Python Visualization Libraries: Matplotlib and Plotly

Table of Contents

Library Overview

Matplotlib

Matplotlib is the grandfather of Python visualization libraries, offering a comprehensive set of tools for creating static, interactive, and publication-quality visualizations. It provides fine-grained control over every element of a plot, making it highly customizable but sometimes verbose in syntax. Matplotlib serves as the foundation for many other visualization libraries and is particularly well-suited for scientific publications and reports.

Plotly

Plotly is a modern visualization library designed with interactivity at its core. It excels at creating web-based, interactive visualizations that can be easily shared or embedded in web applications. Plotly offers a high-level interface for complex visualizations while maintaining flexibility through its lower-level graph objects. It's particularly powerful for dashboards, data exploration, and sharing insights with non-technical audiences.

Graph Types

Matplotlib

1. Line Plot

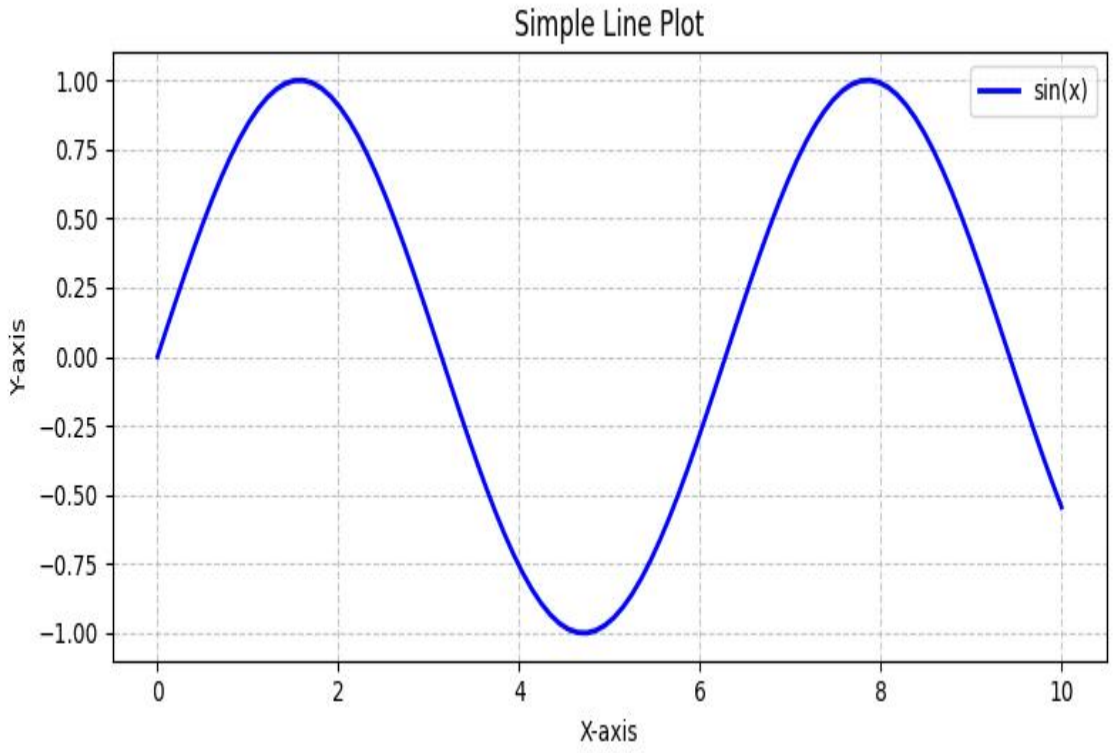
Description: Line plots show relationships between continuous variables, especially useful for time series data or showing trends.

Use Case: Tracking stock prices, temperature changes over time, or any continuous variable measured sequentially.

Code Example:

python

```
import matplotlib.pyplot as pltimport numpy as np
x = np.linspace(0, 10, 100)
y = np.sin(x)
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x, y, color='blue', linestyle='-', linewidth=2,
label='sin(x)')
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Line Plot')
ax.legend()ax.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()plt.show()
```



