

Matplotlib Tutorial

Matplotlib?

Matplotlib is a low level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely. Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Import Matplotlib

Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias: Now the Pyplot package can be referred to as plt.

```
In [2]: import matplotlib.pyplot as plt  
       import numpy as np
```

Checking Matplotlib Version

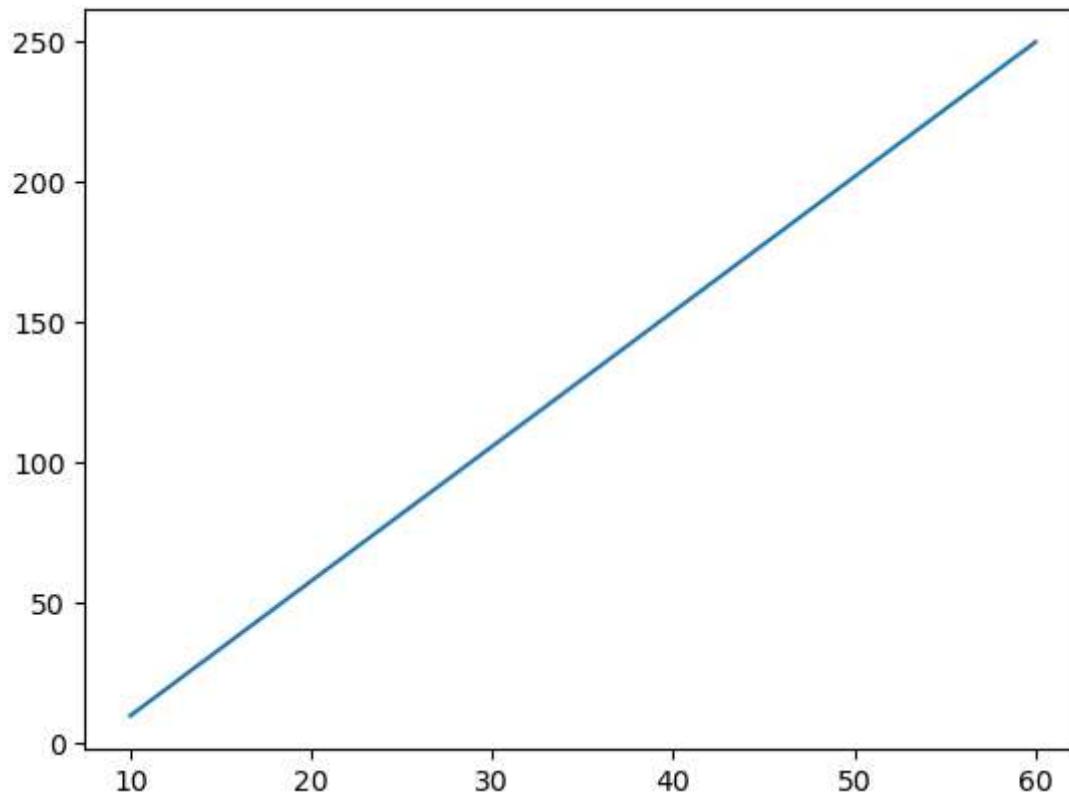
The version string is stored under **version** attribute import matplotlib print(matplotlib.version)

Draw a line in a diagram from position (0,0) to position (6,250):

The plot() function is used to draw points (markers) in a diagram.

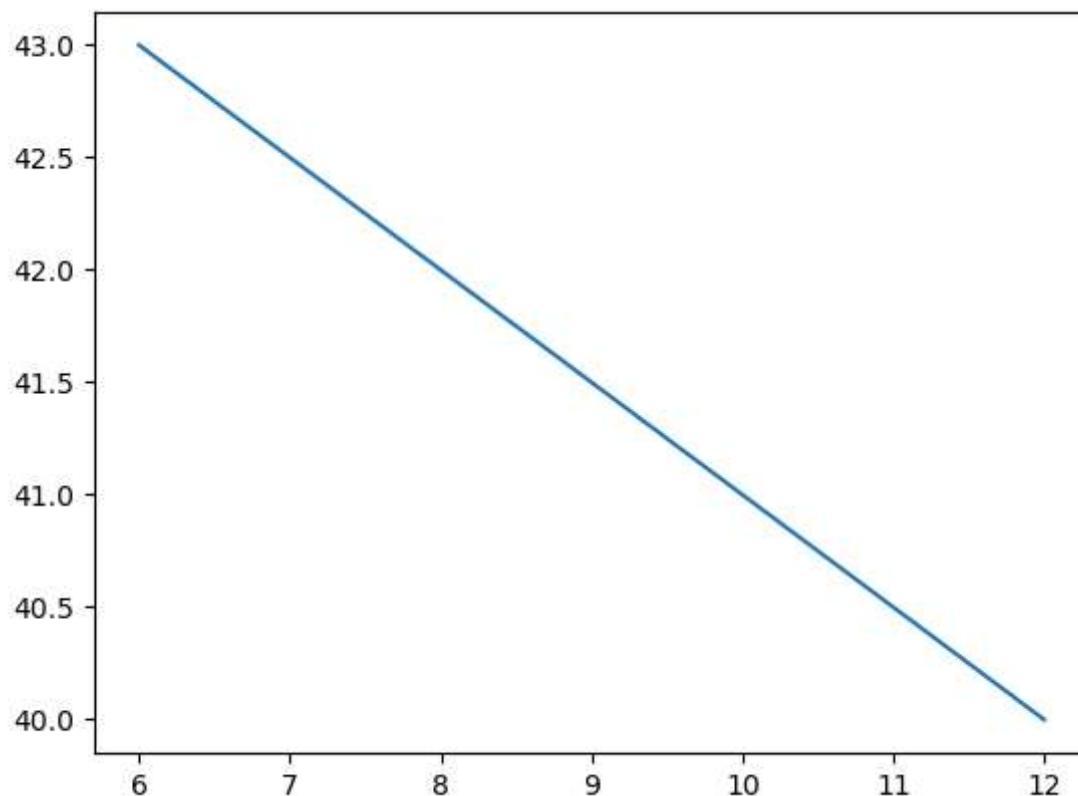
By default, the plot() function draws a line from point to point.

```
In [8]: #Draw a Line chart  
x = np.array([10,60])  
y = np.array([10,250])  
plt.plot(x,y)  
plt.show()
```



```
In [ ]: #Draw a Line in a diagram from position (6,43) to position (12,40):
```

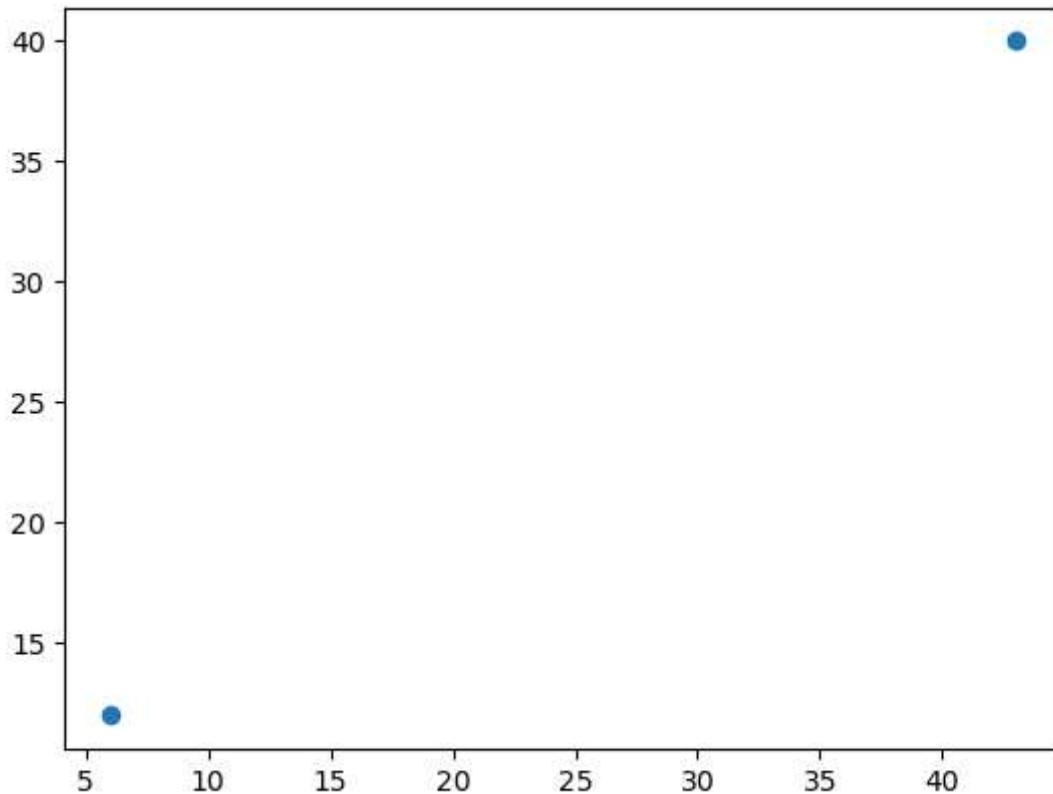
```
In [13]: x =np.array([6,12])
y =np.array([43,40])
plt.plot(x,y)
plt.show()
```



Plotting Without Line (use - markers)

To plot only the markers, you can use shortcut string notation parameter 'o', which means 'rings'.

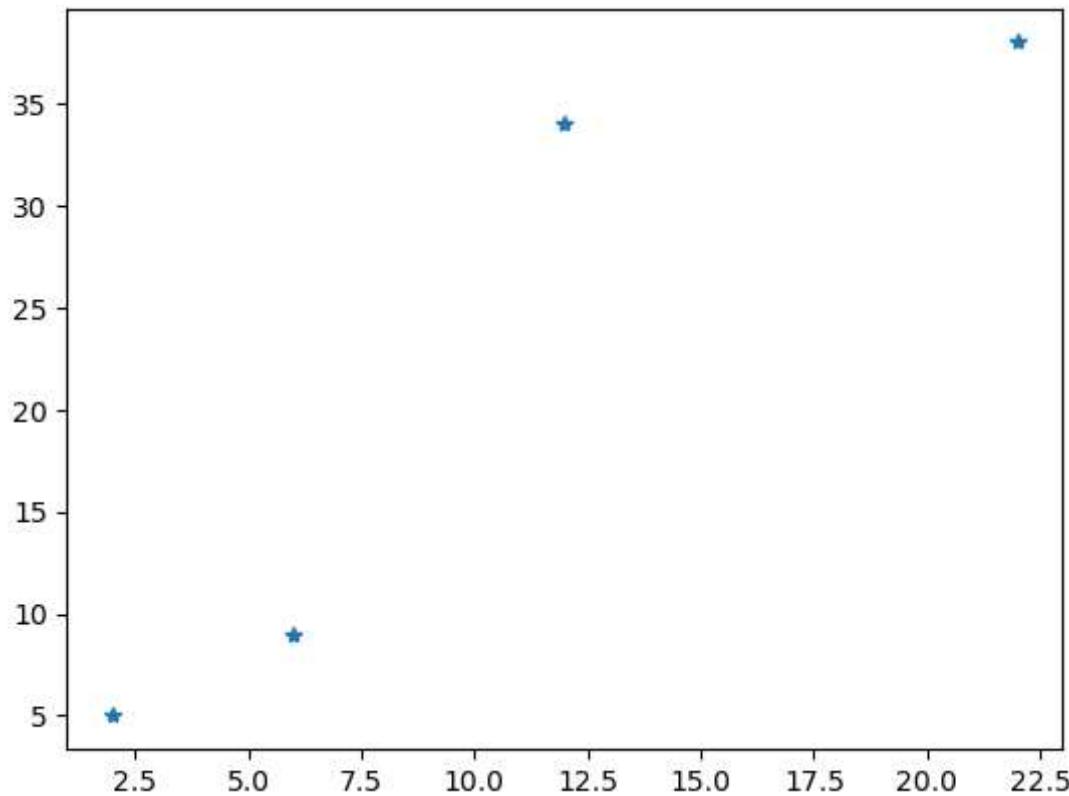
```
In [12]: x =np.array([6,43])  
y =np.array([12,40])  
plt.plot(x,y,'o')  
plt.show()
```



Multiple Points

You can plot as many points as you like, just make sure you have the same number of points in both axis.

