Matplotlib Plotting

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.

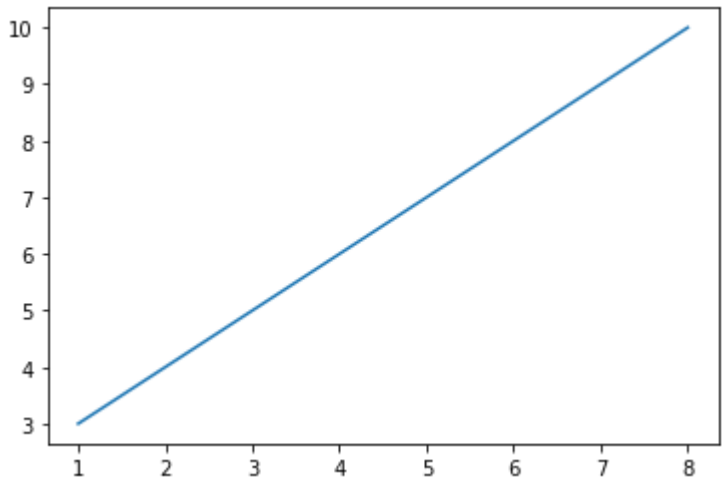
The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the x-axis.

Parameter 2 is an array containing the points on the y-axis.

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.

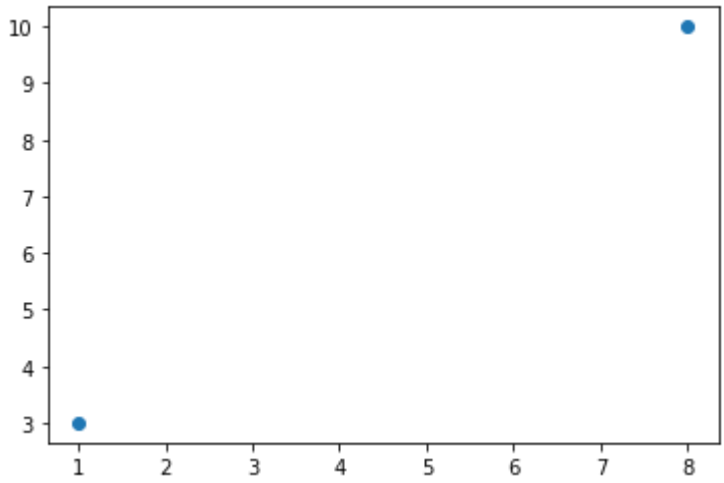
In [ ]:
```
#Draw a line in a diagram from position (1, 3) to position (8, 10):

import matplotlib.pyplot as plt
import numpy as np

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

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



Plotting Without Line

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

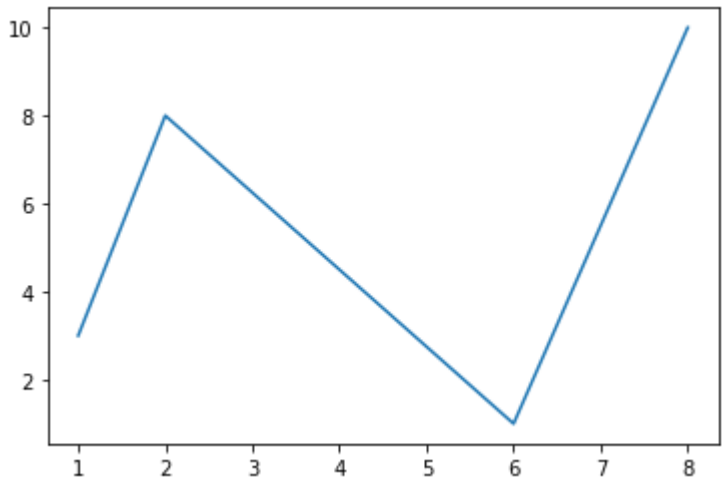
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(xpoints, ypoints, '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 [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

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

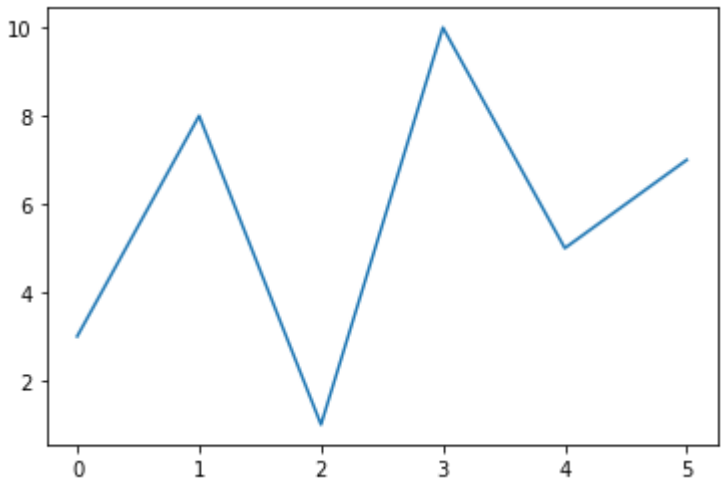


Default X-Points

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.

So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

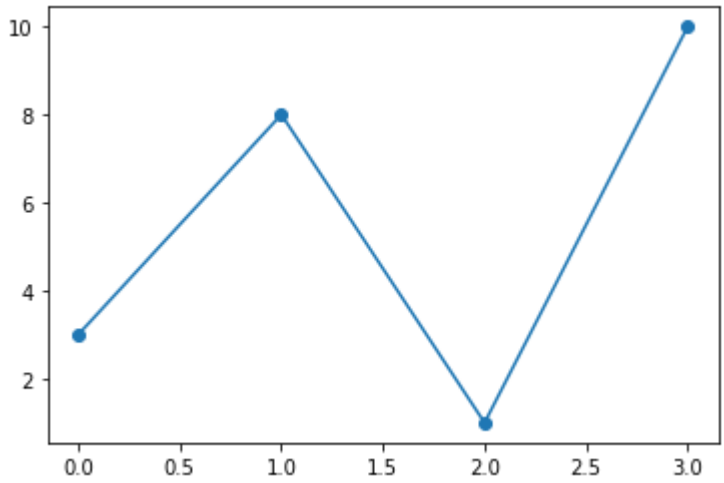
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10, 5, 7])

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



Matplotlib Markers

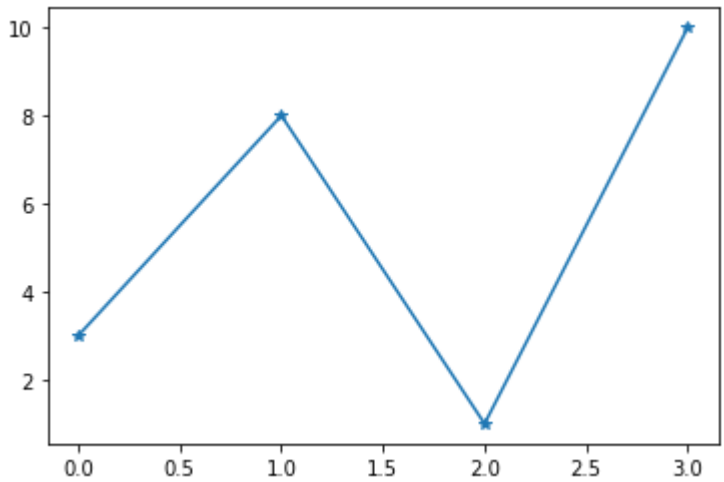
You can use the keyword argument marker to emphasize each point with a specified marker:

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

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

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



Mark each point with a star:

In [ ]:
```
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = '*')
plt.show()
```



Marker Reference

In [ ]:
```
import matplotlib
help(matplotlib.markers)
```

Help on module matplotlib.markers in matplotlib:

NAME
    matplotlib.markers

DESCRIPTION
    This module contains functions to handle markers. Used by both the
    marker functionality of `~matplotlib.axes.Axes.plot` and
    `~matplotlib.axes.Axes.scatter`.

    All possible markers are defined here:

    ============================== ====== ==
    marker   symbol description
    ============================== ====== ==
    ``"."`` |m00| point
    ``","`` |m01| pixel
    ``"o"`` |m02| circle
    ``"v"`` |m03| triangle_down
    ``"^"`` |m04| triangle_up
    ``"<"`` |m05| triangle_left
    ``">"`` |m06| triangle_right
    ``"1"`` |m07| tri_down
    ``"2"`` |m08| tri_up
    ``"3"`` |m09| tri_left
    ``"4"`` |m10| tri_right
    ``"8"`` |m11| octagon
    ``"s"`` |m12| square
    ``"p"`` |m13| pentagon
    ``"P"`` |m23| plus (filled)
    ``"*"`` |m14| star
    ``"h"`` |m15| hexagon1
    ``"H"`` |m16| hexagon2
    ``"+"`` |m17| plus
    ``"x"`` |m18| x
    ``"X"`` |m24| x (filled)
    ``"D"`` |m19| diamond
    ``"d"`` |m20| thin_diamond
    ``"|"`` |m21| vline
    ``"_"`` |m22| hline
    ``0`` (``TICKLEFT``) |m25| tickleft
    ``1`` (``TICKRIGHT``) |m26| tickright
    ``2`` (``TICKUP``) |m27| tickup
    ``3`` (``TICKDOWN``) |m28| tickdown
    ``4`` (``CARETLEFT``) |m29| caretleft
    ``5`` (``CARETRIGHT``) |m30| caretright
    ``6`` (``CARETUP``) |m31| caretup
    ``7`` (``CARETDOWN``) |m32| caretdown
    ``8`` (``CARETLEFTBASE``) |m33| caretleft (centered at base)
    ``9`` (``CARETRIGHTBASE``) |m34| caretright (centered at base)
    ``10`` (``CARETUPBASE``) |m35| caretup (centered at base)
    ``11`` (``CARETDOWNBASE``) |m36| caretdown (centered at base)
    ``"None"``, ``" "`` or ``""``   nothing
    ``'$...$'`` |m37| Render the string using mathtext.
                                           E.g ``"$f$"`` for marker showing the
                                           letter ``f``.
    ``verts``   A list of (x, y) pairs used for Path
                                           vertices. The center of the marker is
                                           located at (0, 0) and the size is
                                           normalized, such that the created path
                                           is encapsulated inside the unit cell.
    path   A `~matplotlib.path.Path` instance.
    ``(numsides, 0, angle)``   A regular polygon with ``numsides``

		sides, rotated by ``angle``.
``(numsides, 1, angle)``		A star-like symbol with ``numsides`` sides, rotated by ``angle``.
``(numsides, 2, angle)``		An asterisk with ``numsides`` sides, rotated by ``angle``.

``None`` is the default which means 'nothing', however this table is
referred to from other docs for the valid inputs from marker inputs and in
those cases ``None`` still means 'default'.

Note that special symbols can be defined via the
:doc:`STIX math font </tutorials/text/mathtext>`,
e.g. ``"$\u266B$"``. For an overview over the STIX font symbols refer to the
`STIX font table <http://www.stixfonts.org/allGlyphs.html>`_.
Also see the :doc:`/gallery/text_labels_and_annotations/stix_fonts_demo`.

Integer numbers from ``0`` to ``11`` create lines and triangles. Those are
equally accessible via capitalized variables, like ``CARETDOWNBASE``.
Hence the following are equivalent::

```
    plt.plot([1, 2, 3], marker=11)
    plt.plot([1, 2, 3], marker=matplotlib.markers.CARETDOWNBASE)
```

Examples showing the use of markers:

* :doc:`/gallery/lines_bars_and_markers/marker_reference`
* :doc:`/gallery/lines_bars_and_markers/marker_fillstyle_reference`
* :doc:`/gallery/shapes_and_collections/marker_path`

.. |m00| image:: /_static/markers/m00.png
.. |m01| image:: /_static/markers/m01.png
.. |m02| image:: /_static/markers/m02.png
.. |m03| image:: /_static/markers/m03.png
.. |m04| image:: /_static/markers/m04.png
.. |m05| image:: /_static/markers/m05.png
.. |m06| image:: /_static/markers/m06.png
.. |m07| image:: /_static/markers/m07.png
.. |m08| image:: /_static/markers/m08.png
.. |m09| image:: /_static/markers/m09.png
.. |m10| image:: /_static/markers/m10.png
.. |m11| image:: /_static/markers/m11.png
.. |m12| image:: /_static/markers/m12.png
.. |m13| image:: /_static/markers/m13.png
.. |m14| image:: /_static/markers/m14.png
.. |m15| image:: /_static/markers/m15.png
.. |m16| image:: /_static/markers/m16.png
.. |m17| image:: /_static/markers/m17.png
.. |m18| image:: /_static/markers/m18.png
.. |m19| image:: /_static/markers/m19.png
.. |m20| image:: /_static/markers/m20.png
.. |m21| image:: /_static/markers/m21.png
.. |m22| image:: /_static/markers/m22.png
.. |m23| image:: /_static/markers/m23.png
.. |m24| image:: /_static/markers/m24.png
.. |m25| image:: /_static/markers/m25.png
.. |m26| image:: /_static/markers/m26.png
.. |m27| image:: /_static/markers/m27.png
.. |m28| image:: /_static/markers/m28.png
.. |m29| image:: /_static/markers/m29.png
.. |m30| image:: /_static/markers/m30.png
.. |m31| image:: /_static/markers/m31.png

```
    .. |m32| image:: /_static/markers/m32.png
    .. |m33| image:: /_static/markers/m33.png
    .. |m34| image:: /_static/markers/m34.png
    .. |m35| image:: /_static/markers/m35.png
    .. |m36| image:: /_static/markers/m36.png
    .. |m37| image:: /_static/markers/m37.png

CLASSES
    builtins.object
        MarkerStyle

    class MarkerStyle(builtins.object)
     |  MarkerStyle(marker=None, fillstyle=None)
     |
     |  Methods defined here:
     |
     |  __bool__(self)
     |
     |  __init__(self, marker=None, fillstyle=None)
     |      Attributes
     |      ----------
     |      markers : list of known marks
     |
     |      fillstyles : list of known fillstyles
     |
     |      filled_markers : list of known filled markers.
     |
     |      Parameters
     |      ----------
     |      marker : str or array-like, optional, default: None
     |          See the descriptions of possible markers in the module docstring.
     |
     |      fillstyle : str, optional, default: 'full'
     |          'full', 'left", 'right', 'bottom', 'top', 'none'
     |
     |  get_alt_path(self)
     |
     |  get_alt_transform(self)
     |
     |  get_capstyle(self)
     |
     |  get_fillstyle(self)
     |
     |  get_joinstyle(self)
     |
     |  get_marker(self)
     |
     |  get_path(self)
     |
     |  get_snap_threshold(self)
     |
     |  get_transform(self)
     |
     |  is_filled(self)
     |
     |  set_fillstyle(self, fillstyle)
     |      Sets fillstyle
     |
     |      Parameters
     |      ----------
     |      fillstyle : string amongst known fillstyles
     |
     |  set_marker(self, marker)
```

```
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  filled_markers = ('o', 'v', '^', '<', '>', '8', 's', 'p', '*', 'h', 'H...
 |
 |  fillstyles = ('full', 'left', 'right', 'bottom', 'top', 'none')
 |
 |  markers = {'.': 'point', ',': 'pixel', 'o': 'circle', 'v': 'triangle_d...

