NumPy Basics Cheat Sheet

NumPy Arrays: are different from Python lists because they must be composed of homogenous elements (same datatype) in order for the mathematical operations to be worked on them.

NumPy arrays are n-dimensional grids composed of elements and they contain information on how to refer to individual elements through an index.

- 1-dimensional array: vector
- 2-dimensional array: matrix
- 3-dimensional array: tensor

These various dimensions are referred to using "axes."

Axes: are the dimensions of the array and are 0-based, just like Python based container-syntax

Making arrays:

```
# %%
import numpy as np

# %%
arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])
```

- Combine the arrays:

```
>>> arr3 = np.concatenate((arr1, arr2))
      array([1, 2, 3, 4, 5, 6])
```

- Make array of zeros:

```
arr_zeros = np.zeros(10)
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

- Maker array of ones:

```
arr_ones = np.ones(10)
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

- Make array of evenly spaced numbers for a set range:

```
linespace_arr = np.linspace(0, 10, num=5)
array([ 0. , 2.5, 5. , 7.5, 10. ])
```

- Make array of a range:

```
arange_arr = np.arange(8)
array([0, 1, 2, 3, 4, 5, 6, 7])
```

Concatenating arrays:

```
arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])

arr3 = np.concatenate((arr1, arr2))
```

```
array([1, 2, 3, 4, 5, 6])
```

Indexing arrays:

```
arr5 = np.array([[1,2,3],
                 [4,5,6],
                 [7,8,9]])
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

We can index for the element with value = 5 by using Python container-like indexing:

```
five = arr5[1,1]
```

Slicing:

We can obtain the entire second row by indexing:

```
second_row = arr5[1,:]
array([4, 5, 6])
```

* It's important to note that when you pull out a slice the output is an array.

```
arr_3d = np.array([[[1, 0, 3],
                    [4, 5, 6],
                    [7, 8, 9]],

                   [[1, 2, 3],
                    [4, 0, 6],
                    [7, 8, 9]],

                   [[1, 2, 3],
                    [4, 5, 6],
                    [7, 0, 9]]])
```

To obtain a matrix of middle columns, we utilize, array[pane, row, column]:

```
arr_3d[:,:,1]
array([[0, 5, 8],
       [2, 0, 8],
       [2, 5, 0]])
```

To obtain a matrix of the top rows:

```
arr_3d[:,0,:]
array([[1, 0, 3],
       [1, 2, 3],
       [1, 2, 3]])
```

NumPy Array Methods:

The methods for arrays primarily used are mathematical:

np.mean() finds the mean value of the array or object within

```
np.mean(arr_3d)
4.444444444444445
```

np.round() rounds the value of a number or elements w/in array or axis to a determined amount of spcaes

```
np.round(np.mean(arr_3d), 2)
4.44
```

np.sum() sums up the values of given elements w/in array or axis

```
np.sum(arr_3d)
120
```

NumPy Attributes: are intrinsic properties about the specific array and can be called on using methods

np.size() counts the number of elements within an array

```
np.size(arr_3d)
27
```

np.ndim() counts the number of dimensions of an array

```
np.ndim(arr_3d)
3
```

Important NumPy functions:

The most common NumPy functions that I've used are the np.mean(), np.sum(), and np.sqrt() functions as the calculations I've been running are more simplistic.