Introduction to Numpy

June 10, 2017

1 Introduction to Numpy

```
In [2]: L = [str(c) for c in 'Unacdemy']
        L

Out[2]: ['U', 'n', 'a', 'c', 'd', 'e', 'm', 'y']

In [3]: index_ = L.index('c')
        L.insert(index_+1, 'a')
        L

Out[3]: ['U', 'n', 'a', 'c', 'a', 'd', 'e', 'm', 'y']

In [4]: List_ = ''.join(L)
        List_

Out[4]: 'Unacademy'
```

Python lists are dynamic, and can be heterogenous.

```
In [5]: L_dynamic = [True, '3', 'Kush', 1]
        type(L_dynamic)
        help(L_dynamic)

Help on list object:

class list(object)
 |  list() -> new empty list
 |  list(iterable) -> new list initialized from iterable's items
 |
 |  Methods defined here:
 |
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x
 |
```

```
 |  __delitem__(...)
 |      x.__delitem__(y) <==> del x[y]
 |
 |  __delslice__(...)
 |      x.__delslice__(i, j) <==> del x[i:j]
 |
 |      Use of negative indices is not supported.
 |
 |  __eq__(...)
 |      x.__eq__(y) <==> x==y
 |
 |  __ge__(...)
 |      x.__ge__(y) <==> x>=y
 |
 |  __getattribute__(...)
 |      x.__getattribute__('name') <==> x.name
 |
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |
 |  __getslice__(...)
 |      x.__getslice__(i, j) <==> x[i:j]
 |
 |      Use of negative indices is not supported.
 |
 |  __gt__(...)
 |      x.__gt__(y) <==> x>y
 |
 |  __iadd__(...)
 |      x.__iadd__(y) <==> x+=y
 |
 |  __imul__(...)
 |      x.__imul__(y) <==> x*=y
 |
 |  __init__(...)
 |      x.__init__(...) initializes x; see help(type(x)) for signature
 |
 |  __iter__(...)
 |      x.__iter__() <==> iter(x)
 |
 |  __le__(...)
 |      x.__le__(y) <==> x<=y
 |
 |  __len__(...)
 |      x.__len__() <==> len(x)
 |
 |  __lt__(...)
 |      x.__lt__(y) <==> x<y
```

```
 |
 |  __mul__(...)
 |      x.__mul__(n) <==> x*n
 |
 |  __ne__(...)
 |      x.__ne__(y) <==> x!=y
 |
 |  __repr__(...)
 |      x.__repr__() <==> repr(x)
 |
 |  __reversed__(...)
 |      L.__reversed__() -- return a reverse iterator over the list
 |
 |  __rmul__(...)
 |      x.__rmul__(n) <==> n*x
 |
 |  __setitem__(...)
 |      x.__setitem__(i, y) <==> x[i]=y
 |
 |  __setslice__(...)
 |      x.__setslice__(i, j, y) <==> x[i:j]=y
 |
 |      Use  of negative indices is not supported.
 |
 |  __sizeof__(...)
 |      L.__sizeof__() -- size of L in memory, in bytes
 |
 |  append(...)
 |      L.append(object) -- append object to end
 |
 |  count(...)
 |      L.count(value) -> integer -- return number of occurrences of value
 |
 |  extend(...)
 |      L.extend(iterable) -- extend list by appending elements from the iterable
 |
 |  index(...)
 |      L.index(value, [start, [stop]]) -> integer -- return first index of value.
 |      Raises ValueError if the value is not present.
 |
 |  insert(...)
 |      L.insert(index, object) -- insert object before index
 |
 |  pop(...)
 |      L.pop([index]) -> item -- remove and return item at index (default last).
 |      Raises IndexError if list is empty or index is out of range.
 |
 |  remove(...)
```

```
 |      L.remove(value) -- remove first occurrence of value.
 |      Raises ValueError if the value is not present.
 |
 |  reverse(...)
 |      L.reverse() -- reverse *IN PLACE*
 |
 |  sort(...)
 |      L.sort(cmp=None, key=None, reverse=False) -- stable sort *IN PLACE*;
 |      cmp(x, y) -> -1, 0, 1
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
 |
 |  __new__ = <built-in method __new__ of type object>
 |      T.__new__(S, ...) -> a new object with type S, a subtype of T
```

```
In [6]: [type(item) for item in L_dynamic]
```

```
Out[6]: [bool, str, str, int]
```

```
In [7]: import array
        L = list(range(10))
        A = array.array('i', L)
        A
```

```
Out[7]: array('i', [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Numpy helps us in efficient operations on Array. Numpy supports homogenous elements in it's array.

