

## Introduction to Data Visualization

### 1. Importance of Data Visualization in Data Analysis (10 minutes)

Concept: Data visualization is crucial because it allows for the clear communication of complex data, making it easier to understand trends, patterns, and outliers.

Real-life Example: In the business world, sales trends over time can be visualized using line plots to identify seasonal peaks and troughs. In healthcare, patient data can be visualized to track disease outbreaks.

Use in Industries: Business: Identifying sales trends, customer behavior analysis, financial forecasting. Healthcare: Tracking patient data, visualizing medical research results. Marketing: Analyzing campaign performance, market research insights. Engineering: Monitoring system performance, quality control.

### 2. Overview of Different Types of Plots and Their Uses

Line Plot: Used to show data trends over time. Bar Plot: Used to compare different groups. Histogram: Used to show the distribution of a dataset. Scatter Plot: Used to show the relationship between two variables. Box Plot: Used to show the distribution of a dataset through its quartiles. Pie Chart: Used to show the proportions of a whole.

## Introduction to Matplotlib

### 1. What is Matplotlib?

Concept: Matplotlib is a plotting library for the Python programming language. It provides an object-oriented API for embedding plots into applications.

Real-life Example: Matplotlib is used to create publication-quality plots in scientific research and engineering.

```
pip install matplotlib
```

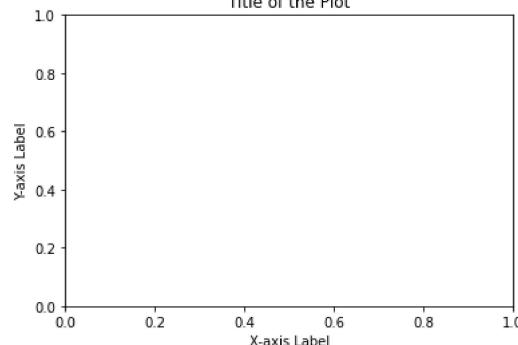
```
Requirement already satisfied: matplotlib in c:\users\hp\anaconda3\lib\site-packages (3.5.1)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (3.0.4)
Requirement already satisfied: packaging>=20.0 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (21.3)
Requirement already satisfied: cycler>=0.10 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: numpy>=1.17 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (1.22.4)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (9.0.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\hp\anaconda3\lib\site-packages (from matplotlib) (1.3.2)
Requirement already satisfied: six>=1.5 in c:\users\hp\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

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

```
#creating a plot
plt.plot(x, y)
```

```
#adding titles and labels
plt.title('Title of the Plot')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
```

```
Text(0, 0.5, 'Y-axis Label')
```



```
#displaying the plot
plt.show()
```

**Line Styles** You can specify the style of the line using the linestyle parameter. Common line styles include:

'-': Solid line (default) '--': Dashed line '-.': Dash-dot line ':': Dotted line

**Markers** Markers are symbols used to mark data points on the plot. You can specify the marker style using the marker parameter. Some common markers include:

'.': Point ';' : Pixel 'o': Circle 'v': Triangle down '^': Triangle up '<': Triangle left '>': Triangle right '1': Tri-down '2': Tri-up '3': Tri-left '4': Tri-right 's': Square 'p': Pentagon '\*': Star 'h': Hexagon1 'H': Hexagon2 '+': Plus 'x': X 'D': Diamond 'd': Thin diamond 'l': Vertical line '\_': Horizontal line

## 1. Line Plot

### Features:

Displays data points connected by straight lines. Shows trends over time or continuous data. Can plot one or multiple lines for different datasets.

### use Cases:

Tracking changes over intervals of time (e.g., stock prices, weather data). Comparing multiple datasets. Visualizing data trends and patterns. Data Type: Numerical data on both axes.

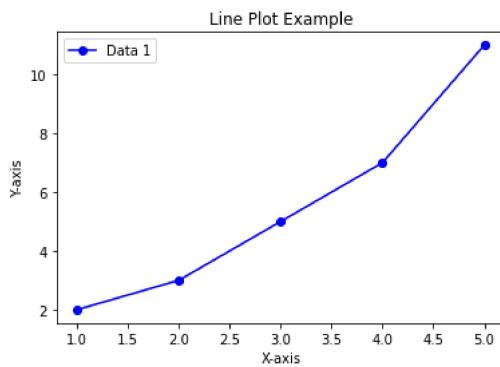
```
import matplotlib.pyplot as plt

# Example data
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

# Create a line plot
plt.plot(x, y, marker='o', linestyle='-', color='b', label='Data 1')

# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot Example')
plt.legend()

# Show the plot
plt.show()
```



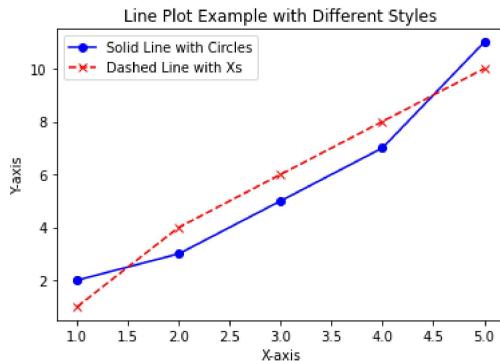
```
import matplotlib.pyplot as plt

# Example data
x = [1, 2, 3, 4, 5]
y1 = [2, 3, 5, 7, 11]
y2 = [1, 4, 6, 8, 10]

# Create line plots with different styles and markers
plt.plot(x, y1, marker='o', linestyle='-', color='b', label='Solid Line with Circles')
plt.plot(x, y2, marker='x', linestyle='--', color='r', label='Dashed Line with Xs')
```

```
# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot Example with Different Styles')
plt.legend()

# Show the plot
plt.show()
```



```
# same X values but different y values
import matplotlib.pyplot as plt

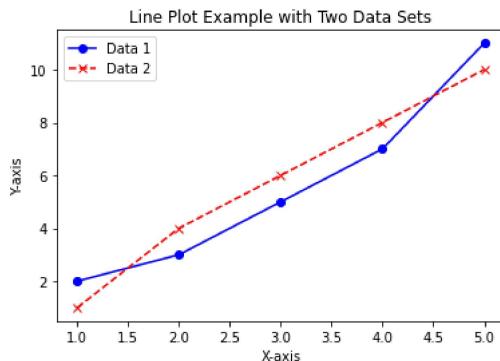
# Example data
x1 = [1, 2, 3, 4, 5]
y1 = [2, 3, 5, 7, 11]

x2 = [1, 2, 3, 4, 5]
y2 = [1, 4, 6, 8, 10]

# Create line plots for both sets of data
plt.plot(x1, y1, marker='o', linestyle='-', color='b', label='Data 1')
plt.plot(x2, y2, marker='x', linestyle='--', color='r', label='Data 2')

# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot Example with Two Data Sets')
plt.legend()

# Show the plot
plt.show()
```



```
# Different X values different y values
import matplotlib.pyplot as plt

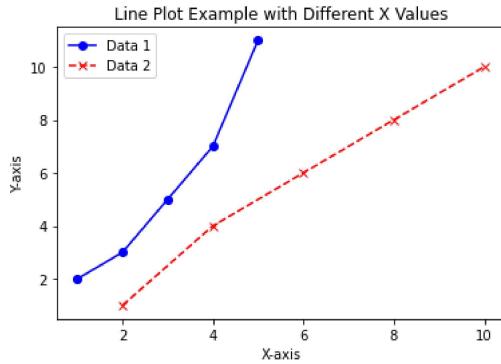
# Example data
x1 = [1, 2, 3, 4, 5]
y1 = [2, 3, 5, 7, 11]

x2 = [2, 4, 6, 8, 10]
y2 = [1, 4, 6, 8, 10]
```

```
# Create line plots for both sets of data
plt.plot(x1, y1, marker='o', linestyle='-', color='b', label='Data 1')
plt.plot(x2, y2, marker='x', linestyle='--', color='r', label='Data 2')

# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot Example with Different X Values')
plt.legend()

# Show the plot
plt.show()
```



2. Bar Plot Features: Displays data using rectangular bars. Bars can be vertical or horizontal. Length of bars represents the value of the data.

Use Cases: Comparing different categories or groups. Showing frequency or count of items.

Data Type: Categorical data on one axis and numerical data on the other.

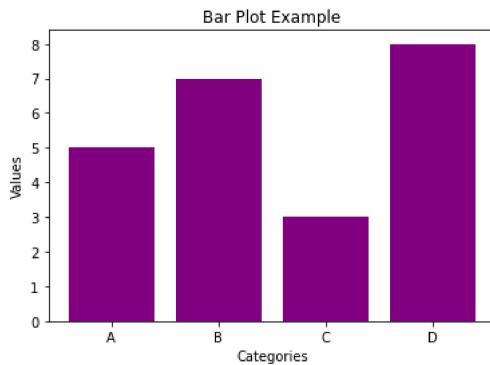
```
import matplotlib.pyplot as plt

# Example data
categories = ['A', 'B', 'C', 'D']
values = [5, 7, 3, 8]

# Create a vertical bar plot
plt.bar(categories, values, color='purple')

# Adding labels and title
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Plot Example')

# Show the plot
plt.show()
```



3. Histogram Features: Displays the distribution of a dataset. Uses bins to group data into ranges. Height of each bin represents the frequency of data points within that range.

Use Cases: Understanding the distribution of a continuous variable. Identifying patterns, such as normal distribution or skewness.

Data Type: Numerical data.

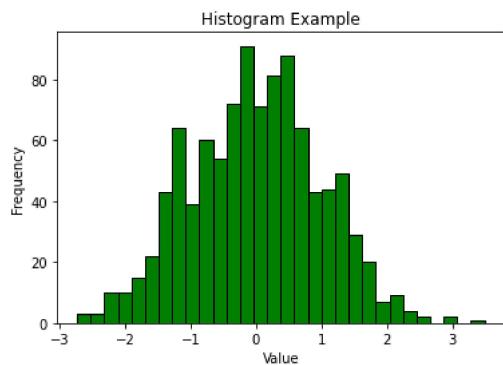
```
import matplotlib.pyplot as plt
import numpy as np