DATA
    CARETDOWN = 7
    CARETDOWNBASE = 11
    CARETLEFT = 4
    CARETLEFTBASE = 8
    CARETRIGHT = 5
    CARETRIGHTBASE = 9
    CARETUP = 6
    CARETUPBASE = 10
    TICKDOWN = 3
    TICKLEFT = 0
    TICKRIGHT = 1
    TICKUP = 2
    rcParams = RcParams({'_internal.classic_mode': False,
         ...nor.widt...

FILE
    /usr/local/lib/python3.8/dist-packages/matplotlib/markers.py
```

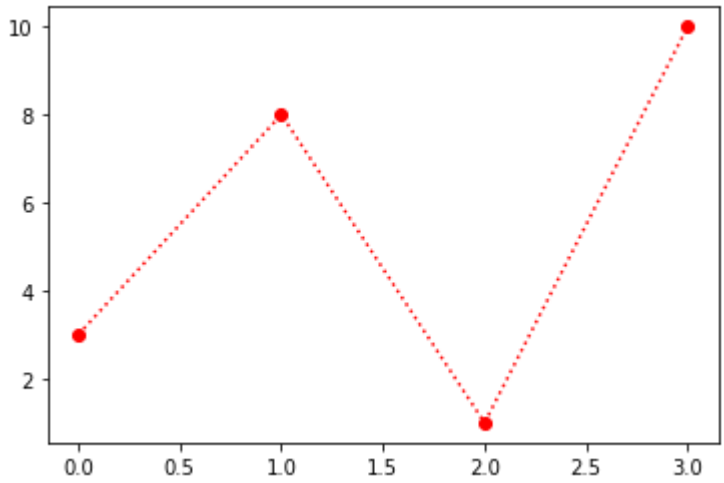
Format Strings fmt

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

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

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

plt.plot(ypoints, 'o:r')
plt.show()
```



The marker value can be anything from the Marker Reference above.

The line value can be one of the following:

Line Syntax	Description
'-'	Solid line
':'	Dotted line
'--'	Dashed line
'-.'	Dashed/dotted line

Note: If you leave out the line value in the fmt parameter, no line will be plotted.

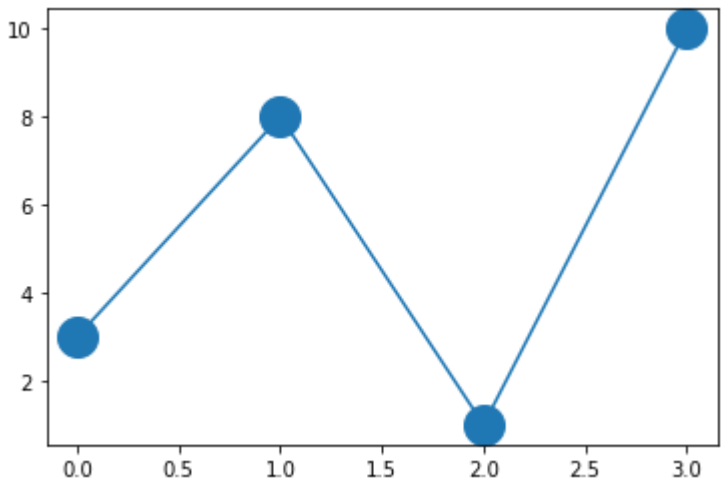
The short color value can be one of the following:

Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Marker Size

You can use the keyword argument markersize or the shorter version, ms to set the size of the markers:

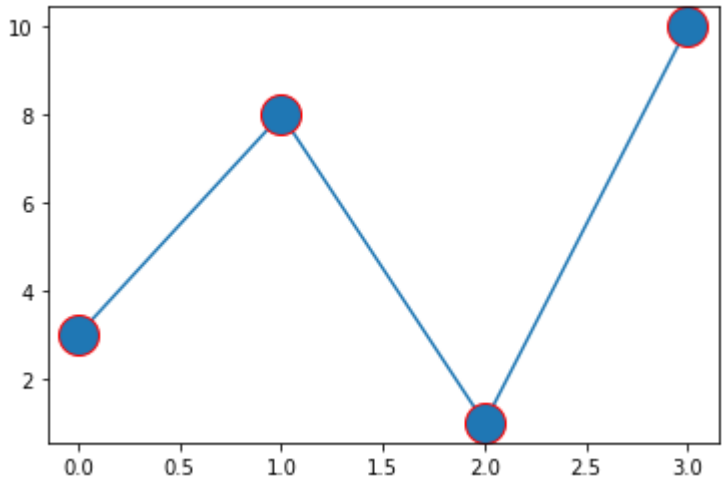
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



Marker Color

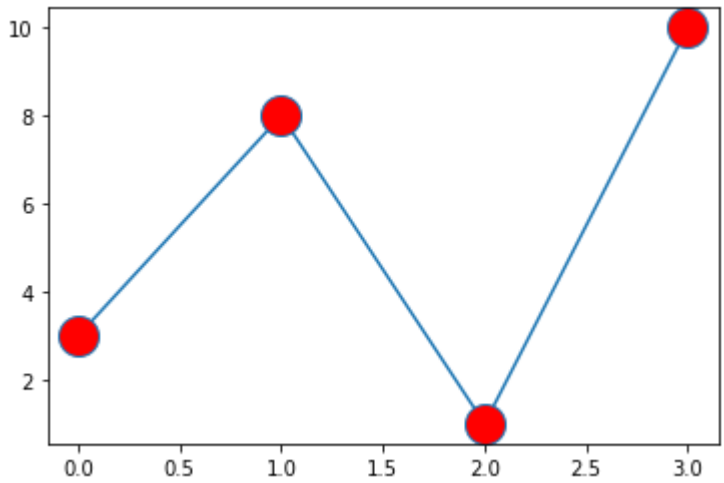
You can use the keyword argument markeredgecolor or the shorter mec to set the color of the edge of the markers:

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

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

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



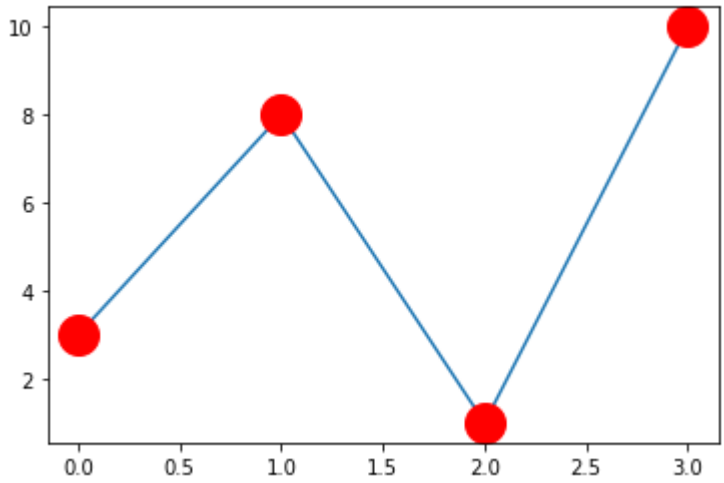
You can use the keyword argument markerfacecolor or the shorter mfc to set the color inside the edge of the markers:

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

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

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

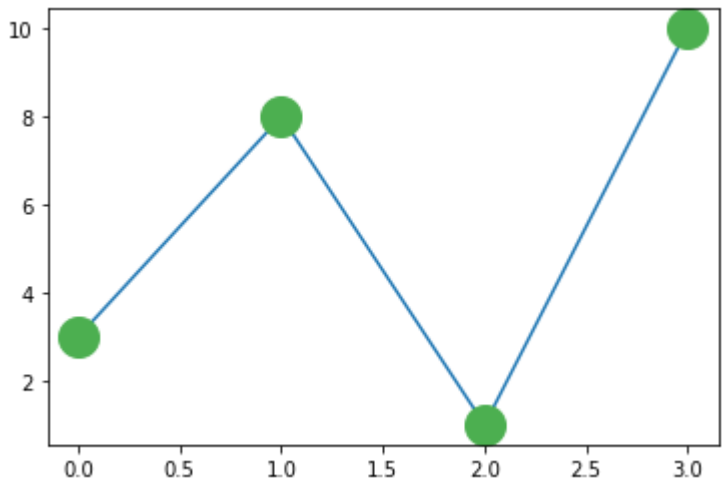


Use both the mec and mfc arguments to color the entire marker:

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

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

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

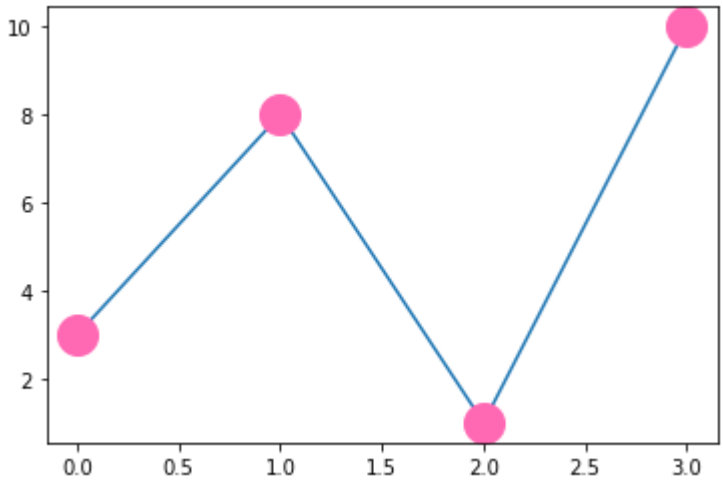


In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc = '#4CAF50')

plt.show()
```



In [ ]:

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

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

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'hotpink', mfc = 'hotpink')

plt.show()
```

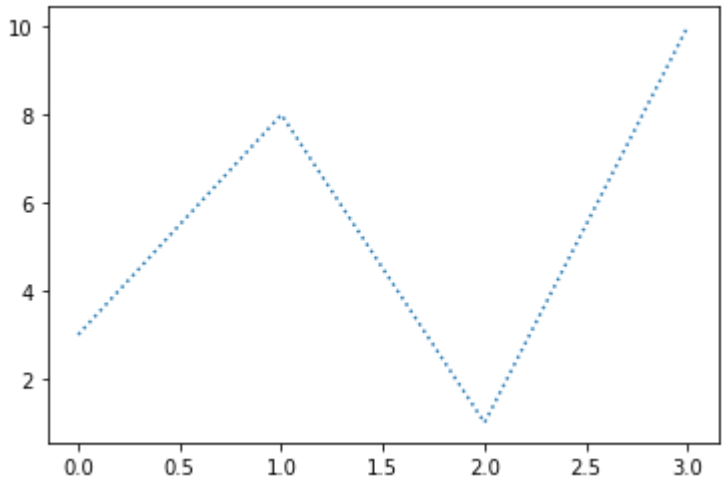


Matplotlib Line

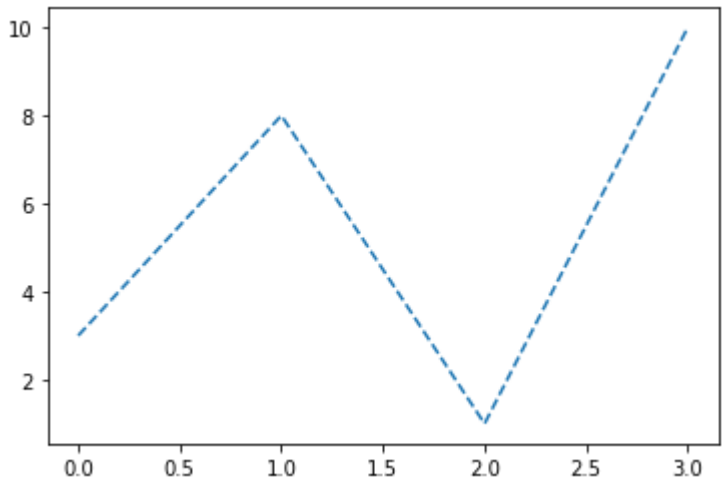
Linestyle

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

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

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

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



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

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

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



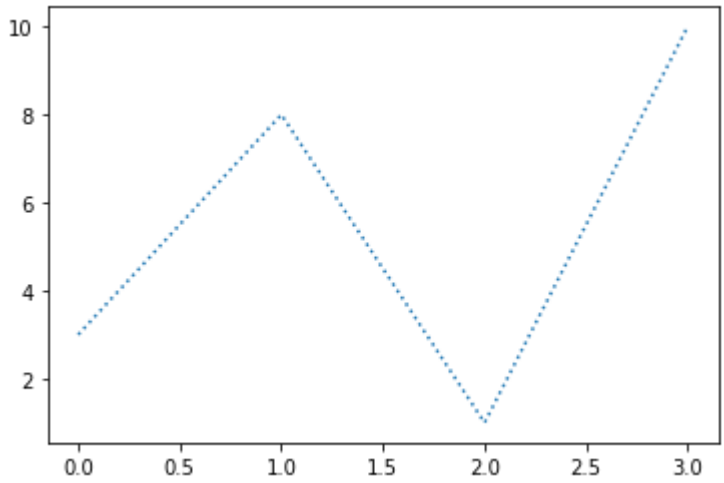
The line style can be written in a shorter syntax:

linestyle can be written as ls.

dotted can be written as :.

dashed can be written as --.

