24. Matplotlib

October 31, 2022

1 Matplotlib: Visualization with Python

- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Installing matplotlib > `pip install matplotlib`

2. Importing matplotlib

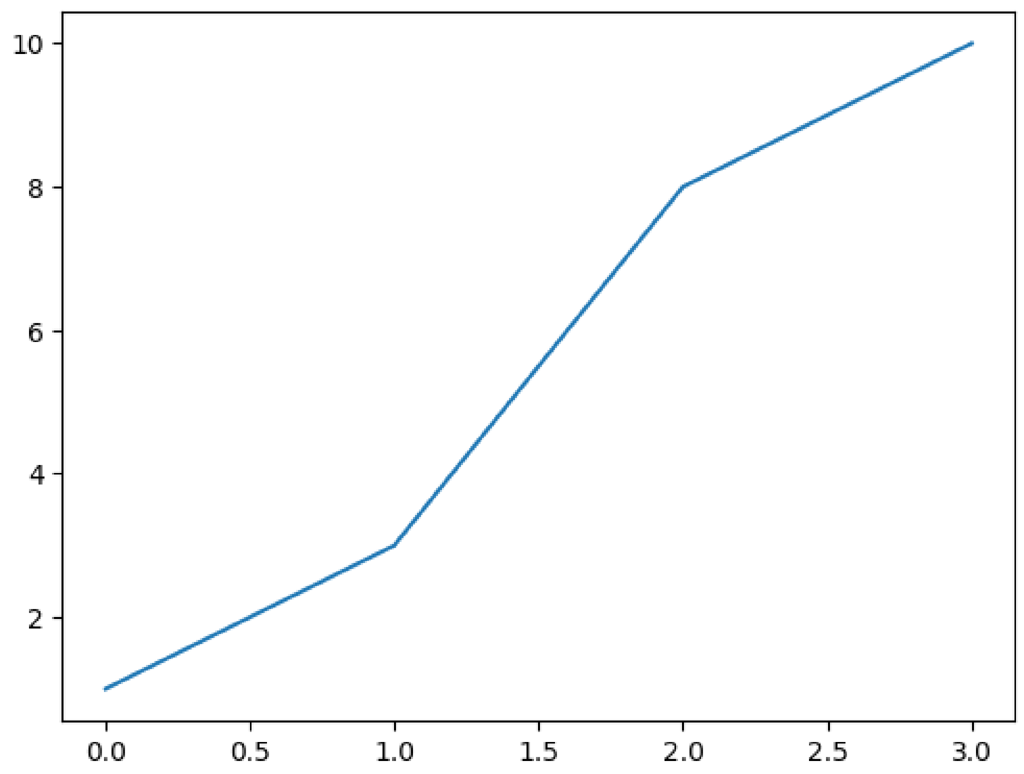
- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt.

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

2 Plotting x and y points

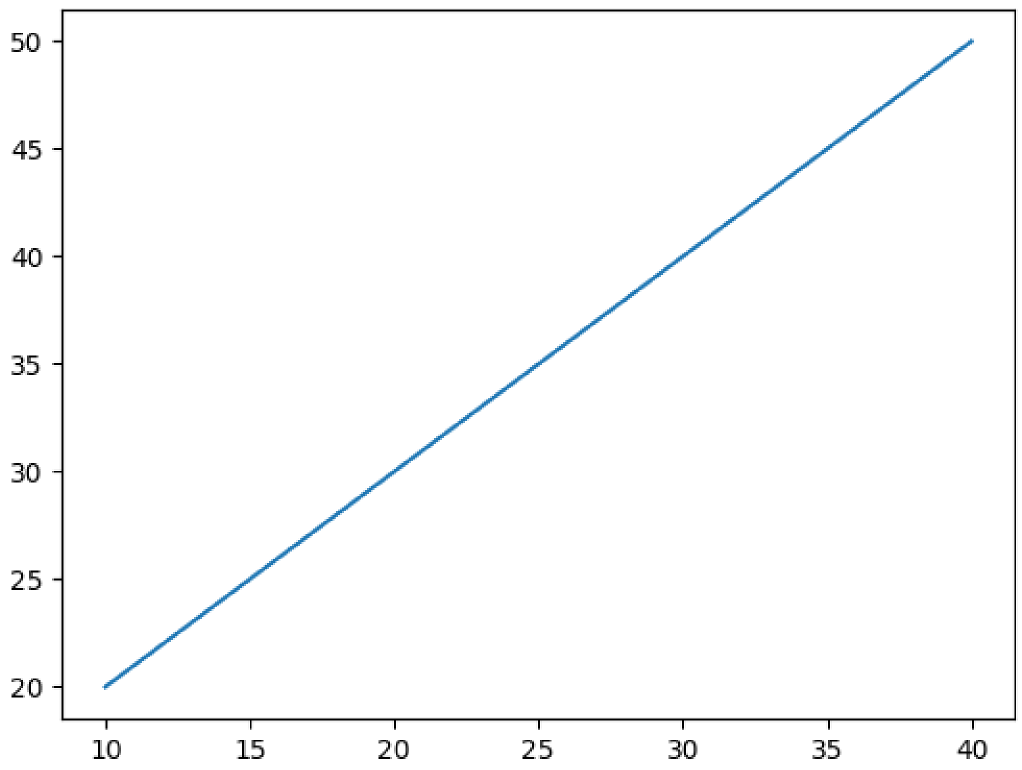
- The `plot()` function is used to draw points (markers) in a diagram.
- By default, the plot() function draws a line from point to point.
- The function takes parameters for specifying points in the diagram.
- Parameter 1 contains the points on the x-axis.
- Parameter 2 contains the points on the y-axis.
- If we need to plot a line from (1, 3) to (8, 10), we have to pass two lists/arrays `[1, 8]` nd `[3, 10]` to the plot function.

[2]:
```python
# Draw a line in a diagram from position (1, 3) to position (8, 10)
xpoints = [1, 8]
ypoints = [3, 10]
plt.plot([1,3,8,10])
#plt.plot(xpoints, ypoints)
plt.show()
```



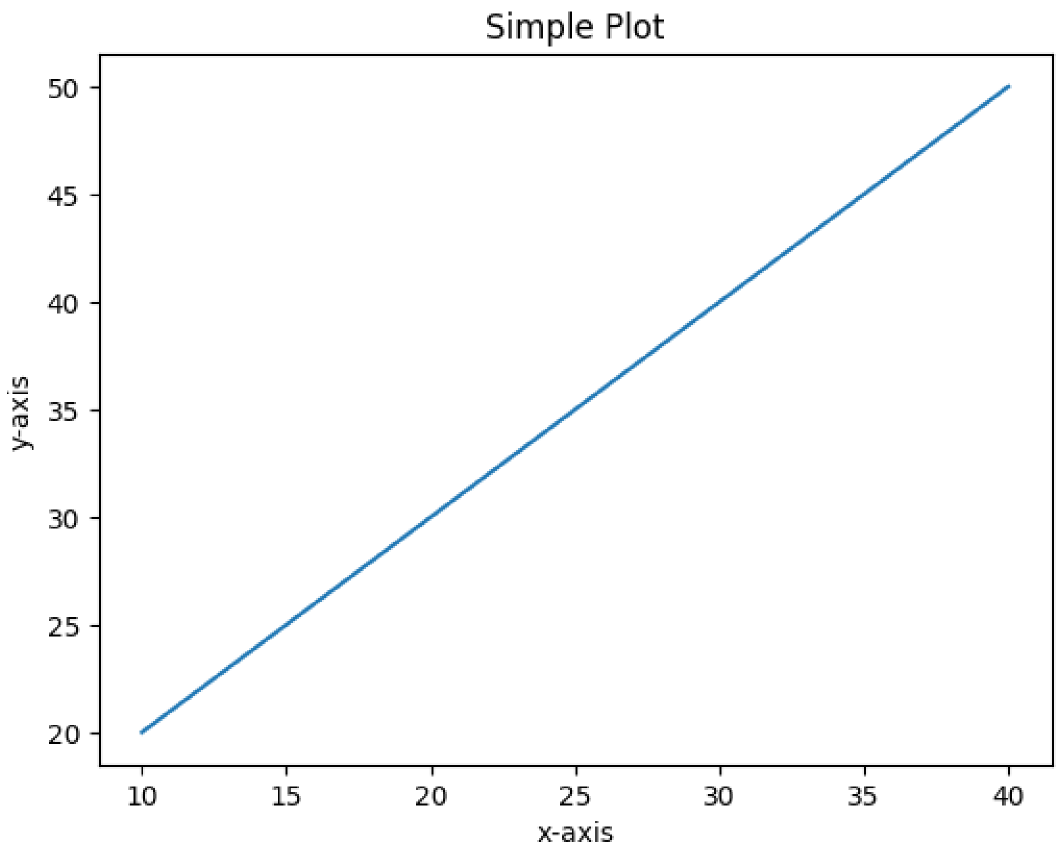
[3]:
```
## Plotting multiple points
```

[4]:
```
# initializing the data
x = np.array([10, 20, 30, 40])
y = np.array([20, 30, 40, 50])

# plotting the data
plt.plot(x, y)

plt.show()
```

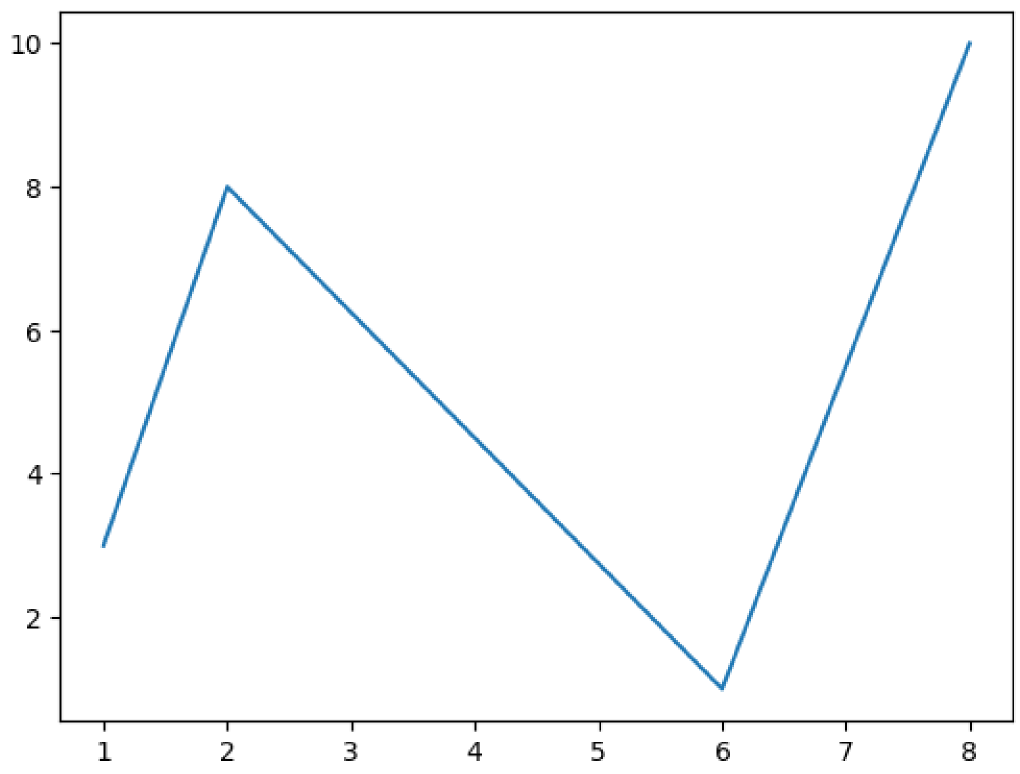


In the above example, the elements of X and Y provides the coordinates for the x axis and y axis and a straight line is plotted against those coordinates.

[5]:
```
# plotting with labels and title

# initializing the data
x = [10, 20, 30, 40]
y = [20, 30, 40, 50]

# plotting the data
plt.plot(x, y)

# Adding the labels
plt.ylabel("y-axis")
plt.xlabel("x-axis")

# Adding the title
plt.title("Simple Plot")

plt.show()
```



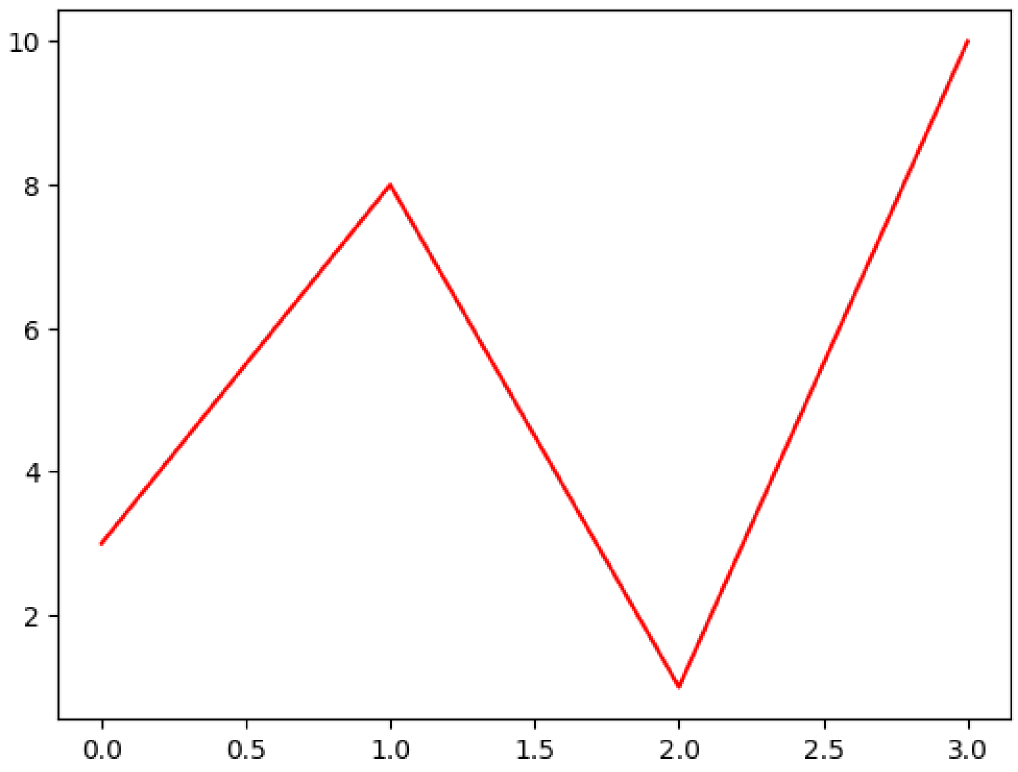
[6]:
```
# Another example

xpoints = [1, 2, 6, 8]
ypoints = [3, 8, 1, 10]

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



3 Default X-Points

- If we do not specify the points in the x-axis, they will get the default values 0, 1, 2, 3, (etc. depending on the length of the y-points.
- So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

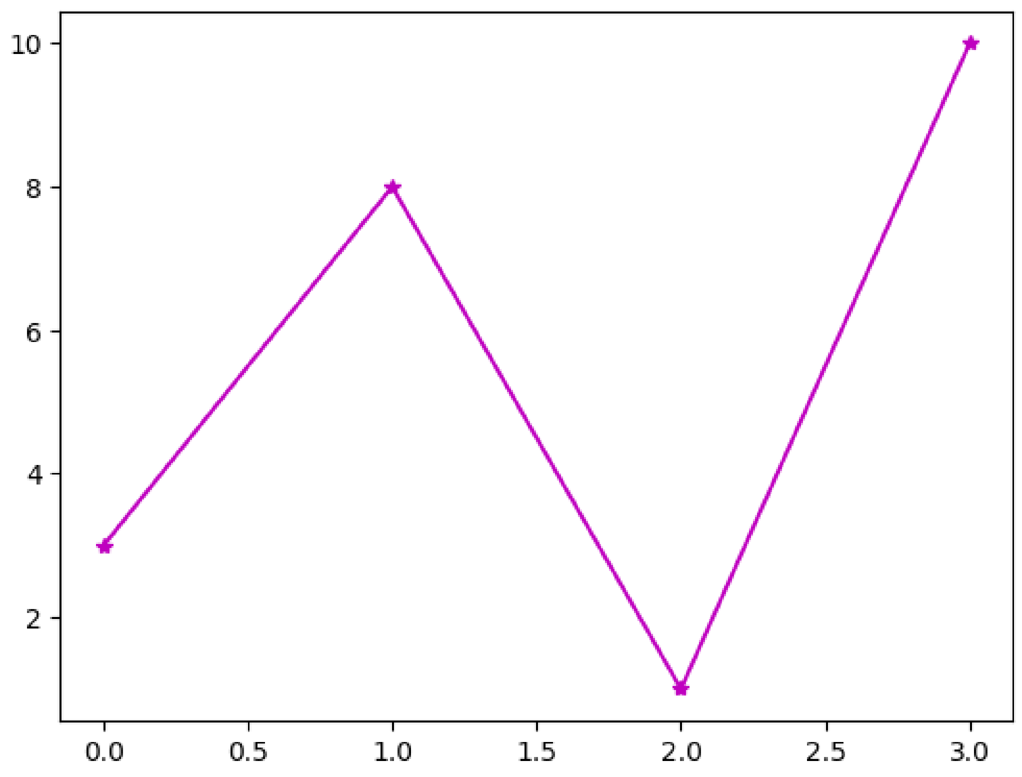
[7]:
```
ypoints = [3, 8, 1, 10]

