기계학습 (Machine Learning)

L04

Computational foundation

한밭대학교

정보통신공학과

최 해 철

국립 한밭대학교

VISUAL MEDIA LAB.
HANBAT NATIONAL UNIVERSITY

ToC

- Computational foundation
  - NumPy
  - (SciPy)
  - Matplotlib

NumPy

NumPy

- 파이썬의 수치 계산을 위한 패키지로 import 하여 사용
- Numerical Python의 줄임말
- 파이썬 기본 자료구조보다 실행 속도 빠름
  - C 언어를 사용하여 NumPy 핵심 모듈을 잘 최적화했기 때문
- 다른 데이터 사이언스 용 파이썬 패키지와 연계성 높음
  - Scipy, Pandas, Matplotlib, Scikit-learn 등의 패키지와 함께 쓰임

NumPy

◆ NumPy의 핵심은 ndarray 객체

- ndarray: fixed-size homogeneous multidimensional array
  - 고정된 크기의 동형 다차원 배열
  - 고정된 크기: 배열 생성할 때 크기 결정
  - 동형: 같은 type의 원소로 구성된 배열

◆ import

```
>>> import numpy as np
```

NumPy : N-dimensional Arrays

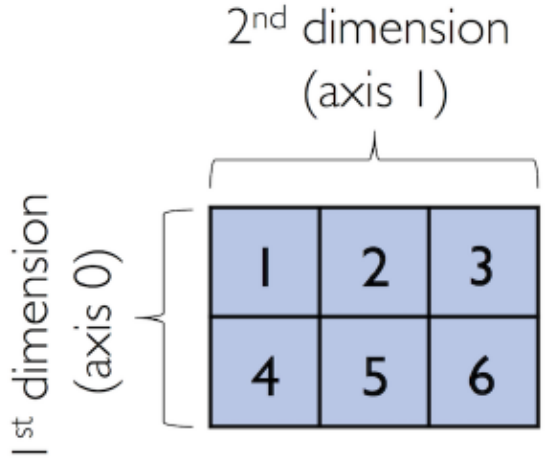
◆ ndarray 는 list, tuple 을 이용하여 생성 가능

```
>>> import numpy as np
>>> x = np.array((0.1,0.2,0.3))    # np.array([0.1,0.2,0.3])도 가능
>>> x
array([0.1, 0.2, 0.3])
>>> x.shape
(3,)
>>> x.dtype
dtype('float64')
```

NumPy : N-dimensional Arrays

◆ ndarray 는 list, tuple 을 이용하여 생성 가능

```
>>> y = np.array(((1,2,3),(4,5,6)))    # [(1,2,3),(4,5,6)],[[1,2,3],[4,5,6]] 등도 가능
>>> y
array([[1, 2, 3],
       [4, 5, 6]])
>>> y.dtype
dtype('int32')   # 정수형 기본타입: int32
>>> y.shape
(2, 3)
```



NumPy : Array Construction Routines

◆ 초기화

```
>>> np.ones((3, 3))

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

```
>>> np.zeros((3, 3))

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

```
>>> np.eye(3)

array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
>>> np.diag((3, 3, 3))

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

NumPy : Array Construction Routines

◆ 초기화

```
>>> np.arange(4., 10.)
array([4., 5., 6., 7., 8., 9.])
```

```
>>> np.arange(5)
array([0, 1, 2, 3, 4])
```

```
>>> np.arange(1., 11., 2)
array([1., 3., 5., 7., 9.])
```

```
>>> np.linspace(0., 1., num=5)
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

NumPy : Array Indexing

- NumPy indexing and slicing works similar to Python lists

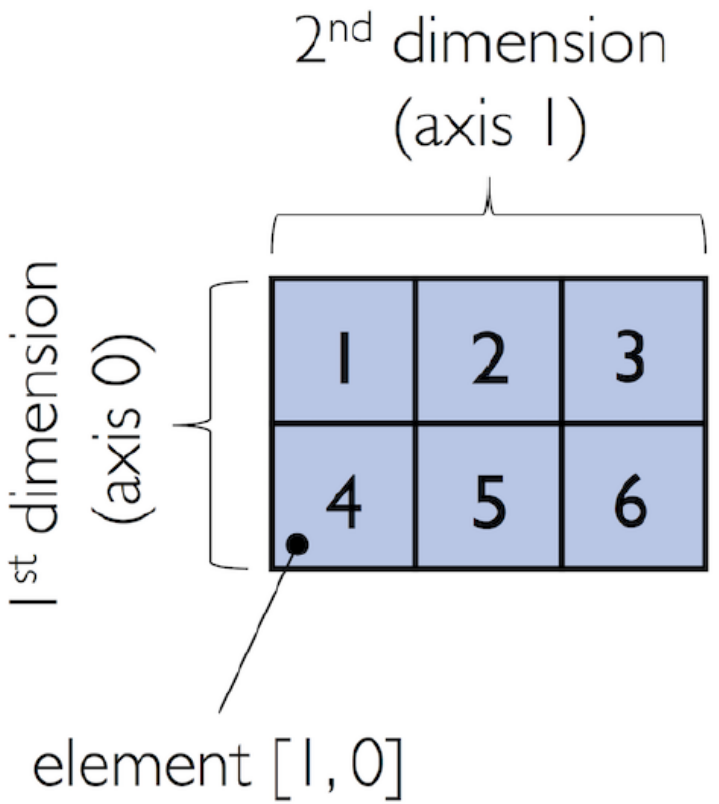
```
>>> ary = np.array([1, 2, 3])
>>> ary[0]

    1

>>> ary[:2] # equivalent to ary[0:2]

array([1, 2])
```

NumPy : Array Indexing

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

>>> ary[0, 0] # upper left
```

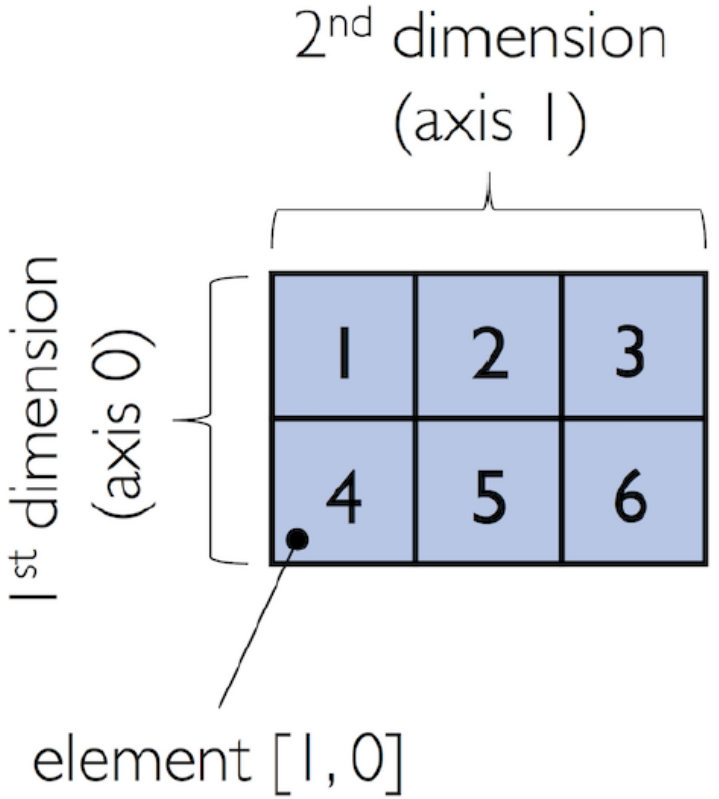
```
>>> ary[0, 1] # first row, second column
```

