Data Visualization with Matplotlib

Principles, Basics, and Customization

Slide 1: Visualization Principles

Why Visualize Data?

- **Understanding**: Reveal patterns, trends, and outliers
- **Communication**: Convey insights quickly and effectively
- **Decision-making**: Support data-driven conclusions
- **Exploration**: Discover relationships in complex datasets

Key Goals

- Accuracy: Represent data truthfully
- Clarity: Make information easy to understand
- Efficiency: Minimize cognitive load

Slide 2: Core Visualization Principles

1. Choose the Right Chart Type

- **Line charts**: Trends over time
- **Bar charts**: Comparisons between categories
- **Scatter plots**: Relationships between variables
- **Histograms**: Distributions
- **Pie charts**: Proportions (use sparingly)

2. Design Principles

- **Simplicity**: Remove unnecessary elements
- **Color**: Use purposefully and consistently
- **Labels**: Clear titles, axes, and legends
- **Scale**: Start y-axis at zero for bar charts

Slide 3: The Data-Ink Ratio

Maximize Information, Minimize Clutter

> "Data-ink ratio = Data-ink / Total ink used in graphic" — Edward Tufte

What to Avoid

- Unnecessary gridlines
- 3D effects without purpose
- Excessive decorations (chartjunk)
- Too many colors or patterns

What to Include

- Clear axis labels
- Informative titles
- Necessary legends
- Data points and trends

Slide 4: Getting Started with Matplotlib

What is Matplotlib?

- Python's most popular plotting library
- Created by John Hunter in 2003

- Flexible and customizable
- Foundation for other libraries (Seaborn, Pandas plotting)

Installation

```
pip install matplotlib
```

Basic Import

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

Slide 5: Matplotlib Architecture

Two Main Interfaces

1. pyplot (MATLAB-style, state-based)

```
plt.plot([1, 2, 3, 4])
plt.ylabel('Values')
plt.show()
```

2. Object-Oriented (explicit, recommended)

```
fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4])
ax.set_ylabel('Values')
plt.show()
```

Key Components

- **Figure**: The entire plotting window
- **Axes**: Individual plot area (can have multiple)
- **Axis**: X and Y axes with ticks and labels

Slide 6: Creating Your First Plot

Simple Line Plot

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

# Data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Create plot
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Sine Wave')
plt.show()
```

Slide 7: Common Plot Types

Line Plots

```
ax.plot(x, y)
```

Scatter Plots

```
ax.scatter(x, y)
```

Bar Charts

```
ax.bar(categories, values)
```

Histograms

```
ax.hist(data, bins=20)
```

Slide 8: Multiple Plots in One Figure

Subplots

```
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

# Access individual subplots
axes[0, 0].plot(x, y1)
axes[0, 1].scatter(x, y2)
axes[1, 0].bar(categories, values)
axes[1, 1].hist(data)

plt.tight_layout()
plt.show()
```

Multiple Lines on One Plot

```
ax.plot(x, y1, label='Line 1')
ax.plot(x, y2, label='Line 2')
ax.legend()
```

Slide 9: Plot Customization - Colors & Styles

Color Options

```
# Named colors
ax.plot(x, y, color='red')

# Hex colors
ax.plot(x, y, color='#FF5733')

# RGB tuples
ax.plot(x, y, color=(0.1, 0.2, 0.5))
```

Line Styles

```
ax.plot(x, y, linestyle='--')    # dashed
ax.plot(x, y, linestyle='-.')    # dash-dot
ax.plot(x, y, linestyle=':')     # dotted
ax.plot(x, y, linewidth=2)       # thickness
```

Slide 10: Markers and Symbols

Marker Styles

```
ax.plot(x, y, marker='o')     # circles
ax.plot(x, y, marker='s')     # squares
ax.plot(x, y, marker='^')     # triangles
ax.plot(x, y, marker='*')     # stars
```

Marker Customization

```
ax.plot(x, y,
        marker='o',
        markersize=8,
        markerfacecolor='red',
        markeredgecolor='black',
        markeredgewidth=2)
```

Slide 11: Labels and Annotations

Text Elements

