

PRESENTATION ON MACHINE LEARNING - LINEAR REGRESSION

PRESENTED BY

DR. RONAK PANCHAL (PhD)

IT Project Manager & Senior Data Scientist

@Cognizant, Pune

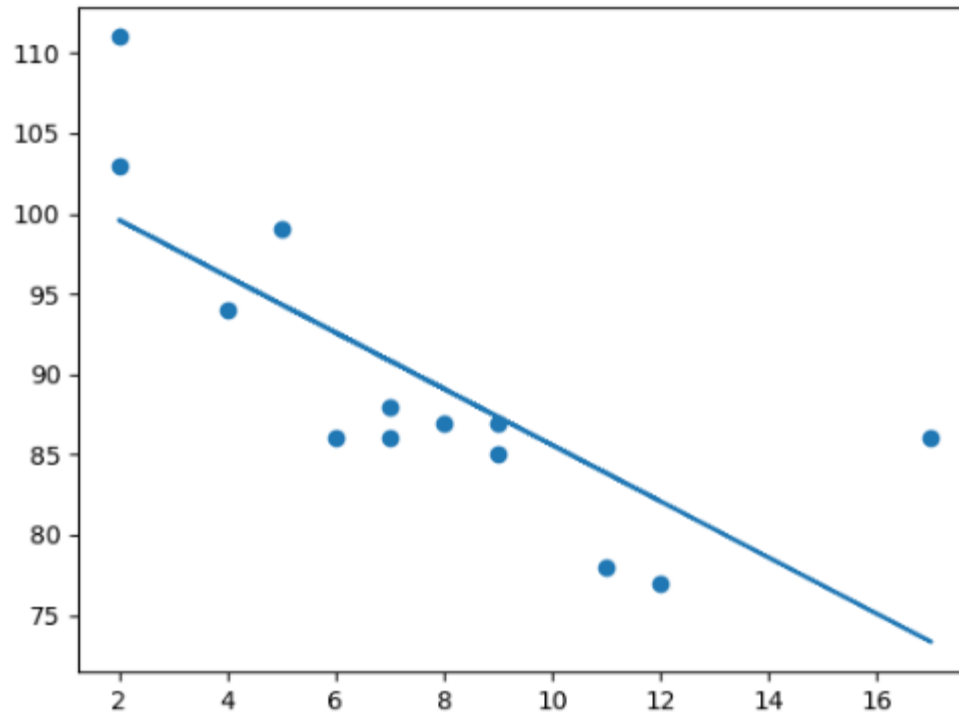
REGRESSION

- The term regression is used when you try to find the relationship between variables.
- In Machine Learning, and in statistical modeling, that relationship is used to predict the outcome of future events.

LINEAR REGRESSION

- Linear regression uses the relationship between the data-points to draw a straight line through all them.

THIS LINE CAN BE USED TO PREDICT FUTURE VALUES.



NOTE

- In Machine Learning, predicting the future is very important.

HOW DOES IT WORK?

- Python has methods for finding a relationship between data-points and to draw a line of linear regression. We will show you how to use these methods instead of going through the mathematic formula.
- In the example below, the x-axis represents age, and the y-axis represents speed. We have registered the age and speed of 13 cars as they were passing a tollbooth. Let us see if the data we collected could be used in a linear regression:

EXAMPLE

- Start by drawing a scatter plot:

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

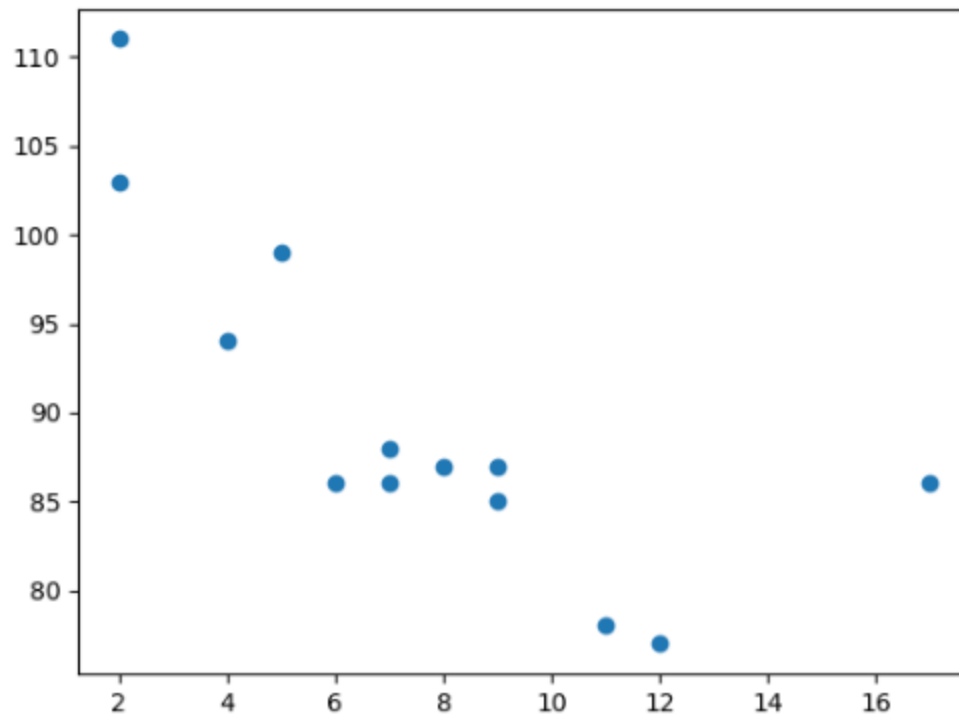
```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
```

```
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

```
plt.scatter(x, y)
```

```
plt.show()
```

RESULT



EXAMPLE

- Import scipy and draw the line of Linear Regression:

```
import matplotlib.pyplot as plt
from scipy import stats
```

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
```

```
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

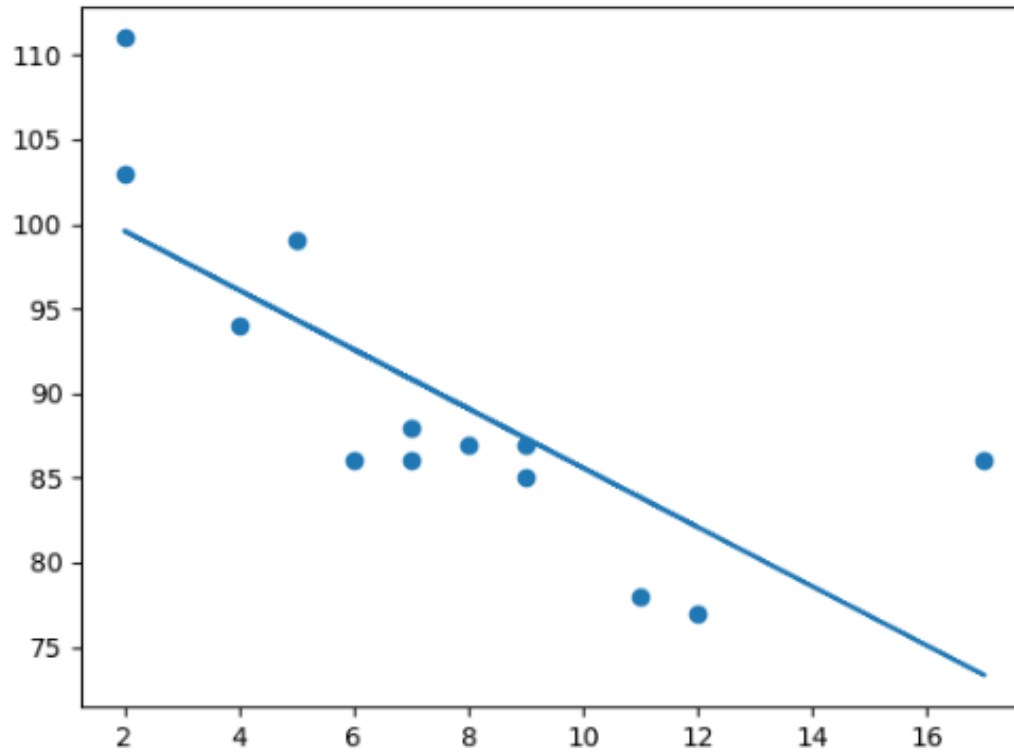
```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

```
def myfunc(x):
    return slope * x + intercept
```

```
mymodel = list(map(myfunc, x))
```

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

RESULT



EXAMPLE EXPLAINED

- Import the modules you need.
- You can learn about the Matplotlib module in our Matplotlib Tutorial.
- You can learn about the SciPy module in our SciPy Tutorial.
- `import matplotlib.pyplot as plt`
`from scipy import stats`

CREATE THE ARRAYS THAT REPRESENT THE VALUES OF THE X AND Y AXIS:

- `x = [5,7,8,7,2,17,2,9,4,11,12,9,6]`
`y = [99,86,87,88,111,86,103,87,94,78,77,85,86]`
- Execute a method that returns some important key values of Linear Regression:
- `slope, intercept, r, p, std_err = stats.linregress(x, y)`

CREATE A FUNCTION THAT USES THE SLOPE AND INTERCEPT VALUES TO RETURN A NEW VALUE. THIS NEW VALUE REPRESENTS WHERE ON THE Y-AXIS THE CORRESPONDING X VALUE WILL BE PLACED:

- `def myfunc(x):`
 `return slope * x + intercept`
- Run each value of the x array through the function. This will result in a new array with new values for the y-axis:
- `mymodel = list(map(myfunc, x))`

DRAW THE ORIGINAL SCATTER PLOT:

```
plt.scatter(x, y)
```

- Draw the line of linear regression:

```
plt.plot(x, mymodel)
```

- Display the diagram:

```
plt.show()
```

R FOR RELATIONSHIP

- It is important to know how the relationship between the values of the x-axis and the values of the y-axis is, if there are no relationship the linear regression can not be used to predict anything.
- This relationship - the coefficient of correlation - is called r .
- The r value ranges from -1 to 1, where 0 means no relationship, and 1 (and -1) means 100% related.
- Python and the Scipy module will compute this value for you, all you have to do is feed it with the x and y values.

EXAMPLE

- How well does my data fit in a linear regression?

```
from scipy import stats
```

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
```

```
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

```
slope, intercept, r, p, std_err = stats.linregress(x,  
y)
```

```
print(r)
```

OutPut

```
-0.7585915243761551
```


NOTE:

- The result -0.76 shows that there is a relationship, not perfect, but it indicates that we could use linear regression in future predictions.

PREDICT FUTURE VALUES

- Now we can use the information we have gathered to predict future values.
- Example: Let us try to predict the speed of a 10 years old car.
- To do so, we need the same myfunc() function from the example above:

```
defmyfunc(x):  
    return slope * x + intercept
```

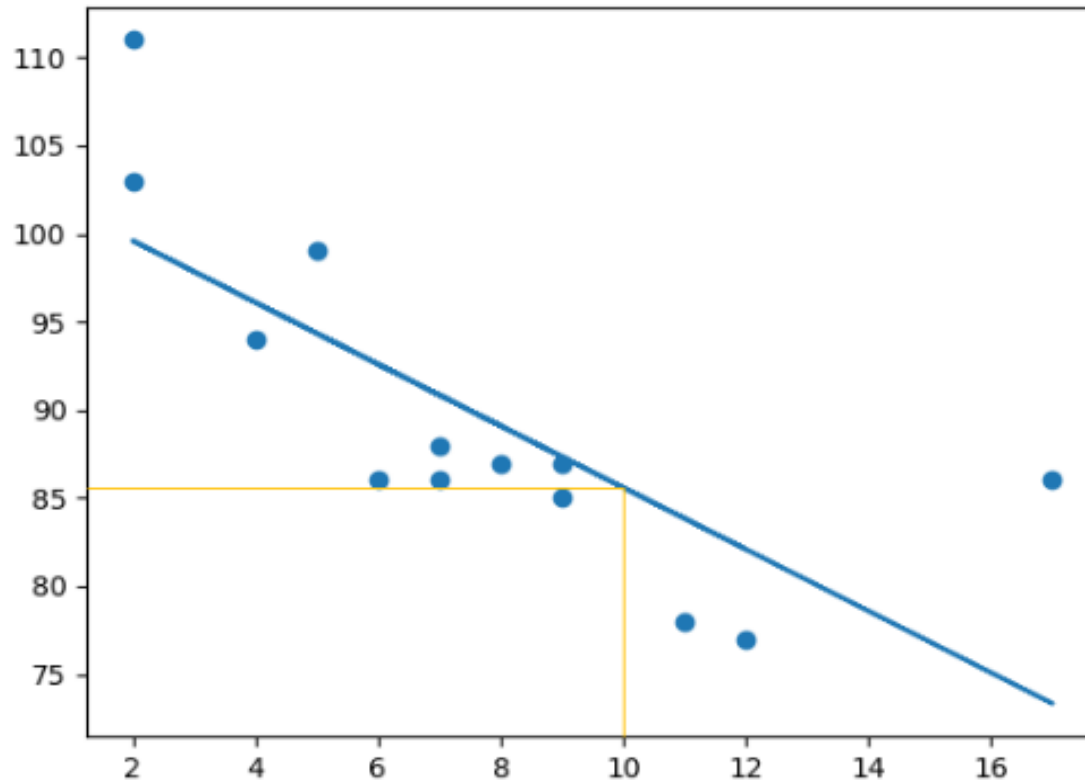
EXAMPLE

- Predict the speed of a 10 years old car:
- from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x,
y)
def myfunc(x):
 return slope * x + intercept
speed = myfunc(10)
print(speed)

OutPut

```
C:\Users\My Name>python demo_ml_r_squared2.py  
85.59308314937454
```

THE EXAMPLE PREDICTED A SPEED AT 85.6,
WHICH WE ALSO COULD READ FROM THE
DIAGRAM:



BAD FIT?

- Let us create an example where linear regression would not be the best method to predict future values.

EXAMPLE

linear_regression&Matplotlib - Colab x ML_LinearRegression.ipynb - Colab x (41) WhatsApp x Meet - zjn-urpx-ygo x

colab.research.google.com/drive/1qmQ-q9GWbAvRNYOmy3bQEwv_tPIA5gBV#scrollTo=GdC6XvrxDz60

Apps New folder Microsoft Teams OpenLink Virtuoso... Google Translate Grocery Tutorial · pr... Be.Cognizant OneCognizant VehicleInventoryOp...

ML_LinearRegression.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

RAM Disk Editing


Files

- ..
- sample_data
- home_prices.csv
- predication_home_area.csv

```
#!pip install matplotlib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from scipy import stats
```

```
[5] x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
    y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

```
[6] plt.scatter(x, y)
    plt.show()
```



Disk 69.92 GB available

0s completed at 10:48 AM

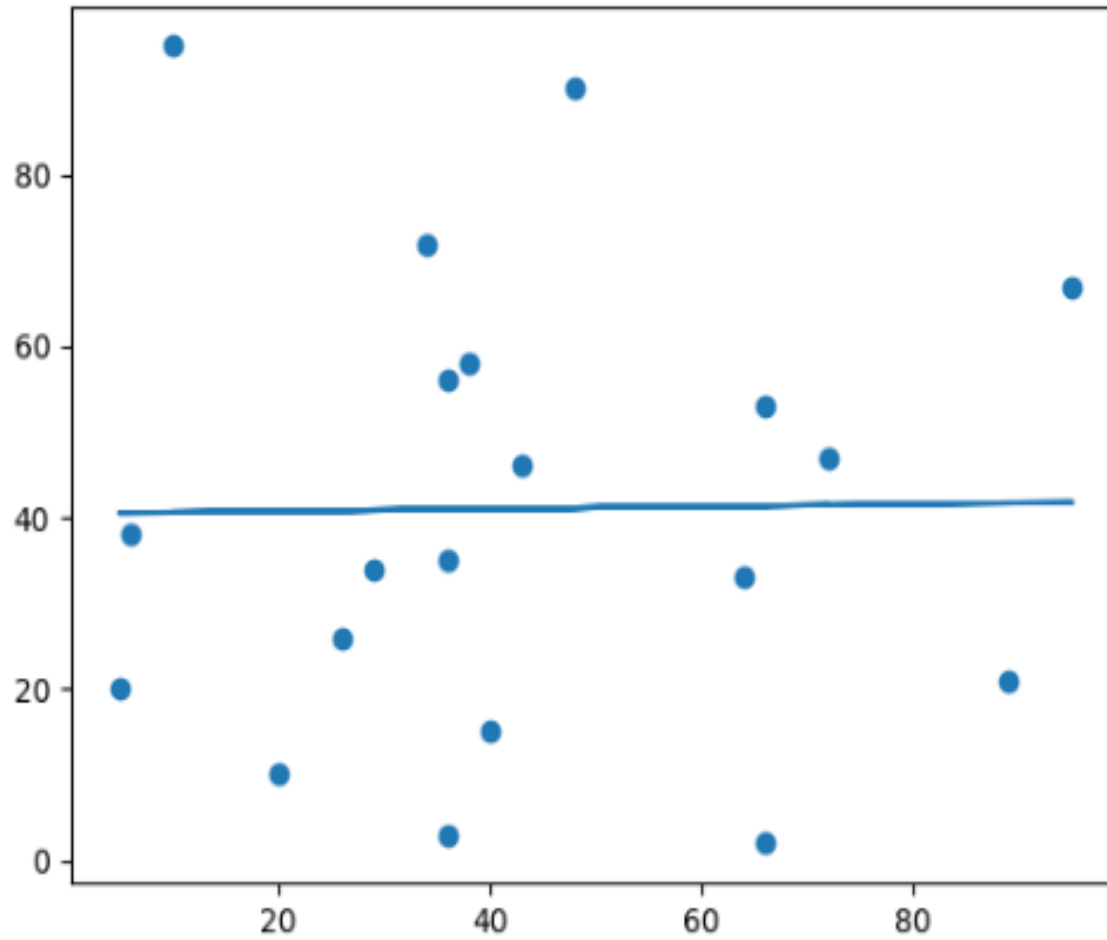
linear_regression....pptx

Show all

33°C Partly sunny

10:51 AM 5/10/2022

RESULT



AND THE R FOR RELATIONSHIP?

Example

- You should get a very low r value.

```
import numpy
from scipy import stats
x= [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,
5,36,66,72,40]
y= [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,
20,56,2,47,15]
slope, intercept, r, p, std_err =
stats.linregress(x, y)
print(r)
```


OUTPUT

```
0.01331814154297491
```

The result: 0.013 indicates a very bad relationship, and tells us that this data set is not suitable for linear regression.

PHYTHON MATPLOTLIB

**PRESENTED BY:
RONAK PANCHAL**

MATPLOTLIB TUTORIAL

- 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.

MATPLOTLIB GETTING STARTED

- Installation of Matplotlib
- If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy.
- Install it using this command:

```
C:\Users\Your Name>pip install matplotlib
```

➤ IF THIS COMMAND FAILS, THEN USE A PYTHON DISTRIBUTION THAT ALREADY HAS MATPLOTLIB INSTALLED, LIKE ANACONDA, SPYDER ETC.

IMPORT MATPLOTLIB

- Once Matplotlib is installed, import it in your applications by adding the **import module** statement:

```
import matplotlib
```

- Now Matplotlib is imported and ready to use:

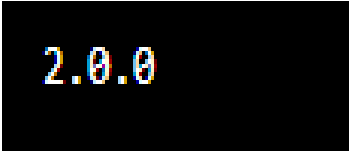
CHECKING MATPLOTLIB VERSION

- The version string is stored under `__version__` attributes.

Example

```
import matplotlib  
print(matplotlib.__version__)
```

OutPut:



2.0.0

MATPLOTLIB PYPLOT

Pyplot

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

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

- Now the Pyplot package can be referred to as plt.

NOTE

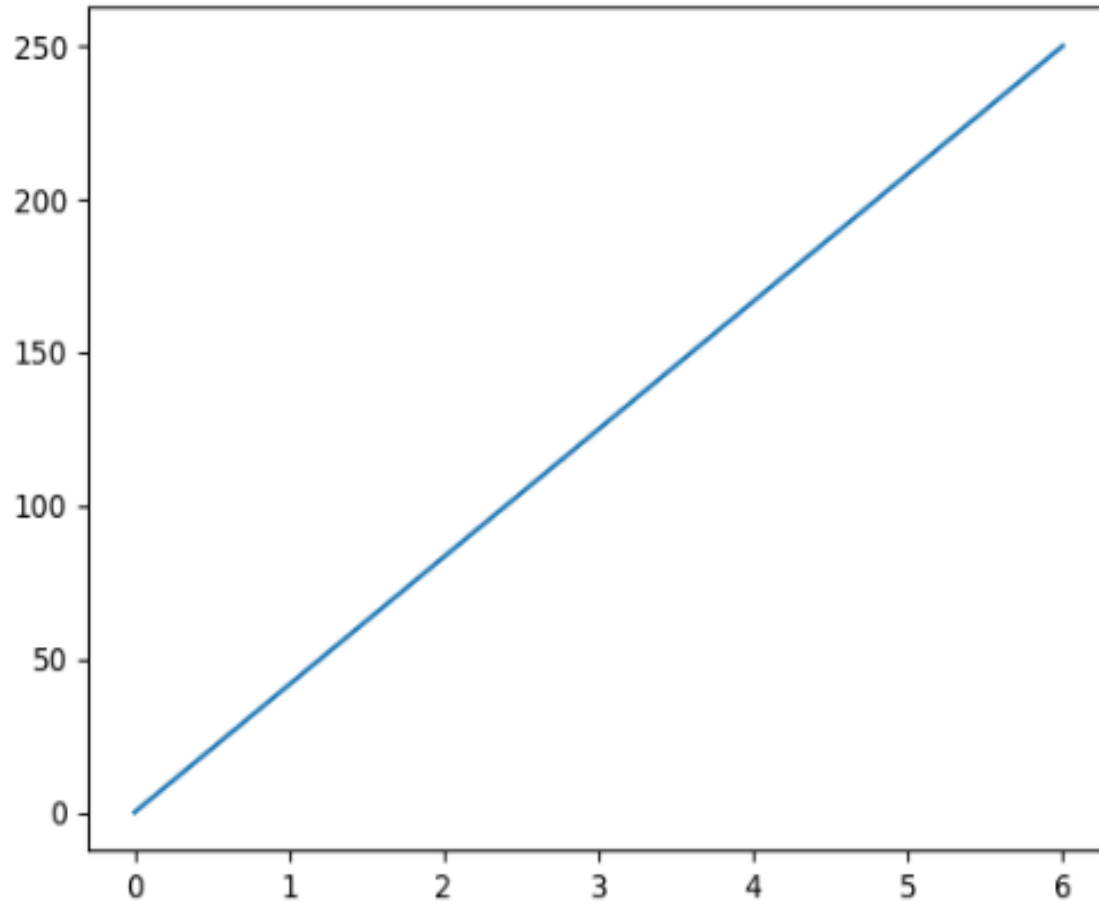
- Two underscore characters are used in_version_.

EXAMPLE

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

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
```

RESULT



MATPLOTLIB PLOTTING

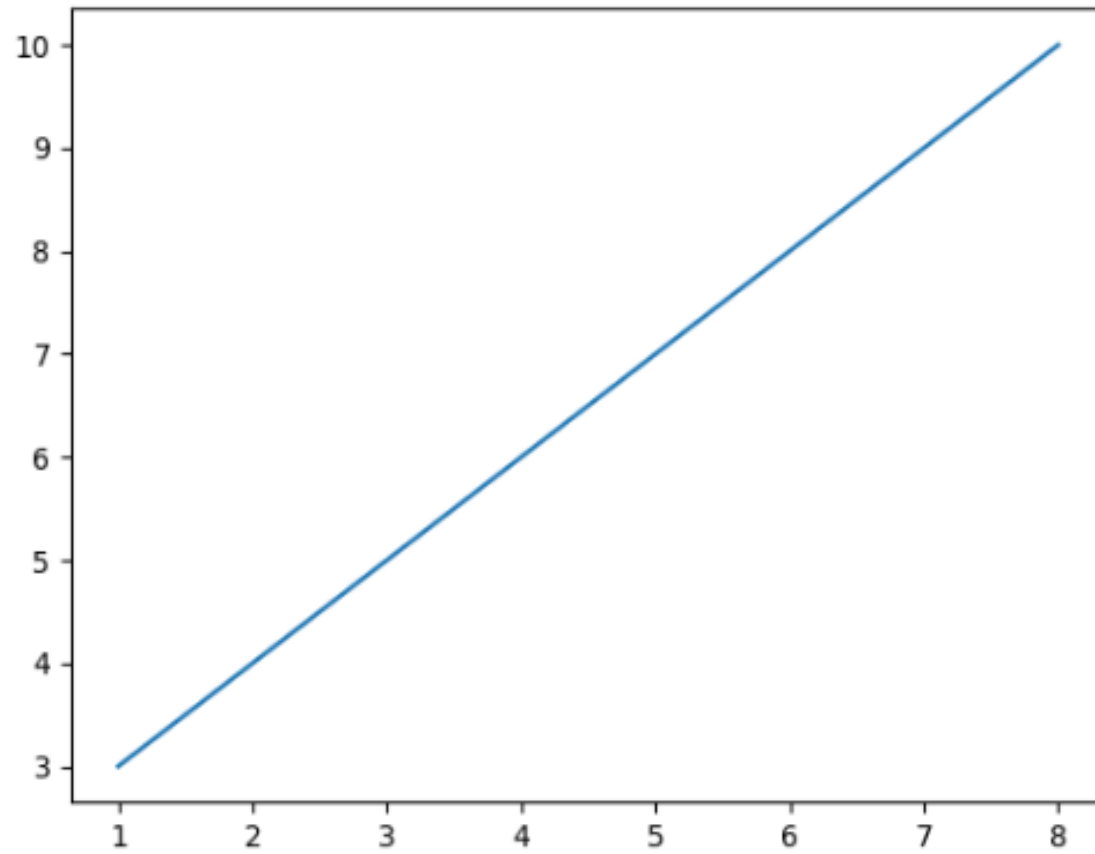
- The `plot()` function is used to draw points (markers) in a diagram.
- By default, the `plot()` function draws a line from point to point.
- The function takes parameters for specifying points in the diagram.
- Parameter 1 is an array containing the points on the **x-axis**.
- Parameter 2 is an array 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.

