라이브러리 임포트

import numpy

A=numpy.array([1,2])

import numpy as np

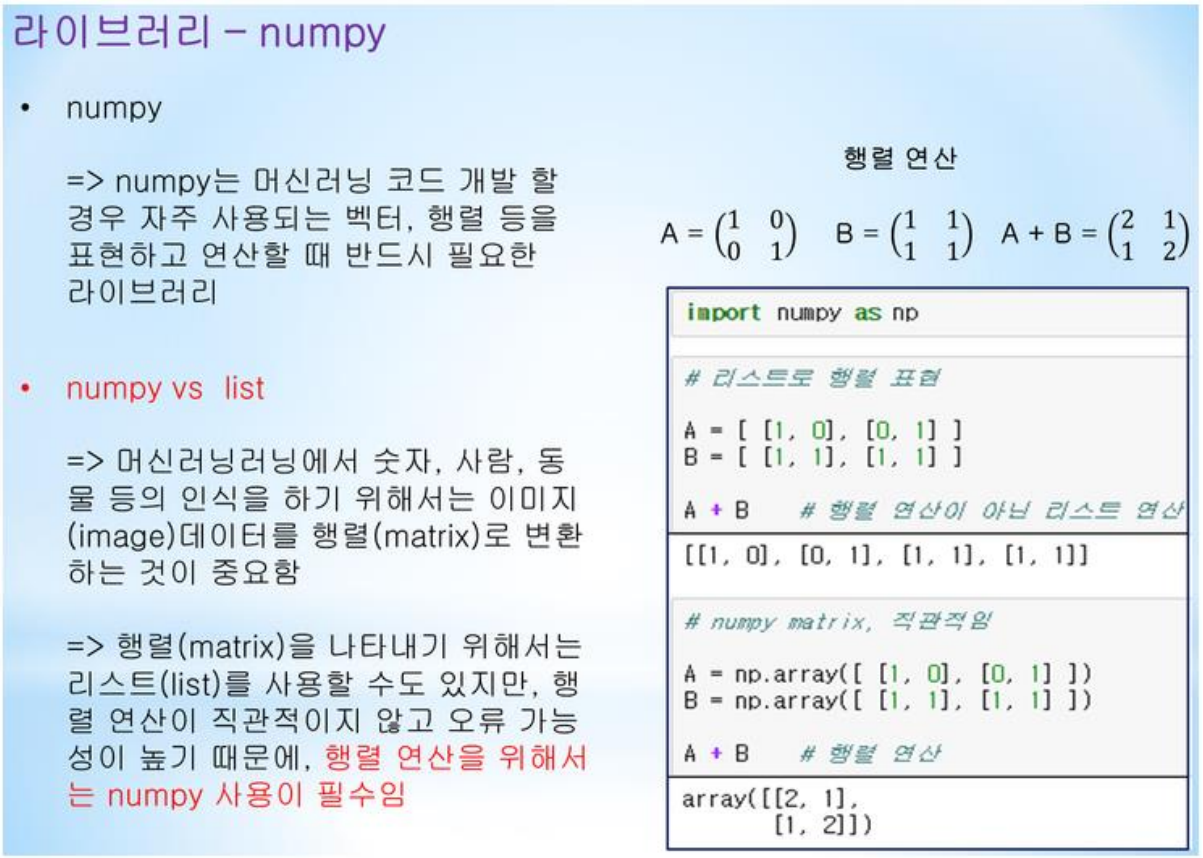
A=np.array([1,2])

from numpy import exp

result = exp(1)

from numpy import *

result = exp(1) + log(1.7) + sqrt(2)



라이브러리 – numpy vector (1차원 배열)

- 벡터(vector) 생성

  => vector는 np.array([…])를 사용하여 생성함 (import numpy as np)

  => 머신러닝 코드 구현 시, 연산을 위해서 vector, matrix 등의 형상(shape), 차원(dimension)을 확인하는 것이 필요함

```
A = np.array([1, 2, 3])
B = np.array([4, 5, 6])

# vector A, B 출력
print("A ==", A, ", B == ", B)

# vector A, B 형상 출력 => shape
print("A.shape ==", A.shape, ", B.shape ==", B.shape)

# vector A, B 차원 출력 => ndim
print("A.ndim ==", A.ndim, ", B.ndim ==", B.ndim)
```

```
A == [1 2 3] , B ==  [4 5 6]
A.shape == (3,) , B.shape == (3,)
A.ndim == 1 , B.ndim == 1
```

- 벡터(vector) 산술연산

  => vector 간 산술연산(+, -, X, /)은 벡터의 각각의 원소에 대해서 행해짐

```
# vector 산술 연산

print("A + B ==", A+B)
print("A - B ==", A-B)
print("A * B ==", A*B)
print("A / B ==", A+B)
```

```
A + B == [5 7 9]
A - B == [-3 -3 -3]
A * B == [ 4 10 18]
A / B == [5 7 9]
```

라이브러리 – numpy matrix (행렬)

- 행렬(matrix) 생성

  => matrix는 vector와 마찬가지로 np.array([[…], […], …])를 사용하여 생성함 (import numpy as np)

$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ shape 2 X 3

$B = \begin{pmatrix} -1 & -2 & -3 \\ -4 & -5 & -6 \end{pmatrix}$ shape 2 X 3

```
A = np.array([ [1, 2 ,3], [4, 5, 6] ])
B = np.array([ [-1, -2, -3], [-4, -5, -6] ])

# matrix A, B 형상 출력 => shape
print("A.shape ==", A.shape, ", B.shape ==", B.shape)

# matrix A, B 차원 출력 => ndim
print("A.ndim ==", A.ndim, ", B.ndim ==", B.ndim)
```

```
A.shape == (2, 3) , B.shape == (2, 3)
A.ndim == 2 , B.ndim == 2
```

- 형 변환 (reshape)

  => vector를 matrix로 변경하거나 matrix를 다른 형상의 matrix로 변경하기 위해서는 reshape() 사용하여 행렬의 shape 을 변경하여야 함

$D = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ shape 2 X 3

$D.reshape(3,2) = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$ shape 3 X 2

```
# vector 생성

C = np.array([1, 2, 3])

# vector C형상 출력 => shape
print("C.shape ==", C.shape)

# vector를 (1,3) 행렬로 형 변환
C = C.reshape(1, 3)

print("C.shape ==", C.shape)
```

```
C.shape == (3,)
C.shape == (1, 3)
```

라이브러리 – numpy 행렬 곱(dot product) (I)

- 행렬 곱(dot product)

  => A 행렬과 B 행렬의 행렬 곱 (dot product)는 np.dot(A,B) 나타내며, 행렬 A 의 열 벡터와 B 행렬의 행 벡터가 같아야 함. 만약 같지 않다면 reshape 또는 전치행렬(transpose)등을 사용하여 형 변환을 한 후에 행렬 곱 실행해야 함

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad B = \begin{pmatrix} -1 & -2 \\ -3 & -4 \\ -5 & -6 \end{pmatrix}$$

형상 2 X 3 3 X 2

$$A \cdot B = \begin{pmatrix} -22 & -28 \\ -49 & -64 \end{pmatrix}$$

(2 X 3) • (3 X 2) = (2 X 2)

```
A = np.array([ [1, 2, 3], [4, 5, 6] ])  # 2X3 행렬
B = np.array([ [-1, -2], [-3, -4], [-5, -6] ])  # 3X2 행렬

# (2X3) dot porduct (3X2) == (2X2) 행렬
C = np.dot(A, B)  # 행렬 곱 수행

# matrix A, B 형상 출력 => shape
print("A.shape ==", A.shape, ", B.shape ==", B.shape)
print("C.shape ==", C.shape)
print(C)
```

```
A.shape == (2, 3) , B.shape == (3, 2)
C.shape == (2, 2)
[[-22 -28]
 [-49 -64]]
```

라이브러리 – numpy 행렬 곱(dot product) (II)

- 행렬 곱(dot product)

  => 행렬 곱은, 행렬의 원소 개수가 같아야만 계산할 수 있는 사칙연산의 한계를 벗어나 ① 행렬곱 조건을 만족하는 다양한 크기의 행렬을 연속으로 만들고 ② 행렬 곱을 연속으로 계산하면서 ③ 결과값을 만들 수 있기 때문에 머신러닝과 이미지 프로세싱 분야에서 자주 사용됨

  [예] 입력 행렬 형상이 64 X 64 이고, 결과 행렬 형상이 64 X 10 이라면, 중간에 dot product 를 만족시키는 어떤 형상의 행렬이라도 가질 수 있음.

  행렬 곱을 사용하지 않고 산술연산만 가능하다면, 입력 행렬의 64 X 64 크기를 가지는 특성 값 만을 사용해야 하기 때문에 다양한 특성을 갖는 필터 개발이 불가능함.

입력 값 특성 값 행렬 결과 값

(64 X 64) • (64 X 256) • (256 X 512) • (512 X 64) • (64 X 10) = (64 X 10)

라이브러리 – numpy broadcast

- 행렬의 사칙연산은 기본적으로 두 개의 행렬 크기가 같은 경우에만 수행할 수 있음. 그러나 numpy 에서는 크기가 다른 두 행렬간에도 사칙연산(+, -, *, /)을 할 수 있는데 이를 브로드캐스트(broadcast)라고 지칭함

  => 차원이 작은 쪽이 큰 쪽의 행 단위로 반복적으로 크기를 맞춘 후에 계산

$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, b = 5 인 경우,

$A + B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + 5 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 5 \\ 5 & 5 \end{pmatrix} = \begin{pmatrix} 6 & 7 \\ 8 & 9 \end{pmatrix}$

broadcast

```
A = np.array([ [1, 2], [3, 4] ])
b = 5

print(A+b)
```

```
[[6 7]
 [8 9]]
```

$C = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, D = (4 5) 인 경우,

$C + D = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + (4\ \ 5) = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 4 & 5 \\ 4 & 5 \end{pmatrix} = \begin{pmatrix} 5 & 7 \\ 7 & 9 \end{pmatrix}$

broadcast

```
C = np.array([ [1, 2], [3, 4] ])
D = np.array([4, 5])

print(C+D)
```

```
[[5 7]
 [7 9]]
```

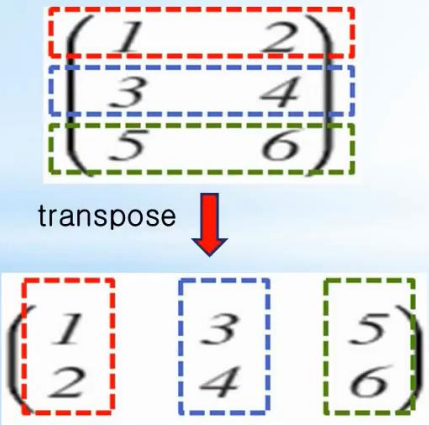
BroadCast는 행렬의 사칙연산에만 해당됨. 즉, 행렬 곱은 해당되지 않음

라이브러리 – numpy 전치행렬 (transpose)

- 전치행렬 (transpose)

  => 어떤 행렬의 전치행렬(transposed matrix)은 원본 행렬의 열은 행으로, 행은 열로 바꾼 것으로서, 원본 행렬을 A라고 하면 전치행렬은 A^T 로 나타냄.

  즉, 1행은 1열로 바꾸고 2행은 2열로, 3행은 3열로 바꾼 행렬을 의미




```
A = np.array([ [1, 2], [3, 4], [5, 6] ])  # 3X2행렬

B = A.T    # A의 전치행렬, 2X3 행렬

print("A.shape ==", A.shape, ", B.shape ==", B.shape)
print(A)
print(B)
```

```
A.shape == (3, 2) , B.shape == (2, 3)
[[1 2]
 [3 4]
 [5 6]]
[[1 3 5]
 [2 4 6]]
```

```
# vector 전치행렬

C = np.array([1, 2, 3, 4, 5])  # vector, matrix 아님
D = C.T      # C 는 vector 이므로 transpose 않됨

E = C.reshape(1, 5)    #  1X5 matrix
F = E.T    # E 의 전치행렬

print("C.shape ==", C.shape, ", D.shape ==", D.shape)
print("E.shape ==", E.shape, ". F.shape ==", F.shape)
print(F)
```

```
C.shape == (5,) , D.shape == (5,)
E.shape == (1, 5) . F.shape == (5, 1)
[[1]
 [2]
 [3]
 [4]
 [5]]
```

라이브러리 – numpy 행렬 indexing / slicing

- 행렬 원소 접근 (I)

  => 행렬 원소를 명시적(explicit)으로 접근하기 위해서는 *리스트(list)*에서처럼, 인덱싱 / 슬라이싱 모두 사용가능 함.

  [예제 1] A[0,0] 또는 A[0][0] 은 1행 1열, A[2, 1] 또는 A[2][1] 은 3행 2열임

  [예제 2] A[: , 0] 은 모든 행 1열을 나타냄

  [예제 3] A[0:-1, 1:2] 인덱스 0인 1 행부터, 인덱스 -1-1=-2인 2행까지의 모든 데이터, 그리고 인덱스 1인 2열부터 인덱스 2-1=1 인 2 열까지의 모든 데이터

- [예제 4] A[:, :] 은 모든 행, 모든 열

참조: [머신러닝 강의 02] 파이썬 데이터타입

```
A = np.array([10, 20, 30, 40, 50, 60]).reshape(3, 2)

print("A.shape ==", A.shape)
print(A)
```

```
A.shape == (3, 2)
[[10 20]
 [30 40]
 [50 60]]
```

```
print("A[0, 0] ==", A[0, 0], ", A[0][0] ==", A[0][0])
print("A[2, 1] ==", A[2, 1], ", A[2][1] ==", A[2][1])
```

```
A[0, 0] == 10 , A[0][0] == 10
A[2, 1] == 60 , A[2][1] == 60
```

```
print("A[0:-1, 1:2] ==", A[0:-1, 1:2])
```

```
A[0:-1, 1:2] == [[20]
 [40]]
```

```
print("A[ :, 0] ==", A[:, 0])
print("A[ :, :] ==", A[:, :])
```

```
A[ :, 0] == [10 30 50]
A[ :, :] == [[10 20]
 [30 40]
 [50 60]]
```

라이브러리 – numpy 행렬 iterator

- 행렬 원소 접근 (II)

  => 명시적(explicit) 인덱싱 / 슬라이싱 이외에, 행렬 모든 원소를 access 하는 경우에는 iterator 사용가능

  (tutorial: https://docs.scipy.org/doc/numpy/reference/generated/numpy.nditer.html)

  => numpy iterator는 C++, Java iterator 처럼 next() 메서드를 통해 데이터 값을 처음부터 끝까지 순차적으로 읽어 들이는 방법을 제공

  [참고] Java iterator interface

```
public interface Iterator {
    boolean hasNext();
    Object  next();
    void    remove();
}
```

```
import numpy as np

A = np.array([ [10, 20, 30, 40], [50, 60, 70, 80] ])

print(A, "₩n")
print("A.shape ==", A.shape, "₩n")

# 행렬 A 의 itrator 생성

it = np.nditer(A, flags=['multi_index'], op_flags=['readwrite'])

while not it.finished:

    idx = it.multi_index

    print("current value => ", A[idx])

    it.iternext()
```

```
[[10 20 30 40]
 [50 60 70 80]]

A.shape == (2, 4)

current value =>  10
current value =>  20
current value =>  30
current value =>  40
current value =>  50
current value =>  60
current value =>  70
current value =>  80
```

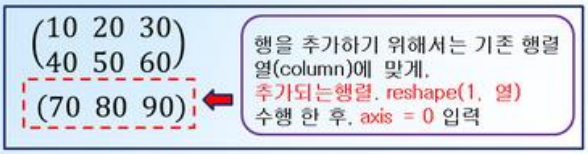
2X4 행렬인 경우,

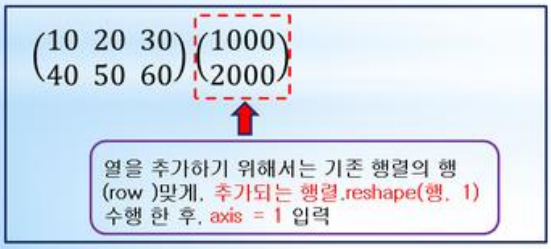
(0,0) => (0,1) => (0,2) => (0,3) => (1,0) => (1,1) => (1,2) => (1,3) 순서로 access

라이브러리 – numpy concatenate(…)

- 행렬에 행(row) 또는 열(columm) 추가하기 위한 numpy.concatenate(…)

  => 머신러닝의 회귀(regression) 코드 구현 시 가중치(weight)와 바이어스(bias)를 별도로 구분하지 않고 하나의 행렬로 취급하기 위한 프로그래밍 구현 기술






```
# 행렬에 열과 행 추가

A = np.array([ [10, 20, 30], [40, 50, 60] ])

print(A.shape)

# A matrix에 행(row) 추가할 행렬, 1행 3열로 reshape
# 행을 추가하기 때문에 우선 열을 3열로 만들어야 함.
row_add = np.array([70, 80, 90]).reshape(1, 3)

# A matrix 에 열(column) 추가할 행렬, 2행 1열 로 생성
# 열을 추가하기 때문에 우선 행을 2행로 만들어야 함.
column_add = np.array([1000, 2000]).reshape(2, 1)
print(column_add.shape)

# numpy.concatenate 에서 axis = 0 행(row) 기준
# A 행렬에 row_add 행렬 추가
B = np.concatenate((A, row_add), axis=0)

print(B)
# numpy.concatenate 에서 axis = 1 열(column) 기준
# B 행렬에 column_add 행렬 추가
C = np.concatenate((A, column_add), axis=1)

print(C)
```

```
(2, 3)
(2, 1)
[[10 20 30]
 [40 50 60]
 [70 80 90]]
[[  10   20   30 1000]
 [  40   50   60 2000]]
```

라이브러리 – numpy useful function (I)

- seperator 로 구분된 파일에서 데이터를 읽기 위한 numpy.loadtxt(…)

  [예제] 다음과 같이 콤마(,)로 분리된 데이터 파일을 read 하기위해서는 np.loadtxt("파일이름",seperator=',') 호출함. 리턴값은 행렬이기 때문에 인덱싱 또는 슬라이싱을 이용하여 데이터를 분리 할 필요 있음

  => 머신러닝 코드에서 입력데이터와 정답데이터를 분리하는 프로그래밍 기법

```
loaded_data = np.loadtxt('./data-01.csv', delimiter=',', dtype=np.float32)

x_data = loaded_data[ :, 0:-1]
t_data = loaded_data[ :, [-1]]

# 데이터 차원 및 shape 확인
print("x_data.ndim = ", x_data.ndim, ", x_data.shape = ", x_data.shape)
print("t_data.ndim = ", t_data.ndim, ", t_data.shape = ", t_data.shape)
```

```
x_data.ndim =  2 , x_data.shape =  (25, 3)
t_data.ndim =  2 , t_data.shape =  (25, 1)
```

./data-01.csv

```
73,80,75,152
93,88,93,185
89,91,90,180
96,98,91,196
73,66,70,142
53,46,55,101
69,74,77,149
47,56,60,115
87,79,90,175
79,70,88,164
69,70,73,141
70,65,74,141
93,95,91,184
79,80,73,152
70,73,78,148
93,89,96,192
78,75,68,147
81,90,93,183
88,92,86,177
78,83,77,159
82,86,90,177
86,82,89,175
78,83,85,175
76,83,71,149
96,93,95,192
```

라이브러리 – numpy useful function (III)

numpy.max(…), numpy.min(…), numpy.argmax(…), numpy.argmin(…)

```
X = np.array([2, 4, 6, 8])

print("np.max(X) ==", np.max(X))
print("np.min(X) ==", np.min(X))
print("np.argmax(X) ==", np.argmax(X))
print("np.argmin(X) ==", np.argmin(X))
```

```
np.max(X) == 8
np.min(X) == 2
np.argmax(X) == 3
np.argmin(X) == 0
```

numpy.ones(…), numpy.zeros(…)

```
A = np.ones([3, 3])

print("A.shape ==", A.shape, ", A ==", A)

B = np.zeros([3, 2])

print("B.shape ==", B.shape, ", B ==", B)
```

```
A.shape == (3, 3) , A == [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
B.shape == (3, 2) , B == [[0. 0.]
 [0. 0.]
 [0. 0.]]
```

```
X = np.array([ [2, 4, 6], [1, 2, 3], [0, 5, 8] ])

print("np.max(X) ==", np.max(X, axis=0)) # axis=0, 열기준
print("np.min(X) ==", np.min(X, axis=0)) # axis=0, 열기준

print("np.max(X) ==", np.max(X, axis=1)) # axis=1, 행기준
print("np.min(X) ==", np.min(X, axis=1)) # axis=1, 행기준

print("np.argmax(X) ==", np.argmax(X, axis=0)) # axis=0, 열기준
print("np.argmin(X) ==", np.argmin(X, axis=0)) # axis=0, 열기준

print("np.argmax(X) ==", np.argmax(X, axis=1)) # axis=1, 행기준
print("np.argmin(X) ==", np.argmin(X, axis=1)) # axis=1, 행기준
```

```
np.max(X) == [2 5 8]
np.min(X) == [0 2 3]
np.max(X) == [6 3 8]
np.min(X) == [2 1 0]
np.argmax(X) == [0 2 2]
np.argmin(X) == [2 1 1]
np.argmax(X) == [2 2 2]
np.argmin(X) == [0 0 0]
```

$$X = \begin{pmatrix} 2 & 4 & 6 \\ 1 & 2 & 3 \\ 0 & 5 & 8 \end{pmatrix}$$

라이브러리 – matplotlib, scatter plot

- 실무에서는 머신러닝 코드를 구현하기 전에,

  => 입력 데이터의 분포와 모양을 먼저 그래프로 그려보고, 데이터의 특성과 분포를 파악 한 후 어떤 알고리즘을 적용할 지 결정하고 있음

  => 데이터 시각화를 위해서는 matplotlib 라이브러리를 사용함

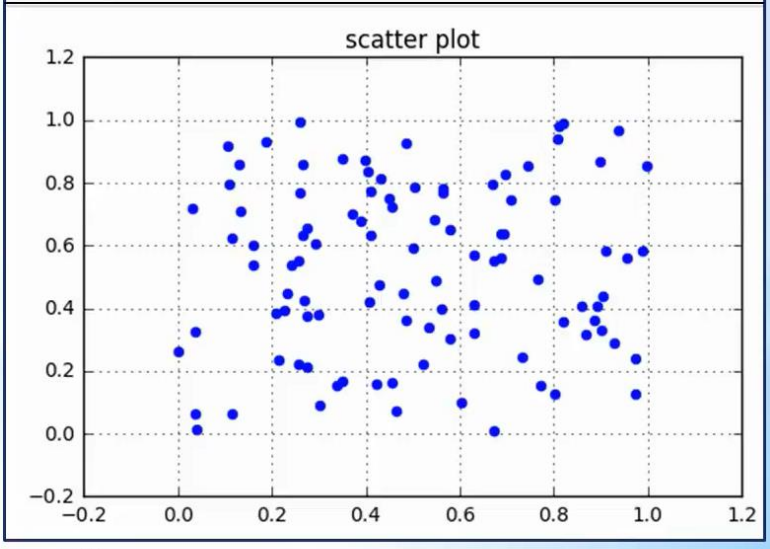
  => 일반적으로 line plot, scatter plot 등을 통해 데이터의 분포와 형태를 파악함

```
import matplotlib.pyplot as plt
import numpy as np

# 주피터 노트북을 사용하는 경우 노트북 내부에 그림 표시
%matplotlib inline

# x data,  y data 생성
x_data = np.random.rand(100)
y_data = np.random.rand(100)

plt.title('scatter plot')
plt.grid()
plt.scatter(x_data, y_data, color='b', marker='o')
plt.show()
```



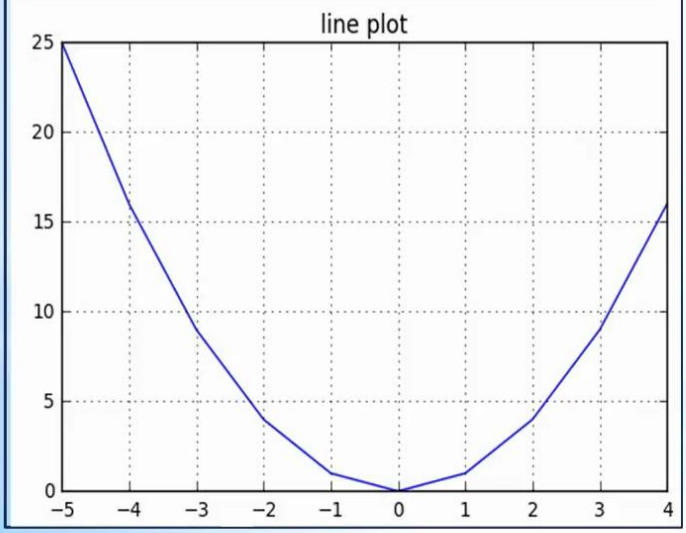
라이브러리 – matplotlib, line plot

```
import matplotlib.pyplot as plt

# 주피터 노트북을 사용하는 경우 노트북 내부에 그림 표시
%matplotlib inline

x_data = [ x for x in range(-5,5) ]
y_data = [ y*y for y in range(-5,5) ]

plt.title('line plot')
plt.grid()
plt.plot(x_data, y_data, color='b')
plt.show()
```

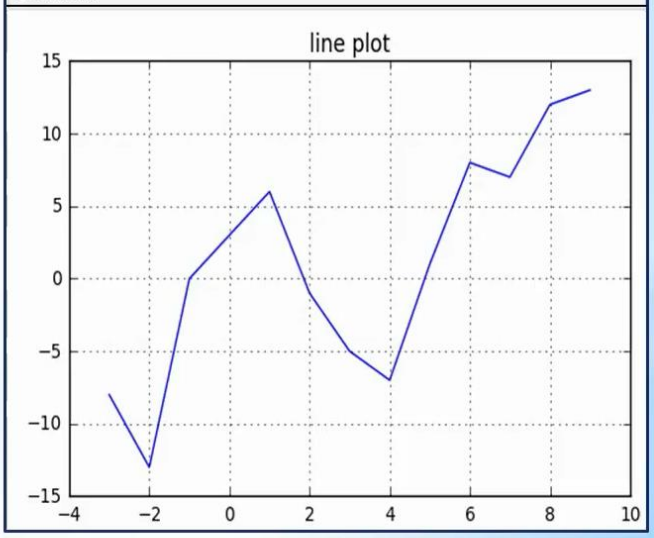



```
import matplotlib.pyplot as plt


# 주피터 노트북을 사용하는 경우 노트북 내부에 그림 표시
%matplotlib inline

x_data = [ -3,  -2, -1, 0, 1, 2,   3,  4, 5, 6, 7,  8, 9 ]
y_data = [ -8, -13, -0, 3, 6, -1, -5, -7, 1, 8, 7, 12, 13 ]

plt.title('line plot')
plt.grid()
plt.plot(x_data, y_data, color='b')
plt.show()
```



Pandas

판다스(Pandas) – 개요

➢ 판다스(Pandas)는 데이터프레임(DataFrame)과 시리즈(Series) 라는 데이터타입(DataType)과 데이터 분석을 위한 다양한 기능을 제공 해주는 파이썬 라이브러리

데이터프레임(DataFrame)

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

시리즈(Series)

- **데이터프레임(DataFrame)**은 엑셀 시트와 같은 테이블 형태의 DataType, 즉 2차원 배열을 나타냄
- **시리즈(Series)**는 데이터프레임에서 한 개의 열(column)을 나타내는 DataType
- 즉, 판다스(Pandas)에서 데이터프레임(DataFrame)이라는 것은 1개 이상의 시리즈(Series)로 구성된 DataType 이라고 할 수 있음

➢ 윈도우 환경에 판다스(Pandas) 설치

- 아나콘다 배포판* 을 설치한 후에, 파이썬 pip 를 이용하여 Pandas 설치 가능

```
(C:₩Program Files₩Anaconda3) C:₩Users₩SungHoPark>pip install pandas
Requirement already satisfied: pandas in c:₩program files₩anaconda3₩lib₩site-packages (0.24.2)
Requirement already satisfied: python-dateutil>=2.5.0 in c:₩program files₩anaconda3₩lib₩site-packages (from pandas)
Requirement already satisfied: numpy>=1.12.0 in c:₩program files₩anaconda3₩lib₩site-packages (from pandas) (1.14.0)
Requirement already satisfied: pytz>=2011k in c:₩program files₩anaconda3₩lib₩site-packages (from pandas) (2016.6.1)
Requirement already satisfied: six>=1.5 in c:₩program files₩anaconda3₩lib₩site-packages (from python-dateutil>=2.5.0
```

※ 아나콘다 배포판 다운로드 : https://www.anaconda.com/distribution/

Quick Tour – 파일에서 데이터 불러오기

➢ 데이터 분석을 하기 위해서는 분석할 데이터를 불러오는 것부터 시작 할 수 있는데, Pandas 에서는 read_csv(…) 를 이용하여 파일에서 데이터를 불러 올 수 있음

- Pandas의 read_csv(…)를 통해서 다음과 같이 콤마로 분리되어 있는 LEC_01_data.csv 을 읽어 들여서 데이터를 불러옴

LEC_01_data.csv

```
Name,Country,Age,Job
John,USA,31,Student
Sabre,France,33,Lawyer
Kim,Korea,28,Developer
Sato,Japan,40,Chef
Lee,Korea,36,Professor
Smith,USA,55,CEO
David,USA,48,Banker
```

- 테이블형태의 데이터타입인 데이터프레임 df가 생성된 것을 알 수 있음

```
import pandas as pd
df = pd.read_csv('data/LEC_01_data.csv')
df      # df 는 테이블 형태의 DataFrame
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

type() 메서드를 통해서 df 데이터 타입을 확인해 보면 DataFrame 임을 알 수 있으며, df 내의 컬럼인 Name 의 데이터타입은 Series 임을 알 수 있음.

```
type(df)  # df 데이터 타입 확인
pandas.core.frame.DataFrame
type(df.Name)  # df.Name 데이터 타입 확인
pandas.core.series.Series
```

Quick Tour – 열(column) 데이터 추출하기

➢ 데이터프레임(DataFrame)에서 열(column) 단위 데이터를 추출하기 위해서는 대괄호 안에 열 이름을 사용함

데이터프레임 df 의 열(column) 이름이 Job인 열 데이터가 추출됨

```
df_job = df['Job']

df_job
```

```
0       Student
1        Lawyer
2     Developer
3          Chef
4     Professor
5           CEO
6        Banker
Name: Job, dtype: object
```

df 에서 Country 열, Job 열을 추출. 추출하고자 하는 열이 2개 이상인 경우 리스트(list) 타입으로 기술되어야 함

```
df_country_job = df[ ['Country', 'Job'] ]

df_country_job
```

리스트(list)

df

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

	Country	Job
0	USA	Student
1	France	Lawyer
2	Korea	Developer
3	Japan	Chef
4	Korea	Professor
5	USA	CEO
6	USA	Banker

Quick Tour – 행(row) 데이터 추출하기

➢ 행 데이터를 추출하기 위해서는 loc, iloc 속성을 사용할 수 있지만 인덱스(index)를 인수로 갖는 loc 가 주로 사용됨 (loc, iloc 자세한 사용법은 이후 '데이터 추출' 강의에서 다룰 예정)

인덱스(index)는 보통 0 부터 시작하지만 행 데이터를 추가, 삭제하면 언제든 변할 수 있으며 숫자만이 아니라 문자열과 datetime 객체도 사용 할 수 있음.

즉, 인덱스는 0, 1 등의 숫자만이 아니라 'first', 'second' 등의 문자열과 2019-01-01 등의 datetime 도 사용 가능함

df

인덱스(index)

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

데이터프레임 df 에서 loc 속성을 이용하여 첫번째 행(row) 데이터를 추출함. 즉, 인덱스 0을 loc 인수로 사용하여 첫번째 행 데이터 추출함

```
df_1st_row = df.loc[0]

df_1st_row
```

```
Name          John
Country        USA
Age             31
Job        Student
Name: 0, dtype: object
```

인덱스 0과 3을 이용하여 1행과 4행 데이터 추출. 추출하고자 하는 행이 2개 이상인 경우 리스트(list) 타입으로 기술되어야 함

```
df_1st_4th_row = df.loc[ [0, 3] ]

df_1st_4th_row
```

리스트(list)

	Name	Country	Age	Job
0	John	USA	31	Student
3	Sato	Japan	40	Chef

Quick Tour – 데이터 분석을 위한 데이터 그룹화 및 통계 계산

➢ 데이터프레임은 데이터를 그룹화 하는 기능과 함께 데이터들의 평균 / 표준편차 / 상관계수 등을 계산하여 데이터 분석을 쉽게 할 수 있는 기능을 제공하고 있음

데이터를 Country 열로 그룹화 하여 df_country_group 변수에 저장

즉, USA / France / Korea / Japan 4개 그룹으로 데이터가 분리됨

Country별로 그룹화된 데이터

	Name	Country	Age	Job
0	John	USA	31	Student
5	Smith	USA	55	CEO
6	David	USA	48	Banker

	Name	Country	Age	Job
2	Kim	Korea	28	Developer
4	Lee	Korea	36	Professor

	Name	Country	Age	Job
1	Sabre	France	33	Lawyer

	Name	Country	Age	Job
3	Sato	Japan	40	Chef

이렇게 분리된 각 그룹에서 Age 열에 대해 평균(mean) 값을 구함

```
df
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO
6	David	USA	48	Banker

```
df_country_group = df.groupby('Country')

age_average = df_country_group['Age'].mean()
```

각 그룹의 Age 열에 대한 평균 값 계산

```
print(age_average)
```

```
Country
France    33.000000
Japan     40.000000
Korea     32.000000
USA       44.666667
Name: Age, dtype: float64
```

파일에서 데이터 불러오기 – csv 파일

➢ 판다스(Pandas)에서는 파일에서 데이터를 불러오는 read_csv(…) 함수 인자(parameter)로 파일이름만 넘겨주면 해당 파일이 콤마(,)로 분리된 데이터 파일이라고 인식함

read_csv(…) 기본 사용법

LEC_02_data.csv

```
Name,Country,Age,Job
John,USA,31,Student
Sabre,France,33,Lawyer
Kim,Korea,28,Developer
Sato,Japan,40,Chef
Lee,Korea,36,Professor
Smith,USA,55,CEO
```

```
import pandas as pd

df1 = pd.read_csv("data/LEC_02_data.csv")

df1
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

데이터프레임 열(column) 이름 변경 / 인덱스 변경

```
# 현재의 column / index 출력

print(df1.columns)
```
← 데이터프레임 칼럼(column) 이름

```
print(df1.index)
```
← 데이터프레임 인덱스

```
print(df1.shape)
```
← 데이터프레임 형상 (행, 열)

```
# column / index 변경
df1.columns = ['이름', '국가', '나이', '직업']

df1.index = ['일', '이', '삼', '사', '오', '육']

df1
```

```
Index(['Name', 'Country', 'Age', 'Job'], dtype='object')
RangeIndex(start=0, stop=6, step=1)
(6, 4)
```

	이름	국가	나이	직업
일	John	USA	31	Student
이	Sabre	France	33	Lawyer
삼	Kim	Korea	28	Developer
사	Sato	Japan	40	Chef
오	Lee	Korea	36	Professor
육	Smith	USA	55	CEO

파일에서 데이터 불러오기 – tsv 파일

➢ 판다스에서 탭(tab)으로 분리되어 있는 데이터 파일을 불러오는 경우에는 파일이름과 함께 데이터 분리자(separator)를 나타내는 sep='₩t' 옵션을 read_csv(…) 인자로 넘겨주어야 함

sep='₩t' 옵션을 사용하지 않는 경우

LEC_02_data_tab.txt

```
Name    Country Age     Job
John    USA     31      Student
Sabre   France  33      Lawyer
Kim     Korea   28      Developer
Sato    Japan   40      Chef
Lee     Korea   36      Professor
Smith   USA     55      CEO
```

```
df2 = pd.read_csv("data/LEC_02_data_tab.txt")
df2
```

	Name Country Age Job
0	John\tUSA\t31\tStudent
1	Sabre\tFrance\t33\tLawyer
2	Kim\tKorea\t28\tDeveloper
3	Sato\tJapan\t40\tChef
4	Lee\tKorea\t36\tProfessor
5	Smith\tUSA\t55\tCEO

데이터가 column 으로 분리되지 않음

sep='₩t' 옵션을 사용하는 경우

```
df2 = pd.read_csv("data/LEC_02_data_tab.txt", sep='₩t')
df2
```

sep='₩t' 옵션을 이용하여 tab으로 분리된 데이터를 column 별로 분리함

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

불러온 데이터 확인하기 – head() 메서드 / tail() 메서드

➢ 판다스(Pandas)의 head() 메서드는 데이터프레임의 가장 앞에 있는 5개의 데이터를 출력해 주며, tail() 메서드는 가장 뒤에 있는 5개의 데이터를 출력해줌

데이터 불러오기

LEC_02_data.csv

```
Name,Country,Age,Job
John,USA,31,Student
Sabre,France,33,Lawyer
Kim,Korea,28,Developer
Sato,Japan,40,Chef
Lee,Korea,36,Professor
Smith,USA,55,CEO
```

```
import pandas as pd
df1 = pd.read_csv("data/LEC_02_data.csv")
df1
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

head() 사용법

```
df1.head()
```

가장 앞의 5개 데이터

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor

```
df1.head(2)
```

가장 앞의 2개 데이터

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer

tail() 사용법

```
df1.tail()
```

가장 뒤의 5개 데이터

	Name	Country	Age	Job
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

```
df1.tail(1)
```

가장 뒤의 1개 데이터

	Name	Country	Age	Job
5	Smith	USA	55	CEO

파일에서 데이터 불러오기 – 헤더(header) 정보가 없는 경우

➢ 헤더(header)정보가 없는 파일에서 데이터를 불러오는 경우 파일이름과 함께 header=None, names = […] 옵션을 read_csv(…) 인자로 넘겨주어야 함

header 옵션 없이 데이터 불러오기

LEC_02_data_no_header.csv

```
John,USA,31,Student
Sabre,France,33,Lawyer
Kim,Korea,28,Developer
Sato,Japan,40,Chef
Lee,Korea,36,Professor
Smith,USA,55,CEO
```

```
df3 = pd.read_csv("data/LEC_02_data_no_header.csv")

df3
```

	John	USA	31	Student
0	Sabre	France	33	Lawyer
1	Kim	Korea	28	Developer
2	Sato	Japan	40	Chef
3	Lee	Korea	36	Professor
4	Smith	USA	55	CEO

1행 데이터가 column 이름으로 인식됨

header=None, names=[…] 옵션 사용하는 경우

```
df3 = pd.read_csv("data/LEC_02_data_no_header.csv", header=None)

df3
```

	0	1	2	3
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

header=None 옵션만 사용하는 경우, column 이름으로 숫자 0, 1, 2 등의 숫자가 지정됨. 즉 데이터 가독성이 떨어짐

```
# header=None 을 사용하지 않고 names 옵션만 주어도 동일한 결과가 나온다
df3 = pd.read_csv("data/LEC_02_data_no_header.csv", header=None, names=['Name', 'Country', 'Age', 'Job'])

df3
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

header=None 옵션과 함께 names=[…] 옵션을 사용하면 데이터 가독성을 높일 수 있음

또한 header=None 을 사용하지 않고 names=[…] 옵션만 사용해도 동일한 결과를 얻을 수 있음

행(row) 데이터 추출하기 – 연속적인 행 데이터 / 불연속적인 행 데이터

➢ 행(row) 데이터를 추출하기 위해서는 일반적으로 ① 연속적인 데이터인 경우 파이썬 슬라이싱(slicing) 기능을 이용하며 ② 불연속적으로 데이터를 추출하기 위해서는 loc[…] 함수를 이용

[머신러닝 강의 02] 10페이지 슬라이싱 설명 참조

데이터 불러오기

LEC_03_data.csv

```
Name,Country,Age,Job
John,USA,31,Student
Sabre,France,33,Lawyer
Kim,Korea,28,Developer
Sato,Japan,40,Chef
Lee,Korea,36,Professor
Smith,USA,55,CEO
```

```
import pandas as pd

df1 = pd.read_csv("data/LEC_03_data.csv")

df1
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

연속적인 행 데이터 추출

```
# 첫 행부터 인덱스 2 행 (3-1) 까지
df1[:3]
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer

```
# 인덱스 3 행부터 4 행의 데이터
df1[3:5]
```

	Name	Country	Age	Job
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor

```
# 인덱스 4행부터 끝까지
df1[4:]
```

	Name	Country	Age	Job
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

불연속적인 행 데이터 추출

```
# 0 행 데이터
df1.loc[0]  # Series 리턴
```

```
Name          John
Country        USA
Age             31
Job        Student
Name: 0, dtype: object
```

```
# 0 행, 3행 데이터
df1.loc[ [0, 3] ]  # DataFrame 리턴
```

	Name	Country	Age	Job
0	John	USA	31	Student
3	Sato	Japan	40	Chef

```
# 1행, 2행, 5행 데이터
df1.loc[ [1, 2, 5] ]
```

	Name	Country	Age	Job
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
5	Smith	USA	55	CEO

열(column) 데이터 추출하기 – 열 이름(name)을 이용한 데이터 추출

➢ 데이터프레임 변수를 df 라고 할 경우, ① df.열이름 또는 df['열이름 '] 을 이용하여 하나의 열 데이터를 가져올 수 있으며 ② 2개 이상의 열 데이터를 가져오기 위해서는 다음과 같이 df[['열이름1', '열이름2', …]] 리스트 타입을 이용하여 복수개의 열 데이터를 가져올 수 있음

리스트(list) 데이터 타입

데이터 불러오기

LEC_03_data.csv

```
Name,Country,Age,Job
John,USA,31,Student
Sabre,France,33,Lawyer
Kim,Korea,28,Developer
Sato,Japan,40,Chef
Lee,Korea,36,Professor
Smith,USA,55,CEO
```

```
import pandas as pd

df1 = pd.read_csv("data/LEC_03_data.csv")

df1
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

하나의 열 데이터 추출

```
# 하나의 열 데이터 추출
df1['Country']  # Series 타입 리턴
```

```
0      USA
1   France
2    Korea
3    Japan
4    Korea
5      USA
Name: Country, dtype: object
```

```
# 하나의 열 데이터 추출
df1.Country  # Series 타입 리턴
```

```
0      USA
1   France
2    Korea
3    Japan
4    Korea
5      USA
Name: Country, dtype: object
```

2개 이상의 열 데이터 추출

```
# 2개의 열 데이터 추출
df1[ ['Name', 'Country'] ]
```

	Name	Country
0	John	USA
1	Sabre	France
2	Kim	Korea
3	Sato	Japan
4	Lee	Korea
5	Smith	USA

```
# 3개의 열 데이터 추출
df1[ ['Name', 'Country', 'Job'] ]
```

	Name	Country	Job
0	John	USA	Student
1	Sabre	France	Lawyer
2	Kim	Korea	Developer
3	Sato	Japan	Chef
4	Lee	Korea	Professor
5	Smith	USA	CEO

열(column) 데이터 추출하기 – 열 인덱스를 이용한 데이터 추출

➢ 데이터프레임에서 열(column) 이름이 없고 열 인덱스(index)만 있는 경우에는 파이썬 슬라이싱(slicing)과 iloc[…] 함수를 이용하여 열 데이터를 추출 할 수 있음

데이터 불러오기

LEC_03_data_no_header.csv

```
John,USA,31,Student
Sabre,France,33,Lawyer
Kim,Korea,28,Developer
Sato,Japan,40,Chef
Lee,Korea,36,Professor
Smith,USA,55,CEO
```

```
df2 = pd.read_csv("data/LEC_03_data_no_header.csv", header=None)

df2
```

	0	1	2	3
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

열(column) 이름 대신에 열 인덱스만 있음

연속적인 열 데이터 추출

```
# 모든 행, 1열부터 2열까지
df2.iloc[ :, 1: 3 ]
```

	1	2
0	USA	31
1	France	33
2	Korea	28
3	Japan	40
4	Korea	36
5	USA	55

```
# 1행부터 3행, 2열부터 끝까지
df2.iloc[ 1:4, 2: ]
```

	2	3
1	33	Lawyer
2	28	Developer
3	40	Chef

불연속적인 열 데이터 추출

```
# 모든 행, 0열, 2열, 3열
df2.iloc[ :, [0, 2, 3] ]
```

	0	2	3
0	John	31	Student
1	Sabre	33	Lawyer
2	Kim	28	Developer
3	Sato	40	Chef
4	Lee	36	Professor
5	Smith	55	CEO

```
# 1행 3행, 1열, 3열
df2.iloc[ [1, 3], [1, 3] ]
```

	1	3
1	France	Lawyer
3	Japan	Chef

열(column) 데이터 추출하기 – 조건에 따른 데이터 추출

➢ 데이터프레임의 연산자 (>, ==, <, !=, &, | 등) 를 이용하여 열 데이터를 추출할 수 있음

데이터 불러오기

LEC_03_data.csv

```
Name,Country,Age,Job
John,USA,31,Student
Sabre,France,33,Lawyer
Kim,Korea,28,Developer
Sato,Japan,40,Chef
Lee,Korea,36,Professor
Smith,USA,55,CEO
```

```
import pandas as pd

df1 = pd.read_csv("data/LEC_03_data.csv")

df1
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

조건에 따른 열 데이터 추출

```
# Country 가 Korea 인 경우
df1[ df1.Country == 'Korea' ]
```

	Name	Country	Age	Job
2	Kim	Korea	28	Developer
4	Lee	Korea	36	Professor

```
# Age > 30 인 경우
df1[ df1.Age > 30 ]
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor
5	Smith	USA	55	CEO

and / or 에 따른 열 데이터 추출

```
# Country != 'USA' & Age >= 36
df1[ (df1.Country != 'USA') & (df1.Age >= 33) ]
```

	Name	Country	Age	Job
1	Sabre	France	33	Lawyer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor

&, | 연산자를 사용할때는 조건식을 괄호로 묶어 주어야 함

```
# Country == Korea | Job == Chef
df1[ (df1.Country == 'Korea') | (df1.Job == 'Chef') ]
```

	Name	Country	Age	Job
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef
4	Lee	Korea	36	Professor

데이터프레임(DataFrame) 만들기 – from dict type

➢ 판다스에서는 dictionary 타입 데이터 또는 순서가 보장되어 있는 OrderedDict 타입 데이터를 DataFrame 클래스의 인자로 넘겨주면 데이터프레임(DataFrame)을 직접 만들 수 있음

dictionary key 값이 데이터프레임 column 이름이며, value 값이 데이터프레임의 실제 column 데이터임

```
import pandas as pd

persons_dict = { 'Name' : [ 'John', 'Sabre', 'Kim', 'Sato' ],
                 'Country' : [ 'USA', 'France', 'Korea', 'Japan' ],
                 'Age' : [ 31, 33, 28, 40 ],
                 'Job' : ['Student', 'Lawyer', 'Developer', 'Chef'] }
```

데이터프레임을 만들 때, 어떤 옵션도 주지 않으면 열(column) 순서가 random 하게 생성됨

```
df1 = pd.DataFrame(persons_dict)

df1
```

	Age	Country	Job	Name
0	31	USA	Student	John
1	33	France	Lawyer	Sabre
2	28	Korea	Developer	Kim
3	40	Japan	Chef	Sato

열 순서가 random 하게 생성됨

columns 옵션을 주어 열(column) 순서를 지정하면 원하는 순서를 갖는 데이터프레임을 만들 수 이음

```
df2 = pd.DataFrame(data = persons_dict,
                   columns = [ 'Name', 'Country', 'Age', 'Job' ] )

df2
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef

columns 옵션으로 지정한 순서대로 열 데이터 생성됨

데이터프레임(DataFrame) 만들기 – from OrderedDict type

데이터 생성 순서가 보장되는 OrderedDict key 값이 데이터프레임 column 이름이며, value 값이 데이터프레임의 실제 column 데이터임

```
from collections import OrderedDict

ordered_persons_dict = OrderedDict( [('Name', [ 'John', 'Sabre', 'Kim', 'Sato' ]),
                                     ('Country', [ 'USA', 'France', 'Korea', 'Japan' ]),
                                     ('Age', [ 31, 33, 28, 40 ]),
                                     ('Job', ['Student', 'Lawyer', 'Developer', 'Chef'])] )
```

column 옵션을 주지 않아도 데이터프레임의 열(column) 데이터가 OrderdDict 을 만들 때 주어진 Name, Country, Age, Job 으로 생성된 것을 알 수 있음

```
df1 = pd.DataFrame(ordered_persons_dict)

df1
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Developer
3	Sato	Japan	40	Chef

columns 옵션을 주지 않아도 열 데이터가 순서에 맞게 생성됨

데이터프레임을 만들 때, index 옵션을 이용하면 인덱스를 숫자가 아닌 문자를 포함한 임의의 데이터로 변경 할 수 있음

```
# DataFrame 생성 시, index 옵션을 주면 index 값을 별도로 지정할 수 있음

df2 = pd.DataFrame(ordered_persons_dict, index = ['one', 'two', 'three', 'four' ])

df2
```

	Name	Country	Age	Job
one	John	USA	31	Student
two	Sabre	France	33	Lawyer
three	Kim	Korea	28	Developer
four	Sato	Japan	40	Chef

index 옵션으로 숫자가 아닌 문자로 인덱스 값을 변경함

데이터프레임 csv / tsv / txt 파일로 저장하기

➢ 데이터프레임을 파일로 저장하기 위해서는 일반적으로 to_csv(…) 함수를 사용하며, to_csv(…) 에서 분리자를 나타내는 sep='… ' 옵션을 사용하면 csv / tsv / txt 등의 다양한 파일 형식으로 저장할 수 있음

데이터프레임 df1 생성

```
import pandas as pd

persons_dict = { 'Name' : [ 'John', 'Sabre', 'Kim' ],
                 'Country' : [ 'USA', 'France', 'Korea' ],
                 'Age' : [ 31, 33, 28 ],
                 'Job' : ['Student', 'Lawyer', 'Chef'] }

df1 = pd.DataFrame(data = persons_dict,
                   columns = [ 'Name', 'Country', 'Age', 'Job' ] )

df1
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Chef

데이터프레임을 csv / tsv / txt 파일로 저장

```
# 분리자(seperator)로서 comma, tab, space 지정하여 저장
# 분리자 옵션을 생략하면 기본적으로 sep=',' 인식

df1.to_csv('persons_01.csv', sep = ',')      # comma로 분리해서 저장

df1.to_csv('persons_01.tsv', sep = '\t')     # tab으로 분리해서 저장

df1.to_csv('persons_01.txt', sep = ' ')      # 공백으로 분리해서 저장
```

persons_01.csv

```
,Name,Country,Age,Job
0,John,USA,31,Student
1,Sabre,France,33,Lawyer
2,Kim,Korea,28,Chef
```

persons_01.tsv

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Chef

persons_01.txt

```
Name Country Age Job
0 John USA 31 Student
1 Sabre France 33 Lawyer
2 Kim Korea 28 Chef
```

데이터프레임 엑셀(excel) 파일로 저장하기

➢ 데이터프레임을 엑셀 파일로 저장하기 위해서는 ① 먼저 xlwt, openpyxl 라이브러리를 설치한 후에 ② to_excel(…) 함수를 이용하여 데이터프레임을 엑셀(excel) 파일로 저장 할 수 있음

라이브러리 설치

```
C:₩Users₩SungHoPark> pip install xlwt
Installing collected packages: xlwt
Successfully installed xlwt-1.3.0

C:₩Users₩SungHoPark> pip install openpyxl
Installing collected packages: openpyxl
Successfully installed openpyxl-2.6.2
```

데이터프레임 df1 생성

```
import pandas as pd

persons_dict = { 'Name' : [ 'John', 'Sabre', 'Kim' ],
                 'Country' : [ 'USA', 'France', 'Korea' ],
                 'Age' : [ 31, 33, 28 ],
                 'Job' : ['Student', 'Lawyer', 'Chef'] }

df1 = pd.DataFrame(data = persons_dict,
                   columns = [ 'Name', 'Country', 'Age', 'Job' ] )

df1
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Chef

데이터프레임을 엑셀 파일로 저장

```
# 데이터프레임을 엑셀의 xls 형식으로 저장
import xlwt

df1.to_excel('persons_03.xls')

# 데이터프레임을 엑셀의 xlsx 형식으로 저장
import openpyxl

df1.to_excel('persons_03.xlsx')
```

persons_03.xls

	A	B	C	D	E
1		Name	Country	Age	Job
2	0	John	USA	31	Student
3	1	Sabre	France	33	Lawyer
4	2	Kim	Korea	28	Chef

persons_03.xlsx

	A	B	C	D	E
1		Name	Country	Age	Job
2	0	John	USA	31	Student
3	1	Sabre	France	33	Lawyer
4	2	Kim	Korea	28	Chef

데이터프레임 JSON 파일로 저장하기

➢ JSON (JavaScript Object Notation)은 네트워크를 통해 데이터를 주고받을 때 자주 사용되고 표준화된 경량 (light) 데이터 형식이며, to_json(…) 함수를 사용하여 데이터프레임을 JSON 파일로 저장 할 수 있음

데이터프레임 df1 생성

```
import pandas as pd

persons_dict = { 'Name' : [ 'John', 'Sabre', 'Kim' ],
                 'Country' : [ 'USA', 'France', 'Korea' ],
                 'Age' : [ 31, 33, 28 ],
                 'Job' : ['Student', 'Lawyer', 'Chef'] }

df1 = pd.DataFrame(data = persons_dict,
                   columns = [ 'Name', 'Country', 'Age', 'Job' ] )

df1
```

	Name	Country	Age	Job
0	John	USA	31	Student
1	Sabre	France	33	Lawyer
2	Kim	Korea	28	Chef

데이터프레임을 JSON 파일로 저장

```
df1.to_json('persons_04.json')
```

persons_04.json


```
{ "Name":{"0":"John","1":"Sabre","2":"Kim"},
  "Country":{"0":"USA","1":"France","2":"Korea"},
  "Age":{"0":31,"1":33,"2":28},
  "Job":{"0":"Student","1":"Lawyer","2":"Chef"} }
```