plt.plot(ypoints, color="r")
plt.show()
```



4 Markers

- Keyword argument marker is used to emphasize each point with a specified marker.
- More markers: click here

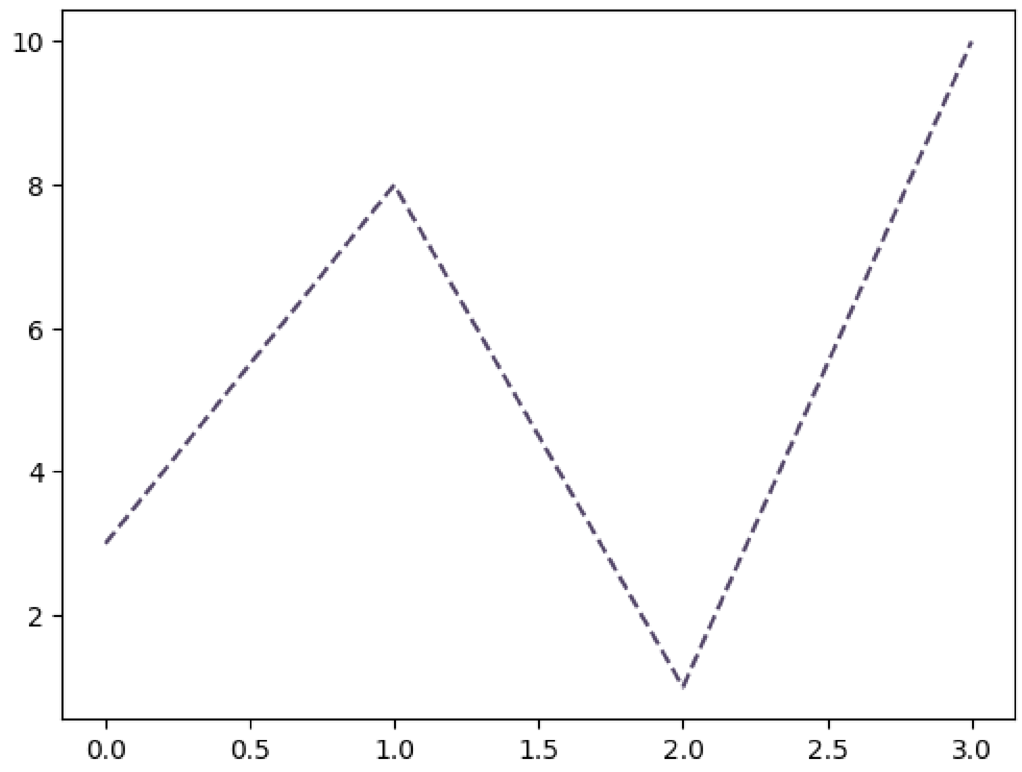
[8]:
```
ypoints = [3, 8, 1, 10]

plt.plot(ypoints, marker = '*', color = "m")
plt.show()
```



4.1 Format Strings fmt

- You can use also use the shortcut string notation parameter to specify the marker.
- This parameter is also called `fmt`, and is written with this syntax: `marker|line|color`

[9]:
```
ypoints = np.array([3, 8, 1, 10])

#plt.plot(ypoints, 'o-.r')
#plt.plot(ypoints, linestyle="dotted")
plt.plot(ypoints, ls="dashed", color = '#524567')
plt.show()
```

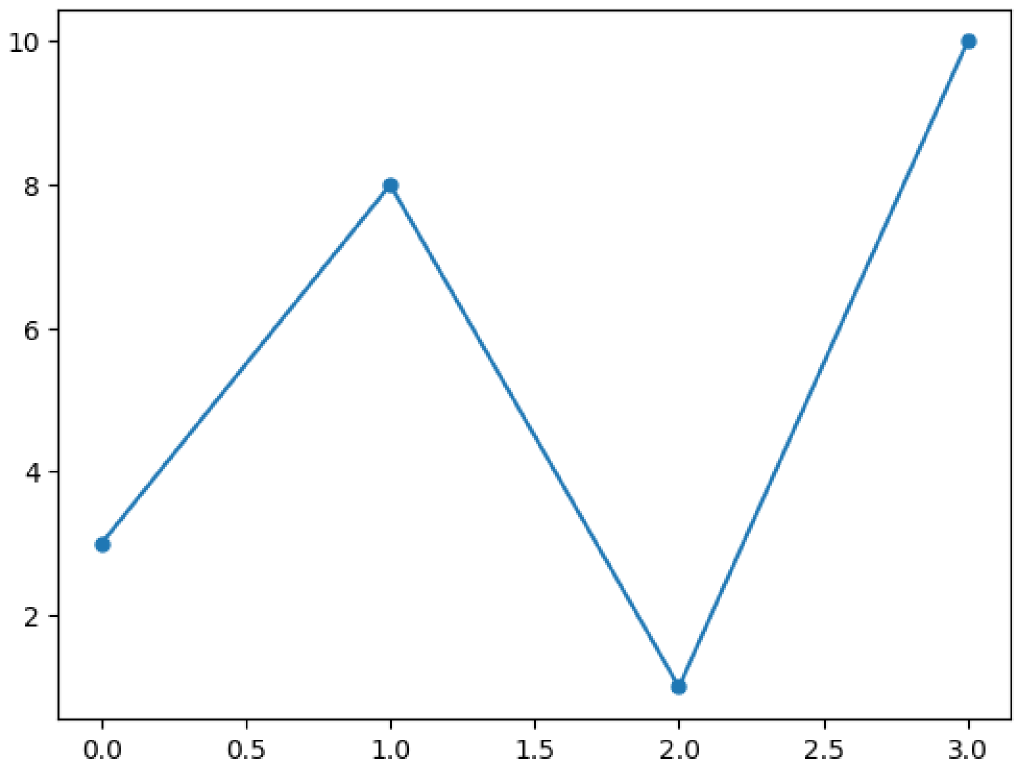


- Lines:
  - Solid line: `-`
  - Dotted line: `:`
  - Dashed line: `--`
  - Dashed/dotted line: `-.`
- Colours:
  - Red: `r`
  - Green: `g`
  - Blue: `b`
  - Cyan: `c`
  - Magenta: `m`
  - Yellow: `y`
  - Black: `k`
  - White: `w`

4.2 Marker Size

- Keyword argument `markersize` or the shorter version `ms` is used to set the size of the markers.

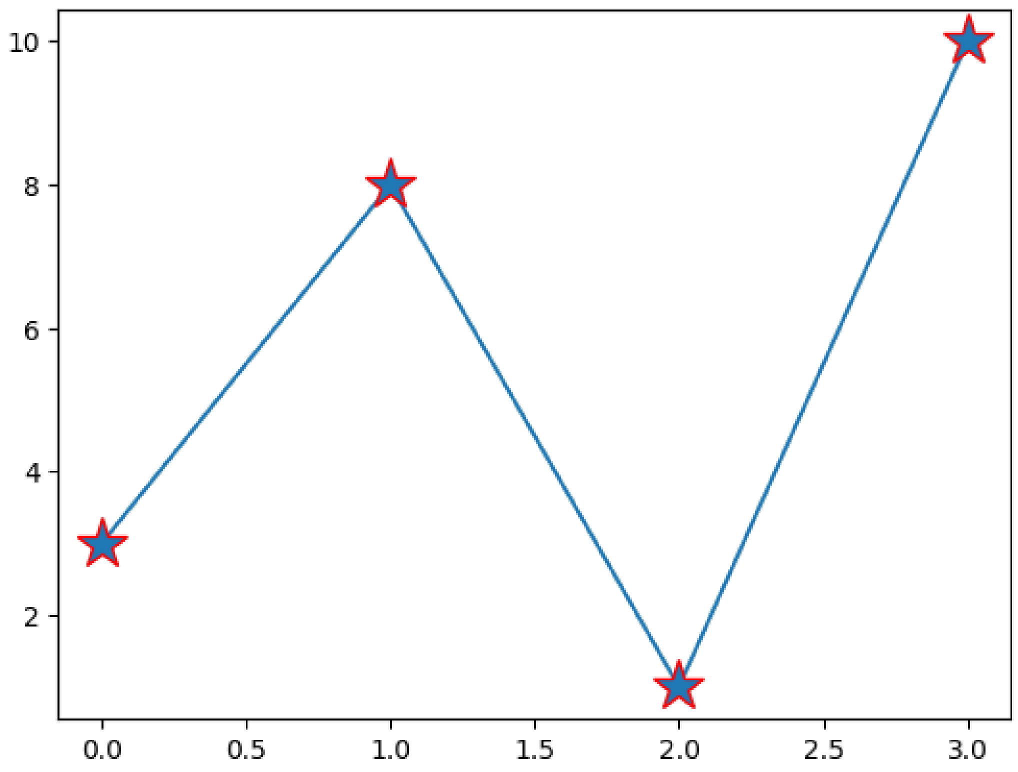
[10]:
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 5)
plt.show()
```



4.3 Marker Colour

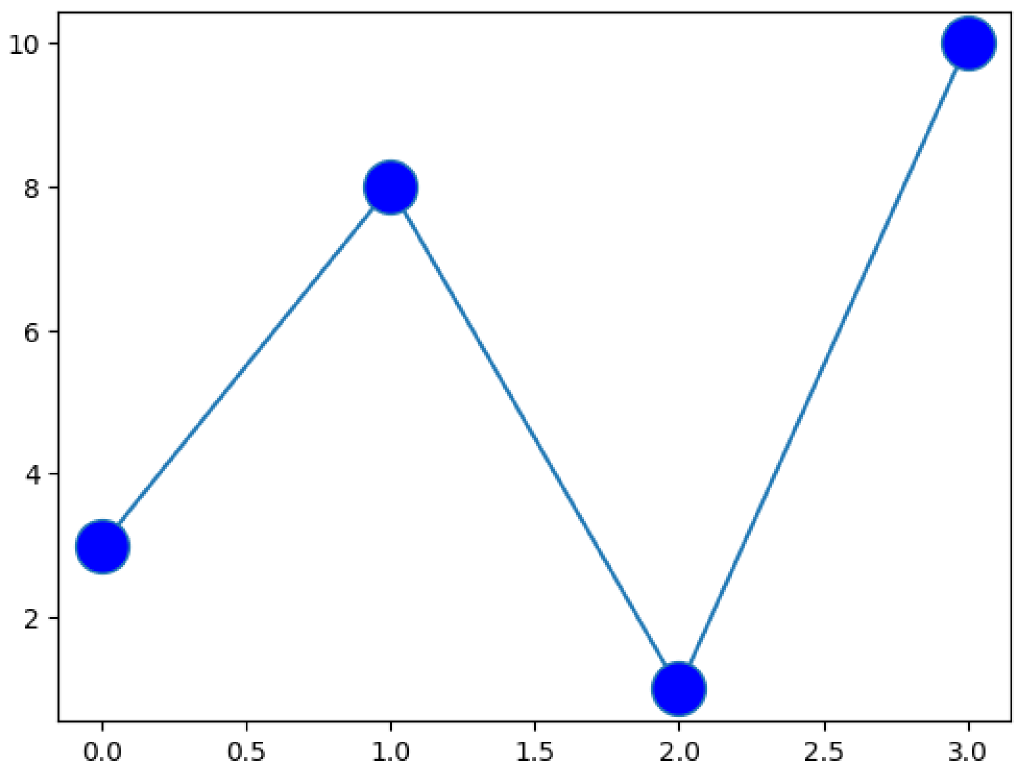
- Keyword argument `markeredgecolor` or the shorter `mec` is used to set the color of the edge of the markers.

