PYTHON VISUALIZATION LIBRARIES

Data visualization is a key step in data analysis and data science. It helps transform raw data into charts and graphs so patterns, relationships, and insights can be easily understood.

Python has powerful libraries for visualization, such as:

- Matplotlib → Traditional, static, highly customizable plots.
- Plotly → Interactive, modern, web-friendly visualizations.

MATPLOTLIB

Matplotlib is one of the most popular and widely used **data visualization libraries in Python**. It provides tools for creating **static, animated, and interactive plots**, making it essential for data science, machine learning, and scientific research.

It was originally developed by **John D. Hunter in 2003** as an open-source project to bring MATLAB-style plotting to Python.

Matplotlibs components are

1. Figure

- The outer container that holds everything in a plot (like a blank canvas).
- A single figure can contain one or multiple plots.
  Use case: If you want to compare different graphs side by side, you create multiple subplots inside one figure.

2. Axes

- The actual plot area inside a figure (like a coordinate system).
- Each axes object has:
  - **x-axis** (horizontal)
  - **y-axis** (vertical)
    Use case: Drawing the graph itself (line, bar, scatter, etc.).

3. Axis

- Refers to the x-axis or y-axis inside a plot.
- Responsible for **ticks**, **labels**, and **scales**.
  Use case: Showing the range of values, e.g., 0–100 on x-axis.

4. Artists

- Everything that appears on the plot is an "artist."
- Includes: lines, bars, text, labels, legends, markers, etc.
  Use case: Customizing the style (colors, labels, legends).

5. Pyplot Module (matplotlib.pyplot)

- A collection of functions that make plotting easier (like a shortcut).
- Provides commands such as:
  - plot() → line plot
  - bar() → bar chart
  - scatter() → scatter plot
  - hist() → histogram
  - pie() → pie chart
    Use case: Most common entry point for beginners to create quick plots.

❖ **Unique Features:**

- Fine-grained control over plots.
- Wide range of chart types.
- Exports plots to multiple file formats (PNG, PDF, SVG).

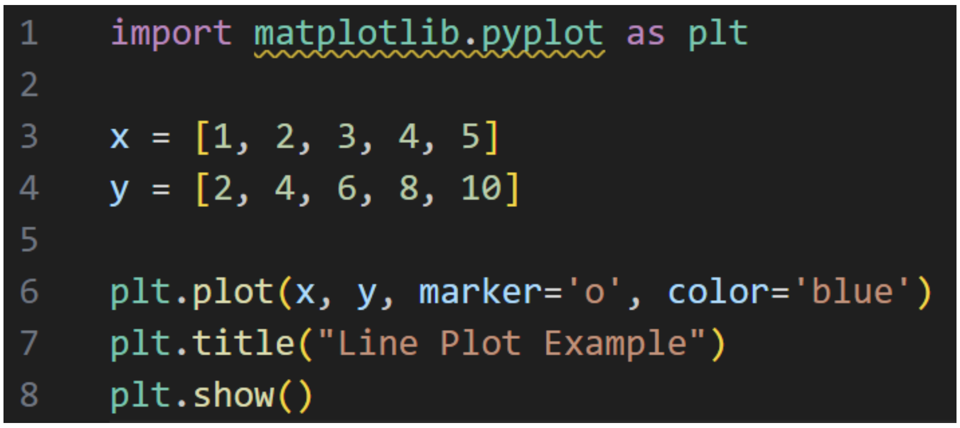
❖ **Typical Use Cases:**

- Scientific research.
- Data analysis reports.
- Publication-quality graphs.

Line Chart

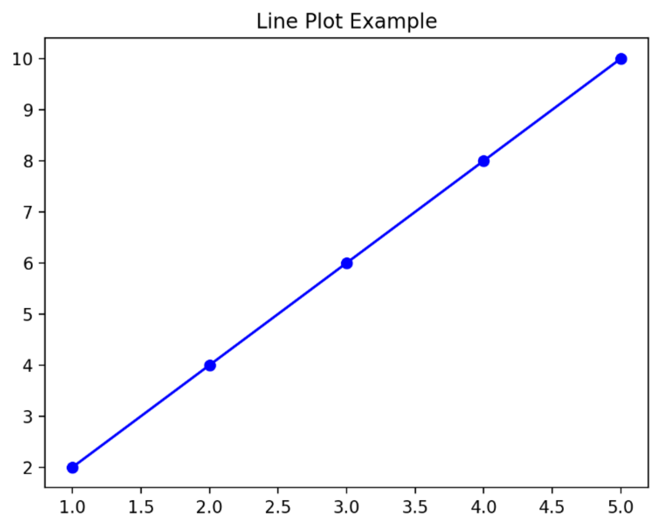
Shows trends/relationships in continuous data.Visualizing stock prices over time. Easy to read trends, but not suitable for categorical data.

Code :



```python
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.plot(x, y, marker='o', color='blue')
plt.title("Line Plot Example")
plt.show()
```

Output :



Description :

It is often used to **show trends** over time (e.g., sales growth per month).

plt.plot() → draws the line.

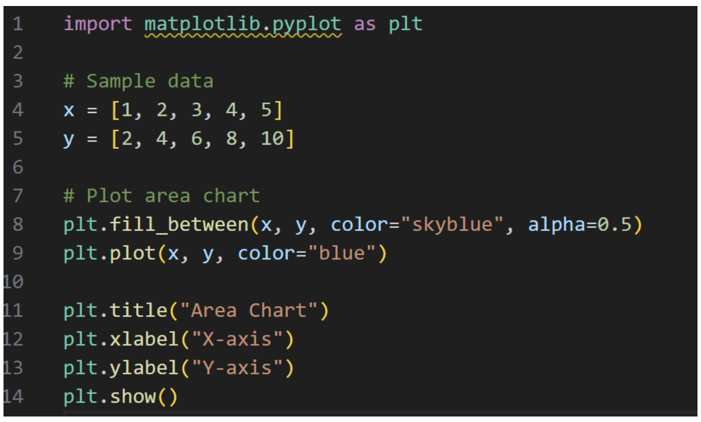
marker='o' → shows circles at each point.

plt.show() → displays the plot window.

Area Chart

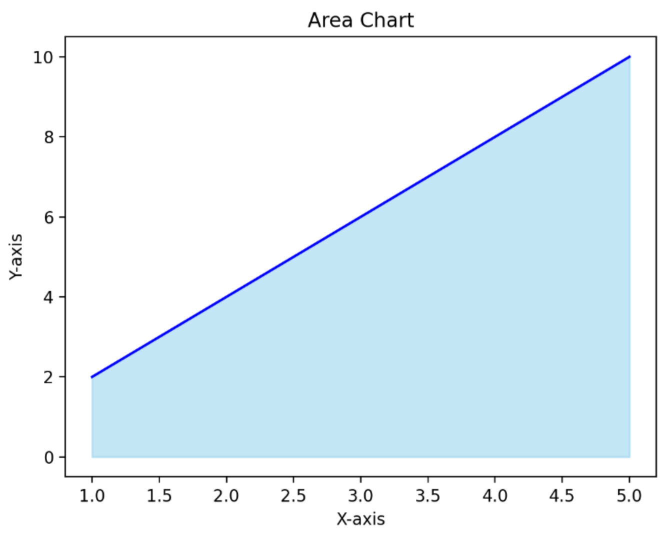
An area chart is similar to a line chart but with the **area under the curve filled with color**. Great for showing **cumulative values over time**. Stacked area plots show how multiple categories contribute to a total.

Code :



```python
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Plot area chart
plt.fill_between(x, y, color="skyblue", alpha=0.5)
plt.plot(x, y, color="blue")

plt.title("Area Chart")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

Output :



Description :

plt.fill_between(x, y, color="skyblue", alpha=0.5) → Fills the **area under the line** with light blue.

plt.plot(x, y, color="blue") → Draws the line on top of the filled area.

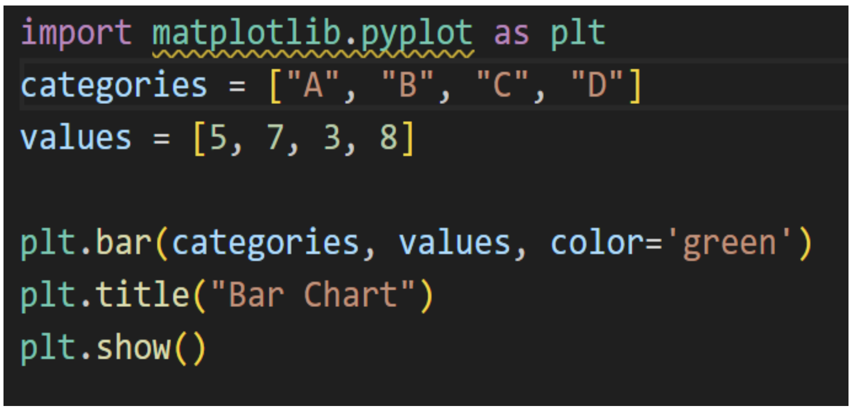
alpha=0.5 → Adds transparency so the area isn't too dark.

Best for **showing trends and accumulated values** over time.

Bar Chart

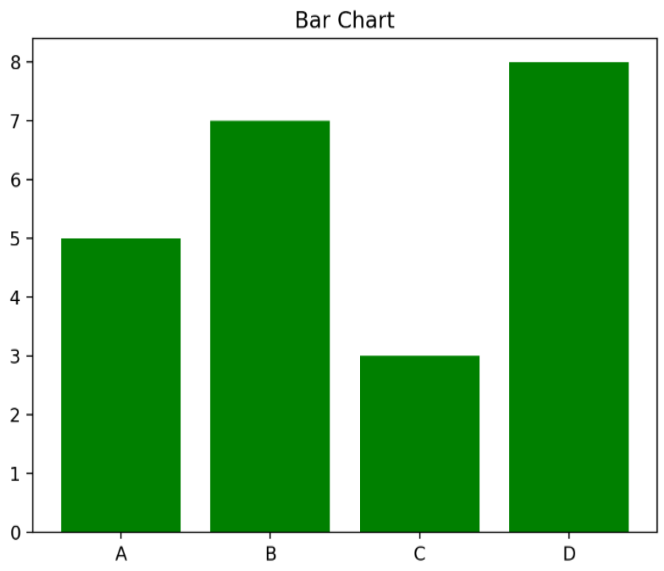
A bar chart represents data with rectangular bars, where the **length of each bar is proportional to its value**. It is useful for comparing **different categories** (e.g., revenue per product, number of students in classes). Works well for categorical data, but can become cluttered if too many categories are used.

Code :



