Lecture 03.3 Numpy

August 31, 2023

1 Numpy

Duncan Callaway

This notebook gives a quick tour of `numpy`. My objective is to give you a sense of how arrays in `numpy` work so that you can then contrast them to Pandas dataframes.

A numpy array is a grid of elements *of the same data type*. (Pandas is like numpy but handles different data types.) The beauty of numpy, like matlab, is that you don't need to specify what sort of data type you're working with – it will guess for you.

[1]:
```
import numpy as np
a = np.array([[1,2,3], [4,5,6]])
a
```

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

That's a numpy array.

1.0.1 Q: What have we seen before that looks a lot like this?

A: a nested list.

A big difference, though, is that we index numpy arrays differently.

[2]:
```
a[1,1]
```

[2]:
```
5
```

What is the first index entry calling? If you're not sure, play around with `a` to see if you can figure it out.

Answer: The row.

And so the second entry is calling the column. Note this is the opposite of the order we used when we indexed into the nested list.

If you're familiar with linear algebra indexing (column, row), then this makes sense.

What happens when we try to add a different data type to the array?

[3]:
```
a[1,1] = 'g'
a
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[3], line 1
----> 1 a[1,1] = 'g'
      2 a

ValueError: invalid literal for int() with base 10: 'g'
```

We can't make the assignment.

1.0.2 Q: What's a slice?

It's a portion of the array that we call with expressions involving a colon `:` in the index locations.

[4]:
```
a[0:1, 0:2]
```

[4]:
```
array([[1, 2]])
```

Perhaps you remember that the indexing is not *inclusive* meaning it does not include data associated with the end index. To get the last value you can just leave the position in the slice syntax empty:

[5]:
```
a
```

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

1.0.3 Q: What will `a[1:, 1:]` give?

[6]:
```
a[1:, 1:]
```

[6]:
```
array([[5, 6]])
```

You can also put a number after a *second* colon in the slice syntax. Negative numbers slice backwards.

[7]:
```
a[:,1::-1]
```

[7]:
```
array([[2, 1],
       [5, 4]])
```

Numbers other than 1 (or -1) after that second colon will pull data in steps, for example this skips the second row:

[8]:
```
a[:,0::2]
```

```
[8]: array([[1, 3],
            [4, 6]])
```

1.1 Finally, multiplication of numpy arrays is a useful operation

```
[9]: b = np.array([1,2])
     c = np.array([3,4])
```

```
[12]: b*c
```

```
[12]: array([3, 8])
```

For those that know linear algebra, this is an element-wise, or Hadamard, vector product.