```
>>> ary[-1, -1] # lower right
```



NumPy : Array Indexing

```
>>> ary[0] # entire first row

array([1, 2, 3])

>>> ary[:, 0] # entire first column

array([1, 4])

>>> ary[:, :2] # first two columns

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



NumPy : Array Math and Universal Functions

◆ Vectorization

Mathematical operators (+, -, /, *, and **)

```
>>> ary = np.array([[1, 2, 3],
...                 [4, 5, 6]])
>>> ary + 1
```

2

```
array([[3, 4, 5],
       [6, 7, 8]])
```

```
>>> ary**2
```

```
array([[ 4,  9, 16],
       [25, 36, 49]])
```

```
>>> ary1 = np.array([1, 2, 3])
>>> ary2 = np.array([4, 5, 6])

>>> ary1 + ary2

array([5, 7, 9])
```

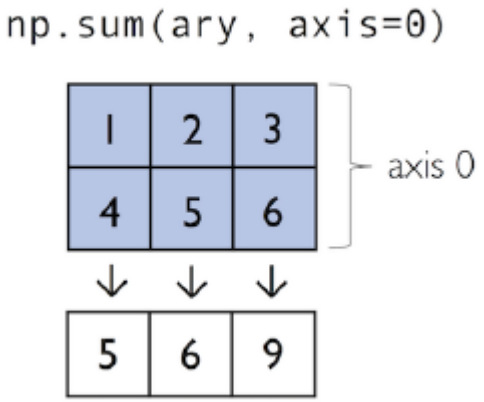
```
ary1 = np.array([1,2,3])
ary2 = np.array([4,5,6])
ary1*ary2
```

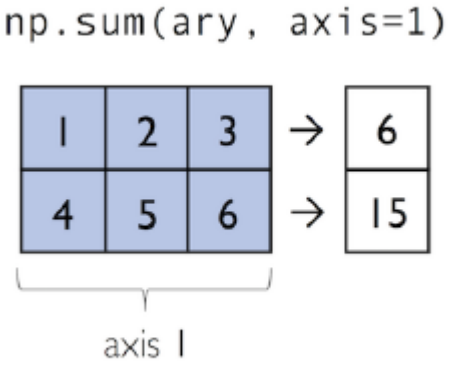
```
array([ 4, 10, 18])
```

NumPy : Array Math and Universal Functions

◆ Sum or product of array element along a given axis

```
>>> ary.sum(axis=0) # column sums

array([5, 7, 9])

>>> np.add.reduce(ary, axis=1) # row sums

array([ 6, 15])

>>> ary.sum()

21
```





NumPy : Array Math and Universal Functions

Other useful unary ufuncs are:

- mean (computes arithmetic average)
- std (computes the standard deviation)
- var (computes variance)
- np.sort (sorts an array)
- np.argsort (returns indices that would sort an array)
- np.min (returns the minimum value of an array)
- np.max (returns the maximum value of an array)
- np.argmin (returns the index of the minimum value)
- np.argmax (returns the index of the maximum value)
- array_equal (checks if two arrays have the same shape and elements)

```
ary = np.array([[1,2,3],[4,5,6]])
ary.mean(axis=0)
```

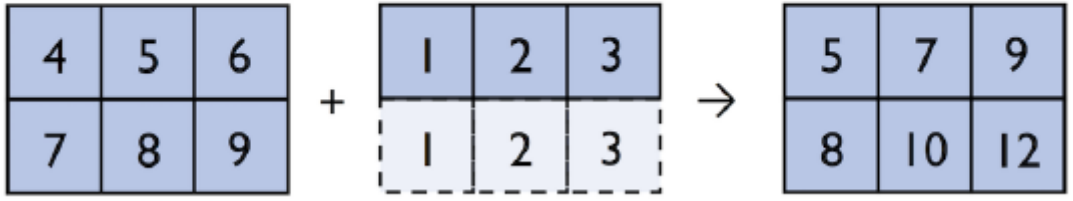
```
array([2.5, 3.5, 4.5])
```

NumPy : Broadcasting

- Broadcasting allows us to perform vectorized operations between two arrays even if their dimensions do not match
- Example of broadcasting.

```
np.array([1, 2, 3]) + 1:
```


```
np.array([[4, 5, 6],
          [7, 8, 9]]) + np.array([1, 2, 3]):
```



NumPy : Advanced Indexing - Memory Views and Copies

◆ 얕은 복사 (*view* of NumPy arrays in memory)

- 메모리 절약

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

>>> first_row = ary[0]
>>> first_row += 99
>>> ary

array([[100, 101, 102],
       [  4,   5,   6]])
```

View 생성

NumPy : Advanced Indexing - Memory Views and Copies

◆ 얕은 복사 (*view* of NumPy arrays in memory)

- Slicing creates views

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

>>> first_row = ary[:1]
>>> first_row += 99
>>> ary

array([[100, 101, 102],
       [  4,   5,   6]])
```

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

>>> center_col = ary[:, 1]
>>> center_col += 99
>>> ary

array([[  1, 101,   3],
       [  4, 104,   6]])
```

NumPy : Advanced Indexing - Memory Views and Copies

◆ 깊은 복사 (*copy* of an array)

- copy method 이용

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

>>> second_row = ary[1].copy()
>>> second_row += 99
>>> ary

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

```
ary1 = np.array([1,2,3])
ary2 = ary1
ary2 += 1
print(ary1)
print(ary2)
```

```
[2 3 4]
[2 3 4]
```

```
ary1 = np.array([1,2,3])
ary2 = ary1.copy()
ary2 += 1
print(ary1)
print(ary2)
```

```
[1 2 3]
[2 3 4]
```

NumPy : Advanced Indexing - Memory Views and Copies

◆ *Fancy* indexing

- non-contiguous integer indices

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

>>> ary[:, [0, 2]] # first and and last column


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

>>> ary[:, [2, 0]] # first and and last column


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

NumPy : Advanced Indexing - Memory Views and Copies

◆ *Fancy* indexing

- Since fancy indexing can be performed with non-contiguous sequences, it cannot return a view → 깊은 복사

```
>>> this_is_a_copy = ary[:, [0, 2]]
>>> this_is_a_copy += 99
>>> ary

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

NumPy : Advanced Indexing - Memory Views and Copies

◆ Boolean masks for indexing

- arrays of True and False values

```
ary = np.array([[10,2,3],[1,5,6],[4,1,1]])
greater3_mask = ary > 3
print(greater3_mask)
```

```
[[ True False False]
 [False  True  True]
 [ True False False]]
```

```
print(ary[greater3_mask])
```

```
[10  5  6  4]
```

NumPy : Advanced Indexing - Memory Views and Copies

◆ Boolean masks for indexing

```
ary = np.array([[10,2,3],[1,5,6],[4,1,1]])
print(ary[(ary>3) & (ary%2 == 0)])
```

```
[10  6  4]
```

