numpy-lab

August 31, 2024

0.1 Name: B. Satwick

0.2 Sec: CSD-A

0.3 Roll.No: A22126551006

1 GETTING FAMILIARITY WITH NUMPY

[1]:
```
import numpy as np
import random
```

Here we import these two librabies in order to work with them

1.0.1 Creating an array using numpy

[2]:
```
arr=np.random.randint(1, 100, 10)
```

[3]:
```
arr
```

[3]:
```
array([ 2, 37, 79, 78, 95, 23, 12, 99, 86, 54])
```

This creates an array with randon 10 values between 1 and 100

[4]:
```
arr_0=np.zeros(5)
arr_0
```

[4]:
```
array([0., 0., 0., 0., 0.])
```

This creates an array with all zeros of a size wished by the user

[5]:
```
arr_1=np.ones(5)
arr_1
```

[5]:
```
array([1., 1., 1., 1., 1.])
```

This creates an array with all ones with the size wished by the user

[6]:
```
arr1=np.arange(8)
arr1
```

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

This creates an array with the numbers in order from 0 to a wished size

[7]:
```
identity_matrix = np.eye(3)
identity_matrix
```

[7]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

This creates an 3x3 matrix with diagonal elements 1

[25]:
```
arr2=np.array([[1,2,3],[4, 5, 6]])
arr2
```

[25]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

Creating a 2D array

1.0.2 Performing basic operations

[15]:
```
arr1=np.array([1, 2, 3, 4, 5,6])
arr1
```

[15]:
```
array([1, 2, 3, 4, 5, 6])
```

[16]:
```
arr2=np.array([6, 5, 4, 3, 2, 1])
arr2
```

[16]:
```
array([6, 5, 4, 3, 2, 1])
```

Addition

[17]:
```
arr1+arr2
```

[17]:
```
array([7, 7, 7, 7, 7, 7])
```

Subtraction

[18]:
```
arr1-arr2
```

[18]:
```
array([-5, -3, -1,  1,  3,  5])
```

Multiplication

[19]:
```
arr1*arr2
```

[19]:
```
array([ 6, 10, 12, 12, 10,  6])
```

Division

[21]:
```
print(arr1/arr2)
```

```
[0.16666667 0.4        0.75       1.33333333 2.5        6.        ]
```

Power

[22]:
```
arr1**2
```

[22]:
```
array([ 1,  4,  9, 16, 25, 36])
```

1.0.3 Understanding array properties

[23]:
```
arr
```

[23]:
```
array([1, 2, 3, 4, 5, 6])
```

[26]:
```
arr2
```

[26]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

To know the shape of the array

[27]:
```
arr2.shape
```

[27]:
```
(2, 3)
```

To know its size

[28]:
```
arr2.size
```

[28]:
```
6
```

To know its dimentionality

[29]:
```
arr2.ndim
```

[29]:
```
2
```

To know he datatype

[33]:
```
arr.dtype
```

[33]:
```
dtype('int32')
```

1.0.4 Data manipulation

[34]:
```
arr
```

[34]:
```
array([1, 2, 3, 4, 5, 6])
```

[36]:
```
arr2=np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
arr2
```

[36]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

INDEXING

[37]:
```
arr[4]
```

[37]:
```
5
```

[38]:
```
arr2[1]
```

[38]:
```
array([4, 5, 6])
```

[39]:
```
arr2[2, 0]
```

[39]:
```
7
```

NEGATIVE INDEXINGNegative indexing means it indexes the elements in backward direction

[40]:
```
arr[-1]
```

[40]:
```
6
```

[41]:
```
arr2[-1, 0]
```

[41]:
```
7
```

SLICING

[42]:
```
arr[2:5]
```

[42]:
```
array([3, 4, 5])
```

[43]:
```
arr2[:, 1]
```

[43]:
```
array([2, 5, 8])
```

NEGATIVE SLICING

[45]:
```
arr[-5:-1]
```

[45]:
```
array([2, 3, 4, 5])
```

NOTE: While slicing you make sure that the left value is always smaller than the right one