```
import matplotlib.pyplot as plt
categories = ["A", "B", "C", "D"]
values = [5, 7, 3, 8]

plt.bar(categories, values, color='green')
plt.title("Bar Chart")
plt.show()
```

Output :



Description :

Good for comparing **different categories** (e.g., sales by product).

plt.bar() → creates vertical bars.

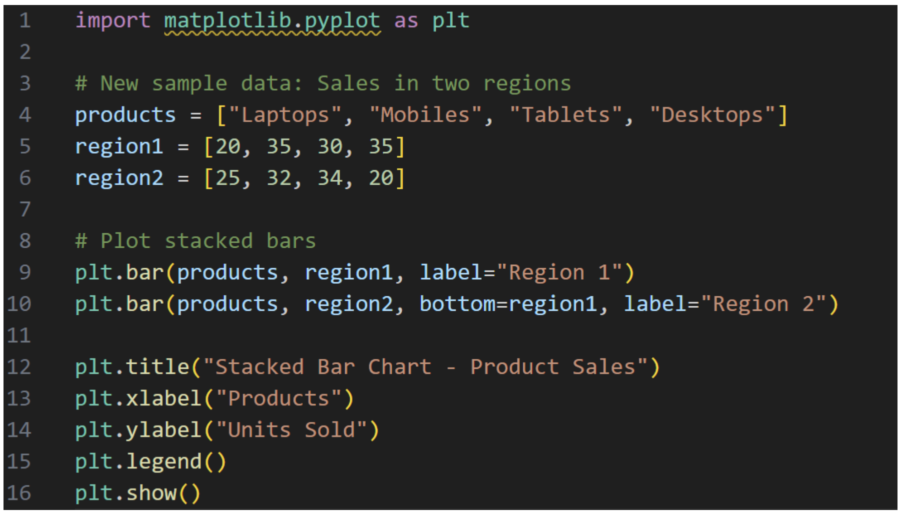
Height of each bar = value.

Labels on x-axis = categories.

Stacked Bar Chart

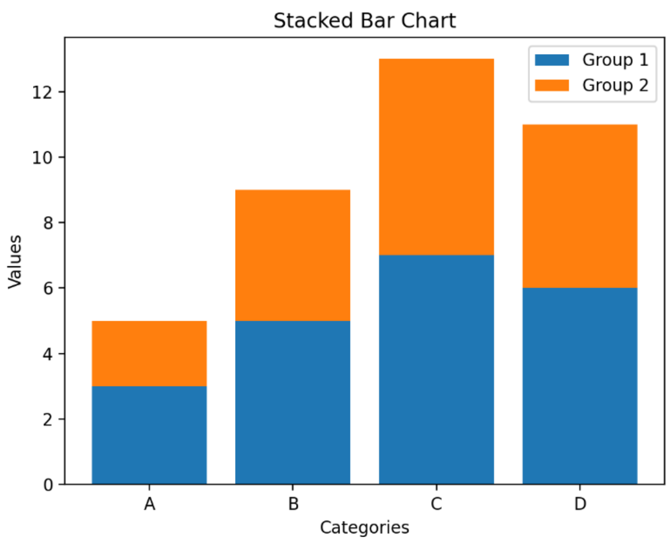
A stacked bar chart shows bars divided into segments. Each bar represents the total, and the segments represent subcategories. Useful for showing **contributions to a whole**. Informative but can get difficult to read with too many categories.

Code :



```python
import matplotlib.pyplot as plt

# New sample data: Sales in two regions
products = ["Laptops", "Mobiles", "Tablets", "Desktops"]
region1 = [20, 35, 30, 35]
region2 = [25, 32, 34, 20]

# Plot stacked bars
plt.bar(products, region1, label="Region 1")
plt.bar(products, region2, bottom=region1, label="Region 2")

plt.title("Stacked Bar Chart - Product Sales")
plt.xlabel("Products")
plt.ylabel("Units Sold")
plt.legend()
plt.show()
```

Output :



Description :

plt.bar(categories, values1, ...) → Creates the first set of bars for **Group 1**.

plt.bar(categories, values2, bottom=values1, ...) → Places **Group 2** values **on top of Group 1** (stacked).

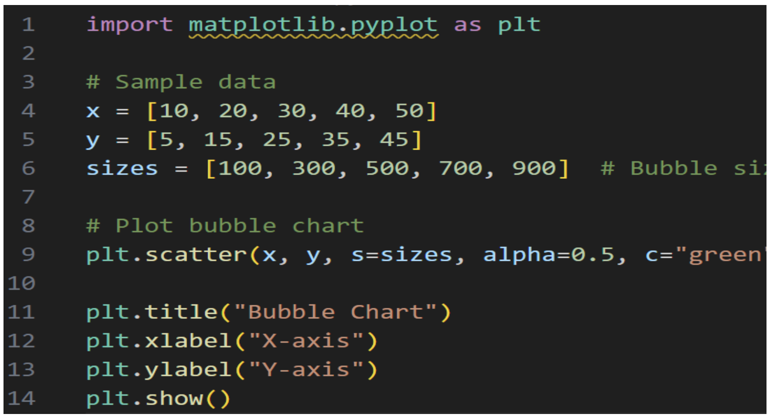
plt.legend() → Adds labels so we know which color belongs to which group.

Useful when you want to compare **parts to a whole**.

Bubble Chart

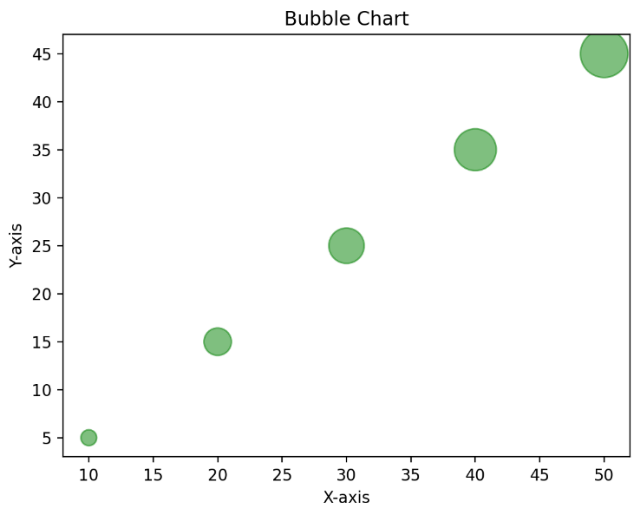
A bubble chart extends a scatter plot by adding a **third dimension through bubble size**. This allows you to represent **magnitude** along with position. Helps visualize multidimensional data in a compact way.Overlapping bubbles can sometimes make interpretation tricky.

Code :



```python
import matplotlib.pyplot as plt

# Sample data
x = [10, 20, 30, 40, 50]
y = [5, 15, 25, 35, 45]
sizes = [100, 300, 500, 700, 900]  # Bubble si

# Plot bubble chart
plt.scatter(x, y, s=sizes, alpha=0.5, c="green

plt.title("Bubble Chart")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

Output :



Description :

plt.scatter(x, y, s=sizes, ...) → Creates scatter plot points, but **s=sizes** makes the points bigger or smaller depending on the list values.

alpha=0.5 → Makes the bubbles slightly transparent so overlapping bubbles are visible.

c="green" → All bubbles are green. (You could also use a list of colors for variety).

Great for showing **three variables** at once:

X → position on X-axis

Y → position on Y-axis

Size → importance/magnitude

PLOTLY

Plotly is an open-source graphing library for Python that allows you to create interactive, publication-quality graphs. Unlike static libraries like Matplotlib, Plotly outputs charts that are zoomable, hoverable, and responsive.

It's widely used in:

- Data Science & Analytics (interactive dashboards, visual storytelling)
- Business Intelligence (KPI reports, sales dashboards)
- Machine Learning (model performance visualization)
- Web Applications (integrates with Flask, Dash)

Unique Features:

- Interactivity (zoom, hover tooltips, filtering).
- High-level API (plotly.express) for quick plotting.
- Low-level API (plotly.graph_objects) for advanced customization.
- Works seamlessly with **Jupyter Notebook, VS Code, or web apps**.

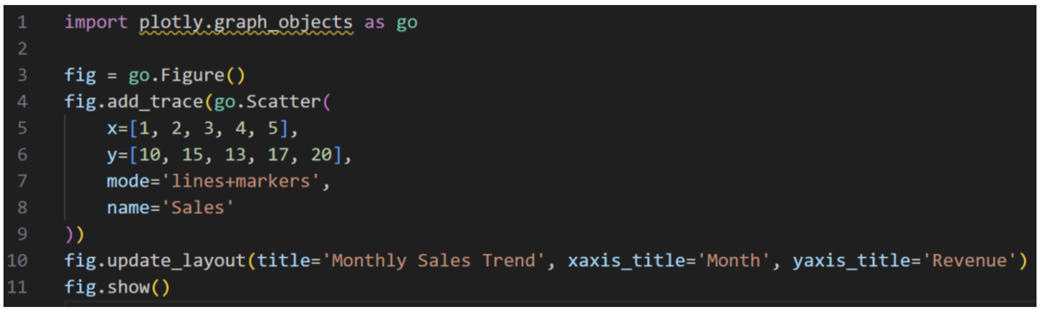
Plotly components are

1. **Figure (go.Figure)**
   - The main container that holds your chart.
2. **Traces**
   - Data points, lines, or bars added to the figure (go.Scatter, go.Bar, etc.).
3. **Layout**
   - Defines chart appearance (titles, axes, legend, background).
4. **Plotly Express (px)**
   - High-level wrapper functions like px.line(), px.bar().
5. **Graph Objects (go)**
   - Detailed, low-level objects for customization.

Line Chart

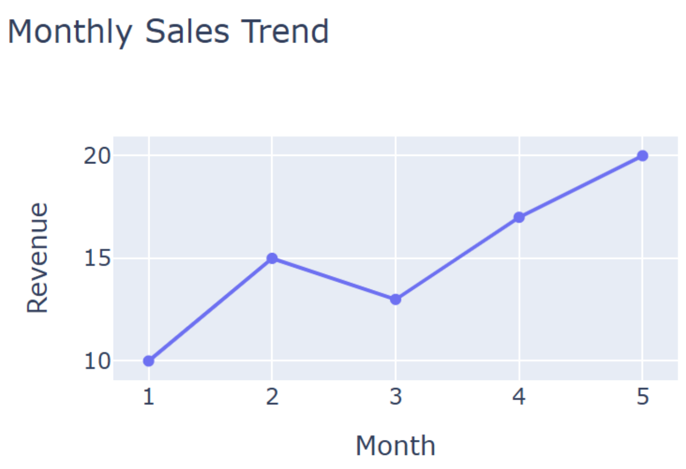
Plotly provides the built-in **line()** function in its high-level plotly.express library. Instead of manually drawing lines (like in low-level plotting libraries), this function automatically takes your dataset and maps the **x-axis**, **y-axis**, and optional styling (title, color, markers, etc.).Displays trends over continuous data (e.g., time series).Stock price over time, monthly sales growth.Predefined Function is plotly.express.line()

Code :



```python
import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(
    x=[1, 2, 3, 4, 5],
    y=[10, 15, 13, 17, 20],
    mode='lines+markers',
    name='Sales'
))
fig.update_layout(title='Monthly Sales Trend', xaxis_title='Month', yaxis_title='Revenue')
fig.show()
```

Output :



Description :

x="Month" and y="Revenue" define axes.

Scatter with mode='lines+markers' → line chart with dots.

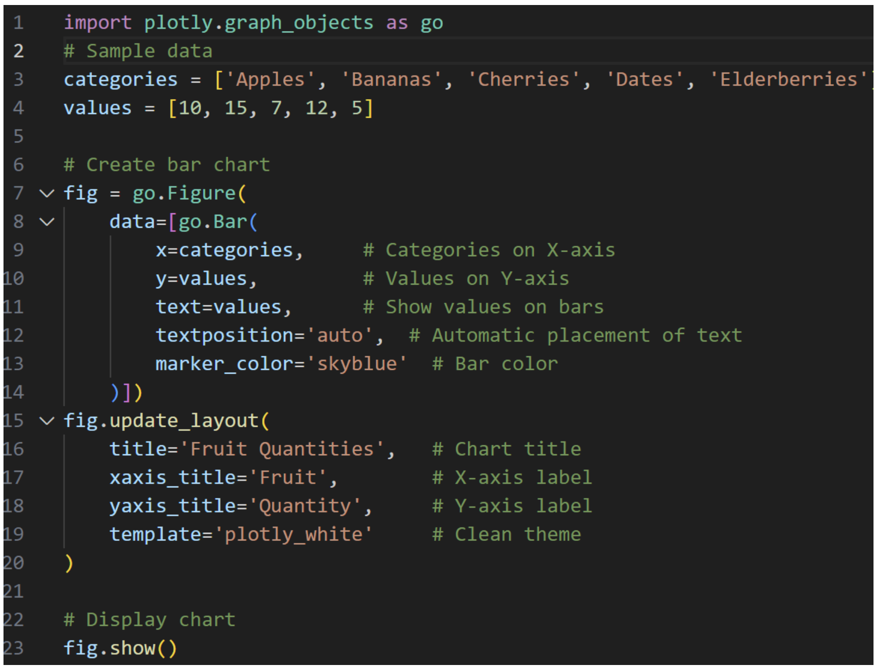
update_layout → customize title and axes.

