Data Analysis - Numpy

July 2, 2024

Abstract

This paper explores data analysis using the NumPy package for scientific computing in Python. It provides support for arrays, matrices, and many mathematical functions to operate on these data structures efficiently

- Credits -
  - zero-to-mastery-ml - https://github.com/mrdbourke/zero-to-mastery-ml/tree/master/data
  - https://www.udemy.com/course/complete-machine-learning-and-data-science-zero-to-mastery

[1]:
```
from IPython.display import display, Image

display(Image(filename="./Resources/Images/Numpy.png"))
```

Core Features

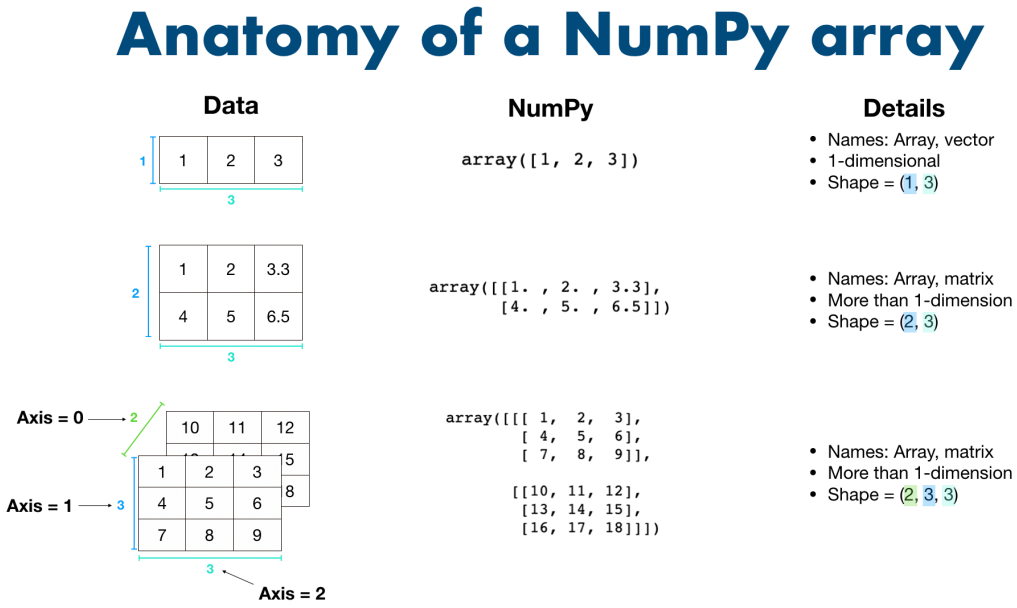
- Multidimensional Arrays: NumPy introduces the ndarray, a powerful n-dimensional array object.
- Element-wise Operations: Perform operations on arrays element-wise.
- Broadcasting: Perform arithmetic operations on arrays of different shapes.
- Linear Algebra Functions: Solve linear algebra problems with functions like dot product, matrix inverse, eigenvalues, etc.
- Random Number Generation: Generate random numbers and samples from various distributions.
- Fourier Transform: Perform fast Fourier transforms.
- Mathematical Functions: Includes functions for trigonometry, statistics, and more.
- Integration with C/C++/Fortran: Interface with low-level languages to optimize perfor-

mance.

[2]:
```
# Importing Numpy Package
import numpy as np
```

[3]:
```
from IPython.display import display, Image

display(Image(filename="./Resources/Images/numpy-anatomy-of-an-array-updated.
↪png"))
```



1. DataTypes and Attributes

[184]:
```
# Numpy's main datatype is ndarray - N Dimensinal Array
a1 = np.array([1,2,3]) # lets create a sample array with a list
a1
a2 = np.array([[1,2,3,3.3],[4,5,7,6]])

a3 = np.array([[[1,2,3],
                [4,5,6],
                [7,8,9]],
                [[10,11,12],
                [13,14,15],
                [16,17,18]]])
```

[5]:
```
# Lets print the arrays
```

[6]:
```
a1, a2, a3
```

[6]:
```
(array([1, 2, 3]),
 array([[1. , 2. , 3. , 3.3],
        [4. , 5. , 7. , 6. ]]),
 array([[[ 1,  2,  3],
         [ 4,  5,  6],
         [ 7,  8,  9]],

        [[10, 11, 12],
         [13, 14, 15],
         [16, 17, 18]]]))
```

[7]:
```
#Lets know the number of dimensions from each of the Array
a1.shape,a2.shape,a3.shape
```

[7]:
```
((3,), (2, 4), (2, 3, 3))
```

[8]:
```
#Lets know the number of dimensions from each of the Array
a1.ndim,a2.ndim,a3.ndim
```

