Visualization Library Documentation ,Plotly and Bokeh

In [ ]:
```
import pandas as pd
import numpy as np import plotly.express
as px from bokeh.plotting import figure,
show from bokeh.io import
output_notebook
```

PLOTLY

Plotly is a powerful, open-source Python library designed for creating interactive, publication-quality visualizations—right in your browser or notebook. It's widely used in data science, analytics, and machine learning for turning raw data into engaging visuals. **Key Features of Plotly :**

- Interactivity by default: Zoom, pan, hover tooltips, and legends are built-in.
- Wide chart support: Over 40 chart types including:Line, bar, scatter, pie, bubble,Box plots, histograms, heatmaps,3D plots, maps, financial charts
- Web integration: Easily embeddable in dashboards and web apps via Dash
- Notebook-friendly: Works seamlessly in Jupyter, Colab, and other notebook environments

Advanced Capabilities

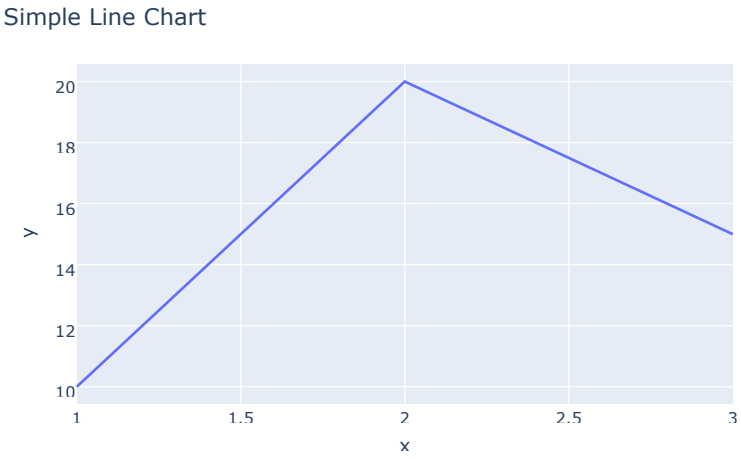
- Animations: Create dynamic transitions between chart states
- Subplots: Combine multiple charts into one figure
- Widgets: Add sliders, dropdowns, and buttons for user
- interaction
- Maps: Choropleth, tile maps, bubble maps with geographic data

3D Visuals: Surface plots, scatter3D, and camera controls

1.Line chart

Line charts are best when you want to see how something changes or flows. Great for showing trends like sales, temperature, or website visits.

In [ ]:
```
fig = px.line(x=[1, 2, 3], y=[10, 20, 15], title="Simple Line Chart")
fig.show()
```



Description:

1. px.line(...)

This is a function from Plotly Express used to create a line chart.

It takes two main arguments:

x=[1, 2, 3]: These are the values for the x-axis (e.g., time, categories, or indices).

y=[10, 20, 15]: These are the corresponding values for the y-axis (e.g., measurements, scores, or counts).

title="Simple Line Chart": Sets the chart's title.

2. fig = ...

The chart object is stored in the variable fig. This object contains all the data and layout info for the plot.

3. fig.show()

This command renders the chart in your notebook or browser.

The result is an interactive chart: you can hover over points, zoom in/out, and pan around.

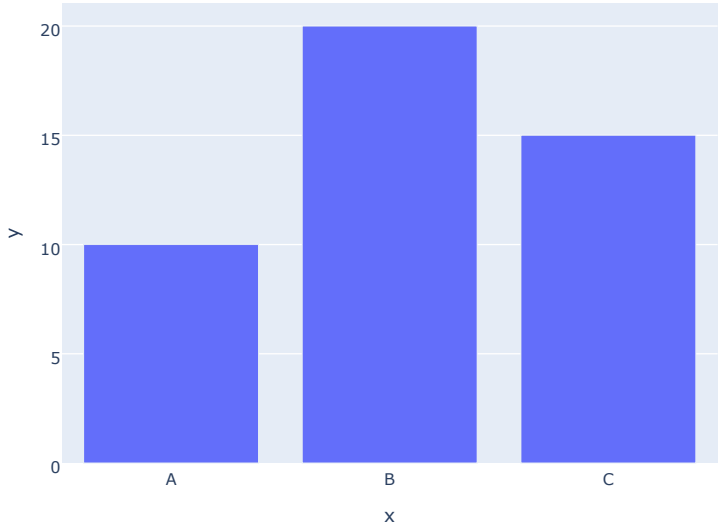
2.Bar chart

Bar charts are super useful when you want to compare values across categories. You can quickly see which category has the highest or lowest value.

The height (or length) of bars makes differences obvious at a glance.

```
# Step 1: Define the data
categories = ['A', 'B', 'C']
values = [10, 20, 15]

# Step 2: Create the bar chart fig = px.bar(x=categories,
y=values, title="Simple Bar Chart")

# Step 3: Show the chart
fig.show()
```

Simple Bar Chart



Descripton:

- **categories and values:** These are your x-axis labels and corresponding y-axis values.

- **px.bar(...):**

This function creates a bar chart. You pass in the x and y data, and optionally a title.

- **fig.show():**

Displays the chart in your notebook or browser. It’s interactive—you can hover, zoom, and pan.

3.Scatterplot

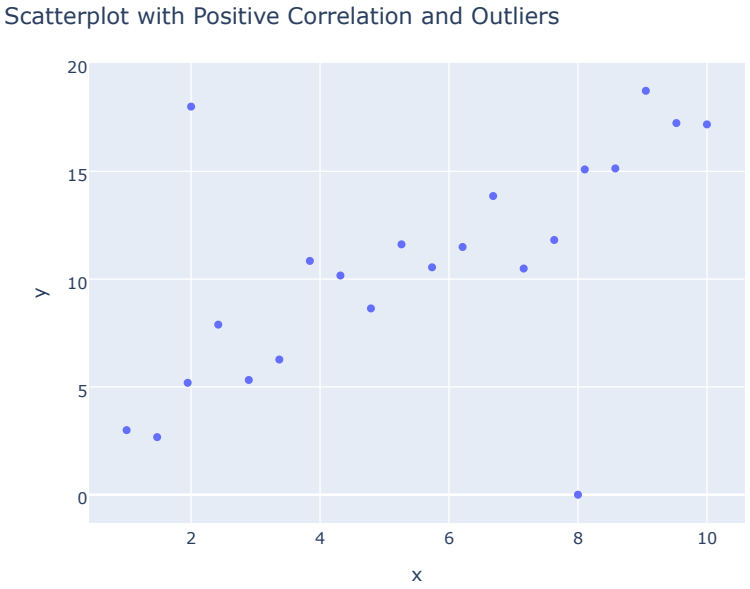
Scatterplots are super important when you want to see relationships between two variables.