```
# Axis labels
ax.set_xlabel('X Label', fontsize=12)
ax.set_ylabel('Y Label', fontsize=12)

# Title
ax.set_title('My Plot', fontsize=14, fontweight='bold')

# Text annotation
ax.text(5, 10, 'Important point', fontsize=10)

# Arrow annotation
ax.annotate('Peak', xy=(7, 12), xytext=(8, 15),
            arrowprops=dict(arrowstyle='->'))
```

Slide 12: Legends and Grids

Legends

```
 ax.plot(x, y1, label='Dataset 1')
ax.plot(x, y2, label='Dataset 2')

# Position legend
ax.legend(loc='upper right')
# or: 'upper left', 'lower right', 'best'

# Customize legend
ax.legend(frameon=True, shadow=True,
          fontsize=10, title='Data')
```

Grids

```
 ax.grid(True)
ax.grid(True, linestyle='--', alpha=0.5)
```

Slide 13: Axis Customization

Limits and Ranges

```
 ax.set_xlim(0, 10)
ax.set_ylim(-1, 1)
```

Scales

```
 ax.set_xscale('log')    # logarithmic
ax.set_yscale('linear') # linear (default)
```

Ticks

```
 ax.set_xticks([0, 2, 4, 6, 8, 10])
ax.set_xticklabels(['A', 'B', 'C', 'D', 'E', 'F'])
ax.tick_params(labelsize=10, rotation=45)
```

Slide 14: Figure Size and Resolution

Figure Dimensions

```
 # Set size when creating figure
fig, ax = plt.subplots(figsize=(10, 6))

# Adjust after creation
fig.set_size_inches(12, 8)
```

Saving Figures

```
 # Save with high resolution
plt.savefig('my_plot.png', dpi=300, bbox_inches='tight')

# Different formats
plt.savefig('my_plot.pdf')
plt.savefig('my_plot.svg')
```

Slide 15: Color Maps and Palettes

Using Colormaps

```
# For scatter plots with color scale
scatter = ax.scatter(x, y, c=values, cmap='viridis')
plt.colorbar(scatter, ax=ax, label='Values')
```

Popular Colormaps

- **Sequential**: 'viridis', 'plasma', 'inferno', 'Blues'
- **Diverging**: 'RdBu', 'coolwarm', 'seismic'
- **Qualitative**: 'tab10', 'Set1', 'Paired'

Custom Color Cycles

```
colors = ['#1f77b4', '#ff7f0e', '#2ca02c']
ax.set_prop_cycle(color=colors)
```

Slide 16: Styles and Themes

Built-in Styles

```
# See available styles
print(plt.style.available)

# Use a style
plt.style.use('seaborn-v0_8-darkgrid')
plt.style.use('ggplot')
plt.style.use('dark_background')
```

Context Managers

```
with plt.style.context('seaborn-v0_8'):
    fig, ax = plt.subplots()
    ax.plot(x, y)
    plt.show()
```

Slide 17: Advanced Customization

Spines and Borders

```
# Remove spines
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Change spine properties
ax.spines['bottom'].set_linewidth(2)
ax.spines['left'].set_color('gray')
```

Tight Layout

```
plt.tight_layout()  # Automatically adjust spacing
```

Transparency

```
ax.plot(x, y, alpha=0.5)  # 50% transparent
```

Slide 18: Best Practices

Do's

✓ Use appropriate chart types for your data ✓ Label all axes clearly with units ✓ Include a descriptive title ✓ Use consistent colors and styles ✓ Provide legends when showing multiple series ✓ Save in appropriate format and resolution

Don'ts

✗ Distort scales to exaggerate differences ✗ Use 3D when 2D is sufficient ✗ Overuse colors and decorations ✗ Truncate axes misleadingly ✗ Use pie charts for more than 5 categories

Slide 19: Complete Example

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

# Generate data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Create figure
fig, ax = plt.subplots(figsize=(10, 6))

# Plot data
ax.plot(x, y1, 'b-', linewidth=2, label='sin(x)')
ax.plot(x, y2, 'r--', linewidth=2, label='cos(x)')

# Customize
ax.set_xlabel('X values', fontsize=12)
ax.set_ylabel('Y values', fontsize=12)
ax.set_title('Trigonometric Functions', fontsize=14, fontweight='bold')
ax.legend(loc='upper right', fontsize=10)
ax.grid(True, alpha=0.3)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.tight_layout()
plt.savefig('trig_plot.png', dpi=300, bbox_inches='tight')
plt.show()
```

