

Problem Solving Using Python

Unit IV: Data Visualization and Practice

Course Code: 25ES142

Credits: 4

Course Type: IPCC

Prepared by:
Mohammed Siraj B
School of Engineering
St Aloysius (Deemed to be University), Mangalore

For Beginner Students
Academic Year 2024-25

Contents

1	Introduction to Data Visualization	3
1.1	What is Data Visualization?	3
1.2	Why is Data Visualization Important?	3
1.3	Introduction to Matplotlib	3
1.4	Installing Matplotlib	3
1.5	Importing Matplotlib	3
1.6	Basic Plotting Workflow	4
2	Line Plots	5
2.1	What is a Line Plot?	5
2.2	When to Use Line Plots?	5
2.3	Basic Syntax	5
2.4	Example 1: Simple Line Plot	5
2.5	Example 2: Line Plot with Labels	6
2.6	Example 3: Multiple Lines in One Plot	7
2.7	Example 4: Customizing Line Appearance	8
2.8	Common Line Styles and Markers	9
2.9	Example 5: Real-World Application	10
3	Bar Graphs	12
3.1	What is a Bar Graph?	12
3.2	When to Use Bar Graphs?	12
3.3	Types of Bar Graphs	12
3.4	Basic Syntax	12
3.5	Example 6: Simple Bar Graph	13
3.6	Example 7: Customizing Bar Colors	14
3.7	Example 8: Horizontal Bar Chart	15
3.8	Example 9: Grouped Bar Chart	16
3.9	Example 10: Stacked Bar Chart	17
4	Pie Charts	19
4.1	What is a Pie Chart?	19
4.2	When to Use Pie Charts?	19
4.3	Basic Syntax	19
4.4	Example 11: Simple Pie Chart	20
4.5	Example 12: Pie Chart with Percentages	21
4.6	Example 13: Customized Pie Chart with Colors	22
4.7	Example 14: Exploded Pie Chart	23
4.8	Example 15: Pie Chart with Legend	24
4.9	Pie Chart Best Practices	25
4.10	Common Pie Chart Parameters	25
5	Customizing Plots	26
5.1	Introduction to Plot Customization	26
5.2	Why Customize Plots?	26
5.3	Essential Customization Elements	26

6 Adding Titles, Labels, and Legends	27
6.1 Syntax for Common Elements	27
6.2 Example 16: Complete Customization	28
6.3 Title Customization Options	29
6.4 Legend Customization Options	29
6.5 Example 17: Multiple Customization Techniques	31
7 Working with Colors	33
7.1 Color Specification Methods	33
7.2 Common Color Codes	33
7.3 Example 18: Using Different Color Methods	34
7.4 Color Palettes and Gradients	35
7.5 Popular Colormaps	36
7.6 Transparency and Alpha Values	37
8 Advanced Customization Techniques	39
8.1 Figure and Subplot Management	39
8.2 Saving Figures	40
8.3 Saving Options	41
9 Integrating Everything - Complete Examples	43
9.1 Example 23: Comprehensive Sales Dashboard	43
9.2 Example 24: Student Performance Analysis	46
10 Practice Exercises	49
11 Quick Reference Guide	50
11.1 Matplotlib Essentials Cheat Sheet	50
11.2 Common Patterns	51
11.3 Best Practices Summary	52
11.4 Common Issues and Solutions	52

1 Introduction to Data Visualization

1.1 What is Data Visualization?

Definition

Data Visualization is the graphical representation of data and information. It uses visual elements like charts, graphs, and maps to help people understand patterns, trends, and insights in data easily.

1.2 Why is Data Visualization Important?

- **Easy to Understand:** Pictures are easier to understand than numbers
- **Quick Insights:** Spot trends and patterns instantly
- **Better Communication:** Share information effectively with others
- **Decision Making:** Make informed decisions based on visual data
- **Story Telling:** Tell compelling stories with data

1.3 Introduction to Matplotlib

What is Matplotlib?

Matplotlib is a comprehensive library in Python for creating static, animated, and interactive visualizations. It's the most popular plotting library in Python.

1.4 Installing Matplotlib

```
1 # Install matplotlib using pip
2 pip install matplotlib
3
4 # Or install with numpy (recommended)
5 pip install matplotlib numpy
```

1.5 Importing Matplotlib

```
1 # Standard way to import matplotlib
2 import matplotlib.pyplot as plt
3
4 # pyplot is the most commonly used module
5 # 'plt' is the standard abbreviation
```

1.6 Basic Plotting Workflow

1. Import matplotlib
2. Prepare your data (lists, arrays, etc.)
3. Create the plot using plt functions
4. Customize the plot (titles, labels, colors, etc.)
5. Display the plot using plt.show()

2 Line Plots

2.1 What is a Line Plot?

Definition

A **Line Plot** displays data as a series of points connected by straight lines. It's perfect for showing trends over time or continuous data.

2.2 When to Use Line Plots?

- Showing changes over time (temperature, stock prices, etc.)
- Displaying trends and patterns
- Comparing multiple data series
- Continuous data representation

2.3 Basic Syntax

```
1 import matplotlib.pyplot as plt
2
3 # Basic line plot
4 plt.plot(x_values, y_values)
5 plt.show()
```

2.4 Example 1: Simple Line Plot

Example 1: Student Marks Over Time

```
1 import matplotlib.pyplot as plt
2
3 # Data: Student marks in 5 tests
4 tests = [1, 2, 3, 4, 5]
5 marks = [65, 70, 75, 80, 85]
6
7 # Create line plot
8 plt.plot(tests, marks)
9
10 # Display the plot
11 plt.show()
12
13 print("Line plot created successfully!")
```

Output/Visualization

This creates a simple line plot showing marks increasing from 65 to 85 over 5 tests. The x-axis shows test numbers (1-5) and y-axis shows marks (65-85).