[11]:
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = '*', ms = 20, mec = 'r')
plt.show()
```



- Keyword argument `markerfacecolor` or the shorter `mfc`to set the color inside the edge of the markers.

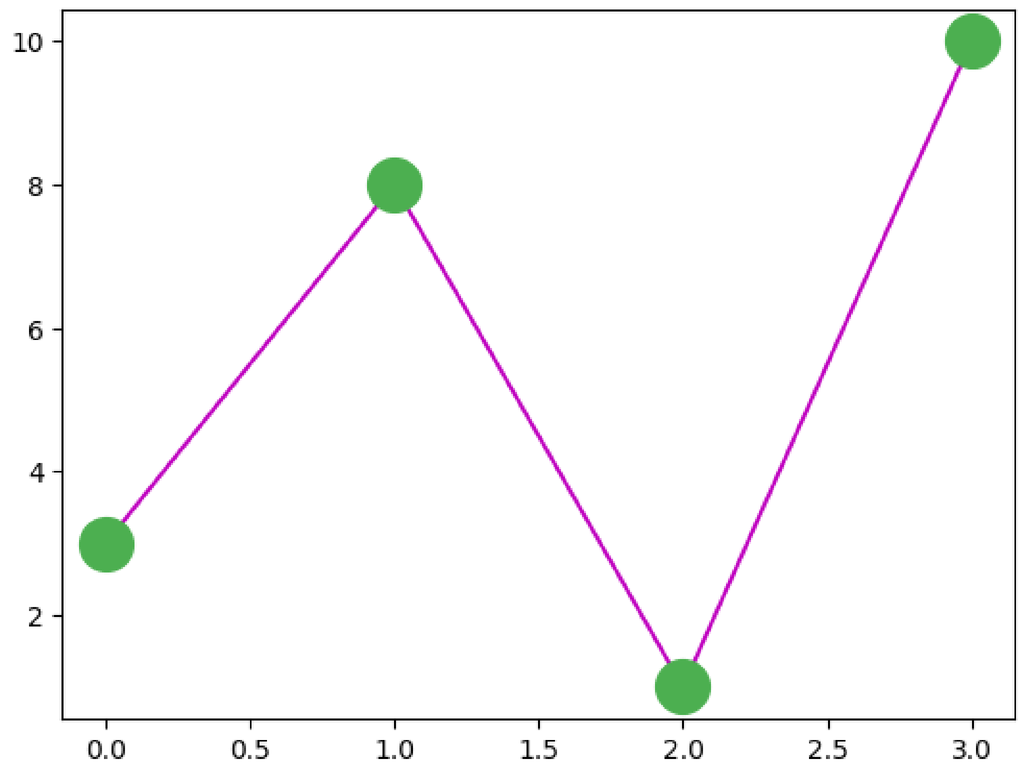
[12]:
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mfc = 'b')
plt.show()
```



- Use both the `mec` and `mfc` arguments to color of the entire marker
- Hexadecimal color values can also be used.

[13]:
```
plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc =␣
 ↪'#4CAF50',color="m")
```

[13]:
```
[<matplotlib.lines.Line2D at 0x7ff41e395c10>]
```



4.3.1 Named Colours

https://matplotlib.org/stable/_images/sphx_glr_named_colors_001.png

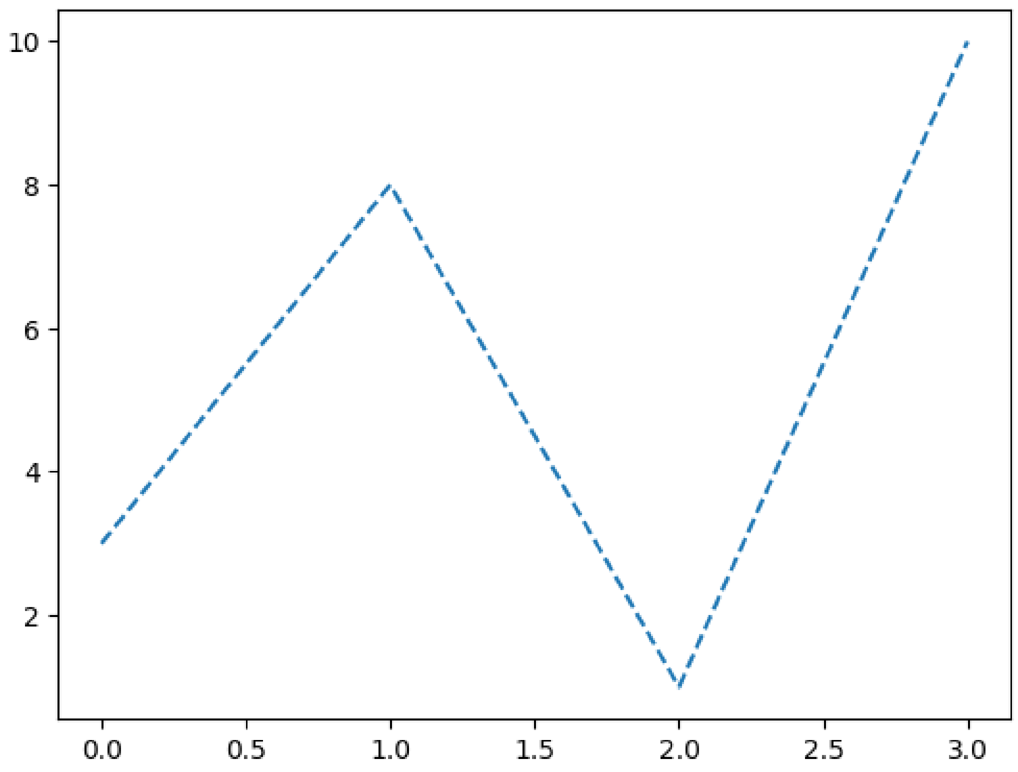
https://matplotlib.org/stable/_images/sphx_glr_named_colors_002.png

https://matplotlib.org/stable/_images/sphx_glr_named_colors_003.png

5 Linestyle

- Keyword argument `linestyle` or shorter `ls` is used to change the style of the plotted line.

[14]:
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dashed')
plt.show()
```

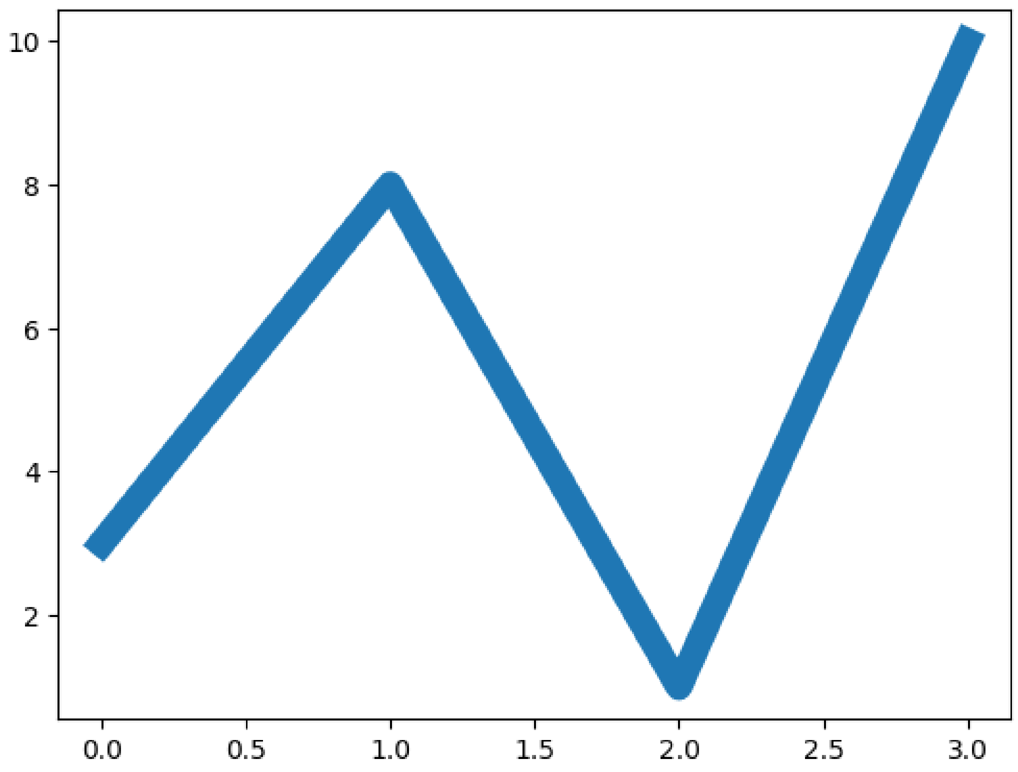


- linestyle:
  - 'solid' (default) or `-`
  - 'dotted' or `:`
  - 'dashed' or `--`
  - 'dashdot' or `-.`

6 Line Width

Keyword argument `linewidth` or the shorter `lw` to change the width of the line.

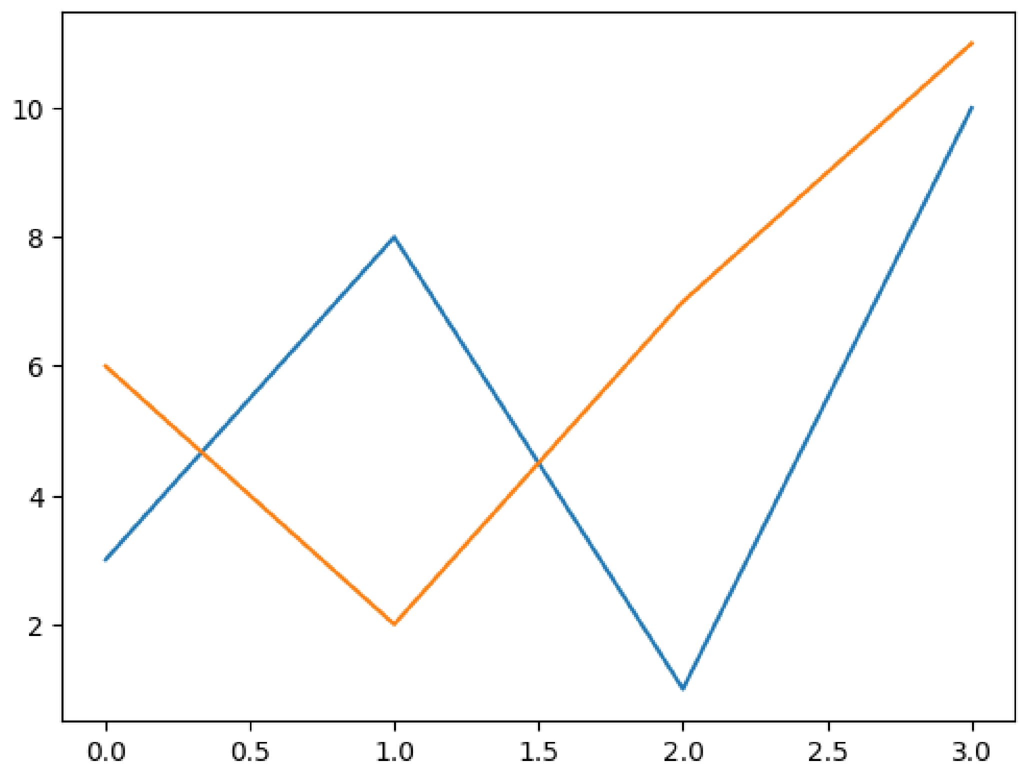
```
[15]: ypoints = np.array([3, 8, 1, 10])

      plt.plot(ypoints, linewidth = '10')
      plt.show()
```



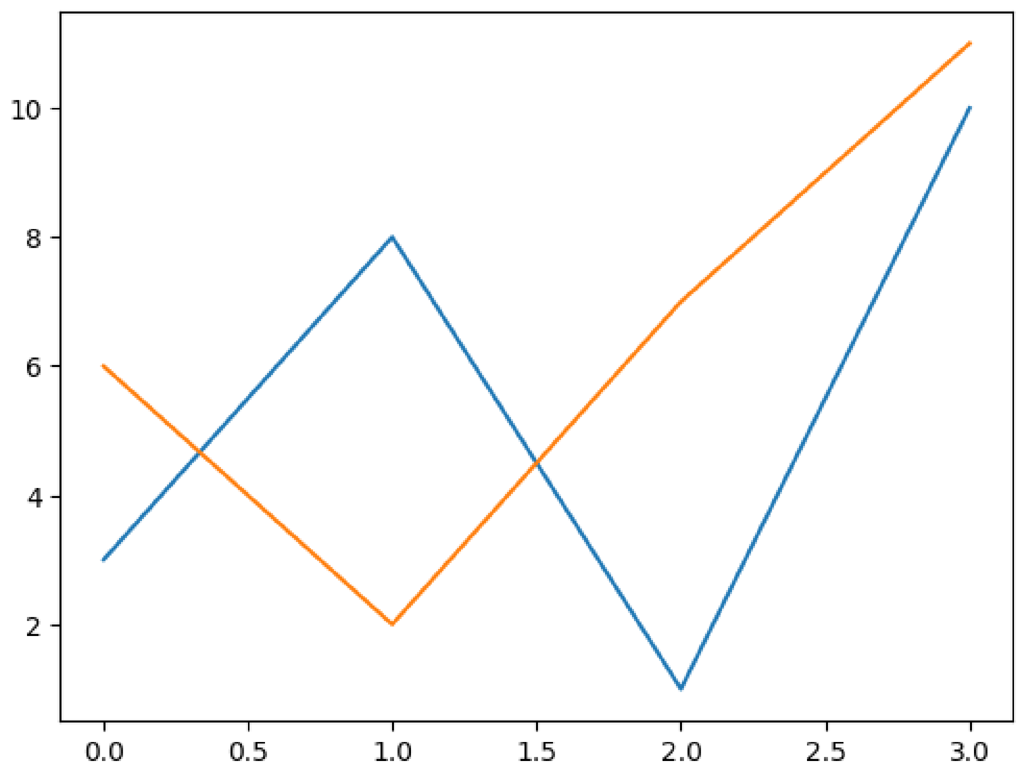
7 Multiple Lines

- By simply adding more plt.plot() functions, more lines can be added in a plot.

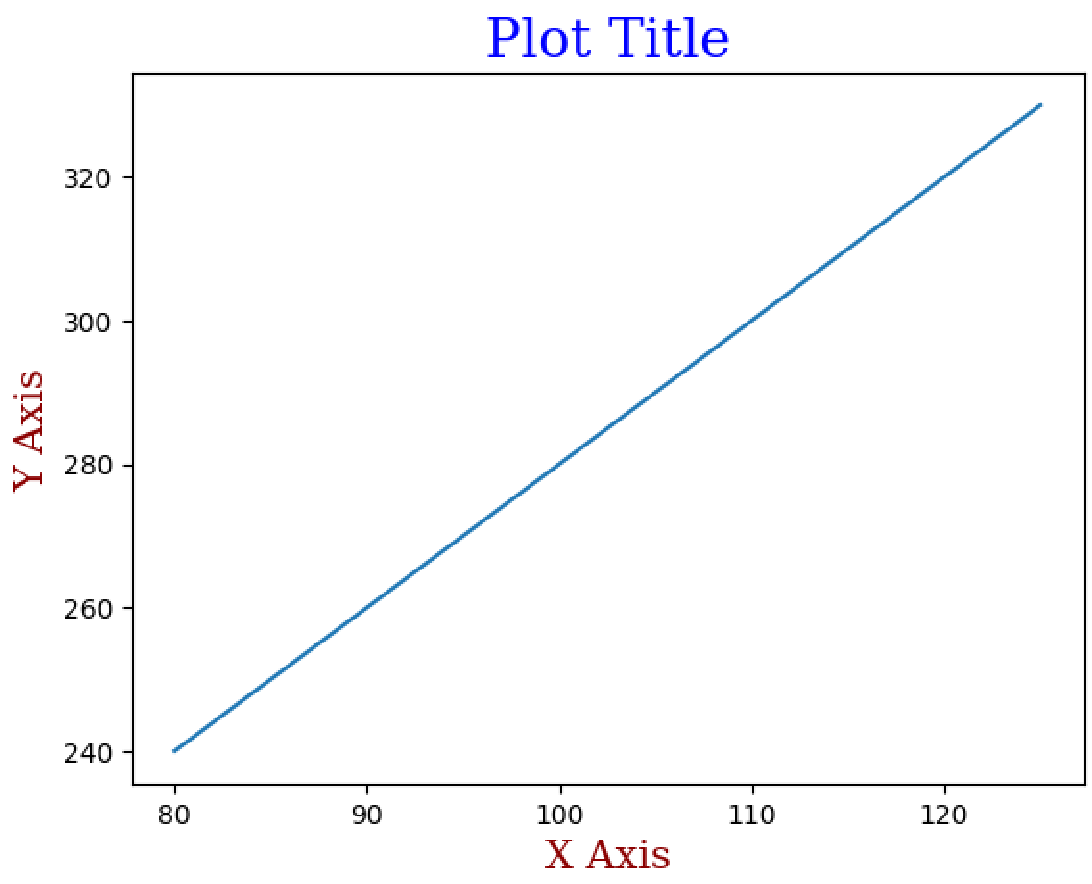
[16]:
```
y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])

plt.plot(y1)
plt.plot(y2)

plt.show()
```



[17]:
```
# Draw two lines by specifiyng the x and y point values for
# both lines in the same plt.plot() function

x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])
x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

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



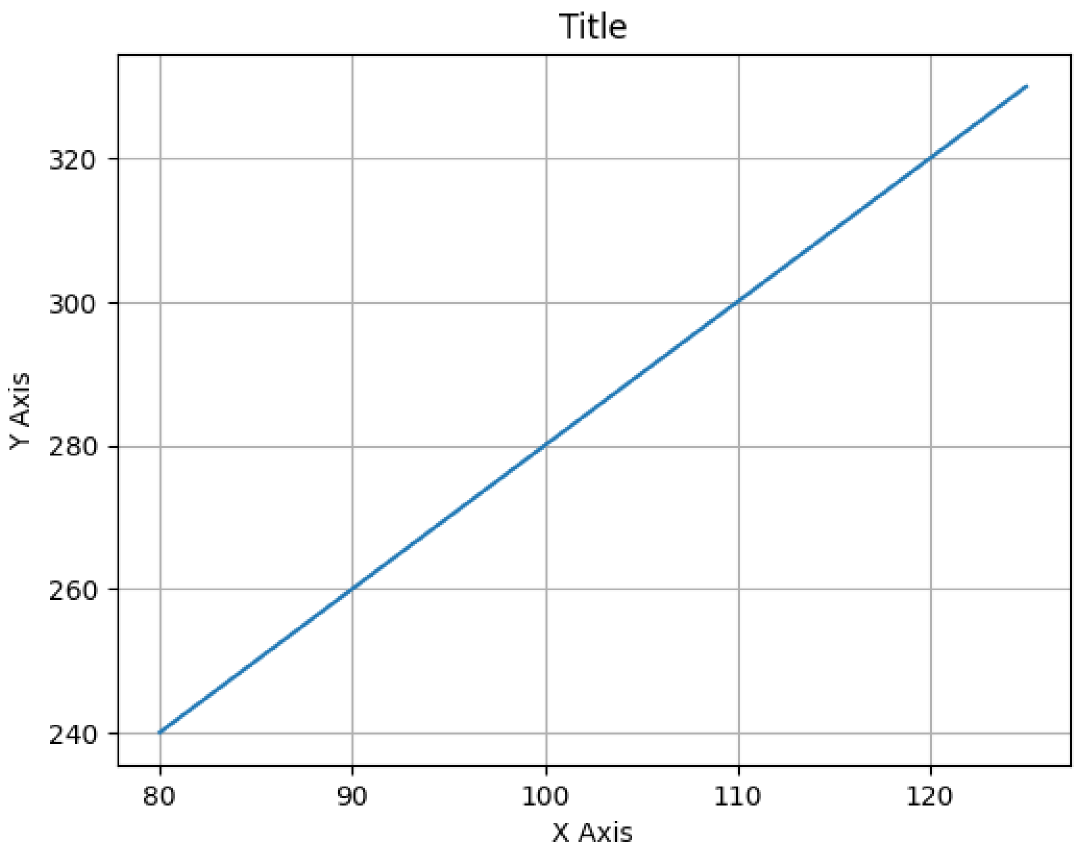
[18]:
```
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("Plot Title", fontdict = font1)#, loc = 'right')
plt.xlabel("X Axis", fontdict = font2)
plt.ylabel("Y Axis", fontdict = font2)

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



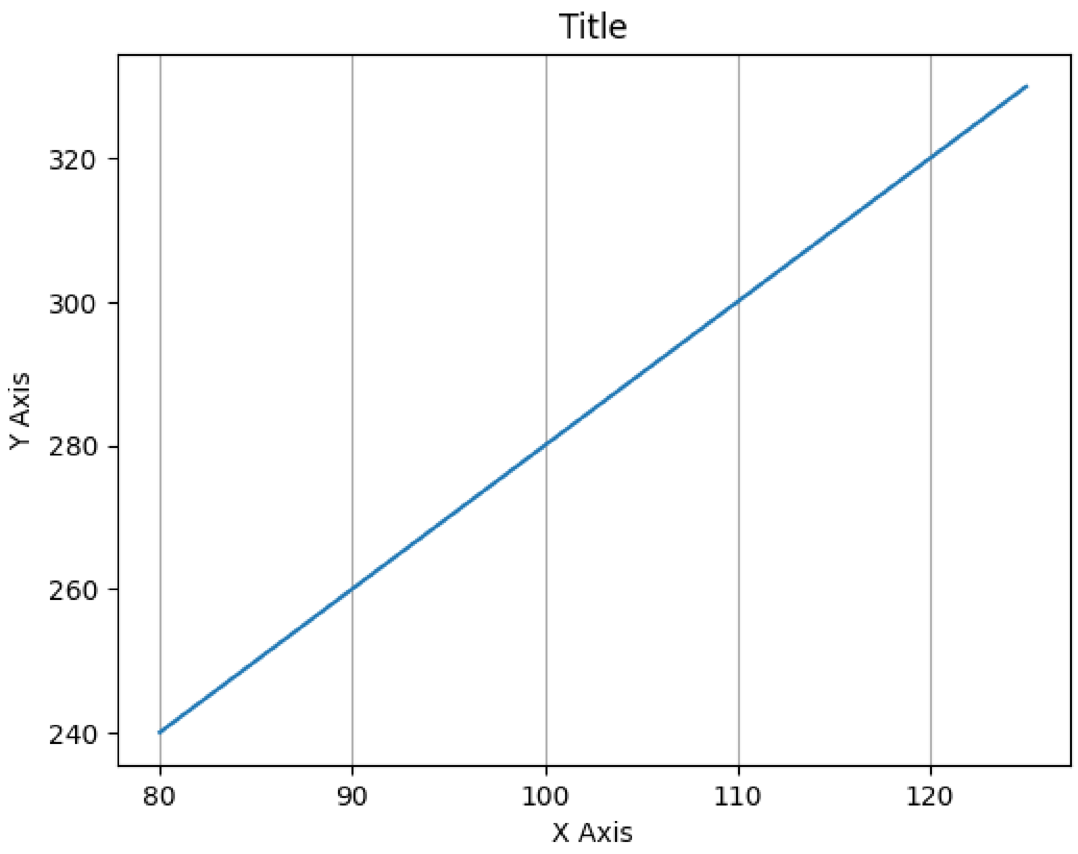
8 Add Grid Lines to a Plot

- Use the `grid()` function to add grid lines to the plot.

[19]:
```python
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Title")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")

plt.plot(x, y)

plt.grid()

plt.show()
```

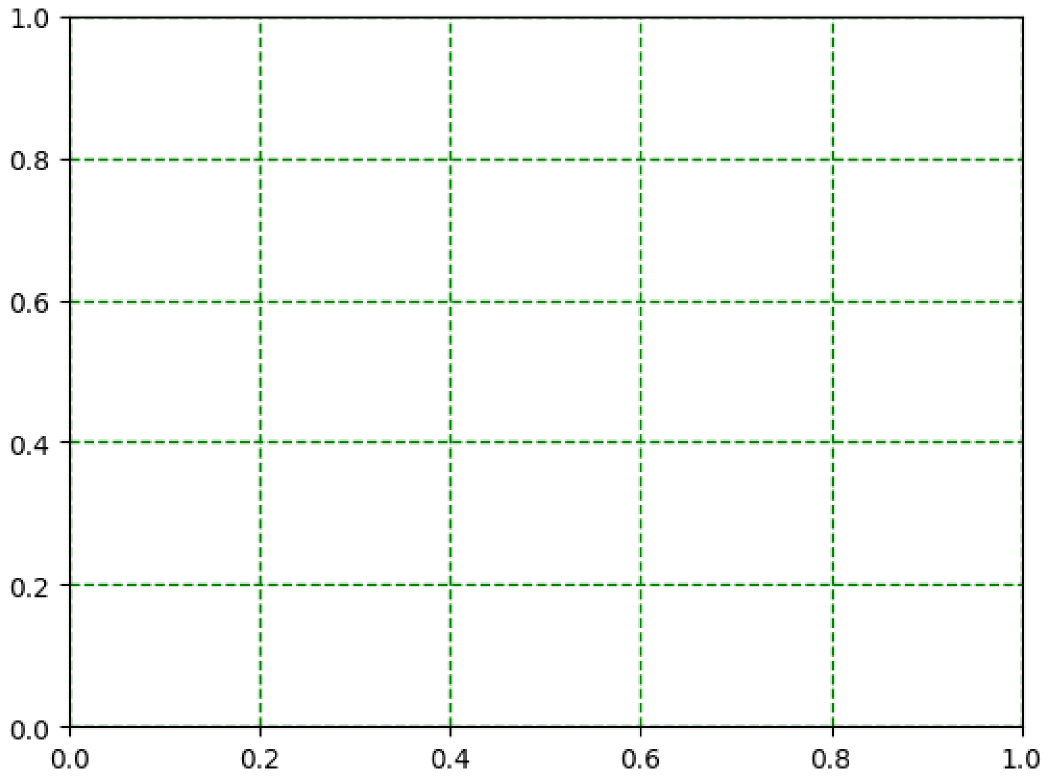


[20]:
```
# Specify Which axis Grid Lines to Display
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Title")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")

plt.plot(x, y)

plt.grid(axis = 'x')

plt.show()
```



8.1 Grid linestyle, linewidth and colour

[21]:
```
plt.grid(color = 'green', linestyle = '--', linewidth = 1)
```

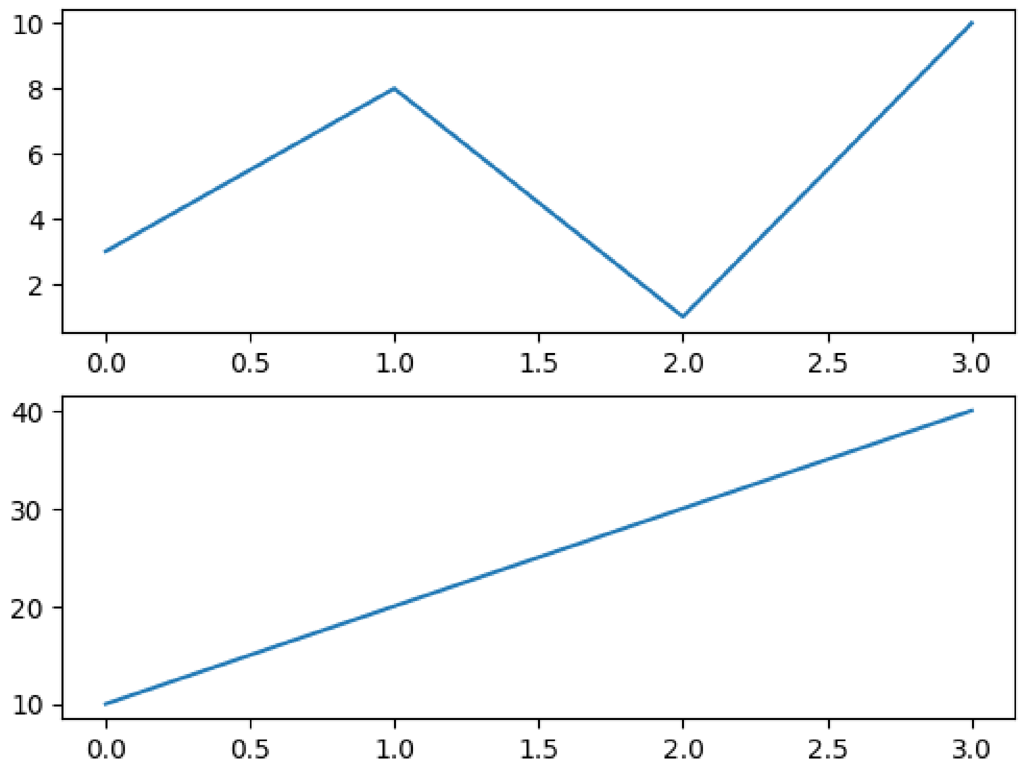


9 Matplotlib Subplots

- Multiple Plots in same image/figure can be displayed using `subplot()` function.
- 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.

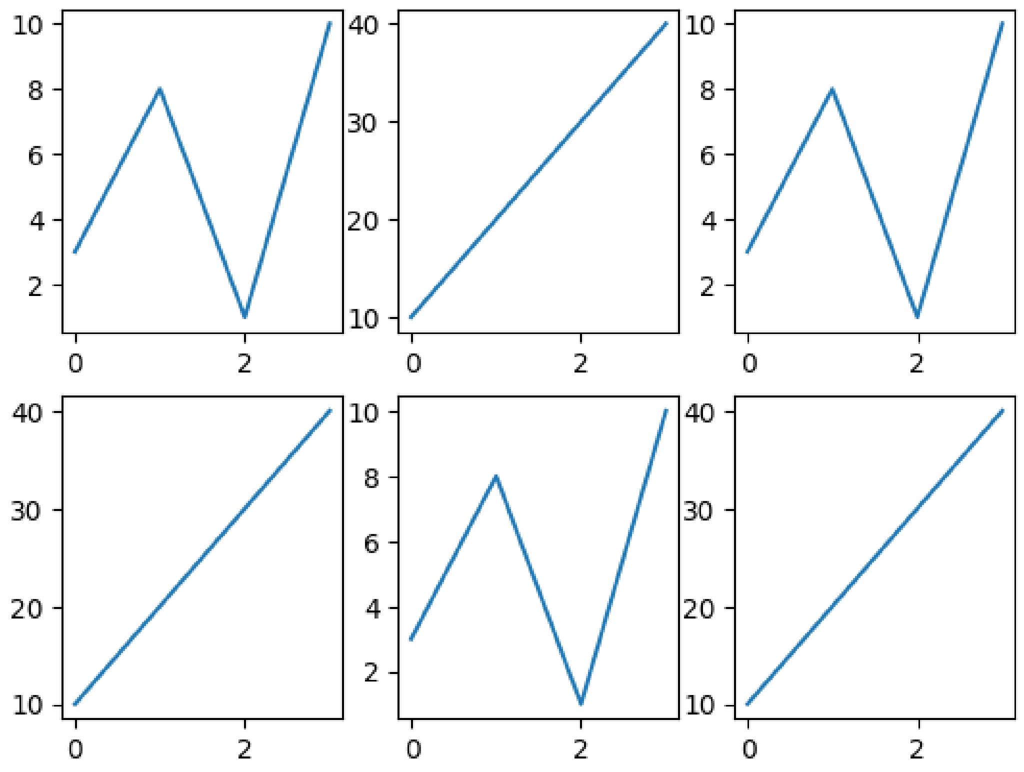
[22]:
```
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1) #the figure has 2 rows, 1 column, & this plot is the first␣
 ↪plot.
plt.plot(x,y)

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

plt.subplot(2, 1, 2) #the figure has 2 rows, 1 column, & this plot is the 2nd␣
 ↪plot.
```

```
plt.plot(x,y)

plt.show()
```

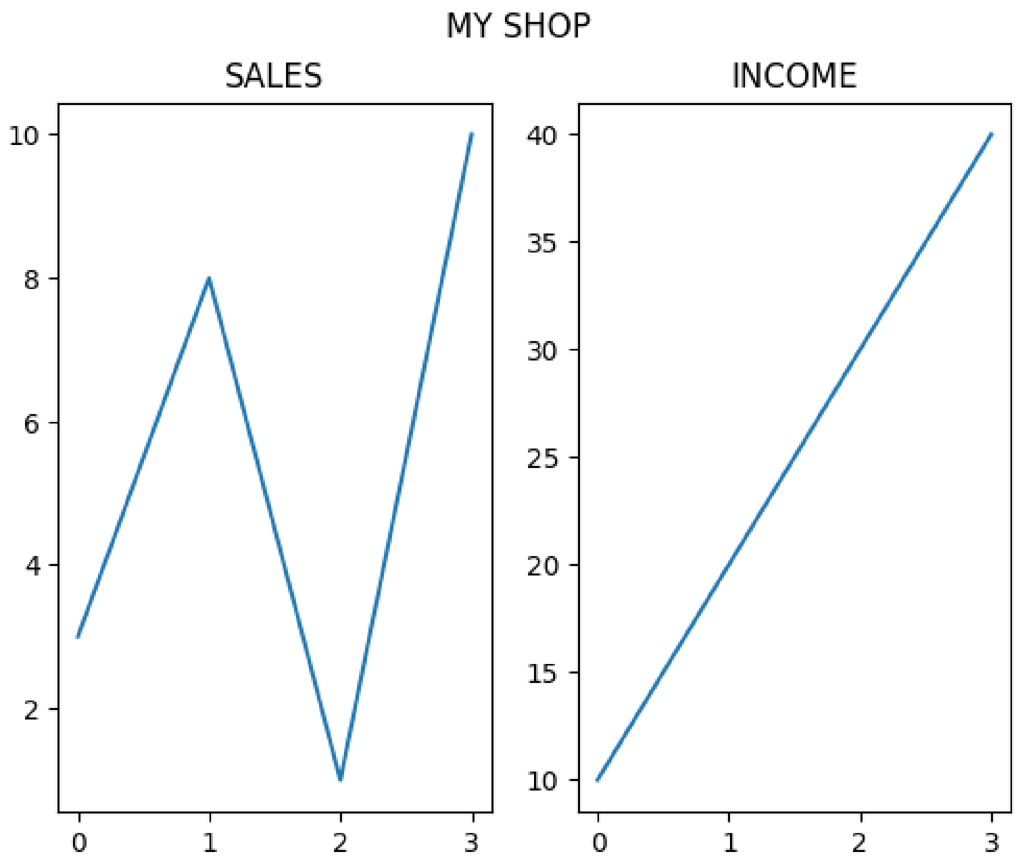


[23]:
```
# 6 plots in one figure
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

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()
```



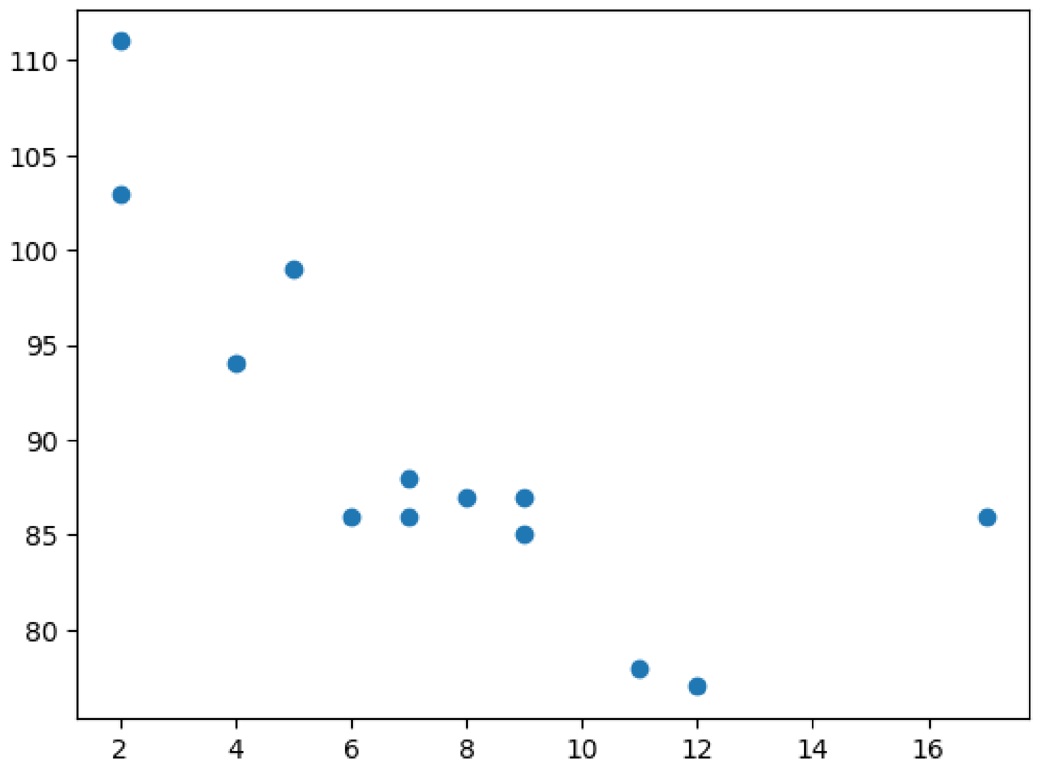
[24]:
```
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

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") #supertitle
plt.show()
```



10 Scatter Plots

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

[25]:
```
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()
```

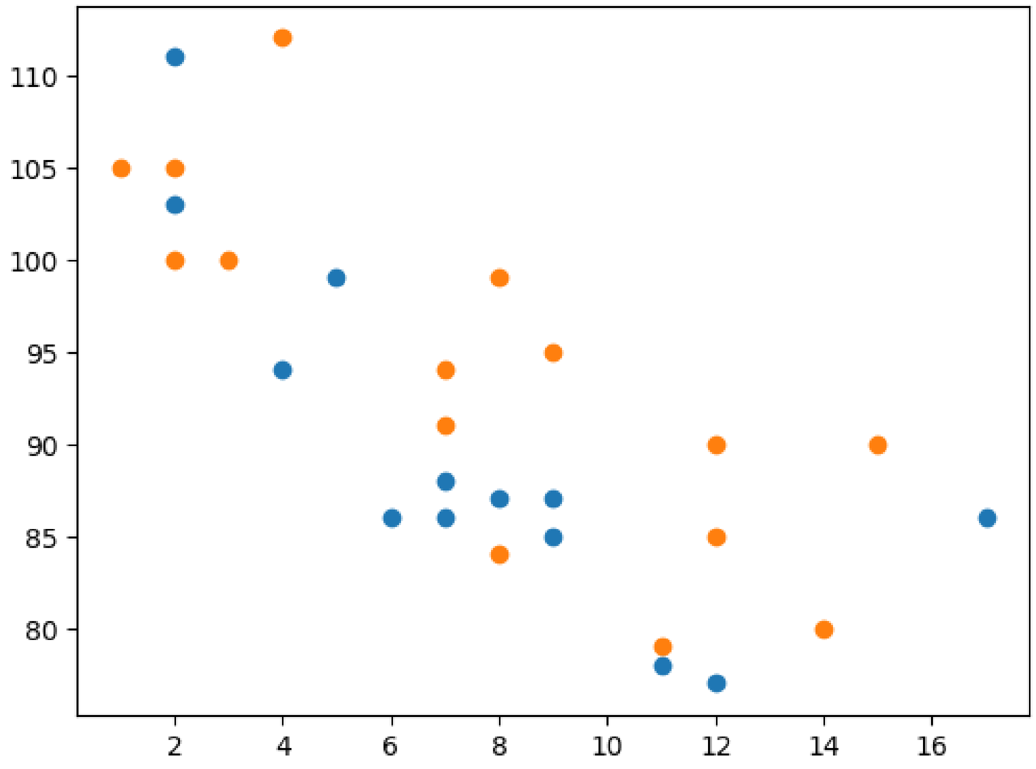


- The observation in the example above is the result of 13 cars passing by.
- The X-axis shows how old the car is.
- The Y-axis shows the speed of the car when it passes.
- Are there any relationships between the observations?
  - It seems that the newer the car, the faster it drives.

10.1 Compare Plots

In the example above, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well? Will the scatter plot tell us something else?

[26]:
```
#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()
```

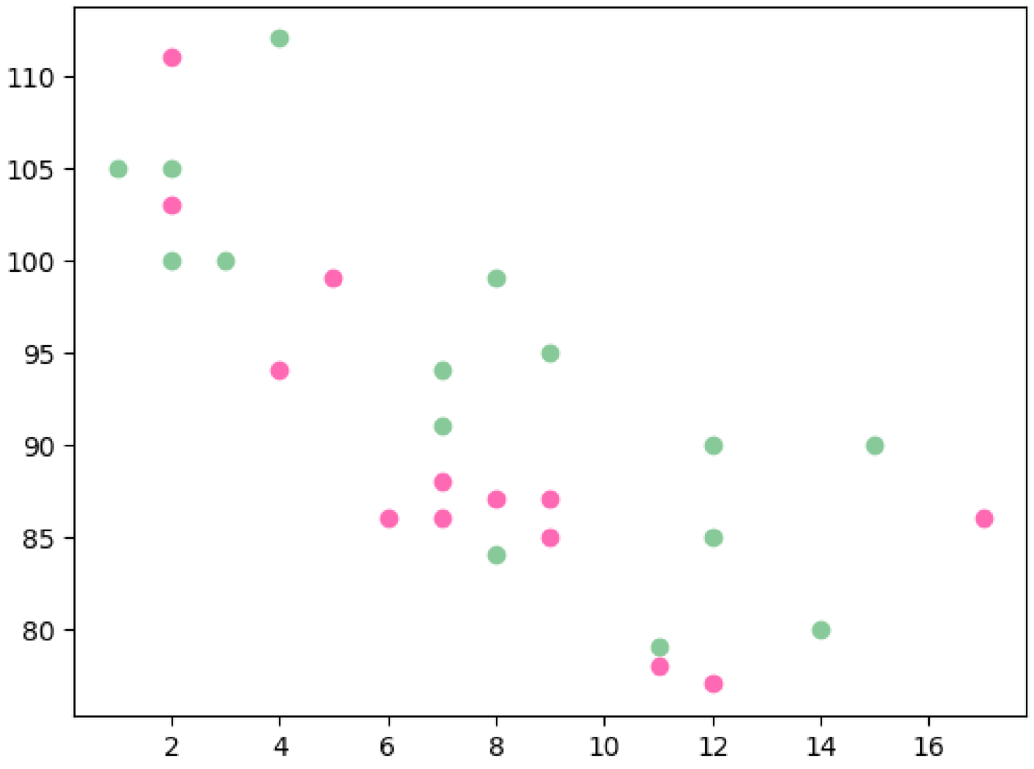


- By comparing the two plots, I think it is safe to say that they both gives us the same conclusion: the newer the car, the faster it drives.

10.2 Marker Colours

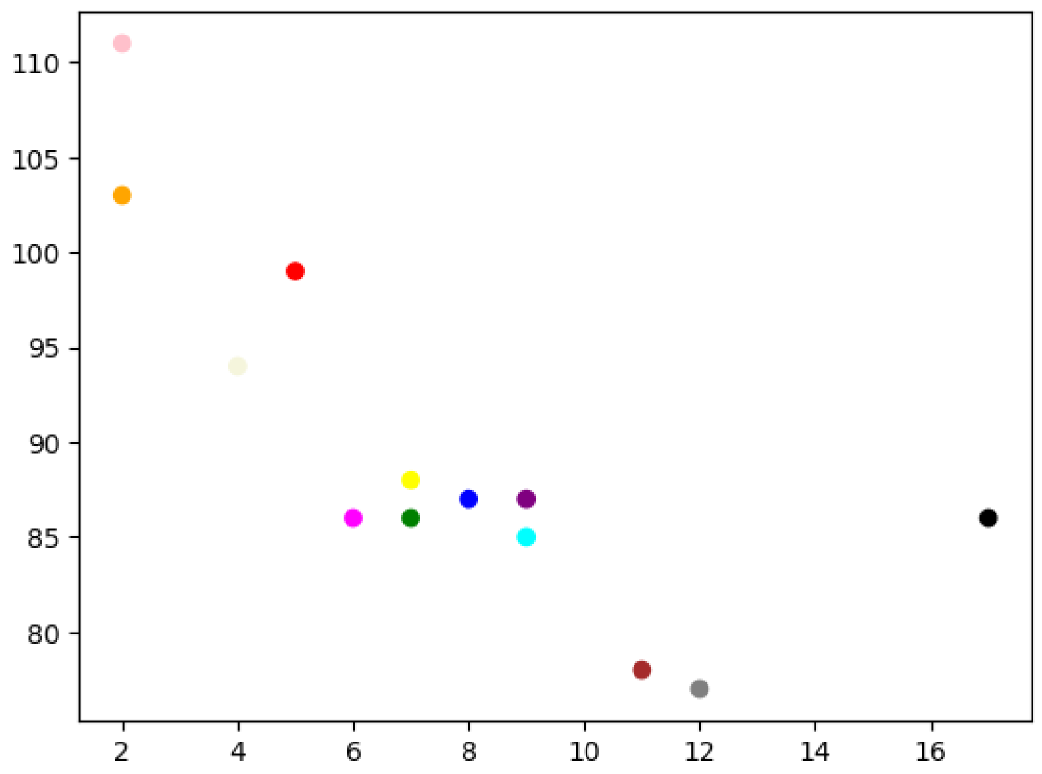
[27]:
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')

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()
```



