UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

MULTIVARIATE STATISTICAL ANALYSIS

Report Practice 01
NumPy

Lecturers Lý Quốc Ngọc
Nguyễn Mạnh Hùng
Phạm Thanh Tùng

Student Võ Nguyễn Hoàng Kim
21127090

PHỤ LỤC

I. SELF-ASSESSMENT FORM

Features		Level of completion
Mean, Median	Mean	100%
	Median	100%
Order Statistics	Max	100%
	Min	100%
	Range	100%
Variance & Stand Deviation	Variance	100%
	Standard Deviation	100%
Correlation	Correlation	100%

II. LIST OF THE FUNCTIONS

Below are the functions that I have used in practical exercise 01 to meet the functionalities stated in the assignment (all these functions are supported by NumPy). Each of function used in the program is accompanied by a proof image:

Note: *Assume X is an array initialized by the* ***np.array()*** *function with values initialized by default in the source code.*

- Mean
  - In the normal case, we use **np.mean()** to compute the Mean of X.
  - Handle with NaN: In this case, we use **np.nanmean()** to compute the Mean of X.
- Median
  - In the normal case, we use **np.median()** to compute the Median of X.
  - Handle with NaN: In this case, we use **np.nanmedian()** to compute the Median of X.
- Max
  - In the normal case, we use **np.max()** to get the Max of X.
  - Handle with NaN: In this case, we use **np.nanmax()** to get the Max of X.
- Min
  - In the normal case, we use **np.min()** to get the Min of X.
  - Handle with NaN: In this case, we use **np.nanmin()** to get the Min of X.
- Range
  - In the normal case, we use **np.ptp()** to compute the Range of X.
- Variance
  - In the normal case, we use **np.var()** to compute the Variance of X.
  - Handle with NaN: In this case, we use **np.nanvar()** to compute the Variance of X.
- Standard Deviation
  - In the normal case, we use **np.std ()** to compute the Standard Deviation of X.

- Handle with NaN: In this case, we use **np.nanstd()** to compute the Standard Deviation of X.

- Correlation
  - In the normal case, we use **np.corrcoef()** to compute the Matrix Correlation of X.

III. IMPLEMENTATION

Note: *Assume X is an **input array** initialized by the **np.array()** function with values initialized by default in the source code:*

1. Mean and Median

a. Mean

- Implement:
  - With **np.mean(),** we can calculate the Mean value of X. It returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis [1]:
    - In this program, *axis* takes values 0 or 1, along which the Means are computed.
      - If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before [1].
      - If *axis = 0*, the Means are computed along the Y-axis (vertical direction).
      - On the contrary, it will be computed along the X-axis (horizontal).
      - The default is to compute the Mean of the flattened array.
  - For single precision, the data type (dtype) plays an important role in the accuracy of the output result. As in the example below, with the initial data type being **np.float32()**, the returned result for the mean value is inaccurate. To improve this, we can use **np.float64()** instead. [1]
  - In cases where there are NaN (Not a Number) values in the dataset, we can use **np.nanmean()** to handle them.
- Result:

  Below are the results obtained from the experiment using the **np.mean()** and **np.nanmean()** functions with the issues mentioned above
  - In normal case:

```
X =  [[42  4]
 [52  5]
 [48  4]
 [58  3]]
Mean X:  27.0
Mean X with axis = 0:  [50.  4.]
Mean X with axis = 1:  [23.  28.5 26.  30.5]
```

  - Exist NaN in dataset:

```
X =  [ 2. nan  5.  9.  7.]
Mean =  nan
nan Mean =  5.75
```

  - Single precision & Double precision

```
a =  [[1.  1.  1.  ... 1.  1.  1. ]
 [0.1 0.1 0.1 ... 0.1 0.1 0.1]]

a.shape:  (2, 262144)
mean a (with f32) =  0.54999924
mean a (with f64) =  0.5500000007450581
```

b. Median

- Implement:
  - For an *input-array* with a length of N, **np.median()** sorts the elements of this array and copies them into another array (*input-array-sorted*). Based on the sorted array, it determines the median value by taking the middle value of the array (the value of the element at position (N-1)/2 when N is odd; the average of the two middle values when N is even) [2].
  - With **np.median(),** we can calculate the Median value of X along the specified axis [2]:
    - In this program, *axis* takes values 0 or 1, along which the Medians are computed.
      - If *axis = 0*, the Medians are computed along the Y-axis (vertical direction).
      - On the contrary, it will be computed along the X-axis (horizontal direction).
      - The default is to compute the Median of the flattened array.
  - In cases where there are NaN (Not a Number) values in the dataset, we can use **np.nanmean()** to handle them.
- Result:

  Below are the results obtained from the experiment using the **np.mean()** and **np.nanmean()** functions with the issues mentioned above
  - In normal case:

```
X =  [[42  4]
 [52  5]
 [48  4]
 [58  3]]
Median =  23.5
Median (axis = 0) =  [50.  4.]
Median (axis = 1) =  [23.  28.5 26.  30.5]
```

  - Exist NaN in dataset:

```
X =  [ 2. nan  5.  9.  7.]
Median =  nan
nan Median =  6.0
```

2. Order statistics

a. Max

- Implement:
  - With **np.max(),** we can get the Maximum of X (in array) or Maximum along an axis.
    - In this program, *axis* takes values 0 or 1, along which to operate [3]:
      - If this is a tuple of ints, the maximum is selected over multiple axes, instead of a single axis or all the axes as before.
      - If we set *axis = 0*, the maximum is selected along the Y-axis (vertical direction).
      - On the contrary, it will be selected along the X-axis (horizontal).
      - By default, flattened input is used.
  - In cases where there are NaN (Not a Number) values in the dataset, we can use **np.nanmax()** to handle them.
- Result:

  Below are the results obtained from the experiment using the **np.max()** and **np.nanmax()** functions with the issues mentioned above
  - In normal case:

```
X =  [[42  4]
 [52  5]
 [48  4]
 [58  3]]
Max =  58
Max (axix = 0):  [58  5]
Max (axix = 1):  [42 52 48 58]
```

- Exist NaN in dataset:

```
X =  [[42.  4.]
 [52.  5.]
 [nan  4.]
 [58. nan]]

Max =  nan
nan Max =  58.0
```

b. Min

- Implement:
  - With **np.min(),** we can get the Minimum of X (in array) or Minimum along an axis
    - In this program, *Axis* takes values 0 or 1, along which to operate [4]:
      - If this is a tuple of ints, the minimum is selected over multiple axes, instead of a single axis or all the axes as before.
      - If we set *axis = 0*, the minimum is selected along the Y-axis (vertical direction).
      - On the contrary, it will be selected along the X-axis (horizontal).
      - By default, flattened input is used.
  - In cases where there are NaN (Not a Number) values in the dataset, we can use **np.nanmin()** to handle them.
- Result:

  Below are the results obtained from the experiment using the **np.min()** and **np.nanmin()** functions with the issues mentioned above
  - In normal case:

```
X =  [[42  4]
 [52  5]
 [48  4]
 [58  3]]
Min =  3
Min (axix = 0):  [42  3]
Min (axix = 1):  [4 5 4 3]
```

  - Exist NaN in dataset:

```
X =  [[42.  4.]
 [52.  5.]
 [nan  4.]
 [58. nan]]

Min =  nan
nan Min =  4.0
```

c. Range

- Implement:
  - Range in this term means the difference between max value and min value along the axis in the dataset. With **np.ptp(),** we can get the Range of value of X.

        - In this program, *axis* takes values 0 or 1, along which to find the peek [5]:
            - If this is a tuple of ints, a reduction is performed on multiple axes, instead of a single axis or all the axes as before.
            - If *axis = 0*, the range is computed along the Y-axis (vertical direction).
            - On the contrary, it will be computed along the X-axis (horizontal).
            - By default, flattened input is used.
    - NaN cases are not supported by NumPy in computing the range of values of X. It will return NaN as the result.
- Result:
  Below are the results obtained from the experiment using the **np.ptp()** function with the issues mentioned above
    - In normal case:

```
X =  [[42  4]
 [52  5]
 [48  4]
 [58  3]]

Range =  55
Range (axis = 0):  [16  2]
Range (axis = 1):  [38 47 44 55]
```

    - Exist NaN in dataset:

```
X =  [[42.  4.]
 [52.  5.]
 [48.  4.]
 [nan  3.]]

Range =  nan
Range (axis = 0):  [nan  2.]
Range (axis = 1):  [38. 47. 44. nan]
```

3. Variance and Standard Deviation

a. Variance

- Implement:
    - With **np.var(),** we can compute the Variance value of X along the specified axis:
        - In this program, *axis* takes values 0 or 1, along which the variance is computed [6]:
            - If this is a tuple of ints, a variance is performed over multiple axes, instead of a single axis or all the axes as before.
            - If *axis = 0*, the variance is computed along the Y-axis (vertical direction).
            - On the contrary, it will be computed along the X-axis (horizontal).
            - By default, flattened input is used.
    - In cases where there are NaN (Not a Number) values in the dataset, we can use **np.nanvar()** to handle them.
- Result:
  Below are the results obtained from the experiment using the **np.var()** and **np.nanvar()** functions with the issues mentioned above
    - In normal case:

```
X =  [[42  4]
 [52  5]
 [48  4]
 [58  3]]

Variance:  546.25
Variance (axis = 0):  [34.   0.5]
Variance (axis = 1):  [361.   552.25 484.   756.25]
```

- Exist NaN in dataset:

```
X =  [[42.  4.]
 [52.  5.]
 [nan  4.]
 [58. nan]]

Variance:  nan
NaN Variance :  558.5833333333334
```

b. Standard Deviation

- Implement:
  - With **np.std(),** we can compute the Standard Deviation value of X along the specified *axis*:
    - In this program, *axis* takes values 0 or 1, along which the variance is computed [7]:
      - If this is a tuple of ints, a standard deviation is performed over multiple axes, instead of a single axis or all the axes as before.
      - If we set *axis = 0*, the variance is computed along the Y-axis (vertical direction).
      - On the contrary, it will be computed along the X-axis (horizontal).
      - By default, flattened input is used.
  - In cases where there are NaN (Not a Number) values in the dataset, we can use **np.nanstd()** to handle them.
- Result:

  Below are the results obtained from the experiment using the **np.std()** and **np.nanstd()** functions with the issues mentioned above
  - In normal case:

```
X =  [[42  4]
 [52  5]
 [48  4]
 [58  3]]

Standard Deviation:  23.371991785040485
Standard Deviation (axis = 0):  [5.83095189 0.70710678]
Standard Deviation (axis = 1):  [19.  23.5 22.  27.5]
```

  - Exist NaN in dataset:

```
X =  [[42.  4.]
 [52.  5.]
 [nan  4.]
 [58. nan]]

Standard Deviation:  nan
NaN Standard Deviation:  23.63436763134003
```

4. Correlation

a. Correlation

- Implement:

- With **np.corrcoef(),** we can get the Correlation Coefficients of X. Elements' values in the Correlation are between -1 and 1. In this function, we need to pay attention to 2 important parameters that are commonly used [8]:
  - *input-array*: as default, it is a 1-D or 2-D array containing multiple variables and observations. Each row of x represents a variable, and each column a single observation of all those variables.
  - *rowvar:* plays an important role, affecting how the input matrix is considered when computing the correlation coefficient matrix.
    - If *rowvar = True* (default), then each row represents a variable, with observations in the columns.
    - Otherwise, the relationship is transposed: each column represents a variable, while the rows contain observations.
- NaN cases are not supported by NumPy in computing the range of values of X. It will return NaN as the result.

- Result:
  Below are the results obtained from the experiment using the **np. corrcoef()** functions (with *rowvar = False)* with the issues mentioned above
  - In normal case:

```
X =  [[42  4]
 [52  5]
 [48  4]
 [58  3]]

Correlation:
 [[ 1.         -0.36380344]
 [-0.36380344  1.        ]]
```

  - Exist NaN in dataset:

```
X =  [[42.  4.]
 [52.  5.]
 [48. nan]
 [58.  3.]]

Correlation:
 [[ 1. nan]
 [nan nan]]
```

IV. SUMARIZE

- In summary, in practice 01, we used various NumPy library functions to compute basic statistical quantities with different value types such as float32, float64, NaN, etc.
- Regarding NaN values, it can be observed that NumPy provides support for computation with datasets containing this data type, and the approach is generally consistent: ignoring (or removing) NaN values and compute the needed value based on the remaining values.
  - Example: To compute Means value as the result above (with *axis = none*), **np.nanmean()** has ignored the presence of NaN values in X (considering them non-existent in X), and computed the mean value based on the remaining numbers, which are 2, 5, 7, 9.

```
X =  [ 2. nan  5.  9.  7.]
Median =  nan
nan Median =  6.0
```

- However, not all quantities are supported by NumPy for computation with NaN, such as Correlation and Range. If we still attempt to use these functions with a dataset containing NaN values, the resulting NaN outputs will not be accurate.

V. REFFERENCES

[1] NumPy Developers, "numpy.mean," NumPy, [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.mean.html.

[2] NumPy Developers, "numpy.median," NumPy, [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.median.html.

[3] NumPy Developers, "numpy.max," NumPy, [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.max.html.

[4] Numpy Developers, "numpy.min," NumPy, [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.min.html.

[5] NumPy Developers, "numpy.ptp," NumPy, [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.ptp.html.

[6] NumPy Developers, "numpy.var," NumPy, [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.var.html.

[7] NumPy Developers, "numpy.std," NumPy, [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.std.html.

[8] NumPy Developers, "numpy.corrcoef," NumPy, [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html.