2.5 Example 2: Line Plot with Labels

Example 2: Temperature Over a Week

```
1 import matplotlib.pyplot as plt
2
3 # Data: Temperature over a week
4 days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
5 temperature = [28, 30, 29, 31, 32, 30, 28]
6
7 # Create line plot
8 plt.plot(days, temperature)
9
10 # Add labels and title
11 plt.xlabel('Day of Week')
12 plt.ylabel('Temperature ( C )')
13 plt.title('Weekly Temperature Chart')
14
15 # Display the plot
16 plt.show()
17
18 print("Temperature chart created!")
```

Output/Visualization

Creates a line plot with: - X-axis labeled "Day of Week" with days Mon-Sun - Y-axis labeled "Temperature (°C)" with values 28-32 - Title "Weekly Temperature Chart" at the top

2.6 Example 3: Multiple Lines in One Plot

Example 3: Comparing Two Students' Performance

```
1 import matplotlib.pyplot as plt
2
3 # Data for two students
4 tests = [1, 2, 3, 4, 5]
5 alice_marks = [65, 70, 75, 80, 85]
6 bob_marks = [70, 68, 72, 75, 80]
7
8 # Plot both lines
9 plt.plot(tests, alice_marks, label='Alice')
10 plt.plot(tests, bob_marks, label='Bob')
11
12 # Add labels and title
13 plt.xlabel('Test Number')
14 plt.ylabel('Marks')
15 plt.title('Student Performance Comparison')
16
17 # Add legend
18 plt.legend()
19
20 # Add grid for better readability
21 plt.grid(True)
22
23 # Display
24 plt.show()
25
26 print("Comparison chart created!")
```

Output/Visualization

Creates a plot with:

- Two lines: one for Alice (showing marks 65-85) and one for Bob (70-80)
- Legend showing which line represents which student
- Grid lines for easier reading of values
- Labels and title properly displayed

2.7 Example 4: Customizing Line Appearance

Example 4: Styling Line Plots

```
1 import matplotlib.pyplot as plt
2
3 # Data
4 months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
5 sales = [15000, 18000, 22000, 19000, 25000, 28000]
6
7 # Create line plot with custom style
8 plt.plot(months, sales,
9           color='green',          # Line color
10          linewidth=3,           # Line thickness
11          marker='o',            # Add circular markers
12          markersize=8,          # Marker size
13          linestyle='--',        # Dashed line
14          label='Monthly Sales')
15
16 # Customize
17 plt.xlabel('Month', fontsize=12)
18 plt.ylabel('Sales (Rs.)', fontsize=12)
19 plt.title('Sales Trend - First Half 2024', fontsize=14,
20           fontweight='bold')
21 plt.legend()
22 plt.grid(True, alpha=0.3)    # alpha controls transparency
23
24 # Display
25 plt.show()
26
27 print("Styled sales chart created!")
```

Output/Visualization

Creates a professional-looking plot with:

- Green dashed line with circular markers
- Thick line (width=3) and large markers (size=8)
- Transparent grid (alpha=0.3)
- Bold title with larger fonts
- Shows sales increasing from 15,000 to 28,000

2.8 Common Line Styles and Markers

Parameter	Options	Description
<code>linestyle</code>	'-' , '--' , '-.' , ':'	Solid, Dashed, Dash-dot, Dotted
<code>marker</code>	'o' , 's' , '^' , '*' , 'D'	Circle, Square, Triangle, Star, Diamond
<code>color</code>	'r' , 'g' , 'b' , 'k'	Red, Green, Blue, Black
	'cyan' , 'magenta'	Or use color names
<code>linewidth</code>	1, 2, 3, ...	Thickness of line
<code>markersize</code>	5, 8, 10, ...	Size of markers

Table 1: Line Plot Customization Options

2.9 Example 5: Real-World Application

Example 5: COVID-19 Cases Tracking (Sample Data)

```
1 import matplotlib.pyplot as plt
2
3 # Sample data: Daily new cases over 10 days
4 days = list(range(1, 11))
5 new_cases = [120, 135, 150, 145, 160, 175, 170, 185, 190,
6               200]
7
8 # Create figure with specific size
9 plt.figure(figsize=(10, 6))
10
11 # Plot the data
12 plt.plot(days, new_cases,
13           color='red',
14           linewidth=2,
15           marker='o',
16           markersize=6,
17           label='Daily New Cases')
18
19 # Customization
20 plt.xlabel('Day', fontsize=12)
21 plt.ylabel('Number of Cases', fontsize=12)
22 plt.title('COVID-19 Daily New Cases Trend', fontsize=14,
23            fontweight='bold')
24
25 # Add grid and legend
26 plt.grid(True, linestyle='--', alpha=0.5)
27 plt.legend(loc='upper left')
28
29 # Add average line
30 average = sum(new_cases) / len(new_cases)
31 plt.axhline(y=average, color='blue', linestyle='--',
32              label=f'Average: {average:.0f}')
33 plt.legend()
34
35 # Display
36 plt.tight_layout() # Adjust layout to prevent label
37 plt.show()
38
39 print(f"Chart created. Average cases: {average:.2f}")
```

Output/Visualization

Creates a comprehensive plot showing:

- Red line with markers showing daily cases (120 to 200)
- Blue dashed horizontal line showing average (164.5 cases)
- Grid for easy value reading
- Legend showing both data series
- Professional layout with proper spacing
- Displays: "Chart created. Average cases: 164.50"

Pro Tip

Best Practices for Line Plots:

- Use different colors for multiple lines
- Always add labels and title
- Include legend when showing multiple series
- Use grid for better readability
- Choose appropriate line styles for different data
- Don't overcrowd - maximum 3-4 lines per chart

3 Bar Graphs

3.1 What is a Bar Graph?

Definition

A **Bar Graph** uses rectangular bars to represent data values. The length or height of each bar is proportional to the value it represents. Great for comparing categories!

3.2 When to Use Bar Graphs?

- Comparing quantities across categories
- Showing discrete data
- Displaying survey results
- Comparing performance of different groups
- Showing rankings or counts

3.3 Types of Bar Graphs

Type	Description
Vertical Bar	Bars grow upward from x-axis (most common)
Horizontal Bar	Bars extend from y-axis
Grouped Bar	Multiple bars for each category (side by side)
Stacked Bar	Bars stacked on top of each other

Table 2: Types of Bar Graphs

3.4 Basic Syntax

```
1 import matplotlib.pyplot as plt
2
3 # Vertical bar chart
4 plt.bar(x_values, y_values)
5
6 # Horizontal bar chart
7 plt.bart(y_values, x_values)
8
9 plt.show()
```

3.5 Example 6: Simple Bar Graph

Example 6: Subject-wise Marks

```
1 import matplotlib.pyplot as plt
2
3 # Data
4 subjects = ['Math', 'Physics', 'Chemistry', 'Biology', 'English']
5 marks = [85, 78, 90, 82, 88]
6
7 # Create bar graph
8 plt.bar(subjects, marks)
9
10 # Add labels
11 plt.xlabel('Subjects')
12 plt.ylabel('Marks')
13 plt.title('My Subject-wise Marks')
14
15 # Display
16 plt.show()
17
18 print("Bar chart created successfully!")
```

Output/Visualization

Creates a vertical bar chart with:

- 5 bars representing each subject
- Heights showing marks (78-90)
- X-axis showing subject names
- Y-axis showing marks
- Simple blue bars (default color)

3.6 Example 7: Customizing Bar Colors

Example 7: Colorful Product Sales

```
1 import matplotlib.pyplot as plt
2
3 # Data
4 products = ['Laptop', 'Mobile', 'Tablet', 'Headphones', 'Watch']
5 sales = [45, 78, 32, 56, 41]
6
7 # Define colors for each bar
8 colors = ['red', 'green', 'blue', 'orange', 'purple']
9
10 # Create bar graph with custom colors
11 plt.bar(products, sales, color=colors, width=0.6)
12
13 # Add labels
14 plt.xlabel('Products', fontsize=12)
15 plt.ylabel('Units Sold', fontsize=12)
16 plt.title('Product Sales Report', fontsize=14, fontweight='bold')
17
18 # Add values on top of bars
19 for i, v in enumerate(sales):
20     plt.text(i, v + 1, str(v), ha='center', fontweight='bold')
21
22 # Display
23 plt.show()
24
25 print("Colorful bar chart with values created!")
```

Output/Visualization

Creates an attractive bar chart with:

- Each bar in a different color (red, green, blue, orange, purple)
- Numbers displayed on top of each bar
- Mobile has highest sales (78), Tablet has lowest (32)
- Bars are 60% of default width (looks better with spacing)

3.7 Example 8: Horizontal Bar Chart

Example 8: Department-wise Student Count

```
1 import matplotlib.pyplot as plt
2
3 # Data
4 departments = ['CSE', 'ECE', 'Mech', 'Civil', 'EEE']
5 students = [120, 95, 80, 70, 85]
6
7 # Create horizontal bar chart
8 plt.bart(departments, students, color='skyblue')
9
10 # Add labels
11 plt.xlabel('Number of Students')
12 plt.ylabel('Departments')
13 plt.title('Department-wise Student Enrollment')
14
15 # Add grid for better readability
16 plt.grid(axis='x', alpha=0.3)
17
18 # Display
19 plt.show()
20
21 print("Horizontal bar chart created!")
```

Output/Visualization

Creates a horizontal bar chart with: - Bars extending from left to right - CSE has most students (120) - Light blue color for all bars - Vertical grid lines for easy reading - Department names on y-axis

3.8 Example 9: Grouped Bar Chart

Example 9: Comparing Two Years' Performance

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Data for two years
5 subjects = ['Math', 'Physics', 'Chemistry', 'Biology']
6 year2023 = [75, 80, 85, 78]
7 year2024 = [82, 85, 88, 85]
8
9 # Set position of bars
10 x = np.arange(len(subjects))
11 width = 0.35 # Width of bars
12
13 # Create bars
14 plt.bar(x - width/2, year2023, width, label='2023', color='lightblue')
15 plt.bar(x + width/2, year2024, width, label='2024', color='lightgreen')
16
17 # Customize
18 plt.xlabel('Subjects')
19 plt.ylabel('Average Marks')
20 plt.title('Year-wise Subject Performance Comparison')
21 plt.xticks(x, subjects)
22 plt.legend()
23 plt.grid(axis='y', alpha=0.3)
24
25 # Display
26 plt.show()
27
28 print("Grouped bar chart created!")
```

Output/Visualization

Creates a grouped bar chart showing:

- Two bars for each subject (side by side)
- Blue bars for 2023, green bars for 2024
- Clear comparison showing improvement in all subjects
- Legend identifying the years
- Proper spacing between groups

3.9 Example 10: Stacked Bar Chart

Example 10: Monthly Expense Breakdown

```
1 import matplotlib.pyplot as plt
2
3 # Data
4 months = ['Jan', 'Feb', 'Mar', 'Apr']
5 rent = [10000, 10000, 10000, 10000]
6 food = [5000, 5500, 5200, 5800]
7 transport = [2000, 2200, 2100, 2300]
8
9 # Create stacked bars
10 plt.bar(months, rent, label='Rent', color='red')
11 plt.bar(months, food, bottom=rent, label='Food', color='green')
12
13 # Calculate bottom for transport (rent + food)
14 bottom_transport = [rent[i] + food[i] for i in range(len(months))]
15 plt.bar(months, transport, bottom=bottom_transport,
16         label='Transport', color='blue')
17
18 # Customize
19 plt.xlabel('Month')
20 plt.ylabel('Expense (Rs.)')
21 plt.title('Monthly Expense Breakdown')
22 plt.legend()
23
24 # Display
25 plt.show()
26
27 print("Stacked bar chart created!")
```

Output/Visualization

Creates a stacked bar chart showing:

- Each month has one bar with three sections
- Red (bottom) = Rent, Green (middle) = Food, Blue (top) = Transport
- Total height shows total monthly expense
- Easy to see both individual and total expenses
- April has highest total expense

Important Note

Important Bar Graph Tips:

- Keep bar width reasonable (0.5 to 0.8 works well)
- Add values on bars for small datasets
- Use horizontal bars for long category names
- Limit number of bars (max 10-12 for readability)
- Start y-axis at zero for accurate comparison
- Use consistent colors for related data

4 Pie Charts

4.1 What is a Pie Chart?

Definition

A **Pie Chart** is a circular statistical graphic divided into slices to illustrate numerical proportion. Each slice represents a category's contribution to the whole (shown as percentage).

4.2 When to Use Pie Charts?

- Showing parts of a whole (percentages)
- Displaying composition or distribution
- When you have 3-7 categories (not too many!)
- Budget allocation
- Market share analysis
- Survey results with categories

4.3 Basic Syntax

```
1 import matplotlib.pyplot as plt
2
3 # Data
4 categories = ['Category1', 'Category2', 'Category3']
5 values = [30, 50, 20]
6
7 # Create pie chart
8 plt.pie(values, labels=categories)
9 plt.show()
```

4.4 Example 11: Simple Pie Chart

Example 11: Time Spent on Activities

```
1 import matplotlib.pyplot as plt
2
3 # Data: Hours spent on different activities
4 activities = ['Studying', 'Sleeping', 'Sports', ,
5   Entertainment']
6 hours = [6, 8, 2, 4]
7
8 # Create pie chart
9 plt.pie(hours, labels=activities)
10
11 # Add title
12 plt.title('Daily Time Distribution (24 hours)')
13
14 # Display
15 plt.show()
16 print("Pie chart created successfully!")
```

Output/Visualization

Creates a basic pie chart with:

- 4 slices representing different activities
- Sleeping takes largest slice (8 hours = 33.3%)
- Studying: 25%, Sports: 8.3%, Entertainment: 16.7%
- Different colors automatically assigned to each slice

4.5 Example 12: Pie Chart with Percentages

Example 12: Market Share Analysis

```
1 import matplotlib.pyplot as plt
2
3 # Data: Market share of smartphone brands
4 brands = ['Samsung', 'Apple', 'Xiaomi', 'Oppo', 'Others']
5 market_share = [28, 25, 18, 12, 17]
6
7 # Create pie chart with percentages
8 plt.pie(market_share,
9         labels=brands,
10        autopct='%.1f%%',    # Show percentages
11        startangle=90)       # Start from top
12
13 # Add title
14 plt.title('Smartphone Market Share 2024')
15
16 # Equal aspect ratio ensures circular shape
17 plt.axis('equal')
18
19 # Display
20 plt.show()
21
22 print("Market share pie chart created!")
```

Output/Visualization

Creates a pie chart showing:

- Each slice labeled with brand name
- Percentage displayed on each slice (28.0%, 25.0%, etc.)
- Samsung has largest market share (28%)
- Chart starts at 90 degrees (top of circle)
- Perfect circular shape maintained

4.6 Example 13: Customized Pie Chart with Colors

Example 13: Budget Allocation

```
1 import matplotlib.pyplot as plt
2
3 # Data: Monthly budget allocation
4 categories = ['Rent', 'Food', 'Transport', 'Entertainment',
5 , 'Savings']
6 amounts = [10000, 6000, 3000, 2000, 4000]
7
8 # Custom colors
9 colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99', '#
10 ff99cc']
11
12 # Create pie chart
13 plt.figure(figsize=(8, 8))
14 plt.pie(amounts,
15         labels=categories,
16         colors=colors,
17         autopct='%1.1f%%',
18         startangle=140,
19         shadow=True) # Add shadow effect
20
21 # Title
22 plt.title('Monthly Budget Allocation (Rs. 25,000)',
23           fontsize=14, fontweight='bold')
24
25 # Display
26 plt.axis('equal')
27 plt.show()
28
29 print("Customized budget pie chart created!")
```

Output/Visualization

Creates an attractive pie chart with:

- Custom pastel colors for each category
- Shadow effect making it look 3D
- Percentages on each slice
- Rent: 40%, Food: 24%, Savings: 16%, Transport: 12%, Entertainment: 8%
- Larger figure size (8x8 inches) for better visibility

4.7 Example 14: Exploded Pie Chart

Example 14: Student Grade Distribution

```
1 import matplotlib.pyplot as plt
2
3 # Data: Number of students in each grade
4 grades = ['A', 'B', 'C', 'D', 'F']
5 students = [25, 35, 20, 12, 8]
6
7 # Explode the 'A' grade slice
8 explode = (0.1, 0, 0, 0, 0) # Only first slice
9
10 # Colors
11 colors = ['gold', 'lightgreen', 'lightblue', 'orange', 'red']
12
13 # Create exploded pie chart
14 plt.pie(students,
15         labels=grades,
16         colors=colors,
17         autopct='%1.1f%%',
18         startangle=90,
19         explode=explode, # Separate first slice
20         shadow=True)
21
22 plt.title('Class Grade Distribution (Total: 100 students)',
23            ,
24            fontsize=12, fontweight='bold')
25 plt.axis('equal')
26
27 # Display
28 plt.show()
29
30 print("Grade distribution chart created!")
```

Output/Visualization

Creates a pie chart with: - 'A' grade slice pulled out (exploded) to emphasize it - Color-coded: Gold (A), Green (B), Blue (C), Orange (D), Red (F) - Shadow effect for depth - Shows A: 25%, B: 35%, C: 20%, D: 12%, F: 8% - B grade has most students (35 students)

4.8 Example 15: Pie Chart with Legend

Example 15: Project Time Distribution

```

1 import matplotlib.pyplot as plt
2
3 # Data: Hours spent on different project phases
4 phases = ['Planning', 'Design', 'Development', 'Testing',
5           'Deployment']
6 hours = [10, 15, 40, 20, 15]
7
8 # Colors
9 colors = ['#ff6b6b', '#4ecdc4', '#45b7d1', '#96ceb4', '#
10   ffeaa7']
11
12 # Create pie chart
13 plt.figure(figsize=(10, 7))
14 plt.pie(hours,
15          labels=phases,
16          colors=colors,
17          autopct='%1.1f%%',
18          startangle=90,
19          counterclock=False) # Clockwise
20
21 # Add legend
22 plt.legend(phases,
23            title="Project Phases",
24            loc="center left",
25            bbox_to_anchor=(1, 0, 0.5, 1))
26
27 plt.title('Software Project Time Distribution\nTotal: 100
28   hours',
29             fontsize=14, fontweight='bold')
30 plt.axis('equal')
31 plt.tight_layout()
32
33 # Display
34 plt.show()
35
36 print("Project time distribution chart with legend created
37   !")

```

Output/Visualization

Creates a professional pie chart with:

- Legend box outside the pie (on the right)
- Different colors for each phase
- Shows Development takes most time (40%)
- Clockwise arrangement starting from top
- Clean layout with title spanning two lines
- Total hours mentioned in subtitle

4.9 Pie Chart Best Practices

Pro Tip

Do's and Don'ts for Pie Charts:

DO:

- Use for 3-7 categories maximum
- Show percentages on slices
- Use contrasting colors
- Order slices by size (largest first)
- Keep it simple and clear

DON'T:

- Use 3D effects (they distort perception)
- Include too many categories (use bar chart instead)
- Use similar colors for adjacent slices
- Make tiny slices (group small values as "Others")
- Forget to show percentages or values

4.10 Common Pie Chart Parameters

Parameter	Description
<code>autopct</code>	Format string for percentages (e.g., <code>'%1.1f%%'</code>)
<code>startangle</code>	Angle to start the pie (default 0, try 90 for top)
<code>explode</code>	Tuple of fractions to offset slices (e.g., <code>(0.1, 0, 0)</code>)
<code>shadow</code>	Boolean, adds shadow effect (True/False)
<code>colors</code>	List of colors for each slice
<code>labels</code>	List of labels for each slice
<code>counterclock</code>	Boolean, arrange slices clockwise (False) or counter-clockwise (True)

Table 3: Pie Chart Customization Parameters

5 Customizing Plots

5.1 Introduction to Plot Customization

What is Plot Customization?

Plot customization means enhancing your visualizations by adding titles, labels, legends, colors, and other visual elements to make them more informative and attractive.

5.2 Why Customize Plots?

- **Clarity:** Make data easier to understand
- **Professionalism:** Create publication-ready charts
- **Communication:** Convey information effectively
- **Aesthetics:** Make charts visually appealing
- **Context:** Provide necessary information to readers

5.3 Essential Customization Elements

1. **Title:** Main heading of the chart
2. **Labels:** Text for X and Y axes
3. **Legend:** Key explaining different data series
4. **Colors:** Visual differentiation
5. **Grid:** Background lines for reference
6. **Figure Size:** Dimensions of the plot

6 Adding Titles, Labels, and Legends

6.1 Syntax for Common Elements