10.3 Colour Each Dot

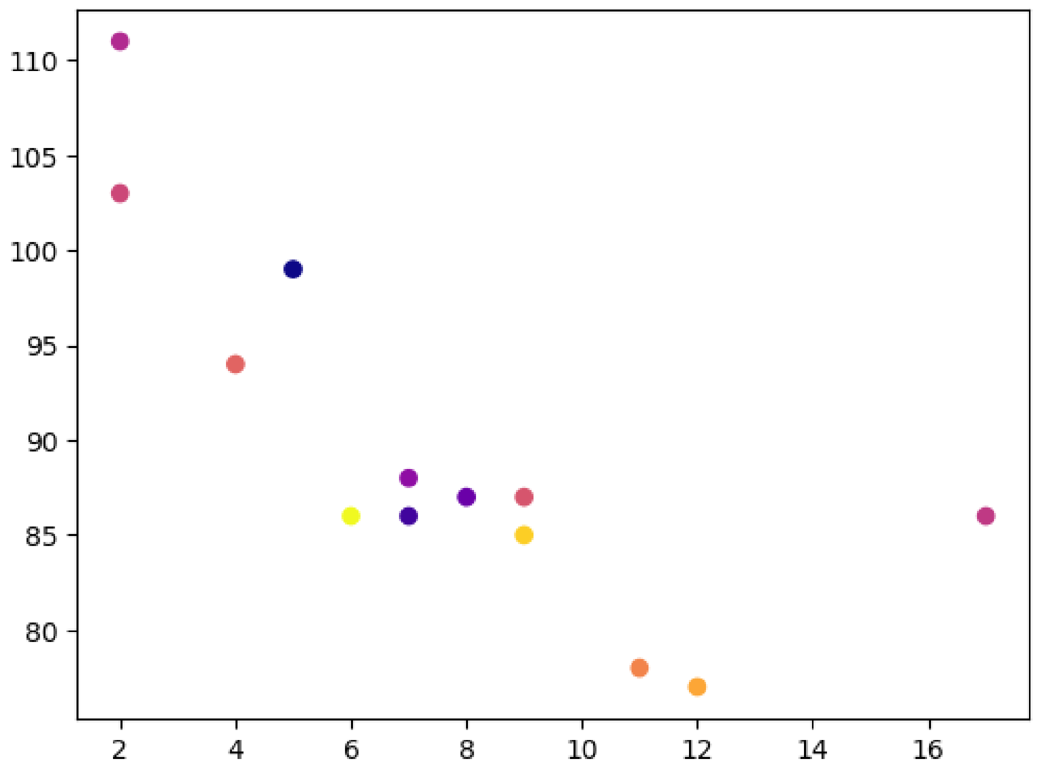
[28]:
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array(["red","green","blue","yellow","pink","black","orange",
                   "purple","beige","brown","gray","cyan","magenta"])

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

plt.show()
```



10.4 Colour Map

- A colormap is like a list of colors, where each color has a value that ranges from 0 to 100.
- The Matplotlib module has a number of available colormaps.
- List of colour maps: click here

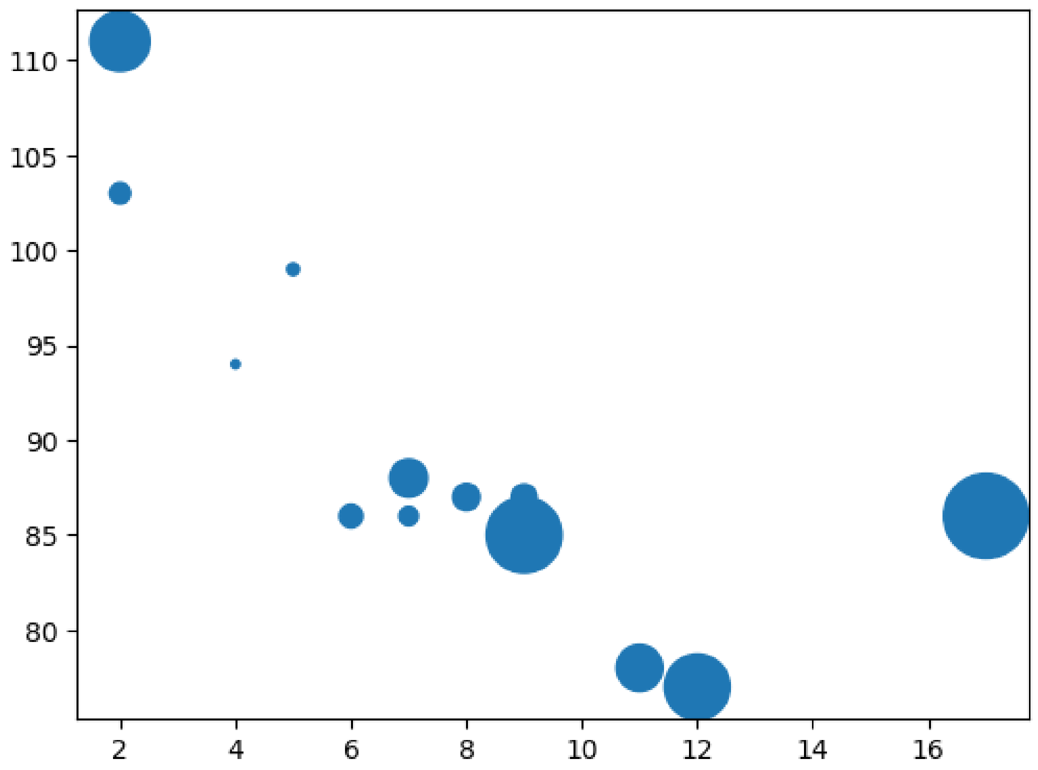
[29]:
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='plasma')
# plt.colorbar()
plt.show()
```



10.5 Size of dots

- Size of the dots can be changed with the `s` argument.
- Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis.

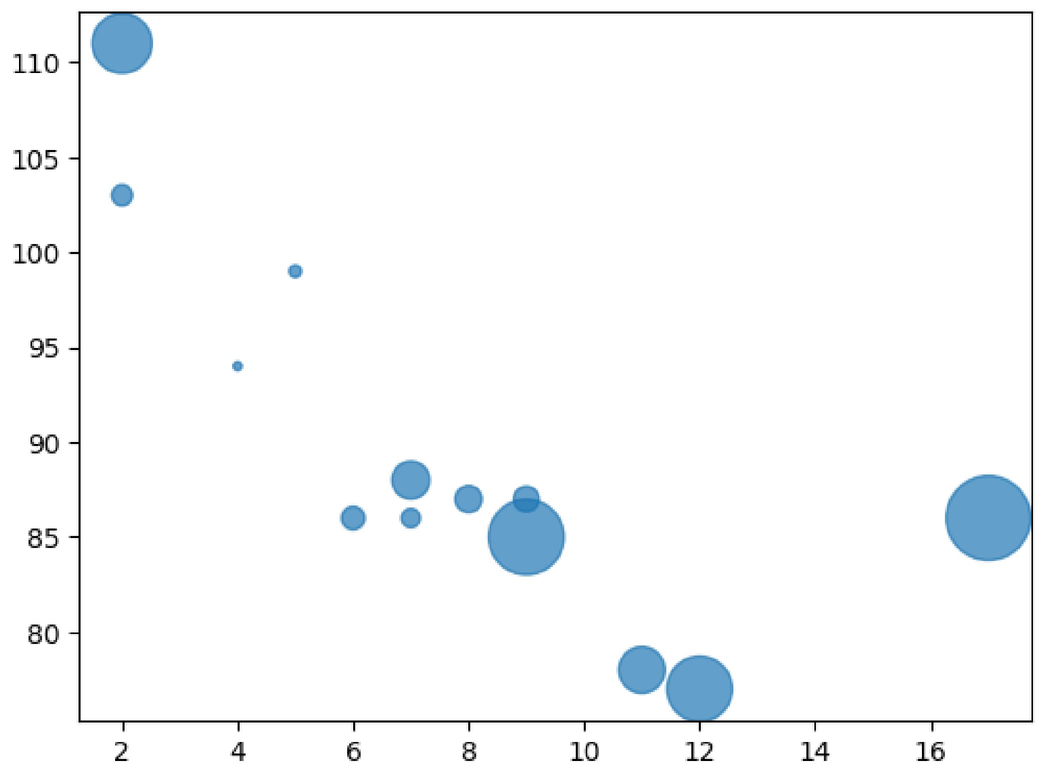
[30]:
```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

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

plt.show()
```



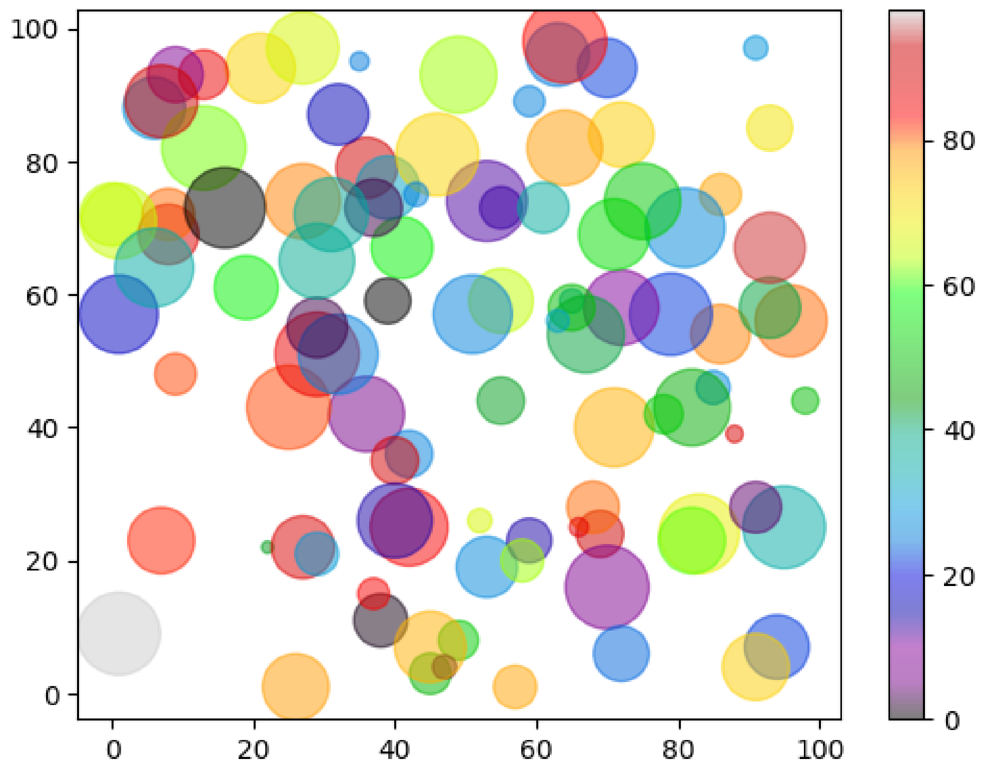
10.6 Transparency of the dots

- Transparency of the dots can be set with the `alpha` argument.

[31]:
```
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=0.7)

plt.show()
```



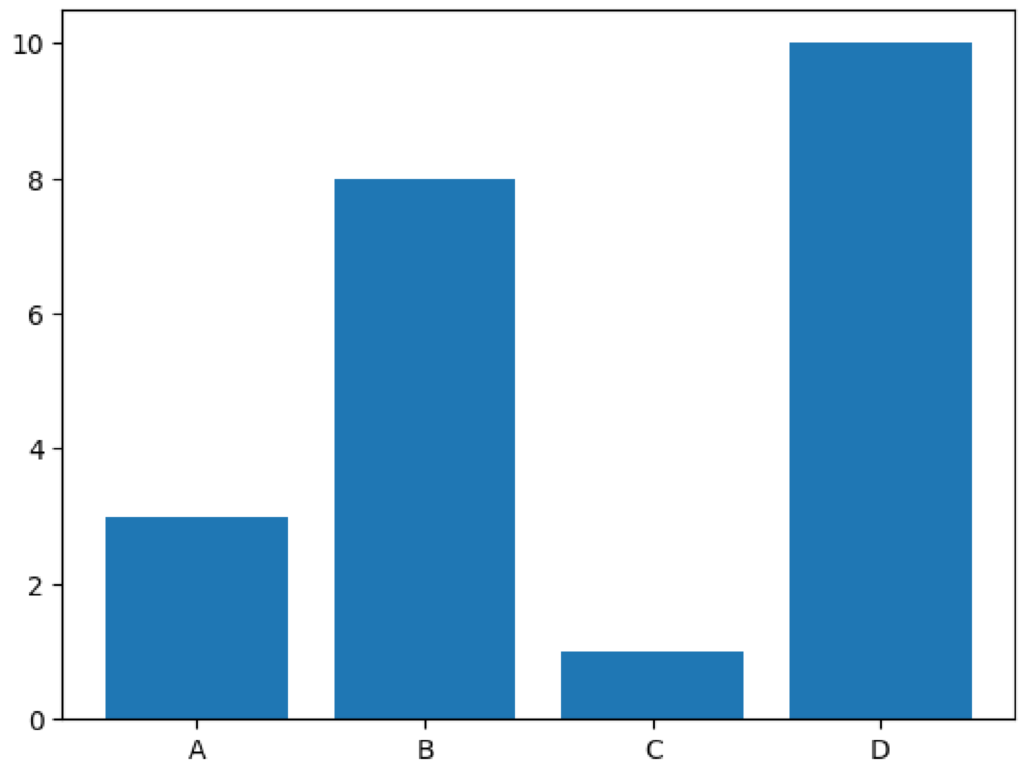
[32]:
```
x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

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

plt.colorbar()

plt.show()
```



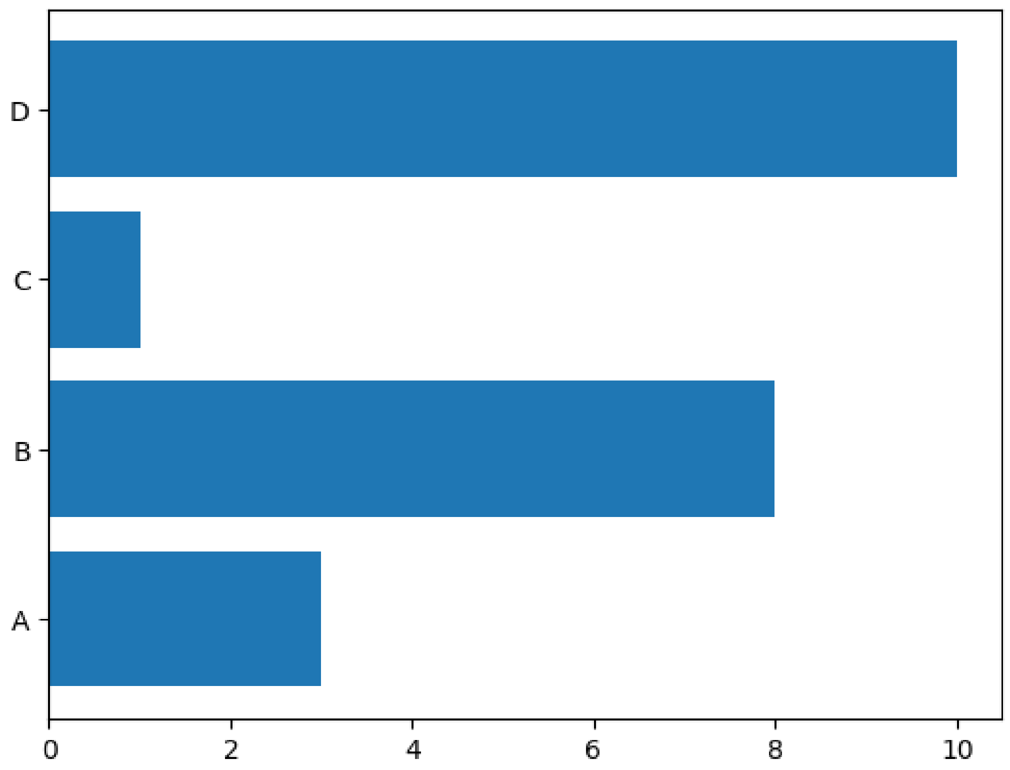
11 Bar Plot

- Using the `bar()` function.
- Use `barh()` for horizontal bar plot.

