Numpy

MULTIDIMENSIONAL ARRAY

Array

- An array is a collection of items stored at contiguous memory locations.
- The idea is to store multiple items of the same type together.
- This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).

- cars = ["Ford", "Volvo", "BMW"]
- x = cars[0]
- Cars[1

Numpy

- NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays.

- Using NumPy, mathematical and logical operations on arrays can be performed.

Operations using NumPy

- Using NumPy, a developer can perform the following operations −
  - Mathematical and logical operations on arrays.
  - Fourier transforms and routines for shape manipulation.
  - Operations related to linear algebra.
  - NumPy has in-built functions for linear algebra and random number generation.

NumPy - Ndarray Object

- The most important object defined in NumPy is an N-dimensional array type called **ndarray**.
- It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index.
- Every item in an ndarray takes the same size of block in the memory. Each element in ndarray is an object of data-type object (called **dtype**).
- The basic ndarray is created using an array function in NumPy as follows −
  - numpy.array

syntax

- numpy.array(object, dtype = None)

Sr.No.	Parameter & Description
1	**object** Any object exposing the array interface method returns an array, or any (nested) sequence.
2	**dtype** Desired data type of array, optional

NumPy - Data Types

Sr.No.	Data Types & Description
1	**bool_** Boolean (True or False) stored as a byte
2	**int_** Default integer type (same as C long; normally either int64 or int32)

float

15	**float32** Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
16	**float64** Double precision float: sign bit, 11 bits exponent, 52 bits mantissa

Data Type Objects (dtype)

A data type object describes interpretation of fixed block of memory corresponding to an array, depending on the following aspects −

- Type of data (integer, float or Python object)
- Size of data
- Byte order (little-endian or big-endian)
- In case of structured type, the names of fields, data type of each field and part of the memory block taken by each field.
- If data type is a subarray, its shape and data type

Import numpy as np

- # using array-scalar type

```
import numpy as np
dt = np.dtype(np.int32)
print dt
```

ndarray.shape

- This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

```
import numpy as np
B=np.array([])      c=np.array([[1,2],[3,4]]])
a = np.array([[1,2,3],[4,5,6]])
print a.shape
```

- The output is as follows −

- (2, 3)

NumPy also provides a reshape function to resize an array.

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print b
```

- The output is as follows −

- [[1, 2]
- [3, 4]
- [5, 6]]

ndarray.ndim

- This array attribute returns the number of array dimensions.
- # an array of evenly spaced numbers

```
import numpy as np
a = np.arange(24)
print a
```

- The output is as follows –

- [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]

NumPy - Array Creation Routines

- numpy.empty
- It creates an uninitialized array of specified shape and dtype. It uses the following constructor −

numpy.empty(shape, dtype = float, order = 'C')

import numpy as np

x = np.empty([3,2], dtype = int)

print x

- The output is as follows −

- [[22649312 1701344351]
- [1818321759 1885959276]
- [16779776 156368896]]

- numpy.zeros
- Returns a new array of specified size, filled with zeros.

- numpy.zeros(shape, dtype = float, order = 'C')
- The constructor takes the following parameters.

- # array of five zeros. Default dtype is float
- import numpy as np
- x = np.zeros(5)
- print x
- The output is as follows −

- [0. 0. 0. 0. 0.]

Numpy.random.rand ----- from uniform distribution (in range [0,1))

- All the values will be generated randomly between 0 and 1

numpy.random.randn() method --generates samples from the normal distribution---any number can be generated

```
import numpy as np
  # 1D Array
array = np.random.randn(5)
print("1D Array filled with random values : \n", array);
```

Output----

```
1D Array filled with randnom values :
[-0.51733692  0.48813676 -0.88147002  1.12901958  0.68026197]
```

randomly constructing 2D array

numpy.random.randn() method

```
import numpy as np
 # 2D Array
array = np.random.randn(3, 4)
print("2D Array filled with random values : \n", array);
```

2D Array filled with random values : output

- [[1.33262386 -0.88922967 -0.07056098 0.27340112]
- [1.00664965 -0.68443807 0.43801295 -0.35874714]
- [-0.19289416 -0.42746963 -1.80435223 0.02751727]]

PANDAS

Introduction to Pandas

- Library for computation with tabular data
- Mixed types of data allowed in a single table
- Columns and rows of data can be named
- Advanced data aggregation and statistical functions

Basic data structures

TYPE	PANDAS NAME
Vector (1 Dimension)	Series
Array (2 Dimensions)	DataFrame

pandas.Series

- pandas.Series(data, index, dtype)

S.No	Parameter & Description
1	**data** data takes various forms like ndarray, list, constants
2	**index** Index values must be unique and hashable, same length as data. Default **np.arrange(n)** if no index is passed.
3	**dtype** dtype is for data type. If None, data type will be inferred

Create a Series from ndarray

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print s
```

- Its output is as follows −

```
0  a
1  b
2  c
3  d
```

- dtype: object

Pandas Series with index

```
#import the pandas library and aliasing as pd

import pandas as pd

import numpy as np

data = np.array(['a','b','c','d'])

s = pd.Series(data,index=[100,101,102,103])

print s
```

- Its output is as follows −

```
100  a
101  b
102  c
103  d
dtype: object
```

Accessing Data from Series with Position

```
import pandas as pd
s = pd.Series([1,2,3,4,5])
#retrieve the first element
print s[0]
```

- Output

```
1
```

Retrieve the first three elements in the Series.

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index =
['a','b','c','d','e'])

#retrieve the first three element
print s[:3]
```

- Its output is as follows −

```
a  1
b  2
c  3
dtype: int64
```

Retrieve the last three elements.

```
import pandas as pd
s = pd.Series([1,2,3,4,5])

#retrieve the last three element
print s[-3:]
```

- Its output is as follows −

```
3
4
5
dtype: int64
```

Python Pandas - DataFrame

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

- Features of DataFrame
- Potentially columns are of different types
- Size – Mutable
- Labeled axes (rows and columns)
- Can Perform Arithmetic operations on rows and columns

Structure

Let us assume that we are creating a data frame with rows and columns.

	0	1
0	3620	10.7
1	7891	0.0
2	9761	NaN
3	3907	2.4
4	4338	15.3
5	5373	10.9

pandas.DataFrame

- A pandas DataFrame can be created using the following constructor −
- pandas.DataFrame(data, index, columns, dtype)

Create DataFrame

A pandas DataFrame can be created using various inputs like −

- Lists
- dict
- Series
- Numpy ndarrays
- Another DataFrame

Create a DataFrame from Lists

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print df
```

- Its output is as follows −

```
   0
0  1
1  2
2  3
3  4
4  5
```

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print df
```

Its output is as follows −

	Name	Age
0	Alex	10
1	Bob	12
2	Clarke	13

Create a DataFrame from Dict of ndarrays / Lists

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print df
```

Its output is as follows −

```
   Age   Name
0  28    Tom
1  34    Jack
2  29    Steve
3  42    Ricky
```

Missing data

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print df
```

Its output is as follows −

```
   a   b     c
0  1   2   NaN
1  5  10  20.0
```

Note − Observe, NaN (Not a Number) is appended in missing areas.

Pandas descriptive statistics

S.No.	Function	Description
1	count()	Number of non-null observations
2	sum()	Sum of values
3	mean()	Mean of Values
4	median()	Median of Values
5	mode()	Mode of values
6	std()	Standard Deviation of the Values
7	min()	Minimum Value
8	max()	Maximum Value

mean()
Returns the average value

```
import pandas as pd
import numpy as np
#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}
#Create a DataFrame
df = pd.DataFrame(d)
print df.mean()
```

- Its output is as follows −

```
Age       31.833333
Rating     3.743333
dtype: float64
```

std()
Returns the standard deviation of the numerical columns.

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame
df = pd.DataFrame(d)
print df.std()
```

Its output is as follows −

```
Age       9.232682
Rating    0.661628
dtype: float64
```

Summarizing Data
The describe() function computes a summary of statistics pertaining to the DataFrame columns.

```
import pandas as pd
import numpy as np
#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame
df = pd.DataFrame(d)
print df.describe()
```

Its output is as follows –

	Age	Rating
count	12.000000	12.000000
mean	31.833333	3.743333
std	9.232682	0.661628
min	23.000000	2.560000
25%	25.000000	3.230000
50%	29.500000	3.790000
75%	35.500000	4.132500
max	51.000000	4.800000

Python Pandas - Indexing and Selecting Data

Indexing	Description
.loc()	Label based
.iloc()	Integer based

.loc()

Pandas provide various methods to have purely label based indexing. When slicing, the start bound is also included. Integers are valid labels, but they refer to the label and not the position.

.loc() has multiple access methods like −

- A single scalar label
- A list of labels
- A slice object
- A Boolean array

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4),
index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C', 'D'])
print(df)
```

	A	B	C	D
a	-0.069384	-0.787414	-0.474020	0.216364
b	-1.265146	1.431168	-0.443679	0.435746
c	-0.483534	1.478549	-0.619949	0.475728
d	-0.770839	-0.272018	-0.361404	0.684284
e	0.141069	-1.162204	0.047874	-0.054955
f	0.056770	0.214658	-0.180290	-1.325190
g	0.976647	0.768103	1.535049	0.682851
h	1.249561	-2.757903	1.181472	-1.311080

By adding .loc in the code

- #select all rows for a specific column
- print df.loc[:,'A']

Its output is as follows

a -0.069384

b -1.265146

c -0.483534

d -0.770839

e 0.141069

f 0.056770

g 0.976647

h 1.249561

.iloc------index location

.iloc()

Pandas provide various methods in order to get purely integer based indexing. Like python and numpy, these are 0-based indexing.

The various access methods are as follows −

- An Integer
- A list of integers
- A range of values

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4), columns = ['A', 'B', 'C', 'D'])

# select all rows for a specific column
print df.iloc[:4]
```

Its **output** is as follows –

	A	B	C	D
0	0.699435	0.256239	-1.270702	-0.645195
1	-0.685354	0.890791	-0.813012	0.631615
2	-0.783192	-0.531378	0.025070	0.230806
3	0.539042	-1.284314	0.826977	-0.026251

- Move to Practical in Jupyter Notebook