2. Scatter Plot

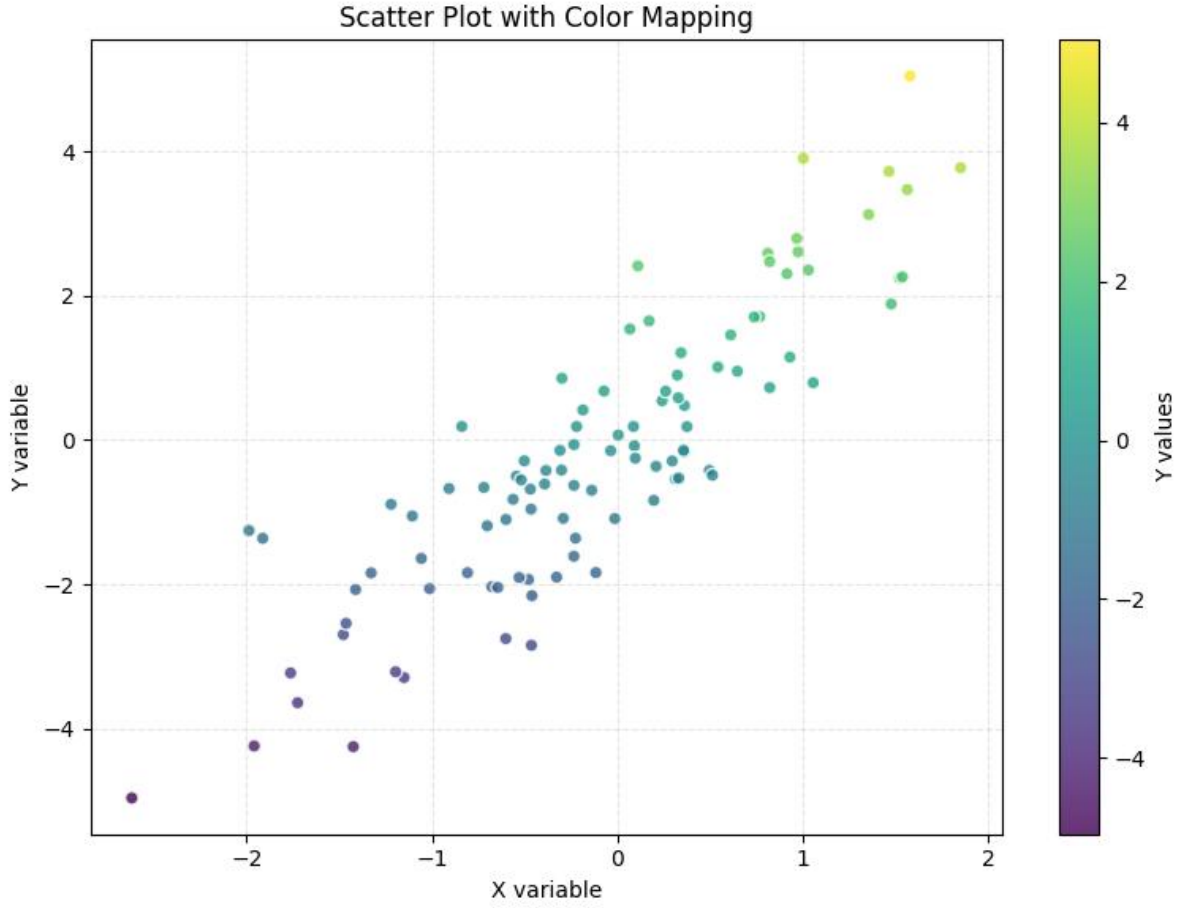
Description: Scatter plots display the relationship between two numerical variables, with each point representing an observation.

Use Case: Examining correlations between variables like height vs. weight, or investigating relationships in scientific data.

Code Example:

python

```python
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(42)
x = np.random.normal(size=100)y = 2*x +
np.random.normal(size=100)
fig, ax = plt.subplots(figsize=(8, 6))
scatter = ax.scatter(x, y, c=y, cmap='viridis', alpha=0.8,
edgecolors='w')
cbar = plt.colorbar(scatter)
cbar.set_label('Y values')
ax.set_xlabel('X variable')
ax.set_ylabel('Y variable')
ax.set_title('Scatter Plot with Color Mapping')
ax.grid(True, linestyle='--', alpha=0.3)
plt.tight_layout()
plt.show()
```