```
1 import matplotlib.pyplot as plt
2
3 # Create plot
4 plt.plot(x, y)
5
6 # Add title
7 plt.title('Chart Title')
8
9 # Add axis labels
10 plt.xlabel('X-axis Label')
11 plt.ylabel('Y-axis Label')
12
13 # Add legend
14 plt.legend(['Data Series Name'])
15
16 # Add grid
17 plt.grid(True)
18
19 # Display
20 plt.show()
```

6.2 Example 16: Complete Customization

Example 16: Fully Customized Line Plot

```
1 import matplotlib.pyplot as plt
2
3 # Data
4 months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
5 sales = [15000, 18000, 22000, 19000, 25000, 28000]
6 expenses = [12000, 13000, 15000, 14000, 16000, 17000]
7
8 # Create larger figure
9 plt.figure(figsize=(10, 6))
10
11 # Plot data
12 plt.plot(months, sales, marker='o', linewidth=2,
13           color='green', label='Sales')
14 plt.plot(months, expenses, marker='s', linewidth=2,
15           color='red', label='Expenses')
16
17 # Add title with custom font
18 plt.title('Monthly Sales and Expenses Report - 2024',
19            fontsize=16,
20            fontweight='bold',
21            color='darkblue',
22            pad=20) # Space above chart
23
24 # Add axis labels
25 plt.xlabel('Month', fontsize=14, fontweight='bold')
26 plt.ylabel('Amount (Rs.)', fontsize=14, fontweight='bold')
27
28 # Add legend
29 plt.legend(loc='upper left', fontsize=12, frameon=True,
30             shadow=True, fancybox=True)
31
32 # Add grid
33 plt.grid(True, linestyle='--', alpha=0.6)
34
35 # Customize tick marks
36 plt.xticks(fontsize=11)
37 plt.yticks(fontsize=11)
38
39 # Tight layout
40 plt.tight_layout()
41
42 # Display
43 plt.show()
44
45 print("Fully customized chart created!")
```

Output/Visualization

Creates a professional chart with:

- Large figure size (10x6 inches)
- Bold, dark blue title with extra spacing
- Two lines: green for sales, red for expenses
- Circular and square markers
- Legend with shadow and rounded box (upper left)
- Dashed grid lines with 60% opacity
- Bold axis labels in larger font
- Profit visible (sales above expenses)

6.3 Title Customization Options

Parameter	Description
<code>fontsize</code>	Size of title text (8, 10, 12, 14, 16...)
<code>fontweight</code>	Weight: 'normal', 'bold', 'light', 'heavy'
<code>color</code>	Color: 'red', 'blue', '#FF5733', etc.
<code>pad</code>	Space between title and plot (in points)
<code>loc</code>	Position: 'left', 'center', 'right'
<code>style</code>	Font style: 'normal', 'italic', 'oblique'

Table 4: Title Customization Parameters

6.4 Legend Customization Options

Parameter	Description
<code>loc</code>	Location: 'upper left', 'upper right', 'lower left', 'lower right', 'center', 'best'
<code>fontsize</code>	Size of legend text
<code>frameon</code>	Show box around legend (True/False)
<code>shadow</code>	Add shadow effect (True/False)
<code>fancybox</code>	Rounded corners (True/False)
<code>title</code>	Title for the legend
<code>ncol</code>	Number of columns in legend

Table 5: Legend Customization Parameters

6.5 Example 17: Multiple Customization Techniques

Example 17: Advanced Styling

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Data for sine and cosine waves
5 x = np.linspace(0, 2*np.pi, 100)
6 y1 = np.sin(x)
7 y2 = np.cos(x)
8
9 # Create figure and axis
10 fig, ax = plt.subplots(figsize=(12, 6))
11
12 # Plot with different styles
13 ax.plot(x, y1, 'b-', linewidth=2.5, label='sin(x)')
14 ax.plot(x, y2, 'r--', linewidth=2.5, label='cos(x)')
15
16 # Title with multiple properties
17 ax.set_title('Sine and Cosine Waves',
18                 fontsize=18,
19                 fontweight='bold',
20                 color='darkgreen',
21                 pad=20,
22                 style='italic')
23
24 # Axis labels
25 ax.set_xlabel('Angle (radians)', fontsize=14, fontweight='bold',
26                 color='navy')
27 ax.set_ylabel('Value', fontsize=14, fontweight='bold',
28                 color='navy')
29
30 # Legend with custom position
31 ax.legend(loc='upper right', fontsize=12, framealpha=0.9,
32             edgecolor='black', fancybox=True, shadow=True)
33
34 # Grid customization
35 ax.grid(True, linestyle=':', linewidth=0.7, color='gray',
36             alpha=0.5)
37
38 # Set axis limits
39 ax.set_xlim([0, 2*np.pi])
40 ax.set_ylim([-1.5, 1.5])
41
42 # Add horizontal line at y=0
43 ax.axhline(y=0, color='k', linewidth=0.8, linestyle='--')
44
45 # Customize ticks
46 ax.tick_params(axis='both', labelsize=11, colors='darkblue')
47
48 # Add text annotation          31
49 ax.text(np.pi, 0, ' ', fontsize=14, ha='center', va='bottom')
```

Output/Visualization

Creates a sophisticated plot showing:

- Sine wave (blue solid line)
- cosine wave (red dashed line)
- Italic, bold green title
- Navy blue bold axis labels
- Legend with semi-transparent background
- Dotted gray grid
- Horizontal black line at $y=0$
- Blue tick labels
- An annotation at $x=$
- Perfect wave patterns from 0 to 2

7 Working with Colors

7.1 Color Specification Methods

Ways to Specify Colors in Matplotlib

Matplotlib supports multiple ways to specify colors:

1. Single character codes: 'r', 'g', 'b', 'k', 'w', 'c', 'm', 'y'
2. Color names: 'red', 'green', 'blue', 'cyan', etc.
3. Hexadecimal: '#FF5733', '#3498DB', etc.
4. RGB tuples: (1.0, 0.5, 0.0) - values between 0 and 1
5. RGBA tuples: (1.0, 0.5, 0.0, 0.7) - last value is transparency

7.2 Common Color Codes

Code	Color	Code	Color
'r'	Red	'b'	Blue
'g'	Green	'c'	Cyan
'y'	Yellow	'm'	Magenta
'k'	Black	'w'	White

Table 6: Single Character Color Codes

7.3 Example 18: Using Different Color Methods

Example 18: Color Variety Demonstration

```
1 import matplotlib.pyplot as plt
2
3 # Data
4 categories = ['A', 'B', 'C', 'D', 'E']
5 values = [25, 40, 30, 55, 45]
6
7 # Different ways to specify colors
8 colors = [
9     'red',                      # Color name
10    '#3498DB',                  # Hex code (blue)
11    (0.2, 0.8, 0.2),            # RGB tuple (green)
12    'gold',                     # Named color
13    '#E74C3C'                  # Hex code (red-orange)
14 ]
15
16 # Create bar chart
17 plt.figure(figsize=(10, 6))
18 bars = plt.bar(categories, values, color=colors, width
19                  =0.6,
20                  edgecolor='black', linewidth=1.5)
21
22 # Add value labels on bars
23 for i, (bar, val) in enumerate(zip(bars, values)):
24     plt.text(bar.get_x() + bar.get_width()/2, val + 1,
25               str(val), ha='center', fontweight='bold',
26               fontsize=11)
27
28 # Customize
29 plt.title('Different Color Specification Methods',
30           fontsize=14, fontweight='bold')
31 plt.xlabel('Categories', fontsize=12)
32 plt.ylabel('Values', fontsize=12)
33 plt.ylim(0, 65)
34 plt.grid(axis='y', alpha=0.3)
35
36 plt.tight_layout()
37 plt.show()
38
39 print("Color demonstration chart created!")
```

Output/Visualization

Creates a colorful bar chart demonstrating:

- 5 different methods of specifying colors
- Black borders around bars for definition
- Value labels centered on top of each bar
- Category D has highest value (55)
- Mix of color names, hex codes, and RGB tuples
- Professional appearance with grid and labels

7.4 Color Palettes and Gradients

Example 19: Using Color Maps

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Data
5 categories = ['Product ' + str(i) for i in range(1, 11)]
6 sales = [45, 67, 23, 89, 34, 56, 78, 45, 90, 67]
7
8 # Create color gradient using colormap
9 colors = plt.cm.viridis(np.linspace(0, 1, len(categories)))
10
11 # Create bar chart
12 plt.figure(figsize=(12, 6))
13 bars = plt.bar(categories, sales, color=colors, edgecolor='black')
14
15 # Customize
16 plt.title('Sales Report with Gradient Colors',
17            fontsize=14, fontweight='bold')
18 plt.xlabel('Products', fontsize=12)
19 plt.ylabel('Sales (Units)', fontsize=12)
20 plt.xticks(rotation=45)
21 plt.grid(axis='y', alpha=0.3)
22
23 # Add colorbar to show gradient
24 sm = plt.cm.ScalarMappable(cmap='viridis',
25                             norm=plt.Normalize(vmin=0, vmax
26                             =len(categories)))
27 sm.set_array([])
28 cbar = plt.colorbar(sm, ax=plt.gca())
29 cbar.set_label('Product Number', rotation=270, labelpad
30 =20)
31
32 plt.tight_layout()
33 plt.show()
34
35 print("Gradient color chart created!")
```

Output/Visualization

Creates an attractive chart with:

- Gradient colors from purple to yellow (viridis colormap)
- Each bar gets progressively different color
- Colorbar on right showing the gradient scale
- Product 9 has highest sales (90 units)
- Rotated x-axis labels for better readability
- Professional gradient effect

7.5 Popular Colormaps

Colormap	Description & Use Case
viridis	Purple to yellow, perceptually uniform (default, best for most uses)
plasma	Purple to yellow-red, good for scientific data
coolwarm	Blue to red through white, good for diverging data
rainbow	Full spectrum, colorful but use carefully
Greys	Grayscale, good for printing
RdYlGn	Red-Yellow-Green, good for heatmaps

Table 7: Commonly Used Colormaps

7.6 Transparency and Alpha Values

Example 20: Using Transparency

```
1 import matplotlib.pyplot as plt
2
3 # Data for overlapping plots
4 x = list(range(1, 11))
5 y1 = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
6 y2 = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
7
8 # Create plot
9 plt.figure(figsize=(10, 6))
10
11 # Filled areas with transparency
12 plt.fill_between(x, y1, alpha=0.3, color='blue', label='Series 1')
13 plt.fill_between(x, y2, alpha=0.3, color='red', label='Series 2')
14
15 # Lines
16 plt.plot(x, y1, 'b-', linewidth=2, marker='o')
17 plt.plot(x, y2, 'r-', linewidth=2, marker='s')
18
19 # Customize
20 plt.title('Overlapping Data with Transparency',
21            fontsize=14, fontweight='bold')
22 plt.xlabel('X Values', fontsize=12)
23 plt.ylabel('Y Values', fontsize=12)
24 plt.legend(loc='upper left', fontsize=11)
25 plt.grid(True, alpha=0.2)
26
27 plt.tight_layout()
28 plt.show()
29
30 print("Transparency demonstration created!")
```

Output/Visualization

Creates a plot showing:

- Two lines with filled areas beneath them
- Transparent fills ($\alpha=0.3$) allow overlap to be visible
- Blue area for Series 1, red area for Series 2
- Overlap appears purple (mix of transparent blue and red)
- Solid lines with markers on top
- Clean visualization of overlapping data

Pro Tip**Color Selection Tips:**

- Use contrasting colors for different data series
- Consider colorblind-friendly palettes
- Avoid too many bright colors in one chart
- Use darker colors for emphasis
- Keep color schemes consistent across related charts
- Test how colors look in grayscale (for printing)
- Use transparency (alpha) when overlapping elements

8 Advanced Customization Techniques

8.1 Figure and Subplot Management

Example 21: Multiple Subplots

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Data
5 months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
6 sales = [15, 18, 22, 19, 25, 28]
7 expenses = [12, 13, 15, 14, 16, 17]
8 profit = [s - e for s, e in zip(sales, expenses)]
9
10 # Create figure with 3 subplots
11 fig, axes = plt.subplots(3, 1, figsize=(10, 12))
12
13 # Subplot 1: Sales
14 axes[0].plot(months, sales, 'g-o', linewidth=2)
15 axes[0].set_title('Monthly Sales', fontsize=12, fontweight='bold')
16 axes[0].set_ylabel('Sales (K Rs.)', fontsize=10)
17 axes[0].grid(True, alpha=0.3)
18
19 # Subplot 2: Expenses
20 axes[1].plot(months, expenses, 'r-s', linewidth=2)
21 axes[1].set_title('Monthly Expenses', fontsize=12,
                     fontweight='bold')
22 axes[1].set_ylabel('Expenses (K Rs.)', fontsize=10)
23 axes[1].grid(True, alpha=0.3)
24
25 # Subplot 3: Profit
26 axes[2].bar(months, profit, color='blue', alpha=0.7)
27 axes[2].set_title('Monthly Profit', fontsize=12,
                     fontweight='bold')
28 axes[2].set_xlabel('Month', fontsize=10)
29 axes[2].set_ylabel('Profit (K Rs.)', fontsize=10)
30 axes[2].grid(True, alpha=0.3)
31
32 # Main title
33 fig.suptitle('Financial Report - First Half 2024',
               fontsize=16, fontweight='bold', y=0.995)
34
35
36 plt.tight_layout()
37 plt.show()
38
39 print("Multiple subplots created!")
```

