S02_T04_numerical_programming

February 1, 2022

1 IT Academy - Data Science Itinerary

1.1 S02-T04: numerical programming

1 :

Create a function that, given an array of one dimension, gives you a basic statistical summary

[2]:
```
import numpy as np


def array_summary(array):
    """array_summary() is a fuction that given an array as an input,
    it return a basic statistics summary"""

    if array.ndim != 1:
        print("Error: array has more than 1 dimension")
    else:
        print("The mean is", np.mean(array))
        print("The max value is", np.amax(array))
        print("The min value is", np.amin(array))
        print("The median value is",np.median(array))
        print("The standard deviation is", np.std(array))
        print ("The 1st quartile:",np.quantile(array,0.25))
        print ("The 3rd quartile:",np.quantile(array,0.75))
```

[3]:
```
# "array_test" is a one-dimensional array. Lets test our fuction with it

array_test = np.array([10,8,10,8,8,4,64,18], dtype = int)
```

[3]:
```
8
```

[3]:
```
array_summary(array_test)
```

```
The mean is 16.25
The max value is 64
The min value is 4
The median value is 9.0
The standard deviation is 18.423829677892705
```

```
The 1st quartile: 8.0
The 3rd quartile: 12.0
```

[13]:
```
# test the function with more than one dimension.

array_test_2 = np.random.randint(101, size = (2,2))

print(array_test_2)

array_summary(array_test_2)
```

```
[[44 43]
 [23 83]]
Error: array has more than 1 dimension
```

2 :

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

[15]:
```
import random

def n_array(N):
    """ n_array() is a fuction that return a N x N array of
    random numbers between 0 and 100"""
    return np.random.randint(101, size = (N,N))
```

[24]:
```
print(n_array(2))
```

```
[[85 94]
 [47 17]]
```

[25]:
```
print(n_array(3))
```

```
[[80 61  3]
 [87 92 29]
 [88 71 75]]
```

3 :

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

[52]:
```
def total_Row_Column(array):
    """total_Row_Column() is a fuction that given a two-dimensional array an␣
 ↪input,
    calculates the totals per row and the totals per column."""

    if array.ndim != 2:
        print("The argument of this function must be a two-dimensional array.")

    else:
```

```
        print("Total value of each column is:",np.sum(array, axis=1))
        print("Total value of each row is:",np.sum(array, axis=0))
```

- Create an array of two dimension in order to test the function:

[59]:
```
array_test = np.random.randint(101, size = (2,2))

total_Row_Column(array_test)
```

```
Total value of each column is: [ 31 103]
Total value of each row is: [ 18 116]
```

- Test the fuction with a one-dimensional array:

[58]:
```
array_test_2 = np.array([10,8,10,8,8,4,64,18], dtype = int)

total_Row_Column(array_test_2)
```

```
The argument of this function must be a two-dimensional array.
```

4 :

```
Manually implement a function that calculates the correlation coefficient.
```

[7]:
```
def coef_corre(X,Y):
    """ coef_corre() is a fuction that given a two one-dimensional arrays as␣
 ↪input (X,Y)
    calculates and returns the Pearson's correlation coefficient"""

    if X.ndim == 1 and Y.ndim == 1:
        if len(X) == len(Y):

            return (np.sum((X-np.mean(X))*(Y-np.mean(Y))))/(np.sqrt(np.
 ↪sum((X-np.mean(X))**2))*np.sqrt(np.sum((Y-np.mean(Y))**2)))

        else:
            print ("Error:The arguments of this function must have the same␣
 ↪numbers of elements")

    else:
        print("Error:The arguments of this function must be two one-dimensional␣
 ↪arrays")
```

- Test the fuction:

[8]:
```
age = np.array([43,21,25,42,57,59])
income = np.array([1500,800,1200,4500,4650,4800])
```

[9]:
```
coef_corre(age,income)
```

[9]:
```
0.8594755764164334
```

[10]:
```
#The NumPy function

np.corrcoef(age,income)
```

[10]:
```
array([[1.        , 0.85947558],
       [0.85947558, 1.        ]])
```

both results are the same: 0.8594

[ ]:
```

```