each marker (optional).
- fig.show() → displays the interactive chart.

Use Case:

- Showing relationships between two variables.
- Identifying patterns, trends, or clusters in data.

Pie Chart

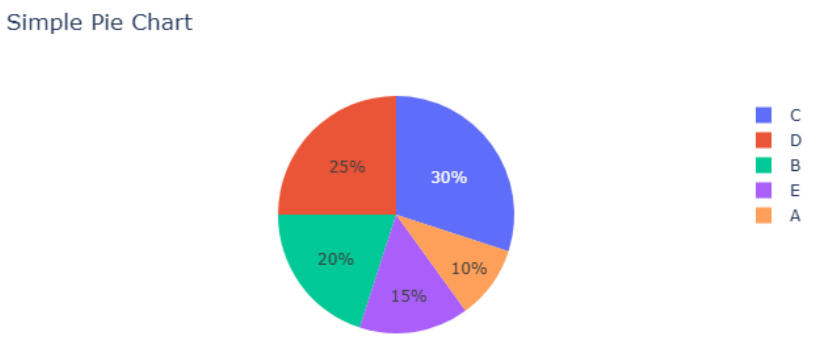
Code snippet:

```
labels = ['A', 'B', 'C', 'D', 'E']
values = [10, 20, 30, 25, 15]

fig = px.pie(names=labels, values=values, title="Simple Pie Chart")

fig.show()
```

Output:



Description:

- labels → list of categories.
- values → list of corresponding values for each category.
- px.pie(names=labels, values=values) → creates a pie chart.
- title → sets the chart title.
- fig.show() → displays the interactive chart.

Use Case:

- Showing proportional distribution of categories.
- Visualizing market share, survey results, or budget allocation.

Area Plot

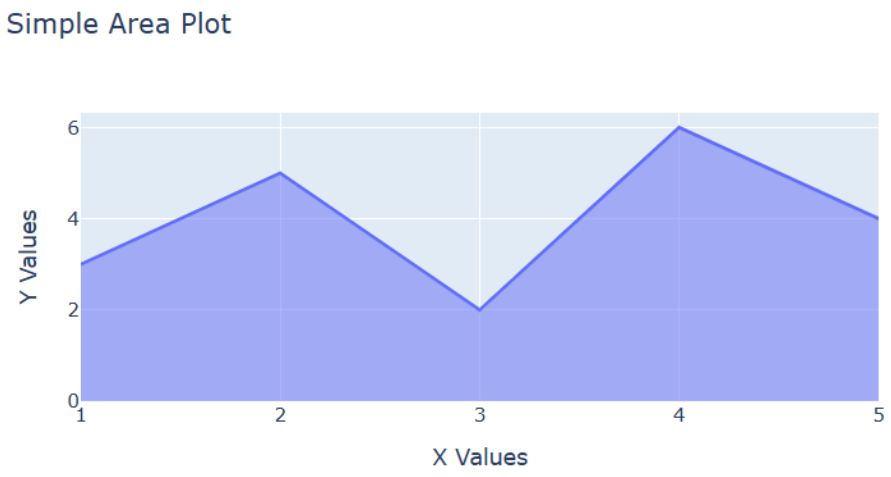
Code Snippet: (for simple area plot)

```
x = [1, 2, 3, 4, 5]
y = [3, 5, 2, 6, 4]

fig = px.area(x=x, y=y, title="Simple Area Plot", labels={'x':'X Values', 'y':'Y Values'})

fig.show()
```

Output:



Description:

- x → list of X-axis values.
- y → list of Y-axis values.
- px.area(x=x, y=y) → creates an area plot by shading the area under the line.
- title → sets the chart title.
- labels → customizes axis labels.
- fig.show() → displays the interactive chart.

Use Case:

- Showing cumulative trends over time.
- Visualizing quantities that change continuously**.**

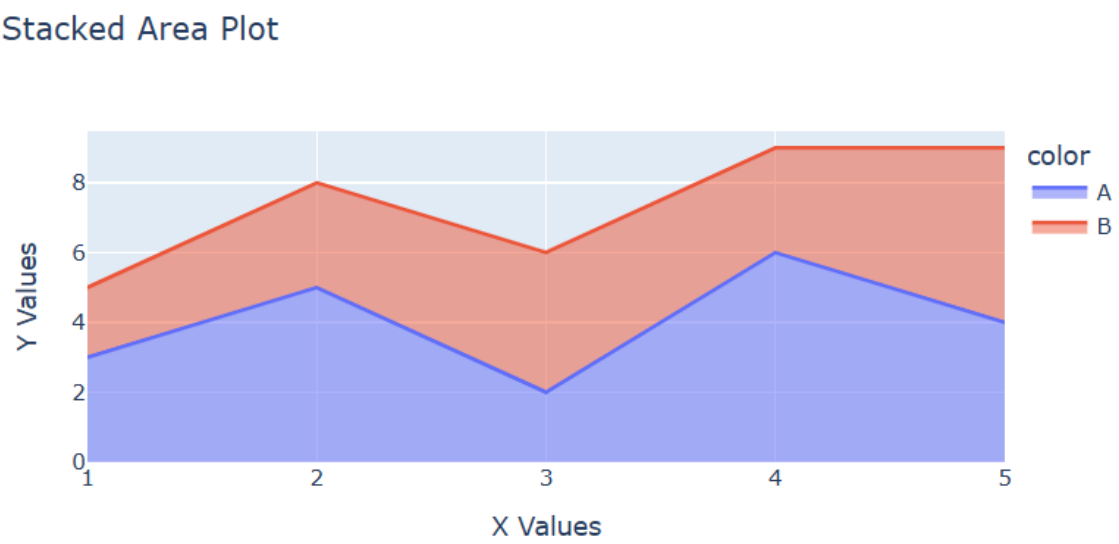
Code Snippet: (for stack plot)

```
x = [1, 2, 3, 4, 5]
y1 = [3, 5, 2, 6, 4]
y2 = [2, 3, 4, 3, 5]

fig = px.area(
    x=x*2,
    y=y1 + y2,
    color=['A']*len(x) + ['B']*len(x),
    title="Stacked Area Plot",
    labels={"x":"X Values","y":"Y Values"}
)

fig.show()
```