You can spot trends like positive or negative correlations.

Easily see points that don’t fit the pattern.

In [ ]:

```
# Generate sample data with positive correlation np.random.seed(42) # for
reproducibility sample_x = np.linspace(1, 10, 20) sample_y = sample_x * 2 +
np.random.normal(0, 2, 20) # Positive correlation

# Add some outliers sample_x =
np.append(sample_x, [2, 8]) sample_y =
np.append(sample_y, [18, 0])

# Create a scatterplot fig = px.scatter(x=sample_x, y=sample_y,
title="Scatterplot with Positive Co

# Show the plot
fig.show()
```



Description:

- **np.random.seed(42)**:

Ensures reproducibility—same random values every time you run it.

- **sample_x**:

Generates 20 evenly spaced values from 1 to 10.

- **sample_y**:

Creates a positively correlated set of values (roughly y = 2x) with added Gaussian noise (mean=0, std=2).

-

- **px.scatter(...):** Creates an interactive scatterplot.
  fig.show(): Displays the plot in your notebook or browser

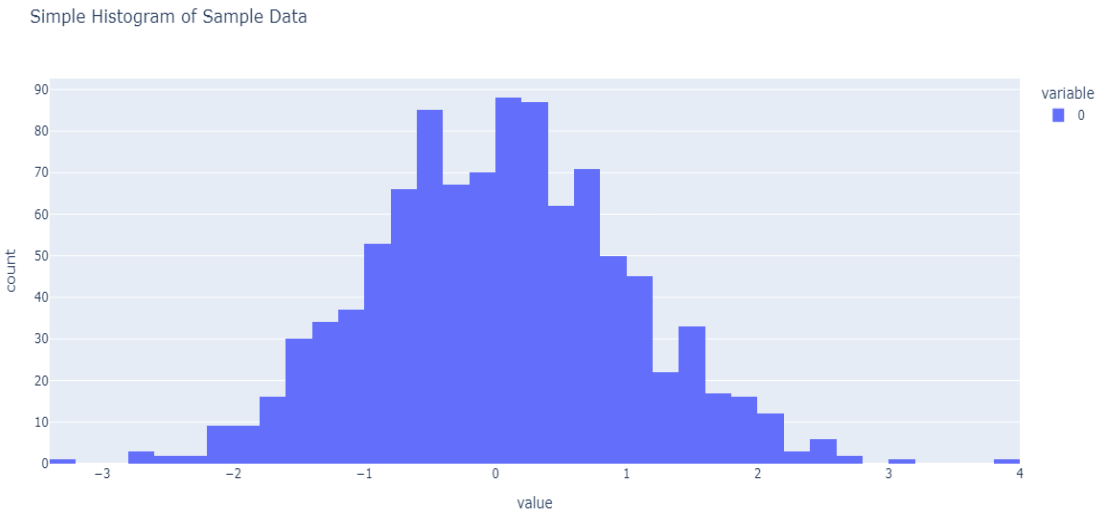
4.Histogram

Reveal Data Distribution Histograms show how data is spread across different ranges (called bins), helping you spot patterns like skewness, symmetry, or multimodal shapes.

Identify Central Tendency & Spread You can quickly see where most values cluster (mean, median) and how widely they vary (range, standard deviation).

In [ ]:
```
# Generate some sample data sample_data = np.random.randn(1000) # 1000 random
numbers from a standard no

# Create a simple histogram fig = px.histogram(sample_data, title="Simple
Histogram of Sample Data")

# Show the plot
fig.show()
```

Simple Histogram of Sample Data



Description:

fig = px.histogram(...) Uses Plotly Express to create

an interactive histogram.

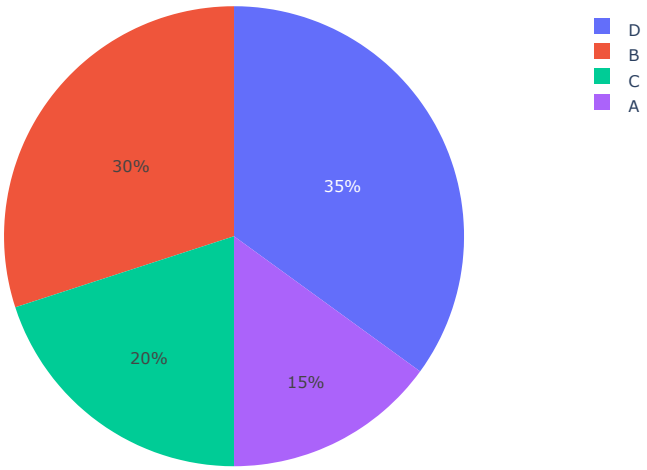
The histogram groups the data into bins and shows how many values fall into each range.

5.Pie Chart

The pie chart is an important type of data representation. It contains different segments and sectors in which each segment and sector of a pie chart forms a specific portion of the total(percentage). The sum of all the data is equal to 360°.

In [ ]:
```
# Sample data for a pie chart data =
{'Category': ['A', 'B', 'C', 'D'],
        'Values': [15, 30, 20, 35]}

sample_df = pd.DataFrame(data)

# Create a simple pie chart fig = px.pie(sample_df, values='Values',
names='Category', title='Simple Pie

# Show the plot
fig.show()
```

Simple Pie Chart of Sample Data



Description: •

px.pie(...):

This is the Plotly Express function for creating pie charts. •

sample_df:

A pandas DataFrame that contains at least two columns: one for categories and one for values.

- **values='Values':**

Specifies which column contains the numerical data that determines the size of each pie slice.

- **names='Category':**

Specifies which column contains the labels for each slice.

- **title='...':**

Adds a title to the chart for clarity.