Slide 20: Introduction to Seaborn

What is Seaborn?

- Built on top of Matplotlib
- High-level interface for statistical graphics
- Beautiful default styles
- Integrated with Pandas DataFrames
- Ideal for exploratory data analysis

Installation & Import

```
pip install seaborn
```

```
import seaborn as sns
import matplotlib.pyplot as plt
```

Key Advantages

- Less code for complex visualizations
- Attractive default themes
- Statistical functions built-in
- Works seamlessly with Pandas

Slide 21: Seaborn Themes and Styles

Setting Themes

```
# Set overall style
sns.set_theme(style='darkgrid')
# Options: 'darkgrid', 'whitegrid', 'dark', 'white', 'ticks'

# Set context for scaling
sns.set_context('notebook')
# Options: 'paper', 'notebook', 'talk', 'poster'

# Set color palette
sns.set_palette('husl')
```

Custom Styling

```
sns.set_theme(style='whitegrid',
              palette='muted',
              font_scale=1.2)
```

Return to Matplotlib defaults

```
sns.reset_defaults()
```

Slide 22: Seaborn Color Palettes

Built-in Palettes

```
# Qualitative palettes
sns.color_palette('deep')
sns.color_palette('pastel')
sns.color_palette('Set2')

# Sequential palettes
sns.color_palette('Blues')
sns.color_palette('rocket')

# Diverging palettes
sns.color_palette('coolwarm')
sns.color_palette('vlag')
```

Viewing Palettes

```
sns.palplot(sns.color_palette('viridis', 10))
```

Slide 23: Distribution Plots

Histogram with KDE

```
sns.histplot(data=df, x='column', kde=True)
```

Kernel Density Estimate

```
sns.kdeplot(data=df, x='column')
```

Distribution Plot

```
sns.displot(data=df, x='column', kind='hist')
# kind options: 'hist', 'kde', 'ecdf'
```

Box Plot

```
sns.boxplot(data=df, x='category', y='value')
```

Violin Plot

```
sns.violinplot(data=df, x='category', y='value')
```

Slide 24: Relationship Plots

Scatter Plot

```
sns.scatterplot(data=df, x='var1', y='var2',
                hue='category', size='weight')
```

Line Plot

```
sns.lineplot(data=df, x='time', y='value',
             hue='category')
```

Regression Plot

```
sns.regplot(data=df, x='var1', y='var2')
# Adds regression line automatically
```

Joint Plot (Scatter + Distributions)

```
sns.jointplot(data=df, x='var1', y='var2',
              kind='scatter')
# kind: 'scatter', 'reg', 'kde', 'hex'
```

Slide 25: Categorical Plots

Bar Plot (with confidence intervals)

```
sns.barplot(data=df, x='category', y='value')
```

Count Plot

```
sns.countplot(data=df, x='category', hue='subcategory')
```

Strip Plot

```
sns.stripplot(data=df, x='category', y='value')
```

Swarm Plot (non-overlapping points)

```
sns.swarmplot(data=df, x='category', y='value')
```

Point Plot

```
sns.pointplot(data=df, x='time', y='value', hue='category')
```

Slide 26: Matrix Plots

Heatmap

```
# Correlation matrix
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm',
            center=0, square=True)
```

Clustermap

```
sns.clustermap(data, cmap='viridis',
               standard_scale=1)
# Hierarchical clustering visualization
```

Pivot Table Heatmap

```
pivot = df.pivot('row', 'col', 'value')
sns.heatmap(pivot, annot=True, fmt='d')
```

