Assignment1_z-scores_Ayushi

September 11, 2020

1 Comparing Data points using z-scores

Z-scores or standard scores are an essential statistic tool for analysts which helps provide comparative insight into scores from different normal distribution. While standard deviations define the line along which a particular data point lies, z-scores calculate how much a given value differs from the standard deviation of a set of values.

Z-scores can be calculated by subtracting the Poplutaion Mean μ (or Sample mean $\bar{x}$, depending upon the data at hand) of the data points from the individual value and then dividing the result by the standard deviation. When translated into a formula it becomes:

$$z = \frac{x_i - \mu}{\sigma}$$

The application of this statistic are widespread in the field of data analysis. For example: In the world of finance, traders z-score to indicate the volatility of the returs. The higher the score the more unexpected the results.

The code below expects the user to provide 2 inputs a numpy array and a scalar value denoting the column index if the user wants to calculate the z-score of a specified column. The program makes use of other user defined functions such as sum, mean and standard deviation of the array input to finally calculate z-scores.

2 Python Code

[1]:
```
import numpy as np

def summation(column):
    sum = 0
    for element in column:
        sum += element
    return sum

def mean(column):
    return summation(column) / column.shape[0]

def standard_dev(column):
    return (sum((column - mean(column))**2)/(column.shape[0]))**(0.5)
```

```
def z_score(array, col_index = None):
    """
    @param array it is an array entered by the user on which z_score␣
↪normalization will be calculated
    @param  col_index it is column index defined to calculate z_scores of a␣
↪specific column entered by the user.
    @return array array after calculating zscore either for entire array or␣
↪specified column
    """
    # Using try statement to handle selected exceptions in user input.
    #instances specified with assert statements
    try:
        # asserting the user to input an array for z_score calculations
        assert type(array) == np.ndarray,'Input should be an array'
        # asserting the user to input a two dimensional array only
        assert len(array.shape) == 2, 'Shape of array should be 2 dimensional'
        # asserting the user to input only a positive integer for column index
        assert (type(col_index) == int and col_index >= 0) or col_index ==␣
↪None, 'Column Index should be none or a positive integer'
        # asserting the user to input a column index value less than the number␣
↪of columns in array because Python starts indexing from zero
        assert col_index == None or col_index < array.shape[1],'Column Index␣
↪should be lesser than the number of columns in the array because Python␣
↪starts indexing from zero'

        array = np.asarray(array,dtype = float)

        if col_index is None:
            for column_index in range(array.shape[1]):
                array[:,column_index] = z_score_column(array,column_index)
        else:
            array[:,col_index] = z_score_column(array,col_index)
        return array

    # generates error message when entered parameters do not meet the given␣
↪condition
    except AssertionError as error:
        print(str(error))

def z_score_column(array, col_index):
    column = array[:,col_index]
    column_mean = mean(column)
    column_standard_dev  = standard_dev(column)
    temp = (column - column_mean) / column_standard_dev
    return temp
```

3 Outputs Generated based on User Input

[3]:
```
x1 = np.array([[4,3,12],[1,5,20],[1,2,3],[10,20,40],[7,2,44]])
x2 = 3
x3 = 6
x4 = 'two'
print(z_score(x1))
```

```
[[-0.17149859 -0.49363572 -0.74427584]
 [-1.02899151 -0.20326177 -0.23968205]
 [-1.02899151 -0.63882269 -1.31194386]
 [ 1.54348727  1.97454287  1.02180243]
 [ 0.68599434 -0.63882269  1.27409932]]
```

[4]:
```
print(z_score(x1,0))
```

```
[[-0.17149859  3.         12.        ]
 [-1.02899151  5.         20.        ]
 [-1.02899151  2.          3.        ]
 [ 1.54348727 20.         40.        ]
 [ 0.68599434  2.         44.        ]]
```

The above input parameters calculate z-scores and transform only the first column values according to the input provided by the user keeping the rest of the column values intact.

[13]:
```
print(z_score(x1,x2))
```

```
Column Index should be lesser than the number of columns in the array because
Python starts indexing from zero
None
```

[14]:
```
print(z_score(x1,x3))
```

```
Column Index should be lesser than the number of columns in the array because
Python starts indexing from zero
None
```

[6]:
```
print(z_score(x2,x3))
```

```
Input should be an array
None
```

[7]:
```
print(z_score(x1,x4))
```

```
Column Index should be none or a positive integer
None
```

4 Lessons Learned

This being my first assignment, there were a number of glitches in the first draft of the code and the outputs of the program were not meeting the standards set. It took a lot of trial and error, time, practice and online help for debugging to come up with a desirable program. My takeaways from this assignments were:

4.0.1 Always start by breaking down the logical sequence flow of the program

For Example: To calculate z_scores in the above program, the program first defines separate functions calculating other elements required by the formula starting with the basic sum of elements column wise to calculate the mean and the standard deviation and then moves on to calculate z_scores once all the the other functions were running properly.

4.0.2 There are plenty of approaches to do things

For Example: To display familiarity and proficiency with basic Python, the above code separately calculates with different functions, the sum, mean and standard deviation of the array. These basic calculations could also have been done using np.mean and np.std functions which are inbuilt in the numpy module. Using these functions, the code could have been shorter and more succinct.

4.0.3 User Friendliness is important

It is important to remember that the program may have different users and should therefore be user friendly. The above program is designed to take 2 inputs - a numpy array and a scalar. To compel the user to meet the specified conditions for input paratmeters, Assert statement was used to test if a condition in the code returns True, if not, the program will raise an AssertionError.

4.0.4 Research is paramount

There are a billion statements and functions in a coding language and it is difficult to recall a lot of them without enough practice. Any beginner should spend enough time on researching appropiate statements to generate a desired output. For Example: While working with the program, I was struggling to find a way to just display an error message for the user and not generate an error. I came across a number of interesting functions using which one could extract, format and print stack traces of Python programs a traceback error message for the user if the user makes use of the traceback package that but to achieve the desired result, I made use of the the try statement with an except clause. The statement did generate assertion errors mentioned in the assert statements which were handled by the use of except clause.

Coding in any language may require a lot of trial and error to get your imagined output or even to fulfill the basic requirements of the assignment. As a neophyte it is important to always give yourself enough time to complete your assignment to not be stuck working till the end of deadlines.