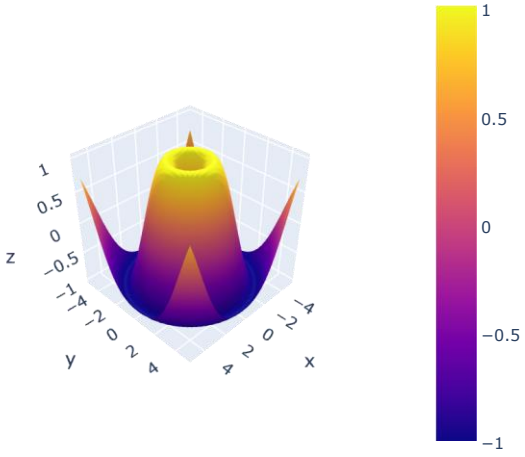
6.Surface Plot

Visualize Relationships Between Three Variables: Surface plots show how a dependent variable (Z) changes based on two independent variables (X and Y).

This is especially useful when you're modeling functions or analyzing multidimensional data.

In [ ]:
```
# Generate sample data for a 3D surface plot
x = np.linspace(-5, 5, 50) y = np.linspace(-
5, 5, 50) x, y = np.meshgrid(x, y) z =
np.sin(np.sqrt(x**2 + y**2))

# Create a 3D surface plot fig =
go.Figure(data=[go.Surface(z=z, x=x, y=y)])

fig.update_layout(title='Simple 3D Surface Plot', autosize=False,
width=500, height=500,                    margin=dict(l=65, r=50,
b=65, t=90)) fig.show()
```

Simple 3D Surface Plot



Description:

-

- **go.Figure(...):** Initializes a Plotly figure object. **go.Surface(...):** Creates a 3D surface plot. **z:** A 2D array of height values (e.g., elevation, intensity).

  x, y: 2D arrays or vectors defining the grid coordinates.

  - **update_layout(...):** Customizes the figure’s appearance. **title:** Sets the chart title.

  autosize=False: Disables automatic resizing. **width &**

  height: Fixes the plot size to 500×500 pixels.

  margin: Controls spacing around the plot (left, right, bottom, top).

BOKEH

Bokeh is a Python library for creating interactive visualizations that can be easily embedded into web applications. It’s designed to handle large datasets and produce rich, browser-based graphics without needing JavaScript. **Core Features:**

- Interactivity: Built-in tools like zoom, pan, hover, and selection make plots dynamic.

- Web Integration: Outputs HTML, JSON, or server-based apps for seamless web embedding.
- Notebook Support: Works smoothly in Jupyter and Colab with output_notebook() for inline rendering.
- High Performance: Optimized for large datasets using WebGL and efficient rendering pipelines.

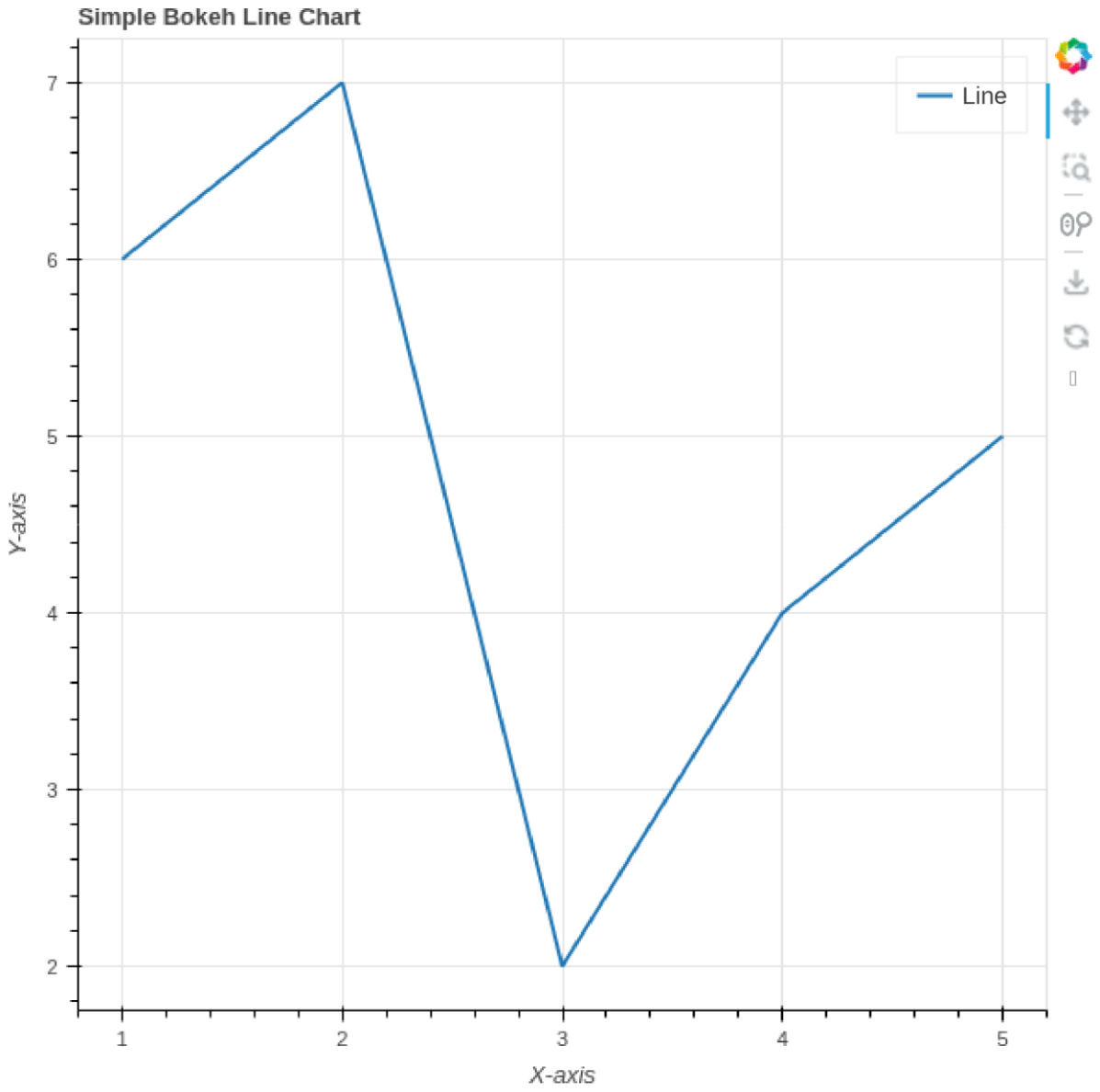
Why Bokeh?

- Customizable: Every element—axes, legends, tooltips—can be
- styled or scripted. • Linked Plots: Sync multiple plots for coordinated views.

