Sprint 2_Numerical Programming

June 7, 2022

1 IT Academy - Data Science

1.1 S02 T04: Estructura de dades

1.1.1 *Exercise 1*

Create a function that, given a one-dimensional array, gives you a basic statistical summary of the data. If it detects that the array is larger than one dimension, it should display an error message.

[1]:

```python
#import requested library
import numpy as np

#create function
def statistics(a):
    if a.ndim == 1:
        print ("The mean of the array is:", np.mean(a))
        print ("The median of the array is:", np.median(a))
        print ("The standard deviation of the array is:", np.std(a, dtype='f2'))
        print ("The variance of the array is:", np.var(a, dtype='f2'))
    else: print("Dimensions of array exceed 1.")
```

[2]:

```python
#create random array of 10 elements, with values between [0,100]
arr1 = np.random.randint(1, 100, 10)
print("Random array:", arr1)

#call function
statistics(arr1)
```

```
Random array: [34 43 55 54 88 54 14 89 22 47]
The mean of the array is: 50.0
The median of the array is: 50.5
The standard deviation of the array is: 23.27
The variance of the array is: 541.5
```

[4]:

```python
#create random array 2D
arr2 = np.random.random((4,2))
print("Random array:\n", arr2)
#calculate statistics: error message.
statistics(arr2)
```

```
Random array:
 [[0.4028788  0.49463848]
 [0.52184416 0.35650548]
 [0.64412505 0.56134421]
 [0.21950207 0.03413224]]
Dimensions of array exceed 1.
```

1.1.2 *Exercise 2*

Create a function that generates an NxN square matrix of random numbers between 0 and 100.

[10]:
```
#create function
def squareMatrix ():
    #random int to generate dimension of square matrix with values between␣
  ↪[1,20]
    n = np.random.randint(1, 20)
    #random square matrix of dimension NxN with values between [0,100]
    a = np.random.randint(0, 100, size=(n,n))
    print ("The square matrix of size",n,":\n", a)

#call function
squareMatrix()
```

```
The square matrix of size 8 :
 [[23 45 65  4 42 41 18 92]
 [10 23 77 78 62 57  9 24]
 [31 64  8 87 10 21 51 13]
 [ 3 84 92 63 33 56 54 59]
 [ 1 16 41 48 96 25 26 17]
 [28 99 15 54 13 78 52  3]
 [56 68  5  2 55 63 21 60]
 [69  5 33 49 80 42 86 27]]
```

1.1.3 *Exercise 3*

Create a function that, given a two-dimensional table, calculates the totals per row and the totals per column.

[12]:
```
#create function
def totalAxis (a):
    print("The column sum of array is:", np.sum(a, axis = 0))
    print("The row sum of array is:\n", np.sum(a, axis = 1).reshape(n,1))
```

[14]:
```
#random int to generate dimensions of  matrix NxM with values between [1,10]
n = np.random.randint(1, 10)
m = np.random.randint(1, 10)

#random square matrix of dimension NxM with values between [0,100]
```

```
arr3 = np.random.randint(0, 100, size=(n,m))
print(arr3)
```

```
[[56 35  7 50 92]
 [53 90 47 54  1]
 [40 51 60 65  4]
 [ 4 90 48 41 62]
 [ 4 76 30 67 97]
 [86 15 99 90 28]]
```

[15]:

```
#call function
totalAxis(arr3)
```

```
The column sum of array is: [243 357 291 367 284]
The row sum of array is:
 [[240]
 [245]
 [220]
 [245]
 [274]
 [318]]
```

1.1.4 *Exercise 4*

Manually implements a function that calculates the correlation coefficient. Learn about its uses and interpretation.

[20]:

```
import math

def pearson_def(x, y):
    assert len(x) == len(y)
    n = len(x)
    assert n > 0
    avg_x = np.mean(x)
    avg_y = np.mean(y)
    diffprod = 0
    xdiff2 = 0
    ydiff2 = 0
    for idx in range(n):
        xdiff = x[idx] - avg_x
        ydiff = y[idx] - avg_y
        diffprod += xdiff * ydiff
        xdiff2 += xdiff * xdiff
        ydiff2 += ydiff * ydiff

    return np.array(diffprod / math.sqrt(xdiff2 * ydiff2), dtype='f2')
```

[21]:

```
#random int to generate dimension of  matrix Nx1 with values between [1,10]
n = np.random.randint(1, 10)
#random square matrix of dimension Nx1 with values between [0,100]
arr4 = np.random.rand(n)
arr5 = np.random.rand(n)
```

[26]:

```
#call function
print("The correlation coefficient is:",pearson_def(arr4, arr5))
```

```
The correlation coefficient is: -0.348
```

[29]:

```
#check result with in-built function of python
corrcoeff = np.array(np.corrcoef([arr4, arr5]),dtype='f2')
print(corrcoeff[0,1])
```

```
-0.348
```