Python Visualization Libraries Guide

Matplotlib & Seaborn

Introduction

Matplotlib and Seaborn are two of the most widely used Python libraries for data visualization, especially in data science and analytics workflows. This comprehensive guide introduces both libraries, showcases common plot types with practical code examples, and compares them for beginner use.

Matplotlib Overview

Matplotlib is a foundational plotting library for Python that can create almost any 2D (and many 3D) visualizations. It exposes a low-level but powerful API to control figures, axes, and individual visual elements.

Key Features:

- Fine-grained control of every visual element (axes, ticks, colors, markers)
- Multiple coding styles: object-oriented (fig, ax = plt.subplots()) and pyplot (plt.plot(...))
- Works in Jupyter notebooks, scripts, and GUI apps; exports to many image formats

Typical Use Cases:

- Custom scientific and engineering figures
- Publication-quality plots with precise layout and styling
- Embedded plots in GUI applications and dashboards

Seaborn Overview

Seaborn is a high-level statistical visualization library built on top of Matplotlib. It focuses on working with tabular data (such as pandas DataFrames) and provides attractive statistical plots with concise code.

Key Features:

- Dataset-oriented API (e.g., data=df, x="col", y="col2")
- Built-in statistical estimation, confidence intervals, regression plots, and distribution plots

- Integrated themes and color palettes for visually appealing defaults

Typical Use Cases:

- Exploratory data analysis (EDA) with DataFrames
- Visualizing distributions and relationships between multiple variables
- Quick creation of complex multi-facet plots

Matplotlib: Common Graph Types

1. Line Plot

Description: Connects data points with lines to show trends over a continuous variable such as time or distance.

Use Case: Plotting time-series (stock price, temperature) or mathematical functions.

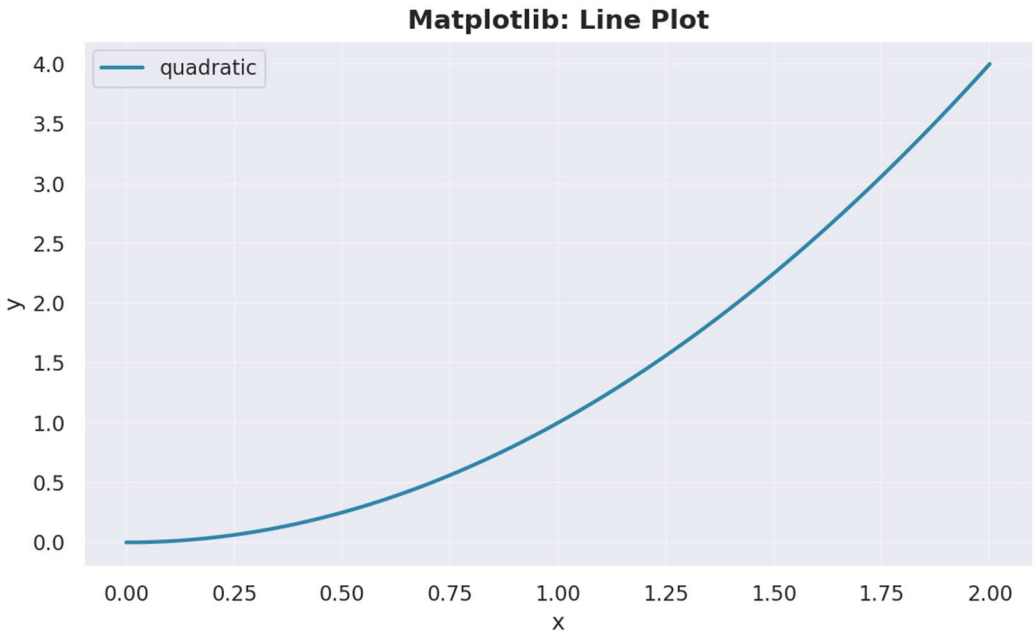
Code Example:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)
y = x**2

