Scatter plot,Histogram and Pie chart by Mrittika Megaraj

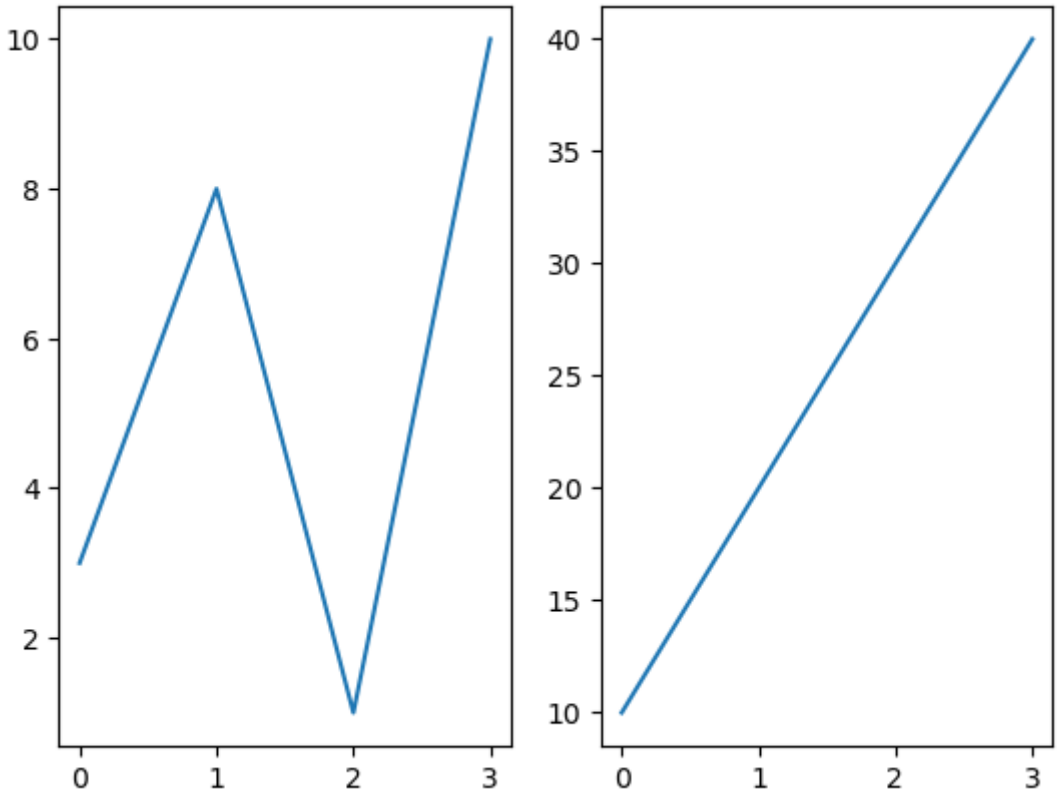
In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Matplotlib Subplot

Matplotlib Subplot:

- With the **subplot()** function we can draw multiple plots in one figure.

In [2]:
```
# Draw 2 plots

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```



subplot() Function:

- The **subplot()** function takes three arguments that describes the layout of the figure.
- The layout is organized in rows and columns, which are represented by the first and second argument.
- The third argument represents the index of the current plot.

- **Syntax:**

```
plt.subplot(rows, cols, index)
```

  - **rows:** Number of rows for organization of subplots.
  - **cols:** Number of columns for organization of subplots.

- The figure has 1 row, 2 columns, and this plot is the first plot.

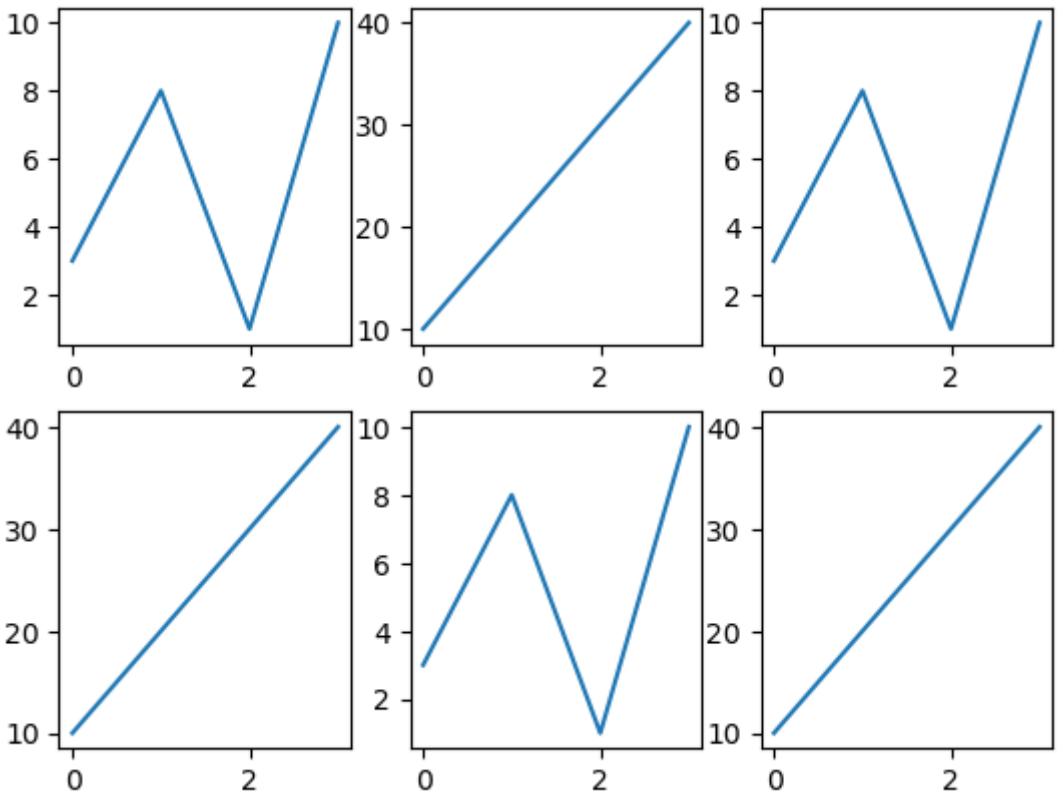
```
plt.subplot(1, 2, 1)
```

- The figure has 1 row, 2 columns, and this plot is the second plot.

```
plt.subplot(1, 2, 2)
```

In [3]:

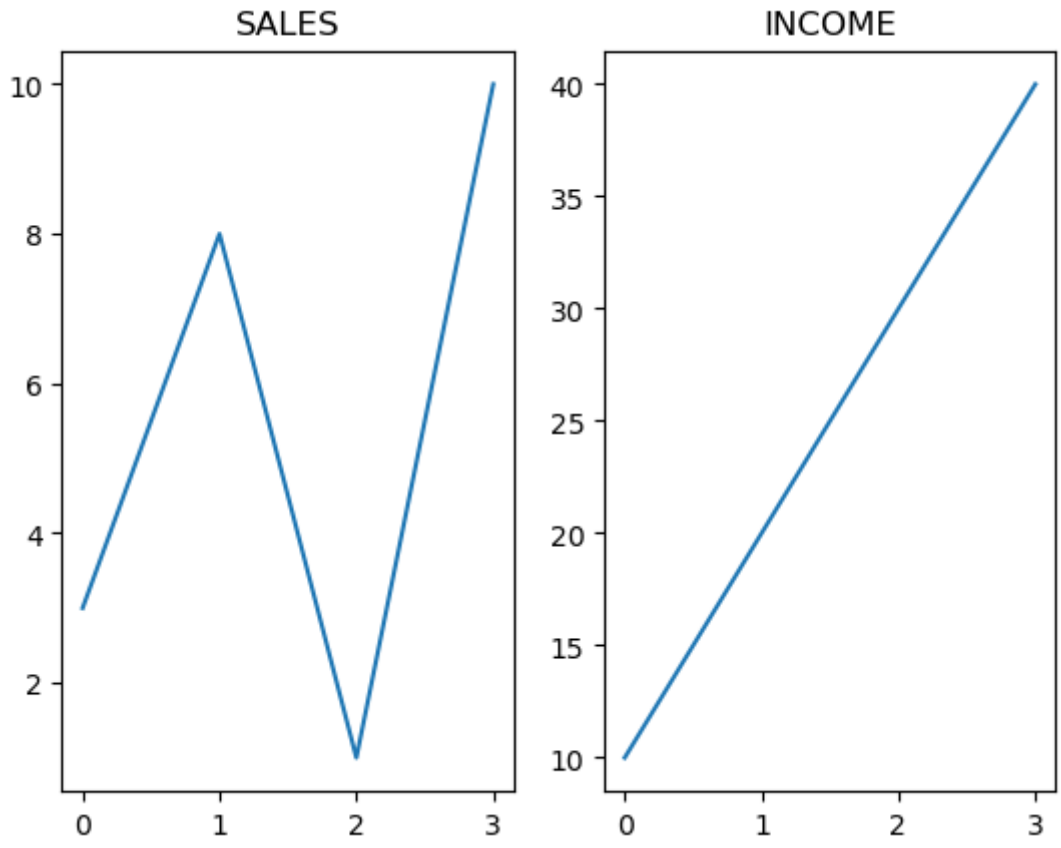
```
# Draw 6 plots

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```



Title each subplot:

- We can add a title to each plot with the **title()** function.

