# 데이터 분석(Python / R)

## 2일차 : 데이터 다루기(NumPy, Pandas)

삼성전자공과대학교 3학년 3학기

# Python example

```
import random

class Die(object):
    def __init__(self):
        self.sides = 6

    def set_sides(self, sides_change):
        if sides_change >= 4:
            if sides_change != 6:
                print("change sides from 6 to ", sides_change, " !")
            else:
                print("sides set to 6")
            self.sides = sides_change
        else:
            print("wrong sides! sides set to 6")

    def roll(self):
        return random.randint(1, self.sides)

if __name__ == '__main__':
    d = Die()
    d1 = Die()
    d.set_sides(4)
    d1.set_sides(4)
    print(d.roll(), d1.roll())
```

# Python types

- Numeric Types
  - integers
  - floats
  - complex numbers
- Booleans
- Sequence Types
  - list
  - tuple
- Text Sequence Types
  - string

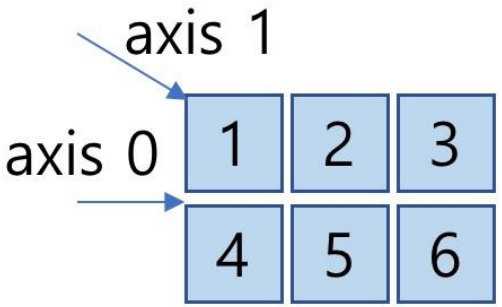
- Set Types
  - set
  - frozenset
- Mapping Types
  - dictionary

| Mutable | Immutable |
|---|---|
| list | tuple |
| set | frozenset |
| dictionary | string |
| byte array | bytes |