# Example data
data = np.random.randn(1000)

# Create a histogram
plt.hist(data, bins=30, color='green', edgecolor='black')

# Adding labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram Example')

# Show the plot
plt.show()
```



4. Scatter Plot Features: Displays data points as individual markers. Shows relationship between two variables. Can use marker size and color to add more dimensions.

Use Cases: Visualizing correlations between variables. Identifying outliers.

Data Type: Numerical data on both axes.

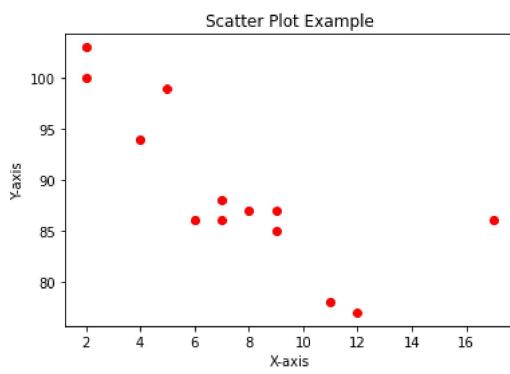
```
import matplotlib.pyplot as plt

# Example data
x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]
y = [99, 86, 87, 88, 100, 86, 103, 87, 94, 78, 77, 85, 86]

# Create a scatter plot
plt.scatter(x, y, color='red')

# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot Example')

# Show the plot
plt.show()
```



5. Box Plot Features: Displays the distribution of data based on a five-number summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum. Shows outliers as individual points. Visualizes the spread and skewness of the data.

Use Cases: Comparing distributions between multiple groups. Identifying outliers and variability in the data.

Data Type: Numerical data.

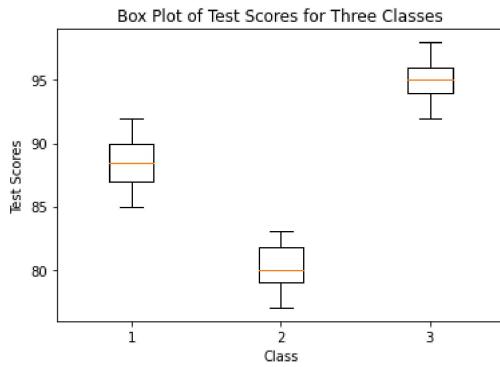
```
import matplotlib.pyplot as plt

# Example data
data = [
    [85, 87, 90, 91, 88, 85, 89, 90, 92, 87], # Class 1
    [78, 80, 82, 79, 81, 77, 83, 79, 80, 82], # Class 2
    [92, 95, 94, 96, 93, 97, 94, 96, 98, 95] # Class 3
]

# Create a basic box plot
plt.boxplot(data)

# Adding labels and title
plt.xlabel('Class')
plt.ylabel('Test Scores')
plt.title('Box Plot of Test Scores for Three Classes')

# Show the plot
plt.show()
```



Class 1: Look at the box for Class 1. The position of the median line within the box tells you the typical score for Class 1. The length of the box shows how variable the scores are. Longer boxes indicate more variability.

Class 2: Compare the box for Class 2 with Class 1. If Class 2's box is shorter, it means the scores in Class 2 are less spread out. The position of the median line will show if the typical score is higher or lower compared to Class 1.

Class 3: Do the same for Class 3. The box might be longer or shorter compared to the other classes, and the median line might be higher or lower. This tells you how the test scores of Class 3 compare with those of the other classes.

Outliers: If you see points outside the whiskers (if the plot includes them), these are outliers. They represent test scores that are unusually high or low compared to the rest of the data.

By focusing on these components, you can help your students understand the basic insights a box plot provides about the data distribution for each class.

6. Pie Chart Features: Displays data as slices of a circle. Shows proportion of each category relative to the whole.

Use Cases: Visualizing relative proportions or percentages. Comparing parts of a whole.

Data Type: Categorical data for slices and numerical data for values.

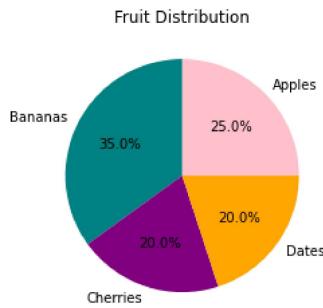
```
import matplotlib.pyplot as plt

# Example data
labels = ['Apples', 'Bananas', 'Cherries', 'Dates']
sizes = [25, 35, 20, 20]
colors = ['pink', 'teal', 'purple', 'orange']

# Create a basic pie chart
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%')

# Adding title
plt.title('Fruit Distribution')

# Show the plot
plt.show()
```



The autopct parameter in Matplotlib's plt.pie() function controls how the percentage values are displayed on the pie chart slices. It allows you to format the text that appears on the chart.

Understanding autopct Purpose: autopct is used to show the percentage of each slice of the pie chart. This helps viewers quickly understand the proportion of each category relative to the whole.

'%1.1f%%': Displays the percentage with one decimal place (e.g., 25.0%). '%1.0f%%': Displays the percentage with no decimal places (e.g., 25%). '%.2f%%': Displays the percentage with two decimal places (e.g., 25.00%).