Data Visualization using Python

Carlos J. Costa

Data visualization plays a crucial role in data analysis, making it easier to understand trends, patterns, and relationships in data[1]. Python provides several powerful libraries for visualizing data, including Pandas, Matplotlib, and Seaborn. Each library serves different purposes and offers a variety of plotting methods. This document will cover essential visualization techniques, including scatter plots, line charts, bar charts, and more advanced visualizations like heatmaps and pair plots.

1. Pandas: Quick and Simple Plots

Pandas provides an easy way to create basic visualizations directly from DataFrames. It is built on top of Matplotlib, which shares many of the same features [3]. The following example shows how to implement Bar Plot using Pandas:

```
import pandas as pd
import matplotlib.pyplot as plt
# Sample DataFrame
data = {'Category': ['A', 'B', 'C', 'D'], 'Values': [4, 7, 1, 8]}
df = pd.DataFrame(data)
# Create a bar plot
df.plot(kind='bar', x='Category', y='Values', color='skyblue')
plt.title('Bar Plot using Pandas')
plt.show()
```

Key Features of Pandas for Visualization are the following:

- df.plot() supports basic plots (line, bar, histogram, scatter).

- Integration with Matplotlib allows complete customization of plots.

- Simplified interface for quick visualizations.

2. Matplotlib: Low-Level Customization

Matplotlib is a powerful and versatile Python library for creating static, animated, and interactive visualizations [6]. It provides fine-grained control over plot elements, making it ideal for users who require detailed customization. As the foundational plotting library in Python, Matplotlib supports a wide range of chart types, including scatter plots, line charts, and bar charts. While highly flexible, it can sometimes feel verbose for simple visualizations. Additionally, it serves as the backbone for many other libraries, such as Seaborn and Plotly, which build upon its capabilities to offer more user-friendly interfaces..

- **Strengths:**

- Fine-grained control over plot elements
- Wide range of plot types
- Suitable for publication-quality figures

- **Weaknesses:**

  - Steeper learning curve for beginners
  - More verbose compared to higher-level libraries

Here's an improved version with examples:

Building Blocks of Matplotlib Plots

Matplotlib plots are composed of several fundamental elements that work together to create clear and informative visualizations. The key components include:

- Figure (fig)
- Axes (ax)
- Subplots
- Axis

Figure (fig) is the top-level container that holds all plot elements, including one or more subplots. Think of it as a blank canvas where multiple charts can be arranged.

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8, 6))
```

Axes (ax) is the area where data is plotted. A figure can contain multiple axes (subplots), each displaying a different visualization.

```
fig, ax = plt.subplots()  # Creates a figure with a single plotting
area (axes)
ax.plot([1, 2, 3], [4, 5, 6])  # Plots a simple line chart
plt.show()
```

Subplots are multiple independent plotting areas within the same figure, which are useful for comparing different datasets side by side.

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))  # Two subplots
in one row
ax1.plot([1, 2, 3], [4, 5, 6], label="Dataset 1")
ax2.scatter([1, 2, 3], [6, 5, 4], color='r', label="Dataset 2")
plt.show()
```

Axis, corresponding to x-axis and y-axis, defines the coordinate space for data visualization. Each axis can be labeled, scaled, and customized.

```
fig, ax = plt.subplots()
ax.plot([10, 20, 30], [5, 15, 25])
ax.set_xlabel("X-Axis Label")
ax.set_ylabel("Y-Axis Label")
ax.set_title("Example Plot with Labeled Axes")
plt.show()
```

Understanding these components makes it possible to create highly customized and effective visualizations using Matplotlib.

Matplotlib's plotting flow typically starts by creating a figure and axes, and then adding various elements (like data points, lines, bars….) to the axes. The immense customization possibilities allow users to manipulate every plot detail, from titles and axis labels to line styles and colors.

2.1 Scatter Plot

Scatter plots are used to visualize relationships between two continuous variables. In Matplotlib, scatter plots can be customized extensively by adding colors or markers based on a third categorical variable:

```
import matplotlib.pyplot as plt

# Create a figure and axis

fig, ax = plt.subplots()

# Scatter plot for 'electoral college %' vs 'popular vote %'

ax.scatter(df['electoral college %'], df['popular vote %'])

# Set titles and labels

ax.set_title('Election Results')

ax.set_xlabel('Electoral College Percentage')

ax.set_ylabel('Popular Vote Percentage')

plt.show()
```

It is possible to customize scatter plots further by using color to represent different categories. In the following example, we color points based on political party affiliation.

```
# Create a color dictionary based on party affiliation

colrs = {'Rep.': 'r', 'Dem.': 'b'}

# Create figure and axis

fig, ax = plt.subplots()

# Plot data points with color based on party

for i in range(len(df['electoral college percentage'])):

    ax.scatter(df['electoral college %'][i], df['popular vote %'][i],
color=colrs[df['party'][i]])
```

```
# Set titles and labels
ax.set_title('US Elections')
ax.set_xlabel('Electoral College Percentage')
ax.set_ylabel('Popular Vote Percentage')
plt.show()
```

2.2 Line Chart

Line charts are commonly used to visualize data trends over time. They are ideal for showing continuous data points and identifying changes in variables. The following example shows how to implement a Basic Line Chart using Matplotlib. This example demonstrates how to plot the trend of the popular vote percentage over several years.

```
# Create a figure and axis
fig, ax = plt.subplots()
# Plot 'popular vote percentage' over the years
ax.plot(df['year'], df['popular vote percentage'])
# Set titles and labels
ax.set_title('US Elections - Popular Vote Over Time')
ax.set_xlabel('Year')
ax.set_ylabel('Popular Vote Percentage')
plt.show()
```

The following example shows how to implement a Multiple Lines Chart using Matplotlib. The following example shows how to plot both the popular vote percentage and the electoral college percentage over time on the same axes.

```
# Create a figure and axis
fig, ax = plt.subplots()
# Plot multiple lines
ax.plot(df['year'], df['popular vote percentage'], label='Popular Vote')
ax.plot(df['year'], df['electoral college percentage'], label='Electoral
College')
# Set title, legend, and labels
ax.set_title('US Elections - Popular Vote and Electoral College')
ax.legend()
ax.set_xlabel('Year')
ax.set_ylabel('Percentage')
```

```
plt.show()
```

2.3 Bar Chart

Bar charts are helpful for visualizing the frequency of categorical data. They can be used to compare quantities across categories. Matplotlib can create grouped, stacked, and horizontal bar charts. The following example shows how to implement a Bar Chart using Matplotlib. In the following example, we use a bar chart to visualize the number of votes by party.

```
# Count the occurrence of each class (party)
data = df['party'].value_counts()
# Get x and y data
points = data.index
frequency = data.values
# Create a figure and axis
fig, ax = plt.subplots()
# Plot a bar chart
ax.bar(points, frequency)
# Set titles and labels
ax.set_title('US Votes by Party')
ax.set_xlabel('Party')
ax.set_ylabel('Frequency')
plt.show()
```

2.4 Advanced Subplots and Layouts

Matplotlib allows multiple plots in the exact figure using subplots. This is especially useful when comparing different visualizations side by side.

Creating Multiple Subplots:

```
# Create a figure with 2x1 subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

# First subplot: Scatter plot
ax1.scatter(df['electoral college %'], df['popular vote %'])
ax1.set_title('Scatter Plot')
ax1.set_xlabel('Electoral College %')
ax1.set_ylabel('Popular Vote %')
```

```
# Second subplot: Line plot
ax2.plot(df['year'], df['popular vote %'])
ax2.set_title('Line Plot')
ax2.set_xlabel('Year')
ax2.set_ylabel('Popular Vote %')

# Adjust layout and display
plt.tight_layout()
plt.show()
```

This approach makes displaying multiple visualizations in a single figure easy, particularly useful for exploratory data analysis or reports.

3. Seaborn: Statistical Data Visualization

Seaborn is built on top of Matplotlib and provides a high-level interface for creating attractive and informative statistical graphics. It works seamlessly with pandas. DataFrames and adds powerful features for visualizing complex datasets[7] [8].

- **Strengths:**
  - Simplifies statistical plotting
  - Aesthetically pleasing default styles
  - Ideal for exploring relationships in data
- **Weaknesses:**
  - Less flexibility in customizing individual plot elements compared to Matplotlib

3.1 Pair Plot

Pair plots help visualize relationships between all variables in a dataset. This function plots pairwise relationships in a dataset and is particularly useful for exploratory data analysis. The following example shows how to implement a Pair Plot using Seaborn.

```
import seaborn as sns
# Load example dataset
df = sns.load_dataset('iris')
# Create a pair plot
sns.pairplot(df, hue='species')
```

```
plt.show()
```

3.2 Heatmap

Heatmaps are used to visualize correlation matrices, showing relationships between variables with colors. You can easily add annotations and color palettes to highlight specific aspects of the data. The following example shows how to implement Heatmap using Seaborn.

```
# Create a correlation matrix
corr = df.corr()
# Create a heatmap
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

3.3 Customizing Seaborn Plots

Seaborn also offers additional features, such as themes and color palettes, to enhance the aesthetics of the plots. Different themes can be applied using sns.set_theme() to match the style of the visualization with the presentation or report.

4. Plotly: Interactive Visualizations

Plotly is a powerful library for creating interactive, web-based visualizations. It supports a wide range of chart types, including 3D plots, and enables dynamic exploration through features like zooming, hovering, and filtering [5]. Designed for interactivity, Plotly excels at embedding visualizations in web applications and dashboards, making it an ideal choice for dynamic data analysis and presentation.

- **Strengths:**
  - Fully interactive visualizations
  - Great for web applications and presentations
  - Supports a variety of complex visualizations (e.g., 3D plots, geographic maps)
- **Weaknesses:**
  - More resource-intensive than static libraries like Matplotlib or Seaborn
  - Slightly different approach to plotting compared to traditional Python libraries

The following example shows how to implement an Interactive Line Chart using Plotly:

```
import plotly.express as px

# Create a sample DataFrame

df = px.data.gapminder().query("country == 'Canada'")

# Create an interactive line chart

fig = px.line(df, x='year', y='gdpPercap', title='GDP Per Capita Over
Time (Canada)')

fig.show()
```

With Plotly, it is possible to add interactive capabilities, making it easy to create dashboards and reports that users can explore.

5. Bokeh: Interactive Visualizations for the Web

Bokeh is a powerful and versatile library for creating interactive visualizations that are well-suited for web applications [2]. Like Plotly, Bokeh allows you to build interactive plots without writing JavaScript code. Its main advantage is the ability to generate highly detailed, interactive plots that can be embedded into websites or Jupyter notebooks. Bokeh's flexibility makes it ideal for developing dashboards and web-based visual analytics.

- **Strengths:**
  - Optimized for handling large datasets
  - Well-suited for web deployment
- **Weaknesses:**
  - Longe learning curve compared to Seaborn or Plotly

The following example shows how to implement an Interactive Line Plot using Bokeh:

```
from Bokeh.plotting import figure, show

from bokeh.io import output_notebook

output_notebook()
```

```
# Create a figure

p = figure(title="Interactive Line Plot in Bokeh", x_axis_label='X', y_axis_label='Y')

# Add a line plot

p.line([1, 2, 3, 4, 5], [2, 4, 6, 8, 10], legend_label="Y=2X", line_width=2)

# Show the plot in the notebook

show(p)
```

In this example, we create a simple line plot with Bokeh's `figure()` method, specifying labels for the x and y axes. The `p.line()` function is used to draw a line based on a list of x and y values, and the `show()` function renders the plot interactively in the notebook.

Here's another example that creates an interactive bar plot using Bokeh:

```
from Bokeh.plotting import figure, show

from bokeh.io import output_notebook

from Bokeh.models import HoverTool

from Bokeh.transform import factor_cmap

# Ensure Bokeh renders in the notebook

output_notebook()

# Sample data

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

values = [10, 20, 30, 40]

# Create a figure object

p = figure(x_range=categories, height=350, title="Category Bar Plot",

           toolbar_location=None, tools="")

# Create bar chart with colors and hover tool

p.vbar(x=categories, top=values, width=0.9, color=factor_cmap('x', palette=['#c9d9d3', '#718dbf', '#e84d60', '#ddb7b1'], factors=categories))

# Add hover tool to show data values

hover = HoverTool()

hover.tooltips = [("Category", "@x"), ("Value", "@top")]

p.add_tools(hover)

# Customize the plot

p.xgrid.grid_line_color = None
```

```
p.y_range.start = 0

# Show the plot in the notebook interactively

show(p)
```

6. Conclusion

Python provides a diverse range of libraries for data visualization, each designed to cater to specific needs: Pandas, Matplotlib, Seaborn, Plotly, and Bokeh.

Pandas library is ideal for quick and straightforward plots directly from DataFrames. Pandas makes generating basic visualizations with minimal code easy, allowing you to explore and summarize data efficiently. Its integration with Matplotlib enables more customization if needed, but Pandas suffices with its simple syntax and direct functionality for most exploratory tasks.

Matplotlib is known for its flexibility and low-level control, and it is the go-to library for Python users who need highly customizable plots. With its core components like figures, axes, and subplots, Matplotlib enables the creation of both simple and complex visualizations, ranging from scatter plots to advanced multi-plot layouts. It is the perfect choice when you require complete control over every aspect of the visual output.

Seaborn is built on top of Matplotlib. Seaborn simplifies the creation of aesthetically pleasing statistical plots. It offers high-level interfaces for drawing attractive and informative visualizations such as heat maps, box plots, and regression plots. Seaborn's strong emphasis on revealing statistical relationships makes it an excellent choice for visualizing complex data patterns cleanly and elegantly.

Plotly specializes in creating interactive and web-ready visualizations. With minimal code, users can generate dynamic, zoomable, and hover-friendly charts easily embedded into websites or shared in reports. Plotly is ideal for creating engaging visualizations, particularly for presentations or interactive dashboards where user engagement with the data is essential [4].

Like Plotly, Bokeh focuses on creating interactive visualizations and emphasizes the ability to handle large datasets and generate web-ready plots. It allows for detailed interactivity, including zooming, panning, and tooltips, making it well-suited for applications where real-time data exploration is necessary.

By understanding and combining these libraries, it is possible to create powerful visualizations tailored to different stages of data analysis, from quick exploration and statistical insights to polished, interactive presentations. Each library serves specific needs, and choosing the right one—or using them in tandem—will allow you to present your data effectively, whether for exploration or final reporting.

References

[1] M. Aparicio and C. J. Costa, "Data visualization," *Communication Design Quarterly Review*, vol. 3, no. 1, pp. 7-11, 2015.

[2] *Bokeh documentation*, Bokeh. Accessed: Oct. 12, 2024. [Online]. Available: https://docs.bokeh.org/en/latest/index.html

[3] *Chart visualization—Pandas 2.2.3 documentation*. Accessed: Oct. 12, 2024. [Online]. Available: https://pandas.pydata.org/docs/user_guide/visualization.html

[4] C. Costa and M. Aparício, "Supporting the decision on dashboard design charts," in *Proceedings of 254th The IIER International Conference 2019, September*, pp. 10-15.

[5] *Getting started with Plotly in Python*. Accessed: Oct. 12, 2024. [Online]. Available: https://plotly.com/python/getting-started/

[6] *Matplotlib documentation—Matplotlib 3.9.2 documentation*. Accessed: Oct. 12, 2024. [Online]. Available: https://matplotlib.org/stable/index.html

[7] *User guide and tutorial—Seaborn 0.13.2 documentation*. Accessed: Oct. 12, 2024. [Online]. Available: https://seaborn.pydata.org/tutorial.html

[8] M. Waskom, "seaborn: Statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. [Online]. Available: https://doi.org/10.21105/joss.03021