Interactive hover tooltips show exact value

Bar Chart

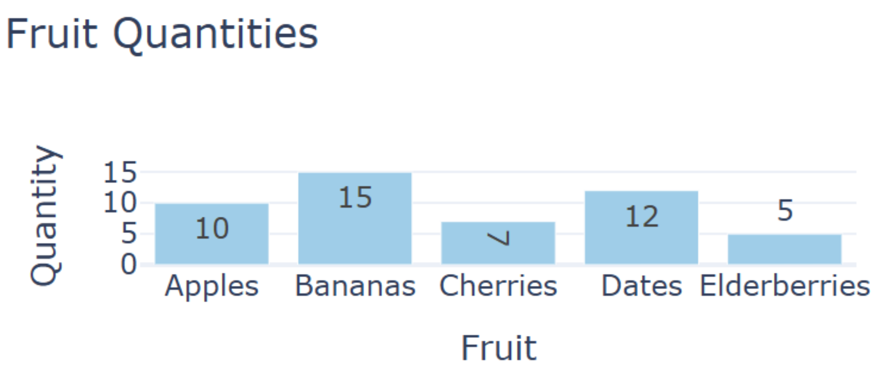
Extension of scatter plot where bubble size = third variable. Comparing countries by GDP (x), Life Expectancy (y), Population (bubble size). plotly.express.scatter() with size & color are predefined functions.

Code :



```python
import plotly.graph_objects as go
# Sample data
categories = ['Apples', 'Bananas', 'Cherries', 'Dates', 'Elderberries'
values = [10, 15, 7, 12, 5]

# Create bar chart
fig = go.Figure(
    data=[go.Bar(
        x=categories,     # Categories on X-axis
        y=values,         # Values on Y-axis
        text=values,      # Show values on bars
        textposition='auto',  # Automatic placement of text
        marker_color='skyblue'  # Bar color
    )])
fig.update_layout(
    title='Fruit Quantities',    # Chart title
    xaxis_title='Fruit',         # X-axis label
    yaxis_title='Quantity',      # Y-axis label
    template='plotly_white'      # Clean theme
)

# Display chart
fig.show()
```

Output :



Description :

go.Bar Specifies that we want a bar chart.

x → categories or labels (fruits).

y → numeric values.

text → numbers displayed on top of bars.

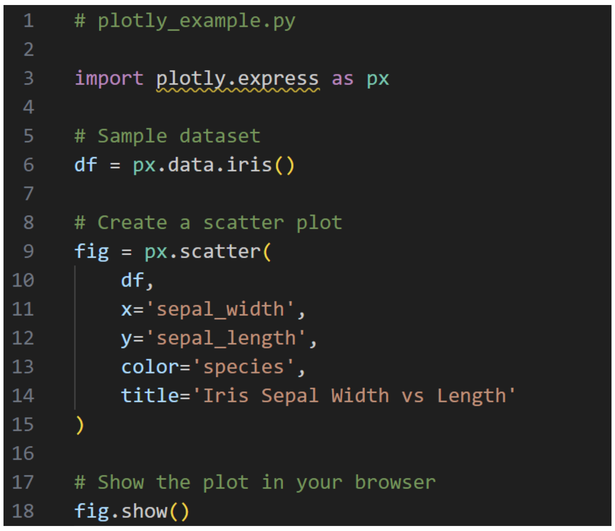
textposition='auto' → Plotly automatically positions the text.

marker_color → sets bar color.

Scatter Plot

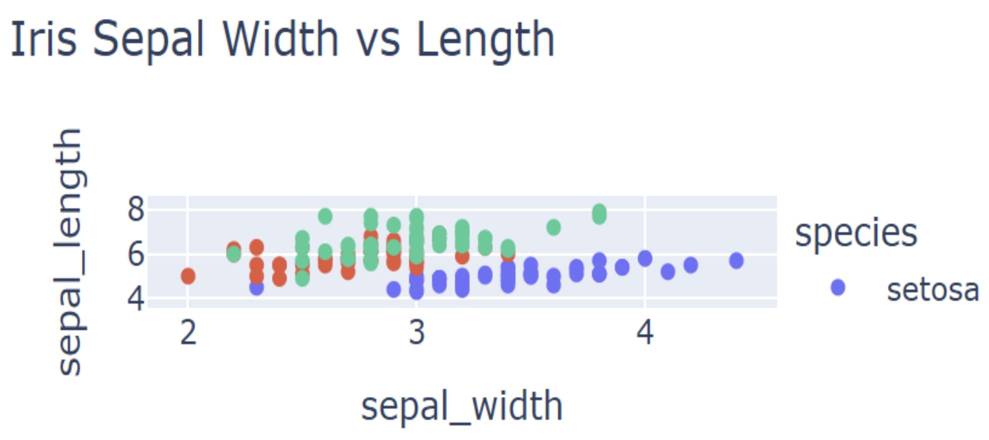
Shows relationships between two variables. Correlation between advertising spend and sales. The **scatter()** function automatically generates scatter plots.
It supports additional parameters to extend into **Bubble Charts** (size, color).

Code :



```python
# plotly_example.py

import plotly.express as px

# Sample dataset
df = px.data.iris()

# Create a scatter plot
fig = px.scatter(
    df,
    x='sepal_width',
    y='sepal_length',
    color='species',
    title='Iris Sepal Width vs Length'
)

# Show the plot in your browser
fig.show()
```

Output :



Description :

Each dot represents a data point.

Hovering reveals details.

x='sepal_width': data for the x-axis.

y='sepal_length': data for the y-axis.

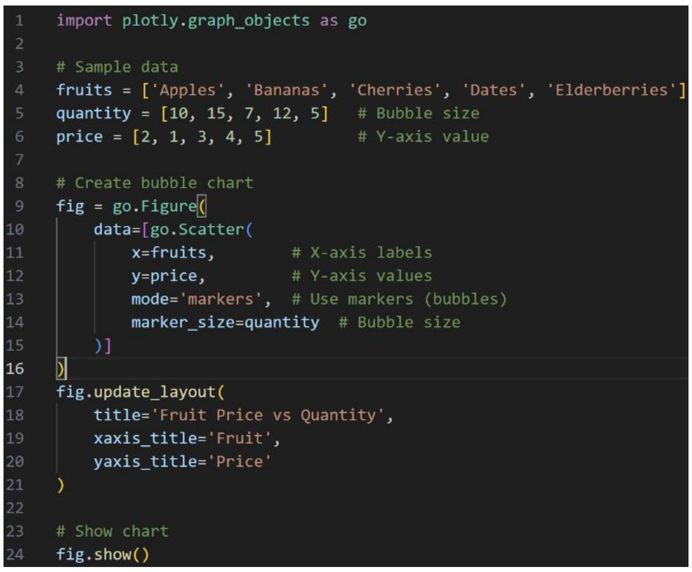
color='species': colors points by species (different categories).

title: chart title.

Bubble Chart

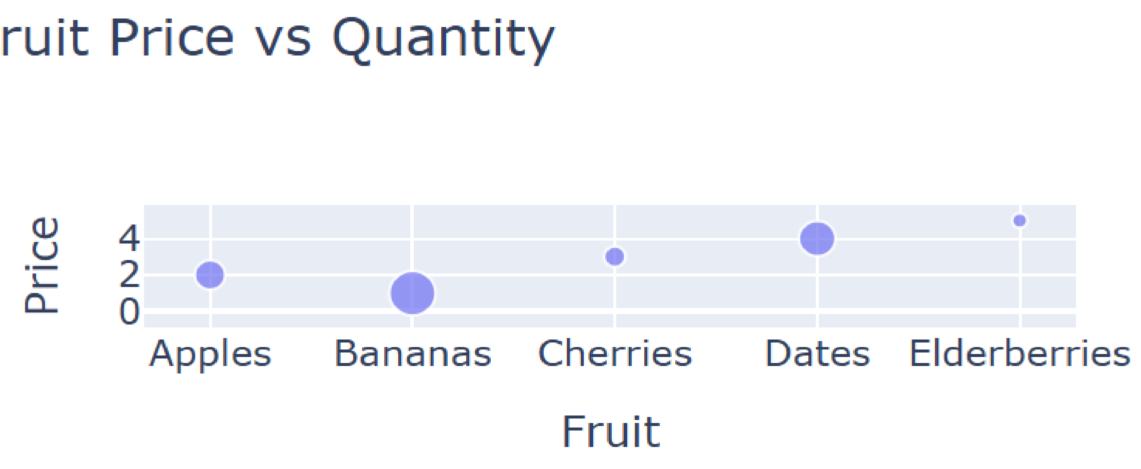
Plotly does not have a **separate function** for bubble charts. Instead, it uses the same **scatter()** function but includes additional parameters.

Code :



```python
import plotly.graph_objects as go

# Sample data
fruits = ['Apples', 'Bananas', 'Cherries', 'Dates', 'Elderberries']
quantity = [10, 15, 7, 12, 5]   # Bubble size
price = [2, 1, 3, 4, 5]         # Y-axis value

# Create bubble chart
fig = go.Figure(
    data=[go.Scatter(
        x=fruits,        # X-axis labels
        y=price,         # Y-axis values
        mode='markers',  # Use markers (bubbles)
        marker_size=quantity  # Bubble size
    )]
)
fig.update_layout(
    title='Fruit Price vs Quantity',
    xaxis_title='Fruit',
    yaxis_title='Price'
)

# Show chart
fig.show()
```

Output :



Description :

go.Scatter → Creates a scatter plot.

mode='markers' → Only points (bubbles) are shown.

x → Categories on X-axis.   y → Values on Y-axis.

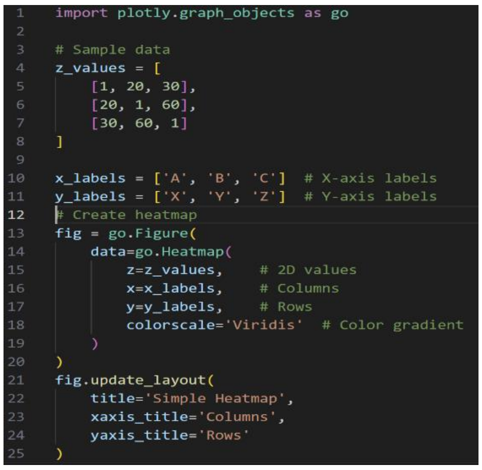
marker_size → Controls bubble size.

fig.show() → Displays the interactive bubble chart.

