Python Visualization:

Visualization is a way of representing. Data visualization is representing of data in a graphical or pictorial format for better understanding of data. Visualization gives a good idea of data and the trends in it.

In python, data visualization has multiple libraries like matplotlib, seaborn, plotly etc.

Let us dive deeper into matplotlib and seaborn.

Matplotlib

1. Library Overview

Matplotlib is the foundational plotting library in Python, created by John Hunter in 2003. It provides a MATLAB-like interface for creating static, publication-quality visualizations. Matplotlib serves as the backbone for many other visualization libraries and offers extensive control over every element of a plot.

Key Features:

- Highly customizable plots with fine-grained control
- Extensive documentation and community support
- Publication-ready static visualizations
- Multiple interfaces (pyplot, object-oriented)
- Integration with NumPy and Pandas

Typical Use Cases:

- Academic publications and research papers
- Static reports and presentations
- Exploratory data analysis
- Custom visualization requirements

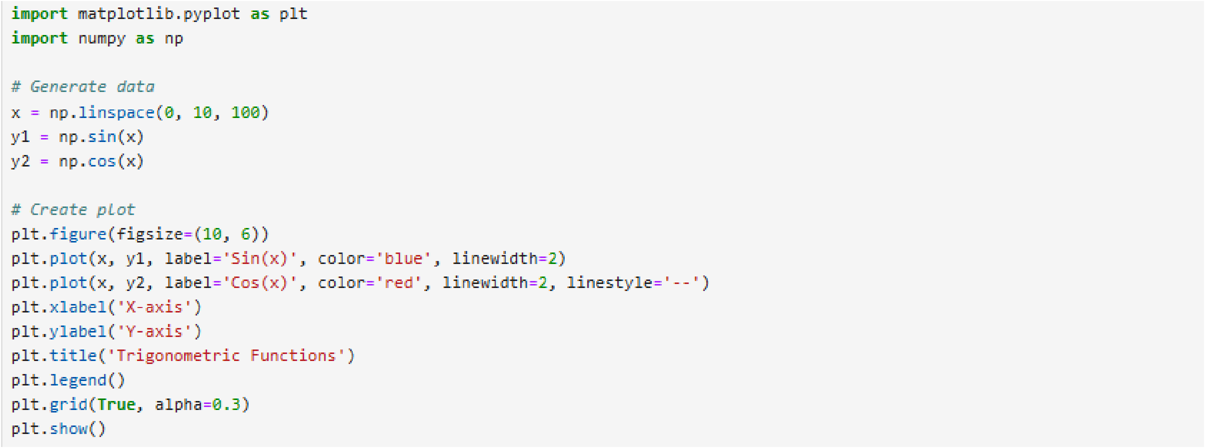
2. Graph Types and Examples

1. Line Plots

Description: Line plots display data points connected by straight lines, ideal for showing trends over continuous intervals.

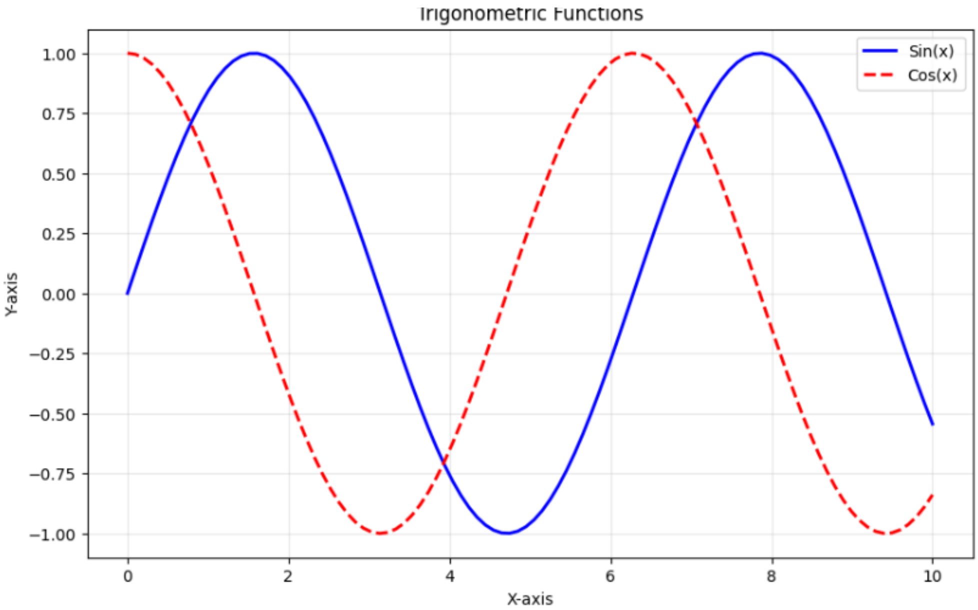
Use Case: Time series analysis, stock prices, temperature changes, performance metrics over time.

Code Snippet:



```python
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 plot
plt.figure(figsize=(10, 6))
plt.plot(x, y1, label='Sin(x)', color='blue', linewidth=2)
plt.plot(x, y2, label='Cos(x)', color='red', linewidth=2, linestyle='--')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Trigonometric Functions')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Output:

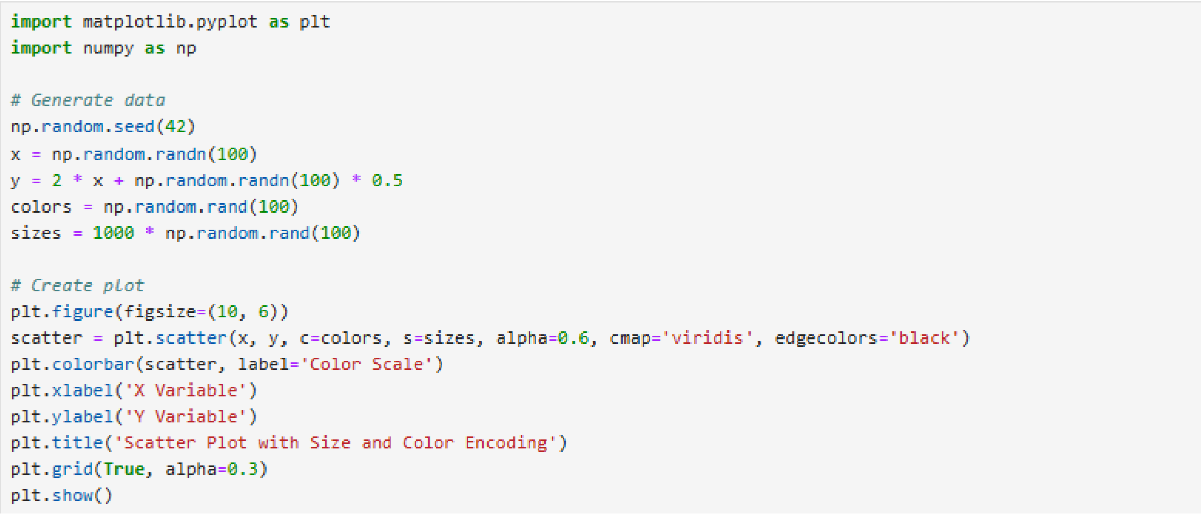


2) Scatter Plots

Description: Scatter plots display individual data points without connecting lines, useful for examining relationships between variables.

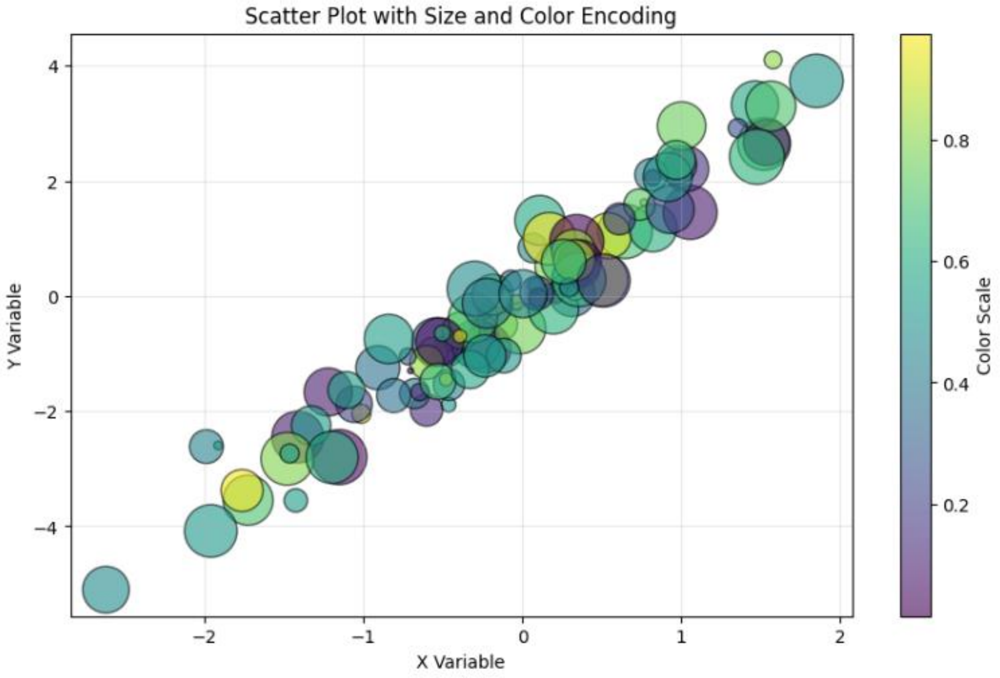
Use Case: Correlation analysis, clustering visualization, outlier detection, pattern recognition.

Code Snippet :



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

# Generate data
np.random.seed(42)
x = np.random.randn(100)
y = 2 * x + np.random.randn(100) * 0.5
colors = np.random.rand(100)
sizes = 1000 * np.random.rand(100)

# Create plot
plt.figure(figsize=(10, 6))
scatter = plt.scatter(x, y, c=colors, s=sizes, alpha=0.6, cmap='viridis', edgecolors='black')
plt.colorbar(scatter, label='Color Scale')
plt.xlabel('X Variable')
plt.ylabel('Y Variable')
plt.title('Scatter Plot with Size and Color Encoding')
plt.grid(True, alpha=0.3)
plt.show()
```