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

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

plt.plot(ypoints, ls = ':')
plt.show()
```



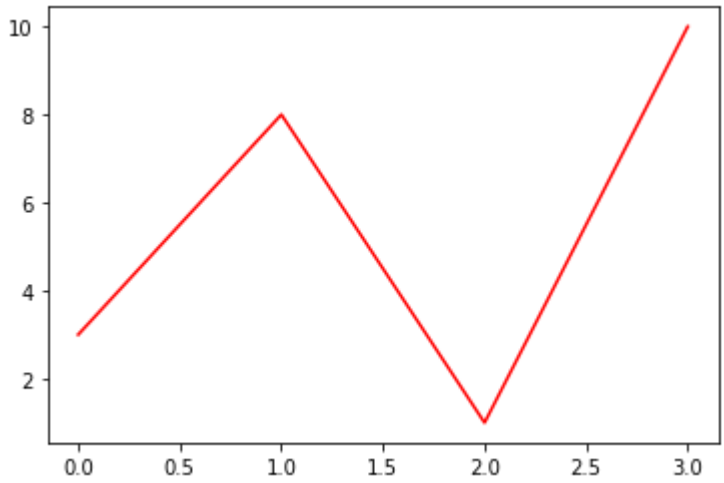
You can choose any of these styles:

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

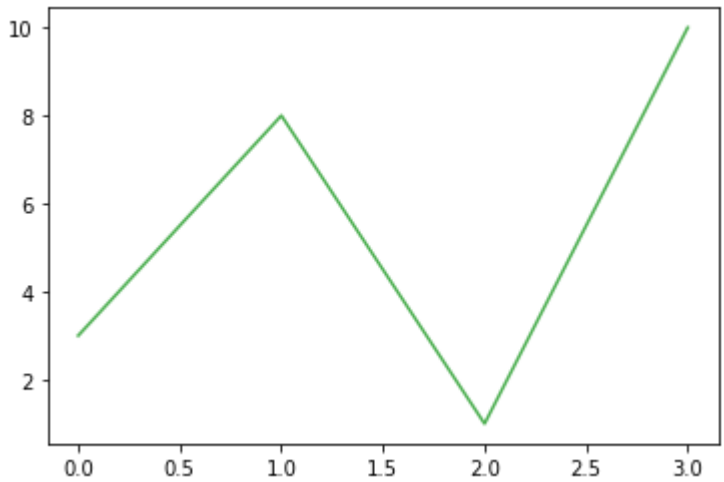
Line Color

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

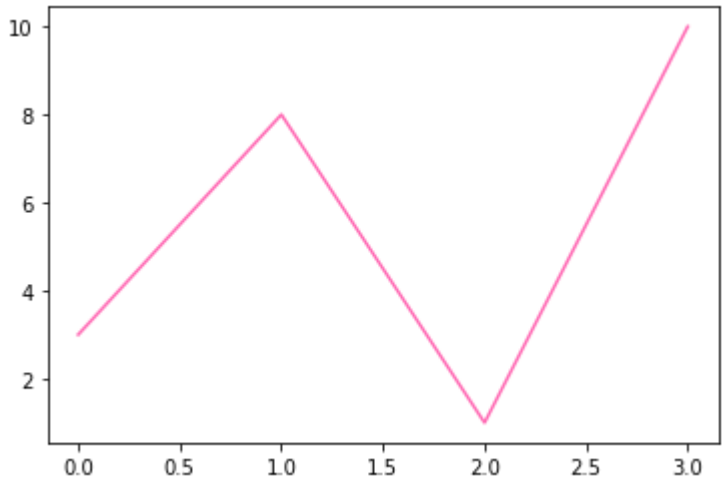
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



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

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

plt.plot(ypoints, c = '#4CAF50')
plt.show()
```



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

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

plt.plot(ypoints, c = 'hotpink')
plt.show()
```

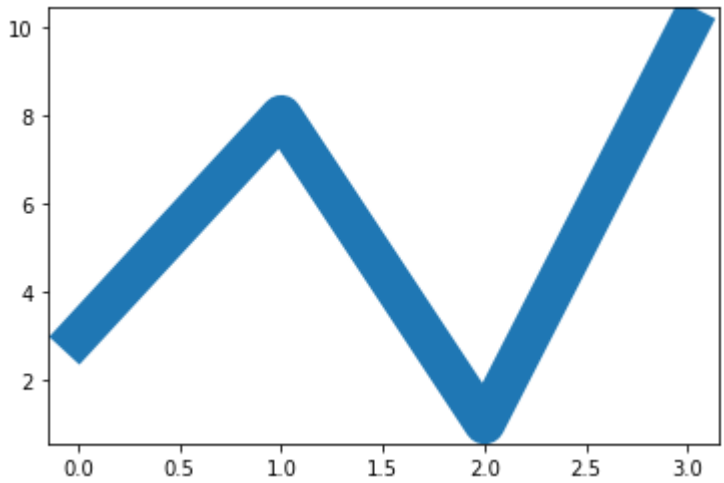


Line Width

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:

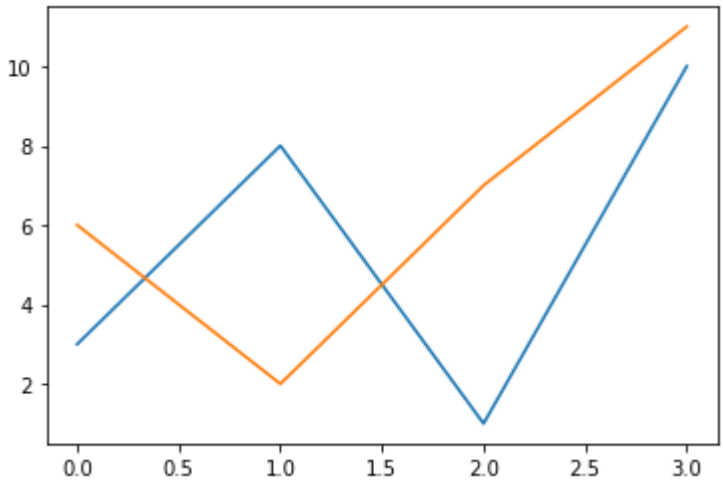
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, linewidth = '20.5')
plt.show()
```



Multiple Lines

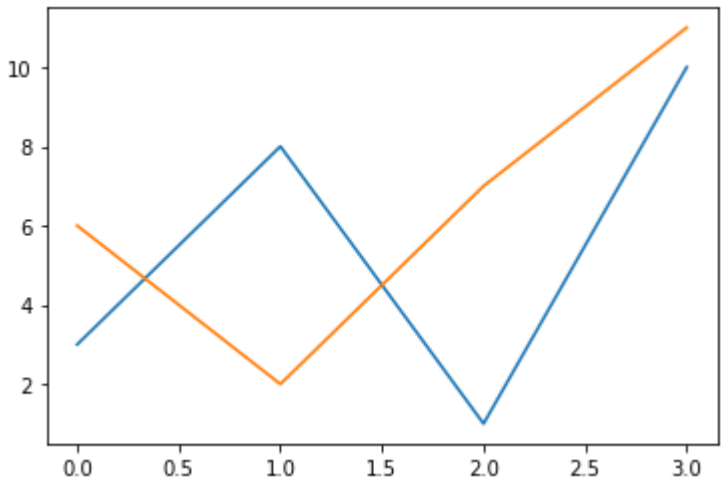
You can plot as many lines as you like by simply adding more plt.plot() functions:

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

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

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

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()
```

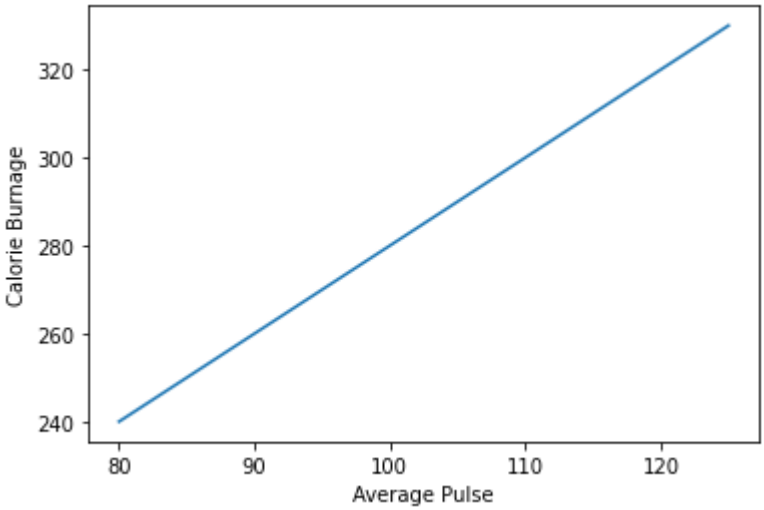


Matplotlib Labels and Title

Create Labels for a Plot

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

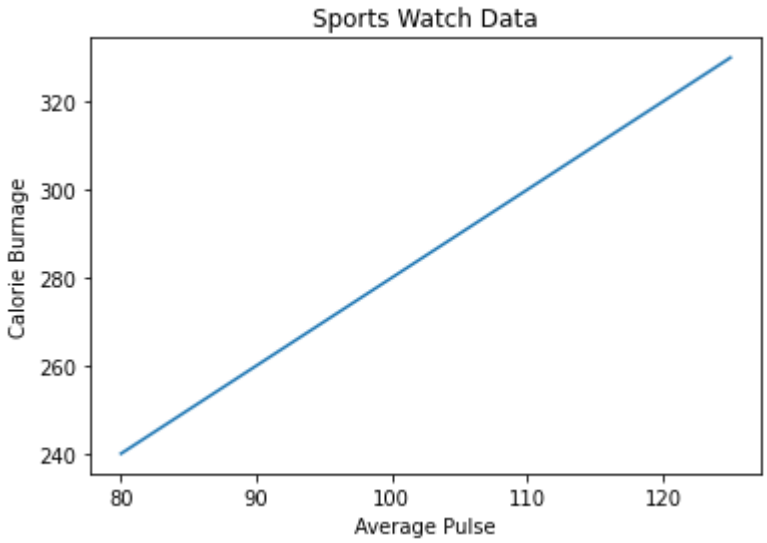
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

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 Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```



Create a Title for a Plot

With Pyplot, you can use the title() function to set a title for the plot.

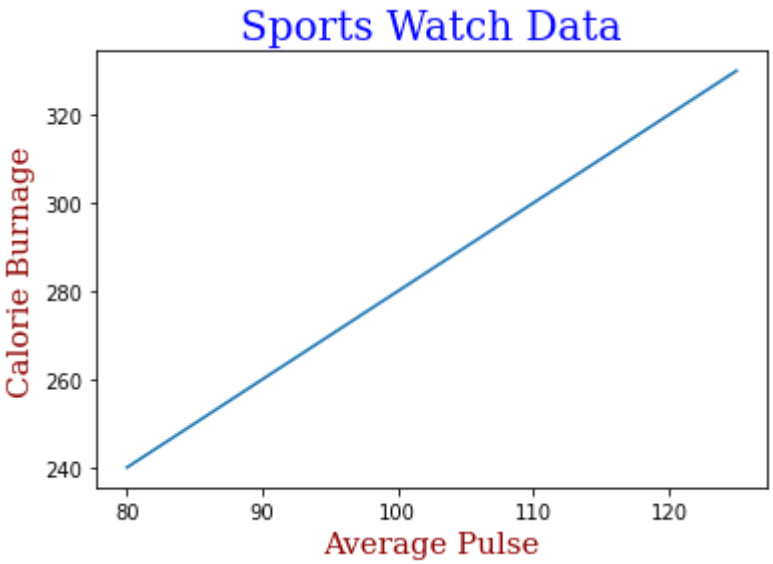
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

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.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```



Set Font Properties for Title and Labels

You can use the fontdict parameter in xlabel(), ylabel(), and title() to set font properties for the title and labels.

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

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 Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

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

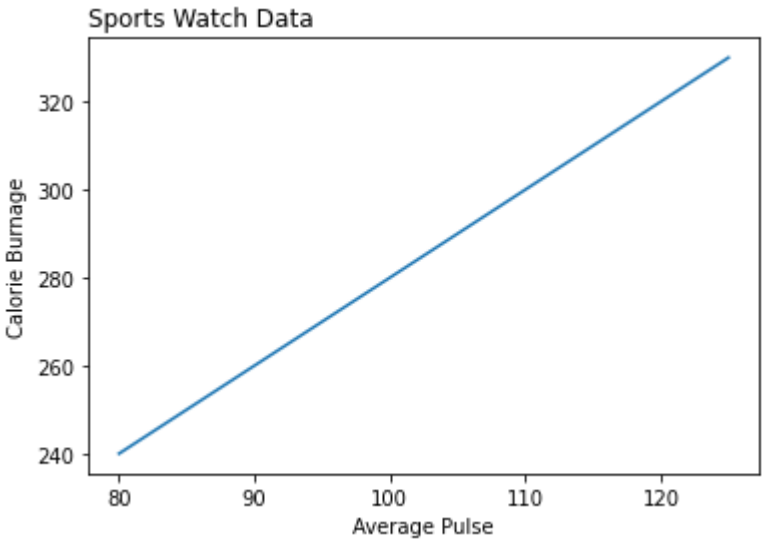


Position the Title

You can use the loc parameter in title() to position the title.

Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

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

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

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

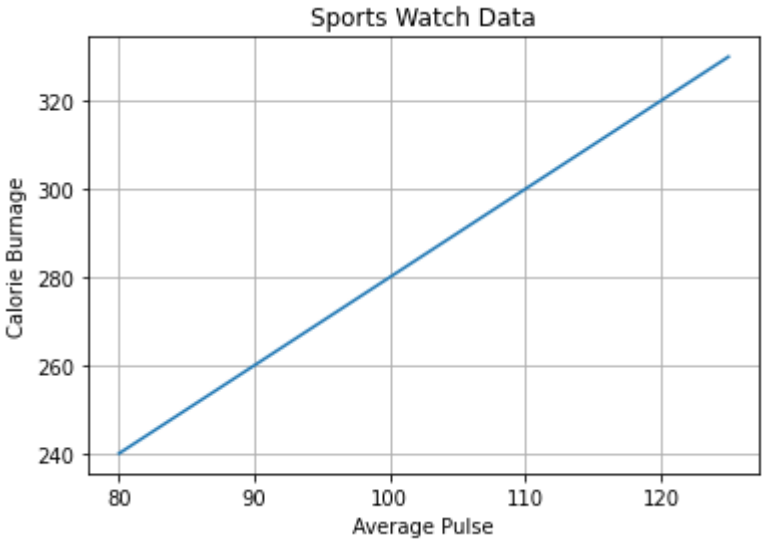


Matplotlib Adding Grid Lines

Add Grid Lines to a Plot

With Pyplot, you can use the grid() function to add grid lines to the plot.

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

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("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()

plt.show()
```



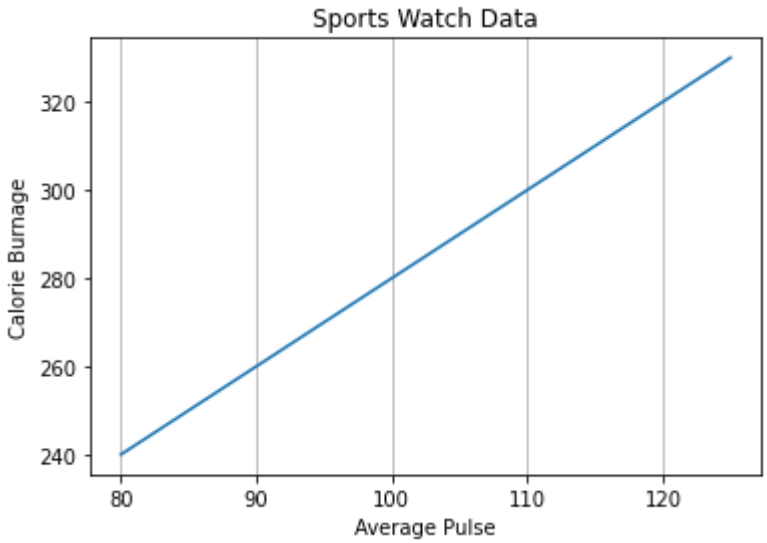
Specify Which Grid Lines to Display

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

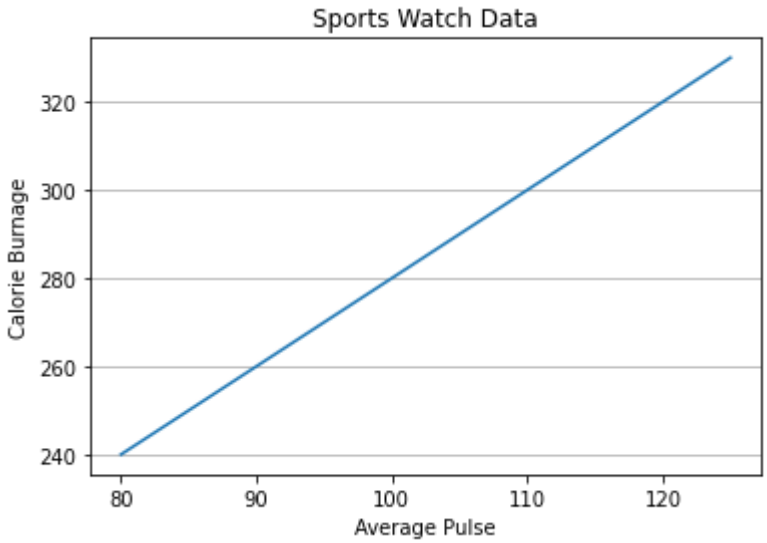
Legal values are: 'x', 'y', and 'both'. Default value is 'both'.

