Numpy and Matplotlib

Objective: Introduce students to Numpy and Matplotlib for numerical computing and data visualization. Perform mathematical operations using Numpy arrays. Visualize data using Matplotlib's plotting functions.

What is NumPy?

NumPy **(Numerical Python)** is a fundamental l**ibrary in Python** for **numerical computations**. It provides support for arrays, matrices, and many mathematical functions to operate on these data structures efficiently.

- An Array is the fundamental object
- Array are used to store **homogenous data elements** in a contiguous block of memory

Why Use NumPy?

In Python we have **lists** that serve the purpose of arrays, but they are **slow to process.**

NumPy aims to provide an **array object** that is up to 50x faster than **traditional Python lists**.

The array object in NumPy is called **ndarray**, it provides a lot of **supporting functions** that make working with ndarray very easy.

NumPy Arrays vs. Python Lists

1. Data type storage
2. Importing module
3. numerical operation
4. consume less memory
5. fast as compared to the python list

Import NumPy

import it in your applications by adding the import keyword:

```
import numpy
```

Now NumPy is imported and ready to use.

```
import numpy
#numpy.array([1, 2, 3, 4, 5]) creates a NumPy array from the list [1, 2, 3, 
arr = numpy.array([1, 2.0, 3, 4, 5])


print(arr)


print(arr)
```

```
[1. 2. 3. 4. 5.]
[1. 2. 3. 4. 5.]
```

NumPy as np

Using NumPy as np is a common convention in Python. When you import NumPy, you typically alias it as np to make the code shorter and more readable.

```
import numpy as n
```

Now the NumPy package can be referred to as np instead of numpy.

```
import numpy as n

arr = n.array([1, 2, 3, 4, 5])

print(arr)
```

```
[1 2 3 4 5]
```

Checking NumPy Version

```
import numpy as np

print(np.__version__)
```

```
1.26.4
```

Create a NumPy ndarray Object

numPy is used to work with arrays. The array object in NumPy is called ndarray.In NumPy, an ndarray (n-dimensional array) is the core data structure used for numerical computations. It is a grid of values, all of the same type, indexed by a tuple of non-negative integers

We can create a **NumPy ndarray objec**t by using the array() function.

```
import numpy as np
# Create a 1D array from a list
arr = np.array([1, 2, 3, 4, 5])

print(arr)
print("1D Array:", arr)


print(type(arr))
print("Type:", type(arr))
```

```
[1 2 3 4 5]
1D Array: [1 2 3 4 5]
<class 'numpy.ndarray'>
Type: <class 'numpy.ndarray'>
```

Creating a 2D Array A 2D array is a matrix with rows and columns.

```
 #Create a 2D array from a nested list
arr2 = n.array([[1, 2, 3], [4, 5, 6]])
print("2D Array:\n", arr2)
print("Shape:", arr2.shape)  # Output: (2, 3)
```

```
2D Array:
 [[1 2 3]
 [4 5 6]]
Shape: (2, 3)
```

Creating a 3D Array A 3D array is an array of matrices.

The shape attribute of a NumPy array returns a tuple representing the dimensions of the array.

For arr3, the shape is (2, 2, 2), which means:

The first dimension has 2 elements (two 2D arrays).

The second dimension has 2 elements (rows in each 2D array).

The third dimension has 2 elements (columns in each 2D array).

```
# Create a 3D array from a nested list
arr3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("3D Array:\n", arr3)
print("Shape:", arr3.shape)  # Output: (2, 2, 2)
```

```
3D Array:
 [[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
Shape: (2, 2, 2)
```

Creating Arrays with Special Functions

```
numpy.ones_arr = n.ones((3, 3))  # Create a 3x3 array of zeros
print("Zeros Array:\n", numpy.ones_arr)
```

```
Zeros Array:
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

NumPy Array Indexing

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has **index 0**, and the second has index 1 etc.

```
import numpy as n

arr = n.array([1, 2, 3, 4])

print(arr[2])
```

```
3
```

```
#Get the second element from the following array.
import numpy as np

arr = np.array([1, 2, 3, 4])
```

```
5
6 print(arr[1])
```

```
2
```

```
1 #Get third and fourth elements from the following array and add them.
2
3 import numpy as np
4
5 arr = np.array([1, 2, 3, 4])
6
7 print(arr[0] + arr[3])
```

```
5
```

Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

```
1 import numpy as np
2
3 arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
4
5 print('2nd element on 1st row: ', arr[0, 1])
```

```
2nd element on 1st row:  2
```

Matplotlib

Matplotlib is a 2D plotting library for Python.It supports a wide variety of plots, including line plots, bar charts, scatter plots, histograms, and more.

```
1 #Matplotlib is a low level graph plotting library in python that serves as a
2 import matplotlib
```

```
1 import matplotlib
2
```

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

```
3.7.1
```

Pyplot

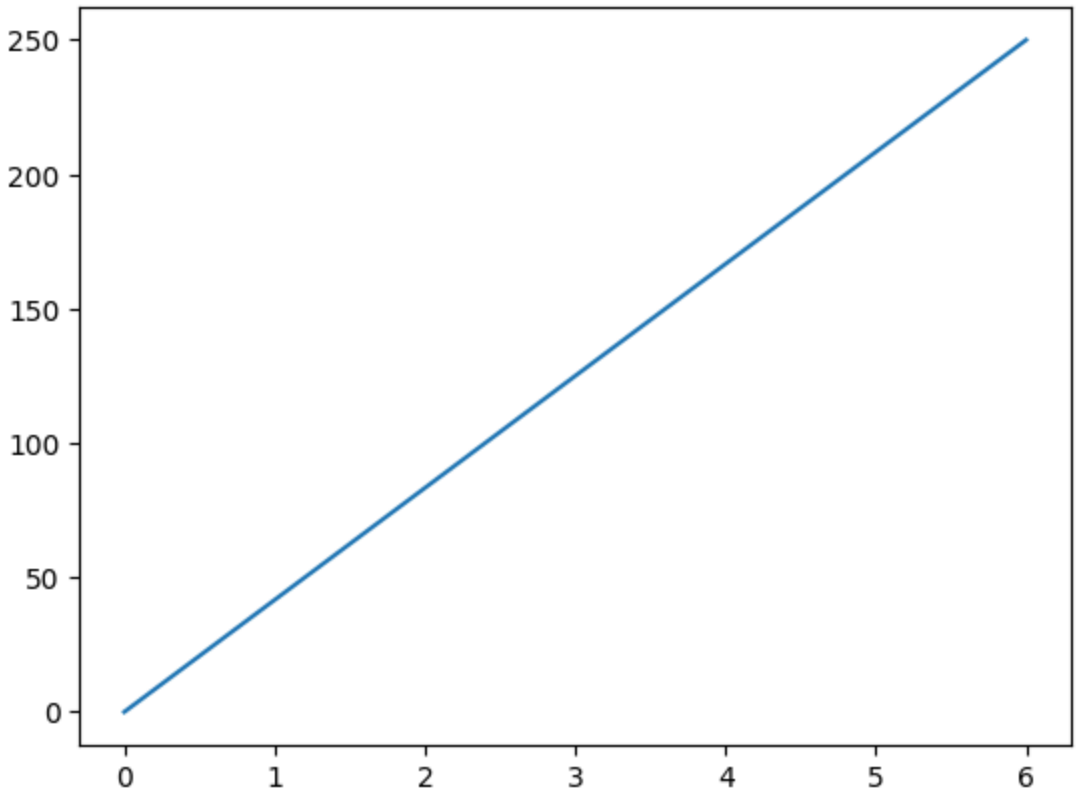
Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

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

Now the Pyplot package can be referred to as plt.

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()
```



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.

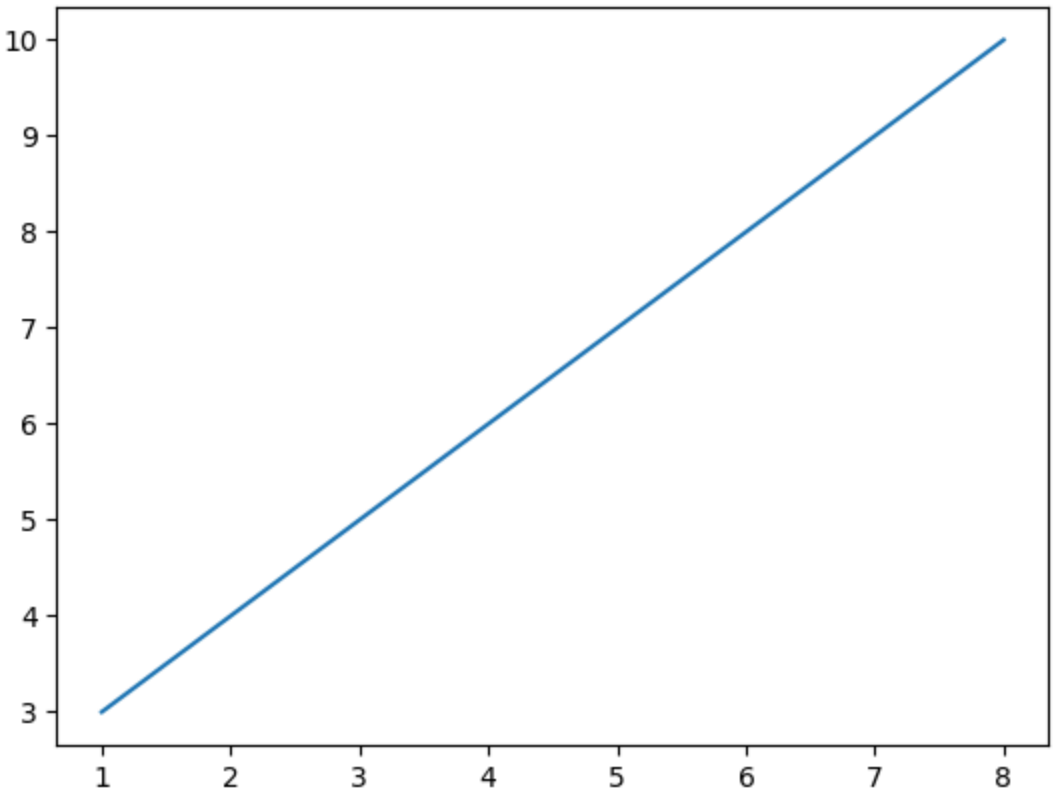
The function takes parameters for specifying points in the diagram.

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

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

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()
```

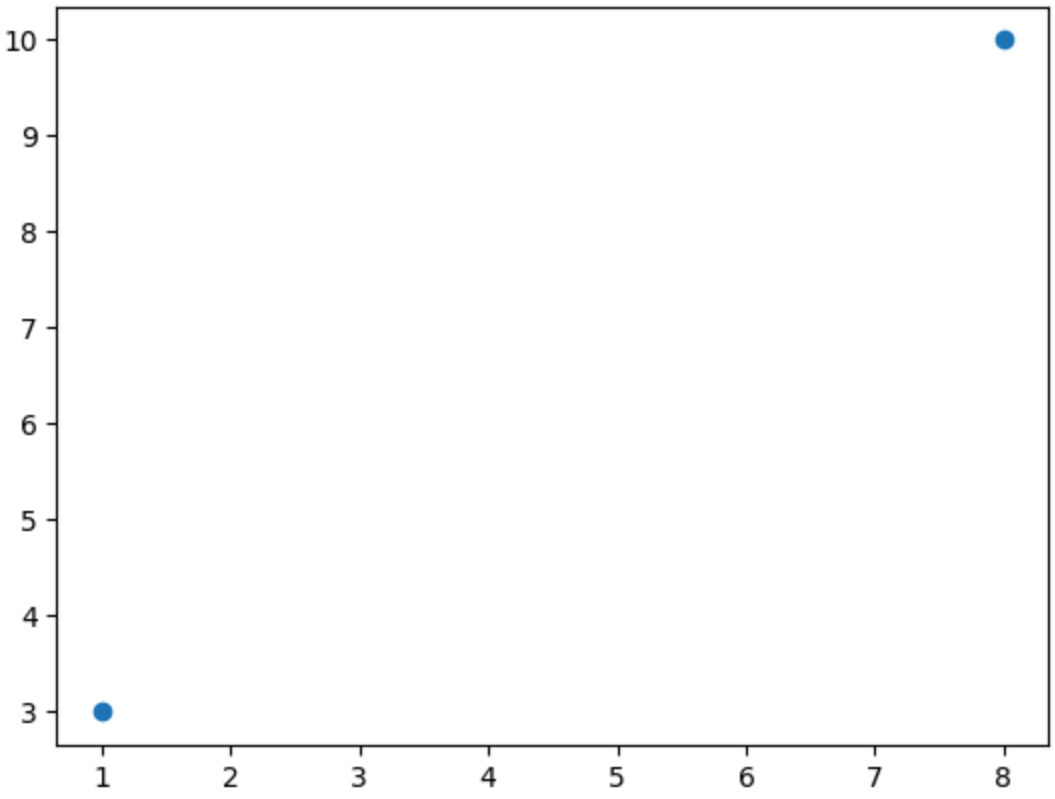


Plotting Without Line

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

Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

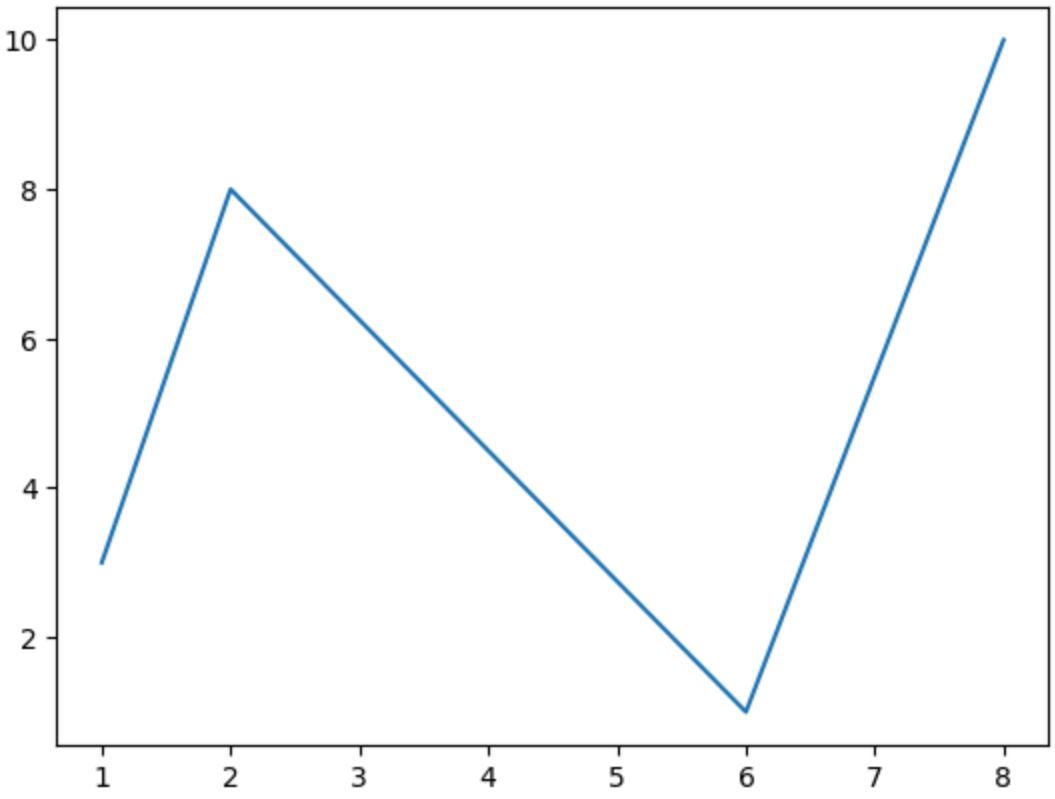
```
import matplotlib.pyplot as plt
import numpy as np

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

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



Multiple Points

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()
```

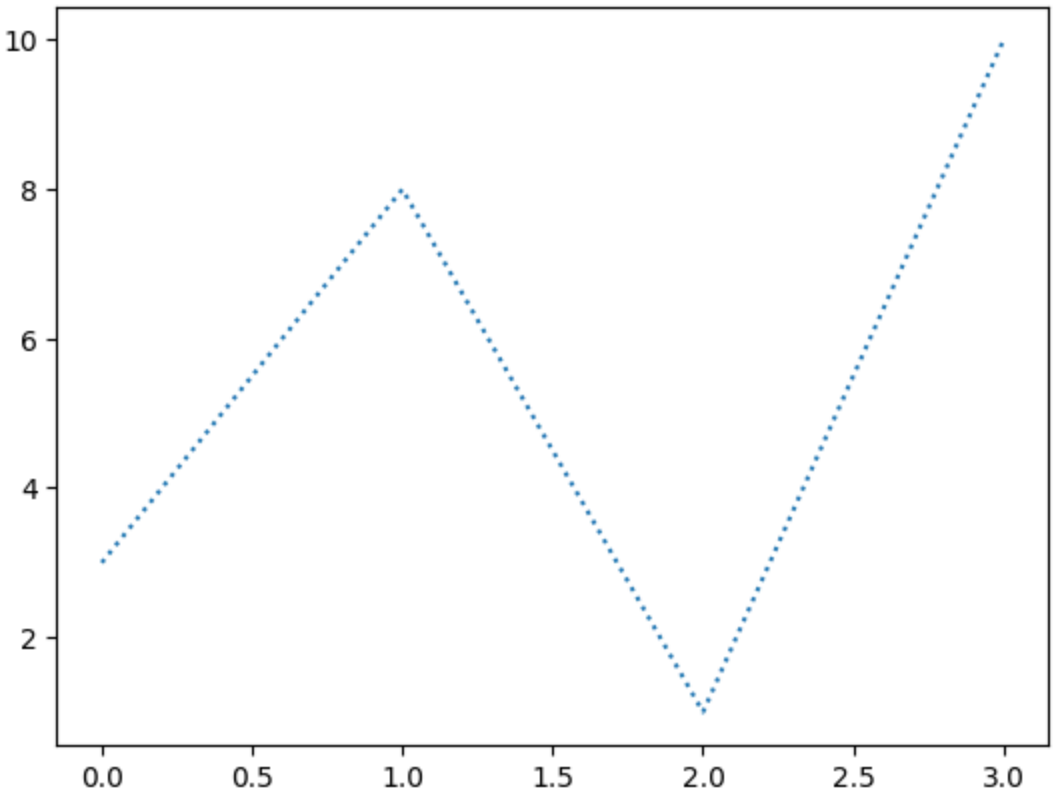


Linestyle

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

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()
```

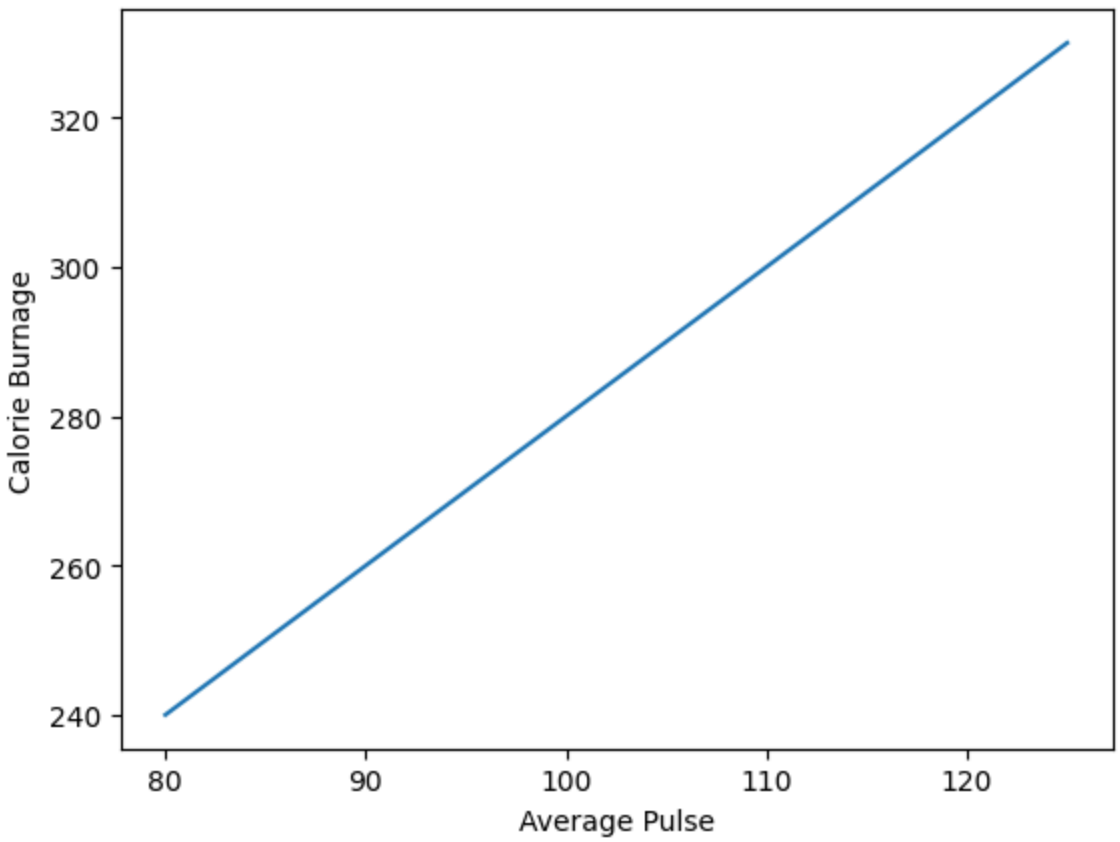


Create Labels for a Plot

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

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()
```

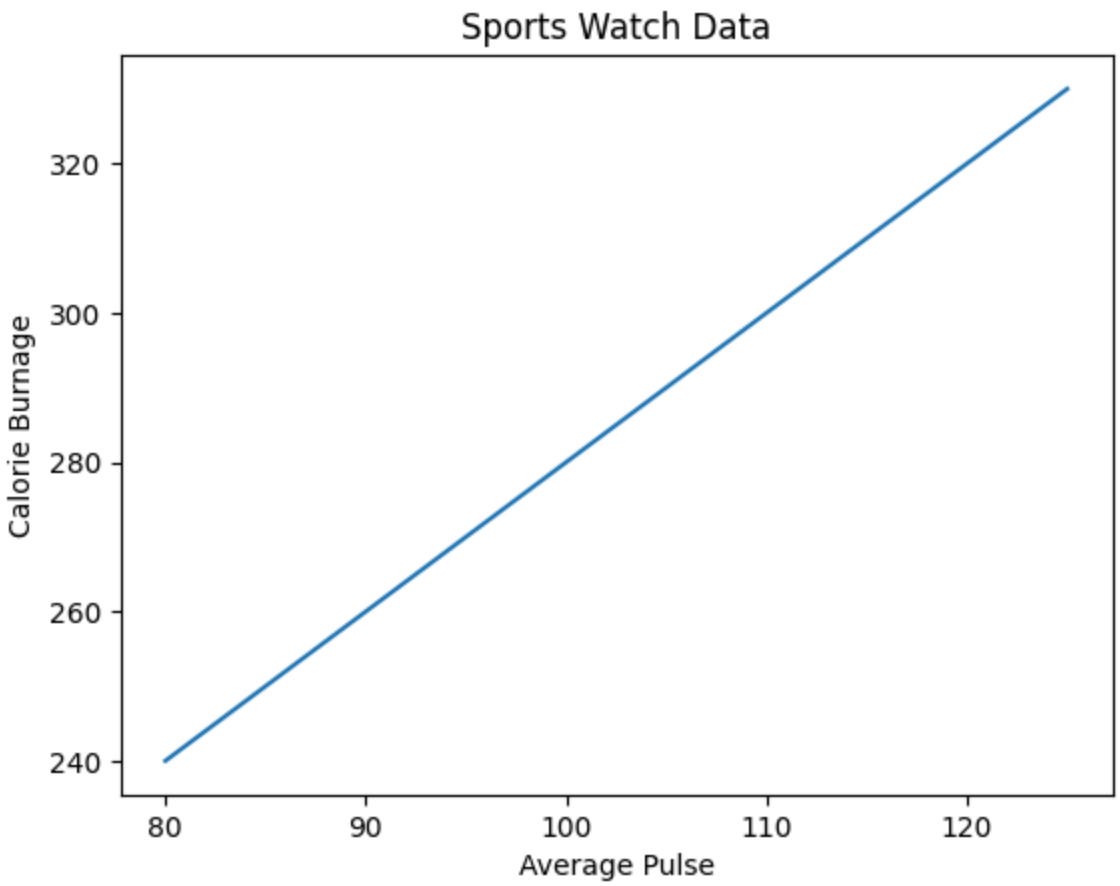


Create a Title for a Plot

Create a Title for a Plot

Add a plot title and labels for the x- and y-axis:

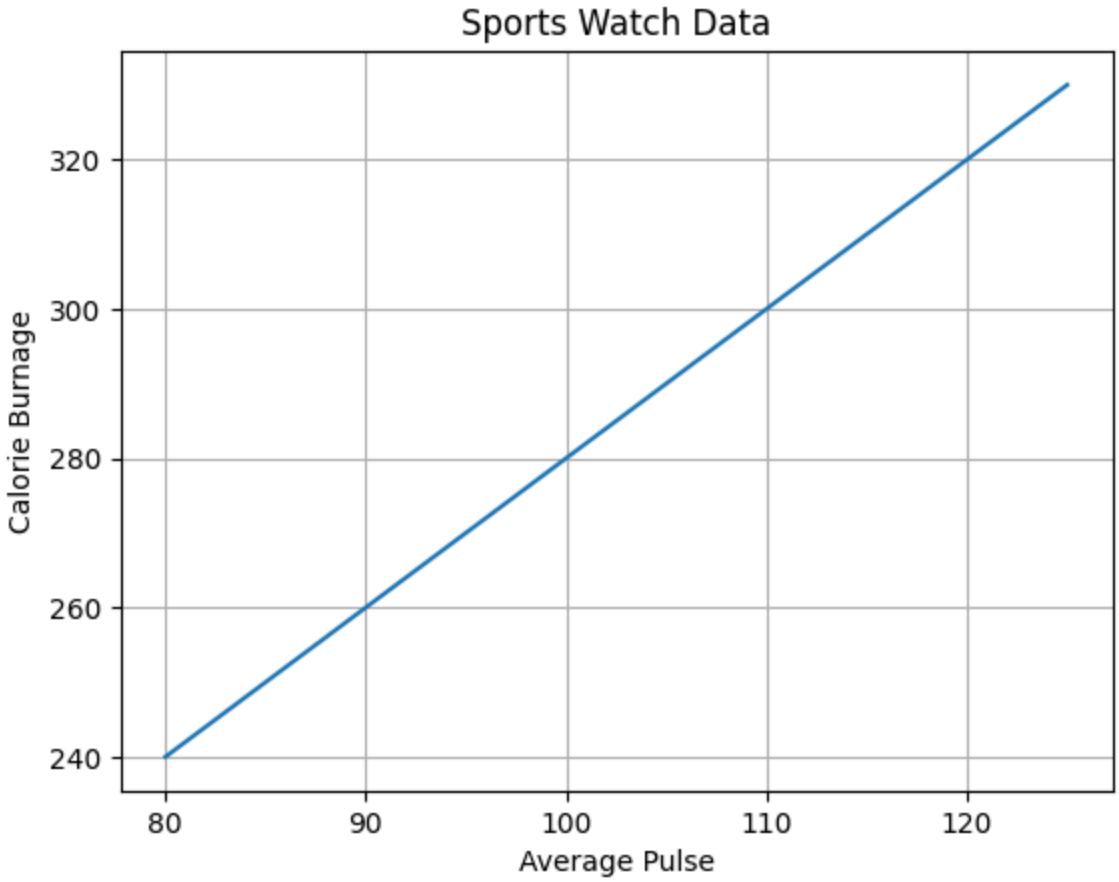
```
import numpy as np
import matplotlib.pyplot as plt