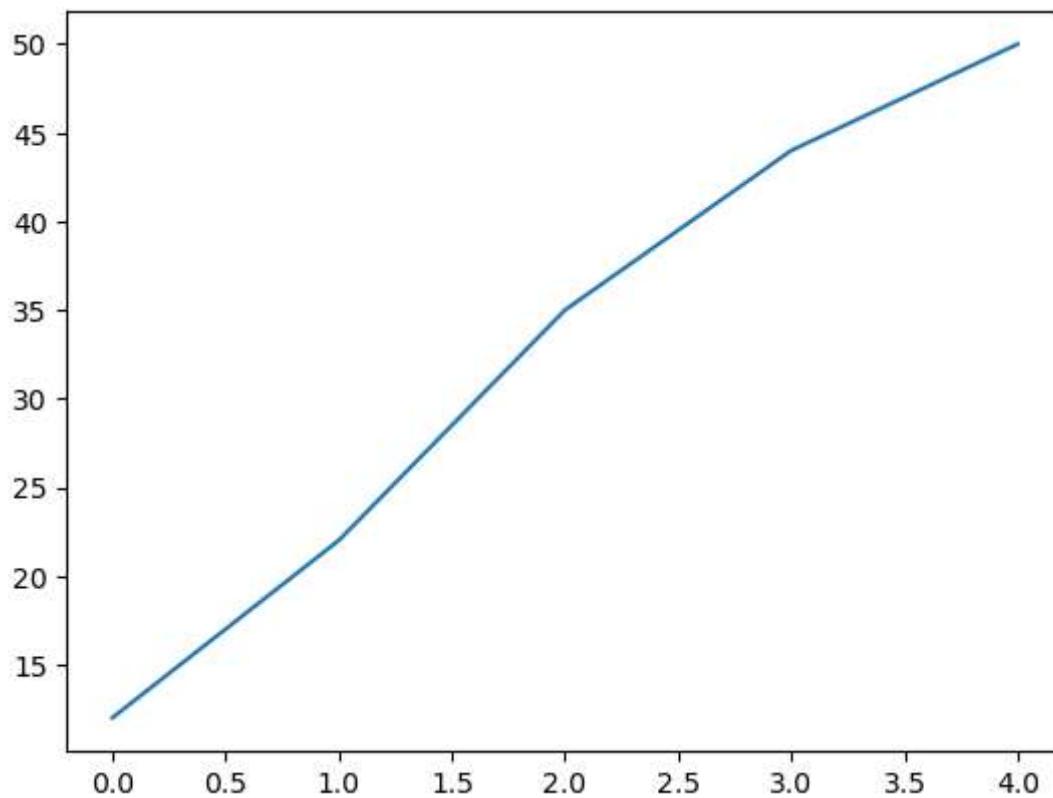
```
In [16]: # Draw a Line in a diagram from position (2,5) to (6,9) then to (12,34) and finally to  
x = np.array([2,6,12,22])  
y = np.array([5,9,34,38])  
#plt.plot(x,y) # Line chart  
plt.plot(x,y,'*') # with marker  
plt.show()
```



Default X-Points

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

```
In [20]: y = np.array([12,22,35,44,50])
plt.plot(y)
plt.show()
```



Markers

You can use the keyword argument marker to emphasize each point with a specified marker: arguments:

marker - to visualize each point with specified marker
ms - use the keyword argument markersize or the shorter version, ms to set the size of the markers.
mec - use the keyword argument markeredgecolor or the shorter version, mec to set the color of the edge of the markers
mfc - use the keyword argument markerfacecolor or the shorter version, mfc to set the color inside the edge of the markers

Marker Description

- 'o' Circle
- '*' Star '.' Point
- ',' Pixel
- 'x' X
- 'X' X (filled)
- '+' Plus 'P' Plus (filled)
- 's' Square
- 'D' Diamond
- 'd' Diamond (thin)
- 'p' Pentagon 'H' Hexagon
- 'h' Hexagon

```
'v' Triangle Down  
'^' Triangle Up  
'<' Triangle Left  
'>' Triangle Right  
'1' Tri Down '2' Tri Up  
'3' Tri Left '4' Tri Right  
'|' Vline  

```

Hexadecimal Colors

You can also use Hexadecimal color values:

Hexadecimal color values are supported in all browsers.

A hexadecimal color is specified with:

rrggbb

Where rr (red), gg (green) and bb (blue) are hexadecimal integers between 00 and ff, specifying the intensity of the color.

For example, #ff0000 is displayed as red, because red is set to its highest value (ff), and the other two (green and blue) are set to 00.

Another example, #00ff00 is displayed as green, because green is set to its highest value (ff), and the other two (red and blue) are set to 00.

To display black, set all color parameters to 00, like this: #000000.

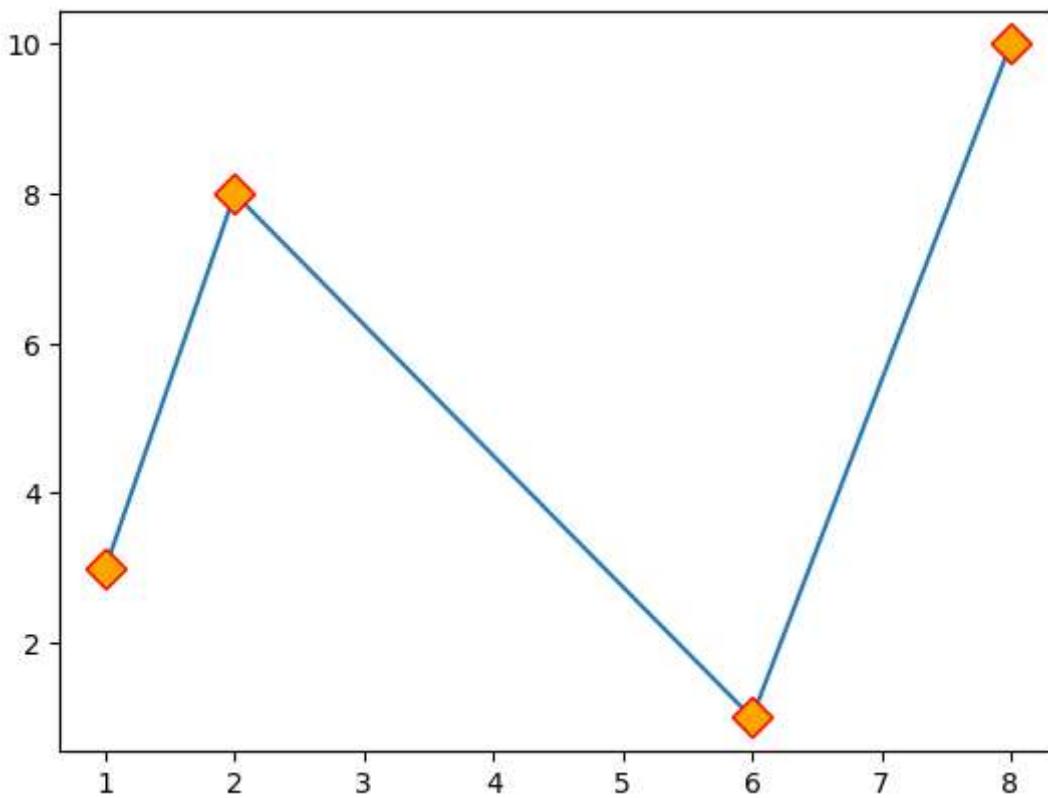
To display white, set all color parameters to ff, like this: #ffffff.

Color Names Supported by All Browsers :

All modern browsers support 140 color names.

```
In [34]: #Mark each point with a circle:
```

```
x=np.array([1,2,6,8])  
y=np.array([3,8,1,10])  
plt.plot(x,y,marker='o',ms=20,mec='red',mfc='green')  
plt.plot(x,y,marker='D',ms=10,mec ='red',mfc='orange')  
plt.show()
```



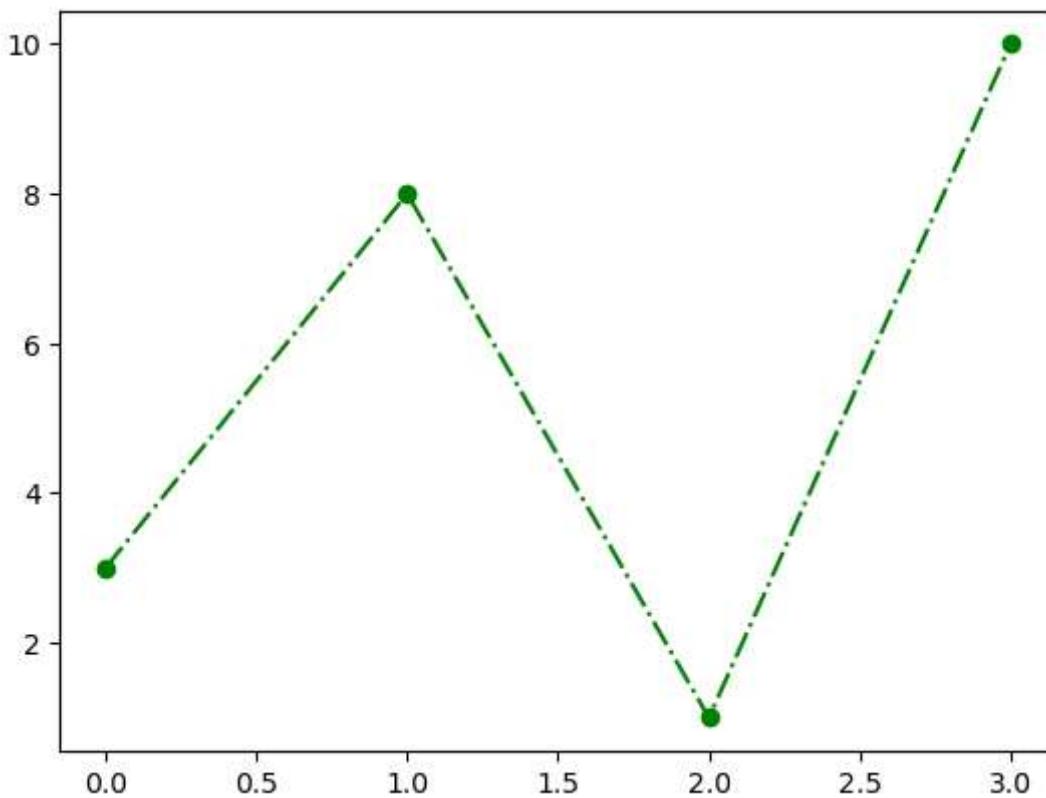
Format Strings - fmt

You can also use the shortcut string notation parameter to specify the marker. This parameter is also called fmt.

shortcut string notation parameter (fmt) : Syntax : marker|line|color

eg: 'o:r' o - marker : - dotted Line r - red color

```
In [41]: # Using fmt
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, '*:b')
plt.plot(ypoints, 'o-.g')
plt.show()
```



Line Reference

The line value can be one of the following:

Line Syntax

'-' Solid line

'.' Dotted line

'--' Dashed line

'-.' Dashed/dotted line If you leave out the line value in the fmt parameter, no line will be plotted.