3. Bar Chart

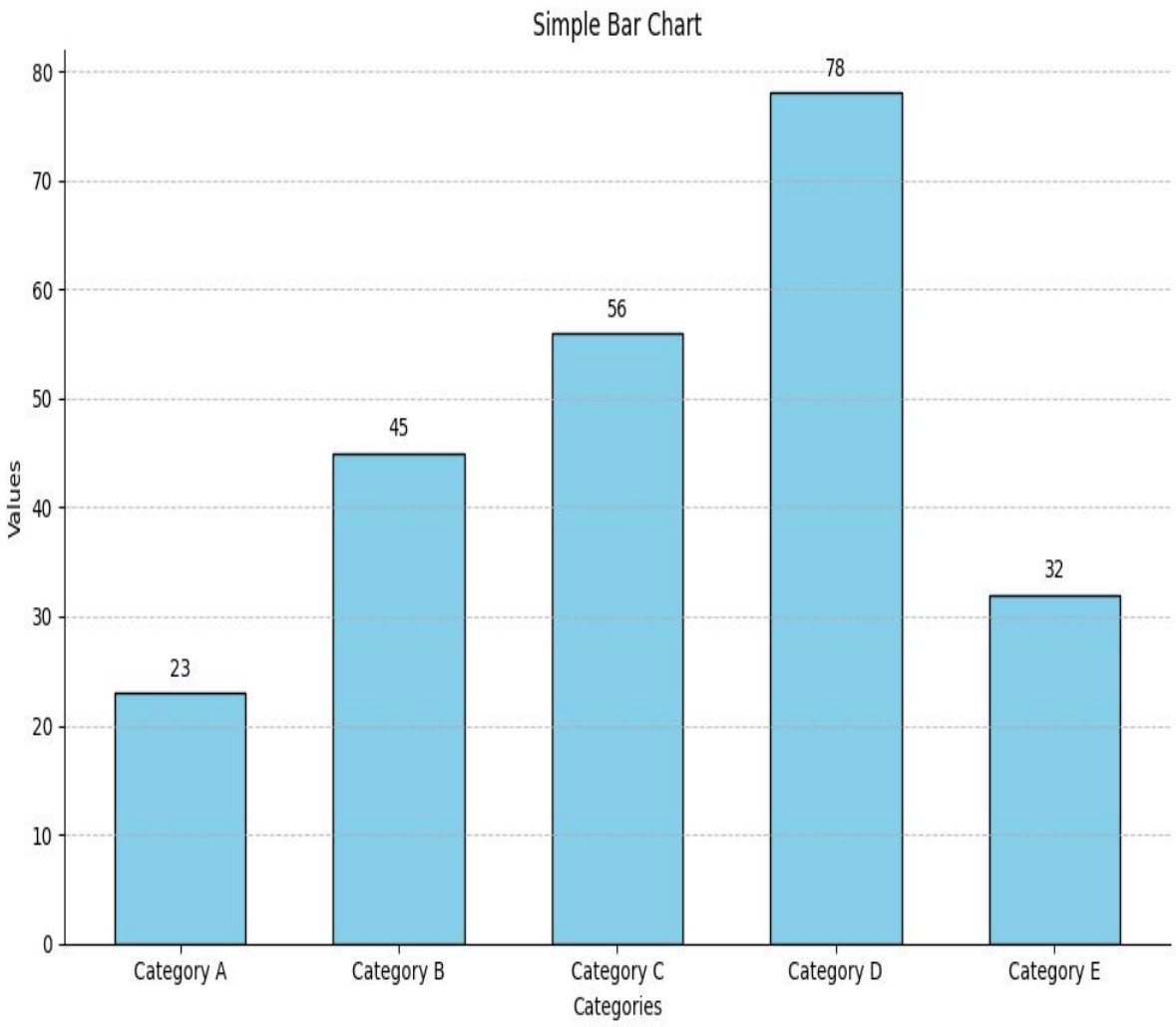
Description: Bar charts display categorical data with rectangular bars proportional to the values they represent.

Use Case: Comparing sales across different products, survey results, or any comparison between distinct categories.

Code Example:

python

```python
import matplotlib.pyplot as plt
import numpy as np
categories = ['Category A', 'Category B', 'Category C', 'Category D', 'Category E']
values = [23, 45, 56, 78, 32]
fig, ax = plt.subplots(figsize=(10, 6))
bars = ax.bar(categories, values, color='skyblue', edgecolor='black', width=0.6)
for bar in bars:    height = bar.get_height()
ax.text(bar.get_x() + bar.get_width()/2., height + 1, f'{height}', ha='center', va='bottom')
ax.set_xlabel('Categories')
ax.set_ylabel('Values')
ax.set_title('Simple Bar Chart')
ax.grid(axis='y', linestyle='--', alpha=0.7)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.tight_layout()
plt.show()
```



4. Histogram

Description: Histograms visualize the distribution of a single variable by dividing the range of values into bins and counting the number of observations in each bin.

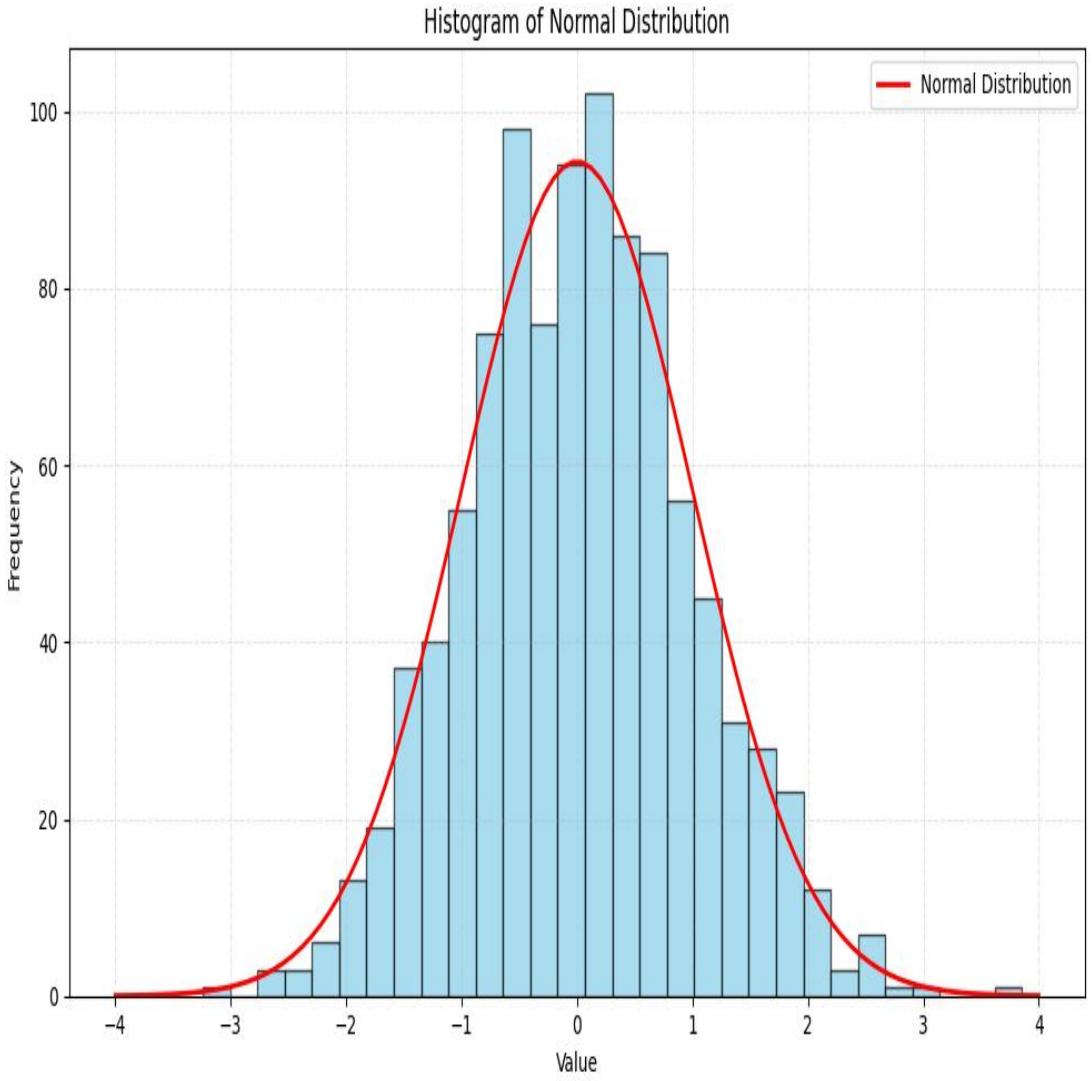
Use Case: Analyzing the distribution of ages in a population, test scores, or any continuous variable.

Code Example:

python

```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(42)data = np.random.normal(loc=0, scale=1, size=1000)
```

```
fig, ax = plt.subplots(figsize=(10, 6))n, bins, patches =
ax.hist(data, bins=30, color='skyblue', edgecolor='black',
alpha=0.7)

x = np.linspace(-4, 4, 100)

y = 1/(1 * np.sqrt(2 * np.pi)) * np.exp(-0.5 * ((x - 0) / 1)
** 2)y = y * len(data) * (bins[1] - bins[0])  ax.plot(x, y, 'r-
', linewidth=2, label='Normal
Distribution')ax.set_xlabel('Value')ax.set_ylabel('Frequency')
ax.set_title('Histogram of Normal Distribution')a

x.legend()

ax.grid(True, linestyle='--', alpha=0.3)

plt.tight_layout()

plt.show()
```



5. Pie Chart

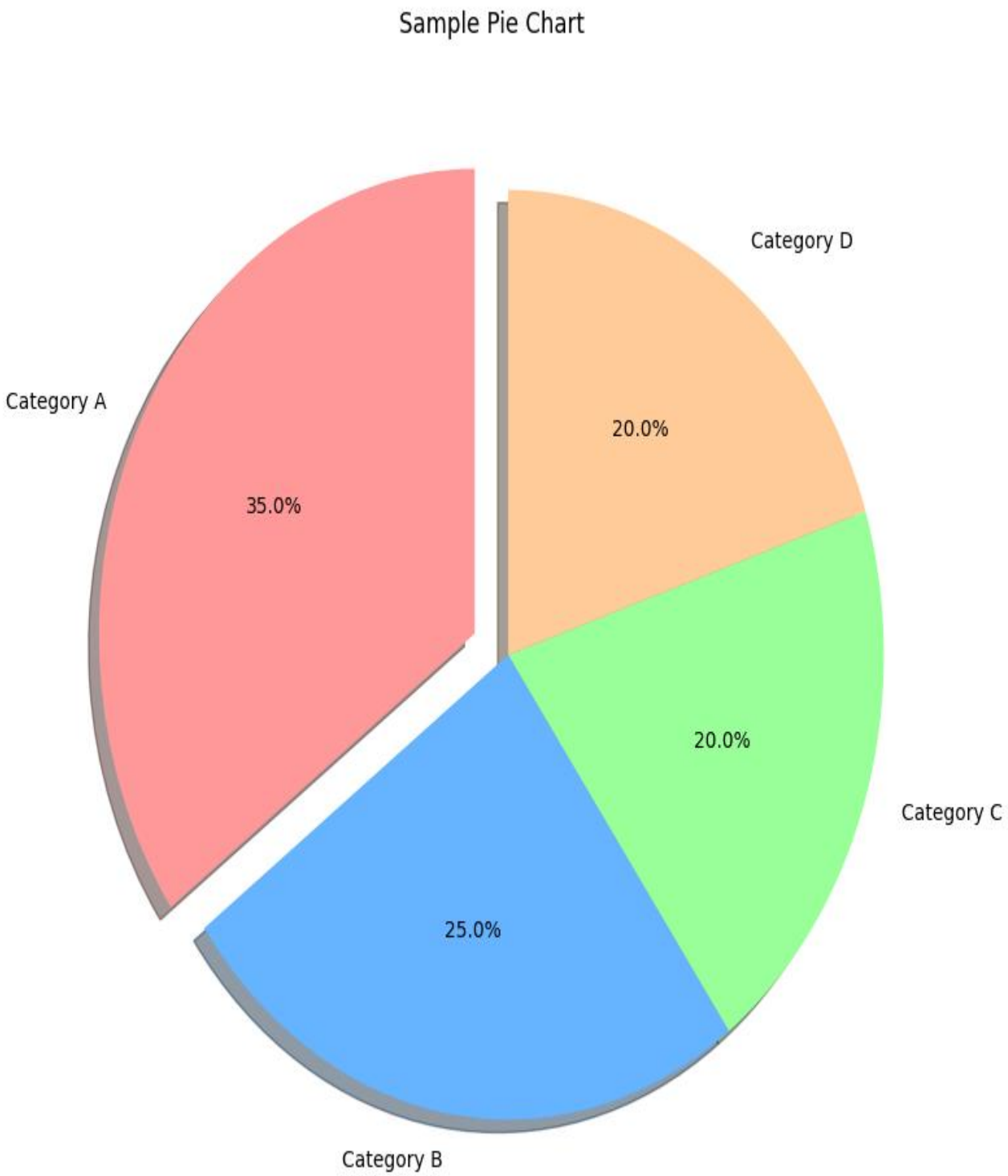
Description: Pie charts display data as slices of a circle, where each slice's size is proportional to the quantity it represents.

Use Case: Showing composition or proportion of categories in a whole, such as market share or budget allocation.

Code Example:

python

```
import matplotlib.pyplot as plt

categories = ['Category A', 'Category B', 'Category C', 'Category D']

sizes = [35, 25, 20, 20]colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']

explode = (0.1, 0, 0, 0)

fig, ax = plt.subplots(figsize=(8, 8))ax.pie(sizes, explode=explode, labels=categories, colors=colors, autopct='%1.1f%%', shadow=True, startangle=90)

ax.axis('equal')

ax.set_title('Sample Pie Chart')

plt.tight_layout()

plt.show()
```


Sample Pie Chart
Category A
35.0%
Category D
20.0%
20.0%
Category C
25.0%
Category B

Plotly

1. Line Plot

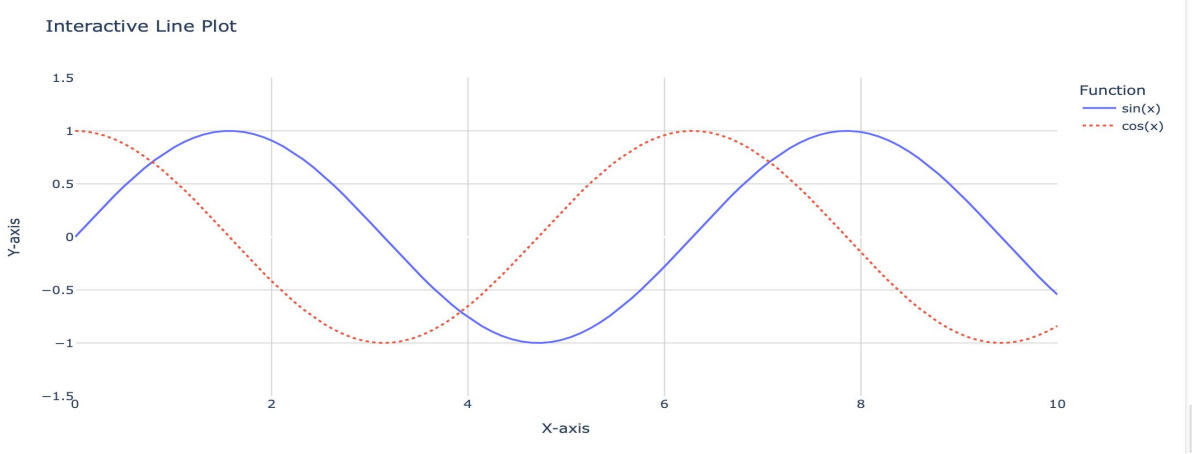
Description: Interactive line plots in Plotly allow zooming, panning, and hovering to see detailed information at each point.

Use Case: Analyzing time series data, showing trends with interactive components.

Code Example:

python

```python
import plotly.express as px
import numpy as np
import pandas as pd
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
df = pd.DataFrame({    'x': np.concatenate([x, x]),    'y': np.concatenate([y1, y2]),    'function': ['sin(x)']*len(x) + ['cos(x)']*len(x)})
fig = px.line(df, x='x', y='y', color='function', line_dash='function',            title='Interactive Line Plot',             labels={'x': 'X-axis', 'y': 'Y-axis'})
fig.update_layout(    legend_title='Function', plot_bgcolor='white',    hovermode='closest', xaxis=dict(        showgrid=True, gridcolor='lightgray',    ), yaxis=dict(        showgrid=True,        gridcolor='lightgray', range=[-1.5, 1.5]    ))
fig.show()
```



2. Scatter Plot

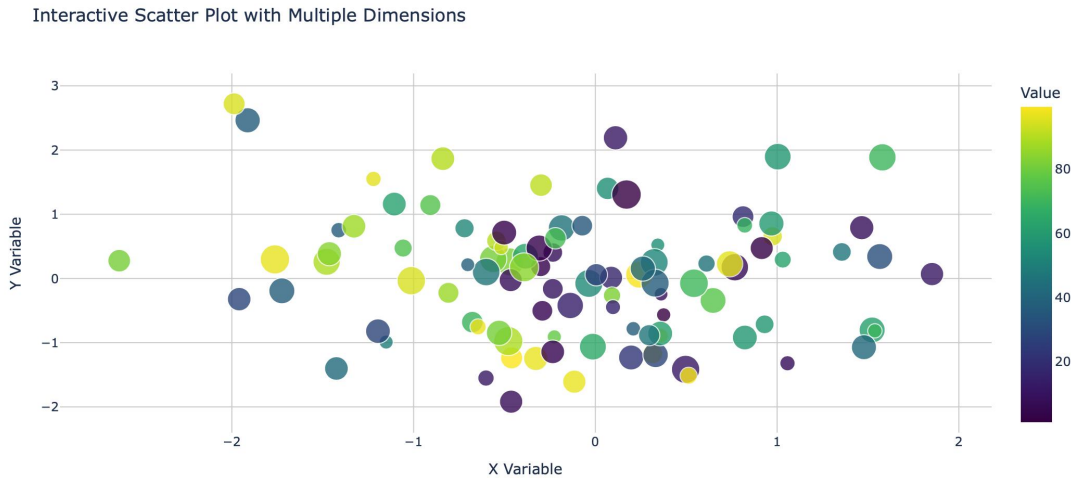
Description: Plotly scatter plots offer interactive features like color mapping, size variation, and hover information.

Use Case: Visualizing relationships between variables with multiple dimensions encoded through color, size, or shape.

Code Example:

python

```
import plotly.express as px
import numpy as np
import pandas as pd
np.random.seed(42)n = 100df = pd.DataFrame({    'x': np.random.normal(0, 1, n),    'y': np.random.normal(0, 1, n),    'size': np.random.uniform(5, 25, n),    'group': np.random.choice(['Group A', 'Group B', 'Group C'], n),    'value': np.random.uniform(0, 100, n)})fig = px.scatter(df, x='x', y='y', color='value', size='size', color_continuous_scale='viridis',                 hover_name='group',                 size_max=20,                 opacity=0.8,                 title='Interactive Scatter Plot with Multiple Dimensions')
fig.update_layout(    plot_bgcolor='white',    xaxis=dict(        title='X Variable',        zeroline=True,        zerolinewidth=1,        zerolinecolor='lightgray',        showgrid=True,        gridcolor='lightgray',    ),    yaxis=dict(        title='Y Variable',        zeroline=True,        zerolinewidth=1,        zerolinecolor='lightgray',        showgrid=True,        gridcolor='lightgray',    ))# Update colorbarfig.update_coloraxes(colorbar_title='Value')# Show plotfig.show()
```



3. Bar Chart

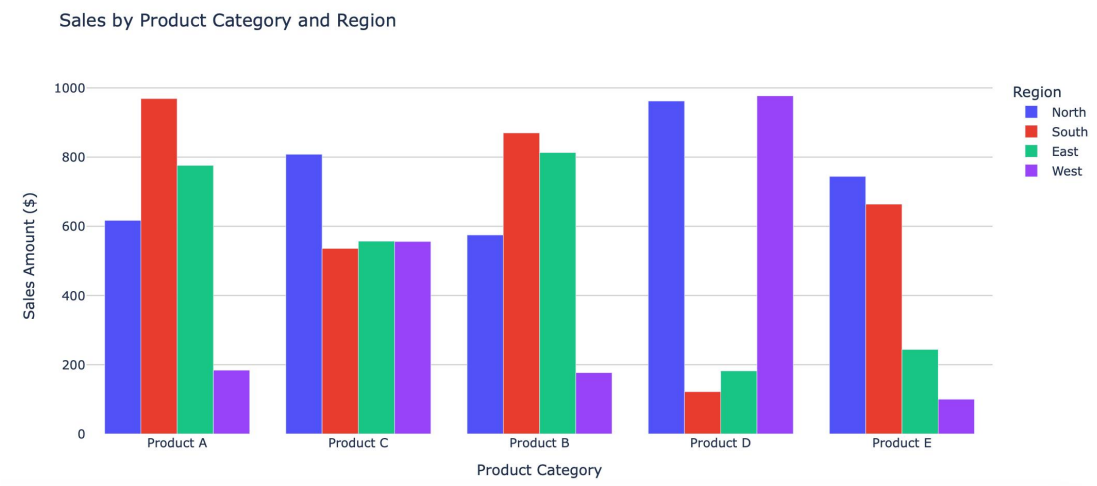
Description: Interactive bar charts in Plotly allow for detailed exploration and provide options for grouped, stacked, or horizontal orientations.

Use Case: Comparing values across categories with interactive elements for deeper analysis.

Code Example:

python

```
import plotly.express as px
import pandas as pd
import numpy as np
categories = ['Product A', 'Product B', 'Product C', 'Product D', 'Product E']
regions = ['North', 'South', 'East', 'West']
data = []for region in regions:    for category in categories:
data.append({            'Category': category,
'Region': region,            'Sales': np.random.randint(100, 1000)        })df = pd.DataFrame(data)# Create interactive bar chartfig = px.bar(df, x='Category', y='Sales',
color='Region', barmode='group',            title='Sales by Product Category and Region',            labels={'Sales': 'Sales Amount ($)'})# Update layoutfig.update_layout(    xaxis_title='Product Category',
yaxis_title='Sales Amount ($)',    legend_title='Region',
plot_bgcolor='white',    xaxis=dict(        categoryorder='total descending',  # Order categories by total sales    ),
yaxis=dict(        showgrid=True,
gridcolor='lightgray',    ))# Show plotfig.show()
```



4. Histogram

Description: Plotly histograms allow for interactive exploration of data distributions with features like zooming and filtering.

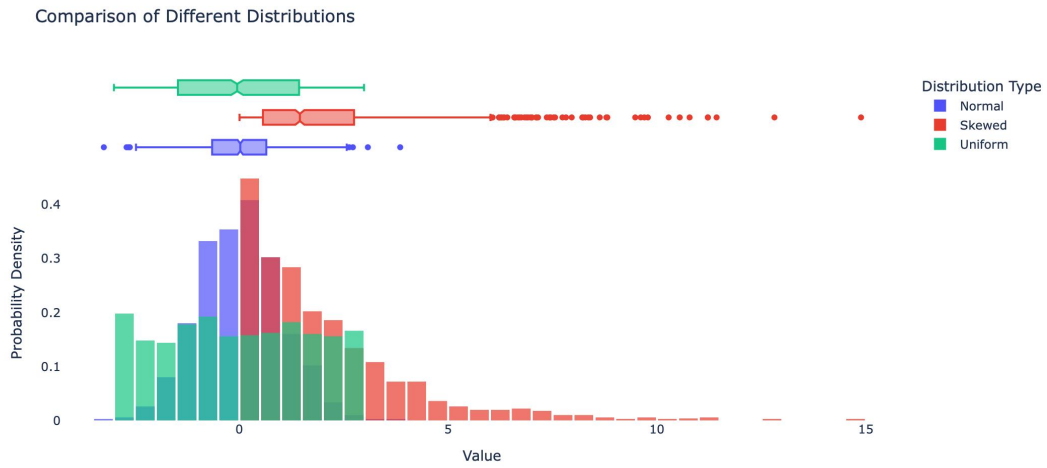
Use Case: Analyzing distributions of data with interactive capabilities to focus on specific ranges.

Code Example:

python

```
import plotly.express as px
import numpy as np
import pandas as pd
   np.random.seed(42)n = 1000
df = pd.DataFrame({    'Normal': np.random.normal(0, 1, n),
'Skewed': np.random.exponential(2, n),    'Uniform':
np.random.uniform(-3, 3, n)})
df_melted = pd.melt(df, var_name='Distribution',
value_name='Value')
fig = px.histogram(df_melted, x='Value', color='Distribution',
nbins=50,                marginal='box',
opacity=0.7,
                barmode='overlay',
                histnorm='probability density',
            title='Comparison of Different Distributions')
fig.update_layout(    xaxis_title='Value',
yaxis_title='Probability Density',
legend_title='Distribution Type',    plot_bgcolor='white',
bargap=0.1)
```

```
fig.show()
```



5. 3D Surface Plot

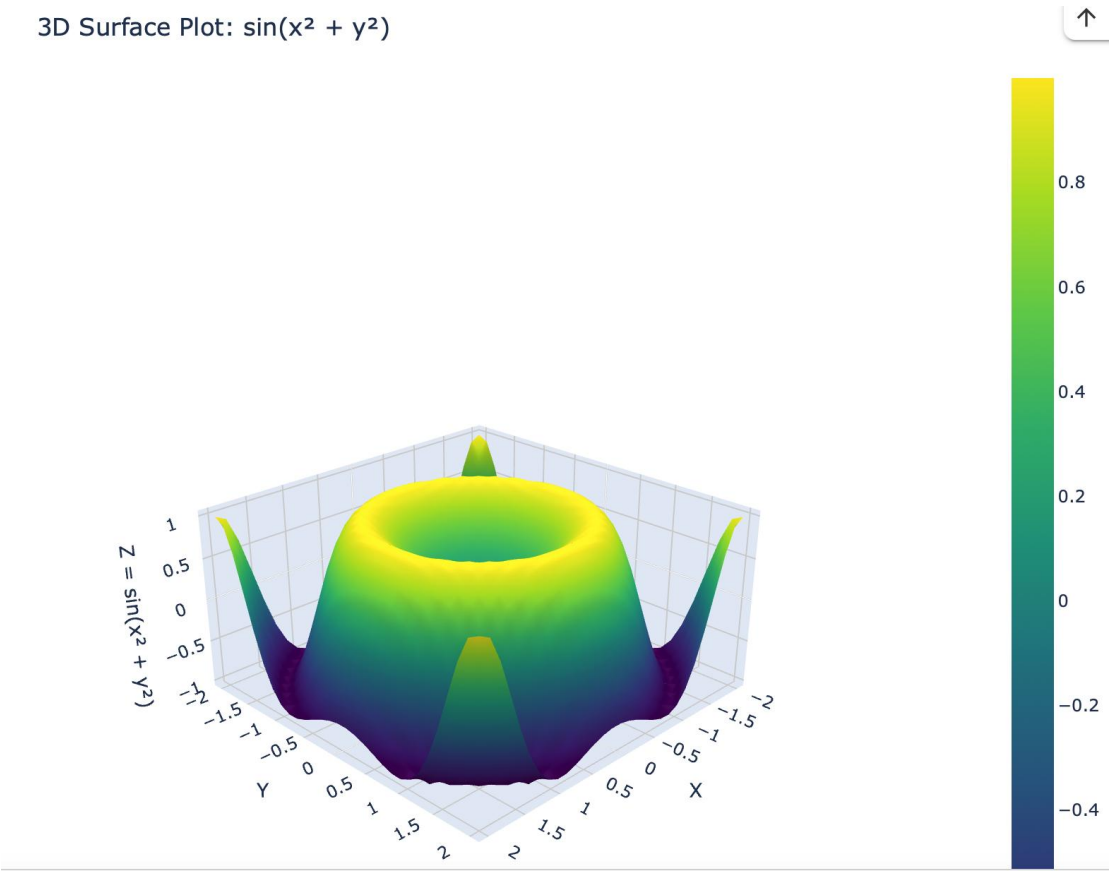
Description: Plotly can create interactive 3D visualizations, allowing users to rotate, zoom, and explore multi-dimensional data.