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()
```



Add Grid Lines to a Plot

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

```
#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, 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()
```

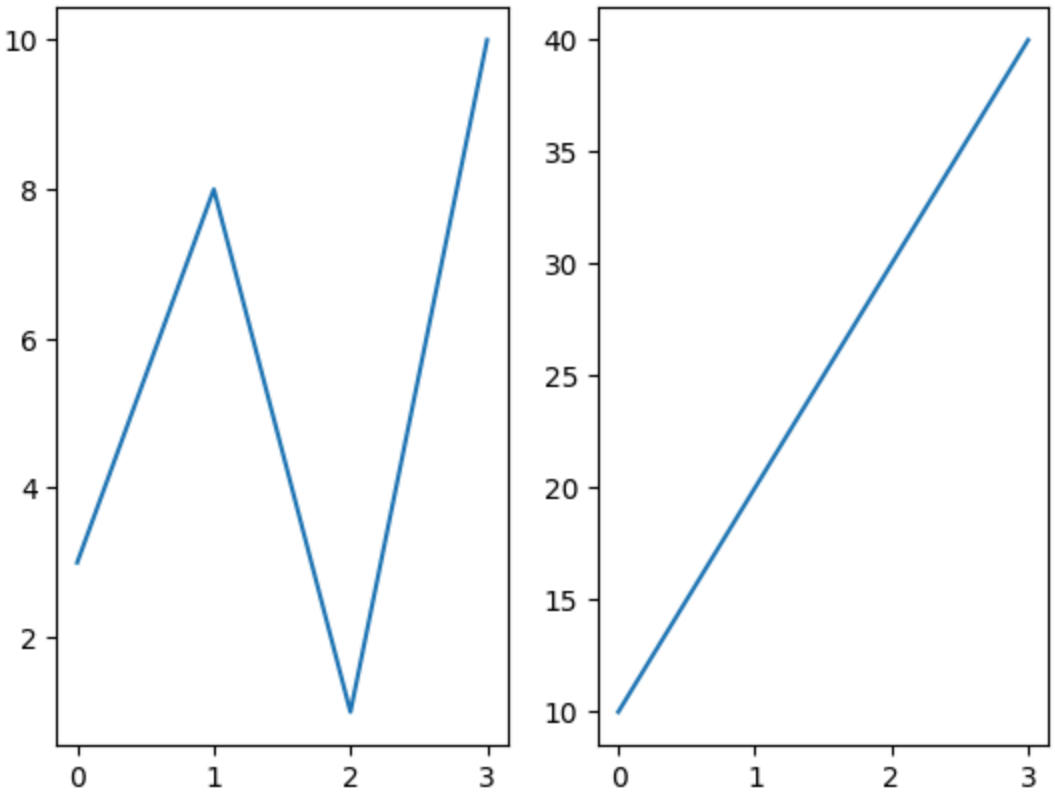


Display Multiple Plots

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

```
#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)
```

```
18
19 plt.show()
```



Creating Scatter Plots

With Pyplot, you can use the scatter() function to draw a scatter plot.

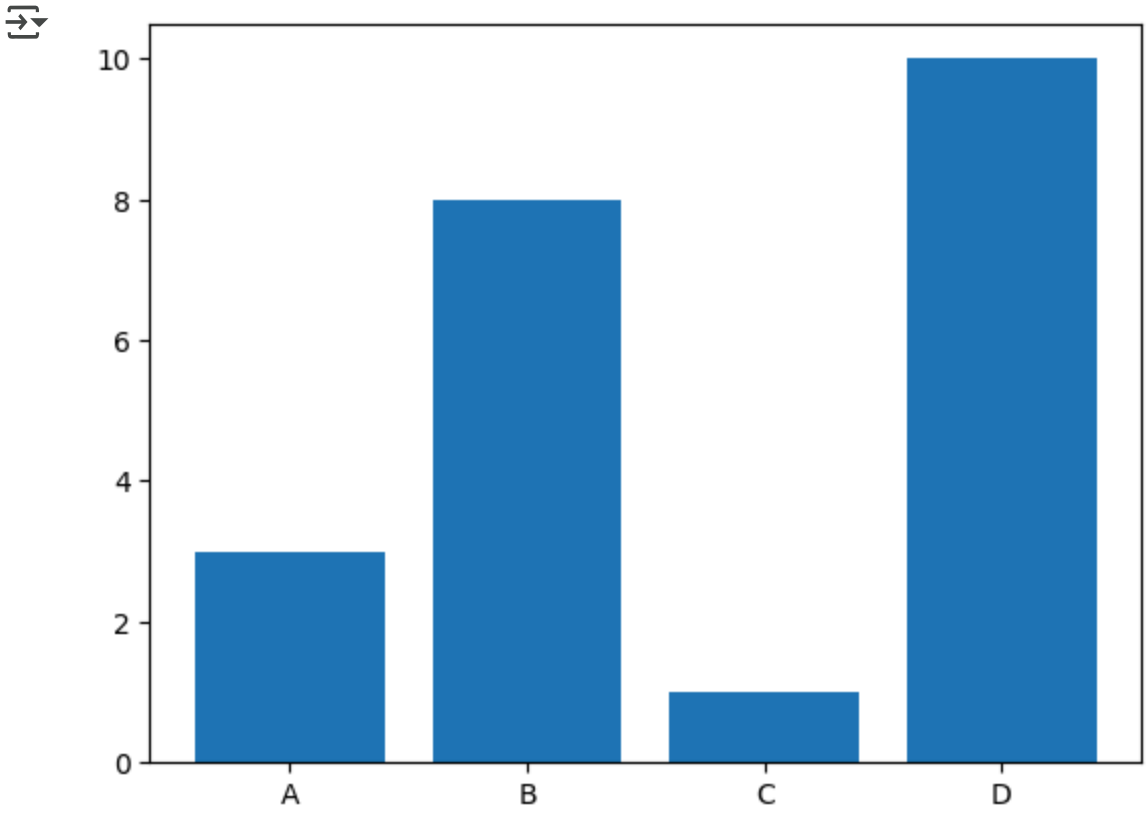
The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

```
1 #A simple scatter plot:
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
6 y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
7
8 plt.scatter(x, y)
9 plt.show()
```

Creating Bars

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

```
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()
```



Creating Pie Charts

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

A simple pie chart:

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

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

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