fig, ax = plt.subplots()
ax.plot(x, y, label="quadratic")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_title("Line plot")
ax.legend()
plt.show()
```

Output:



2. Scatter Plot

Description: Shows individual data points using markers to visualize relationship between two numeric variables.

Use Case: Studying correlation, clusters, or patterns in paired data.

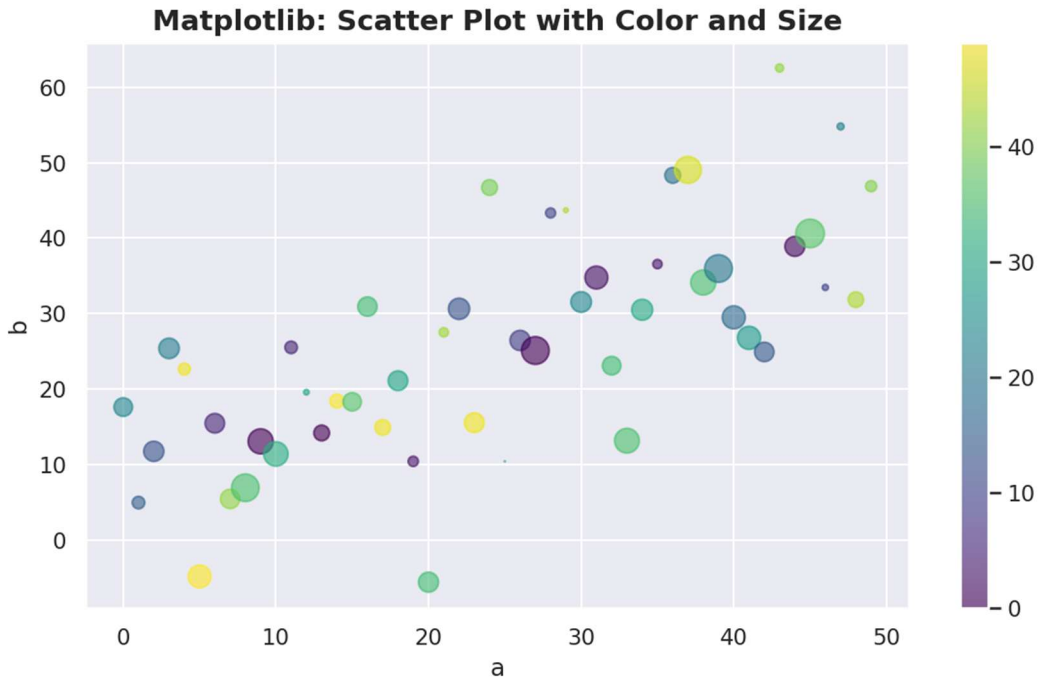
Code Example:

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(0)
data = {
    "a": np.arange(50),
    "b": np.arange(50) + 10 * np.random.randn(50),
    "c": np.random.randint(0, 50, 50),
    "d": np.abs(np.random.randn(50)) * 100,
}

fig, ax = plt.subplots()
ax.scatter("a", "b", c="c", s="d", data=data)
ax.set_xlabel("a")
ax.set_ylabel("b")
ax.set_title("Scatter plot")
plt.show()
```

Output:



3. Bar Chart

Description: Represents data with rectangular bars where height corresponds to category value.

Use Case: Comparing values across categories such as product sales or counts per label.

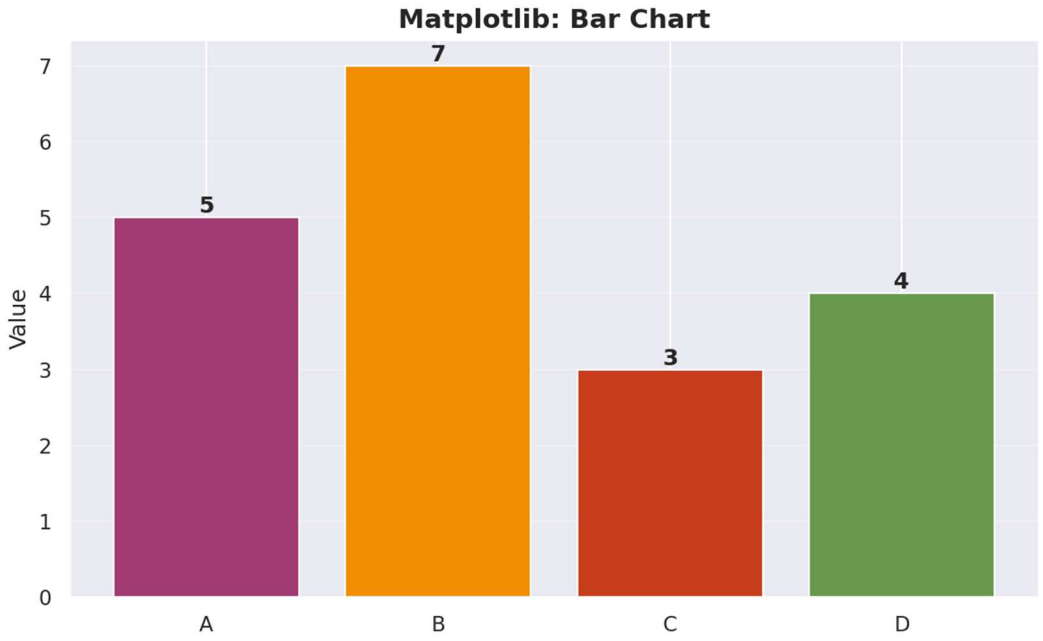
Code Example:

```
import matplotlib.pyplot as plt

categories = ["A", "B", "C", "D"]
values = [5, 7, 3, 4]
```

```
fig, ax = plt.subplots()
ax.bar(categories, values)
ax.set_ylabel("Value")
ax.set_title("Bar chart")
plt.show()
```

