week_7

March 19, 2025

1 Week: 7

1.1 CS2201

1.1.1 TA Solution

1.1.2 Abhisek Sarkar (as20ms091@iiserkol.ac.in)

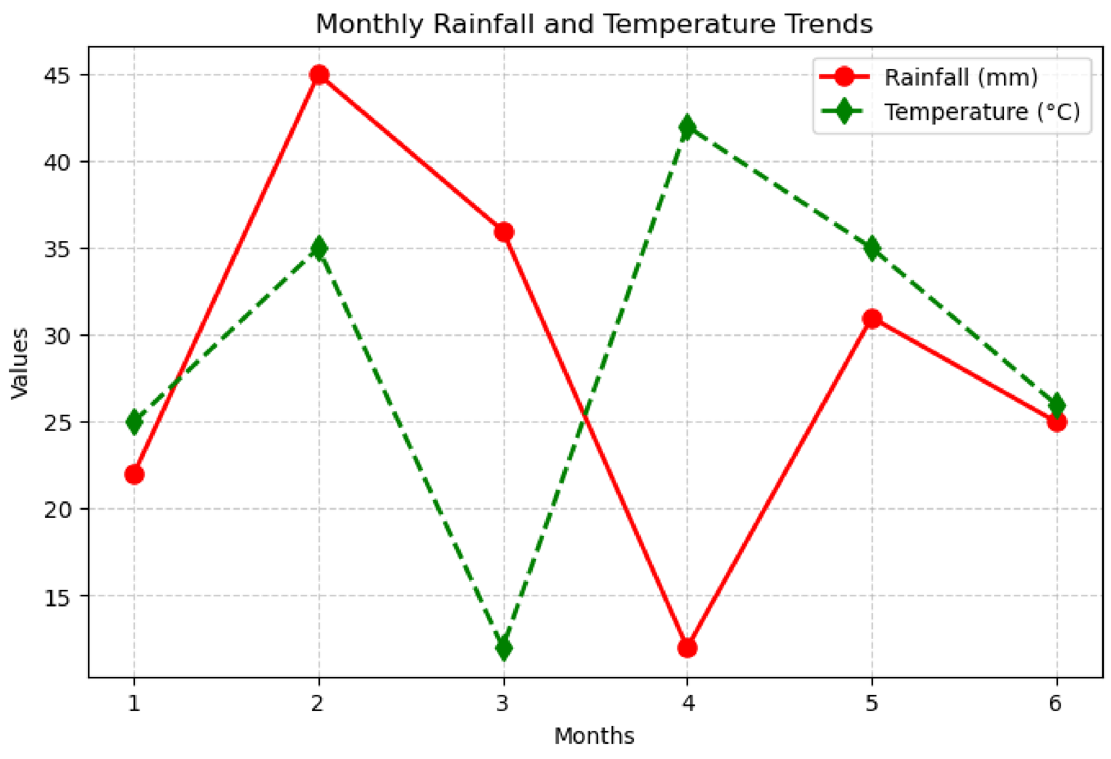
1.1.3 Q1. Take months (say 2, 3 etc.) and rainfall (in millimeters) as input from the user and plot months (x-axis) vs rainfall (y-axis) using plot in matplotlib.

1.1.4 Q2. Add another series for temperature (in degree celsius) and add it in the plot in (1) against months. Make sure you use different colors and markers for the two series (rainfall and temperature) against months.

1.1.5 Q3. Play around with line styles and colors of the two series. Also add title and legend

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

# Taking user input for months, rainfall, and temperature
months = input("Enter space-separated month names: ").split()
rainfall = list(map(float, input("Enter space-separated rainfall values (in mm):
↪ ").split()))
temperature = list(map(float, input("Enter space-separated temperature values␣
↪(in °C): ").split()))

# Converting lists to NumPy arrays
months = np.array(months)
rainfall = np.array(rainfall)
temperature = np.array(temperature)

# Creating the plot
plt.figure(figsize=(8, 5))
plt.plot(months, rainfall, 'o-r', label="Rainfall (mm)", markersize=8,␣
↪linewidth=2)
plt.plot(months, temperature, 'd--g', label="Temperature (°C)", markersize=8,␣
↪linewidth=2)
```

```
# Adding labels, title, and legend
plt.xlabel("Months")
plt.ylabel("Values")
plt.title("Monthly Rainfall and Temperature Trends")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)  # Adding a light grid for better␣
 ↪visualization

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



2 Explanation: Monthly Rainfall and Temperature Visualization

2.1 Objective:

The purpose of this program is to take user input for months, rainfall, and temperature values, and then plot these values on a graph to visualize trends over different months.

2.2 Theory & Concepts:

2.2.1 1. NumPy Library

NumPy (`numpy`) is a powerful Python library used for numerical computing. In this program, we use NumPy to store and manipulate numerical data efficiently.

2.2.2 2. Matplotlib Library

Matplotlib (`matplotlib.pyplot`, imported as `plt`) is a widely used data visualization library in Python. It allows us to create line charts, bar charts, scatter plots, and more.

