



HAI Lecture

Lec 02.Numpy / Matplotlib / Pandas

Data Science Basic

| Introduction

| Numpy

| Matplotlib

| Pandas

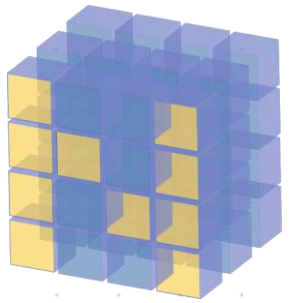
| 실습

| Frameworks for ML

INDEX

A photograph of a workspace with a laptop, a cup of coffee, and a smartphone. The laptop screen shows a code editor with a file explorer on the left. The coffee is in a white cup. The smartphone is in the foreground, displaying a social media post with a video thumbnail and text.

| Introduction



NumPy

Large-scale multi-dimensional array processing



Visualization of data and Plotting



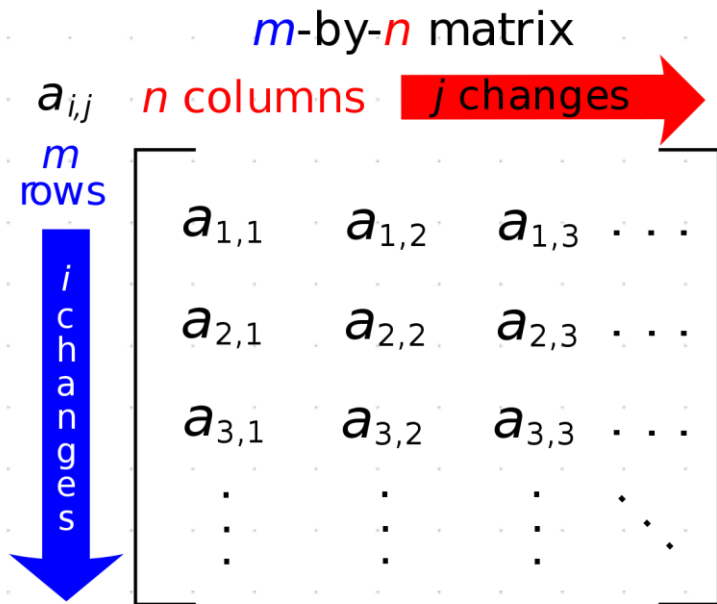
Manipulation and Analysis of Dataframe

| Get Ready : import libraries

이번 수업에서 배울 라이브러리를 import하기 위해
다음 코드를 입력해주세요.

```
import pandas as pd           // Pandas
import numpy as np            // Numpy
import matplotlib.pyplot as plt // Matplotlib
```

Numpy : Numpy의 배열



동일한 자료형을 가지는 값들이 격자판의 형태로 존재하는 것

- 각각의 값들은 튜플의 형태로 인덱싱됨.
- **rank**: 배열의 차원(Dimension)
- **shape**: 배열의 각 차원의 크기

| Numpy : Numpy의 배열

SIMPLE ARRAY CREATION

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

CHECKING THE TYPE

```
>>> type(a)  
numpy.ndarray
```

NUMERIC "TYPE" OF ELEMENTS

```
>>> a.dtype  
dtype('int32')
```

NUMBER OF DIMENSIONS

```
>>> a.ndim  
1
```

| Numpy : Array Operations

SIMPLE ARRAY MATH

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

```
>>> a * b
array([ 2, 6, 12, 20])
```

```
>>> a ** b
array([ 1, 8, 81, 1024])
```

* numpy에서 정한 상수(constant)

- np.pi: 3.14159265359

- np.e: 2.71828182846

| Numpy : Array Operations

```
# multiply entire array by  
# scalar value
```

```
>>> 0.1 * a  
array([0.1, 0.2, 0.3, 0.4])
```

```
# in-place operations
```

```
>>> a *= 2  
>>> a  
array([2, 4, 6, 8])
```

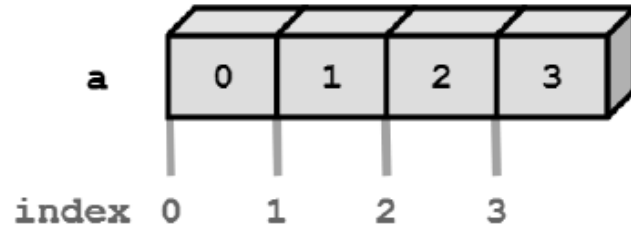
```
# apply functions to array
```

```
>>> x = 0.1*a  
>>> x  
array([0.2, 0.4, 0.6, 0.8])  
>>> y = np.sin(x)  
>>> y  
array([0.19866933, 0.38941834,  
0.56464247, 0.71735609])
```

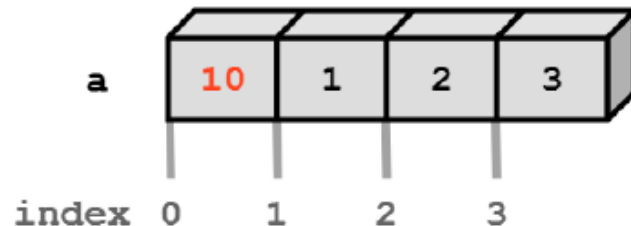

Numpy : Setting array elements

ARRAY INDEXING

```
>>> a[0]  
0
```



```
>>> a[0] = 10  
>>> a  
array([10, 1, 2, 3])
```



BEWARE OF TYPE COERCION

```
>>> a.dtype  
dtype('int32')
```

```
# assigning a float into  
# an int32 array truncates  
# the decimal part
```

```
>>> a[0] = 10.6  
>>> a  
array([10, 1, 2, 3])
```

Numpy array: All elements have the same type and the size.



Python list: Elements can have various sizes and types.

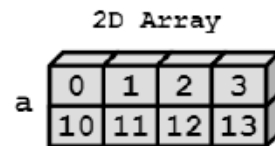
Numpy : Multi-dimensional array

MULTI-DIMENSIONAL ARRAYS

```
>>> a = np.array([[ 0, 1, 2, 3],  
...               [10,11,12,13]])
```

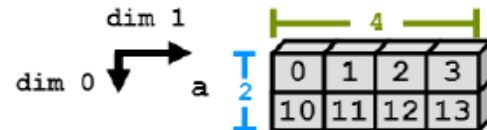
```
>>> a
```

```
array([[ 0,  1,  2,  3],  
       [10, 11, 12, 13]])
```



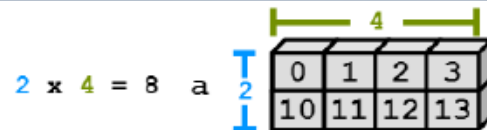
SHAPE = (ROWS, COLUMNS)

```
>>> a.shape  
(2, 4)
```



ELEMENT COUNT

```
>>> a.size  
8
```



NUMBER OF DIMENSIONS

```
>>> a.ndim  
2
```



GET / SET ELEMENTS

```
>>> a[1, 3]
```

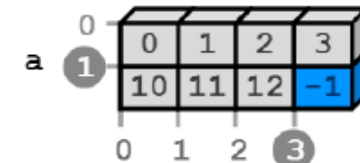
```
13
```



```
>>> a[1, 3] = -1
```

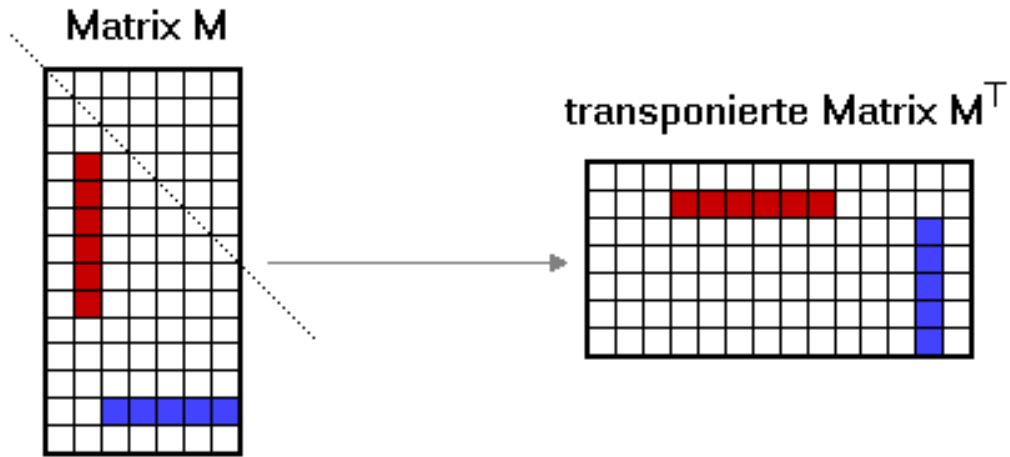
```
>>> a
```

```
array([[ 0,  1,  2,  3],  
       [10, 11, 12, -1]])
```



- # Shape returns a tuple
- # listing the length of the
- # array along each dimension.

Numpy : Transpose operation(전치 연산)



행렬의 행(row)과 열(column)의 위치를
뒤바꾸는 연산

$$\begin{bmatrix} 3 & 3 \\ 5 & -3 \\ 1 & 3 \\ 0 & -2 \\ 5 & -2 \end{bmatrix}^T = \begin{bmatrix} 3 & 5 & 1 & 0 & 5 \\ 3 & -3 & 3 & -2 & -2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}^T$$

Numpy : Transposing 2D array

a.T attribute

In [3]: a

Out[3]:

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

(1-1) transposing 2D array : T attribute

In [4]: a.T

Out[4]:

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

np.transpose() method

In [5]: a

Out[5]:

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

(1-2) transpose method : numpy.transpose(a, axes=None)

In [6]: np.transpose(a)

Out[6]:

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

| Numpy : Broadcasting

: “to spread widely; disseminate” – 흩뿌리고 퍼뜨리다, 전파하다.

```
import numpy as np
```

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

```
b = np.array([5, 6])
```

```
c = a + b
```

Traceback (most recent call last):

File "C:/Users/choig/PycharmProjects/untitled2/main.py", line 6, in <module>

c = a + b

ValueError: operands could not be broadcast together with shapes (3,) (2,)

Process finished with exit code 1

| Numpy : Broadcasting

: shape이 다른 배열들 간의 연산이 특정 조건을 만족할 때
가능해지도록 배열을 자동적으로 변환하는 것

* 브로드캐스팅의 조건

- 차원의 크기가 1일 때

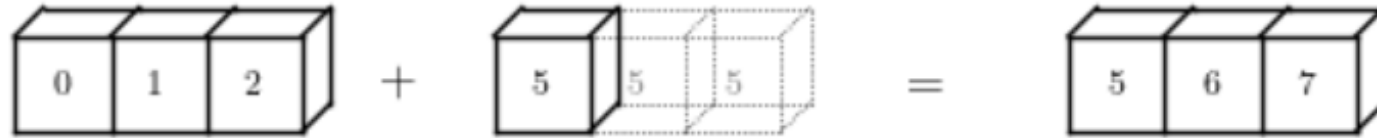
 - : 두 배열 간의 연산에서 최소 1개의 배열의 차원이 1이면 가능하다.

- 차원의 짝이 맞을 때

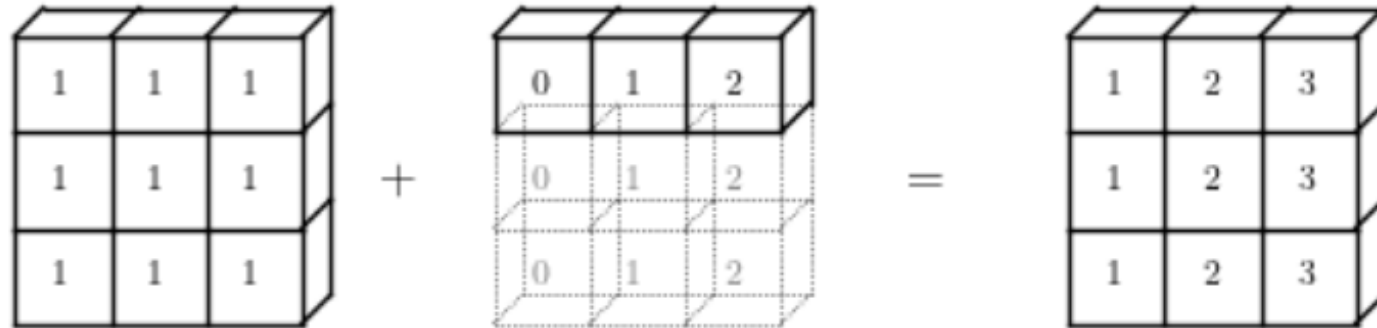
 - : 차원에 대해 축의 길이가 동일할 때

Numpy : Broadcasting

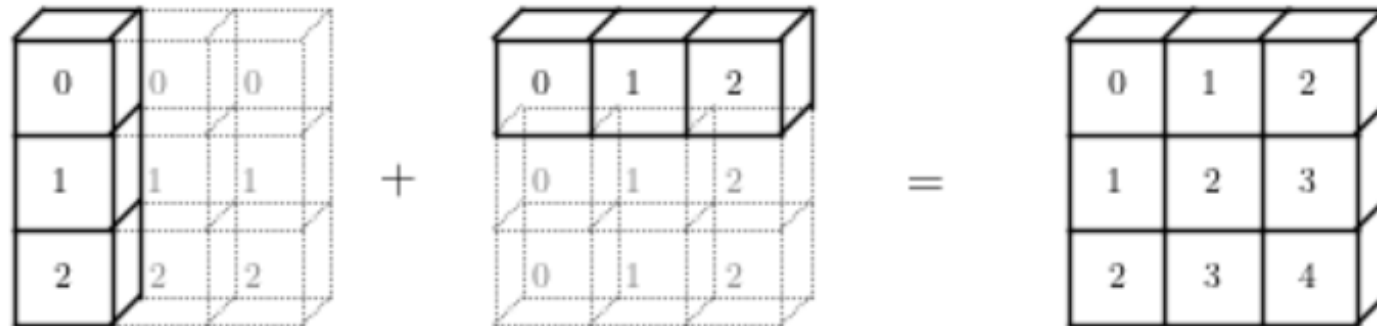
`np.arange(3) + 5`



`np.ones((3, 3)) + np.arange(3)`



`np.arange(3).reshape((3, 1)) + np.arange(3)`



Numpy : Saving data

single array

`np.save ()`

`np.load ()`

Binary file in

Numpy format(.npy)

`np.savez ()`

`np.load ()`

a single file in

uncompressed format(.npz)

several
arrays

`np.savez_compressed ()`

`np.load ()`

a single file in

compressed format(.npz)

`np.savetxt ()`

`np.loadtxt ()`

Text file(.txt)

Numpy : Saving data – single array

> **np.save()** : 1개의 배열을 NumPy format의 바이너리 파일로 저장하기
> **np.load()** : **np.save()**로 저장된 *.npy 파일을 배열로 불러오기


```
In [1]: import numpy as np
```

```
In [2]: x = np.array([0, 1, 2, 3, 4])
```

배열을 저장하기

```
In [3]: np.save('D:/admin/Documents/x_save', x) # x_save.npy
```

[.npy 형식으로 저장된 파일]

 x_save.npy	2018-05-22 오전...	NPY 파일	1KB
--	------------------	--------	-----

배열로 불러오기

```
In [4]: x_save_load = np.load('D:/admin/Documents/x_save.npy')
```

```
In [5]: x_save_load
```

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

Numpy : Saving data – several arrays


> **np.savez()** : 여러개의 배열을 1개의 압축되지 않은 *.npz 포맷 파일로 저장하기
> **np.load()** : np.savez()로 저장된 *.npz 파일을 배열로 불러오기

```
In [6]: x = np.array([0, 1, 2, 3, 4])
```

```
In [7]: y = np.array([5, 6, 7, 8, 9])
```

```
In [8]: np.savez('D:/admin/Documents/xy_savez'  
...: , x=x, y=y) # 각 배열에 이름 부여
```

[.npz 형식으로 저장된 파일]

 xy_savez.npz	2018-05-22 오전...	NPZ 파일	1KB
--	------------------	--------	-----

| Numpy : Saving data – several arrays

- 불러온 파일의 타입: `numpy.lib.npyio.NpzFile`
- 개별 array를 인덱싱하려면 []을 사용한다.

배열로 불러오기

```
In [9]: xy_savez_load = np.load('D:/admin/Documents/xy_savez.npz')
```

```
In [10]: type(xy_savez_load)
```

```
Out[10]: numpy.lib.npyio.NpzFile
```

```
In [11]: xy_savez_load['x']
```

```
Out[11]: array([0, 1, 2, 3, 4])
```

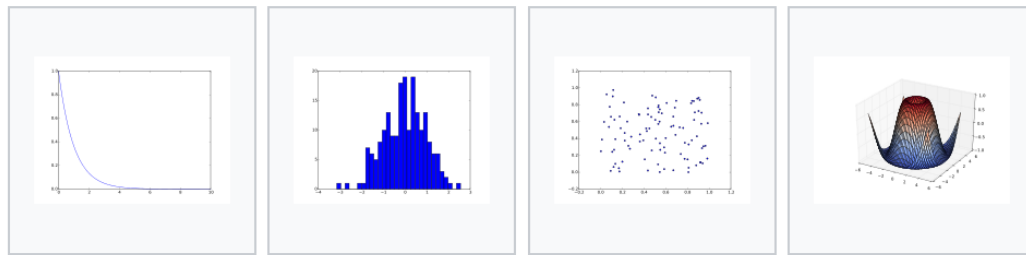
```
In [12]: xy_savez_load['y']
```

```
Out[12]: array([5, 6, 7, 8, 9])
```

Matplotlib : Introduction



- John D. Hunter에 의해 개발되었다.
- 2003년에 최초로 개발됨
- github.com/matplotlib/matplotlib



Line plot

Histogram

Scatter plot

3D plot

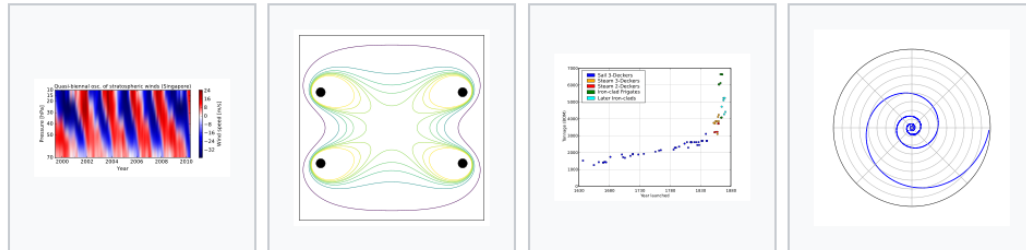


Image plot

Contour plot

Scatter plot

Polar plot

Matplotlib : Data Visualization

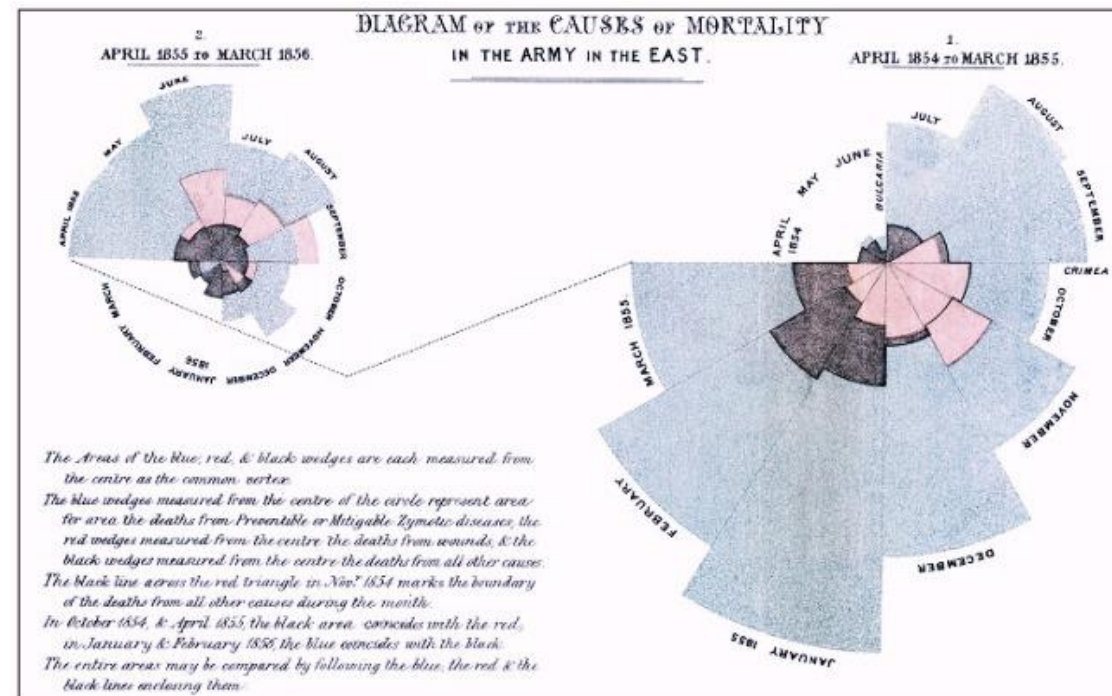
데이터를 시각적으로 표현하는 이유는?

1) 쉬운 이해, 설득하기 위한 목적

e.g. 나이팅게일의 “사망 원인에 따른 분석 그래프”

2) 분석의 용이함

데이터의 분포, 통계 등에 대한 직관적인 이해가 가능하다.



| Matplotlib : How to import?

```
import matplotlib as mpl  
import matplotlib.pyplot as plt
```

matplotlib 패키지의 서브 패키지(sub-package)

패키지의 하위 집합

※ 주피터 노트북을 사용하는 경우 아래의 명령어를 추가해주세요.