Use Case: Visualizing mathematical functions, terrain data, or any relationship between three variables.

Code Example:

python

```
import plotly.graph_objects as go
import numpy as np
x = np.outer(np.linspace(-2, 2, 30), np.ones(30))
y = x.copy().T
z = np.sin(x ** 2 + y ** 2)
fig = go.Figure(data=[go.Surface(z=z, x=x, y=y,
colorscale='viridis')])
fig.update_layout(    title='3D Surface Plot: sin(x² + y²)',
scene=dict( xaxis_title='X',        yaxis_title='Y',
zaxis_title='Z = sin(x² + y²)',
xaxis=dict(gridcolor='lightgray'),
yaxis=dict(gridcolor='lightgray'),
zaxis=dict(gridcolor='lightgray'),
camera=dict(            eye=dict(x=1.5, y=1.5,
z=1)          )    ),    width=800,    height=800,
margin=dict(l=0, r=0, b=0, t=40))
fig.show()
```



Library Comparison

Ease of Use

Matplotlib:

- **Learning Curve**: Steeper learning curve due to its comprehensive and sometimes verbose API
- **API Structure**: Two interfaces - Object-Oriented (recommended) and MATLAB-style (pyplot), which can confuse beginners
- **Documentation**: Extensive documentation, but sometimes challenging to navigate due to the library's depth
- **Boilerplate Code**: Often requires more lines of code to create plots

Plotly:

- **Learning Curve**: Gentler learning curve with the high-level Plotly Express API
- **API Structure**: Consistent structure with both high-level (Express) and low-level (Graph Objects) APIs
- **Documentation**: Well-organized documentation with clear examples
- **Boilerplate Code**: Typically requires less code for basic visualizations with Plotly Express

Customization Options

Matplotlib:

- **Flexibility**: Extremely flexible with full control over every element of plots
- **Component Control**: Fine-grained control over every visual aspect of the plot
- **Style Sheets**: Offers predefined style sheets for quick styling
- **Extensions**: Large ecosystem of extensions and add-ons (Seaborn, mplot3d, etc.)

Plotly:

- **Flexibility**: Good overall flexibility, especially with the Graph Objects API
- **Component Control**: Strong but sometimes less granular than Matplotlib
- **Theming**: Provides comprehensive theming capabilities
- **Templates**: Offers predefined templates for consistent styling

Interactivity

Matplotlib:

- **Native Interactivity**: Limited native interactivity
- **Web Integration**: Not designed primarily for web integration
- **User Interaction**: Basic zooming and panning with certain backends
- **Animation**: Can create animations but requires additional code

Plotly:

- **Native Interactivity**: Excellent built-in interactivity with zooming, panning, hovering
- **Web Integration**: Designed for web integration and easily embeddable in dashboards
- **User Interaction**: Rich set of interactive features without extra code
- **Animation**: Strong support for creating interactive animations

Performance with Large Datasets

Matplotlib:

- **Rendering Speed**: Generally faster for static images with small to medium datasets
- **Large Datasets**: Can struggle with very large datasets without optimization
- **Memory Usage**: Lower memory footprint for basic plots
- **Scaling Strategy**: For large datasets, requires strategies like downsampling

Plotly:

- **Rendering Speed**: May be slower for initial rendering of complex plots
- **Large Datasets**: WebGL rendering option for visualizing millions of points
- **Memory Usage**: Can use more memory, especially for complex interactive plots
- **Scaling Strategy**: Better built-in options for handling large datasets with WebGL

Summary

- **Choose Matplotlib when**:
    - Creating static, publication-quality figures
    - Complete control over every aspect of visualization is needed

    - Working within a scientific or academic context where Matplotlib is the standard
    - Integration with the scientific Python ecosystem is important

- **Choose Plotly when**:

    - Interactivity is a priority
    - Visualizations need to be shared on the web or in dashboards
    - Working with multi-dimensional data that benefits from interactive exploration
    - Quick development of visually appealing plots is needed without extensive code