In [ ]:

```
#Display only grid lines for the x-axis:

import numpy as np
import matplotlib.pyplot as plt

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("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(axis = 'x')

plt.show()
```



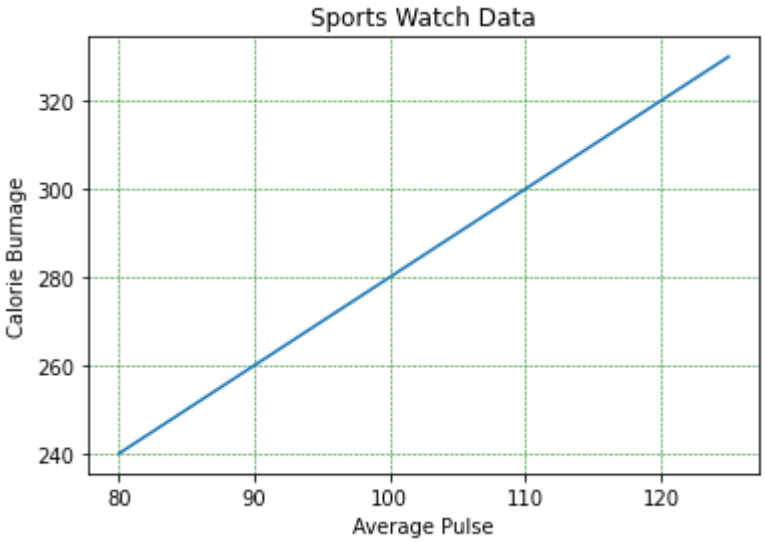
In [ ]:
```
#Display only grid lines for the y-axis:

import numpy as np
import matplotlib.pyplot as plt

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("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(axis = 'y')

plt.show()
```



Set Line Properties for the Grid

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

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

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("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)

plt.show()
```

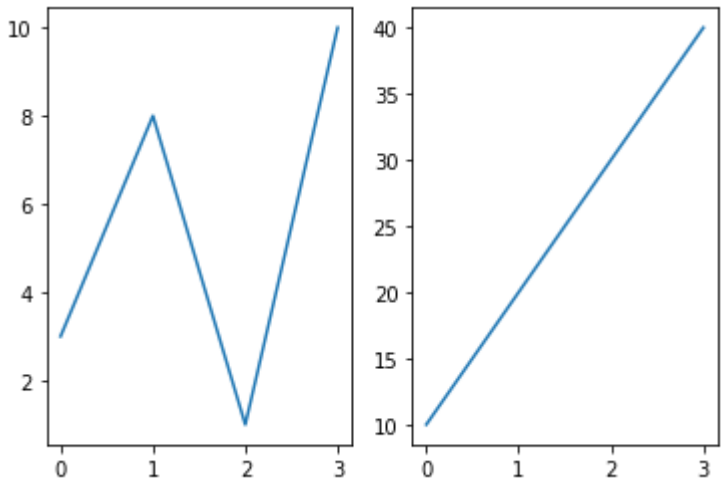


Matplotlib Subplot

Display Multiple Plots

With the subplot() function you can draw multiple plots in one figure:

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

#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()
```



The 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.

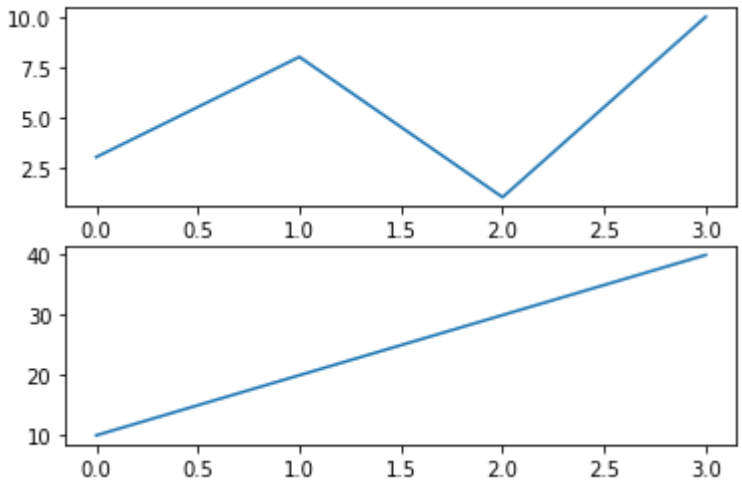
The third argument represents the index of the current plot.

```
plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, and this plot is the first plot.
```

```
plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this plot is the second plot.
```

So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this:

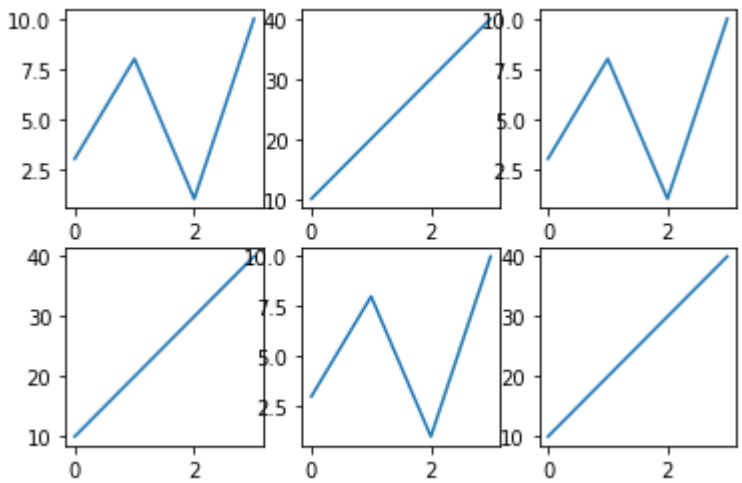
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.subplot(2, 1, 1)
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)
plt.plot(x,y)

plt.show()
```



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

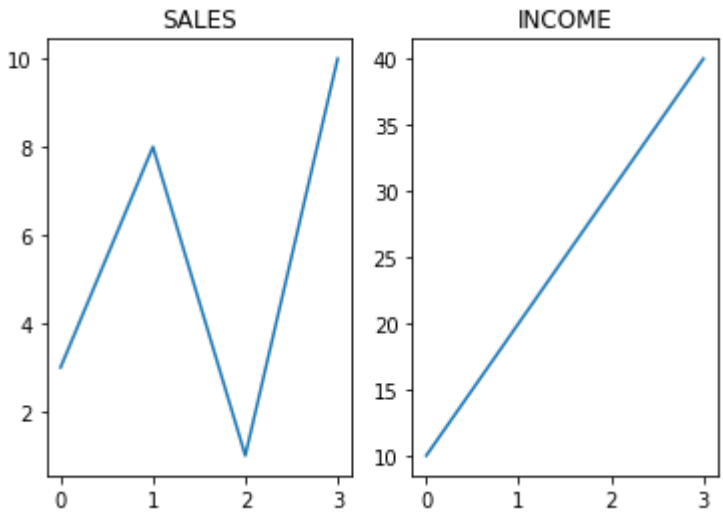
In [1]:

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

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()
```



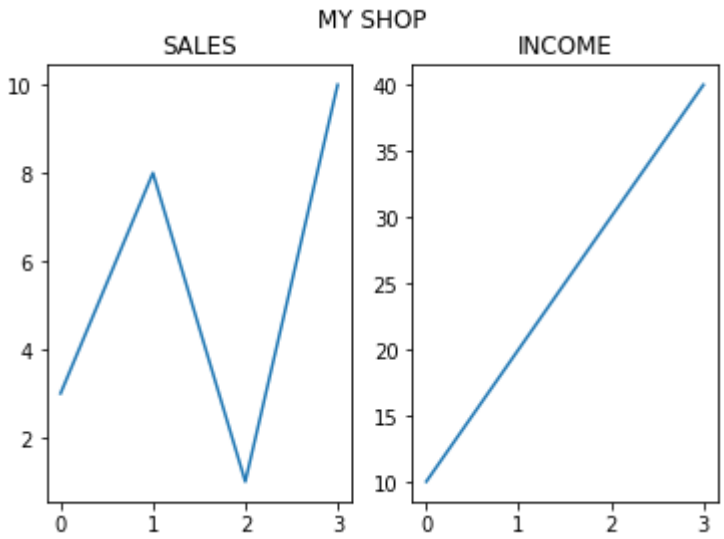
You can add a title to each plot with the title() function:

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

#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()
```



You can add a title to the entire figure with the suptitle() function:

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

#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()
```



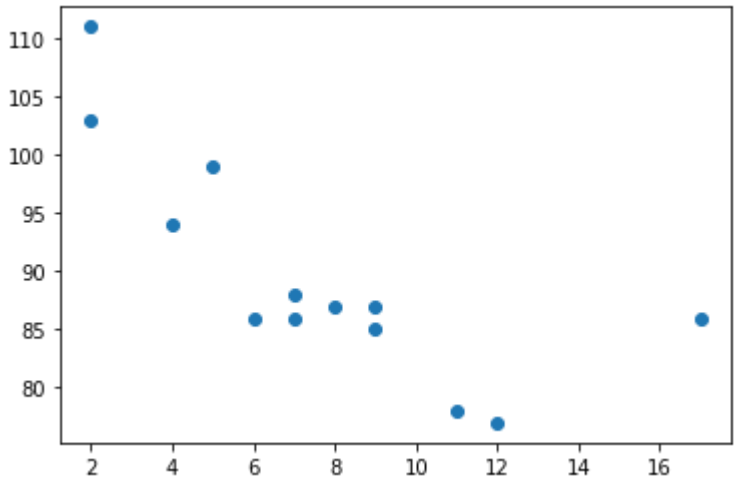
Matplotlib Scatter

Creating Scatter Plots

With Pyplot, 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 [ ]:
```
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])

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.

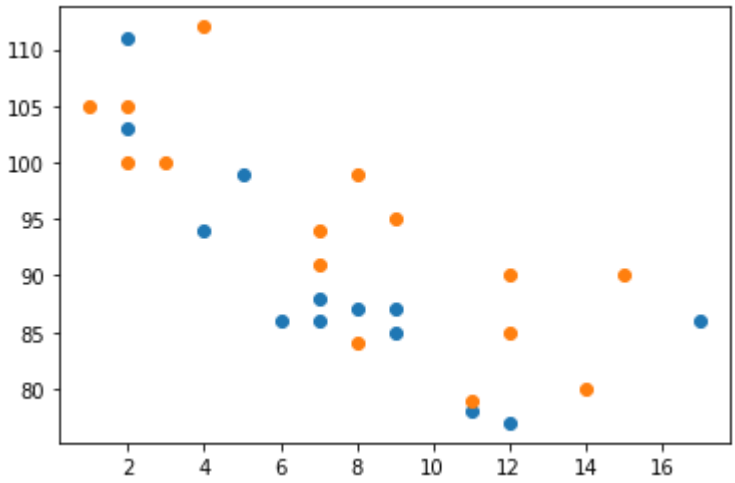
Are there any relationships between the observations?

It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

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?

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

#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()
```

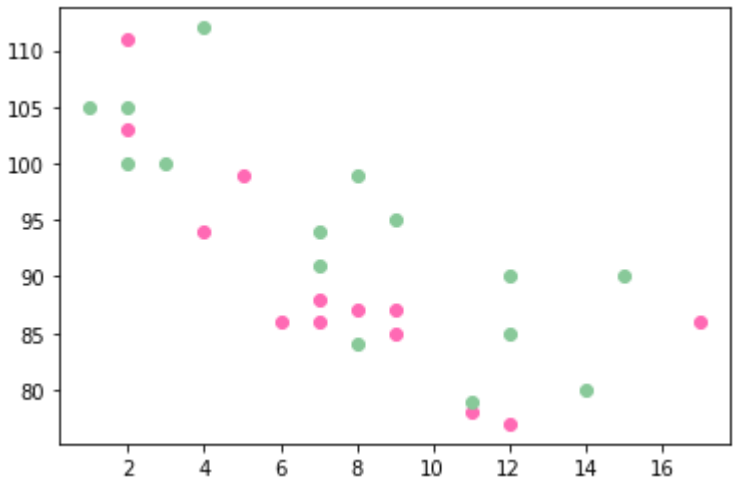


By comparing the two plots, I think it is safe to say that they both gives us the same conclusion: the newer the car, the faster it drives.

Colors

You can set your own color for each scatter plot with the color or the c argument:

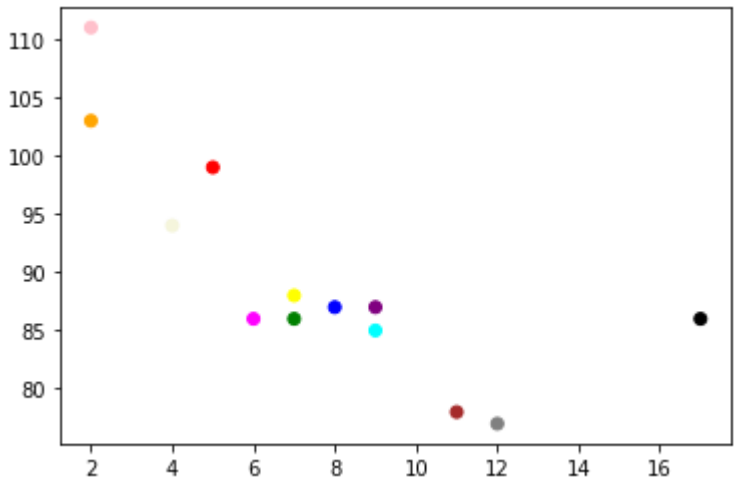
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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])
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:

In [ ]:
```
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","be
ige","brown","gray","cyan","magenta"])

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

plt.show()
```



Colormaps

https://matplotlib.org/stable/tutorials/colors/colormaps.html (https://matplotlib.org/stable/tutorials/colors/colormaps.html)

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.