Color Reference

The short color value can be one of the following:

Color Syntax

'r' Red 'g' Green

'b' Blue

'c' Cyan

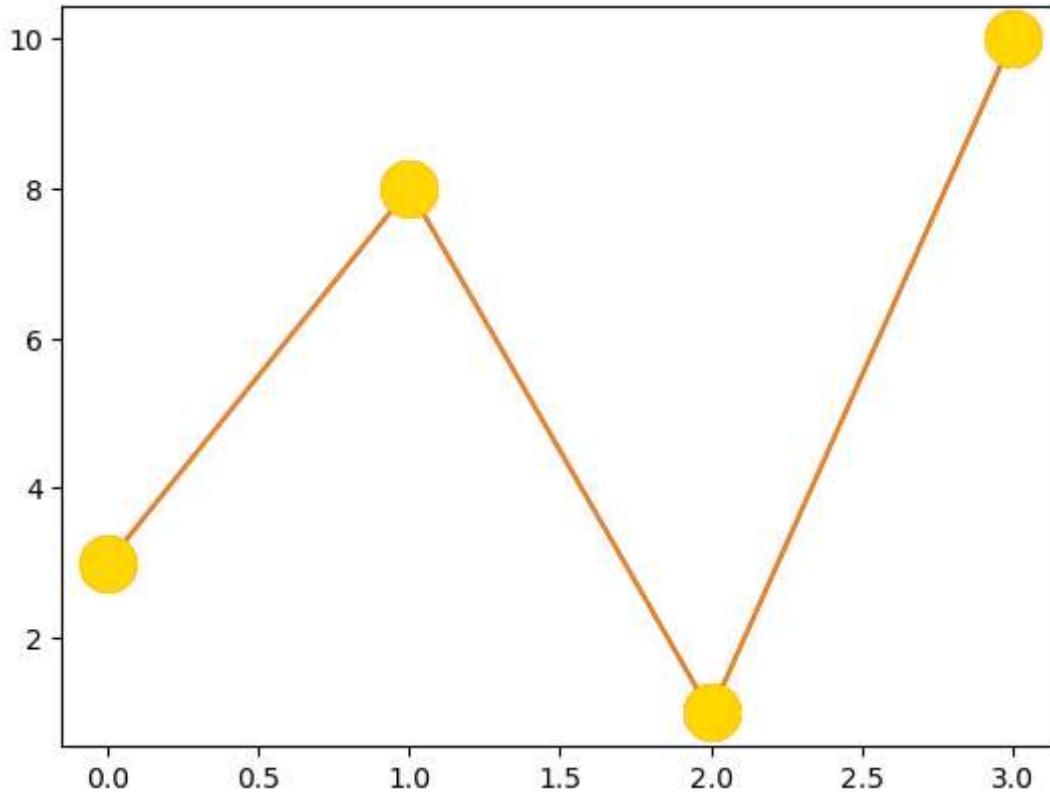
'm' Magenta 'y' Yellow

'k' Black

'w' White

```
In [7]: #Hexadecimal color names(all browsers support 140 colour names):  
y=np.array([3,8,1,10])
```

```
#plt.plot(y, marker = 'o', ms = 20, mec = 'DeepPink', mfc = 'DeepPink')
plt.plot(y, marker = 'o', ms = 20, mec = 'Gold', mfc = 'gold')
plt.show()
```



Matplotlib Line

Linestyle You can use the keyword argument `linestyle`, or shorter `ls`, to change the style of the plotted line: Syntax: `linestyle` or `ls`

You can choose any of these styles:

- 'solid' '-' (default)
- 'dotted' ':'
- 'dashed' '--'
- 'dashdot' '-.'
- 'None' ''

Line Color (color or c)

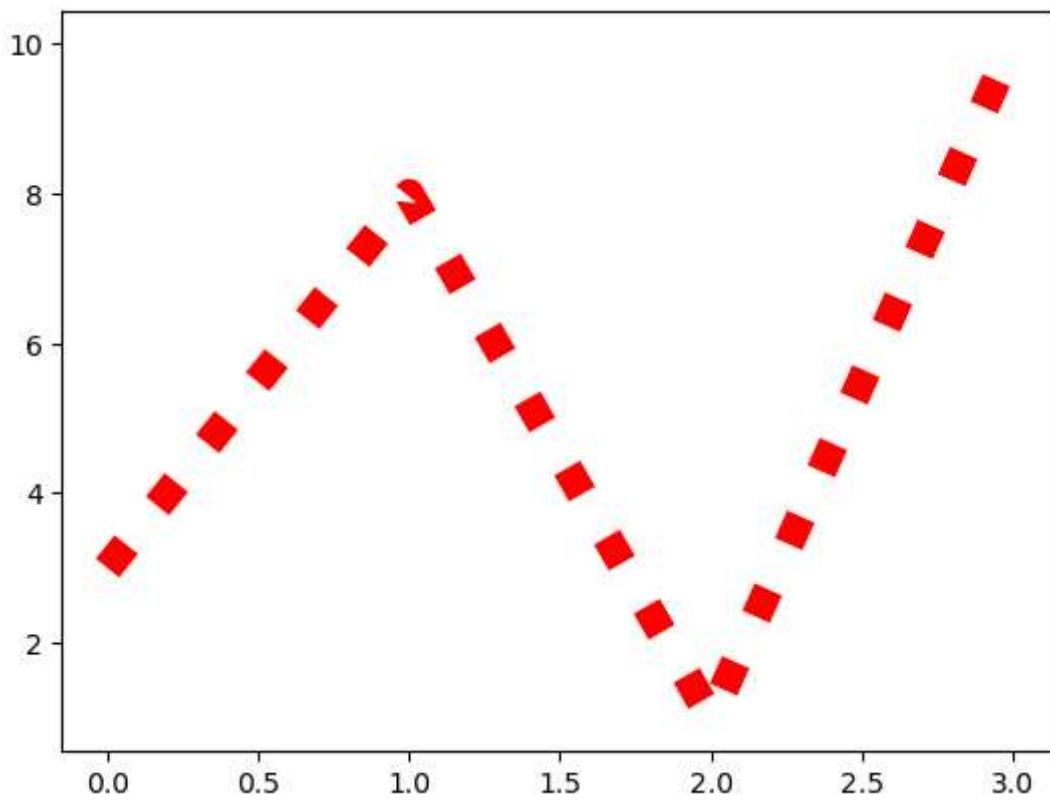
You can use the keyword argument `color` or the shorter `c` to set the color of the line:

Line Width (linewidth or lw)

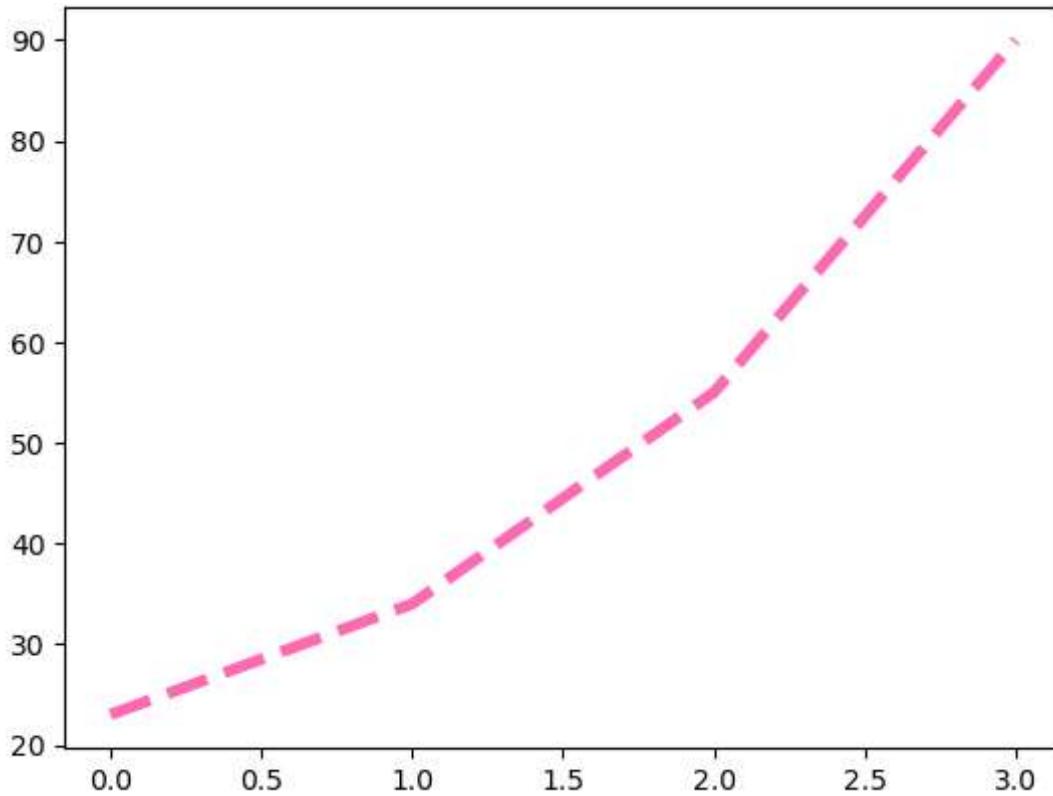
You can use the keyword argument `linewidth` or the shorter `lw` to change the width of the line. The value is a floating number, in points:

```
In [13]: # Linestyle (ls), linewidth(lw) ,color(c)

y=np.array([3,8,1,10])
plt.plot(y,linestyle=':',c="red", lw='10.8')
plt.show()
```



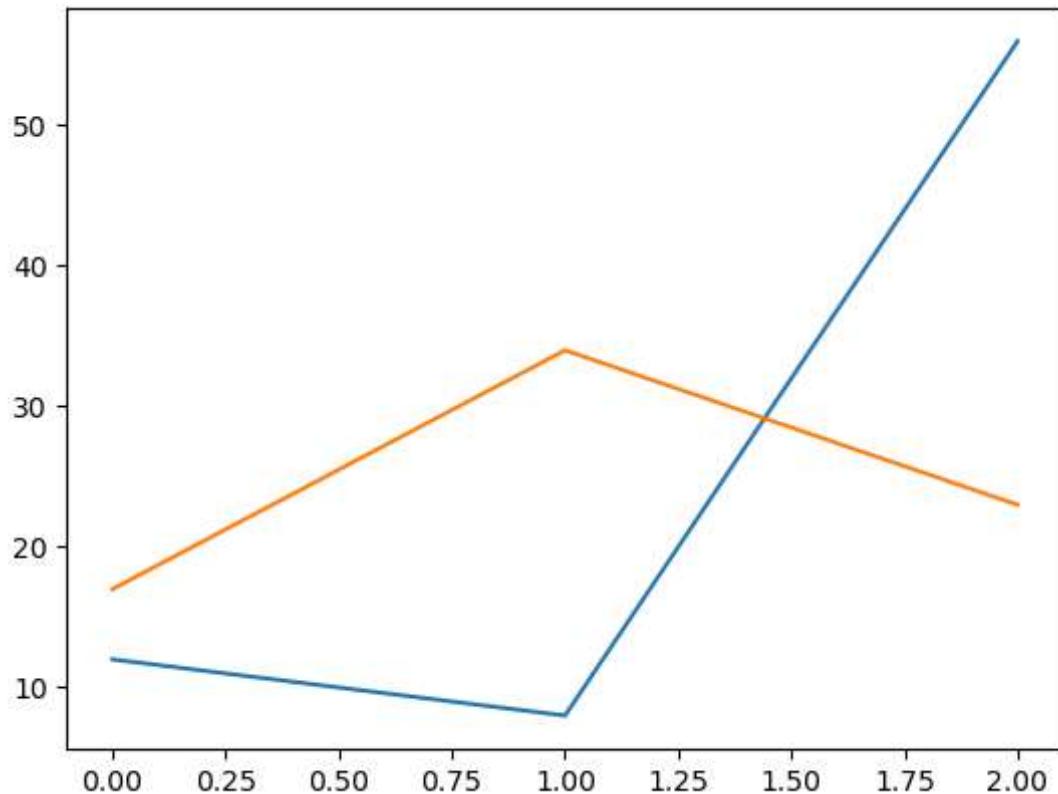
```
In [16]: y = np.array([23,34,55,90])
plt.plot(y,ls='--',lw='4.1',c="HotPink")
plt.show()
```



Multiple Lines

You can plot as many lines as you like by simply adding more plt.plot() functions:

```
In [21]: #Multiple Lines  
y1 = np.array([12,8,56])  
y2 = np.array([17,34,23])  
plt.plot(y1)  
plt.plot(y2)  
plt.show()
```

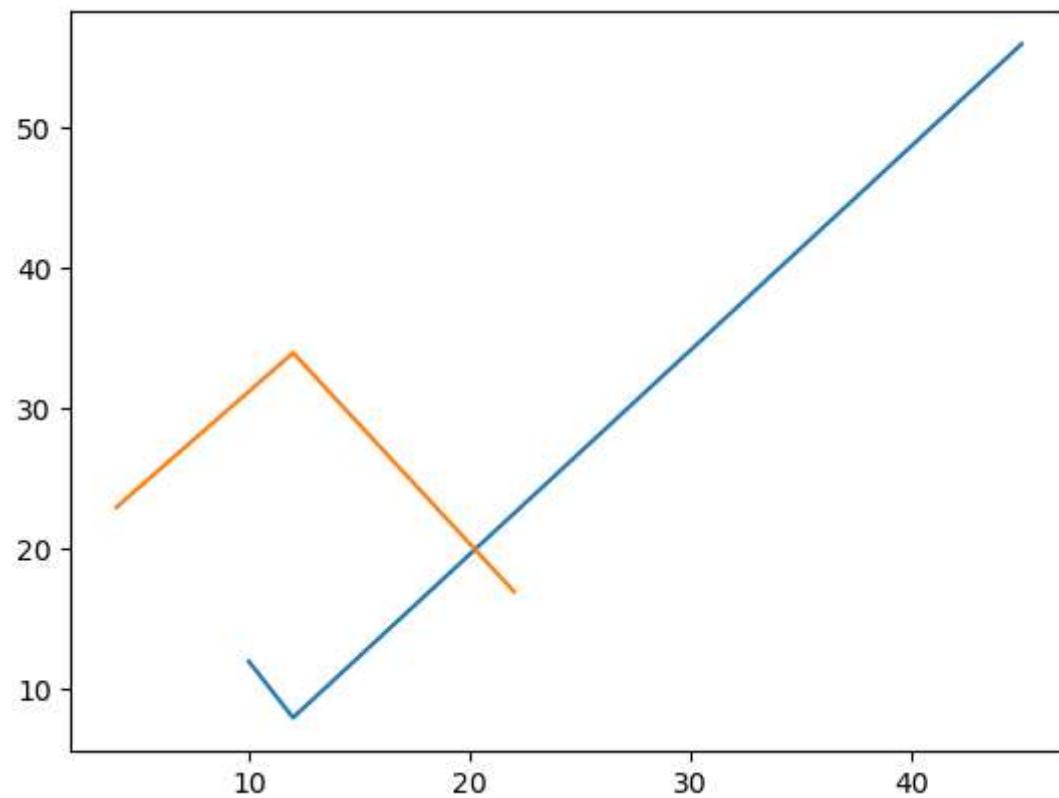


You can also plot many lines by adding the points for the x- and y-axis for each line in the same plt.plot() function.

(In the examples above we only specified the points on the y-axis, meaning that the points on the x-axis got the the default values (0, 1, 2, 3).)

The x- and y- values come in pairs:

```
In [22]: x1 = np.array([10,12,45])  
y1 = np.array([12,8,56])  
x2 = np.array([22,12,4])  
y2 = np.array([17,34,23])  
plt.plot(x1,y1,x2,y2)  
plt.show()
```



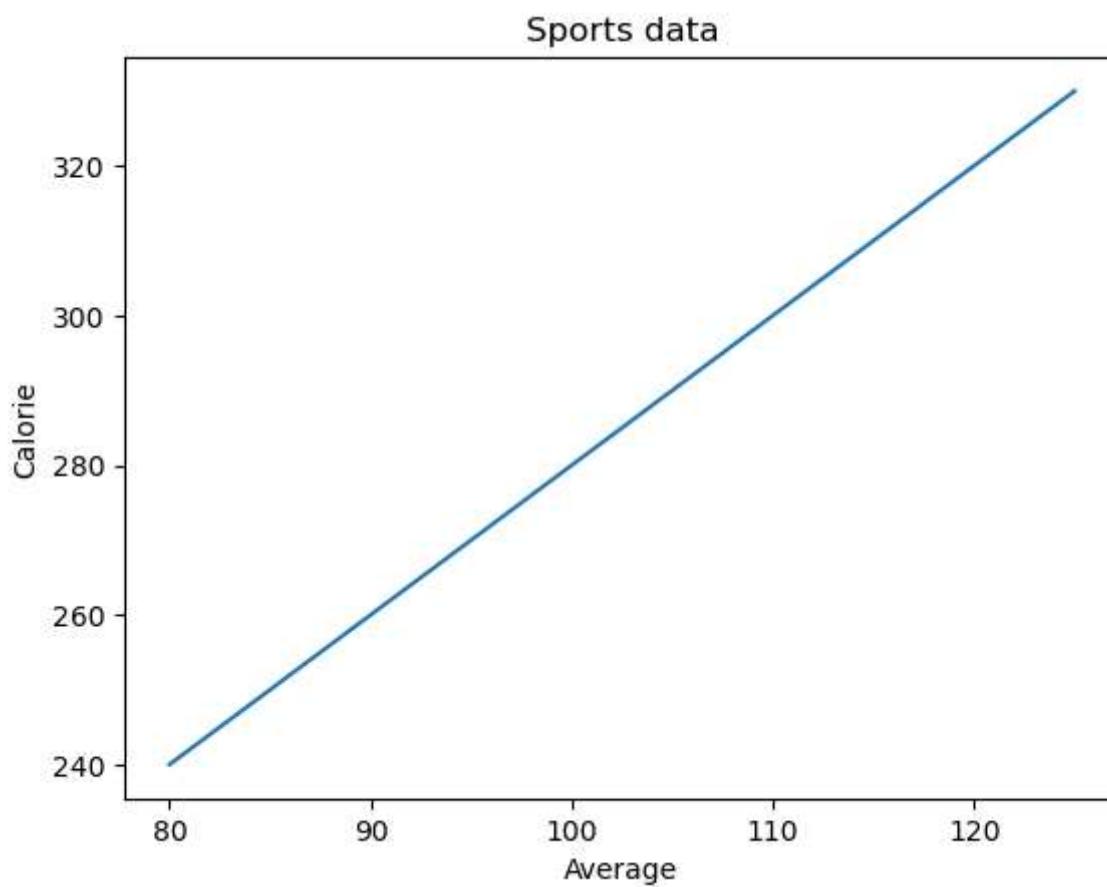
Matplotlib Labels and Title

With Pyplot, you can use the functions : xlabel() - set label for x-axis ylabel() - set label for y-axis title() - set a title for the plot.

loc

You can use the loc parameter in title() to position the title. Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

```
In [23]: #Create Labels for a Plot
# Add Labels to the x- and y-axis:
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.plot(x,y)
plt.xlabel("Average")
plt.ylabel("Calorie")
plt.title("Sports data", loc='center')
plt.show()
```



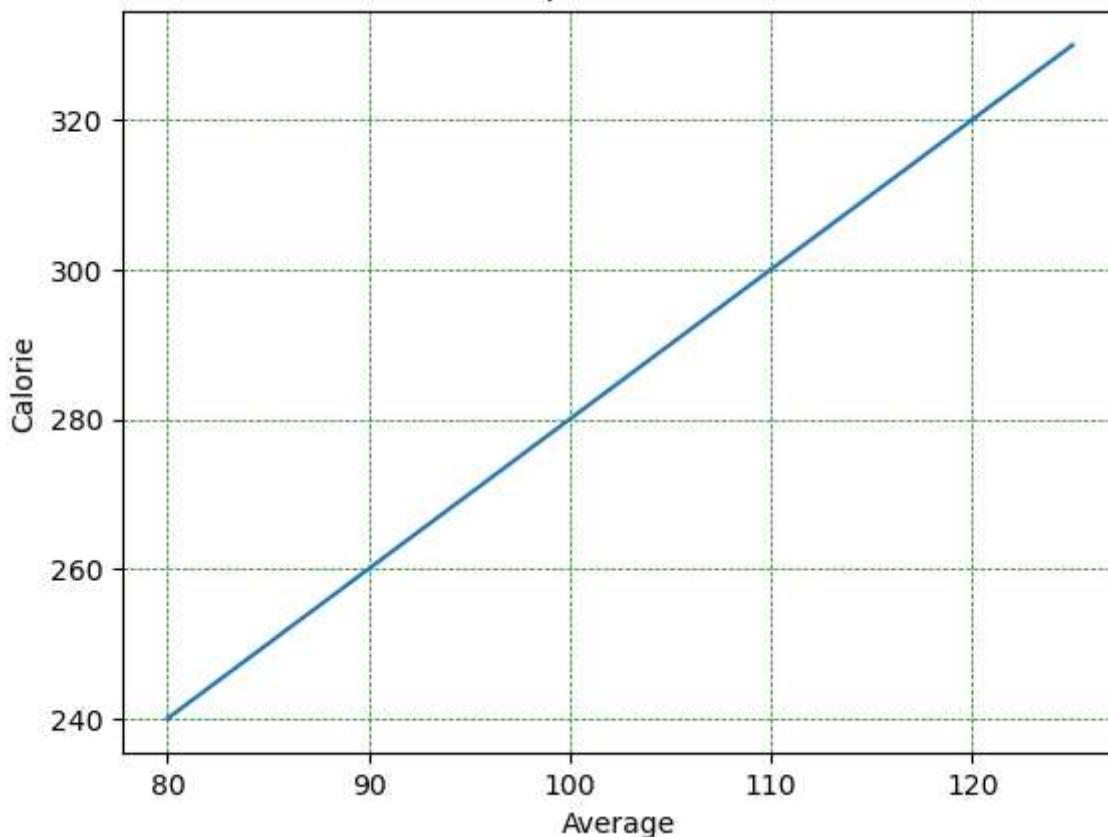
```
In [ ]: #set font properties for title and labels
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}
plt.title("Sports Data", fontdict=font1)
plt.xlabel("Average", fontdict=font2)
plt.ylabel("Calorie", fontdict=font2)
plt.plot(x,y)
plt.show()
```

Add Grid Lines to a Plot

grid() function to add grid lines to the plot. You can also set the line properties of the grid, like this: grid(color = 'color', linestyle = 'linestyle', linewidth = number).

```
In [28]: #Add Grid Lines to a Plot
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.plot(x,y)
plt.xlabel("Average")
plt.ylabel("Calorie")
plt.title("Sports data", loc='center')
plt.grid(c='green', ls='--', lw='0.5')
plt.show()
```

Sports data

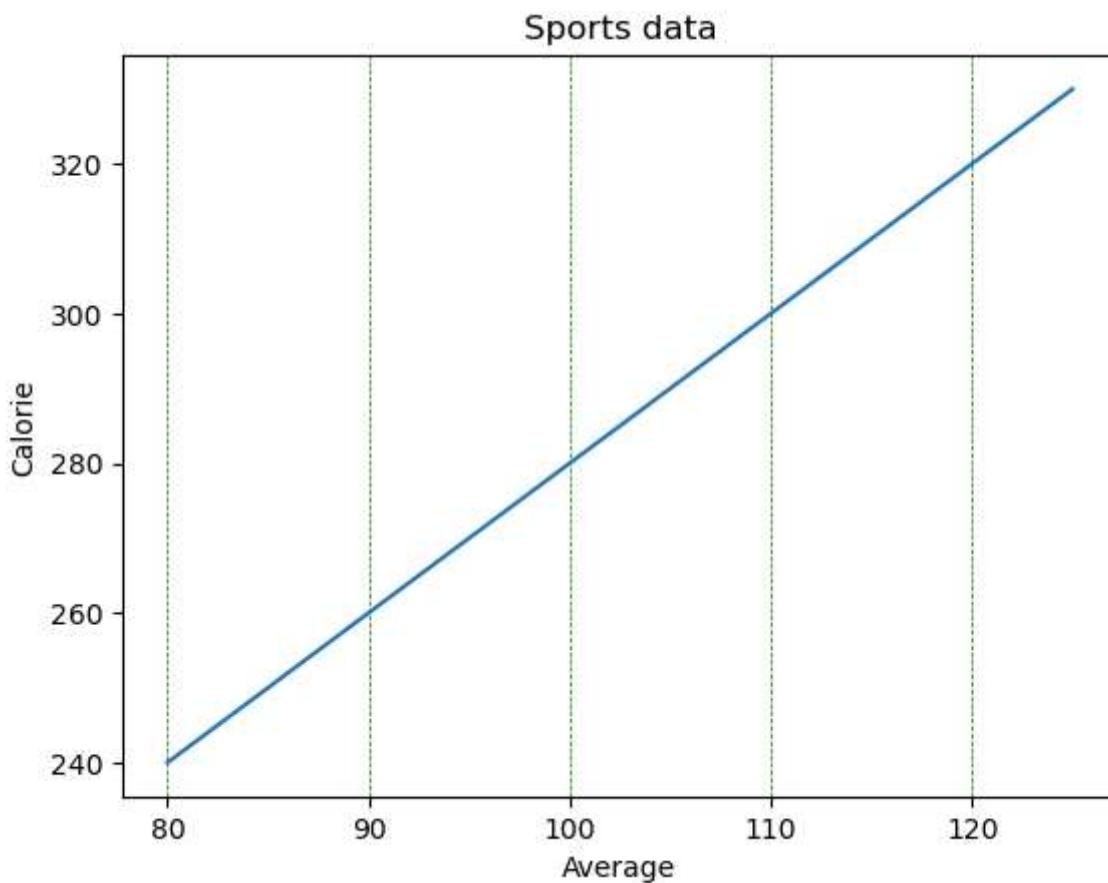


Specify Which Grid Lines to Display

You can use the `axis` parameter in the `grid()` function to specify which grid lines to display.

