numpy

August 31, 2024

1 NUMPY IN DATASCIENCE

2 1.GETTING FAMILIRITY WITH NUMPY

[ ]:
```
NumPy (Numerical Python) is an open source Python library that’s widely used in␣
 ↪science and engineering.
The NumPy library contains multidimensional array data structures, such as the␣
 ↪homogeneous, N-dimensional ndarray, and a large library of functions that
operate efficiently on these data structures.
```

3 How to import NumPy

After installing NumPy, it may be imported into Python code like:

[3]:
```
import numpy as np
```

[ ]:
```
The core functionality of NumPy, or Numerical Python, is to perform numerical␣
 ↪computations on arrays and matrices. NumPy's core functionality includes:
Data structures
NumPy provides multidimensional array and matrix data structures, including␣
 ↪ndarray, a homogeneous n-dimensional array object.
Mathematical functions
NumPy provides a large library of high-level mathematical functions for␣
 ↪performing operations on arrays and matrices, such as vector-vector␣
 ↪multiplication, matrix-matrix multiplication, and element-wise operations.
Efficiency
NumPy is optimized for speed and efficiency, especially for large datasets.␣
 ↪It's written in low-level languages and uses an array-oriented computing␣
 ↪paradigm.
```

4 CREATING 1D ARRAY

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

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

we can create a numpy array using these:

```
[9]: # creating a zeroes array
b=np.zeros(6)
print(b)
```

```
[0. 0. 0. 0. 0. 0.]
```

```
[11]: #creating a ones array
c=np.ones(5)
print(c)
```

```
[1. 1. 1. 1. 1.]
```

The function empty creates an array whose initial content is random and depends on the state of the memory.

```
[13]: # Create an empty array with 2 elements
d=np.empty(2)
print(d)
```

```
[0. 0.]
```

```
[17]: #create an array with a range of elements:

a=np.arange(4)
print(a)
print(np.arange(2, 9, 2))
```

```
[0 1 2 3]
[2 4 6 8]
```

```
[21]: #by generating random numbers
data=np.random.randn(5)
data
```

```
[21]: array([ 0.38541427, -0.04396427, -0.51441549, -0.19888673,  1.75617497])
```

5 ARRAY PROPERTIES

ndarray.ndim explains about the number of axes, or dimensions, of the array.

ndarray.size explains about the total number of elements of the array. This is the product of the elements of the array's shape.

ndarray.shape explains about a tuple of integers that indicate the number of elements stored along each dimension of the array

```
[23]: arr=np.array([[0, 1, 2, 3],[4, 5, 6, 7]])
      arr
```

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

```
[25]: arr.ndim
```

```
[25]: 2
```

```
[27]: arr.size
```

```
[27]: 8
```

```
[29]: arr.shape
```

```
[29]: (2, 4)
```

6 2. Data Manipulation:

INDEXING AND SLICING

```
[31]: # CREATION OF A NUMPY ARRAY


      data = np.array([1, 2, 3])

      data[1]
```

```
[31]: 2
```

```
[33]: data[0:2]
      data[1:]
      data[-2:]
```

```
[33]: array([2, 3])
```

```
[35]: A= np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
      A
```

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

```
[37]: A[1:2]
```

```
[37]: array([[5, 6, 7, 8]])
```

[43]:
```
A[1,3]
```

[43]:
```
8
```

[45]:
```
A[-0:]
```

[45]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

7 RESHAPING

Using arr.reshape() will give a new shape to an array without changing the data.

[47]:
```
a = np.arange(6)
print(a)
```

```
[0 1 2 3 4 5]
```

[49]:
```
b = a.reshape(3, 2)
print(b)
```

```
[[0 1]
 [2 3]
 [4 5]]
```

[53]:
```
c=np.arange(20)
c
```

[53]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

[55]:
```
d=c.reshape(5,4)
d
```

[55]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19]])
```

8 MATHEMATICAL OPERATIONS

[57]:
```
#ADDITION OF ARRAYS

e=d+d
e
```

[57]:
```
array([[ 0,  2,  4,  6],
       [ 8, 10, 12, 14],
       [16, 18, 20, 22],
       [24, 26, 28, 30],
       [32, 34, 36, 38]])
```

[59]:
```
# MULTIPLICATION OF ARRAYS

f=e*e
f
```

[59]:
```
array([[   0,    4,   16,   36],
       [  64,  100,  144,  196],
       [ 256,  324,  400,  484],
       [ 576,  676,  784,  900],
       [1024, 1156, 1296, 1444]])
```

[63]:
```
# SUBTARACTION OF TWO ARRAYS

g=f-e
g
```

[63]:
```
array([[   0,    2,   12,   30],
       [  56,   90,  132,  182],
       [ 240,  306,  380,  462],
       [ 552,  650,  756,  870],
       [ 992, 1122, 1260, 1406]])
```

[67]:
```
# DIVISON
arr=f/2
arr
```

[67]:
```
array([[  0.,   2.,   8.,  18.],
       [ 32.,  50.,  72.,  98.],
       [128., 162., 200., 242.],
       [288., 338., 392., 450.],
       [512., 578., 648., 722.]])
```

[69]:
```
arr1=np.sqrt(arr)
arr1
```

[69]:
```
array([[ 0.        ,  1.41421356,  2.82842712,  4.24264069],
       [ 5.65685425,  7.07106781,  8.48528137,  9.89949494],
       [11.3137085 , 12.72792206, 14.14213562, 15.55634919],
       [16.97056275, 18.38477631, 19.79898987, 21.21320344],
       [22.627417  , 24.04163056, 25.45584412, 26.87005769]])
```

9 3. Data Aggregation:

Use Numpy functions to compute summary statistics like mean, median, standard deviation, and sum.

[71]:
```
array=np.random.randint(1,50,5)
array
```

[71]:
```
array([39, 23, 27, 43,  3])
```

[73]:
```
# TO CALCULATE MEAN

mean=np.mean(array)
mean
```

[73]:
```
27.0
```

[75]:
```
# TO CALCULATE MEDIAN

median=np.median(array)
median
```

[75]:
```
27.0
```

[77]:
```
# TO CALCULATE STANDARD DEVIATION

sd=np.std(array)
sd
```

[77]:
```
14.085453489327207
```

[79]:
```
# TO CALCULATE THE SUM OF ELEMENTS

sum=np.sum(array)
sum
```

[79]:
```
135
```

[102]:
```
# GROUPING DATA AND PERFORMING AGGREGATIONS

classes = np.array(['Class A', 'Class B', 'Class A', 'Class C', 'Class B',␣
 ↪'Class A', 'Class C', 'Class B'])
scores = np.array([85, 78, 92, 88, 75, 95, 90, 82])

# Display the data
print("Classes:", classes)
print("Scores:", scores)
# Get unique classes
```

```
unique_classes = np.unique(classes)

# Initialize an array to store average scores
average_scores = np.zeros(len(unique_classes))

# Calculate the average score for each class
for i, cls in enumerate(unique_classes):
    # Create a mask for the current class
    mask = classes == cls
    # Calculate the average score for this class
    average_scores[i] = np.mean(scores[mask])

# Display the results
for cls, avg_score in zip(unique_classes, average_scores):
    print(f"Average score for {cls}: {avg_score:.2f}")
```

```
Classes: ['Class A' 'Class B' 'Class A' 'Class C' 'Class B' 'Class A' 'Class C'
 'Class B']
Scores: [85 78 92 88 75 95 90 82]
Average score for Class A: 90.67
Average score for Class B: 78.33
Average score for Class C: 89.00
```

10 4.DATA ANALYSIS

#CORRELATION IN NUMPY The correlation coefficient measures the strength and direction of a linear relationship between two variables.

[81]:
```
X = np.array([1, 2, 3, 4, 5])
Y = np.array([2, 4, 6, 8, 10])
```

[85]:
```
correlation_matrix = np.corrcoef(X, Y)

# Extract correlation coefficient
correlation_coefficient = correlation_matrix[0, 1]
print(correlation_matrix)
print(correlation_coefficient)
```

```
[[1. 1.]
 [1. 1.]]
0.9999999999999999
```

11 OUTLIERS IN NUMPY

Outliers are data points that differ significantly from other observations. They can occur due to variability in the data or errors. Identifying and handling outliers is important because they can skew statistical analyses and lead to misleading results.

[94]:
```python
data = np.array([10, 12, 12, 13, 12, 11, 14, 10, 13, 100])
# Calculate the mean and standard deviation
mean = np.mean(data)
std_dev = np.std(data)

# Calculate Z-scores
z_scores = (data - mean) / std_dev

print("Z-scores:", z_scores)

# Identify outliers (Z-score > 3 or Z-score < -3)
outliers = data[np.abs(z_scores) > 3]

# Display the outliers
print("Outliers:", outliers)
```

```
Z-scores: [-0.40436134 -0.32877978 -0.32877978 -0.290989   -0.32877978
-0.36657056
 -0.25319822 -0.40436134 -0.290989    2.99680878]
Outliers: []
```

12 CALCULATING PERCENTILES

Calculating percentiles in NumPy is straightforward using the np.percentile() function. Percentiles are used to understand the distribution of data, with the nth percentile indicating the value below which n% of the data falls.

[96]:
```python
data = np.array([15, 20, 35, 40, 50, 75, 100])
percentile_25 = np.percentile(data, 25)
percentile_50 = np.percentile(data, 50)  # Median
percentile_75 = np.percentile(data, 75)
print(percentile_25)
print(percentile_50)
print(percentile_75)
```

```
27.5
40.0
62.5
```

EFFICIENCY OF NUMPY WITH LARGE DATASETS NumPy is highly efficient for handling large datasets due to its design and underlying implementation. It leverages optimized C and Fortran libraries, allowing it to perform operations faster and use memory more efficiently than standard Python lists or loops.

[100]:
```python
import numpy as np
import time

# Create a large array
```

```
size = 10**7
array = np.random.rand(size)

# Using NumPy
start = time.time()
sum_np = np.sum(array)
end = time.time()
print(f"NumPy sum time: {end - start} seconds")

# Using Python loop
start = time.time()
end = time.time()
print(f"Python sum time: {end - start} seconds")
```

```
NumPy sum time: 0.02160024642944336 seconds
Python sum time: 0.0 seconds
```

13 CONCLUSION

14 APPLICATIONS OF NUMPY IN DATASCIENCE

NumPy is a fundamental package for data science professionals, especially when working with numerical data and performing mathematical operations. Here's why:

Efficient Numerical Computations: Speed: NumPy arrays are more efficient than Python lists because they are stored in contiguous blocks of memory. Vectorization: Many operations in NumPy can be vectorized, meaning they can be applied simultaneously to all elements in an array without the need for explicit loops.

Memory Efficiency: NumPy arrays take up less space compared to Python lists, particularly when dealing with large datasets.

Mathematical Functions: NumPy provides a wide range of built-in mathematical functions that are optimized for performance. These functions can handle operations such as linear algebra, Fourier transforms, and random number generation, which are essential in many data science tasks.

Integration with Other Libraries: NumPy is the backbone of many other data science libraries, such as Pandas, SciPy, and Scikit-learn.

Advantages of Using NumPy Over Traditional Python Data Structures: Performance: NumPy arrays are optimized for numerical computations, providing significant performance improvements over Python lists, especially in large-scale data processing. Flexibility: With NumPy, you can perform complex operations on large datasets with just a few lines of code, thanks to its rich set of functions and broadcasting capabilities. Scalability: As datasets grow in size, the efficiency of NumPy becomes increasingly important. Its ability to handle large, multidimensional arrays and perform computations in parallel is a key advantage.

In summary, using NumPy in your data science projects not only improves performance and memory efficiency but also simplifies code and enhances your ability to work with large and complex datasets. This makes it an indispensable tool for any data science professional.

NumPy's capabilities are crucial in various real-world applications across different fields.

1. Machine Learning: Data Preprocessing: NumPy is often used to handle and manipulate large datasets before feeding them into machine learning algorithms. Training Models: NumPy is integral in implementing the mathematical operations required for training machine learning models.

2. Financial Analysis: Portfolio Optimization: Financial analysts use NumPy to perform complex calculations such as covariance matrices, expected returns, and risk assessments. Option Pricing: NumPy is utilized in financial engineering to model and price options using methods like Monte Carlo simulations, which require handling large datasets and performing repeated numerical computations efficiently.

3. Scientific Research: Physics Simulations: In computational physics, NumPy is used to perform simulations that require heavy numerical computations, such as solving differential equations or modeling physical systems Genomics: In bioinformatics, NumPy is employed to analyze large datasets, such as DNA sequences Climate Modeling: Researchers use NumPy to analyze and model climate data.

In all these fields, NumPy's ability to handle large datasets, perform complex mathematical operations efficiently, and integrate seamlessly with other scientific computing libraries makes it an indispensable tool for professionals.