Introduction to NumPy

This chapter, along with chapter 3, outlines techniques for effectively loading, storing, and manipulating in-memory data in Python. The topic is very broad: datasets can come from a wide range of sources and a wide range of formats, including be collections of documents, collections of images, collections of sound clips, collections of numerical measurements, or nearly anything else. Despite this apparent heterogeneity, it will help us to think of all data fundamentally as arrays of numbers.

For example, images–particularly digital images–can be thought of as simply two-dimensional arrays of numbers representing pixel brightness across the area. Sound clips can be thought of as one-dimensional arrays of intensity versus time. Text can be converted in various ways into numerical representations, perhaps binary digits representing the frequency of certain words or pairs of words. No matter what the data are, the first step in making it analyzable will be to transform them into arrays of numbers.

For this reason, efficient storage and manipulation of numerical arrays is absolutely fundamental to the process of doing data science.

NumPy (short for *Numerical Python*) provides an efficient interface to store and operate on dense data buffers. In some ways, NumPy arrays are like Python's built-in `list` type, but NumPy arrays provide much more efficient storage and data operations as the arrays grow larger in size. NumPy arrays form the core of nearly the entire ecosystem of data science tools in Python, so time spent learning to use NumPy effectively will be valuable no matter what aspect of data science interests you.

If you have already installed the Anaconda stack, you already have NumPy installed and ready to go. If you're more the do-it-yourself type, you can go to http://www.numpy.org/ and follow the installation instructions found there. Once you do, you can import NumPy and double-check the version:

Installing Numpy in notebook

In [ ]:
```
!pip install numpy
```

Installing Numpy in CLI

In [ ]:
```
pip install numpy
```

Double checking the Numpy Installation

In [1]:
```
import numpy
numpy.__version__
```

Out[1]:
```
'1.23.5'
```

By convention, you'll find that most people in the SciPy/PyData world will import NumPy using `np` as an alias:

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

Reminder about Built In Documentation

IPython gives you the ability to quickly explore the contents of a package (by using the tab-completion feature), as well as the documentation of various functions (using the `?`).

For example, to display all the contents of the numpy namespace, you can type this:

```
In [3]: np.<TAB>
```

And to display NumPy's built-in documentation, you can use this:

```
In [4]: np?
```

More detailed documentation, along with tutorials and other resources, can be found at http://www.numpy.org.

In [2]:
```
np?
```

Type: module
String form: <module 'numpy' from 'C:\\Users\\Swati Solanki\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\numpy__init__.py'>
File: c:\users\swati solanki\appdata\local\programs\python\python311\lib\site-packages\numpy__init__.py
Docstring:

```
NumPy
=====

Provides
  1. An array object of arbitrary homogeneous items
  2. Fast mathematical operations over arrays
  3. Linear Algebra, Fourier Transforms, Random Number Generation

How to use the documentation
----------------------------
Documentation is available in two forms: docstrings provided
with the code, and a loose standing reference guide, available from
`the NumPy homepage <https://numpy.org>`_.

We recommend exploring the docstrings using
`IPython <https://ipython.org>`_, an advanced Python shell with
TAB-completion and introspection capabilities.  See below for further
instructions.

The docstring examples assume that `numpy` has been imported as `np`::

  >>> import numpy as np

Code snippets are indicated by three greater-than signs::

  >>> x = 42
  >>> x = x + 1

Use the built-in ``help`` function to view a function's docstring::

  >>> help(np.sort)
  ... # doctest: +SKIP

For some objects, ``np.info(obj)`` may provide additional help.  This is
particularly true if you see the line "Help on ufunc object:" at the top
of the help() page.  Ufuncs are implemented in C, not Python, for speed.
```

```
The native Python help() does not know how to view their help, but our
np.info() function does.

To search for documents containing a keyword, do::

  >>> np.lookfor('keyword')
  ... # doctest: +SKIP

General-purpose documents like a glossary and help on the basic concepts
of numpy are available under the ``doc`` sub-module::

  >>> from numpy import doc
  >>> help(doc)
  ... # doctest: +SKIP

Available subpackages
---------------------
lib
    Basic functions used by several sub-packages.
random
    Core Random Tools
linalg
    Core Linear Algebra Tools
fft
    Core FFT routines
polynomial
    Polynomial tools
testing
    NumPy testing tools
distutils
    Enhancements to distutils with support for
    Fortran compilers support and more.

Utilities
---------
test
    Run numpy unittests
show_config
    Show numpy build configuration
dual
    Overwrite certain functions with high-performance SciPy tools.
```

```
    Note: `numpy.dual` is deprecated.  Use the functions from NumPy or Scipy
    directly instead of importing them from `numpy.dual`.
matlib
    Make everything matrices.
__version__
    NumPy version string

Viewing documentation using IPython
-----------------------------------
Start IPython with the NumPy profile (``ipython -p numpy``), which will
import `numpy` under the alias `np`.  Then, use the ``cpaste`` command to
paste examples into the shell.  To see which functions are available in
`numpy`, type ``np.<TAB>`` (where ``<TAB>`` refers to the TAB key), or use
``np.*cos*?<ENTER>`` (where ``<ENTER>`` refers to the ENTER key) to narrow
down the list.  To view the docstring for a function, use
``np.cos?<ENTER>`` (to view the docstring) and ``np.cos??<ENTER>`` (to view
the source code).

Copies vs. in-place operation
-----------------------------
Most of the functions in `numpy` return a copy of the array argument
(e.g., `np.sort`).  In-place versions of these functions are often
available as array methods, i.e. ``x = np.array([1,2,3]); x.sort()``.
Exceptions to this rule are documented.
```

In [ ]:
```
np.
```