Understanding Numpy & Pandas

Welcome to an introductory tutorial on Numpy and Pandas as two of the most popularly used data science library. This tutorial acquaints you with the fundamentals of NumPy arrays and pandas DataFrames. By the end of this tutorial, you will have a basic understanding of NumPy arrays, and pandas DataFrames and their related functions.

Creating a virtual environment

Now to create a virtual env on in your available text editor, you to follow thid process. However if you are using Pycharm as your editor, once your create a new project it will automatically create a virtual env for you as a default configuration.

so for those using vs code as an editor choice let's follow this steps;
In your editor, click on CTRL + B. This will bring up your editor terminal.

```
#installing venv
py –m pip install ––user virtualenv

#creating virtual env
py –m venv env

#activating virtual env
.\env\Scripts\activate

# installing numpy and pandas library
pip install numpy & pandas
```

Understanding NumPy Arrays

NumPy arrays are a series of homogenous items. Homogenous means the array will have all the elements of the same data type. Let's create an array using NumPy. You can create an array using the array() function with a list of items. Users can also fix the data type of an array. Possible data types are `bool, int, float, long, double, and long double` .

Let's see how to create an array;

```
# importing the numpy library
import numpy as np

# Creating an array
x = np.araay([2,4,6,8,10])
print(x)

# Output : [2,4,6,8,10]
```

Another way to create a NumPy array is with `arange()`. It creates an evenly spaced NumPy array. Three values – start, stop, and step – can be passed to the arange(start,[stop],step) function. The start is the initial value of the range, the stop is the last value of the range, and the step is the increment in that range. The stop parameter is compulsory and you must specific that value.

The `arange(1,15)` function will return 1 to 14 values with one step because the step is, by default, 1. The `arrange()` function generates a value that is one less than the stop parameter value, so observe that as you run your code.

Let's understand this through the following example:

```
import numpy as np

# Creating an array using arrange()
a = np.arrange(1, 15)
print(a)

# Output : [1 2 3 4 5 6 7 8 9 10 11 12 13 14]
```

Apart from the `array()` and `arange()` functions, there are other options, such as `zeros(), ones(), full(), eye(), and random()`, which can also be used to create a NumPy array, as these functions are initial placeholders. Here is a detailed description of each function:

1. `zeros()` : The `zeros()` function creates an array for a given dimension with all zeroes.
2. `ones()` : The `ones()` function creates an array for a given dimension with all ones.
3. `fulls()` : The `full()` function generates an array with constant values.
4. `eyes()` : The `eye()` function creates an identity matrix.
5. `random()` : The `random()` function creates an array with any given dimension.

Now let's look at some cool example to appreciate the functions you see above.

```
import numpy as np
# Create an array of all zeros
p = np.zeros((3,3))
print(p)

# Create an array of all ones
q = np.ones((2,2))
print(q)

# Create a constant array
r = np.full((2,2), 4)
print(a)

# Create a 2x2 identity matrix
s = np.eye(4)
print(s)

# Create an array filled with random values
t = np.random.random((3,3))
print(t)
```

Now when your run your code, try and understand what the Output of the code above.

Array Features

Let's make an array using the arange() function, as we did in the previous section, and let's check its data type:

```
import numpy as np

# Creating an array using arange()
a = np.arange(1,11)
print(type(a))
print(a.dtype)

# Output : <class 'numpy.ndarray'>
# Output :   int64
```

When you use `type()`, it returns `numpy.ndarray`. This means that the type() function returns the type of the container. When you use `dtype()`, it will return `int64`, since it is the type of the elements. You may also get the output as `int32` if you are using `32-bit` Python.

Let's find out the shape of the vector that we produced a few minutes ago:

```
print(a.shape)

# Output: (10,) -> This means the array is a one dimensional array so the 10 is just the
```

Selecting array element

In this section, we will see how to select the elements of the array. Let's see an example of a 2*2 matrix:

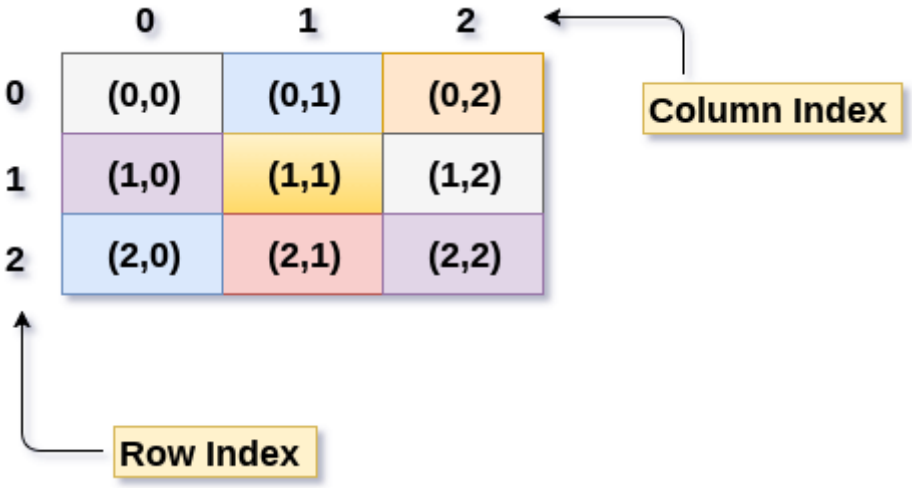
```
a = np.array([[5,6],[7,8]])
print(a)
# Output:
#   [ [5 6]
#     [7 8] ]
# The output shows a 2 x 2 matrix
```

Selecting array elements is pretty simple. We just need to specify the index of the matrix as `a[m,n]` . Here, `m` is the row index and `n` is the column index of the matrix.

```
print(a[0,0])
# Output: 5

print(a[0,1])
# Output: 6

printa([1,0])
# Output: 7

printa([1,1])
# Output: 8
```

I guess you remember this concept from your MATLAB class, if you don't i guess you have to go back and revisit that concept. 🤣 🤣 🤣 🤣 . just joking, understand from the diagram below;

This image is a 3 X 3 matrix image,from that we can find the 2 X 2 matrix. so you now understand it right ?.



Note

Many functions have a data type argument, which is frequently optional:

```
# Creating an array with a specific data type.
arr = np.arange(1,11, dtype= np.float32)
print(arr)

# Output: [ 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.]
```

Manipulating array shapes

our main focus is on array manipulation. Let's learn some new Python functions of NumPy, such as `reshape(), flatten(), ravel(), transpose(), and resize()` :

1. **`reshape()` will change the shape of the array:**

```
# Create an array
arr = np.arange(12)
print(arr)
# Output: [ 0 1 2 3 4 5 6 7 8 9 10 11]

# Reshape the array dimension
new_arr=arr.reshape(4,3)
pprint(new_arr)
# Output:
# [ [ 0, 1, 2],
#   [ 3, 4, 5],
#   [ 6, 7, 8],
#   [ 9, 10, 11] ]

# Reshape the array dimension
new_arr2=arr.reshape(3,4)
print(new_arr2)
# Output:
# array([[ 0,  1,  2,  3],
#        [ 4,  5,  6,  7],
#        [ 8,  9, 10, 11] ])
```

2. The `transpose()` function is a linear algebraic function that transposes the given two-dimensional matrix. The word transpose means converting rows into columns and columns into rows:

```
# Transpose the matrix
print(arr.transpose())

# Output:
#    [[1 4 7]
#     [2 5 8]
#     [3 6 9]]
```

3. The `resize()` function changes the size of the NumPy array. It is similar to reshape() but it changes the shape of the original array:

```
# resize the matrix
arr.resize(1,9)
print(arr)

# Output: [[1 2 3 4 5 6 7 8 9]]
```