week2

January 12, 2021

1 Week 2: Introduction to Numpy and Pandas

Numpy and Pandas are some of the most important data science libraries in python.

[56]:
```
import numpy as np
```

In numpy we will work with numpy arrays, which are similar to lists. Numpy arrays are optimized for data tasks and for effeciency. First, we initialize an array

[57]:
```
# We can initialize an array from a list
a = np.array(['First Element','Second Element','Third Element'])
print(a)
print(type(a))

lst = ['First Element','Second Element','Third Element']
aTest = np.array(lst)
print(a == aTest)
```

```
['First Element' 'Second Element' 'Third Element']
<class 'numpy.ndarray'>
[ True  True  True]
```

Numpy arrays can work similarly to lists. We can loop over them:

[58]:
```
for i in a:
    print(i)
```

```
First Element
Second Element
Third Element
```

And they are indexed similarly

[59]:
```
print(a[0])
print(a[1])
print(a[2])
print(a[3])
```

```
First Element
Second Element
```

```
Third Element


↪---------------------------------------------------------------------------

    IndexError                                Traceback (most recent call␣
↪last)

    <ipython-input-59-a9bb74723f5f> in <module>
      2 print(a[1])
      3 print(a[2])
----> 4 print(a[3])


    IndexError: index 3 is out of bounds for axis 0 with size 3
```

But they have some properties that make them easier to work with than lists for data science.

[ ]:
```python
# Multiplication works as we might like
first = np.array([1,2,3])
second = np.array([4,5,6])
print(3*first)
print(first*second)

# More generally, we can apply functions to the whole array
firstExp = np.exp(first)
print(firstExp)
reverse = np.log(firstExp)
print(reverse)
```

We can use numpy to tell us the shape of the array and to create multi-dimensional arrays.

[ ]:
```python
array2d = np.array([[1,2,3],[4,5,6]])
print(array2d)
print(array2d.shape)
```

As well as to create arrays of zeros and ones:

[ ]:
```python
zeros = np.zeros(2)
print(zeros)
zeros2d = np.zeros([5,4])
print(zeros2d)

ones = np.ones(13)
print(ones)
ones2d = np.ones([2,6])
print(ones2d)
```

We can treat these arrays just like vectors and/or matrices. For any two real vectors $u = (u_1, \ldots u_p)$ and $v = (v_1, \ldots, v_p)$, their dot product is given:

$$u \cdot v = \sum_{i=1}^{p} u_i v_i$$

So if $u = (1, -1, 1)$ and $v = (2, 5, 3)$, we would expect

$$u \cdot v = 2 - 5 + 3 = 0$$

(Note that if $u \cdot v = 0$, we say that u and v are *orthogonal*)

[ ]:
```
u = np.array([1,-1,1])
v = np.array([2,5,3])
print(np.dot(u,v))
```

We can also treat multi-dimensional arrays as matrices. For example, for a square $(p \times p)$ matrix, X, we may be interested in it's inverse (if it exits). The inverse of a matrix is a matrix X^{-1} satisfying

$$XX^{-1} = I_{p \times p} \tag{1}$$

The matrix inverse exists and is unique whenever the determinant of a matrix is 0, $det(X) = 0$.

If X is a 2×2 matrix,

$$X = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

then its determinant is given by

$$det(X) = \frac{1}{ad - bc} \tag{2}$$

and, whenever the determinant is not 0, we can calculate the matrix inverse:

$$X^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{3}$$

So that (as an example):

$$X = \begin{bmatrix} 4 & 7 \\ 2 & 6 \end{bmatrix} \implies X^{-1} = \frac{1}{4 \cdot 6 - 7 \cdot 2} \begin{bmatrix} 6 & -7 \\ -2 & 4 \end{bmatrix}$$
$$X^{-1} = \begin{bmatrix} 0.6 & -0.7 \\ -0.2 & 0.4 \end{bmatrix}$$

[ ]:
```
X = np.array([[4 , 7], [2 , 6]])
print(X)
```

[ ]:
```
Xinv = np.linalg.inv(X)
print(Xinv)

# Test that this returns the identity matrix
```

```
test = np.dot(X,Xinv)
print(np.round(test, 2))
```

This sort of matrix inversion is especially useful when we are computing inverses of large matrices, as we have to do when we are conducting linear regression or fitting other machine learning models.

1.1 Introduction to Pandas

Pandas builds off of numpy and is used to handle datasets. To find a specific dataset to use we are going to import one from the seaborn library.

Once we read in a dataframe, we can use the '.head()' method to take look at the first few observations and see the variable names/features.

[ ]:
```
import pandas as pd
import seaborn as sns
# pd.read_csv("~path/to/data")
iris = sns.load_dataset('iris')
iris.head()
```

The '.count()' method will give us the number of non-empty entries in each column.

[ ]:
```
print(iris.count())
print
```

Just like in numpy, the '.shape' attribute gives the shape of the data frame (observations, number of features):

[ ]:
```
print(iris.shape)
```

You can access specific columns by indexing the data frame by the feature name

[ ]:
```
sepal_length = iris['sepal_length']
sepal_width = iris['sepal_width']
print(sepal_length)
print(sepal_length.shape)
```

Once you have a specific column, I can find its maximum and minimum:

[51]:
```
print(sepal_length.max())
print(sepal_length.min())
print(sepal_length.idxmax())
print(sepal_length.idxmin())
```

```
7.9
4.3
131
13
```

We can also apply these to the whole data frame as well

[54]:
```
iris_numeric = iris[['sepal_length','sepal_width','petal_length','petal_width']]
print(iris_numeric.max())
print(iris_numeric.idxmax())
```

```
sepal_length    7.9
sepal_width     4.4
petal_length    6.9
petal_width     2.5
dtype: float64
sepal_length    131
sepal_width      15
petal_length    118
petal_width     100
dtype: int64
```

We can consolidate these commands using the '.describe()' method

[55]:
```
iris_numeric.describe()
```

[55]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

[ ]:
```

```