Output :

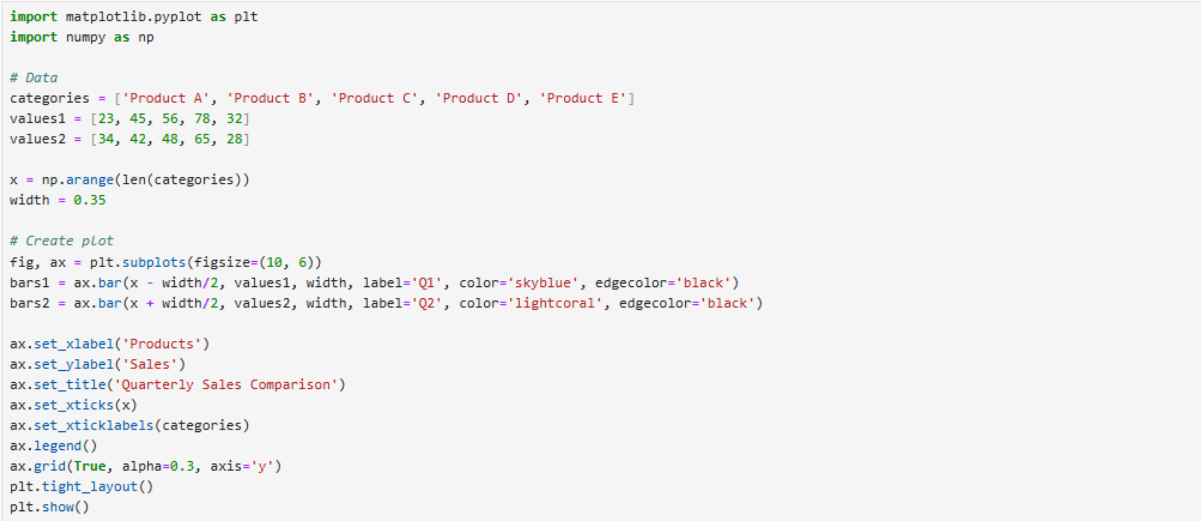


3) Bar Charts

Description: Bar charts use rectangular bars to compare categorical data across different groups.

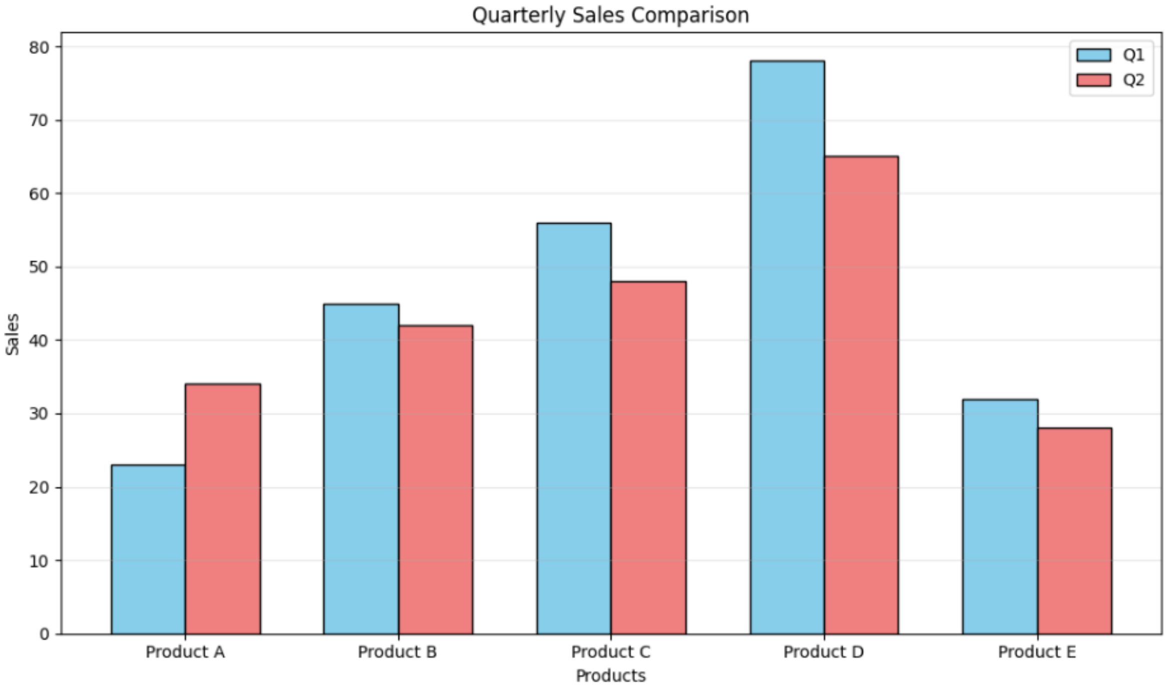
Use Case: Sales comparison, survey results, performance metrics by category, frequency distribution.

Code Snippet:



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

# Data
categories = ['Product A', 'Product B', 'Product C', 'Product D', 'Product E']
values1 = [23, 45, 56, 78, 32]
values2 = [34, 42, 48, 65, 28]

x = np.arange(len(categories))
width = 0.35

# Create plot
fig, ax = plt.subplots(figsize=(10, 6))
bars1 = ax.bar(x - width/2, values1, width, label='Q1', color='skyblue', edgecolor='black')
bars2 = ax.bar(x + width/2, values2, width, label='Q2', color='lightcoral', edgecolor='black')

ax.set_xlabel('Products')
ax.set_ylabel('Sales')
ax.set_title('Quarterly Sales Comparison')
ax.set_xticks(x)
ax.set_xticklabels(categories)
ax.legend()
ax.grid(True, alpha=0.3, axis='y')
plt.tight_layout()
plt.show()
```

Output:

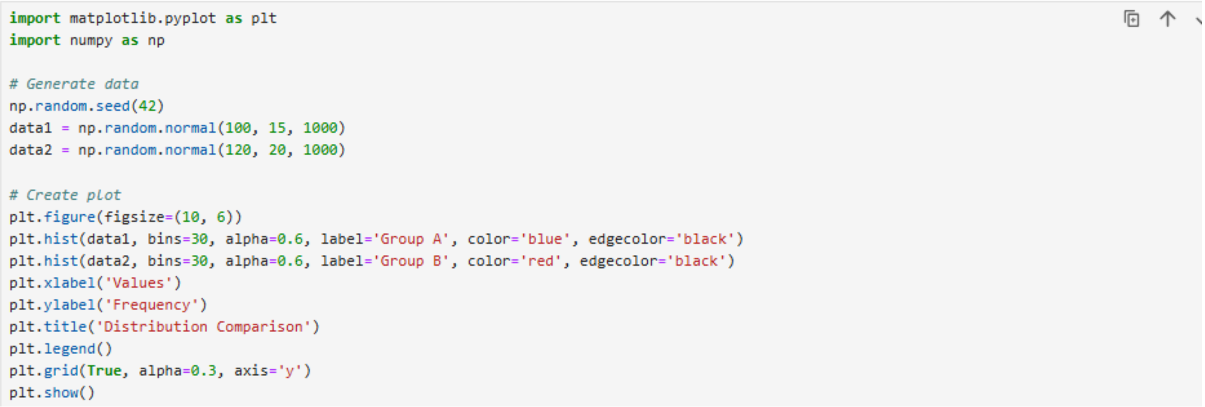


4) Histograms

Description: Histograms show the distribution of numerical data by grouping values into bins.

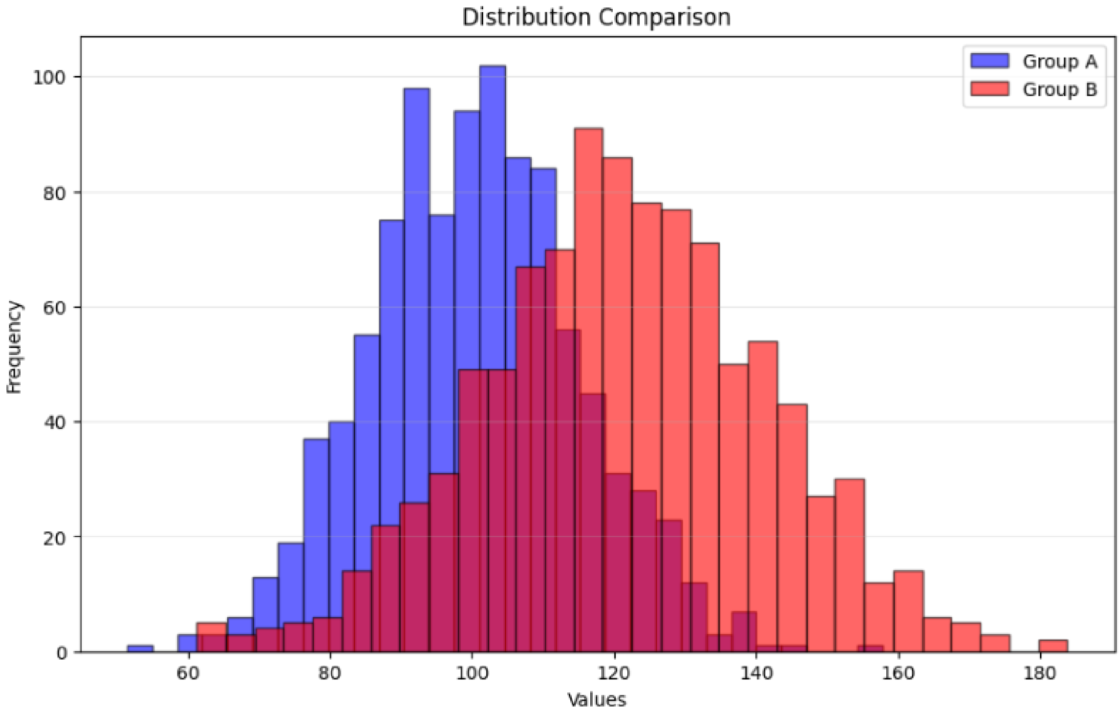
Use Case: Distribution analysis, identifying skewness, understanding data spread, quality control.

Code Snippet:



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

# Generate data
np.random.seed(42)
data1 = np.random.normal(100, 15, 1000)
data2 = np.random.normal(120, 20, 1000)

# Create plot
plt.figure(figsize=(10, 6))
plt.hist(data1, bins=30, alpha=0.6, label='Group A', color='blue', edgecolor='black')
plt.hist(data2, bins=30, alpha=0.6, label='Group B', color='red', edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Distribution Comparison')
plt.legend()
plt.grid(True, alpha=0.3, axis='y')
plt.show()
```

