

Python 분석 기초

모듈 - 2

강사: 장순용 박사

2. 파이썬 라이브러리 활용 분석.

2.1. Numpy 패키지 기초.

2.2. Numpy 연산.

2.3. Numpy 선형 대수학.

2.4. Matplotlib 패키지를 활용한 시각화.

2.5. Pandas 데이터 프레임과 시리즈.

2.6. Pandas 데이터 가공.

Numpy 패키지에 대해서

Numpy 패키지:

- Numpy는 과학 계산과 데이터 다루기를 용이하게 해주는 패키지 이다.
- 고차원 배열을 객체로 제공하며 여러 관련 메서드를 제공한다.
- 이들 메서드의 연산 속도는 최적화 되어 있다 (빠르다).
- 연산의 벡터화 제공 → 코딩하기 편리하며 가독성 증대.

Numpy 패키지 기초 : Numpy 배열

Numpy 배열과 파이썬의 리스트 사이의 차이점:

- Numpy 배열의 크기는 정해져 있다: 크기를 변경하는 경우 새로운 객체가 생성된다.
- Numpy 배열 개개 원소의 자료형은 일치되어 있어야 한다.
- 기존 파이썬 리스트가 제공하지 않는 많은 수학 연산을 기본적으로 제공한다.
- 다른 패키지에서도 Numpy 배열을 기초 자료형으로 사용한다.

Numpy 패키지 기초 : Numpy 배열

Numpy 배열: 생성과 기본 특성

```
In[1] : import numpy as np                # 패키지를 가져와서 np 라고 부름.
In[2] : np.__version__                   # 현재 설치된 Numpy의 버전.
Out[2]: '1.11.3'

In[3] : arr1 = np.array([1,3,5,7,9])      # 리스트 사용하여 배열 생성.
In[4] : arr2 = np.array((1,3,5,7,9))     # 튜플 사용하여 배열 생성.
In[5] : type(arr1)
Out[5]: numpy.ndarray

In[6] : arr3 = arr1
In[7] : id(arr1)                          가      가
Out[7]: 93269488L
In[8] : id(arr3)
Out[8]: 93269488L                        # arr1와 arr3은 메모리 공간 공유.
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열: 생성과 기본 특성

```
In[9] : arr4 = arr1.copy()                # 얕은 복사.  
In[10] : id(arr1)  
Out[10]: 93269488L  
In[11] : id(arr4)  
Out[11]: 93494048L                # 완전히 다른 객체.
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열: arange 함수

```
In[1] : np.arange(10)                                # 0~9.
Out[1] : array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In[2] : np.arange(10,20)                             # 10~19.
Out[2] : array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])

In[3] : np.arange(10, 20, 3)                         # 10~19, step = 3.
Out[3] : array([10, 13, 16, 19])

In[4] : arr = np.arange(10)

In[5] : len(arr)                                     # 리스트의 경우와 같음.
Out[5] : 10

In[5] : arr.size                                     # 리스트의 경우에는 사용 불가능.
Out[5] : 10                                         # Numpy 배열인 경우에만 가능.
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열: linspace 함수

```
In[1] : np.linspace(0, 10, 5)
```

0과 10 사이 5개의 그릿.

```
Out[1] : array([0.0, 2.5, 5.0, 7.5, 10.0])
```


Numpy 패키지 기초 : Numpy 배열

Numpy 배열: zeros 함수 0

```
In[1] : np.zeros(10)
Out[1] : array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]) # 10개의 0.

In[2] : np.zeros((3,4))
Out[2] :
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]]) # 3 x 4 행렬.

In[3] : np.zeros(5, dtype='int64') # 자료형 명시.
Out[3] : array([0, 0, 0, 0, 0], dtype=int64)

In[4] : np.zeros(5, dtype='int64').dtype
Out[4] : dtype('int64')
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열: ones 함수 1

```
In[1] : arr = np.ones((2,3), dtype='int_')
```

```
In[2] : arr
```

```
Out[2] :
```

```
array([[ 1,  1,  1],  
       [ 1,  1,  1]])
```

```
In[3] : arr.dtype
```

```
Out[3] : dtype('int32')
```

```
In[4] : arr.astype('float32')
```

```
Out[4] :
```

```
array([[ 1.,  1.,  1.],  
       [ 1.,  1.,  1.]], dtype=float32)
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열: 원소의 자료형은 일치되어야 한다

```
In[1] : arr1 = np.array([111, 2.3, True, False, False])      # 숫자형과 불 원소 혼재.
In[2] : arr1
Out[2]: ([111., 2.3, 1., 0., 0.])                             # 숫자형으로 자동 변환.
In[3] : arr2 = np.array([111, 2.3, 'python', 'abc'])         # 숫자형과 문자열 혼재.
In[4] : arr2
Out[4]: array(['111', '2.3', 'python', 'abc'],               # 문자열로 자동 변환.
             dtype='<S32')
In[5] : arr3 = np.array([111, True, 'abc'])                  # 숫자형, 문자열, 불 원소 혼재.
In[6] : arr3
Out[6]: array(['111', 'True', 'abc'],                         # 문자열로 자동 변환.
             dtype='<S32')
```

Numpy 패키지 기초 : Numpy 배열

Numpy의 자료형:

자료형	설명
int8, int16, int32, int64, int_ uint8, uint16, uint32, uint64	정수.
float16, float32, float64, float128, float_	실수.
bool_	부울.
string_, unicode_	문자열.

Numpy 패키지 기초 : Numpy 배열

Numpy 배열의 인덱싱과 슬라이싱:

```
In[1] : a = np.array([1, 2, 3, 4, 5])           # 1D 배열.  
In[2] : a[:2]  
Out[2]: array([1, 2])  
In[3] : a[-1]                                  # 끝에서 첫번째 원소.  
Out[3]: 5  
In[4] : a[:]                                    # 전체!  
Out[4]: array([1, 2, 3, 4, 5])
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열의 인덱싱과 슬라이싱:

```
In[1] : a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])      # 3 x 3 행렬.
In[2] : a[1]                                                # 행 1.
Out[2]: array([4, 5, 6])
In[3] : a[-1]                                              # 마지막에서 첫 번째 행.
Out[3]: array([7, 8, 9])
In[4] : a[:]                                              # 전체!
Out[4]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열의 인덱싱과 슬라이싱:

```
In[5] : a[:2]
```

```
Out[5]:
```

```
array([[1, 2, 3],  
       [4, 5, 6]])
```

```
In[6] : a[2][1]
```

```
Out[6]: 8
```

```
In[7] : a[2, 1]
```

```
Out[7]: 8
```

```
In[8] : a[[0, 2]]
```

```
Out[8]:
```

```
array([[1, 2, 3],  
       [7, 8, 9]])
```

```
In[9] : a[1:, 1:]
```

```
Out[9]:
```

```
array([[5, 6],  
       [8, 9]])
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열의 모양:

In[5] : a.size

원소의 갯수.

Out[5]: 9

In[6] : a.shape

전체적인 모양(shape).

Out[6]: (3, 3)

In[7] : a.ndim

디멘전 수.

Out[7]: 2

Numpy 패키지 기초 : Numpy 배열

Numpy 배열의 모양:

```
In[1] : a = np.arange(15)
```

0~14. 전체 15개의 원소. 1D 배열.

```
In[2] : a.reshape(3,5)
```

3 x 5 모양의 2D 행렬로 바꾸어 **보여줌**. (비행구)

```
Out[2]:
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

```
In[3] : a
```

```
Out[3]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

```
In[4] : a.shape = (3, 5)
```

3 x 5 모양의 2D 행렬로 **바꾸어줌**. (항구적)

```
In[5] : a
```

```
Out[5]:
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열의 모양 (주의):

```
In[1] : a = np.array([2,5,1,3])
```

```
In[2] : a.shape
```

```
Out[2]: (4, )
```

rank 1 배열! 벡터는 아님.

```
In[3] : a = a.reshape(4,1)
```

이제는 컬럼 벡터.

```
In[4] : a
```

```
Out[3]:
```

```
array([ [2],  
        [5],  
        [1],  
        [3]])
```

```
In[5] : a.shape
```

```
Out[5]: (4,1)
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열 (주의):

```
In[1] : a = np.arange(10)                                # 0~9. 전체 10개의 원소. 1D 배열.
In[2] : b = a.reshape(2,5)                                # 2 x 5 모양의 2D 행렬.
In[3] : a[0] = -999
In[4] : b
Out[4]:
array([[ -999,  1,  2,  3,  4],
       [  5,  6,  7,  8,  9]])                            # 메모리 공간 공유.
In[5] : c = a.reshape(2,5).copy()                        # 얇은 복사로 새로운 객체 만듦.
In[6] : c[0, 0] = 0
In[7] : a
Out[7]:
array([ -999,  1,  2,  3,  4,  5,  6,  7,  8,  9])          # 메모리 공간이 다르므로 아무런 영향 없음.
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열의 fancy indexing:

```
In[1] : arr = np.arange(100)                # 0~99.
In[2] : arrMask = ( (arr % 5) == 0 )        # 로직배열. 5의 배수인 경우 True.
In[3] : arr[arrMask]
Out[3]:
array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
       85, 90, 95])
In[4] : arrMask = ( (arr % 5) == 0 ) & ( arr > 0 )    # 로직배열. 0 이상의 5의 배수이면 True.
In[5] : arr[arrMask]
Out[5]:
array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85,
       90, 95])
```

Numpy 패키지 기초 : Numpy 배열

Numpy 배열의 확장:

```
In[1] : a = np.array([1, 2, 3])
```

```
In[2] : b = np.append(a, [4, 5, 6])
```

```
In[3] : b
```

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

append

numpy

가

append

2,3

axis

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

```
In[2] : b = np.append(a, [[9, 9]], axis=0)
```

새로운 행으로 추가 (비행구적).

```
In[3] : c = np.append(a, [[9], [9]] , axis=1)
```

새로운 열로 추가 (비행구적).

Numpy 패키지 기초 : Numpy 배열

Numpy 배열의 삭제:

```
In[1] : a = np.array([[1, 2, 3],[4, 5, 6]])
```

```
In[2] : np.delete(a, 0)
```

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

 # 원소 하나 삭제 (비행구적).