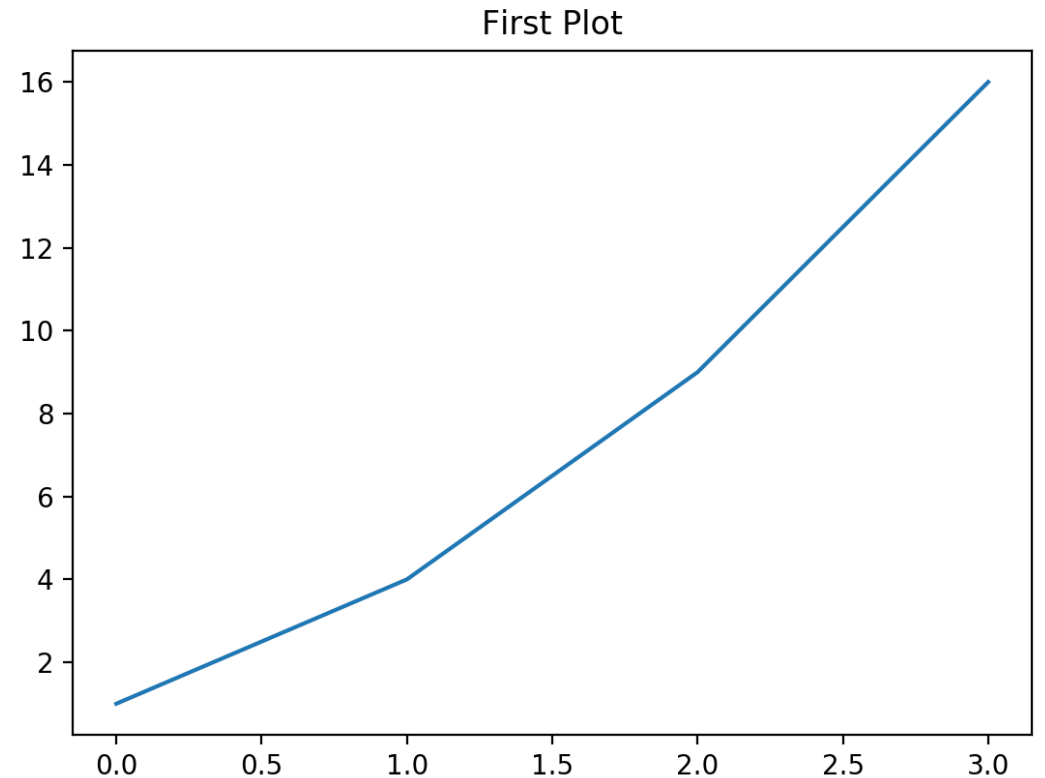
```
%matplotlib inline
```

Matplotlib : Line Plot(라인 플롯)

라인 플롯 (Line Plot) : 꺾은선 그래프

데이터가 시간, 순서 등에 따라 어떻게 변화하는지 보여주기 위해 사용함.

```
plt.title("First Plot")  
plt.plot([1, 4, 9, 16])  
plt.show()
```



| Matplotlib : 그래프 그리기의 기본

```
plt.title("(title)")
```

string을 인자로 받아 그래프의 제목을 정하는 명령어

```
plt.plot([1, 4, 9, 16])
```

그래프를 그리는 명령어, 인자로는 list나 ndarray 객체가 올 수 있다.

이때, 안의 내용은 y값이 되고, x값의 간격은 default로 1이 된다.

x tick

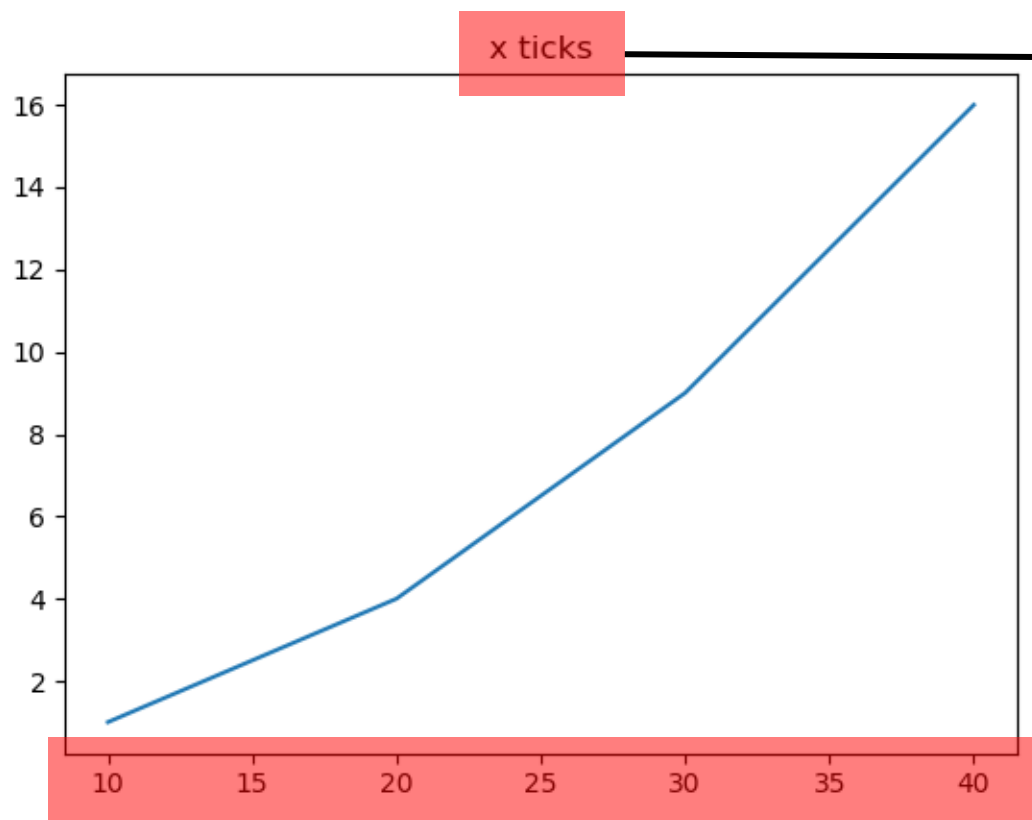
```
plt.show()
```

그래프를 출력하는 명령어

뒷부분에서 자세하게 설명하겠습니다.

Matplotlib : 그래프 제목 설정

```
plt.title("x ticks")  
plt.plot([10, 20, 30, 40], [1, 4, 9, 16])  
plt.show()
```

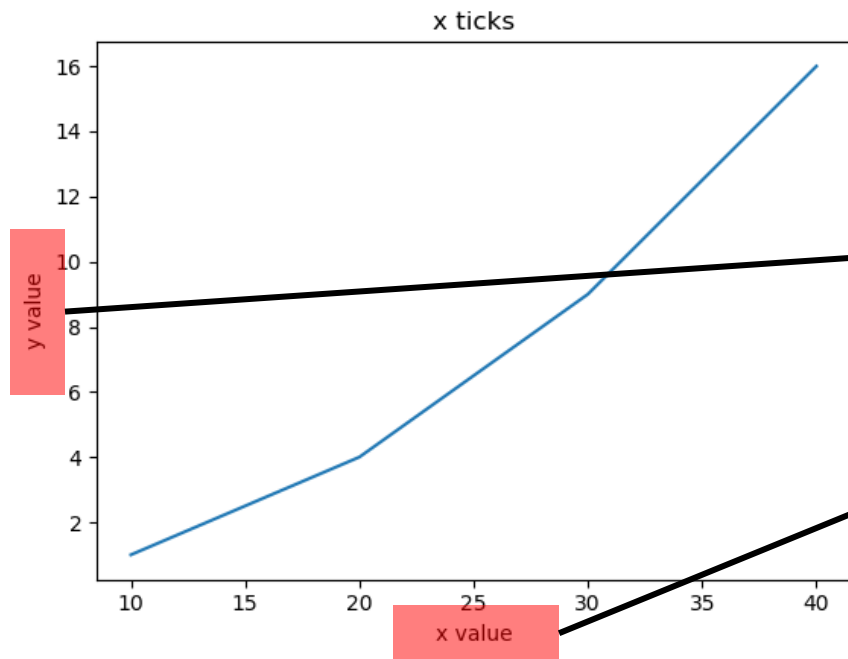


→ 그래프의 제목을 “x ticks”로 정함

↗ x값이 10, 20, 30, 40으로 됨.