  Streaming & Real-Time: Supports live data updates via ColumnDataSource.
- Widgets & Layouts: Add sliders, dropdowns, and buttons for dashboard-like interactivity.

1.Line Chart

In [ ]:
```
from bokeh.plotting import figure, show
from bokeh.io import output_notebook import
numpy as np

# Enable inline plotting in the notebook output_notebook()

# Sample data x =
[1, 2, 3, 4, 5] y =
[6, 7, 2, 4, 5]

# Create a new plot with a title and axis labels p = figure(title="Simple
Bokeh Line Chart", x_axis_label="X-axis", y_axis_la



# Add a line glyph to the plot
p.line(x, y, legend_label="Line", line_width=2)

# Show the plot
show(p)
```



Description:

- **output_notebook()** Purpose: Activates Bokeh's inline rendering.

  Effect: Ensures that the plot appears inside the notebook rather than opening in a separate browser tab.

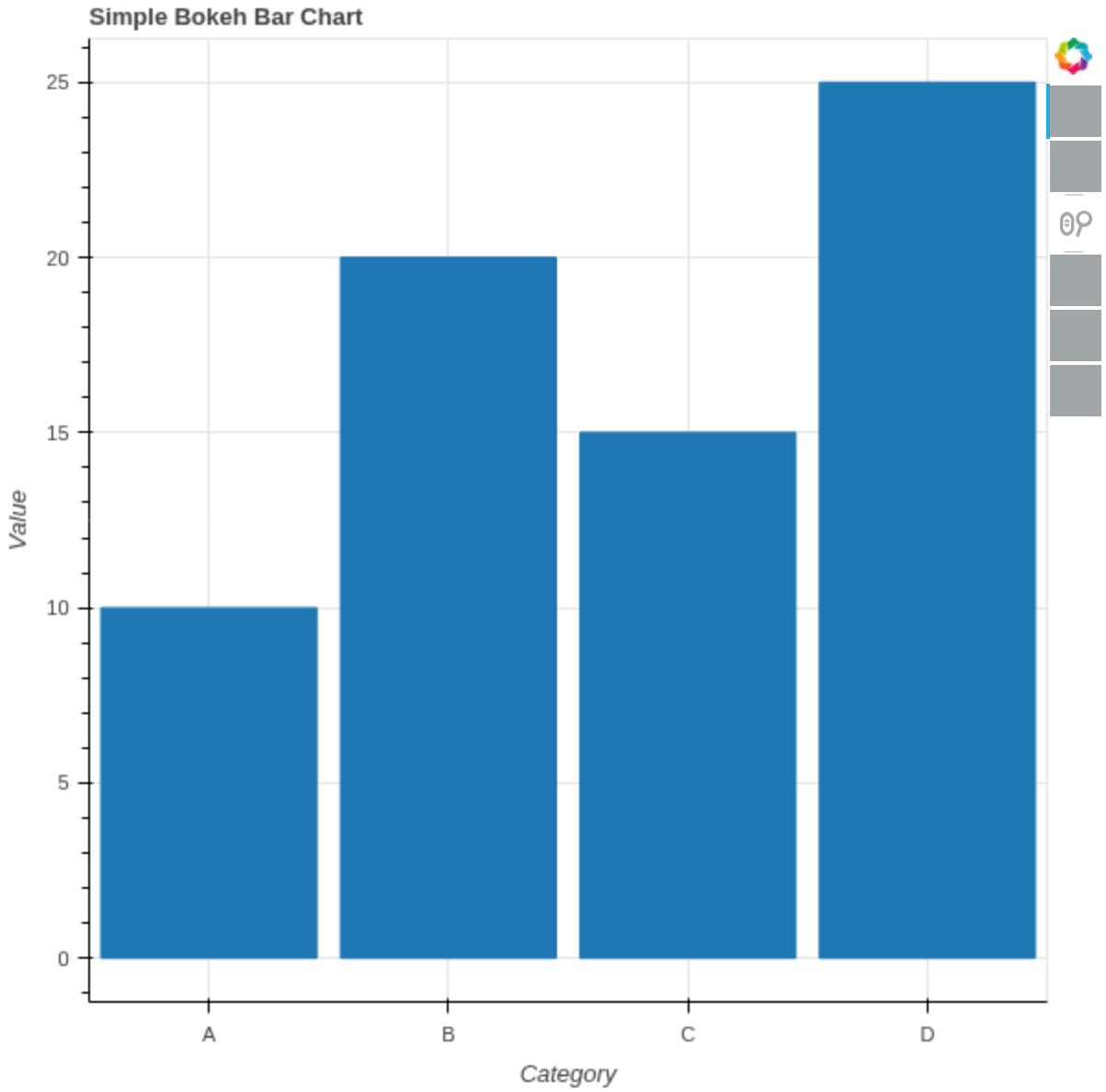
  - **p = figure(...)** Creates a new Bokeh plot object.
- **title="Simple Bokeh Line Chart":** Sets the chart title.
- **x_axis_label="X-axis"** and **y_axis_label="Y-axis":** Labels the axes for clarity.

- **p.line(x, y, legend_label="Line", line_width=2)** Adds a line glyph to the plot. **x and y:** The data points to be
- connected. **legend_label="Line":** Adds a legend entry for
- this line. **line_width=2:** Makes the line thicker for better
- visibility.

2.Bar Chart

In [ ]:

```
# Enable inline plotting in the notebook output_notebook()

# Sample data data = {'categories': ['A',
'B', 'C', 'D'],
        'values': [10, 20, 15, 25]} sample_df
= pd.DataFrame(data)

# Create a new plot with a title and axis labels # Use x_range to define the
categories for the x-axis p = figure(x_range=sample_df['categories'],
title="Simple Bokeh Bar Chart",            x_axis_label="Category",
y_axis_label="Value")

# Add a vertical bar glyph to the plot
p.vbar(x=sample_df['categories'], top=sample_df['values'], width=0.9)

# Show the plot
show(p)
```



- **figure(...):** Initializes a new Bokeh plot.
- **x_range=sample_df['categories']:** Sets the x-axis to use categorical values (like 'A', 'B', 'C') from the DataFrame. **title="...":** Adds a title to the chart.
- **x_axis_label="Category" and y_axis_label="Value":**Labels the axes for clarity. • **vbar(...):** Adds vertical bars to the plot.

  x=...: Positions each bar along the x-axis using the category labels.