Slide 27: FacetGrid - Multiple Subplots

Creating FacetGrid

```
g = sns.FacetGrid(df, col='category', row='subcategory',
                  height=4, aspect=1.5)
g.map(sns.scatterplot, 'var1', 'var2')
g.add_legend()
```

Using relplot (simpler interface)

```
sns.relplot(data=df, x='var1', y='var2',
            col='category', row='subcategory',
            kind='scatter')
```

Categorical FacetGrid

```
sns.catplot(data=df, x='time', y='value',
            col='category', kind='bar')
```

Slide 28: PairGrid and Pairplot

Pairplot (Quick Overview)

```
 sns.pairplot(df, hue='species')
# Scatter plots for all variable pairs
```

Custom PairGrid

```
 g = sns.PairGrid(df, hue='category')
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.histplot)
g.add_legend()
```

Focused Pairplot

```
sns.pairplot(df, vars=['var1', 'var2', 'var3'],
             hue='category', diag_kind='kde')
```

Slide 29: Seaborn + Matplotlib Integration

Customizing Seaborn Plots

```
 # Seaborn returns matplotlib axes
ax = sns.boxplot(data=df, x='category', y='value')

# Use matplotlib to customize
ax.set_title('My Custom Title', fontsize=14)
ax.set_xlabel('Category', fontsize=12)
ax.axhline(y=50, color='r', linestyle='--', label='Threshold')
ax.legend()

plt.tight_layout()
plt.show()
```

Using with Subplots

```
 fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.boxplot(data=df, x='cat', y='val', ax=axes[0])
sns.violinplot(data=df, x='cat', y='val', ax=axes[1])

plt.tight_layout()
```

Slide 30: Complete Seaborn Example

```
 import seaborn as sns
import matplotlib.pyplot as plt

# Load example dataset
df = sns.load_dataset('tips')

# Set theme
sns.set_theme(style='whitegrid', palette='muted')

# Create figure with subplots
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Plot 1: Distribution
sns.histplot(data=df, x='total_bill', kde=True, ax=axes[0,0])
axes[0,0].set_title('Distribution of Total Bill')

# Plot 2: Categorical comparison
sns.boxplot(data=df, x='day', y='total_bill', ax=axes[0,1])
axes[0,1].set_title('Total Bill by Day')

# Plot 3: Relationship
sns.scatterplot(data=df, x='total_bill', y='tip',
                hue='time', size='size', ax=axes[1,0])
axes[1,0].set_title('Bill vs Tip')

# Plot 4: Heatmap
pivot = df.pivot_table(values='tip', index='day',
                       columns='time', aggfunc='mean')
sns.heatmap(pivot, annot=True, fmt='.2f',
            cmap='YlOrRd', ax=axes[1,1])
axes[1,1].set_title('Average Tip by Day and Time')

plt.tight_layout()
plt.savefig('seaborn_example.png', dpi=300)
plt.show()
```

Slide 31: When to Use Matplotlib vs Seaborn

Use Matplotlib When:

- You need precise, fine-grained control
- Creating custom visualizations
- Working with basic plots
- Building from scratch
- Maximum flexibility required

Use Seaborn When:

- Exploring data quickly
- Creating statistical visualizations
- Working with Pandas DataFrames
- Want beautiful defaults out of the box
- Need complex multi-plot layouts (FacetGrid)

Best Practice

Use both together! Seaborn for structure, Matplotlib for fine-tuning.

Slide 32: Resources

Documentation

- Official Matplotlib docs: matplotlib.org
- Seaborn documentation: seaborn.pydata.org
- Gallery of examples: seaborn.pydata.org/examples

Learning More

- Matplotlib tutorials and user guides
- Seaborn tutorial gallery
- Plotly for interactive plots
- "Fundamentals of Data Visualization" by Claus O. Wilke

Practice

- Kaggle datasets
- Real-world projects
- Reproduce published visualizations
- Seaborn example datasets: `sns.load_dataset()`

Thank You!

Questions?