Output:



Description:

- x → list of X-axis values repeated for each series.
- y1, y2 → lists of Y-axis values for each series.
- y=y1 + y2 → combines the Y-values into a single list.
- color=['A']*len(x) + ['B']*len(x) → assigns a category to each series; Plotly automatically stacks the areas by category.
- px.area(...) → creates the area plot with stacked series.
- title → sets the chart title.
- labels → customizes axis labels.
- fig.show() → displays the interactive stacked area chart.

Use Case:

- Showing cumulative trends over time.
- Visualizing quantities that change continuously.

Histogram

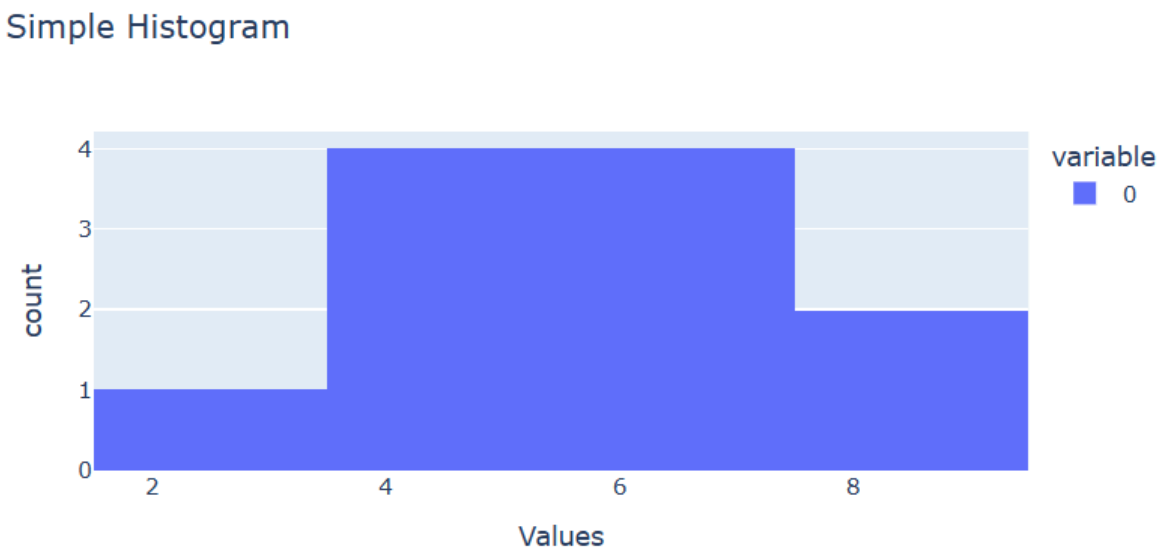
Code snippet:

```
data = [5, 7, 3, 8, 6, 5, 7, 4, 6, 8, 5]

fig = px.histogram(data, nbins=5, title="Simple Histogram", labels={'value':'Values'})

fig.show()
```

Output:



Description:

- data → list of values to be grouped into bins.
- px.histogram(data) → creates a histogram showing frequency distribution.
- nbins → defines the number of bins.
- title → sets the chart title.
- labels → customizes axis labels.
- fig.show() → displays the interactive histogram.

Use Case:

- Understanding data distribution and frequency.
- Identifying patterns, ranges, and outliers in a dataset.

COMPARISON OF MATPLOTLIB AND PLOTLY

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, Plotly, and others, can integrate with it.

- **Customization:** Matplotlib offers extensive customization options, allowing users to adjust colors, line styles, markers, fonts, annotations, and more.

- **Integration:** Matplotlib works seamlessly with libraries like NumPy, Pandas, and SciPy, making it a versatile choice for data visualization in scientific and analytical projects.

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.

Plotly:

- **Interactive Visualizations:** Plotly is a library for creating interactive, web-ready visualizations in Python. Unlike Matplotlib, plots are automatically interactive, allowing zooming, hovering, and dynamic updates.

- **Ease of Use with Plotly Express:** Plotly Express provides a high-level interface to quickly create complex visualizations with minimal code, ideal for beginners and rapid prototyping.

- **Versatility:** Plotly supports a wide range of plot types, including 2D, 3D, geographic maps, time series, polar charts, and specialized charts like bubble charts, treemaps, and candlestick plots.

- **Aesthetics:** Plotly has attractive default styles and color schemes. Charts are visually appealing out-of-the-box without requiring additional styling.

- **Integration:** Plotly integrates with Pandas, NumPy, and Dash (for dashboards), making it easy to create dynamic visualizations for both analysis and web applications.

Advantages of Plotly:

- Interactive and web-friendly visualizations by default.

- Quick plotting with Plotly Express for beginners.

- Wide range of advanced and specialized charts.

- Attractive default styles without extra customization.

- Ideal for dashboards, data exploration, and interactive reporting.

Summary:

- **Matplotlib** is best for fine-grained control, static figures, and publication-quality plots.

- **Plotly** is best for interactive, dynamic, and web-ready visualizations with less effort.

- Many users combine both: Matplotlib for detailed, static plots and Plotly for interactive exploration and dashboards.