```
In[3] : np.delete(a, (0, 2, 4))
```

```
Out[3]: array([2, 4, 6])
```

```
In[4] : np.delete(a, 0, axis = 0)
```

행 전체 삭제 (비행구적).

```
Out[4]: array([4, 5, 6])
```

```
In[5] : np.delete(a, 1, axis = 1)
```

열 전체 삭제 (비행구적).

```
Out[5]:
```

```
array([[1, 3],  
       [4, 6]])
```

실습 #0201

()

→ Numpy 배열에 대해서 알아보니다. ←

→ 사용: [ex_0201.ipynb](#) ←

2. 파이썬 라이브러리 활용 분석.

2.1. Numpy 패키지 기초.

2.2. Numpy 연산.

2.3. Numpy 선형 대수학.

2.4. Matplotlib 패키지를 활용한 시각화.

2.5. Pandas 데이터 프레임과 시리즈.

2.6. Pandas 데이터 가공.

Numpy 배열의 연산

리스트 연산과 Numpy 배열의 연산 비교: '+' 연산자

```
In[1] : a = [1, 2, 3]
In[2] : b = [4, 5, 6]
In[3] : a + b                                # 리스트의 경우는 연결의 의미.
Out[3]: [1, 2, 3, 4, 5, 6]
```

```
In[1] : a = np.array([1, 2, 3])
In[2] : b = np.array([4, 5, 6])
In[3] : a + b                                # Numpy 배열인 경우에는 원소별 연산의 의미.
Out[3]: array([5, 7, 9])
```

Numpy 배열의 연산

리스트 연산과 Numpy 배열의 연산 비교: '*' 연산자

```
In[1] : a = [1, 2, 3]
```

```
In[2] : 3 * a
```

리스트의 경우는 반복의 의미.

```
Out[2]: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
In[3] : b = np.array([1, 2, 3])
```

```
In[4] : 3 * b
```

Numpy 배열인 경우에는 원소별 연산의 의미.

```
Out[4]: array([3, 6, 9])
```

```
In[5] : np.array(3 * [1, 2, 3])
```

```
Out[5]: array([1, 2, 3, 1, 2, 3, 1, 2, 3])
```

```
In[6] : np.repeat(b, 3)
```

```
Out[6]: array([1, 1, 1, 2, 2, 2, 3, 3, 3])
```

Numpy 배열의 연산

Numpy 배열의 연산:

```
In[1] : a = np.array([1, 2, 3])
```

```
In[2] : b = np.array([4, 5, 6])
```

```
In[3] : a + b
```

```
Out[3]: array([5, 7, 9])
```

```
In[4] : b - a
```

```
Out[4]: array([3, 3, 3])
```

```
In[5] : a * b
```

```
Out[5]: array([4, 10, 18])
```

```
In[6] : 1.0*a / b
```

```
Out[6]: array([0.25, 0.4, 0.5])
```

Numpy 배열의 연산 : 벡터화

Numpy 연산의 벡터화 (Universal function):

```
In[1] : x = np.array([0, 1, 2, 3])
In[2] : x**3
Out[2]: array([ 0,  1,  8, 27])
In[3] : np.sqrt(x)
Out[3]: array([ 0.,  1.,  1.41421356,  1.73205081]) # Universal function <= 원소별 적용.
In[4] : np.exp(x)
Out[4]: array([ 1. ,  2.71828183,  7.3890561 , 20.08553692])
In[5] : np.sin(x)
Out[5]: array([0. ,  0.84147098,  0.90929743,  0.14112001])
In[6] : type(np.sin) # Check the type.
Out[6]: numpy.ufunc
```

Numpy의 함수

Numpy가 제공하는 함수:

함수	설명	Universal function?
sin, cos, tan	삼각함수.	Yes
arcsin, arccos, arctan	역삼각함수.	Yes
round	소수점 이하 표기.	Yes
floor	작으면서 제일 가까운 정수.	Yes
ceil	크면서 제일 가까운 정수.	Yes
fix	0 방향으로 가장 가까운 정수.	Yes
prod	배열 원소들의 곱.	No
cumsum	누적합.	No
sum, mean, var, std, median	다양한 통계치.	No
exp, log	지수, 로그 함수.	Yes
unique	고유한 값.	No
min, max, argmax, argmin	최소, 최대값과 위치 함수.	No

Numpy 배열의 메소드

Numpy가 배열의 메서드로 제공하는 통계 함수:

```
In[1] : x = np.arange(1,11)
In[2] : x.sum()
Out[2]: 55                                # 배열의 원소 합.
In[3] : x.mean()
Out[3]: 5.5                              # 평균.
In[4] : x.std()
Out[4]: 2.87228                          # 표준 편차.
In[5] : x.var()
Out[5]: 8.25                             # 분산.
In[6] : x.cumsum()
Out[6]: array([1, 3, 6, 10, 15, 21, 28, 36, 44, 55], dtype=int32) # 누적 배열.
```

Numpy 배열의 메소드

Numpy가 배열의 메서드로 제공하는 통계 함수:

```
In[1] : x = np.arange(1,10)
```

```
In[2] : x = x.reshape((3,-1))
```

(3,-1) 는 (3,3)의 의미!

```
In[3] : print(x)
```

```
Out[3]:
```

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

```
  [4 5 6]
```

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

```
In[4] : print(x.mean(axis=0))
```

```
Out[4]:
```

```
[ 4. 5. 6.]
```

열의 평균.

```
In[5] : print(x.mean(axis=1))
```

```
Out[5]:
```

```
[ 2. 5. 8.]
```

행의 평균.

Numpy 배열의 메소드

Numpy가 배열의 메서드로 제공하는 통계 함수:

```
In[1] : np.random.seed(123)
```

```
In[2] : x = np.random.randint(10, size=1000)
```

```
In[3] : x.max()
```

```
Out[3]: 9
```

```
In[4] : x.min()
```

```
Out[4]: 0
```

```
In[5] : (x > 5).sum()
```

```
Out[5]: 401
```

```
In[6] : np.unique(x)
```

```
Out[6]:
```

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

```
In[7] : sorted(set(x))
```

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

0~9 사이 정수 랜덤으로 1000개.

0~9 가 1000

True를 1로 집계.

Unique한 값만 가져온다.

Unique한 값만 가져온다 (순수 파이썬).

Numpy 배열의 메소드

Numpy가 배열의 메서드로 제공하는 통계 함수:

함수	설명
mean	평균.
var	분산.
std	표준편차.
sum	합.
cumsum	배열 원소들의 누적 합.
max, min	최대값, 최소값.
argmax, argmin	최대값, 최소값의 위치.

*max
*argmax

-
-

가

실습 #0202

→ Numpy 배열의 연산에 대해서 알아봅니다. ←

→ 사용: [ex_0202.ipynb](#) ←

2. 파이썬 라이브러리 활용 분석.

2.1. Numpy 패키지 기초.

2.2. Numpy 연산.

2.3. Numpy 선형 대수학.

2.4. Matplotlib 패키지를 활용한 시각화.

2.5. Pandas 데이터 프레임과 시리즈.

2.6. Pandas 데이터 가공.

Numpy 배열의 연산 : 선형 대수학

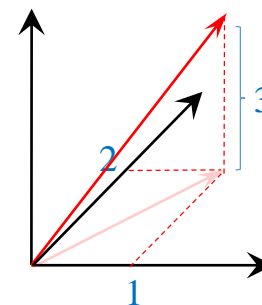
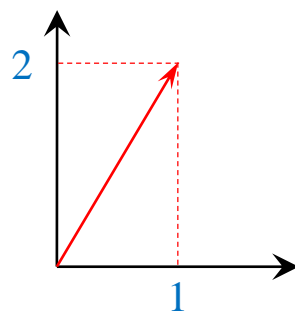
스칼라와 벡터:

=> Scalars는 수치 하나.

Ex). 7 3.141 - 2 9.083

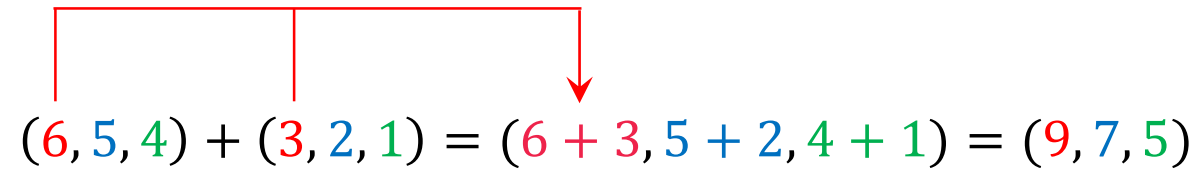
=> Vectors에는 방향과 길이가 있다. 배열의 형태로 나타낸다.

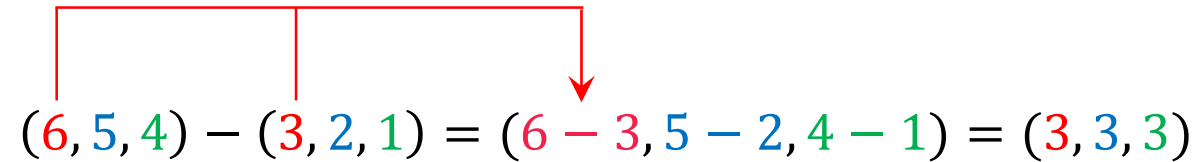
Ex). (1, 2) (1, 2, 3)



벡터 연산: 덧셈과 뺄셈

Ex).


$$(6, 5, 4) + (3, 2, 1) = (6 + 3, 5 + 2, 4 + 1) = (9, 7, 5)$$


$$(6, 5, 4) - (3, 2, 1) = (6 - 3, 5 - 2, 4 - 1) = (3, 3, 3)$$

Numpy 배열의 연산 : 선형 대수학

벡터 연산: 원소별 곱셈

=> '*' 연산자는 원소별 곱셈을 의미한다.

=> 선형대수학의 “내적”이 **아니다**.

Ex).

$$(6, 5, 4) * (3, 2, 1) = (6 \times 3, 5 \times 2, 4 \times 1) = (18, 10, 4)$$

Numpy 배열의 연산 : 선형 대수학

벡터 연산: 내적

=> 선형 대수학의 내적은 다음과 같은 연산이다.

Ex).

$$(6, 5, 4) \cdot (3, 2, 1) = 6 \times 3 + 5 \times 2 + 4 \times 1 = 32$$

=> NumPy의 dot() 메서드 또는 함수를 사용해서 계산한다.

벡터 연산: 스칼라로 곱하기 또는 나누기

Ex).

$$2 \times (1, 2, 3) = (2 \times 1, 2 \times 2, 2 \times 3) = (2, 4, 6)$$

$$(2, 4, 6) / 2 = (2 / 2, 4 / 2, 6 / 2) = (1, 2, 3)$$

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 벡터의 연산