2.2.3 3. Line Plot

A **line plot** is useful for showing trends over time. In this case, we plot: - **Rainfall (in mm) over months** - **Temperature (in °C) over months**

Each dataset is represented with a different color, line style, and marker.

2.3 Step-by-Step Code Explanation

2.3.1 Step 1: Importing Necessary Libraries

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

- `numpy`: Helps in working with numerical data.
- `matplotlib.pyplot`: Used to create and customize plots.

2.3.2 Step 2: Taking User Input

```
months = input("Enter space-separated month names: ").split()
rainfall = list(map(float, input("Enter space-separated rainfall values (in mm): ").split()))
temperature = list(map(float, input("Enter space-separated temperature values (in °C): ").spli
```

- `input()` function takes user input as a string.
- `.split()` separates the input into individual elements.
- `map(float, ...)` converts the input values into floating-point numbers to allow decimal values.

Example Input:

```
Enter space-separated month names: Jan Feb Mar Apr May Jun
Enter space-separated rainfall values (in mm): 80 70 60 90 100 110
Enter space-separated temperature values (in °C): 20 22 25 30 35 38
```

This results in: - `months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']` - `rainfall = [80, 70, 60, 90, 100, 110]` - `temperature = [20, 22, 25, 30, 35, 38]`

2.3.3 Step 3: Converting Lists to NumPy Arrays

```
months = np.array(months)
rainfall = np.array(rainfall)
temperature = np.array(temperature)
```

- NumPy arrays are more efficient than Python lists for numerical operations.

2.3.4 Step 4: Creating the Plot

```
plt.figure(figsize=(8, 5))
plt.plot(months, rainfall, 'o-r', label="Rainfall (mm)", markersize=8, linewidth=2)
plt.plot(months, temperature, 'd--g', label="Temperature (°C)", markersize=8, linewidth=2)
```

- `plt.figure(figsize=(8, 5))`: Sets the figure size to 8 inches by 5 inches.
- `plt.plot(months, rainfall, 'o-r', label="Rainfall (mm)", markersize=8, linewidth=2)`:
  - `'o-r'`: **Red solid line with circular markers (o)**
  - `markersize=8`: Makes markers bigger.
  - `linewidth=2`: Makes lines thicker.
- `plt.plot(months, temperature, 'd--g', label="Temperature (°C)", markersize=8, linewidth=2)`:
  - `'d--g'`: **Green dashed line with diamond markers (d)**

2.3.5 Step 5: Adding Labels and Enhancements

```
plt.xlabel("Months")
plt.ylabel("Values")
plt.title("Monthly Rainfall and Temperature Trends")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)
```

- `plt.xlabel("Months")`: Labels the x-axis.
- `plt.ylabel("Values")`: Labels the y-axis.
- `plt.title("Monthly Rainfall and Temperature Trends")`: Adds a title to the graph.
- `plt.legend()`: Displays a legend to differentiate rainfall and temperature.
- `plt.grid(True, linestyle="--", alpha=0.6)`:
  - Adds a light dashed grid for better readability.
  - `alpha=0.6` makes it semi-transparent.

2.3.6 Step 6: Displaying the Graph

```
plt.show()
```

- This function **displays the final plot**.

2.4 Example Output

Assume the following input values:

```
Enter space-separated month names: Jan Feb Mar Apr May Jun
Enter space-separated rainfall values (in mm): 80 70 60 90 100 110
Enter space-separated temperature values (in °C): 20 22 25 30 35 38
```

The generated **graph** will have: - **Rainfall (mm)** as a **red solid line** with circular markers. - **Temperature (°C)** as a **green dashed line** with diamond markers. - A **title**, **x-axis (Months)**, **y-axis (Values)**, **legend**, and **grid**.