NumPy : Random Number Generators

◆ a uniform distribution via `random.rand` in the half-open interval [0, 1).

```
>>> np.random.seed(123)
>>> np.random.rand(3)

array([0.69646919, 0.28613933, 0.22685145])
```

NumPy : Random Number Generators

◆ `random.seed()` : 전역적, 모든 NumPy의 난수 생성 함수가 동일한 시드를 사용

```
np.random.seed(123)
print(np.random.rand(3))
print(np.random.rand(3))
print(np.random.rand(3))
```

```
[0.69646919 0.28613933 0.22685145]
[0.55131477 0.71946897 0.42310646]
[0.9807642  0.68482974 0.4809319 ]
```

◆ `random.RandomState()` : 개별적인 인스턴스를 생성할 수 있으므로, 각 인스턴스는 독립적으로 시드를 설정하고 난수를 생성

```
rng1 = np.random.RandomState(seed=123)
rng2 = np.random.RandomState(seed=43)
print(rng1.rand(3))
print(rng1.rand(3))
print()
print(rng2.rand(3))
print(rng2.rand(3))
```

```
[0.69646919 0.28613933 0.22685145]
[0.55131477 0.71946897 0.42310646]

[0.11505457 0.60906654 0.13339096]
[0.24058962 0.32713906 0.85913749]
```

NumPy : Reshaping Array

◆ reshape()

- a view of an array with a different shape. → 얕은복사

```
>>> ary1d = np.array([1, 2, 3, 4, 5, 6])
>>> ary2d_view = ary1d.reshape(2, 3)
>>> ary2d_view

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

>>> np.may_share_memory(ary2d_view, ary1d)


True
```

NumPy : Reshaping Array

◆ reshape()

- **we do not need to specify the number elements in each axis**;
  - NumPy is smart enough to figure out how many elements to put along an axis if only one axis is unspecied (**by using the placeholder -1**):

```
>>> ary1d = np.array([1, 2, 3, 4, 5, 6])

>>> ary1d.reshape(2, -1)
array([[1, 2, 3],
       [4, 5, 6]])
```

```
>>> ary1d.reshape(-1, 2)
array([[1, 2],
       [3, 4],
       [5, 6]])
```

NumPy : Reshaping Array

◆ reshape()

- **We can, of course, also use reshape to flatten an array.**

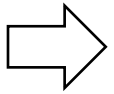
```
>>> ary = np.array([[[1, 2, 3],
...                  [4, 5, 6]]])

>>> ary.reshape(-1)


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

NumPy : Reshaping Array

◆ concatenate()

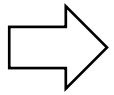
- **To combine two or more array objects,** we can use NumPy's concatenate function as shown in the following examples:

```
>>> ary = np.array([1, 2, 3])

>>> # stack along the first axis
>>> np.concatenate((ary, ary))
```



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

```
>>> ary = np.array([[1, 2, 3]])

>>> # stack along the first axis (here: rows)
>>> np.concatenate((ary, ary), axis=0)
```



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

Default : axis=0

NumPy : Comparison Operators and Masks

◆ Boolean mask

- **Using comparison operators (such as <, >, <=, and >=), we can create a Boolean mask of that array which consists of True and False** elements depending on whether a condition is met in the target array

```
>>> ary = np.array([1, 2, 3, 4])
>>> mask = ary > 2
>>> mask

array([False, False,  True,  True])
```

```
>>> ary[mask]

array([3, 4])

>>> mask.sum()

2
```

NumPy : Comparison Operators and Masks

◆ `np.where()`

- A related, useful function **to assign values to specific elements** in an array

```
>>> ary = np.array([1, 2, 3, 4])
>>> np.where(ary > 2, 1, 0)

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

NumPy : Linear Algebra with NumPy Arrays

◆ Row vector

```
>>> row_vector = np.array([1, 2, 3])
>>> row_vector

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

NumPy : Linear Algebra with NumPy Arrays

◆2D array for **column vector** (1)

```
>>> column_vector = np.array([[1, 2, 3]]).reshape(-1, 1)
>>> column_vector

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

◆2D array for **column vector** (2)

```
>>> row_vector = np.array([1, 2, 3])
>>> row_vector[:, np.newaxis]
```

◆2D array for **column vector** (3)

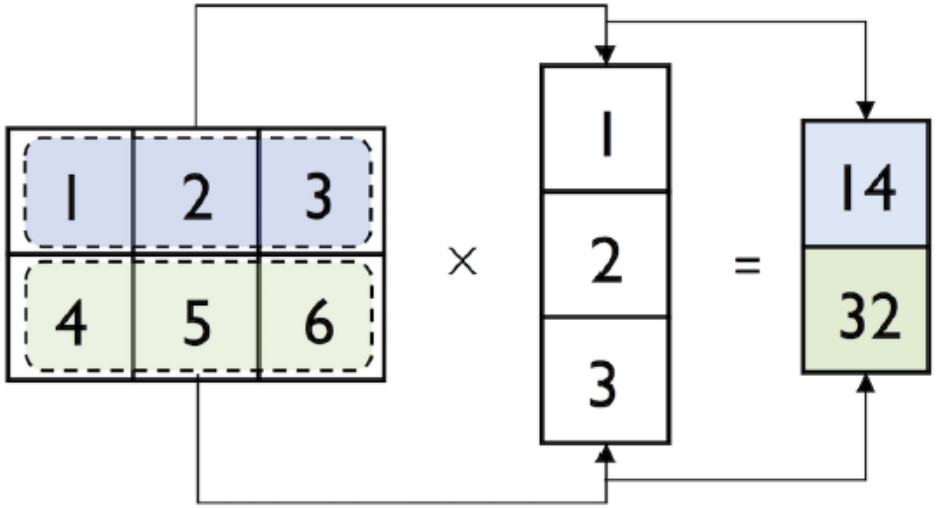
```
>>> row_vector = np.array([1, 2, 3])
>>> row_vector[:, None]
```

NumPy : Linear Algebra with NumPy Arrays

◆ `matmul()`

- **we can perform matrix multiplication** via the `matmul` function:

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

>>> np.matmul(matrix, column_vector)

array([[14],
       [32]])
```



NumPy : Linear Algebra with NumPy Arrays

◆ matmul()

- **we can compute the dot-product between two vectors** (here: the vector norm)

```
>>> row_vector = np.array([1, 2, 3])
>>> row_vector

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

```
>>> np.matmul(row_vector, row_vector)

14
```

```
>>> np.dot(row_vector, row_vector)

14
```

$$[1 \quad 2 \quad 3]\begin{bmatrix}1\\2\\3\end{bmatrix} = 1*1+2*2+3*3$$

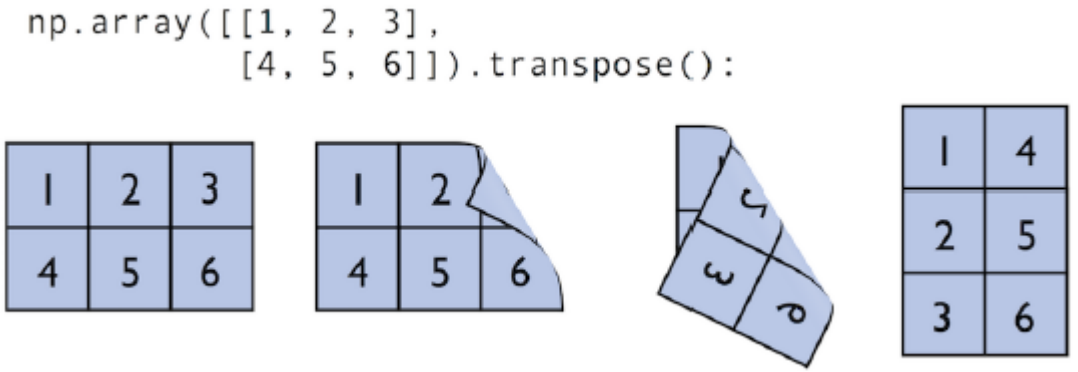
NumPy : Linear Algebra with NumPy Arrays

- `transpose(), T`
  - **to transpose matrices**

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

```
>>> matrix.transpose()

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

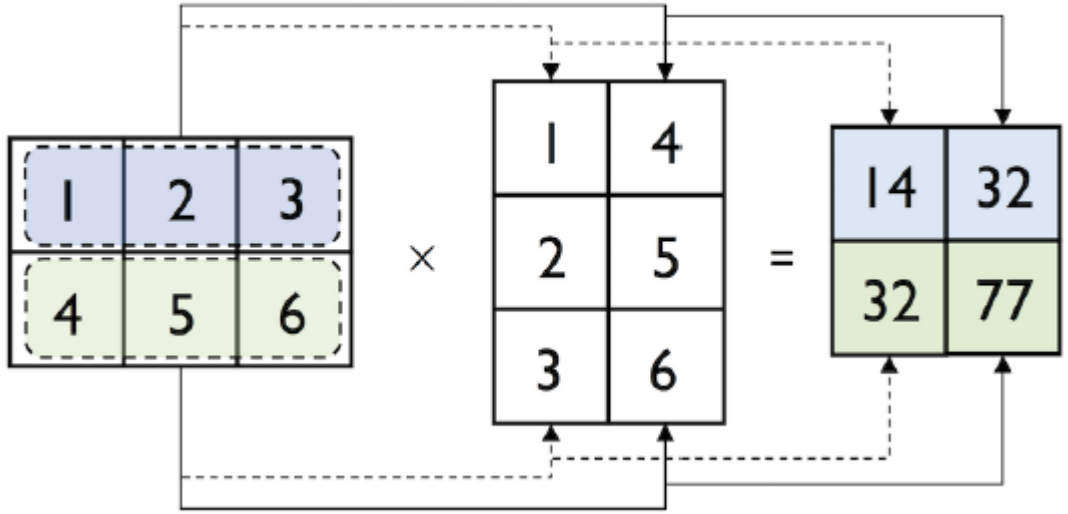
```
>>> matrix.T

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



NumPy : Linear Algebra with NumPy Arrays

- transpose(), T

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

>>> np.matmul(matrix, matrix.transpose())
```



NumPy

- 넘파이에서 제공하는 함수의 종류는 굉장히 다양
  - 실습 진행 시 주로 사용하는 함수 사용법 숙지
  - 그 외 함수 사용법은 넘파이 공식 매뉴얼이나 구글 검색 권장
- 넘파이 소개 자료 (참고)
  - ✓The official NumPy documentation
  - ✓넘파이 기본 사용법
  - ✓넘파이 기본 (총 7강)
  - ✓Linear algebra (numpy.linalg)

SciPy

- 과학 계산용 함수를 모아 놓은 파이썬 패키지
  - 상당히 많이 Numpy를 이용
- Numpy가 지원하지 않는 고성능 선형대수, 함수 최적화, 신호 처리, 특수한 수학 함수와 통계 분포 등을 포함한 많은 기능을 제공
- scikit-learn은 알고리즘을 구현할 때 SciPy의 여러 함수를 사용
- https://docs.scipy.org/doc/scipy/reference/

Subpackage	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions

Matplotlib

- 데이터를 차트나 그래프로 시각화하는 패키지
- 데이터 분석 이전에 데이터 이해를 위한 시각화
- 데이터 분석 이후에 결과를 시각화
- 공식 Matplotlib 갤러리

Matplotlib

◆ 설정

코드 셀에서 그래프를 생성하고 실행한 셀 아래에 이미지 형태로 바로(inline) 출력

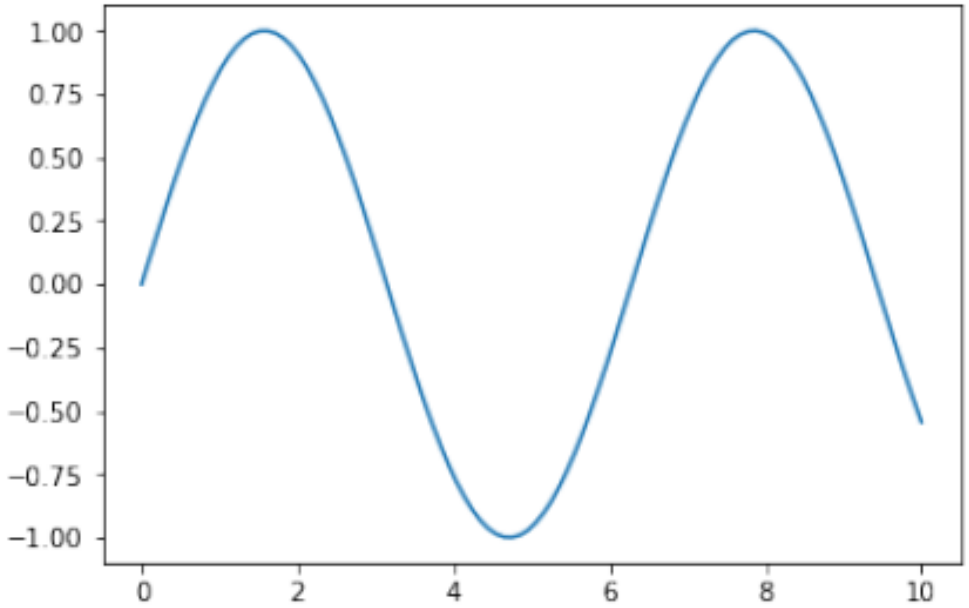
```
%matplotlib inline
import matplotlib.pyplot as plt
```

Matplotlib : Plotting Functions and Lines

◆ `plt.plot(x, y)`

- 선 그래프(Line plot) 생성
- x와 y는 데이터 점의 x축과 y축 좌표를 나타내는 array 또는 list

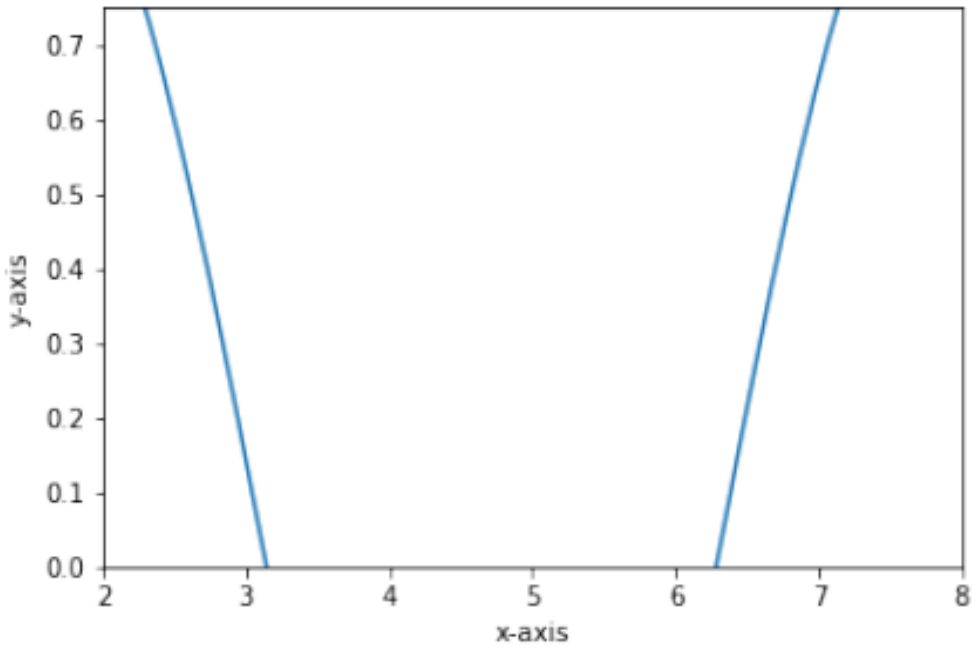
```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))
plt.show()
```



`np.linespace(0, 10, 100) # [0,10] 사이의 100개 포인트 생성`

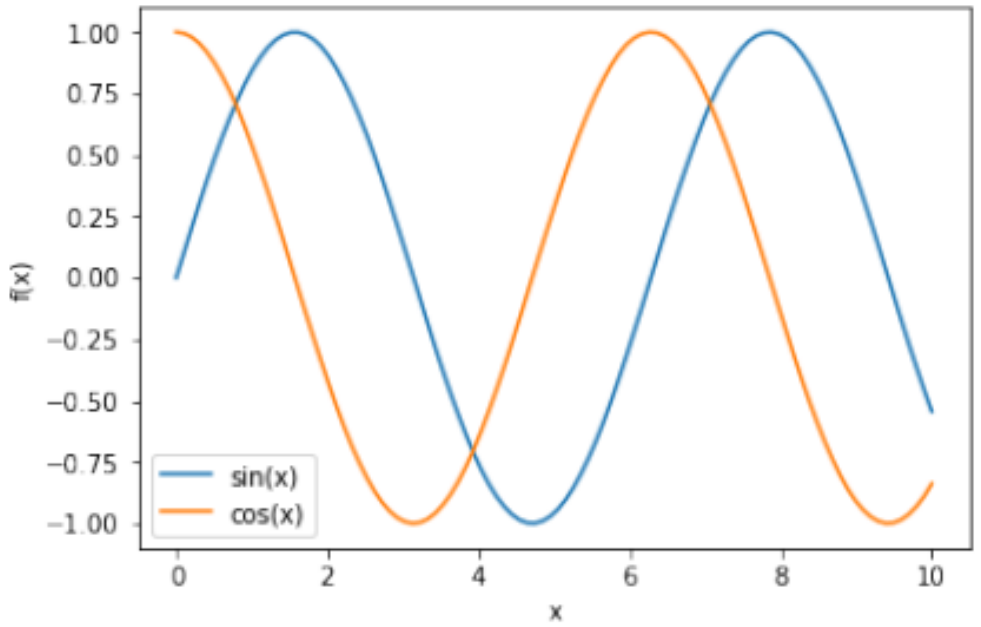
Matplotlib : Plotting Functions and Lines

```
x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))

plt.xlim([2, 8])
plt.ylim([0, 0.75])

plt.xlabel('x-axis')
plt.ylabel('y-axis')

plt.show()
```



Matplotlib : Plotting Functions and Lines

```
x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x), label=('sin(x)'))
plt.plot(x, np.cos(x), label=('cos(x)'))

plt.ylabel('f(x)')
plt.xlabel('x')

plt.legend(loc='lower left')
plt.show()
```



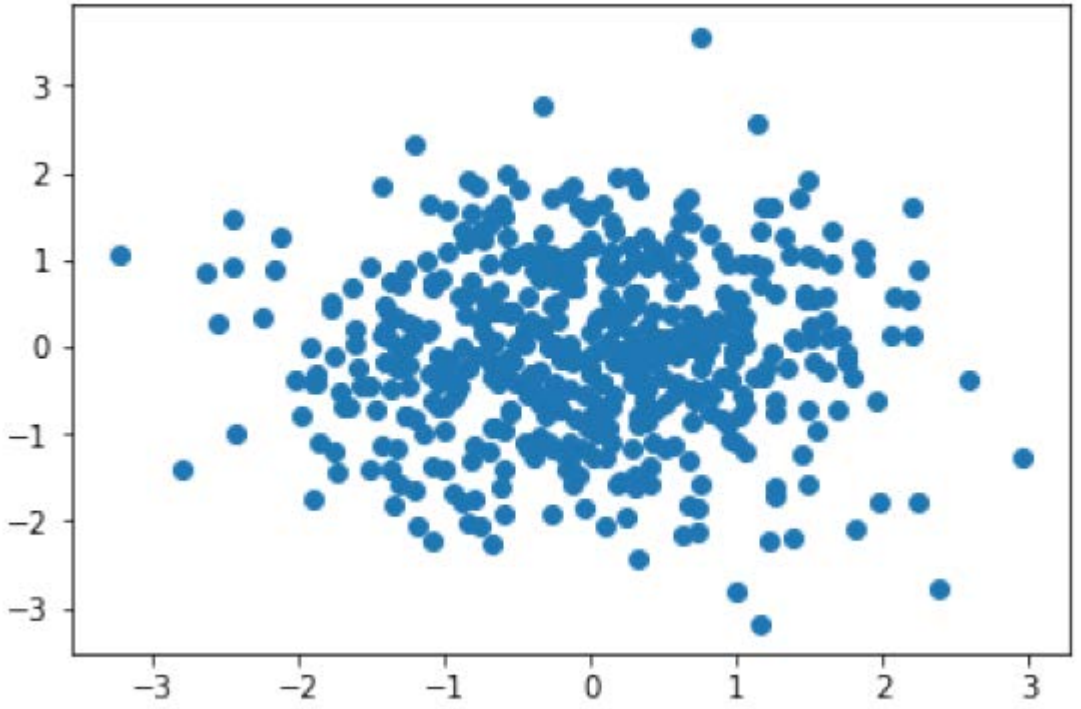
그림에 범례(legend)를 추가
'best': 최적의 위치를 자동으로 선택
'upper right': 오른쪽 상단 / 'upper left': 왼쪽 상단
'lower right': 오른쪽 하단 / 'lower left': 왼쪽 하단

Matplotlib : Scatter Plots

◆ `plt.scatter(x, y)`

- 산점도(Scatter plot)를 생성, 주로 데이터의 점들을 시각화할 때 사용

```
rng = np.random.RandomState(123)
x = rng.normal(size=500)
y = rng.normal(size=500)
plt.scatter(x, y)
plt.show()
```



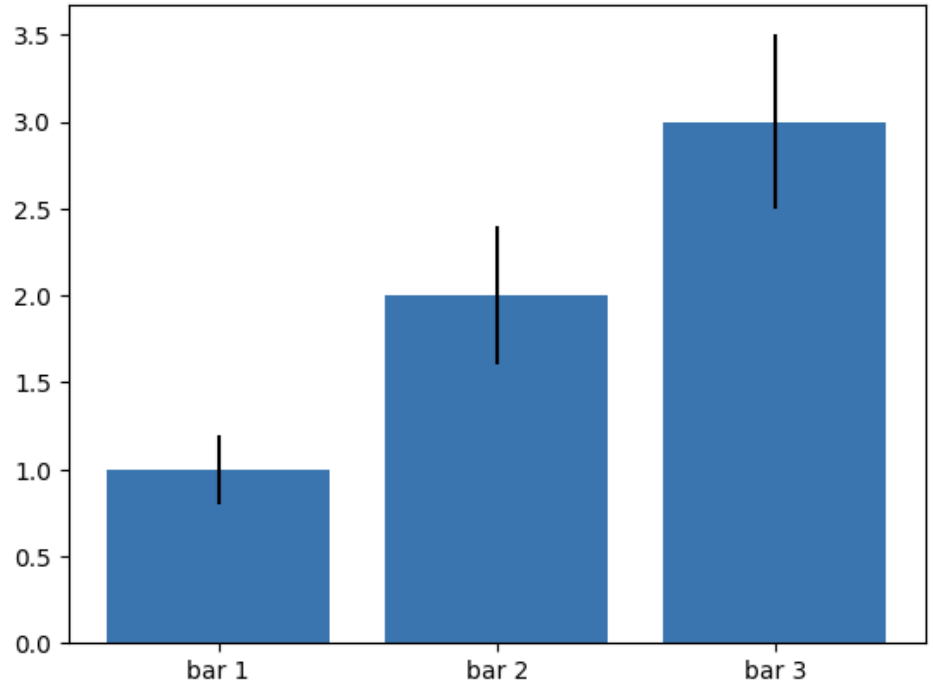
`rng.normal (size = 500) # 정규 분포(평균 0, 표준 편차 1)를 따르는 난수 500개를 생성`

Matplotlib : Bar Plots

◆ `plt.bar(category, values)`

- 막대 그래프(Bar chart)를 생성
- 범주형 데이터(category)의 빈도, 비교, 분포(values)등을 시각화하는 데 효과적

```
# input data
means = [1, 2, 3]
stddevs = [0.2, 0.4, 0.5]
bar_labels = ['bar 1', 'bar 2', 'bar 3']

#plot bars
plt.bar (bar_labels, means, yerr=stddevs)
plt.show()
```



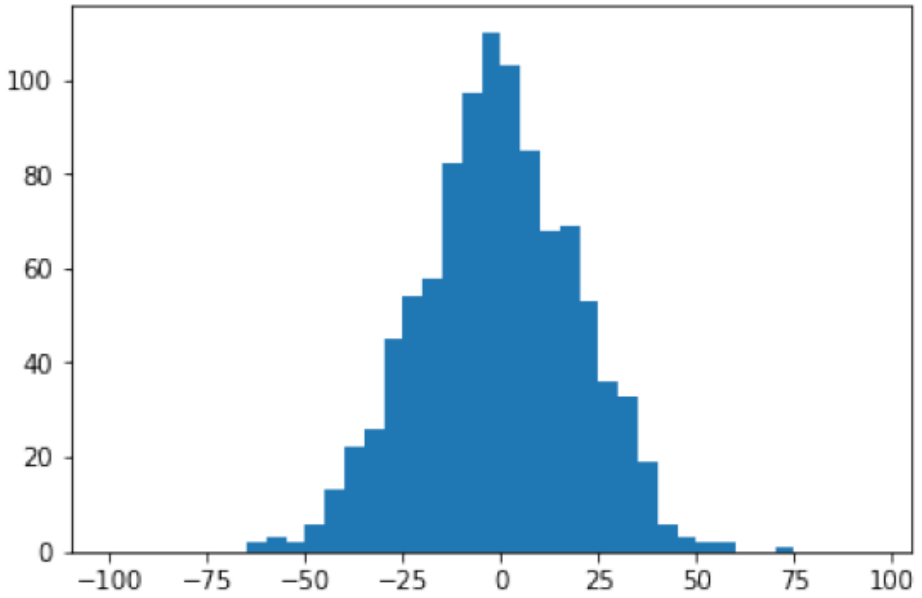
`yerr # (선택인자) 오차 막대의 길이(양수)`

Matplotlib : Histograms

◆ `plt.hist(data, bins= , …)`

- 히스토그램(Histogram)을 생성, 데이터 분포(values)를 시각화하는 데 효과적
- bin : 데이터를 나누는 구간(bin)의 수

```
rng = np.random.RandomState(123)
x = rng.normal(0, 20, 1000)  # (평균, 표준편차, 개수)

# fixed bin size
bins = np.arange(-100, 100, 5) # fixed bin size
                               # [시작, 끝), 간격
plt.hist(x, bins=bins)
plt.show()
```

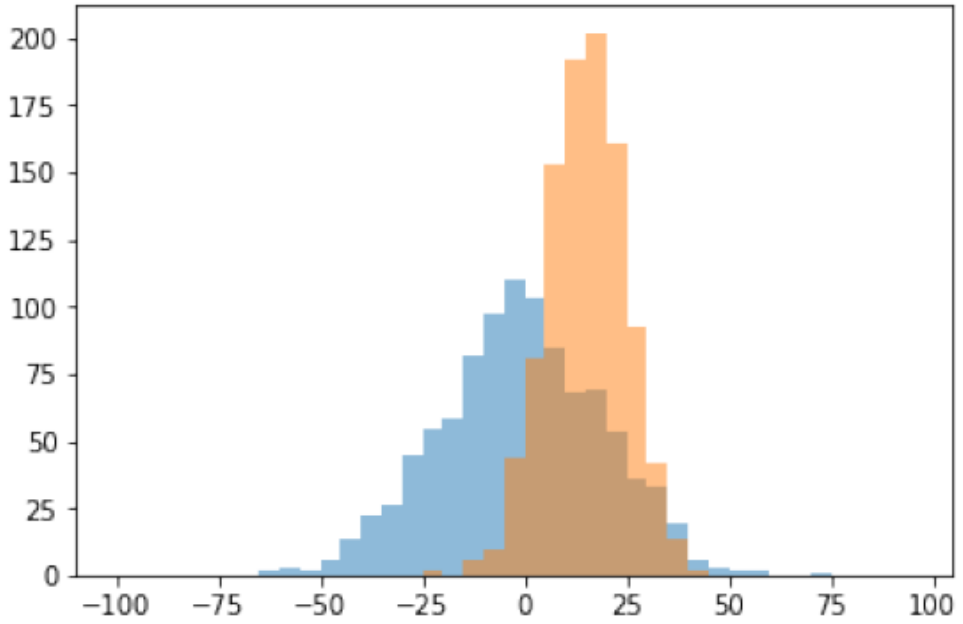


Matplotlib : Histograms

◆ `plt.hist(data, bins= , alpha=, …)`

- 히스토그램(Histogram)을 생성, 데이터 분포(values)를 시각화하는 데 효과적

```
rng = np.random.RandomState(123)
x1 = rng.normal(0, 20, 1000)
x2 = rng.normal(15, 10, 1000)

# fixed bin size
bins = np.arange(-100, 100, 5) # fixed bin size

plt.hist(x1, bins=bins, alpha=0.5)
plt.hist(x2, bins=bins, alpha=0.5)
plt.show()
```

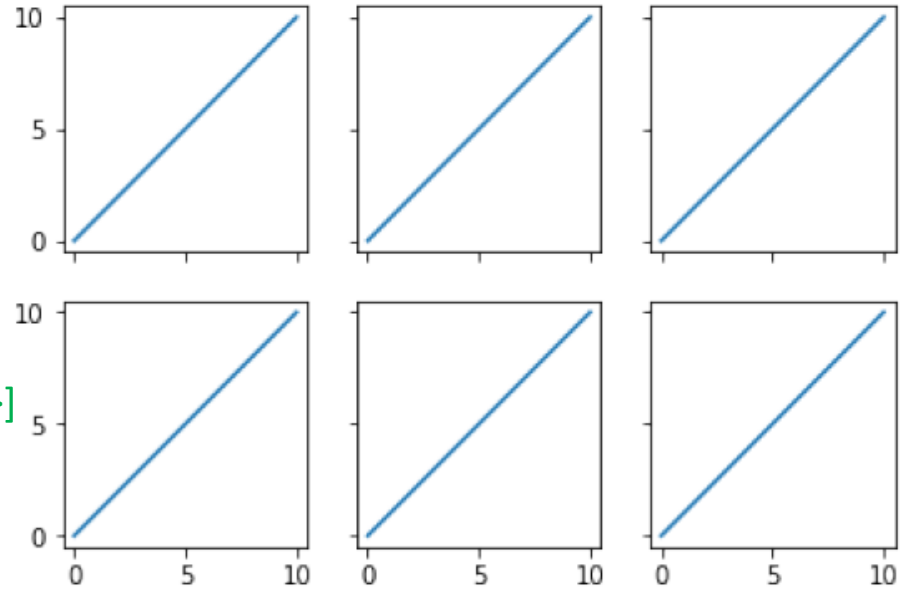


Matplotlib : Subplots

◆ `plt.subplots(nrow=, ncols= , …)`

- 여러 개의 하위 그래프(subplot)를 포함하는 Figure 객체와 그래프 객체(ax[행, 열]) 생성
- 하위 그래프에 접근할 때 ax[행, 열] 형식으로 인덱스를 사용

```
x = range(11)  # [0,11) 정수, 간격 1, [0, 1, 2, ..., 9, 10]
y = range(11)

fig, ax = plt.subplots(nrows=2, ncols=3,
                       sharex=True, sharey=True)
      # ax : 그래프 객체 [[<Axes: > <Axes: > <Axes: >]
                         [<Axes: > <Axes: > <Axes: >]]

for row in ax:  # row : ax의 각 row의 그래프 객체, [<Axes: > <Axes: > <Axes: >]
    for col in row:  # col : row의 각 col의 그래프 객체, <Axes: >
        col.plot(x, y)

plt.show()
```



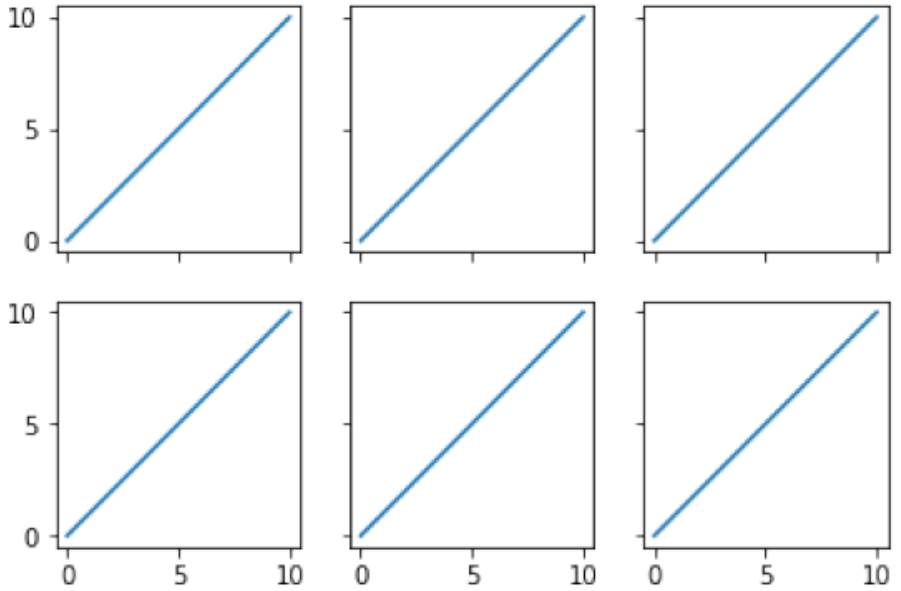
Matplotlib : Subplots

◆ `plt.subplots(nrow=, ncols= , …)`

- 여러 개의 하위 그래프(subplot)를 포함하는 Figure 객체와 그래프 객체(ax[행, 열]) 생성
- 하위 그래프에 접근할 때 ax[행, 열] 형식으로 인덱스를 사용

```
x = range(11)  # [0,11) 정수, 간격 1, [0, 1, 2, ..., 9, 10]
y = range(11)

fig, ax = plt.subplots(nrows=2, ncols=3,
                       sharex=True, sharey=True)
    # ax : 그래프 객체 [[<Axes: > <Axes: > <Axes: >]
                       [<Axes: > <Axes: > <Axes: >]]
```

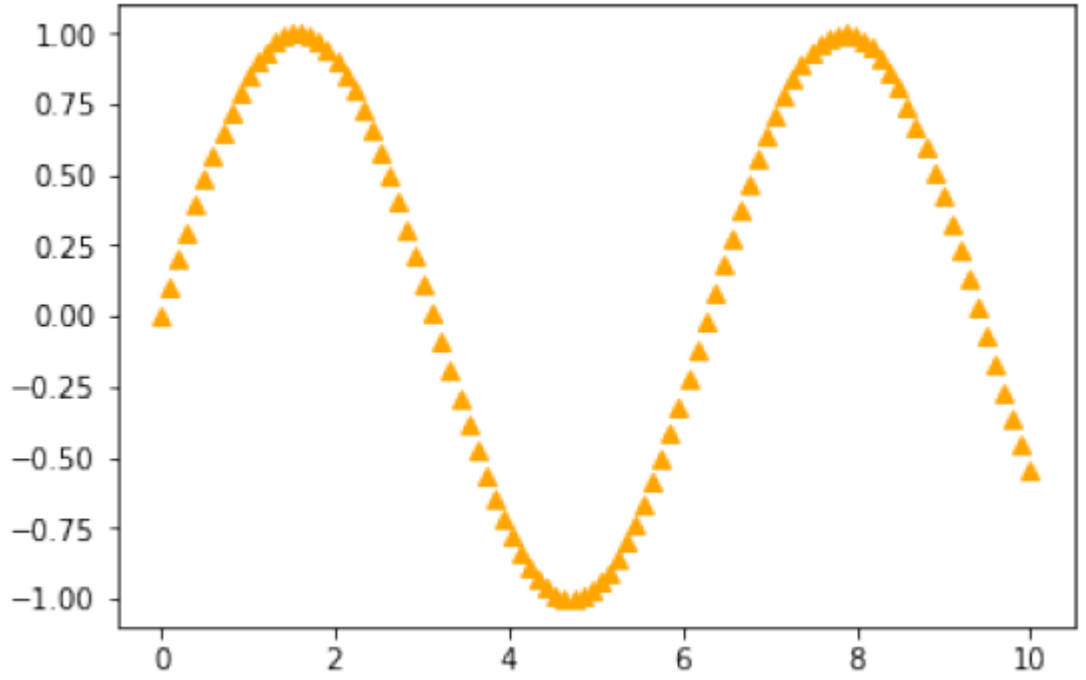
```
for row in range(2) :
  for col in range(3):
    ax[row,col].plot(x,y)

plt.show()
```



Matplotlib : Colors, Markers, LineStyle

◆ color=‘ ’, marker=‘ ’

