In [2]:
```python
import numpy as np
```

DATA MANIPULATION

provides features to create random data

In [3]:
```python
a=np.random.randint(0,9,size=[10,10])
a
```

Out[3]:
```
array([[5, 5, 1, 6, 0, 7, 8, 7, 7, 1],
       [3, 8, 7, 7, 7, 8, 5, 3, 2, 3],
       [3, 7, 5, 3, 2, 1, 4, 6, 6, 4],
       [6, 3, 4, 7, 6, 4, 1, 8, 6, 2],
       [7, 1, 8, 2, 7, 5, 6, 0, 3, 7],
       [3, 1, 7, 4, 4, 6, 7, 7, 6, 3],
       [6, 4, 8, 8, 7, 6, 4, 5, 1, 7],
       [5, 1, 5, 0, 8, 5, 5, 1, 7, 0],
       [3, 0, 4, 5, 1, 8, 8, 7, 8, 0],
       [8, 5, 2, 0, 4, 1, 7, 7, 4, 8]])
```

In [18]:
```python
a.shape
```

Out[18]:
```
(10, 10)
```

reshaping data

In [4]:
```python
a.reshape(20,5)
```

Out[4]:
```
array([[5, 5, 1, 6, 0],
       [7, 8, 7, 7, 1],
       [3, 8, 7, 7, 7],
       [8, 5, 3, 2, 3],
       [3, 7, 5, 3, 2],
       [1, 4, 6, 6, 4],
       [6, 3, 4, 7, 6],
       [4, 1, 8, 6, 2],
       [7, 1, 8, 2, 7],
       [5, 6, 0, 3, 7],
       [3, 1, 7, 4, 4],
       [6, 7, 7, 6, 3],
       [6, 4, 8, 8, 7],
       [6, 4, 5, 1, 7],
       [5, 1, 5, 0, 8],
       [5, 5, 1, 7, 0],
       [3, 0, 4, 5, 1],
       [8, 8, 7, 8, 0],
       [8, 5, 2, 0, 4],
       [1, 7, 7, 4, 8]])
```

index slicing:provides required part's view from the array

In [16]:
```python
a[2:6,4:8]
```

Out[16]:
```
array([[2, 1, 4, 6],
       [6, 4, 1, 8],
       [7, 5, 6, 0],
       [4, 6, 7, 7]])
```

vectorized computation without any loops

In [5]:
```python
print('a+a: \n',a+a)
print('\na*a:\n',a*a)
```

```
a+a: 
 [[10 10  2 12  0 14 16 14 14  2]
 [ 6 16 14 14 14 16 10  6  4  6]
 [ 6 14 10  6  4  2  8 12 12  8]
 [12  6  8 14 12  8  2 16 12  4]
 [14  2 16  4 14 10 12  0  6 14]
 [ 6  2 14  8  8 12 14 14 12  6]
 [12  8 16 16 14 12  8 10  2 14]
 [10  2 10  0 16 10 10  2 14  0]
 [ 6  0  8 10  2 16 16 14 16  0]
 [16 10  4  0  8  2 14 14  8 16]]

a*a:
 [[25 25  1 36  0 49 64 49 49  1]
 [ 9 64 49 49 49 64 25  9  4  9]
 [ 9 49 25  9  4  1 16 36 36 16]
 [36  9 16 49 36 16  1 64 36  4]
 [49  1 64  4 49 25 36  0  9 49]
 [ 9  1 49 16 16 36 49 49 36  9]
 [36 16 64 64 49 36 16 25  1 49]
 [25  1 25  0 64 25 25  1 49  0]
 [ 9  0 16 25  1 64 64 49 64  0]
 [64 25  4  0 16  1 49 49 16 64]]
```