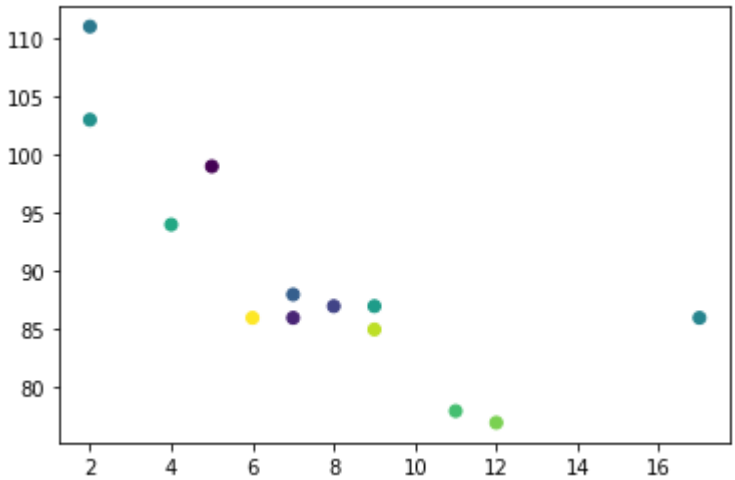
Here is an example of a colormap:

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.

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.

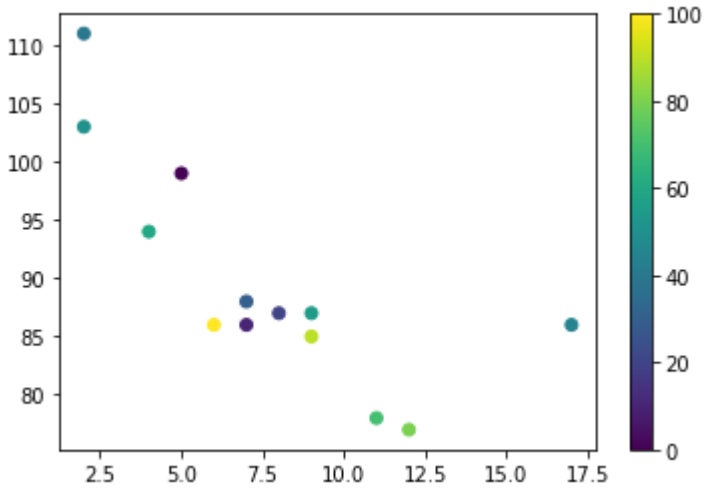
In addition you have to create an array with values (from 0 to 100), one value for each point in the scatter plot:

In [ ]:
```
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()
```

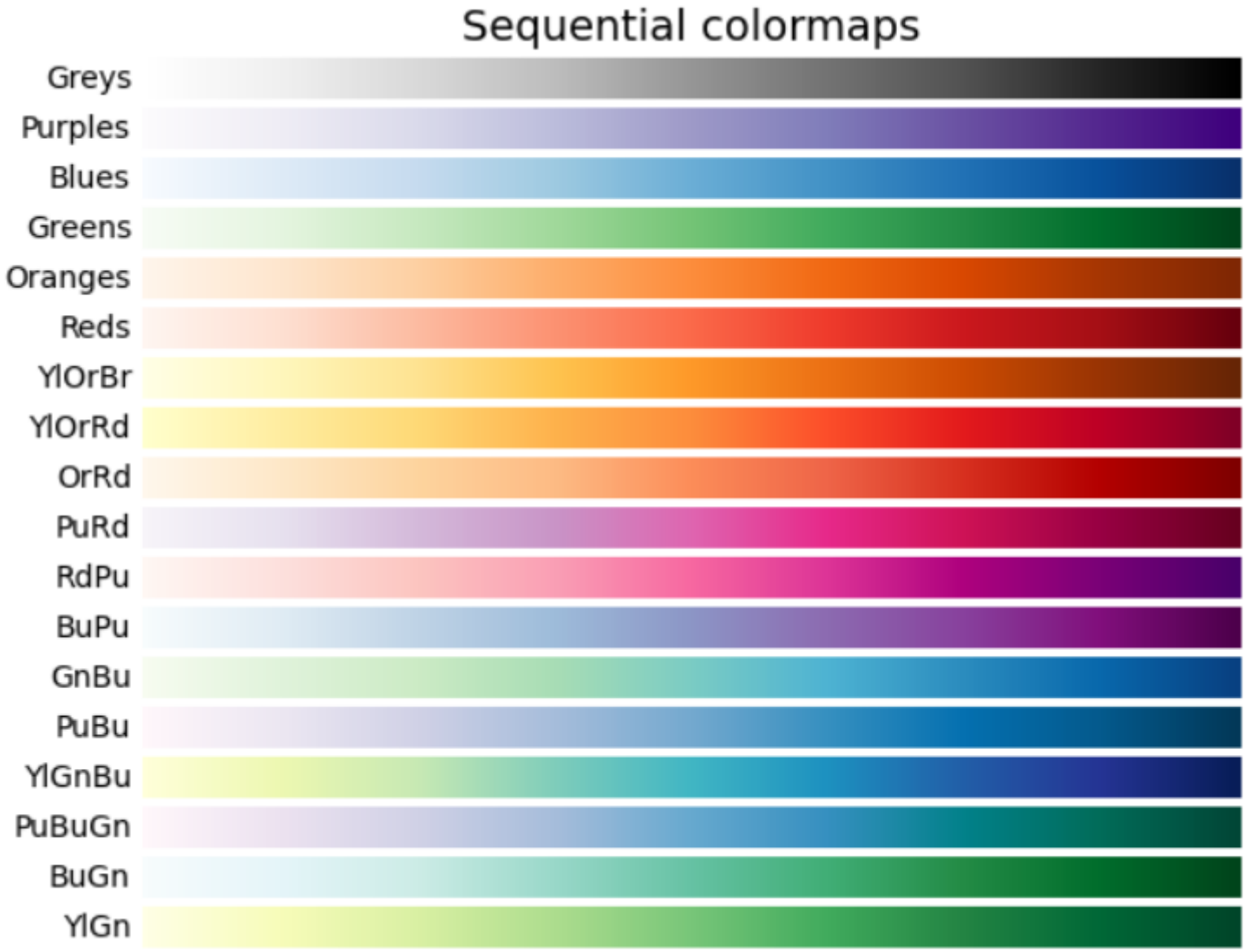


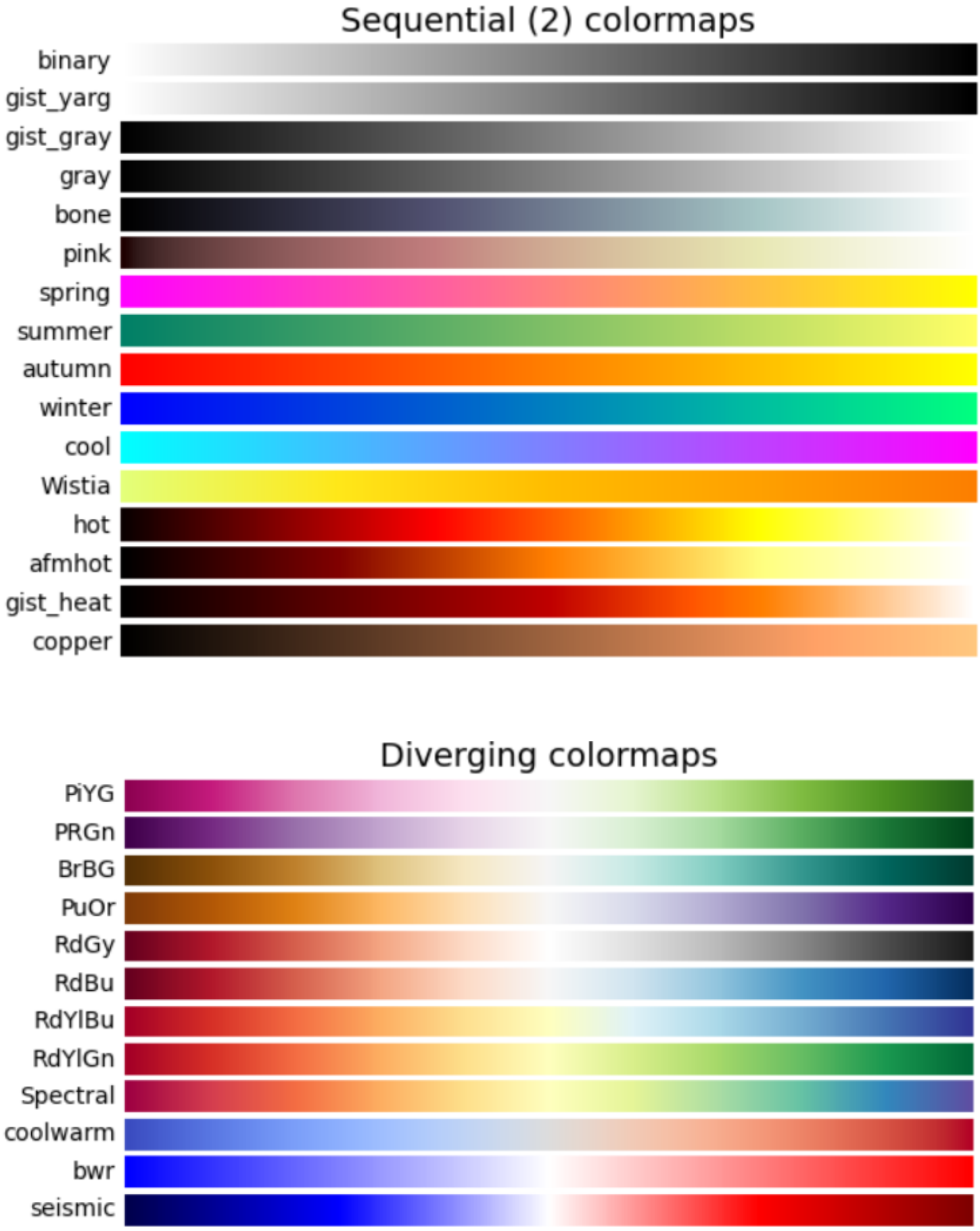
You can include the colormap in the drawing by including the plt.colorbar() statement:

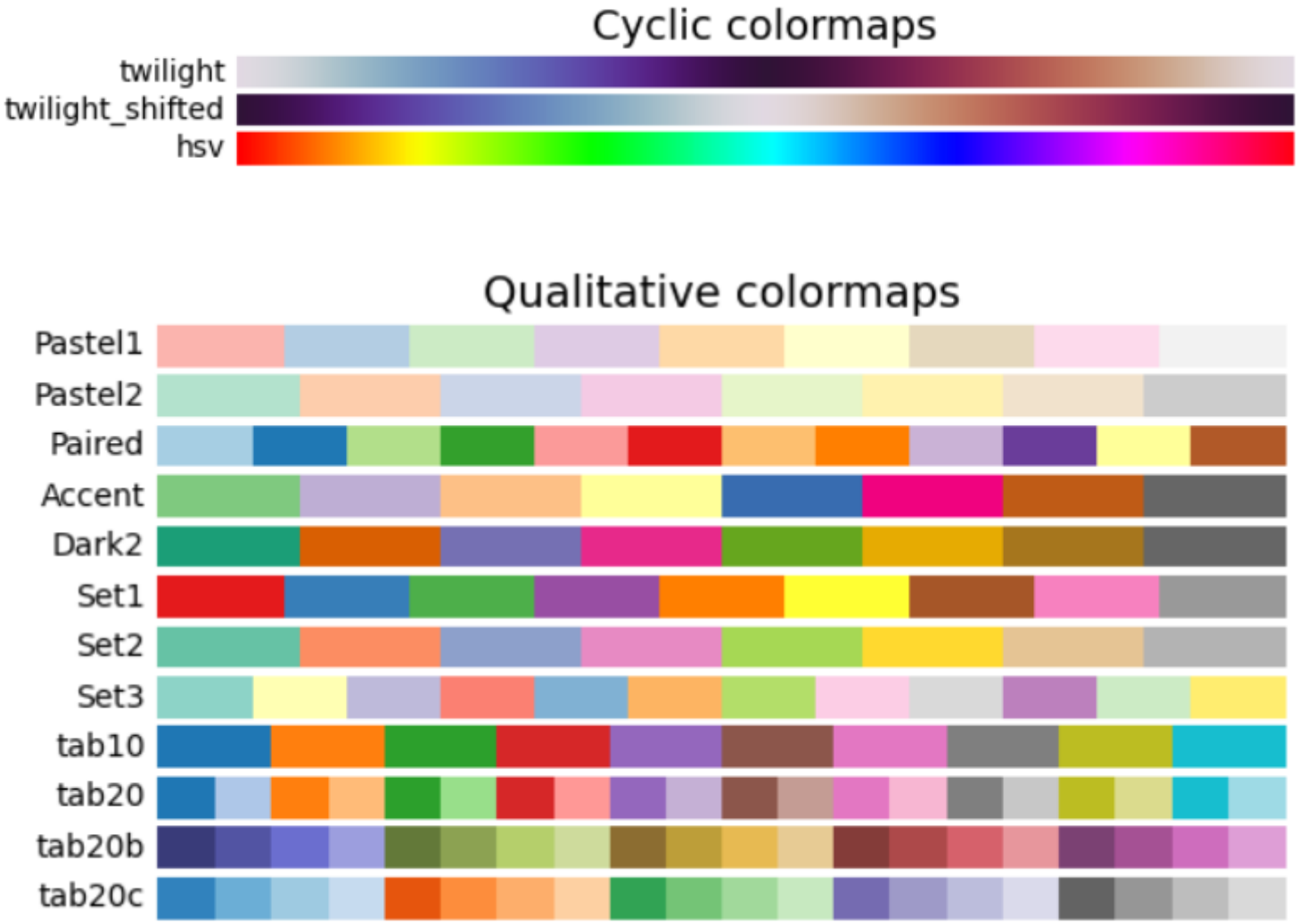
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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()
```

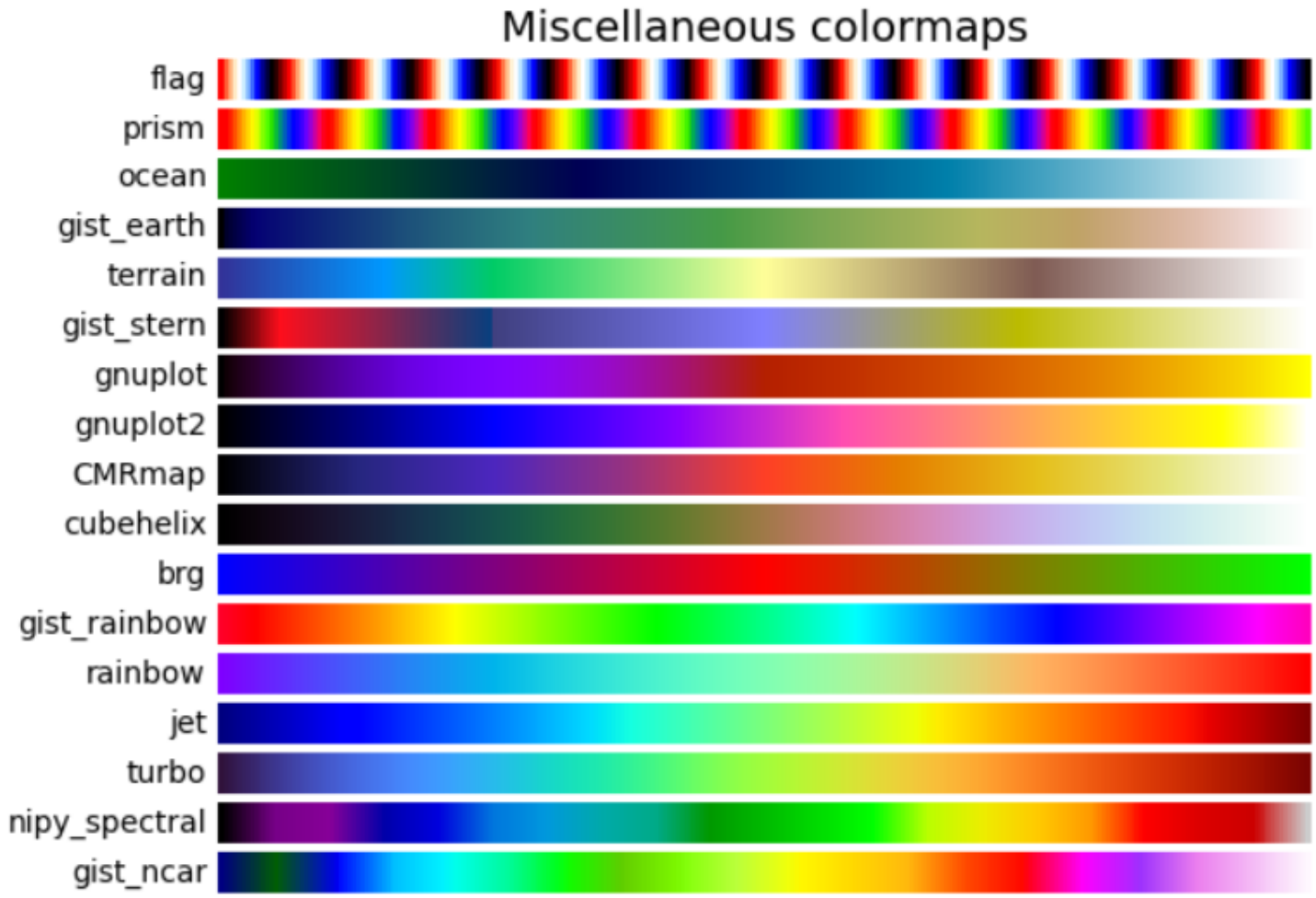


Other Colormaps Available


Sequential colormaps
Greys
Purples
Blues
Greens
Oranges
Reds
YlOrBr
YlOrRd
OrRd
PuRd
RdPu
BuPu
GnBu
PuBu
YlGnBu
PuBuGn
BuGn
YlGn


Sequential (2) colormaps
binary
gist_yarg
gist_gray
gray
bone
pink
spring
summer
autumn
winter
cool
Wistia
hot
afmhot
gist_heat
copper
Diverging colormaps
PiYG
PRGn
BrBG
PuOr
RdGy
RdBu
RdYlBu
RdYlGn
Spectral
coolwarm
bwr
seismic


Cyclic colormaps
twilight
twilight_shifted
hsv
Qualitative colormaps
Pastel1
Pastel2
Paired
Accent
Dark2
Set1
Set2
Set3
tab10
tab20
tab20b
tab20c

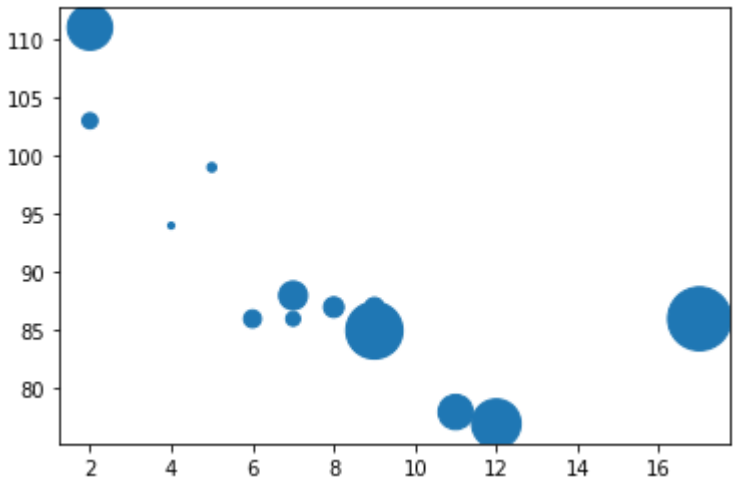


Size

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:

In [ ]:
```
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])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

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

plt.show()
```

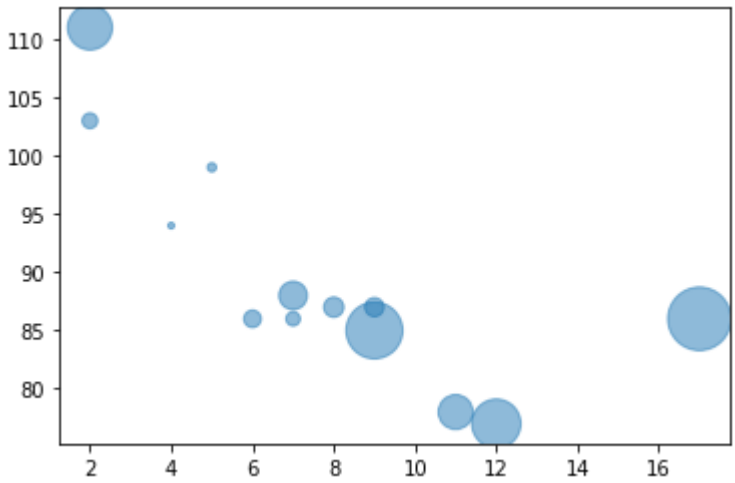


Alpha

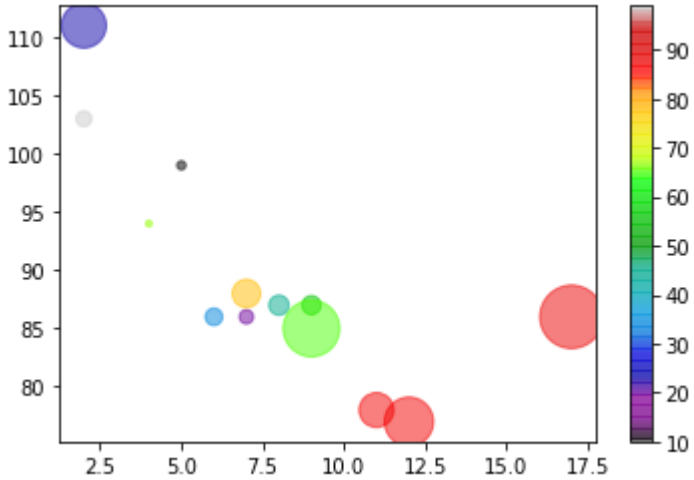
You can adjust the transparency of the dots with the alpha argument.

Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

In [ ]:
```
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])
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()
```



In [ ]:
```
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])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])
colors = [10,20,45,78,23,88,99,56,67,87,87,65,34]

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

plt.colorbar()

plt.show()
```

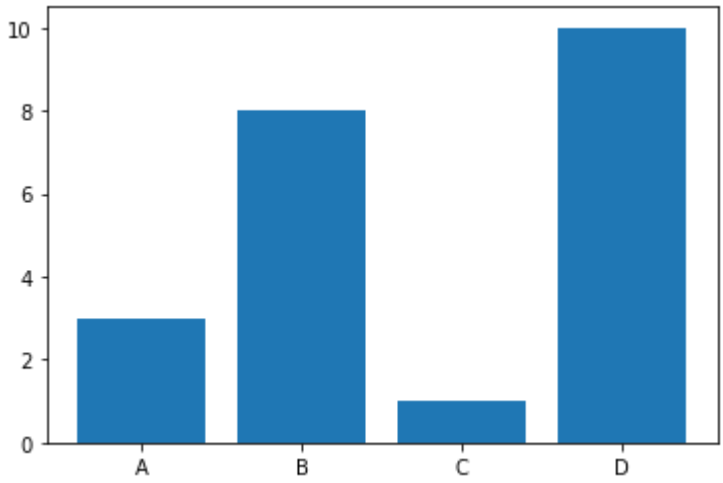


Matplotlib Bars

Creating Bars

With Pyplot, you can use the bar() function to draw bar graphs:

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

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

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

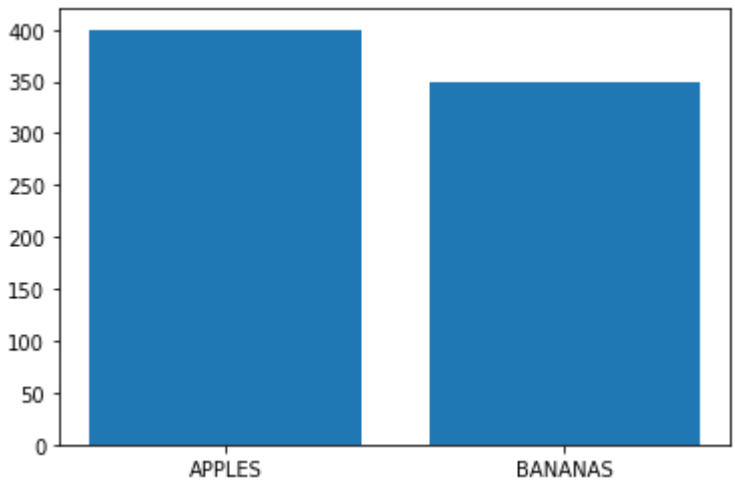


The bar() function takes arguments that describes the layout of the bars.

The categories and their values represented by the first and second argument as arrays.

In [ ]:
```
x = ["APPLES", "BANANAS"]
y = [400, 350]
plt.bar(x, y)
```

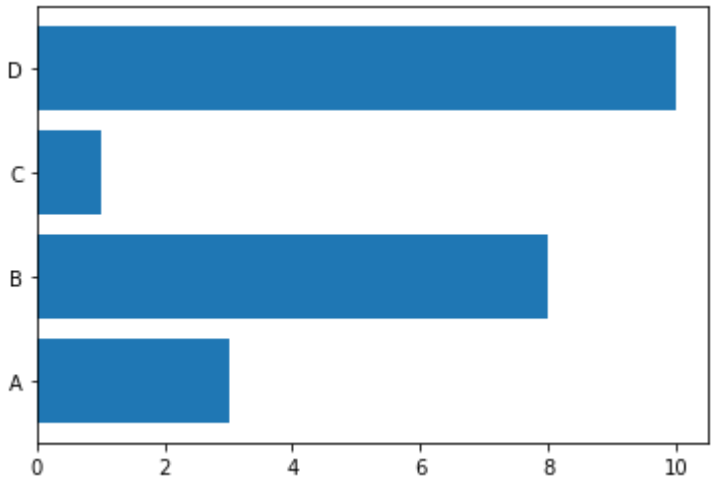
Out[ ]:
```
<BarContainer object of 2 artists>
```



Horizontal Bars

If you want the bars to be displayed horizontally instead of vertically, use the barh() function:

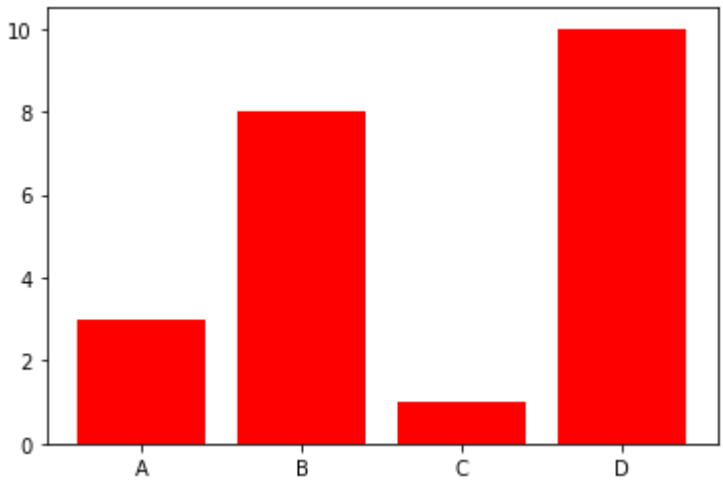
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



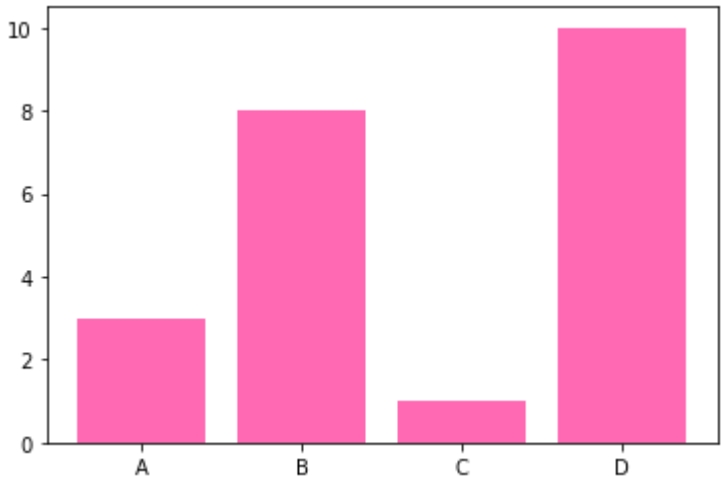
Bar Color

The bar() and barh() take the keyword argument color to set the color of the bars:

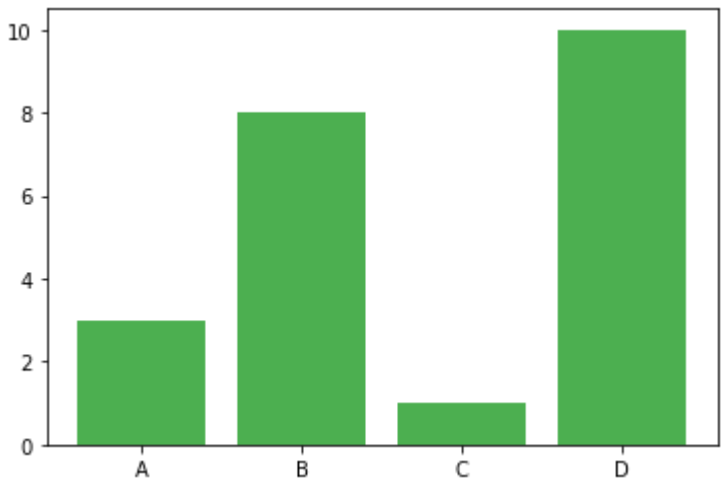
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.bar(x, y, color = "red")
plt.show()
```



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

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

plt.bar(x, y, color = "hotpink")
plt.show()
```



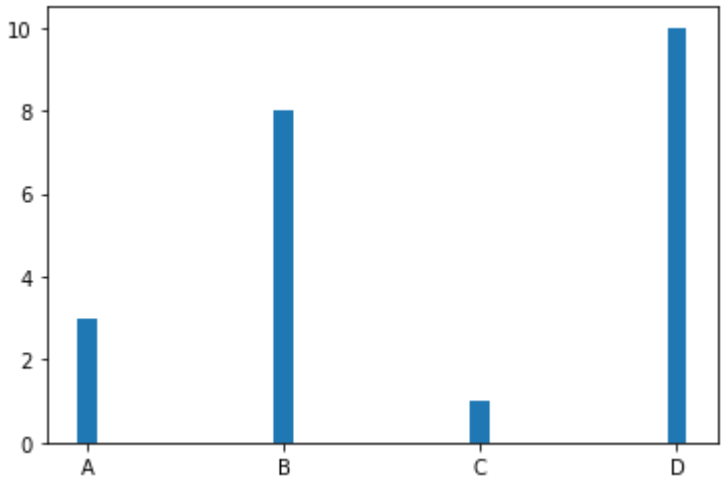
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.bar(x, y, color = "#4CAF50")
plt.show()
```



Bar Width

The bar() takes the keyword argument width to set the width of the bars:

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

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

plt.bar(x, y, width = 0.1)
plt.show()
```



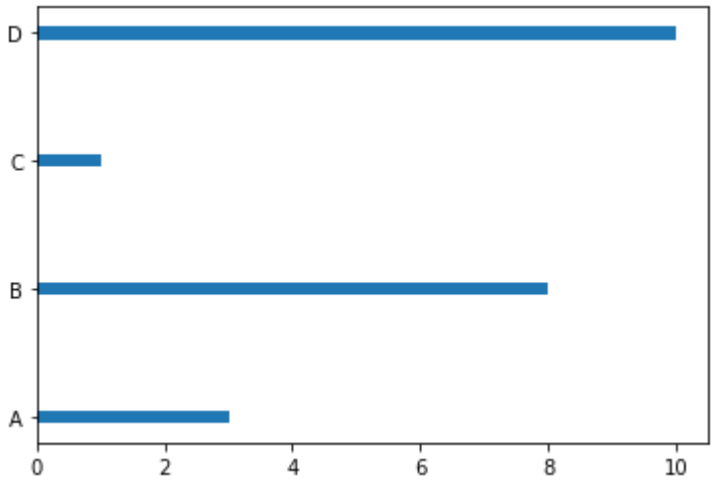
The default width value is 0.8

Note: For horizontal bars, use height instead of width.

Bar Height

The barh() takes the keyword argument height to set the height of the bars:

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

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

plt.barh(x, y, height = 0.1)
plt.show()
```



The default height value is 0.8

Matplotlib Histogram

A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable. It is a kind of bar graph.

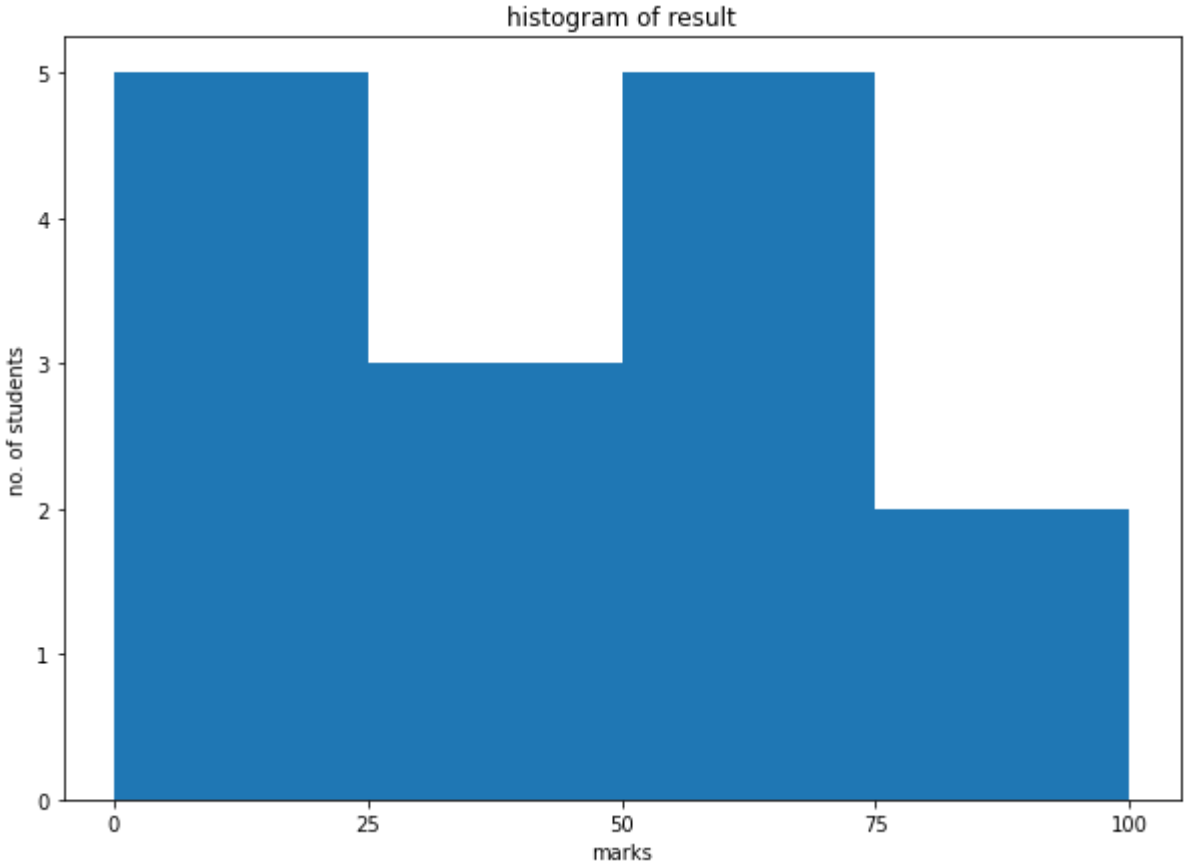
To construct a histogram, follow these steps −

Bin the range of values. Divide the entire range of values into a series of intervals. Count how many values fall into each interval. The bins are usually specified as consecutive, non-overlapping intervals of a variable.

The matplotlib.pyplot.hist() function plots a histogram. It computes and draws the histogram of x.

Following example plots a histogram of marks obtained by students in a class. Four bins, 0-25, 26-50, 51-75, and 76-100 are defined. The Histogram shows number of students falling in this range.

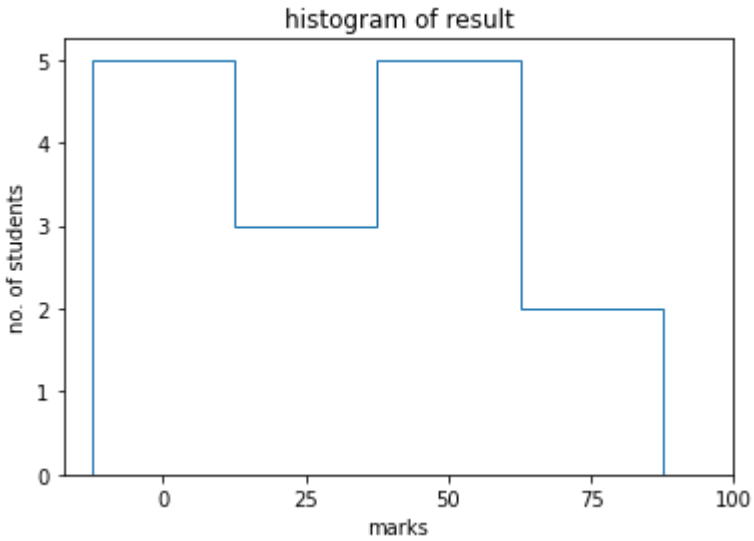
In [4]:
```
from matplotlib import pyplot as plt
import numpy as np
fig, ax = plt.subplots(figsize =(10, 7))
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
ax.hist(a, bins = [0,25,50,75,100])
ax.set_title("histogram of result")
ax.set_xticks([0,25,50,75,100])
ax.set_xlabel('marks')
ax.set_ylabel('no. of students')
plt.show()
```



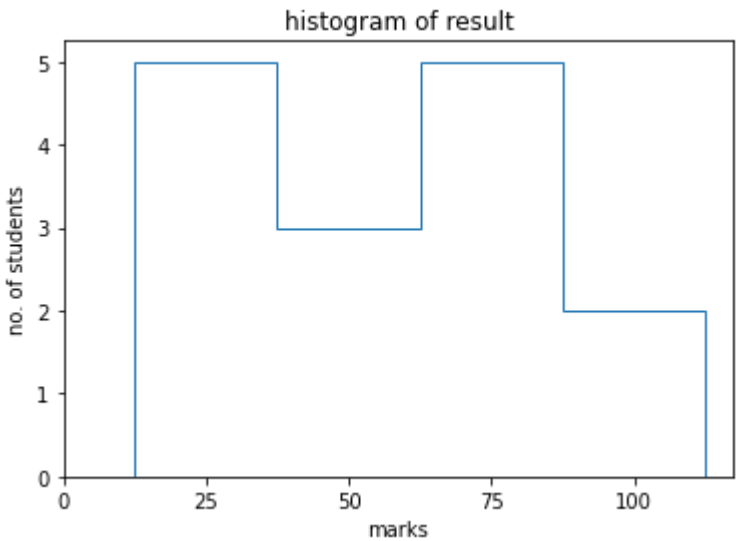
Histogram type and alignment

default histtype is 'bar' and align is 'mid'

In [5]:
```
from matplotlib import pyplot as plt
import numpy as np
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
plt.hist(a, bins = [0,25,50,75,100], histtype = "step", align = 'left')
plt.title("histogram of result")
plt.xticks([0,25,50,75,100])
plt.xlabel('marks')
plt.ylabel('no. of students')
plt.show()
```



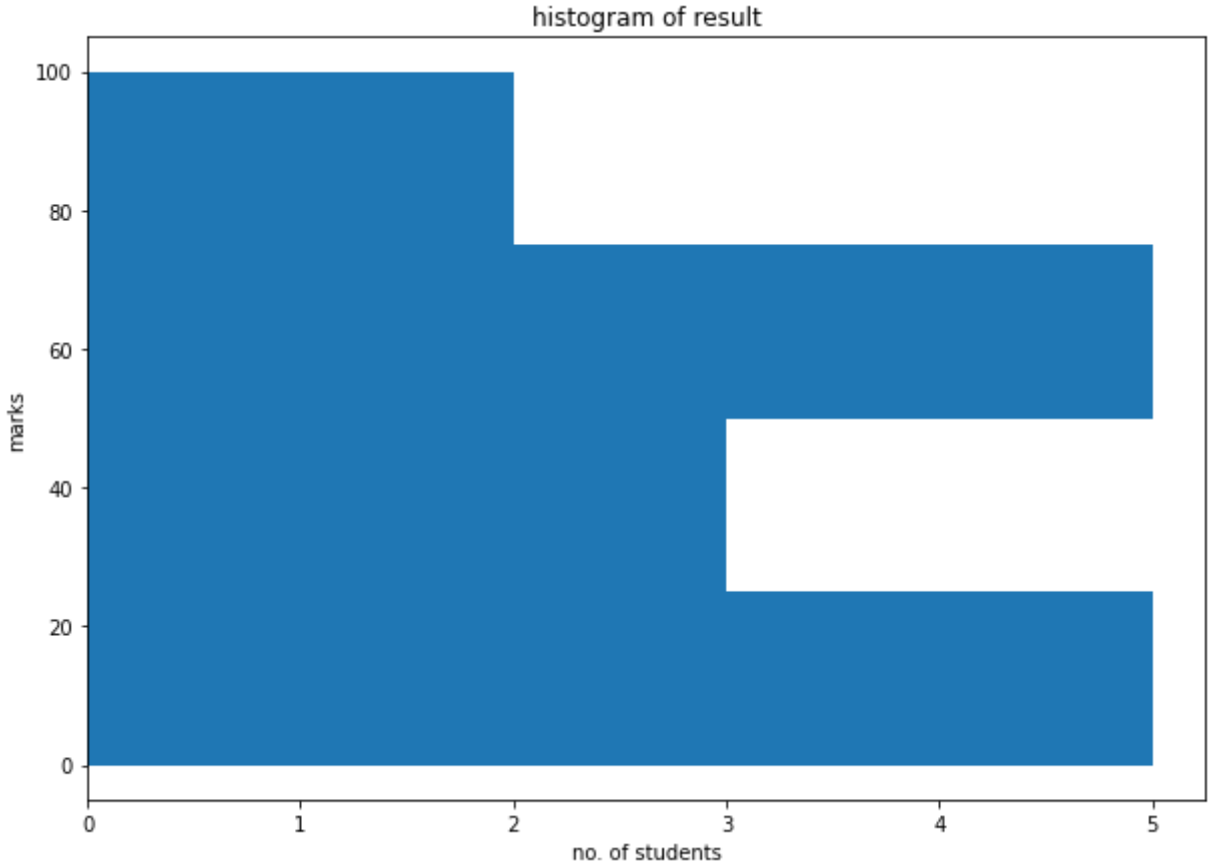
In [6]:
```
from matplotlib import pyplot as plt
import numpy as np
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
plt.hist(a, bins = [0,25,50,75,100], histtype = "step", align = 'right')
plt.title("histogram of result")
plt.xticks([0,25,50,75,100])
plt.xlabel('marks')
plt.ylabel('no. of students')
plt.show()
```



Histogram Orientation

orientation{'vertical', 'horizontal'}, default: 'vertical'

