```
df.loc[[0,1],['b','d']]
'''
  b   d
0 23  23
1 33  65
'''

#RUNNING QUERIES USING LOC[]
df.loc[(df['a']>10) & df['c']<50]
'''
  a   b   c   d
0 12  23  44  23
'''

#LOCATE USING INDEX iloc[]
df.iloc[0,2]
#44


#REMOVE ROW/COLUMN  drop()
df.drop(4,axis=1)   #removes column with label=4
df.drop(3,axis=0)   #removes row with index=3
```

💡 Always use `loc[[row],:]` or `iloc[[rowindex],:]` function to set values rather than direct `df[row] = value`, to avoid view/copy ambiguity

Reading data from CSV, Excel:

```
#IMPORT DATA FROM CSV OR EXCEL FILE
df = pd.read_csv("file.csv")
df = pd.read_excel("data.xlsx")

#EXPORT DATAFRAME TO CSV/EXCEL FILE
df = pd.DataFrame(np.random.randint(10,100,size=[10,5]))
df.columns = ['A','B','C','D','E']
df.to_csv("file.csv")
```

Matplotlib:

- Matplotlib is a low level graph plotting library in python that serves as a visualization utility
- Matplotlib is mostly written in python, a few segments are written in C

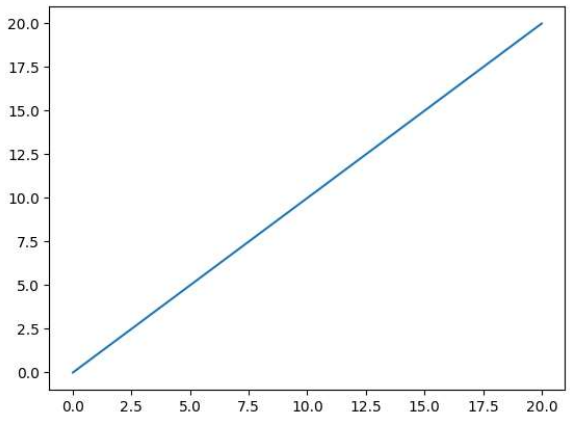
```
import matplotlib.pyplot as plt
```

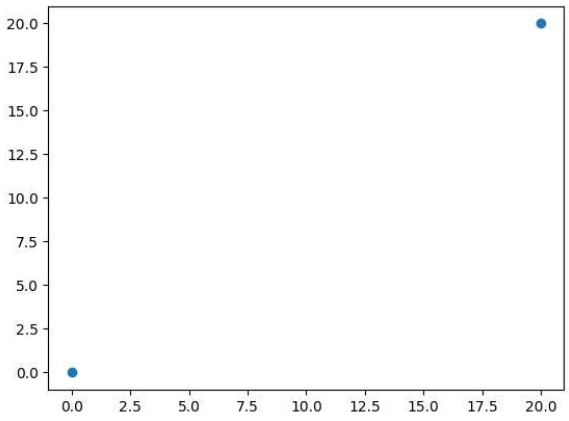
- Types of graphs are:
  - line graph
  - bar graph
  - scatter graph
  - pie chart
  - Histograms and more…

Plot function:

