**NumPy**

NumPy is a Python library.

NumPy is used for working with arrays.

NumPy is short for "Numerical Python".

Joining NumPy Arrays

Joining means putting contents of two or more arrays in a single array.

We pass a sequence of arrays that we want to join to the concatenate() function, along with the axis.

import numpy as np  
  
arr1 = np.array([1, 2, 3])  
  
arr2 = np.array([4, 5, 6])  
  
arr = np.concatenate((arr1, arr2))  
  
print(arr)

## Joining Arrays Using Stack Functions

Stacking is same as concatenation, the only difference is that stacking is done along a new axis.

We can concatenate two 1-D arrays along the second axis which would result in putting them one over the other, ie. stacking.

We pass a sequence of arrays that we want to join to the stack() method along with the axis. If axis is not explicitly passed it is taken as 0.

import numpy as np  
  
arr1 = np.array([1, 2, 3])  
  
arr2 = np.array([4, 5, 6])  
  
arr = np.stack((arr1, arr2), axis=1)  
  
print(arr)

## Stacking Along Rows

NumPy provides a helper function: hstack() to stack along rows.

import numpy as np  
  
arr1 = np.array([1, 2, 3])  
  
arr2 = np.array([4, 5, 6])  
  
arr = np.hstack((arr1, arr2))  
  
print(arr)

## Stacking Along Columns

NumPy provides a helper function: vstack()  to stack along columns.

import numpy as np  
  
arr1 = np.array([1, 2, 3])  
  
arr2 = np.array([4, 5, 6])  
  
arr = np.vstack((arr1, arr2))  
  
print(arr)

## Stacking Along Height (depth)

NumPy provides a helper function: dstack() to stack along height, which is the same as depth.

import numpy as np  
  
arr1 = np.array([1, 2, 3])  
  
arr2 = np.array([4, 5, 6])  
  
arr = np.dstack((arr1, arr2))  
  
print(arr)

## Splitting NumPy Arrays

Splitting is reverse operation of Joining.

Joining merges multiple arrays into one and Splitting breaks one array into multiple.

We use array\_split() for splitting arrays, we pass it the array we want to split and the number of splits.

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5, 6])  
  
newarr = np.array\_split(arr, 3) # 3->How many slit are  
  
print(newarr)

An alternate solution is using hsplit() opposite of hstack()

Use the hsplit() method to split the 2-D array into three 2-D arrays along rows.

import numpy as np  
  
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])  
  
newarr = np.hsplit(arr, 3)  
  
print(newarr)

Similar alternates to vstack() and dstack() are available as vsplit() and dsplit().

## Searching Arrays

You can search an array for a certain value, and return the indexes that get a match.

To search an array, use the where() method.

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5, 4, 4])  
  
x = np.where(arr == 4)  
  
print(x)

## Search Sorted

There is a method called searchsorted() which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

The searchsorted() method is assumed to be used on sorted arrays.

import numpy as np  
  
arr = np.array([6, 7, 8, 9])  
  
x = np.searchsorted(arr, 7)  
  
print(x)

import numpy as np  
  
arr = np.array([6, 7, 8, 9])  
  
x = np.searchsorted(arr, 7, side='right') #left or right  
  
print(x)

Find the indexes where the values 2, 4, and 6 should be inserted:

## Sorting Arrays

Sorting means putting elements in an ordered sequence.

Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a specified array.

import numpy as np  
  
arr = np.array([3, 2, 0, 1])  
  
print(np.sort(arr))

## Filtering Arrays

Getting some elements out of an existing array and creating a new array out of them is called filtering.

In NumPy, you filter an array using a boolean index list.

import numpy as np  
  
arr = np.array([41, 42, 43, 44])  
  
x = [True, False, True, False]  
  
newarr = arr[x]  
  
print(newarr)

Create a filter array that will return only even elements from the original array

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5, 6, 7])  
  
# Create an empty list  
filter\_arr = []  
  
# go through each element in arr  
for element in arr:  
  # if the element is completely divisble by 2, set the value to True, otherwise False  
  if element % 2 == 0:  
    filter\_arr.append(True)  
  else:  
    filter\_arr.append(False)  
  
newarr = arr[filter\_arr]  
  
print(filter\_arr)  
print(newarr)

Create a filter array that will return only values higher than 42:

import numpy as np  
  
arr = np.array([41, 42, 43, 44])  
  
filter\_arr = arr > 42  
  
newarr = arr[filter\_arr]  
  
print(filter\_arr)  
print(newarr)

Create a filter array that will return only even elements from the original array:

import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5, 6, 7])  
  
filter\_arr = arr % 2 == 0  
  
newarr = arr[filter\_arr]  
  
print(filter\_arr)  
print(newarr)

## Generate Random Number

NumPy offers the random module to work with random numbers.

Example: Generate a random integer from 0 to 100:

from numpy import random  
  
x = random.randint(100)  
  
print(x)

## Generate Random Float

The random module's rand() method returns a random float between 0 and 1.

### Example: Generate a random float from 0 to 1:

from numpy import random  
  
x = random.rand()  
  
print(x)

## Generate Random Array

In NumPy we work with arrays, and you can use the two methods from the above examples to make random arrays.

### **Integers**

The randint() method takes a size parameter where you can specify the shape of an array.

### **Example**

Generate a 1-D array containing 5 random integers from 0 to 100:

from numpy import random  
  
x=random.randint(100, size=(5))  
  
print(x)

### **Example**

Generate a 2-D array with 3 rows, each row containing 5 random integers from 0 to 100:

from numpy import random  
  
x = random.randint(100, size=(3, 5))  
  
print(x)

The rand() method also allows you to specify the shape of the array.

### **Example**

Generate a 1-D array containing 5 random floats:

from numpy import random  
  
x = random.rand(5)  
  
print(x)

## Generate Random Number From Array

The choice() method allows you to generate a random value based on an array of values.

The choice() method takes an array as a parameter and randomly returns one of the values.

### **Example**

Return one of the values in an array:

from numpy import random  
  
x = random.choice([3, 5, 7, 9])

The choice() method also allows you to return an array of values.

Add a size parameter to specify the shape of the array.

print(x)

### **Example**

Generate a 2-D array that consists of the values in the array parameter (3, 5, 7, and 9):

from numpy import random  
  
x = random.choice([3, 5, 7, 9], size=(3, 5))  
  
print(x)

The statement from sklearn import metrics is used in Python to import the metrics module from the scikit-learn (often abbreviated as sklearn) library. Here's a step-by-step explanation:

### 1. **Understanding** scikit-learn **(sklearn)**

* **scikit-learn** is a popular machine learning library in Python. It provides simple and efficient tools for data mining and data analysis.
* It includes many modules that allow you to perform various machine learning tasks like classification, regression, clustering, and more.

### 2. **Importing a Specific Module**

* Python allows you to import specific modules from a library. The statement from sklearn import metrics is doing just that—it's importing only the metrics module from the entire sklearn library.
* This is useful because it makes your code more readable and efficient by only loading the part of the library that you need.

### 3. **What is the** metrics **Module?**

* The metrics module in scikit-learn includes functions to evaluate the performance of machine learning models.
* It includes a wide range of tools to assess classification models (like accuracy, precision, recall, F1 score, etc.), regression models (like mean squared error, R^2 score, etc.), and more.

### 4. **How to Use** metrics

* Once you’ve imported the metrics module, you can use any of its functions directly in your code. For example:

from sklearn import metrics

# Assuming you have actual and predicted labels

actual = [0, 1, 1, 0, 1]

predicted = [0, 1, 0, 0, 1]

accuracy = metrics.accuracy\_score(actual, predicted)

print(f"Accuracy: {accuracy}")

### Summary

* from sklearn import metrics is a Python statement that imports the metrics module from the scikit-learn library, allowing you to use various functions to evaluate machine learning models.