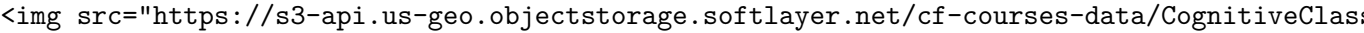


PY0101EN-5-1-Numpy1D

October 17, 2019

https://coc1.us/PY0101EN_edx_add_top


1D Numpy in Python

Welcome! This notebook will teach you about using Numpy in the Python Programming Language. By the end of this lab, you'll know what Numpy is and the Numpy operations.

Table of Contents

- [Preparation](#)
- [What is Numpy?](#)
 - [Type](#)
 - [Assign Value](#)
 - [Slicing](#)
 - [Assign Value with List](#)
 - [Other Attributes](#)
- [Numpy Array Operations](#)
 - [Array Addition](#)
 - [Array Multiplication](#)
 - [Product of Two Numpy Arrays](#)
 - [Dot Product](#)
 - [Adding Constant to a Numpy Array](#)
- [Mathematical Functions](#)
- [Linspace](#)

Estimated time needed: **30 min**

Preparation

```
[4]: # Import the libraries

import time
import sys
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
```

```
[3]: # Plotting functions

def Plotvec1(u, z, v):

    ax = plt.axes()
    ax.arrow(0, 0, *u, head_width=0.05, color='r', head_length=0.1)
    plt.text(*(u + 0.1), 'u')

    ax.arrow(0, 0, *v, head_width=0.05, color='b', head_length=0.1)
    plt.text(*(v + 0.1), 'v')
    ax.arrow(0, 0, *z, head_width=0.05, head_length=0.1)
    plt.text(*(z + 0.1), 'z')
    plt.ylim(-2, 2)
    plt.xlim(-2, 2)

def Plotvec2(a,b):
    ax = plt.axes()
    ax.arrow(0, 0, *a, head_width=0.05, color = 'r', head_length=0.1)
    plt.text(*(a + 0.1), 'a')
    ax.arrow(0, 0, *b, head_width=0.05, color = 'b', head_length=0.1)
    plt.text(*(b + 0.1), 'b')
    plt.ylim(-2, 2)
    plt.xlim(-2, 2)
```

Create a Python List as follows:

```
[3]: # Create a python list

a = ["0", 1, "two", "3", 4]
```

We can access the data via an index:

We can access each element using a square bracket as follows:

```
[4]: # Print each element

print("a[0]:", a[0])
print("a[1]:", a[1])
print("a[2]:", a[2])
```

```
print("a[3]:", a[3])
print("a[4]:", a[4])
```

```
a[0]: 0
a[1]: 1
a[2]: two
a[3]: 3
a[4]: 4
```

What is Numpy?

A numpy array is similar to a list. It's usually fixed in size and each element is of the same type. We can cast a list to a numpy array by first importing numpy:

```
[3]: # import numpy library

import numpy as np
```

We then cast the list as follows:

```
[4]: # Create a numpy array

a = np.array([0, 1, 2, 3, 4])
a
```

```
[4]: array([0, 1, 2, 3, 4])
```

Each element is of the same type, in this case integers:

As with lists, we can access each element via a square bracket:

```
[3]: # Print each element

print("a[0]:", a[0])
print("a[1]:", a[1])
print("a[2]:", a[2])
print("a[3]:", a[3])
print("a[4]:", a[4])
```

```
a[0]: 0
a[1]: 1
a[2]: 2
a[3]: 3
a[4]: 4
```

Type

If we check the type of the array we get `numpy.ndarray`:

```
[4]: # Check the type of the array
```

```
type(a)
```

```
[4]: numpy.ndarray
```

As numpy arrays contain data of the same type, we can use the attribute “dtype” to obtain the Data-type of the array’s elements. In this case a 64-bit integer:

```
[5]: # Check the type of the values stored in numpy array
a.dtype
```

```
[5]: dtype('int64')
```

We can create a numpy array with real numbers:

```
[8]: # Create a numpy array

b = np.array([3.1, 11.02, 6.2, 213.2, 5.2])
b
```

```
[8]: array([ 3.1 , 11.02,  6.2 , 213.2 ,  5.2 ])
```

When we check the type of the array we get numpy.ndarray:

```
[9]: # Check the type of array

type(b)
```

```
[9]: numpy.ndarray
```

If we examine the attribute dtype we see float 64, as the elements are not integers:

```
[10]: # Check the value type

b.dtype
```

```
[10]: dtype('float64')
```

Assign value

We can change the value of the array, consider the array c:

```
[11]: # Create numpy array

c = np.array([20, 1, 2, 3, 4])
c
```

```
[11]: array([20,  1,  2,  3,  4])
```

We can change the first element of the array to 100 as follows:

```
[12]: # Assign the first element to 100
```

```
c[0] = 100  
c
```

```
[12]: array([100,  1,  2,  3,  4])
```

We can change the 5th element of the array to 0 as follows:

```
[13]: # Assign the 5th element to 0
```

```
c[4] = 0  
c
```

```
[13]: array([100,  1,  2,  3,  0])
```

Slicing

Like lists, we can slice the numpy array, and we can select the elements from 1 to 3 and assign it to a new numpy array d as follows:

```
[14]: # Slicing the numpy array
```

```
d = c[1:4]  
d
```

```
[14]: array([1, 2, 3])
```

We can assign the corresponding indexes to new values as follows:

```
[15]: # Set the fourth element and fifth element to 300 and 400
```

```
c[3:5] = 300, 400  
c
```

```
[15]: array([100,  1,  2, 300, 400])
```

Assign Value with List

Similarly, we can use a list to select a specific index. The list 'select' contains several values:

```
[16]: # Create the index list
```

```
select = [0, 2, 3]
```

We can use the list as an argument in the brackets. The output is the elements corresponding to the particular index:

```
[17]: # Use List to select elements
```

```
d = c[select]
d
```

```
[17]: array([100,    2, 300])
```

We can assign the specified elements to a new value. For example, we can assign the values to 100 000 as follows:

```
[18]: # Assign the specified elements to new value

c[select] = 100000
c
```

```
[18]: array([100000,         1, 100000, 100000,         400])
```

Other Attributes

Let's review some basic array attributes using the array a:

```
[19]: # Create a numpy array

a = np.array([0, 1, 2, 3, 4])
a
```

```
[19]: array([0, 1, 2, 3, 4])
```

The attribute size is the number of elements in the array:

```
[20]: # Get the size of numpy array

a.size
```

```
[20]: 5
```

The next two attributes will make more sense when we get to higher dimensions but let's review them. The attribute ndim represents the number of array dimensions or the rank of the array, in this case, one:

```
[21]: # Get the number of dimensions of numpy array

a.ndim
```

```
[21]: 1
```

The attribute shape is a tuple of integers indicating the size of the array in each dimension:

```
[ ]: # Get the shape/size of numpy array

a.shape
```

```
[ ]: # Create a numpy array
```

```
a = np.array([1, -1, 1, -1])
```

```
[ ]: # Get the mean of numpy array
```

```
mean = a.mean()
```

```
mean
```

```
[ ]: # Get the standard deviation of numpy array
```

```
standard_deviation=a.std()
```

```
standard_deviation
```

```
[ ]: # Create a numpy array
```

```
b = np.array([-1, 2, 3, 4, 5])
```

```
b
```

```
[ ]: # Get the biggest value in the numpy array
```

```
max_b = b.max()
```

```
max_b
```

```
[ ]: # Get the smallest value in the numpy array
```

```
min_b = b.min()
```

```
min_b
```

Numpy Array Operations

Array Addition

Consider the numpy array u:

```
[2]: import numpy as np
```

```
u = np.array([1, 0])
```

```
u
```

```
[2]: array([1, 0])
```

Consider the numpy array v:

```
[3]: v = np.array([0, 1])
```

```
v
```

```
[3]: array([0, 1])
```

We can add the two arrays and assign it to z:

```
[4]: # Numpy Array Addition
```

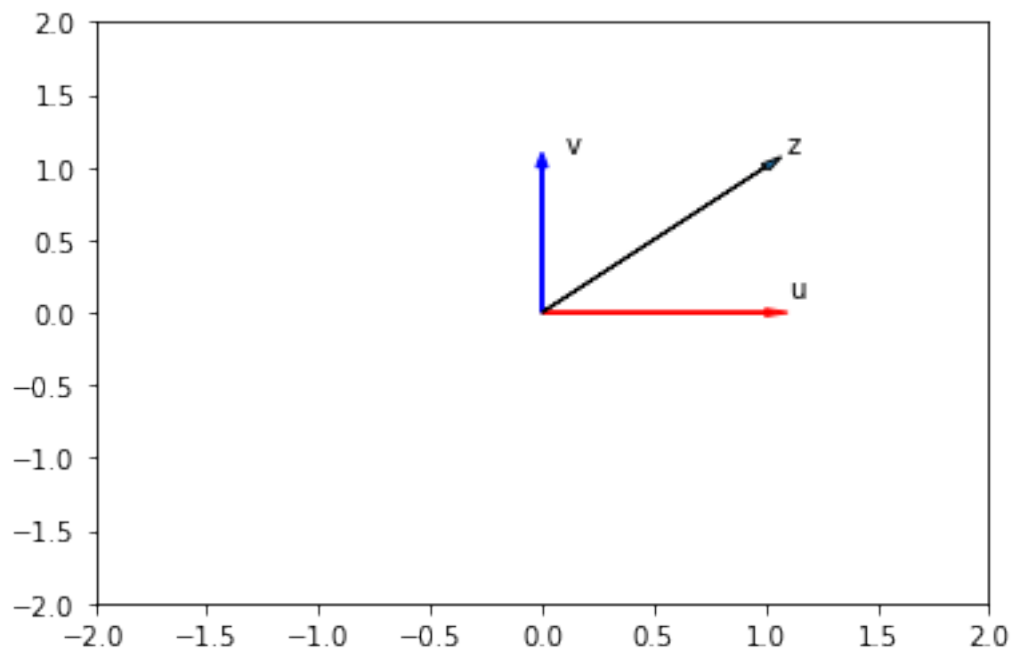
```
z = u + v  
z
```

```
[4]: array([1, 1])
```

The operation is equivalent to vector addition:

```
[9]: # Plot numpy arrays
```

```
Plotvec1(u, z, v)
```



Array Multiplication

Consider the vector numpy array y:

```
[10]: # Create a numpy array
```

```
y = np.array([1, 2])  
y
```

```
[10]: array([1, 2])
```

We can multiply every element in the array by 2:


```
[11]: # Numpy Array Multiplication
```

```
z = 2 * y  
z
```

```
[11]: array([2, 4])
```

This is equivalent to multiplying a vector by a scalar:

Product of Two Numpy Arrays

Consider the following array u:

```
[ ]: # Create a numpy array
```

```
u = np.array([1, 2])  
u
```

Consider the following array v:

```
[ ]: # Create a numpy array
```

```
v = np.array([3, 2])  
v
```

The product of the two numpy arrays u and v is given by:

```
[ ]: # Calculate the production of two numpy arrays
```

```
z = u * v  
z
```

Dot Product

The dot product of the two numpy arrays u and v is given by:

```
[12]: # Calculate the dot product
```

```
np.dot(u, v)
```

```
[12]: 0
```

Adding Constant to a Numpy Array

Consider the following array:

```
[13]: # Create a constant to numpy array
```

```
u = np.array([1, 2, 3, -1])  
u
```

```
[13]: array([ 1,  2,  3, -1])
```

Adding the constant 1 to each element in the array:

```
[14]: # Add the constant to array  
  
u + 1
```

```
[14]: array([2, 3, 4, 0])
```

The process is summarised in the following animation:

Mathematical Functions

We can access the value of pie in numpy as follows :

```
[15]: # The value of pie  
  
np.pi
```

```
[15]: 3.141592653589793
```

We can create the following numpy array in Radians:

```
[6]: # Create the numpy array in radians  
  
x = np.array([0, np.pi/2 , np.pi])  
x
```

```
[6]: array([0.          , 1.57079633, 3.14159265])
```

We can apply the function sin to the array x and assign the values to the array y; this applies the sine function to each element in the array:

```
[7]: # Calculate the sin of each elements  
  
y = np.sin(x)  
y
```

```
[7]: array([0.0000000e+00, 1.0000000e+00, 1.2246468e-16])
```

Linspace

A useful function for plotting mathematical functions is “linspace”. Linspace returns evenly spaced numbers over a specified interval. We specify the starting point of the sequence and the ending point of the sequence. The parameter “num” indicates the Number of samples to generate, in this case 5:

```
[ ]: # Makeup a numpy array within [-2, 2] and 5 elements
```

```
np.linspace(-2, 2, num=5)
```

If we change the parameter num to 9, we get 9 evenly spaced numbers over the interval from -2 to 2:

```
[ ]: # Makeup a numpy array within [-2, 2] and 9 elements  
  
np.linspace(-2, 2, num=9)
```

We can use the function line space to generate 100 evenly spaced samples from the interval 0 to 2 :

```
[ ]: # Makeup a numpy array within [0, 2] and 100 elements  
  
x = np.linspace(0, 2*np.pi, num=100)
```

We can apply the sine function to each element in the array x and assign it to the array y:

```
[ ]: # Calculate the sine of x list  
  
y = np.sin(x)
```

```
[ ]: # Plot the result  
  
plt.plot(x, y)
```

Quiz on 1D Numpy Array

Implement the following vector subtraction in numpy: $u-v$

```
[8]: # Write your code below and press Shift+Enter to execute  
  
u = np.array([1, 0])  
v = np.array([0, 1])  
z=u-v  
z
```

```
[8]: array([ 1, -1])
```

Double-click **here** for the solution.

Multiply the numpy array z with -2:

```
[9]: # Write your code below and press Shift+Enter to execute  
  
z = np.array([2, 4])  
e=-2*z  
e
```

```
[9]: array([-4, -8])
```

Double-click [here](#) for the solution.

Consider the list $[1, 2, 3, 4, 5]$ and $[1, 0, 1, 0, 1]$, and cast both lists to a numpy array then multiply them together:

```
[ ]: # Write your code below and press Shift+Enter to execute
import numpy as np
a=np.array([1,2,3,4,5])
b=np.array([1,0,1,0,1])
c=a*b
c
```

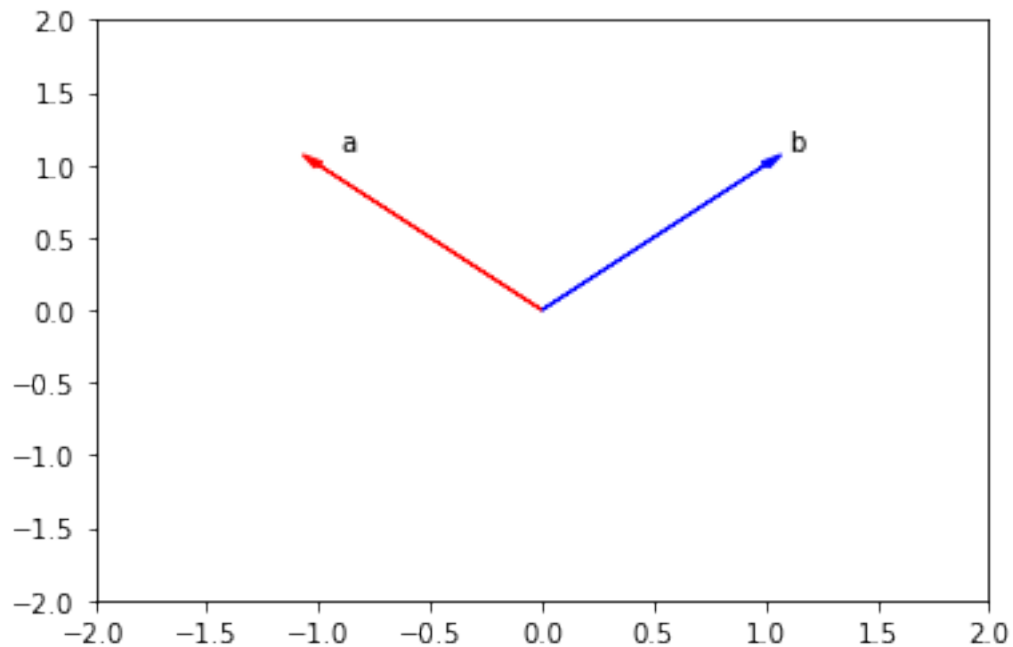
```
[ ]: array([1, 0, 3, 0, 5])
```

Double-click [here](#) for the solution.

Convert the list $[-1, 1]$ and $[1, 1]$ to numpy arrays a and b. Then, plot the arrays as vectors using the function Plotvec2 and find the dot product:

```
[5]: # Write your code below and press Shift+Enter to execute
import numpy as np
a=np.array([-1,1])
b=np.array([1,1])
Plotvec2(a,b)
np.dot(a,b)
```

```
[5]: 0
```

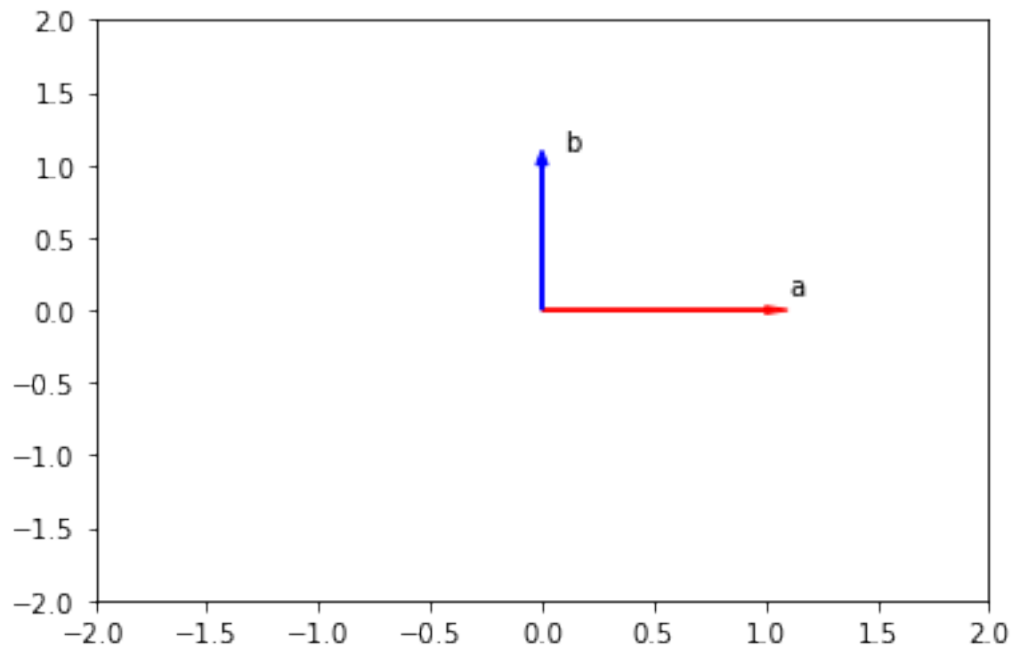


Double-click [here](#) for the solution.

Convert the list $[1, 0]$ and $[0, 1]$ to numpy arrays a and b . Then, plot the arrays as vectors using the function `Plotvec2` and find the dot product:

```
[6]: # Write your code below and press Shift+Enter to execute
a=np.array([1,0])
b=np.array([0,1])
Plotvec2(a,b)
np.dot(a,b)
```

[6]: 0

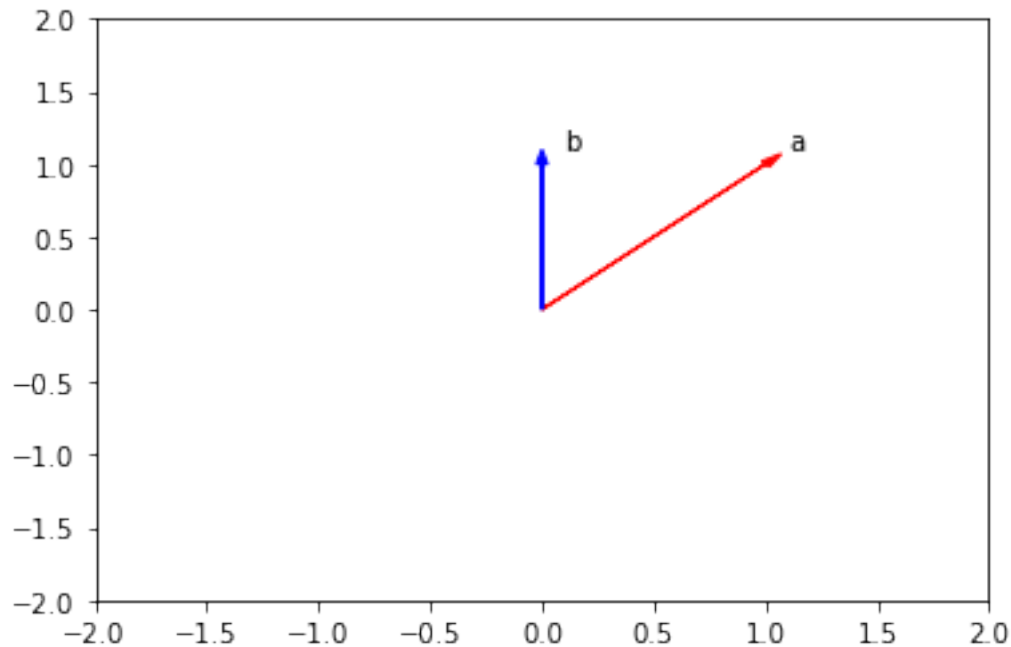


Double-click [here](#) for the solution.

Convert the list $[1, 1]$ and $[0, 1]$ to numpy arrays a and b . Then plot the arrays as vectors using the function `Plotvec2` and find the dot product:

```
[7]: # Write your code below and press Shift+Enter to execute
a=np.array([1,1])
b=np.array([0,1])
Plotvec2(a,b)
np.dot(a,b)
```

[7]: 1



Double-click [here](#) for the solution.

Why are the results of the dot product for $[-1, 1]$ and $[1, 1]$ and the dot product for $[1, 0]$ and $[0, 1]$ zero, but not zero for the dot product for $[1, 1]$ and $[0, 1]$?

Hint: Study the corresponding figures, pay attention to the direction the arrows are pointing to.

```
[9]: # Write your code below and press Shift+Enter to execute
a=np.array([-1,1])
b=np.array([1,1])
Plotvec2(a,b)
np.dot(a,b)

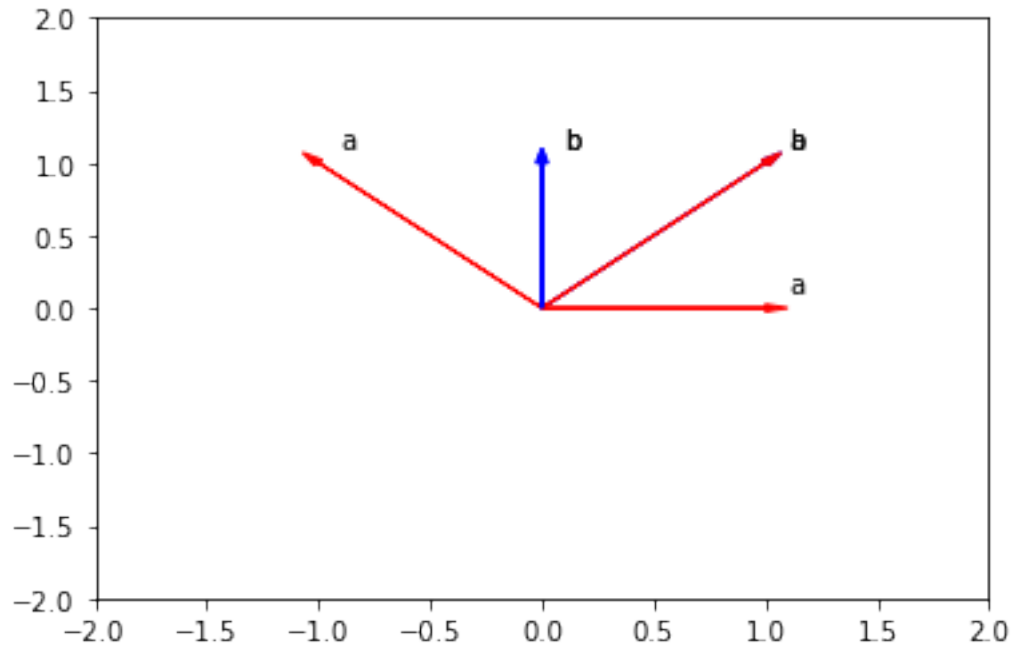
c=np.array([1,0])
d=np.array([0,1])
Plotvec2(c,d)
np.dot(c,d)

e=np.array([1,1])
f=np.array([0,1])
Plotvec2(e,f)
np.dot(e,f)
```

/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/ipykernel_launcher.py:17: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and

returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

[9]: 1



Double-click **here** for the solution.

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow this article to learn how to share your work.

Get IBM Watson Studio free of charge!

<p><img src="https://s3-api.us-geo.objectst

About the Authors:

Joseph Santarcangelo is a Data Scientist at IBM, and holds a PhD in Electrical Engineering. His research focused on using Machine Learning, Signal Processing, and Computer Vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Other contributors: Mavis Zhou

Copyright © 2018 IBM Developer Skills Network. This notebook and its source code are released under the terms of the MIT License.