EXAMPLE

- 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()
```

RESULT:



THE X-AXIS IS THE HORIZONTAL AXIS.
THE Y-AXIS IS THE VERTICAL AXIS.

PLOTTING WITHOUT LINE

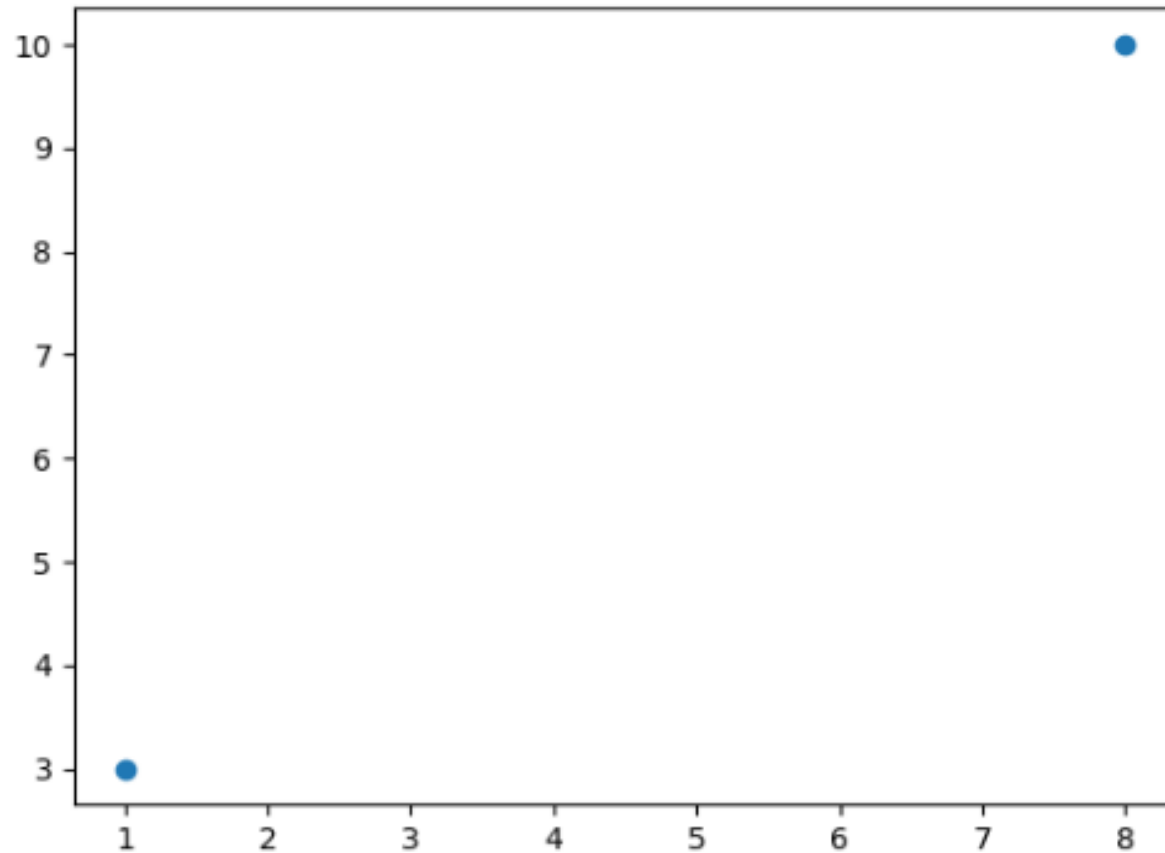
- To plot only the markers, you can use *shortcut string notation* parameter 'o', which means 'rings'.

Example

- Draw two points in the diagram, one at position (1, 3) and one in 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, 'o')
plt.show()
```


RESULT



MULTIPLE POINTS

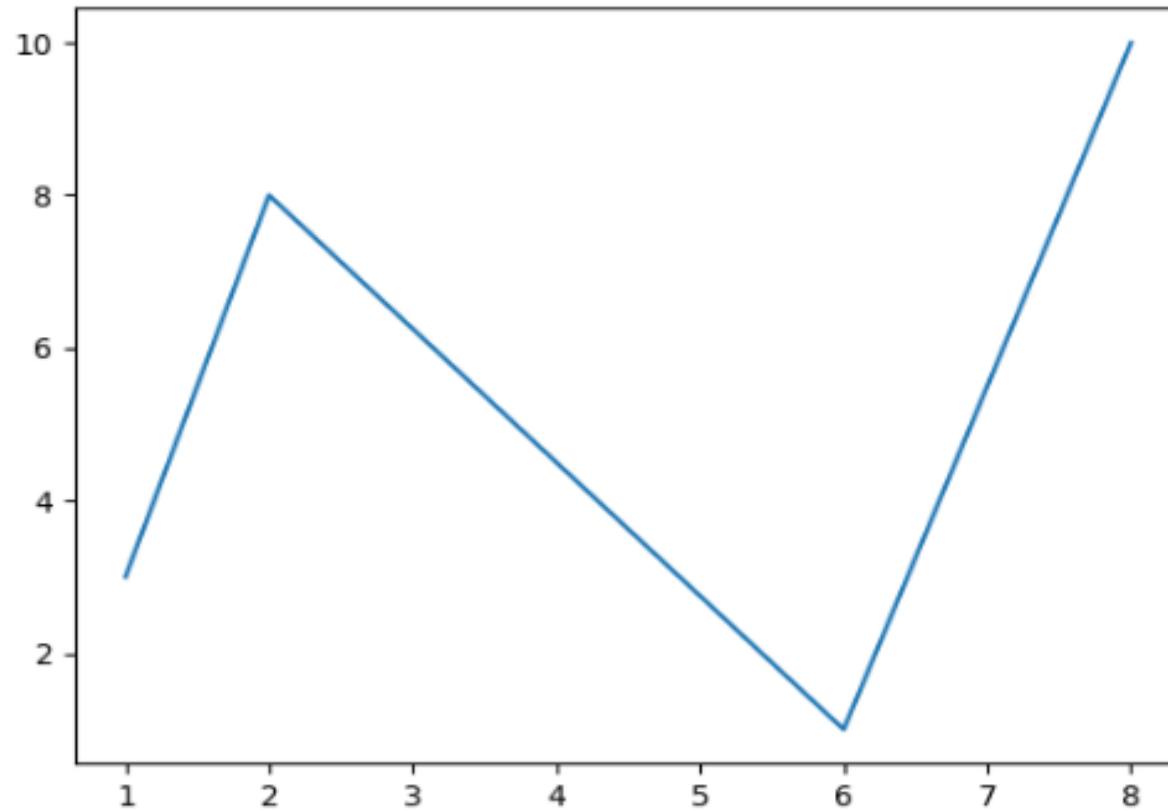
- You can plot as many points as you like, just make sure you have the same number of points in both axis.

EXAMPLE

- Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):

```
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()
```

RESULT



DEFAULT X-POINTS

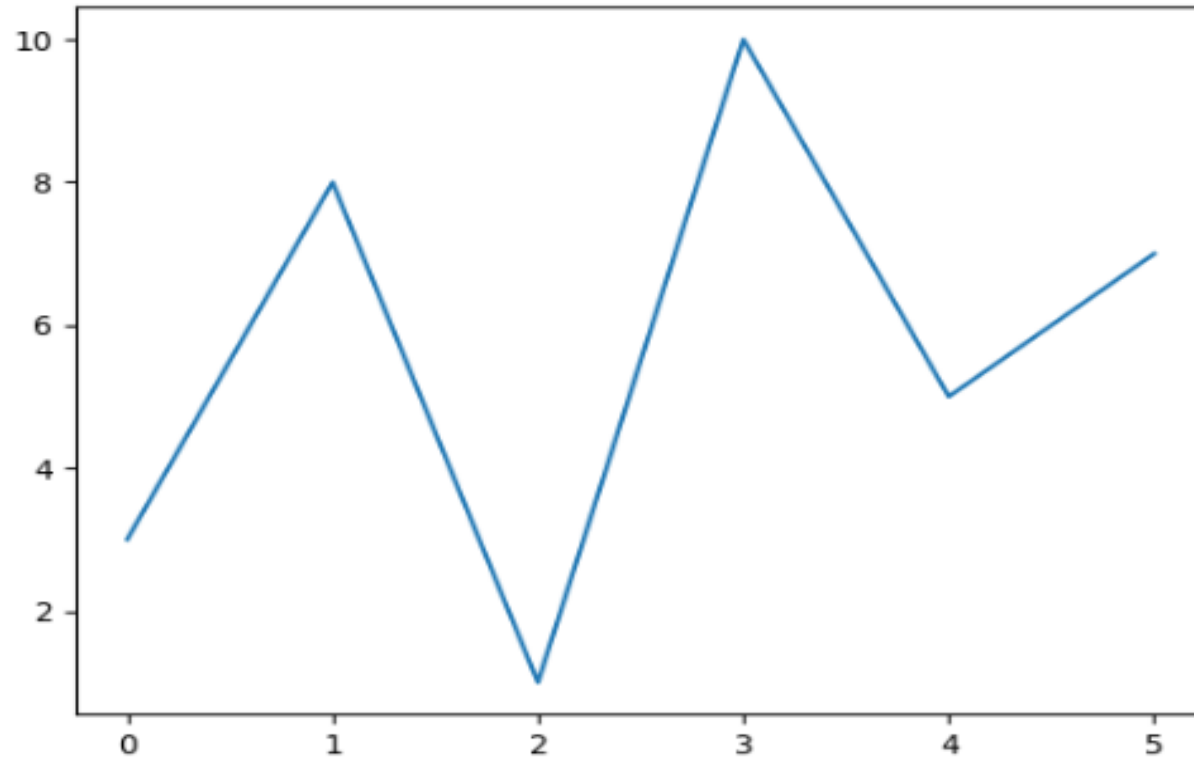
- If we do not specify the points in the x-axis, they will get the default values 0, 1, 2, 3, (etc. depending on the length of the y-points).
- So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

EXAMPLE

- Plotting without x-points:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10, 5, 7])
plt.plot(ypoints)
plt.show()
```

RESULT



MATPLOTLIB MARKERS

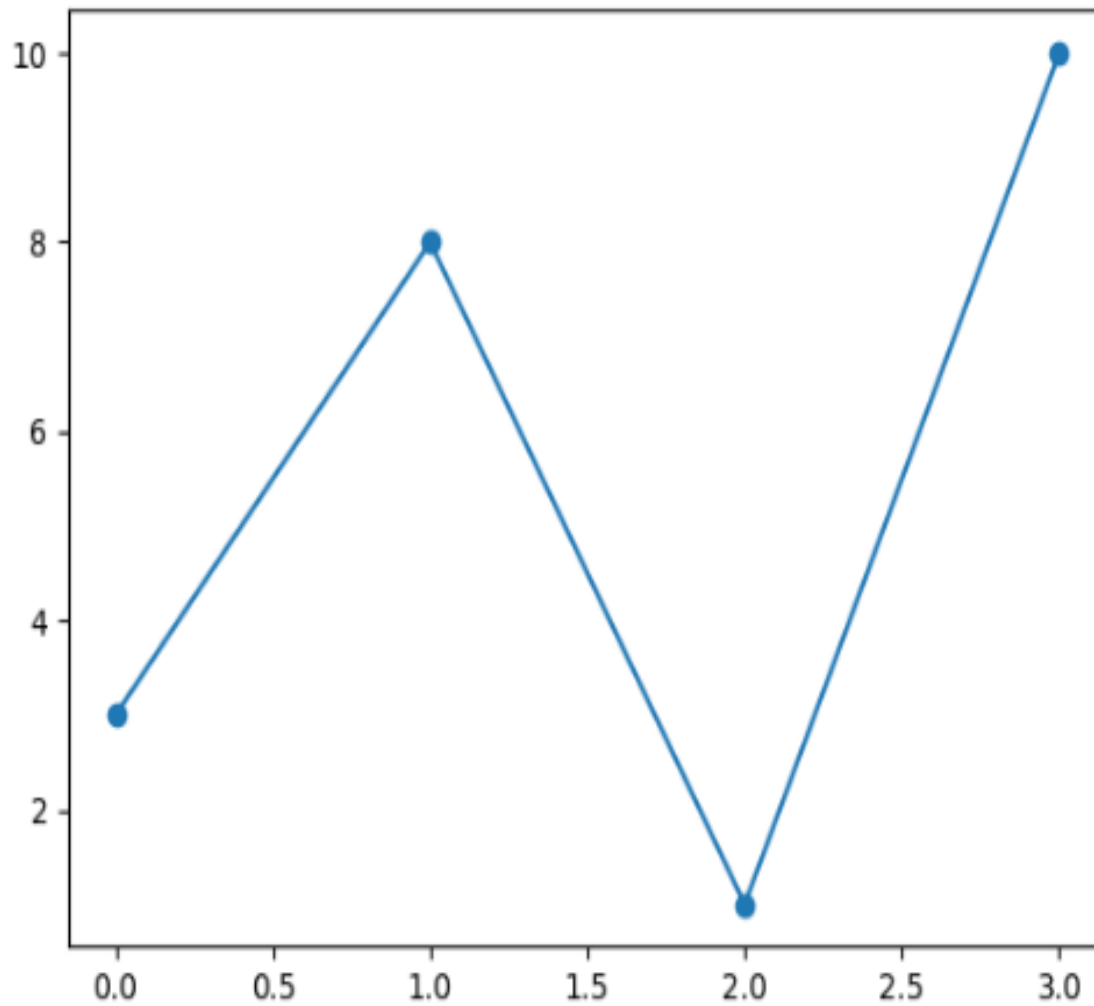
- Markers
- You can use the keyword argument `marker` to emphasize each point with a specified marker:

Example

- Mark each point with a circle:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o')
plt.show()
```


RESULT



EXAMPLE

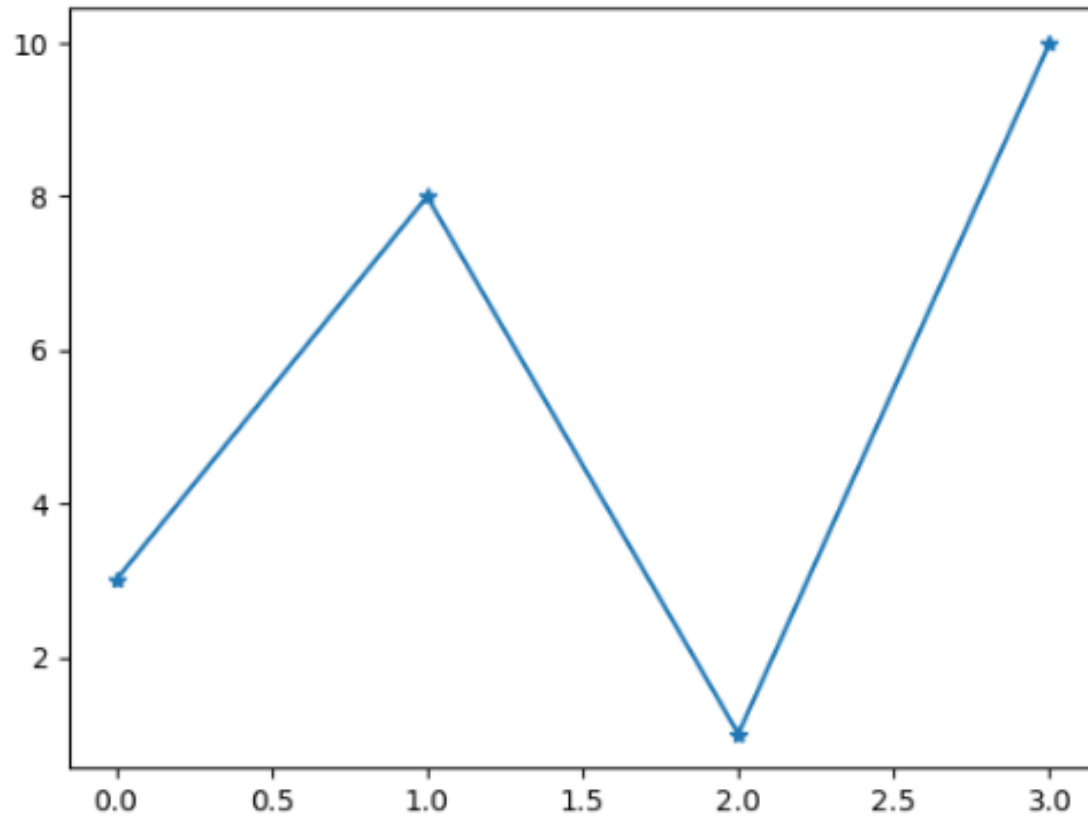
- Mark each point with a star:

...

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

...

RESULT



MARKER REFERENCE

FORMAT STRINGS FMT

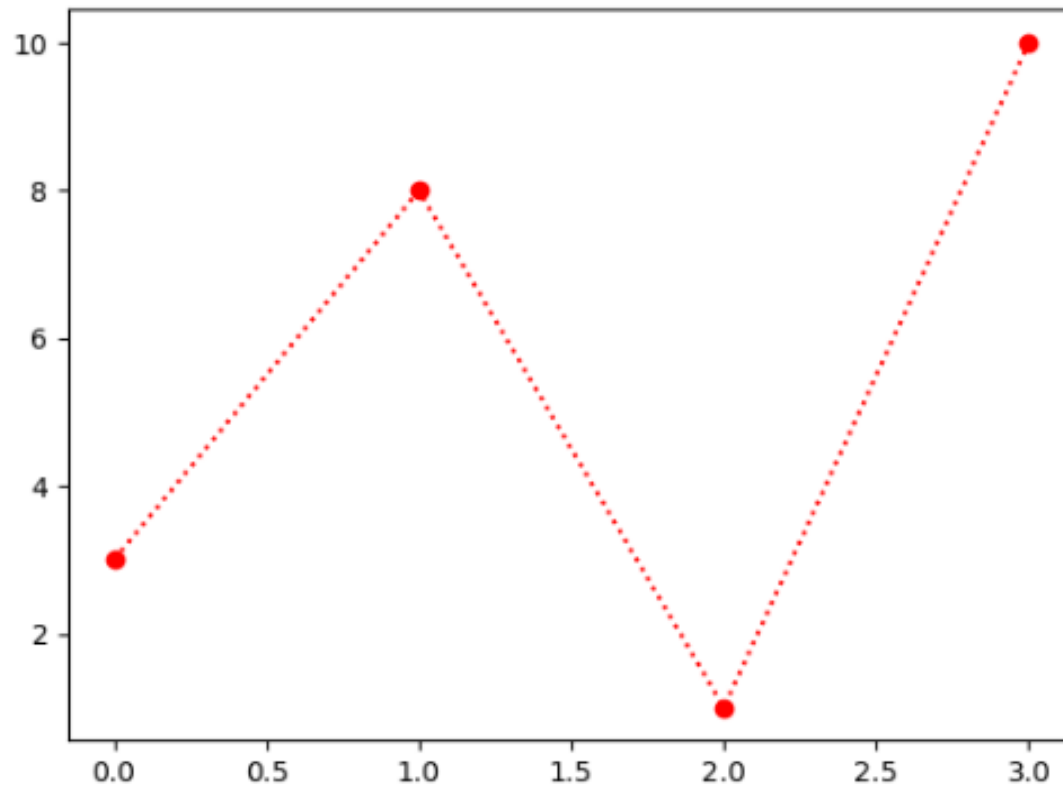
- You can also use the *shortcut string notation* parameter to specify the marker.
- This parameter is also called `fmt`, and is written with this syntax:
- ***marker | line | color***

EXAMPLE

- Mark each point with a circle:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, 'o:r')
plt.show()
```

RESULT



LINE REFERENCE

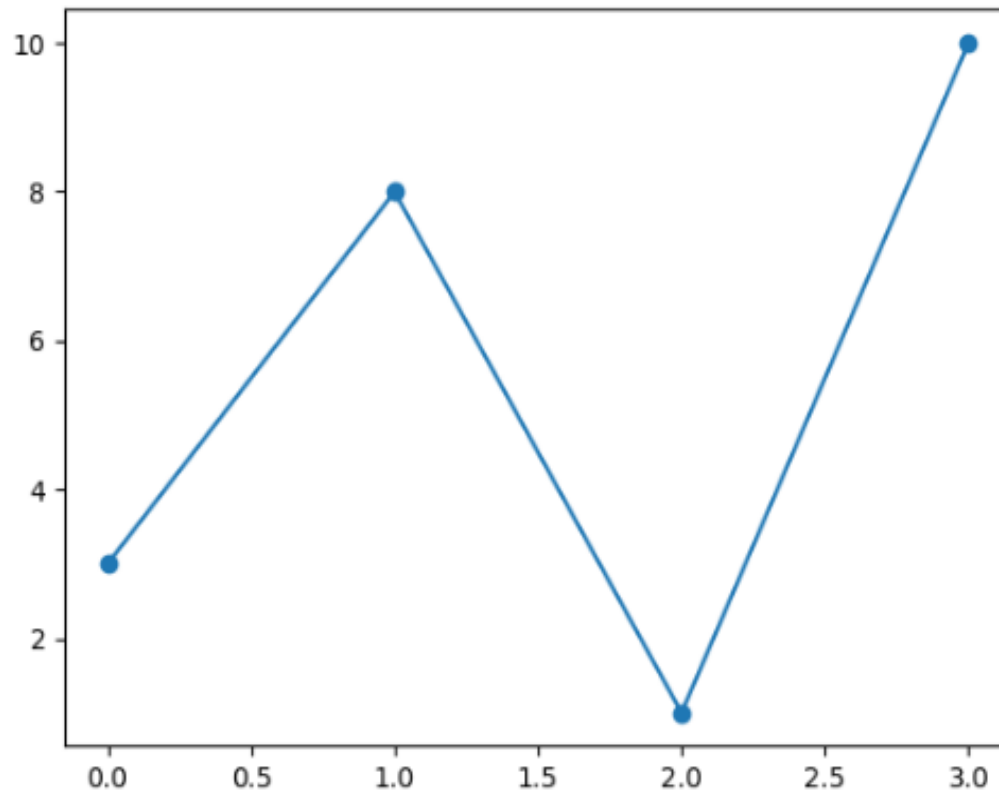
➤ '-' Solid line

Example

Three lines to make our compiler able to draw:

```
import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
y    points = np.array([3, 8, 1, 10])
plt.plot(y, points, 'o-')
plt.show()
```


RESULT



':'DOTTED LINE

➤ Example

Three lines to make our compiler able to draw:

```
import sys
```

```
import matplotlib
```

```
matplotlib.use('Agg')
```

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

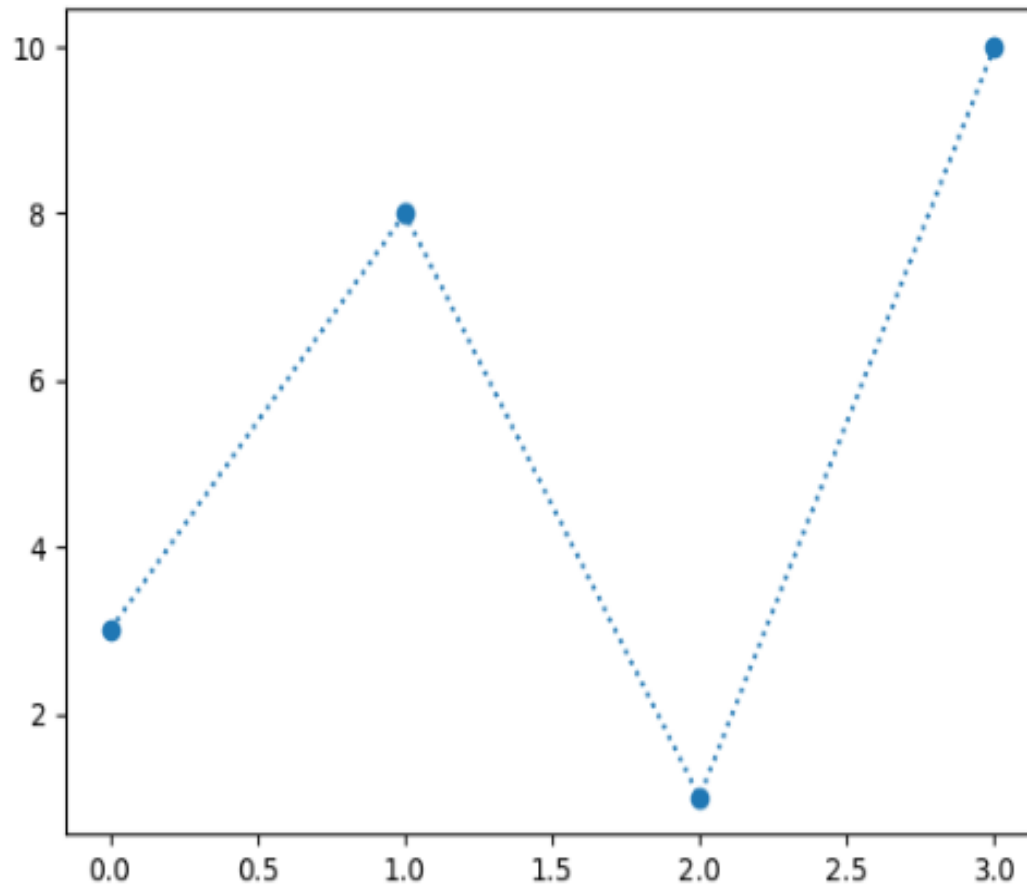
```
import numpy as np
```

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

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

```
plt.show()
```

RESULT



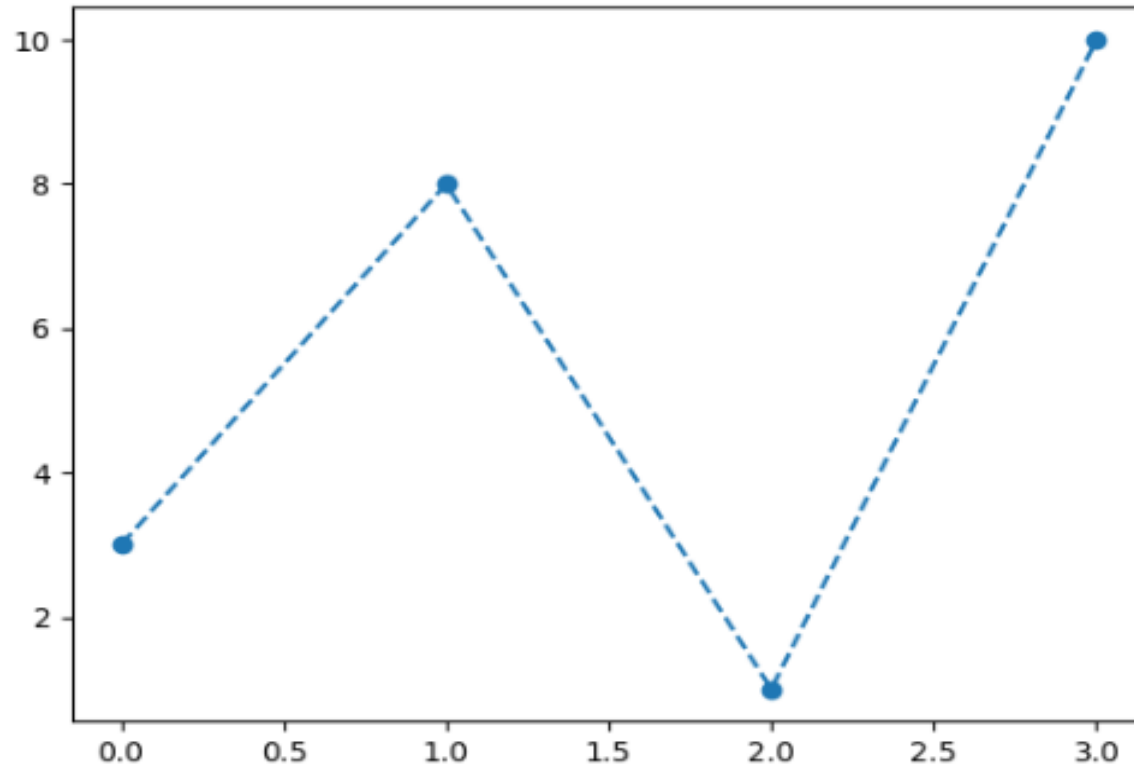
'--'DASHED LINE

Example

- Three lines to make our compiler able to draw:

```
import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, 'o--')
plt.show()
```

RESULT



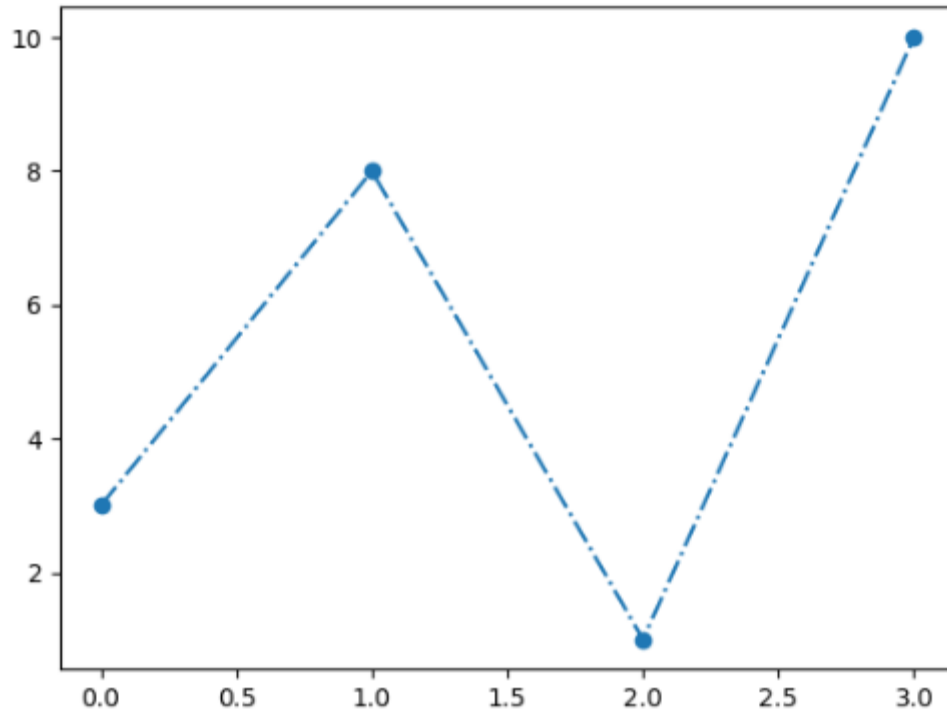
'-.'DASHED/DOTTED LINE

Example

- Three lines to make our compiler able to draw:

```
import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
y points = np.array([3, 8, 1, 10])
plt.plot(y points, 'o-.')
plt.show()
```

RESULT



MATPLOTLIB LINE

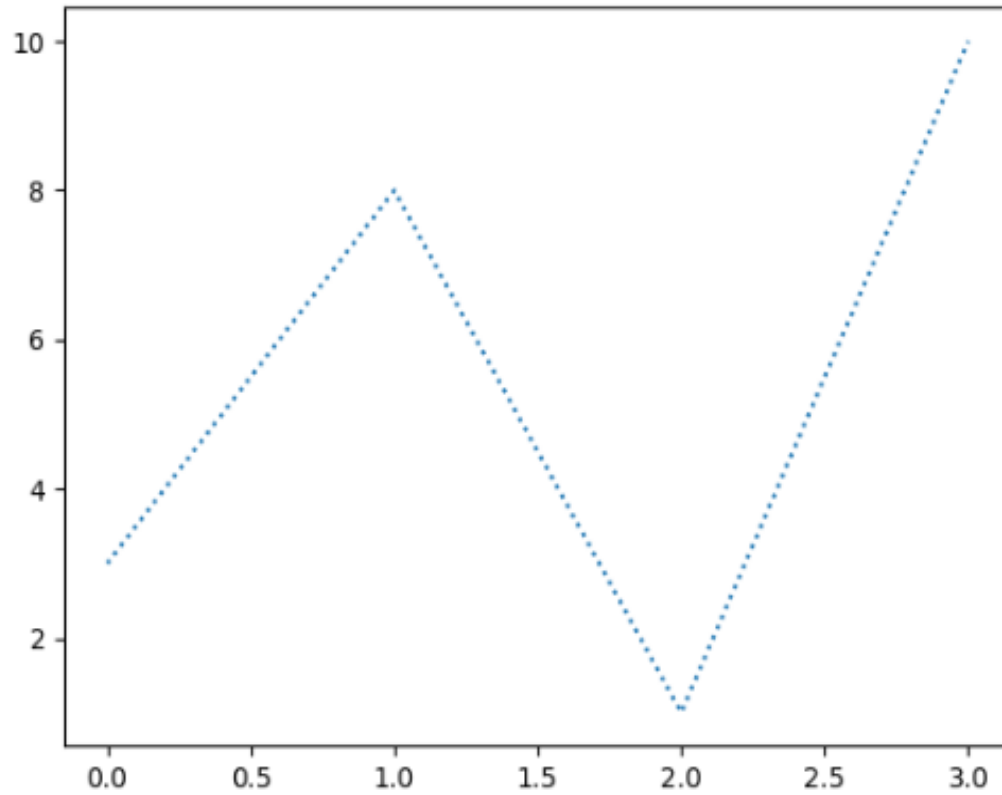
- Linestyle
- You can use the keyword argument `linestyle`, or shorter `ls`, to change the style of the plotted line:

Example

Use a dotted line:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```

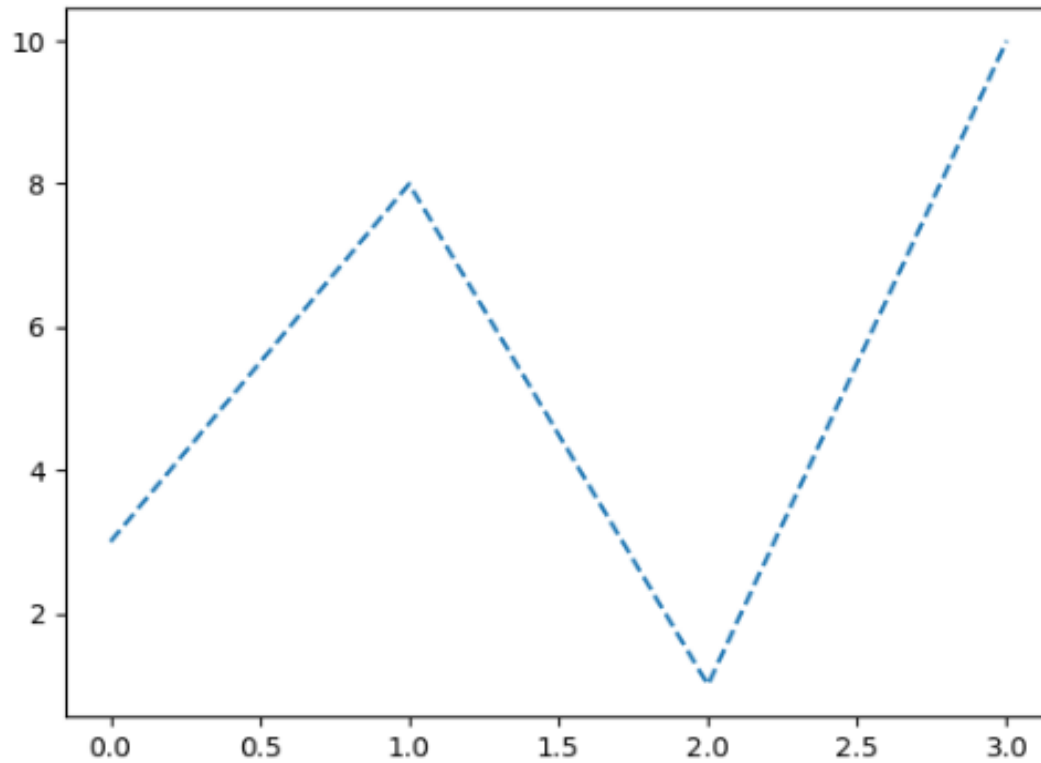

RESULT



EXAMPLE

- Use a dashed line:
`plt.plot(ypoints, linestyle = 'dashed')`

RESULT



SHORTER SYNTAX

- 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 **--**.

EXAMPLE

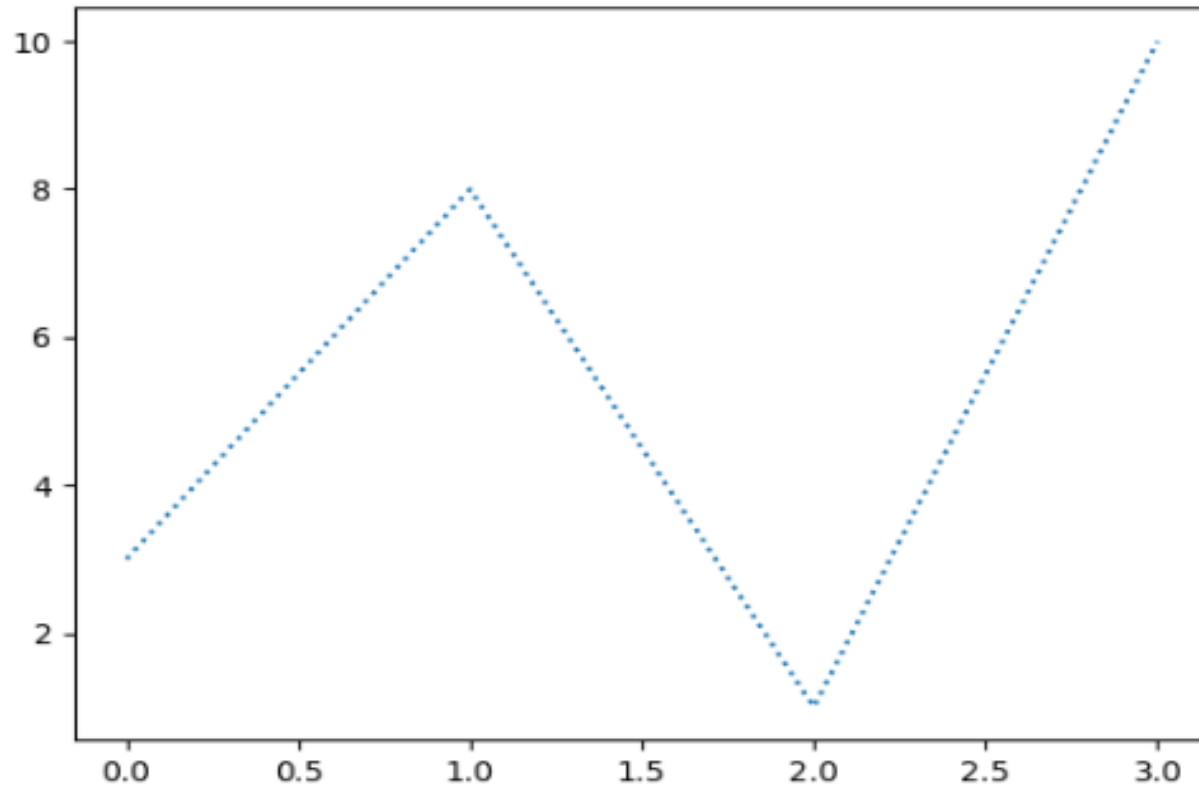
- Shorter syntax:

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

Example:

```
import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, ls = ':')
plt.show()
```

RESULT



LINE STYLES

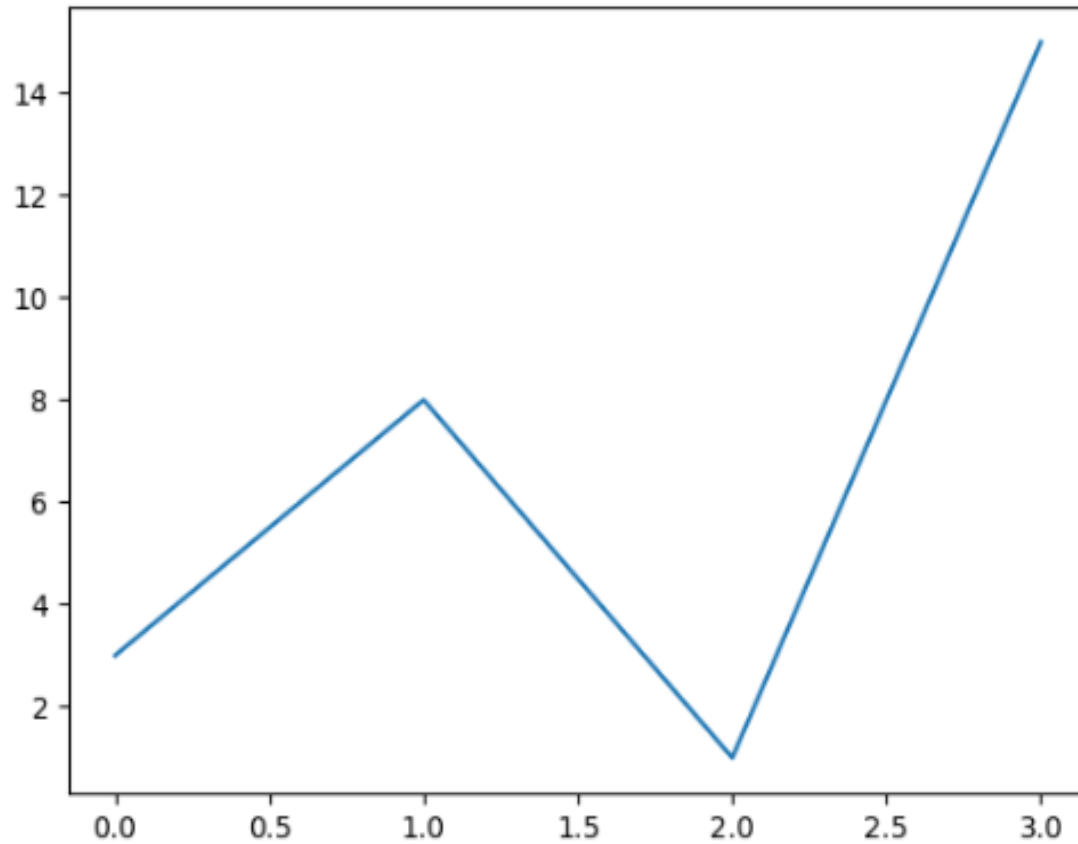
1. 'solid' (default) ‘-‘
2. 'dotted' ":"
3. Dashed "--"
4. Dashdot "-."
5. None "" or ' '

'SOLID' (DEFAULT) '-'

Example

```
import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 15])
plt.plot(ypoints, ls = '-')
plt.show()
```


RESULT

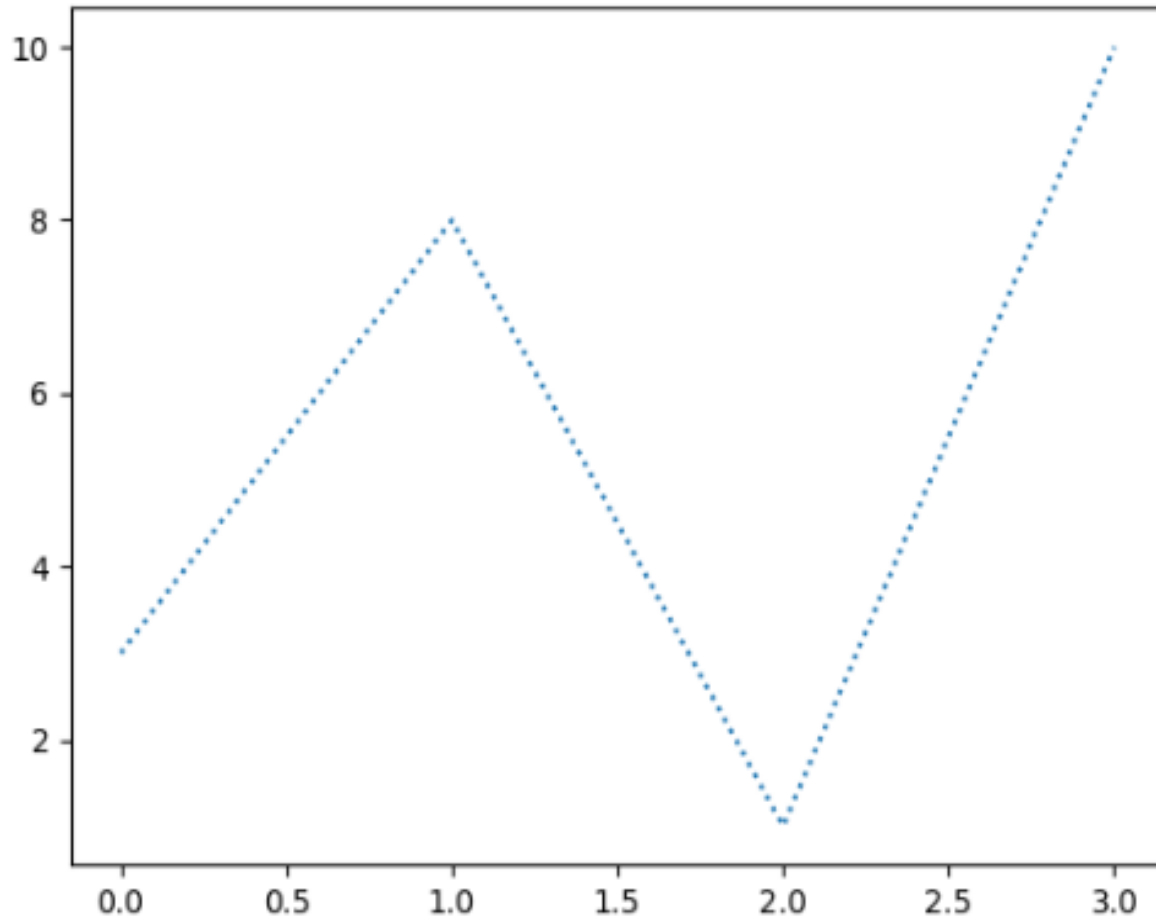


'DOTTED':

Example:

```
import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, ls = ':')
plt.show()
```

RESULT

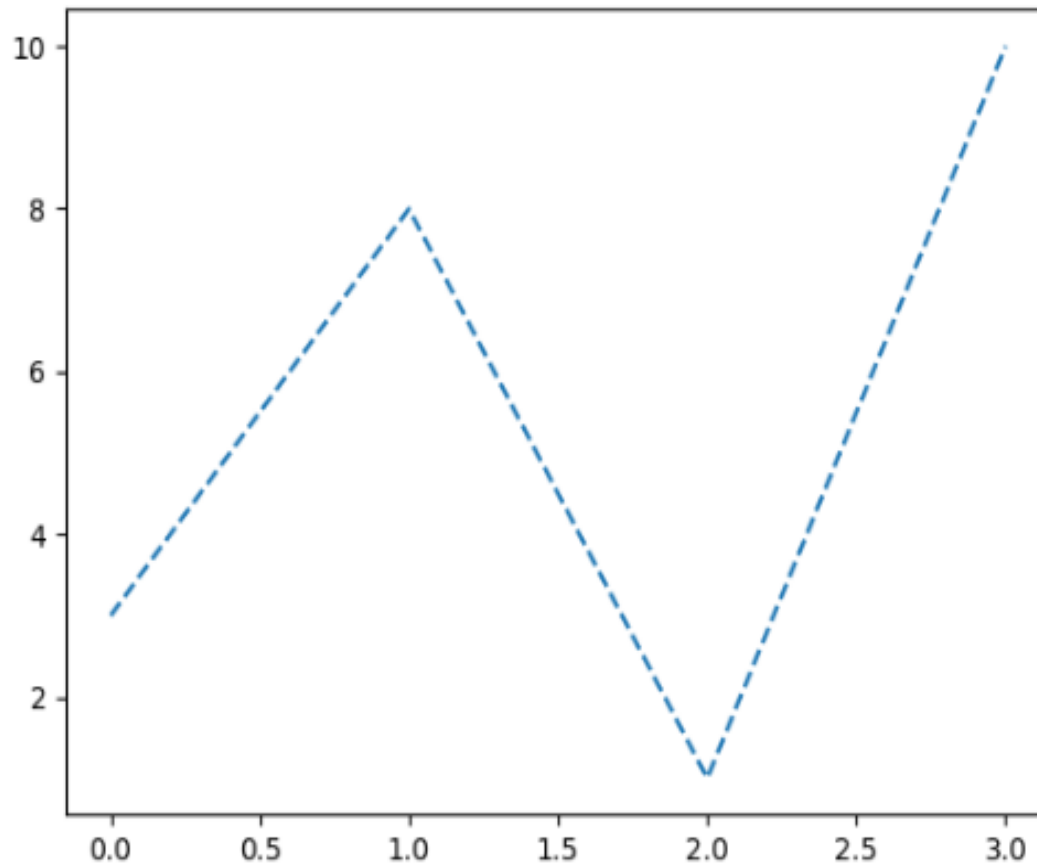


'DASHED"--'

Example:

```
import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, ls = '--')
plt.show()
```

RESULT

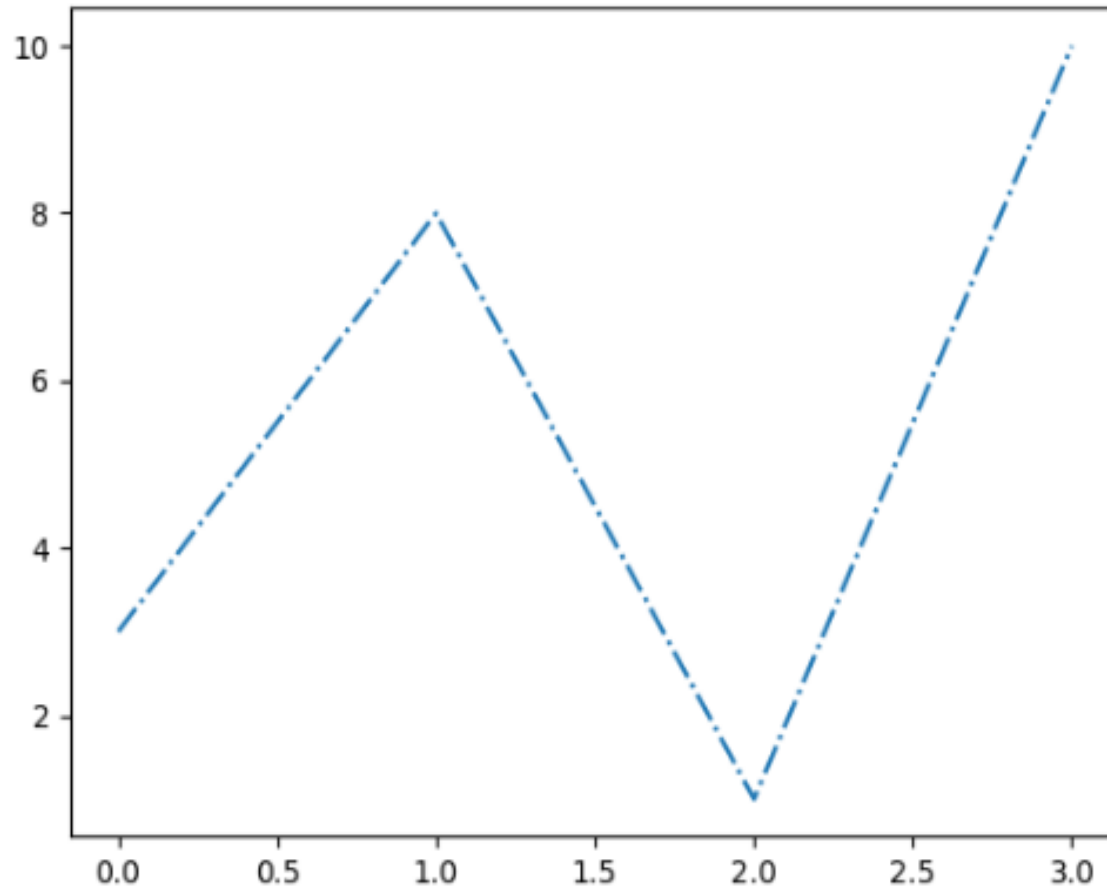


'DASHDOT'-'.'

Example

```
import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, ls = '-.')
plt.show()
```

RESULT

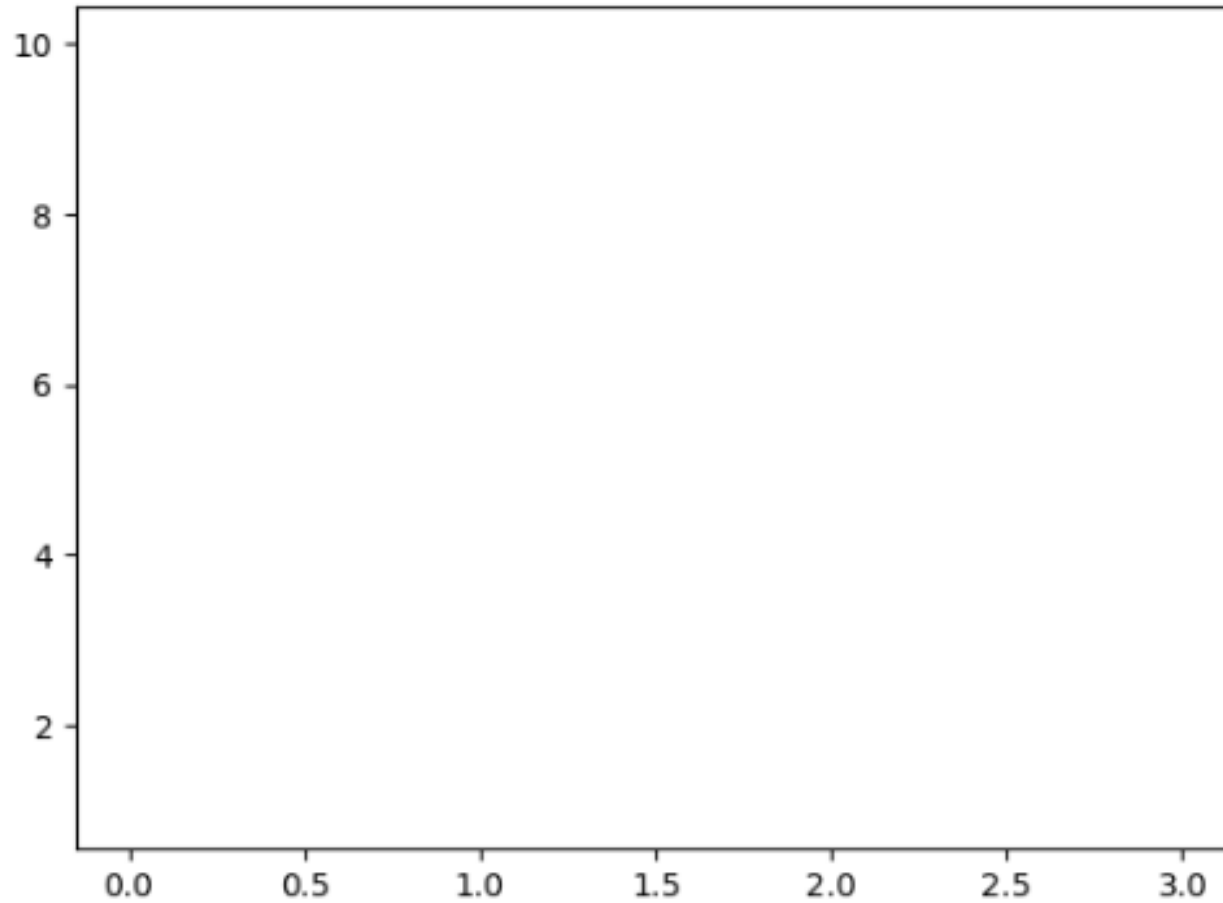


‘NONE’ OR ‘ ’

Example

```
import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, ls = "")
plt.show()
```


RESULT



LINE COLOR

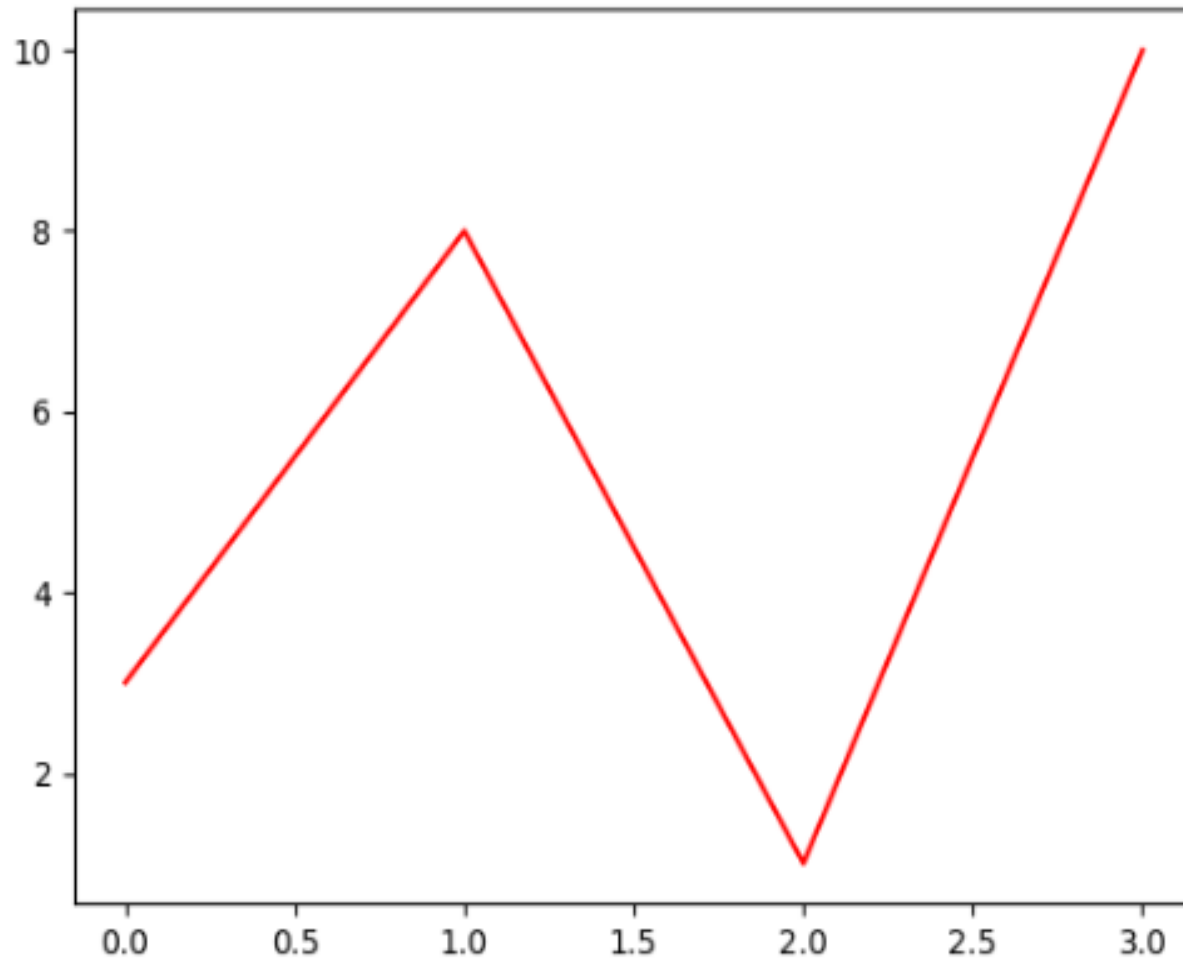
- You can use the keyword argument `color` or the shorter `c` to set the color of the line:

Example:

- Set the line color to red:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, color = 'r')
plt.show()
```