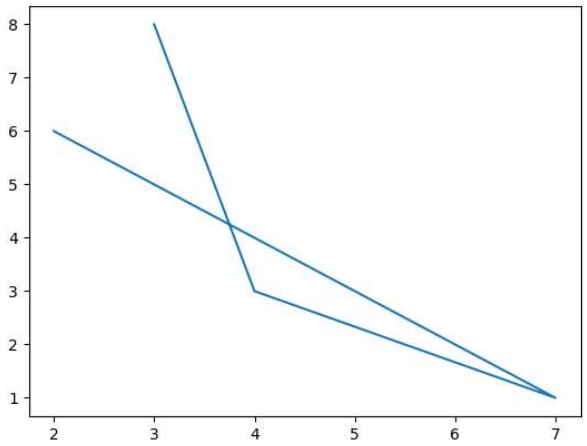
- The `plot()` function is used to draw points (markers) in a diagram
- By default, the `plot()` function draws a line from point to point
- It takes input as - numpy array or list of x-points and y-points
- To display the plot, use `plt.show()`

```
xpoints = [0,20]   #points on x-axis
ypoints = [0,20]   #points on y-axis
#so, it draws a line from (0,0) to (20,20)

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

#To plot only the points, you can use shortcut parameter 'o'
#which means 'rings'.

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

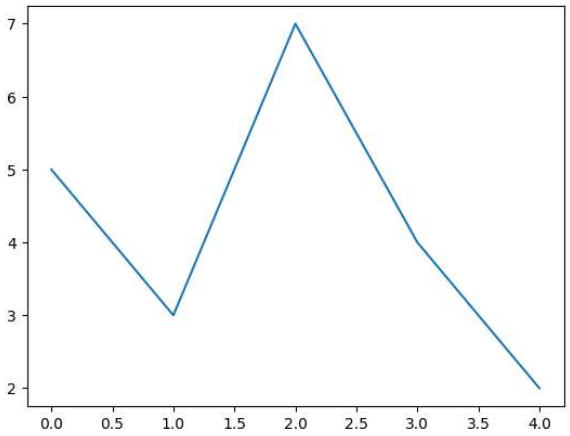





```
#Multiple points
plt.plot([3,4,7,2],[8,3,1,6])
plt.show()

#Default x-points
'''If we do not specify the points on the x-axis,
they will get the default values 0,1,2,3...'''
plt.plot([5,3,7,4,2])
plt.show()
```

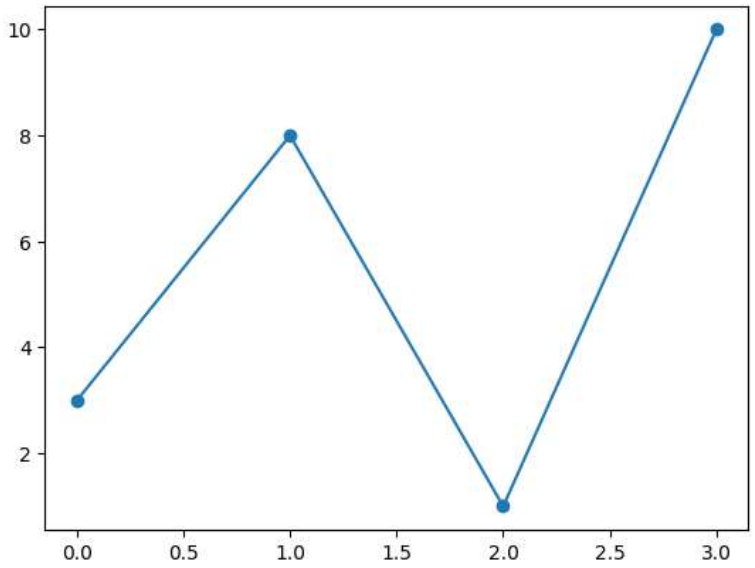


Multiple points



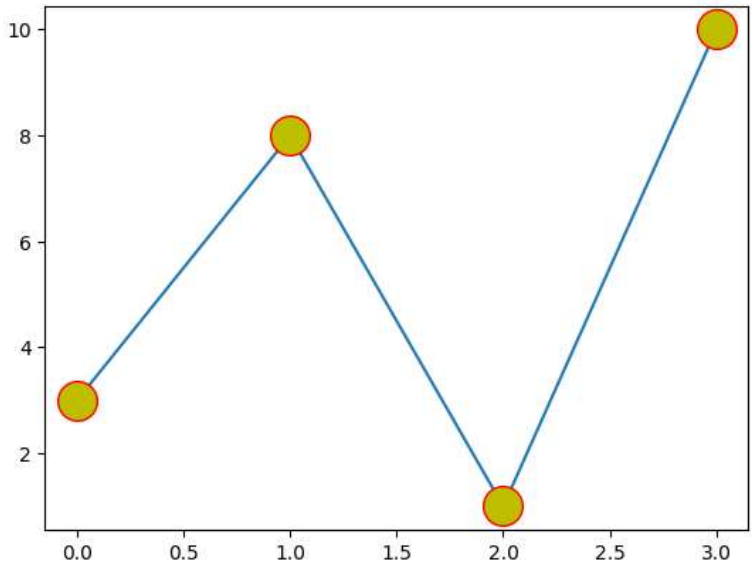
Default X points