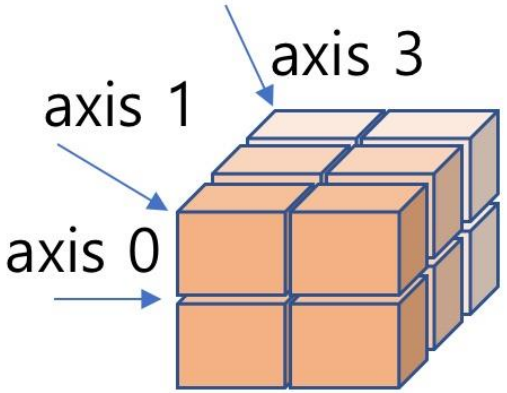
# Numpy

- ndarray : 같은 type을 가지는 데이터들의 N 차원 배열


1D array



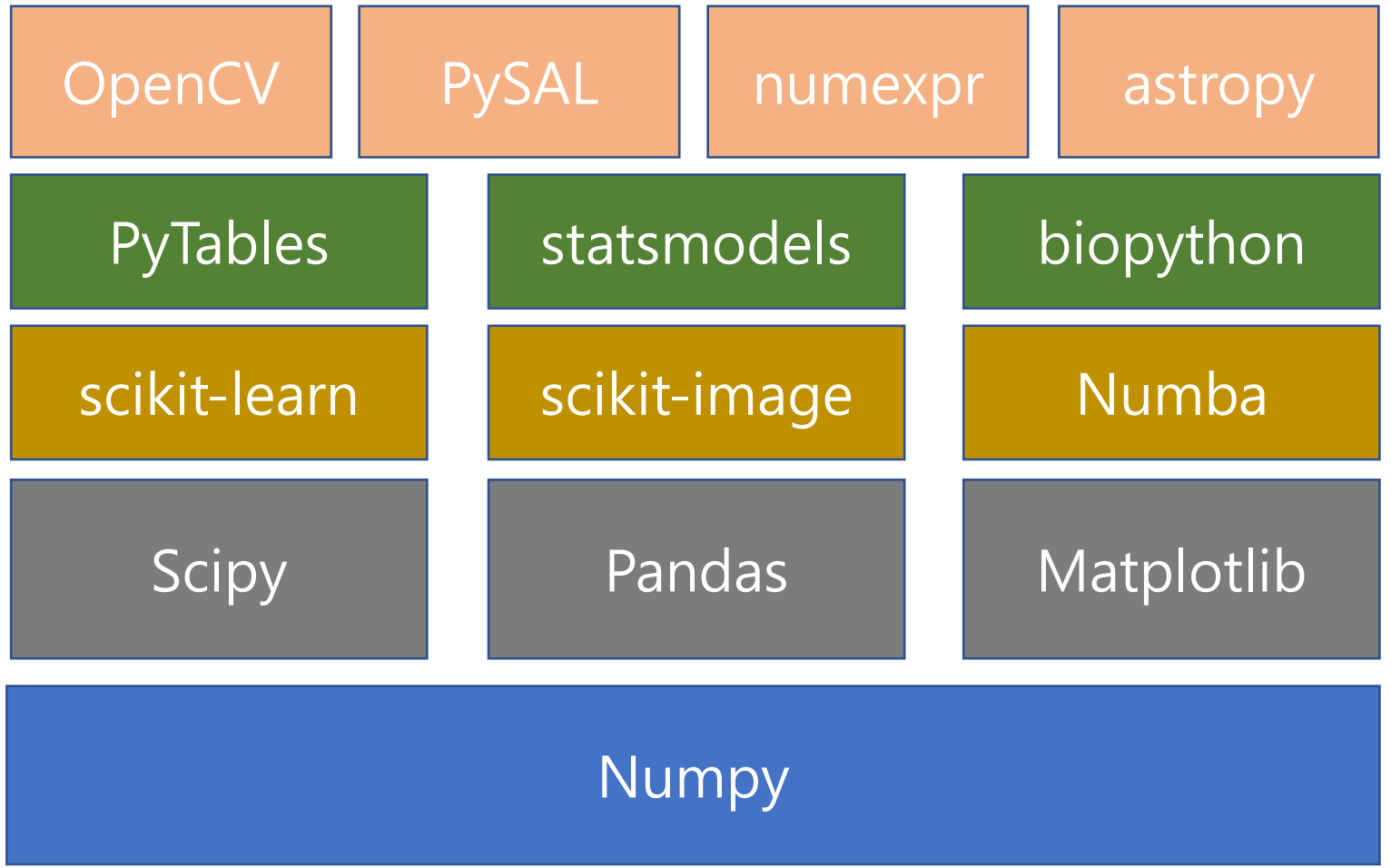

2D array




3D array

- Multi-dimensional arrays
- Built-in array operations
- Simplified, but powerful array interactions -〉 broadcasting
- Integration of other languages (Fortran, C, C++)

# Numpy – ecosystem



# Numpy – 기초

```
>>> import numpy as np
>>> a = np.array([1,2,3,4,5,6,7,8,9])
>>> a
array([1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> b = a.reshape((3,3))
>>> b
array([[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])

>>> b * 10 + 4
array([[14, 24, 34],
[44, 54, 64],
[74, 84, 94]])
```

- List

```
>>> a = [1,3,5,7,9]
>>> b = [3,5,6,7,9]
>>> a + b
[1, 3, 5, 7, 9, 3, 5, 6, 7, 9]
```

- numpy array

```
>>> a = np.array([1,3,5,7,9])
>>> b = np.array([3,5,6,7,9])
>>> a + b
array([ 4,  8, 11, 14, 18])
```

# Numpy - 생성

- Vector

```
# as vectors from lists
>>> a = np.array([1,3,5,7,9])
>>> b = np.array([3,5,6,7,9])
>>> c = a + b
>>> c
[4, 8, 11, 14, 18]

>>> type(c)
(<type 'numpy.ndarray'>)

>>> c.shape
(5,)
```

- One type

```
>>> M[0,0] = "hello"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for long()
with base 10: 'hello'
```

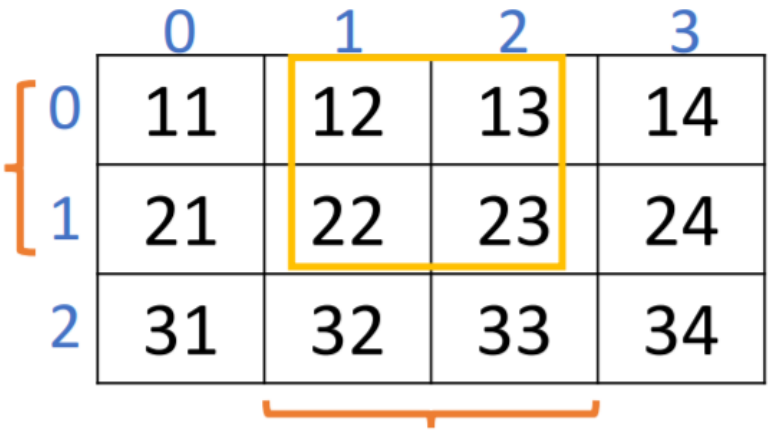
- Matrix

```
>>> l = [[1, 2, 3], [3, 6, 9], [2, 4, 6]]
>>> a = np.array(l)
>>> a
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> a.shape
(3, 3)
>>> a.dtype
int64

>>> M = np.array([[1, 2], [3, 4]])
>>> M.shape
(2,2)
>>> M.dtype
dtype('int64')
# 타입 지정
>>> M = numpy.array([[1, 2], [3, 4]], dtype=complex)
>>> M
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

# Numpy – slicing

an_array =




a_slice = an_array[:2, 1:3]

# Numpy – slicing

```
>>> print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> print(a[0])  # 리스트와 동일하게 동작
[1 2 3]
>>> print(a[1, 2])  # row, column
9
>>> print(a[1, 1:3])
[6 9]
>>> print(a[:,1])
[2 6 4]
>>> a[1, 2] = 7
>>> print(a)
[[1 2 3]
 [3 6 7]
 [2 4 6]]
>>> a[:, 0] = [0, 9, 8]
>>> print(a)
[[0 2 3]
 [9 6 7]
 [8 4 6]]
```

# Numpy - 생성 함수

```
>>> x = arange(0, 10, 1) # start, stop, step
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.linspace(0, 10, 25)
array([  0.        ,   0.41666667,   0.83333333,   1.25      ,
         1.66666667,   2.08333333,   2.5       ,   2.91666667,
         3.33333333,   3.75      ,   4.16666667,   4.58333333,
         5.        ,   5.41666667,   5.83333333,   6.25      ,
         6.66666667,   7.08333333,   7.5       ,   7.91666667,
         8.33333333,   8.75      ,   9.16666667,   9.58333333,  10.        ])

>>> np.logspace(0, 10, 10, base=numpy.e)
array([  1.00000000e+00,   3.03773178e+00,   9.22781435e+00,
         2.80316249e+01,   8.51525577e+01,   2.58670631e+02,
         7.85771994e+02,   2.38696456e+03,   7.25095809e+03,
         2.20264658e+04])
```

# Numpy - 생성 함수

```
# 대각 행렬
>>> np.diag([1,2,3])
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])

>>> b = np.zeros(5)
>>> print(b)
[ 0.  0.  0.  0.  0.]
>>> b.dtype
dtype('float64')

>>> n = 1000
>>> my_int_array = np.zeros(n, dtype=np.int)
>>> my_int_array.dtype
dtype('int32')

>>> c = numpy.ones((3,3))
>>> c
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

# Numpy - array사용

```
>>> d = numpy.arange(5)
>>> print(d)
[0 1 2 3 4]

>>> d[1] = 9.7
>>> print(d)  # 기존 type 보존
[0 9 2 3 4]

>>> print(d*0.4)  # 연산을 수행하면 새로운 array 생성(type은 결과에 맞게 변경)
[ 0.   3.6  0.8  1.2  1.6]

>>> d = np.arange(5, dtype=np.float)
>>> print(d)
[ 0.  1.  2.  3.  4.]

>>> np.arange(3, 7, 0.5)  # type
array([ 3. , 3.5, 4. , 4.5, 5. , 5.5, 6. , 6.5])
```

# Numpy - array사용

```
>>> np.mgrid[0:5, 0:5] # meshgrid 생성
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3],
       [4, 4, 4, 4, 4]])

array([[0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4]])

# random data
>>> mp.random.rand(5,5)
array([[ 0.51531133,  0.74085206,  0.99570623,  0.97064334,  0.5819413 ],
       [ 0.2105685 ,  0.86289893,  0.13404438,  0.77967281,  0.78480563],
       [ 0.62687607,  0.51112285,  0.18374991,  0.2582663 ,  0.58475672],
       [ 0.72768256,  0.08885194,  0.69519174,  0.16049876,  0.34557215],
       [ 0.93724333,  0.17407127,  0.1237831 ,  0.96840203,  0.52790012]])
```

# Numpy - 파일로 부터 읽어오기

```
>>> birth = np.genfromtxt('births.csv', delimiter=',', skip_header=1)
>>> birth
array([[1.96900e+03, 1.00000e+00, 1.00000e+00,         nan, 4.04600e+03],
       [1.96900e+03, 1.00000e+00, 1.00000e+00,         nan, 4.44000e+03],
       [1.96900e+03, 1.00000e+00, 2.00000e+00,         nan, 4.45400e+03],
       ...,
       [2.00800e+03, 1.10000e+01,         nan,         nan, 1.65468e+05],
       [2.00800e+03, 1.20000e+01,         nan,         nan, 1.73215e+05],
       [2.00800e+03, 1.20000e+01,         nan,         nan, 1.81235e+05]])

>>> np.savetxt('births.txt', birth)  # 텍스트 포멧으로 저장
>>> np.save('saved-matrix.npy', birth) # numpy 포멧으로 저장
>>> np.load('saved-matrix.npy')
array([[1.96900e+03, 1.00000e+00, 1.00000e+00,         nan, 4.04600e+03],
       [1.96900e+03, 1.00000e+00, 1.00000e+00,         nan, 4.44000e+03],
       [1.96900e+03, 1.00000e+00, 2.00000e+00,         nan, 4.45400e+03],
       ...,
       [2.00800e+03, 1.10000e+01,         nan,         nan, 1.65468e+05],
       [2.00800e+03, 1.20000e+01,         nan,         nan, 1.73215e+05],
       [2.00800e+03, 1.20000e+01,         nan,         nan, 1.81235e+05]])
```

# Numpy – ndarray

- NumPy's main object is the homogeneous multidimensional array called `ndarray`.
  - This is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. Typical examples of multidimensional arrays include vectors, matrices, images and spreadsheets.
  - Dimensions usually called axes, number of axes is the rank

| | |
|---|---|
| [7, 5, -1] | An array of rank 1 i.e. It has 1 axis of length 3 |
| [ [ 1.5, 0.2, -3.7] ,<br>[ 0.1, 1.7, 2.9] ] | An array of rank 2 i.e. It has 2 axes, the first length 3, the second of length 3 (a matrix with 2 rows and 3 columns |

# Numpy – ndarray attributes

❖ **ndarray.ndim**

- 차원(dimensions) 수

❖ **ndarray.shape**

- row, column의 개수를 tuple 로 반환, (# of rows, # of columns)

❖ **ndarray.size**

- 요소의 개수, # of rows x # of columns

❖ **ndarray.dtype**

- type, 예) bool_, character, int_, int8, int16, int32, int64, float_, float8, float16, float32, float64, complex_, complex64, object_.

❖ **ndarray.itemsize ( ndarray.dtype.itemsize)**

- 요소의 byte 크기, 예) float64 - 8 (=64/8), complex32 - 4

❖ **ndarray.data**

- 요소를 담고 있는 buffer(memory)

# Numpy - 복사

```
>>> x = np.array([1,2,3,4])
>>> y = x
>>> x is y
True
>>> id(x), id(y)
(139814289111920, 139814289111920)
>>> x[0] = 9
>>> y
array([9, 2, 3, 4])

>>> x[0] = 1
>>> z = x[:]
>>> x is z
False
>>> id(x), id(z)
(139814289111920, 139814289112080)
>>> x[0] = 8
>>> z
array([8, 2, 3, 4])
```

```
>>> x = np.array([1,2,3,4])
>>> y = x.copy()
>>> x is y
False
>>> id(x), id(y)
(139814289111920, 139814289111840)
>>> x[0] = 9
>>> x
array([9, 2, 3, 4])
>>> y
array([1, 2, 3, 4])
```

```
a = [1, 2, 3]
b = a[:]
a[0] = 100
b
```

결과는?

# Numpy - 연산자

```
>>> a = np.arange(4.0)
>>> b = a * 23.4
>>> c = b/(a+1)
>>> c += 10
>>> print c
[ 10.   21.7  25.6  27.55]

>>> arr = np.arange(100, 200)
>>> select = [5, 25, 50, 75, -5]
>>> print(arr[select])  # index로 가져오기 - integer lists
[105, 125, 150, 175, 195]

>>> arr = np.arange(10, 20)
>>> div_by_3 = arr%3 == 0
>>> print(div_by_3)
[ False False  True False False  True False False  True False]
>>> print(arr[div_by_3])  # boolean lists
[12 15 18]

>>> arr = numpy.arange(10, 20).reshape((2,5))
[[10 11 12 13 14]
 [15 16 17 18 19]]
```

# Numpy - 연산 함수

```
>>> arr.sum()
145
>>> arr.mean()
14.5
>>> arr.std()
2.8722813232690143
>>> arr.max()
19
>>> arr.min()
10
>>> div_by_3.all()
False
>>> div_by_3.any()
True
>>> div_by_3.sum()
3
>>> div_by_3.nonzero()
(array([2, 5, 8]),)
```

❖ 대부분의 array method는 동일한 함수를 가지고 있음

```
>>> arr.sum()
45
>>> np.sum(arr)
45
```

# Numpy - 정렬

```
>>> arr = np.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])
>>> arr.sort() # array 자체 정렬
>>> print(arr)
[ 1.2  1.8  2.3  4.5  5.5  6.7]

>>> x = np.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])
>>> np.sort(x) # 정렬된 array 반환
array([ 1.2,  1.8,  2.3,  4.5,  5.5,  6.7])

>>> print(x)
[ 4.5  2.3  6.7  1.2  1.8  5.5]

>>> s = x.argsort() # 정렬된 index 반환
>>> s
array([3, 4, 1, 0, 5, 2])
>>> x[s]
array([ 1.2,  1.8,  2.3,  4.5,  5.5,  6.7])
```

# Numpy - 배열 연산

```
>>> a = np.array([[1.0, 2.0], [4.0, 3.0]])
>>> print(a)
[[ 1. 2.]
 [ 3. 4.]]

>>> a.transpose() # 전치 행렬
array([[ 1., 3.],
       [ 2., 4.]])

>>> u = np.eye(2) # 대각 행렬

>>> u
array([[ 1., 0.],
       [ 0., 1.]])

>>> j = np.array([[0.0, -1.0], [1.0, 0.0]])

>>> np.dot (j, j) # matrix product
array([[-1., 0.],
       [ 0., -1.]])
```

# Numpy - 기초통계

❖ 평균, 분산, 표준편차 등의 기초 통계

```
>>> a = np.array([1, 4, 3, 8, 9, 2, 3], float)
>>> np.median(a)
3.0
```

❖ 상관 계수

```
>>> a = np.array([[1, 2, 1, 3], [5, 3, 1, 8]], float)
>>> c = np.corrcoef(a)
>>> c
array([[ 1.        ,  0.72870505],
       [ 0.72870505,  1.        ]])
```

❖ 공분산 행렬

```
>>> np.cov(a)
array([[ 0.91666667,  2.08333333],
       [ 2.08333333,  8.91666667]])
```

# Numpy – arrays, matrices

```
>>> m = np.mat([[1,2],[3,4]])

>>> a = np.array([[1,2],[3,4]])
>>> m = np.mat(a)

>>> a = np.array([[1,2],[3,4]])
>>> m = np.asmatrix(a)

>>> a[0, 0] = 100
>>> print(a)
[[100 2]
 [3 4]]

>>> print(m)
[[100 2]
 [3 4]]
```

# Numpy - matrices

```
>>> a = array([[1,2],[3,4]])
>>> m = mat(a)
>>> m = matrix([[1, 2], [3, 4]])
>>> a[0]
array([1, 2])
>>> m[0]
matrix([[1, 2]])
>>> a*a
array([[ 1, 4], [ 9, 16]])
>>> m*m           # np.dot
matrix([[ 7, 10], [15, 22]])
>>> a**3
array([[ 1, 8], [27, 64]])
>>> m**3
matrix([[ 37, 54], [ 81, 118]])
>>> m.T
matrix([[1, 3], [2, 4]])
>>> m.H
matrix([[1, 3], [2, 4]])
>>> m.I
matrix([[-2. , 1. ], [ 1.5, -0.5]])
```

# Numpy - array 수학 연산

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([5,2,6], float)
>>> a + b
array([6., 4., 9.])
>>> a - b
array([-4., 0., -3.])
>>> a * b
array([5., 4., 18.])
>>> b / a
array([5., 1., 2.])
>>> a % b
array([1., 0., 3.])
>>> b**a
array([5., 4., 216.])