[48]:
```
arr[3]=10
arr
```

[48]:
```
array([ 1,  2,  3, 10,  5,  6])
```

[49]:
```
arr2[:, 2]=18
arr2
```

[49]:
```
array([[ 1,  2, 18],
       [ 4,  5, 18],
       [ 7,  8, 18]])
```

RESHAPE

[50]:
```
arr
```

[50]:
```
array([ 1,  2,  3, 10,  5,  6])
```

[51]:
```
arr2
```

[51]:
```
array([[ 1,  2, 18],
       [ 4,  5, 18],
       [ 7,  8, 18]])
```

[52]:
```
arr.reshape(2, 3)
```

[52]:
```
array([[ 1,  2,  3],
       [10,  5,  6]])
```

[56]:
```
arr2.reshape(9)
```

[56]:
```
array([ 1,  2, 18,  4,  5, 18,  7,  8, 18])
```

1.0.5 Data aggregation

[57]:
```
arr=np.random.randint(1, 100, 15)
arr
```

[57]:
```
array([95, 17, 46, 42,  8, 64, 46, 69, 97, 90, 12, 40, 10, 90, 88])
```

To know the sum of an array

[61]:
```
arr.sum()
```

[61]:
```
814
```

To know the mean of an array

[62]:
```
arr.mean()
```

[62]:
```
54.266666666666666
```

To know the median

[64]:
```
np.median(arr)
```

[64]:
```
46.0
```

To know the standard deviation

[65]:
```
np.std(arr)
```

[65]:
```
31.89036775510053
```

To know the variance

[66]:
```
np.var(arr)
```

[66]:
```
1016.9955555555556
```

To know the minimum

[67]:
```
np.min(arr)
```

[67]:
```
8
```

To know the maximun

[68]:
```
np.max(arr)
```

[68]:
```
97
```

To know the cummilative sum

[69]:
```
np.cumsum(arr)
```

[69]:
```
array([ 95, 112, 158, 200, 208, 272, 318, 387, 484, 574, 586, 626, 636,
       726, 814])
```

To know the percentiles

[70]:
```
np.percentile(arr, 75)
```

[70]:
```
89.0
```

[71]:
```
np.percentile(arr, 50)
```

[71]:
```
46.0
```

1.0.6 Data analysis

We will be using iris dataset

[72]:
```
import seaborn as sns
```

[73]:
```
df=sns.load_dataset('iris')
```

[74]:
```
df
```

[74]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
..	…	…	…	…	…
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

[150 rows x 5 columns]

[75]:
```
arr=df.to_numpy()
```

[76]:
```
arr
```

[76]:
```
array([[5.1, 3.5, 1.4, 0.2, 'setosa'],
       [4.9, 3.0, 1.4, 0.2, 'setosa'],
       [4.7, 3.2, 1.3, 0.2, 'setosa'],
       [4.6, 3.1, 1.5, 0.2, 'setosa'],
       [5.0, 3.6, 1.4, 0.2, 'setosa'],
       [5.4, 3.9, 1.7, 0.4, 'setosa'],
       [4.6, 3.4, 1.4, 0.3, 'setosa'],
       [5.0, 3.4, 1.5, 0.2, 'setosa'],
       [4.4, 2.9, 1.4, 0.2, 'setosa'],
       [4.9, 3.1, 1.5, 0.1, 'setosa'],
       [5.4, 3.7, 1.5, 0.2, 'setosa'],
       [4.8, 3.4, 1.6, 0.2, 'setosa'],
```

```
[4.8, 3.0, 1.4, 0.1, 'setosa'],
[4.3, 3.0, 1.1, 0.1, 'setosa'],
[5.8, 4.0, 1.2, 0.2, 'setosa'],
[5.7, 4.4, 1.5, 0.4, 'setosa'],
[5.4, 3.9, 1.3, 0.4, 'setosa'],
[5.1, 3.5, 1.4, 0.3, 'setosa'],
[5.7, 3.8, 1.7, 0.3, 'setosa'],
[5.1, 3.8, 1.5, 0.3, 'setosa'],
[5.4, 3.4, 1.7, 0.2, 'setosa'],
[5.1, 3.7, 1.5, 0.4, 'setosa'],
[4.6, 3.6, 1.0, 0.2, 'setosa'],
[5.1, 3.3, 1.7, 0.5, 'setosa'],
[4.8, 3.4, 1.9, 0.2, 'setosa'],
[5.0, 3.0, 1.6, 0.2, 'setosa'],
[5.0, 3.4, 1.6, 0.4, 'setosa'],
[5.2, 3.5, 1.5, 0.2, 'setosa'],
[5.2, 3.4, 1.4, 0.2, 'setosa'],
[4.7, 3.2, 1.6, 0.2, 'setosa'],
[4.8, 3.1, 1.6, 0.2, 'setosa'],
[5.4, 3.4, 1.5, 0.4, 'setosa'],
[5.2, 4.1, 1.5, 0.1, 'setosa'],
[5.5, 4.2, 1.4, 0.2, 'setosa'],
[4.9, 3.1, 1.5, 0.2, 'setosa'],
[5.0, 3.2, 1.2, 0.2, 'setosa'],
[5.5, 3.5, 1.3, 0.2, 'setosa'],
[4.9, 3.6, 1.4, 0.1, 'setosa'],
[4.4, 3.0, 1.3, 0.2, 'setosa'],
[5.1, 3.4, 1.5, 0.2, 'setosa'],
[5.0, 3.5, 1.3, 0.3, 'setosa'],
[4.5, 2.3, 1.3, 0.3, 'setosa'],
[4.4, 3.2, 1.3, 0.2, 'setosa'],
[5.0, 3.5, 1.6, 0.6, 'setosa'],
[5.1, 3.8, 1.9, 0.4, 'setosa'],
[4.8, 3.0, 1.4, 0.3, 'setosa'],
[5.1, 3.8, 1.6, 0.2, 'setosa'],
[4.6, 3.2, 1.4, 0.2, 'setosa'],
[5.3, 3.7, 1.5, 0.2, 'setosa'],
[5.0, 3.3, 1.4, 0.2, 'setosa'],
[7.0, 3.2, 4.7, 1.4, 'versicolor'],
[6.4, 3.2, 4.5, 1.5, 'versicolor'],
[6.9, 3.1, 4.9, 1.5, 'versicolor'],
[5.5, 2.3, 4.0, 1.3, 'versicolor'],
[6.5, 2.8, 4.6, 1.5, 'versicolor'],
[5.7, 2.8, 4.5, 1.3, 'versicolor'],
[6.3, 3.3, 4.7, 1.6, 'versicolor'],
[4.9, 2.4, 3.3, 1.0, 'versicolor'],
[6.6, 2.9, 4.6, 1.3, 'versicolor'],
```

```
[5.2, 2.7, 3.9, 1.4, 'versicolor'],
[5.0, 2.0, 3.5, 1.0, 'versicolor'],
[5.9, 3.0, 4.2, 1.5, 'versicolor'],
[6.0, 2.2, 4.0, 1.0, 'versicolor'],
[6.1, 2.9, 4.7, 1.4, 'versicolor'],
[5.6, 2.9, 3.6, 1.3, 'versicolor'],
[6.7, 3.1, 4.4, 1.4, 'versicolor'],
[5.6, 3.0, 4.5, 1.5, 'versicolor'],
[5.8, 2.7, 4.1, 1.0, 'versicolor'],
[6.2, 2.2, 4.5, 1.5, 'versicolor'],
[5.6, 2.5, 3.9, 1.1, 'versicolor'],
[5.9, 3.2, 4.8, 1.8, 'versicolor'],
[6.1, 2.8, 4.0, 1.3, 'versicolor'],
[6.3, 2.5, 4.9, 1.5, 'versicolor'],
[6.1, 2.8, 4.7, 1.2, 'versicolor'],
[6.4, 2.9, 4.3, 1.3, 'versicolor'],
[6.6, 3.0, 4.4, 1.4, 'versicolor'],
[6.8, 2.8, 4.8, 1.4, 'versicolor'],
[6.7, 3.0, 5.0, 1.7, 'versicolor'],
[6.0, 2.9, 4.5, 1.5, 'versicolor'],