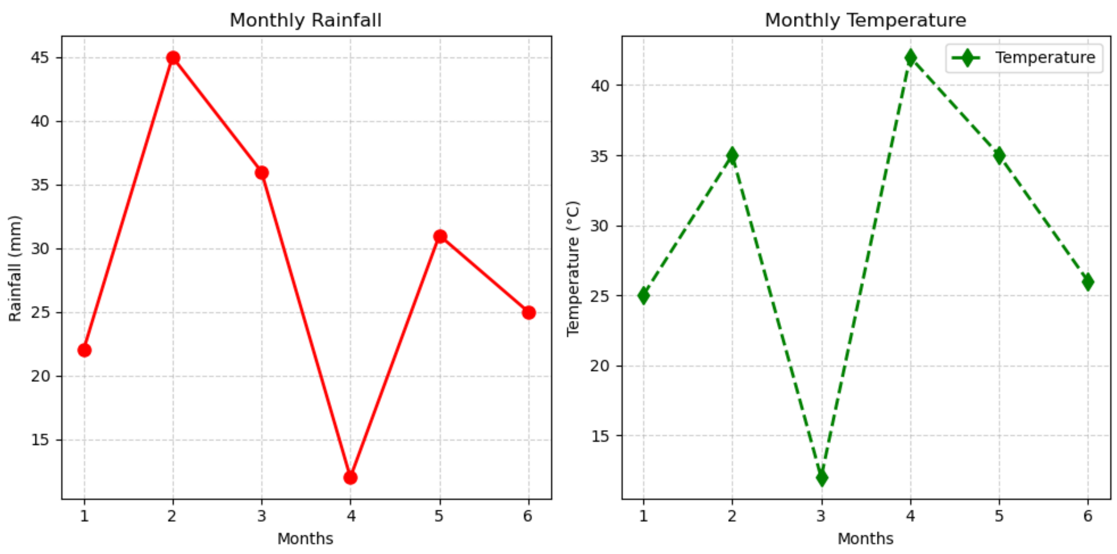
2.5 Final Summary

User inputs month names, rainfall, and temperature values.
Data is converted to NumPy arrays for efficient processing.
Matplotlib is used to plot the data with different markers and colors.
Enhancements like grid, labels, title, and legend improve readability.
Final graph displays trends for both rainfall and temperature over months.

This makes the program **efficient, visually appealing, and easy to understand!**

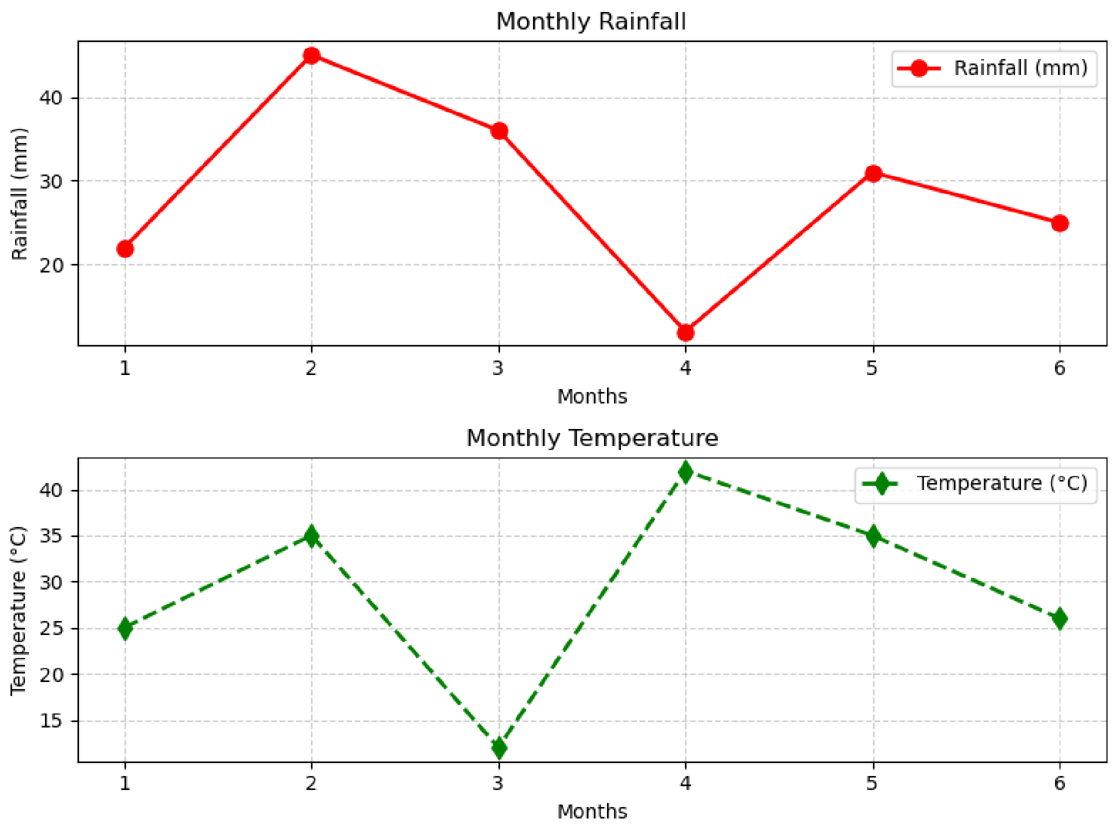
2.5.1 Q4. Show the above two plots as subplots i) along the same row and ii) along the same column. Put the title on each subplot.

[2]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Taking user input for months, rainfall, and temperature
months = input("Enter space-separated month names: ").split()
rainfall = list(map(float, input("Enter space-separated rainfall values (in mm):
↪ ").split()))
temperature = list(map(float, input("Enter space-separated temperature values␣
↪(in °C): ").split()))

# Converting lists to NumPy arrays
months = np.array(months)
rainfall = np.array(rainfall)
temperature = np.array(temperature)

# Creating subplots
plt.figure(figsize=(10, 5))

# Subplot 1: Rainfall
plt.subplot(1, 2, 1)
plt.plot(months, rainfall, 'o-r', markersize=8, linewidth=2)
plt.title("Monthly Rainfall")
plt.xlabel("Months")
plt.ylabel("Rainfall (mm)")
plt.grid(True, linestyle="--", alpha=0.6)

# Subplot 2: Temperature
plt.subplot(1, 2, 2)
plt.plot(months, temperature, 'd--g', markersize=8, linewidth=2,␣
 ↪label="Temperature")
plt.title("Monthly Temperature")
plt.xlabel("Months")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)