Output:

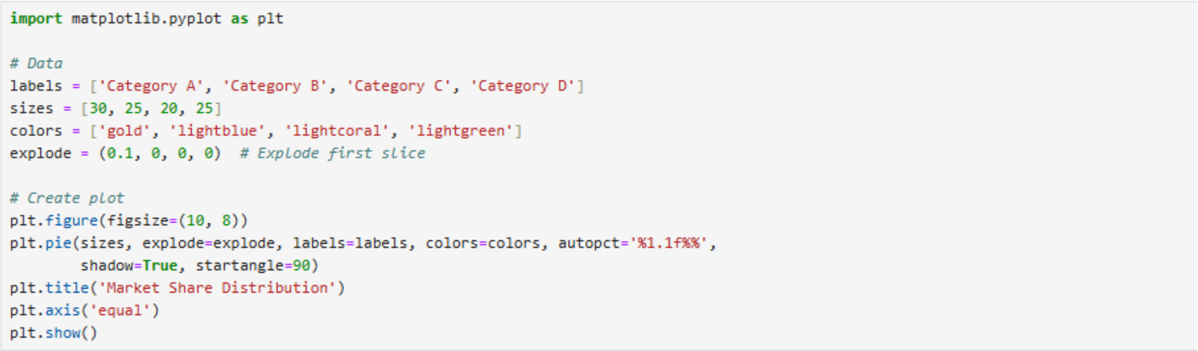


5) Pie Charts

Description: Pie charts display proportions of a whole using circular sectors.

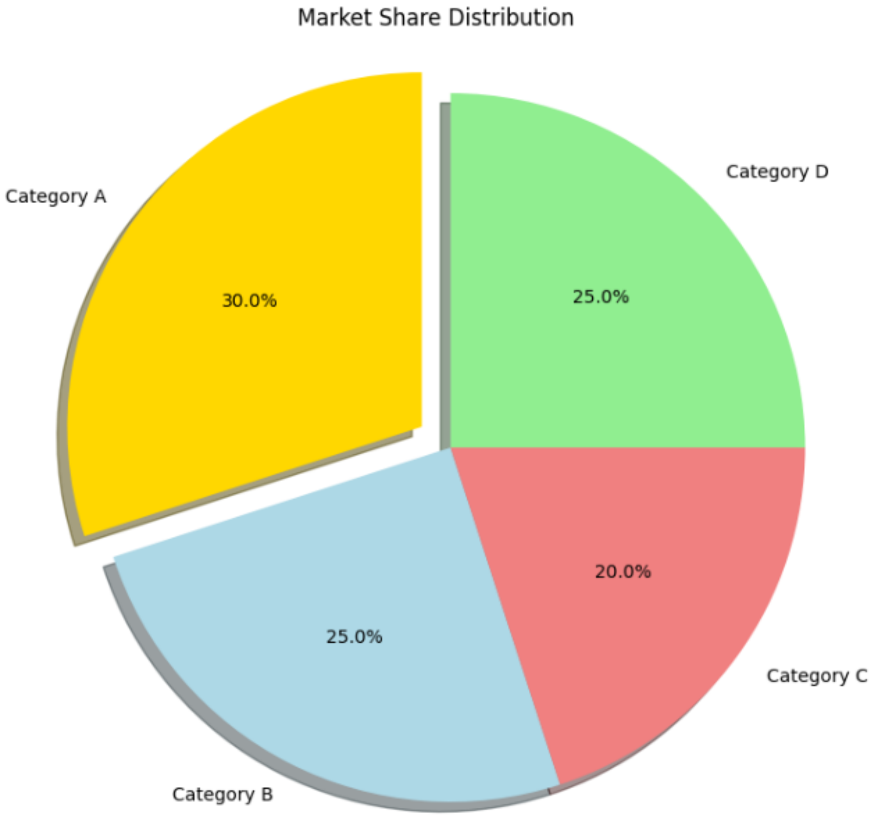
Use Case: Market share analysis, budget allocation, demographic composition, survey results**.**

Code Snippet :



```
import matplotlib.pyplot as plt

# Data
labels = ['Category A', 'Category B', 'Category C', 'Category D']
sizes = [30, 25, 20, 25]
colors = ['gold', 'lightblue', 'lightcoral', 'lightgreen']
explode = (0.1, 0, 0, 0)  # Explode first slice

# Create plot
plt.figure(figsize=(10, 8))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
        shadow=True, startangle=90)
plt.title('Market Share Distribution')
plt.axis('equal')
plt.show()
```

Ouptut:

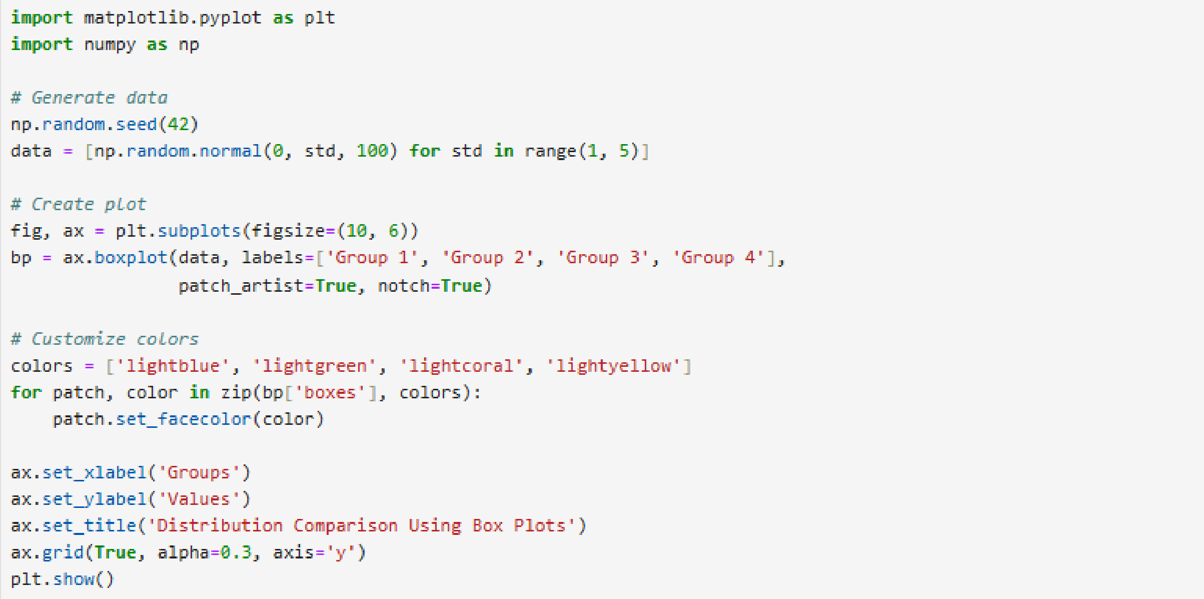


6) Box Plots

Description: Box plots display the distribution of data through quartiles, showing median, outliers, and spread.

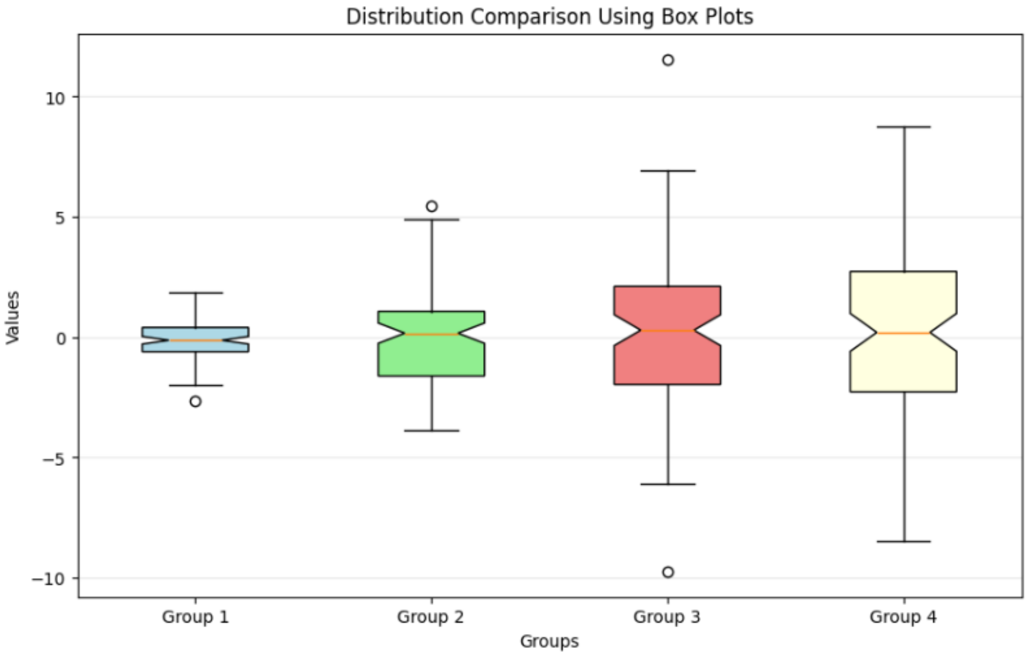
Use Case: Statistical analysis, comparing distributions, identifying outliers, quality assurance.

