Numpy Library

The notebook presents a very brief introduction to the Numpy Library The Numpy Library is often used for the implementation of basic to complex Machine Learning Implementations

In [52]:
```
import numpy as np

a = np.random.randn(5)
```

the np.random.randn(5) command creates a data structure that is a random sample of 5 values from the uniform guassian distribution with the mean at zero and unifrom distribution from -1 till 1.

In [53]:
```
a
```

Out[53]:
```
array([0.73493653, 0.90059485, 0.11364175, 1.06141116, 0.75269699])
```

In [54]:
```
a.shape
```

Out[54]:
```
(5,)
```

This is called a rank one array. Keep in mind that this rank one array is neither a row vector or a column vector. Due to this if you try and transpose this array, you will get the same array back in exactly the same dimensions.

In [55]:
```
a.T
```

Out[55]:
```
array([0.73493653, 0.90059485, 0.11364175, 1.06141116, 0.75269699])
```

And if we find the product of array a with its transpose then rather than getting a matrix/array, we will be getting a number.

In [56]:
```
print(np.dot(a, a.T))
```

```
3.0572636447554045
```

Compared to this if we explicitly declare the array a to have a complete dimension then it is a very different story.

In [57]:
```
a = np.random.randn(5,1)
```

In [58]:
```
a
```

Out[58]:
```
array([[ 0.60353888],
       [ 1.43756302],
       [-1.2367828 ],
       [-0.0210852 ],
       [ 0.47765206]])
```

In [59]:
```
a.shape
```

Out[59]:
```
(5, 1)
```

In [60]:
```
a.T
```

Out[60]:
```
array([[ 0.60353888,  1.43756302, -1.2367828 , -0.0210852 ,  0.47765206]])
```

In [61]:
```
print(np.dot(a, a.T))
```

```
[[ 3.64259185e-01  8.67625179e-01 -7.46446512e-01 -1.27257364e-02
   2.88281591e-01]
 [ 8.67625179e-01  2.06658743e+00 -1.77795321e+00 -3.03112996e-02
   6.86654935e-01]
 [-7.46446512e-01 -1.77795321e+00  1.52963170e+00  2.60778092e-02
  -5.90751852e-01]
 [-1.27257364e-02 -3.03112996e-02  2.60778092e-02  4.44585538e-04
  -1.00713878e-02]
 [ 2.88281591e-01  6.86654935e-01 -5.90751852e-01 -1.00713878e-02
   2.28151490e-01]]
```

An important concept over here is to implement programming assertions to ensure that any vector coming in to your model is of the desired dimension. For example, for our implementation we need an a vector that needs to have the dimension (5, 1). to ensure that the code does not run into trouble if a different dimension of A is passed on an assertion can be made.

In [62]:
```
assert (a.shape == (5,1))
```

if the assertion is false, python will prompt an assertion error

In [63]:
```
assert (a.shape == (2,1))
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[63], line 1
----> 1 assert (a.shape == (2,1))

AssertionError:
```

Reshaping of Arrays

Another useful type of operation is reshaping of arrays. The most flexible way of doing this is with the reshape method. For example, if you want to put the numbers 1 through 9 in a 3×3

In [64]:
```
grid = np.arange(1, 10).reshape((3, 3))
print(grid)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Another common reshaping pattern is the conversion of a one-dimensional array into a two-dimensional row or column matrix. This can be done with the `reshape` method, or more easily done by making use of the `newaxis` keyword within a slice operation:

In [65]:
```
a = np.random.randn(5)
```

In [66]:
```
print(a.shape)
```

```
(5,)
```

In [67]:
```
a = a.reshape(5,1)
```

In [68]:
```
print(a.shape)
```

```
(5, 1)
```