# Adjust layout and display the plots
plt.tight_layout()
plt.show()
```



[3]:
```
import numpy as np
import matplotlib.pyplot as plt

# Taking user input for months, rainfall, and temperature
months = input("Enter space-separated month names: ").split()
rainfall = list(map(float, input("Enter space-separated rainfall values (in mm):
 ↪ ").split()))
temperature = list(map(float, input("Enter space-separated temperature values␣
 ↪(in °C): ").split()))

# Converting lists to NumPy arrays
```

```
months = np.array(months)
rainfall = np.array(rainfall)
temperature = np.array(temperature)

# Creating subplots
plt.figure(figsize=(8, 6))

# Subplot 1: Rainfall
plt.subplot(2, 1, 1)
plt.plot(months, rainfall, 'o-r', markersize=8, linewidth=2, label="Rainfall␣
  ↪(mm)")
plt.title("Monthly Rainfall")
plt.xlabel("Months")
plt.ylabel("Rainfall (mm)")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)

# Subplot 2: Temperature
plt.subplot(2, 1, 2)
plt.plot(months, temperature, 'd--g', markersize=8, linewidth=2,␣
  ↪label="Temperature (°C)")
plt.title("Monthly Temperature")
plt.xlabel("Months")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)

# Adjust layout and display the plots
plt.tight_layout()
plt.show()
```



3 Understanding `subplot()` in Matplotlib

3.1 Objective:

The goal of this manual is to help students understand how `subplot()` works in Matplotlib to create multiple plots within a single figure.

3.2 Introduction to `subplot()`

In **Matplotlib**, the `subplot()` function allows us to **create multiple plots** in the same figure. Instead of displaying just one graph, we can divide the figure into sections and plot multiple graphs side by side (horizontally) or stacked (vertically).

This is useful when we want to **compare different datasets** within the same figure.

3.3 Basic Syntax of `subplot()`

```
plt.subplot(nrows, ncols, index)
```

Where: - `nrows` → **Number of rows** (how many plots stacked vertically) - `ncols` → **Number of columns** (how many plots side by side) - `index` → **Position of the current plot** (which subplot to work on)

3.3.1 Example Layouts:

Syntax	Figure Layout
`plt.subplot(1, 2, 1)`	1 row, 2 columns → First plot (Left)
`plt.subplot(1, 2, 2)`	1 row, 2 columns → Second plot (Right)
`plt.subplot(2, 1, 1)`	2 rows, 1 column → First plot (Top)
`plt.subplot(2, 1, 2)`	2 rows, 1 column → Second plot (Bottom)

3.4 Example 1: Two Plots Side by Side

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

x = np.array([1, 2, 3, 4, 5])
y1 = np.array([10, 20, 30, 40, 50])  # First dataset
y2 = np.array([5, 15, 25, 35, 45])   # Second dataset

plt.figure(figsize=(10, 4))

# First subplot (left)
plt.subplot(1, 2, 1)
plt.plot(x, y1, 'o-r', label="Dataset 1")
plt.title("First Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()

# Second subplot (right)
plt.subplot(1, 2, 2)
plt.plot(x, y2, 'd--g', label="Dataset 2")
plt.title("Second Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()

plt.tight_layout()  # Adjusts spacing between subplots
plt.show()
```

3.4.1 Output:

- **Two plots displayed side by side**

- Each plot has **different datasets**, colors, and markers.

3.5 Example 2: Two Stacked Plots (One Above Another)

```python
plt.figure(figsize=(6, 6))

# First subplot (Top)
plt.subplot(2, 1, 1)
plt.plot(x, y1, 'o-r', label="Dataset 1")
plt.title("Top Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)

# Second subplot (Bottom)
plt.subplot(2, 1, 2)
plt.plot(x, y2, 'd--g', label="Dataset 2")
plt.title("Bottom Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)

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

3.5.1 Output:

- **Two plots stacked vertically (one on top, one below)**
- Each plot has **labels, legends, and grid for clarity**.

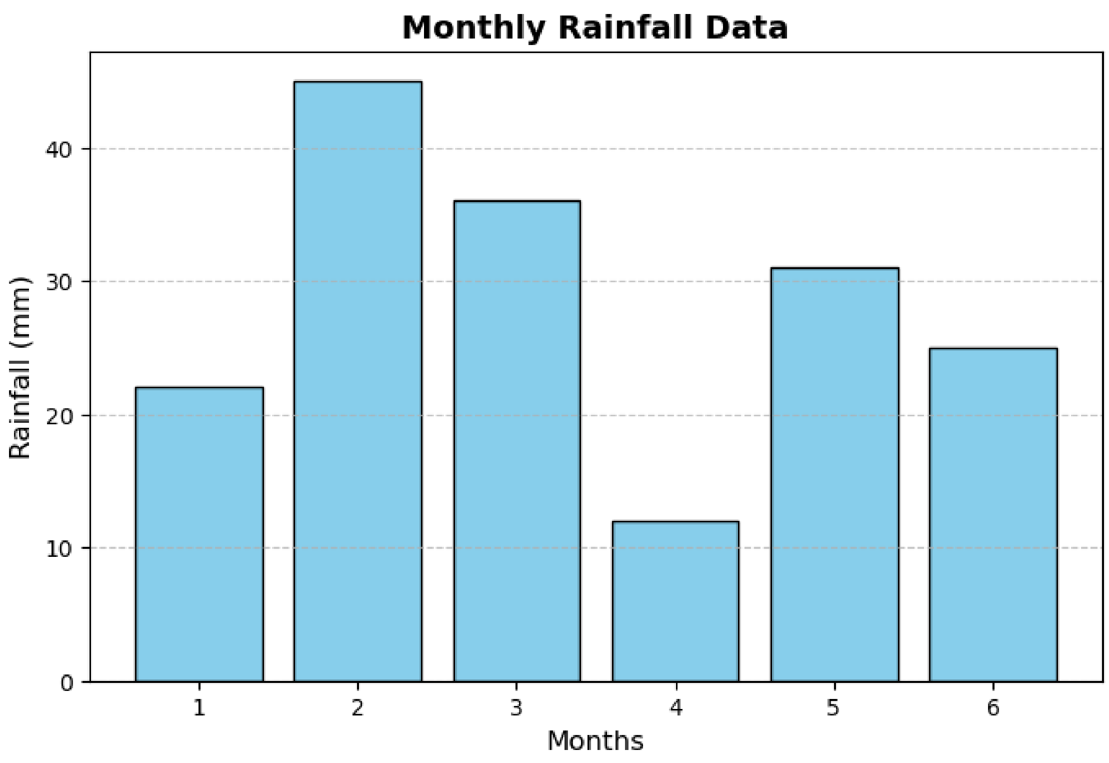
3.6 Key Takeaways:

- `subplot(nrows, ncols, index)` helps create **multiple plots** in a single figure.
- Helps in **visual comparison** between datasets.
- `plt.tight_layout()` ensures **plots don't overlap**.
- Grid and labels make **each subplot easy to read**.

With `subplot()`, we can **efficiently organize multiple plots** in a single figure while keeping our visualizations neat and professional.

3.6.1 Q5. Use the data of Q1 to draw a bar chart.

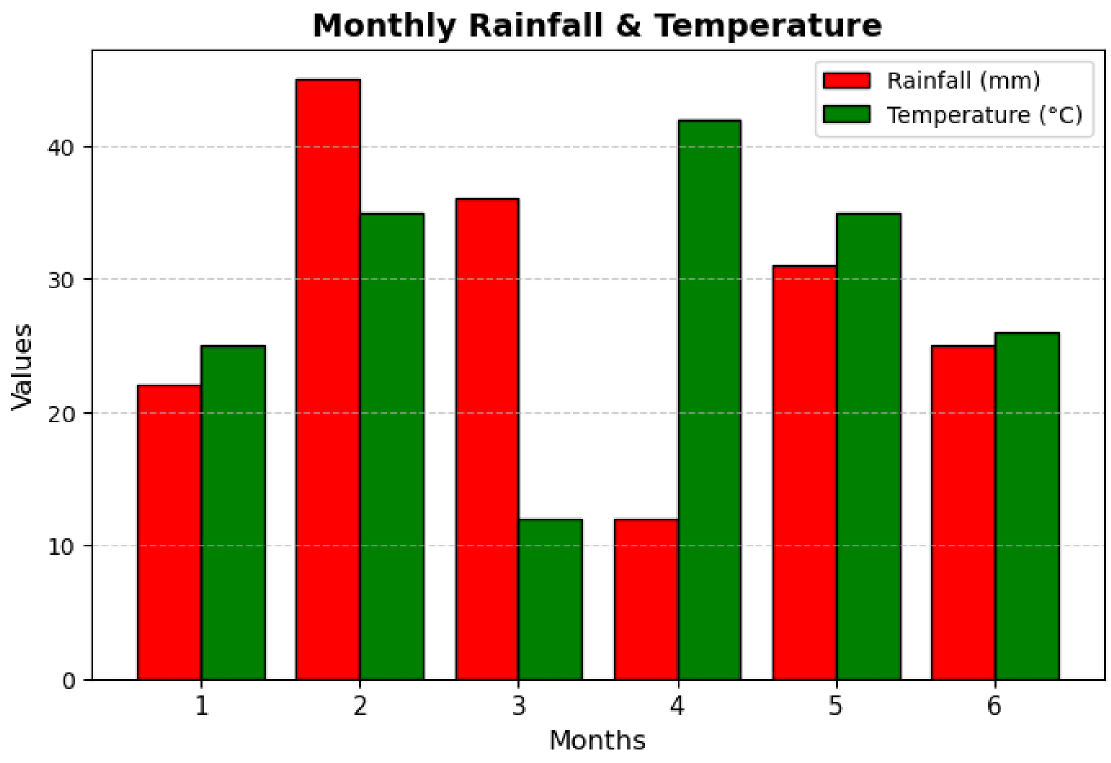
[4]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Taking user input for months and rainfall values
#months = input("Enter space-separated month names: ").split()
#rainfall = list(map(float, input("Enter space-separated rainfall values (in␣
 ↪mm): ").split()))
# No need to take input as we are taking the input of previous problem

# Converting lists to NumPy arrays
months = np.array(months)
rainfall = np.array(rainfall)

# Creating a bar chart
plt.figure(figsize=(8, 5))
plt.bar(months, rainfall, color='skyblue', edgecolor='black')

# Adding labels and title
plt.xlabel("Months", fontsize=12)
plt.ylabel("Rainfall (mm)", fontsize=12)
plt.title("Monthly Rainfall Data", fontsize=14, fontweight='bold')

# Adding grid for better readability
plt.grid(axis='y', linestyle="--", alpha=0.7)

# Displaying the chart
plt.show()
```