```
In [8]: import numpy as np
        arr1 = np.ones((5,5), dtype='float')
        arr2 = np.zeros((3,4), dtype='int')
```

```
In [9]: print("5 by 5 Matrix : (filled with ones) \n")
        print(arr1)

        print("\n3 by 4 Matrix : (filled with zeroes)\n")
        print(arr2)
```

```
5 by 5 Matrix : (filled with ones)

[[ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
```

```
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]]

3 by 4 Matrix : (filled with zeroes)

[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

```
In [10]: # Start from 0 to 20, jump of 2
         np.arange(0,20,2)

Out[10]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

Explicitly setting data type of the output array

```
In [11]: arr_ = np.array([1,2,3,4], dtype='float32')

In [12]: arr_

Out[12]: array([ 1.,  2.,  3.,  4.], dtype=float32)
```

Nested lists:

```
In [15]: arr_ = np.array(np.array([1,2,3,4]*2))

In [16]: arr_

Out[16]: array([1, 2, 3, 4, 1, 2, 3, 4])

In [19]: arr_ = np.array([range(i,i+3) for i in [1,2,3,4]])

In [20]: arr_

Out[20]: array([[1, 2, 3],
                [2, 3, 4],
                [3, 4, 5],
                [4, 5, 6]])

In [30]: # creating m by n matrix - filled with same number

         arr_ = np.full((3,5),12, dtype='int')
         arr_

Out[30]: array([[12, 12, 12, 12, 12],
                [12, 12, 12, 12, 12],
                [12, 12, 12, 12, 12]])

In [33]: # Array - evenly spaced
         arr_ = np.linspace(10, 20, 10)
         arr_
```

```
Out[33]: array([ 10.        ,  11.11111111,  12.22222222,  13.33333333,
                 14.44444444,  15.55555556,  16.66666667,  17.77777778,
                 18.88888889,  20.        ])
```

```
In [63]: np.random.seed(0) # Whenever code runs, keep it same

         X = np.random.randint(80,100, size=19)
         print("Standard Deviation: ", X.std())
         print("Mean: ", X.mean())
         print("Maximum: ", X.max())
         print("Minimum: ", X.min())
```

```
('Standard Deviation: ', 5.7308031952910312)
('Mean: ', 89.0)
('Maximum: ', 99)
('Minimum: ', 80)
```

```
In [65]: np.random.seed()


         X = np.random.randint(80,100, size=19)
         print("Standard Deviation: ", X.std())
         print("Mean: ", X.mean())
         print("Maximum: ", X.max())
         print("Minimum: ", X.min())
```

```
('Standard Deviation: ', 4.6506783230262148)
('Mean: ', 88.94736842105263)
('Maximum: ', 96)
('Minimum: ', 80)
```

```
In [66]: np.random.seed()


         X = np.random.randint(80,100, size=19)
         print("Standard Deviation: ", X.std())
         print("Mean: ", X.mean())
         print("Maximum: ", X.max())
         print("Minimum: ", X.min())
```

```
('Standard Deviation: ', 6.0524027416694945)
('Mean: ', 90.0)
('Maximum: ', 98)
('Minimum: ', 80)
```

```
In [67]: import pandas as pd
         data = pd.read_csv('president_heights.csv')
```

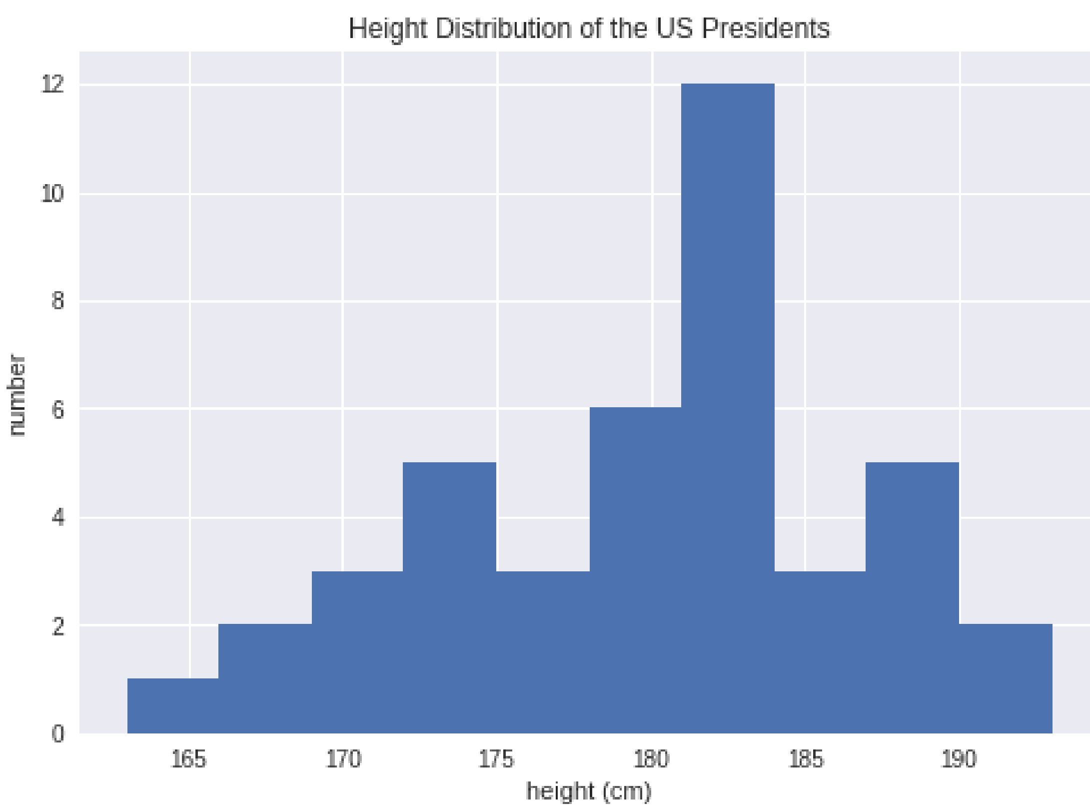
```
heights = np.array(data['height(cm)'])
print(heights)
```

```
[189 170 189 163 183 171 185 168 173 183 173 173 175 178 183 193 178 173
 174 183 183 168 170 178 182 180 183 178 182 188 175 179 183 193 182 183
 177 185 188 188 182 185]
```

```
In [68]: print("Standard Deviation: ", heights.std())
         print("Mean: ", heights.mean())
         print("Maximum: ", heights.max())
         print("Minimum: ", heights.min())
```

```
('Standard Deviation: ', 6.931843442745892l)
('Mean: ', 179.73809523809524)
('Maximum: ', 193)
('Minimum: ', 163)
```

```
In [74]: import seaborn; seaborn.set()
         import matplotlib.pyplot as plt
         plt.hist(heights)
         plt.title('Height Distribution of the US Presidents')
         plt.xlabel('height (cm)')
         plt.ylabel('number')
         plt.show()
```



Structured Data : Numpy's Structured Arrays
Using compound data types

```
In [14]: import numpy as np

         # Method 1 - Simple linear arrays (3)
         name = ['X', 'Y', 'Z']
         age = [25,19,21]
         weight = [55, 75, 70]
```

```
In [15]: # Method 2 - Compound Data Type
         data = np.zeros(3, dtype={'names':('name', 'age', 'weight'),'formats':('U1
         print(data.dtype)
```

```
[('name', '<U10'), ('age', '<i4'), ('weight', '<f8')]
```

Here `'U10'` translates to "Unicode string of maximum length 10", `'i4'` translates to integer of 4 byte (32 bits), `'f8'` translates to "8-byte float (64-bits)".
Empty container array - created, fill them now.

```
In [16]: data['name'] = name
         data['age'] = age
         data['weight'] = weight
         print(data)
```

```
[(u'X', 25, 55.0) (u'Y', 19, 75.0) (u'Z', 21, 70.0)]
```

```
In [17]: print(data[-1]['name'])
```

```
Z
```

```
In [18]: print(data[data['age'] < 30])
```

```
[(u'X', 25, 55.0) (u'Y', 19, 75.0) (u'Z', 21, 70.0)]
```

```
In [19]: print(data[data['weight'] > 70])
```

```
[(u'Y', 19, 75.0)]
```

Sorting using Numpy

In [9]:
```
# Selection sort : O(N^2)

import numpy as np

def selection_sort(x):
    for i in range(len(x)):
        # print("Minimum from %d is : %d", i, np.argmin(x[i:]))
        swap = i + np.argmin(x[i:])
        (x[i], x[swap]) = (x[swap], x[i])
    return x


x = np.array([5,1,10,9,2])

x

selection_sort(x)
```

In [14]:
```
# Bogo Sort : O(N * N!)
def bogo_sort(x):
    while np.any(x[:-1] > x[1:]):
        np.random.shuffle(x)
    return x
x = np.array([2,1,10,5,7])
bogo_sort(x)
```

Out[14]:
```
array([ 1,  2,  5,  7, 10])
```

** Fast Sorting in numpy : np.sort and np.argsort **

By default `np.sort` uses an $\mathcal{O}[N \log N]$, *quicksort* algorithm, though *mergesort* and *heapsort* are also available. For most applications, the default quicksort is more than sufficient.

To return a sorted version of the array without modifying the input, you can use `np.sort`:

In [15]:
```
x = np.array([5,1,9,2])
np.sort(x)
```

Out[15]:
```
array([1, 2, 5, 9])
```

In [18]:
```
i = np.argsort(x)
print(i)
```

```
[1 3 0 2]
```

** Sorting along rows and columns **

In [24]:
```
rand = np.random.RandomState(42)
# np.random.RandomState?
X = rand.randint(0,10,(4,6))
print(X)
```

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

```
In [28]: # Sort each column
         np.sort(X, axis = 0)
```

```
Out[28]: array([[2, 1, 4, 0, 1, 5],
                [5, 2, 5, 4, 3, 7],
                [6, 3, 7, 4, 6, 7],
                [7, 6, 7, 4, 9, 9]])
```

```
In [29]: # Sort each row
         np.sort(X, axis = 1)
```

```
Out[29]: array([[3, 4, 6, 6, 7, 9],
                [2, 3, 4, 6, 7, 7],
                [1, 2, 4, 5, 7, 7],
                [0, 1, 4, 5, 5, 9]])
```

** Partitioning **

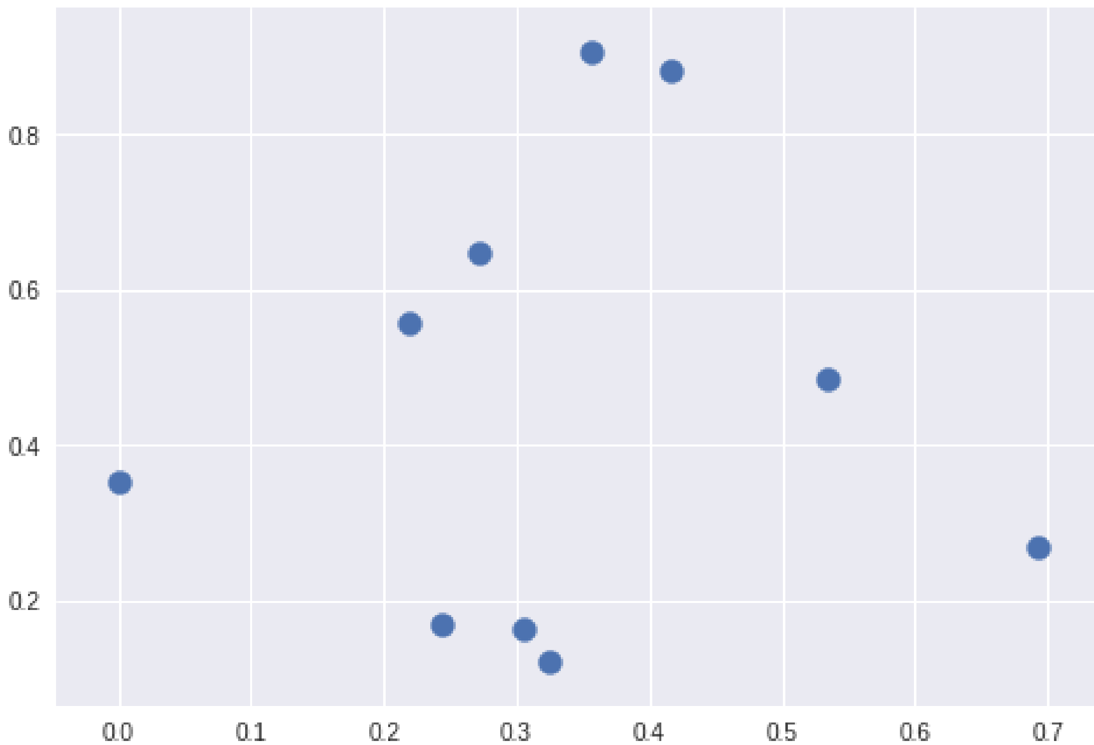
```
In [30]: X = np.array([7,2,3,1,6,5,4])
         np.partition(X, 3)
```

```
Out[30]: array([2, 1, 3, 4, 6, 5, 7])
```

2 Example : K-Nearest Neighbours

```
In [40]: X = rand.rand(10,2)
         print(X)
         import matplotlib.pyplot as plt
         import seaborn; seaborn.set() # Plot styling
         plt.scatter(X[:, 0], X[:, 1], s = 100)
         plt.show()
```

```
[[  4.16509948e-01   8.83280259e-01]
 [  3.24345021e-01   1.22087955e-01]
 [  3.56297838e-01   9.06828442e-01]
 [  2.72132249e-01   6.47690121e-01]
 [  5.20376995e-04   3.52568856e-01]
 [  3.04781258e-01   1.64655853e-01]
 [  5.34089419e-01   4.84829971e-01]
 [  6.92436033e-01   2.69412334e-01]
 [  2.44125522e-01   1.68291042e-01]
 [  2.18764220e-01   5.58102002e-01]]
```



In [39]:
```
dist_sq = np.sum((X[:, np.newaxis, :] - X[np.newaxis, :, :]) ** 2, axis =
```

In [41]:
```
# Breaking down in steps

# Step - 1

differences = X[:, np.newaxis, :] - X[np.newaxis, :, :]
differences.shape
```

Out[41]:
```
(10, 10, 2)
```

In [43]:
```
sq_dif = differences ** 2
sq_dif.shape
print(sq_dif)
```

```
[[[  0.00000000e+00   0.00000000e+00]
  [  8.49437374e-03   5.79413724e-01]
  [  3.62549817e-03   5.54516905e-04]
  [  2.08449198e-02   5.55027133e-02]
  [  1.73047323e-01   2.81654593e-01]
  [  1.24833001e-02   5.16421037e-01]
  [  1.38249321e-02   1.58762632e-01]
  [  7.61352044e-02   3.76833829e-01]
```

```
 [  2.97163902e-02   5.11209580e-01]
 [  3.91033731e-02   1.05740899e-01]]

[[  8.49437374e-03   5.79413724e-01]
 [  0.00000000e+00   0.00000000e+00]
 [  1.02098252e-03   6.15817632e-01]
 [  2.72617352e-03   2.76257637e-01]
 [  1.04862400e-01   5.31214460e-02]
 [  3.82740817e-04   1.81202598e-03]
 [  4.39927126e-02   1.31581771e-01]
 [  1.35490993e-01   2.17044727e-02]
 [  6.43516798e-03   2.13472529e-03]
 [  1.11473056e-02   1.90108249e-01]]

[[  3.62549817e-03   5.54516905e-04]
 [  1.02098252e-03   6.15817632e-01]
 [  0.00000000e+00   0.00000000e+00]
 [  7.08384632e-03   6.71526694e-02]
 [  1.26577602e-01   3.07203688e-01]
 [  2.65395801e-03   5.50820151e-01]
 [  3.16098464e-02   1.78082709e-01]
 [  1.12988886e-01   4.06299294e-01]
 [  1.25826284e-02   5.45437490e-01]
 [  1.89154962e-02   1.21610130e-01]]

[[  2.08449198e-02   5.55027133e-02]
 [  2.72617352e-03   2.76257637e-01]
 [  7.08384632e-03   6.71526694e-02]
 [  0.00000000e+00   0.00000000e+00]
 [  7.37730092e-02   8.70965606e-02]
 [  1.06595777e-03   2.33322103e-01]
 [  6.86215589e-02   2.65234282e-02]
 [  1.76655270e-01   1.43094084e-01]
 [  7.84376765e-04   2.29823476e-01]
 [  2.84814661e-03   8.02603098e-03]]

[[  1.73047323e-01   2.81654593e-01]
 [  1.04862400e-01   5.31214460e-02]
 [  1.26577602e-01   3.07203688e-01]
 [  7.37730092e-02   8.70965606e-02]
 [  0.00000000e+00   0.00000000e+00]
 [  9.25746838e-02   3.53112968e-02]
 [  2.84695923e-01   1.74930025e-02]
 [  4.78747275e-01   6.91500724e-03]
 [  5.93434668e-02   3.39583128e-02]
 [  4.76303748e-02   4.22438740e-02]]