Output:



4. Histogram

Description: Splits a continuous variable into bins and shows frequency or density of data in each bin.

Use Case: Inspecting distribution of measurements such as scores, lengths, or durations.

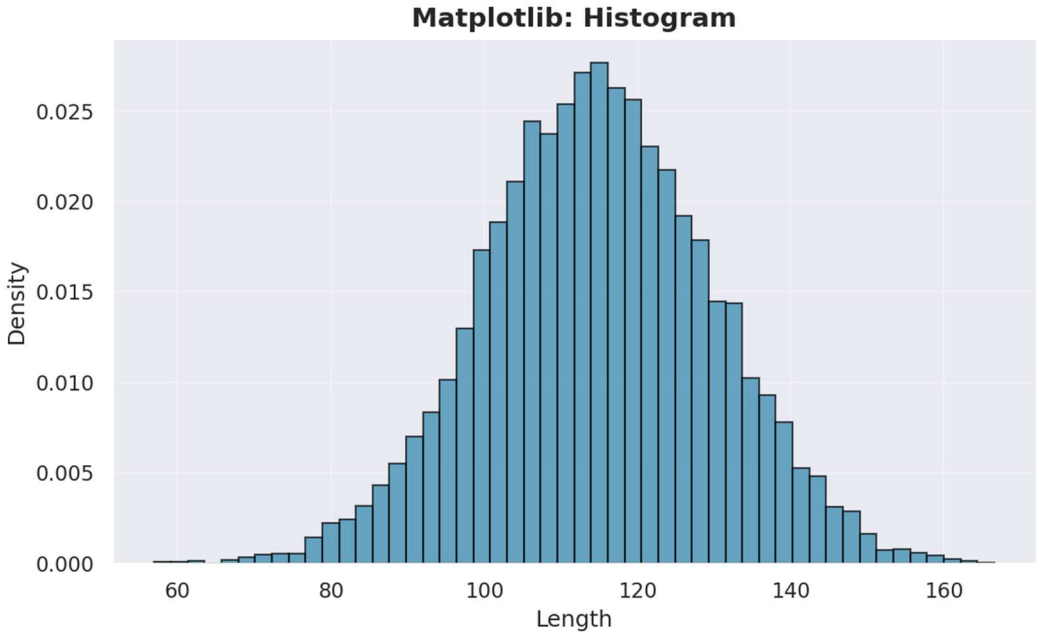
Code Example:

```
import matplotlib.pyplot as plt
import numpy as np

mu, sigma = 115, 15
x = mu + sigma * np.random.randn(10000)

fig, ax = plt.subplots()
ax.hist(x, bins=50, density=True, alpha=0.75)
ax.set_xlabel("Length")
ax.set_ylabel("Density")
ax.set_title("Histogram")
plt.show()
```

Output:



5. Pie Chart

Description: Displays data as slices of a circle, with slice size representing proportion of total.

Use Case: Showing percentage breakdown for a small number of categories.