3.6.2 Q6. Use the data of Q2 to draw a bar graph showing the rainfall and temperature for each month.

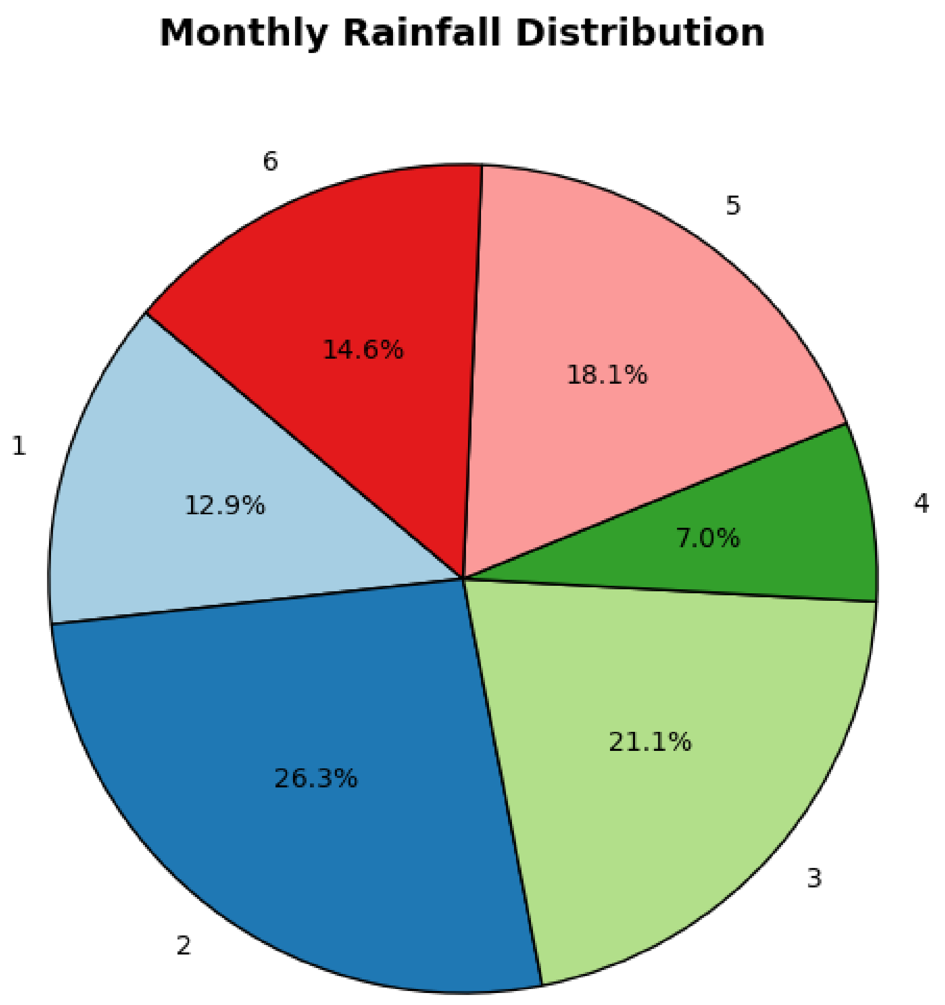
[5]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Taking user input for months, rainfall, and temperature
months = input("Enter space-separated month names: ").split()
rainfall = list(map(float, input("Enter space-separated rainfall values (in mm):
 ↪ ").split()))
temperature = list(map(float, input("Enter space-separated temperature values␣
 ↪(in °C): ").split()))

# Converting lists to NumPy arrays
months = np.array(months)
rainfall = np.array(rainfall)
temperature = np.array(temperature)

# Creating the x-axis positions
x_axis = np.arange(len(months))

# Plotting grouped bar charts
```

```
plt.figure(figsize=(8, 5))
bar_width = 0.4  # Width of each bar

plt.bar(x_axis - bar_width / 2, rainfall, bar_width, label="Rainfall (mm)",␣
↪color="red", edgecolor="black")
plt.bar(x_axis + bar_width / 2, temperature, bar_width, label="Temperature␣
↪(°C)", color="green", edgecolor="black")

# Customizing x-axis labels
plt.xticks(x_axis, months, fontsize=11)

# Adding labels, title, and legend
plt.xlabel("Months", fontsize=12)
plt.ylabel("Values", fontsize=12)
plt.title("Monthly Rainfall & Temperature", fontsize=14, fontweight='bold')
plt.legend()
plt.grid(axis='y', linestyle="--", alpha=0.6)

# Displaying the chart
plt.show()
```



3.6.3 Q7. Use data of Q1 to draw a pie chart.

[6]:
```
import numpy as np
import matplotlib.pyplot as plt

# Taking user input for months and rainfall values
#months = input("Enter space-separated month names: ").split()
#rainfall = list(map(float, input("Enter space-separated rainfall values (in␣
 ↪mm): ").split()))
# No need to take input we will use from previous inputs

# Converting lists to NumPy arrays
months = np.array(months)
rainfall = np.array(rainfall)

# Creating the pie chart
plt.figure(figsize=(7, 7))
plt.pie(
    rainfall, labels=months, autopct="%1.1f%%", startangle=140,
    colors=plt.cm.Paired.colors, wedgeprops={"edgecolor": "black"}
)

# Adding title
plt.title("Monthly Rainfall Distribution", fontsize=14, fontweight="bold")

# Displaying the chart
plt.show()
```



4 Understanding Pie Charts & Bar Diagrams in Data Visualization**

4.1 Introduction

Data visualization is an essential tool in statistics and data science. Among the most commonly used visualizations are **pie charts** and **bar diagrams**. These help in understanding and interpreting data efficiently.

This guide explains **what pie charts and bar diagrams are, when to use them, and how they work** in a simple and professional way.

---

5 1. Pie Chart

5.0.1 What is a Pie Chart?

A **pie chart** is a **circular graph** that represents data as slices of a pie. Each slice corresponds to a **proportion** of the total dataset. It is mainly used to **show the percentage or relative size of different categories** in a dataset.

5.0.2 When to Use a Pie Chart?

When you need to **show proportions or percentages**.
When comparing **a few categories (ideally less than 6-8)**.
When the total sum of all values **represents 100%** of something.

5.0.3 Basic Structure of a Pie Chart:

- **Each slice represents a category** in the dataset.
- The **size of each slice** is proportional to its percentage of the total.
- Labels are used to **identify each slice**.
- **Different colors** distinguish categories visually.

5.0.4 Example of a Pie Chart:

```
import matplotlib.pyplot as plt

# Data for the pie chart
labels = ["Apple", "Banana", "Grapes", "Mango"]
values = [30, 20, 25, 25]  # Percentage distribution

# Creating the pie chart
plt.figure(figsize=(7, 7))
plt.pie(values, labels=labels, autopct="%1.1f%%", startangle=140, colors=plt.cm.Paired.colors)
plt.title("Fruit Sales Distribution", fontsize=14, fontweight="bold")

# Displaying the chart
plt.show()
```

5.0.5 Key Features in the Code:

`autopct="%1.1f%%"` → Displays the percentage on the chart.
`startangle=140` → Rotates the pie for better visualization.
`plt.cm.Paired.colors` → Uses a predefined color scheme.

5.0.6 Limitations of Pie Charts:

Not suitable for **large datasets**.
Difficult to **compare exact values** between slices.

If categories have **similar sizes**, differences may not be clear.

---

6 2. Bar Diagram

6.0.1 What is a Bar Diagram?

A **bar diagram (or bar chart)** is a graph that represents data using **rectangular bars**. The length or height of each bar is **proportional to the value it represents**.

6.0.2 When to Use a Bar Diagram?

When comparing **individual values across categories**.
When working with **large datasets with many categories**.
When showing **exact differences** between values.

6.0.3 Basic Structure of a Bar Diagram:

- **X-axis:** Represents different categories.
- **Y-axis:** Represents numerical values.
- **Bars:** Heights of bars indicate the values.
- **Color & labels** help in clear visualization.

6.0.4 Example of a Bar Diagram:

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

# Data for the bar chart
categories = ["Jan", "Feb", "Mar", "Apr", "May"]
values = [50, 70, 40, 90, 60]

# Creating the bar chart
plt.figure(figsize=(8, 5))
plt.bar(categories, values, color="blue", edgecolor="black")

# Adding labels and title
plt.xlabel("Months", fontsize=12)
plt.ylabel("Sales (in units)", fontsize=12)
plt.title("Monthly Sales Data", fontsize=14, fontweight="bold")
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Displaying the chart
plt.show()
```

6.0.5 Key Features in the Code:

`plt.bar(categories, values, color="blue")` → Creates a bar chart with blue bars.
`edgecolor="black"` → Adds a black border to bars for better visibility.
`plt.grid(axis="y", linestyle="--", alpha=0.7)` → Adds a **dashed grid** for readability.

6.0.6 Types of Bar Diagrams:

1 **Vertical Bar Chart** → Bars are **standing (default)**.
2 **Horizontal Bar Chart** → Bars are **laid flat** using `plt.barh()`.
3 **Grouped Bar Chart** → Two sets of bars for **comparing multiple datasets**.

6.0.7 Limitations of Bar Charts:

If too many categories, **bars may become too small to read**.
Does not show **proportions like a pie chart**.

---

7 Comparison: Pie Chart vs. Bar Diagram

Feature	Pie Chart	Bar Diagram
Use Case	Show **proportions**	Show **comparisons**
Best for	Small datasets (6 categories)	Large datasets
Data Type	**Percentage-based**	**Exact values**
Easy to Read?	Harder for similar values	Easier
Exact Values?	No, only relative sizes	Yes

---

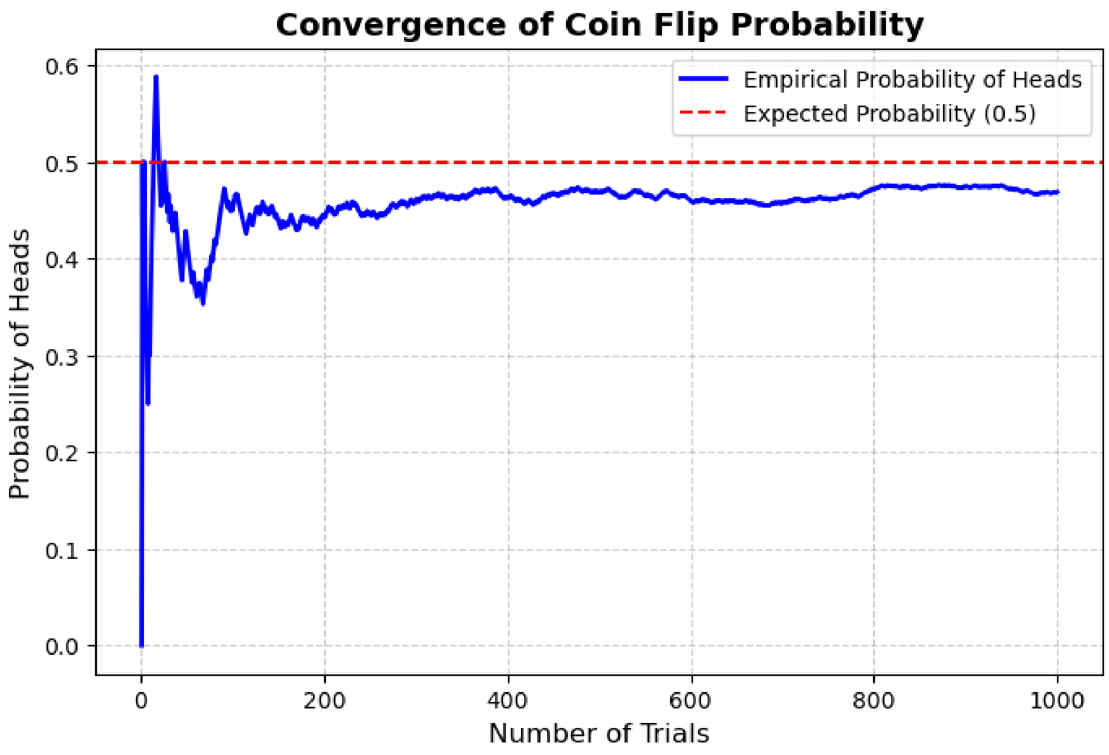
8 Key Takeaways

Pie charts are best for showing **percentage distributions**.
Bar charts are best for comparing **exact values across categories**.
Choose the **right chart** based on **what you want to communicate**.
Use **labels, colors, and gridlines** for clear visualization.

By understanding these concepts, you can **effectively present and analyze data** in a **clear and professional way**!

8.0.1 Q8. Consider a trial of tossing an unbiased coin n (a large value) times. Use random.random() [generates a random float number between 0 and 1] to simulate the observation after each coin toss as follows: if the random number is <0.5, assume that H has occurred, tail otherwise. Count the number of heads and compute the fraction of heads generated after each coin toss. This is basically to estimate the probability of head (that is 0.5). Plot the estimated probability (using matplotlib.pyplot.plot) along the y-axis, while the trial numbers are plotted along the x-axis.

[7]:
```
import random
import matplotlib.pyplot as plt

# Initialize variables
n = 1000  # Number of trials
head_count = 0
x = []
y = []

# Simulating coin flips
for i in range(1, n + 1):
    x.append(i)
    if random.random() < 0.5:  # Probability of getting heads
        head_count += 1
    y.append(head_count / i)  # Calculate probability of heads

# Plotting the results
plt.figure(figsize=(8, 5))
plt.plot(x, y, color="blue", linewidth=2, label="Empirical Probability of␣
 ↪Heads")

# Formatting the plot
plt.xlabel("Number of Trials", fontsize=12)
plt.ylabel("Probability of Heads", fontsize=12)
plt.title("Convergence of Coin Flip Probability", fontsize=14,␣
 ↪fontweight="bold")
plt.axhline(y=0.5, color="red", linestyle="--", label="Expected Probability (0.
 ↪5)")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)

# Save and display the plot
plt.savefig("head_probability.png", dpi=300, bbox_inches="tight")
plt.show()
```


Convergence of Coin Flip Probability
Empirical Probability of Heads
Expected Probability (0.5)
Probability of Heads
Number of Trials
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
200
400
600
800
1000