Code Snippet:



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

# Generate data
np.random.seed(42)
data = [np.random.normal(0, std, 100) for std in range(1, 5)]

# Create plot
fig, ax = plt.subplots(figsize=(10, 6))
bp = ax.boxplot(data, labels=['Group 1', 'Group 2', 'Group 3', 'Group 4'],
                patch_artist=True, notch=True)

# Customize colors
colors = ['lightblue', 'lightgreen', 'lightcoral', 'lightyellow']
for patch, color in zip(bp['boxes'], colors):
    patch.set_facecolor(color)

ax.set_xlabel('Groups')
ax.set_ylabel('Values')
ax.set_title('Distribution Comparison Using Box Plots')
ax.grid(True, alpha=0.3, axis='y')
plt.show()
```

Output:

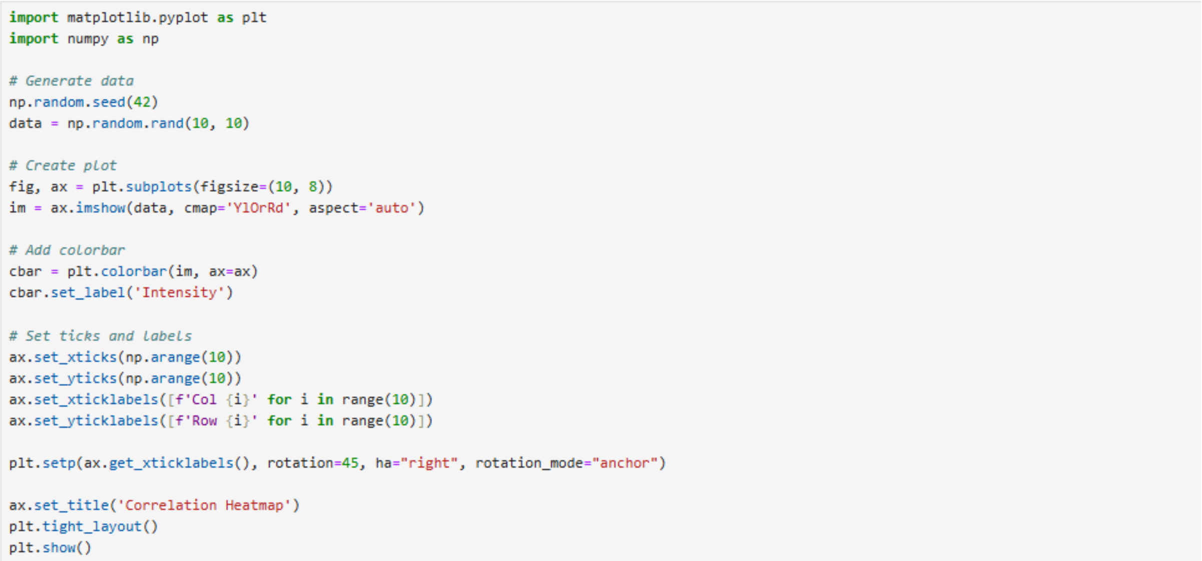


7) Heatmaps

Description: Heatmaps use color intensity to represent values in a matrix format.

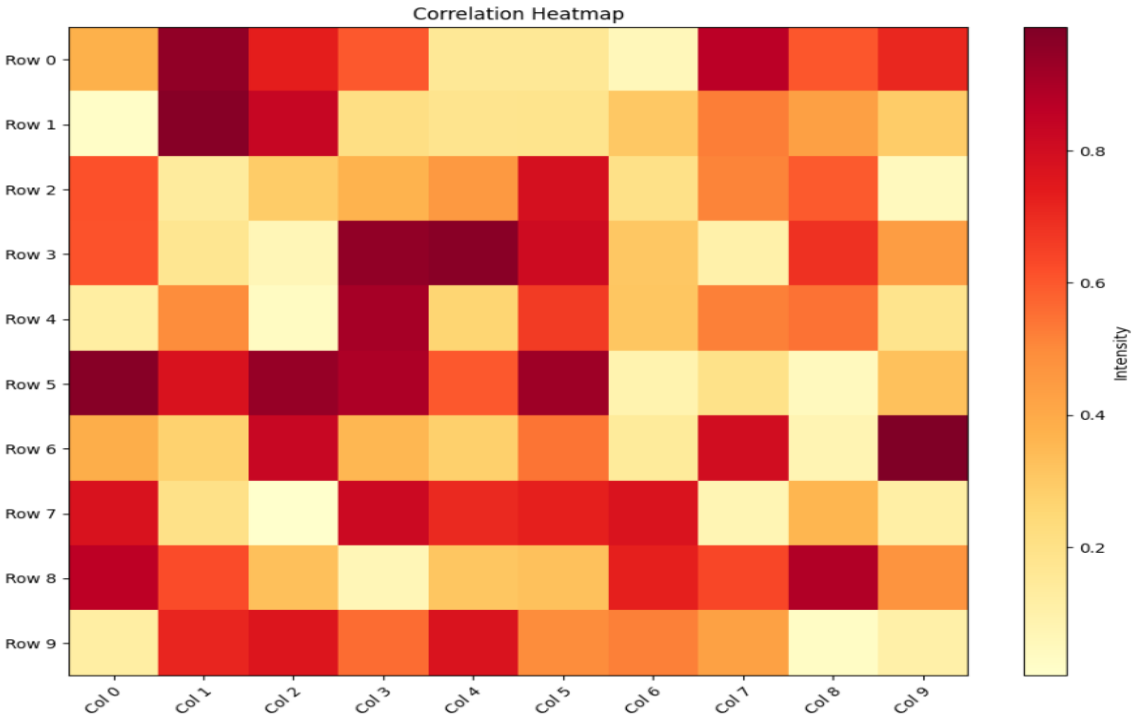
Use Case: Correlation analysis, time-based patterns, geographical data, confusion matrices.

Code Snippet :



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

# Generate data
np.random.seed(42)
data = np.random.rand(10, 10)

# Create plot
fig, ax = plt.subplots(figsize=(10, 8))
im = ax.imshow(data, cmap='YlOrRd', aspect='auto')

# Add colorbar
cbar = plt.colorbar(im, ax=ax)
cbar.set_label('Intensity')

# Set ticks and labels
ax.set_xticks(np.arange(10))
ax.set_yticks(np.arange(10))
ax.set_xticklabels([f'Col {i}' for i in range(10)])
ax.set_yticklabels([f'Row {i}' for i in range(10)])

plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

ax.set_title('Correlation Heatmap')
plt.tight_layout()
plt.show()
```

Output:

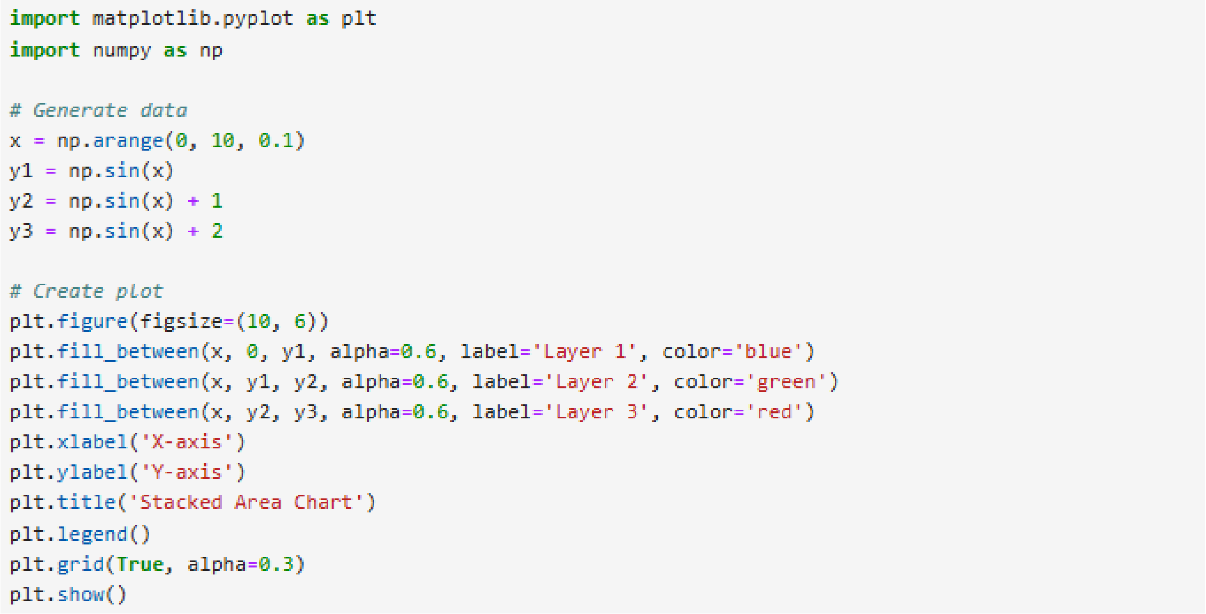


8) Area Plots

Description: Area plots are similar to line plots but with the area below the line filled.

