A DOCUMENTATION ON NUMPY AND PANDAS LIBRARIES

Numpy is a python library that provides high performance support for n dimensional arrays (vectors, matrices, tensors), and enables users to perform fast mathematical operations over them.

You would surely need a software that support the use of python libraries e.g. Anaconda, PyCharm installed on your laptop before you get started. Since numpy is a module, it has to be imported into the working space, and this is done by typing “import numpy as np”; np then becomes a substitutionary term for “numpy”.

A few of the operations that can be carried out using numpy are:

* ***np.array*** **:** This function is used to create an array out of a regular python list. It can be used to create any dimension of matrix that we want. e.g.

**np.array([ [1,2,3], [3,5,6], [2,4,1] ])** returns 3x3 matrix.

* ***np.arange*** **:** It kinds of give the same output as the regular “range” function. The only difference is that it returns the output as an array. e.g.

**range(10)** returns **0,1,2,3,4,5,6,7,8,9**

**np.arange(10)** returns **array([0,1,2,3,4,5,6,7,8,9])**

**np.arange( 10, 40, 5 )** returns **array([10, 15, 20, 25, 30, 35])** … **5 signifies step**.

* ***np.save* :** This function is used to save an array. e.g.

**array = np.array([ [1,2,3,4], [56,'Sola', 'love',4] ])**

**np.save(‘array\_1’, arr)** saves arr with the name ‘array\_1’.

* ***np.savez* :** This function is used to save multiple arrays. e.g.

**array\_1 = np.array([1,2,3,4,5])**

**array\_2 = np.array([2,3,4,5,6])**

**np.savez(‘array.npz’, x= array\_1, y = array\_2)**

* ***np.load :*** This function is useful for loading an already saved file.

Numpy also has standard functions like mean, sum, variance, standard deviation, median etc. Let’s just demonstrate one here.

**array = np.array([ [1,2,3],[2,3,5] ])**

**array.sum()** returns **16** as the output. It sums up all the elements in the array.

N.B.: To sum along the row, use **array.sum(1);** and to sum along the columns, use **array.sum(0).**

Pandas is a set of data analysis libraries built around NumPy. Pandas provides two main datatypes which are: series and dataframes. These essentially add labeling and indexing to the raw vectors and matrices, but also many additional methods for handling the data.  It provides high-performance, easy to use structures and data analysis tools. Here are some of the operations pandas can do for you

First of all, like we did for numpy, pandas also has to imported manually into our working space. We do this by executing the command: “import pandas as pd”. We want to work with Series and Dataframes. For easier implementation of our codes (to avoid the repetition of the use of pd), we can simply import Series and DataFrame from pandas by the command: **”from pandas import Series”** or, **“from pandas import DataFrame”**.

A series is created by passing a list of values, letting pandas create a default integer index e.g.

**Series([1,2,3,’John’, np.nan])** returns

1. ***1***
2. ***2***
3. ***3***
4. ***John***
5. ***NaN***

A dataframe can also be created by converting a dictionary e.g.

**sample = {'company': ['A','B'], 'Profit' : [1000, 5000]}**

**sample\_dataframe = DataFrame(sample)**

The code above returns

***company Profit***

***0 A 1000***

***1 B 5000***

Now, sample has been transformed to a dataframe named sample\_dataframe. We can run the following codes using pandas library.

sample\_dataframe.index : returns the indices of the dataframe

sample\_dataframe.values : returns the values of the indices

sample\_dataframe.reindex: create a new index from exiting index e.g.

**sample\_dataframe.reindex([0,1,2])** returns

***company Profit***

***0 A 1000.0***

***1 B 5000.0***

***2 NaN NaN***

The column argument can also be passed into the sample dataframe such that we have.

**samp\_df = sample\_dataframe.reindex([0,1,2])**

**print(samp\_df.reindex(columns = ['company','Profit','Year'] ) )**

The code above returns

**company Profit Year**

**0 A 1000.0 NaN**

**1 B 5000.0 NaN**

**2 NaN NaN NaN**

Note that it returns NaN for the newly created column, “Year”.

Columns can also be dropped from a dataframe by passing the code. We will drop a column from our dataframe, sample\_df, demonstrated in the code below.

**samp\_df.drop('company', axis =1)** returns

**Profit**

**0 1000.0**

**1 5000.0**

**2 NaN**

Note that the column “year” was not displayed because pandas assumed the column is irrelevant to the next course of action. Also, did you see ‘axis = 1’? What does it mean? It means that we are basically telling pandas to drop it along the columns. Axis = 0 means pandas should drop it along the row. The code can generate error if pandas can’t find the specified variable in the row or column. e.g.

**samp\_df.drop('company', axis =0)** would generate error because there is no “company” along the rows.

Also, note that if axis is not defined, it is automatically taken to be zero.

**sample\_df.dropna()** drops all rows containing null values.

**sample\_df.fillna()** is used for filling all null values in a dataframe.

The dropna function also accepts the argument axis = 1 and axis = 0. “axis =1” stands for column-wise dropping while “axis = 0” stands for row-wise dropping.

You can explore more by checking the official pandas documentation. I wish you an enjoyable ride!