```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x),
         color='orange',
         marker='^',
         linestyle='')
plt.show()
```



Colors

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Line Styles

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

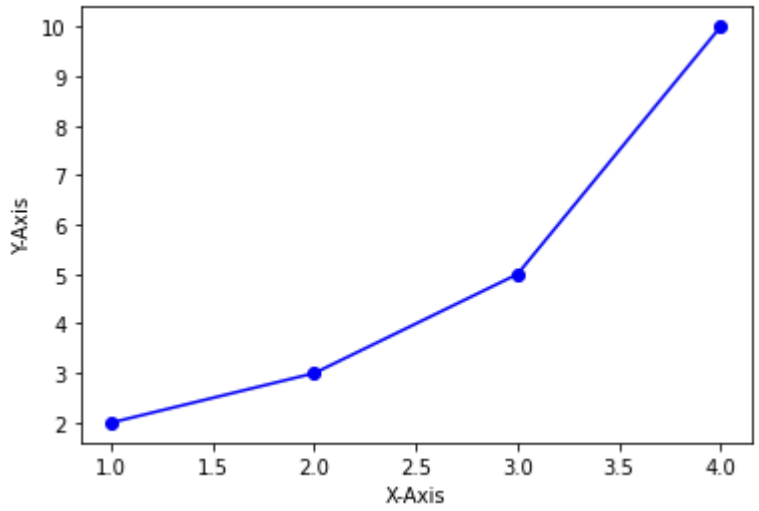
Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

Matplotlib : Colors, Markers, LineStyle

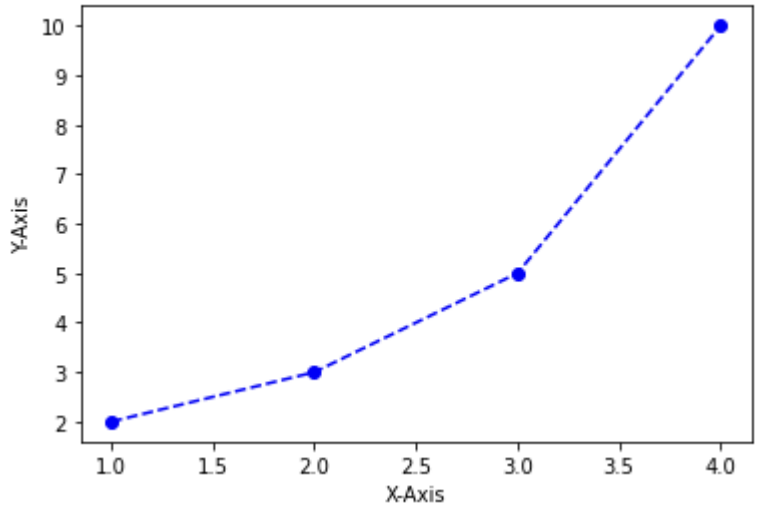
blue circle solid line

```
plt.plot(x, y, 'bo-')
```



blue circle dashed line

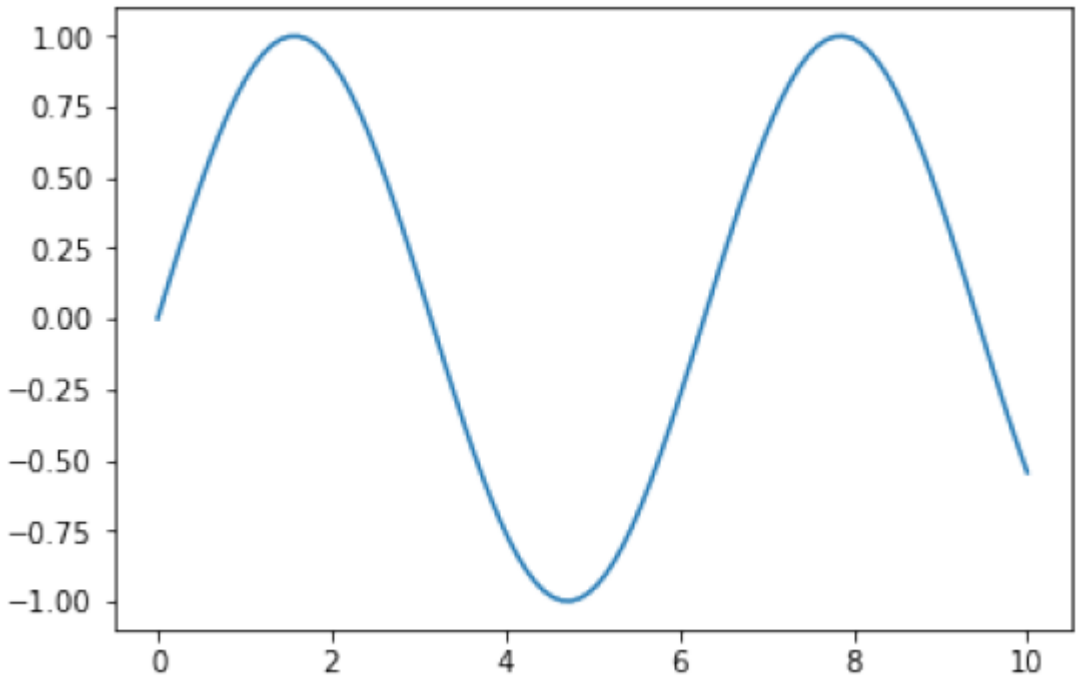
```
plt.plot(x, y, 'bo--')
```

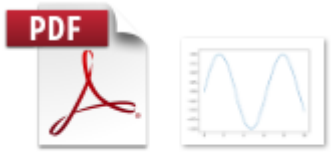


Matplotlib : Saving Plots

- Savefig('filename')
  - .eps, .svg, .jpg, .bmp, .png, .pdf, .tiff, …

```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))

plt.savefig('myplot.png', dpi=300)
plt.savefig('myplot.pdf')

plt.show()
```





myplot.pdf myplot.png

Matplotlib

◆ Matplotlib에서 제공하는 함수의 종류는 굉장히 다양

- 함수 사용법은 matplotlib 공식 매뉴얼이나 구글 검색 권장

◆ 유용한 소개 자료

- Matplotlib의 기초
  - 실습자료
- Matplotlib 다루기
  - 실습자료
- Matplotlib 공식 매뉴얼
- Matplotlib 공식 튜토리얼

Recommendation

◆ 읽을 거리 : Sebastian Raschka

- ML_L04_참고자료 04-scipython__notes.pdf

감사합니다.