Heatmap

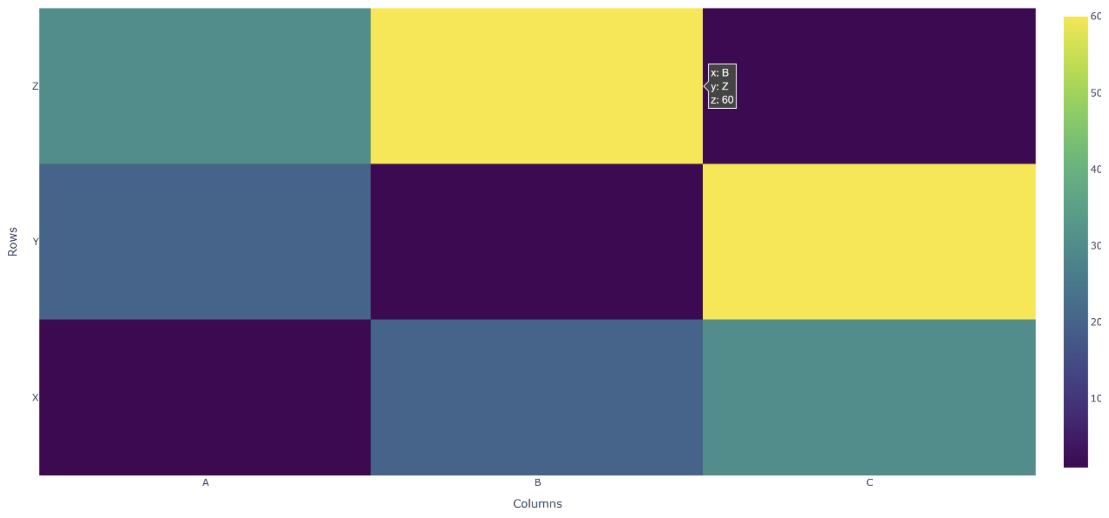
Visualizes data density using colors. Correlation matrix, website traffic by day/hour. plotly.express.imshow() or go.Heatmap() are predefined functions. No need to write loops to color cells. Just pass a 2D dataset.

Code :



```python
import plotly.graph_objects as go

# Sample data
z_values = [
    [1, 20, 30],
    [20, 1, 60],
    [30, 60, 1]
]

x_labels = ['A', 'B', 'C']  # X-axis labels
y_labels = ['X', 'Y', 'Z']  # Y-axis labels
# Create heatmap
fig = go.Figure(
    data=go.Heatmap(
        z=z_values,     # 2D values
        x=x_labels,     # Columns
        y=y_labels,     # Rows
        colorscale='Viridis'  # Color gradient
    )
)
fig.update_layout(
    title='Simple Heatmap',
    xaxis_title='Columns',
    yaxis_title='Rows'
)
```

Output :



Description :

z → 2D list of values (rows × columns).

x → Labels for columns.

y → Labels for rows.

colorscale → Sets the color gradient.

update_layout → Add chart title and axis labels.

fig.show() → Displays the interactive heatmap.

COMPARISON OF MATPLOTLIB AND PLOTLY

Matplotlib:

- Core Library: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It provides fine-grained control over every aspect of a plot, including colors, markers, line styles, and annotations.
- Flexibility: Users can create simple line and scatter plots as well as complex 3D plots, geographical maps, and custom charts.
- Mature Ecosystem: Matplotlib has been around for a long time and is widely used in the Python ecosystem. Many other libraries, such as Seaborn, are built on top of it.
- Integration: Easily integrates with NumPy, Pandas, SciPy, and other libraries, making it suitable for scientific computing and data analysis.
- Customization: Offers extensive customization options to control every element of a plot, including fonts, figure size, and subplots.

Advantages of Matplotlib:

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

Plotly:

- Interactive & Web-Based: Plotly is an interactive graphing library that creates web-ready plots with hover, zoom, and click functionality. It supports Python, R, and JavaScript.
- High-Level Interface: Provides easy-to-use high-level APIs via plotly.express for fast plotting, and plotly.graph_objects for advanced customization.
- Interactive Charts: Supports line, bar, scatter, bubble, 3D plots, maps, and dashboards with interactive features.
- Aesthetics & Themes: Built-in color palettes, templates, and styling options allow attractive visualizations without much effort.
- Integration: Works seamlessly with Pandas dataframes and web frameworks like Dash and Streamlit.
-

Advantages of Plotly:

- Fully interactive plots with hover, zoom, and click features.
- Ideal for web-based dashboards and data exploration.
- Built-in themes, color palettes, and templates for visually appealing charts.
- Supports 3D plotting, maps, and advanced visualizations.
- Easy integration with Pandas and web applications.

Overall:

- Matplotlib is preferred for static, publication-quality plots and fine-grained control.
- Plotly is preferred for interactive visualizations, dashboards, and web applications.
- Many users combine both libraries, leveraging Matplotlib for static plots and Plotly for interactive data exploration.