Legal values are: 'x', 'y', and 'both'. Default value is 'both'. eg: `grid(axis='x')`

```
In [34]: #Display only grid lines for the x-axis:  
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.plot(x,y)  
plt.xlabel("Average")  
plt.ylabel("Calorie")  
plt.title("Sports data", loc='center')  
#plt.grid(c='green', ls='--', lw='0.5', axis='y')  
plt.grid(c='green', ls='--', lw='0.5', axis='x')  
#plt.grid(c='green', ls='--', lw='0.5', axis='both')  
plt.show()
```



Matplotlib Subplot

`subplot()` : you can draw multiple plots in one figure. 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. eg:
`plt.subplot(1, 2, 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.

You can draw as many plots you like on one figure, just describe the number of rows, columns, and the index of the plot.

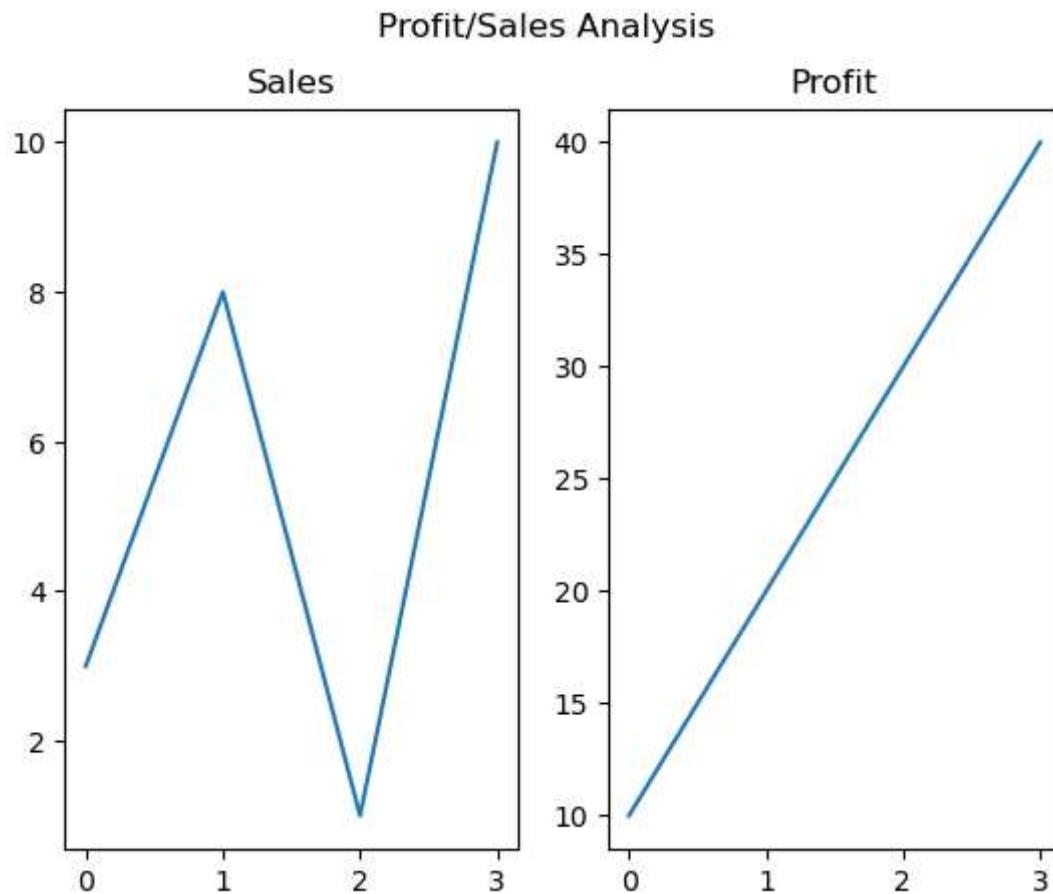
`title()` : add a title to each plot
`suptitle()` : add a title to the entire figure

In [43]: #Display 2 Plots (1 row and 2 column)

```
#plot1
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1,2,1)
plt.title("Sales")

#plot2
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1,2,2)
plt.title("Profit")

plt.suptitle("Profit/Sales Analysis")
plt.plot(x,y)
plt.show()
```



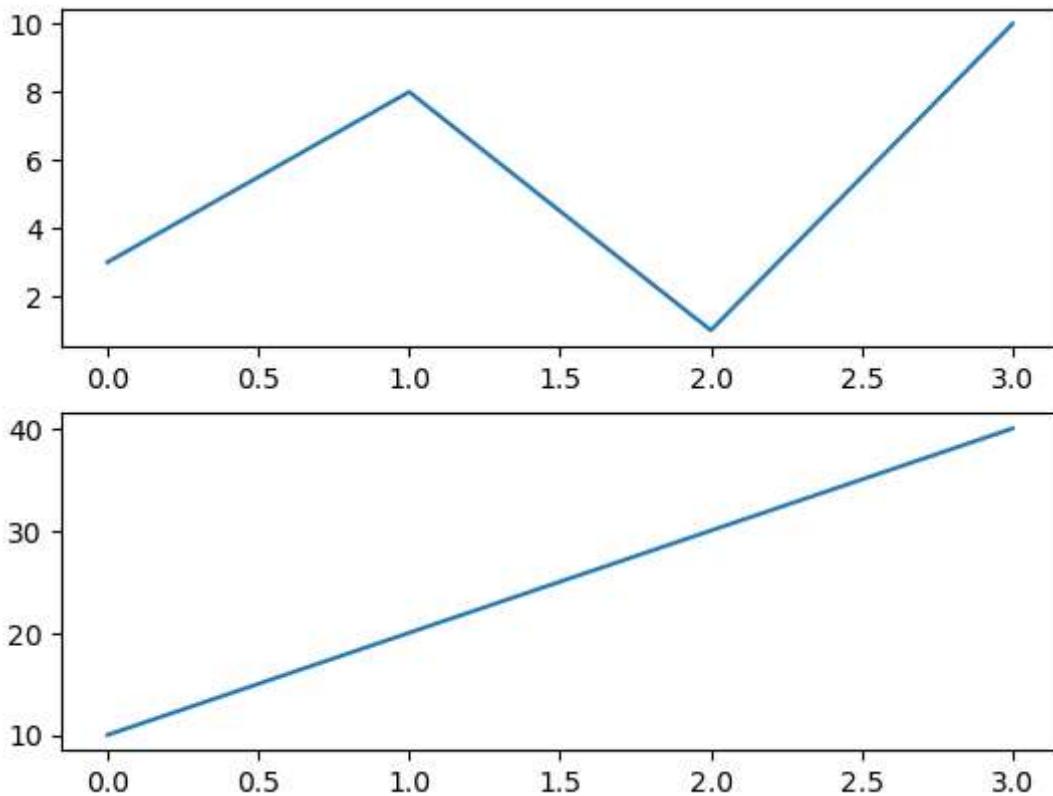
In [37]: #Display 2 Plots (2 row 1 column)

```
#plot1
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2,1,1)

#plot2
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
```

```
y = np.array([10, 20, 30, 40])
plt.subplot(2,1,2)

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

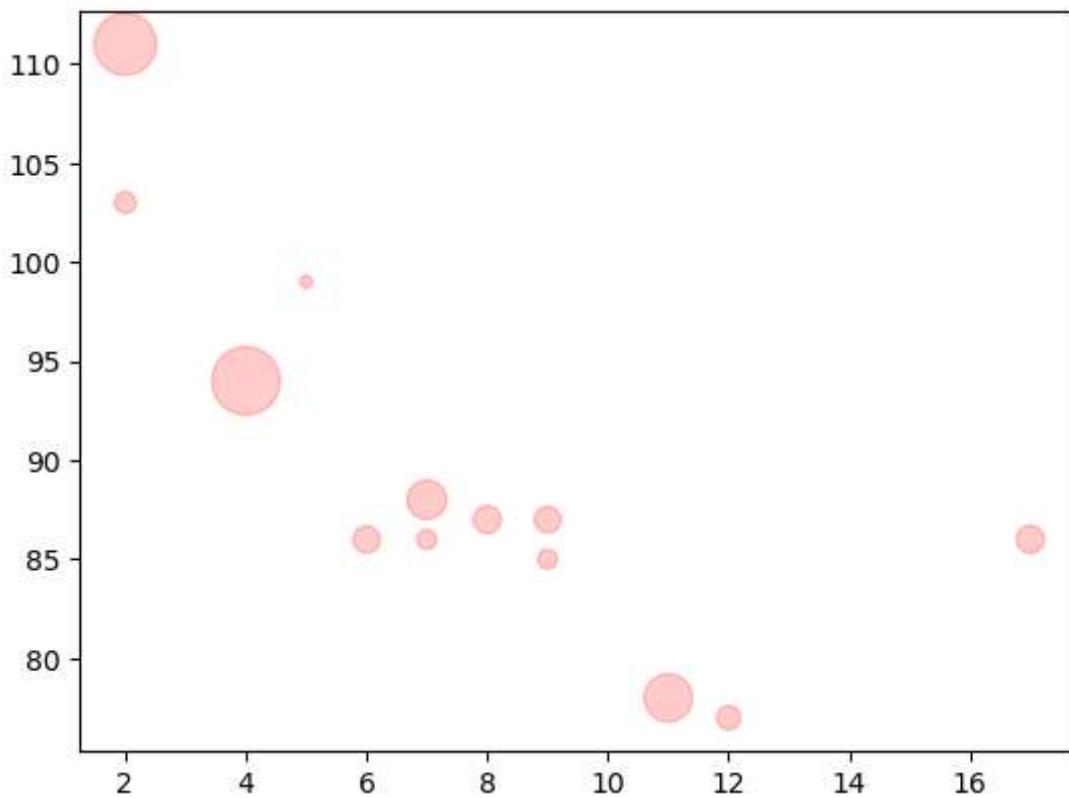


Creating Scatter Plots

With Pyplot, you 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 [44]: #Creating Scatter Plots
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([20,50,100,200,500,100,60,90,600,300,75,50,95])
plt.scatter(x,y,color='red',s=sizes,alpha=0.2) # alpha = set the opacity(transparency)
plt.show()
```



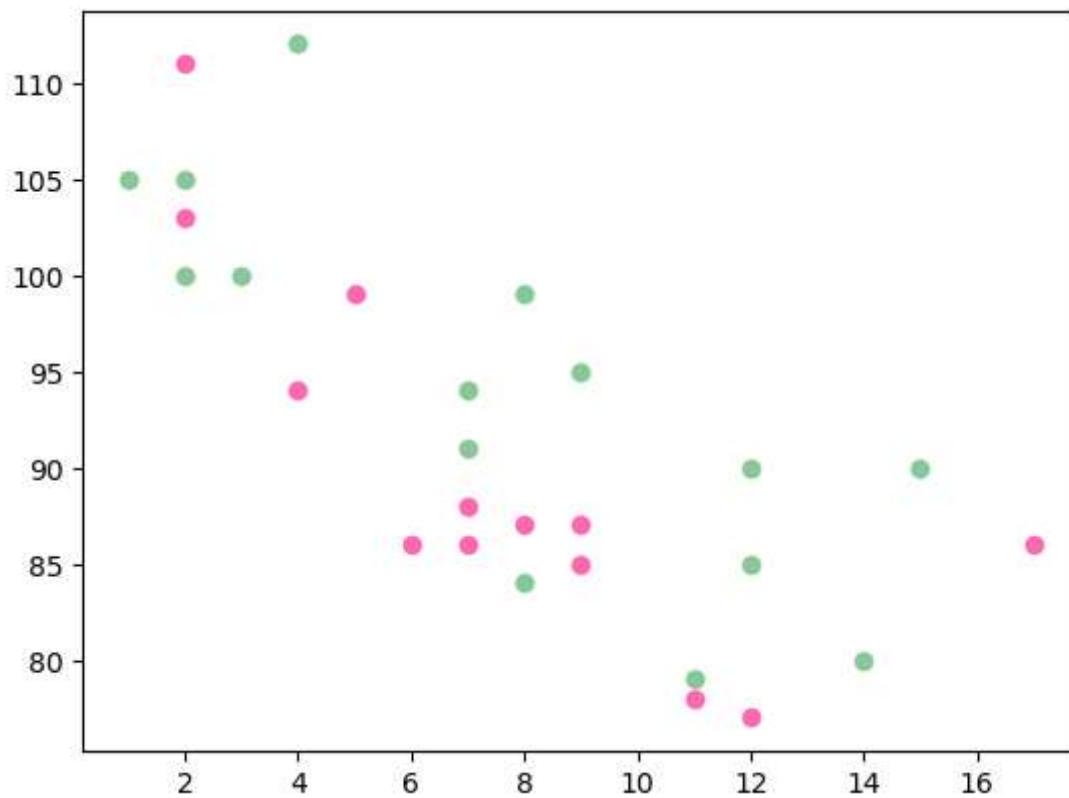
In [45]: *#Creating multiple plot and set color your own color for the marker*