RESULT



EXAMPLE

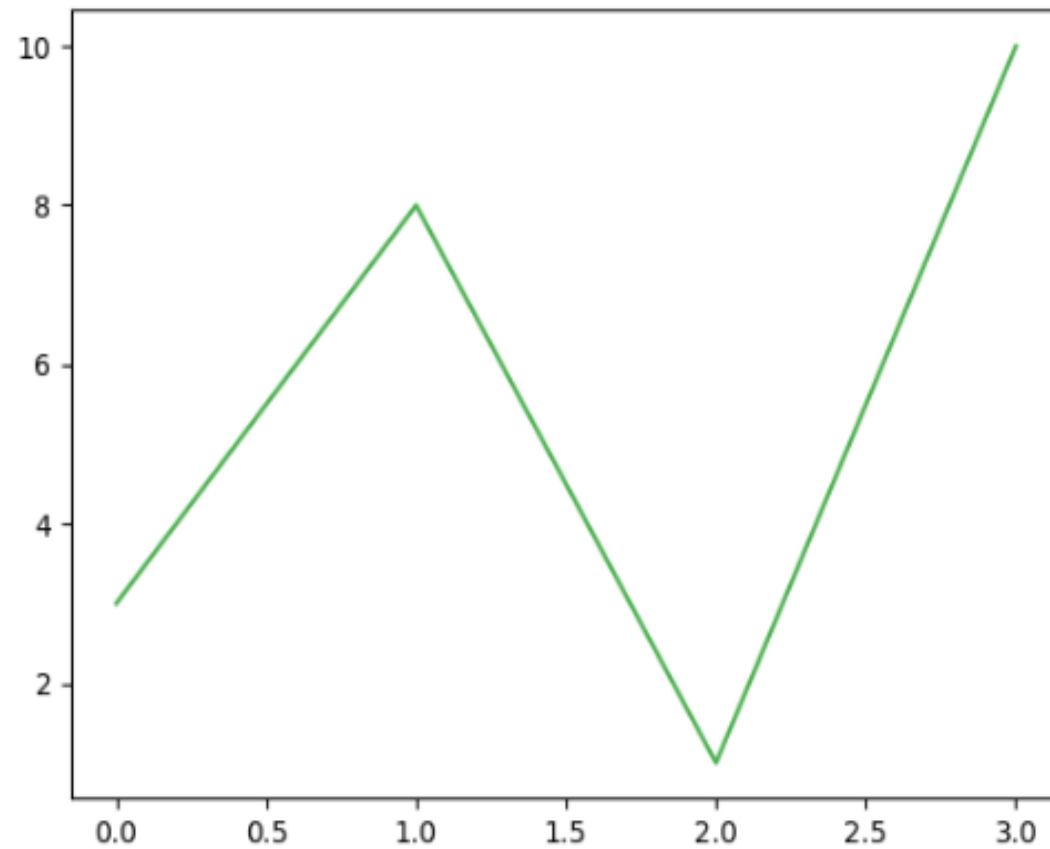
- Plot with a beautiful **green** line:

...

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

...

RESULT



EXAMPLE

- Plot with the color named "**hotpink**":

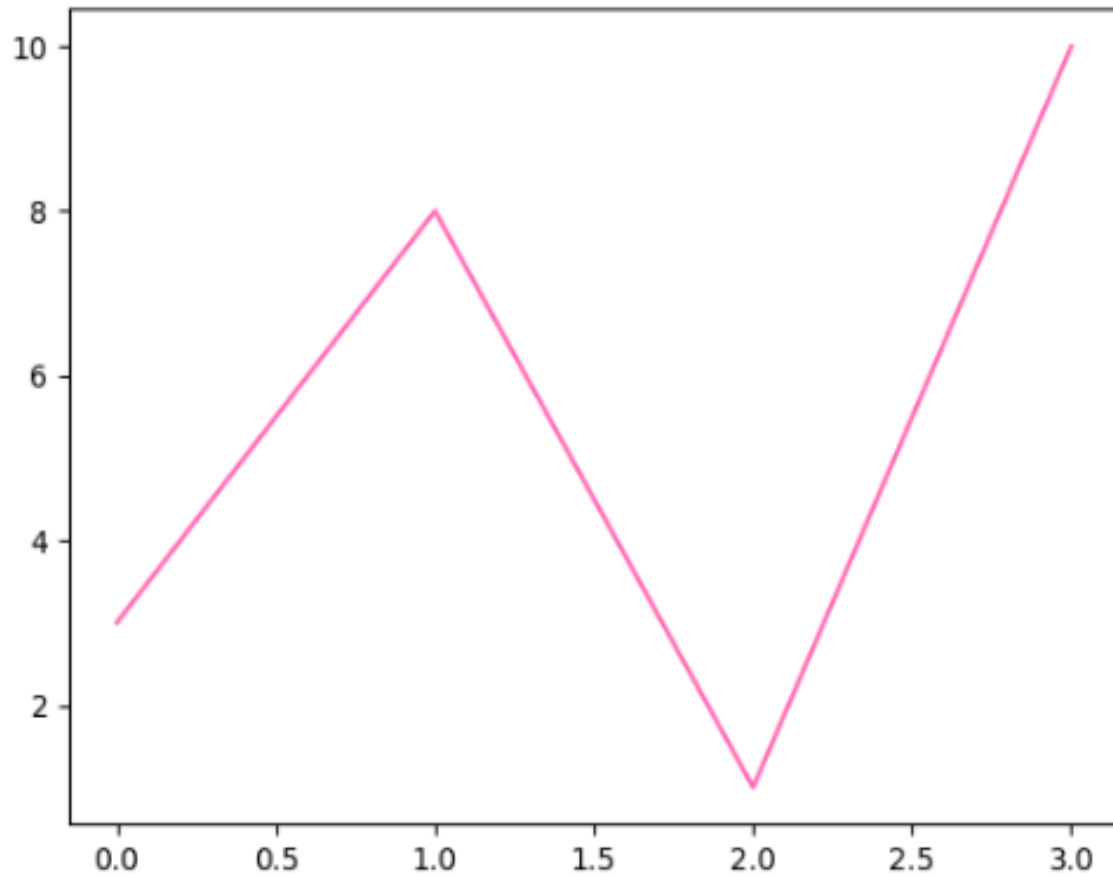
...

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

```

...

RESULT



LINE WIDTH

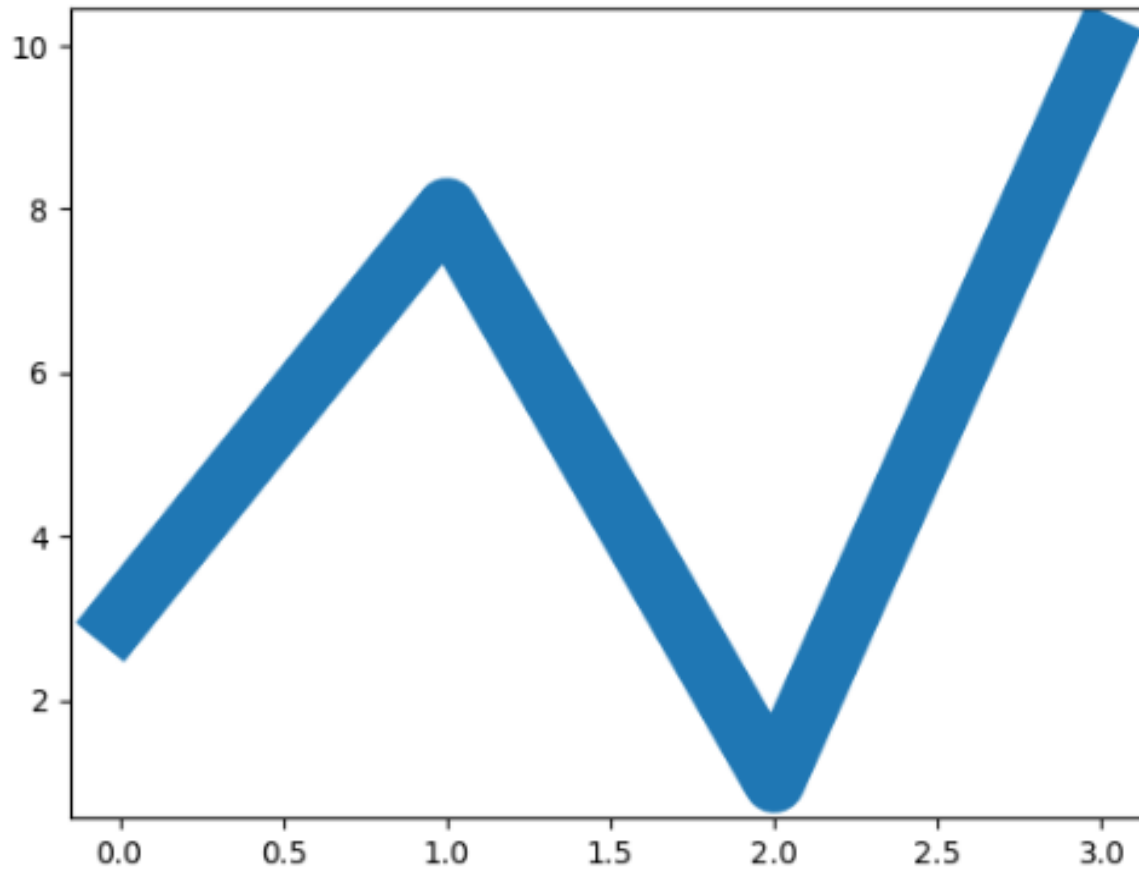
- You can use the keyword argument `linewidth` or the shorter `lw` to change the width of the line.
- The value is a floating number, in points:

Example

- Plot with a 20.5pt wide line:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, linewidth = '20.5')
plt.show()
```


RESULT



MULTIPLE LINES

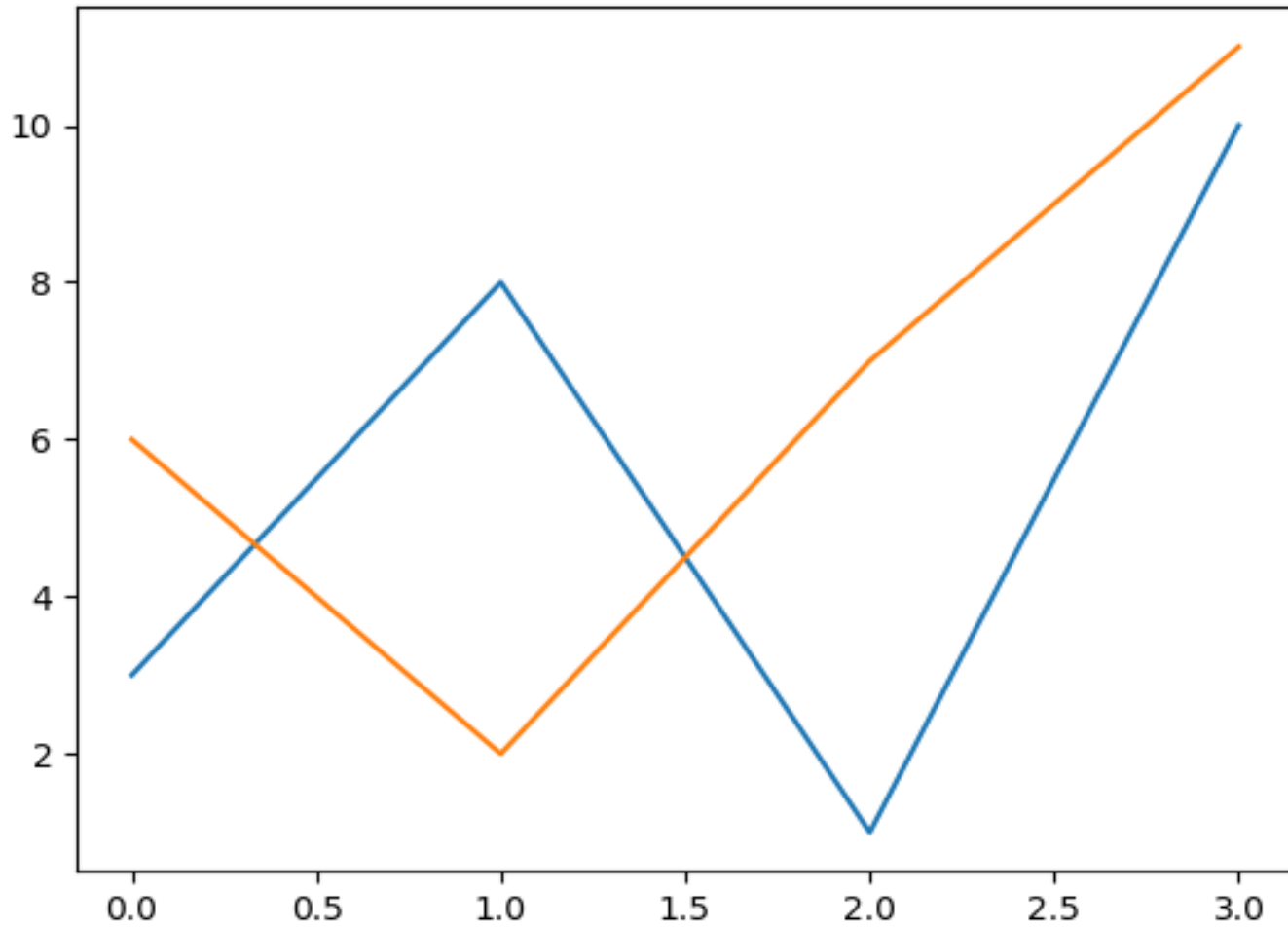
- You can plot as many lines as you like by simply adding more `plt.plot()` functions:

Example

- Draw two lines by specifying a `plt.plot()` function for each line:

```
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()
```

RESULT



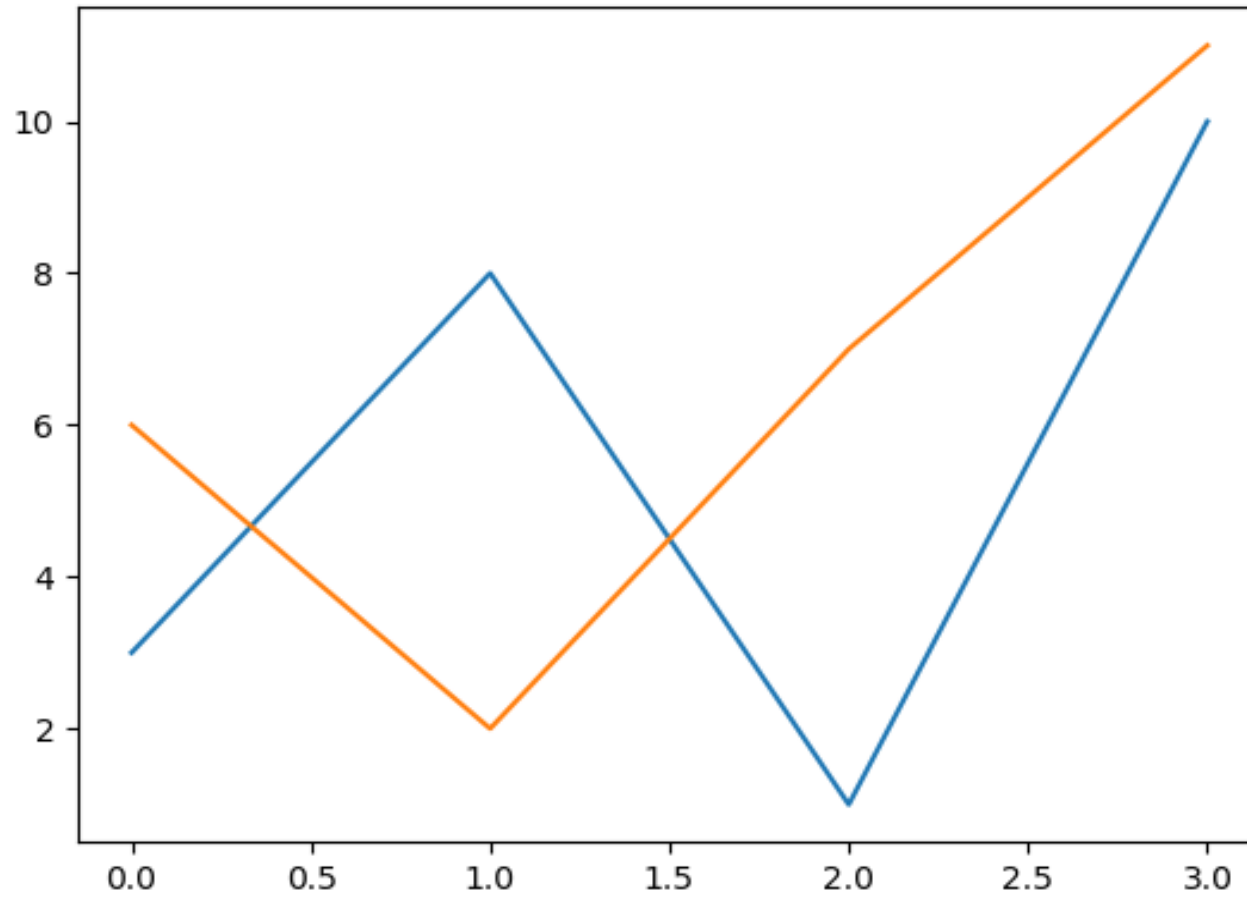
- You can also plot many lines by adding the points for the x- and y-axis for each line in the same `plt.plot()` function.
- (In the examples above we only specified the points on the y-axis, meaning that the points on the x-axis got the the default values (0, 1, 2, 3).)
- The x- and y- values come in pairs:

EXAMPLE

- Draw two lines by specifying the x- and y-point values for both lines:

```
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()
```

RESULT



MATPLOTLIB LABELS AND TITLE

Create Labels for a Plot

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

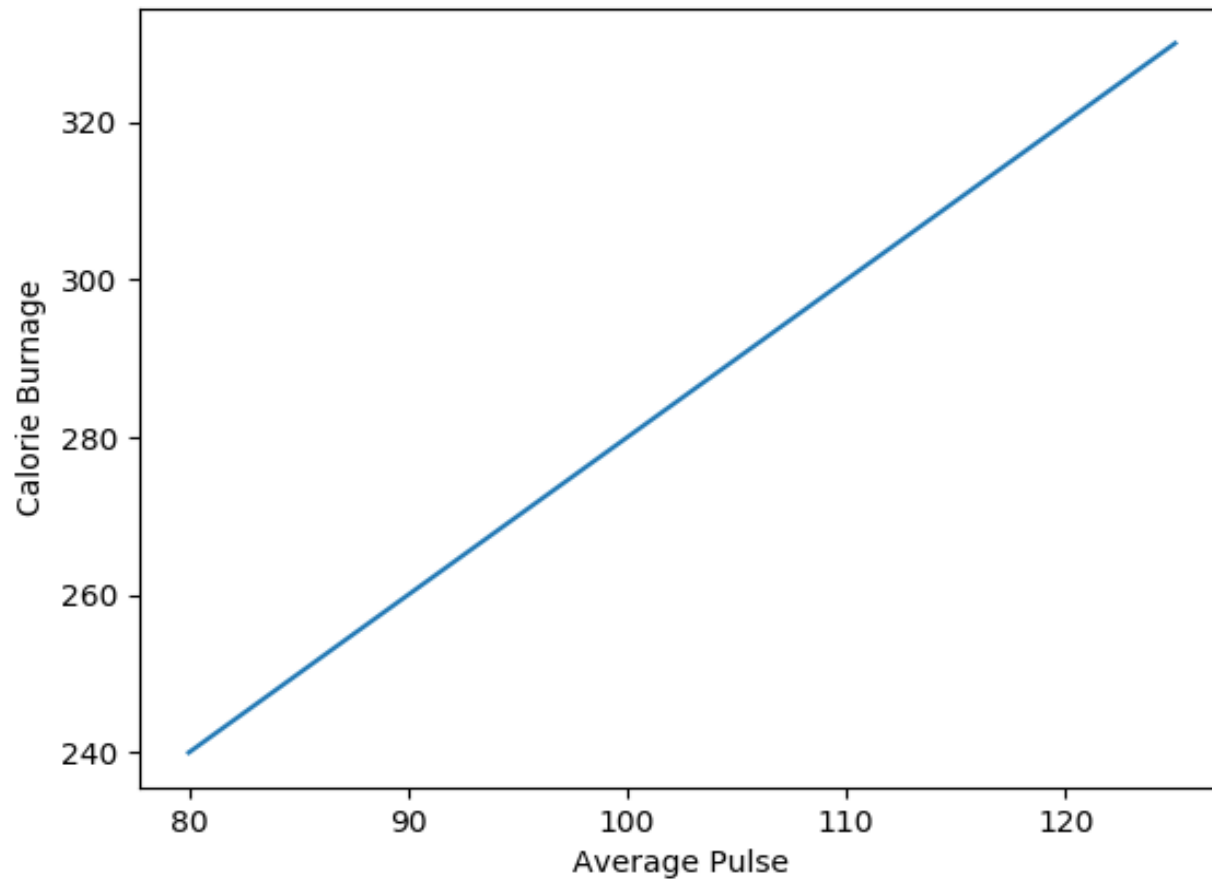
EXAMPLE

- Add labels to the x- and 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.plot(x, y)
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.show()
```


RESULT



CREATE A TITLE FOR A PLOT

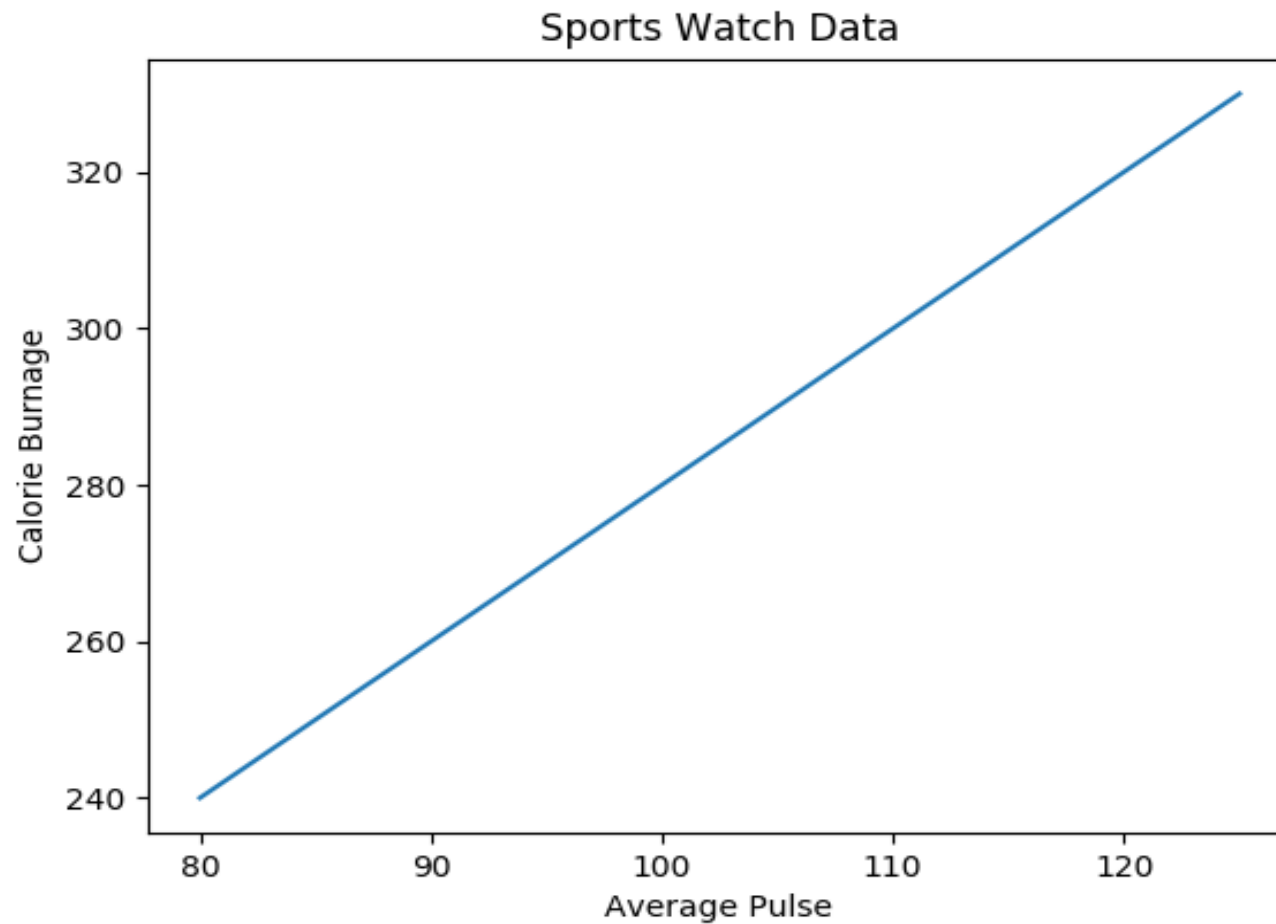
- With Pyplot, you can use the `title()` function to set a title for the plot.

EXAMPLE

- Add a plot title and labels for the x- and 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.plot(x, y)
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.show()
```

RESULT



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.

EXAMPLE

- Set font properties for the title and labels:

```
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])
```

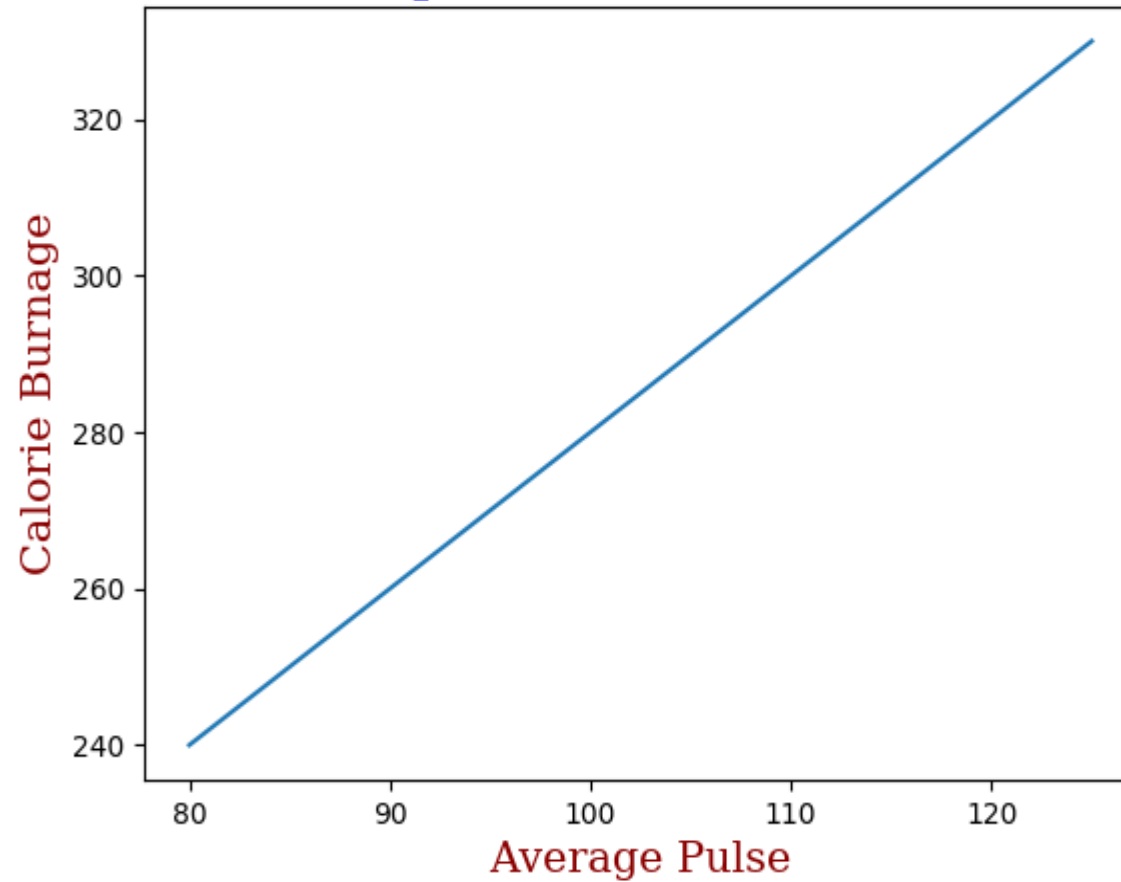
```
font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}
```

```
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()
```

RESULT

Sports Watch Data



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'.