[8]:
```
(1, 2, 3)
```

[9]:
```
#Lets know the datatypes
a1.dtype,a2.dtype,a3.dtype
```

[9]:
```
(dtype('int64'), dtype('float64'), dtype('int64'))
```

[10]:
```
#Lets know the size
a1.size,a2.size,a3.size
```

[10]:
```
(3, 8, 18)
```

[11]:
```
#Lets know the type
type(a1),type(a2),type(a3)
```

[11]:
```
(numpy.ndarray, numpy.ndarray, numpy.ndarray)
```

[12]:
```
# Create a Pandas dataFrame from a Numpy array

import pandas as pd

df=pd.DataFrame(a2)
df
```

[12]:

	0	1	2	3
0	1.0	2.0	3.0	3.3
1	4.0	5.0	7.0	6.0

2. Creating Arrays

[13]:
```
sample_array=np.array([1,2,3])
sample_array,sample_array.dtype
```

[13]:
```
(array([1, 2, 3]), dtype('int64'))
```

[14]:
```
# creating array with ones by giving the shape
ones=np.ones([1,2,2,3])
ones,ones.dtype,type(ones)
```

[14]:
```
(array([[[[1., 1., 1.],
          [1., 1., 1.]],

         [[1., 1., 1.],
          [1., 1., 1.]]]]),
 dtype('float64'),
 numpy.ndarray)
```

[15]:
```
# creating array with zeros by giving the shape
zeros=np.zeros([1,2,2,3])
zeros,zeros.dtype,type(zeros)
```

[15]:
```
(array([[[[0., 0., 0.],
          [0., 0., 0.]],

         [[0., 0., 0.],
          [0., 0., 0.]]]]),
 dtype('float64'),
 numpy.ndarray)
```

[16]:
```
# creating array with start, stop and step value.
range_array=np.arange(0,6,1)
range_array
```

[16]:
```
array([0, 1, 2, 3, 4, 5])
```

0.0.1 Generating Pseudo-Random Numbers with NumPy

NumPy provides a powerful random module that allows you to generate pseudo-random numbers. Here are some common methods for generating random numbers using NumPy:

[17]:
```
#random array
random_array=np.random.randint(0,10, size=(3,5))
random_array,random_array.dtype
```

[17]:
```
(array([[7, 6, 3, 2, 3],
        [3, 9, 9, 9, 4],
        [3, 3, 2, 6, 3]]),
 dtype('int64'))
```

[18]:
```
#random array
random_array2=np.random.random((3,5))
random_array2,random_array2.dtype
```

[18]:
```
(array([[0.37623571, 0.14909837, 0.78008197, 0.6463277 , 0.5497461 ],
        [0.50222091, 0.23726681, 0.19542584, 0.61581266, 0.12889903],
        [0.77831658, 0.08703917, 0.70452577, 0.25474942, 0.80363963]]),
 dtype('float64'))
```

[19]:
```
#random array

random_array3=np.random.random((3,5))
random_array3,random_array3.dtype
```

[19]:
```
(array([[0.67048783, 0.21066622, 0.44218973, 0.98989719, 0.34210572],
        [0.52934468, 0.57144134, 0.78231732, 0.50726528, 0.43252198],
        [0.96501912, 0.20822561, 0.53199887, 0.43032072, 0.9628018 ]]),
 dtype('float64'))
```

[24]:
```
#Pseudo-random-numbers
# Seed the random number generator
np.random.seed(7)
random_array4 = np.random.rand(4, 5)
random_array4,random_array4.dtype
```

[24]:
```
(array([[0.07630829, 0.77991879, 0.43840923, 0.72346518, 0.97798951],
        [0.53849587, 0.50112046, 0.07205113, 0.26843898, 0.4998825 ],
        [0.67923   , 0.80373904, 0.38094113, 0.06593635, 0.2881456 ],
        [0.90959353, 0.21338535, 0.45212396, 0.93120602, 0.02489923]]),
 dtype('float64'))
```

3. Viewing array and metrics

[27]:
```
# Finidng unique
np.unique(random_array3)
```

[27]:
```
array([0.20822561, 0.21066622, 0.34210572, 0.43032072, 0.43252198,
       0.44218973, 0.50726528, 0.52934468, 0.53199887, 0.57144134,
       0.67048783, 0.78231732, 0.9628018 , 0.96501912, 0.98989719])
```

[44]:
```
a3,a3.shape,a3[1,1,:2]
```

[44]:
```
(array([[[ 1,  2,  3],
         [ 4,  5,  6],
         [ 7,  8,  9]],

        [[10, 11, 12],
         [13, 14, 15],
```

```
        [16, 17, 18]]]),
 (2, 3, 3),
 array([13, 14]))
```