[[  1.24833001e-02   5.16421037e-01]
```

```
 [  3.82740817e-04   1.81202598e-03]
 [  2.65395801e-03   5.50820151e-01]
 [  1.06595777e-03   2.33322103e-01]
 [  9.25746838e-02   3.53112968e-02]
 [  0.00000000e+00   0.00000000e+00]
 [  5.25822328e-02   1.02511466e-01]
 [  1.50276224e-01   1.09739202e-02]
 [  3.67911830e-03   1.32145993e-05]
 [  7.39893093e-03   1.54799872e-01]]

[[  1.38249321e-02   1.58762632e-01]
 [  4.39927126e-02   1.31581771e-01]
 [  3.16098464e-02   1.78082709e-01]
 [  6.86215589e-02   2.65234282e-02]
 [  2.84695923e-01   1.74930025e-02]
 [  5.25822328e-02   1.02511466e-01]
 [  0.00000000e+00   0.00000000e+00]
 [  2.50736500e-02   4.64047586e-02]
 [  8.40790616e-02   1.00196894e-01]
 [  9.94299816e-02   5.36879048e-03]]

[[  7.61352044e-02   3.76833829e-01]
 [  1.35490993e-01   2.17044727e-02]
 [  1.12988886e-01   4.06299294e-01]
 [  1.76655270e-01   1.43094084e-01]
 [  4.78747275e-01   6.91500724e-03]
 [  1.50276224e-01   1.09739202e-02]
 [  2.50736500e-02   4.64047586e-02]
 [  0.00000000e+00   0.00000000e+00]
 [  2.00982314e-01   1.02255156e-02]
 [  2.24364987e-01   8.33417245e-02]]

[[  2.97163902e-02   5.11209580e-01]
 [  6.43516798e-03   2.13472529e-03]
 [  1.25826284e-02   5.45437490e-01]
 [  7.84376765e-04   2.29823476e-01]
 [  5.93434668e-02   3.39583128e-02]
 [  3.67911830e-03   1.32145993e-05]
 [  8.40790616e-02   1.00196894e-01]
 [  2.00982314e-01   1.02255156e-02]
 [  0.00000000e+00   0.00000000e+00]
 [  6.43195673e-04   1.51952584e-01]]

[[  3.91033731e-02   1.05740899e-01]
 [  1.11473056e-02   1.90108249e-01]
 [  1.89154962e-02   1.21610130e-01]
 [  2.84814661e-03   8.02603098e-03]
 [  4.76303748e-02   4.22438740e-02]
```

```
 [  7.39893093e-03   1.54799872e-01]
 [  9.94299816e-02   5.36879048e-03]
 [  2.24364987e-01   8.33417245e-02]
 [  6.43195673e-04   1.51952584e-01]
 [  0.00000000e+00   0.00000000e+00]]]
```

```
In [46]: dist_sq = sq_dif.sum(-1)
         dist_sq.shape
         print(dist_sq)
```

```
[[ 0.          0.5879081   0.00418002  0.07634763  0.45470192  0.52890434
   0.17258756  0.45296903  0.54092597  0.14484427]
 [ 0.5879081   0.          0.61683861  0.27898381  0.15798385  0.00219477
   0.17557448  0.15719547  0.00856989  0.20125556]
 [ 0.00418002  0.61683861  0.          0.07423652  0.43378129  0.55347411
   0.20969256  0.51928818  0.55802012  0.14052563]
 [ 0.07634763  0.27898381  0.07423652  0.          0.16086957  0.23438806
   0.09514499  0.31974935  0.23060785  0.01087418]
 [ 0.45470192  0.15798385  0.43378129  0.16086957  0.          0.12788598
   0.30218893  0.48566228  0.09330178  0.08987425]
 [ 0.52890434  0.00219477  0.55347411  0.23438806  0.12788598  0.
   0.1550937   0.16125014  0.00369233  0.1621988 ]
 [ 0.17258756  0.17557448  0.20969256  0.09514499  0.30218893  0.1550937
   0.          0.07147841  0.18427596  0.10479877]
 [ 0.45296903  0.15719547  0.51928818  0.31974935  0.48566228  0.16125014
   0.07147841  0.          0.21120783  0.30770671]
 [ 0.54092597  0.00856989  0.55802012  0.23060785  0.09330178  0.00369233
   0.18427596  0.21120783  0.          0.15259578]
 [ 0.14484427  0.20125556  0.14052563  0.01087418  0.08987425  0.1621988
   0.10479877  0.30770671  0.15259578  0.        ]]
```

```
In [47]: dist_sq.diagonal()
```

```
Out[47]: array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

```
In [49]: nearest = np.argsort(dist_sq, axis=1) # Sorting each row
         print(nearest)
```

```
[[0 2 3 9 6 7 4 5 8 1]
 [1 5 8 7 4 6 9 3 0 2]
 [2 0 3 9 6 4 7 5 8 1]
 [3 9 2 0 6 4 8 5 1 7]
 [4 9 8 5 1 3 6 2 0 7]
 [5 1 8 4 6 7 9 3 0 2]
 [6 7 3 9 5 0 1 8 2 4]
 [7 6 1 5 8 9 3 0 4 2]
 [8 5 1 4 9 6 7 3 0 2]
```

```
 [9 3 4 6 2 0 8 5 1 7]]
```

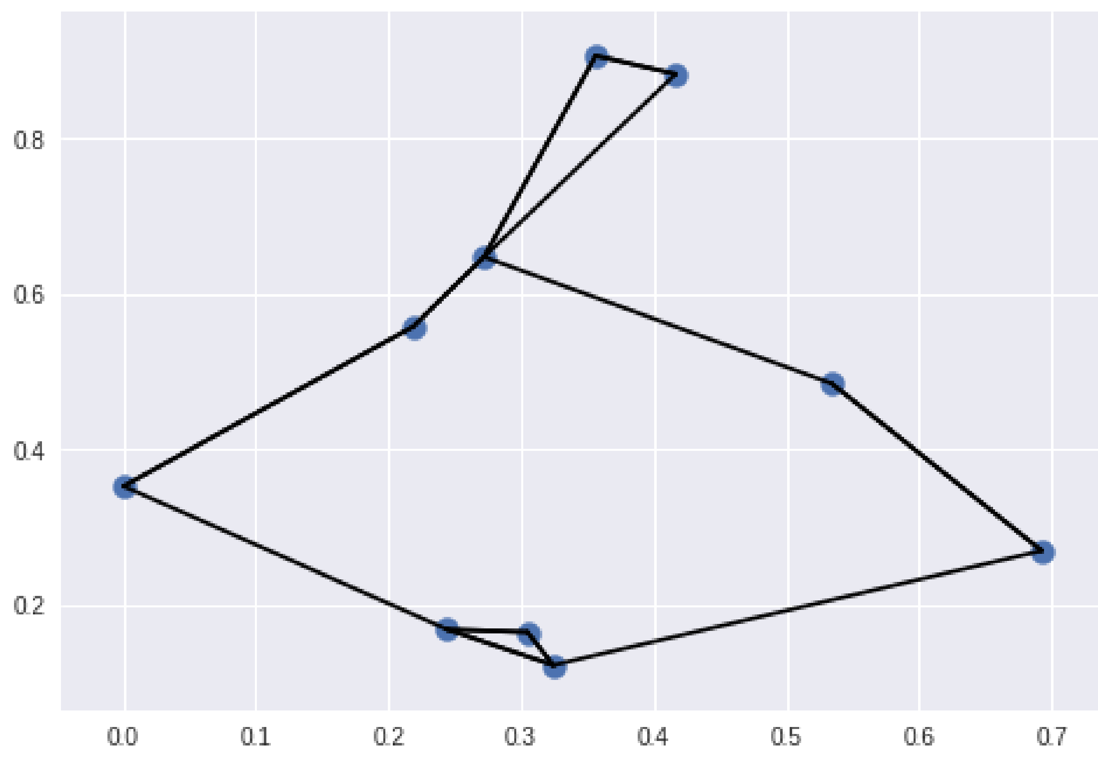
```
In [50]: # To get k-nearest neighbours, k + 1 left partitioned positions

         K = 2
         nearest_part = np.argpartition(dist_sq, K+1, axis = 1)
```

```
In [55]: plt.scatter(X[:, 0], X[:, 1], s=100)

         K = 2
         for i in range(X.shape[0]):
             for j in nearest_part[i, :K+1]:
                 # Plot from X[i] to X[j]
                 # Use some zip magic to make it happen.
                 plt.plot(*zip(X[j], X[i]), color = 'black')

         plt.show()
```




```
In [77]: # Revision - K-nearest numbers

         # Create 7 by 2 array
         X = rand.rand(7,2)
         X
```

```
Out[77]: array([[ 0.98342314,  0.39882444],
                [ 0.81643187,  0.79834512],
                [ 0.15071754,  0.50819878],
                [ 0.69581281,  0.8583588 ],
