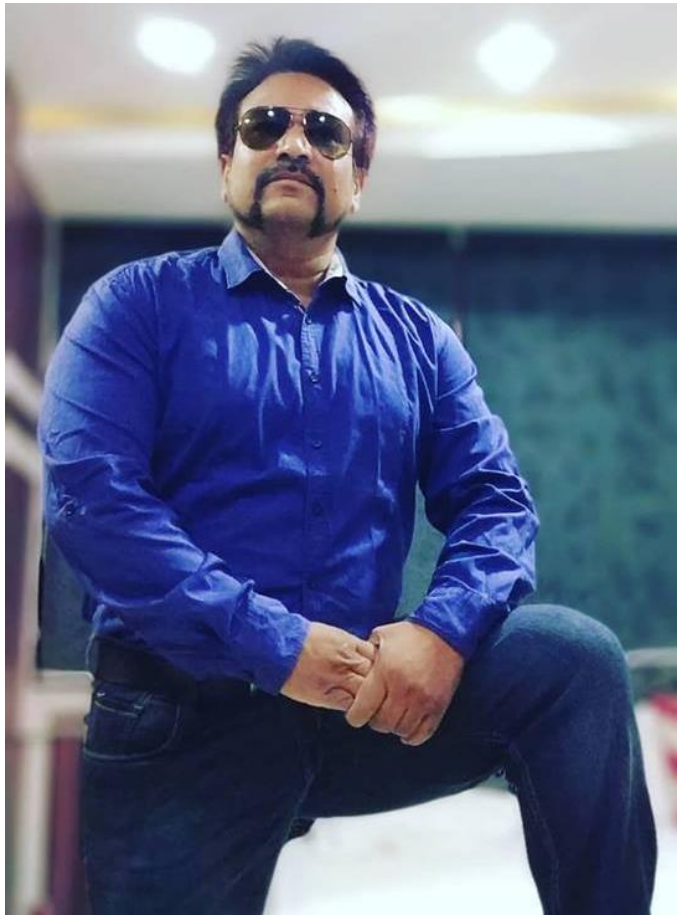
Working with NumPy

सीबीएसई पाठ्यक्रम पर आधारित

इन्फार्मेटिक्स प्रैक्टिसेज कक्षा -12



अध्याय -1

द्वारा:
संजीव भदौरिया
स्नातकोत्तर शिक्षक (संगणक विज्ञान)
के० वि० बाराबंकी (लखनऊ संभाग)

NumPy Arrays

- Pandas के structure बारे में जानने से पहले NumPy arrays को समझ लेते हैं | क्योंकि -
  1. Pandas के कुछ फंक्शन NumPy array के रूप में अपना परिणाम देते हैं |
  2. यह data structure के साथ एक त्वरित शुरुआत भी देगा
- NumPy ("Numerical Python" या Numeric Python") भी पाइथन का एक open source module है जो arrays और matrices पर तेज़ गणितीय फंक्शन प्रदान करता है |
- NumPy कोप्रयोग करने के लिए इसे import करना होता है| जिसका statement निम्न है

```
>>>import numpy as np
```

(इसमें np, numpy का बाद में उसे किये जाने वाला नाम है जो की वैकल्पिक होता है |)

- NumPy arrays 2 रूपों में होता है -
  - 1-D array –
    इसे Vectors के नाम से भी जानते हैं |
  - Multidimentional arrays –
    जिन्हें हम Matrices के नाम से भी जानते हैं |

```
>>> import numpy as np
>>> lst = [1,2,3,4]
>>> a1=np.array(lst)
>>> lst
[1, 2, 3, 4]
>>> print(a1)
[1 2 3 4]
>>> a1
array([1, 2, 3, 4])
>>>
```

List और array में अन्तेर देखिये

NumPy Arrays बनाम Python Lists

- यद्यपि NumPy array, Python List जैसे ही data को store करते हैं और list के जैसे ही array से डाटा को access भी करते हैं फिर भी NumPy array, पाइथन list से भिन्न होते हैं | उनके बीच में मुख्या अंतर निम्न हैं –
- एक बार NumPy array में size निर्धारित कर दिया फिर आप इसे बदल नहीं सकते | इसके लिए आपको या तो नया array बनाना होगा या इस array को नए array से overwrite करना होगा, जबकि list में ऐसा नहीं होता |
- NumPy array में सारे elements एक जैसे ही होने चाहिए अर्थात एक प्रकार के datatype के ही, जबकि list में ऐसा नहीं होता |
- एक NumPy array, Python list की तुलना में कम जगह घेरता है |
- NumPy array, Vectorized operation को सपोर्ट करता है अर्थात एक एक element क्रमवार जबकि list या list की list में ऐसा नहीं होता|

```
>>> import numpy as np
>>> lst=[1,2,3,4]
>>> a1=np.array(lst)
>>> lst+2
Traceback (most recent
  File "<pyshell#4>", l
```

इसमें list error देगा जबकि array में ये ऑपरेट होगा |

```
>>> a1+2
array([3, 4, 5, 6])
```

NumPy Data Types

NumPy निम्न प्रकार के data types को support करता है |

.No.	Data Type	Description	Size
1.	bool_	Boolean data type (stores *True* or *False*)	1 byte
2.	int_	Default type to store integers in ***int32*** or ***int64***	4 or 8 bytes
3.	int8	Stores signed integers in range –128 to 127	1 byte
4.	int16	Stores signed integers in range –32768 to 32767	2 bytes
5.	int32	Stores signed integers in range -2^{16} to $2^{16}-1$	4 bytes
6.	int64	Stores signed integers in range -2^{32} to $2^{32}-1$	8 bytes
7.	uint8	Stores unsigned integers in range 0 to 255	1 byte
8.	uint16	Stores integers in range 0 to $2^{16}-1$	2 bytes
9.	uint32	Stores integers in range 0 to $2^{32}-1$	4 bytes
10.	uint64	Stores integers in range 0 to $2^{64}-1$	8 bytes
11.	float_	Default type to store floating point (*float64*)	8 bytes
12.	float16	Stores **half precision floating point values** (*5 bits* exponent, *10 bit* mantissa)	2 bytes
13.	float32	Stores **single precision floating point values** (*8 bits* exponent, *23 bit* mantissa)	4 bytes
14.	float64	Stores **double precision floating point values** (*11 bits* exponent, *52 bit* mantissa)	8 bytes
15.	complex_	Default type to store complex numbers *(complex128)*	16 bytes
16.	complex64	Complex numbers represented by *two float32 numbers* for *real* and *imaginary* value components.	8 bytes
17.	complex128	Complex numbers represented by *two float64numbers* for *real* and *imaginary* value components.	16 bytes
18.	string_	Fixed-length string type.	1 byte per character
19.	unicode_	Fixed-length Unicode type.	number of bytes platform specific

NumPy Arrays बनाने के तरीके

1. array() function का प्रयोग array बनाने के लिए किया जाता है-

numpy.array(<arrayconvertible object>,[<datatype>])

*यह मान कर चलते हैं कि numpy को np के नाम से import किया गया है .

```
>>> import numpy as np
>>> nar1=np.array([2,5.2,1.0])
>>> nar1
array([2. , 5.2, 1. ])
>>>
```

उपरोक्त कोड निम्न कार्य करेगा

- nar1 एक ndarray object के रूप में बनेगा
- nar1 में 3 element होंगे (जैसे कि list में pass किये गए हैं).
- ndarray के elements का एक datatype का निर्धारण हो जायेगा जो default होगा | आप dtype argument का प्रयोग करके स्वयं datatype का निर्धारण कर सकते हैं |
- Elements के datatype के अनुसार elements का size भी निर्धारित हो जायेगा |

स्वयं के datatype के साथ array बनाना

```
>>> l=[1,2,3,4]
>>> ar=np.array(l,dtype=np.int64)
>>> ar
array([1, 2, 3, 4], dtype=int64)
>>>
```

type, dtype और size की जांच -

```
>>> l=[1,2,3,4]
>>> ar=np.array(l,dtype=np.int64)
>>> ar
array([1, 2, 3, 4], dtype=int64)
>>> print(type(ar))
<class 'numpy.ndarray'>
>>> print(ar.dtype)
int64
>>> print(ar.itemsize)
8
>>>
```

NumPy Arrays बनाने के तरीके

2. fromiter() का प्रयोग करके ndarray बनाना -

fromiter() function उस समय बहुत कारगर साबित होता है जब आप किसी non-numeric sequence से ndarray बनाना चाहते हैं |

numpy. fromiter (<iterable sequence name>,<target data type>,[<count>])

```
>>> ad={1:"A",2:"B",3:"C",4:"D",5:"E"}
>>> ar2=np.fromiter(ad,dtype=np.int32)
>>> print(ar2)
[1 2 3 4 5]
>>> ar2.dtype
dtype('int32')
>>> ar2.itemsize
4
>>> print(ar2[0],ar2[3])
1 4
```

NumPy Arrays को बनाने के तरीके

3. arange( ) से range का array तैयार किया जाता है |

<arrayname> = numpy.arange([start],stop,[step],[dtype])

```
>>> import numpy as np
>>> arr1=np.arange(5)
>>> arr1
array([0, 1, 2, 3, 4])
```

यहाँ सिर्फ stop value ही पास की है |

```
>>> import numpy as np
>>> arr2=np.arange(1,7,2,dtype=np.float32)
>>> arr2
array([1., 3., 5.], dtype=float32)
```

यहाँ 1 से 7 तक 2- 2 के अंतर पर

4. linspace( ) से range का array तैयार किया जाता है |

<arrayname> = numpy.linspace([start],stop,[dtype])

```
>>> import numpy as np
>>> arr3 = np.linspace(2,3,6)
>>> arr3
array([2. , 2.2, 2.4, 2.6, 2.8, 3. ])
```

यहाँ 2 से 3 के बीच की 6 values का array बना |

```
>>> import numpy as np
>>> arr4 = np.linspace(2.5,5,8)
>>> arr4
array([2.5       , 2.85714286, 3.21428571, 3.57142857, 3.92857143,
       4.28571429, 4.64285714, 5.        ])
```

यहाँ 2.5 से 5 तक 8 values का array बना |

2D NumPy Arrays

```
>>> import numpy as np
>>> A=np.array([[10,11,12,13],[21,22,23,24]])
>>> A[1,3]
24
>>> A[1][3]
24
>>> A
array([[10, 11, 12, 13],
       [21, 22, 23, 24]])
>>> print(A)
[[10 11 12 13]
 [21 22 23 24]]
>>> type(A),type(a1)
(<class 'numpy.ndarray'>, <class 'numpy.ndarray'>)
>>> a1.shape
(4,)
>>> A.shape
(2, 4)
>>> A.itemsize
4
```

यह list के प्रयोग से 2-D array बना है

Array के elemets को index के द्वारा access करना

Array को print करना |

Array का प्रकार देखना

Array की बनावट देखना (विभिन्न फंक्शन का प्रयोग)

NumPy arrays को ndarray भी कहते हैं (n-dimentional array)

2D NumPy Arrays

2. arange() का प्रयोग करके 2D array बनाना -

जैसे हमने 1D ndarrays को arange() function के साथ बनाया था ठीक उसी प्रकार हम 2D ndarrays को भी arange() function से बना सकते हैं साथ में हमें reshape() function का प्रयोग करना होगा.

<ndarray>.reshape(<shape tuple>)

```
>>> ar1=np.arange(10)
>>> ar1
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> ar3=ar1.reshape(5,2)
>>> ar3
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

3. ARRAYS बनाने के अन्य methods

a. empty() function का प्रयोग करके-

empty() function का प्रयोग करके empty array अथवा specified shape और dtype का unintialized array बनाया जा सकता है.

numpy.empty(Shape,[dtype=<datatype>,] [order = 'C' or 'F']

जहां :**dtype: एक python का data type अथवा numpy है जो initial values को set करता है |**

Shape: dimension है |

Order : 'C' का मतलब data का row wise arrangement (C means C like).

Order : 'F' का मतलब data का row wise arrangement (F means Fortran like)

```
>>> arr1=np.empty([2,3],dtype=np.int64, order='C')
>>> arr1
array([[0, 0, 0],
       [0, 0, 0]], dtype=int64)
```

b. zeros() function का प्रयोग करके -

numpy.zeros (Shape,[dtype=<datatype>,] [order = 'C' or 'F'])

```
>>> ar3=np.zeros([2,3],dtype=np.int64,order='F')
>>> ar3
array([[0, 0, 0],
       [0, 0, 0]], dtype=int64)
```

c. ones() function का प्रयोग करके -

numpy.ones(Shape,[dtype=<datatype>,] [order = 'C' or 'F'])

```
>>> ar4=np.ones([2,3],dtype=np.float32)
>>> ar4
array([[1., 1., 1.],
       [1., 1., 1.]], dtype=float32)
```

Array Slicing-

NumPy arrays से इसका subset निकलने के लिए slices का प्रयोग list की भांति कर सकते हैं |

<Arrayname>[<start>:<stop>:<step>]

- जब <start><stop> अथवा <step> values को न दिया गया हो तब Python default values को निम्नत: मानेगा |

  Start=0; Stop=dimension size ;Step=1

```
>>> arr=np.array([2,4,6,8,10,12,14,16])
>>> arr[3:7]
array([ 8, 10, 12, 14])
>>> arr[:5]
array([ 2,  4,  6,  8, 10])
>>> arr[4:]
array([10, 12, 14, 16])
>>> arr[:-1]
array([ 2,  4,  6,  8, 10, 12, 14])
>>> arr[:-3]
array([ 2,  4,  6,  8, 10])
>>> arr[2:7:2]
array([ 6, 10, 14])
>>>
```

Numpy Arrays को जोड़ना अथवा concatinate करना-

दो या अधिक उपस्थित ndarrays को join अथवा concatenate करने के लिए python निम्न functions प्रदान करता है -

1. *hstack() and vstack()*
2. *concatenate()*

existing arrays horizontally अथवा vertically जोड़ना -

यदि आपके पास 2 array हैं -

1	4	9	3	6	5	7	2

तो आपके pass horizontally array जोड़ने के बाद ऐसा array प्राप्त होगा -

Horizontally -

1	4	9	3	6	5	7	2

अथवा vertically ऐसा प्राप्त होगा -

1	4	9	3
6	5	7	2

इसके लिए आप hstack() function और vstack () function का प्रयोग कर सकते हैं |

Syntax-

Numpy.hstack(<tuple containing names of 1D arrays to be stacked>)

Numpy.vstack(<tuple containing names of 1D arrays to be stacked>)

```
>>> arr[2:7:2]
array([ 6, 10, 14])
>>> list1=[1,2,3,4,5]
>>> list2=[6,7,8,9,10]
>>> a1=np.array(list1)
>>> a1
array([1, 2, 3, 4, 5])
>>> a2=np.array(list2)
>>> a2
array([ 6,  7,  8,  9, 10])
>>> join1=np.hstack(a1,a2)
```

```
>>> join1
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
>>> join2=np.vstack((a1,a2))
>>> join2
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

यही operations 2D arrays पर भी लगाये जा सकते हैं .

EXISTING ARRAYS को CONCATENATE() के प्रयोग से जोड़ना –

इस function का प्रयोग करके आप NumPy arrays को axis 0(rows) or axis 1(column) के अनुसार जोड़ सकते हैं .

```
numpy.concatenate((<tuple of arrays to be joined>),[axis=<n>])
```

जहां :

-Axis argument यह बताता है की आपको किस axis के अनुसार जोड़ना है |

. यदि इसे छोड़ दिया जाये तो axisis को 0 मान लिया जाता है अर्थात rows के अनुसार|

जोड़े जाने वाले arrays सामान shape में होने चाहिए i.e.-

- यदि axis 0 है तो shape को column dimension के साथ मेल खाना चाहिए |
- यदि axis 1 है तो shape को rows dimension के साथ मेल खाना चाहिए |

```
>>> a1=np.array([1,2,3,4,5,6,7,8,9]).reshape(3,3)
>>> a1
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> a2=np.array([1,2,3,4,5,6]).reshape(2,3)
>>> a2
array([[1, 2, 3],
       [4, 5, 6]])
>>> a3=np.array([1,2,3,4,5,6]).reshape(3,2)
>>> a3
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
>>> a4=np.array([1,2,3]).reshape(3,1)
>>> a4
array([[1],
       [2],
       [3]])
>>>
```

ऊपर दिए गए arrays को देखें
a1 का shape (3,3).
a2 का shape(2,3)
a3 का shape(3,2)
a4 का shape(3,1)

```
>>> j1=np.concatenate((a1,a2),axis=0)
>>> j1
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9],
       [1, 2, 3],
       [4, 5, 6]])
>>> j2=np.concatenate((a1,a3),axis=1)
>>> j2
array([[1, 2, 3, 1, 2],
       [4, 5, 6, 3, 4],
       [7, 8, 9, 5, 6]])
>>> j3=np.concatenate((a1,a4),axis=1)
>>> j3
array([[1, 2, 3, 1],
       [4, 5, 6, 2],
       [7, 8, 9, 3]])
```

ARRAYS के subset निकालना (Slicing)-

Subsets, contiguous या non-contiguous हो सकते हैं |.

a. Contiguous Subsets को लेने के लिए NumPy Arrays को split करना

b. NumPy कुछ functions प्रदान करता है जैसे split(), hpslit(), vsplit() जिनकी मदद से subset निकाले जा सकते हैं . Syntax हैं -

```
numpy.hsplit(<array>,<n>)

numpy.vsplit(<array>,<n>)
```

जहां :

<array> NumPy array है और <n> number of sections/subsets हैं जिनमे array को विभक्त (divide)करना है |

<n> को ऐसे लिया जाना चाहिए ताकि <array> का सामान विभाजन हो सके अन्यथा एक error आयेगी |

```
>>> ary=np.arange(24.0).reshape(4,6)
>>> ary
array([[ 0.,  1.,  2.,  3.,  4.,  5.],
       [ 6.,  7.,  8.,  9., 10., 11.],
       [12., 13., 14., 15., 16., 17.],
       [18., 19., 20., 21., 22., 23.]])
>>> np.hsplit(ary,2)
[array([[ 0.,  1.,  2.],
       [ 6.,  7.,  8.],
       [12., 13., 14.],
       [18., 19., 20.]]), array([[ 3.,  4.,  5.],
       [ 9., 10., 11.],
       [15., 16., 17.],
       [21., 22., 23.]])]

>>> np.vsplit(ary,2)
[array([[ 0.,  1.,  2.,  3.,  4.,  5.],
       [ 6.,  7.,  8.,  9., 10., 11.]]), array([[12., 13., 14., 15., 16., 17.],
       [18., 19., 20., 21., 22., 23.]])]
>>>
```

b. split() function का प्रयोग -

numpy.split(<array>,<n>|<1D array>, [axis=0])

- <array> Numpy array है जिसको split किया जाना है .
- यदि 2nd arugument <n> है , तो <array> को <n> equal subarrays में विभाजित किया जाता है |
- यदि 2nd argument <n> और axis=0 है , तो यह vsplit() की तरह कार्य करता है और axis=1है तो यह hsplit() की तरह कार्य करता है |
- यदि 2nd argument 1D array है तब <array> बराबर subarrays में split होती है |
- axis argument, optional होता हिया और यदि छोड़ दिया जाए तो यह value 0 लेता है अर्थात horizontal axis पर | axis=1, के लिए split, vertical axis पर होता है |

```
>>> ar1d=[10,11,12,13,14,15,16,17,18,19]
>>> np.split(ar1d,[2,6])
[array([10, 11]), array([12, 13, 14, 15]), array([16, 17, 18, 19])]
>>>
```

0:2, 2:6, 6:

दिए गए argument 2,6 ने array को into 3 slices में विभाजित किया है i.e. 0:2, 2:6 and 6:

```
>>> ary=np.arange(24.0).reshape(4,6)
>>> ary
array([[ 0.,  1.,  2.,  3.,  4.,  5.],
       [ 6.,  7.,  8.,  9., 10., 11.],
       [12., 13., 14., 15., 16., 17.],
       [18., 19., 20., 21., 22., 23.]])
>>> np.split(ary,[1,4])
[array([[0., 1., 2., 3., 4., 5.]]), array([[ 6.,  7.,  8.,  9., 10., 11.],
       [12., 13., 14., 15., 16., 17.],
       [18., 19., 20., 21., 22., 23.]]), array([], shape=(0, 6), dtype=float64)]
>>> np.split(ary,[2,5],axis=1)
[array([[ 0.,  1.],
       [ 6.,  7.],
       [12., 13.],
       [18., 19.]]), array([[ 2.,  3.,  4.],
       [ 8.,  9., 10.],
       [14., 15., 16.],
       [20., 21., 22.]]), array([[ 5.],
       [11.],
       [17.],
       [23.]])]
```

divided as 0:1, 1:4,,,4: horizontally (axis=0 because skipped)

divided as 0:2, 2:5, 5: vertically (axis=1)

Condition based Non-Contiguous Subsets को निकालना

इसके लिए extract() function प्रयोग किया जाता है as per the syntax-

numpy.extract (<condition>,<array>)

extract() function सदैव दिए गए array के elements को 1D array के रूप में return करता है जो <condition> के criteria को पूरा करता है |

Condition बनाना -

एक 2D ndarray का subset खोजने के लिए जो 5 से पूरी तरह से विभाज्य है, तो आपको लिखना होगा- *condition=no.mod(ary,5)==0*

```
>>> ary
array([[ 0.,  1.,  2.,  3.,  4.,  5.],
       [ 6.,  7.,  8.,  9., 10., 11.],
       [12., 13., 14., 15., 16., 17.],
       [18., 19., 20., 21., 22., 23.]])
```

```
>>> cond=np.mod(ary,5)==0
>>> cond
array([[ True, False, False, False, False,  True],
       [False, False, False, False,  True, False],
       [False, False, False,  True, False, False],
       [False, False,  True, False, False, False]])
```

Ndarray के लिए विभिन्न functions जो numpy में उपलब्ध हैं

Function	विवरण
sin(), cos(), tan() और अन्य त्रिकोणमितीय function	त्रिकोणमितीय मान निकलने के लिए जैसे sin(x) इत्यादि
around(x)	round करता है दिए गए दशमलव के स्थानों तक
rint(x)	round करता है नज़दीकी पूर्ण संख्या तक
fix(x)	round करता है नज़दीकी पूर्ण संख्या तक शून्य की तरफ
floor(x)	Floor value देता है
ceil(x)	Ceiling वैल्यू देता है
trunc(x)	Truncated वैल्यू प्रदान करता है |
log 10(x)	Log base 10 प्रदान करता है
इसी पकार के ढेर functions प्रदान किये जाते हैं numpy के द्वारा |	

2D arrays पर Arithmetic operations -

a. Using operators-

<ndarray1> + <n>|<ndarray2>
<ndarray1> - <n>|<ndarray2>
<ndarray1> * <n>|<ndarray2>
<ndarray1> / <n>|<ndarray2>
<ndarray1> % <n>|<ndarray2>

b. Using NumPy Functions-

Numpy.add (<ndarray1>,<n>|<ndarray2>)
Numpy.subtract (<ndarray1>,<n>|<ndarray2>)
Numpy.multiply (<ndarray1>,<n>|<ndarray2>)
Numpy.divide (<ndarray1>,<n>|<ndarray2>)
Numpy.mod (<ndarray1>,<n>|<ndarray2>)
Numpy.remainder (<ndarray1>,<n>|<ndarray2>)

```
>>> ary+1.5
array([[ 1.5,  2.5,  3.5,  4.5,  5.5,  6.5],
       [ 7.5,  8.5,  9.5, 10.5, 11.5, 12.5],
       [13.5, 14.5, 15.5, 16.5, 17.5, 18.5],
       [19.5, 20.5, 21.5, 22.5, 23.5, 24.5]])
>>> new=ary+2.1
>>> ary+new
array([[ 2.1,  4.1,  6.1,  8.1, 10.1, 12.1],
       [14.1, 16.1, 18.1, 20.1, 22.1, 24.1],
       [26.1, 28.1, 30.1, 32.1, 34.1, 36.1],
       [38.1, 40.1, 42.1, 44.1, 46.1, 48.1]])

>>> np.multiply(ary,3)
array([[ 0.,  3.,  6.,  9., 12., 15.],
       [18., 21., 24., 27., 30., 33.],
       [36., 39., 42., 45., 48., 51.],
       [54., 57., 60., 63., 66., 69.]])
```

NumPy Arrays पर Applications -

1. **Covariance-** सहसंयोजक(Covariance) के पीछे सहज विचार यह है कि यह बताता है कि दो डेटासेट कैसे भिन्न होते हैं। 2 डेटासेट के बीच एक उच्च सकारात्मक सहसंयोजक (High Positive Covariance) का अर्थ है कि वे बहुत दृढ़ता से समान हैं। इसी तरह, 2 डेटासेट के बीच एक उच्च नकारात्मक सहसंयोजक (High Negative Covariance) का अर्थ है कि वे बहुत भिन्न हैं।

numpy.cov(<arr1> , <arr2>)

```
>>> a=np.array([1,2,3,4,5])
>>> b=np.array([3,4,0,-1,-3])
>>> check=np.cov(a,b)
>>> check
array([[ 2.5 , -4.25],
       [-4.25,  8.3 ]])
```

Negative values यह दर्शाती हैं की वे सामान नहीं है |

output एक 2X2 matrix है ,जो निम्नवत बनी है -
Check[0][0]=var(a)
Check[0][1]=covariance(a,b)
Check[1][0]=covariance(b,a)=covariance(a,b)
Check[1][1]=var(b)

2. **Correlation**- सहसंबंध (Correlation) मूल रूप से सहसंयोजक (Covariance) है। यह दो मान देता है: 1 यदि डेटा सेट में सकारात्मक सहसंयोजक और -1 है यदि डेटासेट में नकारात्मक सहसंयोजक हैं।

np.corrcoef(<array1>, <array2>)

```
>>> corr=np.corrcoef(a,b)
>>> corr
array([[ 1.        , -0.93299621],
       [-0.93299621,  1.        ]])
```

- कृपया हमारे ब्लॉग को फॉलो करिए और youtube channel को subscribe करिए | ताकि आपको और सारे chapters मिल सकें |

www.pythontrends.wordpress.com