[47]:
```
a4 = np.random.randint(10,size=(2,3,4,5))
a4,a4.shape
```

[47]:
```
(array([[[[4, 0, 4, 1, 5],
          [2, 4, 8, 7, 5],
          [7, 5, 3, 7, 6],
          [2, 3, 1, 1, 8]],

         [[4, 2, 1, 7, 0],
          [8, 8, 1, 2, 3],
          [0, 1, 1, 5, 5],
          [6, 0, 9, 3, 2]],

         [[4, 2, 8, 4, 9],
          [0, 0, 2, 6, 3],
          [9, 5, 4, 6, 9],
          [9, 9, 4, 8, 2]]],


        [[[4, 6, 5, 3, 5],
          [2, 6, 4, 3, 2],
          [5, 9, 7, 5, 8],
          [5, 2, 0, 9, 3]],

         [[2, 2, 4, 3, 4],
          [8, 7, 0, 2, 7],
          [8, 8, 4, 0, 4],
          [1, 7, 4, 9, 0]],

         [[1, 0, 6, 2, 0],
          [2, 4, 4, 1, 0],
          [0, 2, 3, 7, 6],
          [9, 3, 5, 6, 8]]]]),
 (2, 3, 4, 5))
```

[51]:
```
#Get the first 4 numbers of inner most array
a4[:,:,:,:4]
```

[51]:
```
array([[[[4, 0, 4, 1],
         [2, 4, 8, 7],
         [7, 5, 3, 7],
         [2, 3, 1, 1]],
```

```
[[4, 2, 1, 7],
 [8, 8, 1, 2],
 [0, 1, 1, 5],
 [6, 0, 9, 3]],

[[4, 2, 8, 4],
 [0, 0, 2, 6],
 [9, 5, 4, 6],
 [9, 9, 4, 8]]],


[[[4, 6, 5, 3],
  [2, 6, 4, 3],
  [5, 9, 7, 5],
  [5, 2, 0, 9]],

 [[2, 2, 4, 3],
  [8, 7, 0, 2],
  [8, 8, 4, 0],
  [1, 7, 4, 9]],

 [[1, 0, 6, 2],
  [2, 4, 4, 1],
  [0, 2, 3, 7],
  [9, 3, 5, 6]]])
```

4. Manipulating and Comparing Arrays

Arithmetic

[56]:
```
ones = np.ones(3)
a1,ones
```

[56]:
```
(array([1, 2, 3]), array([1., 1., 1.]))
```

[57]:
```
a1 + ones
```

[57]:
```
array([2., 3., 4.])
```

[60]:
```
a1-ones
```

[60]:
```
array([-1.,  0.,  1.])
```

[63]:
```
a2
```

[63]:
```
array([[1. , 2. , 3. , 3.3],
       [4. , 5. , 7. , 6. ]])
```

[70]:
```
sample_multiplication_array= a2[:,:3] * a1
sample_multiplication_array
```

[70]:
```
array([[ 1.,  4.,  9.],
       [ 4., 10., 21.]])
```

[72]:
```
# Division
a2[:,:3] / a1
```

[72]:
```
array([[1.        , 1.        , 1.        ],
       [4.        , 2.5       , 2.33333333]])
```

[74]:
```
#Floor Division
a2[:,:3] //a1
```

[74]:
```
array([[1., 1., 1.],
       [4., 2., 2.]])
```

[76]:
```
a2[:,:3] **2 , np.square(a2[:,:3])
```

[76]:
```
(array([[ 1.,  4.,  9.],
        [16., 25., 49.]]),
 array([[ 1.,  4.,  9.],
        [16., 25., 49.]]))
```

[82]:
```
a2[:,:3] % 1
```

[82]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

[83]:
```
np.exp(a1), np.log(a1)
```

[83]:
```
(array([ 2.71828183,  7.3890561 , 20.08553692]),
 array([0.        , 0.69314718, 1.09861229]))
```

Aggregation

[86]:
```
list_list=[1,2,3]
type(list_list),sum(list_list)
```

[86]:
```
(list, 6)
```

[87]:
```
a1
```

[87]:
```
array([1, 2, 3])
```

[89]:
```
np.sum(a1)
```

[89]:
```
6
```

[90]:
```
np.mean(a1)
```

[90]:
```
2.0
```

[93]:
```
np.max(a1),np.min(a1)
```

[93]:
```
(3, 1)
```

[94]:
```
np.std(a1)
```

[94]:
```
0.816496580927726
```

[95]:
```
np.var(a1)
```

[95]:
```
0.6666666666666666
```

Reshaping and Transposing

[102]:
```
a2, a2.shape
```

[102]:
```
(array([[1. , 2. , 3. , 3.3],
        [4. , 5. , 7. , 6. ]]),
 (2, 4))
```