Output/Visualization

Creates a comprehensive dashboard with:

- Three vertically stacked subplots
- Top: Line plot of sales (green, increasing trend)
- Middle: Line plot of expenses (red, slight increase)
- Bottom: Bar chart of profit (blue, showing profit = sales - expenses)
- Each subplot has its own title and labels
- Main title at the top spanning all subplots
- Consistent grid across all three

8.2 Saving Figures

Example 22: Saving Plots to Files

```

1 import matplotlib.pyplot as plt
2
3 # Create a simple plot
4 categories = ['A', 'B', 'C', 'D']
5 values = [23, 45, 56, 78]
6
7 plt.figure(figsize=(8, 6))
8 plt.bar(categories, values, color='skyblue')
9 plt.title('Sample Data Visualization')
10 plt.xlabel('Categories')
11 plt.ylabel('Values')
12 plt.grid(axis='y', alpha=0.3)
13
14 # Save in different formats
15 plt.savefig('chart.png', dpi=300, bbox_inches='tight')
16 plt.savefig('chart.pdf', bbox_inches='tight')
17 plt.savefig('chart.jpg', dpi=200, bbox_inches='tight')
18
19 # Display
20 plt.show()
21
22 print("Chart saved in PNG, PDF, and JPG formats!")
23 print("- chart.png (high resolution: 300 dpi)")
24 print("- chart.pdf (vector format)")
25 print("- chart.jpg (medium resolution: 200 dpi)")
```

Output/Visualization

Creates and saves a chart as:

- PNG file (300 dpi) - best for presentations
- PDF file - best for documents (scalable)
- JPG file (200 dpi) - smaller file size

All files saved with tight bounding box (no extra whitespace)

Displays: Success messages for all three formats

8.3 Saving Options

Parameter	Description
<code>dpi</code>	Dots per inch (resolution): 72 (screen), 150 (print), 300 (high quality)
<code>bbox_inches</code>	'tight' removes extra whitespace
<code>transparent</code>	True makes background transparent
<code>facecolor</code>	Background color
<code>format</code>	File format: 'png', 'pdf', 'jpg', 'svg'

Table 8: savefig() Parameters

9 Integrating Everything - Complete Examples

9.1 Example 23: Comprehensive Sales Dashboard

Example 23: Complete Business Dashboard

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Quarterly sales data
5 quarters = ['Q1', 'Q2', 'Q3', 'Q4']
6 product_a = [120, 150, 180, 200]
7 product_b = [90, 110, 140, 170]
8 product_c = [75, 85, 95, 110]
9
10 # Market share data
11 products = ['Product A', 'Product B', 'Product C', 'Others']
12 market = [35, 28, 20, 17]
13
14 # Create figure with 4 subplots in a 2x2 grid
15 fig = plt.figure(figsize=(14, 10))
16
17 # Subplot 1: Line plot - Product A trend
18 ax1 = plt.subplot(2, 2, 1)
19 ax1.plot(quarters, product_a, 'go-', linewidth=3,
20         markersize=10)
21 ax1.set_title('Product A Sales Trend', fontsize=12,
22               fontweight='bold')
23 ax1.set_xlabel('Quarter')
24 ax1.set_ylabel('Sales (Units)')
25 ax1.grid(True, alpha=0.3)
26 ax1.set_ylim([100, 220])
27
28 # Subplot 2: Grouped bar chart - All products comparison
29 ax2 = plt.subplot(2, 2, 2)
30 x = np.arange(len(quarters))
31 width = 0.25
32 ax2.bar(x - width, product_a, width, label='Product A',
33          color='green')
34 ax2.bar(x, product_b, width, label='Product B', color='blue')
35 ax2.bar(x + width, product_c, width, label='Product C',
36          color='orange')
37 ax2.set_title('Quarterly Sales Comparison', fontsize=12,
38               fontweight='bold')
39 ax2.set_xlabel('Quarter')
40 ax2.set_ylabel('Sales (Units)')
41 ax2.set_xticks(x)
42 ax2.set_xticklabels(quarters)
43 ax2.legend()
44 ax2.grid(axis='y', alpha=0.3)
45
46 # Subplot 3: Pie chart - Market share
47 ax3 = plt.subplot(2, 2, 3)
48 colors = ['#2ecc71', '#3498db', '#f39c12', '#95a5a6']
49 explode = (0.1, 0, 0, 0)
```

Output/Visualization

Creates a professional 2x2 dashboard showing:

Top Left: Line plot with green markers showing Product A growth from 120 to 200 units

Top Right: Grouped bars comparing three products across quarters

Bottom Left: Pie chart with Product A having largest share (35%, exploded slice)

Bottom Right: Stacked area showing total sales increasing from 285 to 480 units

All charts have:

- Professional styling with grids
- Proper titles, labels, and legends
- Color-coordinated (green for Product A throughout)
- Saved as high-resolution PNG file

9.2 Example 24: Student Performance Analysis

Example 24: Educational Data Visualization

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 # Student performance data
5 students = ['Alice', 'Bob', 'Charlie', 'David', 'Eve']
6 math = [85, 78, 92, 88, 95]
7 science = [88, 82, 90, 85, 92]
8 english = [80, 85, 88, 90, 87]
9
10 # Average marks calculation
11 averages = [(m + s + e) / 3 for m, s, e in zip(math,
12               science, english)]
13
14 # Subject-wise class average
15 subjects = ['Math', 'Science', 'English']
16 class_avg = [
17     np.mean(math),
18     np.mean(science),
19     np.mean(english)
20 ]
21
22 # Create figure
23 fig = plt.figure(figsize=(14, 10))
24
25 # Plot 1: Individual student performance (grouped bar)
26 ax1 = plt.subplot(2, 2, 1)
27 x = np.arange(len(students))
28 width = 0.25
29 ax1.bar(x - width, math, width, label='Math', color='#e74c3c')
30 ax1.bar(x, science, width, label='Science', color='#3498db')
31 ax1.bar(x + width, english, width, label='English', color='#2ecc71')
32 ax1.set_xlabel('Students', fontweight='bold')
33 ax1.set_ylabel('Marks', fontweight='bold')
34 ax1.set_title('Subject-wise Student Performance',
35                 fontsize=12, fontweight='bold')
36 ax1.set_xticks(x)
37 ax1.set_xticklabels(students, rotation=45)
38 ax1.legend()
39 ax1.grid(axis='y', alpha=0.3)
40
41 # Plot 2: Overall student ranking (horizontal bar)
42 ax2 = plt.subplot(2, 2, 2)
43 sorted_indices = np.argsort(averages)
44 sorted_students = [students[i] for i in sorted_indices]
45 sorted_averages = [averages[i] for i in sorted_indices]
46 colors_rank = plt.cm.RdYlGn(np.linspace(0.3, 0.9, len(
47     students))) 46
48 ax2.banh(sorted_students, sorted_averages, color=
49           colors_rank)
50 ax2.set_xlabel('Average Marks', fontweight='bold')