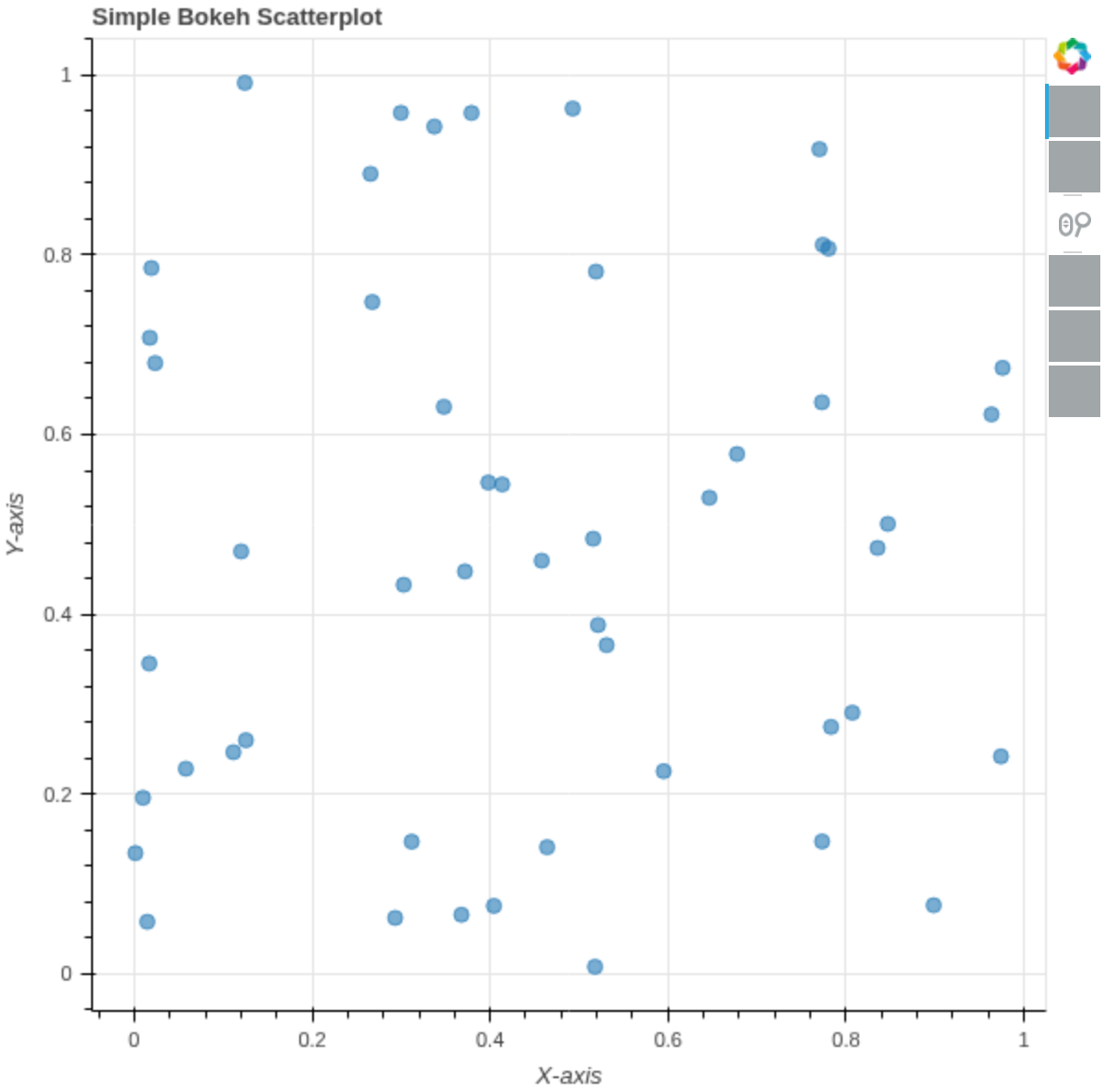
  top=...: Sets the height of each bar using the corresponding values.

width=0.9: Controls the thickness of each bar (0.9 means almost full width of each category slot).

3.Scatterplot

In [ ]:

```
# Enable inline plotting in the notebook output_notebook()

# Sample data x =
np.random.rand(50) y =
np.random.rand(50)

# Create a new plot with a title and axis labels p = figure(title="Simple
Bokeh Scatterplot", x_axis_label="X-axis", y_axis_l

# Add a scatter glyph to the plot
p.scatter(x, y, size=8, alpha=0.6)

# Show the plot
show(p)
```



Description:

- **p = figure(title="Simple Bokeh Scatterplot", x_axis_label="Xaxis", y_axis_label="Y-axis")** Creates a new plot object called
  p.

  title: Sets the title of the plot.

  x_axis_label and y_axis_label: Label the X and Y

  axes respectively. This sets up the canvas for your

  scatterplot.

- **p.scatter(x, y, size=8, alpha=0.6)** Adds a scatter glyph (i.e., a collection of dots) to the plot.

x and y: These are lists or arrays of coordinates for each point.