Code Example:

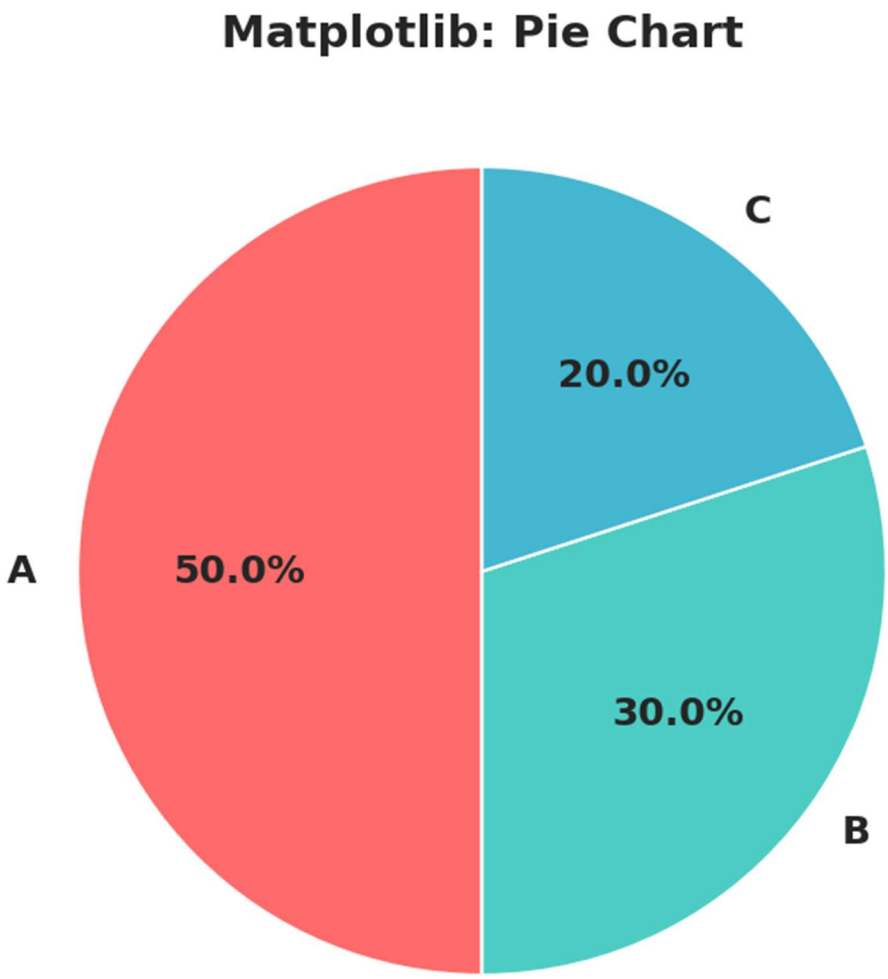
```
import matplotlib.pyplot as plt

labels = ["A", "B", "C"]
```

```
sizes = [50, 30, 20]

fig, ax = plt.subplots()
ax.pie(sizes, labels=labels, autopct="%1.1f%%")
ax.set_title("Pie chart")
plt.show()
```

Output:



6. Heatmap / 2D Color Plot

Description: Displays values on a 2D grid as colors to represent intensity or magnitude.

Use Case: Correlation matrices, image-like data, or function values over a grid.

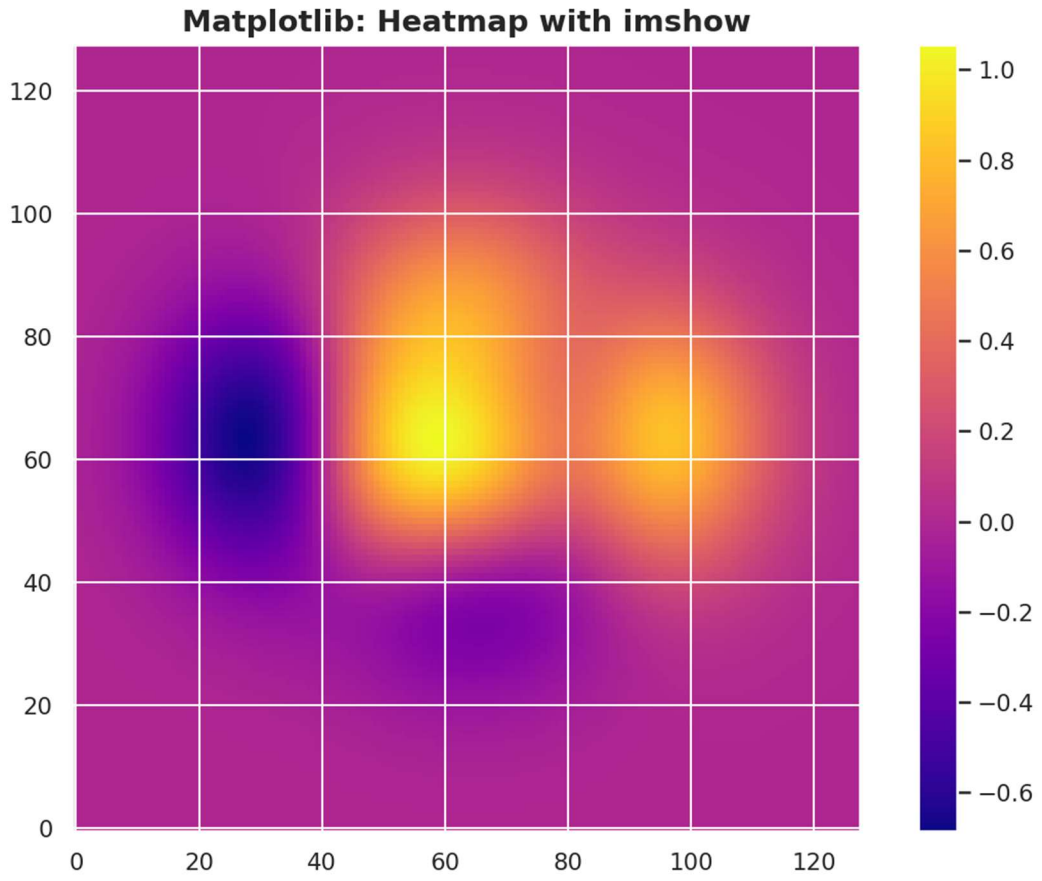
Code Example:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
X, Y = np.meshgrid(np.linspace(-3, 3, 128),
                   np.linspace(-3, 3, 128))
Z = (1 - X/2 + X**5 + Y**3) * np.exp(-X**2 - Y**2)