[5.7, 2.6, 3.5, 1.0, 'versicolor'],
[5.5, 2.4, 3.8, 1.1, 'versicolor'],
[5.5, 2.4, 3.7, 1.0, 'versicolor'],
[5.8, 2.7, 3.9, 1.2, 'versicolor'],
[6.0, 2.7, 5.1, 1.6, 'versicolor'],
[5.4, 3.0, 4.5, 1.5, 'versicolor'],
[6.0, 3.4, 4.5, 1.6, 'versicolor'],
[6.7, 3.1, 4.7, 1.5, 'versicolor'],
[6.3, 2.3, 4.4, 1.3, 'versicolor'],
[5.6, 3.0, 4.1, 1.3, 'versicolor'],
[5.5, 2.5, 4.0, 1.3, 'versicolor'],
[5.5, 2.6, 4.4, 1.2, 'versicolor'],
[6.1, 3.0, 4.6, 1.4, 'versicolor'],
[5.8, 2.6, 4.0, 1.2, 'versicolor'],
[5.0, 2.3, 3.3, 1.0, 'versicolor'],
[5.6, 2.7, 4.2, 1.3, 'versicolor'],
[5.7, 3.0, 4.2, 1.2, 'versicolor'],
[5.7, 2.9, 4.2, 1.3, 'versicolor'],
[6.2, 2.9, 4.3, 1.3, 'versicolor'],
[5.1, 2.5, 3.0, 1.1, 'versicolor'],
[5.7, 2.8, 4.1, 1.3, 'versicolor'],
[6.3, 3.3, 6.0, 2.5, 'virginica'],
[5.8, 2.7, 5.1, 1.9, 'virginica'],
[7.1, 3.0, 5.9, 2.1, 'virginica'],
[6.3, 2.9, 5.6, 1.8, 'virginica'],
[6.5, 3.0, 5.8, 2.2, 'virginica'],
[7.6, 3.0, 6.6, 2.1, 'virginica'],
```

```
       [4.9, 2.5, 4.5, 1.7, 'virginica'],
       [7.3, 2.9, 6.3, 1.8, 'virginica'],
       [6.7, 2.5, 5.8, 1.8, 'virginica'],
       [7.2, 3.6, 6.1, 2.5, 'virginica'],
       [6.5, 3.2, 5.1, 2.0, 'virginica'],
       [6.4, 2.7, 5.3, 1.9, 'virginica'],
       [6.8, 3.0, 5.5, 2.1, 'virginica'],
       [5.7, 2.5, 5.0, 2.0, 'virginica'],
       [5.8, 2.8, 5.1, 2.4, 'virginica'],
       [6.4, 3.2, 5.3, 2.3, 'virginica'],
       [6.5, 3.0, 5.5, 1.8, 'virginica'],
       [7.7, 3.8, 6.7, 2.2, 'virginica'],
       [7.7, 2.6, 6.9, 2.3, 'virginica'],
       [6.0, 2.2, 5.0, 1.5, 'virginica'],
       [6.9, 3.2, 5.7, 2.3, 'virginica'],
       [5.6, 2.8, 4.9, 2.0, 'virginica'],
       [7.7, 2.8, 6.7, 2.0, 'virginica'],
       [6.3, 2.7, 4.9, 1.8, 'virginica'],
       [6.7, 3.3, 5.7, 2.1, 'virginica'],
       [7.2, 3.2, 6.0, 1.8, 'virginica'],
       [6.2, 2.8, 4.8, 1.8, 'virginica'],
       [6.1, 3.0, 4.9, 1.8, 'virginica'],
       [6.4, 2.8, 5.6, 2.1, 'virginica'],
       [7.2, 3.0, 5.8, 1.6, 'virginica'],
       [7.4, 2.8, 6.1, 1.9, 'virginica'],
       [7.9, 3.8, 6.4, 2.0, 'virginica'],
       [6.4, 2.8, 5.6, 2.2, 'virginica'],
       [6.3, 2.8, 5.1, 1.5, 'virginica'],
       [6.1, 2.6, 5.6, 1.4, 'virginica'],
       [7.7, 3.0, 6.1, 2.3, 'virginica'],
       [6.3, 3.4, 5.6, 2.4, 'virginica'],
       [6.4, 3.1, 5.5, 1.8, 'virginica'],
       [6.0, 3.0, 4.8, 1.8, 'virginica'],
       [6.9, 3.1, 5.4, 2.1, 'virginica'],
       [6.7, 3.1, 5.6, 2.4, 'virginica'],
       [6.9, 3.1, 5.1, 2.3, 'virginica'],
       [5.8, 2.7, 5.1, 1.9, 'virginica'],
       [6.8, 3.2, 5.9, 2.3, 'virginica'],
       [6.7, 3.3, 5.7, 2.5, 'virginica'],
       [6.7, 3.0, 5.2, 2.3, 'virginica'],
       [6.3, 2.5, 5.0, 1.9, 'virginica'],
       [6.5, 3.0, 5.2, 2.0, 'virginica'],
       [6.2, 3.4, 5.4, 2.3, 'virginica'],
       [5.9, 3.0, 5.1, 1.8, 'virginica']], dtype=object)
```

