Numpy an External Library

This lesson introduces an external numpy library by discussing in detail how numpy provides support to handle single or multidimensional arrays and different functionalities.

WE'LL COVER THE FOLLOWING

Handling arrays with NumPy

Single dimensional arrays

For the sake of simplicity, you can think of Numpy as a fast and efficient library for storing and manipulating data - usually through array data structures (which are similar to Python lists). You can find all the documentation for Numpy here. Numpy arrays are different from standard Python lists in many ways; notably, they are faster, take up less space, and have more functionality. It is important to note, though, that these arrays are of a fixed size and type that you define at creation. You cannot infinitely append new values as you might with a list.

Let's see some of the ways you can use arrays in action.

```
import numpy as np

# This creates our array
```

```
np_array = np.array([5, 10, 15, 20, 25, 30])
print("--0--")


# Gets the unique values
print(np.unique(np_array))
print("--1--")

# Calculates the standard deviation
print(np.std(np_array))
print("--2--")

# Calculates the maximum
print(np_array.max())
print("--3--")

# Squares each value in the array
print(np_array ** 2)
print("--4--")

# Adds the arrays together element wise
print(np_array + np_array)
print("--5--")

# The sum of the squares of the elements
print(np.sum(np_array ** 2))
print("--6--")

# Gives you the shape: (rows, columns)
print(np_array.shape)
```

Two dimensional arrays

You will also want to understand how to use 2D arrays, arrays that have both *rows* and *columns*. This is typically how you would represent a **matrix** of data using `numpy`.

```
import numpy as np

# Create 2d array
print("--0--")
np_2d_array = np.array([[1,2,3],
                        [4,5,6]])
print(np_2d_array)

# Calculate the transpose, which is when you swap the columns and rows.
print("--1--")
np_2d_array_T = np_2d_array.T
print(np_2d_array_T)

# Print the shape of the array as (number of rows, number of columns)
print("--3--")
print(np_2d_array.shape)
```

```
# Access elements in the 2d array by index.
# First index is the row number
# Second index is the column number

# Index numbers start from 0
print("--4--")
print(np_2d_array[1,1])
print(np_2d_array[0,2])
```

Important functionalities

Calculating dot product

A useful `numpy` function is the ability to calculate the *dot product* of two vectors. If you need a refresher on what a dot product is, you can check out the Wikipedia page. Here is how you do it with code:

```
import numpy as np
np_array = np.array([5, 10, 15, 20, 25, 30])
dot_product = np.dot(np_array, np_array)
print(dot_product)
```

Generating random values

`Numpy` has some excellent functions to help you randomly sample your data.

First, the `rand()` function returns a random number or numbers between 0 and 1. The number of random values returned depends on the shape provided to the function. If no shape is given, just a single number is returned.

```
import numpy as np

# Generage a single random number in range [0,1)
print("--0--")
print(np.random.rand())

# Generate a matrix of random numbers in range [0,1) with shape (3,2)
print("--1--")
print(np.random.rand(3,2))
```

You can also use `randint()` to return random integers from low (inclusive) to high (exclusive). You pass the low and high values as parameters to the function as well as the size of the output.

```
import numpy as np

# Low=5, High=15, Size=2. Generate 2 values between 5 and 15 (exclusive)
print("--0--")
print(np.random.randint(5, 15, 2))

# Low=5, High=15, Size=(3,2). Generate a matrix of shape (3,2) with values between 5 and 15 (
print("--1--")
print(np.random.randint(5, 15, (3,2)))
```

Sampling the data

Once you have some data, it can be useful to sample from it.

As a refresher, sampling is a way to take a smaller group from a population. For example, you might create a random sample from the U.S. population randomly knocking on 10 doors in the U.S. That would be a sample size of 10.

The `choice()` function allows you to pass an array, specify how many values to sample, and decide whether sampling should be done with or without replacement. Sampling without replacement means the same value can't be sampled more than once.

```
import numpy as np
array = np.array([1,2,3,4,5])

# Sample 10 data points with replacement.
print("--0--")
print(np.random.choice(array, 10, replace=True))

# Sample 3 data points without replacement.
print("--1--")
print(np.random.choice(array, 3, replace=False))
```

Randomly shuffling values

Randomly shuffling values

Sometimes, it can be helpful to randomly shuffle your array. `Numpy` has a `shuffle()` function that does just that.

> **Note:** The shuffle happens in place, so you don't have to reassign the array to a new value.

```
import numpy as np

x = [1,2,3,4,5]  # Create a list of 5 elements
np.random.shuffle(x)  # Randomly shuffle the order of the elements in the list

print(x)
```

Also, when using randomness, if you want the random selections to be the same every time, you need to set a **seed**. This is possible using the function: `numpy.random.seed()`. We have to supply the function with an integer to *seed* the randomness, so the results can be replicated as long as the same seed is used.

```
import numpy as np
np.random.seed(42)
```

If we put the above code at the beginning of our script, it will ensure the randomness is always the same and tied to the seed 42.

We will end our discussion on Python by examining the library **Scipy**, in the next lesson.