VISUALIZATION

Visualization is a way of representing information. Data visualization refers to the graphical or pictorial representation of data for better understanding and interpretation. It helps in identifying patterns, trends, and relationships within the data easily.

In Python, data visualization can be performed using several powerful libraries such as Matplotlib, Seaborn, and Plotly etc….

These libraries allow users to create different types of charts and graphs to make data more meaningful and visually appealing.

Let us explore Matplotlib and Pandas in detail.

MATPLOTLIB

Matplotlib is one of the most popular data visualization libraries in Python, created by John Hunter in 2002. It is mainly used for creating 2D plots and charts, such as line graphs, bar charts, histograms, and scatter plots.

It provides great control over every element of a figure — from axes and labels to colors and styles — making it ideal for detailed and customizable visualizations.

1.Figure

2.Axes

3.Axis

4.Artists

Figure

The Figure acts as the main container that holds all the plots or graphical elements. It can contain one or multiple plots within it.

Axes

Axes represent the area where the actual data is plotted. They are added inside the figure and can have two or three coordinate axes. Functions such as set_xlabel() and set_ylabel() are used to label the X and Y axes.

Axis

The Axis handles the range, scale, and ticks of the plot. It is responsible for setting the limits and intervals on the axes.

Artists

Artists are the visible elements that make up the figure, such as lines, texts, labels, and shapes. Everything you see on a plot is an artist in Matplotlib.

PYPLOT

Pyplot is a module within the Matplotlib library that provides a simple interface for creating various types of visualizations. It includes functions to generate a wide range of plots such as bar charts, scatter plots, pie charts, histograms, and area plots.

We can access Pyplot by importing it from the Matplotlib package as shown below:

```
import matplotlib.pyplot as plt
```

Additionally, NumPy is often imported alongside Matplotlib to handle numerical data efficiently and to support data operations for plot.

```
import numpy as np
```

Some of the plots in matplotlib:

1.Line Plot

A line plot displays information as a series of data points connected by straight lines.

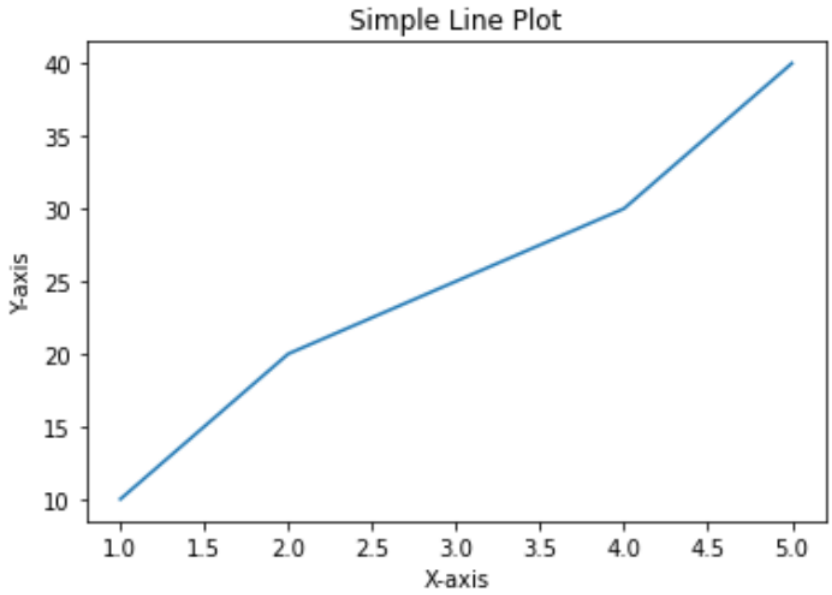
Use Case:

Used to show trends or changes over time.

Code snippet:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]

plt.plot(x, y)
plt.title("Simple Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

Output:



Description:

-To represent changes or trends over time (like sales growth or temperature variation).

-To compare multiple datasets on the same graph.

-To visualize continuous data clearly and effectively.

-We have used ,the plot() function draws a line connecting the data points in x and y. The title(), xlabel(), and ylabel() functions add a title and axis labels to make the plot more informative.

2.Scatter Plot:

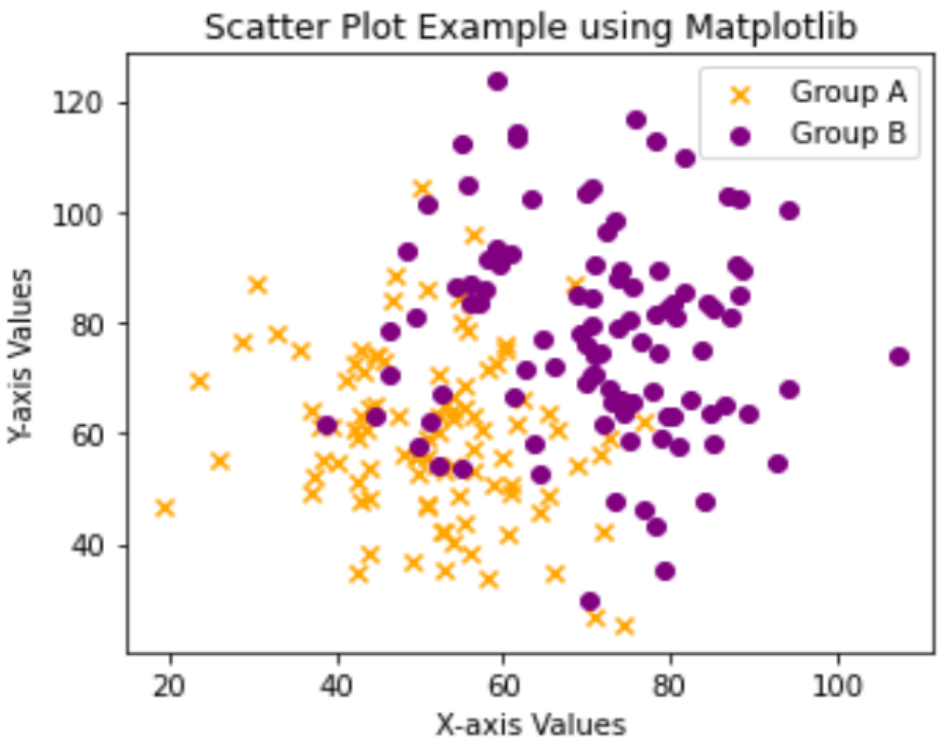
A Scatter Plot visualizes data points for two variables using dots on a two-dimensional graph. Each dot represents a pair of values — one on the X-axis and one on the Y-axis.

