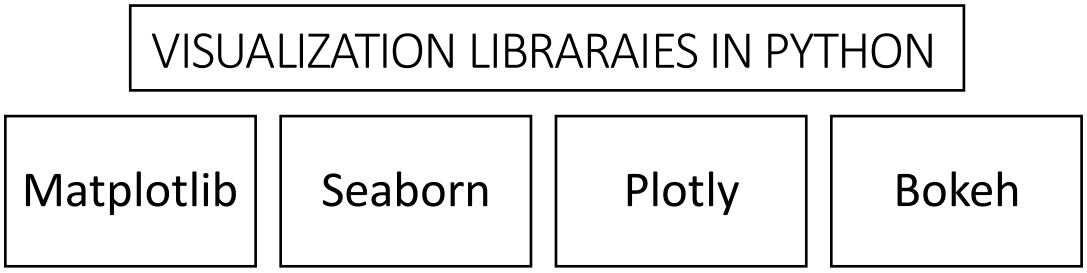
# VISUALIZATION

Visualization is a way of representing information and data in a visual context. Data visualization involves the graphical or pictorial representation of data, making it easier to understand complex datasets. By transforming data into visual formats, such as charts and graphs, visualization provides insights into trends, patterns, and relationships within the data.

In Python, data visualization is supported by several powerful libraries, including Matplotlib, Seaborn, and Plotly, among others. These libraries offer a range of tools and functionalities to create informative and aesthetically pleasing visualizations.




Let us dive deeper into **Matplotlib** and **Seaborn**.

# MATPLOTLIB

- Matplotlib is an open-source visualization library for Python, widely used for creating static, animated, and interactive plots.
- Matplotlib supports a variety of plotting functionalities, including line plots, bar charts, histograms, scatter plots, and 3D visualizations.
- Originally created by John D. Hunter in 2003, Matplotlib has evolved into a fundamental tool for data visualization in Python, extensively utilized by data scientists, researchers, and engineers globally.
- Its versatility and extensive customization options make it suitable for both simple and complex visualizations, while its integration with other libraries like NumPy and Pandas enhances its functionality for data analysis.
- Additionally, Matplotlib's active community contributes to its continuous improvement and a wealth of resources for users.

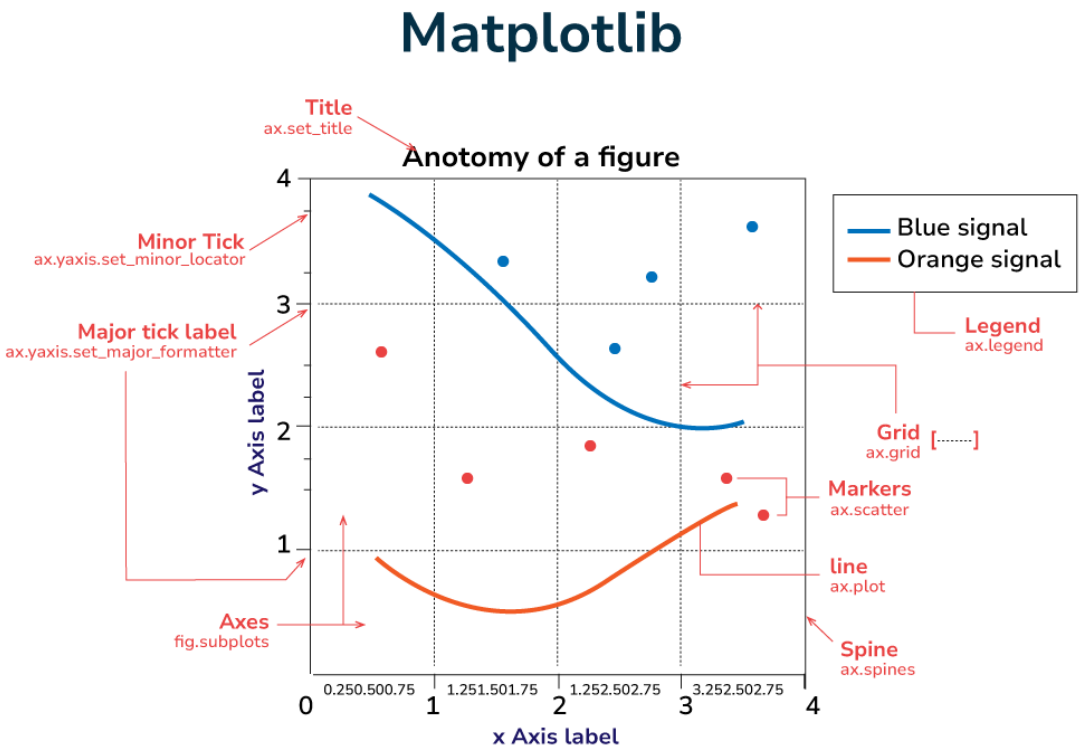
# FEATURES

- **Customizable Plots**: Extensive options for customizing every aspect of a plot, including colors, markers, and annotations.
- **Subplot Functionality**: Ability to create multiple plots in a single figure for easy comparison using the **subplot()** function.
- **Interactive Backends**: Supports interactive features that allow users to zoom, pan, and update plots in real-time.
- **3D Plotting Capabilities**: Includes a toolkit for creating three-dimensional visualizations with **mpl_toolkits.mplot3d**.
- **Animation Support**: Provides functionality for creating animated plots to visualize changes in data over time.

# USE CASES :

- **Data Exploration**: Visualizing datasets to identify trends, patterns, and outliers during the exploratory data analysis phase.
- **Scientific Research**: Creating publication-quality plots for research papers, including graphs for experimental results and statistical analyses.
- **Financial Analysis**: Plotting stock prices, financial indicators, and market trends to analyse performance and make investment decisions.
- **Machine Learning**: Visualizing model performance metrics, such as confusion matrices, ROC curves, and learning curves to evaluate and compare algorithms.
- **Time Series Analysis**: Plotting time series data to observe changes over time, such as temperature variations, sales data, or sensor readings.
- **Comparative Analysis**: Creating side-by-side plots to compare different datasets or variables, such as before-and-after scenarios in experiments.
- **Educational Purposes**: Teaching concepts in mathematics, statistics, and data science through visual representations of data and functions.

# Components or Parts of Matplotlib Figure




A Matplotlib figure consists of several key components or parts, which include:

1. **Figure**: The overall window or page that contains all the elements of the plot. It serves as the top-level container for everything.
2. **Axes**: The area where the data is plotted. A figure can contain multiple axes, and each axes can have its own set of data, scales, and labels.
3. **Axis**: The x-axis and y-axis (and z-axis for 3D plots) that define the coordinate system for the data. Each axis can have its own scale, ticks, and labels.
4. **Ticks**: The markers along the axes that indicate the scale. Ticks can be customized in terms of their position, labels, and appearance.
5. **Labels**: Text labels for the axes (x-label and y-label) and the title of the plot. These provide context and information about the data being visualized.
6. **Legend**: A box that describes the elements of the plot, such as different lines or markers, helping to identify what each represents.

7. **Grid**: Optional lines that run across the plot area, helping to improve readability and make it easier to interpret the data.
8. **Annotations**: Text or markers added to specific points in the plot to provide additional information or highlight important features.
9. **Colorbar**: A visual representation of the color mapping used in the plot, often used in heatmaps or contour plots to indicate the scale of values.
10. **Subplots**: Individual axes within a figure that allow for multiple plots to be displayed in a grid layout, facilitating comparison between different datasets or visualizations.

These components work together to create a comprehensive and informative visualization in Matplotlib.

# Pyplot Overview

A sub-library of Matplotlib that provides a collection of functions for creating various types of plots and visualizations. It simplifies the process of generating plots by providing a MATLAB-like interface.

## Installing Matplotlib in Jupyter Notebook :

```
[1]: pip install matplotlib
     Requirement already satisfied: matplotlib in c:\users\rk73i\anaconda3\lib\site-packages (3.9.2)
```

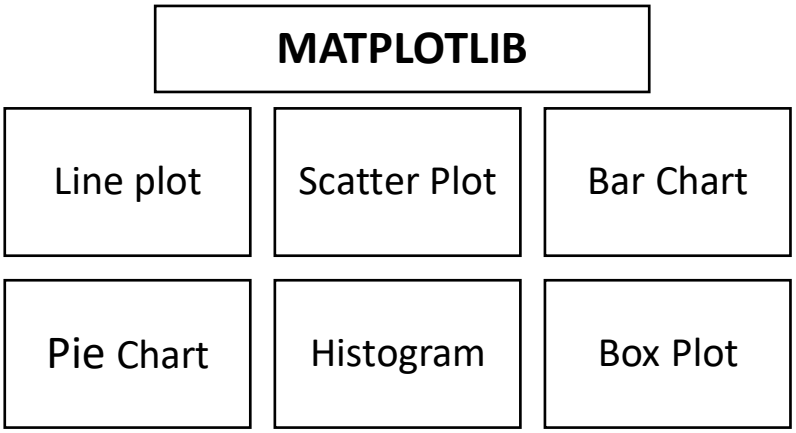
## Importing Matplotlib in Jupyter Notebook :

```
[2]: import matplotlib.pyplot as plt
```

## We also import numpy as a support for the matplotlib :

```
[3]: import numpy as np
```

# Some of the Plots in Matplotlib :




## 1. LINE PLOT :

A line plot displays data points connected by straight lines, making it ideal for showing trends over time or ordered categories. It is commonly used in time-series analysis, such as tracking stock prices, temperature changes, or sales growth.

**Use Case :**

- Tracking monthly sales performance.
- Visualizing temperature fluctuations over a week.

**Code Snippet :**

```
[1]: import matplotlib.pyplot as plt

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May']
sales = [150, 210, 180, 300, 250]

plt.plot(months, sales, marker='o', linestyle='--', color='b', label='Monthly Sales')
plt.xlabel('Months')
plt.ylabel('Sales (in thousands)')
plt.title('Monthly Sales Trend')
plt.legend()
plt.grid(True)
plt.show()
```

**Output :**


**Description :**

- plt.plot() creates the line plot with months on the x-axis and sales on the y-axis.
- marker='o' adds circular markers at each data point.
- linestyle='--' makes the line dashed.
- plt.xlabel(), plt.ylabel(), and plt.title() add axis labels and a title.
- plt.legend() displays the label defined in plot().
- plt.grid(True) adds a grid for better readability.

## 2. Bar Chart :

A bar chart represents categorical data with rectangular bars, where the height/length corresponds to the value. It is useful for comparing quantities across different groups.

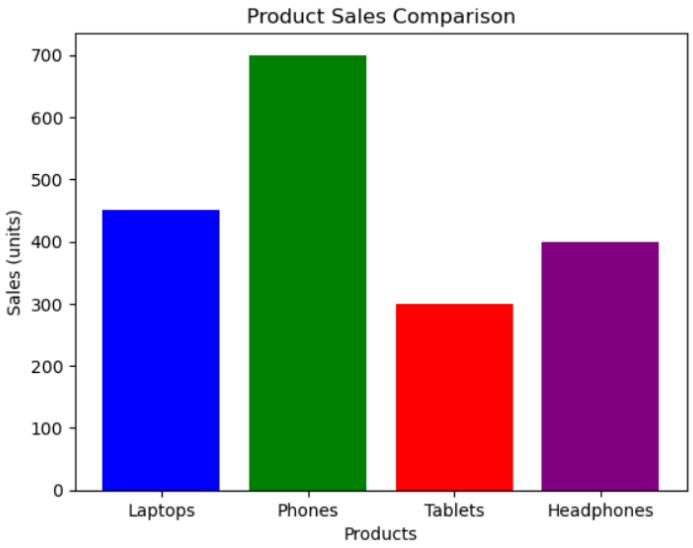
**Use Case:**

- Comparing product sales across different regions.
- Survey results for different age groups.

**Code Snippet:**

```
[3]: import matplotlib.pyplot as plt
products = ['Laptops', 'Phones', 'Tablets', 'Headphones']
sales = [450, 700, 300, 400]
plt.bar(products, sales, color=['blue', 'green', 'red', 'purple'])
plt.xlabel('Products')
plt.ylabel('Sales (units)')
plt.title('Product Sales Comparison')
plt.show()
```

**Output :**




**Description :**

- plt.bar() creates vertical bars for each product.
- The color parameter assigns different colors to each bar.
- plt.xlabel() and plt.ylabel() label the axes.
- plt.title() sets the chart title.

## 3. Scatter Plot :

A scatter plot displays individual data points as dots, making it useful for identifying correlations, clusters, or outliers between two numerical variables.

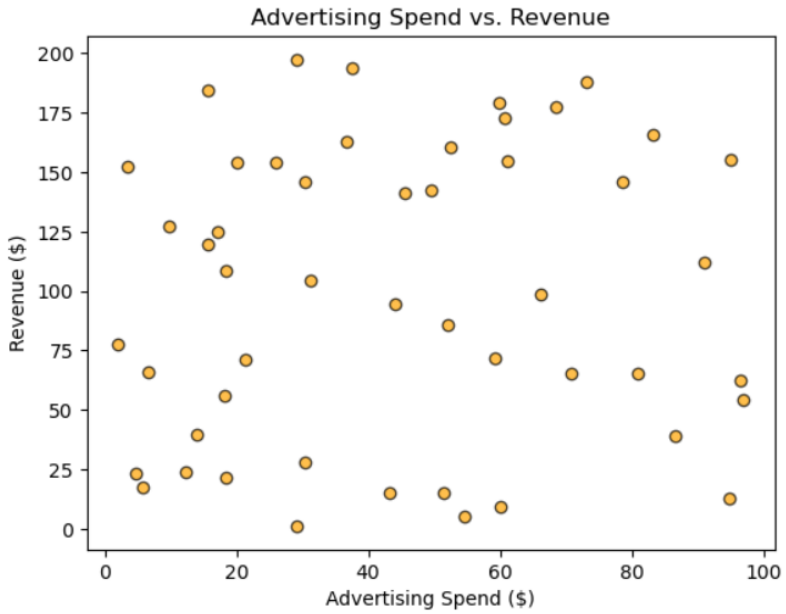
**Use Case:**

- Relationship between height and weight.
- Correlation between advertising spend and revenue.

**Code Snippet:**

```
[4]: import matplotlib.pyplot as plt
     import numpy as np
     np.random.seed(42)
     x = np.random.rand(50) * 100  # Random X values (0-100)
     y = np.random.rand(50) * 200  # Random Y values (0-200)
     plt.scatter(x, y, color='orange', edgecolor='black', alpha=0.7)
     plt.xlabel('Advertising Spend ($)')
     plt.ylabel('Revenue ($)')
     plt.title('Advertising Spend vs. Revenue')
     plt.show()
```

**Output :**




**Description :**

- plt.scatter() plots individual points with x and y values.
- edgecolor='black' adds a border to each point.
- alpha=0.7 makes points slightly transparent to avoid overplotting.

## 4. Histogram :

A histogram visualizes the distribution of numerical data by dividing it into bins and showing the frequency of values in each bin. It helps in understanding data spread, skewness, and outliers.

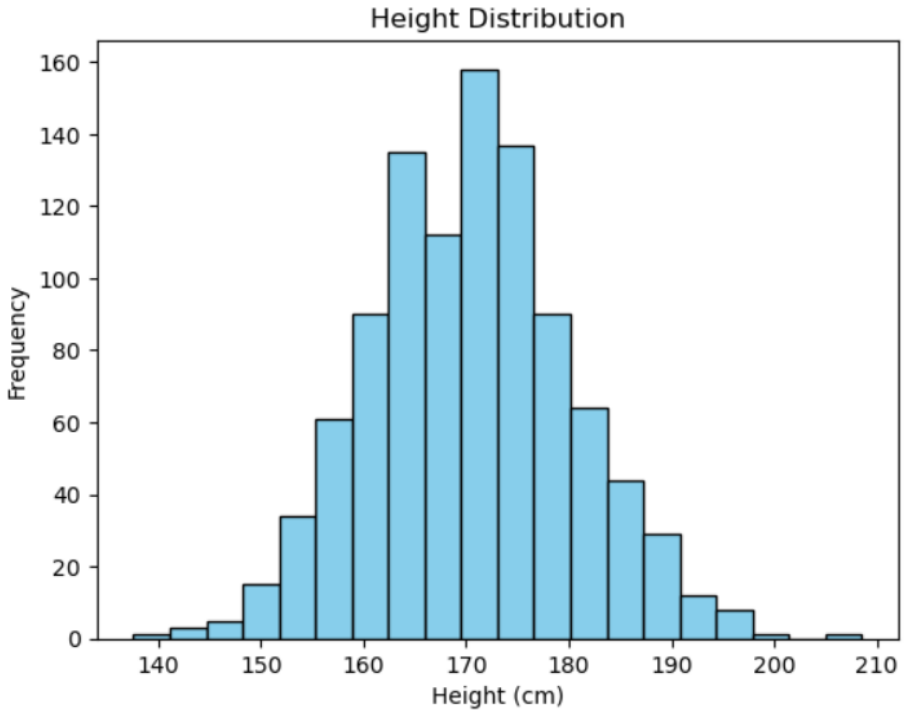
**Use Case:**

- Analyzing exam score distributions.
- Checking the age distribution of survey respondents.

**Code Snippet:**

```
[5]: import matplotlib.pyplot as plt
     import numpy as np
     data = np.random.normal(170, 10, 1000)  # Mean=170, Std=10, 1000 samples
     plt.hist(data, bins=20, color='skyblue', edgecolor='black')
     plt.xlabel('Height (cm)')
     plt.ylabel('Frequency')
     plt.title('Height Distribution')
     plt.show()
```

**Output :**




**Description :**

- plt.hist() creates bins for the data distribution.
- bins=20 divides the data into 20 intervals.
- edgecolor='black' adds borders to bars for clarity.

## 5. Pie Chart :

A pie chart represents parts of a whole as slices, where each slice's size corresponds to its proportion. It is best for showing percentage distributions.

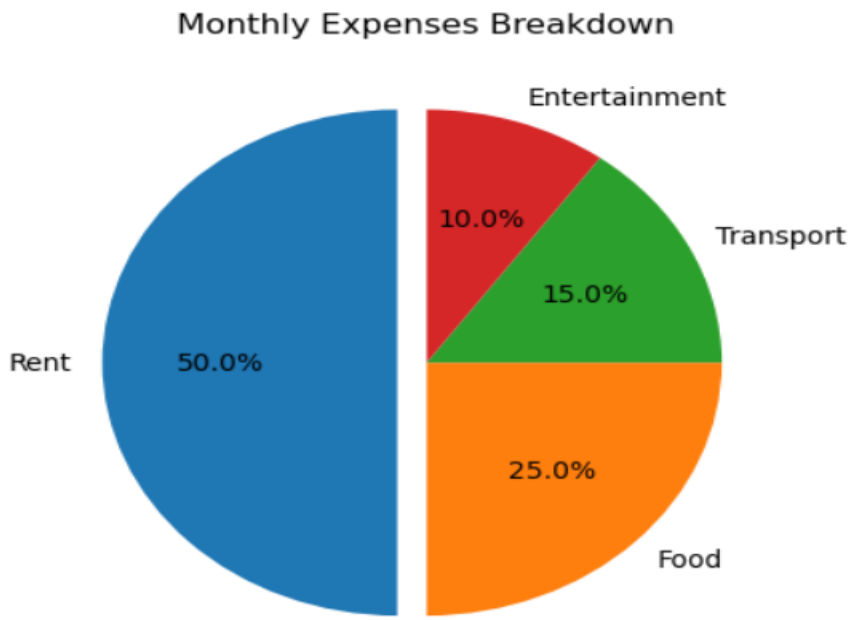
**Use Case:**

- Market share of different companies.
- Budget allocation in a project.

**Code Snippet:**

```
[6]: import matplotlib.pyplot as plt
     categories = ['Rent', 'Food', 'Transport', 'Entertainment']
     expenses = [1000, 500, 300, 200]
     plt.pie(expenses, labels=categories, autopct='%1.1f%%', startangle=90, explode=(0.1, 0, 0, 0))
     plt.title('Monthly Expenses Breakdown')
     plt.show()
```

**Output :**




**Description :**

- plt.pie() creates the pie chart with expenses as values.
- autopct='%1.1f%%' displays percentages with 1 decimal place.
- startangle=90 rotates the chart for better readability.
- explode=(0.1, 0, 0, 0) offsets the first slice for emphasis.

# SEABORN

• Seaborn is a high-level Python data visualization library built on Matplotlib that provides beautiful statistical graphics with minimal code.
• Developed by Michael Waskom in 2012 as an extension to Matplotlib, focusing on statistical visualization.
• Integrates seamlessly with Pandas DataFrames, making it ideal for data analysis workflows.
• Provides attractive default styles and color palettes to create publication-ready visualizations.
• Particularly strong for visualizing relationships between multiple variables and statistical modeling results.

## FEATURES

• **Statistical Visualization**: Specialized in creating plots that reveal relationships in data.
• **Attractive Defaults**: Beautiful styling and color palettes out-of-the-box.
• **Pandas Integration**: Works natively with DataFrame objects.
• **Advanced Plot Types**: Supports complex visualizations like violin plots and cluster maps.
• **Faceting Capabilities**: Easy creation of multiple plot grids with FacetGrid.
• **Regression Visualization**: Built-in functions for plotting linear regression models.
• **Theme Customization**: Multiple built-in themes (darkgrid, whitegrid, dark, white, ticks).

## Plot Categories

**1. Relational Plots :**

Visualize relationships between variables:

- **scatterplot()**: Shows relationship between two numerical variables.
- **lineplot()**: Displays trends over time or ordered categories.
- **relplot()**: Flexible higher-level function for relational plots (combines scatter and line plots).

**2. Categorical Plots**

Visualize distributions and comparisons of categorical data:

- **barplot()**: Shows point estimates and confidence intervals.

- **countplot()**: Displays counts of observations in each category.
- **boxplot()**: Shows distribution quartiles and outliers.
- **violinplot()**: Combines boxplot with kernel density estimation.
- **swarmplot()**: Displays all observations with points adjusted to avoid overlap.
- **pointplot()**: Shows point estimates and confidence intervals.
- **catplot()**: Higher-level interface for categorical plots.

**3. Distribution Plots**

Visualize univariate and bivariate distributions:

- **histplot()**: Histogram of a single variable's distribution.
- **kdeplot()**: Kernel Density Estimate of distribution.
- **rugplot()**: Shows marginal distributions with ticks.
- **distplot()**: Combined histogram and KDE plot (deprecated in favor of histplot).

**4. Regression Plots**

Visualize linear relationships and models:

- **regplot()**: Scatter plot with regression line.
- **lmplot()**: Higher-level regression plotting with faceting.

**5. Matrix Plots**

Visualize matrix-style data:

- **heatmap()**: Color-encoded matrix values.
- **clustermap()**: Heatmap with hierarchical clustering.

## Use Cases

• Statistical Analysis: Visualizing relationships between multiple variables.
• Data Exploration: Quickly understanding distributions and patterns.
• Machine Learning: Evaluating feature relationships and model results.
• Scientific Research: Creating publication-quality statistical graphics.
• Business Analytics: Comparing metrics across categories.

## 1. **Scatter Plot with Hue :**

Visualizes the relationship between two numeric variables using colored dots to represent categories.

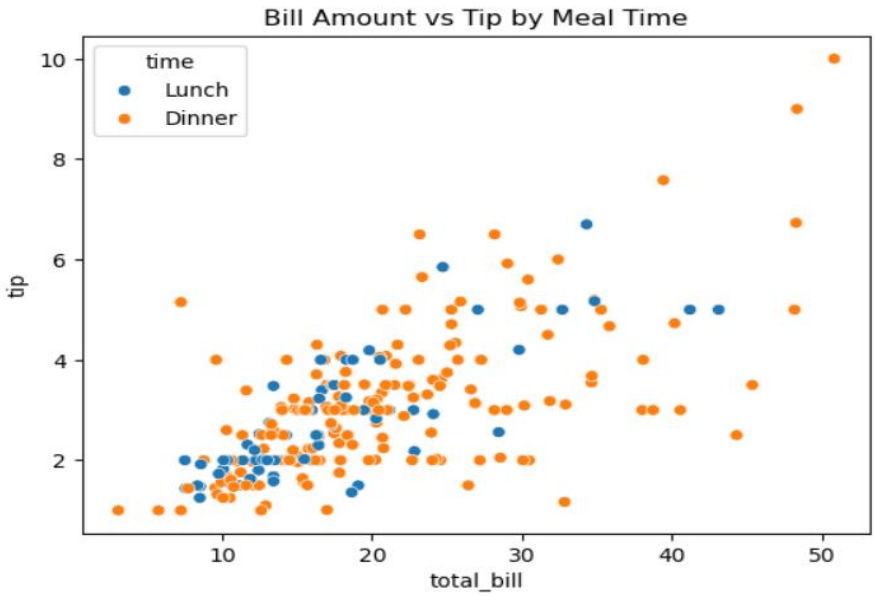
**Use Cases:**

- Customer analysis: Spending vs. Income by gender.
- Biology: Plant height vs. leaf size by species.

**Code snippet :**

```
[10]: import seaborn as sns
      import matplotlib.pyplot as plt
      tips = sns.load_dataset('tips', cache=False)
      sns.scatterplot(data=tips, x='total_bill', y='tip', hue='time')
      plt.title('Bill Amount vs Tip by Meal Time')
      plt.show()
```

**Output :**




**Description :**

A scatter plot showing:

- X-axis: Total bill amount ($)
- Y-axis: Tip amount ($)
- Blue dots: Lunch meals
- Orange dots: Dinner meals
- Clear positive correlation between bill and tip amounts

## 2. Grouped Bar Chart :

Compares values across categories using grouped bars.

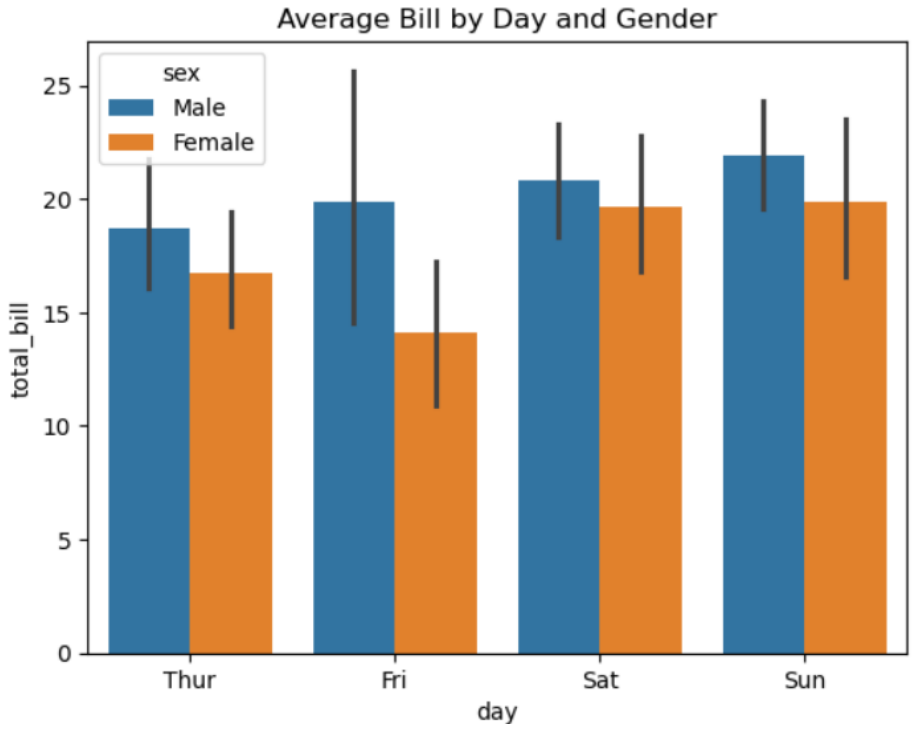
**Use Cases:**

- Sales: Quarterly revenue by product category.
- Education: Test scores by grade and gender.

**Code snippet :**

```
[12]: import seaborn as sns
      import matplotlib.pyplot as plt
      sns.barplot(data=tips, x='day', y='total_bill', hue='sex')
      plt.title('Average Bill by Day and Gender')
      plt.show()
```

**Output :**




**Description :**

Four groups of bars (Thursday-Sunday):

- Each group contains two bars (male/female)
- Blue bars: Male customers
- Orange bars: Female customers
- Sunday shows highest average bills

## 3. **Histogram with KDE :**

Shows data distribution using bars with a smoothed line.

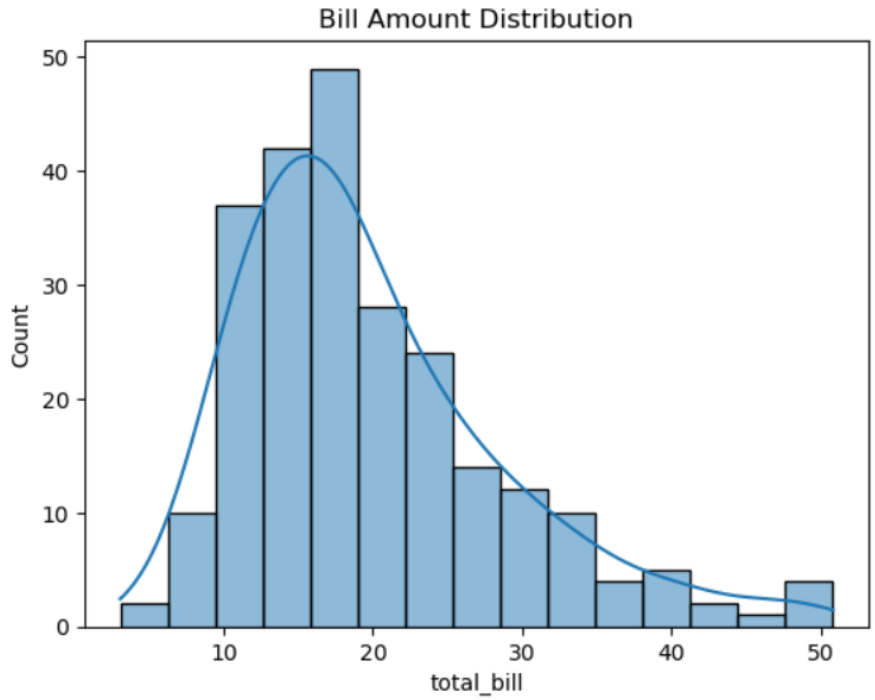
**Use Cases:**

- Finance: Income distribution analysis
- Healthcare: Patient age distribution

**Code snippet :**

```
[13]: import seaborn as sns
      import matplotlib.pyplot as plt
      sns.histplot(data=tips, x='total_bill', bins=15, kde=True)
      plt.title('Bill Amount Distribution')
      plt.show()
```

**Output :**




**Description :**

- 15 blue bars showing bill frequency
- Smooth gold line showing density pattern
- Right-skewed distribution (most bills $10-$30)

## 4. **Box Plot :**

Displays data distribution through quartiles and outliers.

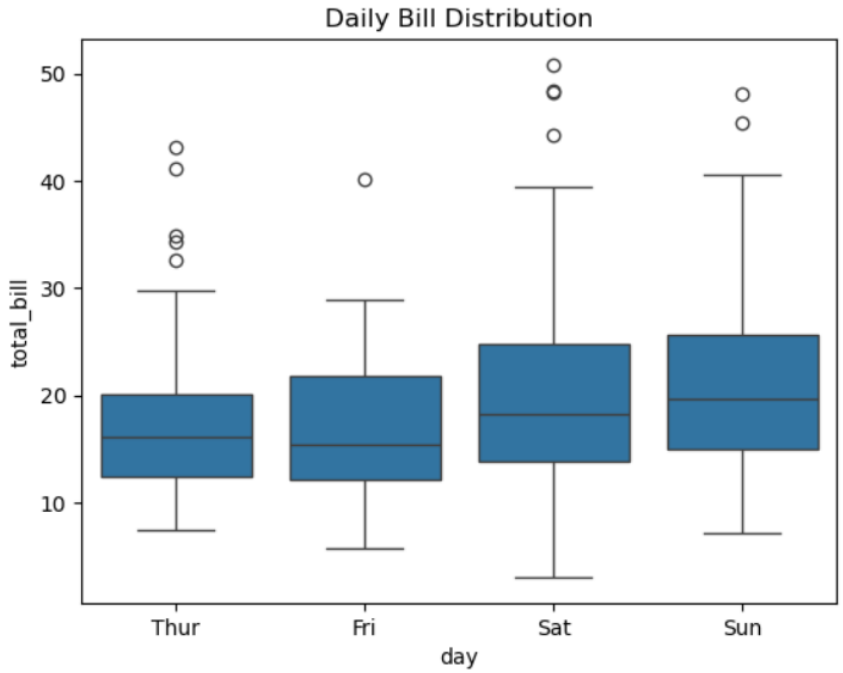
**Use Cases:**

- HR: Salary distribution by department
- Manufacturing: Product defect analysis

**Code snippet :**

```
[14]: import seaborn as sns
      import matplotlib.pyplot as plt
      sns.boxplot(data=tips, x='day', y='total_bill')
      plt.title('Daily Bill Distribution')
      plt.show()
```

**Output :**




**Description :**

Four boxplots (Thursday-Sunday):

- Boxes show 25th-75th percentiles
- Whiskers extend to non-outlier range
- Sunday has widest bill range and highest median

## 5. Heatmap :

Color-coded matrix showing values.

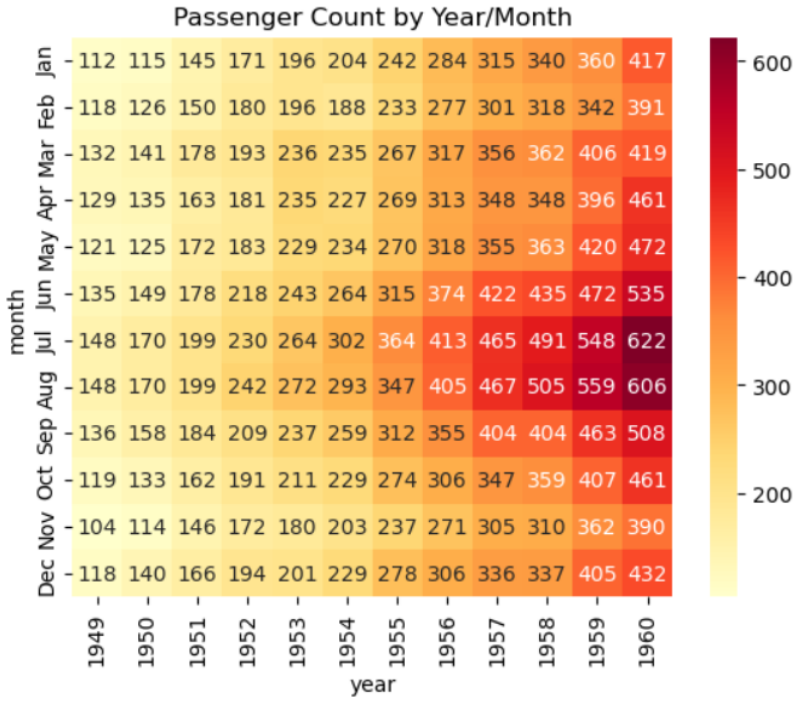
**Use Cases:**

- Marketing: Customer segment analysis
- Biology: Gene expression patterns

**Code snippet :**

```
[17]: import seaborn as sns
      import matplotlib.pyplot as plt
      flights = sns.load_dataset('flights')
      flights_pivot = flights.pivot(index='month', columns='year', values='passengers')
      sns.heatmap(flights_pivot, cmap='YlOrRd', annot=True, fmt='d')
      plt.title('Passenger Count by Year/Month')
      plt.show()
```

**Output :**




**Description :**

- 12x12 color grid (months x years)
- Yellow: Low passenger counts
- Red: High passenger counts
- Summer months show brightest colors

## 6. Violin Plot :

Combines box plot and density estimation.

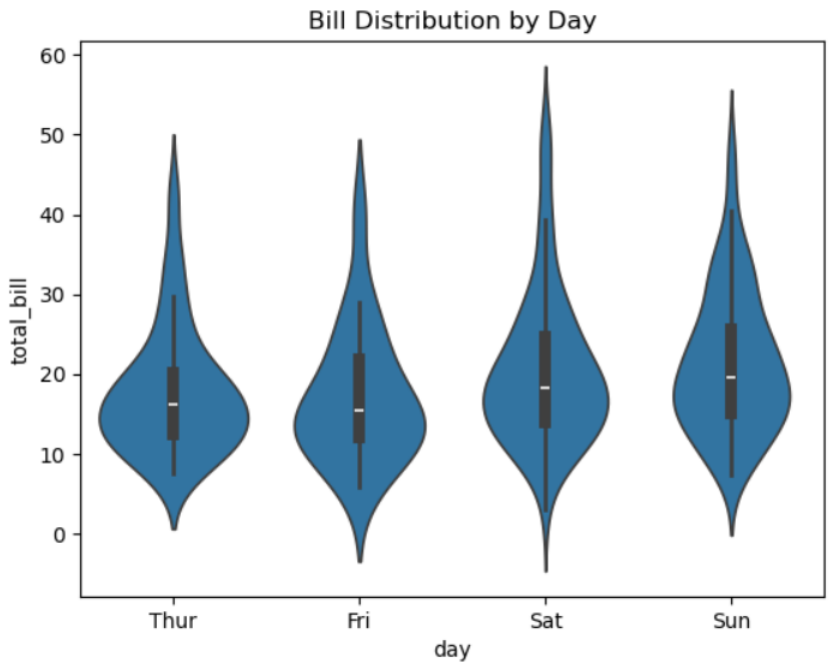
**Use Cases:**

- Psychology: Response time analysis
- Sports: Player performance distribution

**Code snippet :**

```
[18]: import seaborn as sns
      import matplotlib.pyplot as plt
      sns.violinplot(data=tips, x='day', y='total_bill')
      plt.title('Bill Distribution by Day')
      plt.show()
```

**Output :**




**Description :**

- Four violin shapes (Thursday-Sunday)
- Thick sections: High density of bills
- Thin tails: Rare bill amounts
- Sunday shows bimodal distribution

# Comparison Between Matplotlib and Seaborn

**1. Ease of Use**

- **Matplotlib**: Demands explicit configuration of plot elements (axes, legends). More verbose syntax for complex visualizations.
- **Seaborn**: Abstracts complexity with high-level functions. Automatic handling of categorical variables via hue/style parameters.

**Key Distinction**: Seaborn's displot() generates distribution plots with automatic bin selection, while Matplotlib requires manual histogram calculations.

**2. Customization Depth**

- **Matplotlib**: Enables pixel-level control (e.g., custom tick formatters, artist-level modifications).
- **Seaborn**: Limited to statistical visualization conventions but offers beautiful default styles.

**Key Distinction**: Only Matplotlib supports creating entirely new plot types (e.g., custom polar charts or CAD-like diagrams).

**3. Statistical Functionality**

- **Matplotlib**: Requires manual implementation of statistical features (error bars, confidence intervals).
- **Seaborn**: Built-in statistical computations (regression lines, distribution fitting).

**Key Distinction**: Seaborn's pairplot() automatically visualizes high-dimensional correlations - equivalent functionality in Matplotlib would require 50+ lines of code.

**4. Data Structure Handling**

- **Matplotlib**: Agnostic to data format (works with lists/arrays).
- **Seaborn**: Optimized for Pandas DataFrames with automatic metadata utilization.

**Key Distinction**: Seaborn's FacetGrid enables automated small multiples creation based on DataFrame columns, a feature absent in core Matplotlib.

**5. Performance Characteristics**

- **Matplotlib**: Closer to metal (better for >1M datapoints).
- **Seaborn**: Adds statistical computation overhead but smarter defaults for medium data.

**Key Distinction**: Matplotlib's hexbin() handles extreme-scale density plots better than Seaborn's kdeplot().