```
#Display markers
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o')
plt.show()
```



Customize markers:

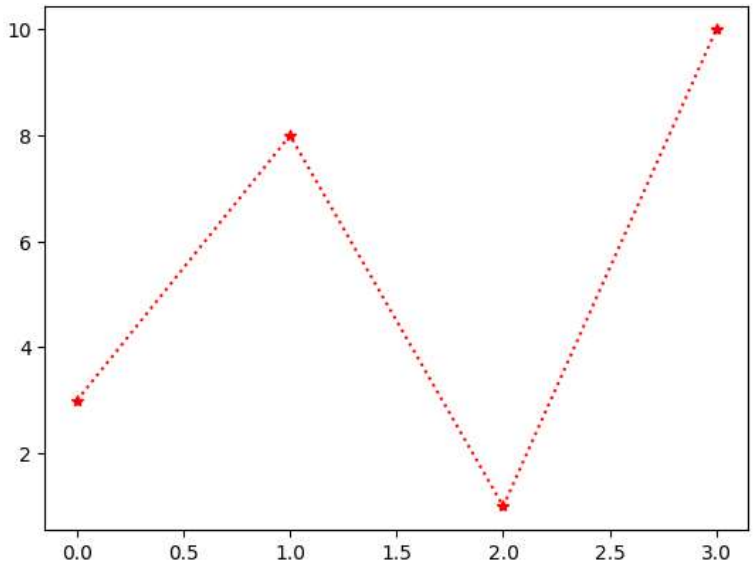
```
#Marker type: marker keyword
plt.plot(ypoints, marker = '*')   #for * as marker
plt.plot(ypoints, marker = 'o')   #for circle (regular) marker
#and similarly, '.' , 'x' etc.

#Marker size: ms keyword

#Marker edge colour: mec keyword

#Marker face colour: mfc keyword

ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r', mfc = 'y' )
plt.show()
```



Format string:

- This parameter is also called `fmt`, and is written with this syntax: `marker|line|color`

```
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, '*:r')
'''
'*:r' fmt signifies
* marker
: dotted line
r red color
'''
plt.show()
```

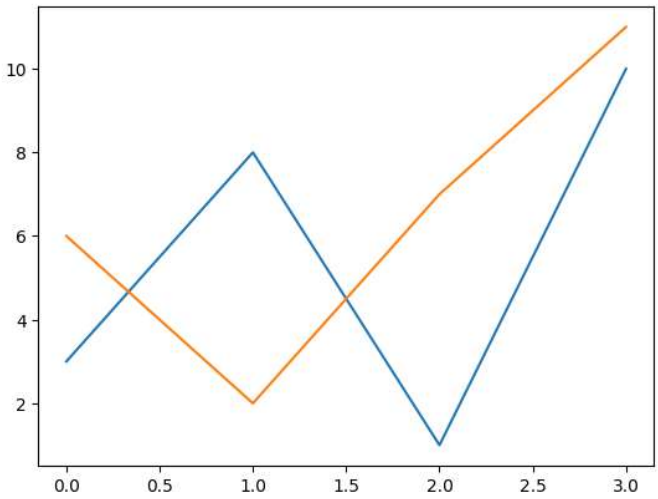