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

x = np.array([5,7,8,2,17,2,9,4,11,12,9,6])
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()
```



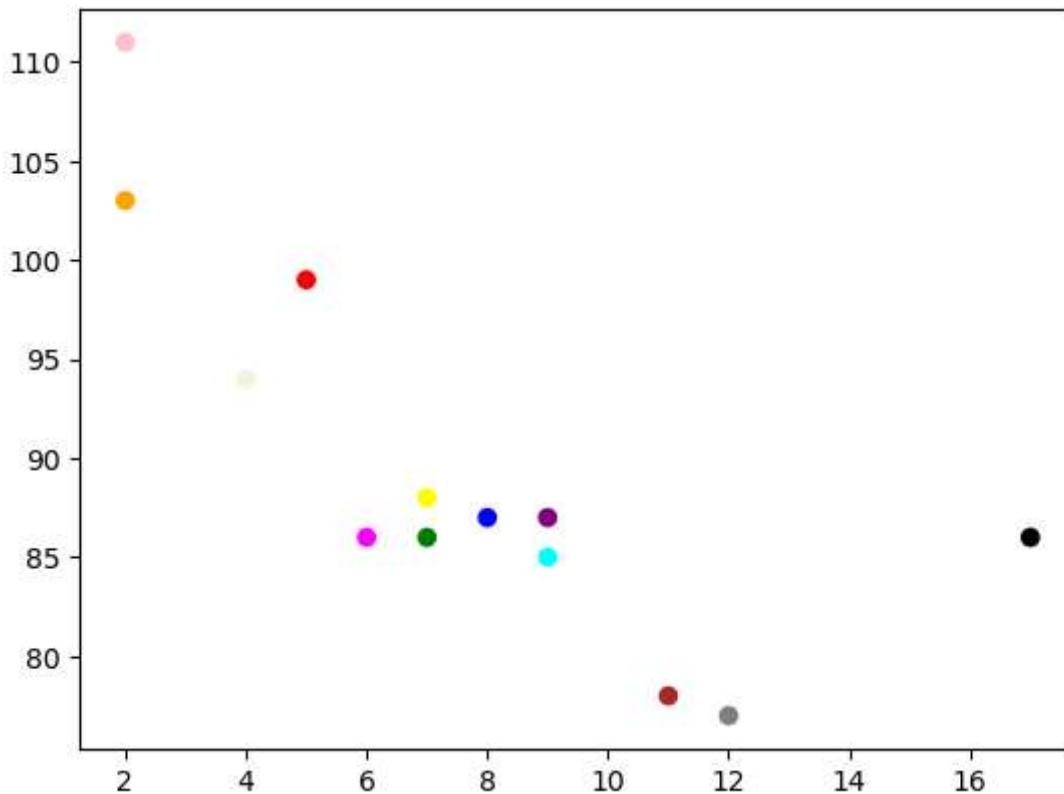
Color Each Dot

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

You cannot use the color argument for this, only the c argument.

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

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"])
plt.scatter(x, y, c= colors)
plt.show()
```



ColorMap

The Matplotlib module has a number of available colormaps. A colormap is like a list of colors, where each color has a value that ranges from 0 to 100. This colormap is called 'viridis' and as you can see it ranges from 0, which is a purple color, up to 100, which is a yellow color.

How to Use the ColorMap

You 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. In addition you have to create an array with values (from 0 to 100), one value for each point in the scatter plot:

Available ColorMaps

You can choose any of the built-in colormaps:

Name Reverse Accent Accent_r
Blues Blues_r BrBG BrBG_r
BuGn BuGn_r
BuPu BuPu_r
CMRmap CMRmap_r
Dark2 Dark2_r GnBu GnBu_r
Greens Greens_r
Greys Greys_r OrRd OrRd_r
Oranges Oranges_r

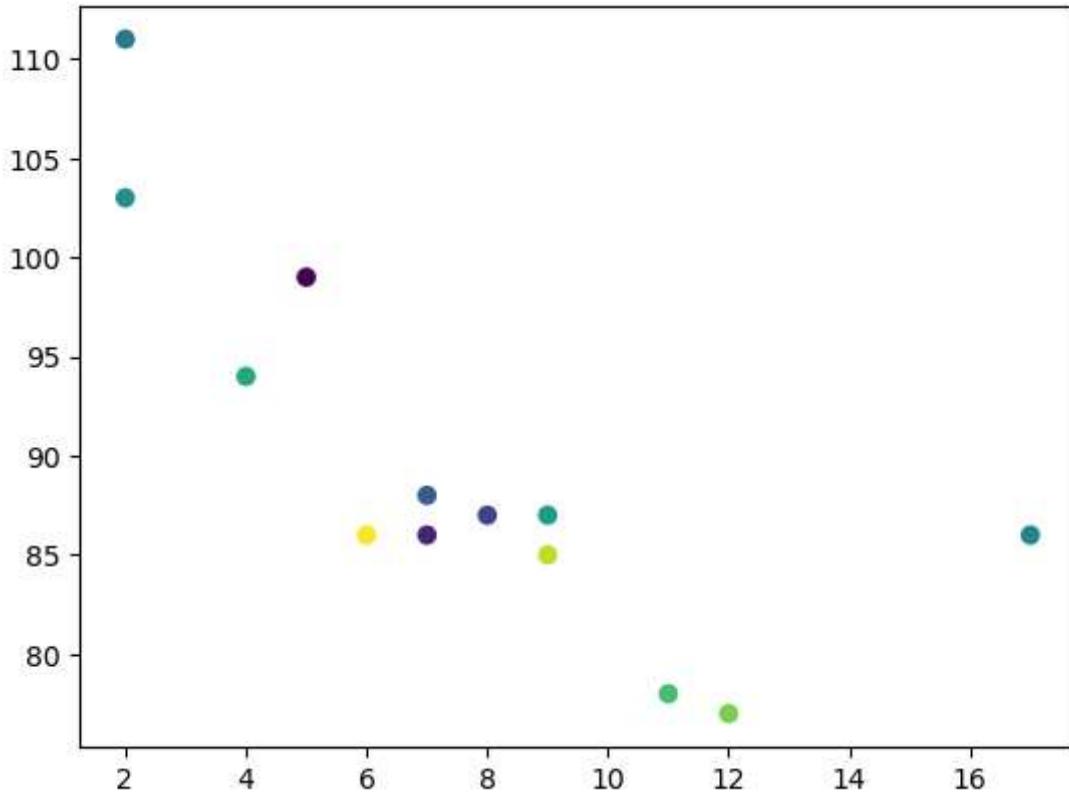
PRGn PRGn_r
Paired Paired_r
Pastel1 Pastel1_r
Pastel2 Pastel2_r
PiYG PiYG_r
PuBu PuBu_r
PuBuGn PuBuGn_r
PuOr PuOr_r
PuRd PuRd_r
Purples Purples_r
RdBu RdBu_r
RdGy RdGy_r
RdPu RdPu_r
RdYIBu RdYIBu_r
RdYIGn RdYIGn_r
Reds Reds_r
Set1 Set1_r
Set2 Set2_r
Set3 Set3_r
Spectral Spectral_r
Wistia Wistia_r
YlGn YlGn_r
YlGnBu YlGnBu_r
YlOrBr YlOrBr_r
YlOrRd YlOrRd_r
afmhot afmhot_r
autumn autumn_r
binary binary_r
bone bone_r
brg brg_r
bwr bwr_r
cividis cividis_r
cool cool_r
coolwarm coolwarm_r
copper copper_r
cubehelix cubehelix_r flag flag_r
gist_earth gist_earth_r
gist_gray gist_gray_r gist_heat gist_heat_r gist_ncar gist_ncar_r gist_rainbow gist_rainbow_r
gist_stern gist_stern_r
gist_yarg gist_yarg_r gnuplot gnuplot_r
gnuplot2 gnuplot2_r
gray gray_r
hot hot_r

```
hsv hsv_r
inferno inferno_r
jet jet_r
magma magma_r nipy_spectral nipy_spectral_r ocean ocean_r pink pink_r
plasma plasma_r
prism prism_r rainbow rainbow_r
seismic seismic_r
spring spring_r
summer summer_r
tab10 tab10_r tab20 tab20_r tab20b tab20b_r
tab20c tab20c_r
terrain terrain_r
twilight twilight_r
twilight_shifted twilight_shifted_r
viridis viridis_r
winter winter_r
```

In [2]: *#Create a color array, and specify a colormap in the scatter plot:*

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

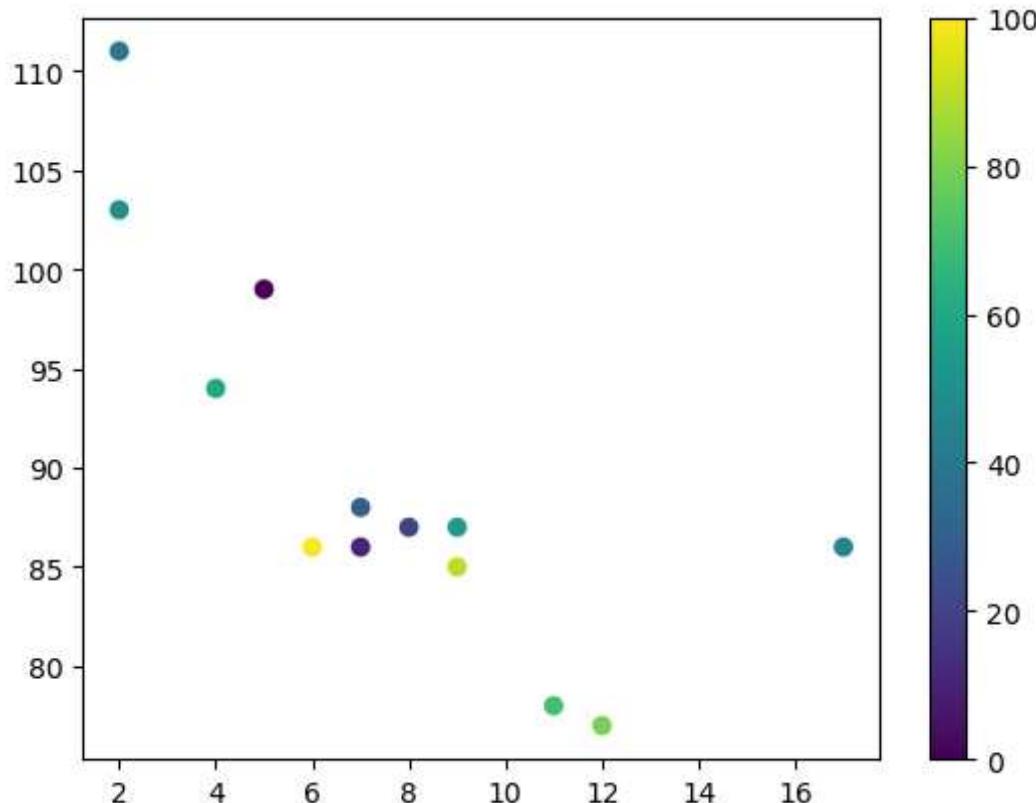
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])
plt.scatter(x,y,c=colors,cmap='viridis')
plt.show()
```



plt.colorbar() : To include the colormap in the drawing

```
In [3]: #Create a color array, and specify a colormap in the scatter plot:
import matplotlib.pyplot as plt
import numpy as np