```
In[1] : x = np.array([1, 3, 5])           # 길이가 3인 배열.
In[2] : y = np.array([2, 4, 6])           # 길이가 3인 배열.
In[3] : x + y                             # 벡터 덧셈.
Out[3]: array([ 3, 7, 11])
In[4] : x - y                             # 벡터 뺄셈.
Out[4]: array([-1, -1, -1])
In[5] : x * y                             # 벡터 원소별 곱셈.
Out[5]: array([ 2, 12, 30])
In[6] : np.dot(x, y)                      # 내적.
Out[6]: 44
In[7] : x.dot(y) # y.dot(x)              # 내적.
Out[7]: 44
```

행렬:

=> 행렬은 자료를 행과 열로 나열해 놓은 일종의 배열이다:

Ex).

$$\begin{bmatrix} 3 \\ 7 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 \\ 7 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 5 & 2 \\ 7 & 6 & 4 \\ 8 & 0 & 10 \end{bmatrix}$$

=> “전치행렬”:

Ex).

$$\begin{bmatrix} 5 \\ 3 \\ 1 \end{bmatrix}^t = [5 \quad 3 \quad 1]$$

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}^t = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Numpy 선형 대수학: 행렬 만들기

```
In[1] : np.zeros((2,3))
```

0을 원소로 하는 행렬.

```
Out[1]:
```

```
array([[ 0.,  0.,  0.],  
       [ 0.,  0.,  0.]])
```

```
In[2] : np.ones((2,3))
```

1을 원소로 하는 행렬.

```
Out[2]:
```

```
array([[ 1.,  1.,  1.],  
       [ 1.,  1.,  1.]])
```

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬 만들기

In[1] : np.random.seed(n)

랜덤 시드 초기화 (n=seed).

In[2] : np.random.random((2,2))

0과 1사이 균등분포 랜덤 행렬. 튜플 두겹.

Out[2]:

array([[0.60050608, 0.0590288],
 [0.00072301, 0.72163516]])

가1

!!!

In[3] : np.random.randn(2,2)

정규분포 랜덤 행렬. 튜플 한겹.

Out[3]:

array([[-1.3059906 , 0.98549986],
 [-0.97344165, -0.89474788]])

np.random.seed(222)

np.random.random((2,2))

#0 1

#

(0~1)

(0,1,2,3~) 2가

0~1

- (

0 -

- (

)

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬 만들기

```
In[1] : m = np.diag([1,2,3])
```

대각 행렬.

가 가

```
In[2] : m
```

Out[2]:

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

```
In[3] : np.diag(m)
```

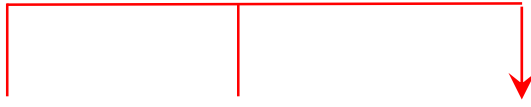
행렬의 대각선상의 원소 배열.

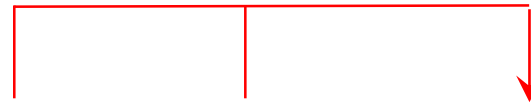
Out[3]: array([1, 2, 3])

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬의 연산 (+, -)

가


$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$


$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} - \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1-5 & 2-6 \\ 3-7 & 4-8 \end{bmatrix} = \begin{bmatrix} -4 & -4 \\ -4 & -4 \end{bmatrix}$$

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬의 연산

In[1] : m1 = np.array([[1, 2, 3],[4, 5, 6]]) # 크기가 2 x 3인 행렬.

In[2] : m2 = np.array([[6, 5, 4],[3, 2, 1]]) # 크기가 2 x 3인 행렬.

In[3] : m1 + m2

Out[3]:

array([[7, 7, 7],
 [7, 7, 7]])

In[4] : m1 - m2

Out[4]:

array([[-5, -3, -1],
 [1, 3, 5]])

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬의 연산 (* 와 dot의 비교)

		*가			
				dot	
가		*			
	가			dot	-

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} * \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 1 \times 7 & 2 \times 8 & 3 \times 9 \\ 4 \times 10 & 5 \times 11 & 6 \times 12 \end{bmatrix} = \begin{bmatrix} 7 & 16 & 27 \\ 40 & 55 & 72 \end{bmatrix}$$

~~$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} = [?????]$$~~

, 가 가

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬의 연산 (* 와 dot의 비교)

```
In[5] : m1 * m2
```

```
Out[5]:
```

```
array([[ 6, 10, 12],  
       [12, 10,  6]])
```

```
In[6] : np.dot(m1 , m2)
```

```
ValueErrorTraceback (most recent call last)
```

```
<ipython-input-263-321e200e8b3a> in <module>()
```

```
----> 1 np.dot(m1,m2)
```

```
ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```

사이즈 오류 발생!

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬의 연산 (* 와 dot의 비교)

$$\begin{aligned} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}^t \\ &= \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 & 10 \\ 8 & 11 \\ 9 & 12 \end{bmatrix} \quad \boxed{\text{가}} \\ &= \begin{bmatrix} 1 \times 7 + 2 \times 8 + 3 \times 9 & 1 \times 10 + 2 \times 11 + 3 \times 12 \\ 4 \times 7 + 5 \times 8 + 6 \times 9 & 4 \times 10 + 5 \times 11 + 6 \times 12 \end{bmatrix} \\ &= \begin{bmatrix} 50 & 68 \\ 122 & 167 \end{bmatrix} \end{aligned}$$

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬의 연산 (* 와 dot의 비교)

```
In[7] : np.transpose(m2)
```

2 x 3 행렬을 3 x 2로 변환 (전치 행렬).

```
Out[7]:
```

```
array([[6, 3],  
       [5, 2],  
       [4, 1]])
```

```
In[8] : np.dot(m1 , np.transpose(m2))
```

이제는 2 x 3 행렬과 3 x 2를 서로 곱할 수 있다.

```
Out[8]:
```

```
array([[28, 10],  
       [73, 28]])
```

결과는 2 x 2 행렬.

2*3 2*3

*

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬의 연산 (* 와 dot의 비교)

$$[1 \quad 2 \quad 3] \cdot \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1 \times 4 + 2 \times 5 + 3 \times 6 = 32$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \cdot [4 \quad 5 \quad 6] = \begin{bmatrix} 1 \times 4 & 1 \times 5 & 1 \times 6 \\ 2 \times 4 & 2 \times 5 & 2 \times 6 \\ 3 \times 4 & 3 \times 5 & 3 \times 6 \end{bmatrix} = \begin{bmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{bmatrix}$$

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬의 연산 (* 와 dot의 비교)

```
In[1] : x = np.array([1, 3, 5])
```

길이가 3인 배열.

```
In[2] : y = np.array([2, 4, 6])
```

길이가 3인 배열.

```
In[3] : x = x.reshape((3,1))
```

열 벡터 = 3x1 행렬.

```
In[4] : y = y.reshape((1,3))
```

행 벡터 = 1x3 행렬.

```
In[5] : y.dot(x)
```

행렬의 곱.

```
Out[5]: [[44]]
```

하나의 원소를 갖는 행렬.

```
In[6] : y.dot(x)[0,0]
```

```
Out[6]: 44
```

```
In[7] : x.dot(y)
```

행

```
Out[7]:
```

```
array([[ 2,  4,  6],
       [ 6, 12, 18],
       [10, 20, 30]])
```

x numpy y
numpy
x.shape 3

shape - 가
(3,), (,3)
x,y 가

가 가

reshape 가 ,

reshape
x.reshape((-1,1))/((-1,3))
reshape (

- 1 3 5 15가

44
44

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬의 연산 (스칼라와 행렬의 곱, 나누기)

$$3 \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 3 \times 1 & 3 \times 2 & 3 \times 3 \\ 3 \times 4 & 3 \times 5 & 3 \times 6 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} / 2 = \begin{bmatrix} 1/2 & 2/2 & 3/2 \\ 4/2 & 5/2 & 6/2 \end{bmatrix} = \begin{bmatrix} 0.5 & 1.0 & 1.5 \\ 2.0 & 2.5 & 3.0 \end{bmatrix}$$

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 행렬의 연산 (스칼라와 행렬의 곱, 나누기)

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

```
In[2] : 3 * m
```

스칼라와 행렬의 곱.

```
Out[2]:
```

```
array([[3, 6],  
       [9, 12]])
```

```
In[3] : m / 2.0
```

행렬을 스칼라로 나눔.

```
Out[3]:
```

```
array([[0.5, 1.0],  
       [1.5, 2.0]])
```

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 역행렬

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} -2 & 1 \\ 1.5 & -0.5 \end{bmatrix} = \begin{bmatrix} -2 + 3 & 1 - 1 \\ -6 + 6 & 3 - 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

\uparrow \uparrow \uparrow
 A A^{-1} I

역행렬

x가 0	가	가	1/3
x가			?
-	0		
-	가		

Numpy 배열의 연산 : 선형 대수학

Numpy 선형 대수학: 역행렬

```
In[1] : m = np.array([[1, 2],[3, 4]])           # 정사각형 shape의 행렬.
In[2] : minv = np.linalg.inv(m)                # m의 역행렬.
In[3] : minv
Out[3]:
array([[ -2. ,  1. ],
       [ 1.5, -0.5]])
In[4] : mres = np.dot(m, minv)                  # 행렬의 곱.
In[5] : np.round(mres,2)
Out[5]:
array([[1., 0.],
       [0., 1.]])
```

Numpy 배열의 연산 : 선형 대수학

연립 방정식:

1. **m**개의 방정식과 **m**원 일차 연립 방정식은 다음과 같다:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \dots a_{1m}x_m &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \dots a_{2m}x_m &= b_2 \\ \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 \dots a_{1m}x_m &= b_m \end{aligned}$$

2. 행렬과 벡터를 사용해서 표기하면 다음과 같다:

$$A$$

$$Ax = b$$

3. 해(x)는 다음과 같이 구한다:

$$x = A^{-1}b$$

가?

?

1. 5 , 가 3 ?
- ,
()

가

가

2. 5 5 1
3. 3 5 ?
-

?

?

가

가 (가 가?
- ()
가
()

Numpy 배열의 연산 : 선형 대수학

연립 방정식:

- 다음과 같이 두명의 작업자가 생산에 투입되는 상황을 가정해 본다:

→ x가 5시간 작업에 투입되고 y가 8시간 작업에 투입되니 생산량은 30이 된다:

$$5x + 8y = 30$$

→ x가 6시간 작업에 투입되고 y가 4시간 작업에 투입되니 생산량은 25이 된다:

$$6x + 4y = 25$$

→ 그러면 연립 일차 방정식은 다음과 같다:

$$5x + 8y = 30$$

$$6x + 4y = 25$$

→ 작업자들의 시간당 생산량을 구하라.

Numpy 배열의 연산 : 선형 대수학

연립 방정식:

가 가 (가)

```
In[1] : A = np.array([[5, 8],[6, 4]])
```

정사각형 shape의 행렬.

```
In[2] : b = np.array([[30], [25]])
```

```
In[3] : Ainv = np.linalg.inv(A)
```

A의 역행렬.

```
In[4] : np.dot(Ainv,b)
```

해.

Out[4]:

```
array([[ 2.85714286],  
       [ 1.96428571]])
```

x의 시간당 생산량.

y의 시간당 생산량.

```
In[5] : np.linalg.solve(A, b)
```

Numpy의 linalg.solve 메서드를 사용해서 구함.

Out[5]:

```
array([[ 2.85714286],  
       [ 1.96428571]])
```

?

가

(

)

실습 #0203

→ Numpy 행렬과 선형대수학에 대해서 알아봅니다. ←

→ 사용: [ex_0203.ipynb](#) ←

2. 파이썬 라이브러리 활용 분석.

2.1. Numpy 패키지 기초.

2.2. Numpy 연산.

2.3. Numpy 선형 대수학.

2.4. Matplotlib 패키지를 활용한 시각화.

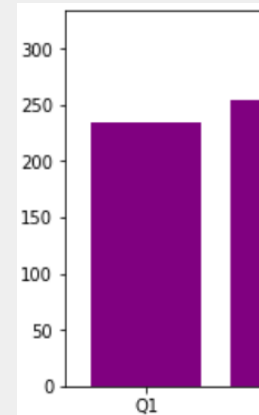
2.5. Pandas 데이터 프레임과 시리즈.

2.6. Pandas 데이터 가공.

Matplotlib 패키지 : 바차트

Bar chart:

```
In[1] : import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
x = np.array(['Q1', 'Q2', 'Q3','Q4'])
y = np.array([ 234.0, 254.7, 144.6, 317.6])
plt.bar(x,y,color = 'purple')
plt.show()
```



boxplot

, boxplot

가?

EDA - -

EDA

- , ,

EDA 가

-

1.

- 가 (a,b,c) 가

-

가

가 1 () bar

가 2 () ab

()

가

가 1

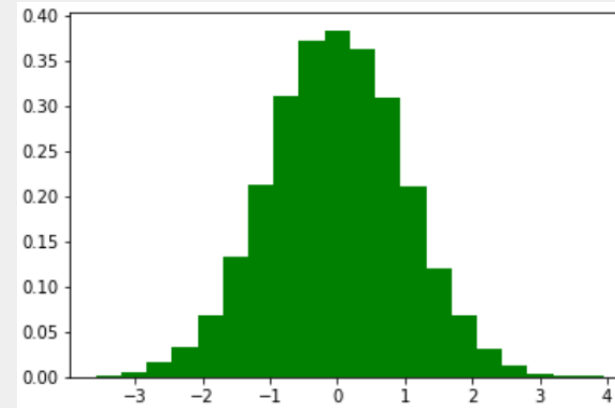
Matplotlib 패키지 : 히스토그램

Histogram:

```
In[1] : x = np.random.randn(10000)
plt.hist(x, bins=20, color='green', density=True)
plt.show()
```

, 가
1 가

1. 가 .
2. 가 .
3. .

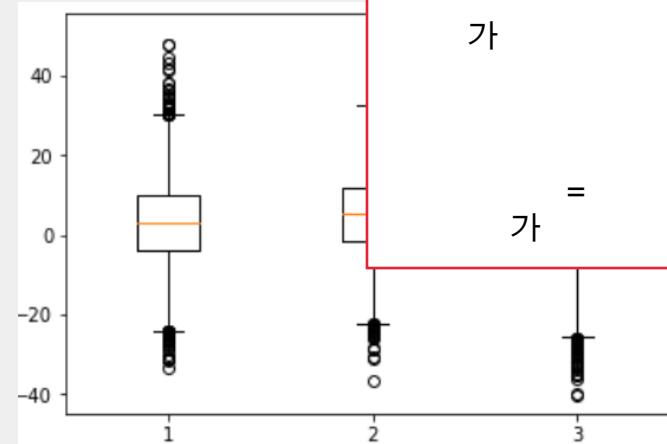


- ?
1. 가 ,
 2. ,
= (,
)

Box plot:

```
In[1] : x = np.random.randn(10000)*10+3
        y = np.random.randn(10000)*10+5
        z = np.random.randn(10000)*10+1
        plt.boxplot([x, y, z], 0)
        plt.show()
```

- 가
가



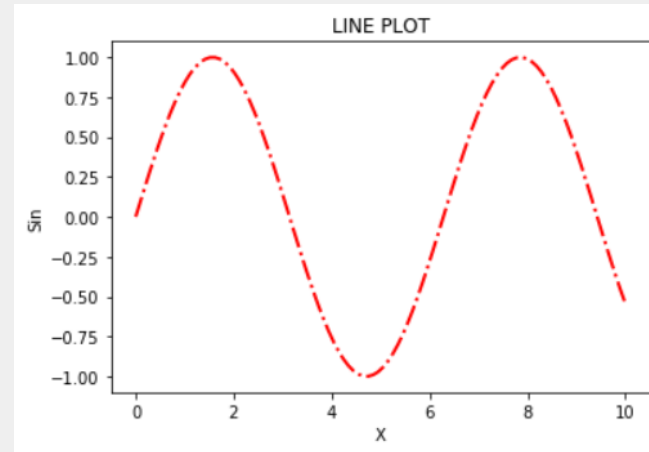
1 1
,
가 ?
가 []
25,50,75 (4 4)
가
=
가

Matplotlib 패키지 : 라인

Line plot:

```
In[1] : x = np.linspace(0,10,100)
        y = np.sin(x)
        plt.plot(x,y,color='red',linestyle='-.',linewidth=2)
        plt.xlabel('X')
        plt.ylabel('Sin')
        plt.title('LINE PLOT')
        plt.show()
```

가 ,



Matplotlib 패키지 : plot 메서드의 인자

plot 메서드의 인자:

인자	설명
color	컬러.
alpha	투명도.
linewidth	선 굵기.
linestyle	선 스타일.
marker	마커 타입.
markersize	마커 크기.
markerfacecolor	마커 내부 컬러.
markeredgecolor	마커 테두리선 컬러.
markeredgewidth	마커 테두리선 굵기.

Matplotlib 패키지 : 라인스타일

라인스타일 유형:

linestyle	설명
'none'	라인 없음.
'.'	점선.
'--'	단속 선.
'-.'	선과 점.
'--'	연속선.
'steps'	스텝.

Matplotlib 패키지 : 마커 타입

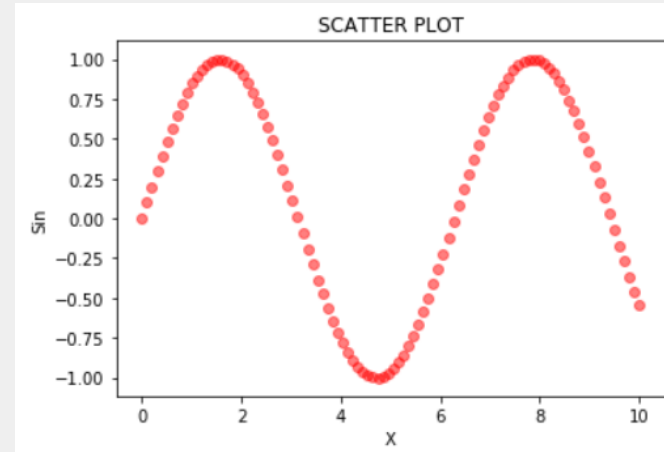
마커 타입:

marker	설명
'.'	점.
','	픽셀.
'o'	원.
'^'	삼각형.
'v'	역삼각형.
's'	사각형.
'*'	별표.
'+'	플러스.
'x'	엑스.
'D'	다이아몬드 (마름모).
'p'	정오각형.

Matplotlib 패키지 : 산점도

Scatter plot:

```
In[1] : x = np.linspace(0,10,100)
        y = np.sin(x)
        plt.scatter(x,y,c='red',marker='o',alpha=0.5)
        plt.xlabel('X')
        plt.ylabel('Sin')
        plt.title('SCATTER PLOT')
        plt.show()
```



실습 #0204

→ Matplotlib 패키지를 활용한 시각화에 대해서 알아본다. ←

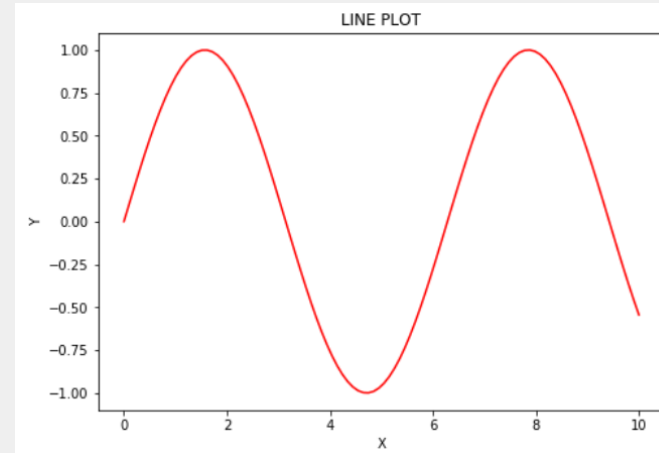
→ 사용: [ex_0204.ipynb](#) ←

Matplotlib 패키지 : 객체 지향형 시각화

객체 지향형 시각화:

```
In[1] : fig=plt.figure()           0 ,0           ,           1=100,           1=100
        axes = fig.add_axes([0,0,1,1])           # Left, Bottom, Width, Height
        axes.plot(x,y,color='red',linestyle='-')
        axes.set_xlabel('X')
        axes.set_ylabel('Y')
        axes.set_title('LINE PLOT')
        plt.show()
```

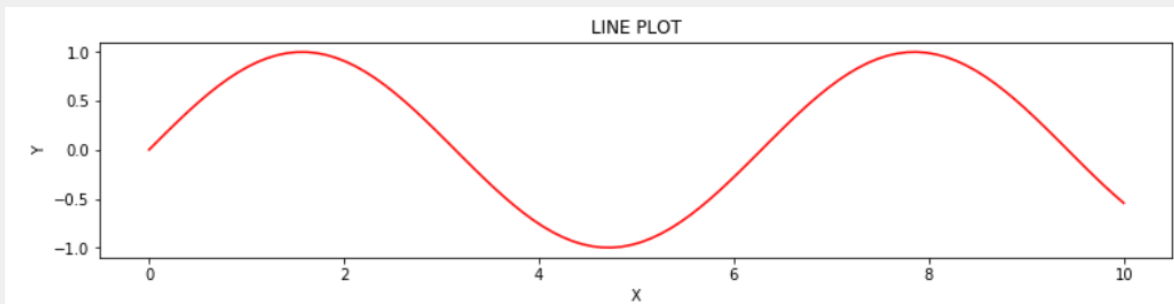
- 1.
2. ()



Matplotlib 패키지 : 객체 지향형 시각화

figsize 인자를 통한 비율 조정:

```
In[1] : fig=plt.figure(figsize=(10,2))      # 가로, 세로 크기.  
        axes = fig.add_axes([0,0,1,1])      # Left, Bottom, Width, Height  
        axes.plot(x,y,color='red',linestyle='-')  
        axes.set_xlabel('X')  
        axes.set_ylabel('Y')  
        axes.set_title('LINE PLOT')  
        plt.show()
```



Matplotlib 패키지 : 객체 지향형 시각화

하나의 패널에 다중 시각화 (범례 사용):

axis -
axes - (가 ,)

```
In[1] : fig=plt.figure(figsize=(8,5), dpi=100)
```

가로, 세로 크기. DPI 세팅.

```
axes = fig.add_axes([0,0,1,1])
```

Left, Bottom, Width, Height

```
axes.plot(x,y,color='red',linestyle=':', label='Sin')
```

범례를 위한 label 인자.

```
axes.plot(x,z,color='blue',linestyle='-.', label='Cos')
```

범례를 위한 label 인자.

```
axes.legend(loc=0)
```

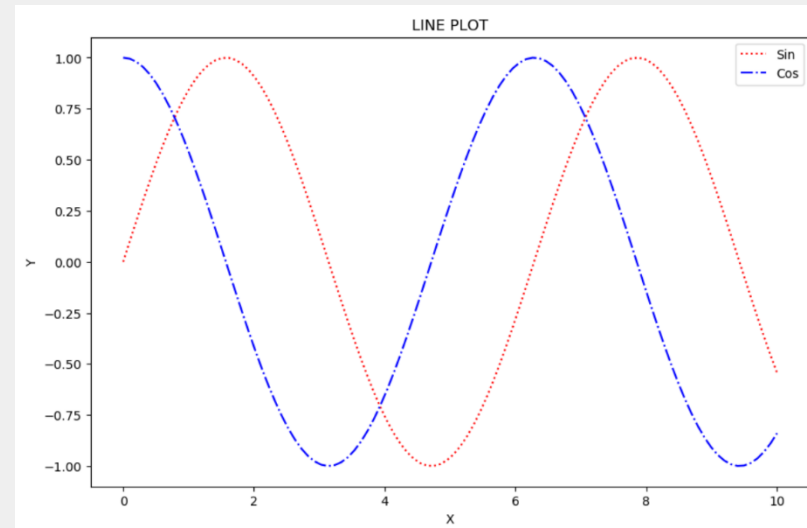
Top-Right에 범례 출력.

```
axes.set_xlabel('X')
```

```
axes.set_ylabel('Y')
```

```
axes.set_title('LINE PLOT')
```

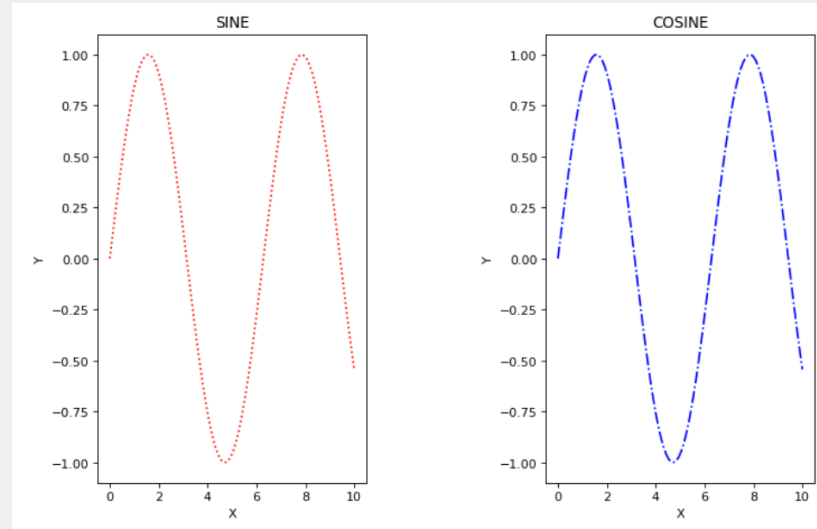
```
plt.show()
```



Matplotlib 패키지 : 객체 지향형 시각화

여러개의 패널이 있는 다중 시각화 1:

```
In[1] : fig=plt.figure(figsize=(10,5), dpi=80)      # 가로, 세로 크기. DPI 세팅.
        axes1 = fig.add_axes([0,0,0.3,1])           # Left, Bottom, Width, Height
        axes2 = fig.add_axes([0.5,0,0.3,1])         # Left, Bottom, Width, Height
        axes1.plot(x,y,color='red',linestyle=':')
        axes2.plot(x,y,color='blue',linestyle='-.')
        axes1.set_xlabel('X')
        axes1.set_ylabel('Y')
        axes1.set_title('SINE')
        axes2.set_xlabel('X')
        axes2.set_ylabel('Y')
        axes2.set_title('COSINE')
        plt.show()
```



Matplotlib 패키지 : 객체 지향형 시각화

여러개의 패널이 있는 다중 시각화 2:

()

```
In[1] : fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10,10))
```

```
# (0,0)
```

```
(0,0) axes[0,0].plot(x,y,color='green',linestyle=':')
```

```
axes[0,0].set_xlabel('X')
```

```
axes[0,0].set_ylabel('Y')
```

```
axes[0,0].set_title('SINE')
```

```
# (0,1)
```

```
:
```

```
:
```

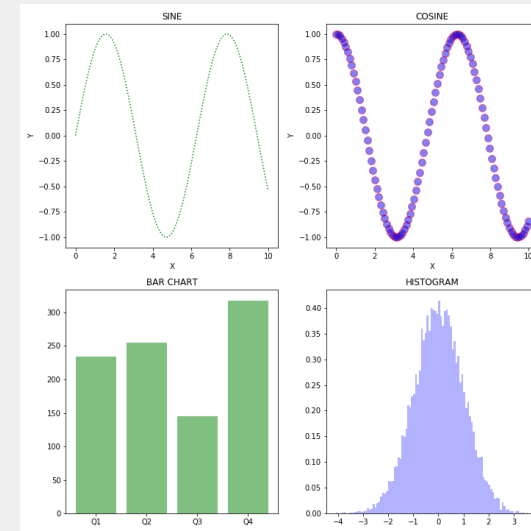
```
:
```

```
:
```

```
plt.tight_layout()
```

서로 겹치는 현상을 피함.

```
plt.show()
```



실습 #0205

→ Matplotlib 패키지를 활용한 시각화에 대해서 조금 더 깊이 알아본다. ←

→ 사용: [ex_0205.ipynb](#) ←

2. 파이썬 라이브러리 활용 분석.

2.1. Numpy 패키지 기초.

2.2. Numpy 연산.

2.3. Numpy 선형 대수학.

2.4. Matplotlib 패키지를 활용한 시각화.

2.5. Pandas 데이터 프레임과 시리즈.

2.6. Pandas 데이터 가공.

구조화 데이터 vs 비구조화 데이터:

- 비구조화 데이터: 스크레이핑 방법으로 내려받은 인터넷 데이터, 로그 파일, 등.
- 구조화 데이터: CSV 파일, 엑셀 파일, SQL 테이블, 등.
가

Pandas 패키지 : 특징

Pandas 패키지의 특징:

- Pandas 패키지는 Numpy 패키지를 바탕으로 만들어짐 ⇒ 함수의 호환성.
- Pandas는 Series와 DataFrame 객체를 다루는 목적으로 특화됨.
()
- 통계, 결측치 처리, 시각화 등 많은 기능이 있음.

시리즈 (Series):

- 1차원 Numpy 배열과 유사하다.
- index라는 속성이 있어서 인덱싱 목적으로 사용된다.
- 벡터 연산도 지원한다. `dot` `numpy` ,

가

가

Pandas 시리즈: 기초

가

```
In[1] : type(df)
Out[1]: pandas.core.frame.DataFrame # 데이터 프레임 객체.
In[2] : type(df.a) df.
Out[2]: pandas.core.series.Series # 시리즈 객체.
In[3] : my_data = np.array([220, 215, 93,64])
In[4] : eye = pd.Series(data=my_data, index=['Brown','Blue','Hazel','Green']) # 시리즈 생성.
In[5] : eye
Out[5]:
Brown 220
Blue 215
Hazel 93
Green 64
dtype: int32
```

Pandas 패키지 : 시리즈

Pandas 시리즈: 기초

```
In[1] : ser.namea          ser.name    a          # Name 속성.  
In[2] : ser.index          # Index 속성.  
In[3] : ser.values         # Series를 NumPy 배열로 출력.
```

```
In[1] : ser.sort_values()   sorting          # Series values를 소팅.  
In[2] : ser.sort_index()    sorting          # Series indices를 소팅.  
In[3] : ser.nunique()       가    (4)          가  
In[4] : ser.unique()        (a,b,ab,o)  
In[5] : ser.value_counts()  가          # 도수분포표.          가
```

(,) value

실습 #0206

→ Pandas의 시리즈에 대해서 알아본다. ←

→ 사용: [ex_0206.ipynb](#) ←

데이터 프레임 (DataFrame):

- 행렬과도 유사한 2D 객체이다. 하지만 개개 열의 자료형이 서로 일치하지 않을수도 있다.
- CSV 파일, 엑셀 파일, SQL 테이블 등의 형식의 데이터를 담기에 적합하다.
- Pandas의 데이터 프레임에는 columns (열)과 index (행)의 속성이 있음.

Pandas 패키지 : 데이터 프레임

Pandas 데이터 프레임 기초:

```
In[1] : import pandas as pd                # 패키지를 불러옴.
In[2] : df = pd.read_csv('my_file.csv', header='infer', encoding='ISO-8859-1')
In[3] : type(df)
Out[3]: pandas.core.frame.DataFrame
In[4] : df.info()                          # 데이터 프레임의 구조.
In[5] : df.head(n)                         # 데이터 프레임의 상단 n 행 보여줌.
In[6] : df.tail(n)                        # 데이터 프레임의 하단 n 행 보여줌
In[7] : df.columns                        (      ) # 데이터 프레임의 헤더를 보여줌.
In[8] : df.columns = ['A', 'B', 'C',...]  # 헤더의 이름을 바꾼다.
In[9] : header = df.columns               가 # 헤더의 이름을 저장.
In[10] : X = np.array(df)                 # 값은 Numpy array로 별도 저장.
```

Pandas 패키지 : 데이터 프레임

Pandas 데이터 프레임 슬라이싱:

```

In[11] : df.A           # 컬럼 A를 뽑아서 보여줌.
In[12] : df.B           # 컬럼 B를 뽑아서 보여줌.
In[13] : df.loc[:, ['A','B']] a      가      # 컬럼 A와 B를 뽑아서 보여줌.
In[14] : df.iloc[:, [0,1]]           :      # 컬럼 0과 1을 뽑아서 보여줌.
In[15] : df.loc[n]         가      # n번째 행을 보여줌.
In[16] : df.iloc[n]        # n번째 행을 보여줌.
In[17] : df.loc[n:m]       가      # n에서 m 번째 행을 보여줌.
In[18] : df.iloc[n:m]      # n에서 m-1 번째 행을 보여줌.
In[19] : df.drop(columns=['B','C'])   # 컬럼 B와 C 를 제외한 나머지.
In[20] : df.loc[:, (header != 'B' ) & (header != 'C' )] # 컬럼 B와 C 를 제외한 나머지.
In[21] : df.loc[:, (header == 'A' ) | (header == 'B' )] # 컬럼 A와 B 를 뽑아서 보여줌.

```

가 b,c가

df=

inplace=True

?

1000

- 999

- 1000

2

998

가

Pandas 패키지 : 데이터 프레임

Pandas 데이터 프레임 생성: 딕셔너리 사용

가

```
In[1] : data = { 'NAME' : ['Jake', 'Jennifer', 'Paul', 'Andrew'], 'AGE': [24,21,25,19], 'GENDER':['M','F','M','M']}
```

```
In[2] : df = pd.DataFrame(data)
```

```
In[3] : df
```

Out[3]:

	AGE	GENDER	NAME
--	-----	--------	------

0	24	M	Jake
---	----	---	------

가

1	21	F	Jennifer
---	----	---	----------

가 (가) -

2	25	M	Paul
---	----	---	------

3	19	M	Andrew
---	----	---	--------

```
In[4] : df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D', 'E']) # 랜덤 데이터 프레임.
```

10 5

5

가 가

Pandas 패키지 : 데이터 프레임

Pandas 데이터 프레임 조건부 슬라이싱:

가 ,

```
In[1] : df[ df.GENDER == 'M' ] # 성별이 M인 경우만 가져온다.
In[2] : df[ -(df.GENDER == 'M') ] != # 성별이 M이 아닌 경우만 가져온다.
In[3] : df[ df.HEIGHT > 170 ] .loc # 신장이 170 이상인 경우만 가져온다.
In[4] : df[ (df.HEIGHT > 170) & (df.HEIGHT < 180) ] # AND 조건의 조합.
In[5] : df[ (df.GENDER == 'M' ) & (df.HEIGHT < 180) ] # AND 조건의 조합.
In[6] : df[ (df.HEIGHT < 160) | (df.HEIGHT > 180) ] # OR 조건의 조합.
In[7] : df[ (df.GRADE == 1 ) | (df.GRADE == 4 ) ] # OR 조건의 조합.
In[8] : df[ (df.GENDER == 'M' ) & ((df.HEIGHT < 160) | (df.HEIGHT > 180)) ] # AND 와 OR 조건의 조합.
```

데이터 테이블의 결합 SQL

TABLE A

이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

TABLE B

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

데이터 테이블의 결합

TABLE A

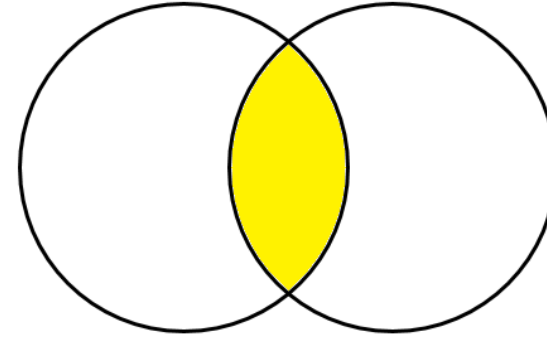
이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

TABLE B

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

데이터 테이블의 결합 : Inner Join

조건: A.이름 = B.직원이름



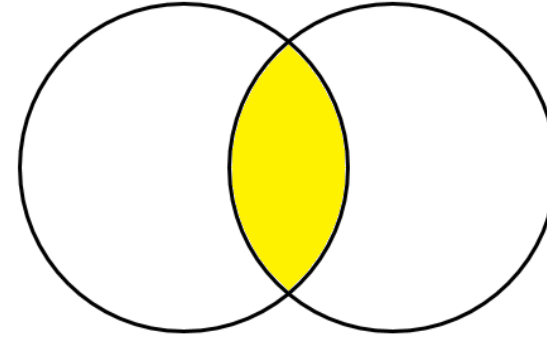
이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

데이터 테이블의 결합 : Inner Join

조건: A.이름 = B.직원이름

가

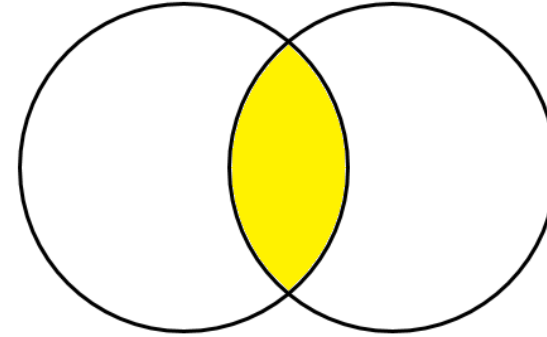


이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

데이터 테이블의 결합 : Inner Join

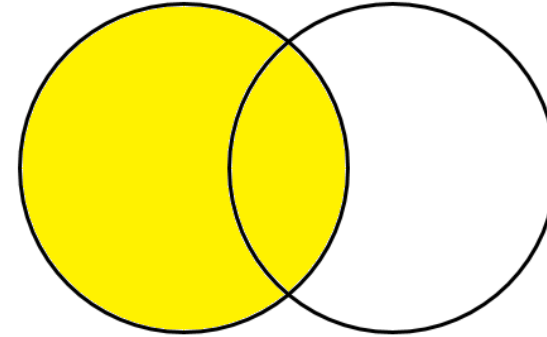
조건: A.이름 = B.직원이름



이름	성별	나이	직원이름	직책	시급
홍길동	남	28	홍길동	직원	9000
사임당	여	30	사임당	과장	20000

데이터 테이블의 결합 : Left Join

조건: A.이름 = B.직원이름

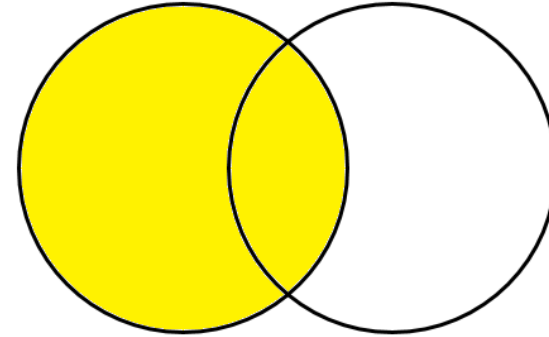


이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

데이터 테이블의 결합 : Left Join

조건: A.이름 = B.직원이름

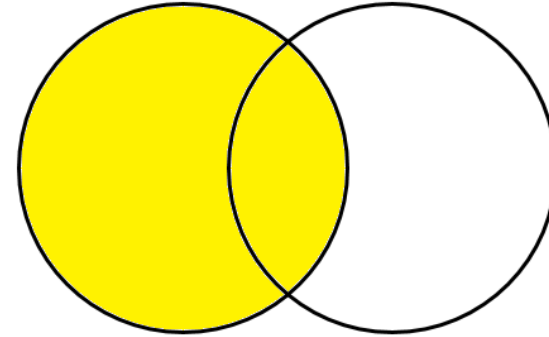


이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

데이터 테이블의 결합 : Left Join

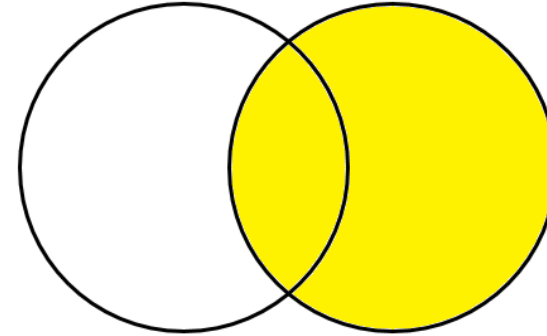
조건: A.이름 = B.직원이름



이름	성별	나이	직원이름	직책	시급
김철수	남	23			
이민수	남	31			
홍길동	남	28	홍길동	직원	9000
임꺽정	남	36			
사임당	여	30	사임당	과장	20000

데이터 테이블의 결합 : Right Join

조건: A.이름 = B.직원이름



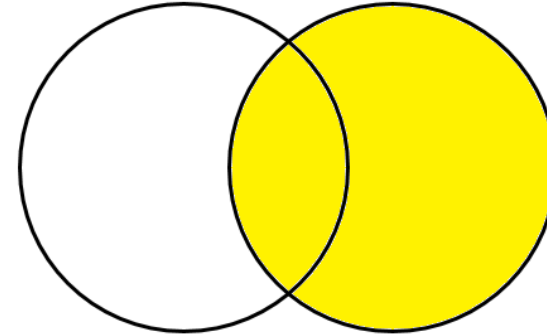
이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

데이터 테이블의 결합 : Right Join

조건: A.이름 = B.직원이름

가
가

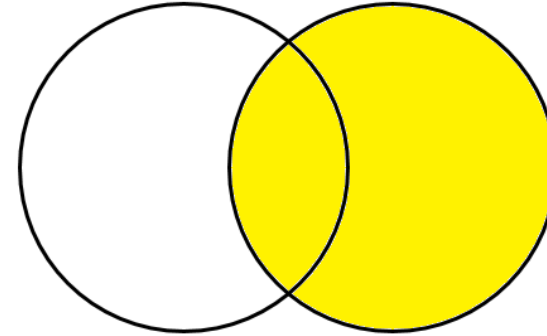


이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

데이터 테이블의 결합 : Right Join

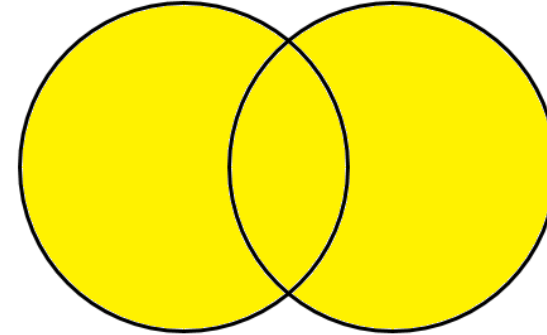
조건: A.이름 = B.직원이름



이름	성별	나이	직원이름	직책	시급
홍길동	남	28	홍길동	직원	9000
			이세종	직원	9500
			대장금	과장	19000
사임당	여	30	사임당	과장	20000

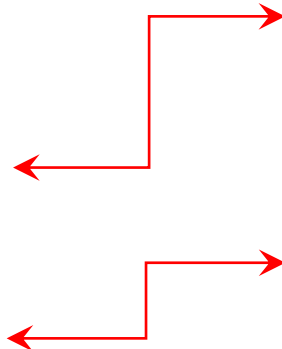
데이터 테이블의 결합 : Full Outer Join

조건: A.이름 = B.직원이름



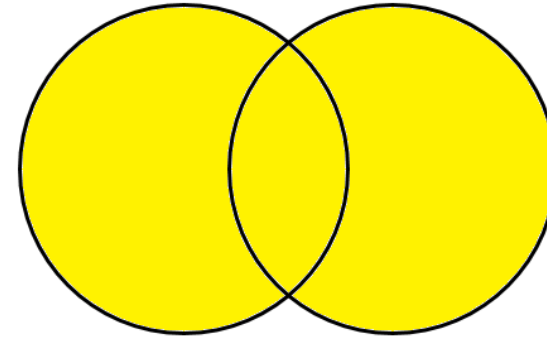
이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000



데이터 테이블의 결합 : Full Outer Join

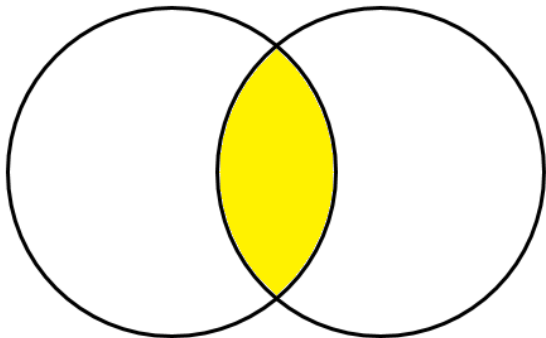
조건: A.이름 = B.직원이름



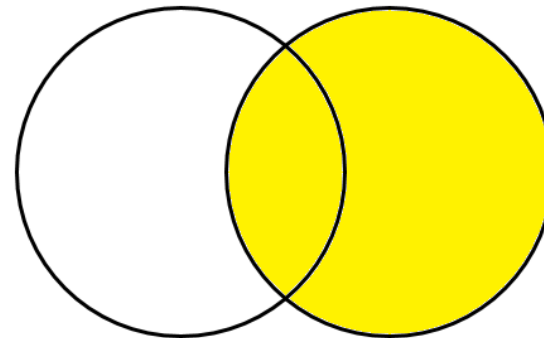
이름	성별	나이	직원이름	직책	시급
김철수	남	23			
이민수	남	31			
홍길동	남	28	홍길동	직원	9000
임꺽정	남	36			
사임당	여	30	사임당	과장	20000
			이세종	직원	9500
			대장금	과장	19000

데이터 테이블의 결합 : 정리

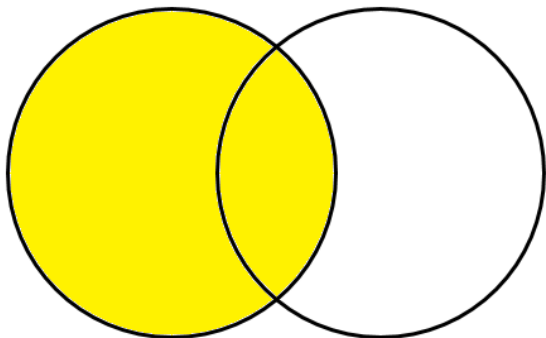
Inner Join



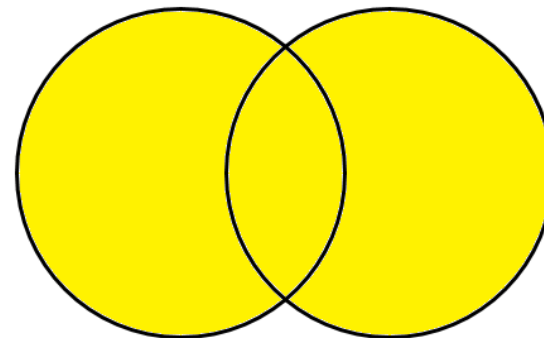
Right Join



Left Join



Full Outer Join



실습 #0207

→ Pandas 데이터 프레임에 대해서 알아봅니다. ←

→ 사용: [ex_0207.ipynb](#) ←

Mini-Project #0201

mini-project_0401.ipynb의 물음에 답하십시오.

참고: [**sol_0201.ipynb**](#)

Mini-Project #0202

mini-project_0402.ipynb의 물음에 답하십시오.

참고: [sol_0202.ipynb](#)

2. 파이썬 라이브러리 활용 분석.

2.1. Numpy 패키지 기초.

2.2. Numpy 연산.

2.3. Numpy 선형 대수학.

2.4. Matplotlib 패키지를 활용한 시각화.

2.5. Pandas 데이터 프레임과 시리즈.

2.6. Pandas 데이터 가공.

Pandas 패키지 : 데이터 프레임

Pandas 데이터 프레임: 메서드로 제공되는 통계 함수

In[1] : df.sum(axis=0)	가 1 가	가	# 개개 열을 따라서 더함.
In[2] : df.sum(axis=1)	가 True	가	# 개개 행을 따라서 더함.
In[3] : df.mean(axis=0, skipna=False)			# 열 평균을 구하는데 NA를 떨구지 않음.
In[4] : df.describe()	,		# 사분위수 등 기술통계 요약을 보여줌.
In[5] : df.count(axis=0)		가	# NA가 아닌 값의 갯수.
In[6] : df.A.corr(df.B)	가		# A 컬럼과 B 컬럼 사이의 상관계수.
In[7] : df.corr()			# 상관계수 행렬을 계산한다.
In[8] : df.corrwith(df.A)			# A와 나머지 변수 사이의 상관계수. a

가

axis

axis

Pandas 패키지 : 데이터 프레임

Pandas 데이터 프레임: 결측치 NA관련 메서드

```
na_values=[' ']  
In[1] : df.isnull() # NA인 위치에는 True인 데이터 프레임.  
In[2] : (df.isnull()).sum(axis=0) # 컬럼 별 결측치의 갯수. TF sum / T 1 F 0  
In[3] : (df.isnull()).mean(axis=0) (0,1) # 컬럼 별 결측치의 비중. na  
In[4] : df.dropna(axis = 0) # NA가 포함된 행은 drop 한다.  
In[5] : df.dropna(axis = 1) 가 # NA가 포함된 열은 drop 한다.  
In[6] : df.dropna(axis=0, thresh = 3) # 최소 3개 이상 정상값이 있는 행은 제외하고 drop. 가  
In[7] : df.fillna(value=0) # 결측치를 0으로 채워 넣는다.
```

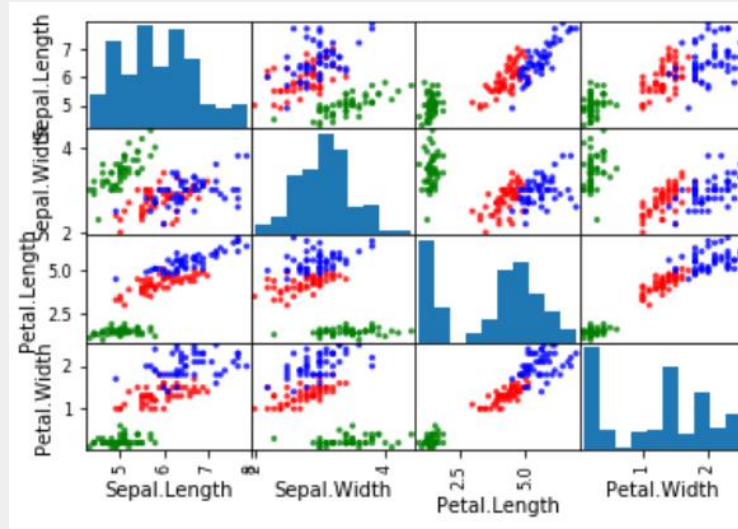
- inplace = T

가 ()
- - 0 ~3 ~
? ?
, 가
가? 가 ()
in[3]

Pandas 패키지 : 시각화

Pandas 데이터 프레임: 시각화

```
In[1] : df.plot(x = 'A' , y = 'B' )
In[2] : df.plot.scatter(x = 'A' , y = 'B' )
In[3] : my_cols_dict = {'setosa':'red', 'virginica':'green', 'versicolor':'blue'}
        my_cols = df0['Species'].apply(lambda x: my_cols_dict[x])
        pd.plotting.scatter_matrix(df, c=my_cols, marker='o', alpha=0.5)
        plt.show()
```



apply

가

A 대 B 라인 플롯.

A 대 B 산점도.

Species 유형을 컬러로 번역. species

산점도 행렬.

가 ?

가

apply
apply

실습 #0208

→ Pandas의 통계 함수, 결측치 처리, 시각화에 대해서 알아봅니다. ←

→ 사용: [ex_0208.ipynb](#) ←

실습 #0209

→ Pandas의 시각화 응용 예. ←

→ 사용: [ex_0209.ipynb](#) ←

실습 #0210

→ Pandas의 시각화 또 다른 응용 예. ←

→ 사용: [ex_0210.ipynb](#) ←

Mini-Project #0203 : 전국 커피숍 규모 파악

1. “data_coffee.csv” 데이터를 읽어옵니다.
2. 매장의 규모(sizeOfsite) 변수를 요약해 봅니다.
3. sizeOfsite의 결측치(NA)를 제거하고 히스토그램을 그려봅니다.
4. 매장 규모가 500이상인 outlier를 제외하고 히스토그램을 그려봅니다.

참고: [sol_0203.ipynb](#)

Mini-Project #0204 : 판매 실적

1. “data_officesupplies.csv” 데이터를 읽어옵니다.
2. Region을 West, Central, East로 구별하여 판매 수량 (Units)를 다중 박스플롯으로 출력한다.

참고: [sol_0204.ipynb](#)

Pandas 패키지 : 멀티 인덱스

Pandas 멀티 인덱스:

1 123, 2 123 2

```
In[1] : my_header = ['a','b','c']
        my_index_out = ['G1']*3 + ['G2']*3
        my_index_in = [1,2,3]*2
        my_index_zipped = list(zip(my_index_out, my_index_in)) # 두 개의 리스트 기반으로 튜플 생성.
        my_index = pd.MultiIndex.from_tuples(my_index_zipped)
        df = pd.DataFrame(data=np.random.randn(6,3),index=my_index,columns=my_header)
```

In[2] : df

Out[2]:

		a	b	c
G1	1	0.708643	1.526325	-0.522276
	2	-0.112115	0.366355	-0.127317
	3	-0.020839	0.023037	0.167214
G2	1	-1.300622	-0.310416	-0.840097
	2	0.088279	-1.596302	1.367721
	3	-0.998479	-0.476471	-2.038437

Pandas 패키지 : 그루핑 후 연산

그루핑 후 연산:

```
In[1] : df.groupby('gender').mean()           # 성별 모든 변수의 개개 평균.
In[2] : df.groupby('gender')['height'].mean()  # 성별 신장 평균.
In[3] : df.groupby('gender')[['height','weight']].mean()  # 성별 신장, 체중 평균.
In[4] : df.groupby('gender')['height'].describe()  # 성별 신장 통계적 요약.
In[5] : df.groupby(['gender','bloodtype'])['height'].mean()  # 멀티 인덱싱된 시리즈!!
```

Out[5]:

gender	bloodtype	
F	A	172.450000
	AB	170.100000
	B	158.200000
	O	164.433333
M	A	165.700000
	AB	181.050000
	B	174.550000
	O	166.200000

Name: height, dtype: float64

[]
[]

apply 메서드:

```
In[1] : df['height'].apply(lambda x: x/100)
```

람다함수와의 조합 즐겨 사용.

Out[1]:

```
0      1.653
1      1.701
2      1.750
3      1.821
4      1.680
5      1.620
6      1.552
7      1.769
8      1.785
9      1.761
10     1.671
11     1.800
12     1.622
13     1.761
14     1.582
15     1.686
16     1.692
Name: height, dtype: float64
```

가

Pandas 패키지 : 정렬

정렬:

```
In[1] : df.sort_values(by='bloodtype')           # 데이터 프레임 정렬.
In[2] : df.sort_values(by='bloodtype', ascending=False)      (          )
In[3] : df.sort_values(by=['bloodtype','gender'])           , 2      ?
In[4] : sr.sort_values()                                   # 시리즈 값으로 정렬.
In[5] : sr.sort_index()                                   # 시리즈 인덱스로 정렬.
```

Pandas 패키지 : 명목형 변수 요약

명목형 변수 요약:

```
In[1] : df['bloodtype'].unique()      o,a,b,ab      # 고유한 값 (유형).
```

```
Out[1]:
```

```
array(['O', 'AB', 'B', 'A'], dtype=object)
```

```
In[2] : df['bloodtype'].nunique()      4가      # 유형의 가지수.
```

```
Out[2]:
```

```
4
```

```
In[3] : df['bloodtype'].value_counts()      가      # 도수 분포표.
```

```
Out[3]:
```

```
O 5
```

```
B 5
```

```
A 4
```

```
AB 3
```

```
Name: bloodtype, dtype: int64
```


Pandas 패키지 : 피보팅

피보팅:

```
In[1] : # A,B의 값으로 인덱스, C의 값으로 컬럼, 실제 셀에 들어가는 값은 E의 평균.  
# aggregate(E ~ A+B+C, data=df, mean)과 유사 (R).  
pd.pivot_table(df, index=['A','B'], columns='C', values='E')
```

Out[1]:

		C	
		large	small
A	B		
bar	one	6.0	8.0
	two	9.0	9.0
foo	one	4.5	2.0
	two	NaN	5.5

a, b, c

e

aggregate

aggregate - apply

가

피보팅:

In[2] : # A,B의 값으로 인덱스, C의 값으로 컬럼, 실제 셀에 들어가는 값은 E의 중앙값.

aggregate(E ~ A+B+C, data=df, median)과 유사 (R).

pd.pivot_table(df, index=['A','B'], columns='C', values='E', **aggfunc=np.median, fill_value=0**)

Out[2]:

		C	
		large	small
A	B		
bar	one	6.0	8.0
	two	9.0	9.0
foo	one	4.5	2.0
	two	0.0	5.5

피보팅:

```
In[3] : # D, E의 그룹평균.      a,b      d      , e
        # aggregate( cbind(D, E) ~ A+B, data=df, mean)과 유사 (R).
        # df.groupby(['A','B'])[['D','E']].mean()과 동일 (Python, Pandas).
        pd.pivot_table(df, index=['A','B'], values=['D','E'], aggfunc=np.mean)
```

Out[3]:

		D	E
A	B		
bar	one	4.500000	7.000000
	two	6.500000	9.000000
foo	one	1.666667	3.666667
	two	3.000000	5.500000

ab

de

피보팅:

In[4] : # D, E를 다른 방법으로 집계함.

```
pd.pivot_table(df, index=['A','B'], values=['D','E'], aggfunc={'D':np.mean,'E':np.median})
```

Out[4]:

		D	E
A	B		
bar	one	4.500000	7.0
	two	6.500000	9.0
foo	one	1.666667	4.0
	two	3.000000	5.5

d , e

#group by
#

실습 #0211

→ 멀티 인덱스, groupby, apply, pivot_table 등에 대해서 알아본다. ←

→ 사용: [ex_0211.ipynb](#) ←

Mini-Project #0205 : 혈액형과 신장

1. “data_studentlist_en.csv” 데이터를 읽어온다.
2. 혈액형 별 신장을 구한다.
 - groupby 메서드 사용.
 - pivot_table 메서드 사용.

참고: [sol_0205.ipynb](#)

Mini-Project #0206 : 전국 인구조사 데이터

1. “data_population.csv” 데이터를 읽어온다.

읽어 올 때 빈 셀은 결측치 (NaN) 로 인식한다. HINT: na_values 인자.

1. 결측치가 포함된 행은 제거한다.

`na_values=[' ']`

2. 도별로 인구를 **합계**한다 (sum).

→ groupby 메서드 사용.

→ pivot_table 메서드 사용.

참고: [sol_0206.ipynb](#)

Mini-Project #0207 : 커피숍 영업 현황

1. “data_coffee.csv” 데이터를 읽어오시오.
2. 1997~2014년도 사이의 개업 횟수를 (yearOfStart) 도수분포표로 요약하고 시각화 하시오.
3. 운영 중인 커피숍 대상으로 1997~2014년도 사이의 개업 회수를 도수분포표로 요약하시오.
4. 지금은 폐업된 커피숍 대상으로 1997~2014년도 사이의 개업 회수를 도수분포표로 요약하시오.
5. 위 #3과 #4 결과를 하나의 그래프로 시각화한다. 범례도 포함한다.

참고: [sol_0207.ipynb](#)

문의:

sychang1@gmail.com