It is used to identify relationships, patterns, or correlations between two variables, such as height vs. weight, or temperature vs. sales

Code snippet:

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

x1 = np.random.normal(50, 10, 100)
y1 = np.random.normal(60, 15, 100)

x2 = np.random.normal(70, 12, 100)
y2 = np.random.normal(80, 20, 100)

plt.figure(figsize=(5, 4))

plt.scatter(x1, y1, color='orange', marker='x', label='Group A')
plt.scatter(x2, y2, color='purple', marker='o', label='Group B')

plt.title("Scatter Plot Example using Matplotlib")
plt.xlabel("X-axis Values")
plt.ylabel("Y-axis Values")
plt.legend()

plt.show()
```

Output:



Description:

np.random.normal() generates random numbers following a normal (bell-shaped) distribution.

Two different groups (Group A and Group B) are plotted in different colors and markers.

plt.legend() adds a small label box to identify each group.

figsize=(5,4) controls the size of the figure.

3.Bar Chart

A Bar Chart (or Bar Graph) is a type of graph that represents categorical data with rectangular bars.

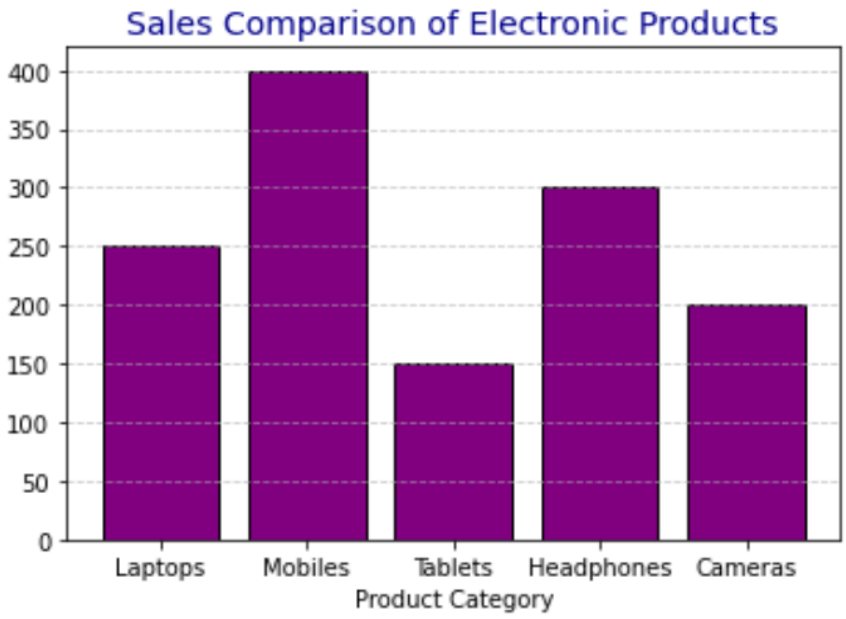
Each bar's height (or length) corresponds to the value of the category it represents.

It's mainly used to compare values across different categories or groups.

Code snippet:

```
import matplotlib.pyplot as plt

products = ['Laptops', 'Mobiles', 'Tablets', 'Headphones', 'Cameras']
sales = [250, 400, 150, 300, 200]

plt.bar(products, sales, color='purple', edgecolor='black')

plt.title("Sales Comparison of Electronic Products", fontsize=14, color='darkblue')
plt.xlabel("Product Category")
plt.ylabel("Units Sold")