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()
```



To change Size of each point in scatter plot

You can change the size of the dots 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:

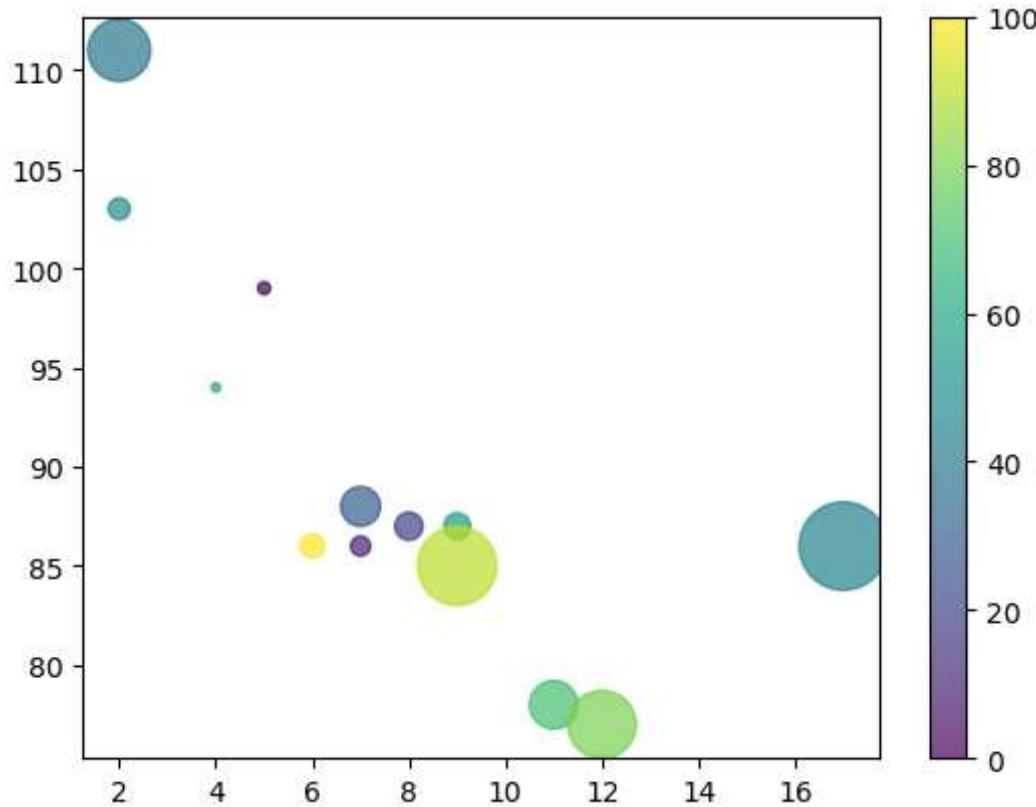
Alpha

You can adjust the transparency of the dots with the `alpha` argument.

```
In [11]: #Create a color array, and specify a colormap in the scatter plot:
import matplotlib.pyplot as plt
import numpy as np

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])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])
plt.scatter(x,y,c=colors,cmap='viridis',s=sizes,alpha=0.7)
plt.colorbar()
plt.show()
```



Combine Color Size and Alpha

You can combine a colormap with different sizes of the dots. This is best visualized if the dots are transparent:

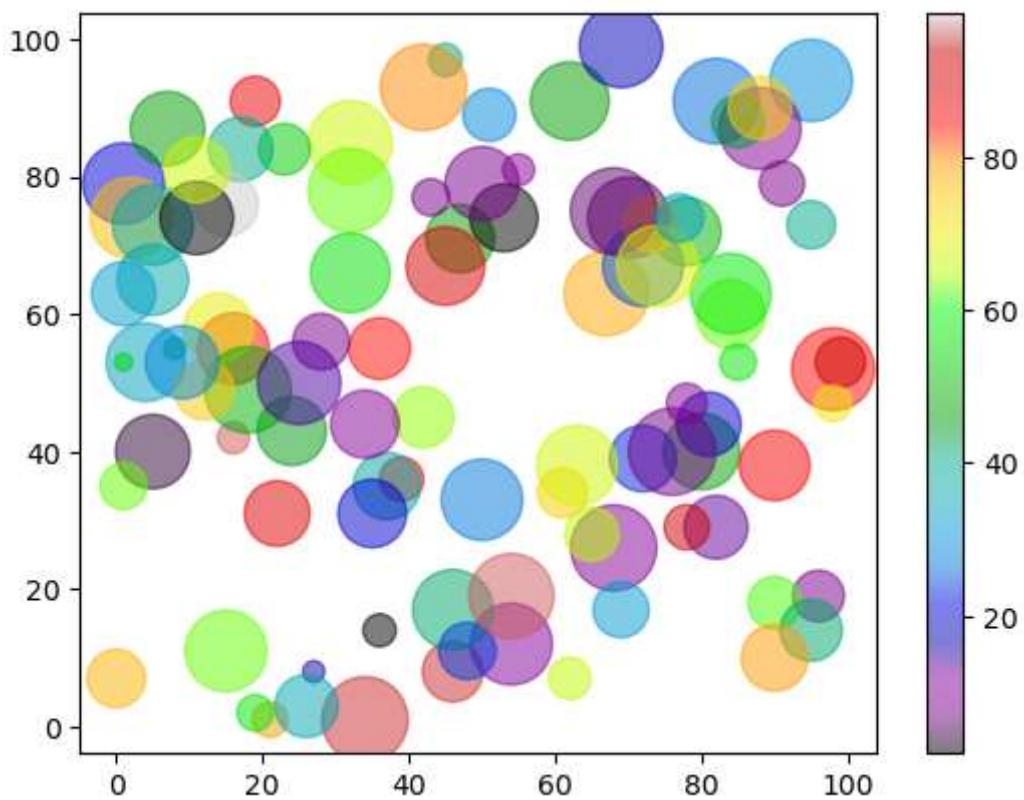
```
In [13]: #Create random arrays with 100 values for x-points, y-points, colors and sizes:
import matplotlib.pyplot as plt
import numpy as np

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()

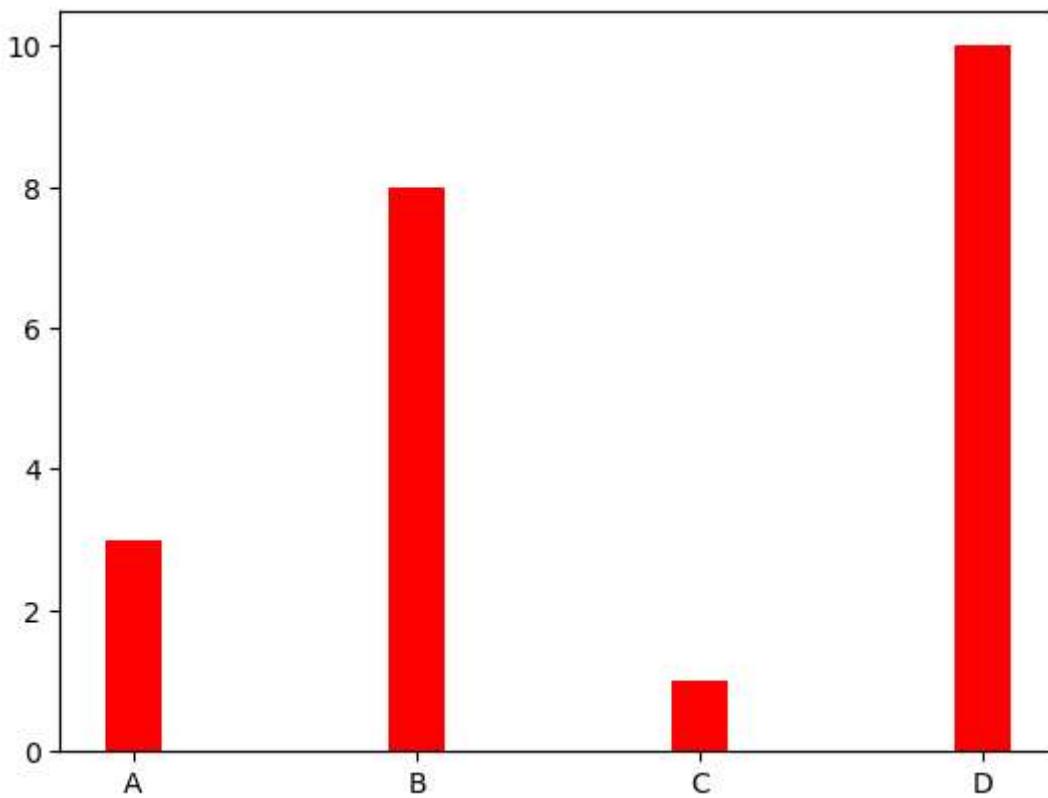
plt.show()
```



Matplotlib Bars

bar() : function to draw bar graphs The bar() function takes arguments that describes the layout of the bars. arguments: color - to set the color of the bars width - to set the width of the bars The default width value is 0.8

```
In [19]: #Creating Bars
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x,y,color="red",width=0.2)
plt.show()
```

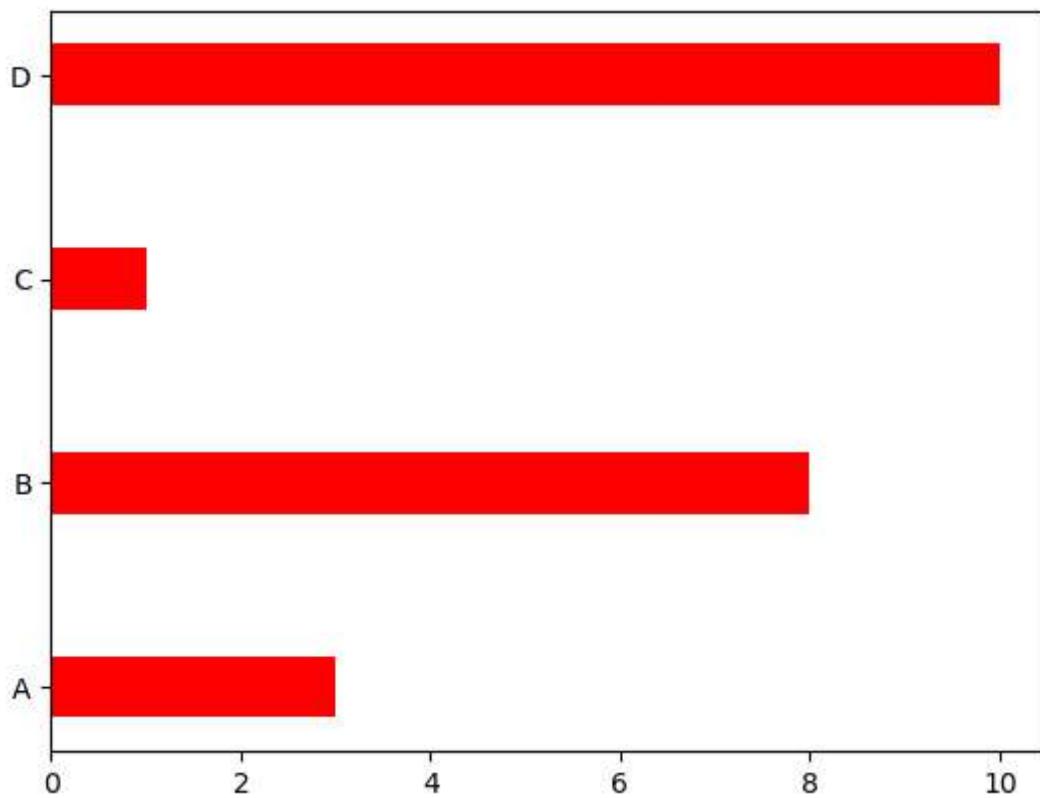


Horizontal Bars

barh() :If you want the bars to be displayed horizontally instead of vertically. arguments:

color - to set the color of the bars
height - to set the width of the bars The default height value is 0.8