>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> b = np.array([-1, 3], float)
>>> a + b
array([[ 0.,  5.],
       [ 2.,  7.],
       [ 4.,  9.]])
```

```
>>> a = np.array([[1, 2], [3, 4]
, [5, 6]], float)
>>> b = np.array([-1, 3], float)

>>> a * a
array([[  1.,   4.],
       [  9.,  16.],
       [ 25.,  36.]])

>>> b * b
array([ 1.,  9.])

>>> a * b
array([[ -1.,   6.],
       [ -3.,  12.],
       [ -5.,  18.]])
```

# Numpy - array 수학 연산

```
>>> A = np.array([[n+m*10 for n in range(5)] for m in range(5)])
>>> v1 = arange(0, 5)
>>> A
array([[ 0, 1, 2, 3, 4],
[10, 11, 12, 13, 14],
[20, 21, 22, 23, 24],
[30, 31, 32, 33, 34],
[40, 41, 42, 43, 44]])
>>> v1
array([0, 1, 2, 3, 4])
>>> np.dot(A,A)
array([[ 300,  310,  320,  330,  340],
       [1300, 1360, 1420, 1480, 1540],
       [2300, 2410, 2520, 2630, 2740],
       [3300, 3460, 3620, 3780, 3940],
       [4300, 4510, 4720, 4930, 5140]])
>>>
>>> np.dot(A,v1)
array([ 30, 130, 230, 330, 430])
>>> np.dot(v1,v1)
30
```

# Numpy - array 수학 연산

```
>>> M = np.matrix(A)
>>> v = np.matrix(v1).T
>>> v
matrix([[0],
        [1],
        [2],
        [3],
        [4]])
>>> M*v
matrix([[ 30],
        [130],
        [230],
        [330],
        [430]])
>>> v.T * v   # inner product
matrix([[30]])
# standard matrix algebra applies
>>> v + M*v
matrix([[ 30],
        [131],
        [232],
        [333],
        [434]])
```

# Pandas – 데이터 가져오기

```
#Read csv file
df = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/Salaries.csv ")
```

```
pd.read_excel('myfile.xlsx',sheet_name='Sheet1)
pd.read_stata('myfile.dta')
pd.read_sas('myfile.sas7bdat')
pd.read_hdf('myfile.h5','df')
```

https://pandas.pydata.org/pandas-docs/stable/reference/io.html

# Pandas – Data Frame data types

| Pandas Type | Native Python Type | Description |
|---|---|---|
| object | string | 컬럼 데이터가 숫자와 문자가 혼합된 데이터의 경우 object로 자동 매핑 |
| int64 | int | 정수(64는 메모리의 크기) |
| float64 | float | 부동 소수점 숫자, 컬럼 데이터가 숫자와 NaN으로 구성되어 있으면 float64로 매핑 |
| datetime64, timedelta [ns] | N/A | 날짜, 시간 |

# Pandas – Data Frame data types

```
In [4]:  #Check a particular column type
         df['salary'].dtype
```

```
Out[4]: dtype('int64')
```

```
In [5]:  #Check types for all the columns
         df.dtypes
```

```
Out[4]: rank          object
        discipline    object
        phd           int64
        service       int64
        sex           object
        salary        int64
        dtype: object
```

# Pandas – Data Frames attributes

| 속성 | 설명 |
|---|---|
| dtypes | 컬럼들의 type list 반환 |
| columns | 컬럼들의 이름 list 반환 |
| axes | 로우, 컬럼 정보 list 반환 |
| ndim | 차원 정보 반환 |
| size | 요소들의 개수 반환 |
| shape | 차원수에 대한 tuple 반환 |
| values | Data들을 numpy array로 반환 |

# Pandas – Data Frames methods

| 함수 | 설명 |
|---|---|
| head( [n] ), tail( [n] ) | 처음/마지막 n개의 행 |
| describe() | 숫자 데이터에 대한 기술 통계량 산출 |
| max(), min() | 모든 숫자 컬럼에 대한 최대/최소 값 반환 |
| mean(), median() | 모든 숫자 컬럼에 대한 평균/중앙 값 반환 |
| std() | 모든 숫자 컬럼에 대한 표준편차 반환 |
| sample([n]) | 임의의 샘플 데이터 반환 |
| dropna() | NA 값이 들어 있는 모든 행 삭제 |

# Data Frames *groupby* method

- 데이터를 특정 조건에 의해 몇 개의 그룹으로 나눔
- 각 그룹에 대한 통계량 계산

In [ ]:
```
#Group data using rank
df_rank = df.groupby(['rank'])
```

In [ ]:
```
#Calculate mean value for each numeric column per each group
df_rank.mean()
```

| rank | phd | service | salary |
|---|---|---|---|
| AssocProf | 15.076923 | 11.307692 | 91786.230769 |
| AsstProf | 5.052632 | 2.210526 | 81362.789474 |
| Prof | 27.065217 | 21.413043 | 123624.804348 |

In [ ]:
```
#Calculate mean salary for each professor rank:
df.groupby('rank')[['salary']].mean()
```

| rank | salary |
|---|---|
| AssocProf | 91786.230769 |
| AsstProf | 81362.789474 |
| Prof | 123624.804348 |

# Pandas – Data Frames *groupby* method

동작 방식:

- Lazy evaluation : groupby를 실행하면 grouping 조건에 대한 적합성 여부만 판단하고 실제 grouping은 필요한 시점에 이루어 짐
- grouping 동작이 일어나면 key에 대한 sorting 이 수행됨, 실행 속도를 향상 하려면 “sort=False” 파라메터를 지정

In [ ]:
```
#Calculate mean salary for each professor rank:
df.groupby(['rank'], sort=False)[['salary']].mean()
```

# Pandas – Data Frame: filtering

```
In [ ]: #Calculate mean salary for each professor rank:
        df_sub = df[df['salary'] > 120000]
```

- 〉 greater
- 〈 less
- == equal
- 〉= greater or equal;
- 〈= less or equal;
- != not equal;

```
In [ ]: #Select only those rows that contain female professors:
        df_f = df[df['sex'] == 'Female']
```

# Pandas – Data Frames: Slicing

In [ ]:
```
#Select column salary:
df['salary']
```

Series

In [ ]:
```
#Select column salary:
df[['rank','salary']]
```

DataFrame

In [ ]:
```
#Select rows by their position:
df[10:20]
```

In [ ]:
```
#Select rows by their labels:
df.loc[10:20,['rank','sex','salary']]
```

Out[ ]:

| | rank | sex | salary |
|---|---|---|---|
| **10** | Prof | Male | 128250 |
| **11** | Prof | Male | 134778 |
| **13** | Prof | Male | 162200 |
| **14** | Prof | Male | 153750 |
| **15** | Prof | Male | 150480 |
| **19** | Prof | Male | 150500 |

# Pandas – Data Frames: Sorting

In [ ]:
```
# Create a new data frame from the original sorted by the column Salary
df_sorted = df.sort_values( by ='service')
df_sorted.head()
```

Out[ ]:

| | rank | discipline | phd | service | sex | salary |
|---|---|---|---|---|---|---|
| **55** | AsstProf | A | 2 | 0 | Female | 72500 |
| **23** | AsstProf | A | 2 | 0 | Male | 85000 |
| **43** | AsstProf | B | 5 | 0 | Female | 77000 |
| **17** | AsstProf | B | 4 | 0 | Male | 92000 |
| **12** | AsstProf | B | 1 | 0 | Male | 88000 |

In [ ]:
```
df_sorted = df.sort_values( by =['service', 'salary'], ascending = [True, False])
df_sorted.head(10)
```

Out[ ]:

| | rank | discipline | phd | service | sex | salary |
|---|---|---|---|---|---|---|
| **52** | Prof | A | 12 | 0 | Female | 105000 |
| **17** | AsstProf | B | 4 | 0 | Male | 92000 |
| **12** | AsstProf | B | 1 | 0 | Male | 88000 |
| **23** | AsstProf | A | 2 | 0 | Male | 85000 |
| **43** | AsstProf | B | 5 | 0 | Female | 77000 |
| **55** | AsstProf | A | 2 | 0 | Female | 72500 |
| **57** | AsstProf | A | 3 | 1 | Female | 72500 |
| **28** | AsstProf | B | 7 | 2 | Male | 91300 |
| **42** | AsstProf | B | 4 | 2 | Female | 80225 |
| **68** | AsstProf | A | 4 | 2 | Female | 77500 |

# Pandas – Missing Values

In [ ]:
```
# Read a dataset with missing values
flights = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/flights.csv")
```

In [ ]:
```
# Select the rows that have at least one missing value
flights[flights.isnull().any(axis=1)].head()
```

Out[ ]:

| | year | month | day | dep_time | dep_delay | arr_time | arr_delay | carrier | tailnum | flight | origin | dest | air_time | distance | hour | minute |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **330** | 2013 | 1 | 1 | 1807.0 | 29.0 | 2251.0 | NaN | UA | N31412 | 1228 | EWR | SAN | NaN | 2425 | 18.0 | 7.0 |
| **403** | 2013 | 1 | 1 | NaN | NaN | NaN | NaN | AA | N3EHAA | 791 | LGA | DFW | NaN | 1389 | NaN | NaN |
| **404** | 2013 | 1 | 1 | NaN | NaN | NaN | NaN | AA | N3EVAA | 1925 | LGA | MIA | NaN | 1096 | NaN | NaN |
| **855** | 2013 | 1 | 2 | 2145.0 | 16.0 | NaN | NaN | UA | N12221 | 1299 | EWR | RSW | NaN | 1068 | 21.0 | 45.0 |
| **858** | 2013 | 1 | 2 | NaN | NaN | NaN | NaN | AA | NaN | 133 | JFK | LAX | NaN | 2475 | NaN | NaN |

# Pandas – Missing Values

| 함수 | 설명 |
|---|---|
| dropna() | NA 값이 하나라도 들어 있는 모든 행 삭제 |
| dropna(how='all') | 모든 값이 NA인 행 삭제 |
| dropna(axis=1, how='all') | 모든 값이 NA인 열 삭제 |
| dropna(thresh = 5) | NA값이 5개 이하로 들어 있는 행 삭제 |
| fillna(0) | NA 값을 0으로 채움 |
| isnull() | 값이 NA이면 True 반환 |
| notnull() | 값이 NA가 아니면 True 반환 |

# Pandas – Missing Values

pd.read_xxx :

- na_values : NA/NaN으로 인식할 값 지정
- keep_default_na : 데이터를 파싱할 때 기본 NA 값을 포함할지 여부
- na_filter : NA 파싱 여부

| na_values | keep_default_na | 동작 |
|---|---|---|
| not None | True | 지정된 na_values 값이 기본 NA 값과 함께 NA로 인식 |
| None | True | 기본 NA 값만 NA로 인식 |
| not None | False | 기본 NA 값만 NA로 인식 |
| None | False | 모든 값이 NA로 인식 안됨 |

❖ na_filter 가 False이면 na_values, keep_default_na 값이 무시됨

# Pandas – Missing Values

- 합계를 구할 때는 missing value는 0으로 취급
- 모든 값이 missing value이면 합계는 NaN
- cumsum() , cumprod() 실행할 때, missing values는 무시되지만, 결과에는 남아 있음
- GroupBy를 실행할 때 missing values는 제외 됨
- 대부분의 기술 통계 함수는 "*skipna*" 파라메터를 가지고 있어서 missing value를 제외할 지 여부를 결정. 기본값은 True

# Pandas – Aggregation Functions in Pandas

Aggregation - 각 그룹에 대한 요약 통계

- compute group sums or means
- compute group sizes/counts

aggregation functions:

- min, max
- count, sum, prod
- mean, median, mode, mad
- std, var

In [ ]:
```
flights[['dep_delay','arr_delay']].agg(['min','mean','max'])
```

Out[ ]:

| | dep_delay | arr_delay |
|---|---|---|
| min | -16.000000 | -62.000000 |
| mean | 9.384302 | 2.298675 |
| max | 351.000000 | 389.000000 |

# Pandas – Basic Descriptive Statistics

| 함수 | 설명 |
|---|---|
| describe | 기본 통계량 (count, mean, std, min, quantiles, max) |
| min, max | 값의 최소 / 최대값 |
| mean, median, mode | 평균, 중앙값, 최빈값 |
| var, std | 분산, 표준편차 |
| sem | 편향되지 않은 표준 오류 |
| skew | 편향되지 않은 비대칭도(왜도) |
| kurt | 첨도 |



43