                [ 0.32595891,  0.22024105],
                [ 0.71114953,  0.80950105],
                [ 0.34866599,  0.09617655]])
```

```
In [78]: differences = X[:, np.newaxis, :] - X[np.newaxis, :, :]
         differences.shape
         differences

         sq_dif = differences ** 2
         sq_dif.shape

         dist_sq = sq_dif.sum(-1)
         dist_sq.shape

         dist_sq.diagonal() # To verify - dist. (x,x) = 0
```

```
Out[78]: array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

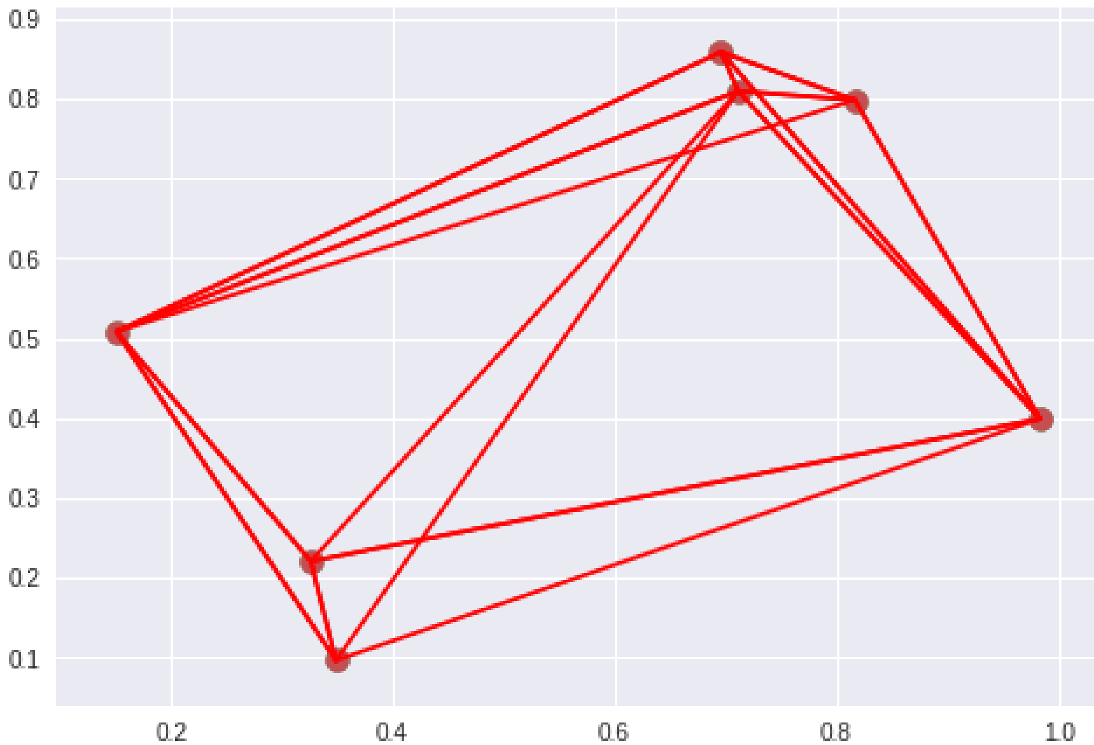
** Working on K - Nearest Neighbours ** (K = 2)

```
In [79]: K = 4
         nearest_parti = np.argpartition(dist_sq, K + 1, axis = 1)
         nearest_parti
```

```
Out[79]: array([[1, 0, 5, 3, 4, 6, 2],
                [1, 3, 5, 0, 2, 4, 6],
                [4, 6, 2, 5, 3, 1, 0],
                [1, 3, 5, 0, 2, 4, 6],
                [4, 6, 2, 0, 5, 3, 1],
                [1, 3, 5, 0, 2, 4, 6],
                [4, 6, 2, 0, 5, 3, 1]])
```

```
In [82]: plt.scatter(X[:, 0], X[:,1], s = 100)
         K = 4
         for i in range(X.shape[0]):
             for j in nearest_parti[i, :K+1]:
                 plt.plot(*zip(X[j], X[i]), color = 'red')

         plt.show()
```


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.2
0.4
0.6
0.8
1.0