```
In [14]: x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.barh(x,y,color="red",height=0.3)
plt.show()
```



Matplotlib Histograms

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

Create Histogram

`hist()` : Function to create histograms. `x=np.random.normal(170,10,250)` - For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10.

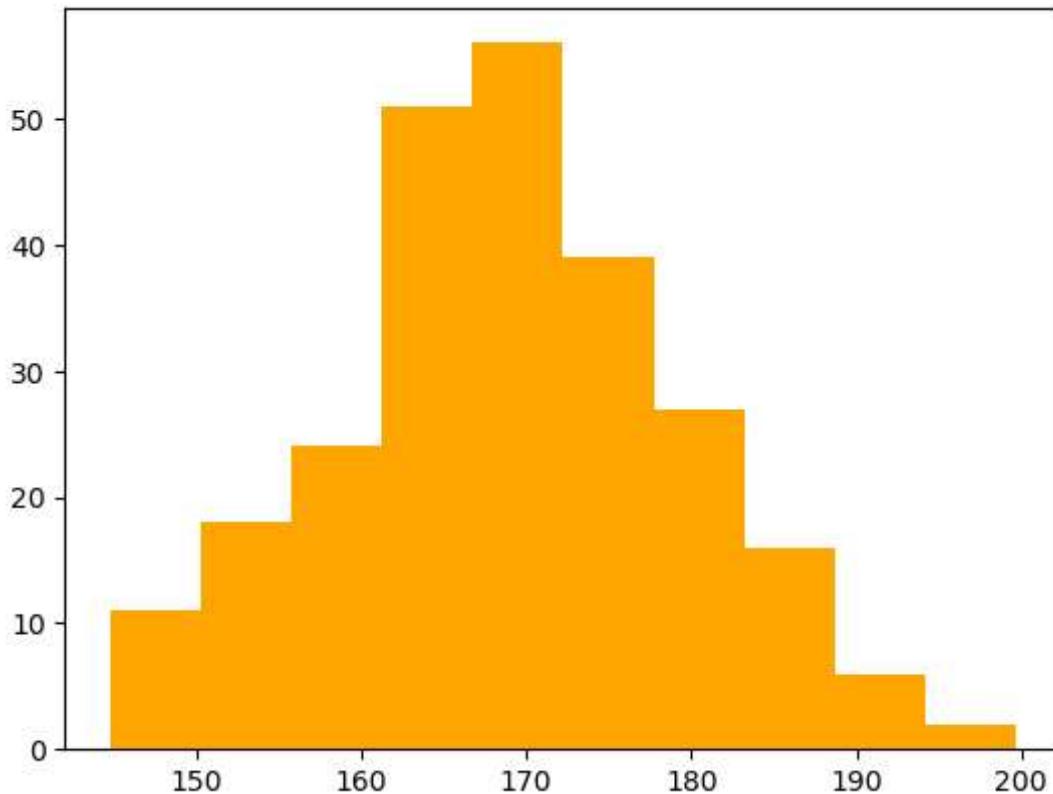
```
In [24]: #A Normal Data Distribution by NumPy:  
x=np.random.normal(170,10,250)  
x
```

```
Out[24]: array([149.40984156, 172.20070137, 182.9825461 , 160.01629785,
 176.37062242, 154.81384488, 156.34181739, 168.96183924,
 182.10927525, 161.3148854 , 173.61579562, 178.38917689,
 185.08803301, 157.36922626, 178.77196659, 182.15528275,
 175.87898964, 172.11882993, 161.2085065 , 161.96483925,
 170.89600639, 176.36523919, 169.66685858, 176.21736014,
 174.90555096, 172.26790753, 155.69966784, 172.34704686,
 151.02141614, 166.21432287, 157.62067683, 165.70995922,
 160.70515073, 152.35829313, 175.53122496, 154.12898471,
 176.65331354, 169.0977088 , 193.27993461, 166.31327322,
 163.30469878, 166.61218706, 161.24598764, 167.395348 ,
 154.09518328, 168.15596491, 186.68538725, 187.97084984,
 178.38292037, 176.38470834, 152.53630323, 194.2066996 ,
 171.91109451, 176.01981051, 166.47759272, 176.11298804,
 173.1154006 , 157.93635071, 185.92016578, 176.03189202,
 159.12310952, 163.88045872, 159.79106996, 163.25688143,
 169.8942413 , 167.83210379, 177.96283374, 172.7871974 ,
 181.48940516, 170.59849588, 164.31484649, 171.7071571 ,
 185.67480648, 167.41363232, 170.71623178, 170.93483208,
 167.2271047 , 165.70887638, 196.1598835 , 176.61071181,
 161.22187639, 149.26682445, 172.52784895, 162.48860003,
 162.80840426, 180.47031873, 174.72529991, 167.08215839,
 163.46230334, 179.13211322, 196.49919045, 167.07802945,
 175.21880432, 169.16221753, 180.7358291 , 175.97954131,
 159.51737313, 173.85307462, 168.49449885, 157.53328 ,
 166.21000269, 165.26533832, 154.97113662, 185.3216149 ,
 164.88646587, 160.9930798 , 169.77887991, 177.28812451,
 176.89011354, 157.69895066, 183.36523731, 162.74602948,
 173.8089871 , 197.19779051, 183.31040258, 162.81846124,
 170.96412207, 161.19143707, 192.59773793, 175.86052451,
 185.62136077, 157.29718082, 170.55385429, 162.3632044 ,
 170.53284382, 152.18482716, 167.15523102, 176.7752647 ,
 158.7306299 , 174.64446533, 182.89550758, 175.95762605,
 169.8065228 , 172.42910273, 168.94388681, 169.84928465,
 162.41353817, 165.65817992, 175.20382808, 190.18713572,
 153.16655711, 176.92939333, 161.24610379, 186.5037073 ,
 170.04592301, 168.73810767, 158.90563409, 177.59629083,
 152.19812591, 160.84202192, 169.53621394, 144.65980683,
 164.59462457, 177.31609348, 176.58617121, 194.41873642,
 173.44676735, 174.76379947, 183.41699968, 173.893927 ,
 178.24738564, 179.85777337, 193.28814015, 156.13063574,
 157.52443894, 158.72903419, 179.34333749, 189.56355356,
 184.96501702, 175.95624024, 169.55285931, 175.6456961 ,
 185.06839624, 177.56850178, 167.99177286, 176.30887846,
 180.392496 , 158.86011632, 180.05261905, 178.25827999,
 172.61269905, 181.81153026, 163.76320173, 177.52191965,
 190.49690673, 170.49275519, 160.38738428, 186.79713057,
 170.80070972, 173.11554995, 172.29916559, 160.67739243,
 169.23211219, 166.2337565 , 158.10995182, 165.0015334 ,
 152.00686398, 164.64347268, 170.20562455, 159.132724 ,
 141.73079518, 163.83998393, 158.39650916, 162.00630963,
 193.34124791, 147.88520027, 144.31157996, 179.01346468,
 169.4643758 , 168.18547279, 172.33903597, 183.97264103,
 164.60420952, 166.25686 , 183.35995494, 173.71196173,
 170.85334784, 169.02973608, 167.36633248, 171.12028343,
 181.71234651, 178.69538492, 169.90728597, 176.82079898,
 169.40321002, 171.91883193, 168.22823517, 154.73059476,
 175.10020451, 188.45769881, 166.51943569, 168.06052995,
 170.55328534, 169.18283863, 180.20958887, 179.82334677,
 168.23093642, 143.12784714, 172.78027548, 171.97454958,
```

```
158.16601619, 172.1732723 , 187.83841031, 156.77193663,
171.0019862 , 175.00143941, 162.81718772, 159.64994263,
170.44018167, 181.71912705])
```

In [25]: #17. Histogram for frequency distribution

```
x=np.random.normal(170,10,250) #Normal data Distribution
plt.hist(x,color="orange") #The hist() function will read the array and produce a hist
plt.show()
```



Matplotlib Pie Charts

pie() : Function to draw pie charts. As you can see the pie chart draws one piece (called a wedge) for each value in the array. By default the plotting of the first wedge starts from the x-axis and moves counterclockwise.

The size of each wedge is determined by comparing the value with all the other values, by using this formula: The value divided by the sum of all values: $x/\text{sum}(x)$

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

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

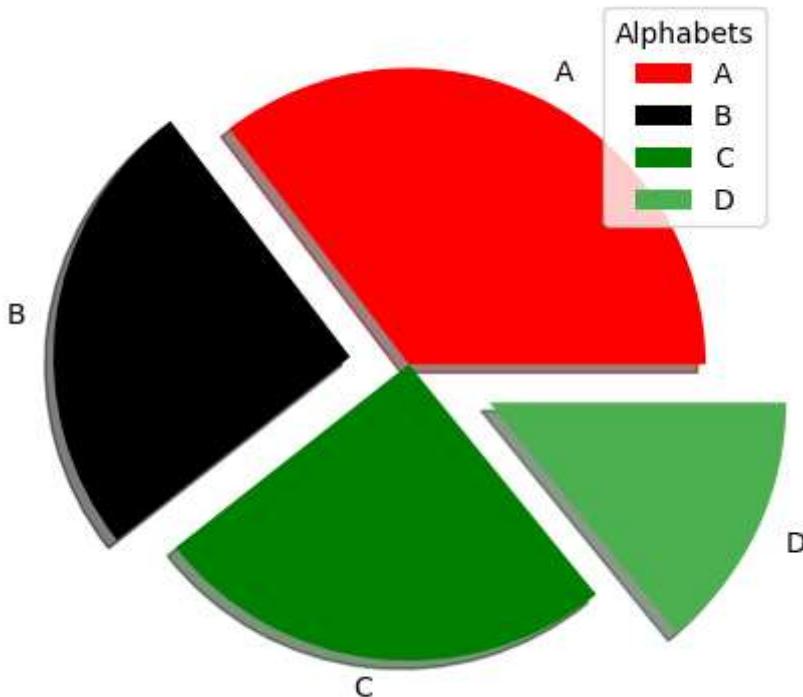
explode - The explode parameter allows you to the wedges to stand out. The explode parameter, if specified, and not None, must be an array with one value for each wedge. Each

value represents how far from the center each wedge is displayed. shadow - Add a shadow to the pie chart by setting the shadows parameter to True

color - 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 [30]:

```
#Creating Pie Charts
y = np.array([35, 25, 25, 14])
myexplode=[0,0.2,0,0.3]
mylabels=["A", "B", "C", "D"]
mycolors=["red", "black", "green", "#4CAF50"]
# plt.pie(y, explode=myexplode, labels=myLabels, colors=myColors, shadow= True, startangle=90)
plt.pie(y, explode=myexplode, labels=mylabels, colors=mycolors, shadow= True, startangle=90)
# plt.Legend(title="Alphabets")
plt.show()
```



Legend

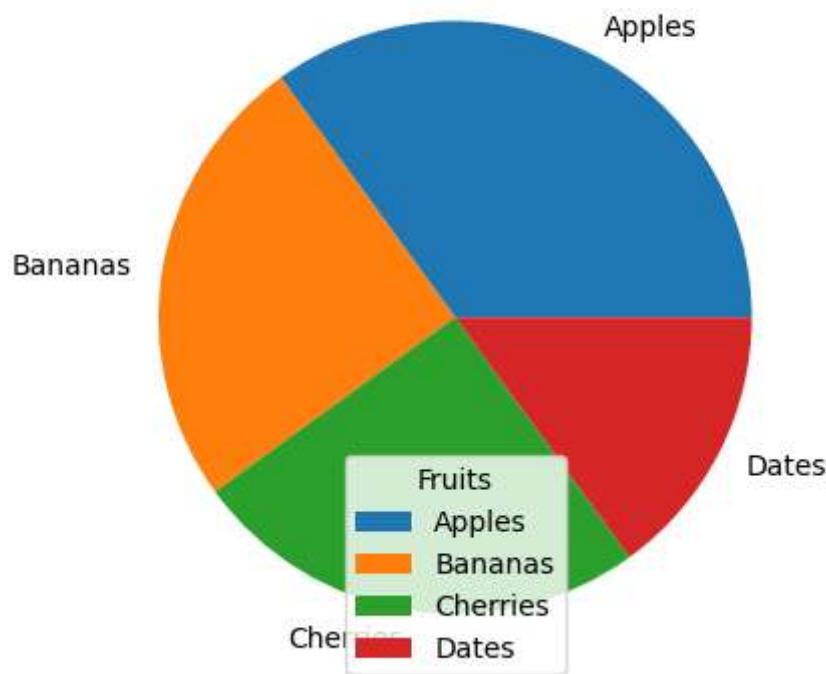
legend() -To add a list of explanation for each wedge in pie chart. Arguments: title - add a header to the legend.

In [32]:

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

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

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



THANK YOU...