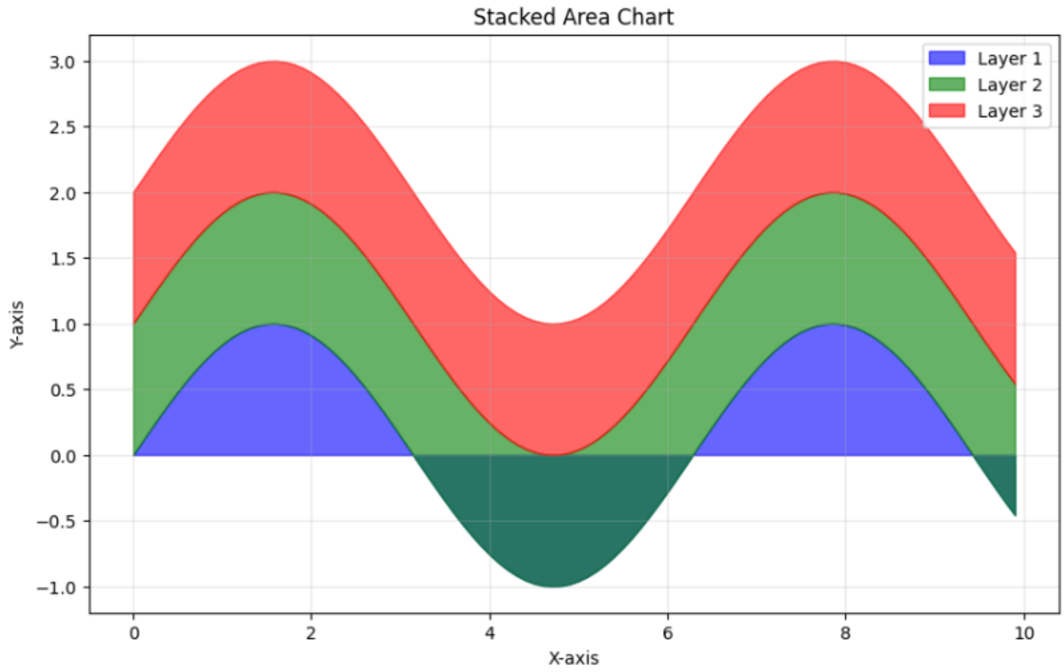
Use Case: Stacked data visualization, cumulative totals, showing contribution of components over time.

Code Snippet :



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

# Generate data
x = np.arange(0, 10, 0.1)
y1 = np.sin(x)
y2 = np.sin(x) + 1
y3 = np.sin(x) + 2

# Create plot
plt.figure(figsize=(10, 6))
plt.fill_between(x, 0, y1, alpha=0.6, label='Layer 1', color='blue')
plt.fill_between(x, y1, y2, alpha=0.6, label='Layer 2', color='green')
plt.fill_between(x, y2, y3, alpha=0.6, label='Layer 3', color='red')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Stacked Area Chart')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Output:



Summary Recommendations

Choose Matplotlib when:

- Creating publication-quality static figures
- You need maximum customization control
- Working on academic or scientific papers
- Output format needs to be vector graphics (PDF, SVG)
- You're working within the scientific Python ecosystem
- Performance with moderately large datasets is critical

Ease of Use

Matplotlib:

- Steeper learning curve, especially for beginners
- Two interfaces (pyplot and object-oriented) can be confusing initially
- Requires more code for complex visualizations
- Extensive documentation but sometimes overwhelming

Interactivity

Matplotlib:

- Primarily static visualizations
- Limited interactivity through widgets in Jupyter
- Can add basic zoom/pan with toolbar in GUI backends
- Not designed for web-based interactive experiences

Performance with Large Datasets

Matplotlib:

- Efficient for datasets up to hundreds of thousands of points
- Performance degrades with very large datasets (millions of points)
- Rendering can be slow for complex figures
- Limited by Python's single-threaded nature
- Can use data aggregation techniques for large data

Seaborn: Statistical Data Visualization Library

Library Overview

Seaborn is a Python data visualization library built on top of Matplotlib that provides a high-level interface for creating attractive and informative statistical graphics. Created by Michael Waskom, Seaborn is designed to make visualization a central part of exploring and understanding data.

Unique Features:

- **Built-in themes and color palettes** that make it easy to create visually appealing plots
- **Statistical plotting functions** that automatically compute and visualize statistical relationships
- **Dataset-oriented API** that works seamlessly with pandas DataFrames
- **Automatic handling of missing data** and statistical aggregation
- **Built-in support for multi-plot grids** for complex visualizations
- **Integration with the PyData stack** (NumPy, pandas, matplotlib)

Typical Use Cases:

- Exploratory data analysis (EDA)
- Statistical modeling and hypothesis testing visualization
- Correlation and relationship analysis
- Distribution analysis and comparison
- Time series analysis
- Academic research and publication-quality figures

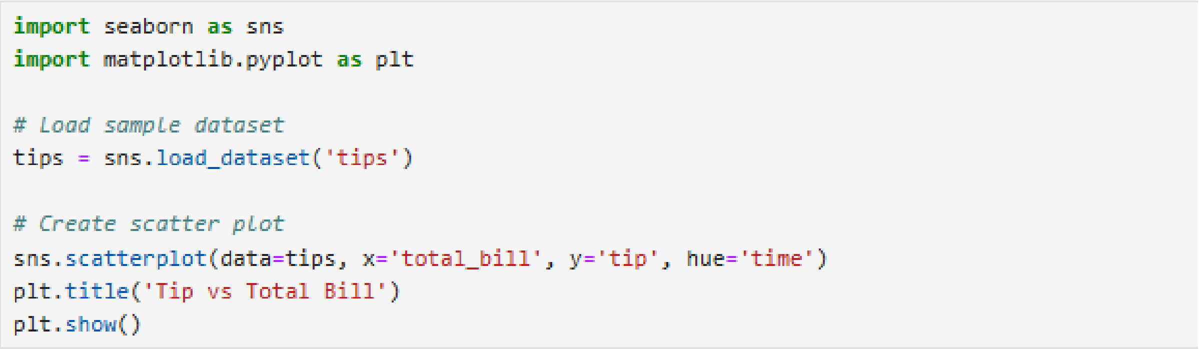
Graph Types in Seaborn

1. Scatter Plot (scatterplot)

Description: Displays the relationship between two continuous variables as points in 2D space.

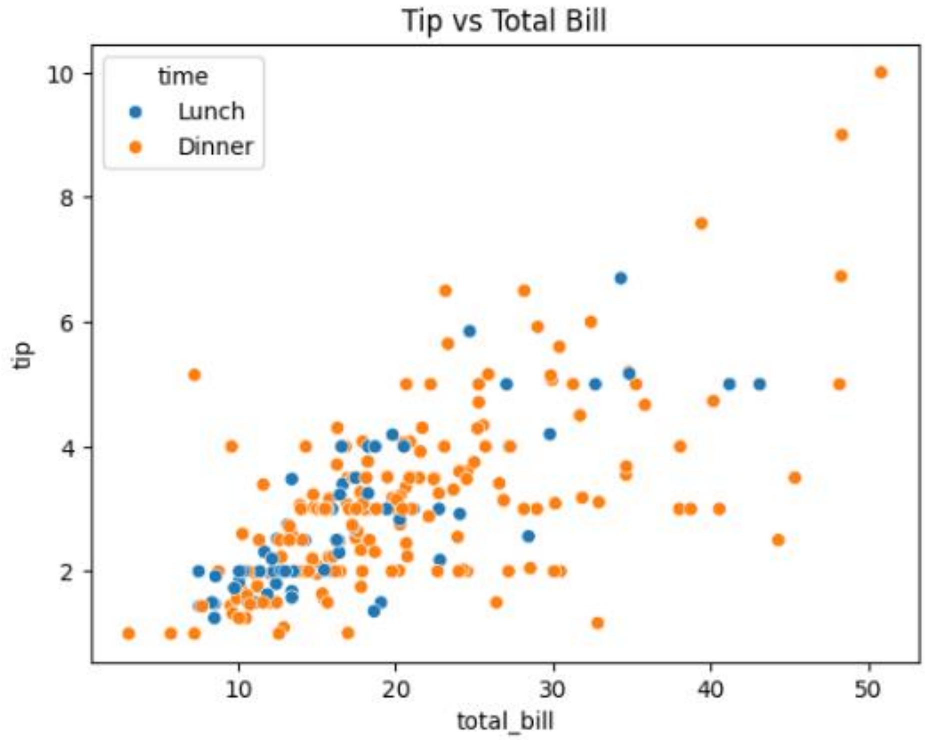
Use Case: Examining correlations, identifying outliers, and visualizing clusters in data.

Code Snippet:



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

# Load sample dataset
tips = sns.load_dataset('tips')

# Create scatter plot
sns.scatterplot(data=tips, x='total_bill', y='tip', hue='time')
plt.title('Tip vs Total Bill')
plt.show()
```

Output:

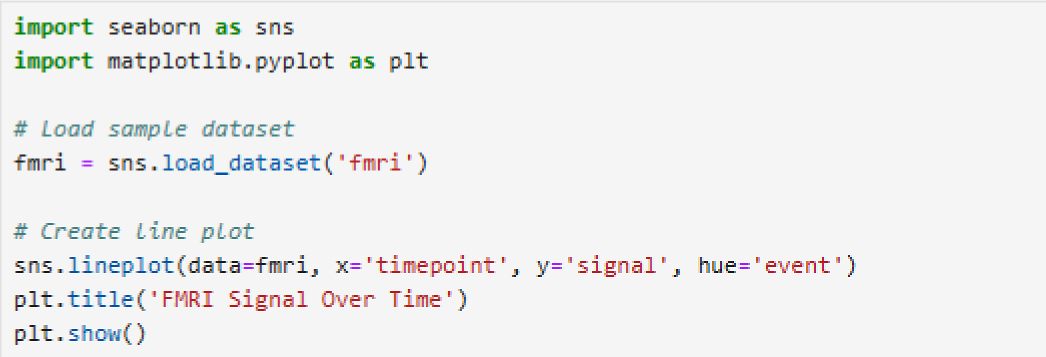


2. Line Plot (lineplot)

Description: Shows trends over a continuous variable (often time) with confidence intervals.

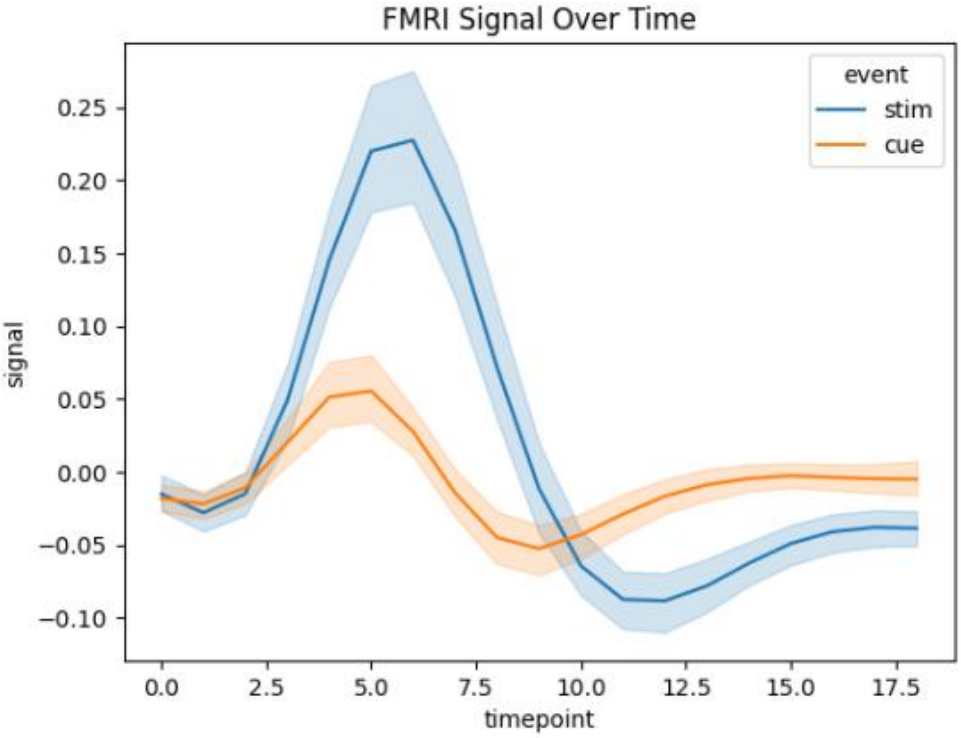
Use Case: Time series analysis, tracking changes over time, comparing trends across categories.

Code Snippet:



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

# Load sample dataset
fmri = sns.load_dataset('fmri')

# Create line plot
sns.lineplot(data=fmri, x='timepoint', y='signal', hue='event')
plt.title('FMRI Signal Over Time')
plt.show()
```

Output:

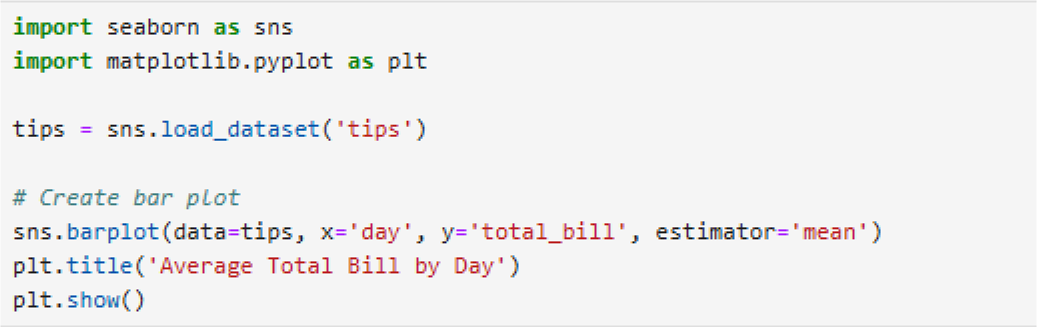


3. Bar Plot (barplot)

Description: Shows point estimates and confidence intervals with rectangular bars.

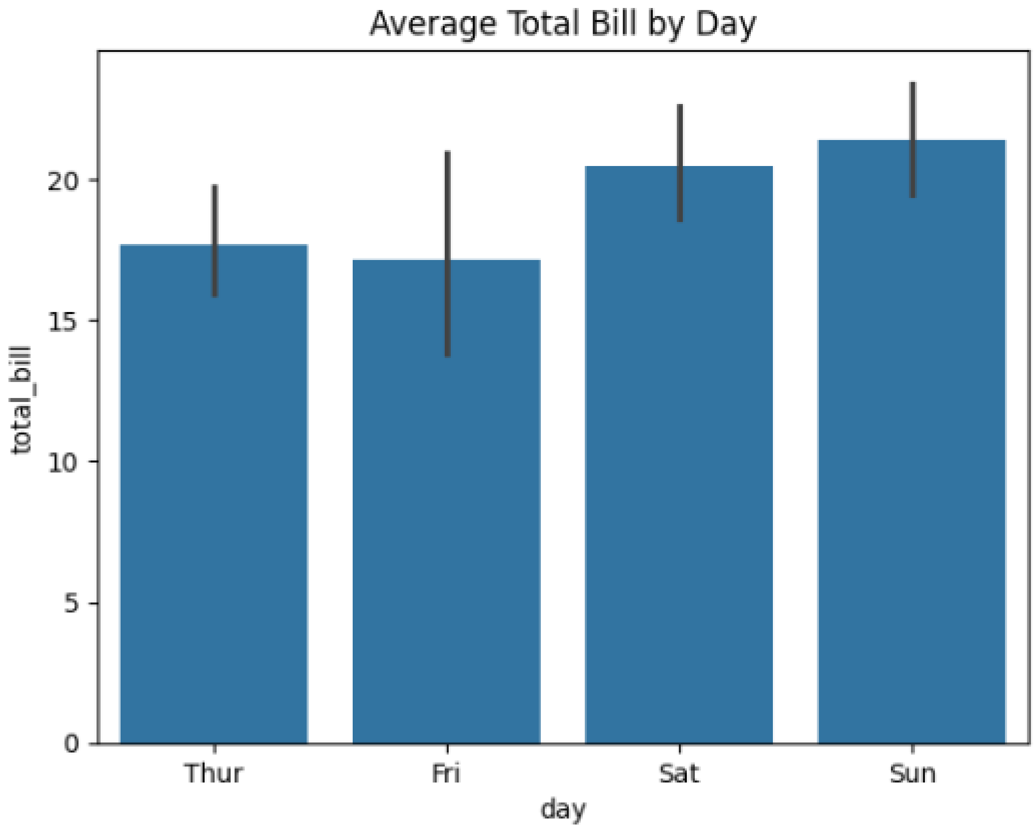
Use Case: Comparing means or other statistics across categories.

Code Snippet:



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

tips = sns.load_dataset('tips')

# Create bar plot
sns.barplot(data=tips, x='day', y='total_bill', estimator='mean')
plt.title('Average Total Bill by Day')
plt.show()
```

Output:

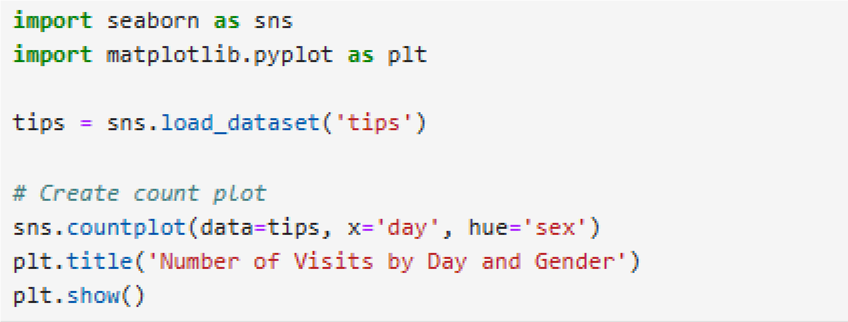


4. Count Plot (countplot)

Description: Shows the count of observations in each categorical bin using bars.

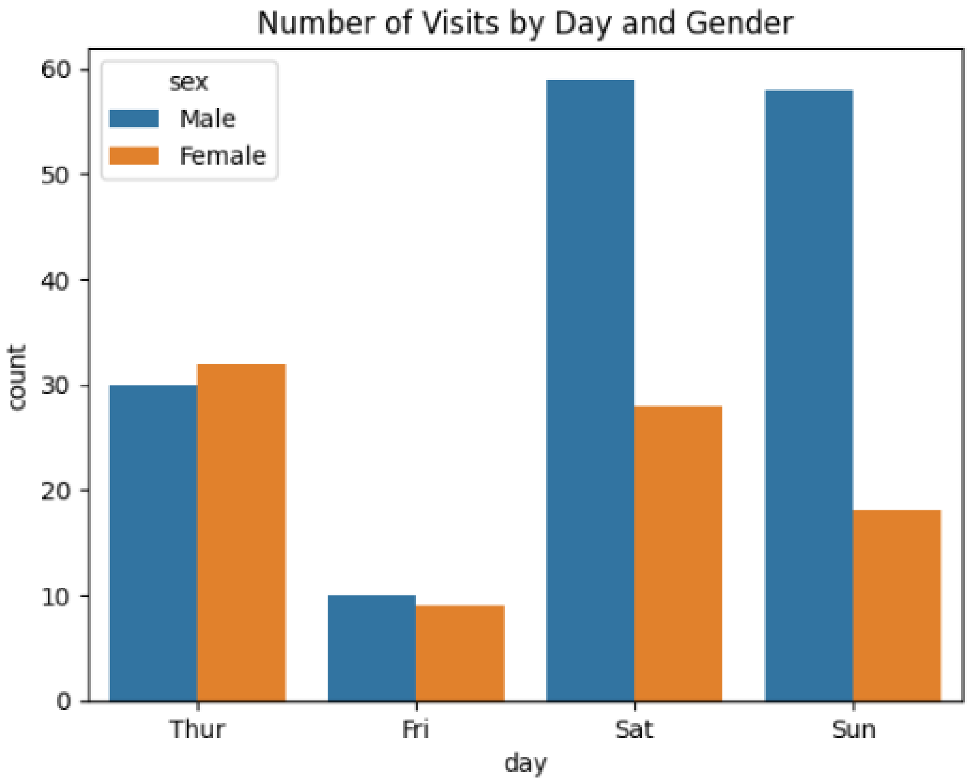
Use Case: Displaying frequency distributions of categorical variables.

Snippet :



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

tips = sns.load_dataset('tips')

# Create count plot
sns.countplot(data=tips, x='day', hue='sex')
plt.title('Number of Visits by Day and Gender')
plt.show()
```

Output:

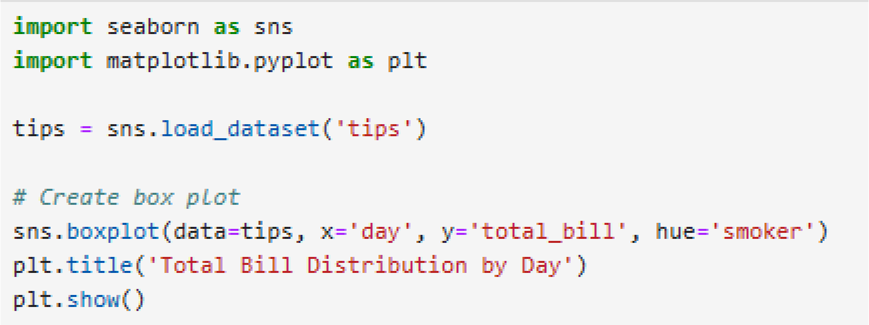


5. Box Plot (boxplot)

Description: Displays the distribution of data through quartiles, showing median, outliers, and spread.

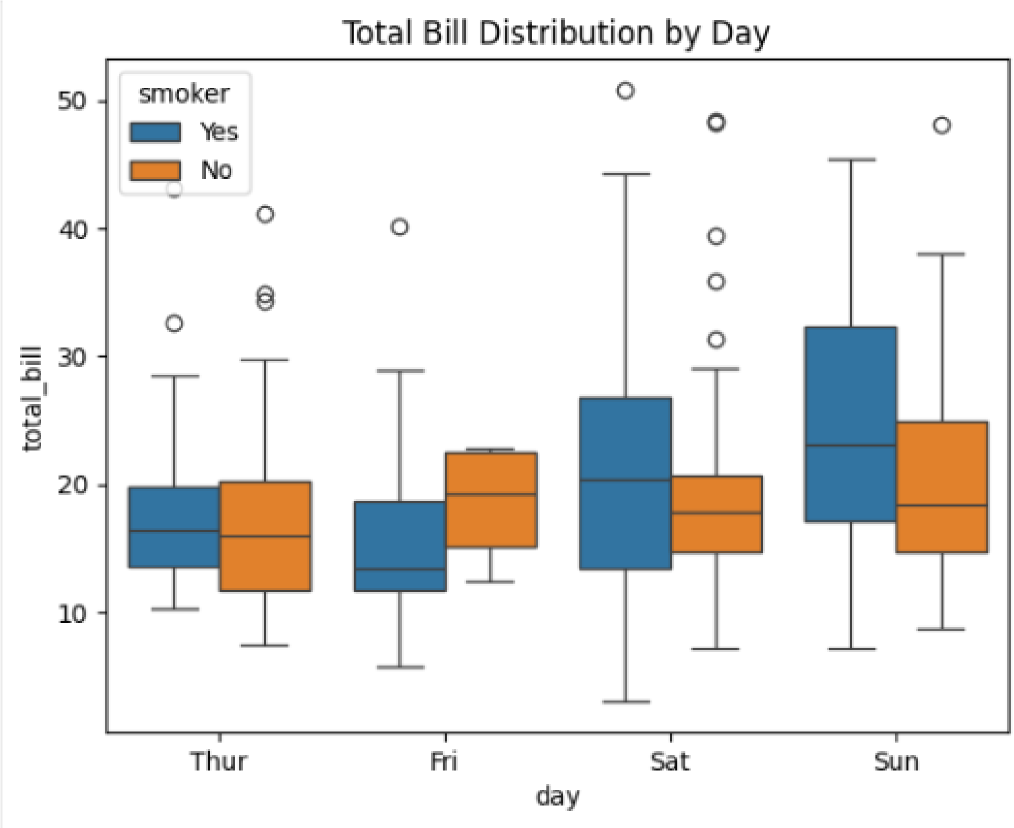
Use Case: Comparing distributions across categories, identifying outliers, understanding data spread.

Snippet:



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

tips = sns.load_dataset('tips')

# Create box plot
sns.boxplot(data=tips, x='day', y='total_bill', hue='smoker')
plt.title('Total Bill Distribution by Day')
plt.show()
```

Output:

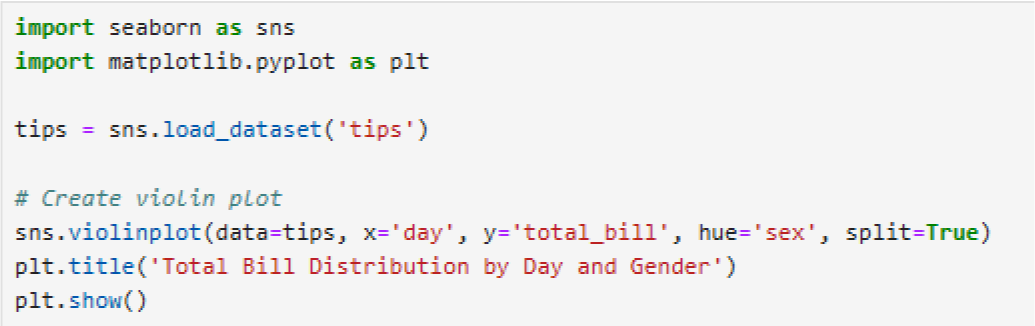


6. Violin Plot (violinplot)

Description: Combines box plot with a kernel density estimate to show distribution shape.

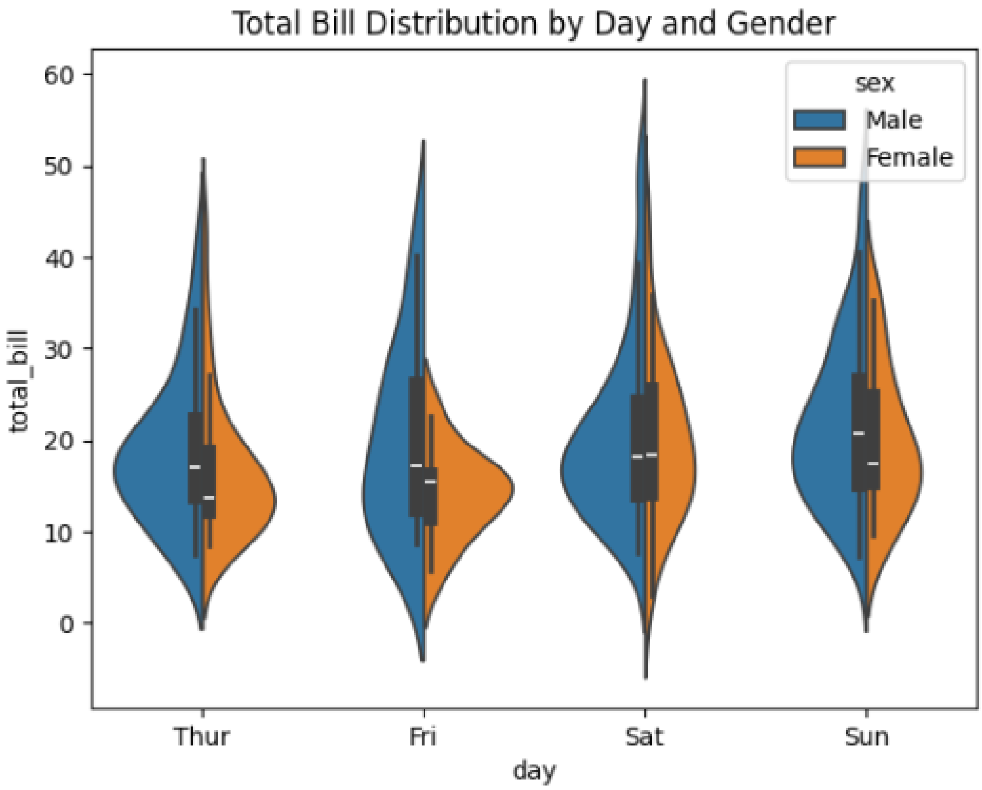
Use Case: Comparing distributions when you need to see the full probability density.

Snippet:



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

tips = sns.load_dataset('tips')

# Create violin plot
sns.violinplot(data=tips, x='day', y='total_bill', hue='sex', split=True)
plt.title('Total Bill Distribution by Day and Gender')
plt.show()
```

Output:

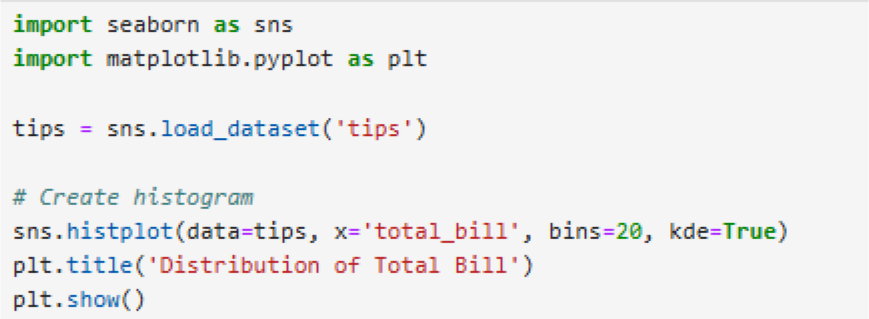


7. Histogram (histplot)

Description: Shows the distribution of a single continuous variable using bins.

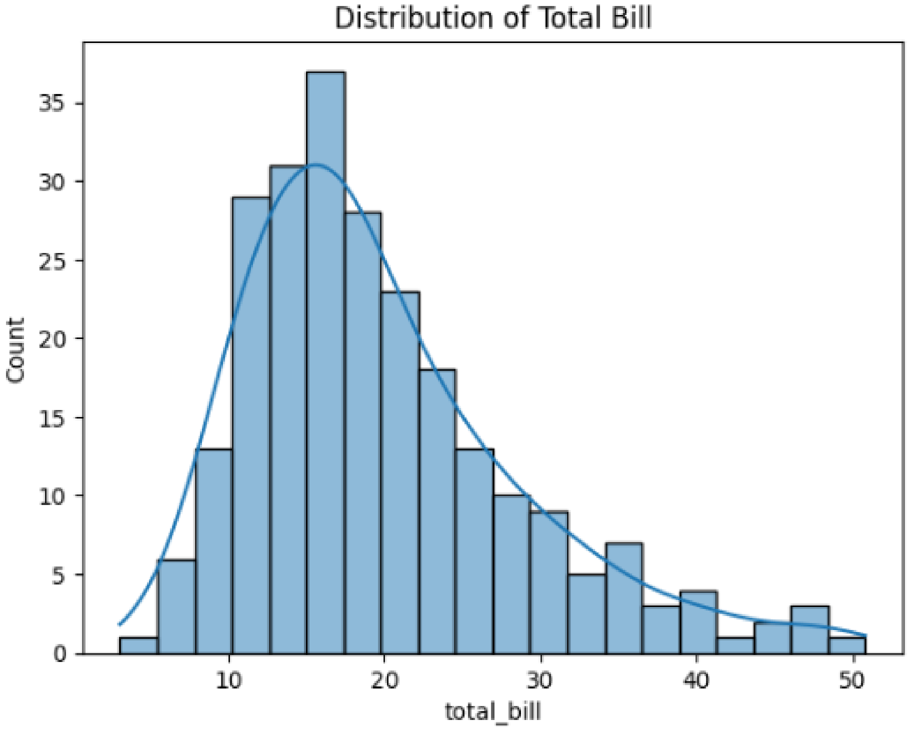
Use Case: Understanding the shape, center, and spread of a distribution.

Snippet:



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

tips = sns.load_dataset('tips')

# Create histogram
sns.histplot(data=tips, x='total_bill', bins=20, kde=True)
plt.title('Distribution of Total Bill')
plt.show()
```

Output:

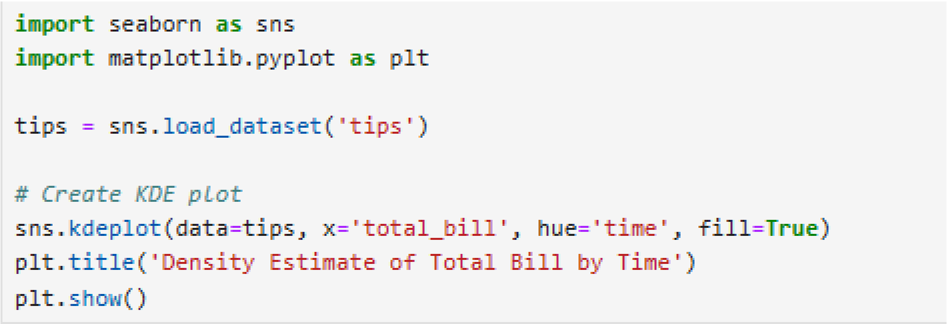


KDE Plot (kdeplot)

Description: Represents the probability density function of a continuous variable using a smooth curve.

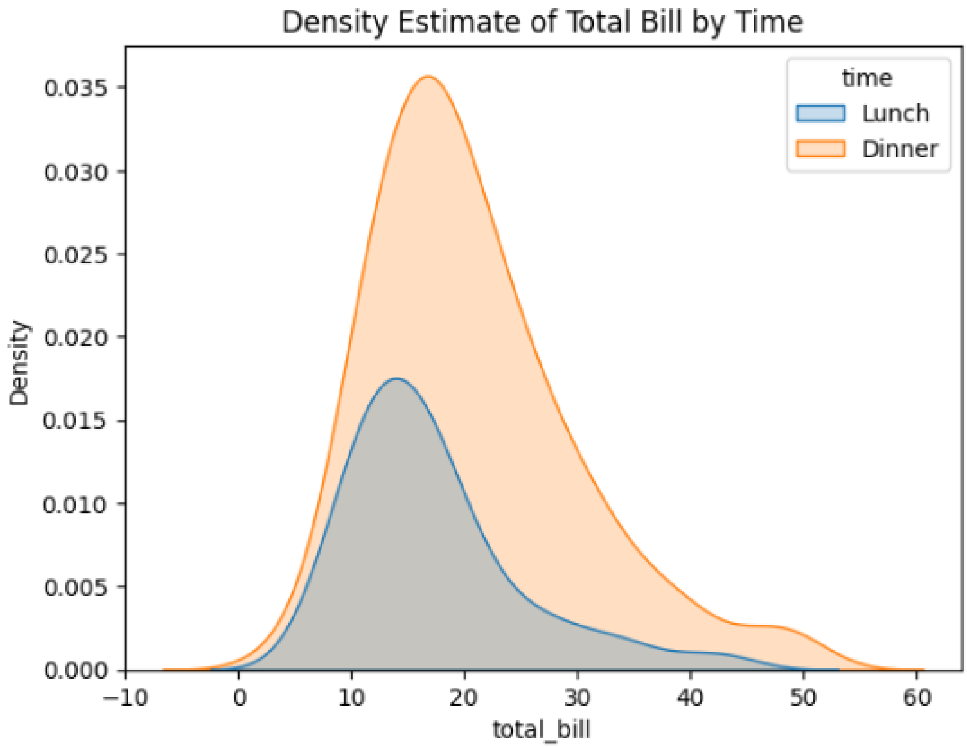
Use Case: Visualizing smooth distribution estimates, comparing multiple distributions.

Snippet:



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

tips = sns.load_dataset('tips')

# Create KDE plot
sns.kdeplot(data=tips, x='total_bill', hue='time', fill=True)
plt.title('Density Estimate of Total Bill by Time')
plt.show()
```

Output:



3. Comparison: Seaborn vs Matplotlib

Ease of Use

Seaborn:

- **Very intuitive and beginner-friendly**
- **High-level interface requires less code**
- **Automatic statistical calculations and visualizations**
- **Works seamlessly with Pandas DataFrames**
- **Built-in beautiful themes and color palettes**
- **One line of code can create complex statistical plots**
- **Rating: ★★★★★**

Matplotlib:

- **Steeper learning curve for beginners**
- **Requires more code for similar visualizations**
- **Two interfaces (pyplot and object-oriented) can be confusing**
- **More manual setup needed for statistical plots**
- **Need to manually configure aesthetics**
- **Rating: ★★★☆☆**

Winner: Seaborn - significantly easier for statistical visualizations and data exploration

---

Customization Options

Seaborn:

- **Good customization through parameters**
- **Limited low-level control compared to Matplotlib**
- **Built on Matplotlib, so you can use Matplotlib commands for fine-tuning**

- **Pre-defined themes and color palettes are excellent but limiting for unique styles**
- **Some plot elements harder to customize**
- **Rating: ★★★☆☆**

Matplotlib:

- **Extremely granular control over every plot element**
- **Can modify virtually any aspect of visualization**
- **Complete control over axes, ticks, labels, colors, styles**
- **Supports custom artists and patches**
- **Best for publication-quality custom graphics**
- **Rating: ★★★★★**

Winner: Matplotlib - unmatched control and customization capabilities

---

Statistical Visualization

Seaborn:

- **Designed specifically for statistical analysis**
- **Automatic calculation of confidence intervals, regression lines**
- **Built-in statistical estimation functions**
- **Excellent for categorical data analysis**
- **Native support for multi-variable relationships (pair plots, joint plots)**
- **Automatic handling of aggregation and error bars**
- **Rating: ★★★★★**

Matplotlib:

- **Requires manual implementation of statistical calculations**
- **No built-in statistical functions**

- Need to use NumPy/SciPy for statistics, then plot manually
- More work to create statistical visualizations
- No automatic error bars or confidence intervals
- Rating: ★★☆☆☆

Winner: Seaborn decisively - purpose-built for statistical visualization

---

Plot Types and Variety

Seaborn:

- Specialized in statistical and categorical plots
- Excellent for: distributions, relationships, categorical comparisons
- Limited for: 3D plots, network graphs, specialized scientific plots
- Focused set of plot types optimized for data science
- Great for standard data analysis workflows
- Rating: ★★★★☆

Matplotlib:

- Comprehensive library with vast plot types
- Supports: basic plots, 3D visualization, animations, custom shapes
- Can create virtually any type of 2D/3D visualization
- More flexibility for non-standard visualizations
- Foundation for specialized plotting libraries
- Rating: ★★★★★

Winner: Matplotlib - broader range of visualization types