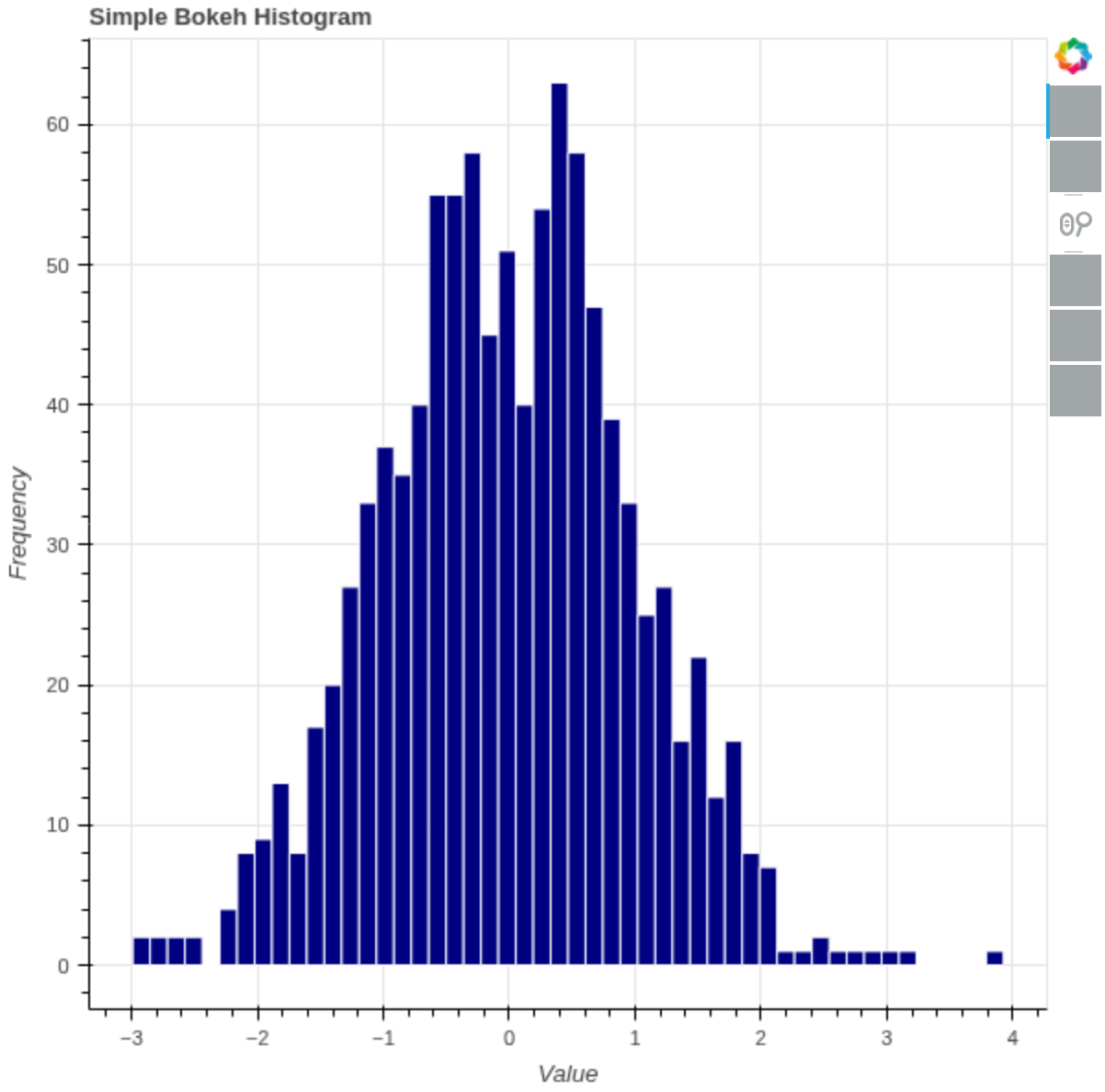
size=8: Sets the diameter of each point.

alpha=0.6: Controls transparency (0 is fully transparent, 1 is fully opaque). This helps when points overlap.

4.Histogram

In [ ]:

```
# Enable inline plotting in the notebook output_notebook()

# Generate some sample data sample_data = np.random.randn(1000) # 1000 random
numbers from a standard no

# Calculate histogram data hist, edges =
np.histogram(sample_data, bins=50)

# Create a new plot p = figure(title="Simple Bokeh Histogram",
x_axis_label="Value", y_axis_labe

# Add a quad glyph to represent the bars of the histogram
p.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:],
fill_color="navy", line_color="white")

# Show the plot
show(p)
```



Description:

sample_data: A NumPy array of numerical values.

np.histogram(...): Calculates how many values fall into each of 50 bins (ranges). **hist:** Contains the frequency (count) of values in each bin.

edges: Contains the boundaries of the bins (length = 51 for 50 bins). **figure(...):** Initializes a new Bokeh plot. **title:** Sets the plot title. **x_axis_label and y_axis_label:** Label the axes for clarity. **quad(...):** Adds rectangular bars (quads) to the plot.

top=hist: Height of each bar corresponds to the frequency.

bottom=0: Bars start from y=0. **left=edges[:-1]:** Left edge of each bin. **right=edges[1:]:** Right edge of each bin.

fill_color="navy": Fills bars with navy blue.

line_color="white": Outlines bars in white for contrast.

5. Geo/Map vizualization

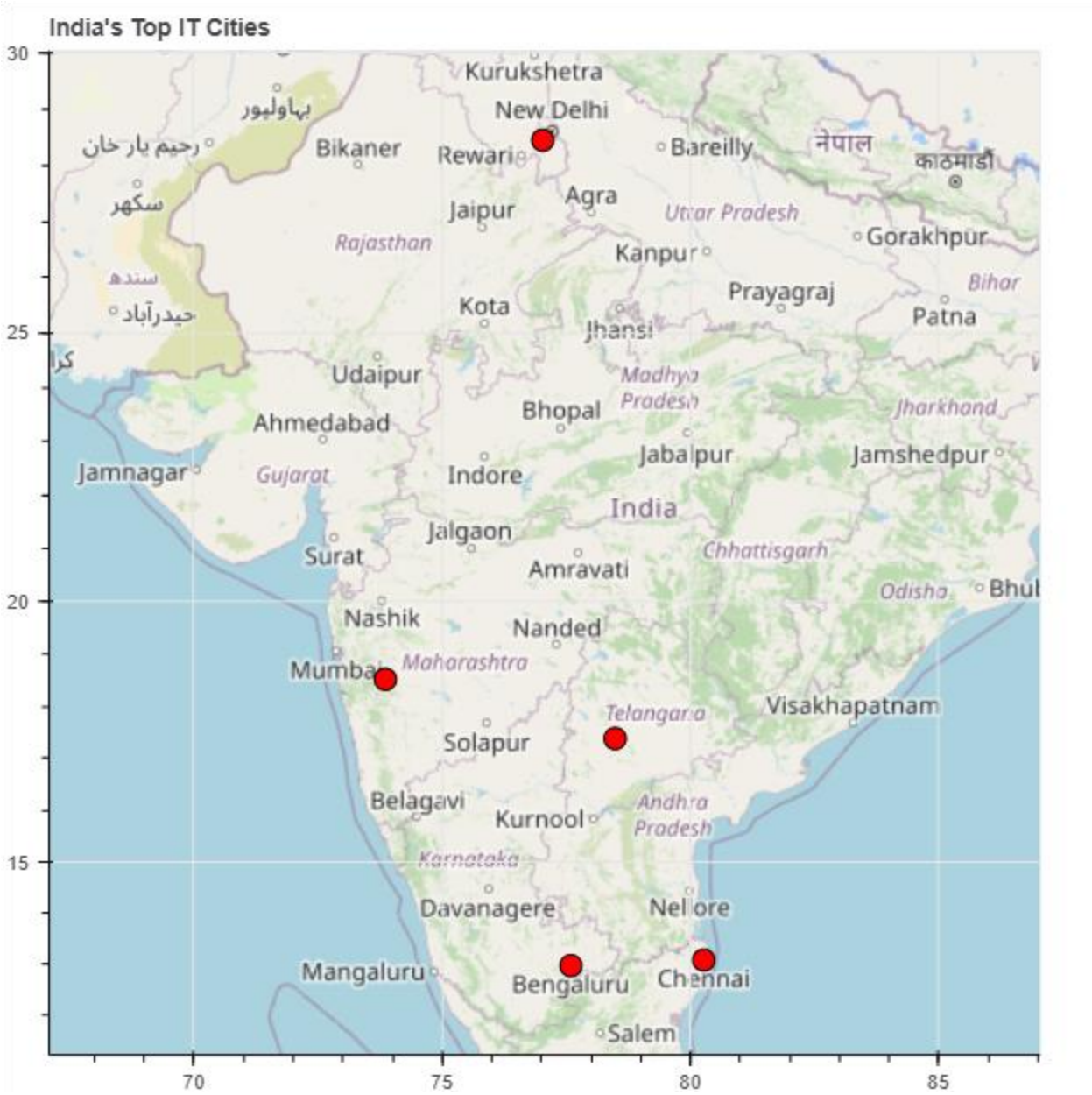
Tile renderers (p.add_tile) let you plot points/paths over realworld maps (OpenStreetMap, Google-style tiles). Bokeh integrates with geographic data very well.