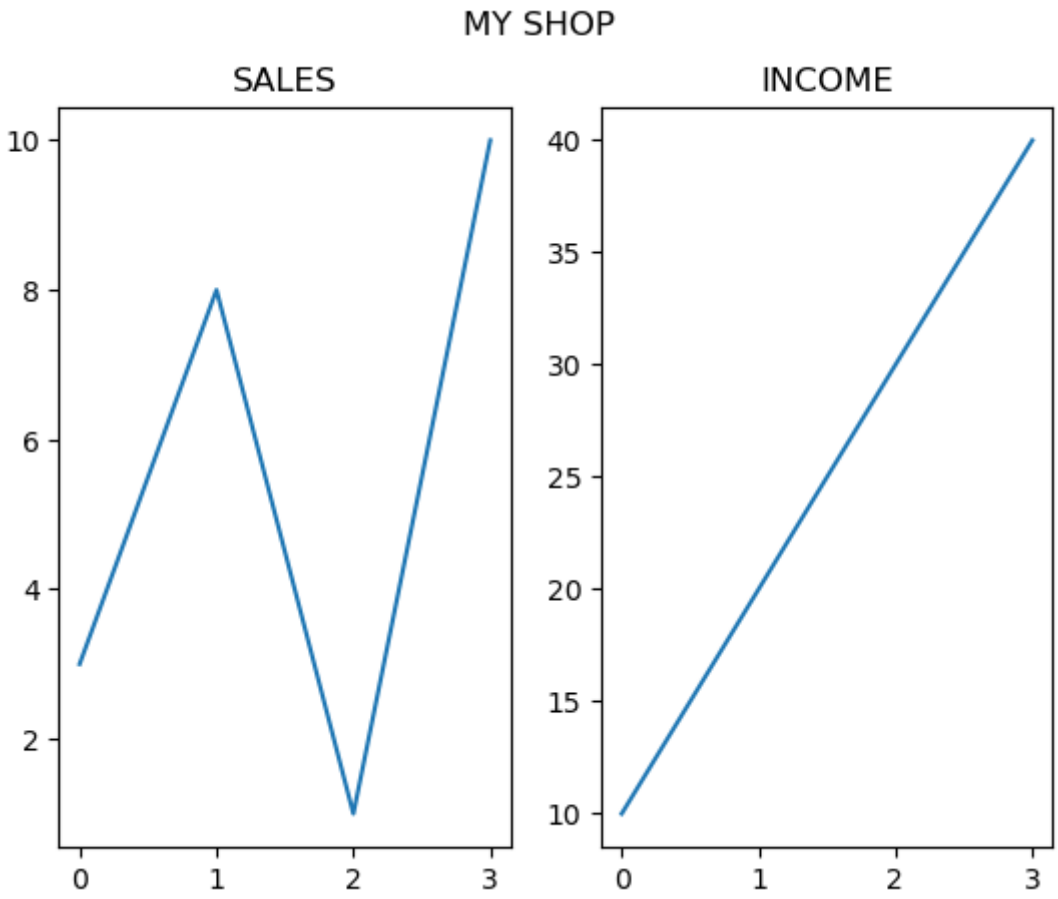
In [4]:
```
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.show()
```



Super Title:

- We can add a title to the entire figure with the **suptitle()** function.

In [5]:
```
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.suptitle("MY SHOP")
plt.show()
```

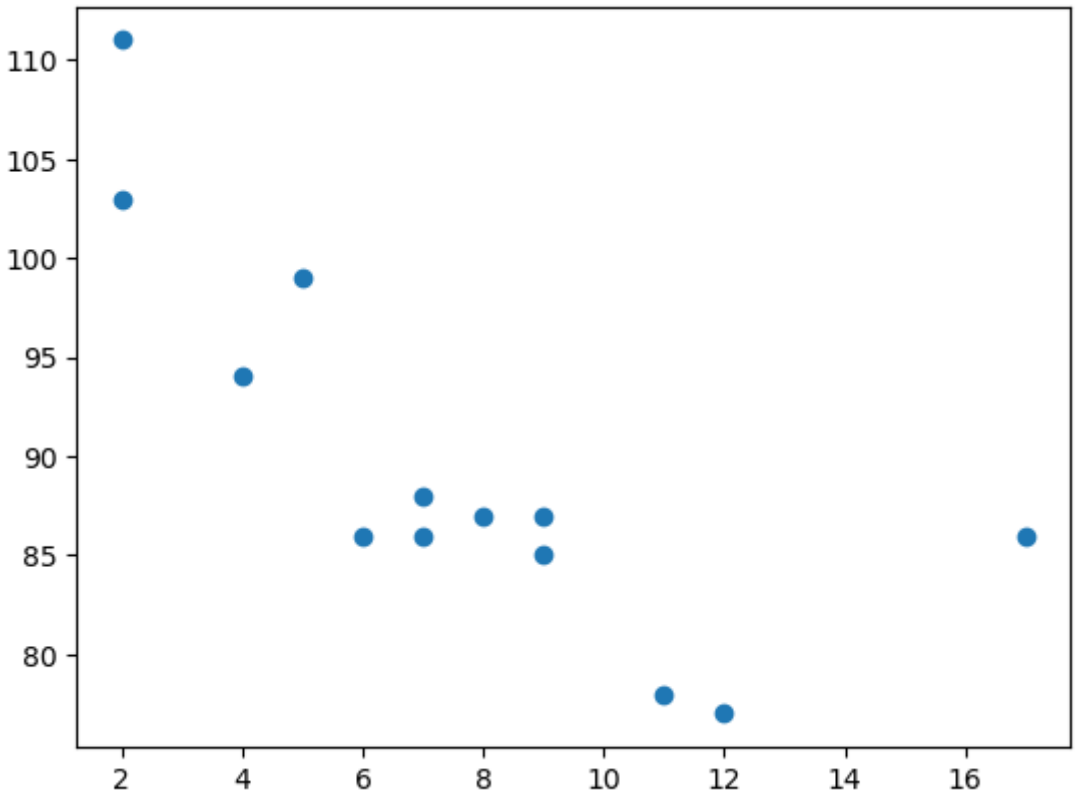


Matplotlib Scatter

Creating Scatter Plots:

- With Pyplot, we can use the **scatter()** function to draw a scatter plot.
- The **scatter()** function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

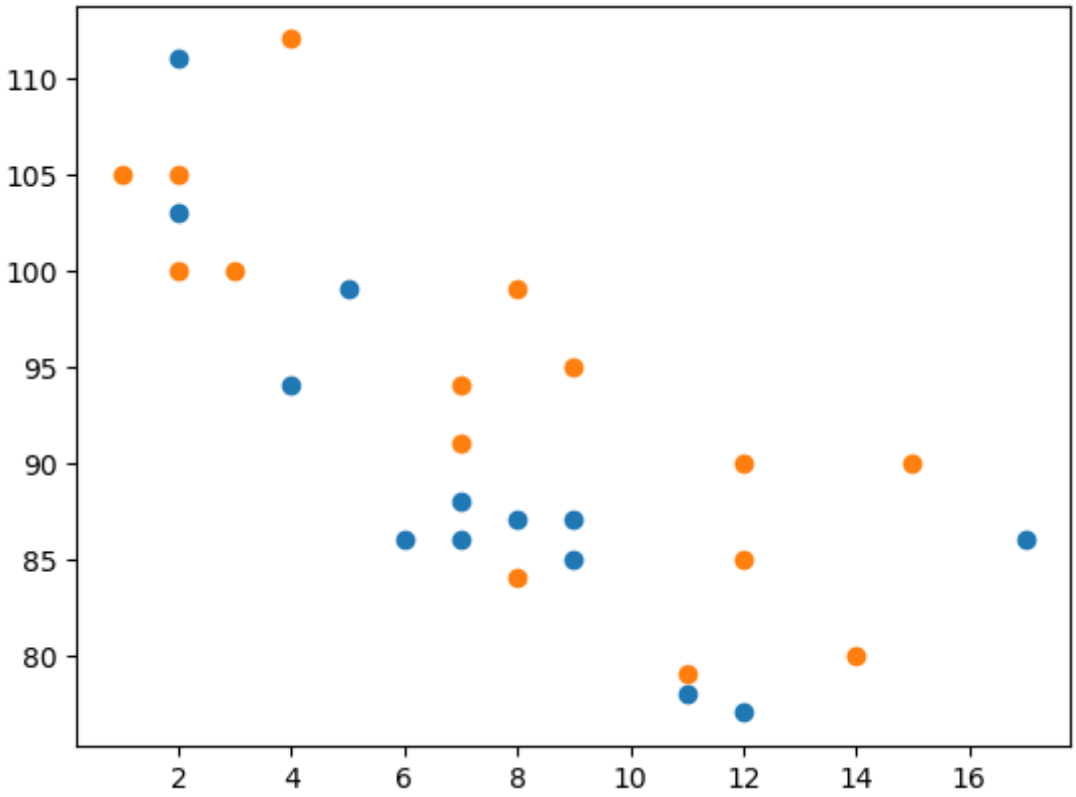
In [6]:
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```



Compare Plots:

- The two plots are plotted with two different colors, by default blue and orange.

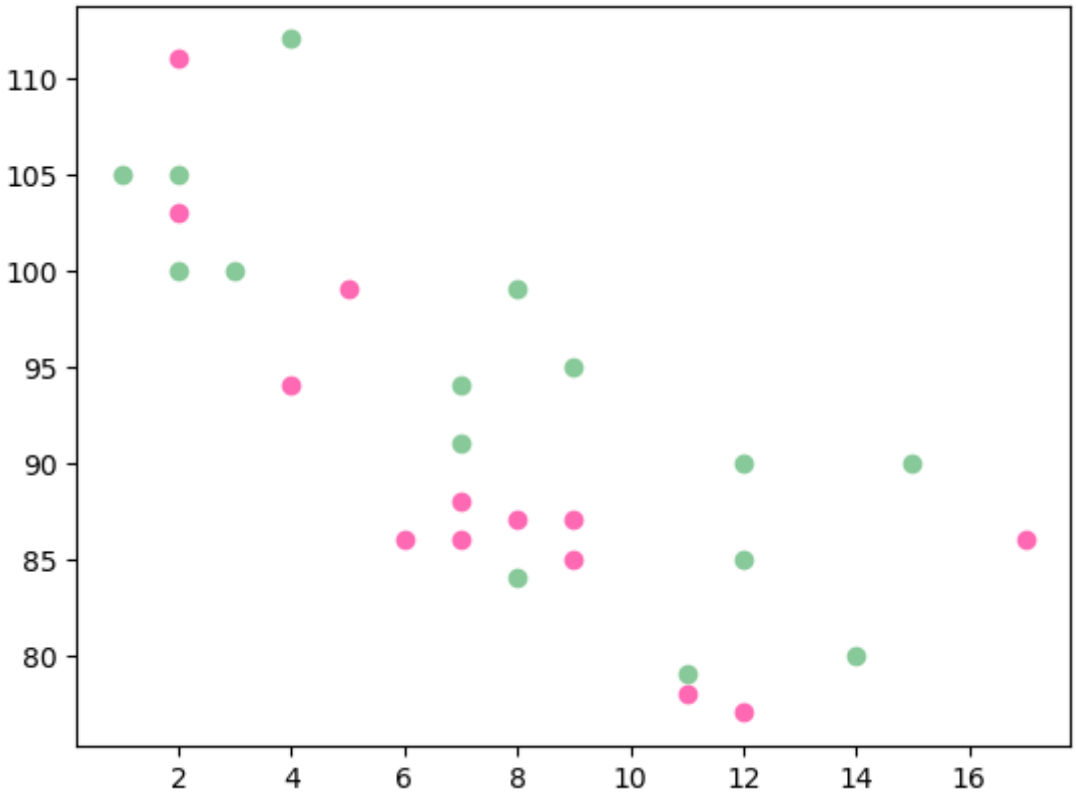
In [7]:
```
#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```



Colors:

- We can set your own color for each scatter plot with the **color** or the **c** argument.

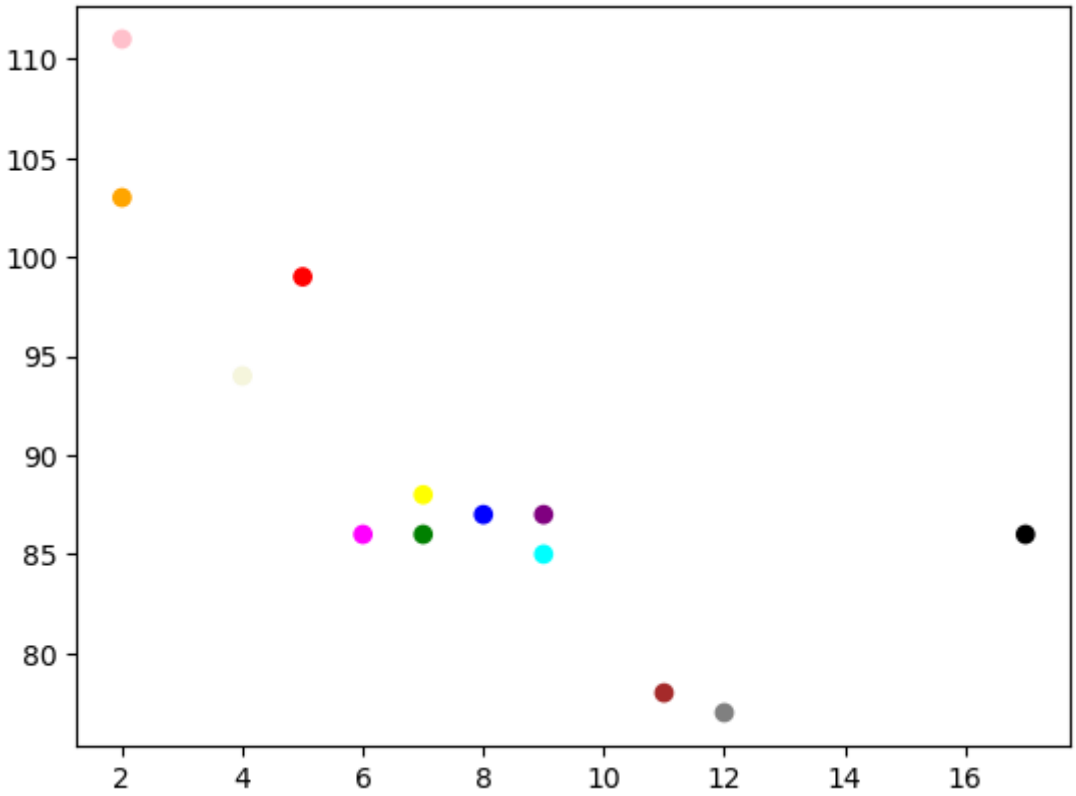
In [8]:
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')

x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#88c999')

plt.show()
```



Color Each Dot:

- We can even set a specific color for each dot by using an array of colors as value for the c argument.

In [9]:
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array(["red","green","blue","yellow","pink","black","orange","p
                   "beige","brown","gray","cyan","magenta"])

plt.scatter(x, y, c=colors)

plt.show()
```

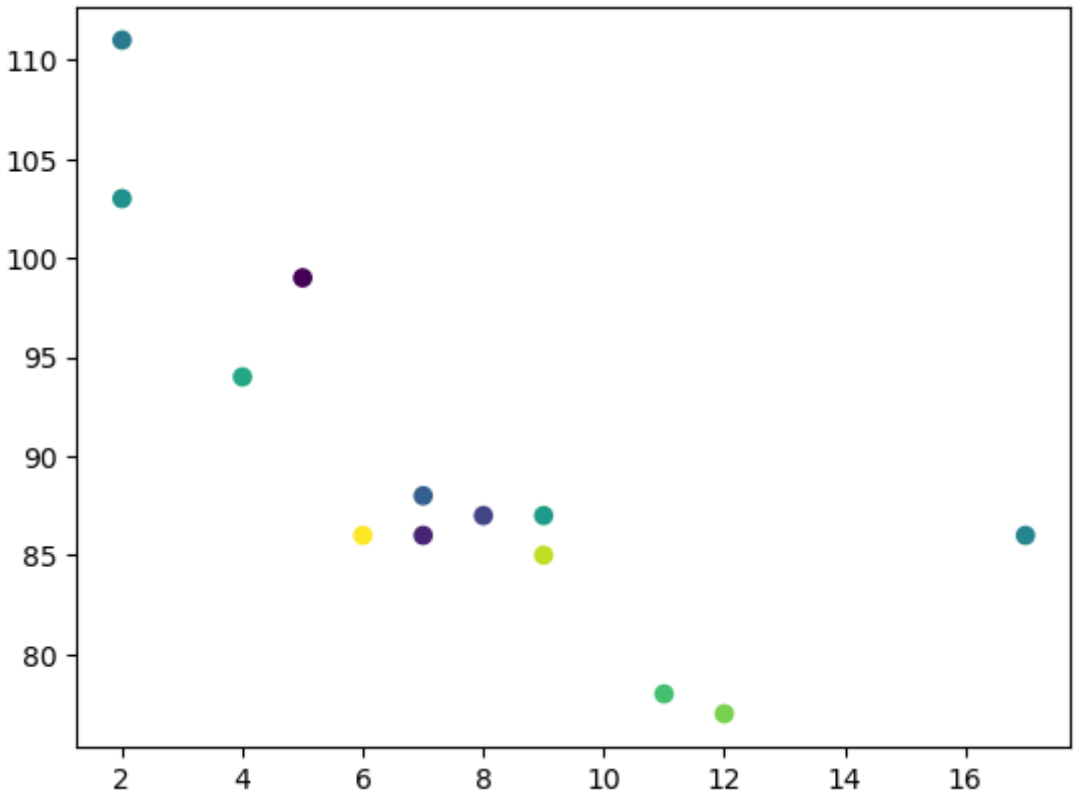


ColorMap:

- A colormap is like a list of colors, where each color has a value that ranges from 0 to 100.
- We can specify the colormap with the keyword argument **cmap** with the value of the colormap, in this case **'viridis'** which is one of the built-in colormaps available in Matplotlib.

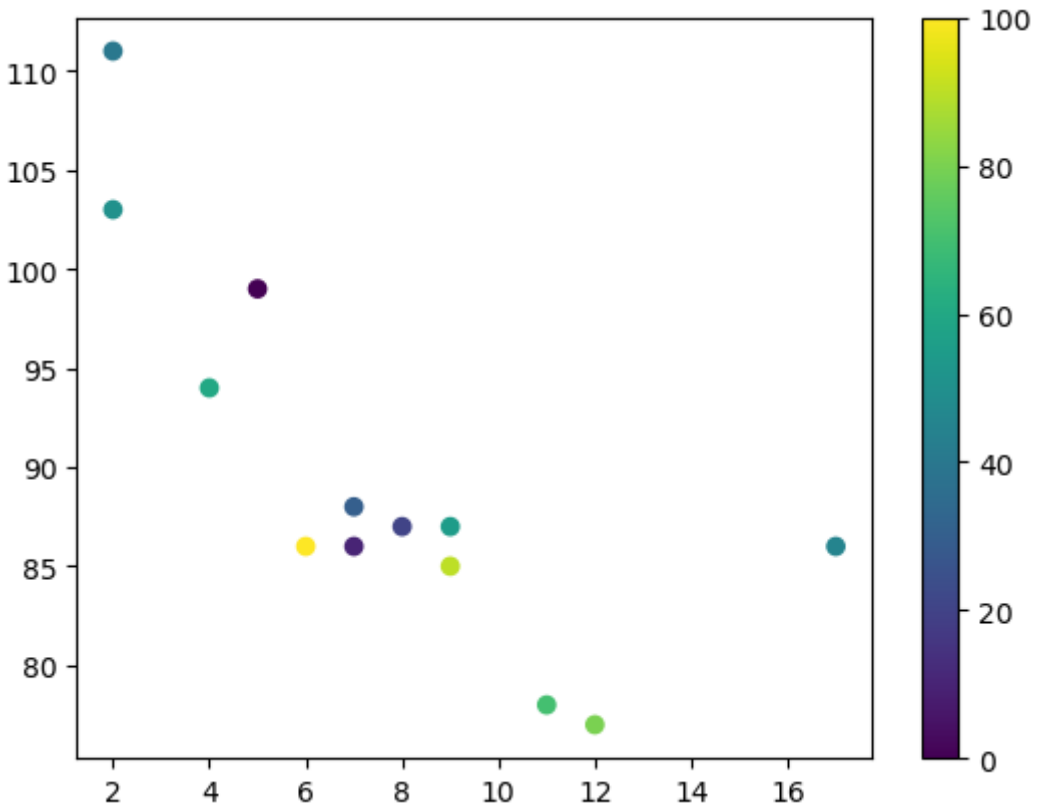
- **Names of colormaps:** winter, twilight, terrain, spring, rainbow, ocean, pink, magma, jet, copper, coolwarm, cool, cividis, bone, binary, Spectral, Set1, Reds, RdBu, Purples, Paired, Oranges, Greys, Greens, BrBG.

In [10]:
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.show()
```



- We can include the colormap in the drawing by including the **plt.colorbar()** statement.

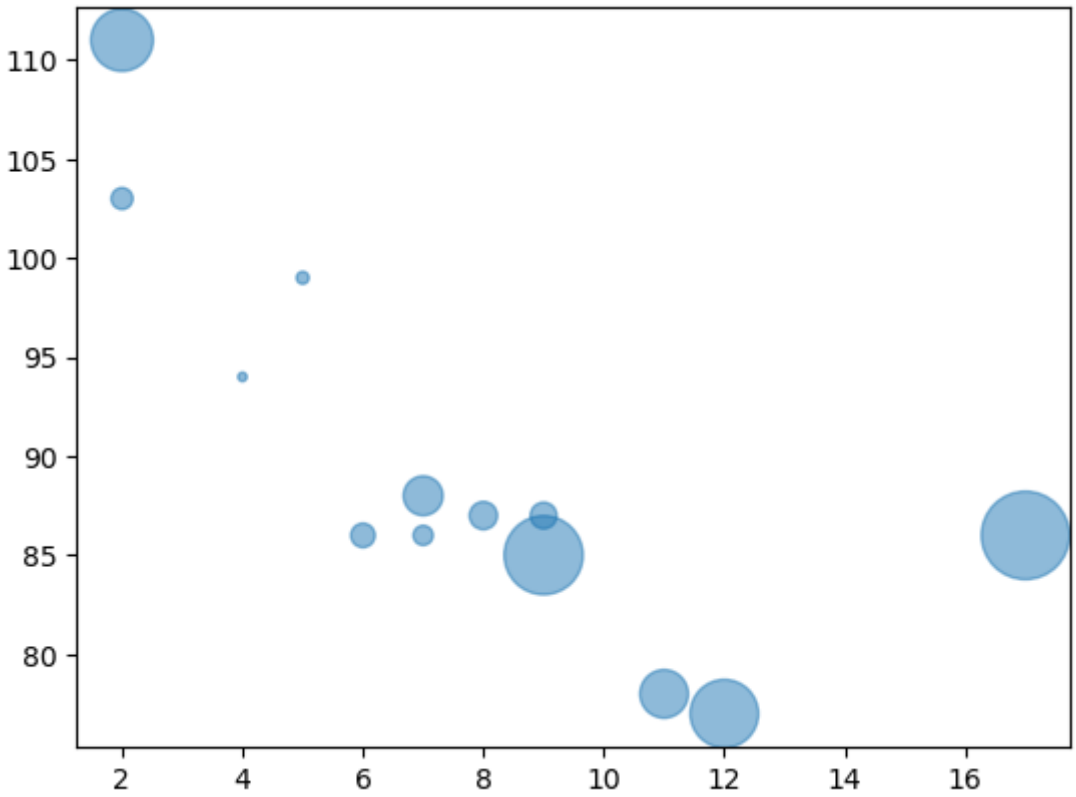
In [11]:
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.colorbar()

plt.show()
```



Alpha:

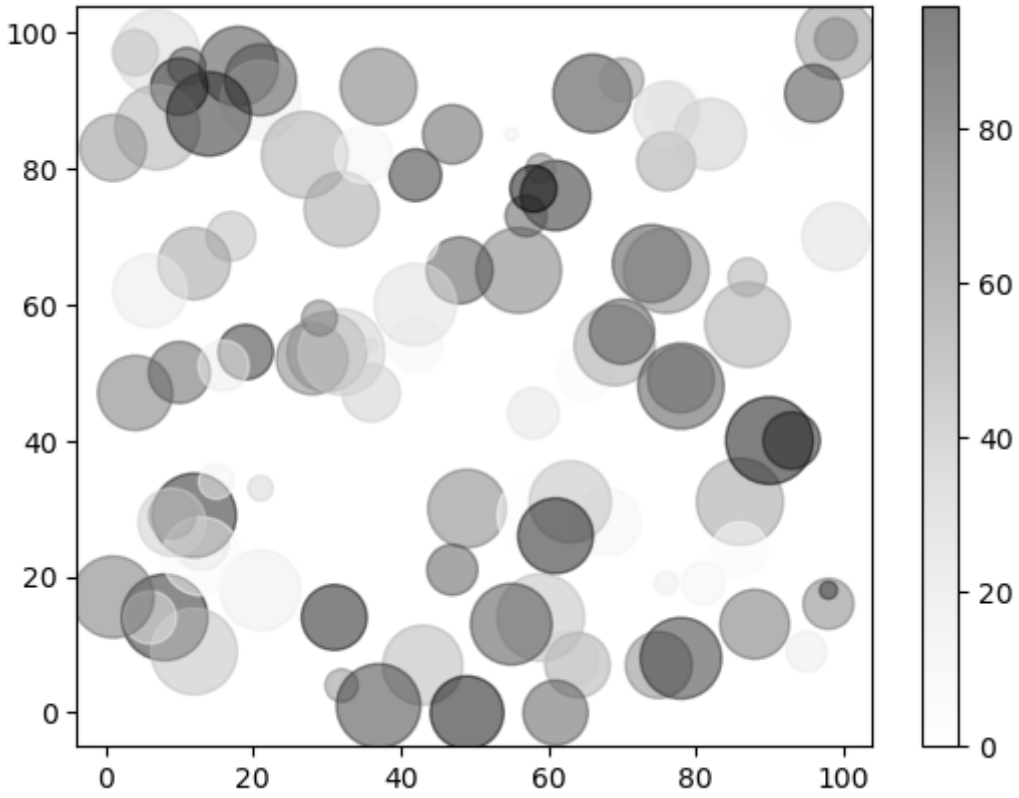
- We can adjust the transparency of the dots with the **alpha** argument.
- Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis.

In [12]:
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes, alpha=0.5)

plt.show()
```