```
#Types of formats
markers = '*','o','x','.' etc
lines =
'-' solid line
':' dotted line
'--' dashed line
'-.' dashed dotted line
colors =  r,b,g,y,k(black),w(white) etc
```

Plot Multiple lines:

- Add as many line as we want, be simply adding multiple `plot()` functions

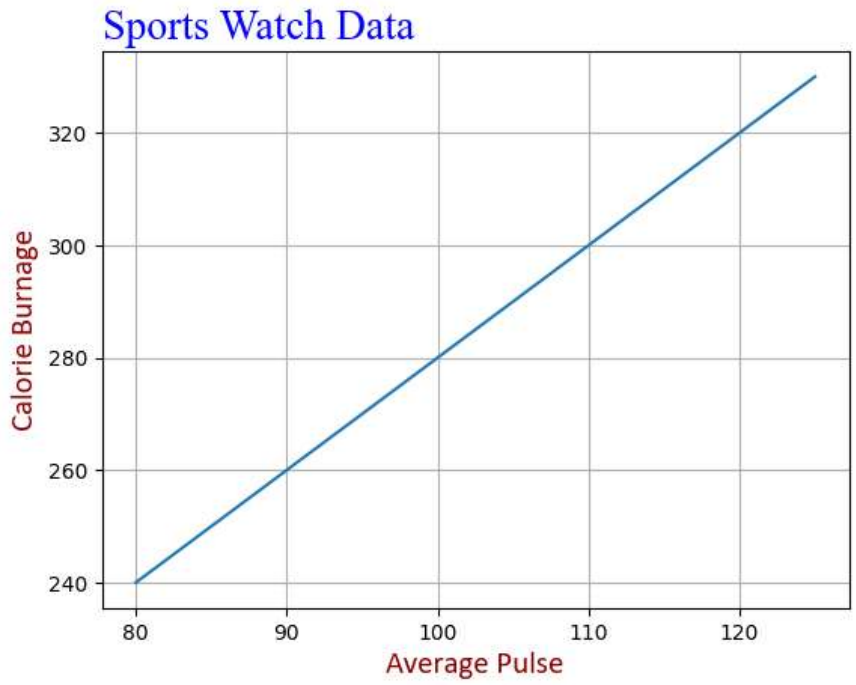
```
y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])
plt.plot(y1)   #will take default x values
plt.plot(y2)
plt.show()
```



Labels:

- you can use the `xlabel()` and `ylabel()` functions to set a label for the x- and y-axis
- the `title()` function to set a title for the plot,
- use `loc` and `fontdict` to set the alignment and font properties respectively
- use `grid()` to display gridlines

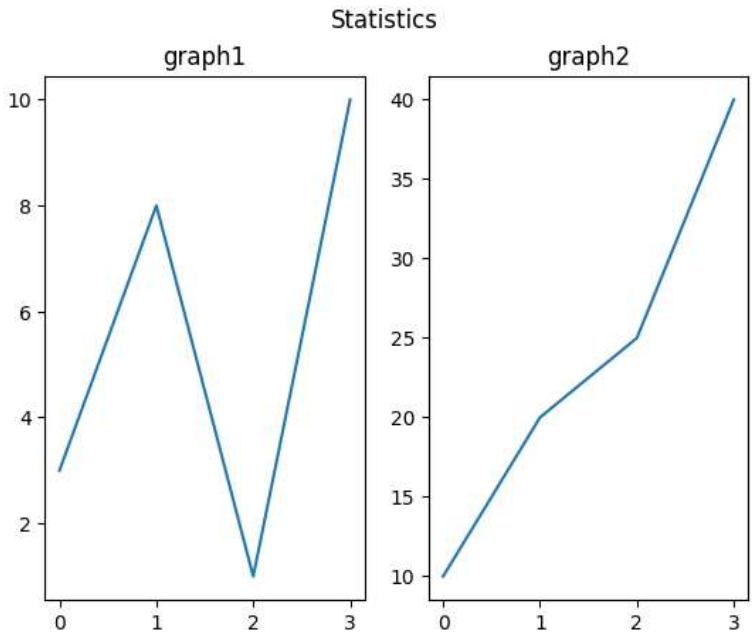
```
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'Times new roman','color':'blue','size':20}
font2 = {'family':'calibri','color':'darkred','size':15}

