Matplotlib

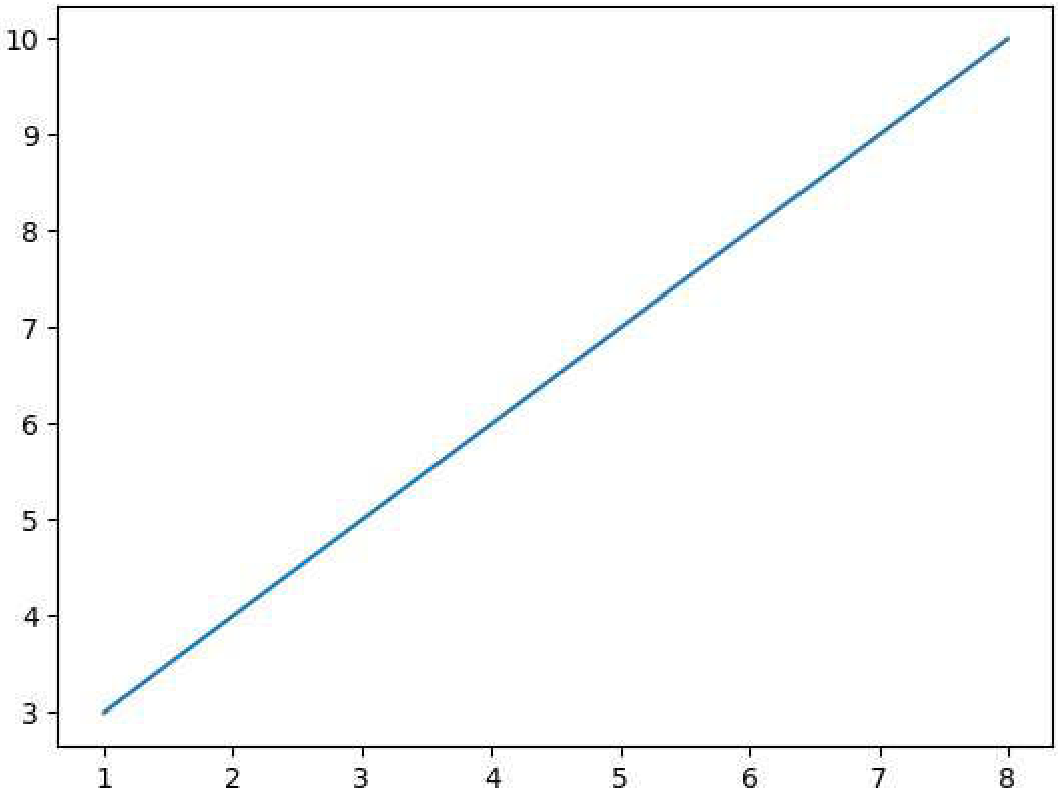
- Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
- Most of the Matplotlib utilities lies under the **pyplot** submodule, and are usually imported under the plt alias
- The **plt.plot(x_points, y_points, marker = 'marker_type')** function is used to draw points (markers) in a diagram. Here marker and x_points are optional
- Eg: If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function

In [1]:

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

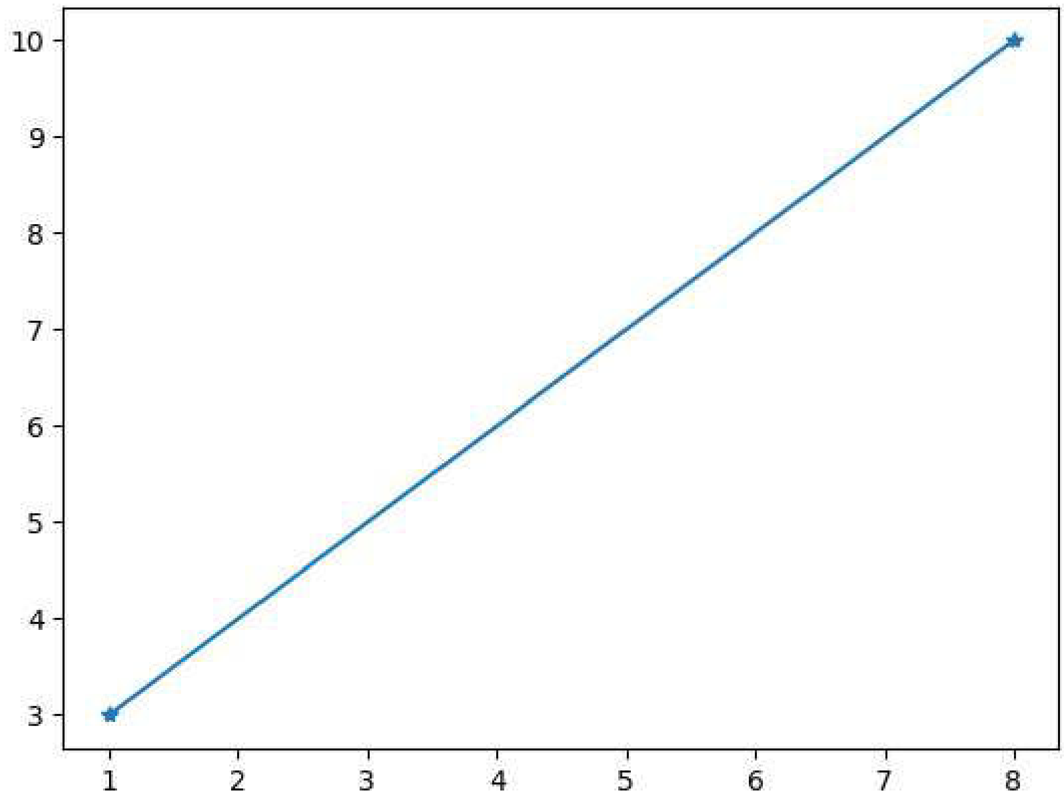
In [2]:

```
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```



In [3]:

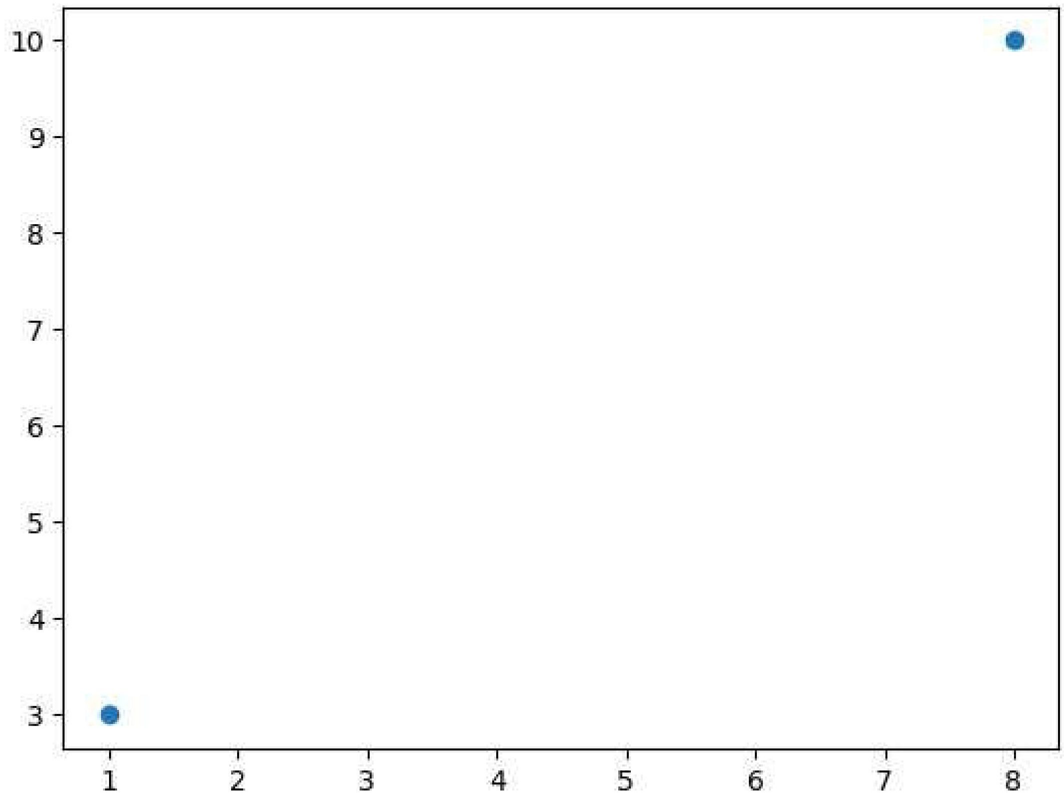
```
plt.plot(xpoints, ypoints, marker = '*')
plt.show()
```



plt.plot(x_points, y_points, 'marker') : To plot only the markers/points without line, add marker as below

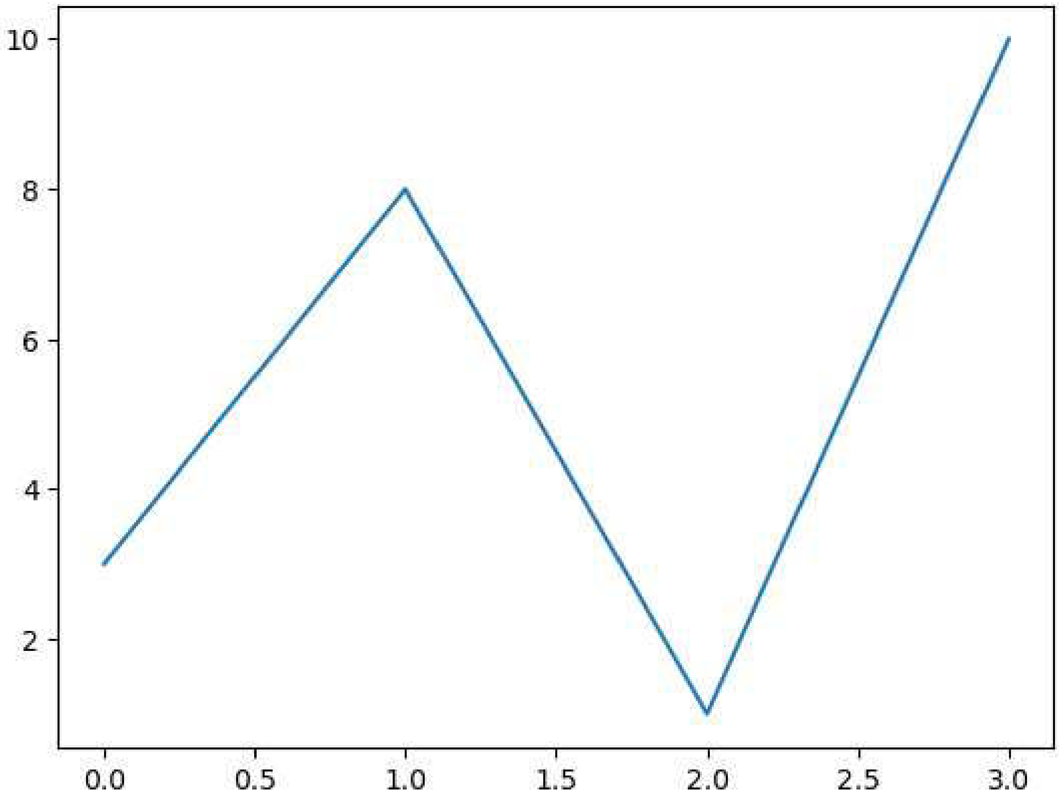
- o means rings
- * means star
- . means point etc.

In [4]:
```
plt.plot(xpoints, ypoints, 'o')
plt.show()
```



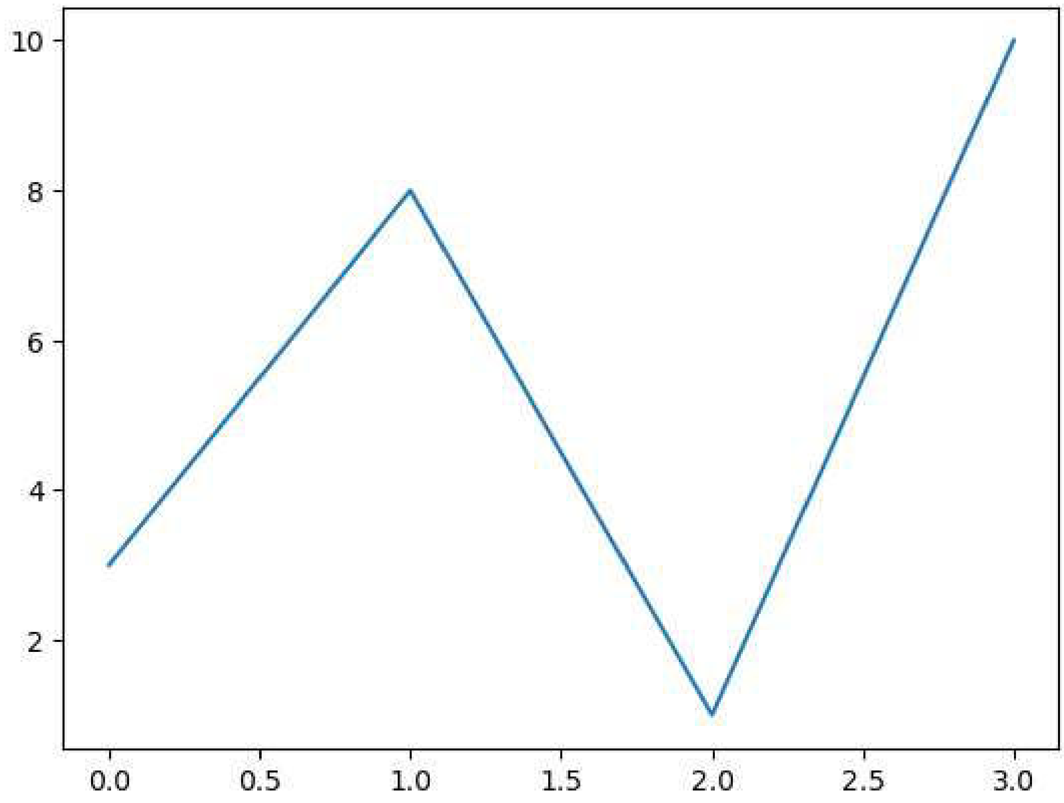
We can plot as many points as we like, we need to ensure that we have the same number of points in both axis.

In [5]:
```
x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])

plt.plot(x1, y1)
plt.show()
```



Note: If we do not specify the points on the x-axis, they will get the default values 0, 1, 2, 3 (etc., depending on the length of the y-points.

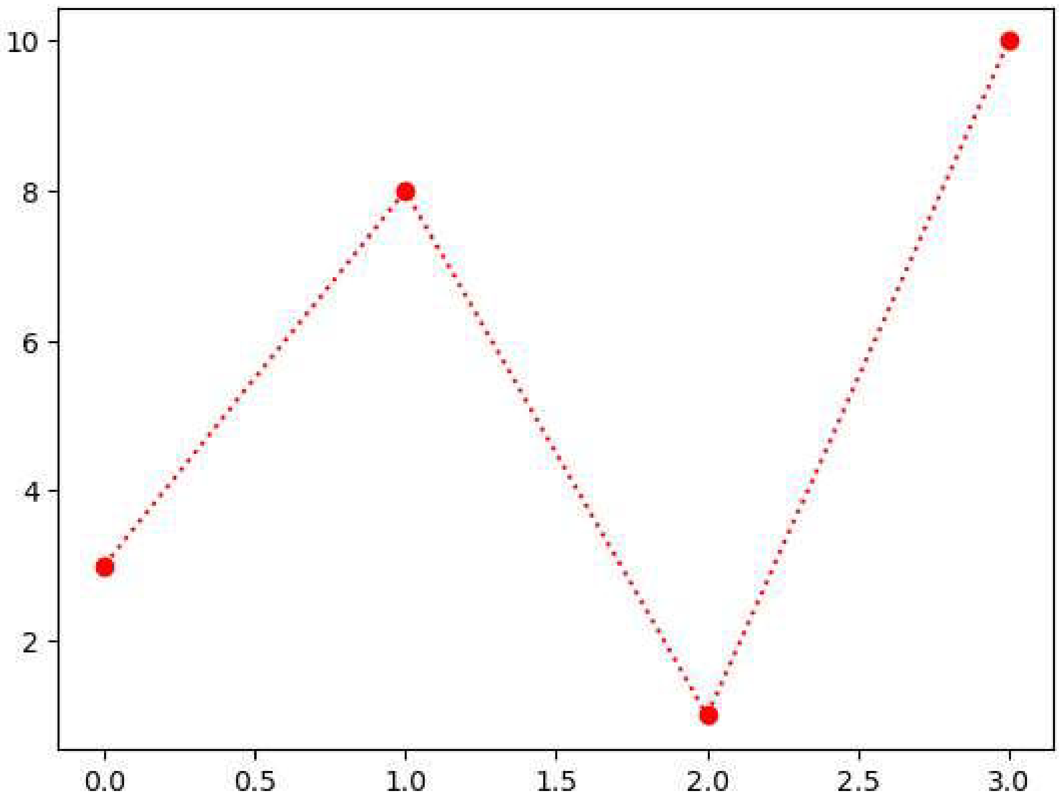
In [6]:
```
plt.plot(y1)
plt.show()
```



Format Strings (fmt)

- It is used in the format **marker|line|color**
- If you leave out the line value in the fmt parameter, no line will be plotted.
- Eg: o:r, o-.r, or etc.

```
In [7]: plt.plot(y1, 'o:r')
        #plt.plot(y1, 'o-r')
        #plt.plot(y1, 'o-.r')
        #plt.plot(y1, '*--g')
        plt.show()
```

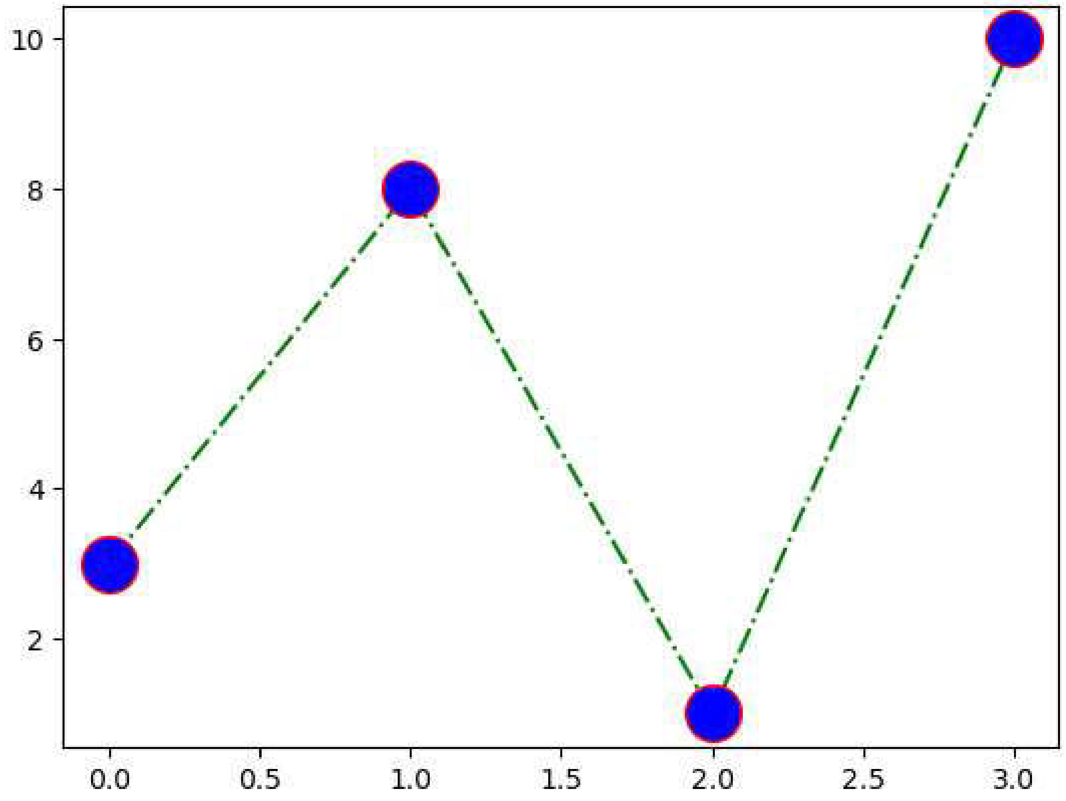


Marker Attributes

- **ms/markersize = value** is used to give size of the marker
- **mec/markeredgecolor = color** is used to give edge color of the marker
- **mfc/markerfacecolor = color** is used to give face color of the marker
- **Note:** We can use hexadecimal color values to give colors. We have 140 preferred colors available.

```
In [8]: plt.plot(x1, y1, 'o-.g', marker = 'o', ms = 20, mec = 'r', mfc = 'b')
        plt.show()
```

```
C:\Users\srbhk\AppData\Local\Temp\ipykernel_8440\3750214434.py:1: UserWarnin
g: marker is redundantly defined by the 'marker' keyword argument and the fmt
string "o-.g" (-> marker='o'). The keyword argument will take precedence.
  plt.plot(x1, y1, 'o-.g', marker = 'o', ms = 20, mec = 'r', mfc = 'b')
```

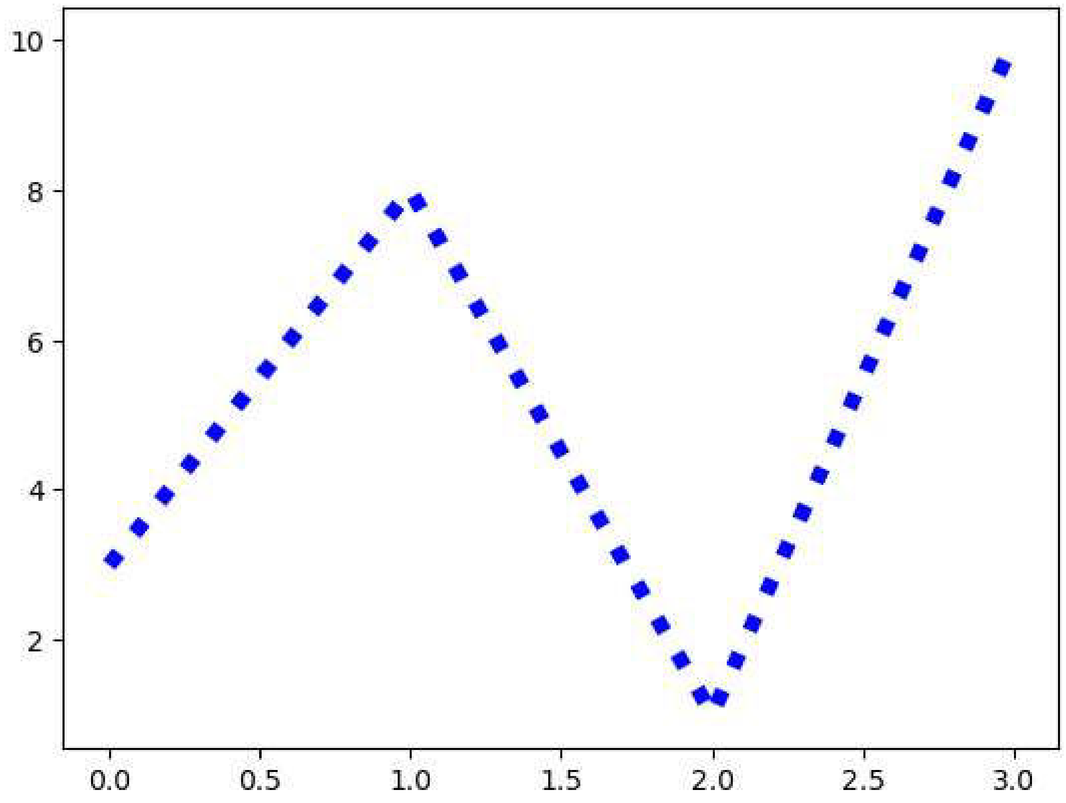


Line Attributes

- **ls/linestyle = 'line_type'** is used to define type of line (dotted/dashed/solid/dashdot)
- **c/linecolor = 'color'** is used to give required color to the line
- **lw/linewidth = 'size'** is used to give width of the line

In [9]:

```
plt.plot(x1, y1, ls = 'dotted', c = 'b', lw = '5.5')
plt.show()
```



Title and Labels of Plot

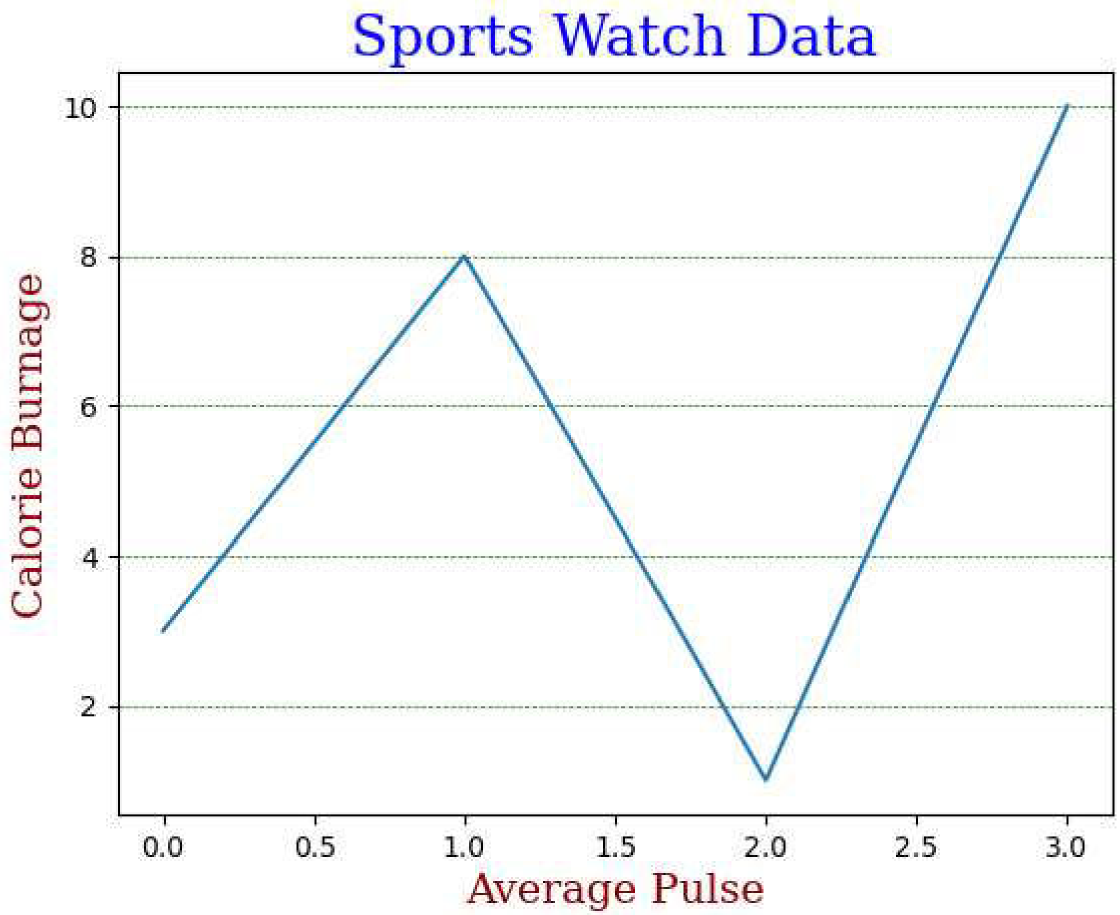
- **plt.title('title')** is used to give the title of the plot
    - We can use **loc = left/right/center** argument to specify the position of title. Default is center
- **plt.xlabel('label_name)** and **plt.ylabel('label_name)** are used to plot x and y labels respectively

Note: We can use fontdict parameter in title and labels to specify the font details if required. Font can have family(font name), color and size

Plot Grid

- **plt.grid()** is used to have a grid on the plot.
- We can give axis as argument if we require any one of the grid.Eg: **plt.grid(axis = 'x')**
- We can also give color, linestype and width as arguments, like **plt.grid(color = 'color', linestyle = 'linestyle', linewidth = number)**

```
In [10]: font1 = {'family':'serif','color':'blue','size':20}
         font2 = {'family':'serif','color':'darkred','size':15}

         plt.title("Sports Watch Data", fontdict = font1, loc ='center')
         plt.xlabel("Average Pulse", fontdict = font2)
         plt.ylabel("Calorie Burnage", fontdict = font2)

         plt.plot(x1, y1)
         plt.grid(axis = 'y', color = 'green', linestyle = '--', linewidth = 0.5)
         plt.show()
```



Multi Plots

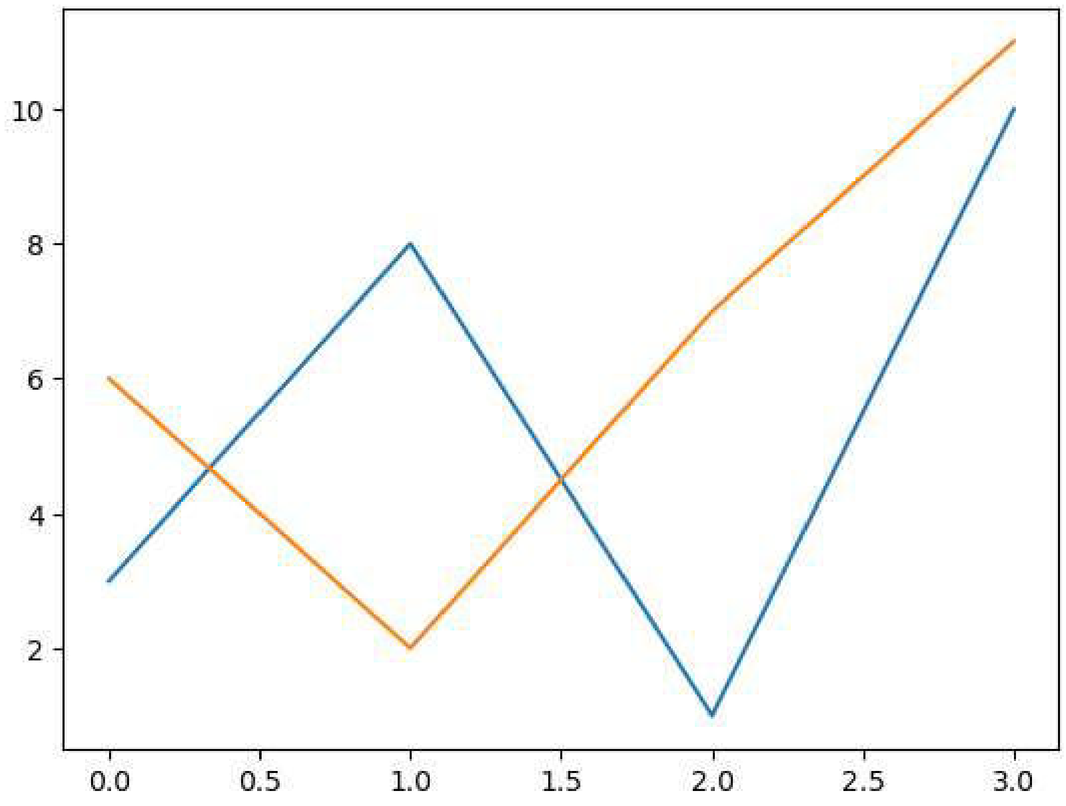
- We can plot 2 or more points in the same graph as follows:
    1. plt.plot(xpoints1, ypoints 1, xpoints2, ypoints 2....)
    2. plt.plot(xpoints1, ypoints1) plt.plot(xpoints2, ypoints2)

Subplots

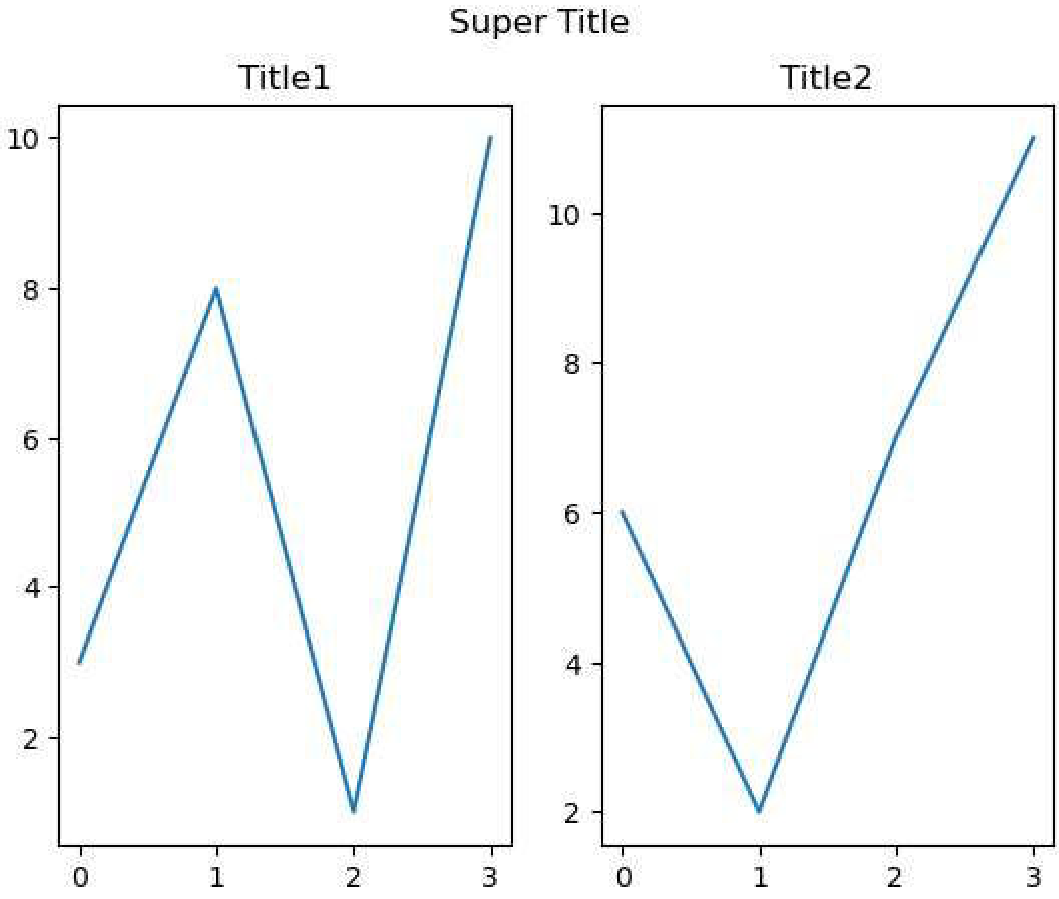
- **plt.subplot(pos, kwargs)** here, pos is Either a 3-digit integer or three separate integers describing the position of the subplot
- Here the first, second, and third integer are no of rows,no of cols, index of current plot.
- We can give individual title for each plot
- **Note: plt.suptitle("Super_title")** is used to give super title to subplots

In [11]:

```
#x1, y1 declared previously
x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

plt.plot(x1, y1)
plt.plot(x2, y2)
# or plt.plot(x1, y1, x2, y2)
plt.show()
```



In [12]:
```
#plot 1
plt.subplot(1, 2, 1)
plt.plot(x1,y1)
plt.title("Title1")

#plot2
plt.subplot(1, 2, 2)
plt.plot(x2,y2)
plt.title("Title2")

plt.suptitle("Super Title")
plt.show()
```

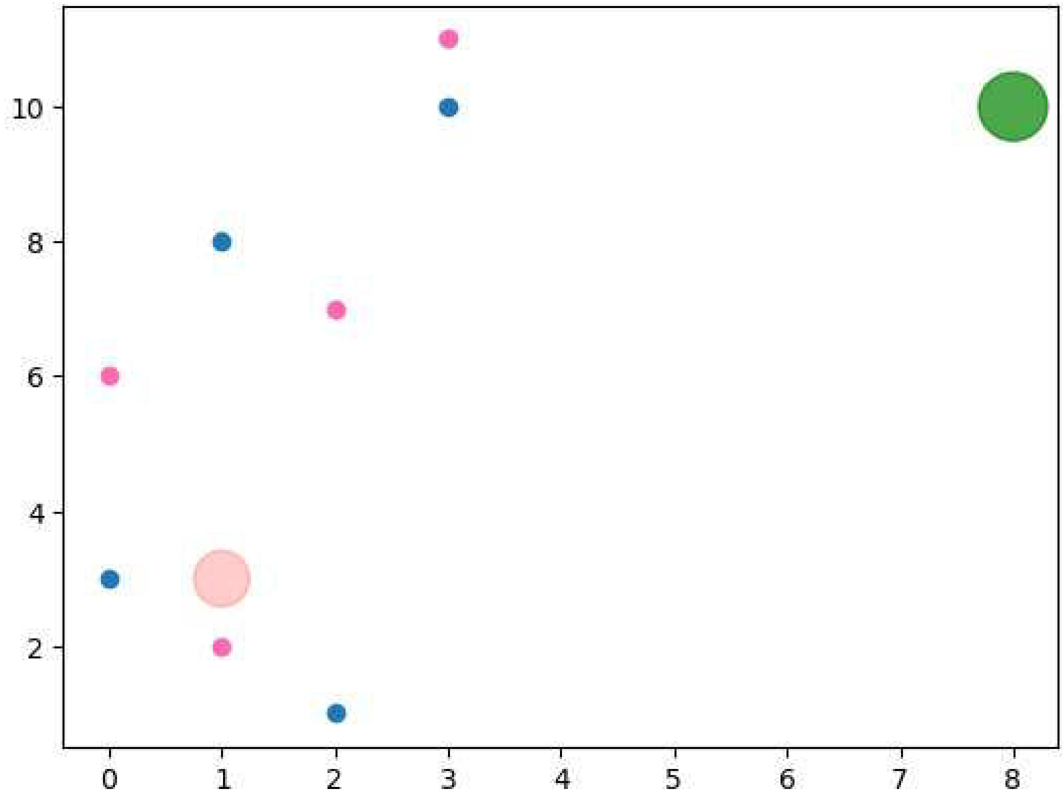


Scatter Plots

- **plt.scatter(xpoints, ypoints, c= 'color', s = size, alpha = transparency_value)** is used to plot scatter plot.
- Color, size, alpha is optional.
- We can color, size and alpha (transparency) each dot seperately by passing an array of equal colors/elements as of markers.
- We can also pass color map with **cmap** argument. A colormap is like a list of colors, where each color has a value that ranges from 0 to 100.
- **plt.colorbar()** is used to print colormap scale along with graph

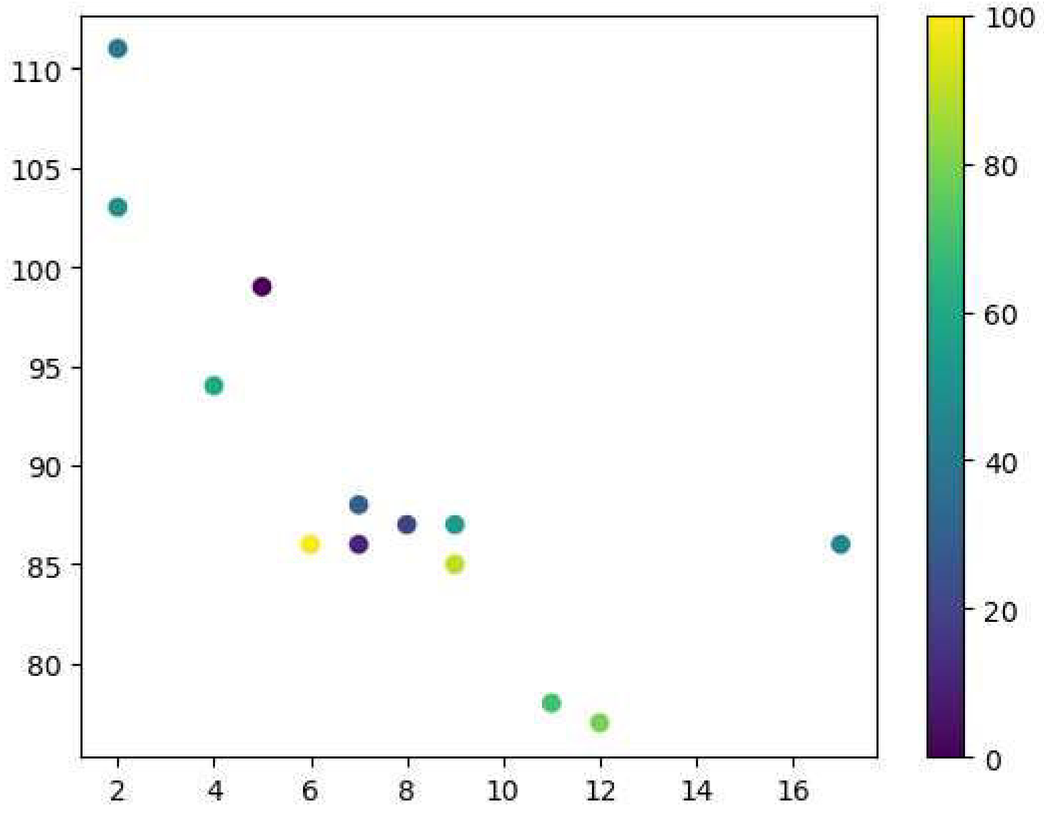
```
In [13]: plt.scatter(x1, y1)
         plt.scatter(x2,y2, c = 'hotpink')

         # passing arrays for each element
         colors = np.array(["red","green"])
         sizes = np.array([400, 600])
         alphas = np.array([0.2, 0.7])
         plt.scatter(xpoints,ypoints, c = colors, s = sizes, alpha = alphas)
```

```
Out[13]: <matplotlib.collections.PathCollection at 0x20522b2f8e0>
```



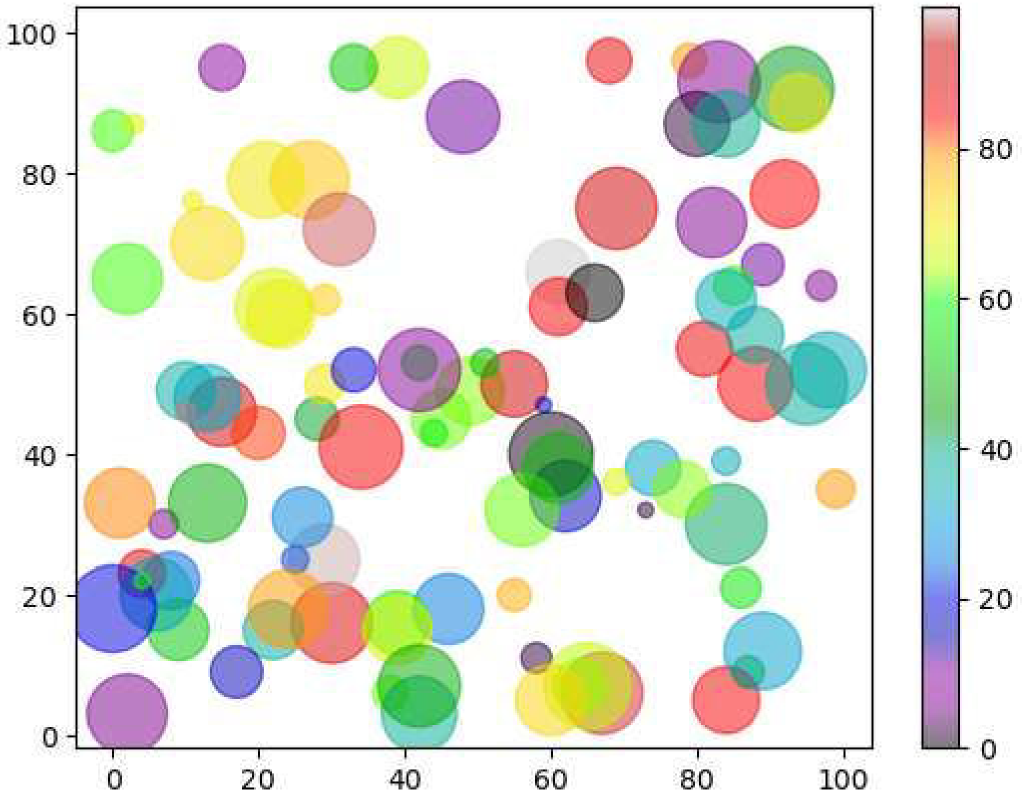
```
In [14]: #Colormap
         x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
         y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
         colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

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

```
Out[14]: <matplotlib.colorbar.Colorbar at 0x20521666dd0>
```



Practice Question: Create random arrays with 100 values for x-points, y-points, colors and sizes.

```
In [15]: 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='nipy_spectral')
         plt.colorbar()
         plt.show()
```



Bar Plots

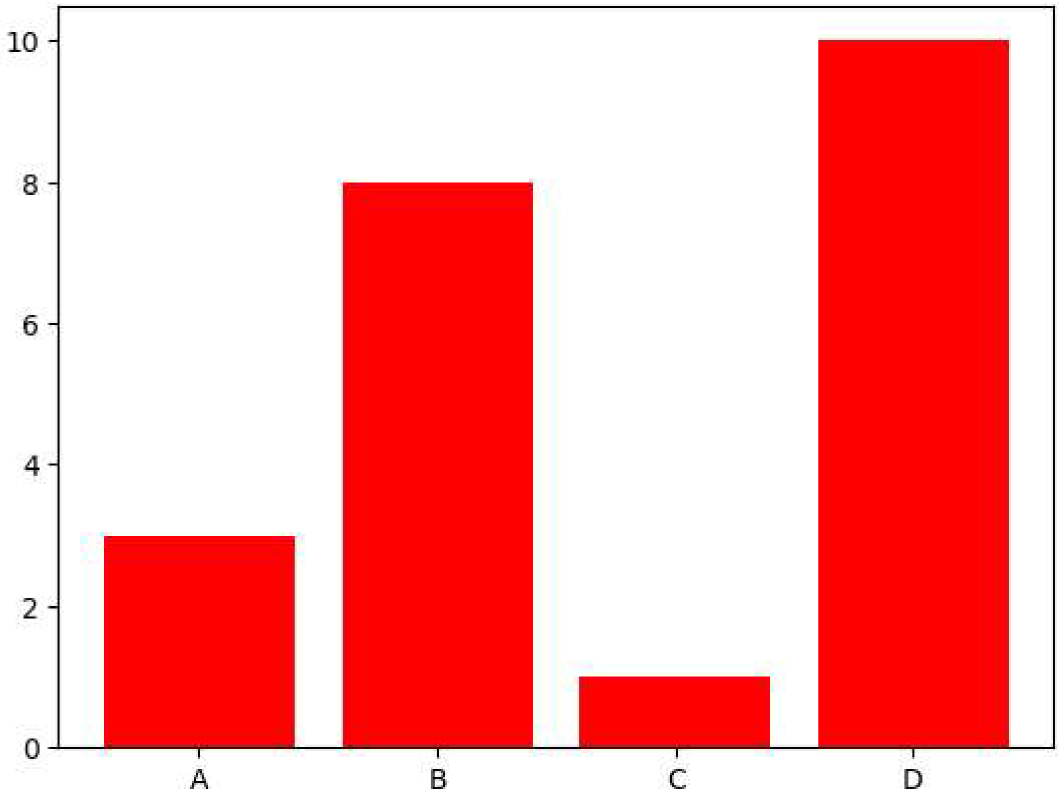
- graphical representation of categorical data and has equal space between each pair of consecutive bars
- **plt.bar(x,y, c= 'color', width = value)** or **plt.barh(x,y, c= 'color', height = value)** is used to plot bar graph and horizontal bar graph respectively.
- The default width/height is 0.8.
- color, width and height are optional.

In [16]:

```
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y, color = 'red', width = 0.8)
```

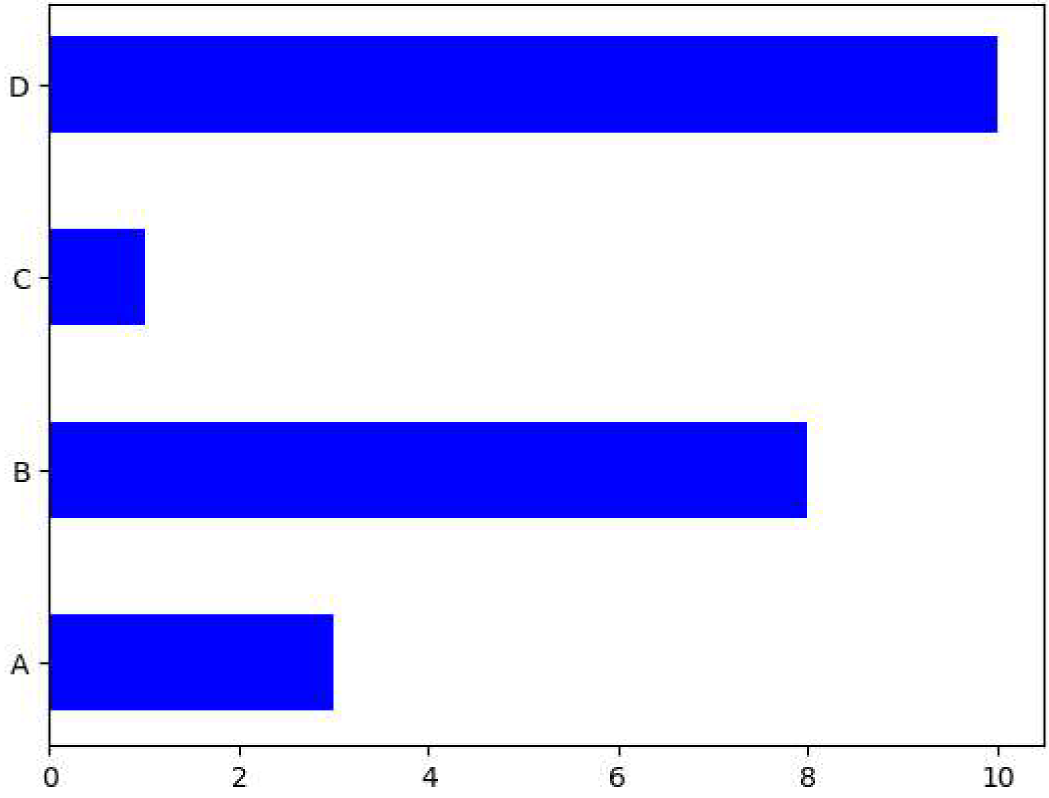
Out[16]:

```
<BarContainer object of 4 artists>
```



```
In [17]: plt.barh(x,y, color = 'blue', height =0.5)
```

```
Out[17]: <BarContainer object of 4 artists>
```

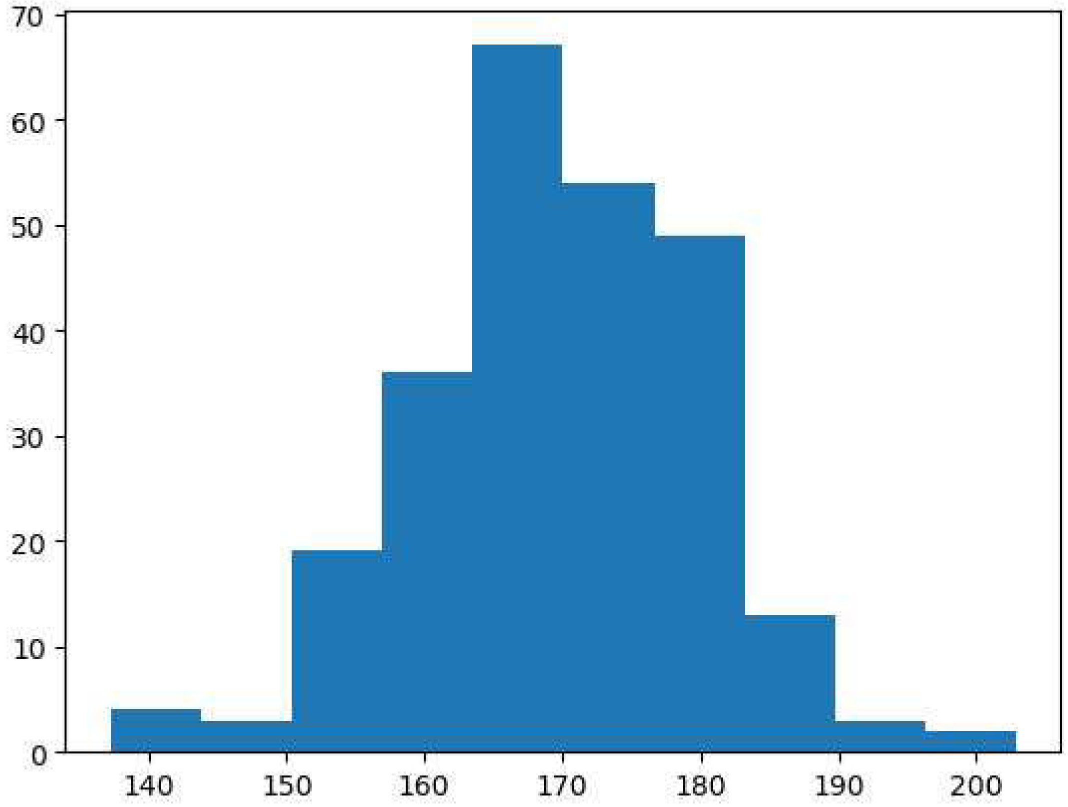


Histograms

- graphical representation of quantitative data and has no space between the consecutive bars.
- histogram is a graph showing frequency distributions.
- It is a graph showing the number of observations within each given interval.
- **plt.hist(xpoints)** is used to plot the histogram

In [18]:

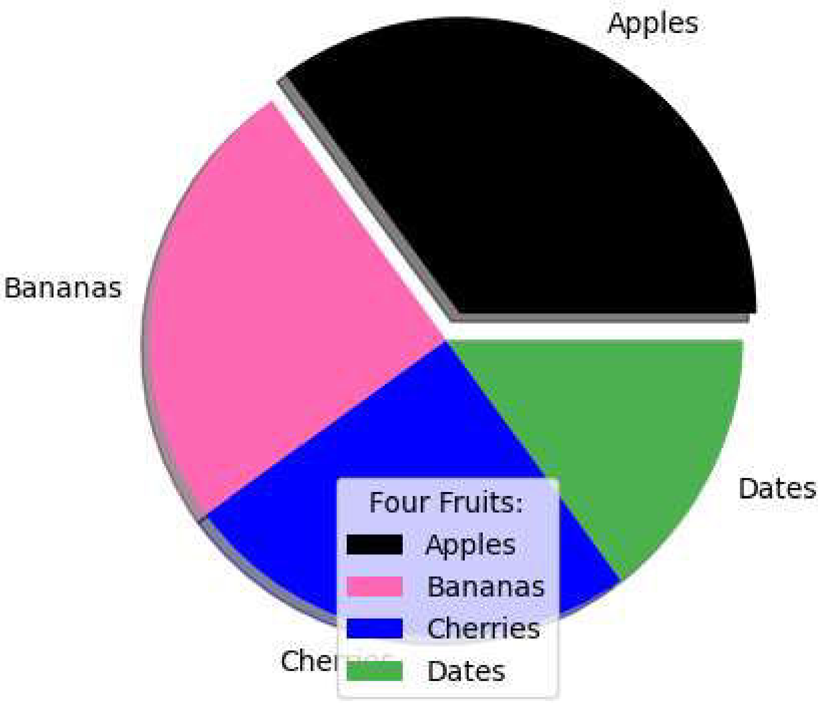
```
x = np.random.normal(170, 10, 250)

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



Pie Charts

- We can plot the pie chart with **plt.pie(x, labels = label_array, startangle = value, explode = explode_array, shadow = True)
  - **labels(optional):** We can add required labels with the optional labels attribute
  - **startangle(optional):** The start angle by default is 0 degrees (x axis). We can change it by adding optional argument startangle
  - **explode(optional):** We can define how far the wedge will be from center
  - **shadow = True(optional):** is added if we require shadow of pie chart
  - **colors(optional):** We can add custom colors with the optional colors arguments
- **plt.legend(title = 'Title')** is used to display list of explanation for each wedge
  - **title(optional):** is used to give title to legend

```
In [19]: x = np.array([35, 25, 25, 15])
         mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
         myexplode = [0.1, 0., 0, 0]
         mycolors = ["black", "hotpink", "b", "#4CAF50"]

         plt.pie(x, labels = mylabels, startangle = 0, explode = myexplode, shadow = Tr
         plt.legend(title = "Four Fruits:")
         plt.show()
```


Apples
Bananas
Cherries
Dates
Four Fruits:
Apples
Bananas
Cherries
Dates