Matplotlib : 축 레이블 설정

```
plt.title("x ticks")  
plt.plot([10, 20, 30, 40], [1, 4, 9, 16])  
plt.xlabel("x value") // x축의 라벨을 설정  
plt.ylabel("y value") // y축의 라벨을 설정  
plt.show()
```



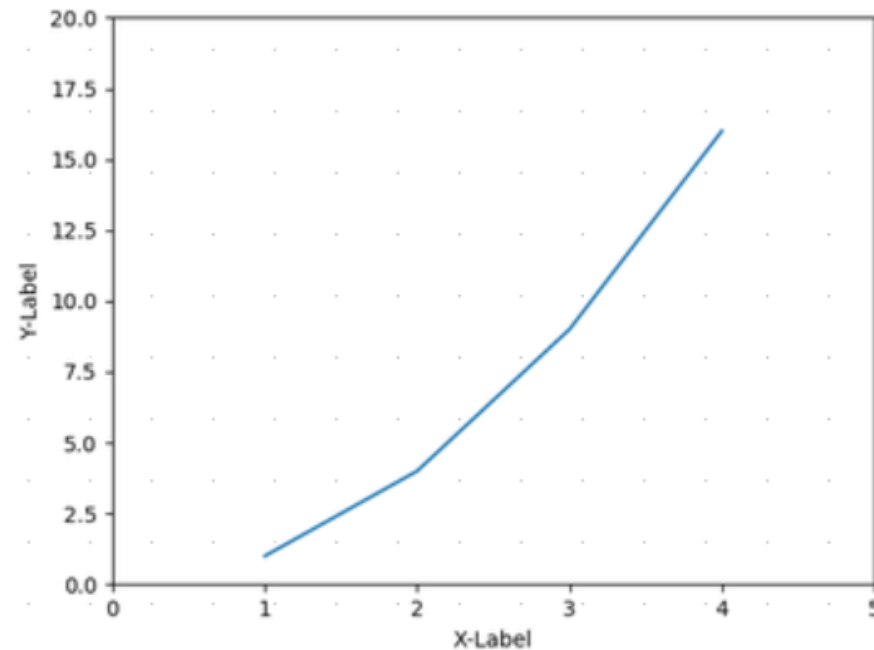
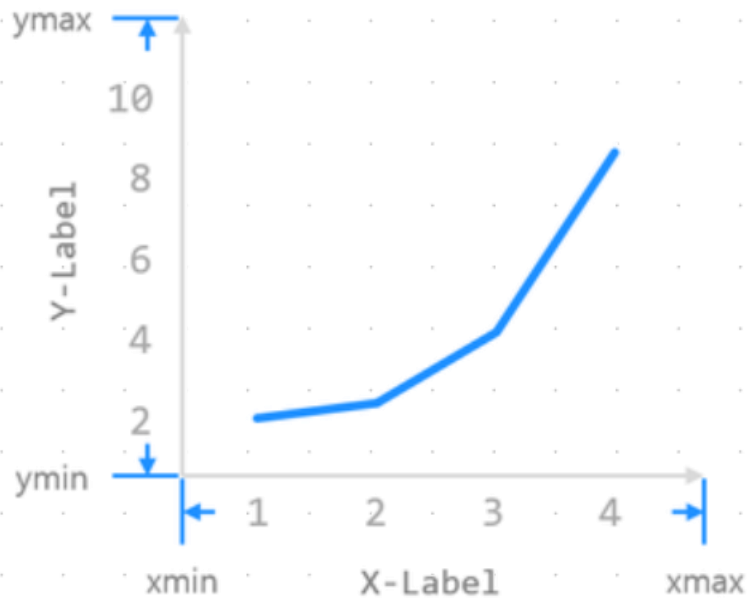
그래프의 제목을 "y ticks"로 정함

그래프의 제목을 "x ticks"로 정함

Matplotlib : 축 범위 설정

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])  
plt.xlabel('X-Label')  
plt.ylabel('Y-Label')  
plt.axis([0, 5, 0, 20])  
plt.show()
```

[xmin, xmax, ymin, ymax]

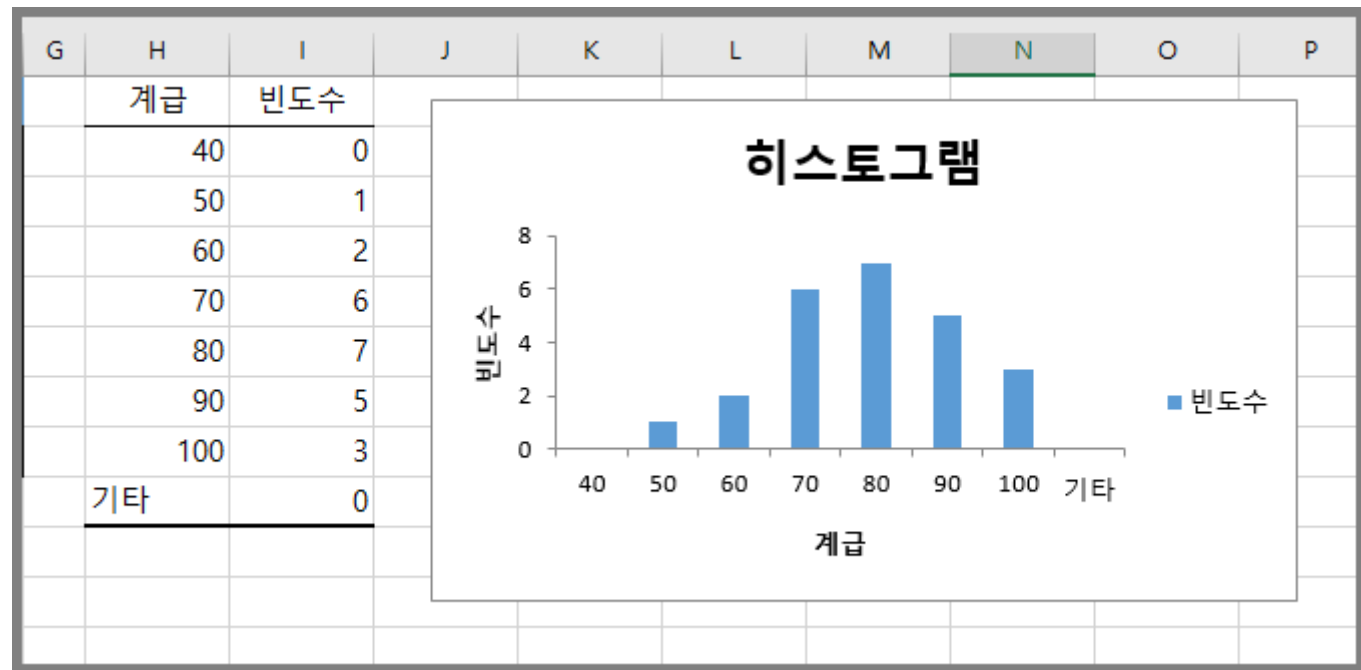


Matplotlib : 히스토그램(Histogram)

도수분포표를 그래프로 나타낸 것

가로축 = 계급, 세로축 = 도수(횟수 또는 개수)

* 계급 : 변수의 구간, 서로 겹치지 아니한다.



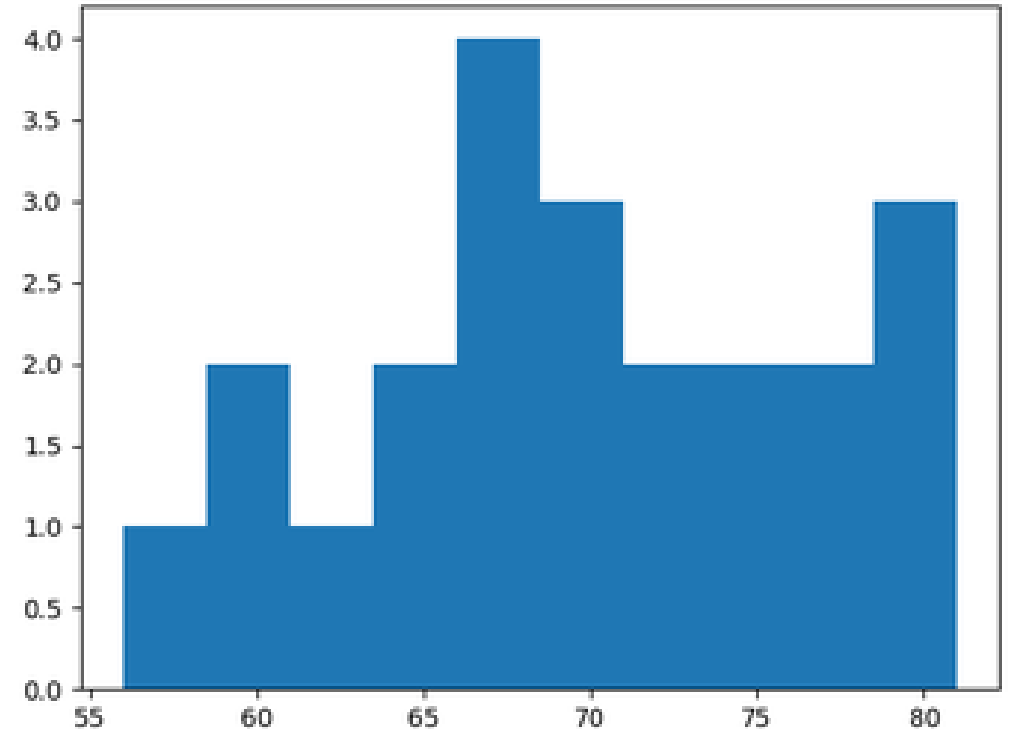
Matplotlib : 히스토그램(Histogram)

```
import matplotlib.pyplot as plt
```

```
weight = [68, 81, 64, 56, 78, 74,  
61, 77, 66, 68, 59, 71, 80, 59,  
67, 81, 69, 73, 69, 74, 70, 65]
```

```
plt.hist(weight)  
# 리스트의 형식을 넣는다.
```

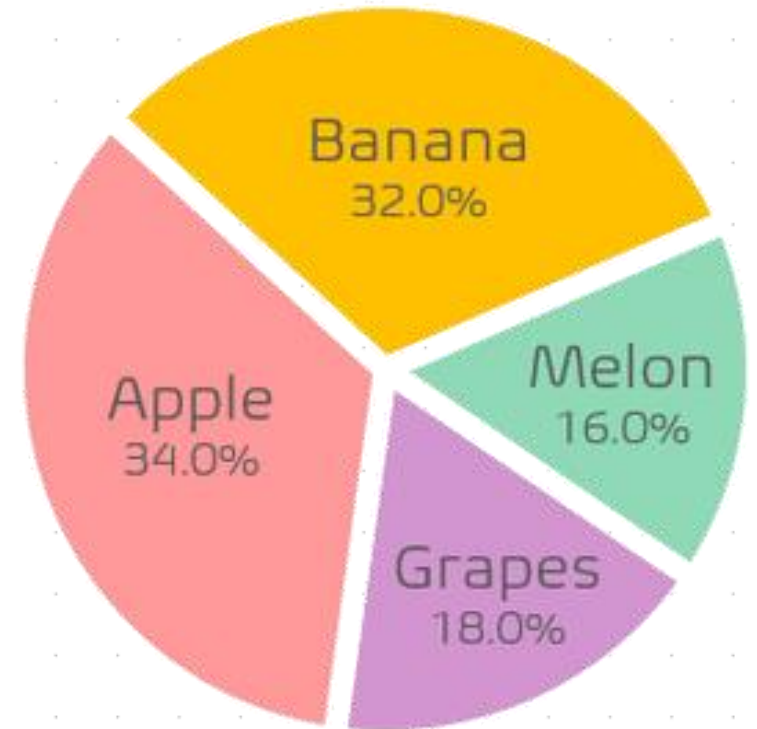
```
plt.show()
```



Matplotlib : Pie Chart(원형 그래프)

범주별 구성 비율을 원형으로 표현한 그래프

- 부채꼴의 중심각을 구성 비율에 비례하도록 표현한다.
- in matplotlib : **pie()** 함수



Matplotlib : Pie Chart(원형 그래프)

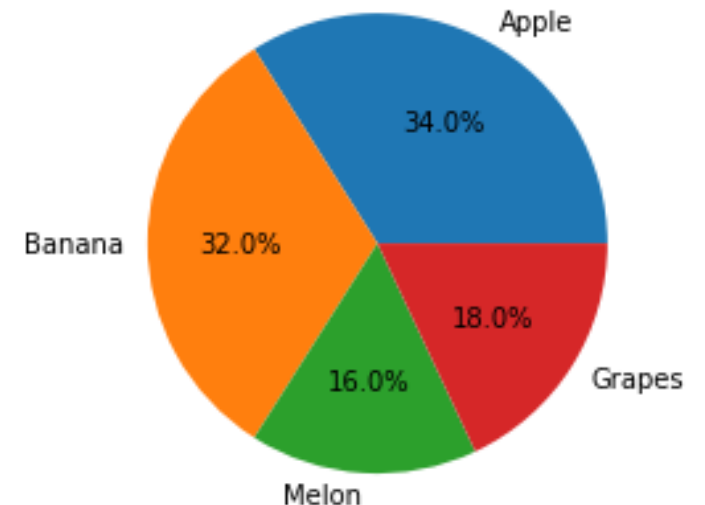
```
import matplotlib.pyplot as plt
```

```
ratio = [34, 32, 16, 18] # 각 영역의 비율
```

```
labels = ['Apple', 'Banana', 'Melon', 'Grapes'] # 각 영역의 이름
```

```
plt.pie(ratio, labels=labels, autopct='%.1f%%')  
plt.show()
```

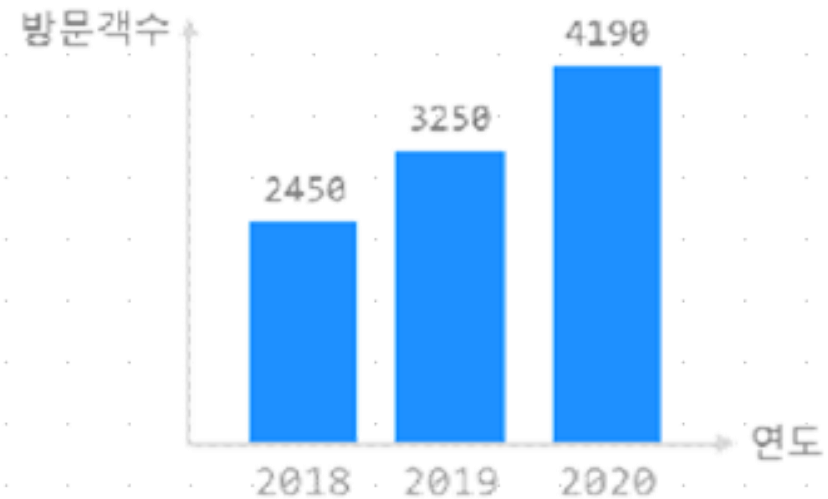
- pie() 함수에 ratio와 label 순서로 넣는다.
- autopct : 부채꼴 내에 표시될 숫자의 형식을 지정



| Matplotlib : Bar Chart(막대 그래프)

범주가 있는 데이터 값을 직사각형의 막대로 표현하는 그래프

- in matplotlib : **bar()** 함수

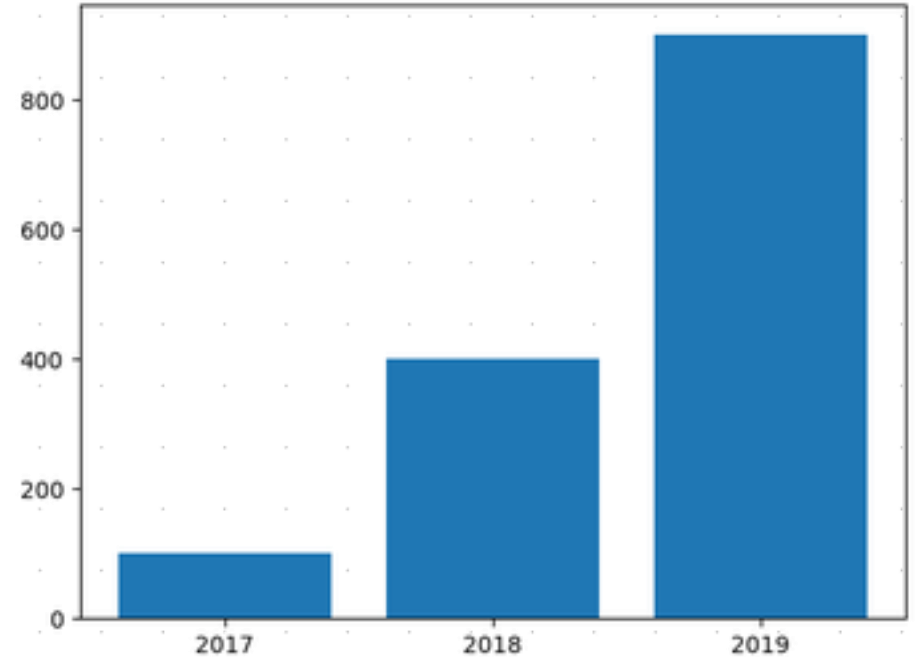


Matplotlib : Bar Chart(막대 그래프)

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

```
x = np.arange(3)  
years = ['2017', '2018', '2019']  
values = [100, 400, 900]
```

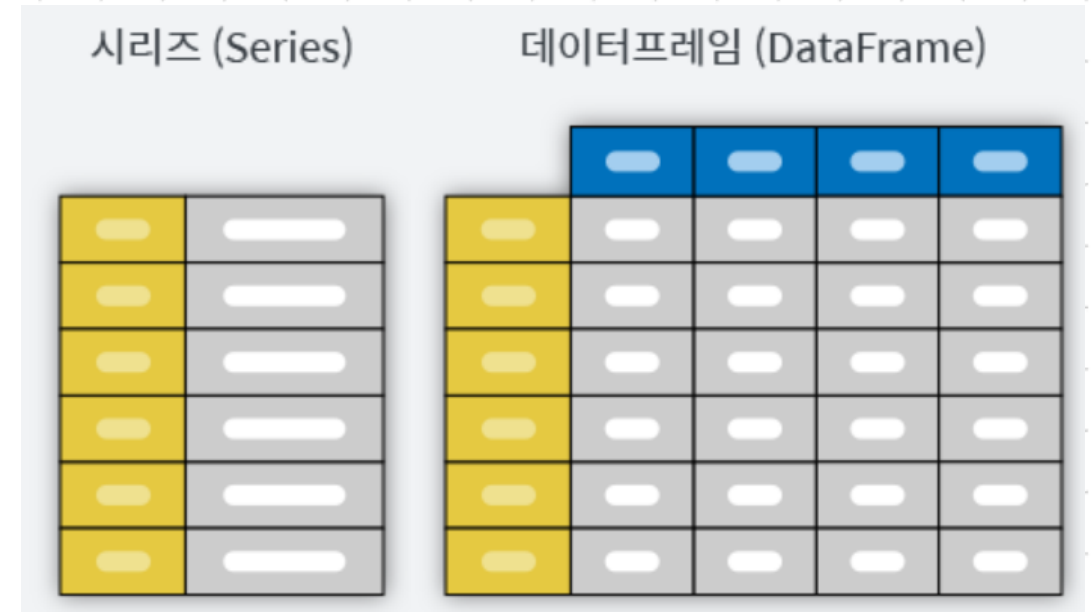
```
plt.bar(x, values)  
plt.xticks(x, years)  
plt.show()
```



| Pandas : Data Object

데이터를 담고 있는 그릇

- **Series**: 1차원 배열의 형태 -> 1가지 기준에 의하여 데이터가 저장된다.
- **DataFrame**: 2차원 배열의 형태
인덱스와 컬럼이라는 두 가지 기준으로
표 형태처럼 데이터가 저장된다.



Pandas : Series and DataFrame

```
import pandas as pd
import numpy as np
s = pd.Series([1, 3, 5, np.nan, 6, 8])
```

```
# 0    1.0
# 1    3.0
# 2    5.0
# 3    NaN
# 4    6.0
# 5    8.0
# dtype: float64
```

```
dates = pd.date_range('20130101', periods=6)
# DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
#                '2013-01-05', '2013-01-06'],
#                dtype='datetime64[ns]', freq='D')
```

```
df = pd.DataFrame(np.random.randn(6,4), index=dates,
                  columns=list('ABCD'))
```

#	A	B	C	D
# 2013-01-01	0.469112	-0.282863	-1.509059	-1.135632
# 2013-01-02	1.212112	-0.173215	0.119209	-1.044236
# 2013-01-03	-0.861849	-2.104569	-0.494929	1.071804
# 2013-01-04	0.721555	-0.706771	-1.039575	0.271860
# 2013-01-05	-0.424972	0.567020	0.276232	-1.087401
# 2013-01-06	-0.673690	0.113648	-1.478427	0.524988

| Pandas : Viewing Data – 1

.head() : 맨 앞의 자료를 확인 (default: 5개)

.tail() : 맨 뒤의 자료를 확인 (default: 5개)

첫 5개 행의 데이터를 보여줍니다.

`df.head()`

#	A	B	C	D
# 2013-01-01	0.469112	-0.282863	-1.509059	-1.135632
# 2013-01-02	1.212112	-0.173215	0.119209	-1.044236
# 2013-01-03	-0.861849	-2.104569	-0.494929	1.071804
# 2013-01-04	0.721555	-0.706771	-1.039575	0.271860
# 2013-01-05	-0.424972	0.567020	0.276232	-1.087401

마지막 3개 행의 데이터를 보여줍니다.

`df.tail(3)`

#	A	B	C	D
# 2013-01-04	0.721555	-0.706771	-1.039575	0.271860
# 2013-01-05	-0.424972	0.567020	0.276232	-1.087401
# 2013-01-06	-0.673690	0.113648	-1.478427	0.524988

| Pandas : Viewing Data - 2

.index : 인덱스 속성 확인

.columns : 컬럼 속성 확인

.values : 안에 들어있는 numpy 데이터 확인

```
df.index
# DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
#                '2013-01-05', '2013-01-06'],
#                dtype='datetime64[ns]', freq='D')
```

```
df.columns
# Index(['A', 'B', 'C', 'D'], dtype='object')
```

Pandas : Viewing Data - 3

.describe() : DataFrame의 간단한 통계 정보를 요약하여 보여줌

- 컬럼별 데이터 개수(count)
- 데이터의 평균값(mean), 표준편차(std)
- 최솟값(min), 최댓값(max)
- 4분위수(25%, 50%, 75%)

```
df.describe()
```

#		A	B	C	D
# count		6.000000	6.000000	6.000000	6.000000
# mean		0.073711	-0.431125	-0.687758	-0.233103
# std		0.843157	0.922818	0.779887	0.973118
# min		-0.861849	-2.104569	-1.509059	-1.135632
# 25%		-0.611510	-0.600794	-1.368714	-1.076610
# 50%		0.022070	-0.228039	-0.767252	-0.386188
# 75%		0.658444	0.041933	-0.034326	0.461706
# max		1.212112	0.567020	0.276232	1.071804

Pandas : Selection

DataFrame 자체가 가지고 있는 [] 슬라이싱 기능을 이용합니다.

A라는 이름을 가진 컬럼의 데이터만 갖고옵니다.

df['A']

```
# 2013-01-01    0.469112
# 2013-01-02    1.212112
# 2013-01-03   -0.861849
# 2013-01-04    0.721555
# 2013-01-05   -0.424972
# 2013-01-06   -0.673690
# req: D, Name: A, dtype: float64
```

df.A 또는 df['A']

	A	B	C	D
2013-01-01	0.469112	-0.282863	-1.509059	-1.135632
2013-01-02	1.212112	-0.173215	0.119209	-1.044236
2013-01-03	-0.861849	-2.104569	-0.494929	1.071804
2013-01-04	0.721555	-0.706771	-1.039575	0.271860
2013-01-05	-0.424972	0.567020	0.276232	-1.087401
2013-01-06	-0.673690	0.113648	-1.478427	0.524988

맨 처음 3개의 행을 가져옵니다.

df[0:3]

```
#           A           B           C           D
# 2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
# 2013-01-02  1.212112 -0.173215  0.119209 -1.044236
# 2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
```

df[0:3]

	A	B	C	D
2013-01-01	0.469112	-0.282863	-1.509059	-1.135632
2013-01-02	1.212112	-0.173215	0.119209	-1.044236
2013-01-03	-0.861849	-2.104569	-0.494929	1.071804
2013-01-04	0.721555	-0.706771	-1.039575	0.271860
2013-01-05	-0.424972	0.567020	0.276232	-1.087401
2013-01-06	-0.673690	0.113648	-1.478427	0.524988

Pandas : Selection

특정 조건을 만족하는 부분만 추출할 수 있습니다.

```
df[df.A > 0]
```

```
#           A           B           C           D
# 2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
# 2013-01-02  1.212112 -0.173215  0.119209 -1.044236
# 2013-01-04  0.721555 -0.706771 -1.039575  0.271860
```

```
df[df > 0]
```

```
#           A           B           C           D
# 2013-01-01  0.469112      NaN      NaN      NaN
# 2013-01-02  1.212112      NaN  0.119209      NaN
# 2013-01-03      NaN      NaN      NaN  1.071804
# 2013-01-04  0.721555      NaN      NaN  0.271860
# 2013-01-05      NaN  0.567020  0.276232      NaN
# 2013-01-06      NaN  0.113648      NaN  0.524988
```

df[df.A > 0]

	A	B	C	D
2013-01-01	0.469112	-0.282863	-1.509059	-1.135632
2013-01-02	1.212112	-0.173215	0.119209	-1.044236
2013-01-03	-0.861849	-2.104569	-0.494929	1.071804
2013-01-04	0.721555	-0.706771	-1.039575	0.271860
2013-01-05	-0.424972	0.567020	0.276232	-1.087401
2013-01-06	-0.673690	0.113648	-1.478427	0.524988

df[df > 0]

	A	B	C	D
2013-01-01	0.469112	-0.282863	-1.509059	-1.135632
2013-01-02	1.212112	-0.173215	0.119209	-1.044236
2013-01-03	-0.861849	-2.104569	-0.494929	1.071804
2013-01-04	0.721555	-0.706771	-1.039575	0.271860
2013-01-05	-0.424972	0.567020	0.276232	-1.087401
2013-01-06	-0.673690	0.113648	-1.478427	0.524988

* 선택되지 않은 값들은 결측치(NaN)로 표현됩니다.

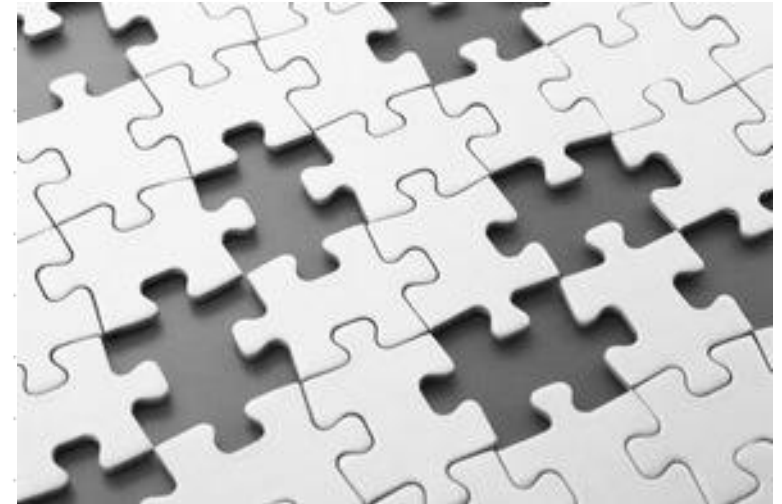
| Pandas : 결측치(missing data)

실제 연구에서는 여러 가지 이유로 데이터를 전부 다 수집하지 못하는 경우가 발생합니다.

이처럼 측정되지 못하여 비어 있는 데이터를 일컫는 말

np.nan

- pandas에서는 기본적으로 결측치를 연산에서 제외한다.



Pandas : 결측치의 제거

`.dropna()` : 결측치가 하나라도 존재하는 행들을 모두 버린다.

#		A	B	C	D	F	E
# 2013-01-01	0.000000	0.000000	-1.509059	5	NaN	1.0	
# 2013-01-02	1.212112	-0.173215	0.119209	5	1.0	1.0	
# 2013-01-03	-0.861849	-2.104569	-0.494929	5	2.0	NaN	
# 2013-01-04	0.721555	-0.706771	-1.039575	5	3.0	NaN	



```
df1.dropna(how='any')
```

#		A	B	C	D	F	E
# 2013-01-02	1.212112	-0.173215	0.119209	5	1.0	1.0	

Pandas : 결측치의 처리와 확인

.fillna() : 결측치가 있는 부분을 다른 값으로 채우는 메서드

```
df1.fillna(value=5)
```

#		A	B	C	D	F	E
# 2013-01-01	0.000000	0.000000	-1.509059	5	5.0	1.0	
# 2013-01-02	1.212112	-0.173215	0.119209	5	1.0	1.0	
# 2013-01-03	-0.861849	-2.104569	-0.494929	5	2.0	5.0	
# 2013-01-04	0.721555	-0.706771	-1.039575	5	3.0	5.0	

.isna() : 해당 값이 결측치인지의 여부를 Boolean형으로 반환하는 메서드

```
pd.isna(df1)
```

#		A	B	C	D	F	E
# 2013-01-01	False	False	False	False	True	False	
# 2013-01-02	False	False	False	False	False	False	
# 2013-01-03	False	False	False	False	False	True	
# 2013-01-04	False	False	False	False	False	True	

| Pandas : 데이터의 입력과 출력

CSV

DataFrame 객체를 CSV 파일 형식으로 저장한다.

```
df.to_csv('foo.csv')
```

CSV 파일로부터 DataFrame 객체를 읽어온다.

```
pd.read_csv('foo.csv')
```

```
#      Unnamed: 0      A      B      C      D
# 0  2000-01-01  0.266457 -0.399641 -0.219582  1.186860
# 1  2000-01-02 -1.170732 -0.345873  1.653061 -0.282953
# 2  2000-01-03 -1.734933  0.530468  2.060811 -0.515536
# ..      ...      ...      ...      ...      ...
# 997  2002-09-24 -9.902058 -9.340490 -4.386639 30.105593
# 998  2002-09-25 -10.216020 -9.480682 -3.933802 29.758560
# 999  2002-09-26 -11.856774 -10.671012 -3.216025 29.369368
#
# [1000 rows x 5 columns]
```




Thank You