In [ ]:

```
from bokeh.io import show, output_notebook from
bokeh.plotting import figure, ColumnDataSource from
bokeh.models import WMTSTileSource from bokeh.models
import HoverTool import math import pandas as pd
output_notebook()  # For Jupyter

# --- 1. Sample data: Top IT cities in India --cities =
pd.DataFrame({
    'city': ['Bengaluru', 'Hyderabad', 'Pune', 'Chennai', 'Gurugram'],
    'lat': [12.9716, 17.3850, 18.5204, 13.0827, 28.4595],
    'lon': [77.5946, 78.4867, 73.8567, 80.2707, 77.0266]
})

# --- 2. Convert lat/lon to Web Mercator --def wgs84_to_web_mercator(lon,
lat):     k = 6378137     x = lon * (k * math.pi / 180.0)     y =
math.log(math.tan((90 + lat) * math.pi / 360.0)) * k     return (x, y)
cities["x"], cities["y"] = zip(*cities.apply(lambda row: wgs84_to_web_mercat

# --- 3. Create ColumnDataSource for interactivity --source =
ColumnDataSource(cities)

# --- 4. Create figure with map background --tile_provider =
WMTSTileSource(url="http://c.tile.openstreetmap.org/{Z}/{X}/
```

```
p = figure(x_range=(cities["x"].min()-2e5, cities["x"].max()+2e5),
y_range=(cities["y"].min()-2e5, cities["y"].max()+2e5),
x_axis_type="mercator", y_axis_type="mercator",            title="India's
Top IT Cities")

p.add_tile(tile_provider)

# --- 5. Add city markers ---
p.circle(x="x", y="y", size=12, fill_color="red", line_color="black", source

# --- 6. Add hover tool --hover = HoverTool() hover.tooltips =
[("City", "@city"), ("Lat", "@lat"), ("Lon", "@lon")]
p.add_tools(hover) show(p)
```

```
/tmp/ipython-input-1641445211.py:40: BokehDeprecationWarning:

'circle() method with size value' was deprecated in Bokeh 3.4.0 and will be r
emoved, use 'scatter(size=...) instead' instead.
```



** Description:**

This Bokeh visualization displays India's top IT cities on an interactive OpenStreetMap background. Each city is plotted as a red marker with hover tooltips showing the city name and coordinates.

steps:

- step: "Prepare Data" details:
  - Define a DataFrame with city names, latitude, and longitude.
  - Cities: Bengaluru, Hyderabad, Pune, Chennai, Gurugram.
- step: "Convert Coordinates" details:

- Use WGS84 (lat/lon) to Web Mercator conversion.
- Formula ensures compatibility with tile-based maps.
- Adds 'x' and 'y' columns for plotting.

- step: "Setup Data Source" details:
  - Create a ColumnDataSource from the DataFrame.
  - Enables interactivity (hover tooltips, linked updates).
- step: "Create Map" details:
  - Use WMTSTileSource with OpenStreetMap URL.
  - Define Bokeh figure with Mercator axes.
  - Set initial x/y ranges to focus on India region.
- step: "Plot Markers" details:
  - Add red circle markers at each city’s (x, y) location.
  - Marker size: 12px, black border.
- step: "Enable Hover" details:
  - Add HoverTool to show:
    - City name (@city)
    - Latitude (@lat)
    - Longitude (@lon).