Combine Color Size and Alpha:

In [13]:
```
x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='Greys')

plt.colorbar()

plt.show()
```

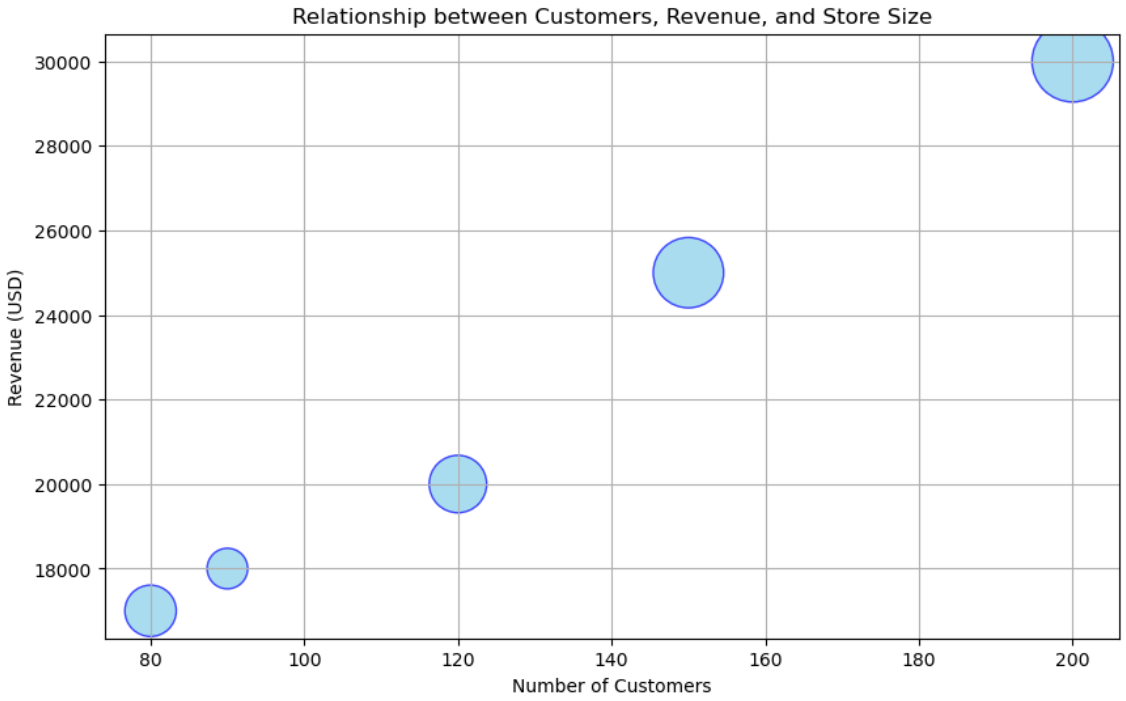


Use Cases:

- Investigating the relationship between study hours and exam scores.
- Analyzing the correlation between temperature and ice cream sales.

In [14]:

```
# Sample data for stores
stores = ['Store A', 'Store B', 'Store C', 'Store D', 'Store E']
customers = [120, 90, 150, 80, 200]
revenue = [20000, 18000, 25000, 17000, 30000]
store_size = [10, 5, 15, 8, 20]  # Represents the size of each store in 100

# Scale the store sizes for point sizes in the scatter plot
point_sizes = [size * 100 for size in store_size]

# Create a scatter plot with different point sizes
plt.figure(figsize=(10, 6))  # Set the figure size
plt.scatter(customers, revenue, s=point_sizes, c='skyblue', alpha=0.7, edge

# Add labels, a title, and a legend
plt.xlabel('Number of Customers')
plt.ylabel('Revenue (USD)')
plt.title('Relationship between Customers, Revenue, and Store Size')

# Display the plot
plt.grid(True)  # Add a grid for better readability
plt.show()
```



Matplotlib Histograms

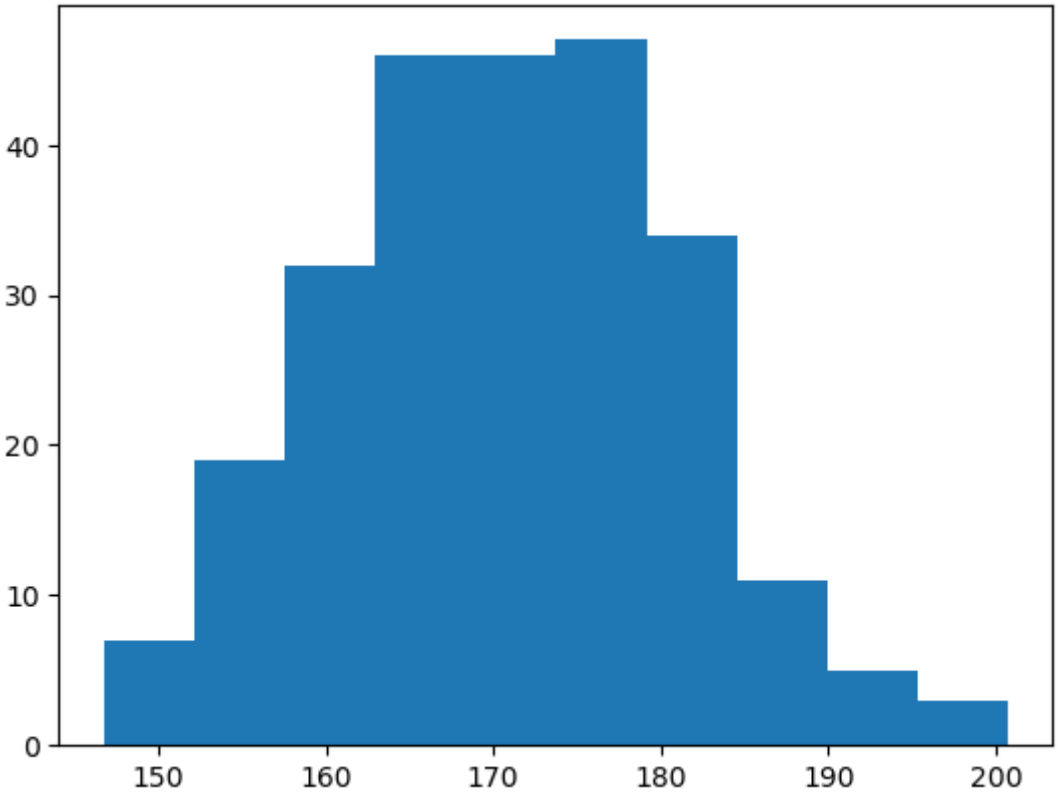
Histogram

- A histogram is a graph showing **frequency** distributions.
- It is a graph showing the number of observations within each given interval.

Create Histogram:

- In Matplotlib, we use the **hist()** function to create histograms.
- The **hist()** function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

In [15]:
```
x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```

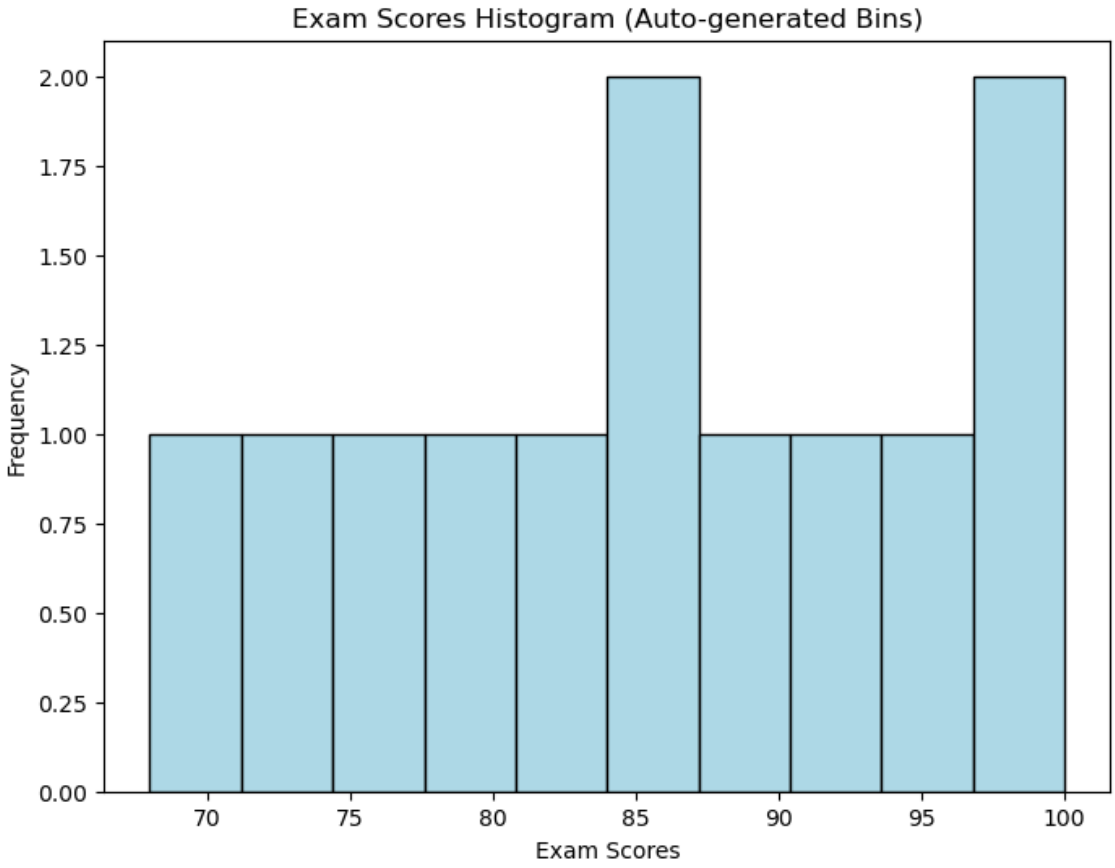


Use Cases:

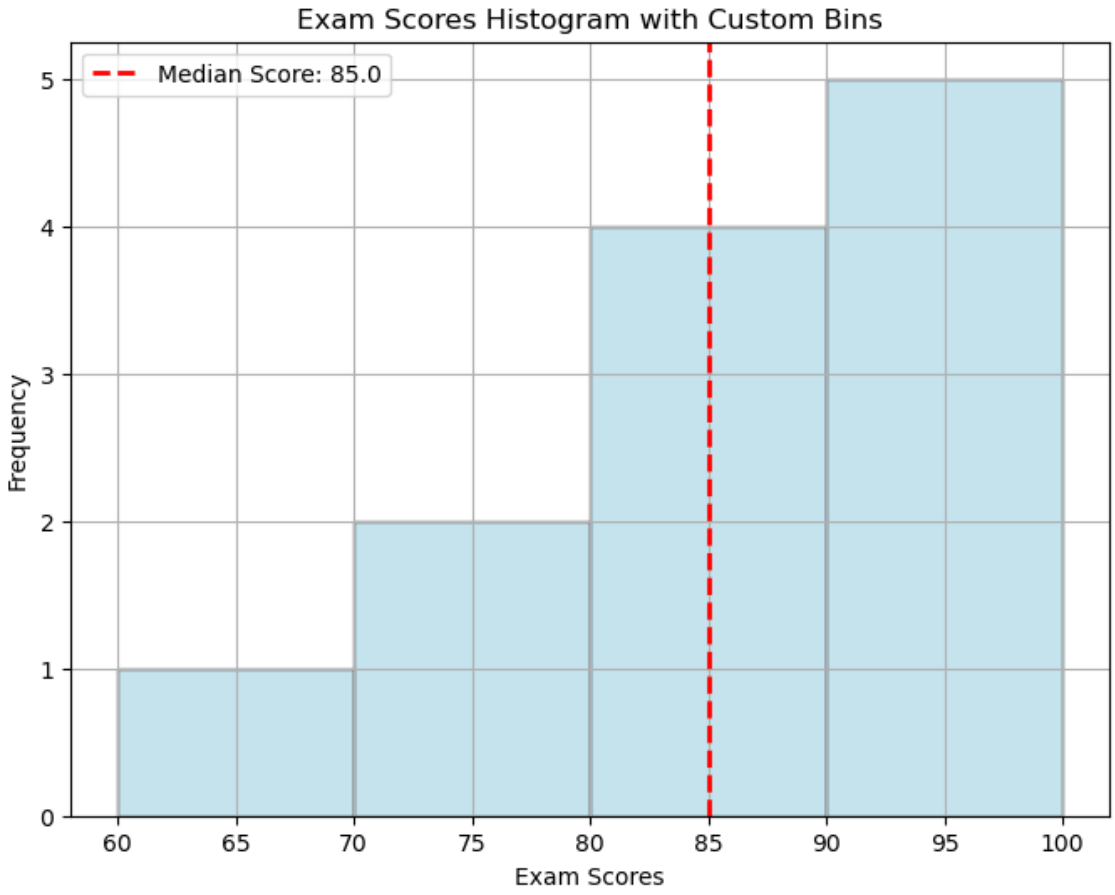
- Analyzing age distribution in a population.
- Examining exam score distribution in a classroom.

In [16]:

```
# Sample exam scores data
exam_scores = [68, 72, 75, 80, 82, 84, 86, 90, 92, 95, 98, 100]

# Create a histogram without specifying bin ranges
plt.figure(figsize=(8, 6))  # Set the figure size
plt.hist(exam_scores, color='lightblue', edgecolor='black')

# Add labels and a title
plt.xlabel('Exam Scores')
plt.ylabel('Frequency')
plt.title('Exam Scores Histogram (Auto-generated Bins)')
plt.show()
```



In [17]:
```
# Sample exam scores data
exam_scores = [68, 72, 75, 80, 82, 84, 86, 90, 92, 95, 98, 100]

# Custom bin ranges
bin_ranges = [60, 70, 80, 90, 100]

# Create a histogram with custom bin ranges
plt.figure(figsize=(8, 6))  # Set the figure size
plt.hist(exam_scores, bins=bin_ranges, color='lightblue', edgecolor='black'

# Add labels and a title
plt.xlabel('Exam Scores')
plt.ylabel('Frequency')
plt.title('Exam Scores Histogram with Custom Bins')

# Calculate and add a median line
median_score = np.median(exam_scores)
plt.axvline(median_score, color='red', linestyle='dashed', linewidth=2, lab

# Add a legend
plt.legend()

# Display the plot
plt.grid(True)  # Add a grid for better readability
plt.show()
```



Matplotlib Pie Charts

Creating Pie Charts:

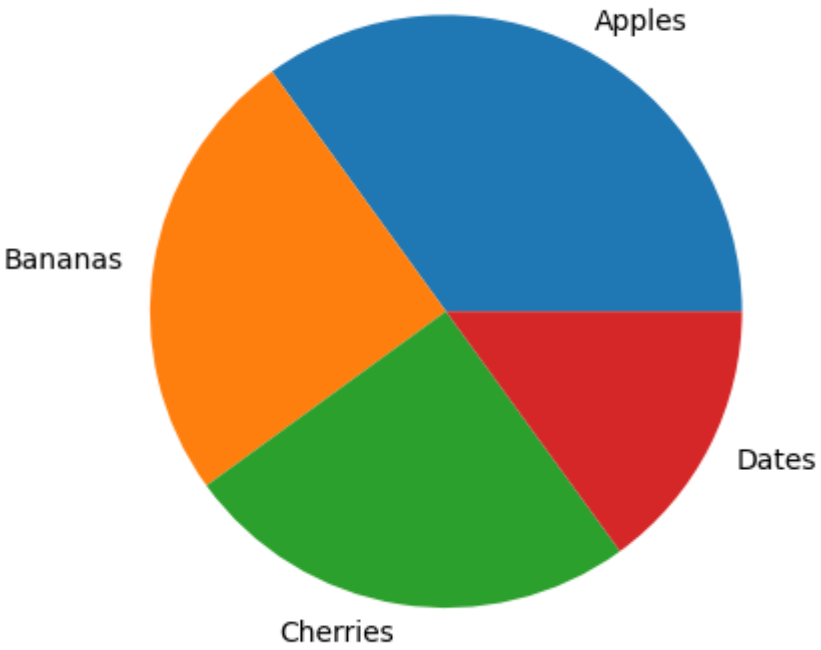
- With Pyplot, we can use the **pie()** function to draw pie charts.
- By default the plotting of the first wedge starts from the **x-axis** and moves **counterclockwise**.

In [18]:
```
y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```

Labels:

- Add labels to the pie chart with the **label** parameter.
- The **label** parameter must be an array with one label for each wedge

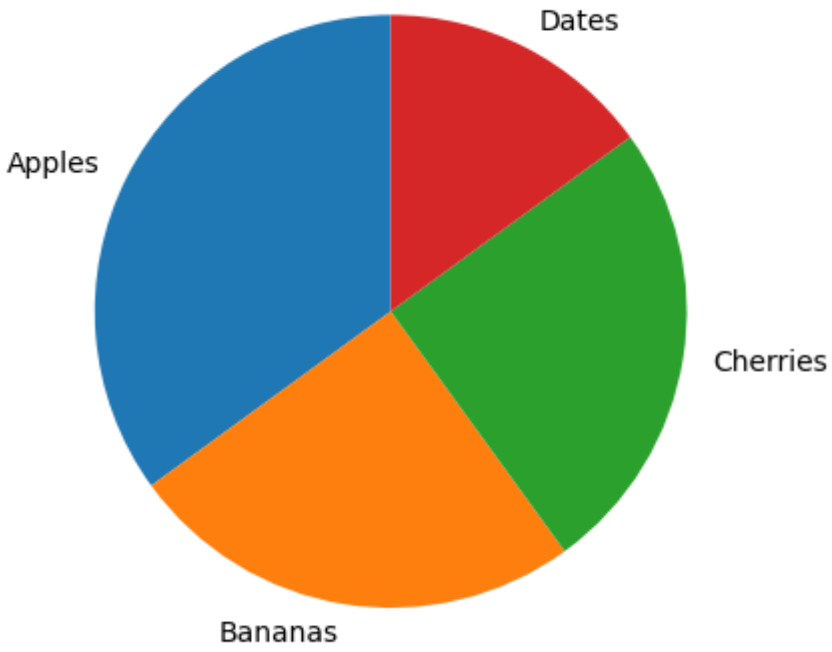
In [19]:
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```



Start Angle:

- As mentioned the default start angle is at the **x-axis**, but we can change the start angle by specifying a **startangle** parameter.
- The **startangle** parameter is defined with an angle in degrees, default angle is 0

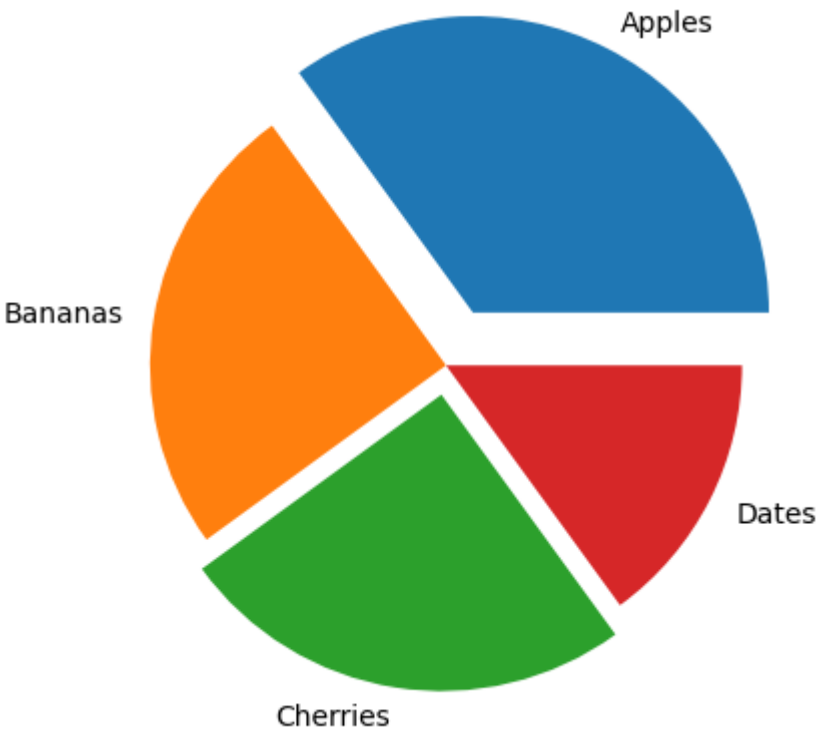
In [20]:
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle = 90)
plt.show()
```



Explode:

- The **explode** parameter, if specified, and not **None**, must be an array with one value for each wedge.
- Each value represents how far from the center each wedge is displayed.

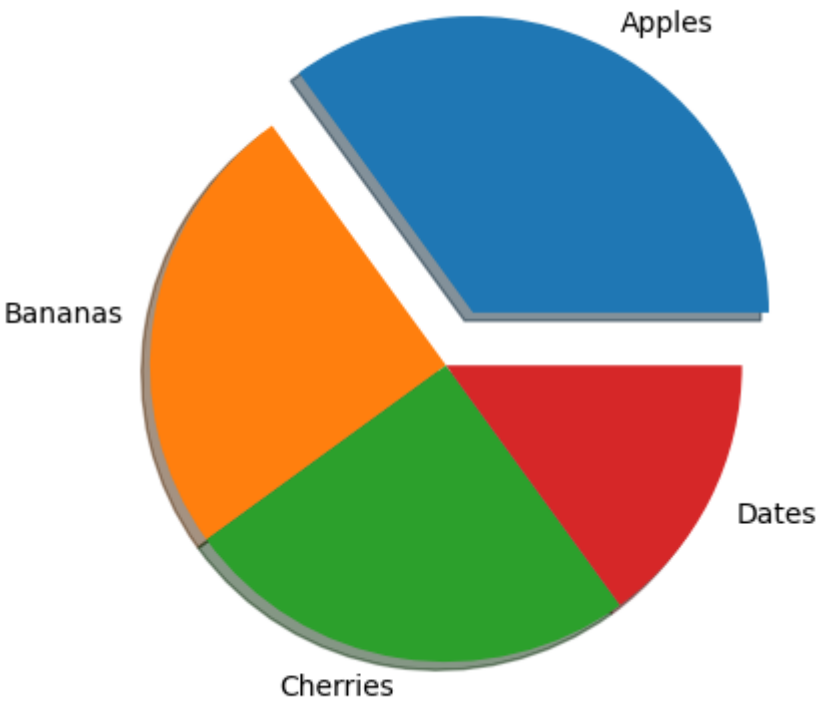
In [21]:
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0.1, 0]

plt.pie(y, labels = mylabels, explode = myexplode)
plt.show()
```



Shadow:

- Add a shadow to the pie chart by setting the **shadows** parameter to **True**.

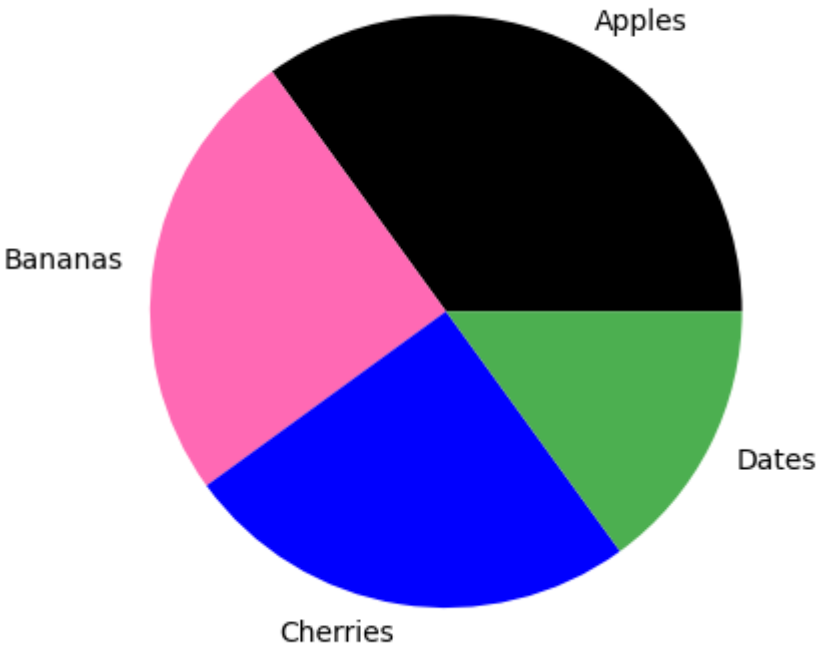
In [22]:
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
plt.show()
```



Colors:

- We can set the color of each wedge with the **colors** parameter.
- The **colors** parameter, if specified, must be an array with one value for each wedge.

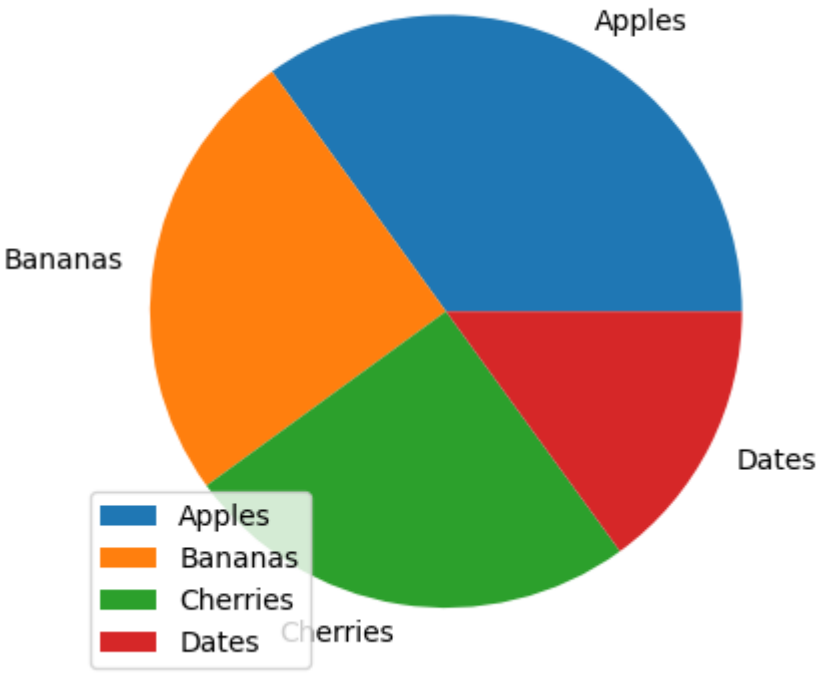
In [23]:
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]

plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```



Legend:

- To add a list of explanation for each wedge, use the **legend()** function.

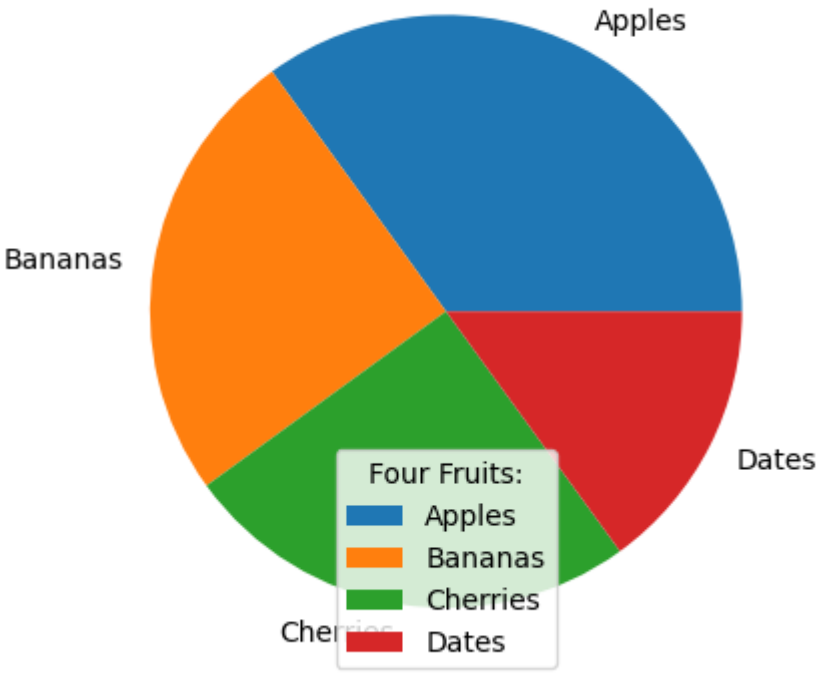
In [24]:
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend()
plt.show()
```



Legend With Header:

- To add a header to the legend, add the **title** parameter to the **legend** function.

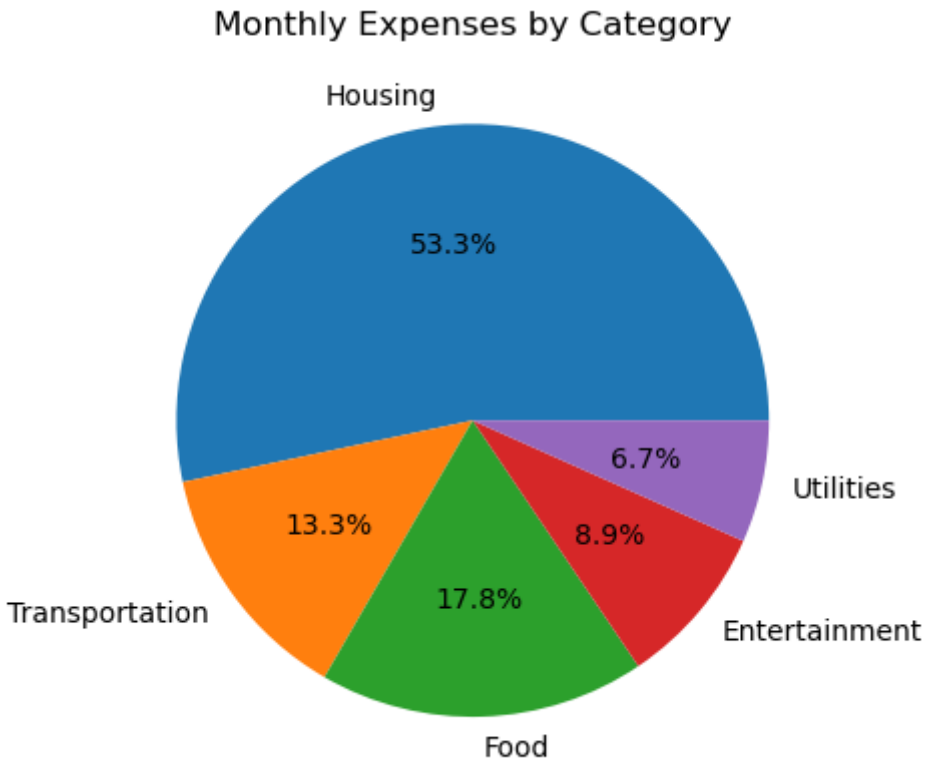
In [25]:
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend(title = "Four Fruits:")
plt.show()
```



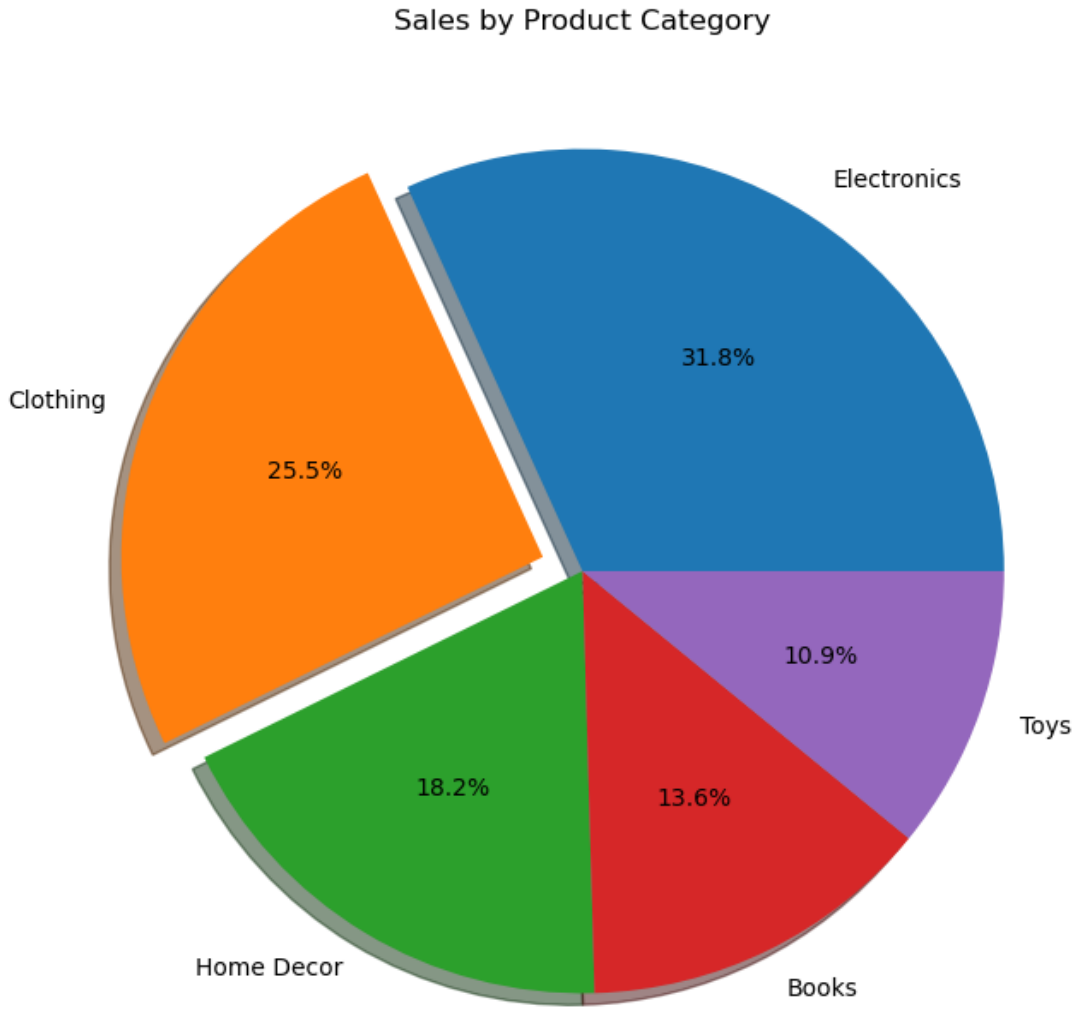
Use Cases:

- Displaying the distribution of a budget by expense categories.
- Showing the market share of various smartphone brands.

In [26]:
```
# Sample data (expenses)
categories = ['Housing', 'Transportation', 'Food', 'Entertainment', 'Utilit
expenses = [1200, 300, 400, 200, 150]

# Create a pie chart
plt.pie(expenses, labels=categories, autopct='%1.1f%%')

# Add a title
plt.title('Monthly Expenses by Category')

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



In [27]:
```
# Product categories
categories = ['Electronics', 'Clothing', 'Home Decor', 'Books', 'Toys']

# Sales data for each category
sales = [3500, 2800, 2000, 1500, 1200]

# Explode a specific segment (e.g., 'Clothing')
explode = (0, 0.1, 0, 0, 0)  # The second value (0.1) is the amount by whic

# Create a pie chart with explode and shadow
plt.figure(figsize=(8, 8))  # Set the figure size
plt.pie(sales, labels=categories, explode=explode, shadow=True, autopct='%1
plt.title('Sales by Product Category')

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



In [ ]: