Introduction to Scipy

- SciPy is an open-source Python library which is used to solve scientific and mathematical problems.
- **It is built on the NumPy extension and allows the user to manipulate and visualize data with a wide range of high-level commands.**
- SciPy builds on NumPy and therefore if you import SciPy, there is no need to import NumPy.

NumPy vs SciPy

- Both NumPy and SciPy are Python libraries used for used mathematical and numerical analysis.
- NumPy contains array data and basic operations such as sorting, indexing, etc whereas, SciPy consists of all the numerical code.
- Though NumPy provides a number of functions that can help resolve linear algebra, Fourier transforms, etc, SciPy is the library that actually contains fully-featured versions of these functions along with many others.
- However, if you are doing scientific analysis using Python, you will need to install both NumPy and SciPy since SciPy builds on NumPy.

Subpackages in SciPy:

SciPy has a number of subpackages for various scientific computations which are shown in the following table:

Name	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output

linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions

These packages need to be imported exclusively prior to using them. For example:

```
from scipy import cluster
```

Basic Functions:

Interaction with NumPy:

- SciPy builds on NumPy and therefore you can make use of NumPy functions itself to handle arrays.
- You can simply make use of help(), info() or source() functions.

help():

To get information about any function, you can make use of the help() function. There are two ways in which this function can be used:

- without any parameters
- using parameters

```
from scipy import cluster
help(cluster)                #with parameter
help()                        #without parameter
```

```
Help on package scipy.cluster in scipy:

NAME
    scipy.cluster

DESCRIPTION
    ========================================
    Clustering package (:mod:`scipy.cluster`)
    ========================================
```

info():

This function returns information about the desired functions, modules, etc.

```
import scipy
scipy.info(cluster)
```

```
========================================
Clustering package (:mod:`scipy.cluster`)
========================================
```

source():

The source code is returned only for objects written in Python. This function does not return useful information in case the methods or objects are written in any other language such as C.

```
scipy.source(cluster)
```

```
In file: C:\Users\Cloud Analogy\anaconda3\Lib\site-packages\scipy\cluster\__init__.py

"""
========================================
Clustering package (:mod:`scipy.cluster`)
========================================
```

Special Functions:

Exponential and Trigonometric Functions:

```
from scipy import special
a = special.exp10(3)
print(a)

b = special.exp2(3)
print(b)

c = special.sindg(90)          # sin 90 = 1
print(c)

d = special.cosdg(45)          #cos 45
print(d)
```

```
1000.0
8.0
1.0
0.7071067811865475
```

Linear Algebra:

Finding the Inverse of a Matrix:

```
import numpy as np
from scipy import linalg
A = np.array([[1,2], [4,3]])
B = linalg.inv(A)
print(B)
```

```
[[-0.6  0.4]
 [ 0.8 -0.2]]
```

Finding the Determinants:

```
import numpy as np
from scipy import linalg
A = np.array([[1,2], [4,3]])
B = linalg.det(A)
print(B)
```

```
-5.0
```