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

plt.plot(x, y)
plt.grid()   #we can give style to grid in same way as line styling
plt.show()
```



Display multiple plots:

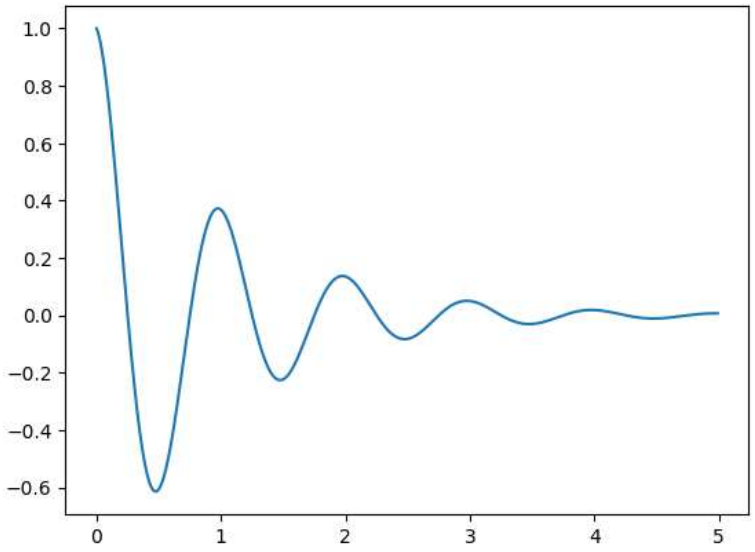
- We use the `subplot()` function
- It takes 3 arguements, total rows, total columns and index of current plot

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

plt.subplot(1, 2, 1)    '1 row, 2 column, this is 1st subplot'
plt.plot(x,y)
plt.title("graph1")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 25, 40])

plt.subplot(1, 2, 2)    '1 row, 2 column, this is 2nd subplot'
plt.plot(x,y)
plt.title("graph2")

plt.suptitle("Statistics")   #super title
plt.show()
```



Plot a function:

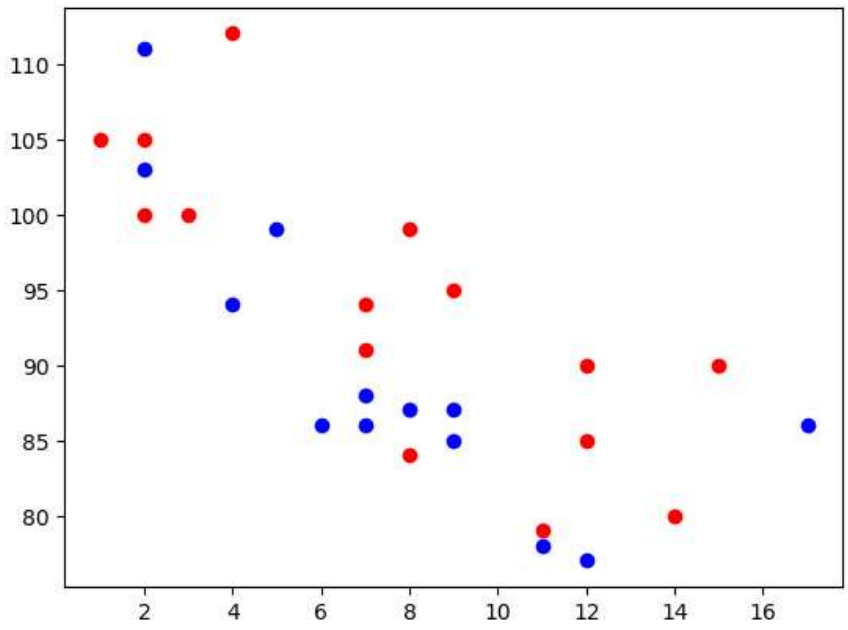
```
#function plot
def para(x):
    return np.exp(-x)*np.cos(2*(np.pi)*x)
x = np.arange(0,5,0.01)
plt.plot(x,para(x))
plt.show()
```



Scatter plots:

- Use `plt.scatter()`

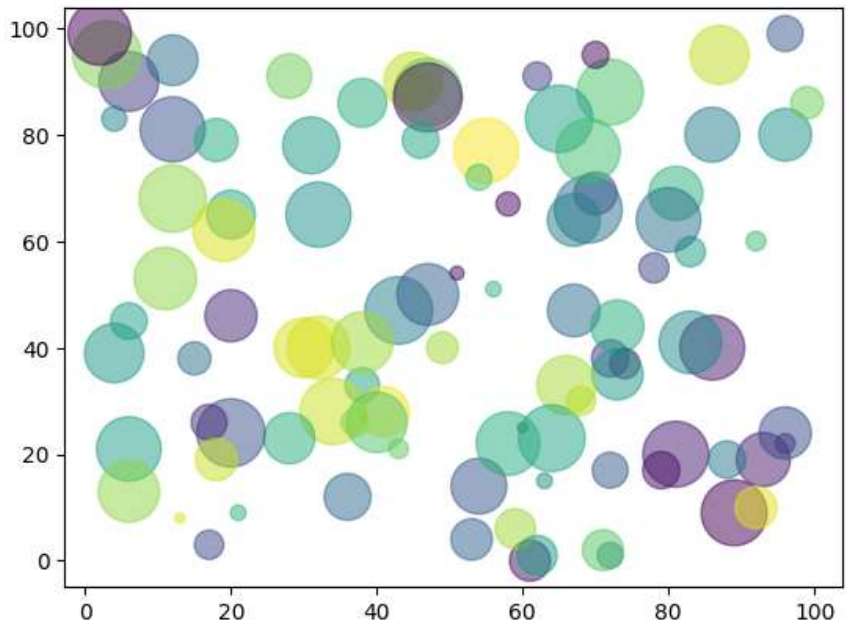
```
#scatter1
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'blue')

#scatter2
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 = 'red')

plt.show()
```



- We can use all the styling mentioned for `plot()` in `scatter()` also
- Some more parameters:

```
c= adjust colour of individual dots (pass a list of colors)
alpha= adjust the transparency of the dots
size= adjust size of dots
```

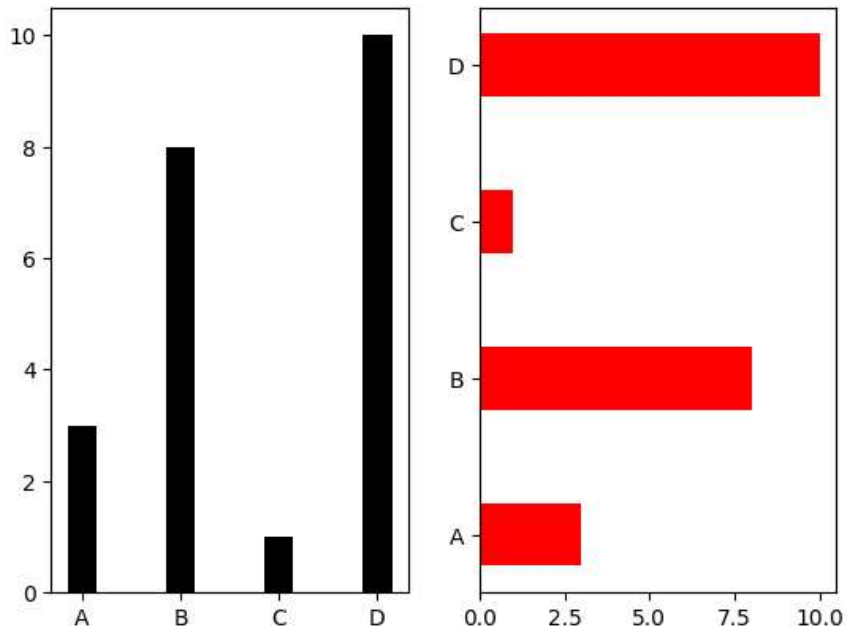
```
#Example using some attributes
x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5)
plt.show()
```



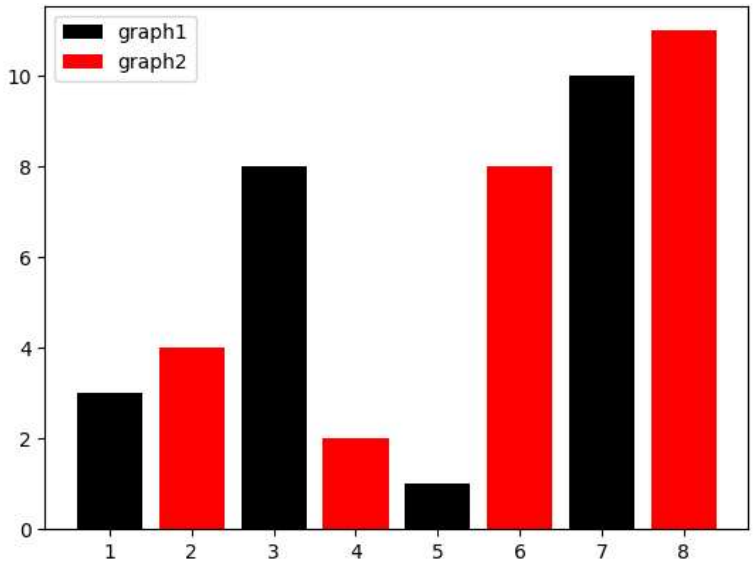
Bar diagrams:

- Use the `bar()` function for vertical bar graph
- Use `barh()` for horizontal bar graph
- Useful attributes: `color=, width=, height=`

```
#bar1
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.subplot(1,2,1)
plt.bar(x, y, width = 0.3, color = 'black')

#bar2
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.subplot(1,2,2)
plt.barh(x, y, height = 0.4, color = 'red')

plt.show()
```



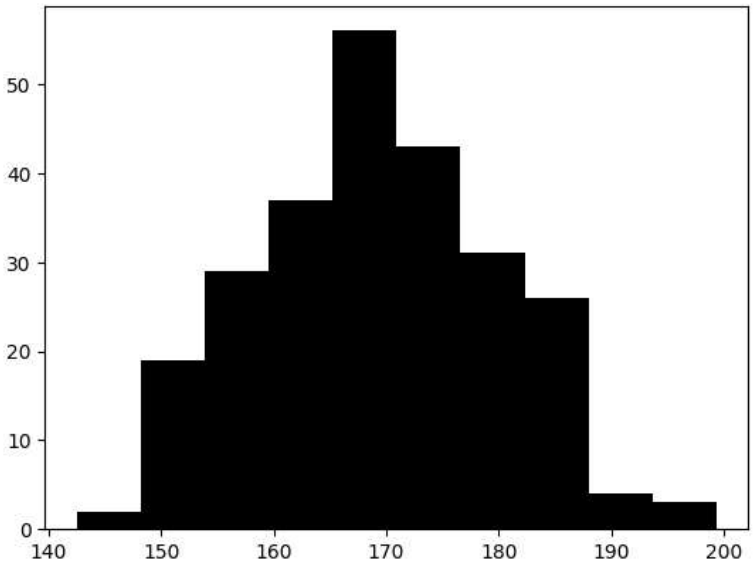

```
#multiple bars in same graph
#bar1
x = np.array([1,3,5,7])
y = np.array([3, 8, 1, 10])
plt.bar(x, y, color = 'black', label="graph1")

#bar2
w = np.array([2,4,6,8])
z = np.array([4, 2, 8, 11])
plt.bar(w, z, color = 'red', label="graph2")

plt.legend()
plt.show()
```



Historgram:

- Use `hist()` function
- A histogram is a graph showing *frequency* distributions
- We pass an array to `hist()` function

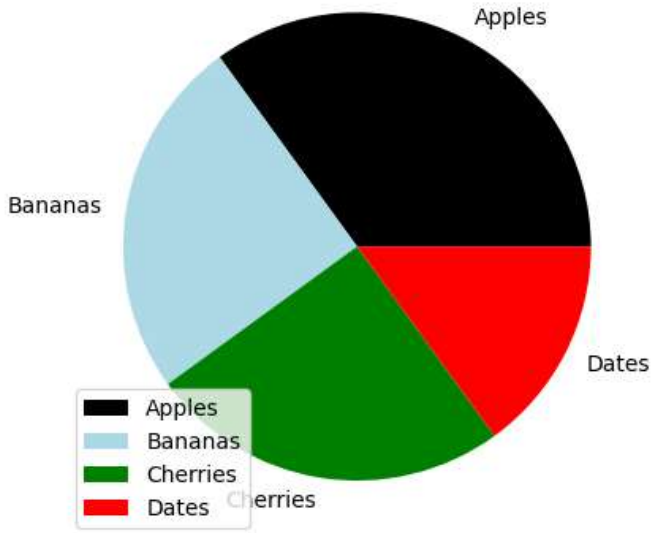
```
x = np.random.normal(170, 10, 250)
plt.hist(x, color='black')
plt.show()
```



Pie chart:

- Use `pie()` function
- Important attributes: `labels=[list], startangle=, colors=[list]`
- Important functions: `plt.legend()`

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

plt.pie(y, labels = mylabels, colors = mycolors)   #default startangle = 0 (i.e x-axis)
plt.legend()
plt.show()
```


Apples
Bananas
Dates
Cherries
Apples
Bananas
Cherries
Dates