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

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

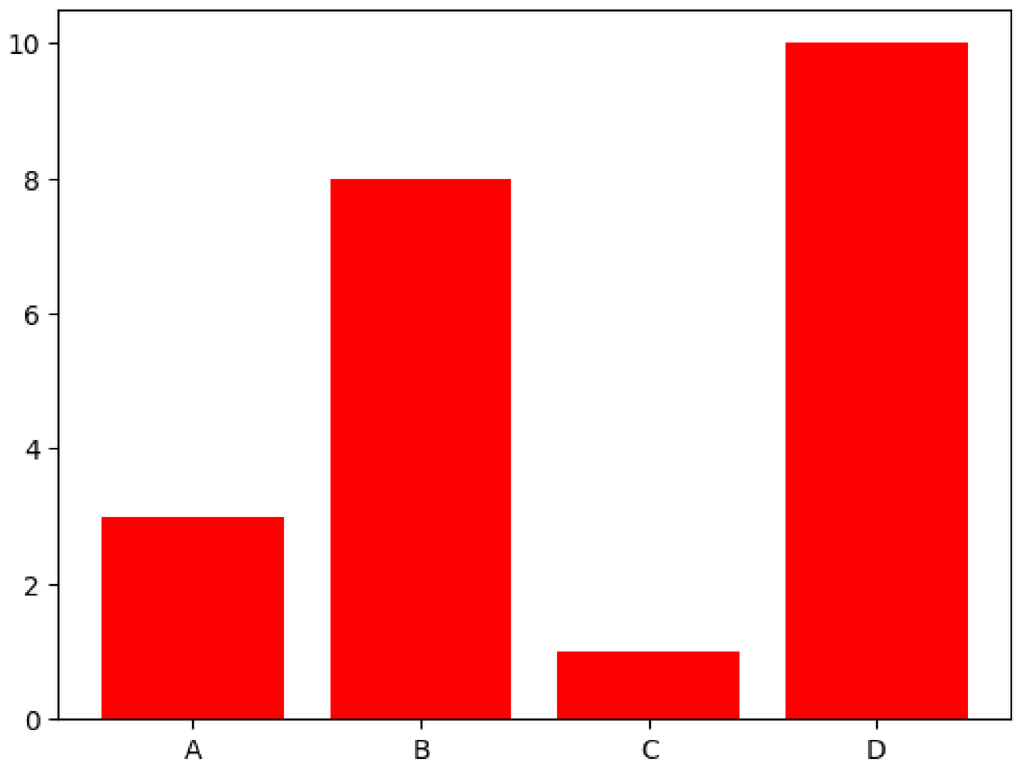


[34]:
```
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

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



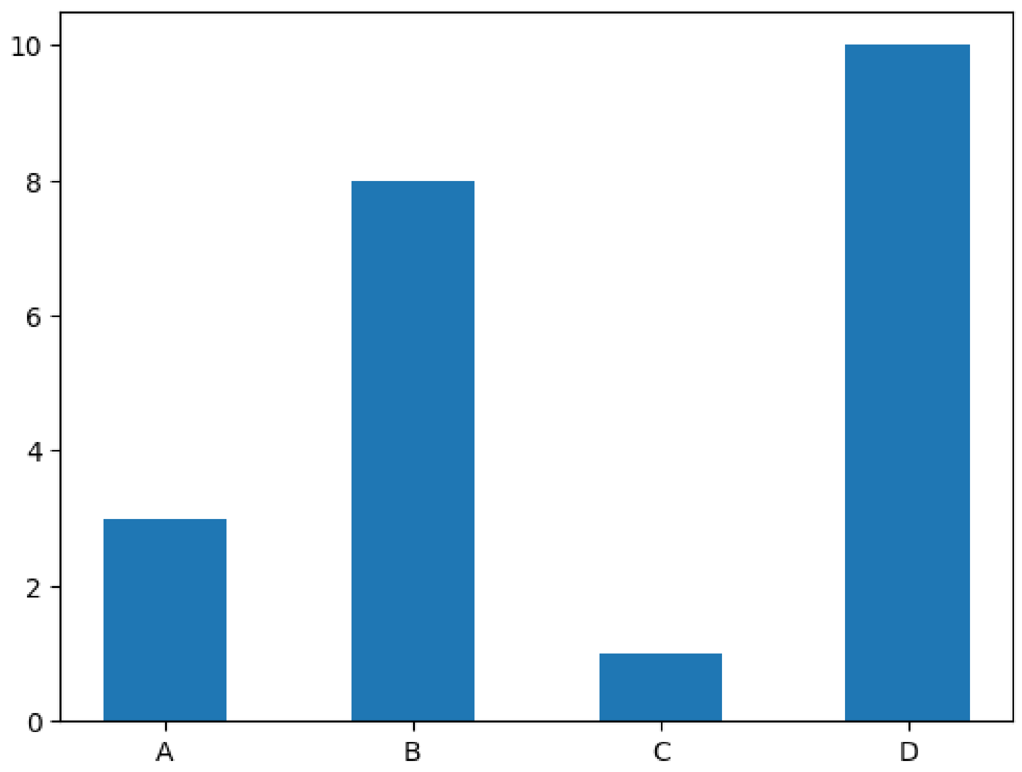
11.1 Bar colours

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

      plt.bar(x, y, color = "red") # can use colornames or hexcodes
      plt.show()
```



[36]:
```
## Bar width
```

[37]:
```
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, width = 0.5)
plt.show()
```



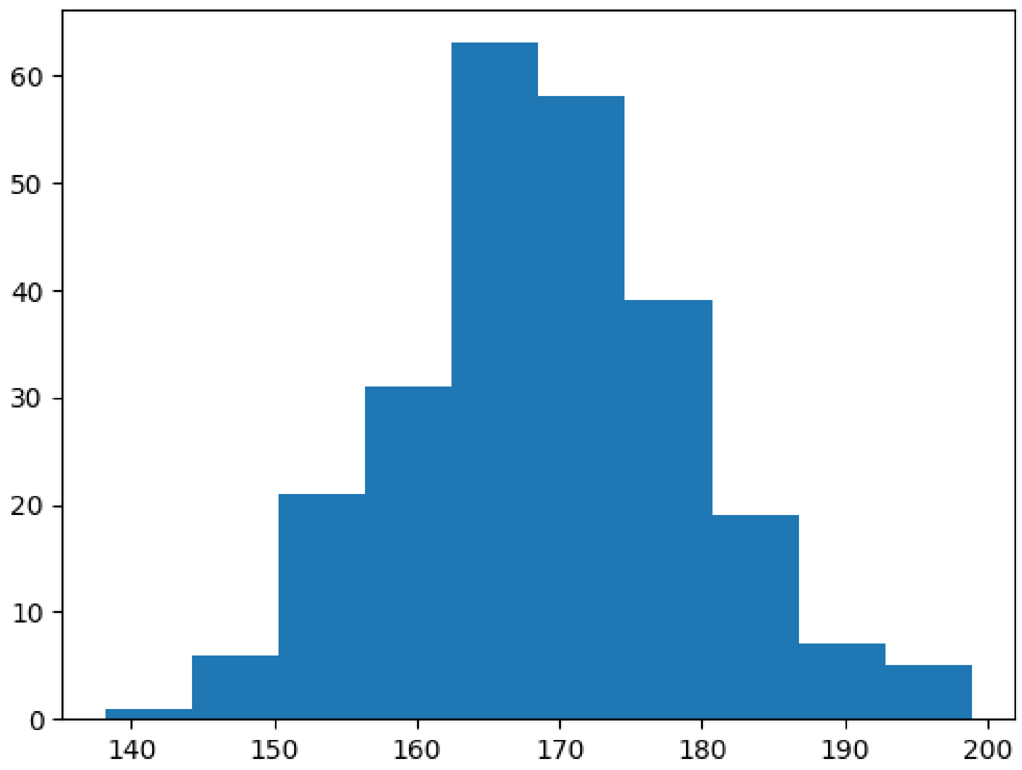
12 Histogram

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

[38]:
```
x = np.random.normal(170, 10, 250)

print(x)
```

```
[181.78296248 188.86639443 179.56341734 171.30588066 158.76223144
 176.35829936 170.31077113 177.54255671 172.96909113 176.30101021
 176.56289783 161.21193381 179.33531659 169.55808316 175.08967886
 169.05775594 183.49929852 173.51354004 166.56228619 163.65174493
 167.49852352 180.97345958 168.20447381 161.50033345 148.44465329
 166.33773999 183.14385922 192.32556449 165.69419882 166.17950668
 164.59045057 174.01188284 187.33827941 163.83560334 171.0603065
 166.85558176 169.95103472 166.65113162 172.55902269 184.5388055
 174.58031004 165.46015018 172.05446089 182.81245294 167.99491774
 174.66274476 173.18880769 172.00898051 181.59637292 172.40521551
 171.54287468 184.2540212  165.66168693 166.59131723 170.74025233
 164.55715374 157.12617225 156.71626733 173.85248397 163.02439649
 175.80623478 167.83340399 156.54597082 184.5636715  165.46931794
 174.24141082 168.04320362 159.37480065 182.41206115 178.08091157
 174.69667458 161.5561223  191.35825071 167.40064574 168.47927494
 178.96681531 160.4278518  177.99803651 168.35115231 167.97578214
 176.10382991 164.18679007 169.34692337 169.01156683 154.98123856
 183.89818417 173.24106634 178.00487927 149.69137748 175.65226972
 166.36951611 192.52996465 153.12996645 155.18549535 172.60273962
 174.97291377 160.97462633 168.00423475 158.34801411 171.48945332
 173.24044756 180.71162617 179.18444478 158.81175818 166.62425847
 154.95733599 179.8481358  166.80155136 178.42077837 166.21245814
 169.73028282 173.25795442 142.9363764  159.41166143 166.39473582
 169.79243829 168.6663418  169.94944181 187.41151917 172.99203474
 157.17128542 180.4736662  173.5806897  177.03467659 173.03337806
 165.3571837  178.29532769 162.84705068 174.14369772 149.42154576
 176.99142627 165.22709338 176.55501028 166.51515534 171.43264913
 161.19499731 162.73027975 146.80173532 169.23888893 188.44695026
 166.53118185 181.2497023  179.96676912 161.33966143 162.8200808
 179.58331121 177.74477461 167.82720431 200.41962927 177.56513405
 173.46516743 159.15474622 178.1744522  169.26231217 181.57569166
 173.93900739 155.78198    175.33468322 159.82265547 188.57415522
 174.09456179 160.9941767  174.87799219 163.91220048 166.32665413
 180.92840732 157.93335398 166.49399226 172.8630794  156.71663878
 159.63618585 176.95342907 165.60866082 173.43701685 172.09116248
 148.46606737 173.47089704 183.93952098 173.99421887 170.94692143
 175.69943593 160.46661961 157.10849217 178.18469906 188.0788699
 170.58344815 160.64350762 171.11816885 177.10279    186.17415094
 163.33504375 164.92204089 164.93395235 154.55830931 183.73610613
 157.79416214 162.32868087 181.27740835 172.57752245 166.91631097
 159.35101083 155.46046604 165.47423168 163.24646203 154.91024992
 169.48896079 167.1290409  163.57598325 166.15500159 170.18700514
 169.3728887  180.34388084 166.46271668 173.38488241 172.42290629
 167.88869915 169.62349305 171.50299946 167.85192283 187.36303148
 154.85264854 162.73629751 164.34449823 158.5600208  173.38969438
 169.91519307 173.7293449  172.88803862 182.59179337 148.79021692
 178.59842649 176.22468466 176.34563967 174.15147317 164.78744192
 169.98907439 174.25416506 175.98967147 173.03650484 179.80162053
 160.32194852 181.68996924 176.02886756 172.44480432 175.95143147
 184.16928385 158.33091141 154.3381366  176.08780358 167.71893103]
```

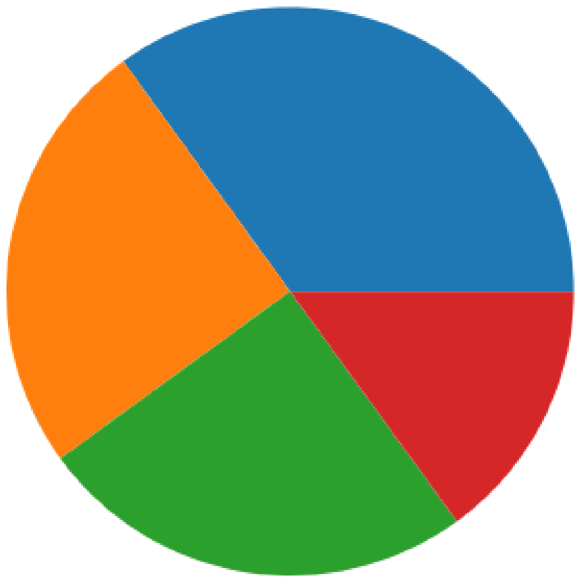
[39]:
```
x = np.random.normal(170, 10, 250)

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



Say you ask for the height of 250 people, you might end up with a histogram like this: 2 people from 140 to 145cm 5 people from 145 to 150cm 15 people from 151 to 156cm 31 people from 157 to 162cm 46 people from 163 to 168cm 53 people from 168 to 173cm 45 people from 173 to 178cm 28 people from 179 to 184cm 21 people from 185 to 190cm 4 people from 190 to 195cm

13 Pie Charts

[40]:
```
y = np.array([35, 25, 25, 15])

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

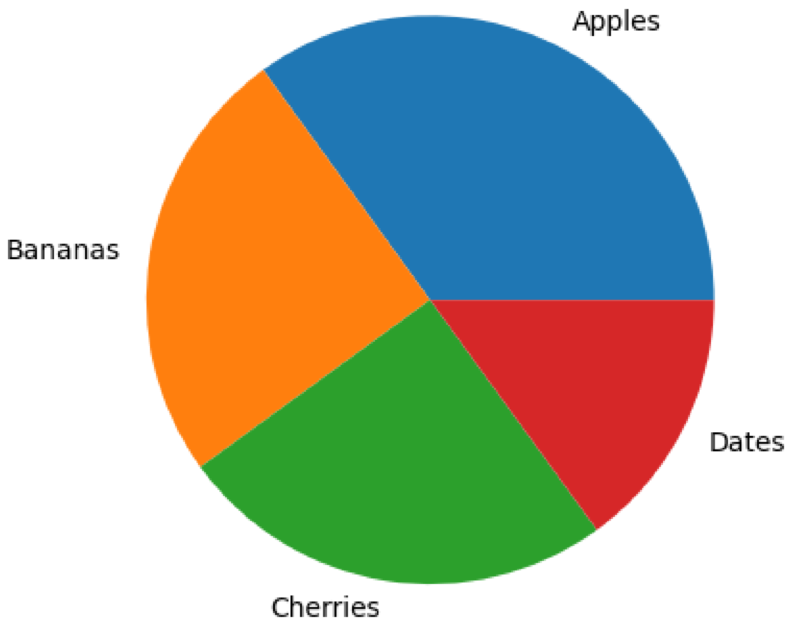


- By default the plotting of the first wedge starts from the x-axis and move counterclockwise

13.1 Labels

- Add labels to the pie chart with the `label` parameter.

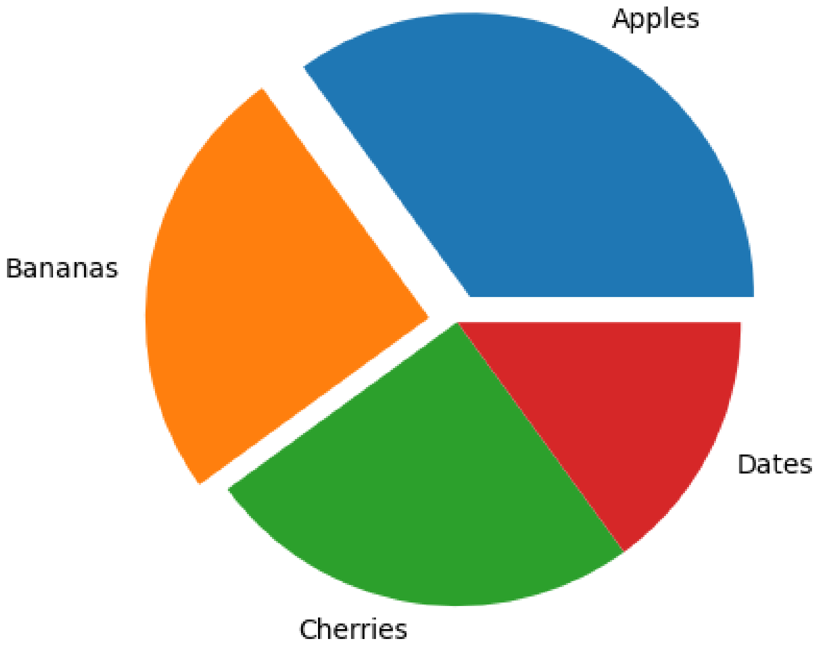
```
[41]: y = np.array([35, 25, 25, 15])
      mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

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



13.2 Explode

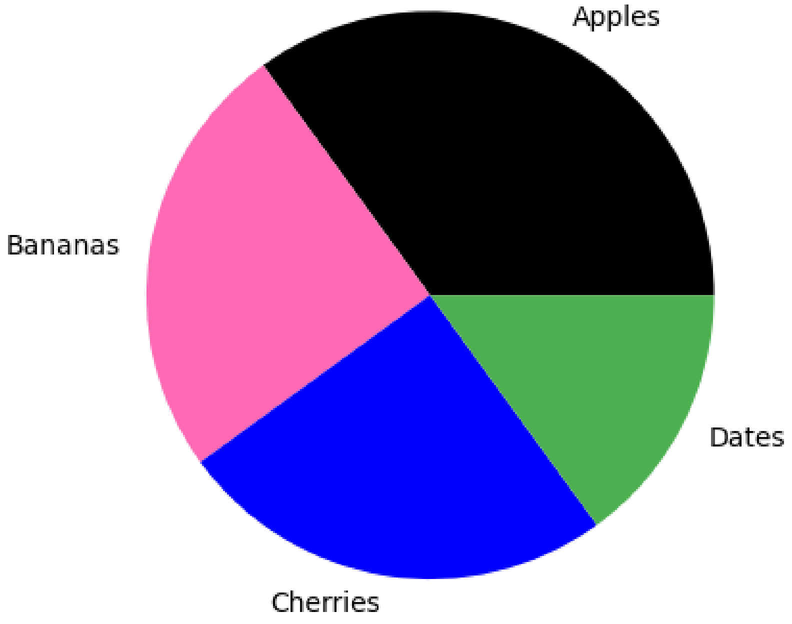
- The explode parameter allows to make one of the wedges to stand out .

[42]:
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.1, 0.1, 0, 0]

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



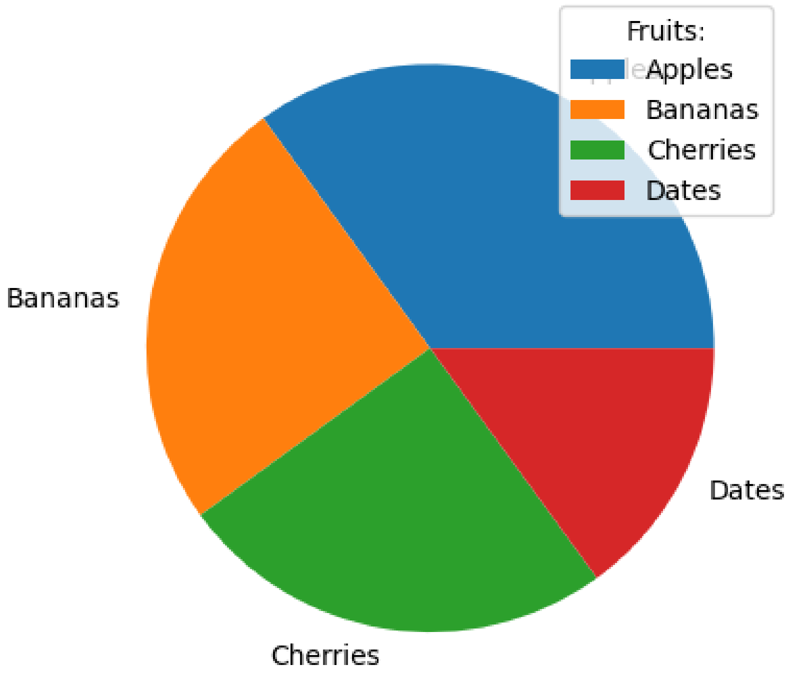
13.3 Colors

[43]:
```
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()
```



```
[44]: ## Legend
```

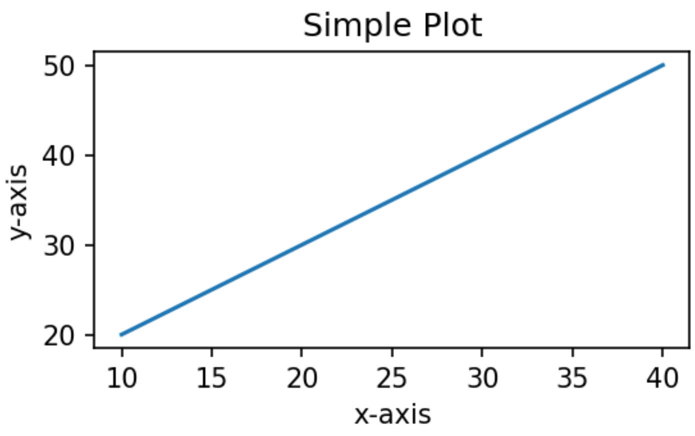
```
[45]: y = np.array([35, 25, 25, 15])
      mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

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



14 figure()

- The `figure()` function in pyplot module of matplotlib library is used to create a new figure.

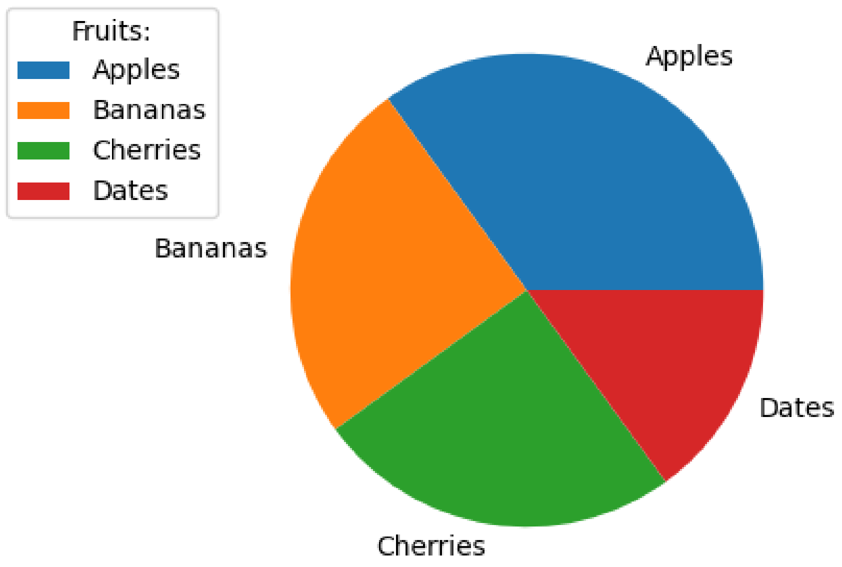
[46]:
```
fig = plt.figure(figsize =(4,2), dpi=150) # size in inches

x = [10, 20, 30, 40]
y = [20, 30, 40, 50]

# plotting the data
plt.plot(x, y)

# Adding the labels
plt.ylabel("y-axis")
plt.xlabel("x-axis")

# Adding the title
plt.title("Simple Plot")

plt.show()
```



Parameters of `figure()`

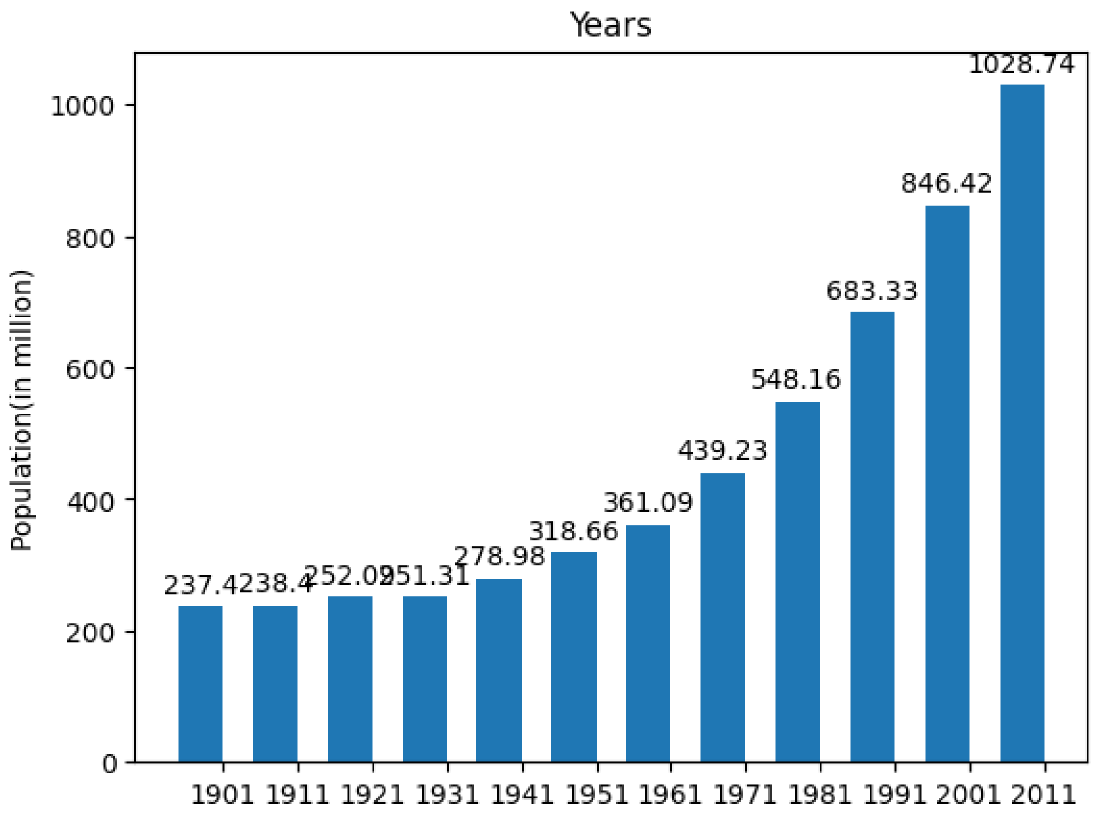
- figsize(float, float): These parameter are the width, height in inches.
- dpi : This parameter is the resolution of the figure.
- facecolor : This parameter is the the background color.
- edgecolor : This parameter is the border color.
- clear : This parameter if True and the figure already exists, then it is cleared.

[47]:
```
fig = plt.figure(figsize =(4, 4), dpi=100, facecolor="w")

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

plt.pie(y, labels = mylabels)
plt.legend(title = "Fruits:",loc="upper right", bbox_to_anchor=(0,1))
plt.savefig("pie.png",dpi=300,bbox_inches="tight") # save plot to disk
plt.show()
```



15 Bar Chart with Annotations

```
[48]: years = [1901, 1911, 1921, 1931, 1941, 1951, 1961, 1971, 1981, 1991, 2001, 2011]
      population = [237.4, 238.4, 252.09, 251.31, 278.98, 318.66, 361.09, 439.23, 548.
       ↪16, 683.33, 846.42, 1028.74]

      x = np.arange(len(years)) # the label locations #12
      width = 0.6 # the width of the bars

      fig, ax = plt.subplots()

      ax.set_ylabel('Population(in million)')
      ax.set_title('Years')
      ax.set_xticks(x)
      ax.set_xticklabels(years)

      pps = ax.bar(x - width/2, population, width, label='population')
      for p in pps:
          height = p.get_height()
          ax.annotate('{}'.format(height),
            xy=(p.get_x() + p.get_width() / 2, height),
            xytext=(0, 3), # 3 points vertical offset
            textcoords="offset points",
            ha='center', va='bottom')

      plt.show()
```



- Explore:
  - https://matplotlib.org/stable/gallery/index.html
  - https://www.pythoncharts.com/