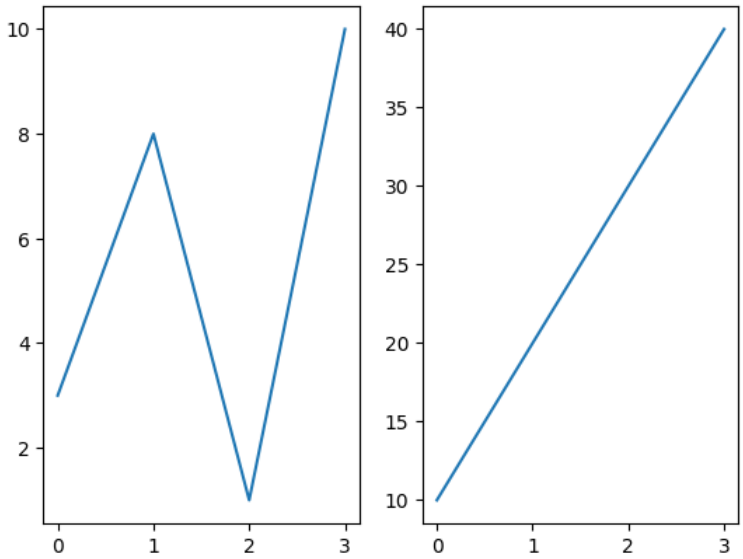
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

# Matplotlib Subplot

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

```
# 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.
> - **index:** Index number of the current subplot, starts from 1.

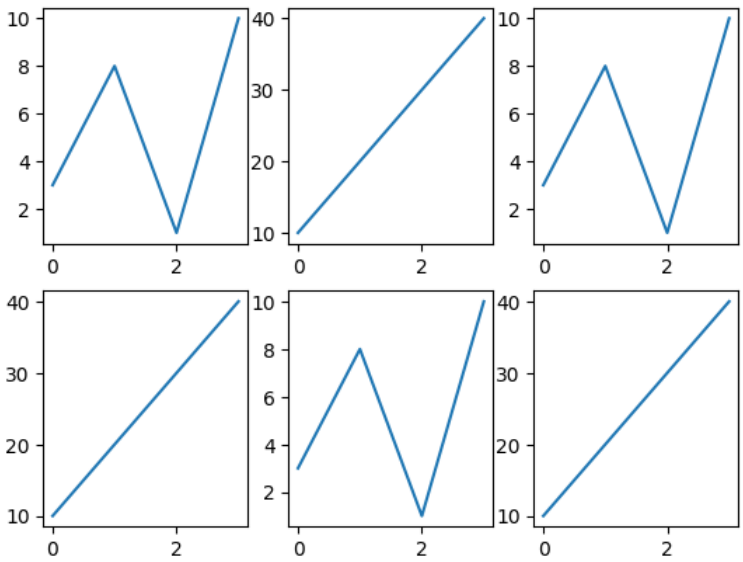
- The figure has 1 row, 2 columns, and this plot is the first plot.

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

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

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

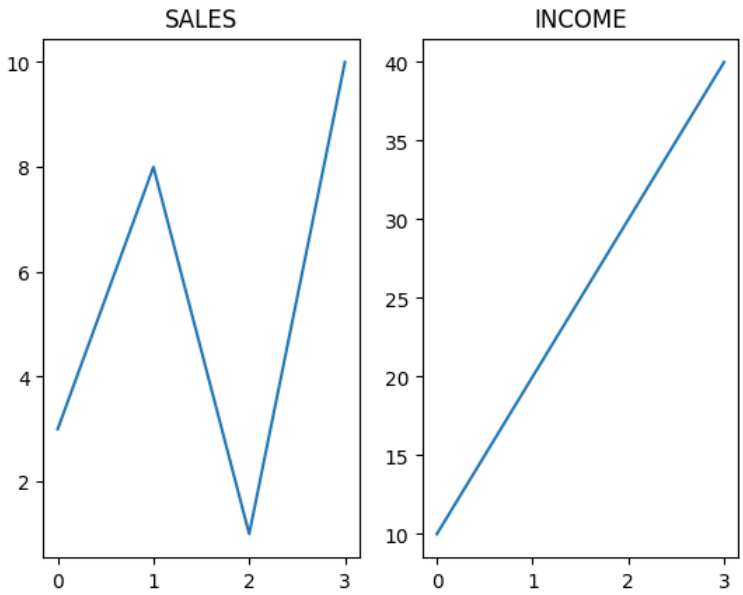
```
# Draw 6 plots

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.

```
#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.

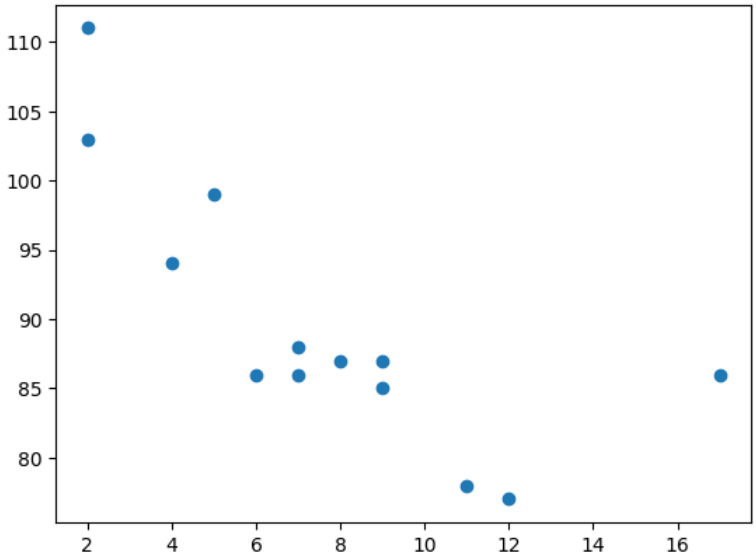
```
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

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:

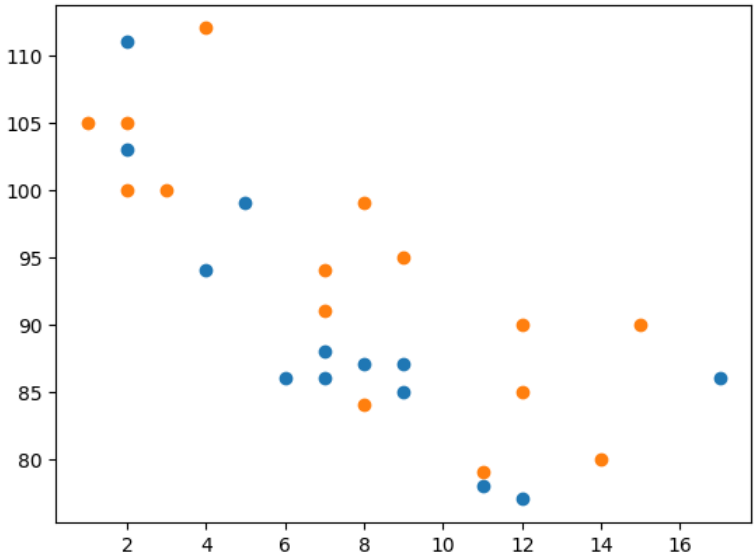
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

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




> **Compare Plots:**

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

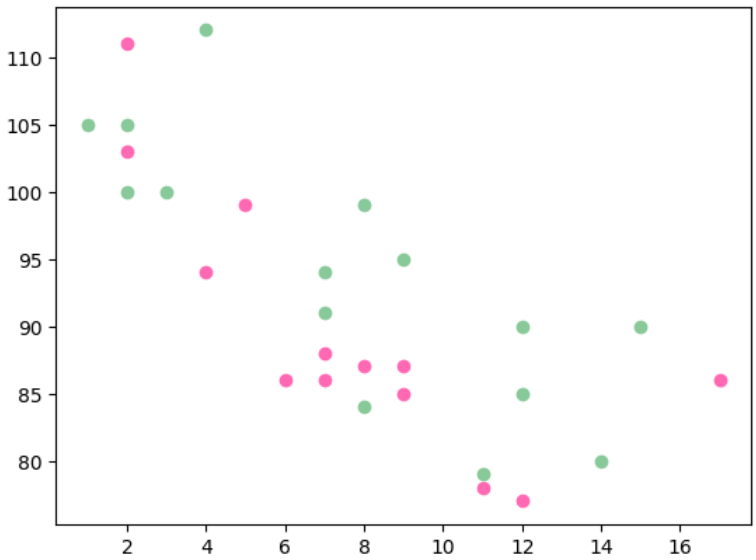
```
#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#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.

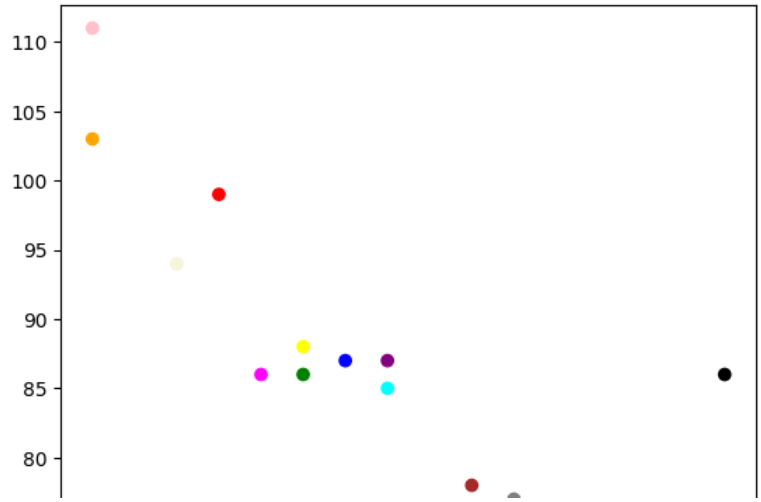
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')

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.

```
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","purple",
                   "beige","brown","gray","cyan","magenta"])

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

plt.show()
```

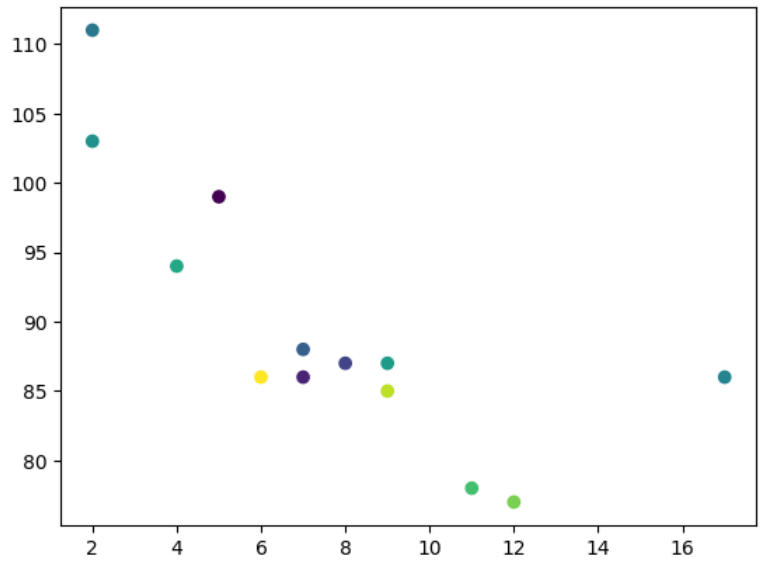



> **ColorMap:**

- 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.

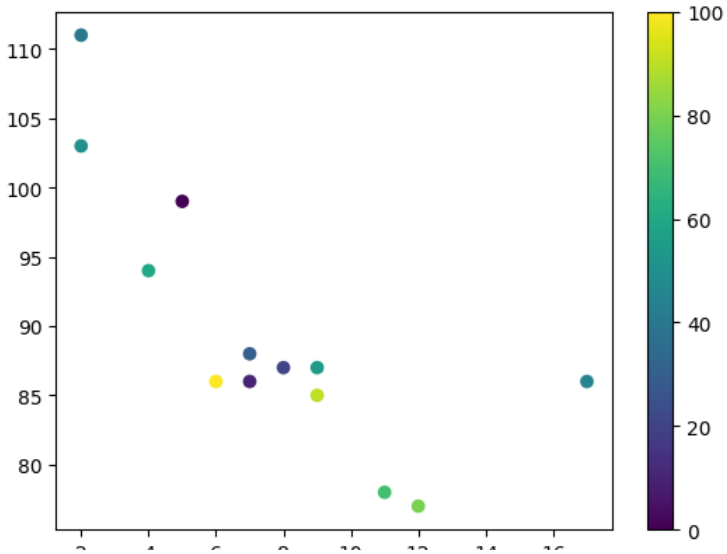
- **Names of colormaps:** winter, twilight, terrain, spring, rainbow, ocean, pink, magma, jet, copper, coolwarm, cool, cividis, bone, binary, Spectral, Set1, Reds, RdBu, Purples, Paired, Oranges, Greys, Greens, BrBG.

```
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.

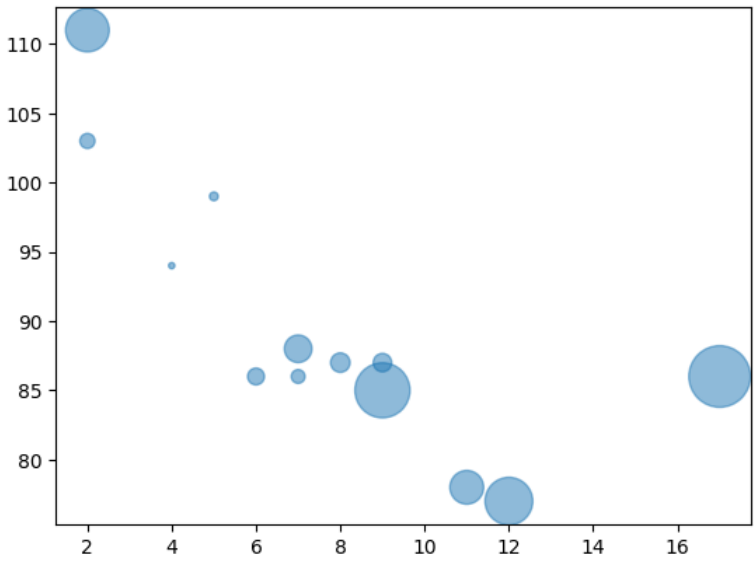
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

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

plt.colorbar()

plt.show()
```




> **Alpha:**

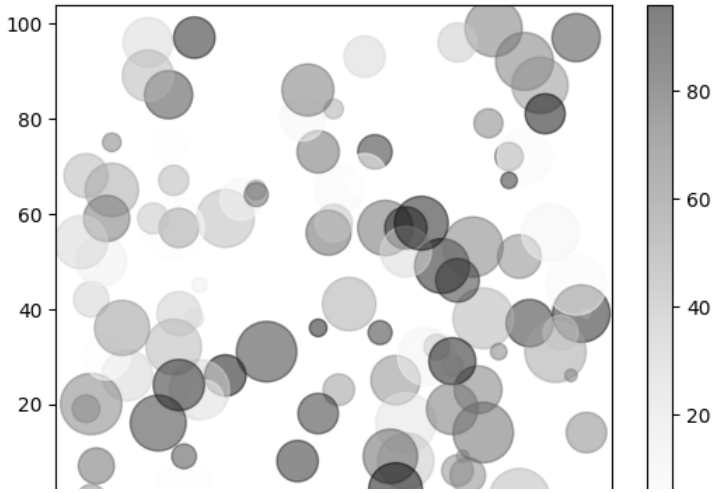
- We can adjust the transparency of the dots with the **alpha** argument.
- Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis.

```
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:**

```
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()
```




## Matplotlib Histograms

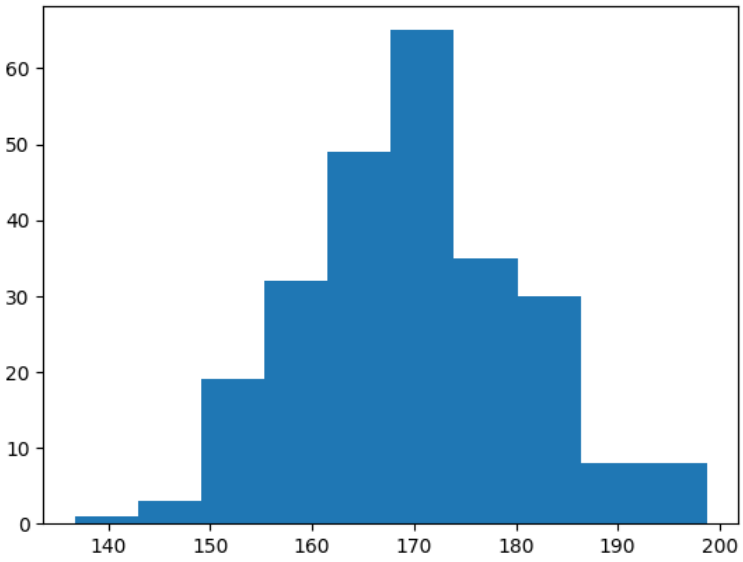
> **Histogram**

- 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.

```
x = np.random.normal(170, 10, 250)

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




## Matplotlib Pie Charts

> **Creating Pie Charts:**

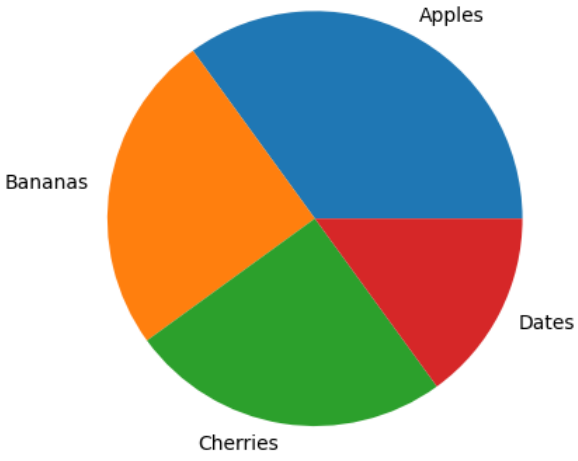
- With Pyplot, we can use the **pie()** function to draw pie charts.
- By default the plotting of the first wedge starts from the **x-axis** and moves **counterclockwise**.

```
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

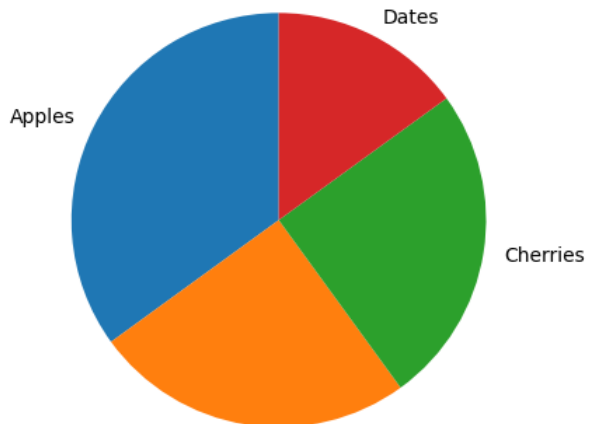
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

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

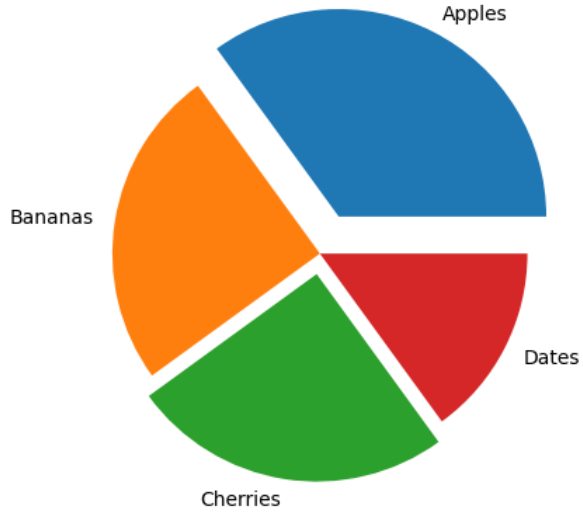
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

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.

```
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:**

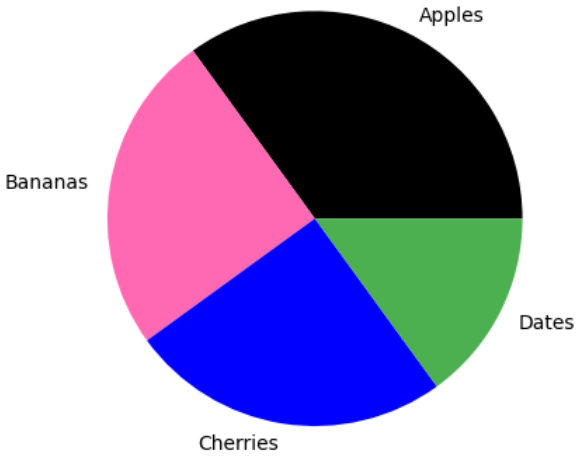
- Add a shadow to the pie chart by setting the **shadows** parameter to **True**.

```
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.

```
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.

```
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.

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

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