In [6]:
```python
a1=np.arange(1,11)
a2=np.arange(11,21)
print('a1:',a1)
print('a2:',a2)
print('a1+a2:',a1+a2)
print('a1*a2:',a1*a2)
print('a2-a1:',a2-a1)
print('a2/a1:',a2/a1)
```

```
a1: [ 1  2  3  4  5  6  7  8  9 10]
a2: [11 12 13 14 15 16 17 18 19 20]
a1+a2: [12 14 16 18 20 22 24 26 28 30]
a1*a2: [ 11  24  39  56  75  96 119 144 171 200]
a2-a1: [10 10 10 10 10 10 10 10 10 10]
a2/a1: [11.          6.          4.33333333  3.5         3.          2.66666667
  2.42857143  2.25        2.11111111  2.        ]
```

linear algerbra

In [7]:
```python
x = np.array([[1., 2., 3.], [4., 5., 6.]])
y = np.array([[6., 23.], [-1, 7], [8, 9]])
print('x:\n',x,'\ny:\n',y)
```

```
x:
 [[1. 2. 3.]
 [4. 5. 6.]] 
y:
 [[ 6. 23.]
 [-1.  7.]
 [ 8.  9.]]
```

In [8]:
```python
np.dot(x,y)
```

Out[8]:
```
array([[ 28.,  64.],
       [ 67., 181.]])
```

In [9]:
```python
x.dot(y)
```

Out[9]:
```
array([[ 28.,  64.],
       [ 67., 181.]])
```

In [10]:
```python
y.dot(x)
```

Out[10]:
```
array([[ 98., 127., 156.],
       [ 27.,  33.,  39.],
       [ 44.,  61.,  78.]])
```

DATA AGGREGATION

data aggregation(row wise,column wise,wholedata)

In [30]:
```python
print('row wise mean:',a.mean(axis=1))
print('column wise mean:',a.mean(axis=0))
print('row wise median:',np.median(a,axis=1))
print('column wise median:',np.median(a,axis=0))
print('row wise sum:',a.sum(axis=1))
print('column wise sum:',a.sum(axis=0))
print('whole mean:',a.mean())
print('whole sum:',a.sum())
print('whole median:',np.median(a))
print('row standard deviation:',np.std(a,axis=1))
print('row standard deviation:',np.std(a,axis=0))
print('whole standard deviation:',np.std(a))
```

```
row wise mean: [4.7 5.3 4.1 4.7 4.6 4.8 5.6 3.7 4.4 4.6]
column wise mean: [4.9 3.5 5.1 4.2 4.6 5.1 5.5 5.1 5.  3.5]
row wise median: [5.5 6.  4.  5.  5.5 5.  6.  5.  4.5 4.5]
column wise median: [5.  3.5 5.  4.5 5.  5.5 5.5 6.5 6.  3. ]
row wise sum: [47 53 41 47 46 48 56 37 44 46]
column wise sum: [49 35 51 42 46 51 55 51 50 35]
whole mean: 4.65
whole sum: 465
whole median: 5.0
row standard deviation: [2.79463772 2.23830293 1.81383571 2.14709106 2.72763634 1.98997487
 2.05912603 2.79463772 3.13687743 2.76405499]
row standard deviation: [1.75783958 2.61725047 2.3        2.74954542 2.69072481 2.38537209
 2.06155281 2.66270539 2.23606798 2.80178515]
whole standard deviation: 2.535251466817444
```