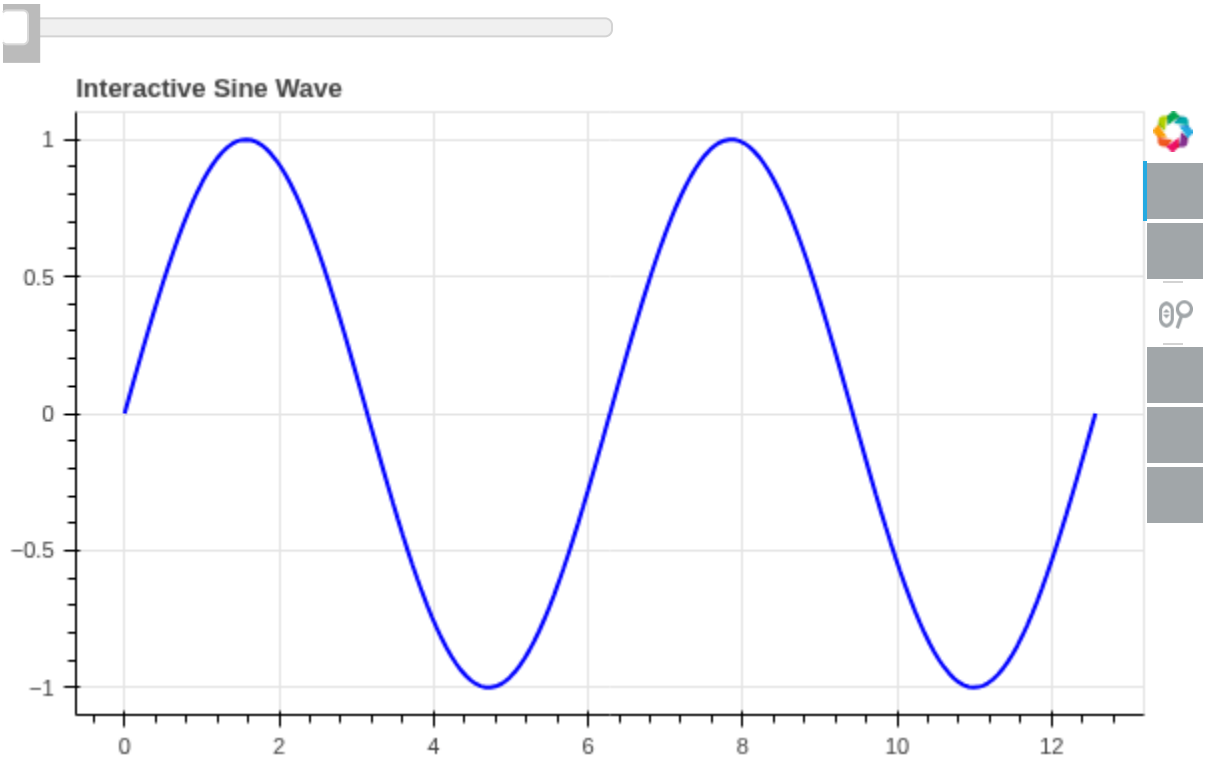
6.Interactive Plot with Slider

Sliders, dropdowns, checkboxes directly tied to plots without needing Dash/Streamlit. Example: dynamically update a sine wave with a frequency slider.

In [ ]:

```
import numpy as np from bokeh.plotting import figure,
show, ColumnDataSource from bokeh.io import
output_notebook, curdoc from bokeh.models import Slider
from bokeh.layouts import column from
bokeh.models.callbacks import CustomJS output_notebook()
# for Jupyter

# --- 1. Create sample data --x =
np.linspace(0, 4*np.pi, 200) y = np.sin(x)
source = ColumnDataSource(data=dict(x=x, y=y))

# --- 2. Create figure --p = figure(height=350, width=600,
title="Interactive Sine Wave")
p.line('x', 'y', source=source, line_width=2, color="blue")

# --- 3. Slider widget --slider = Slider(start=1, end=10, value=1,
step=1, title="Frequency")

# --- 4. Define callback in JavaScript (no server needed) ---



callback = CustomJS(args=dict(source=source, slider=slider), code="""
const data = source.data;     const freq = slider.value;     const x =
data['x'];     const y = data['y'];     for (let i = 0; i < x.length;
i++) {         y[i] = Math.sin(freq * x[i]);     }
source.change.emit();
""") slider.js_on_change('value',

callback)

# --- 5. Layout --layout =
column(slider, p)
show(layout)
```

Frequency: **1**



** Description:**

A simple interactive visualization created using Bokeh. The plot displays a sine wave and allows the user to change its frequency using a slider widget.

steps:

- step: "Create Data" details:
  - Generate x values from 0 to 4π.
  - Compute y = sin(x).
  -
- Store data in a ColumnDataSource for interactivity.
  step: "Build Figure" details:
  - Initialize a Bokeh figure (height=350, width=600).
  - Add a line plot with x and y values.
  - Style: blue line, width 2px.
- step: "Add Slider Widget"
  details:
  - Create a slider with values from 1 to 10.
  - Slider controls the frequency of the sine wave.
- step: "Define Callback" details:

- JavaScript CustomJS callback updates y-values.
- Formula: y[i] = sin(freq * x[i]).
- Emit changes to refresh the plot dynamically.

- step: "Combine Layout" details:

- Arrange slider above the plot using column layout. Display the interactive visualization with show().

Comparison:

Plotly:

Overview: Plotly is a powerful, high-level library for creating interactive and web-based visualizations.

It is widely used for dashboards, data apps, and publishing interactive charts online.

Plotly supports both Python and JavaScript, making it versatile across platforms. **Features:**

- Interactivity: Built-in interactivity with zooming, panning, hovering, and tooltips.
  - Ease of Use: High-level API makes it quick to create professional visuals with minimal code. - Variety of Charts: Supports a wide range of charts (line, bar, scatter, 3D plots, maps, financial charts, etc.).
  - Dash Integration: Works seamlessly with Plotly Dash for creating full-featured web dashboards. - Export Options: Charts can be exported as static images (PNG, PDF) or shared as interactive HTML.

Advantages:

- Easy to use with simple syntax.
- Great for interactive dashboards and web
- applications. Large library of chart types, including

- 3D and map visualizations. • Excellent support for hover effects and tooltips.
- Cross-platform (works in Jupyter, browser, mobile)

Best for: Rapid development of interactive dashboards, data storytelling, and sharing visualizations online with minimal effort.

Bokeh:

Overview:

Bokeh is a Python interactive visualization library focused on providing elegant, interactive plots for modern web browsers. It is highly flexible and allows fine-grained customization. **Features:**

- Interactivity: Provides interactive tools (zoom, pan, hover) and supports linking multiple plots. • Widgets: Built-in support for sliders, dropdowns, and buttons to create interactive dashboards without external tools. • Streaming Data: Can handle real-time data updates with streaming and patching. • Flexibility: Allows integration with Flask/Django for custom apps. • Extensions: Works with Holoviews and Datashader for large datasets and advanced visualizations.

Advantages: - High flexibility with strong Python-first design. Can handle large/streaming datasets efficiently. - Direct integration of widgets for interactive apps. - Suitable for custom web-based dashboards without needing another framework. - Strong support for linked brushing and multiple synchronized plots.

Best_for: > Interactive data exploration, scientific dashboards, and applications needing streaming or linked visualizations.

Summary:

Key_differences:

- Plotly is easier for beginners and quick dashboarding, while Bokeh is more customizable and flexible for developers. • Plotly integrates with Dash for apps, while Bokeh has its own widget system for interactivity. • Plotly excels in variety (3D, maps, financial charts), while Bokeh excels in linking, streaming, and scientific workflows. • Both generate browser-based visualizations, but Plotly emphasizes simplicity, while Bokeh emphasizes flexibility.

Usage_recommendation:

Use Plotly if you want fast, attractive dashboards and ease of use. Use Bokeh if you need highly customized, scientific, or streaming interactive visualizations.