EXAMPLE

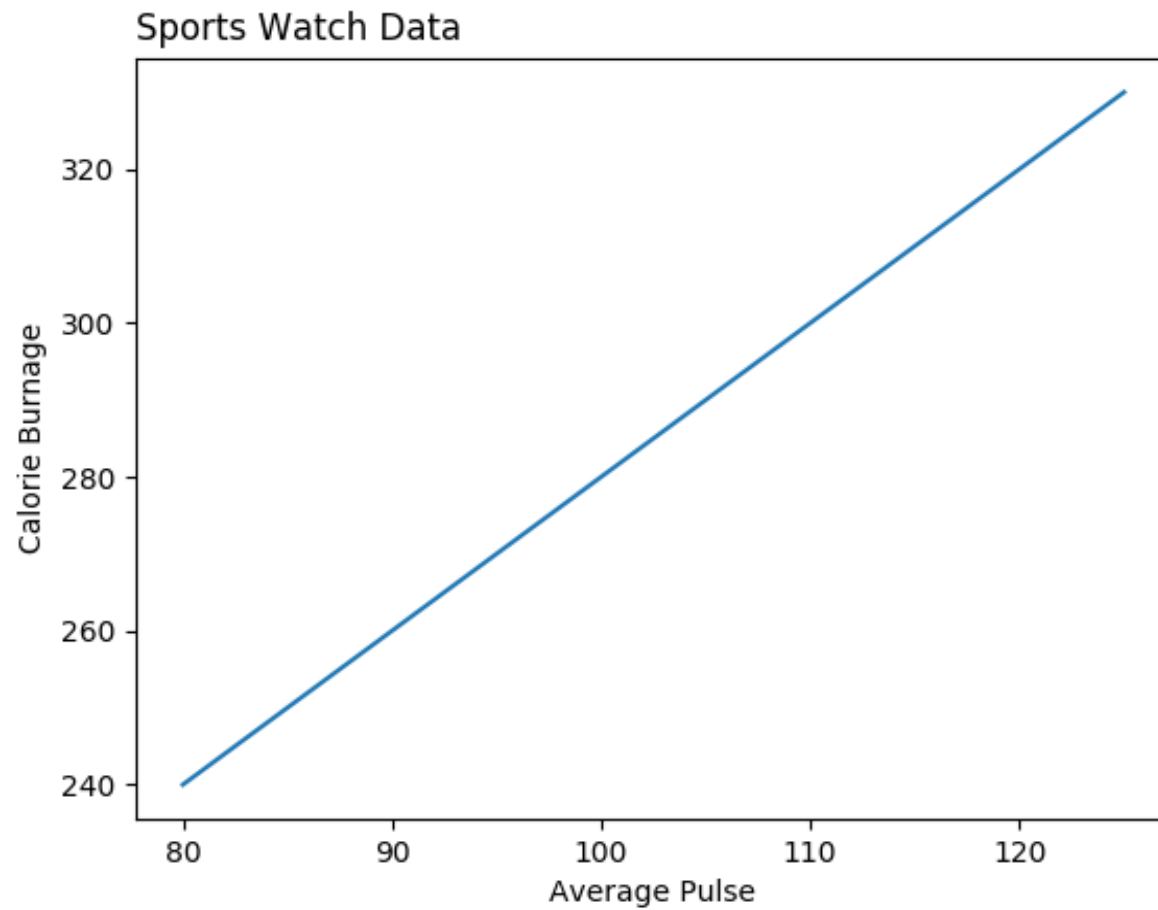
- Position the title to the left:

```
import numpy as np
import matplotlib.pyplot as plt
x =
np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 1
25])
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()
```

RESULT



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.

EXAMPLE

Add grid lines to the plot:

- ```
import numpy as np
import matplotlib.pyplot as plt
x =
np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 1
25])
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()
```

# RESULT

## 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'.

## EXAMPLE

- Display only grid lines for the x-axis:

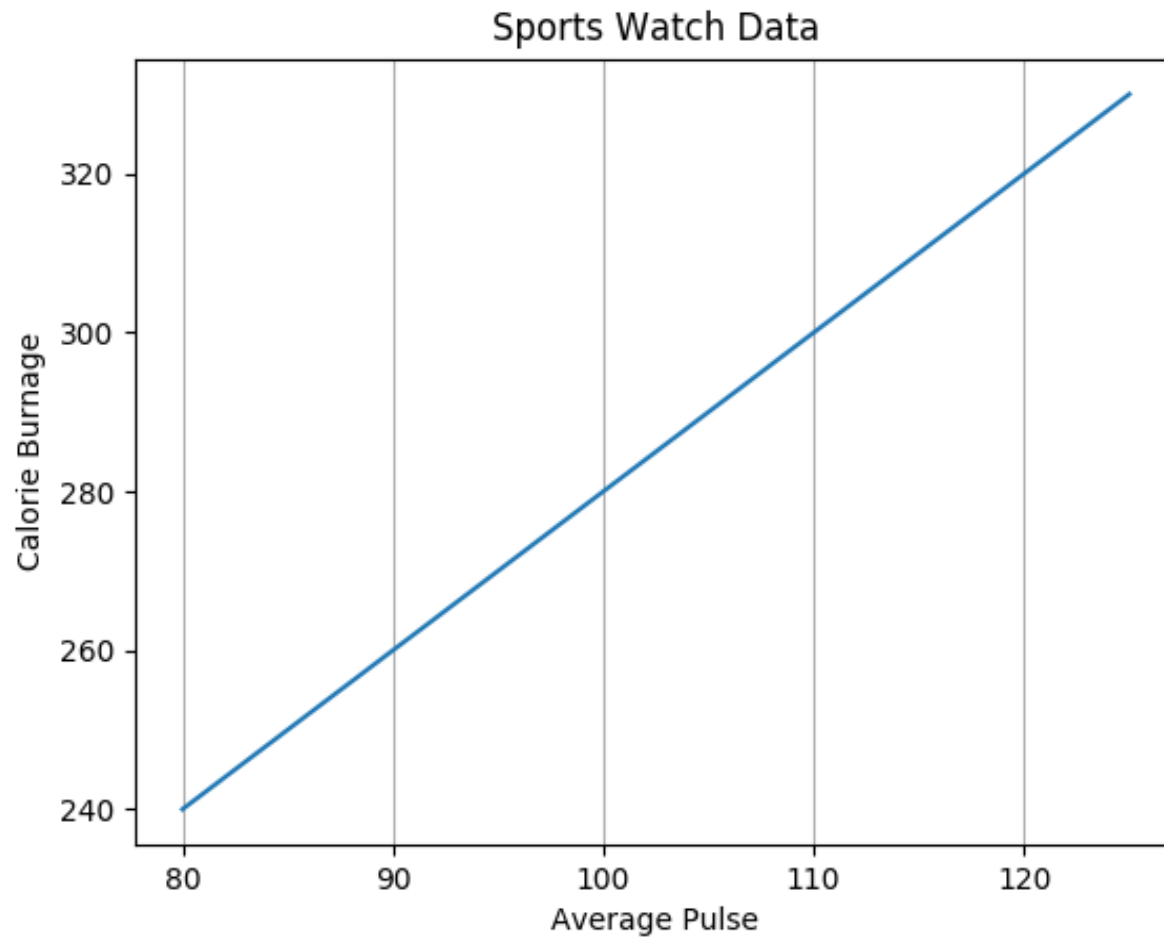
```
import numpy as np
import matplotlib.pyplot as plt

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()
```

# RESULT





# EXAMPLE

- Display only grid lines for the y-axis:

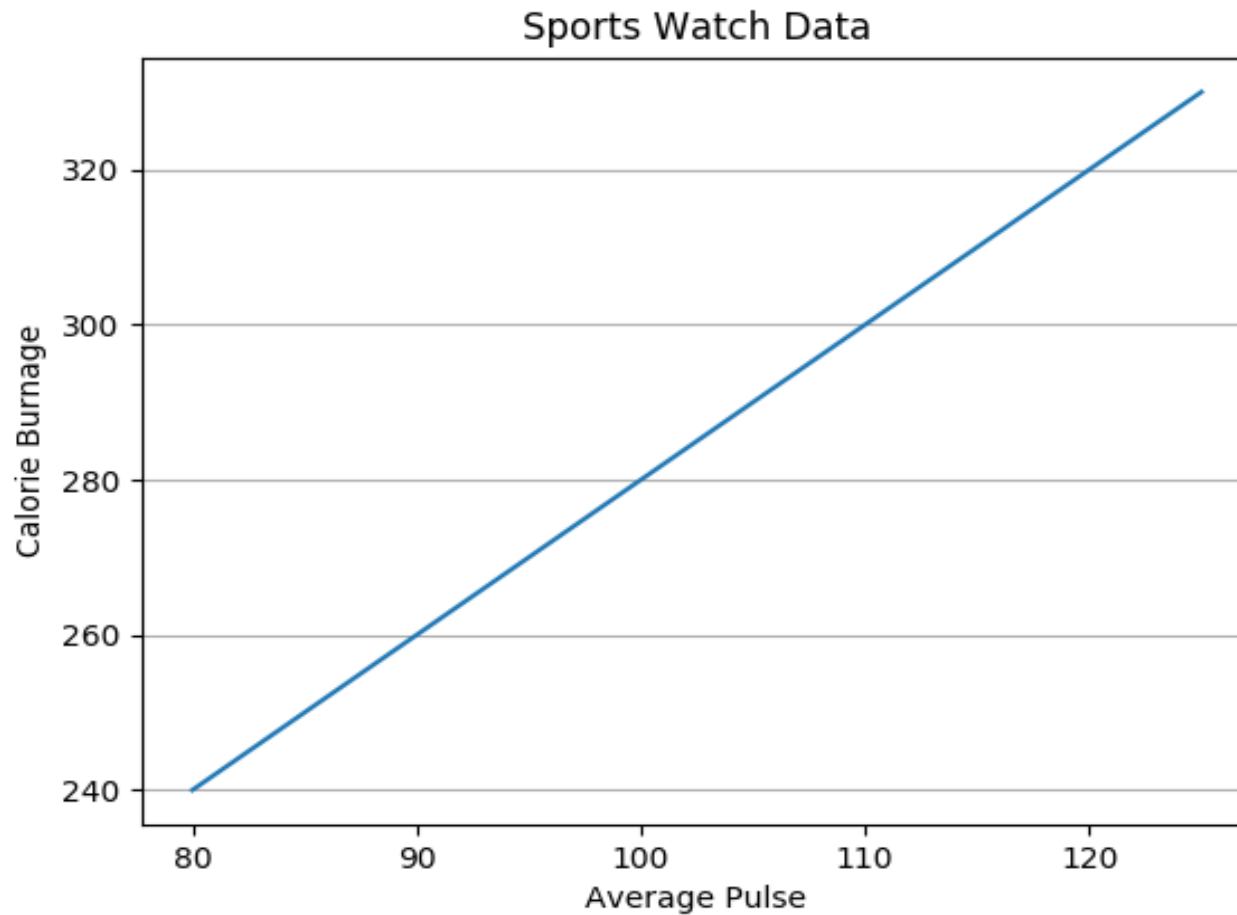
```
import numpy as np
import matplotlib.pyplot as plt

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()
```

# RESULT



# 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).`

- Example

Set the line properties of the grid:

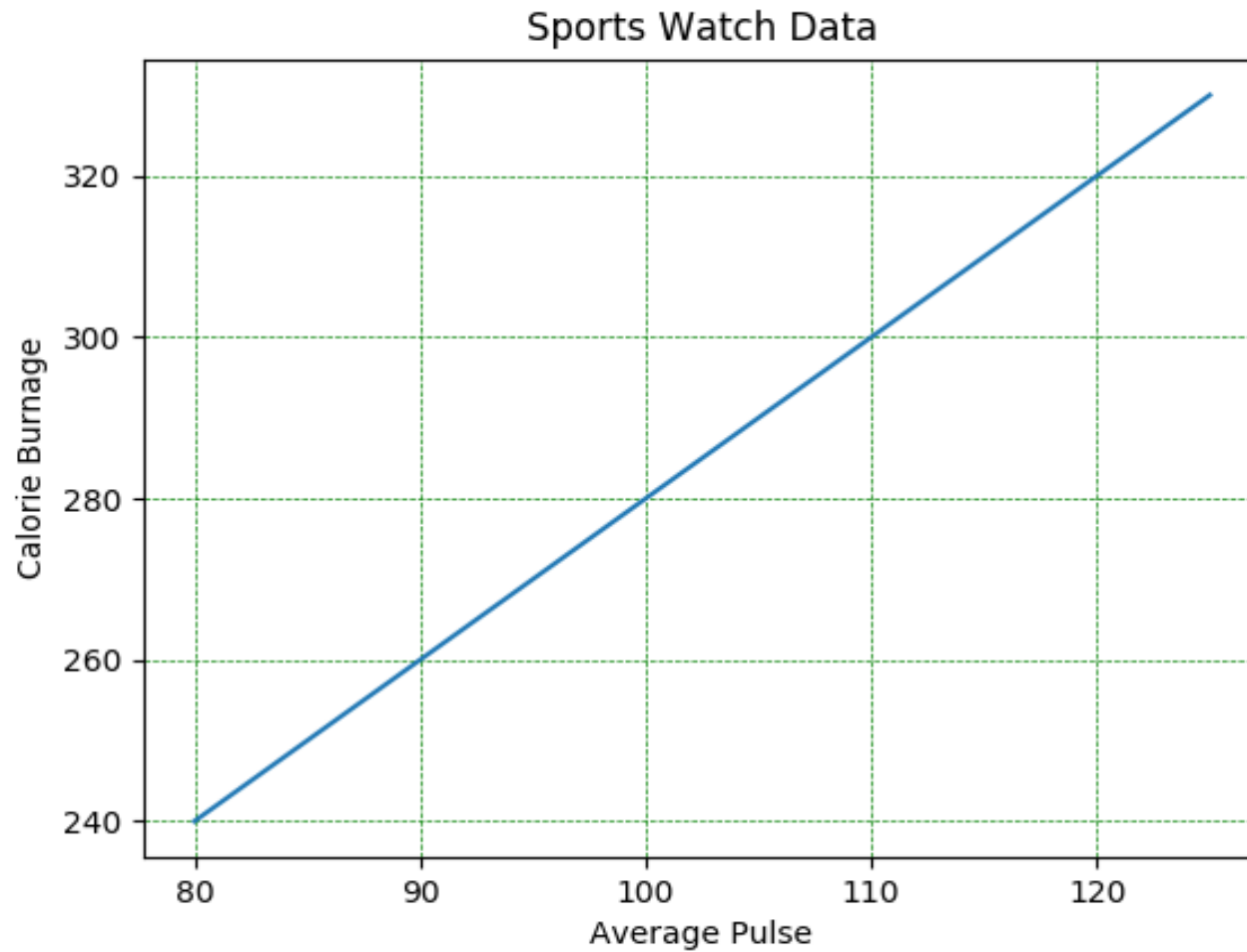
```
import numpy as np
import matplotlib.pyplot as plt

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()
```

# RESULT:



# MATPLOTLIB SUBPLOT

- Display Multiple Plots
- With the `subplot()` function you can draw multiple plots in one figure:

## EXAMPLE

Draw 2 plots:

```
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)
```

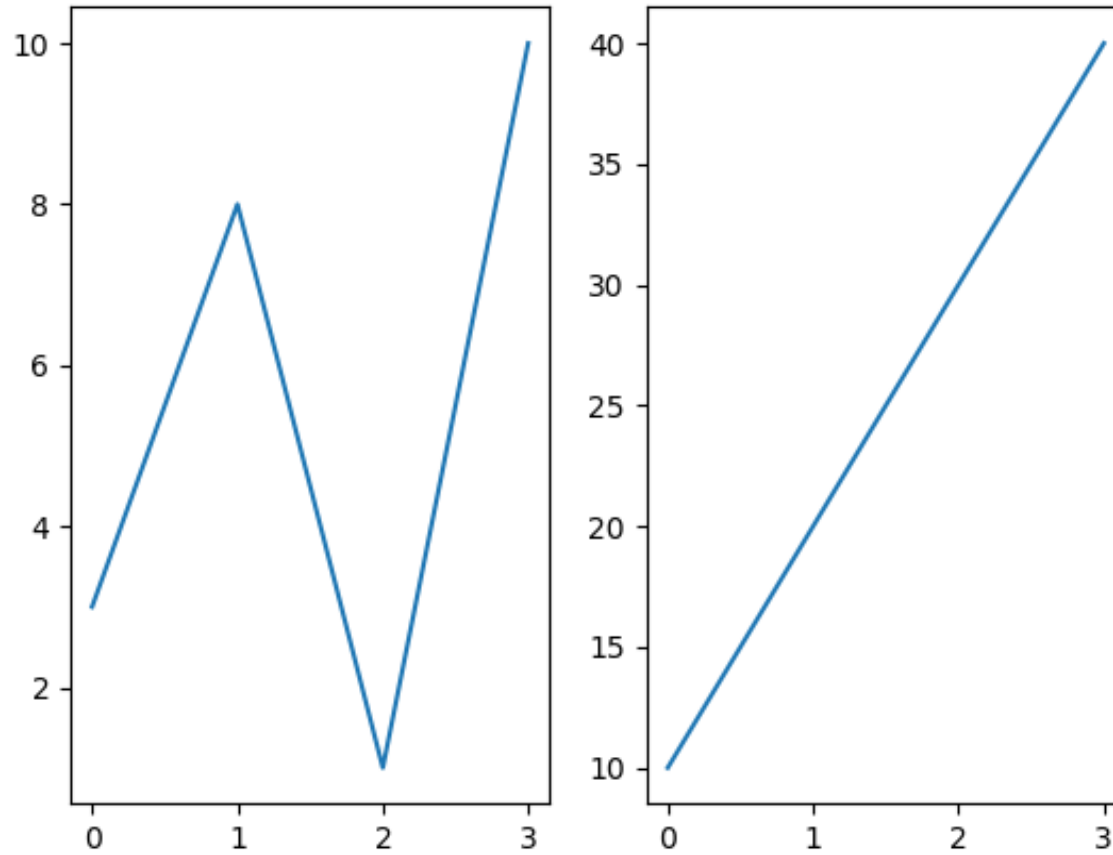
```
#plot 2:
```

```
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()
```

# RESULT



# 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 and 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:



# EXAMPLE

- Draw 2 plots on top of each other:

```
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()
```

# RESULT

# EXAMPLE

- Draw 6 plots:

```
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])
```

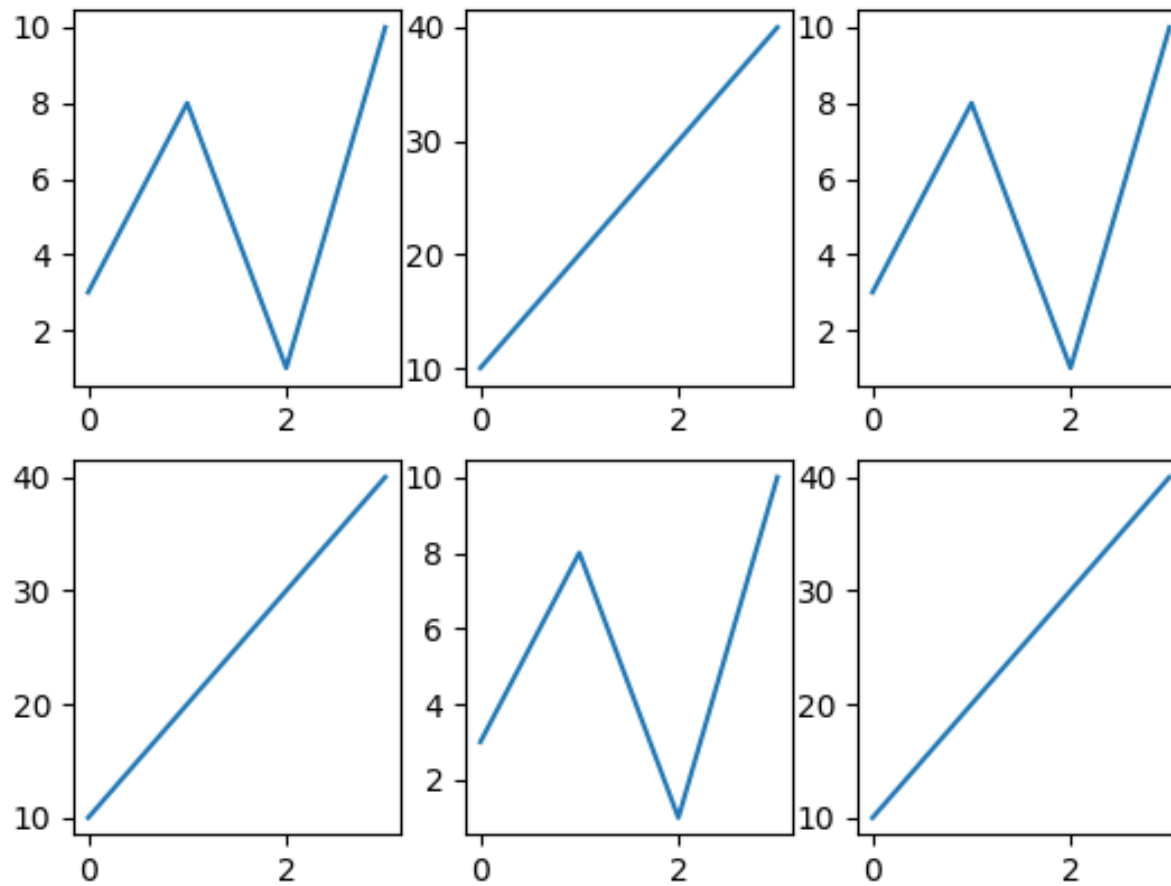
```
plt.subplot(2, 3, 4)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
```

```
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()
```

# RESULT



# TITLE

- You can add a title to each plot with the `title function()`.

# EXAMPLE

- 2 plots, with titles:

```
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")
```

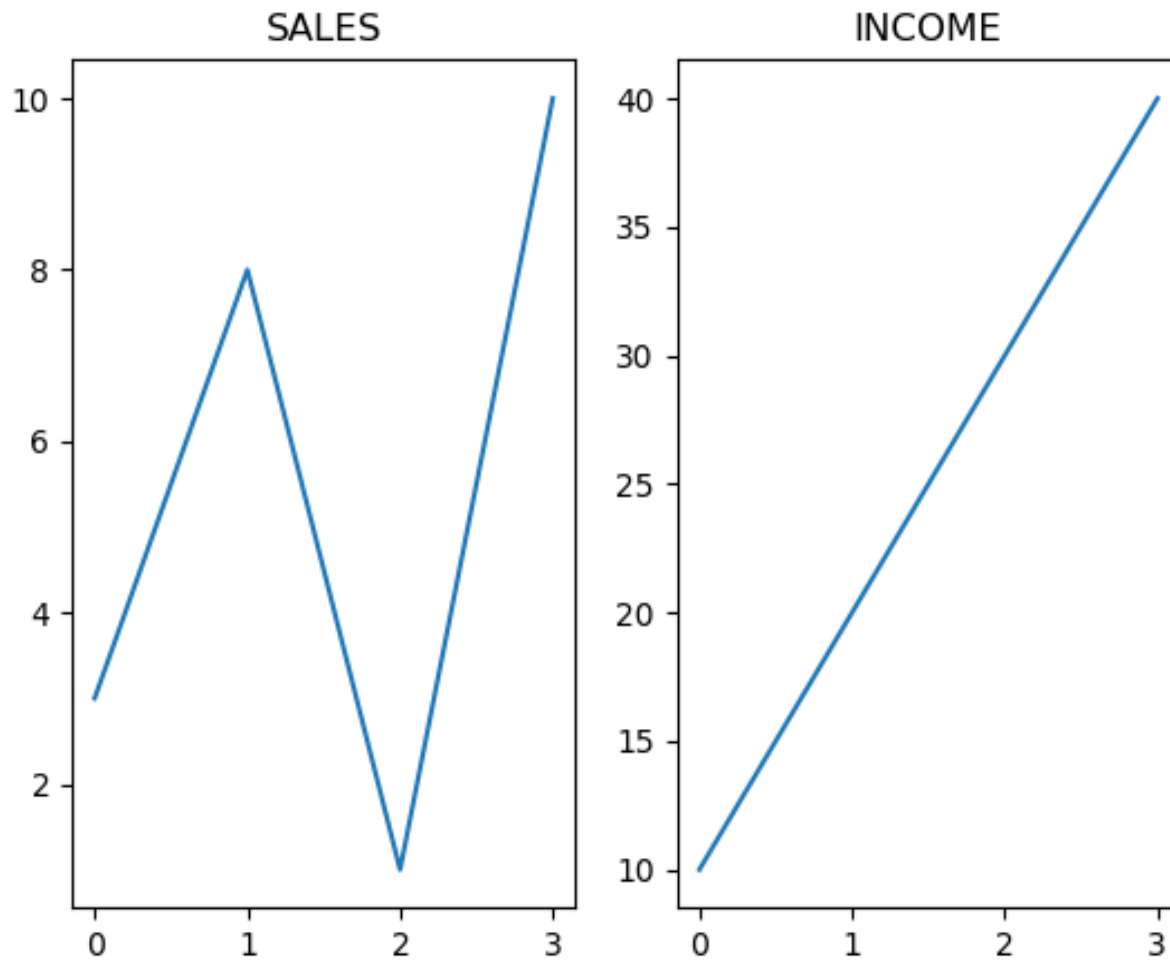
```
#plot 2:
```

```
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()
```

# RESULT



# SUPER TITLE

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

## Example

- Add a title for the entire figure:

```
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:
```