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()
```

Output:



Description:

The x-axis shows the product categories.

The y-axis shows the number of units sold.

Taller bars indicate higher sales.

You can visually compare which product performed best (here, Mobiles).

4.Histogram

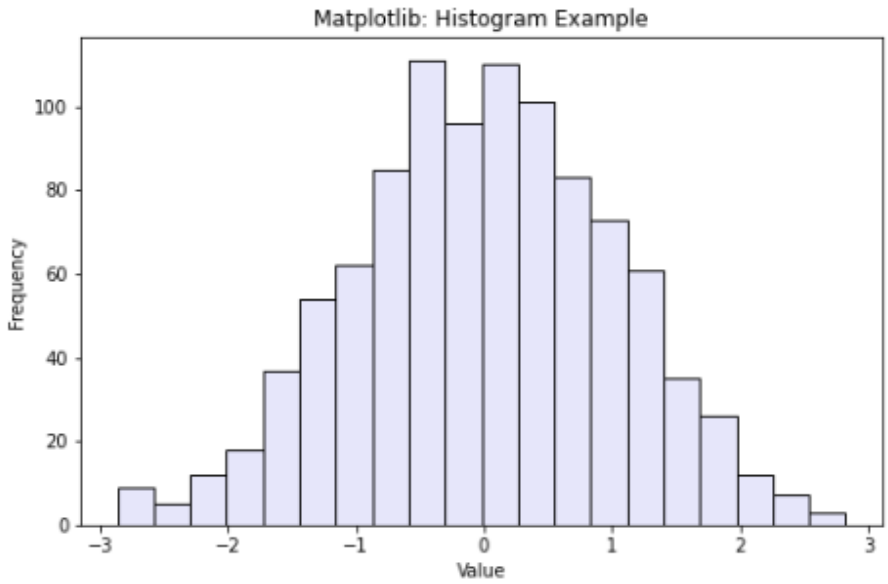
Shows the distribution of a dataset by grouping values into bins.

Used to understand the frequency distribution of numerical data.

Code snippet:

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

data = np.random.randn(1000)

plt.figure(figsize=(8,5))
plt.hist(data, bins=20, color='lavender', edgecolor='black')

plt.title('Matplotlib: Histogram Example')
plt.xlabel('Value')
plt.ylabel('Frequency')

plt.show()
```

Output:



Description:

It divides the data into bins (intervals) and displays how many values fall into each range.

The height of each bar represents the frequency of data within that interval.

In this graph, the histogram reveals the shape, spread, and center of the data — helping identify patterns such as normal distribution, skewness, or outliers.

Pie chart

A pie chart is a circular graph divided into slices to illustrate numerical proportions.

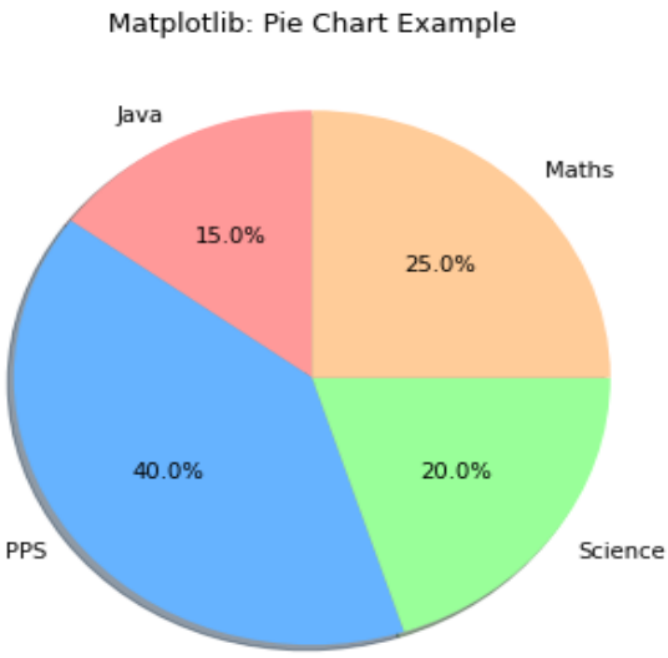
Each slice represents a category's contribution to the whole.

The total value of all slices equals 100%

code snippet:

```
import matplotlib.pyplot as plt

categories = ['Java', 'PPS', 'Science', 'Maths']
values = [15, 40, 20, 25]

plt.figure(figsize=(6,6))
plt.pie(values,
        labels=categories,
        autopct='%1.1f%%',
        startangle=90,
        colors=['#ff9999','#66b3ff','#99ff99','#ffcc99'],  shadow=True)

plt.title('Matplotlib: Pie Chart Example')

plt.show()
```

Output:



Description:

Here in the code, time is the x variable and labels are defined using labels keyword.

wedgeprops attribute is used to define the linewidth and the color to separate the elements in the pie chart.

color array contains the colors to each element.

Area chart

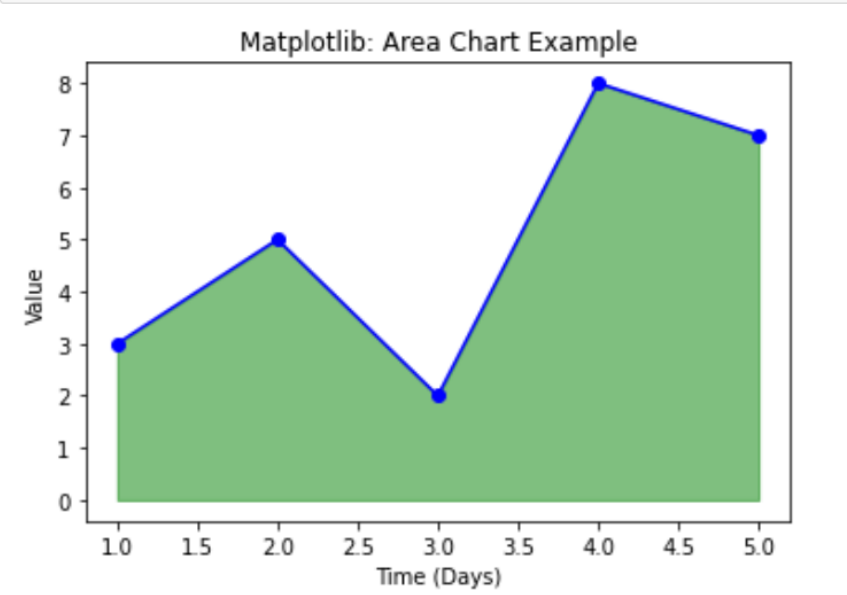
An Area Chart is similar to a Line Chart, but the area under the line is filled with color.