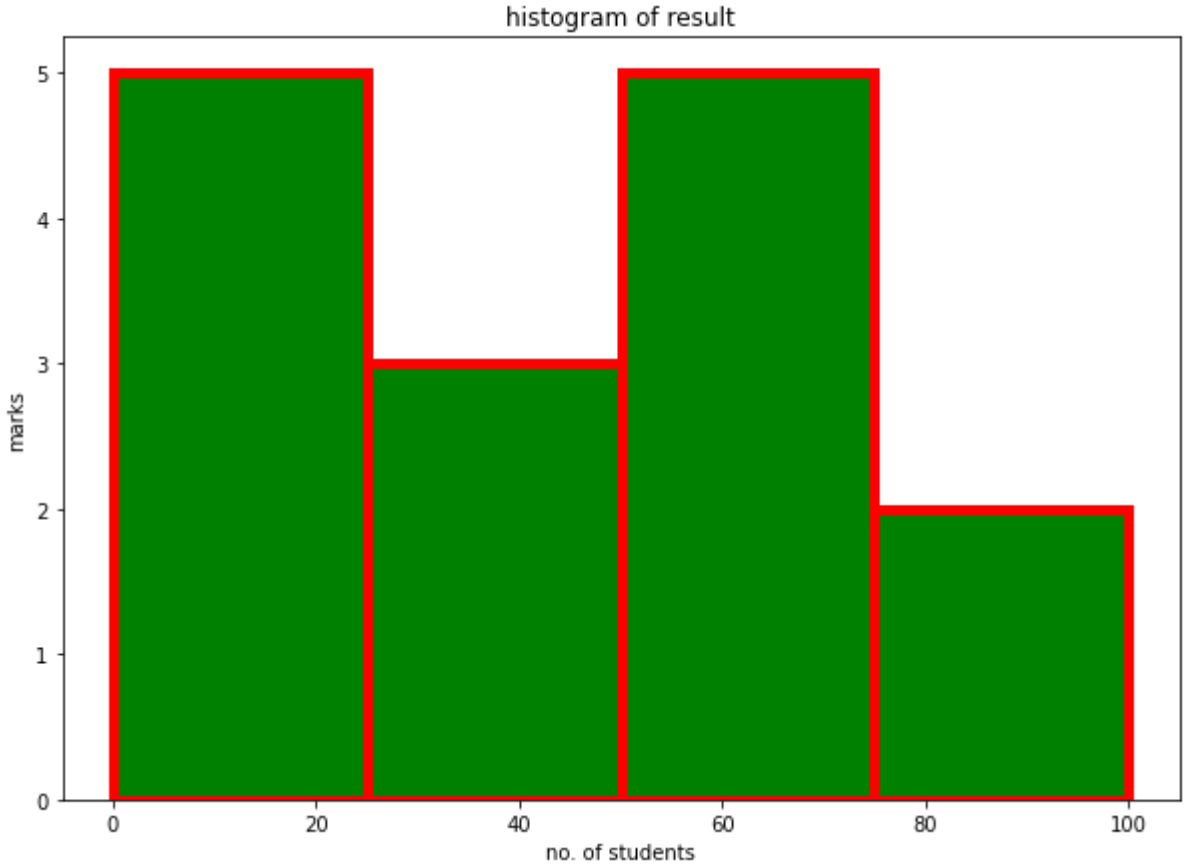
In [ ]:
```
from matplotlib import pyplot as plt
import numpy as np
fig, ax = plt.subplots(figsize =(10, 7))
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
ax.hist(a, bins = [0,25,50,75,100], orientation = 'horizontal')
ax.set_title("histogram of result")
#ax.set_xticks([0,25,50,75,100])
ax.set_xlabel('no. of students')
ax.set_ylabel('marks')
plt.show()
```



Histogram Colour Customization

lw is linewidth and ec is edge colour

In [7]:
```
from matplotlib import pyplot as plt
import numpy as np
fig, ax = plt.subplots(figsize =(10, 7))
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
ax.hist(a, bins = [0,25,50,75,100], color = 'green', lw = '5', ec = 'red')
ax.set_title("histogram of result")
#ax.set_xticks([0,25,50,75,100])
ax.set_xlabel('no. of students')
ax.set_ylabel('marks')
plt.show()
```



Matplotlib Pie Charts

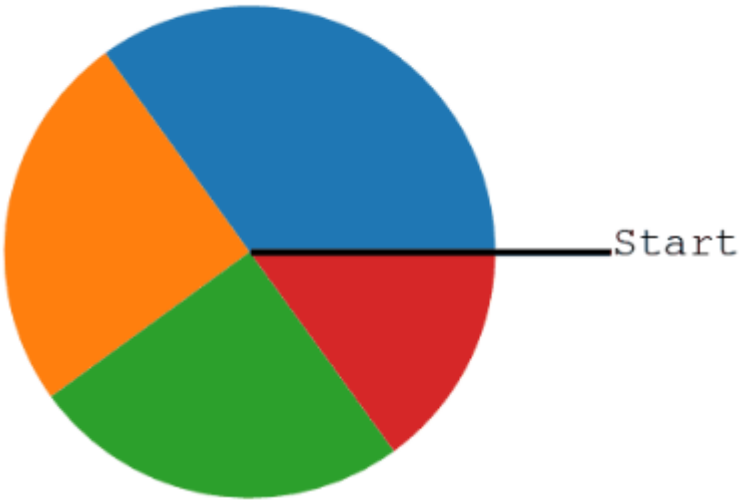
Creating Pie Charts

With Pyplot, you can use the pie() function to draw pie charts:

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

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

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

As you can see the pie chart draws one piece (called a wedge) for each value in the array (in this case [35, 25, 25, 15]).

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



Note: The size of each wedge is determined by comparing the value with all the other values, by using this formula:

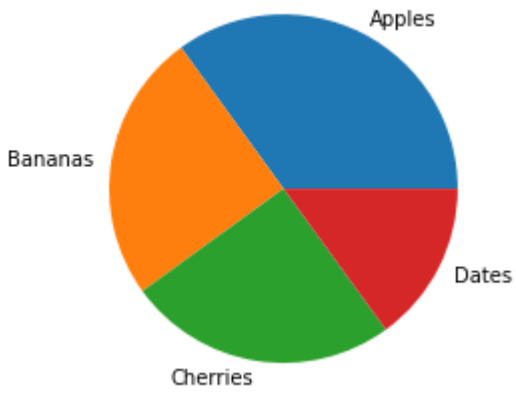
The value divided by the sum of all values: x/sum(x)

Labels

Add labels to the pie chart with the label parameter.

The label parameter must be an array with one label for each wedge:

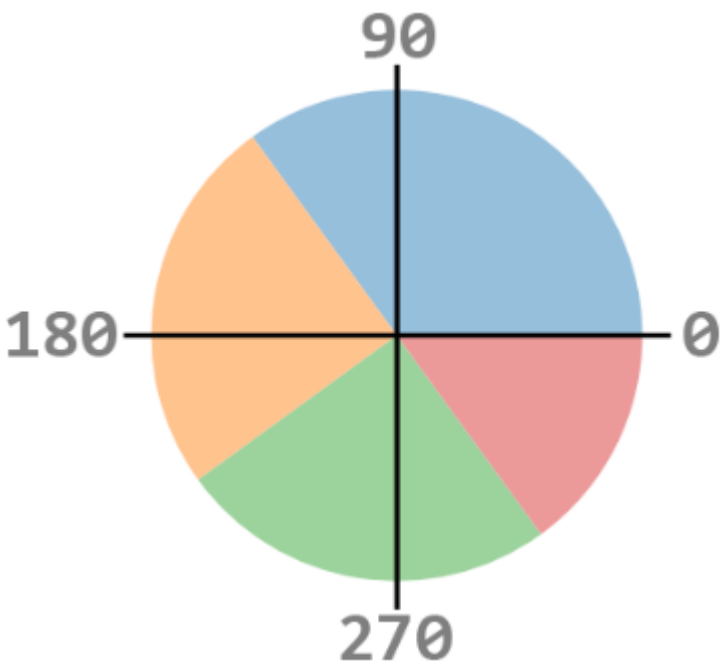
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



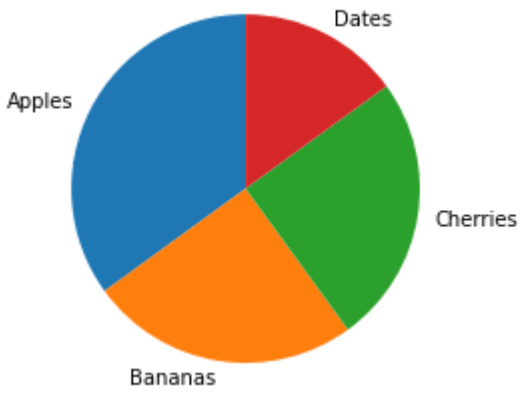
Start Angle

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:



In [ ]:
```
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, startangle = 90)
plt.show()
```



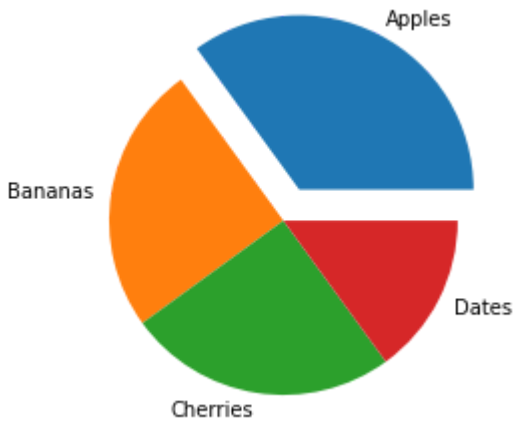
Explode

Maybe you want one of the wedges to stand out? The explode parameter allows you to do that.

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:

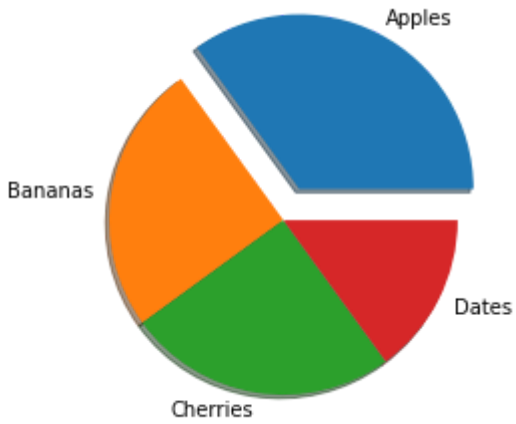
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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)
plt.show()
```



Shadow

Add a shadow to the pie chart by setting the shadows parameter to True:

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

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

You can set the color of each wedge with the colors parameter.

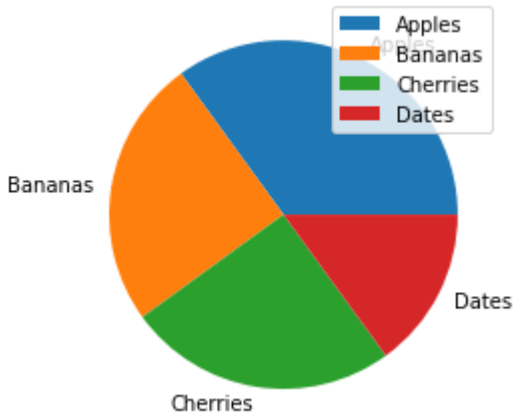
The colors parameter, if specified, must be an array with one value for each wedge:

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

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:

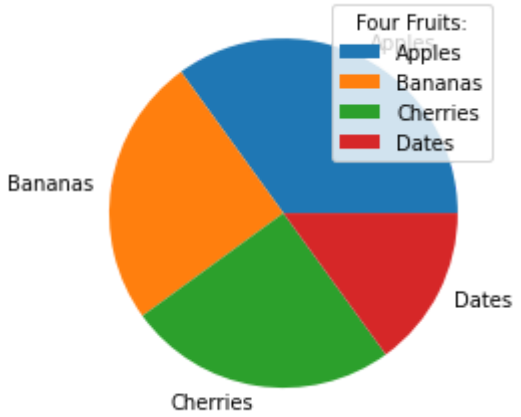
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



Legend With Header

To add a header to the legend, add the title parameter to the legend function.

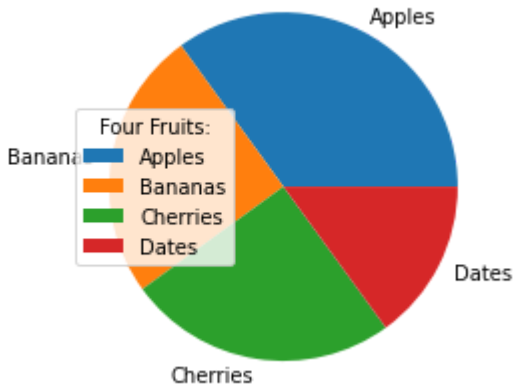
In [11]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



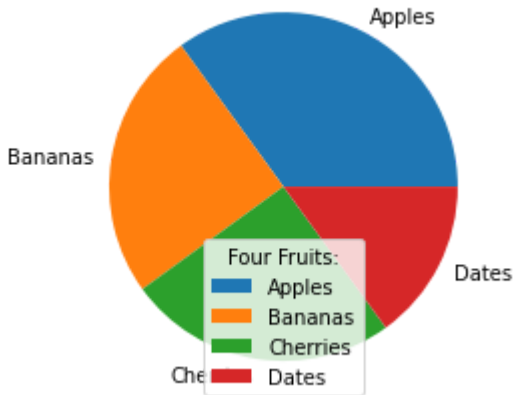
Legend Location

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

In [12]:
```
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 = "Four Fruits:", loc = 6)
plt.show()
```

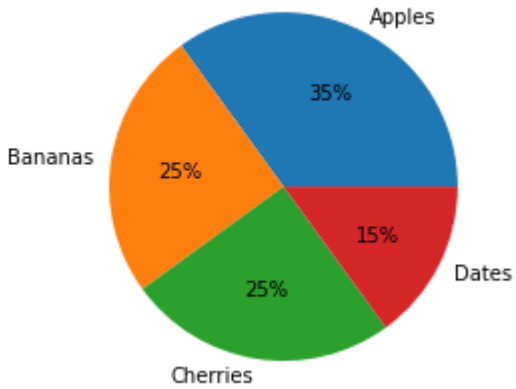


In [10]:
```
import matplotlib.pyplot as plt
import numpy as np

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

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



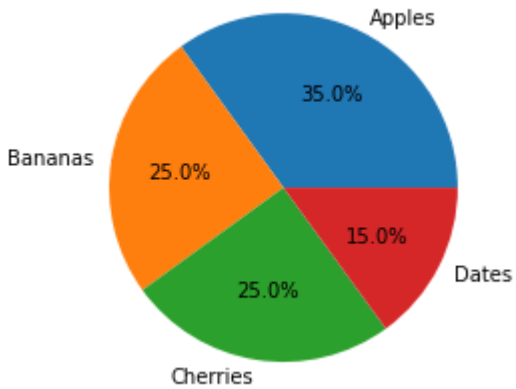
Autopct can be used to show percentages of pie slices

In [14]:
```
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, autopct = '%1.f%%')
plt.show()
```



For accuracy after decimal point, use %.1f%% for one

In [15]:
```
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, autopct = '%.1f%%')
plt.show()
```



For accuracy after decimal point, use %.2f%% for two

In [16]:
```
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, autopct = '%.2f%%')
plt.show()
```