[103]:
```
a3, a3.shape
```

[103]:
```
(array([[[ 1,  2,  3],
         [ 4,  5,  6],
         [ 7,  8,  9]],

        [[10, 11, 12],
         [13, 14, 15],
         [16, 17, 18]]]),
 (2, 3, 3))
```

[113]:
```
#Reshape
a2[:,:3].reshape(2,3,1)*a3
```

[113]:
```
array([[[  1.,   2.,   3.],
        [  8.,  10.,  12.],
        [ 21.,  24.,  27.]],

       [[ 40.,  44.,  48.],
        [ 65.,  70.,  75.],
        [112., 119., 126.]]])
```

[114]:
```
#Transpose
a2.T
```

[114]:
```
array([[1. , 4. ],
       [2. , 5. ],
       [3. , 7. ],
       [3.3, 6. ]])
```

Dot Product

[117]:
```
np.random.seed=0
matrix1=np.random.randint(10,size=(5,3))
matrix2=np.random.randint(10,size=(5,3))
```

[118]:
```
matrix1
```

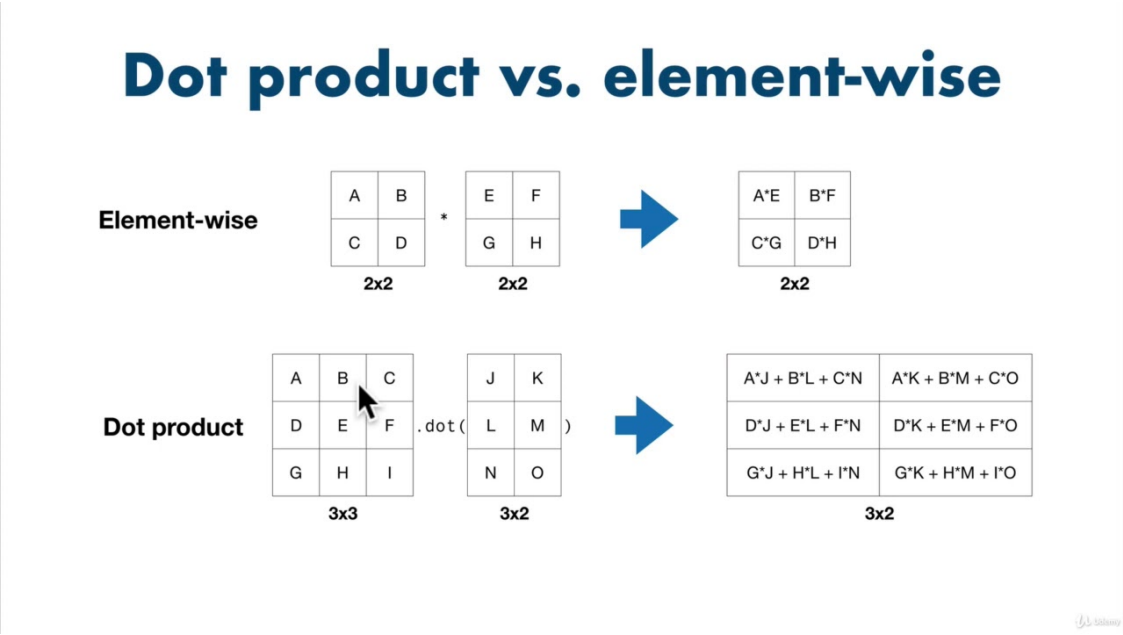
[118]:
```
array([[7, 6, 8],
       [5, 4, 3],
       [0, 7, 4],
       [0, 8, 5],
       [0, 7, 5]])
```

[119]:
```
matrix2
```

[119]:
```
array([[1, 5, 9],
       [4, 1, 9],
       [6, 5, 6],
       [8, 5, 9],
       [4, 6, 0]])
```

[127]:
```
from IPython.display import display, Image

display(Image(filename="./Resources/Images/matrix_mulitplication.jpg"))
```



[121]:
```
#Element wise multiplication (Hadarmard product)
matrix1 * matrix2
```

[121]:
```
array([[ 7, 30, 72],
       [20,  4, 27],
       [ 0, 35, 24],
       [ 0, 40, 45],
       [ 0, 42,  0]])
```

[128]:
```
matrix1.shape, matrix2.shape
```

[128]:
```
((5, 3), (5, 3))
```

[131]:
```
matrix2, matrix2.T.shape
```

[131]:
```
(array([[1, 5, 9],
        [4, 1, 9],
        [6, 5, 6],
        [8, 5, 9],
        [4, 6, 0]]),
 (3, 5))
```

[133]:
```
# Dot Product Multiplication
np.dot(matrix1,matrix2.T),np.dot(matrix1,matrix2.T).shape
```

[133]:
```
(array([[109, 106, 120, 158,  64],
        [ 52,  51,  68,  87,  44],
        [ 71,  43,  59,  71,  42],
        [ 85,  53,  70,  85,  48],
        [ 80,  52,  65,  80,  42]]),
 (5, 5))
```

Dot Product example (Nut Butter sales)

[161]:
```
np.random.seed=0
#Number of Jars sold
sales_amount=np.random.randint(20,size=(5,3))
sales_amount
```

[161]:
```
array([[ 6,  2, 18],
       [18, 11,  5],
       [17, 17,  4],
       [11,  5,  3],
       [13,  5,  7]])
```

[162]:
```
#Create Weekly Sales
weekly_sales = pd.DataFrame(sales_amount,index=["Mon","Tue","Wed","Thu","Fri"],␣
↪columns=["Almond Butter",
                                                                              ␣
↪          "Peanut Butter","Cashew Butter"])
weekly_sales
```

[162]:

	Almond Butter	Peanut Butter	Cashew Butter
Mon	6	2	18
Tue	18	11	5
Wed	17	17	4
Thu	11	5	3
Fri	13	5	7

[163]:
```
#create prices array
prices=np.array([10,8,12])
butter_Prices= pd.DataFrame(prices.
↪reshape(1,3),index=["Price"],columns=["Almond Butter","Peanut␣
↪Butter","Cashew Butter"])
butter_Prices
```

[163]:

	Almond Butter	Peanut Butter	Cashew Butter
Price	10	8	12

[164]:
```
Total_sales=np.dot(weekly_sales,butter_Prices.T)
Total_sales
```

[164]:
```
array([[292],
       [328],
       [354],
       [186],
       [254]])
```

[165]:
```
Total_sales.shape
```

[165]:
```
(5, 1)
```

[166]:
```
weekly_sales.shape
```

[166]:
```
(5, 3)
```

[167]:
```
weekly_sales["Total $"]=Total_sales
weekly_sales
```

[167]:

	Almond Butter	Peanut Butter	Cashew Butter	Total $
Mon	6	2	18	292
Tue	18	11	5	328
Wed	17	17	4	354

```
Thu             11              5               3       186
Fri             13              5               7       254
```

Comparison Operators

[168]:
```
a1
```

[168]:
```
array([1, 2, 3])
```

[176]:
```
a2[:,:3]
```

[176]:
```
array([[1., 2., 3.],
       [4., 5., 7.]])
```

[179]:
```
 a1> a2[:,:3]
```

[179]:
```
array([[False, False, False],
       [False, False, False]])
```

[185]:
```
 a1 == a2[:,:3]
```

[185]:
```
array([[ True,  True,  True],
       [False, False, False]])
```

[186]:
```
a1
```

[186]:
```
array([1, 2, 3])
```

Sorting the Arrays

[194]:
```
np.random.seed=(10)
random_array=np.random.randint(10,size=(3,5))
random_array
```

[194]:
```
array([[7, 4, 8, 7, 7],
       [8, 3, 3, 7, 0],
       [0, 6, 7, 0, 2]])
```

[195]:
```
random_array.shape
```

[195]:
```
(3, 5)
```

[196]:
```
np.sort(random_array)
```

[196]:
```
array([[4, 7, 7, 7, 8],
       [0, 3, 3, 7, 8],
       [0, 0, 2, 6, 7]])
```

[197]:
```
np.argsort(random_array)
```

[197]:
```
array([[1, 0, 3, 4, 2],
       [4, 1, 2, 3, 0],
       [0, 3, 4, 1, 2]])
```

[202]:
```
np.argmin(random_array,axis=0)
```

[202]:
```
array([2, 1, 1, 2, 1])
```

[204]:
```
np.argmin(random_array,axis=1)
```

[204]:
```
array([1, 4, 0])
```

5. Practical Examples using Numpy Array

[206]:
```
from IPython.display import display, Image

display(Image(filename="./Resources/Images/panda.png"))
```

[211]:
```
# Turn the image into a Numpy array

from matplotlib.image import imread

panda = imread("./Resources/Images/panda.png")
```

```
panda.shape , panda.size
```

[211]:
```
((2330, 3500, 3), 24465000)
```

[ ]:
```

```