[79]:
```
# Convert feature columns to NumPy array
features = df.drop(columns='species')
```

```
features_array = features.to_numpy()
```

[80]:
```
# Compute the correlation matrix
correlation_matrix = np.corrcoef(features_array, rowvar=False)

# Print the correlation matrix
print("Correlation Matrix:\n", correlation_matrix)
```

```
Correlation Matrix:
 [[ 1.         -0.11756978  0.87175378  0.81794113]
 [-0.11756978  1.         -0.4284401  -0.36612593]
 [ 0.87175378 -0.4284401   1.          0.96286543]
 [ 0.81794113 -0.36612593  0.96286543  1.        ]]
```

Calculating percentiles

[83]:
```
np.percentile(arr[:,2], 50)
```

[83]:
```
4.35
```

[84]:
```
per=[0, 25, 50, 75, 100]
np.percentile(arr[:, 2], per)
```

[84]:
```
array([1.0, 1.6, 4.35, 5.1, 6.9], dtype=object)
```

1.1 Conclusion

NumPy significantly boosts efficiency in data science by offering fast, memory-efficient operations on large datasets through its powerful array-based computations. Unlike traditional Python lists, NumPy arrays support vectorized operations, allowing for quick calculations and reduced code complexity. Features like built-in statistical functions, multi-dimensional arrays, and broadcasting simplify complex numerical analyses and data manipulations. This makes NumPy an essential tool for handling and analyzing data, ensuring faster execution and streamlined workflows compared to native Python data structures.

1.2 Role of numpy in data science

NumPy is really important in many real-world scenarios. For instance, in machine learning, it helps handle and process large amounts of data quickly, which is essential for training and testing algorithms. In financial analysis, NumPy is used to do complex calculations like evaluating investment risks and analyzing stock market trends. For scientific research, it supports tasks such as running simulations, solving mathematical equations, and analyzing experimental data. Its ability to perform these operations efficiently makes it a key tool in these areas.

In addition to these areas, NumPy is also widely used in fields like engineering and healthcare. For engineers, it helps in modeling physical systems and analyzing data from sensors or simulations. In healthcare, NumPy is valuable for processing and analyzing medical images or genetic data, enabling advancements in diagnostics and personalized medicine. Its efficiency and power make it

a versatile tool across various disciplines, making complex numerical and data analysis tasks more manageable and accurate.

[ ]: