Day 05 - Python Numpy Array Notes

Introduction to Numpy

Numpy (Numerical Python) is a fundamental package for scientific computing in Python. It provides support for arrays, mathematical functions, random number generation, and more.

Numpy: NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

Key features:

- Efficient storage and manipulation of numerical data.
- Powerful N-dimensional array object.
- Functions for performing element-wise computations and array operations.

To install Numpy, use:

```
pip install numpy
```

To import Numpy in your code:

```
import numpy as np
```

Array

An **array** is a grid of values, all of the same type, indexed by a tuple of non-negative integers. Numpy arrays are more efficient than Python lists for numerical operations.

An **array** is a data structure that stores values of same data type. In Python, this is the main difference between arrays and lists. While python lists can contain values corresponding to different data types, arrays in python can only contain values corresponding to same data type

Creating Arrays

```
import numpy as np

# Creating a 1D array
arr1d = np.array([1, 2, 3, 4, 5])
print(arr1d)

# Creating a 2D array
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2d)
```

Shape

The **shape** of an array is a tuple that gives the dimensions of the array. For a matrix with n rows and m columns, the shape will be (n, m).

Example

```
# Shape of a 1D array
print(arr1d.shape)  # Output: (5,)

# Shape of a 2D array
print(arr2d.shape)  # Output: (2, 3)
```

Single and Multidimensional Arrays

- **Single-dimensional arrays**: Also known as 1D arrays or vectors.
- **Multidimensional arrays**: Arrays with more than one dimension, such as 2D matrices or 3D tensors.

Examples

```
# Single-dimensional array
arr1d = np.array([1, 2, 3])

# Multidimensional array
arr3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print(arr3d.shape)  # Output: (2, 2, 2)
```

Exploratory Data Analysis (EDA) over Arrays

EDA involves summarizing the main characteristics of arrays, often with visual methods. This includes:

- Calculating statistics like mean, median, standard deviation.
- Visualizing data with plots (using libraries like Matplotlib).

Examples

```
# Basic statistical operations
mean = np.mean(arr1d)
std_dev = np.std(arr1d)
print(f"Mean: {mean}, Standard Deviation: {std_dev}")

# Using Matplotlib for visualization
import matplotlib.pyplot as plt
```

```
plt.hist(arr1d, bins=5)
plt.show()
```

reshape()

The **reshape()** method changes the shape of an array without changing its data.

Example

```
arr = np.array([1, 2, 3, 4, 5, 6])
reshaped_arr = arr.reshape((2, 3))
print(reshaped_arr)
# Output:
# [[1 2 3]
#  [4 5 6]]
```

arange()

The **arange()** function returns evenly spaced values within a given interval, similar to Python's built-in `range()` but for Numpy arrays.

Example

```
arr = np.arange(0, 10, 2)
print(arr)  # Output: [0 2 4 6 8]
```

ones()

The **ones()** function creates an array filled with ones.

Example

```
arr = np.ones((3, 3))
print(arr)
# Output:
# [[1. 1. 1.]
#  [1. 1. 1.]
#  [1. 1. 1.]]
```

zeros()

The **zeros()** function creates an array filled with zeros.

Example

```
arr = np.zeros((2, 4))
print(arr)
# Output:
# [[0. 0. 0. 0.]
#  [0. 0. 0. 0.]]
```

zeros_like()

The **zeros_like()** function returns an array of zeros with the same shape and type as a given array.

Example

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
zeros_like_arr = np.zeros_like(arr)
print(zeros_like_arr)
# Output:
# [[0 0 0]
#  [0 0 0]]
```

random methods and submethods

Numpy's **random** module provides methods to generate random numbers and arrays.

Examples

```
# Random number between 0 and 1
random_num = np.random.rand()
print(random_num)

# Random array of shape (2, 3)
random_arr = np.random.rand(2, 3)
print(random_arr)

# Random integers
random_int = np.random.randint(1, 10)
print(random_int)

# Random choice from a list
choices = np.random.choice([1, 2, 3, 4, 5], size=3)
print(choices)

# Random normal distribution
normal_arr = np.random.randn(3, 3)
print(normal_arr)
```

Some Methods of Arrays

- `np.array()`: Creates a NumPy array from a list.
- `arr.shape`: Returns the shape of the array.
- `arr.dtype`: Returns the data type of the array.
- `arr.reshape()`: Reshapes the array to a specified shape.
- `np.arange()`: Creates an array with a range of values.
- `np.ones()`: Creates an array filled with ones.
- `np.zeros()`: Creates an array filled with zeros.
- `np.random.randint()`: Generates random integers within a specified range.
- `np.random.randn()`: Generates random numbers from a standard normal distribution.
- `np.random.random_sample()`: Generates random numbers between 0 and 1.
- `np.dot()`: Computes the dot product of two arrays.
- `np.sum()`: Computes the sum of array elements.
- `np.mean()`: Computes the mean of array elements.
- `np.max()`: Finds the maximum value in an array.
- `np.min()`: Finds the minimum value in an array.
- `np.std()`: Computes the standard deviation of array elements.
- `np.var()`: Computes the variance of array elements.
- `df.plot()`: Plots the data in a DataFrame.

These notes cover the essential concepts and functionalities related to Numpy arrays, complete with examples to illustrate usage.