It shows quantitative data over time and helps visualize trends and magnitude.

Code snippet:

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

x = np.arange(1, 6)
y = [3, 5, 2, 8, 7]

plt.fill_between(x, y, color='green', alpha=0.5)
plt.plot(x, y, color='blue', marker='o')  # outline line

plt.title('Matplotlib: Area Chart Example')
plt.xlabel('Time (Days)')
plt.ylabel('Value')

plt.show()
```

Output:



Description:

A blue area under a curved line showing how the data changes over 5 time points.

The filled region helps highlight total volume or magnitude beneath the line.

Smooth and shaded — easy to interpret trend and intensity together.

PANDAS

Pandas is an open-source data manipulation and analysis library in Python.

It provides fast, flexible, and expressive data structures designed to make working with structured data (like tables, spreadsheets, or databases) easy and efficient.

Pandas is built on top of NumPy and integrates well with other libraries such as Matplotlib, Seaborn, and Scikit-learn.

Let us see the different plots in each category.

- Line Plot – to visualize trends over time.
- Bar Chart / Barh Chart – for comparing categorical data.
- Histogram – to show data distribution,
- Pie Chart – to represent proportions of a whole.
- Box Plot – to identify spread and outliers.
- Area Plot – to emphasize cumulative values over a range.
- Scatter Plot – to examine relationships between variables.
- KDE (Density) Plot – to visualize probability distributions.

Unique Features of Pandas

- Simplified data handling (loading, cleaning, filtering, grouping)
- Built-in plotting functions (powered by Matplotlib)
- Can visualize data directly from a DataFrame or Series
- Supports multiple chart types — line, bar, pie, histogram, boxplot, etc.
- Ideal for Exploratory Data Analysis (EDA)

Here are some sample codes for some of the graphs:

1.Line graph

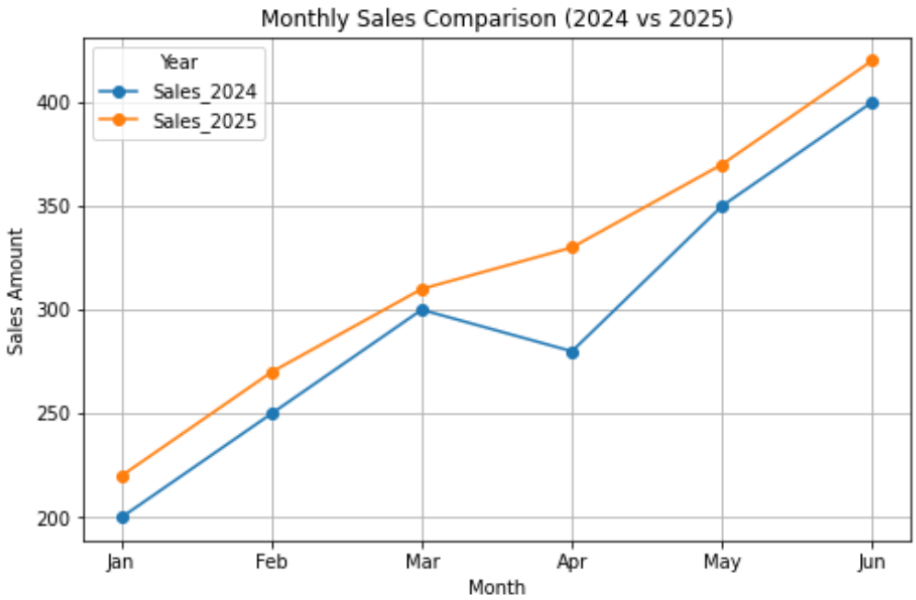
A Line Graph (or Line Plot) is a type of chart used to display information as a series of data points connected by straight lines.

It shows trends or changes over time — such as growth, decline, or fluctuations in data.

Code snippet:

```
import pandas as pd
import matplotlib.pyplot as plt
data = {
    'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'],
    'Sales_2024': [200, 250, 300, 280, 350, 400],
    'Sales_2025': [220, 270, 310, 330, 370, 420]
}

df = pd.DataFrame(data)

df.set_index('Month', inplace=True)

df.plot(kind='line', marker='o', linestyle='-', figsize=(8, 5))