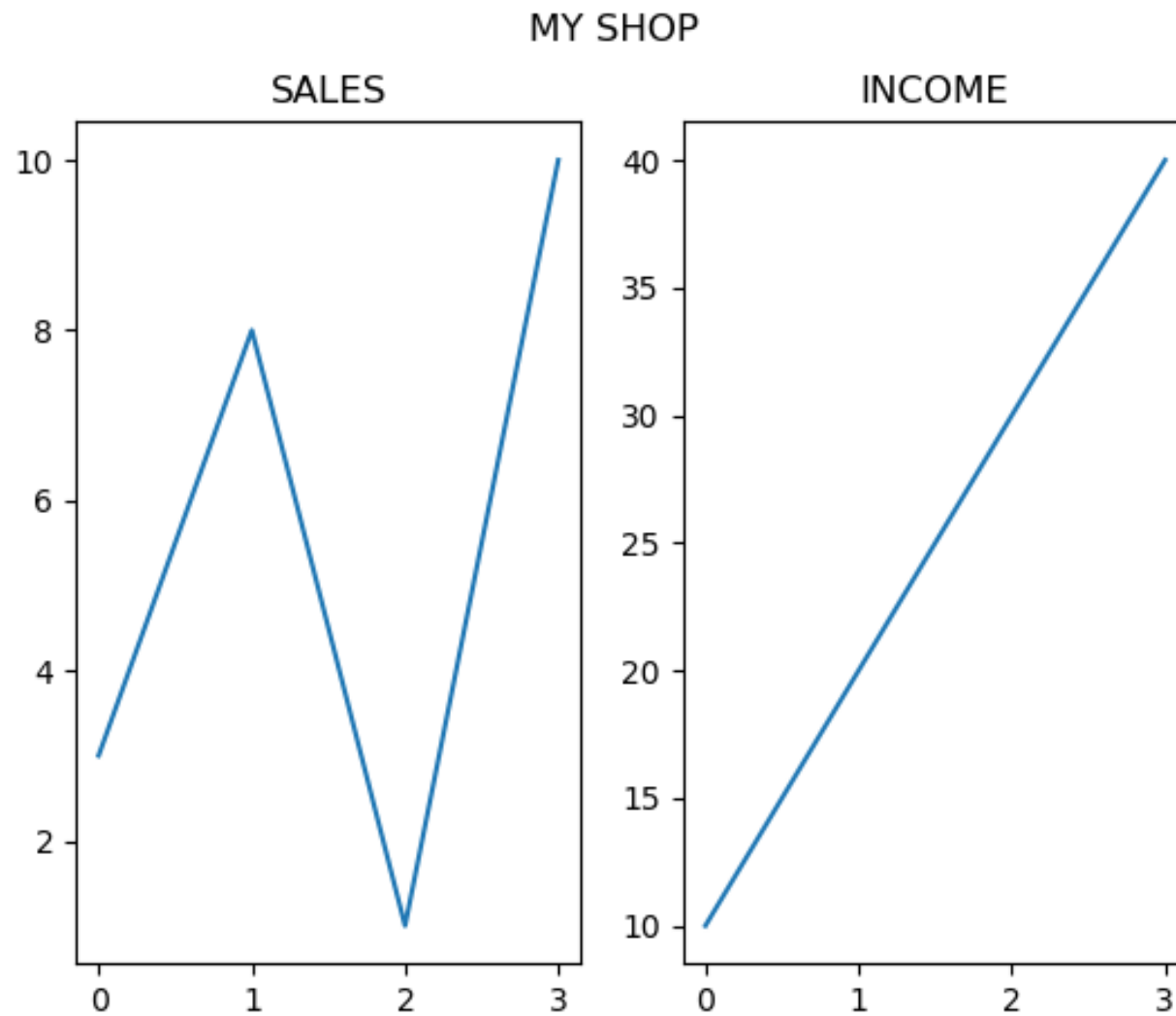
```
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
```

```
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")
```

```
plt.suptitle("MY SHOP")
plt.show()
```



# RESULT



# 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:

## EXAMPLE

- A simple scatter plot:

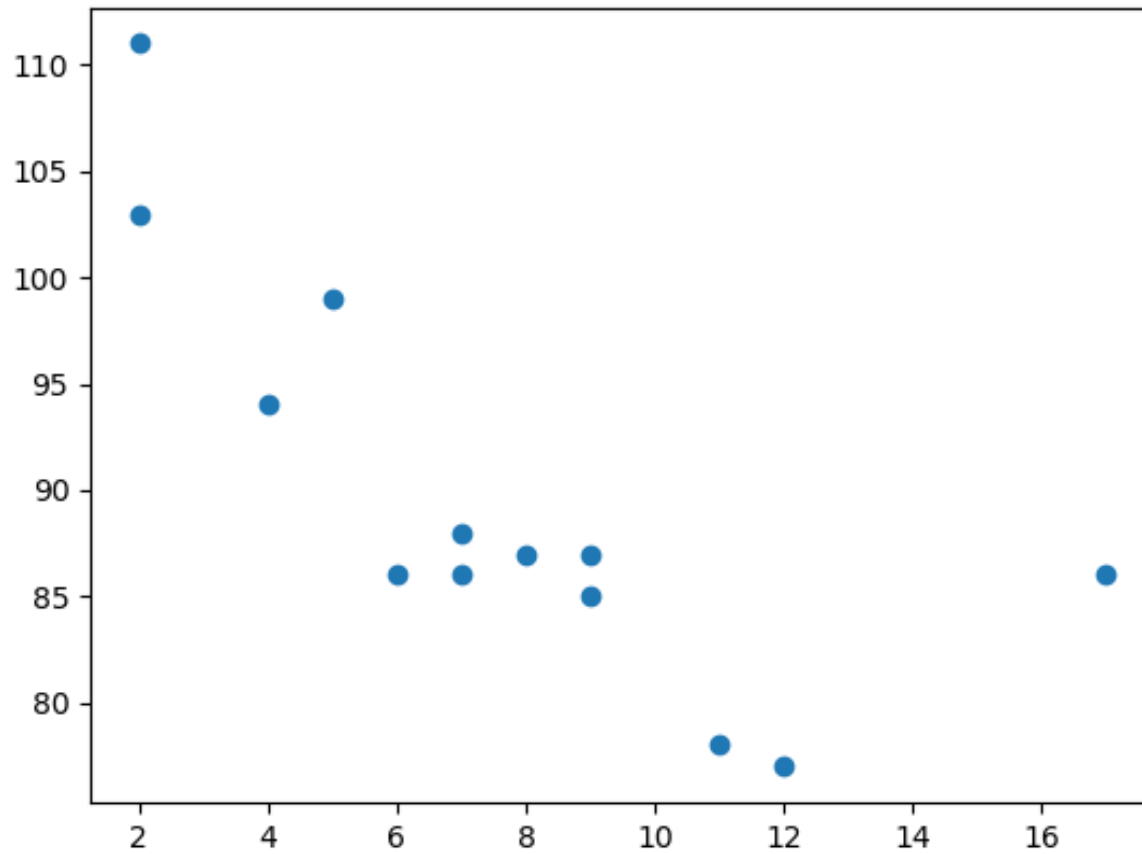
```
import matplotlib.pyplot as plt
import numpy as np
```

```
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])
```

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

# RESULT



- 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?

## EXAMPLE

Draw two plots on the same figure:

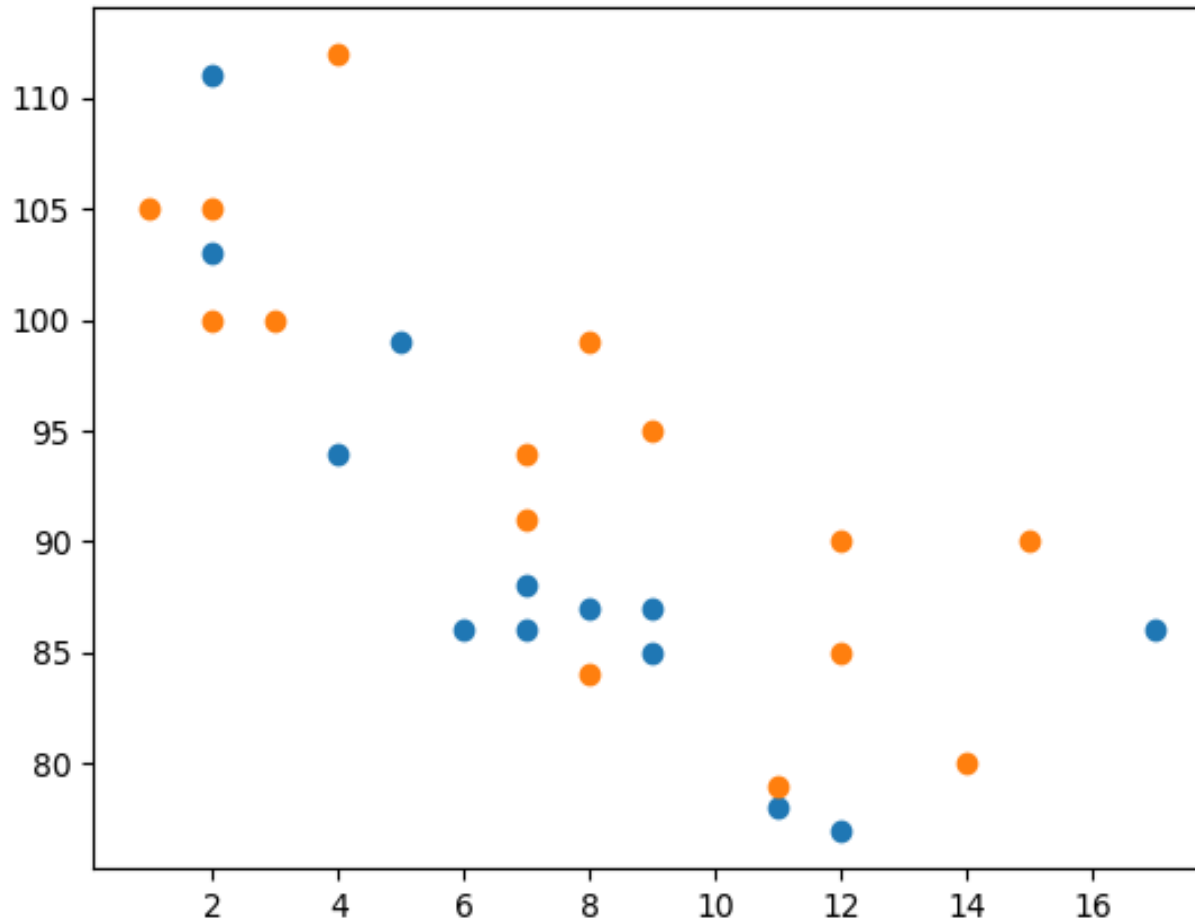
```
import matplotlib.pyplot as plt
import numpy as np

#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([100,105,84,105,90,99,90,95,94,100,79,112,9
1,80,85])
plt.scatter(x, y)

plt.show()
```

# RESULT





## NOTE

- The two plots are plotted with two different colors, by default blue and orange, you will learn how to change colors later in this chapter.
- 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:

- **Example**

- Set your own color of the markers:

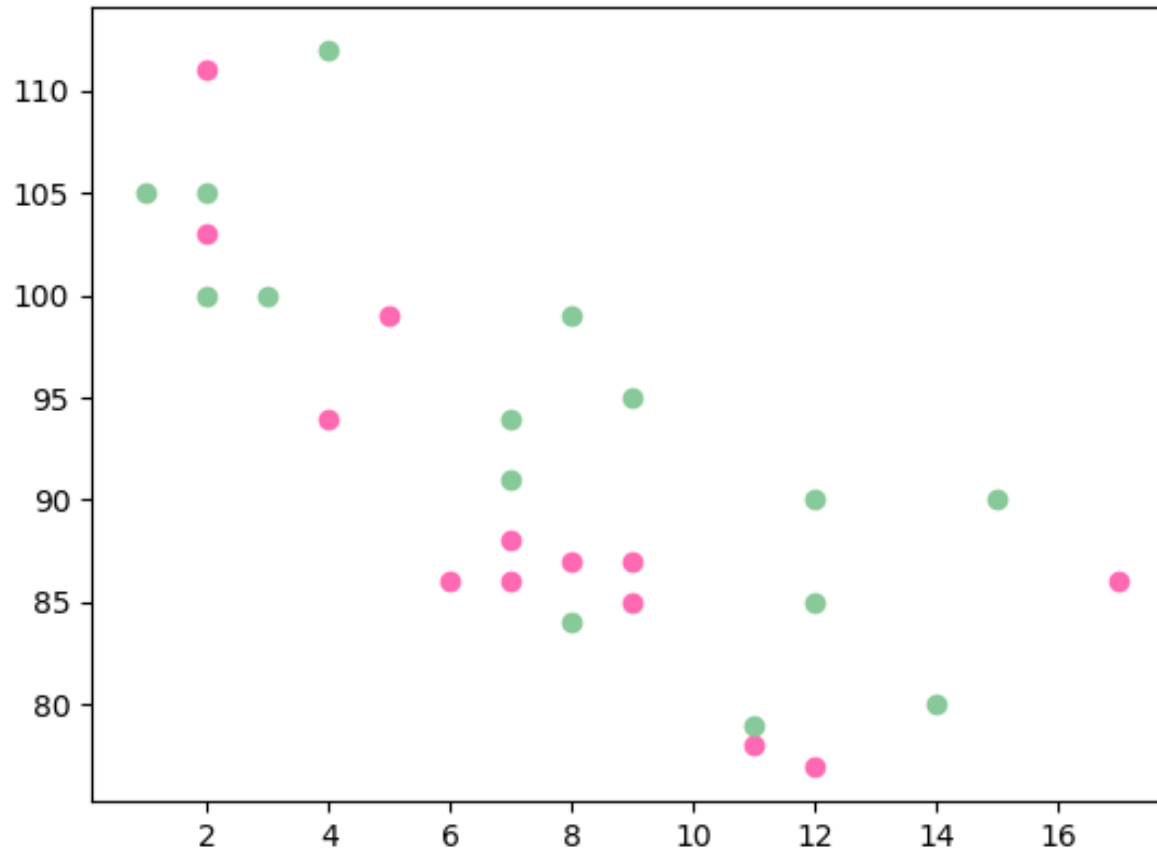
```
import matplotlib.pyplot as plt
import numpy as np
```

```
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([100,105,84,105,90,99,90,95,94,100,79,112,91,80,8
5])
plt.scatter(x, y, color = '#88c999')
```

```
plt.show()
```

# RESULT



# 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:

## Note:

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

## EXAMPLE

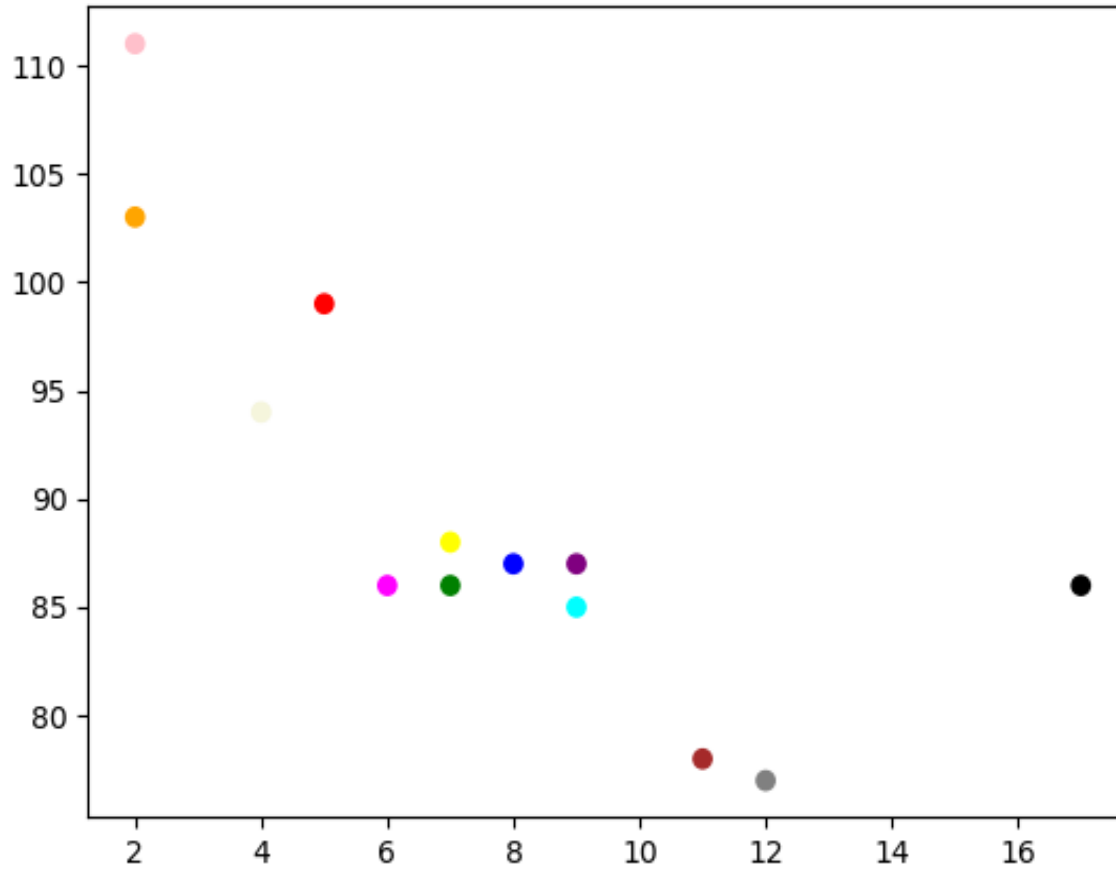
- Set your own color of the markers:

```
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","brown","gray","cyan",
"magenta"])
```

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

# RESULT



# COLOR MAP

- 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, and 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 of the point in the scatter plot:

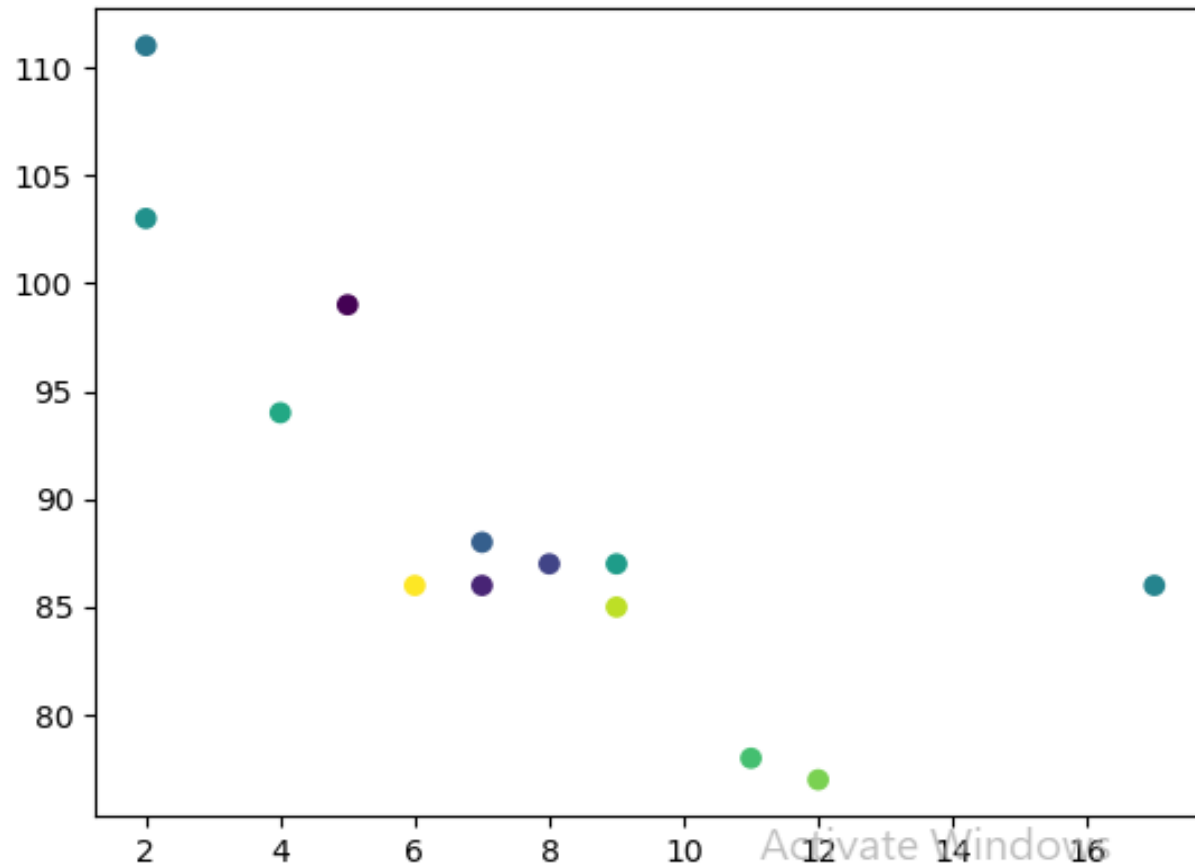


## EXAMPLE

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

# RESULT



➤ YOU CAN INCLUDE THE COLORMAP IN  
THE DRAWING BY INCLUDING  
THE `PLT.COLORBAR()` STATEMENT:

## EXAMPLE

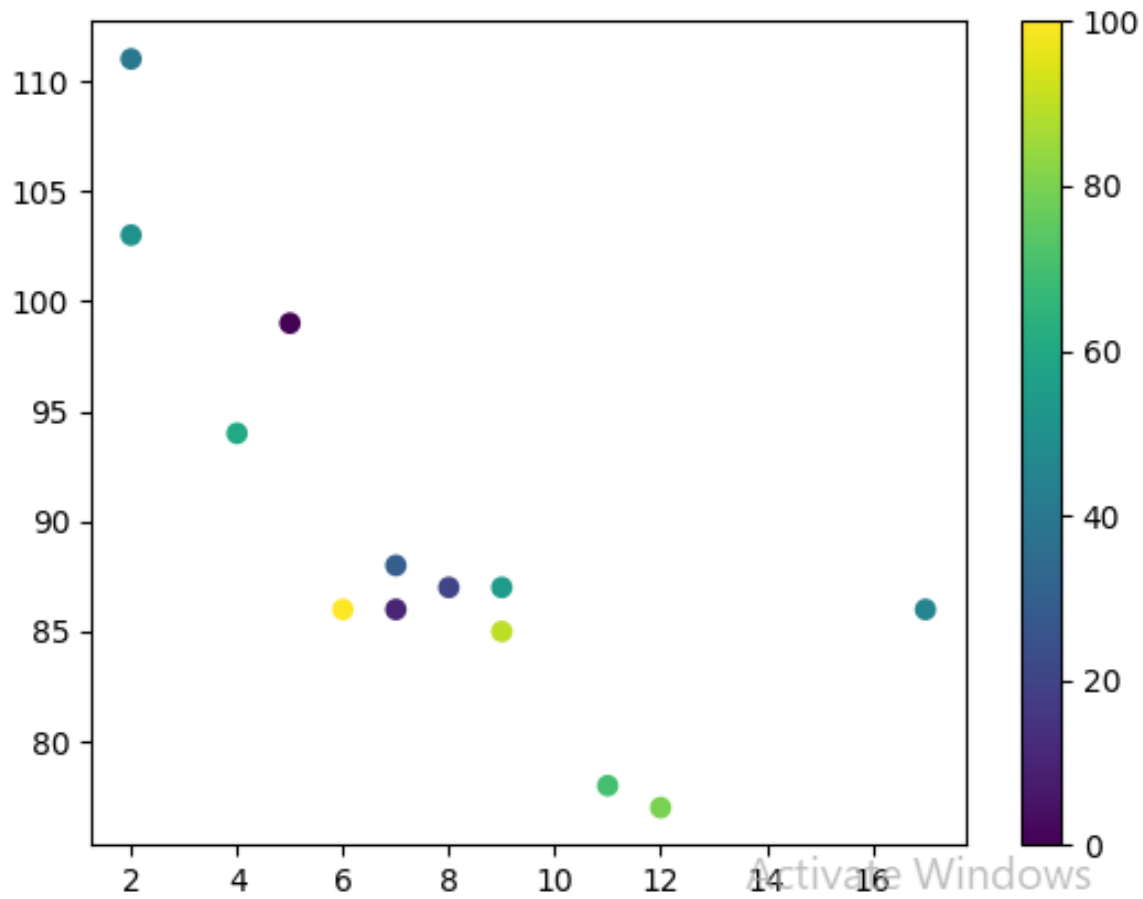
- Include the actual colormap:

```
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([99,86,87,88,111,86,103,87,94,78,77,85,8
6])
colors =
np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 9
0, 100])
```

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

# RESULT



## 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:

## EXAMPLE

- Set your own size for the markers:

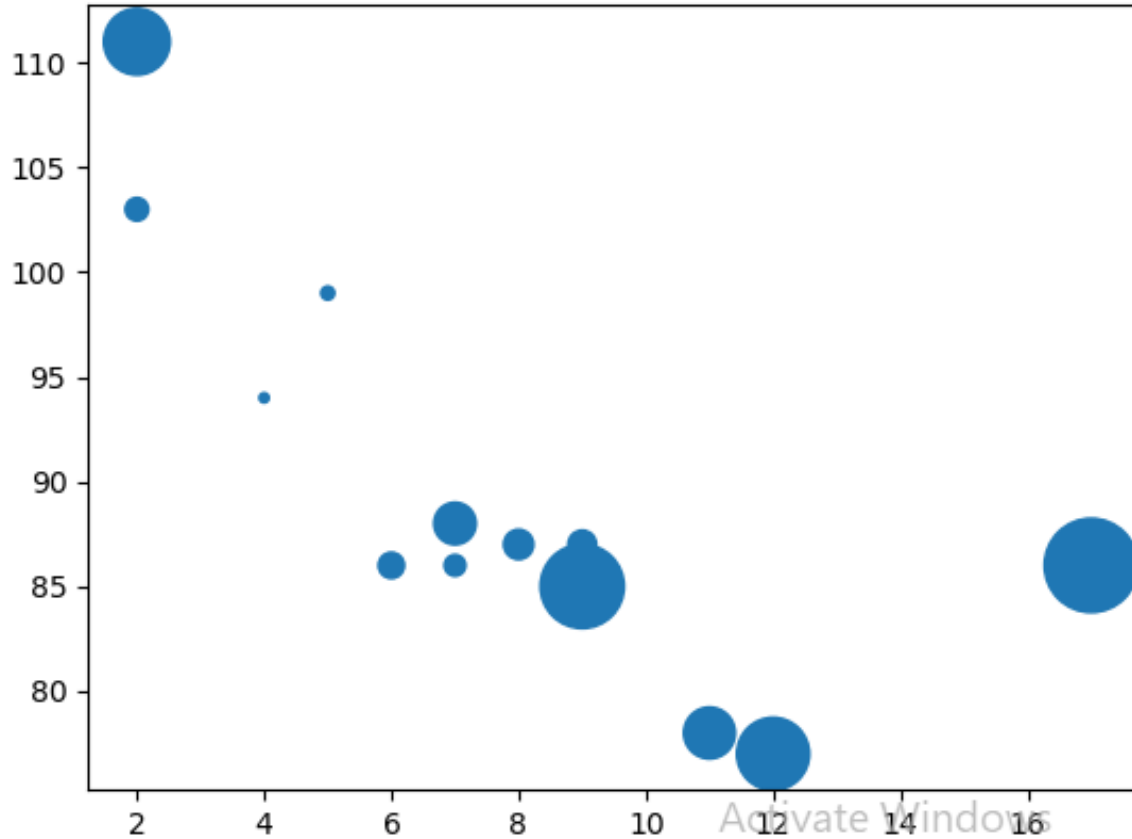
```
import matplotlib.pyplot as plt
import numpy as np
```

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

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

plt.show()
```

# RESULT





# 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:

## EXAMPLE

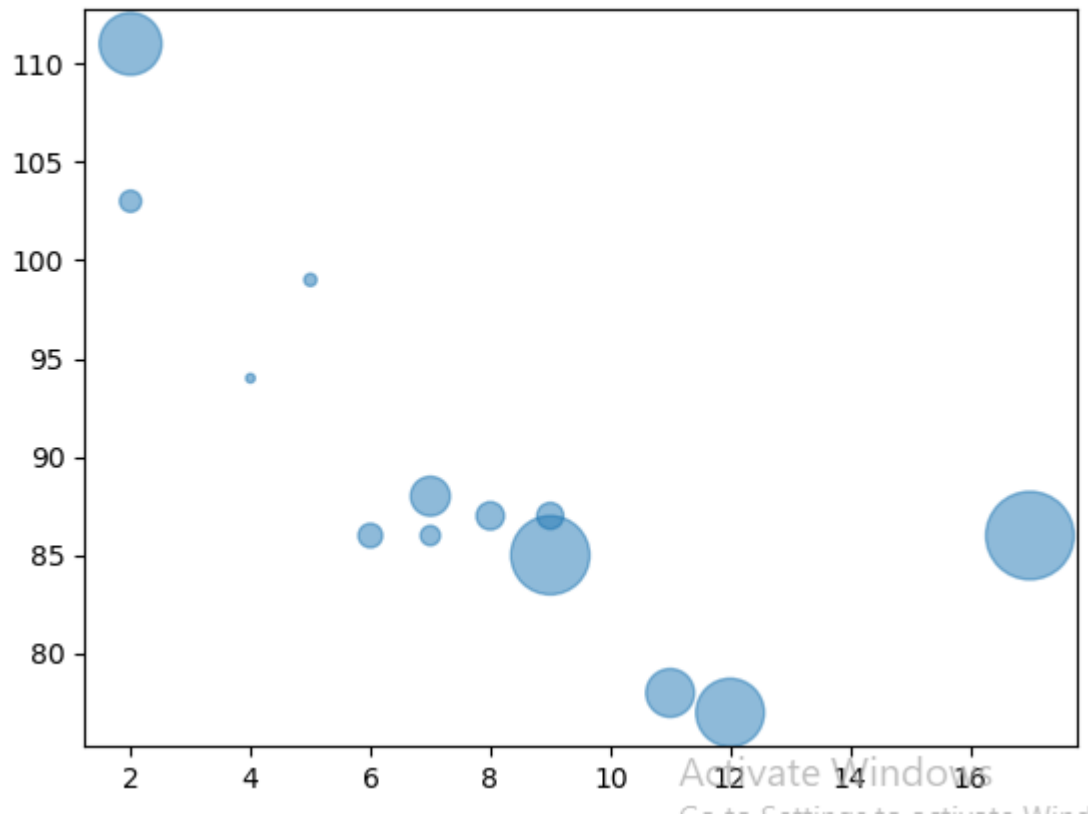
- Set your own size for the markers:

```
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()
```

# RESULT



## COMBINE COLOR SIZE AND ALPHA

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

## EXAMPLE

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

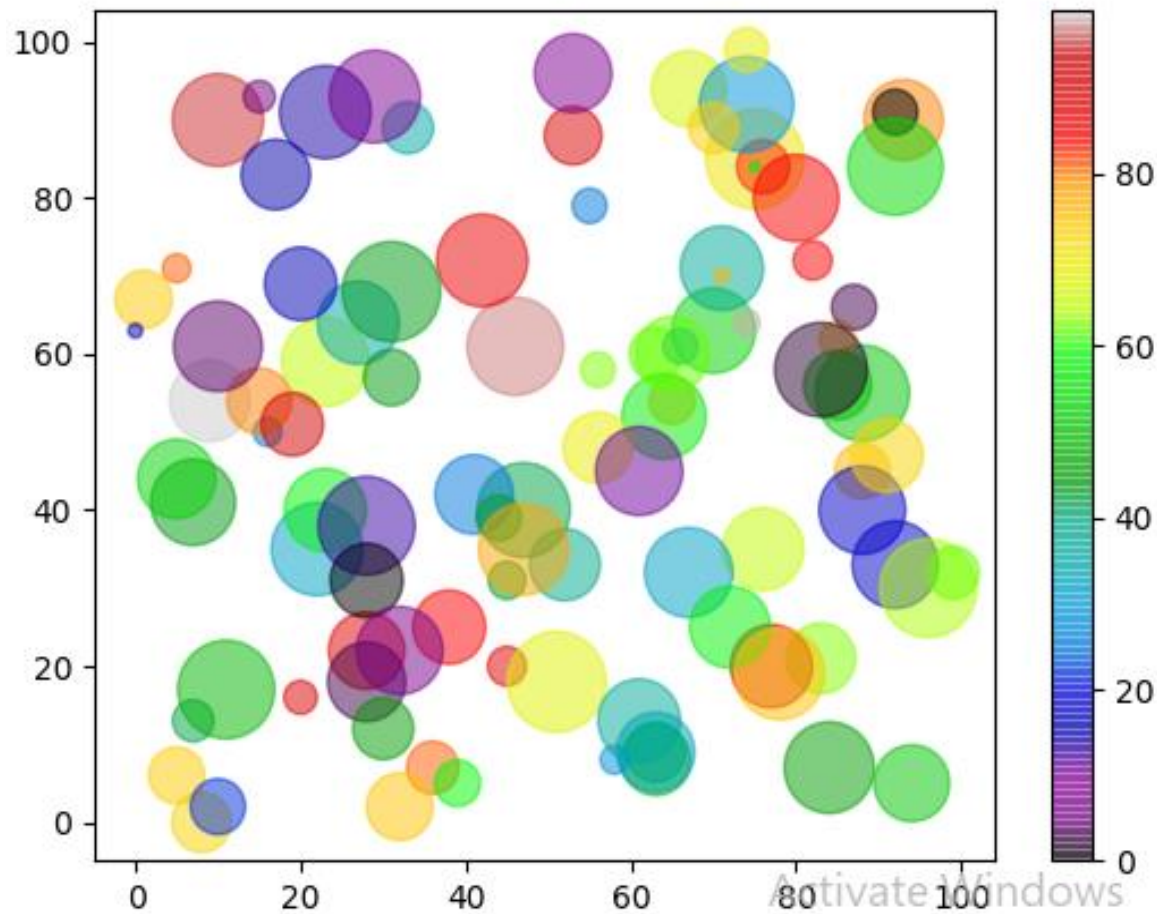
```
import matplotlib.pyplot as plt
import numpy as np
```

```
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()
```

# RESULT



# MATPLOTLIB BARS

## Creating Bars

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

## Example

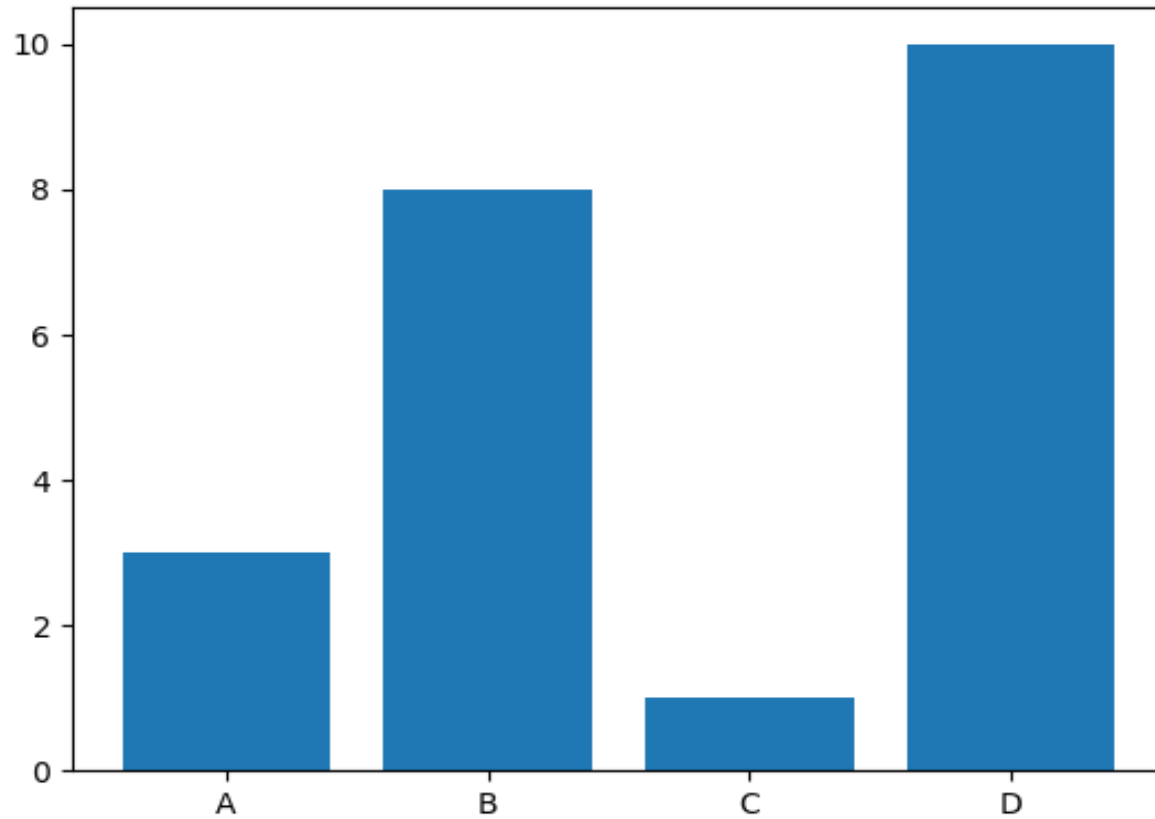
- Draw 4 bars:

```
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()
```

# RESULT



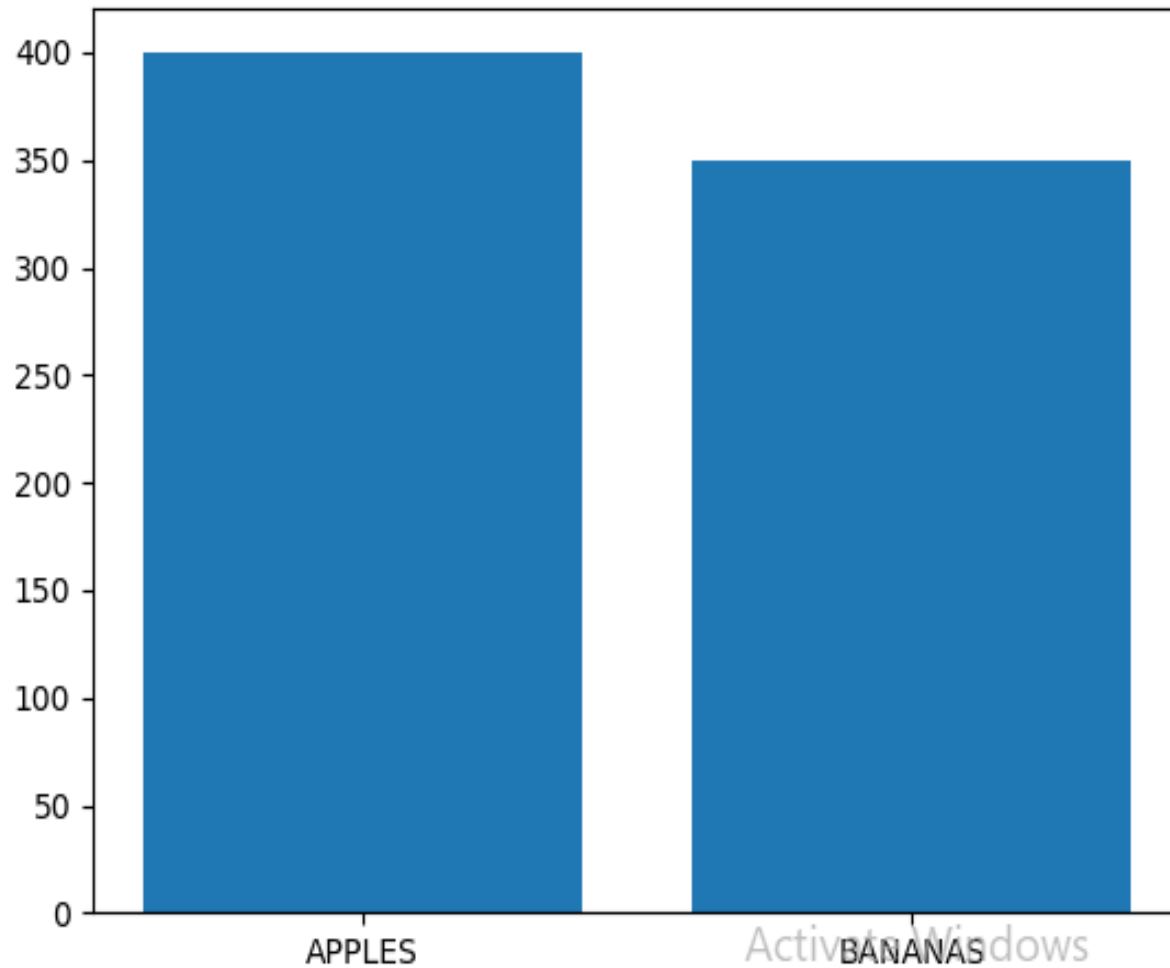


➤ THE `BAR()` FUNCTION TAKES ARGUMENTS THAT DESCRIBES THE LAYOUT OF THE BARS. THE CATEGORIES AND THEIR VALUES REPRESENTED BY THE *FIRST* AND *SECOND* ARGUMENT AS ARRAYS.

## EXAMPLE

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

# RESULT



# HORIZONTAL BARS

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

## EXAMPLE

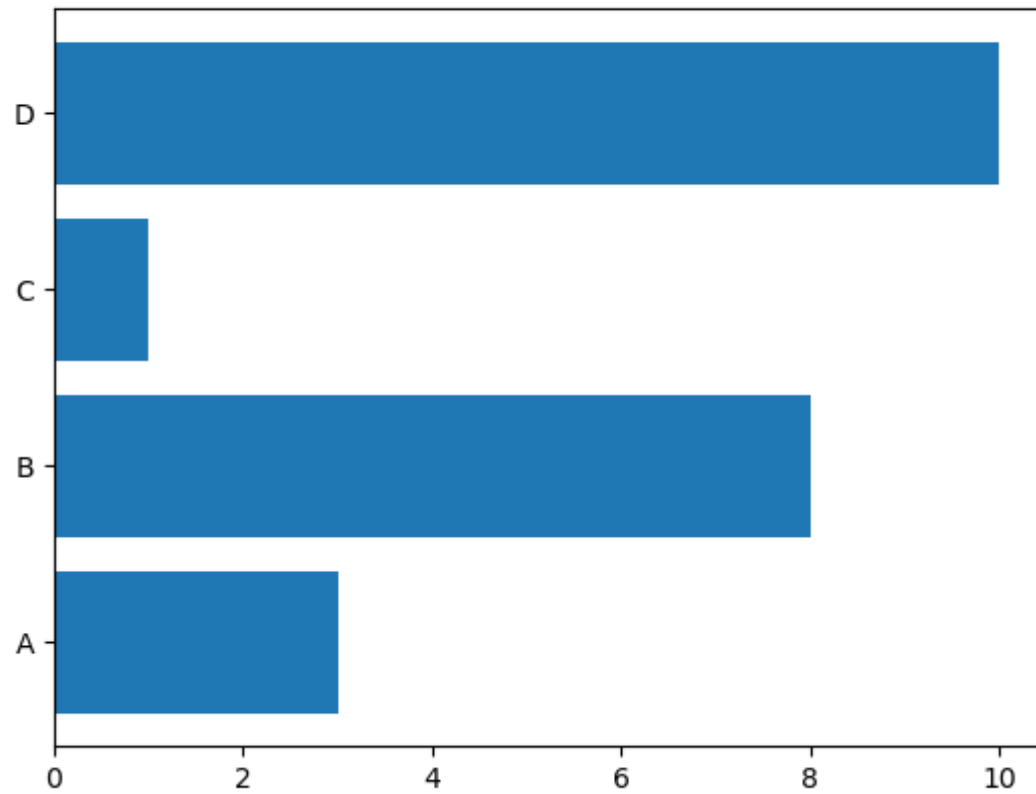
- Draw 4 horizontal bars:

```
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()
```

# RESULT



# BAR COLOR

- The **bar()** and **barh()** takes the keyword argument **color** to set the color of the bars:

## Example

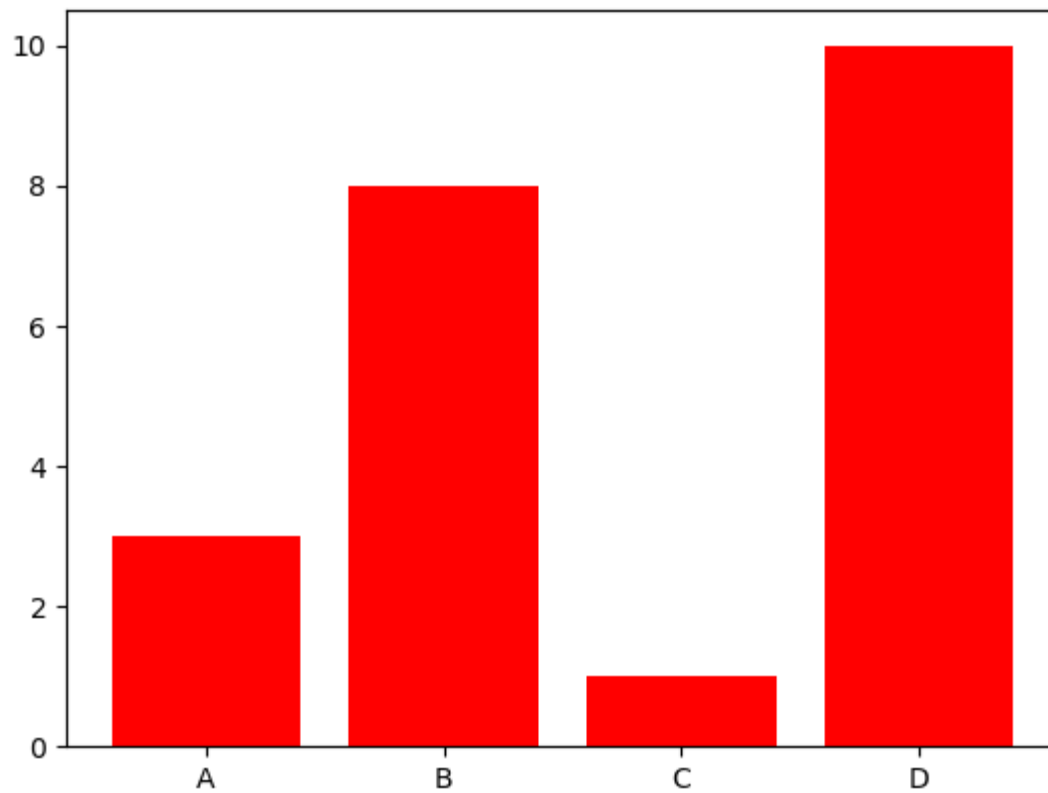
Draw 4 red bars:

```
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()
```

# RESULT





# COLOR NAMES

- You can use any of the 140 supported color names.

Example

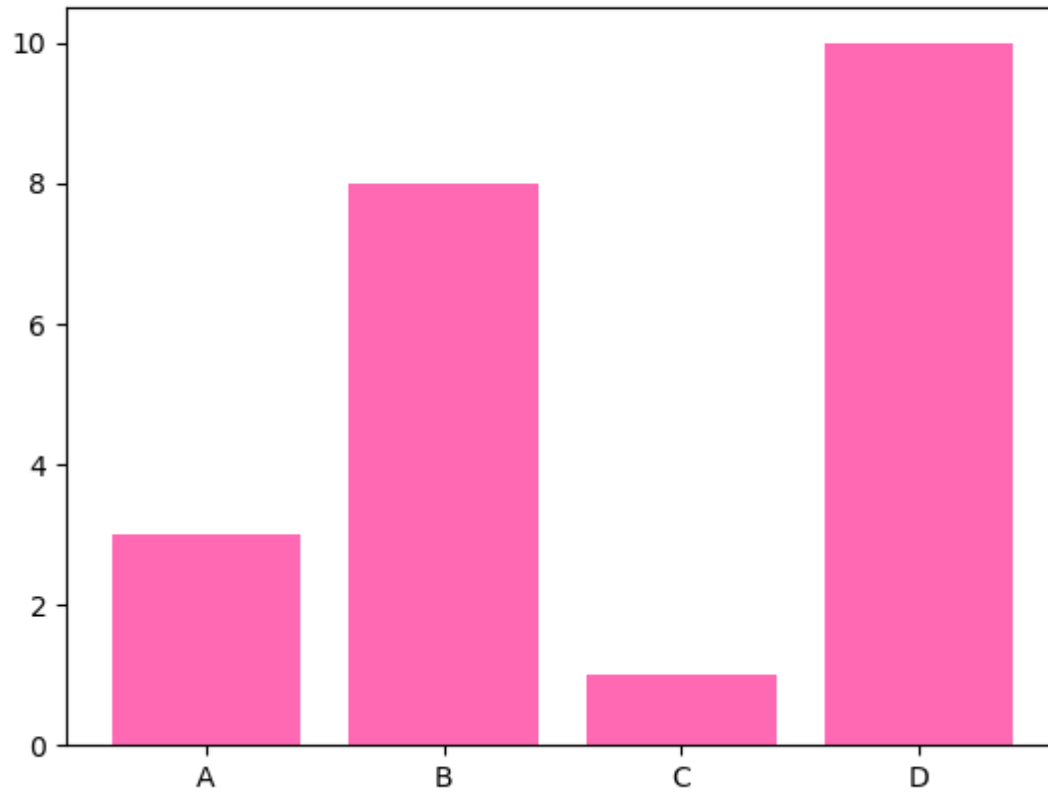
Draw 4 "hot pink" bars:

```
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()
```

# RESULT



# COLOR HEX

➤ Or you can use Hexadecimal color values:

## Example

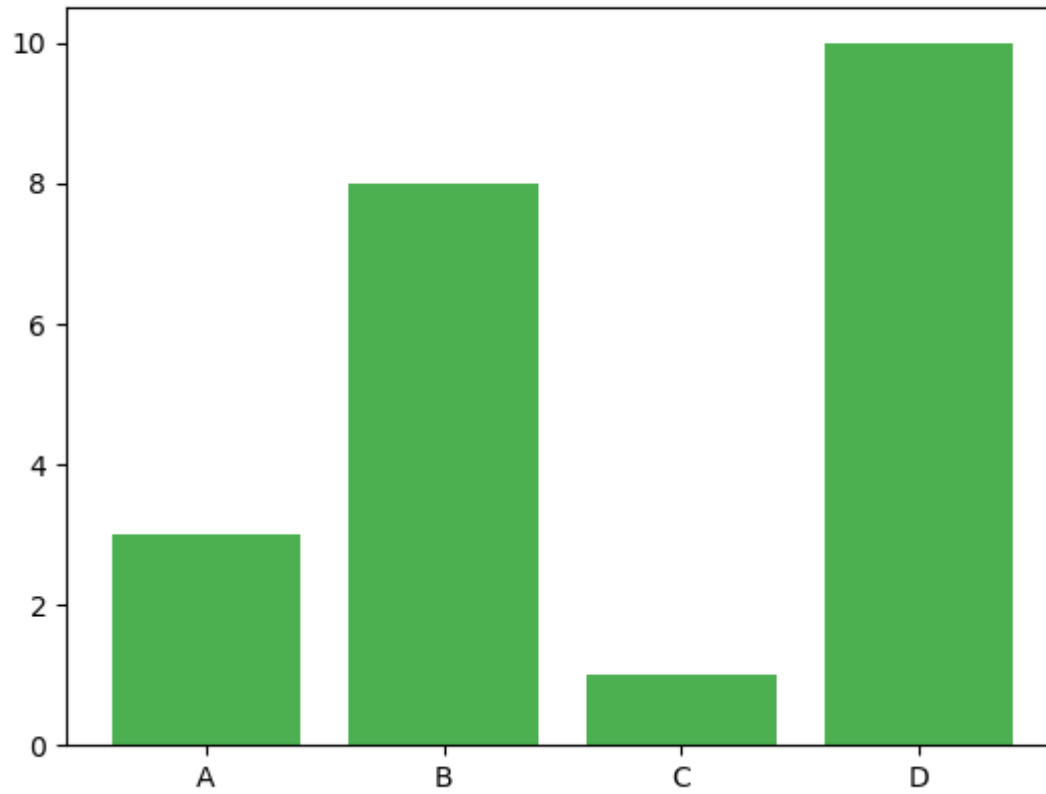
Draw 4 bars with a beautiful green color:

```
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()
```

# RESULT



# BAR WIDTH

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

## Example

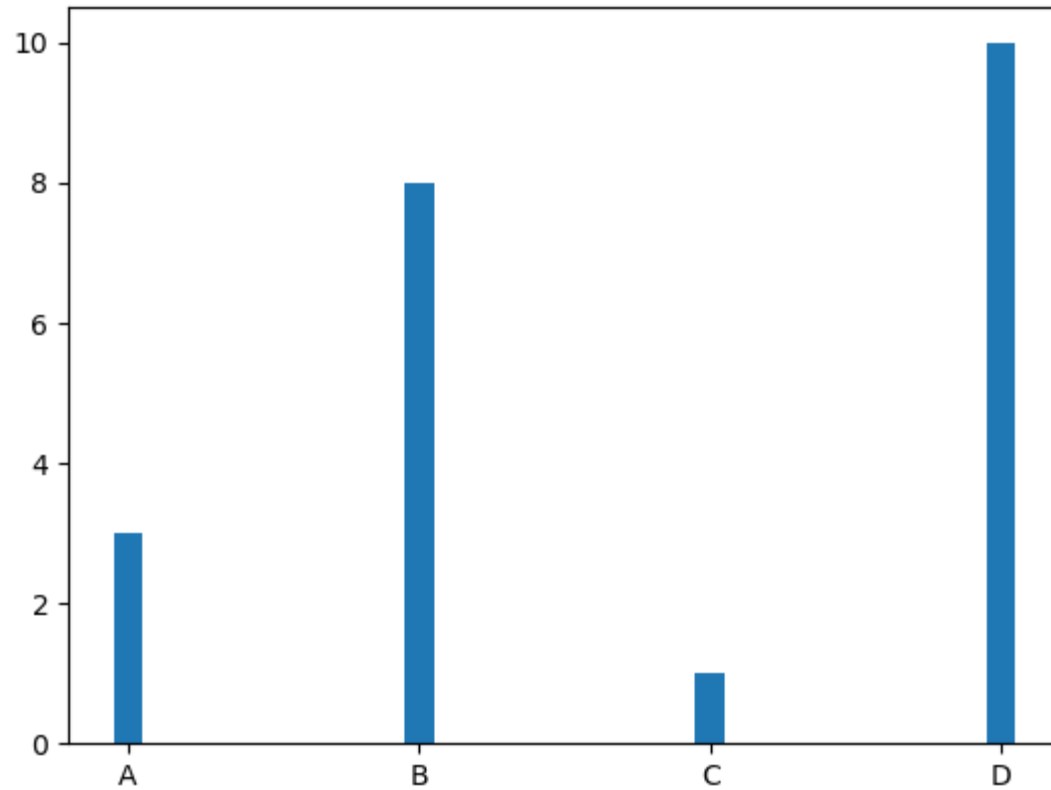
- Draw 4 very thin bars:

```
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()
```

## Default



The default width value is 0.8

## NOTE

- For horizontal bars, use height instead of width.

Files

sample\_data

Company.csv

Disk 69.93 GB available

```
[] real_x = data.iloc[:,0].values
real_x

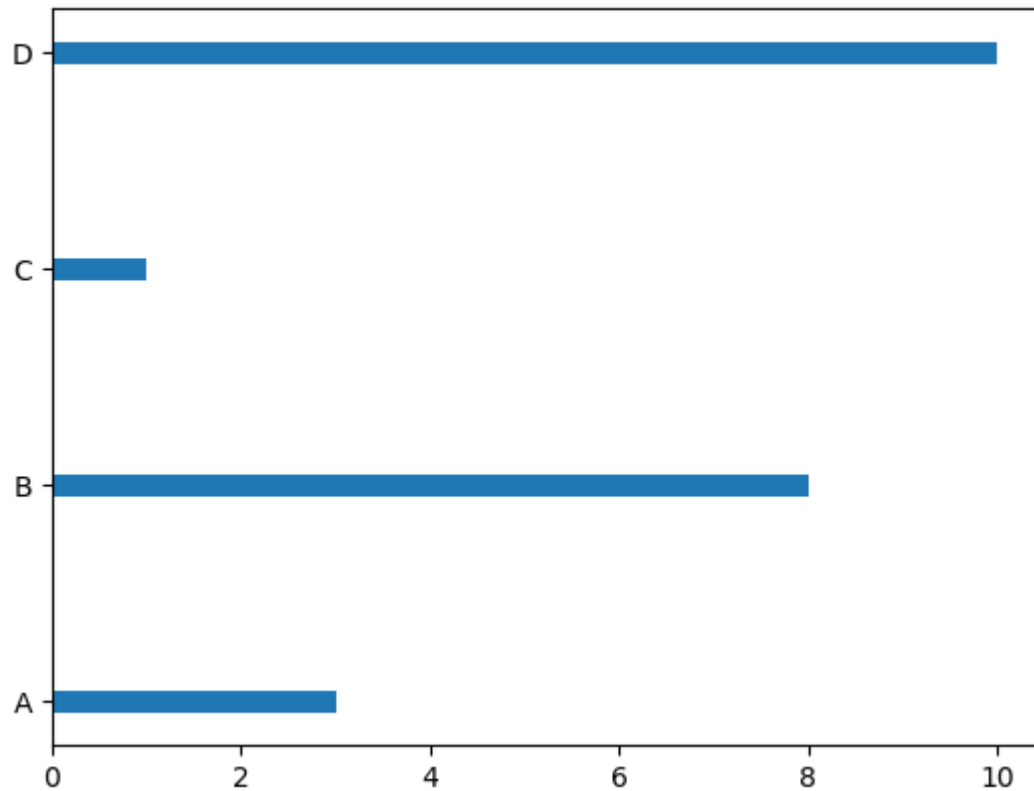
array([1.1, 1.3, 1.5, 2. , 2.2, 2.9, 3. , 3.2, 3.2, 3.7, 3.9,
 4. , 4. , 4.1, 4.5, 4.9, 5.1, 5.3, 5.9, 6. , 6.8, 7.1,
 7.9, 8.2, 8.7, 9. , 9.5, 9.6, 10.3, 10.5])