```

Output/Visualization

Creates a comprehensive educational dashboard with:

Top Left: Grouped bars showing each student's marks in three subjects

Top Right: Horizontal bars ranking students by average (gradient colors from red to green)

Bottom Left: Pie chart showing how class average is distributed across subjects

Bottom Right: Line plots showing each student's performance trend across subjects

Console output:

Class Averages:

Math: 87.60

Science: 87.40

English: 86.00

Top performer: Eve (Average: 91.33)

Dashboard saved as 'student_performance.png'

10 Practice Exercises

Practice Exercises

Basic Visualization Exercises:

1. Create a line plot showing the temperature changes in your city over 7 days. Add appropriate title, labels, and grid.
2. Make a bar graph comparing the number of students in 5 different departments. Use different colors for each bar.
3. Create a pie chart showing your monthly expenses breakdown (Rent, Food, Transport, Entertainment, Others). Include percentages.
4. Plot a line graph showing monthly sales data for 6 months. Customize line color, style, and add markers.
5. Create a horizontal bar chart showing the top 5 most popular programming languages with their usage percentages.

Intermediate Customization Exercises:

6. Create a grouped bar chart comparing sales of three products (A, B, C) over four quarters. Add legend and grid.
7. Make a stacked bar chart showing income and expenses for 6 months. Calculate and display profit.
8. Create a pie chart with one slice exploded showing student grade distribution. Use custom colors and shadows.
9. Plot two lines on the same graph: one for temperature and one for humidity over a week. Use different colors and styles.
10. Create a bar graph with gradient colors showing sales figures. Add value labels on top of each bar.

Advanced Integration Exercises:

11. Create a 2x2 subplot figure showing:
 - Line plot of daily COVID cases
 - Bar chart of cases by age group
 - Pie chart of vaccination status
 - Stacked area of recovered vs active cases
12. Build a sales dashboard with:
 - Monthly sales trend (line plot)
 - Product-wise comparison (grouped bar)
 - Market share (pie chart)
 - Save as high-resolution PNG 49
13. Create a weather visualization showing:
 - Temperature variation (line plot with markers)

11 Quick Reference Guide

11.1 Matplotlib Essentials Cheat Sheet

Basic Plotting

```

1 import matplotlib.pyplot as plt
2
3 # Line Plot
4 plt.plot(x, y, color='blue', linewidth=2, marker='o')
5
6 # Bar Chart
7 plt.bar(x, y, color='green', width=0.6)
8
9 # Horizontal Bar
10 plt.bardh(x, y, color='red')
11
12 # Pie Chart
13 plt.pie(values, labels=labels, autopct='%1.1f%%')
14
15 # Always show the plot
16 plt.show()
17
18 # Save plot
19 plt.savefig('filename.png', dpi=300)

```

Customization Commands

```

1 # Titles and Labels
2 plt.title('My Chart Title', fontsize=14, fontweight='bold')
3 plt.xlabel('X Axis Label', fontsize=12)
4 plt.ylabel('Y Axis Label', fontsize=12)
5
6 # Legend
7 plt.legend(['Series 1', 'Series 2'], loc='upper right')
8
9 # Grid
10 plt.grid(True, linestyle='--', alpha=0.5)
11
12 # Figure Size
13 plt.figure(figsize=(10, 6))
14
15 # Axis Limits
16 plt.xlim([0, 10])
17 plt.ylim([0, 100])
18
19 # Tight Layout (prevents label cutoff)
20 plt.tight_layout()

```

11.2 Common Patterns

Complete Plot Template

```
1 import matplotlib.pyplot as plt
2
3 # Data
4 x = [1, 2, 3, 4, 5]
5 y = [10, 20, 15, 25, 30]
6
7 # Create figure
8 plt.figure(figsize=(10, 6))
9
10 # Plot
11 plt.plot(x, y, marker='o', linewidth=2, color='blue',
12           label='Data')
13
14 # Customize
15 plt.title('My Chart Title', fontsize=14, fontweight='bold')
16 plt.xlabel('X Label', fontsize=12)
17 plt.ylabel('Y Label', fontsize=12)
18 plt.legend()
19 plt.grid(True, alpha=0.3)
20
21 # Display
22 plt.tight_layout()
23 plt.show()
```

11.3 Best Practices Summary

Important Note

Visualization Best Practices:

DO:

- Always add title and axis labels
- Use appropriate chart type for your data
- Choose colors carefully (consider colorblind viewers)
- Add grid for better readability
- Include legend for multiple series
- Keep it simple and clean
- Test different sizes for best appearance
- Save in appropriate format (PNG for web, PDF for documents)

DON'T:

- Overload chart with too much data
- Use too many different colors
- Forget to label axes
- Make text too small or too large
- Use 3D effects unless necessary
- Ignore aspect ratio (keep plots proportional)

11.4 Common Issues and Solutions

Issue	Solution
Labels cut off	Use <code>plt.tight_layout()</code> before <code>plt.show()</code>
Legend blocks data	Use <code>loc='best'</code> or position outside: <code>bbox_to_anchor=(1, 1)</code>
Plot too small	Increase figure size: <code>plt.figure(figsize=(12, 8))</code>
Colors look similar	Use distinct colors or colormaps like 'tab10'
Too much white space	Use <code>bbox_inches='tight'</code> when saving
Text overlapping	Rotate labels: <code>plt.xticks(rotation=45)</code>
Multiple plots overlap	Create subplots: <code>plt.subplot(2, 2, 1)</code>

Table 9: Common Matplotlib Issues and Fixes

End of Unit IV Notes

Congratulations! You've completed the Data Visualization unit!

Practice regularly by creating different types of visualizations.

Remember: The best way to learn is by doing!

Prepared with care for beginner students

Happy Visualizing!