fig, ax = plt.subplots()
cax = ax.imshow(Z, cmap="plasma", origin="lower")
fig.colorbar(cax, ax=ax)
ax.set_title("Heatmap with imshow")
plt.show()
```

Output:



Comparison: Matplotlib vs Seaborn

Aspect	Matplotlib	Seaborn
Primary Focus	General-purpose plotting	Statistical visualization
API Style	Low-level, OO / pyplot	High-level, declarative, DataFrame-friendly
Defaults	Minimal styling	Themed, attractive defaults
Statistical Tools	User must compute stats manually	Built-in estimation, CIs, regression, distributions
Customization	Very high (full artist-level control)	High via parameters; full control via underlying MPL
Learning Curve	Steeper for advanced plots	Gentler for common EDA plots
Typical Inputs	Arrays, lists, dict-like	pandas DataFrames and arrays
Interactivity	Depends on Matplotlib backends	Same as Matplotlib (not web-focused)

Ease of Use

Seaborn is often easier for EDA with DataFrames because you specify data=df and map columns to roles (x, y, hue, col, etc.). Matplotlib is straightforward for simple function plots or when you think in terms of figures and axes.

Customization

Matplotlib provides maximum control: layout, annotations, multiple axes, custom ticks and scales, and more, ideal for publication-ready figures. Seaborn offers high-level settings and themes but relies on Matplotlib underneath for final fine-tuning.

Interactivity

Both libraries use Matplotlib's backends, so interactivity mainly depends on environment (Jupyter, GUI toolkits) rather than the library itself. Neither focuses on rich browser-based interactivity like Plotly or Bokeh.

Performance with Large Datasets

Matplotlib can plot large arrays but may slow down if too many individual artists are drawn (e.g., millions of markers). Seaborn adds statistical computations and grouping, so for very large datasets, pre-aggregating in pandas and then plotting can be more efficient.

Key Takeaways

1. 1. Use Matplotlib when you need fine-grained control, custom layouts, or publication-quality figures.
2. 2. Use Seaborn when you're doing exploratory data analysis with DataFrames and want statistical plots with minimal code.
3. 3. Combine them: Seaborn functions return Matplotlib axes, so you can use both in the same workflow.
4. 4. Learn one well first: Most beginners benefit from starting with Seaborn for quick visualization, then learning Matplotlib for customization.

Resources

- Matplotlib Quick Start: https://matplotlib.org/stable/users/explain/quick_start.html
- Seaborn Introduction: https://seaborn.pydata.org/tutorial/introduction.html
- Matplotlib Gallery: https://matplotlib.org/stable/gallery/index.html
- Seaborn Gallery: https://seaborn.pydata.org/examples.html

This guide is intended for beginners learning Python data visualization. For advanced topics and deeper exploration, refer to the official documentation of each library.