[] real_x=real_x.reshape(-1,1)
real_x

array([[1.1],
 [1.3],
 [1.5],
 [2.],
 [2.2],
 [2.9],
 [3.],
 [3.2],
 [3.2],
 [3.7],
 [3.9],
 [4.],
 [4.],
 [4.1],
 [4.5],
 [4.9],
 [5.1],
 [5.3],
 [5.9],
 [6.],
 [6.8],
 [7.1],
 [7.9],
 [8.2],
 [8.7],
 [9.],
 [9.5],
 [9.6],
 [10.3],
 [10.5]])
```



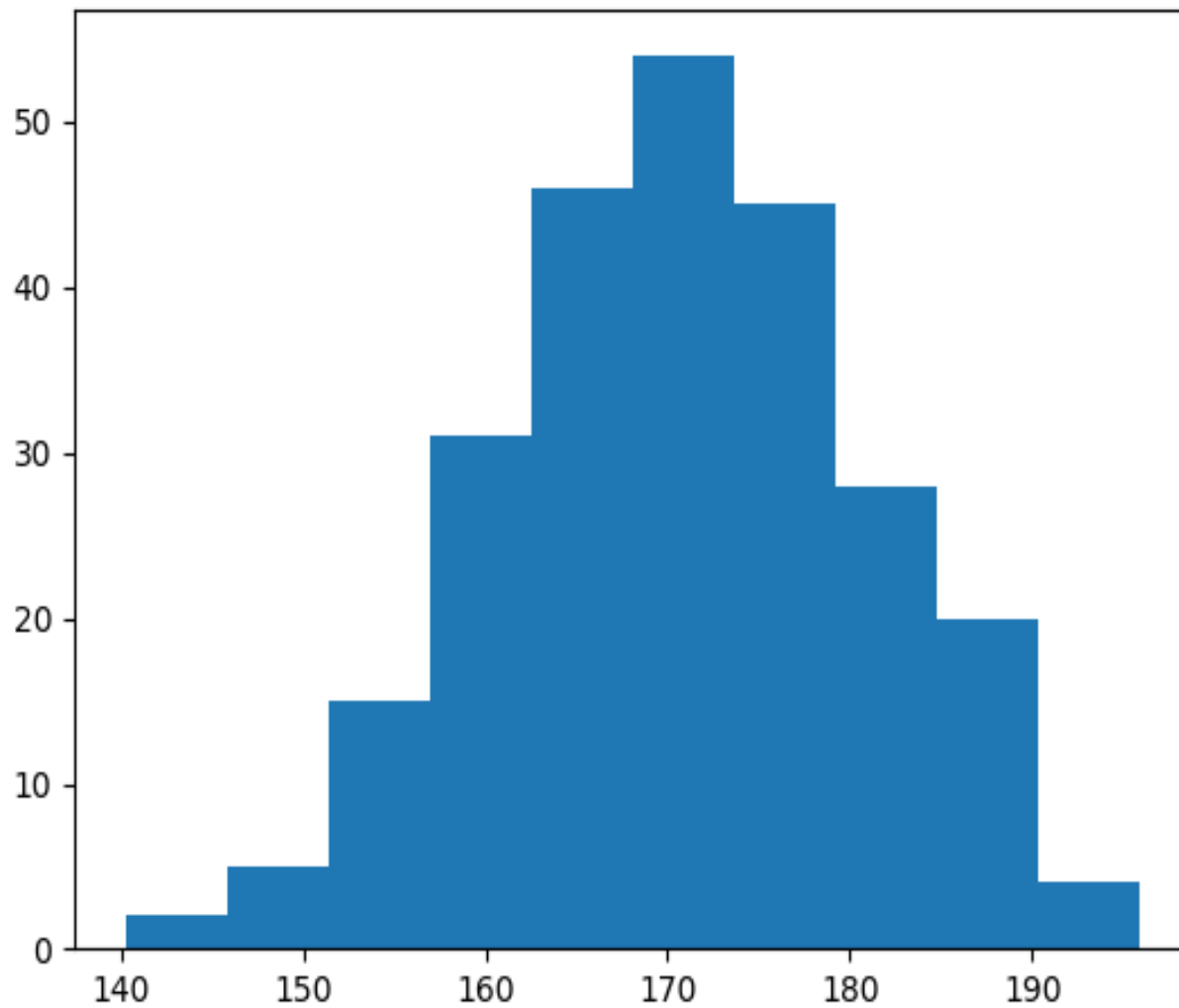
# RESULT



The default height value is 0.8

# MATPLOTLIB HISTOGRAMS

- Histogram
- A histogram is a graph showing *frequency* distributions.
- It is a graph showing the number of observations within each given interval.
- Example: Say you ask for the height of 250 people, you might end up with a histogram like this:



## YOU CAN READ FROM THE HISTOGRAM THAT THERE ARE APPROXIMATELY:

- 2 people from 140 to 145cm
- 5 people from 145 to 150cm
- 15 people from 151 to 156cm
- 31 people from 157 to 162cm
- 46 people from 163 to 168cm
- 53 people from 168 to 173cm
- 45 people from 173 to 178cm
- 28 people from 179 to 184cm
- 21 people from 185 to 190cm
- 4 people from 190 to 195cm

# CREATE HISTOGRAM

- In Matplotlib, we use the `hist()` function to create histograms.
- The `hist()` function will use an array of numbers to create a histogram, the array is sent into the function as an argument.
- 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. Learn more about [Normal Data Distribution](#) in our [Machine Learning Tutorial](#).

## EXAMPLE

- A Normal Data Distribution by NumPy:

```
import numpy as np
x = np.random.normal(170, 10, 250)
print(x)
```

# RESULT

```
[162.75967739 167.16615502 156.90278097 158.37683878 169.35497389
171.27104589 172.07514066 155.64540933 171.85702826 155.95220143
169.62077721 156.77653396 171.68215965 174.69214325 167.10123891
174.10290247 183.64738602 162.02658768 175.92083941 170.06006115
150.88992415 181.25423383 162.22870993 171.42244885 166.2789325
177.04392063 174.43295632 186.58362923 182.22324188 163.85183445
181.40074724 161.26403847 161.96105405 157.73800235 176.7960842
161.72878387 180.85454475 150.19914929 159.67373109 168.91899711
166.93992106 178.78059503 171.67455144 172.31356972 175.60220077
161.96919219 166.95418229 158.81259899 166.95191916 161.90490279
174.39196338 181.87940083 162.32508549 181.1592168 169.98334175
181.20583806 159.64875433 167.37244343 184.03026475 175.74427237
174.84827266 167.30343291 167.40246992 177.28271663 160.46469111
183.20865711 175.15235757 174.49666228 178.77333657 167.40459998
173.15061073 166.41925169 167.71830765 169.9051103 165.93278748
173.0218388 160.14066368 175.08291809 159.77349393 178.71234545
165.47906339 160.18408343 189.80623563 165.7280748 179.05639205
183.13656973 174.45368035 166.12490101 173.12621207 169.74763656
168.96445419 173.57190205 189.49145873 163.92794158 179.85507386
158.7147013 173.23618407 179.20953651 163.11438249 180.49574395
173.59730523 182.75853518 177.60525261 182.58002439 179.4391518
```

# THE HIST() FUNCTION WILL READ THE ARRAY AND PRODUCE A HISTOGRAM

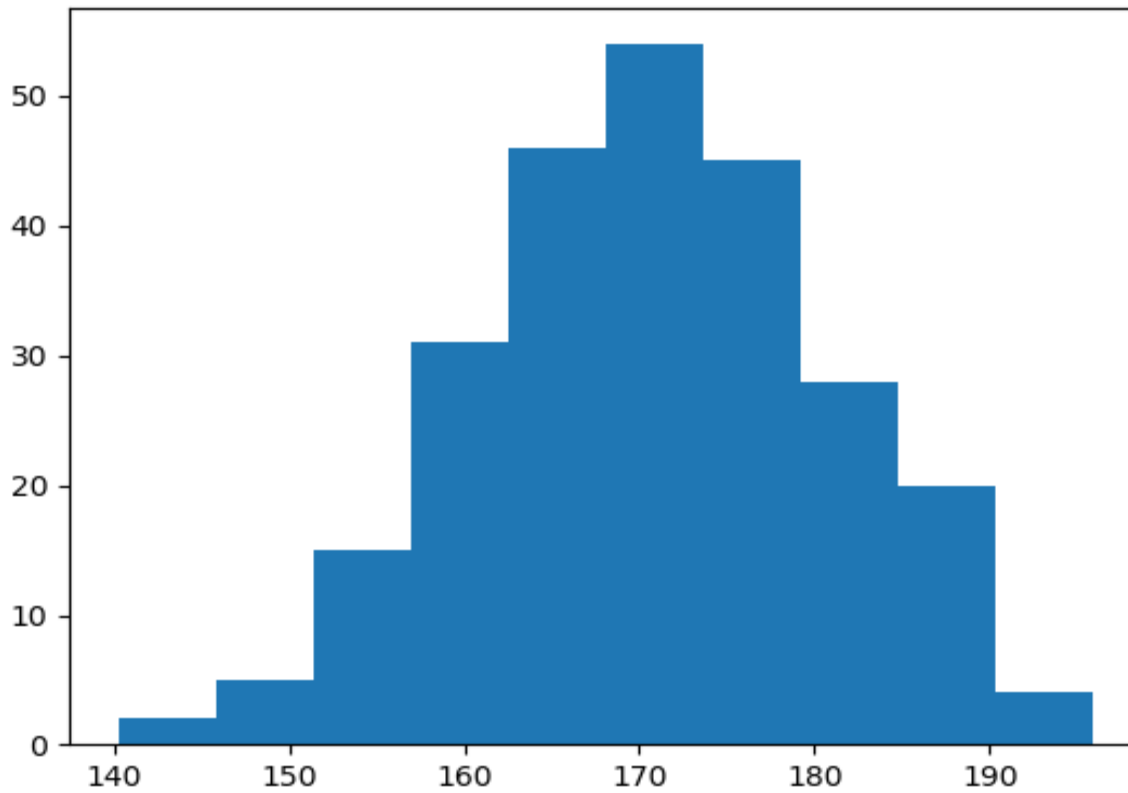
## Example

- A simple histogram:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```



# RESULT



# MATPLOTLIB PIE CHARTS

- Creating Pie Charts
- With Pyplot, you can use the `pie()` function to draw pie charts:

## Example

- A simple pie chart:
- `import matplotlib.pyplot as plt`  
`import numpy as np`

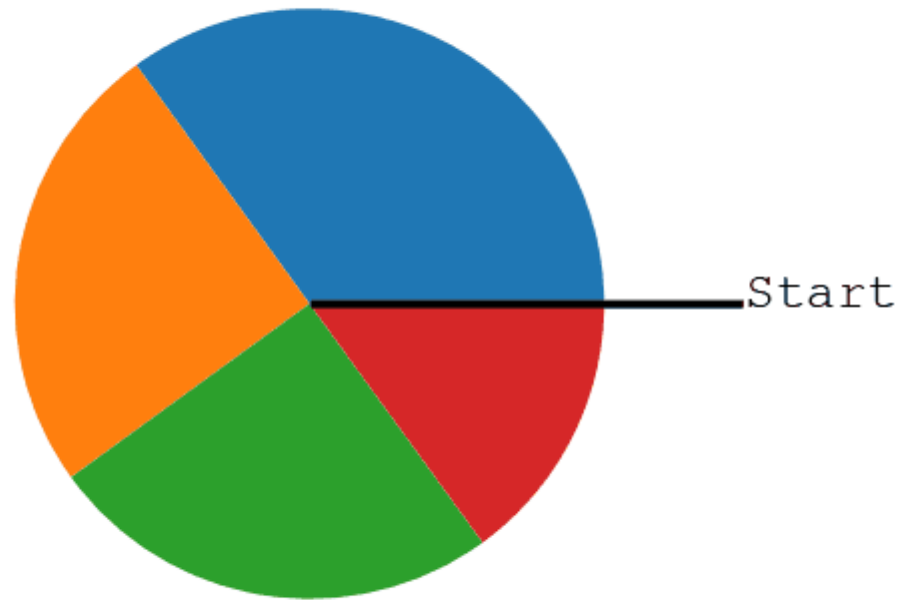
```
y = np.array([35, 25, 25, 15])
```

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

# RESULT



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  
MOVE *COUNTERCLOCKWISE*



# LABEL

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

## Example

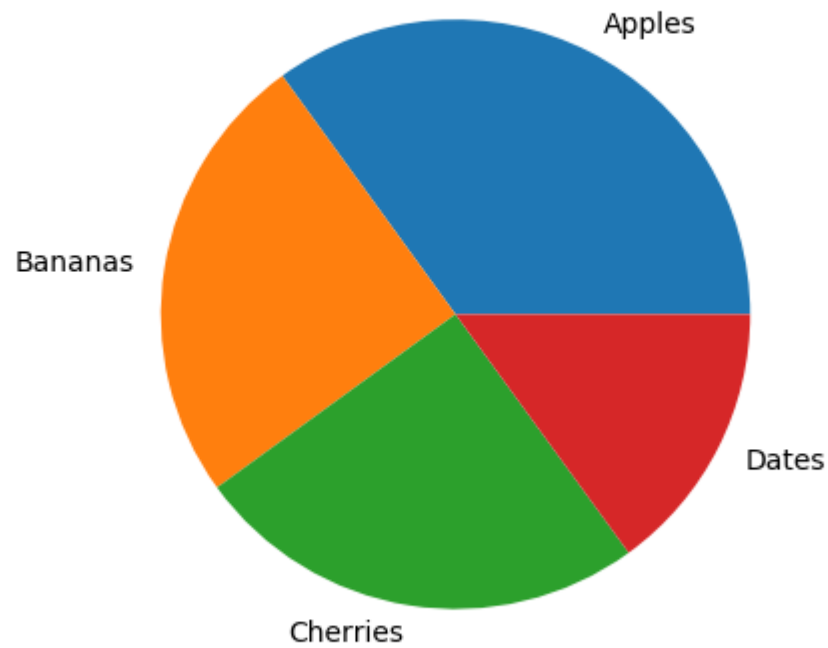
- A simple pie chart:

```
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()
```

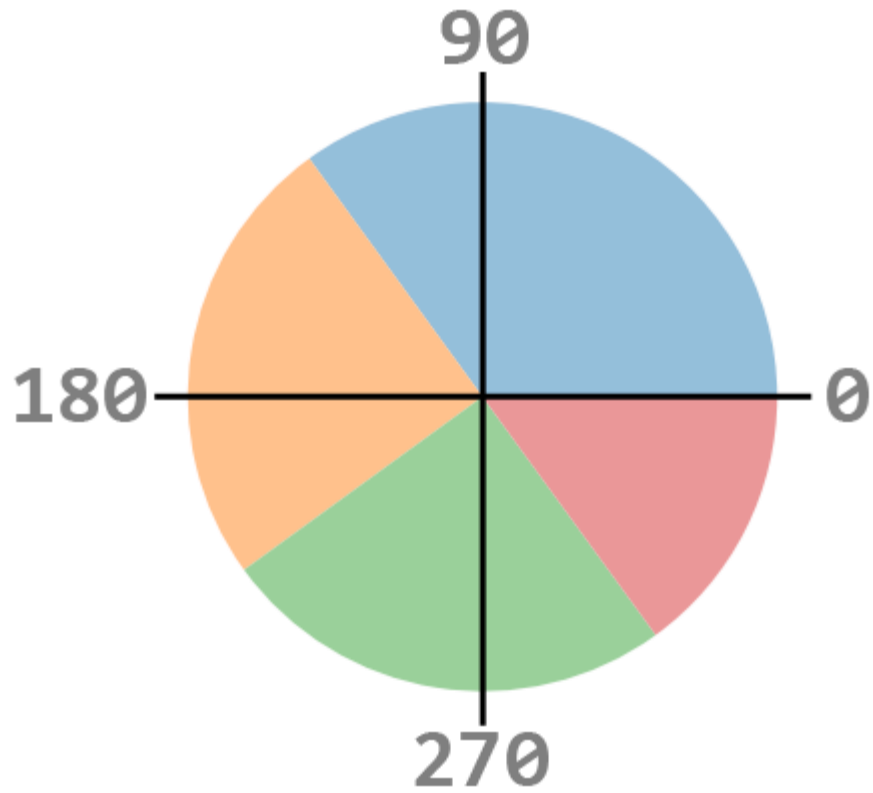
# RESULT



# 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:





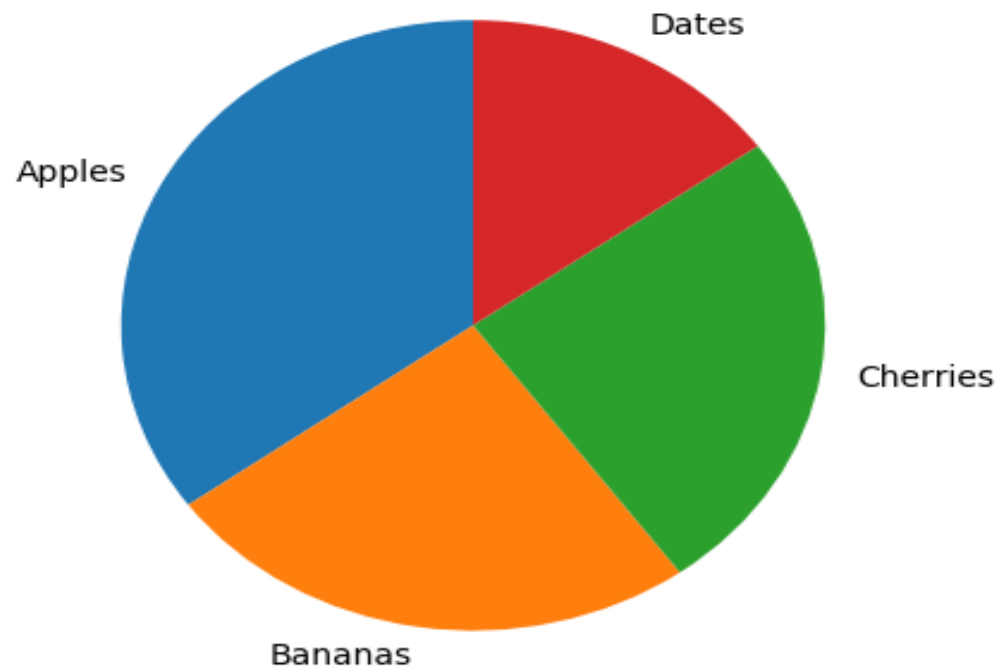
## EXAMPLE

- Start the first wedge at 90 degrees:

```
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:

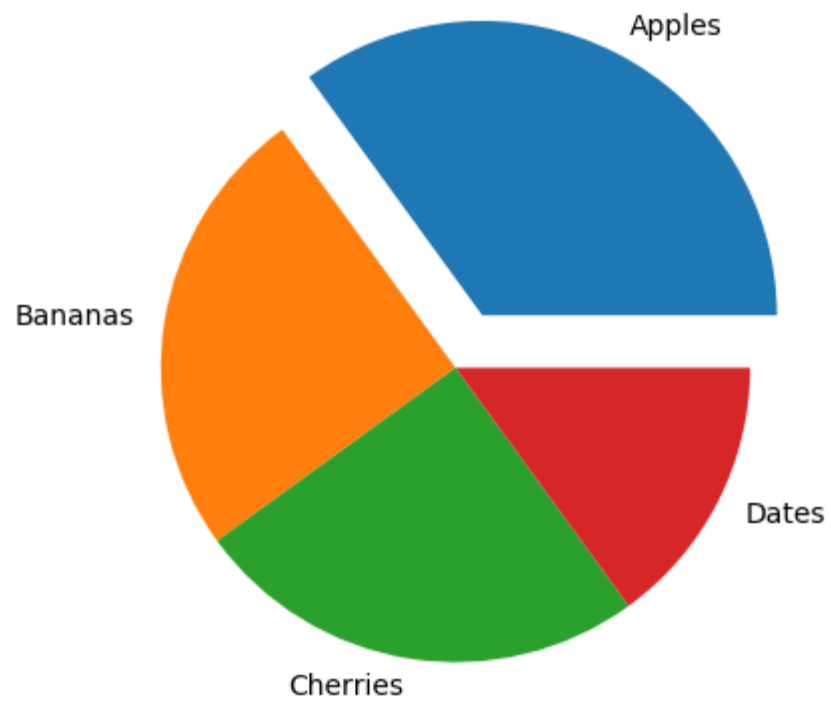
## EXAMPLE

- Pull the "Apples" wedge 0.2 from the center of the pie:

```
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:

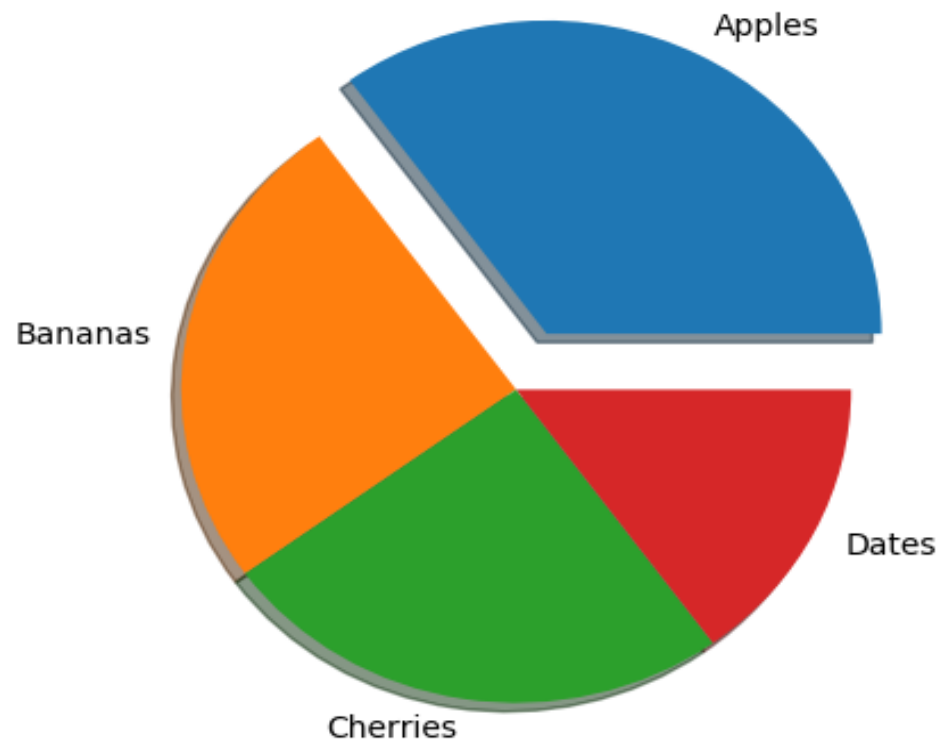
## Example

- Add a shadow:

```
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:

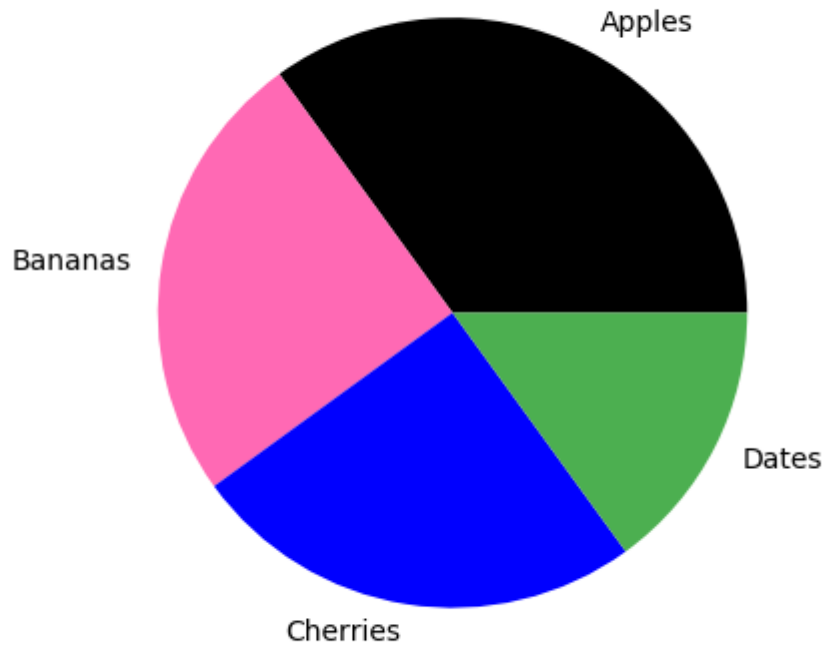
## Example

- Specify a new color for each wedge:

```
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()
```



YOU CAN USE HEXADECIMAL COLOR VALUES,  
ANY OF THE 140 SUPPORTED COLOR NAMES, OR  
ONE OF THESE SHORTCUTS:

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

# LEGAND

- To add a list of explanation for each wedge, use the `legend()` function:

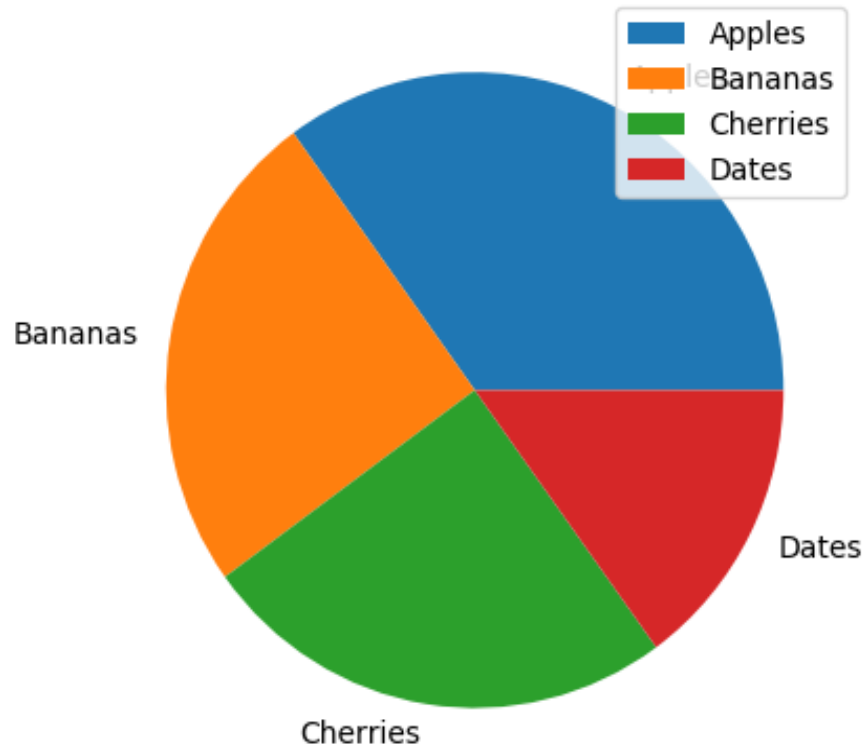
## Example

- Add a legend:

```
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.

## Example

- Add a legend with a header:

```
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()
```

