SANJEEV SHARMA
sanjivksharma@yahoo.com

Informatics Practices(New)

CLASS XII Code No. 065 -2019-20

Unit 1: Data Handling (DH-2)

Numeric Python or Numpy

Before using Numeric Python let us quickly revise the List Data structure:

A list can hold any type and can hold different types of elements at the same time. We can also change, add and remove elements from the List.

But one operation that is sometime important for us is not possible in with the List. That operation is to perform calculation on all the elements of the List at once. We have to perform elements by element operation on the List, which is a slow process.

Let's take two List of marks and find the sum of elements of two Lists

```
First_List = [20,25,30,40]
Second_List = [25,15,35,30]
Third_List= First_List  + Second_List
print(Third_List)
```

Output :

```
[25, 15, 35, 30, 20, 25, 30, 40]
```

The above operation will combine these two Lists and make a new List. Instead of adding each element of first list with each element of second list this operation will simply combine both list.

Let's take one more example and try to use multiplication operator with two Lists:

```
Values1 = [20,25,30,40]
Values2 = [25,15,35,30]
print(Values1 * Values2)
```

The above operation will produce an error

TypeError: can't multiply sequence by non-int of type 'list'

We can solve this by going through each list element one after the other, and finding the multiplication/sum of each element of both Lists separately, but this is terribly inefficient and tiresome for large Lists.

Numpy Array :

More elegant solution of the above problem is to use NumPy, or Numeric Python. It's a Python package that, among others, provides an alternative to the regular python List: the Numpy array. The Numpy array is pretty similar to a regular Python List, but has one additional feature:

We can perform calculations over entire arrays at once. It's really easy, and fast as well.

To actually use Numpy array in our Python script, we have to import the numpy package like this.

```
import numpy as np
```

Example 1:

```
# Python script with the use of numpy package
import numpy as np    # Import Numeric Python Library
# Creation of Two List
Values1 = [20,25,30,40]
Values2 = [25,15,35,30]

print("\n First we make Two new Numpy Arrays with above Two Lists")
Values3=np.array(Values1)
Values4=np.array(Values2)
```

```
print("\n Use of multiplication operator with Numpy Arrays")
print(Values3 * Values4)

print("\n The above operation will multiply each element of First Array with each element of the Second Array")
```

Output :

```
First we make Two new Numpy Arrays with above Two Lists

 Use of multiplication operator with Arrays
[ 500  375 1050 1200]

 The above operation will multiply each element of First Array wit
h each element of the Second Array
```

The calculation is performed element-wise. The first element of array one is multiplied with the first element of second array and so on.

If our list has different types of elements in it for example:

```
Values1 = [1 , "String Value", True]
Element 1 is an Integer
Element 2 is a String
Element 3 is a Boolean
```

And if we create a numpy array with the above list

```
My_array=np.array(Values1)
```

The My_array will have all three elements of String Type, this means that If our List contains elements with different datatypes the numpy array will convert the datatype of all elements into String.

Example :

```
# Python script with the use of different elements in the List
```

```
import numpy as np
# Creation of List with three different elements:
#1. integer 2. String 3. Boolean
Values1 = [1, "String Value", True]
print("Original List")
print(Values1)
print("\n Use of numpy array\n")
Values3=np.array(Values1)
print("Above statement will create an numpy array with string data type of all elements ")
print(Values3)
```

Output:

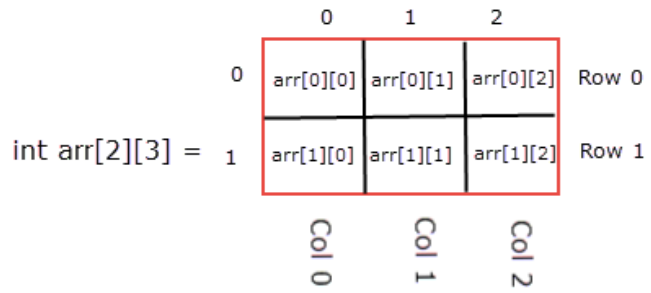
```
Original List
[1, 'String Value', True]
Use of numpy array
Above statement will create an numpy array with string data type of all elements
['1' 'String Value' 'True']
```

The resulting Numpy array will contain a single type, String in this case. Numpy array is simply a new kind of Python type, like the float, string and List types. We can work with Numpy arrays pretty much the same as we can with regular Python Lists.

When we want to get elements from our array, we can use square brackets. Suppose we want to get the second element from the Value1 array we can write the statement like this:

```
Element = Value1[1]   # Index starts at 0
```

2D Numpy Arrays :



A conceptual representation of 2-D array

A 2D arrays is a collection of Rows and Columns. Let's create a 2D array using numpy :

```
TwoD_array = np.array([[10 , 20 , 30] , [ 40 , 50 ,60] ])
```

If we print this array using the statement

```
print(TwoD_array)
```

The output will be

```
[[10 20 30]
 [40 50 60]]
```

We can access each element of the 2D array using square brackets and index of the elements.

```
print(TwoD_array[0])  # This will print the First row
print(TwoD_array[0][0])  # This will print the First element of First row
print(TwoD_array[1])  # This will print the Second row
print(TwoD_array[1][2])  # This will print the Third element of Second row
```

Subsetting:

```
print(TwoD_array[1,1:3) #This will print the Second and Third element of Second row
 print(TwoD_array[:,0:2)(# This will print the First and Second element of all rows


print(TwoD_array[1,:) #This will print the Second  row
```

Example 1:

```
# 2D Array

import numpy as np

# Creation of  2D Array

TwoD_array = np.array([[10 , 20 , 30] , [ 40 , 50 ,60] ]) # use of nested lists


print("\nOriginal  2D Array")

print(TwoD_array)

print("\nPrinting the Size of the Array using shape attribute\n")

print(TwoD_array.shape)  #  use  of  shape  attribute  to  find  the number of roww and columns in each row


print("\nAccessing Individual elements from the 2D Array\n")

# This will print the First row

print("\nThis will print the First row",TwoD_array[0])


# This will print the First element of First row

print("\nThis  will  print  the  First  element  of  First row",TwoD_array[0][0])


# This will print the Second row

print("\nThis will print the Second row",TwoD_array[1])

# This will print the Third element of Second row
```

Output:

```
Original  2D Array

[[10 20 30]
```

```
 [40 50 60]]
Printing the Size of the Array using shape attribute
(2, 3)
Accessing Individual elements from the 2D Array
This will print the First row [10 20 30]
This will print the First element of First row 10
This will print the Second row [40 50 60]
This will print the Third element of Second row 60
This will also print the Third element of Second row 60
```

Example 2:

```
# 2D Array and Subsetting
import numpy as np

# Creation of  2D Array
TwoD_array = np.array([[10 , 20 , 30] , [ 40 , 50 ,60] ]) # use of nested lists

print("\nOriginal  2D Array")
print(TwoD_array)

print("\nUse of Subsetting to access elements\n")
# This will print the Second and Third element of Second row
print("\nThis will print the Second and Third element of Second row",TwoD_array[1,1:3])
 # This will print the First and Second element of all rows
print("\nThis will print the First and Second element of all rows",TwoD_array[:,0:2])
# This will print the Second row
```

```
print("\nThis will print the Second  row",TwoD_array[1,:])
```

Output:

```
Original  2D Array
[[10 20 30]
 [40 50 60]]
Use of Subsetting to access elements
This will print the Second and Third element of Second row [50 60]
This will print the First and Second element of all rows [[10 20]
 [40 50]]
This will print the Second  row [40 50 60]
```

Slicing:

Slicing a 2D array means to extract elements from a previously created 2D array by specifying start, stop and step values.

The basic slice syntax is **i:j:k** where i is the starting index, j is the stopping index, and k is the step.

Negative i and j are interpreted as n + i and n + j where n is the number of elements. Negative k makes stepping go towards smaller indices.

Assume n is the number of elements being sliced. Then, if i is not given it defaults to 0 for k > 0 and n - 1 for k < 0. If j is not given it defaults to n for k > 0 and -n-1 for k < 0 . If k is not given it defaults to 1.

Note that :: is the same as : and means select all elements.

Example :

```
# 2D Array and Slicing
import numpy as np
# Creation of  2D Array
```

```
Myarray = np.array([10 , 20 , 30, 40 , 50 ,60] )


print("\nOriginal  1D Array")
print(Myarray)
print("\nUse of Slicing - START - STOP - STEP \n")
print("\nIn this example we are writing i, j , k in place of  START , STOP , STEP ")


print("\nPrinting starts from Second to Fourth element using Step value 1")
print(Myarray[1:4:1])
print("\n If i is not given it defaults to 0 for k > 0 and n - 1 for k < 0")
print(Myarray[:4:1])
print("\nIf j is not given it defaults to n-1")
print(Myarray[0::1])
print("\nIf i,j,k are not given it defaults to all elements")
print(Myarray[:])
print("\nWe can aslo use :: column to print all elements")
print(Myarray[::]) # We can aslo use :: column to print all elements
print("\nNegative i and j are interpreted as n + i and n + j where n is the number of elements")
print(Myarray[-4:-1:1])
print("\nNegative k makes stepping go towards smaller indices.")
print(Myarray[4:1:-1])
```

Output:

```
Original  1D Array
[10 20 30 40 50 60]
```

Use of Slicing - START - STOP - STEP

In this example we are writing i, j , k in place of START , STOP , STEP

Printing starts from Second to Fourth element using Step value 1

[20 30 40]

If i is not given it defaults to 0 for k > 0 and n - 1 for k < 0

[10 20 30 40]

If j is not given it defaults to n-1

[10 20 30 40 50 60]

If i,j,k are not given it defaults to all elements

[10 20 30 40 50 60]

We can aslo use :: column to print all elements

[10 20 30 40 50 60]

Negative i and j are interpreted as n + i and n + j where n is the number of elements

[30 40 50]

Negative k makes stepping go towards smaller indices.

[50 40 30]

Subsets:

Subset means to create a new numpy array or subset from the existing numpy array.

```
#  Use of slicing to create new Subset of Array
import numpy as np
# Creation of  2D Array
Myarray1D = np.array([10 , 20 , 30, 40 , 50 ,60] )
Myarray2D = np.array([[10 , 20 , 30], [40 , 50 ,60]] )


print("\nOriginal 1D Array")
```

```
print(Myarray1D)
print("\nOriginal 2D Array")
print(Myarray2D)
print("\n Subset of Myarray1D from Second to Fourth Element ")
subset=Myarray[1:4]
print(subset)
print("\n Subset of Myarray2D from Second row First Elements to Second Element ")
subset=Myarray2D[1,0:2]
print(subset)
```

Output:

```
Original 1D Array
[10 20 30 40 50 60]
Original 2D Array
[[10 20 30]
 [40 50 60]]
Use of slicing to create new Array
Subset of Myarray1D from Second to Fourth Element
[20 30 40]
Subset of Myarray2D from Second row First Elements to Second Element
[[40 50]]
```

Arithmetic operations on 2D arrays :

Some basic Arithmetic operations such as + ,- ,/ ,* ,% ,** etc.

```
# Arithmetic operations on 2D array
import numpy as np
# Creation of  2D Array
```

```
TwoD_array = np.array([[10 , 20 , 30] , [ 40 , 50 ,60] ]) # use of nested lists

TwoD_arraySecond = np.array([[2 , 4 , 6] , [ 8 , 10 ,12] ]) # use of nested lists

print("\nOriginal  2D Array")

print(TwoD_array)

print("\nUse of Adition Operator +")

print(TwoD_array +2)

print("\nUse of Substraction Operator -")

print(TwoD_array - 2)

print("\nUse of Multiplication Operator -")

print(TwoD_array * 2)

print("\nUse of Division Operator /")

print(TwoD_array / 2)

print("\nUse of Division Operator /")

print(TwoD_array % 2)

print("\nUse of Exponential Operator **")

print(TwoD_array ** 2)

print("\nUse of Addition Operator + to add Two 2D arrays")

print(TwoD_array + TwoD_arraySecond)
```

Output:

```
Original  2D Array

[[10 20 30]

 [40 50 60]]

Use of Adition Operator +

[[12 22 32]

 [42 52 62]]

Use of Substraction Operator -
```

```
[[ 8 18 28]
 [38 48 58]]
Use of Multiplication Operator -
[[ 20  40  60]
 [ 80 100 120]]
Use of Division Operator /
[[ 5. 10. 15.]
 [20. 25. 30.]]
Use of Division Operator /
[[0 0 0]
 [0 0 0]]
Use of Exponential Operator **
[[ 100  400  900]
 [1600 2500 3600]]
Use of Addition Operator + to add Two 2D arrays
[[12 24 36]
 [48 60 72]]
```

Covariance, Correlation and Linear Regression:

Covariance:

Covariance is a measure of relationship between 2 variables. It measures the degree of change in the variables, i.e. when one variable changes, will there be the same/a similar change in the other variable. This measure is scale dependent because it is not standardized. We can easily find the Covariance by using inbuilt functions cov().

Example:

```
#Python program to calculate Covariance
import pandas as pd
```

```
data = pd.DataFrame({
'name':['Naresh','Mohit','Rajesh','TC Soni'],
    'experience':[1,2,3,4],
'salary':[25000,35000,30000,20000],
'join_year' :[2007,2008,2007,2008]
})
print("Data for the Covariance is \n")
print(data)
print("\n Covariance of above Data \n")
print(data.cov())  # cov() function is used to find the Covariance
```

Output:

```
Data for the Covariance is
     name  experience  salary  join_year
0   Naresh           1   25000       2007
1    Mohit           2   35000       2008
2   Rajesh           3   30000       2007
3  TC Soni           4   20000       2008
 Covariance of above Data
             experience        salary  join_year
experience     1.666667 -3.333333e+03   0.333333
salary     -3333.333333  4.166667e+07   0.000000
join_year      0.333333  0.000000e+00   0.333333
```

Correlation:

Correlation is a measure of relationship between variables that is measured on a -1 to 1 scale. The closer the correlation value is to -1 or 1 the stronger the relationship, the closer to 0, the weaker the relationship. It measures how change in one variable is associated with change in another variable. We can easily find the Correlation by using inbuilt functions corr().

Example:

```
#Python program to calculate Correlation
import pandas as pd
data = pd.DataFrame({
'name':['Naresh','Mohit','Rajesh','Amit'],
    'experience':[1,2,3,4],
'salary':[25000,35000,30000,20000],
'join_year' :[2007,2008,2010,2008]
})
print("Data for the Correlation is \n")
print(data)
print("\n Correlation of above data \n")
# cor() function is used to find the Correlation
print(data.corr(method='pearson'))
# method parameter is optional default is pearson
```

Output:

```
Data for the Correlation is
     name  experience  salary  join_year
0  Naresh           1   25000       2007
1   Mohit           2   35000       2008
2  Rajesh           3   30000       2010
3    Amit           4   20000       2008
 Correlation of above data
                experience    salary  join_year
experience        1.000000 -0.400000   0.512989
salary           -0.400000  1.000000   0.307794
join_year         0.512989  0.307794   1.000000
```

Linear Regression:

The objective of a linear regression model is to find a relationship between one or more features (independent variables) and a continuous target variable (dependent variable)

Where can Linear Regression be used?

It is a very powerful technique and can be used to understand the factors that influence profitability. It can be used to forecast sales in the coming months by analyzing the sales data for previous months.

Example:

```
# A Python program demonstrating Linear Regression
import pandas as pd
# Use of matplotlib to Draw Diagram
import matplotlib.pyplot as plt
House_Loan = {'Year': [2018,2018,2018,2018,2017,2017,2017,2017],
        'Month': [12,11,10,9,8,12,11,9],
        'Interest_Rate': [2.75,2.5,2.5,2.25,1.75,1.75,1.75,1.75],
        'Defaulter_Rate': [5.3,5.3,5.4,5.6,6.2,6.1,5.9,6.2],
        'Bank_Index_Price': [1464,1394,1357,1293,965,943,958,971]
        }
df =pd. DataFrame(House_Loan,columns= ['Year','Month', 'Interest_Rate', 'Defaulter_Rate', 'Bank_Index_Price'])

plt.scatter(df['Interest_Rate'], df['Bank_Index_Price'], color='red')
plt.title('Bank Index Price Vs Interest Rate', fontsize=14)
plt.xlabel('Interest Rate', fontsize=14)
```

```
plt.ylabel('Bank Index Price', fontsize=14)
plt.grid(True)
plt.show()

plt.scatter(df['Defaulter_Rate'],df['Bank_Index_Price'], color='green')
plt.title('Bank Index Price Vs Defaulter Rate', fontsize=14)
plt.xlabel('Defaulter Rate', fontsize=14)
plt.ylabel('Bank Index Price', fontsize=14)
plt.grid(True)
plt.show()
```

Output: