NUMPY

Introduction to Python's Numpy

INTRODUCTION

One of the, if not the, most important part of Data Science is computation of numerical data. NumPy, short for "Numerical Python" is an **incredible library** that aids in mathematical, scientific, and data science computations and programming.

It is **memory efficient**; it can **handle vast amounts of data** better than any other library. **Very convenient** to work with, especially for matrix multiplication and reshaping. Most importantly - **NumPy is fast**. In fact, big names like TensorFlow and Scikit-Learn use NumPy for their matrix operations.

It is implemented in C - which by default outperforms traditional python. Thus giving Numpy its famous speed.

There are many other factors, which are overwhelmingly hard to remember for my little brain. You can check out a detailed comparison here at the link provided in your notebooks under "Why Numpy?" Python Lists vs. Numpy Arrays - What is the difference?: IST Advanced Topics Primer.

OVERVIEW & OBJECTIVES

- Installation
    - Install
        - So, very firstly let's install the Numpy package. We are gonna use pip - pip install numpy.
        - Now, it shows "Requirement already satisfied" because Google Colab's notebook environment has it installed by default.
    - Import and Version Check
        - Now let's import numpy as np; and print its version to check if it got imported successfully.
        - Good. Okay now we can use the numpy functionalities in our notebook.

- Numpy Arrays
    - Array initialization (using lists)
        - At the very core of the numpy library is the “N-dimensional Array Object” which is just a fancy phrase for a flexible and multi-dimensional list-like object with many functions built around it.
        - Let’s see how to use it - we can do: array = np.array and pass a list to it.
        - On printing it, we see we have successfully initialized the numpy array.
    - Arithmetical Operations on Arrays
        - A big requirement of working with data is computing it. Numpy arrays makes calculation easy by letting us perform arithmetic operations:-
            - So we can add a scalar
            - Subtract a scalar
            - Multiply a scalar
            - And Divide a scalar
    - Array Attributes
        - Numpy array has a couple important attributes for its’ arrays too
            - Shape
                - Shows the shape of a numpy array
            - DType
                - Shows the data type of a numpy array

Everyone is with me right now yeah? Please feel free to tell me if I need to slow down.

- Multi-dimensional Arrays
    - Now Numpy also supports multi-dimensional array objects so for e.g. we can have:-
        - 2D Array Object
            - Initialised by passing a 2D list object.
            - Let’s print it and its shape too.
        - 3D
            - Initialised similarly using a 3D list Object

        - On printing its shape we see it a 3D dimensional object with dimensions - “”

- Ones and Zeros
    - Now Numpy provides us with a useful little function to initialise numpy arrays with just ones and zeros.
        - For zeros - we have array = np.zeros and pass the desired shape of the array.
        - On printing shape and dtype we see we got a 2D object as we specified. Now please take note that (scroll up and demo) by default the dtype of the array is int64, but for ones and zeros we have it as float64.
        - We can also pass a dtype object to initialise our own desired datatype. For e.g. we pass int and when we print its dtype. There it is.
        - For ones we have everything similar except that the function becomes “np.ones”

Everyone still on track? Or do I need to pace down? Any questions up until now?

- Reshaping
    - The Numpy library also has some built in reshaping functionalities.
    - To demonstrate let’s initialise a 2D array object and print its shape.
    - First we Reshape
        - It can be used as array.reshape and we pass the desired shape.
        - So we had an array of shape (2, 3) and now we want to reshape it to (3, 2).
        - Done.
    - Somewhat similarly we have Flatten.
        - We use flatten to, obvious by the name, flatten the multi-dimensional array object.
        - On printing the array we see that now its a 1D array object.
- Horizontal and Vertical Stacking
    - We also functions for stacking two or more arrays either on top of each other or in front of each other.
    - To demonstrate, let’s have a couple of 1D arrays.
    - To stack the arrays in front of each other or horizontally, we have “np.hstack”.

        - Seeing the output we can observe that it works similarly to “concatenation” or just joining two arrays.
    - To stack the arrays on top of each other or vertically, we have “np.vstack”.
        - We can see that now we have a 2D object with, please note, the former array on top and the latter array below it.

Any questions at this point?

- Generating Data
    - Coming to generating some data. We can use numpy in cases where we want to generate a range or a set of numbers in a particular order.
        - To get a range of numbers between a start and an endpoint we have “**np.arange**” and we pass the starting and the ending point. We can also specify the desired gap between each number for eg.-
        - To generate a set of numbers spaced “ linearly” or spaced “evenly” we have “**np.linspace”**. Here we specify the start, the end and the “number of elements” we want. Numpy calculates the required spacing for each element returns us an array object.
        - Similarly to generate numbers that are spaced logarithmically we can use np.logspace. The same parameters are passed.
        - Now to generate numbers randomly we have a whole functionality called “**np.random**” and to showcase it, we can see a set of random numbers distributed “normally” - now a normal distribution is when the set of numbers are distributed such that they are symmetrical about the mean. You can see it as a visual representation in your notebooks. Mu being the mean and the distribution is plotted such that it is symmetrical about the mean. This is also known as a bell-curve, since obviously because it looks like a bell. So we pass in the mean, the standard deviation or the amount of deviation we want from the mean, and the number of elements to be returned. And there it is.

Any doubts? Need a break?

- Indexing
    - In order to access specific elements in the numpy arrays we can use something called “indexing”

    - For 1D, we can access elements by specifying its index. Or we can also use "slicing", where we specify a range of indices we want to "slice-out" of the array.
        - X:y
        - :y
        - X:
    - For 2D, we can use indices of rows or columns. Or we can use slicing to return multiple columns or rows.
- Basic Statistical Functions
    - Numpy has many useful built-in functionalities to assist in computation and statistics. Although there is a very long list of such functions we can go over some basic ones such as:-
    - Min - which returns the minimum element of the array.
    - Max - which returns the maximum element of the array.
    - Mean - which returns the mean of the set of elements of the array.
    - Median - which returns the median of the set of elements of the array.
    - Standard Deviation - which returns the standard deviation of the array.
- Product of Arrays
    - Lastly we have product of two arrays. To demonstrate, let's have two 1D numpy arrays.
    - Elemental Product - Numpy computes the element-by-element product of the two arrays on multiplying two arrays directly.
    - Dot Product - To compute the dot product of two arrays we have the "np.dot" function which takes the first and the second array to be multiplied.
    - Matrix Multiplication - And for our conventional matrix multiplication we have the "np.matmul" which also takes the two arrays.
- Determinant
    - Additionally, to compute the determinant of matrices, we have the np.linalg.det function. Which returns the determinant of matrix which we pass as a parameter.

SUMMARISING

Numpy has a lot of built-in functionalities for its array object. You can check out the further documentation on “numpy.org” (link provided in your notebooks).

So now we come to the end of the Numpy part of the workshop. We have covered :-

- How to install and import Numpy
- Initializing a Numpy array from a list and with ones or zeros
- Arithmetical and Matrix Operations on Numpy arrays
- Reshaping and stacking multiple Numpy arrays
- And some basic statistical functions built-into the Numpy library

If any one has any doubts regarding this session please feel free to present them now…

Okay, so thank you for being such patient listeners and have a nice day.