ANALYSING DATA

co-variance and correlation matrix of a given data

In [13]:
```python
np.cov(a)
```

Out[13]:
```
array([[ 8.67777778, -1.23333333,  0.81111111,  0.67777778, -5.02222222,
         1.93333333, -3.8       , -0.87777778,  7.46666667, -0.13333333],
       [-1.23333333,  5.56666667, -1.36666667, -1.01111111,  0.13333333,
        -0.82222222,  2.24444444, -0.01111111, -1.46666667, -4.64444444],
       [ 0.81111111, -1.36666667,  3.65555556, -0.07777778, -2.84444444,
        -0.2       , -2.17777778, -1.74444444, -0.48888889,  1.71111111],
       [ 0.67777778, -1.01111111, -0.07777778,  5.12222222, -2.8       ,
         0.48888889,  0.31111111,  0.23333333,  1.35555556, -1.57777778],
       [-5.02222222,  0.13333333, -2.84444444, -2.8       ,  8.26666667,
         0.46666667,  2.6       ,  4.2       , -2.26666667,  0.93333333],
       [ 1.93333333, -0.82222222, -0.2       ,  0.48888889,  0.46666667,
         4.4       , -0.53333333,  2.15555556,  5.64444444, -1.08888889],
       [-3.8       ,  2.24444444, -2.17777778,  0.31111111,  2.6       ,
        -0.53333333,  4.71111111, -1.68888889, -2.93333333, -1.84444444],
       [-0.87777778, -0.01111111, -1.74444444,  0.23333333,  4.2       ,
         2.15555556, -1.68888889,  8.67777778,  2.57777778, -0.91111111],
       [ 7.46666667, -1.46666667, -0.48888889,  1.35555556, -2.26666667,
         5.64444444, -2.93333333,  2.57777778, 10.93333333, -2.37777778],
       [-0.13333333, -4.64444444,  1.71111111, -1.57777778,  0.93333333,
        -1.08888889, -1.84444444, -0.91111111, -2.37777778,  8.48888889]])
```

In [14]:
```python
np.corrcoef(a)
```

Out[14]:
```
array([[ 1.        , -0.17745106,  0.14401224,  0.10166088, -0.59296144,
         0.3128788 , -0.59431633, -0.10115237,  0.76656009, -0.01553492],
       [-0.17745106,  1.        , -0.3029621 , -0.18935295,  0.01965513,
        -0.16613658,  0.43827791, -0.00159866, -0.18799986, -0.67563282],
       [ 0.14401224, -0.3029621 ,  1.        , -0.0179742 , -0.5174347 ,
        -0.04986858, -0.52477749, -0.30972495, -0.07733163,  0.30716809],
       [ 0.10166088, -0.18935295, -0.0179742 ,  1.        , -0.43029234,
         0.10298041,  0.06333221,  0.03499801,  0.18113894, -0.23927163],
       [-0.59296144,  0.01965513, -0.5174347 , -0.43029234,  1.        ,
         0.0773776 ,  0.41662609,  0.49588368, -0.2384219 ,  0.11141563],
       [ 0.3128788 , -0.16613658, -0.04986858,  0.10298041,  0.0773776 ,
         1.        , -0.11714148,  0.34884188,  0.81380166, -0.17816886],
       [-0.59431633,  0.43827791, -0.52477749,  0.06333221,  0.41662609,
        -0.11714148,  1.        , -0.26414059, -0.4087177 , -0.29166108],
       [-0.10115237, -0.00159866, -0.30972495,  0.03499801,  0.49588368,
         0.34884188, -0.26414059,  1.        ,  0.26464575, -0.10615529],
       [ 0.76656009, -0.18799986, -0.07733163,  0.18113894, -0.2384219 ,
         0.81380166, -0.4087177 ,  0.26464575,  1.        , -0.24681386],
       [-0.01553492, -0.67563282,  0.30716809, -0.23927163,  0.11141563,
        -0.17816886, -0.29166108, -0.10615529, -0.24681386,  1.        ]])
```

the functions cov(co-varaiance matrix) and corr(corrlation matrix) reduce the need for coding to calulation covariance or correlation

here the data contains 10 rows and 10 columns,consider a data having some 10,000 rows and some 1000 columns in genral format using for loops we need size of the data to iterate through the data and perform any required opertaion making it 3 to 5 lines of code this is where numpy enters the field completing the computation with a single line of code(vectorized computation)

APPLICATION IN DATA SCIENCE

due to is vast features and vast functions it reduces the effort to code for basic requirements(aggregaton(mean,median,sum,standard deviation), correlation coefficient calculation,data selction(indexing)).making the job easy to analyse the data irrespective of it's size.the machine learning models(pca,kmeans,regression,svm,etc) takes the input as an numpy ndarray.