plt.title("Monthly Sales Comparison (2024 vs 2025)")
plt.xlabel("Month")
plt.ylabel("Sales Amount")
plt.grid(True)
plt.legend(title="Year")
plt.show()
```

Output:



Description:

This code uses pandas and matplotlib to create a line graph showing monthly sales for 2024 and 2025. The data is stored in a DataFrame, with months on the x-axis and sales on the y-axis. The graph displays trends over time with titles, labels, and a legend for clarity.

2.Bar graph:

A bar graph (or bar chart) displays data using rectangular bars.

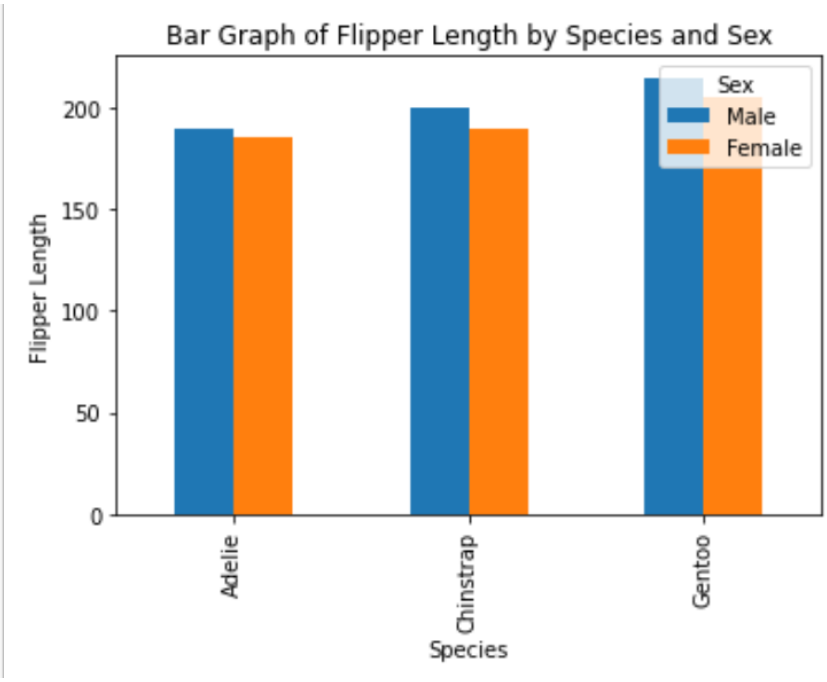
Each bar represents a category, and its height shows the value of that category.

Bar graphs are useful for comparing values across different groups or categories.

code snipper:

```
import pandas as pd
import matplotlib.pyplot as plt

data = {
    'Species': ['Adelie', 'Chinstrap', 'Gentoo'],
    'Male': [190, 200, 215],
    'Female': [185, 190, 205]
}
df = pd.DataFrame(data)
df.plot(x='Species', kind='bar', figsize=(6, 4))
plt.title("Bar Graph of Flipper Length by Species and Sex")
plt.xlabel("Species")
plt.ylabel("Flipper Length")
plt.legend(title="Sex")
plt.show()
```

Output:



Description:

This code uses Pandas and Matplotlib to create a bar graph showing male and female flipper lengths for different penguin species. The data is stored in a Data Frame and plotted

using df.plot(kind='bar') with labels and a title for clarity.

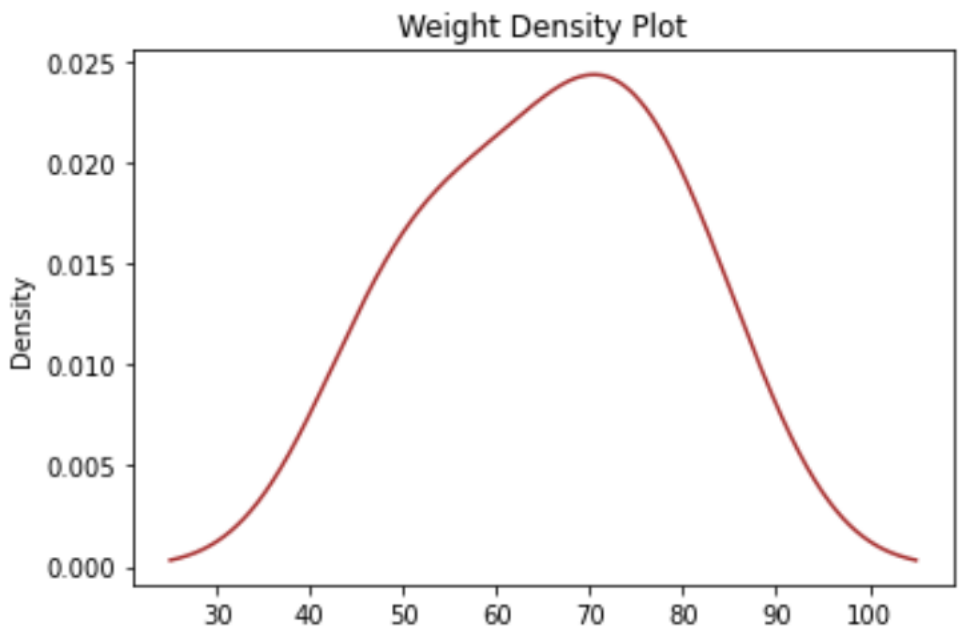
3.KDE (Density) Plot

Show smooth distribution curve (like a histogram's continuous version).

Code snipper:

```
data = {'Weight': [45, 50, 55, 60, 65, 70, 72, 75, 80, 85]}
df = pd.DataFrame(data)

df['Weight'].plot(kind='kde', color='brown', title='Weight Density Plot')
plt.show()
```

Output:



Description:

A KDE (Kernel Density Estimate) plot is a smooth, continuous curve that shows the probability density of numerical data.

It's like a smoothed version of a histogram and helps identify where values are most concentrated.

Used for understanding data distribution and variability.

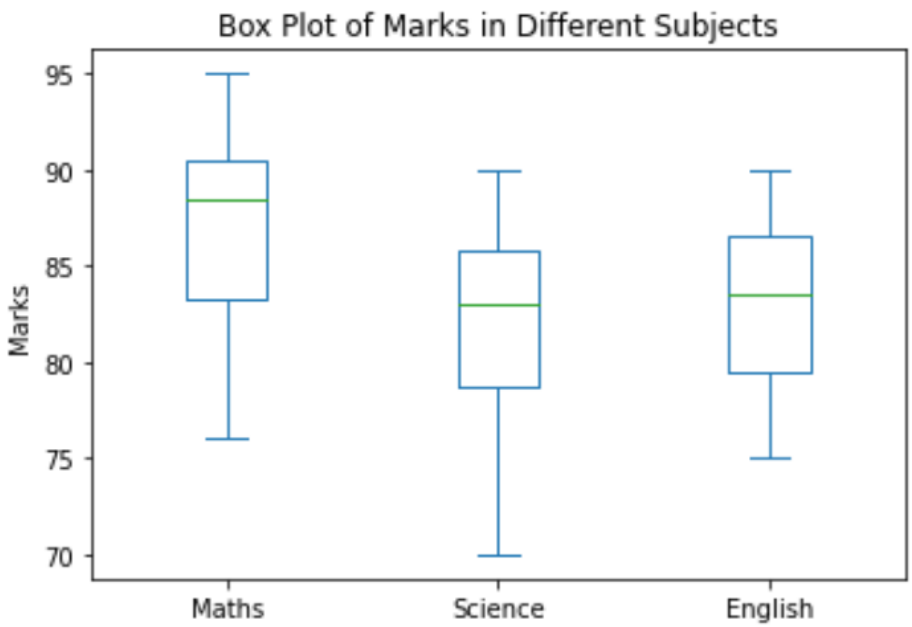
4.Box plot

Show data spread, median, and outliers.

Code snippet:

```
import pandas as pd
import matplotlib.pyplot as plt

data = {
    'Maths': [85, 90, 78, 92, 88, 76, 95, 89],
    'Science': [80, 85, 75, 90, 82, 70, 88, 84],
    'English': [78, 82, 85, 88, 80, 75, 90, 86]
}
df = pd.DataFrame(data)
df.plot(kind='box', figsize=(6,4))
plt.title("Box Plot of Marks in Different Subjects")
plt.ylabel("Marks")
plt.show()
```

Output:



Description:

A box plot (or box-and-whisker plot) shows the spread and skewness of a dataset.

It displays the median, quartiles, and outliers, making it easy to compare the variability of different data groups.

Used in statistical analysis to detect outliers or data imbalance.

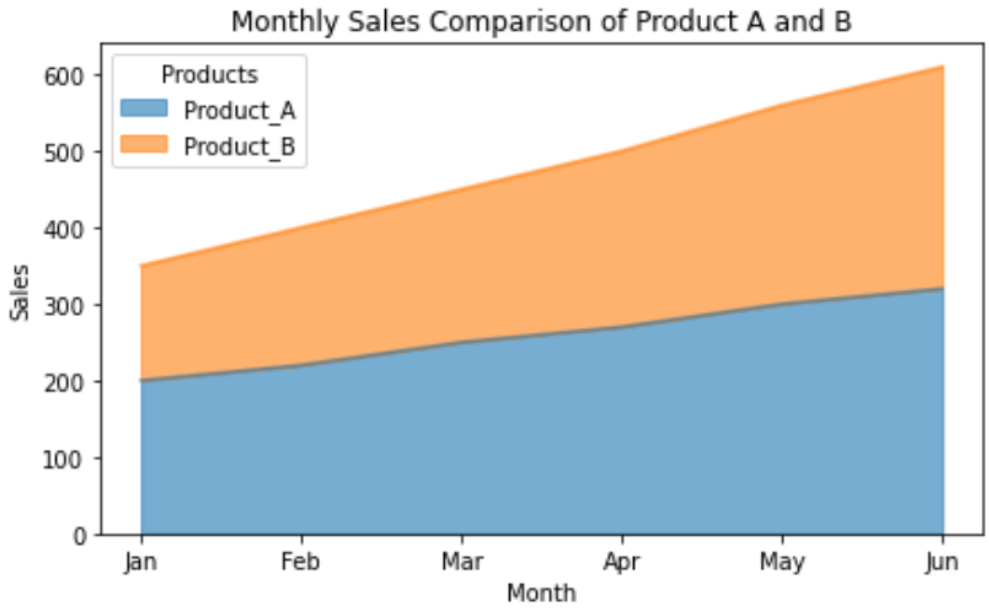
5.Area plot

Highlight total magnitude over a range (cumulative trends).

Code snippet:

```
import pandas as pd
import matplotlib.pyplot as plt

data = {
    'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'],
    'Product_A': [200, 220, 250, 270, 300, 320],
    'Product_B': [150, 180, 200, 230, 260, 290]
}
df = pd.DataFrame(data)
df.set_index('Month', inplace=True)
df.plot(kind='area', alpha=0.6, figsize=(7, 4))
plt.title("Monthly Sales Comparison of Product A and B")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.legend(title="Products")
plt.show()
```

Output:



Description:

An area plot fills the space below a line plot with color to emphasize the magnitude of change.

It shows how quantities vary over time and is helpful for displaying cumulative totals or growth trends.

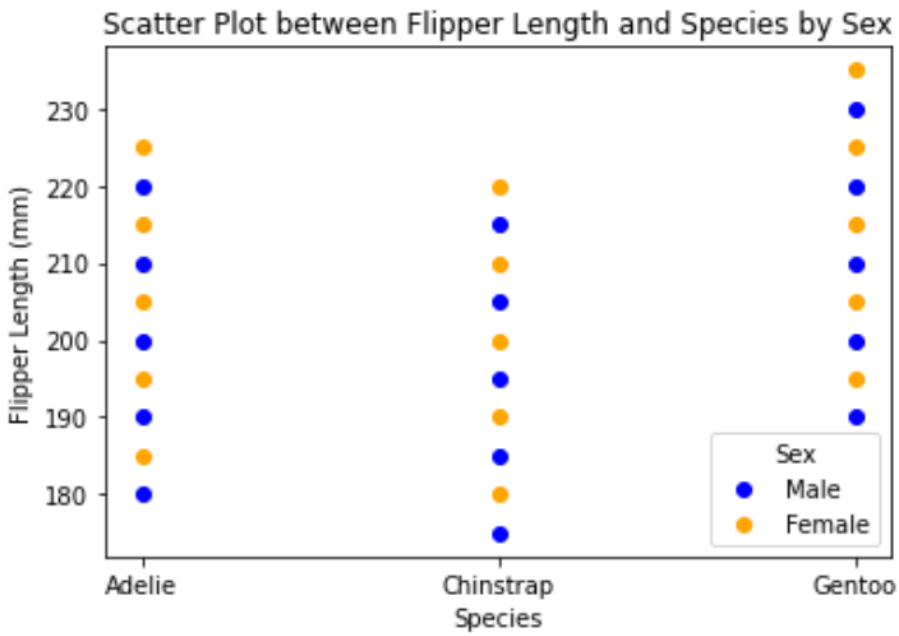
6.Scatter Plot

Show relationship between two numeric variables.

Code snippet:

```
import pandas as pd
import matplotlib.pyplot as plt
data = {
    'species': ['Adelie']*10 + ['Chinstrap']*10 + ['Gentoo']*10,
    'flipper_length_mm': [180, 185, 190, 195, 200, 205, 210, 215, 220, 225,
                          175, 180, 185, 190, 195, 200, 205, 210, 215, 220,
                          190, 195, 200, 205, 210, 215, 220, 225, 230, 235],
    'sex': ['Male', 'Female'] * 15
}
df = pd.DataFrame(data)
colors = {'Male': 'blue', 'Female': 'orange'}
for gender, color in colors.items():
    subset = df[df['sex'] == gender]
    plt.scatter(subset['species'], subset['flipper_length_mm'], label=gender, color=color)
plt.title("Scatter Plot between Flipper Length and Species by Sex")
plt.xlabel("Species")
plt.ylabel("Flipper Length (mm)")
plt.legend(title="Sex")

plt.show()
```

Output:



Description:

This code uses Pandas and Matplotlib to create a scatter plot comparing penguin species and their flipper lengths for male and female groups, using color to distinguish them and adding labels and a legend for clarity.

7.Heatmap

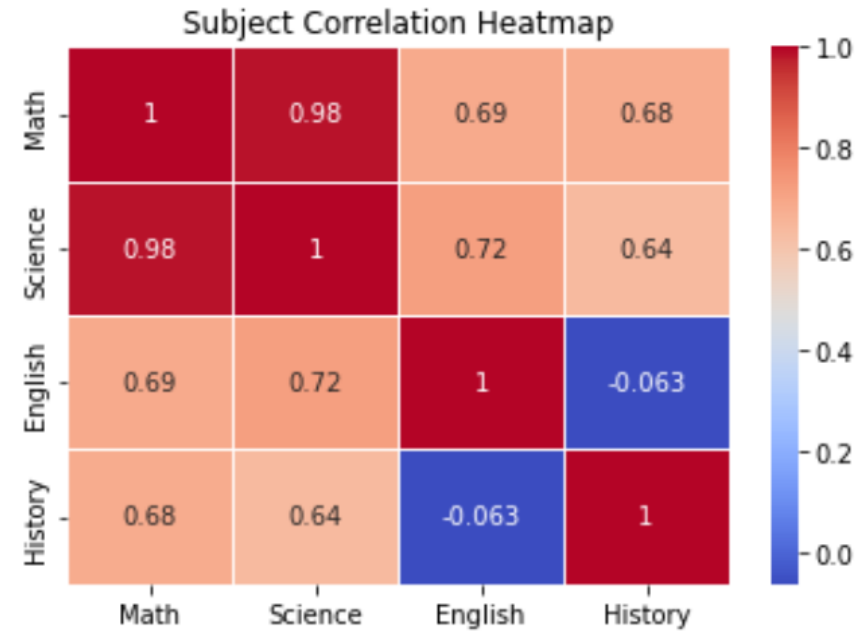
A Heatmap is a data visualization technique that uses colors to represent numerical values in a matrix or 2D data table.

It helps quickly identify patterns, correlations, and intensity within datasets.

Code snippet:

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

data = {
    'Math': [88, 92, 80, 89, 100],
    'Science': [85, 90, 78, 88, 95],
    'English': [78, 85, 82, 87, 90],
    'History': [90, 88, 84, 86, 89]
}

df = pd.DataFrame(data, index=['Student1', 'Student2', 'Student3', 'Student4', 'Student5'])
corr_matrix = df.corr()
plt.figure(figsize=(6, 4))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Subject Correlation Heatmap')
plt.show()
```

Output:



Description:

-A colorful grid showing correlation values between each pair of subjects.

-Diagonal values (1.0) represent perfect self-correlation.

-Warm colors (red/orange) show strong positive correlations,

-while cool colors (blue) show negative or weaker correlations.

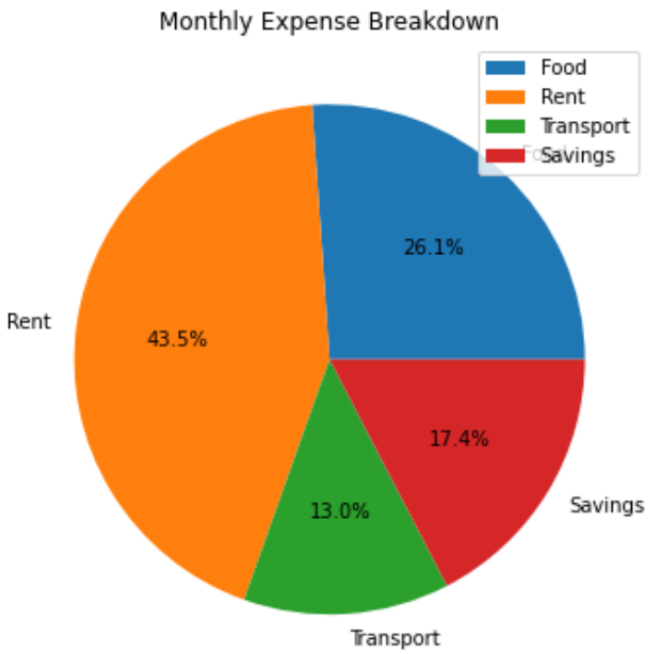
Pie chart

Display proportions or percentages of a whole.

code snippet:

```
import pandas as pd
import matplotlib.pyplot as plt
data = {'Category': ['Food', 'Rent', 'Transport', 'Savings'],
        'Amount': [300, 500, 150, 200]}
df = pd.DataFrame(data)

df.set_index('Category').plot(kind='pie', y='Amount', autopct='%1.1f%%', figsize=(6,6),
                              title='Monthly Expense Breakdown')
plt.ylabel('')
plt.show()
```

Output:



Description:

-A pie chart represents parts of a whole using slices of a circle.

-Each slice's size corresponds to its percentage contribution to the total.

-It's ideal for showing proportional data, such as budget allocation or market share.

Comparission of matplotlib and pandas

Matplotlib:

- Core Library: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It provides a low-level interface for creating plots with fine-grained control over every aspect of the figure.
- Flexibility: Matplotlib allows users to create any type of plot imaginable, from basic line plots and scatter plots to complex 3D plots and geographical maps.
- Mature: Matplotlib has been around for a long time and is a well-established library in the Python ecosystem. Many other libraries, including Seaborn, are built on top of Matplotlib.
- Customization: Matplotlib offers extensive customization options, allowing users to customize every aspect of a plot, including colors, line styles, markers, fonts, annotations, and more.
- Integration: Matplotlib can be easily integrated with other libraries and frameworks, such as NumPy, Pandas, SciPy, which makes it a versatile choice for data visualization in various contexts.

Advantages of Matplotlib:

- High degree of customization and control over plots.
- Wide range of plot types and styles.
- Strong community support and extensive documentation.
- Well-suited for creating publication-quality figures and graphics.

Pandas:

- Core Library: Pandas is a powerful data analysis and manipulation library in Python that also includes built-in data visualization features. It provides a high-level interface for creating quick and simple plots directly from DataFrame and Series objects without requiring extensive code.
- Ease of Use: Pandas makes data visualization extremely easy and beginner-friendly. With a single command using the .plot() function, users can generate a variety of charts such as line plots, bar charts, histograms, pie charts, and more. It is ideal for Exploratory Data Analysis (EDA) and quick insights.
    - Integration:

    Pandas integrates seamlessly with Matplotlib, as its plotting functions are built on top of it. It also works efficiently with NumPy and Seaborn, allowing smooth data manipulation and visualization workflows within the same environment.

    - Data Handling: Unlike other visualization libraries, Pandas can directly plot from structured datasets (DataFrames). It handles missing data, indexing, grouping, and aggregation effortlessly, making it perfect for real-world data exploration before

advanced visualization.

- Customization: While Pandas offers fewer customization options than Matplotlib, it supports basic styling features like color, labels, titles, gridlines, and figure size. For advanced customization, users can access the underlying Matplotlib axes to modify plot details further.
- Performance:

Pandas is efficient for small to medium-sized datasets and provides instant visual feedback. However, for extremely large datasets or complex visuals, direct Matplotlib or other specialized libraries (like Plotly) may perform better.

Advantages of pandas:

- Pandas allows users to create visualizations with just one line of code using the .plot() function.
- Ideal for beginners and for quick Exploratory Data Analysis (EDA)
- Great for quickly checking trends, distributions, and relationships in data before performing detailed analysis.
- Helps analysts gain insights fast without complex coding.

Overall the comparison between Matplotlib and Pandas are both popular libraries used for data visualization in Python, but they serve slightly different purposes and audiences.

Matplotlib is a low-level, highly customizable library designed for creating detailed and professional plots, while Pandas provides a high-level, easy-to-use interface for quick and simple data visualizations directly from DataFrames.