Python Libraries for Data Science

NumPy is a numeric processing API for Python, good for fast mathematical computation of numeric arrays and matrices.

Pandas provides additional features based on NumPy. It provides additional support for indexing, reading/writing CSV/Excel, etc

Matplotlib is a graphical plotting API for Python. Similar to Matlab, it allows you to plot graphs and charts.

Scikit-Learn is a machine learning library for Python. It can be used to implement many supervised/unsupervised learning algorithms.

Install the libraries we're going to use

numpy
openpyxl
xlrd
matplotlib
pandas

- Getting Started with NumPy arrays
- Techniques for creating NumPy arrays
- Reading CSV data
- Visualizing data

NumPy holds data in N-dimensional arrays
- An array is an instance of the numpy.ndarray class
- https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html

All the data in a NumPy array is the same type
- This allows NumPy to store and process the data dfficently

Why are NumPy arrays more efficient than Python lists?
- Python is dynamically typed, so every object contains metadata that identifies the type at run time
- In a Python list, every item contains this metadata - eek!
- In a NumPy array, only the array itself contains the metadata

Examples 1-3

Reading CSV Data

A common requirement is to read data from a CSV file
- The easiest way to do this is via the Pandas read_csv() function

Pandas reads values into a multi-column DataFrame
- You can then extract a column into a NumPy array

We discuss Pandas in detail later in the course

Example 4

Visualizing Data

Visualization is an important aid to help you understand the shape and meaning of data

You can use the MatPlotLib library to visualize data in lots of different ways
- Line graphs
- Scatter graphs
- Bar-charts
- Pie-charts
- Histograms
- Etc.

Example 5

- Indexing into an array
- Slicing an array
- Accessing a specific column or row
- Aside: Views vs. copies

Indexing into a NumPy array is quite intuitive
- [i] Access element from start, first element is at [0]
- [-i] Access element from end, last element is at [-1]
- [r,c] Access element in 2-D array (etc. for higher dimensions)

Ex 6

You can slice into an array using a [start:stop:step] index
- start Default start is 0
- stop Default stop is the size of the dimension
- step Default step is 1

Ex 7

To get a specific column or row in a multidimension array:
- Use an empty slice to skip a dimension
- E.g. in a 2D array, [:,1] gets column 1
- E.g. in a 2D array, [1,:] gets row 1
- step Default step is 1

Ex 8

When you get an array slice/row/column, you get a view on the data
- If you make any changes, it will change the actual data

If you want to get a copy of the data:
- Call copy() on the slice/row/column

Ex 9

- Reshaping an array
- Creating new axes
- Concatenating arrays
- Stacking arrays vertically or horizontally
- Splitting an array

Reshaping is a simple and common technique for creating multidimensional arrays
- Create a 1D array initially (typically)
- Reshape it to a multidimensional array (must be compatible shape)
- The multidimensional array is a view onto the original 1D array

Ex 10

Another useful technique is create new axes for an array
- Create a 1D array initially (typically)
- Create a new column or row, using np.newaxis

Ex 11

You can concatenate same-size arrays together
- np.concatenate() - you can specify the axis to concatenate on

Ex 12

Here's an example that concatenates 2D arrays
- Note the optional axis parameter (default is 0)

Ex 13

You can stack different-size arrays together
- np.vstack() - stack vertically (must have same no. of cols)
- np.hstack() - stack horizontally (must have same no. of rows)

Ex 14

You can split an array into subarrays
- np.split()
- np.vsplit()
- np.hsplit()

Ex 15