

데이터분석 결과 보기

경남대학교 전하용

Pandas

(Python Data Analysis Library)

Why Pandas?

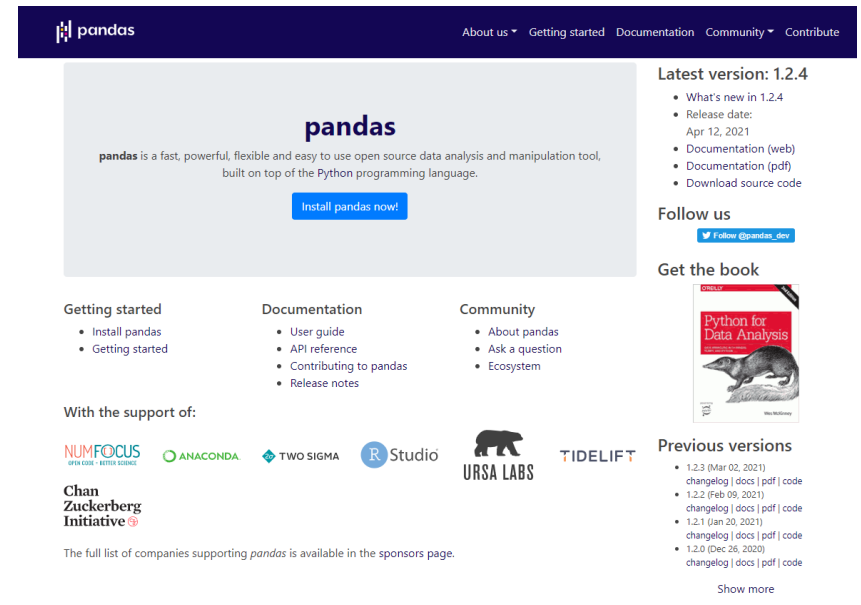
■ 데이터과학자가 판다스를 배우는 이유

- **빅데이터의 시대. 데이터 과학이라는 새로운 영역의 출현.**
 - 클라우드 컴퓨팅의 확산. 빅데이터 저장, 분석에 필요한 컴퓨팅 자원이 매우 저렴해짐.
 - 컴퓨팅 파워의 대중화는 최적의 학습환경과 연구 인프라를 제공.
- **데이터과학은 데이터를 연구하는 분야이고, 데이터 자체가 가장 중요한 자원**
 - 데이터 분석 업무의 80~90%는 데이터를 수집하고 정리하는 일이 차지.
 - 나머지 10~20%는 알고리즘을 선택하고, 모델링 결과를 분석하여 데이터로부터 유용한 정보 (information)을 뽑아내는 분석 프로세스의 몫.
- **판다스는 데이터를 수집하고 정리하는데 최적화된 도구.**
 - 가장 배우기 쉬운 프로그래밍 언어, 파이썬(Python) 기반.
 - 오픈소스(open source)로 무료로 이용 가능.

Pandas 정의

■ Pandas란?(Python Data Analysis Library)

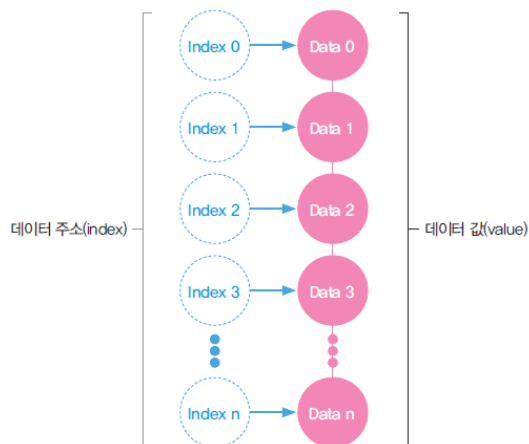
- 데이터 분석 및 가공에 사용되는 파이썬 라이브러리
- 엑셀과 상당히 유사
- 데이터의 수정/가공 및 분석이 용이
- 데이터 가공을 위한 수많은 함수를 지원
- Numpy 기반으로 데이터 처리가 상당히 빠름



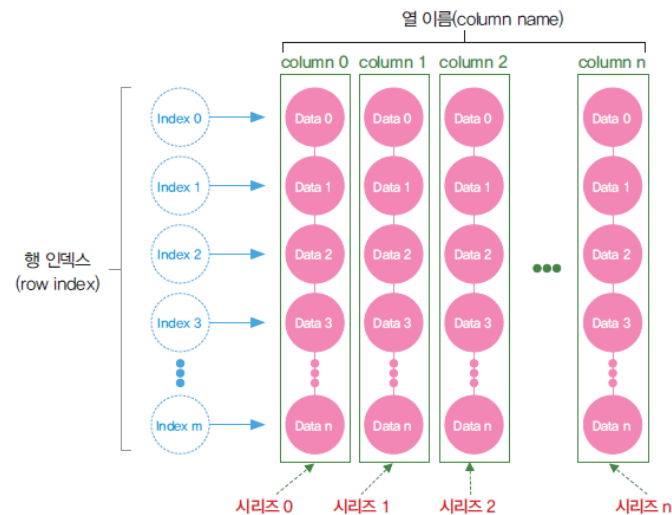
Pandas 자료구조

■ 판다스 자료구조

- 판다스의 일차적인 목적은 **형식적으로 서로 다른 여러 가지 유형의 데이터를 공통의 포맷으로 정리**하는 것
- 판다스는 **시리즈(Series)**와 **데이터프레임(DataFrame)**이라는 구조화된 데이터 형식을 제공



<시리즈 구조>

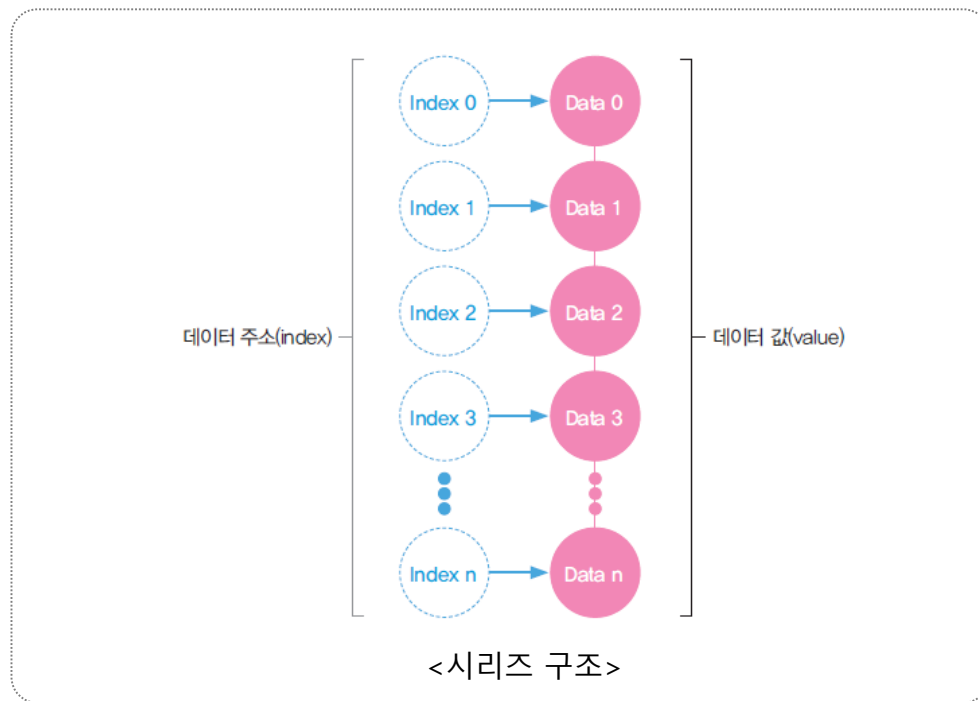


<데이터프레임 구조>

시리즈(Series)

■ 시리즈(Series)

- 일차원 레이블 배열과 같은 구조
- 모든 데이터 유형을 포함 할 수 있음
(정수, 부동 소수점, 문자열, 파이썬 객체 등)
- 두개의 배열(Array) 즉, 인덱스(즉 레이블) 기능을 하는 배열과 실제 데이터를 포함하는 배열의 구조로 볼 수 있음
- 산술 연산이 가능함



시리즈(Series)

■ 시리즈 만들기

- 딕셔너리와 시리즈의 구조가 비슷하기 때문에, 딕셔너리를 시리즈로 변환하는 방법을 많이 사용.
- 판다스 내장 함수인 Series()를 이용하고, 딕셔너리를 함수의 매개변수(인자)로 전달.

```
pandas.Series(딕셔너리)
```



시리즈(Series)

- 예제 : 딕셔너리의 시리즈 변환

```
import pandas as pd
```

```
# k:v 구조를 갖는 딕셔너리를 만들고, 변수 dict_data에 저장  
dict_data = {'a': 1, 'b': 2, 'c': 3}
```

```
# 판다스 Series() 함수로 딕셔너리(dict_data)를 시리즈로 변환.  
sr = pd.Series(dict_data)
```

```
# 변수 sr의 자료형 출력  
print(type(sr))  
print('\n')
```

```
# 변수 sr에 저장되어 있는 시리즈 객체를 출력  
print(sr)
```

```
<class 'pandas.core.series.Series'>
```

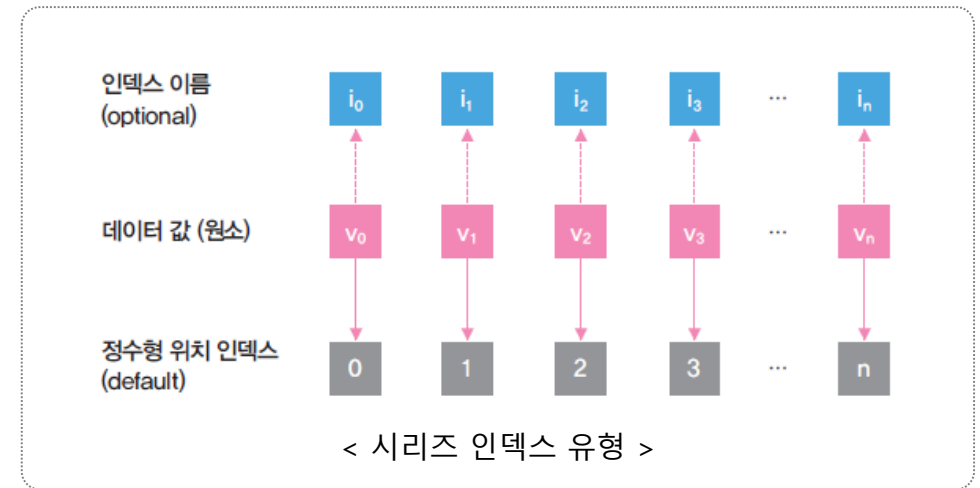
```
a    1  
b    2  
c    3  
dtype: int64
```

<실행결과>

시리즈(Series)

■ 인덱스 구조

- 인덱스는 자기와 짝을 이루는 원소의 순서와 주소를 저장.
- 인덱스를 활용하여 데이터 값의 탐색, 정렬, 선택, 결합 등 데이터 조작 함.
- 인덱스의 종류
 - ① 정수형 위치 인덱스(integer position)
 - ② 인덱스 이름(index name) 또는 인덱스 라벨(index label)



시리즈(Series)

- 예제 : 리스트의 시리즈 변환

```
import pandas as pd
```

```
# 리스트를 시리즈로 변환하여 변수 sr에 저장
```

```
list_data = ['2019-01-02', 3.14, 'ABC', 100, True]
```

```
sr = pd.Series(list_data)
```

```
print(sr)
```

```
print('\n')
```

```
# 인덱스 배열은 변수 idx에 저장. 데이터 값 배열은 변수 val에 저장
```

```
idx = sr.index
```

```
val = sr.values
```

```
print(idx)
```

```
print('\n')
```

```
print(val)
```

```
0    2019-01-02
```

```
1         3.14
```

```
2         ABC
```

```
3         100
```

```
4         True
```

```
dtype: object
```

```
RangeIndex(start=0, stop=5, step=1)
```

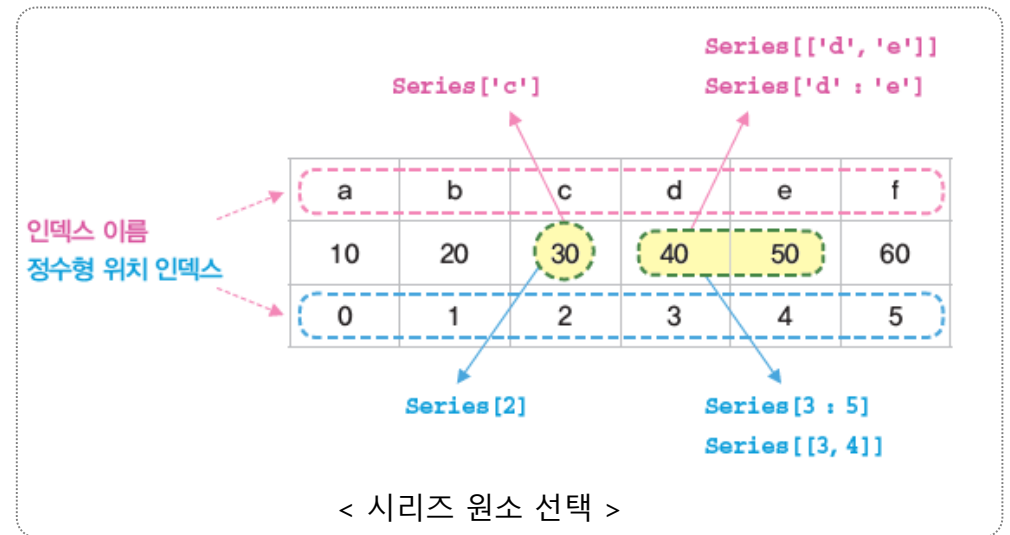
```
['2019-01-02' 3.14 'ABC' 100 True]
```

<실행결과>

시리즈(Series)

■ 원소 선택

- 인덱스를 이용하여, 시리즈의 원소를 선택.
- 하나의 원소를 선택하거나, 여러 원소를 한꺼번에 선택 가능.
- 인덱스 범위를 지정하여 여러 개의 원소 선택 가능.
- 인덱스의 유형에 따라 사용법이 조금 다름
 - 정수형 인덱스: 대괄호([]) 안에 숫자 입력. (0부터 시작)
 - 인덱스 이름(라벨): 대괄호([]) 안에 이름과 함께 따옴표를 입력.
 - 큰 따옴표(" "), 작은 따옴표(' ') 모두 사용.



시리즈(Series)

- 예제 : 리스트의 시리즈 변환

```
import pandas as pd
```

```
# 튜플을 시리즈로 변환(index 옵션에 인덱스 이름을 지정)
```

```
tup_data = ('영인', '2010-05-01', '여', True)
```

```
sr = pd.Series(tup_data, index=['이름', '생년월일', '성별', '학생여부'])
```

```
print(sr)
```

```
# 원소를 1개 선택
```

```
print(sr[0])          # sr의 1 번째 원소를 선택 (정수형 위치 인덱스를 활용)
```

```
print(sr['이름'])      # '이름' 라벨을 가진 원소를 선택 (인덱스 이름을 활용)
```

```
# 여러 개의 원소를 선택 (인덱스 리스트 활용)
```

```
print(sr[[1, 2]])
```

```
print(sr[['생년월일', '성별']])
```

```
# 여러 개의 원소를 선택 (인덱스 범위 지정)
```

```
print(sr[1 : 2])
```

```
print(sr['생년월일' : '성별'])
```

```
이름          영인
생년월일      2010-05-01
성별          여
학생여부      True
dtype: object
```

```
영인
영인
```

```
생년월일      2010-05-01
성별          여
dtype: object
```

```
생년월일      2010-05-01
성별          여
dtype: object
```

```
생년월일      2010-05-01
dtype: object
```

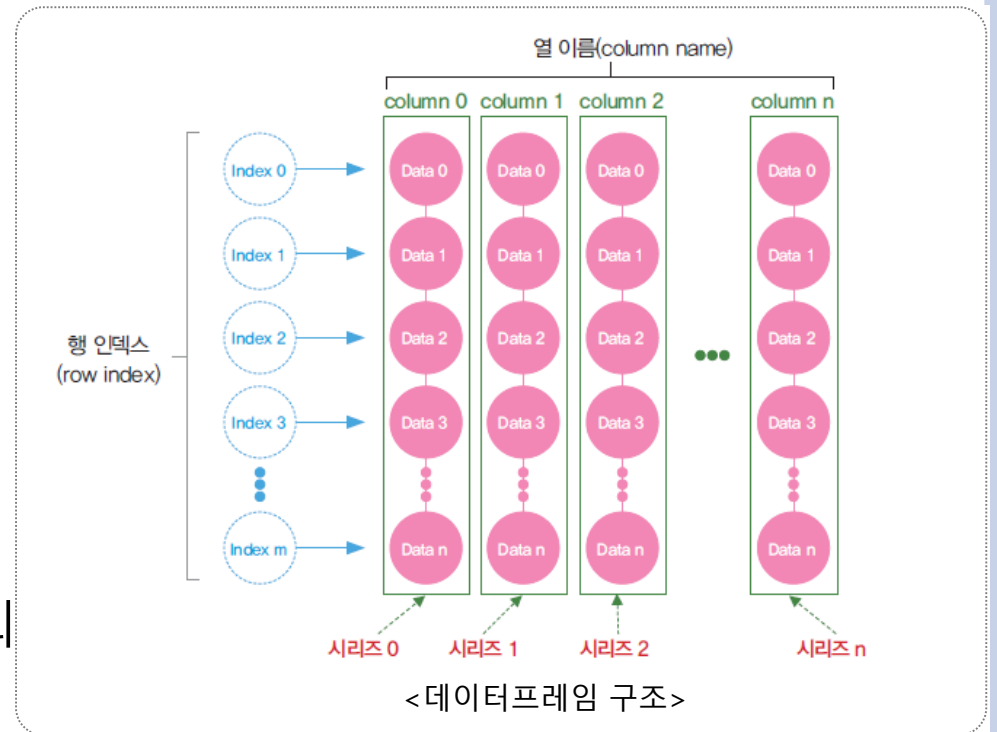
```
생년월일      2010-05-01
성별          여
dtype: object
```

<실행결과>

데이터프레임(DataFrame)

■ 데이터프레임(DataFrame)

- 2차원 배열.
- 열은 시리즈 객체.
- 여러 개의 열들이 같은 행 인덱스를 기준으로 줄지어 결합된 2차원 벡터 또는 행렬.
- 행 인덱스(row index)와 열 이름(column name 또는 column label)으로 구분.
- 각 열은 공통의 속성을 갖는 일련의 데이터를 나타냄.
- 각 행은 개별 관측대상에 대한 다양한 속성 데이터들의 모음인 레코드(record).




데이터프레임(DataFrame)

[예시]

다음 주식종목 리스트에서, 각 행은 하나의 주식종목에 관한 관측값(observation)을 나타낸다.

각 열은 종목코드, 회사이름, 액면가, 총주식수 등 공통의 속성이거나 범주를 나타내는데, 보통 변수(variable)로 활용된다



종목 코드	회사 이름	액면가	총 주식수
005930	삼성전자	100원	5,970백만 주
017670	SK텔레콤	500원	81백만 주
005380	현대자동차	5000원	214백만 주

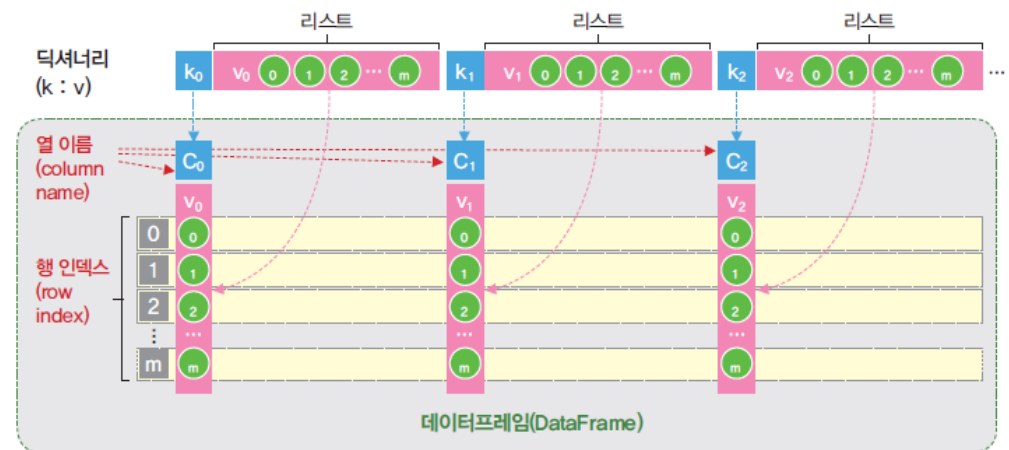
<주식 종목 리스트>

데이터프레임(DataFrame)

■ 데이터프레임 만들기

- 같은 길이(원소의 개수가 동일한)의 배열 여러 개가 필요.
- 데이터프레임은 여러 개의 시리즈(열, column)를 모아 놓은 집합.
- 판다스 DataFrame() 함수를 사용.
- 여러 개의 리스트를 원소로 갖는 딕셔너리를 함수에 전달하는 방식을 주로 활용.

```
pandas.DataFrame(딕셔너리)
```



<딕셔너리의 데이터프레임 변환>

데이터프레임(DataFrame)

- 예제 : 딕셔너리의 데이터프레임 변환

```
import pandas as pd
```

```
# 열이름을 key로 하고, 리스트를 value로 갖는 딕셔너리 정의(2차원 배열)
```

```
dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9],  
             'c3':[10,11,12], 'c4':[13,14,15]}
```

```
# 판다스 DataFrame() 함수로 딕셔너리를 데이터프레임으로 변환. 변수 df에 저장.
```

```
df = pd.DataFrame(dict_data)
```

```
# df의 자료형 출력
```

```
print(type(df))
```

```
print('\n')
```

```
# 변수 df에 저장되어 있는 데이터프레임 객체를 출력
```

```
print(df)
```

```
<class 'pandas.core.frame.DataFrame'>
```

	c0	c1	c2	c3	c4
0	1	4	7	10	13
1	2	5	8	11	14
2	3	6	9	12	15

<실행결과>

■ 행 인덱스/열 이름 설정

- 데이터프레임의 행 인덱스와 열 이름을 사용자가 지정 가능

```
pandas.DataFrame(2차원배열,  
                  index=행 인덱스 배열,  
                  columns=열 이름 배열)
```

■ 행 인덱스/열 이름 설정

```
# 행 인덱스/열 이름 지정하여, 데이터프레임 만들기  
df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']],  
                  index=['준서', '예은'],  
                  columns=['나이', '성별', '학교'])
```

```
# 행 인덱스, 열 이름 확인하기  
print(df) #데이터프레임  
print('\n')  
print(df.index) #행 인덱스  
print('\n')  
print(df.columns) #열 이름  
print('\n')
```

```
# 행 인덱스, 열 이름 변경하기  
df.index=['학생1', '학생2']  
df.columns=['연령', '남녀', '소속']
```

```
print(df) #데이터프레임  
print('\n')  
print(df.index) #행 인덱스  
print('\n')  
print(df.columns) #열 이름
```

데이터프레임(DataFrame)

- 예제 : 행/열 이름 변경(rename) 메소드

```
# 행 인덱스/열 이름 지정하여, 데이터프레임 만들기
df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']],
                   index=['준서', '예은'],
                   columns=['나이', '성별', '학교'])

# 데이터프레임 df 출력
print(df)
print("\n")

# 열 이름 중, '나이'를 '연령'으로, '성별'을 '남녀'로, '학교'를 '소속'으로 바꾸기
df.rename(columns={'나이': '연령', '성별': '남녀', '학교': '소속'}, inplace=True)

# df의 행 인덱스 중에서, '준서'를 '학생1'로, '예은'을 '학생2'로 바꾸기
df.rename(index={'준서': '학생1', '예은': '학생2'}, inplace=True)

# df 출력(변경 후)
print(df)
```

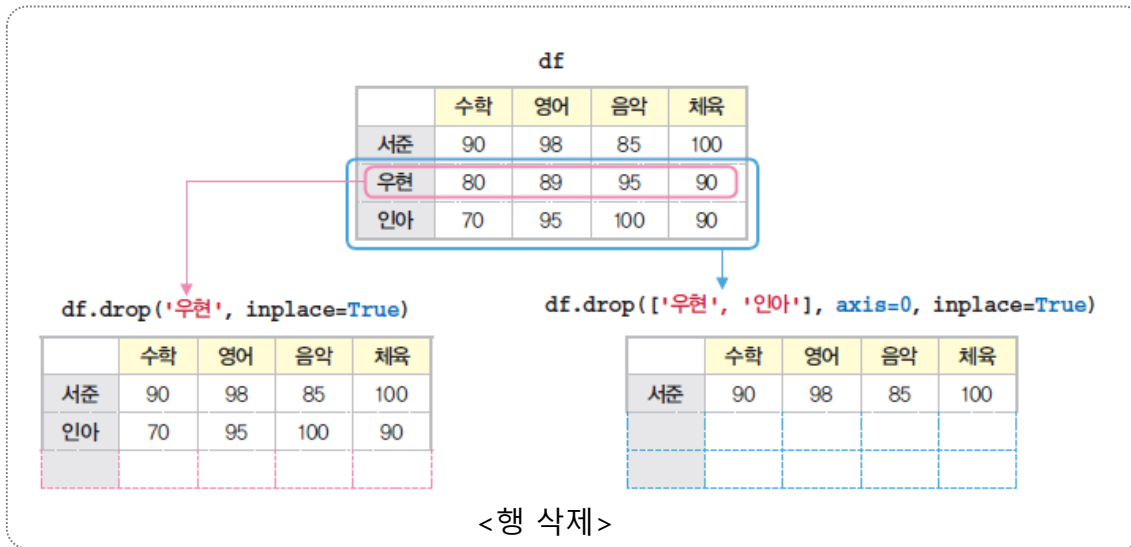
	나이	성별	학교
준서	15	남	덕영중
예은	17	여	수리중

	연령	남녀	소속
학생1	15	남	덕영중
학생2	17	여	수리중

데이터프레임(DataFrame)

■ 행/열 삭제

- 행 삭제 : DataFrame 객체.drop(행 인덱스 또는 배열, axis=0)
- 열 삭제 : DataFrame 객체.drop(열 이름 또는 배열, axis=1)
- drop()메소드로 삭제
- 원본 객체를 직접 변경하기 위해서는 inplace=True 옵션 추가



데이터프레임(DataFrame)

- 예제 : 행 삭제(drop) 메소드

```
# DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
exam_data = {'수학' : [ 90, 80, 70], '영어' : [ 98, 89, 95],
              '음악' : [ 85, 95, 100], '체육' : [ 100, 90, 90]}
```

```
df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])
print(df)
print('\n')
```

```
# 데이터프레임 df를 복제하여 변수 df2에 저장. df2의 1개 행(row)을 삭제
df2 = df[:]
df2.drop('우현', inplace=True)
print(df2)
print('\n')
```

```
# 데이터프레임 df를 복제하여 변수 df3에 저장. df3의 2개 행(row)을 삭제
df3 = df[:]
df3.drop(['우현', '인아'], axis=0, inplace=True)
print(df3)
```

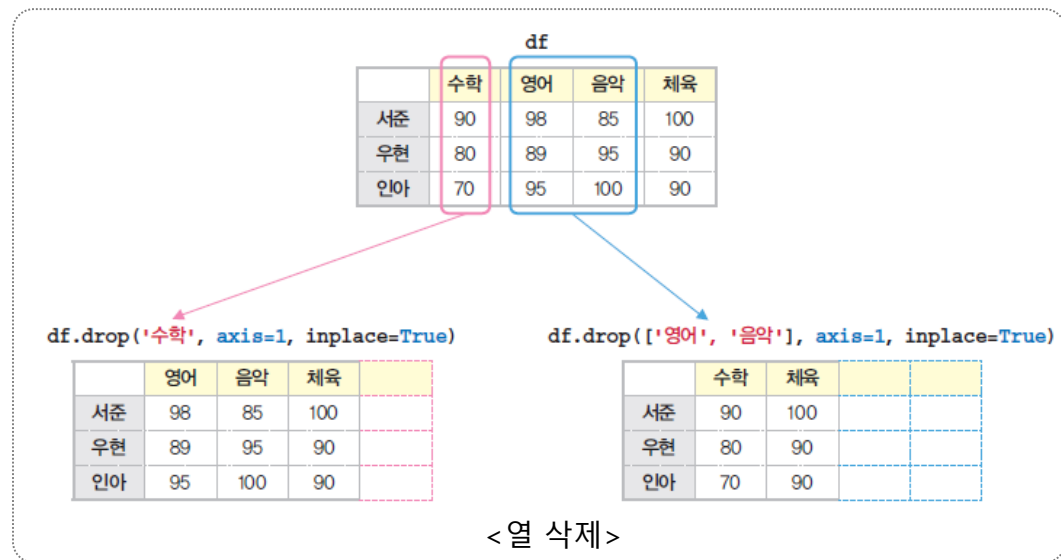
	수학	영어	음악	체육
서준	90	98	85	100
우현	80	89	95	90
인아	70	95	100	90

	수학	영어	음악	체육
서준	90	98	85	100
인아	70	95	100	90

	수학	영어	음악	체육
서준	90	98	85	100

<실행결과>

데이터프레임(DataFrame)



■ 예제 : 행 삭제(drop() 메소드)

```
# df4의 2개 열(column)을 삭제
df4 = df[:]
df4.drop('수학', axis=1, inplace=True)
print(df4)
print('\n')
```

```
# df5의 2개 열(column)을 삭제
df5 = df[:]
df5.drop(['영어', '음악'], axis=1, inplace=True)
print(df5)
```

	영어	음악	체육
서준	98	85	100
우현	89	95	90
인아	95	100	90

	수학	체육
서준	90	100
우현	80	90
인아	70	90

<실행결과>

데이터프레임(DataFrame)

■ 행 선택

- **loc**과 **iloc** 인덱서를 사용.
- 인덱스 이름을 기준으로 행을 선택할 때는 **loc**을 이용하고, 정수형 위치 인덱스를 사용할 때는 **iloc**을 이용.

구분	loc	iloc
탐색 대상	인덱스 이름(index label)	정수형 위치 인덱스(integer position)
범위 지정	가능(범위의 끝 포함) 예) ['a':'c'] → 'a', 'b', 'c'	가능(범위의 끝 제외) 예) [3:7] → 3, 4, 5, 6 (* 7 제외)

< loc와 iloc >

```
label1 = df.loc['서준']    # loc 인덱서 활용
position1 = df.iloc[0]    # iloc 인덱서 활용
print(label1)
print('\n')
print(position1)
print('\n')
```

```
# 행 인덱스를 사용하여 2개 이상의 행 선택
label2 = df.loc[['서준', '우현']]
position2 = df.iloc[[0, 1]]
print(label2)
print('\n')
print(position2)
print('\n')
```

```
# 행 인덱스의 범위를 지정하여 행 선택
label3 = df.loc['서준':'우현']
position3 = df.iloc[0:1]
print(label3)
print('\n')
print(position3)
```

데이터프레임(DataFrame)

■ 열 선택

DataFrame 객체['열이름'] 또는 DataFrame 객체.열이름

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90

df['수학']

	수학
0	90
1	80
2	70

시리즈

df[['음악', '체육']]

	음악	체육
0	85	100
1	95	90
2	100	90

데이터프레임

< 데이터프레임 열 선택 >

```
# '수학' 점수 데이터만 선택. 변수 math1에 저장
math1 = df['수학']
print(math1)
print(type(math1))
print('\n')
```

```
# '영어' 점수 데이터만 선택. 변수 english에 저장
english = df.영어
print(english)
print(type(english))
print('\n')
```

```
# '음악', '체육' 점수 데이터를 선택. 변수 music_gym에 저장
music_gym = df[['음악', '체육']]
print(music_gym)
print(type(music_gym))
print('\n')
```

```
# '수학' 점수 데이터만 선택. 변수 math2에 저장
math2 = df[['수학']]
print(math2)
print(type(math2))
```

데이터프레임(DataFrame)

■ 원소 선택

〈원소 1개 선택〉

	0	1	2	3
	수학	영어	음악	체육
0 서준	90	98	85	100
1 우현	80	89	95	90
2 인아	70	95	100	90

행, 열 좌표 { `df.loc['서준', '음악']` } 0 서준 음악 원소
`df.iloc[0, 2]`

〈원소 2개(시리즈) 선택〉

	0	1	2	3
	수학	영어	음악	체육
0 서준	90	98	85	100
1 우현	80	89	95	90
2 인아	70	95	100	90

행, 열 좌표 { `df.loc['서준', ['음악', '체육']]` } 0 서준 음악 체육 시리즈
`df.iloc[0, [2, 3]]`
`df.loc['서준', '음악' : '체육']`
`df.iloc[0, 2:]`

〈데이터프레임(df)의 일부분 선택〉

	0	1	2	3
	수학	영어	음악	체육
0 서준	90	98	85	100
1 우현	80	89	95	90
2 인아	70	95	100	90

행, 열 좌표 { `df.loc[['서준', '우현'], ['음악', '체육']]` } 0 서준 85 100
`df.iloc[[0, 1], [2, 3]]`
`df.loc['서준': '우현', '음악': '체육']`
`df.iloc[0:2, 2:]`
1 우현 95 90
데이터프레임

< 데이터프레임 [행, 열] 데이터 선택 >

데이터프레임(DataFrame)

```
# DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
exam_data = {'이름' : [ '서준', '우현', '인아'],
              '수학' : [ 90, 80, 70],
              '영어' : [ 98, 89, 95],
              '음악' : [ 85, 95, 100],
              '체육' : [ 100, 90, 90]}
df = pd.DataFrame(exam_data)

# '이름' 열을 새로운 인덱스로 지정, df 객체에 변경사항 반영
df.set_index('이름', inplace=True)
print(df)
print('\n')

# 데이터프레임 df의 특정 원소 1개 선택 ('서준'의 '음악' 점수)
a = df.loc['서준', '음악']
print(a)
b = df.iloc[0, 2]
print(b)
print('\n')
```

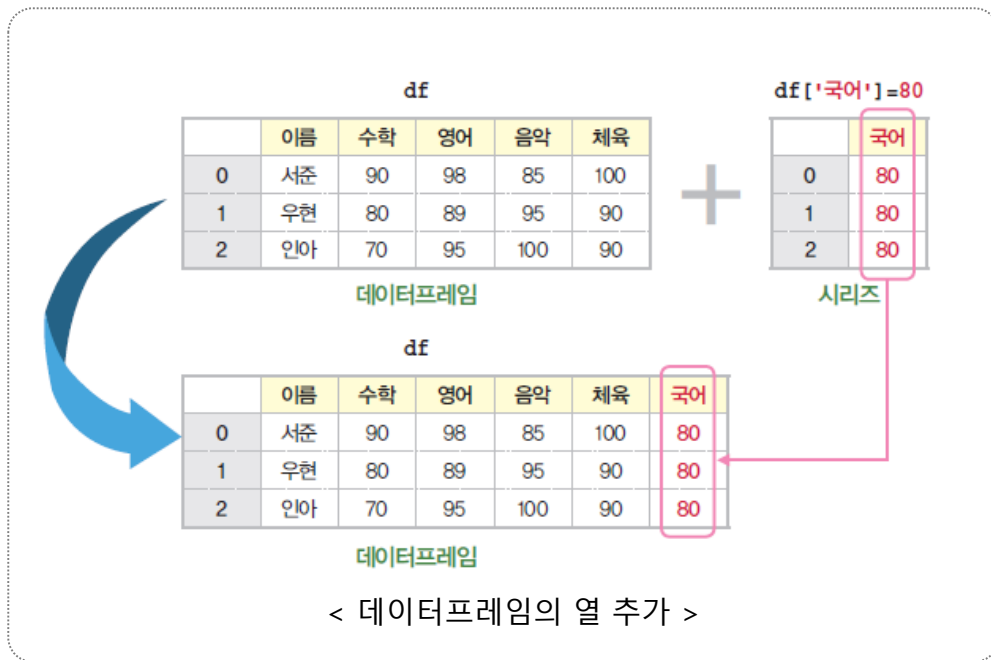
```
# 데이터프레임 df의 특정 원소 2개 이상 선택
c = df.loc['서준', ['음악', '체육']]
print(c)
d = df.iloc[0, [2, 3]]
print(d)
e = df.loc['서준', '음악':'체육']
print(e)
f = df.iloc[0, 2:]
print(f)
print('\n')

# df의 2개 이상의 행과 열로부터 원소
g = df.loc[['서준', '우현'], ['음악', '체육']]
print(g)
h = df.iloc[[0, 1], [2, 3]]
print(h)
i = df.loc['서준':'우현', '음악':'체육']
print(i)
j = df.iloc[0:2, 2:]
print(j)
```

데이터프레임(DataFrame)

■ 열 추가

DataFrame 객체[‘추가하려는 열 이름’] = 데이터 값



```
exam_data = {'이름' : [ '서준', '우현', '인아'],  
             '수학' : [ 90, 80, 70],  
             '영어' : [ 98, 89, 95],  
             '음악' : [ 85, 95, 100],  
             '체육' : [ 100, 90, 90]}
```

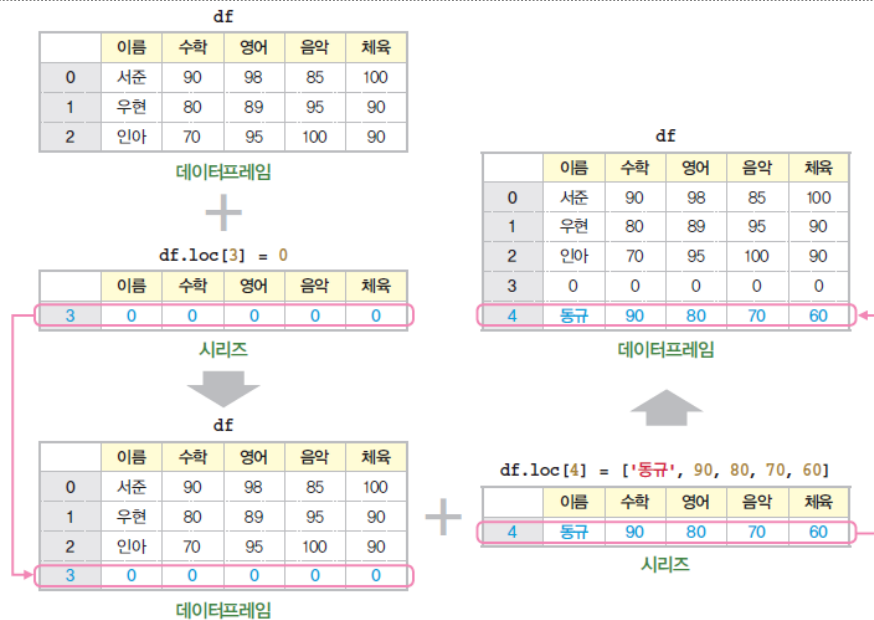
```
df = pd.DataFrame(exam_data)  
print(df)  
print('\n')
```

```
# 데이터프레임 df에 '국어' 점수 열(column)을 추가.  
df['국어'] = 80  
print(df)
```

데이터프레임(DataFrame)

■ 행 추가

DataFrame 객체.loc[‘새로운 행 이름’] = 데이터 값



< 데이터프레임의 행 추가 >

```
# 새로운 행(row)을 추가 - 같은 원소 값을 입력
df.loc[3] = 0
print(df)
print('\n')
```

```
# 새로운 행(row)을 추가 - 원소 값 여러 개의 배열 입력
df.loc[4] = ['동규', 90, 80, 70, 60]
print(df)
print('\n')
```

```
# 새로운 행(row)을 추가 - 기존 행을 복사
df.loc['행5'] = df.loc[3]
print(df)
```

데이터프레임(DataFrame)

■ 원소값 변경

DataFrame 객체의 일부분 또는 원소 선택 = 새로운 값

```
# 데이터프레임 df의 특정 원소를 변경하는 방법
df.iloc[0][3] = 80
print(df)
print('\n')
```

```
df.loc['서준']['체육'] = 90
print(df)
print('\n')
```

```
df.loc['서준', '체육'] = 100
print(df)
print('\n')
```

```
# 데이터프레임 df의 원소 여러 개를 변경하는 방법
df.loc['서준', ['음악', '체육']] = 50
print(df)
print('\n')
```

```
df.loc['서준', ['음악', '체육']] = 100, 50
print(df)
```

데이터프레임(DataFrame)

■ 행/열이 위치 바꾸기

DataFrame 객체.transpose() 또는 DataFrame 객체.T

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90

행 → 열

	0	1	2
이름	서준	우현	인아
수학	90	80	70
영어	98	89	95
음악	85	95	100
체육	100	90	90

열 → 행

< 행/열 바꾸기 >

데이터프레임 df를 전치하기 (메소드 활용)

```
df = df.transpose()
print(df)
print('\n')
```

데이터프레임 df를 다시 전치하기 (클래스 속성 활용)

```
df = df.T
print(df)
```

데이터프레임(DataFrame)

■ 특정 열을 행 인덱스로 설정

- set_index() 메소드 사용
- 기존 행 인덱스는 삭제 됨

df

	이름	수학	영어	음악	체육	국어
0	서준	90	98	85	100	80
1	우현	80	89	95	90	80
2	인아	70	95	100	90	80

행 주소(인덱스)

df.set_index(['이름'])

이름	수학	영어	음악	체육	국어
서준	90	98	85	100	80
우현	80	89	95	90	80
인아	70	95	100	90	80

서로 다른 객체

< 특정 열을 행 인덱스로 설정 >

```
# DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
exam_data = {'이름' : [ '서준', '우현', '인아'],
              '수학' : [ 90, 80, 70],
              '영어' : [ 98, 89, 95],
              '음악' : [ 85, 95, 100],
              '체육' : [ 100, 90, 90]}
```

```
df = pd.DataFrame(exam_data)
print(df)
print('\n')
```

```
# 특정 열(column)을 데이터프레임의 행 인덱스(index)로 설정
ndf = df.set_index(['이름'])
print(ndf)
print('\n')
ndf2 = ndf.set_index('음악')
print(ndf2)
print('\n')
ndf3 = ndf.set_index(['수학', '음악'])
print(ndf3)
```

산술 연산

■ 데이터프레임 연산

- 데이터프레임 vs 숫자
 - 데이터프레임에 어떤 숫자를 더하면, 모든 원소에 숫자를 더함
 - 덧셈, 뺄셈, 곱셈, 나눗셈 모두 가능
 - 기존 데이터프레임의 형태를 그대로 유지한 채, 원소 값만 새로운 값으로 바뀜

```
exam_data = {'수학' : [ 90, 80, 70],
             '영어' : [ 98, 89, 95],
             '음악' : [ 85, 95, 100],
             '체육' : [ 100, 90, 90]}

df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])
print(df)
print('\n')

df['수학2'] = df['수학']+5
print(df)
print('\n')

df['총점'] = df['수학'] + df['영어'] + df['음악'] + df['체육']
df['평균'] = df['총점']/4
df
```

	수학	영어	음악	체육	수학2	총점	평균
서준	90	98	85	100	95	373	93.25
우현	80	89	95	90	85	354	88.50
인아	70	95	100	90	75	355	88.75

데이터 입출력

■ 외부 파일 읽기

- 판다스는 다양한 형태의 외부 파일을 읽어와서 데이터프레임으로 변환하는 함수를 제공
- 데이터프레임을 다양한 유형의 파일로 저장 가능

File Format	Reader	Writer
CSV	read_csv	to_csv
JSON	read_json	to_json
HTML	read_html	to_html
Local clipboard	read_clipboard	to_clipboard
MS Excel	read_excel	to_excel
HDF5 Format	read_hdf	to_hdf
SQL	read_sql	to_sql

< Pandas 외부 파일 입출력 도구 >

데이터 입출력

■ CSV 파일

- 데이터 값을 쉼표(,)로 구분하고 있다는 의미
- CSV(comma-separated values)라고 부르는 텍스트 파일
- 쉼표(,)로 열을 구분하고 줄바꿈으로 행 구분

CSV 파일 -> 데이터프레임 : `pandas.read_csv('파일경로(이름)')`

옵션	설명
path	파일의 위치(파일명 포함), URL
sep(또는 delimiter)	텍스트 데이터를 필드별로 구분하는 문자
header	열 이름으로 사용될 행의 번호(기본값은 0) header가 없고 첫 행부터 데이터가 있는 경우 None으로 지정 가능
index_col	행 인덱스로 사용할 열의 번호 또는 열 이름
names	열 이름으로 사용할 문자열의 리스트
skiprows	처음 몇 줄을 skip할 것인지 설정(숫자 입력) skip하려는 행의 번호를 담은 리스트로 설정 가능(예: [1, 3, 5])
parse_dates	날짜 텍스트를 datetime64로 변환할 것인지 설정(기본값은 False)
skip_footer	마지막 몇 줄을 skip할 것인지 설정(숫자 입력)
encoding	텍스트 인코딩 종류를 지정(예: 'utf-8')

< read_csv() 함수 옵션 >

데이터 입출력

read_csv() 주요 옵션

<CSV 파일>

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

* header 옵션

- '열 이름'이 되는 행을 지정
- `read_csv(file, header=?)`

① **header=0** (기본 값: 0행을 열 지정): `df = read_csv(file)`

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

→

	c0	c1	c2	c3
0	0	1	4	7
1	1	2	5	8
2	2	3	6	9

② **header=1** (1행을 열 지정): `df = read_csv(file, header=1)`

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

→

	0	1	4	7
0	1	2	5	8
1	2	3	6	9

③ **header=None** (행을 열 지정하지 않음): `df = read_csv(file, header=None)`

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

→

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

< read_csv() 함수 - header 옵션 비교 >

<CSV 파일>

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

* index_col 옵션

- '행 주소'가 되는 열을 지정
- `read_csv(file, index_col=?)`

① **index_col=False** (인덱스 지정하지 않음)

: `df = read_csv(file, index_col=False)`

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

→

	c0	c1	c2	c3
0	0	1	4	7
1	1	2	5	8
2	2	3	6	9

② **index_col='c0'** ('c0'열을 인덱스 지정)

: `df = read_csv(file, index_col='c0')`

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

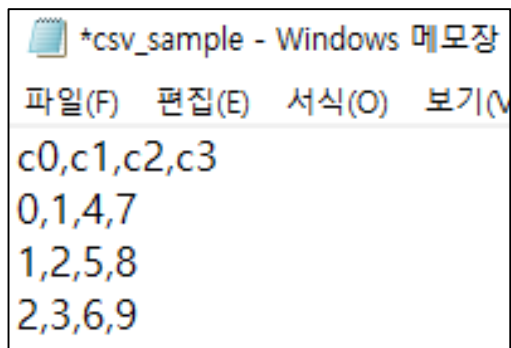
→

	c1	c2	c3
0	1	4	7
1	2	5	8
2	3	6	9

< read_csv() 함수 - index_col 옵션 비교 >

데이터 입출력

■ read_csv() 예제



c0,c1,c2,c3
0,1,4,7
1,2,5,8
2,3,6,9

```
import pandas as pd

# 파일경로를 찾고, 변수 file_path에 저장
file_path = 'data/csv_sample.csv'

# read_csv() 함수로 데이터프레임 변환. 변수 df1에 저장
df1 = pd.read_csv(file_path)
print(df1)
print('\n')

# header=None 옵션
df2 = pd.read_csv(file_path, header=None)
print(df2)
print('\n')

# index_col=None 옵션
df3 = pd.read_csv(file_path, index_col=None)
print(df3)
print('\n')

# index_col='c0' 옵션
df4 = pd.read_csv(file_path, index_col='c0')
print(df4)
```

	c0	c1	c2	c3
0	0	1	4	7
1	1	2	5	8
2	2	3	6	9

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

	c0	c1	c2	c3
0	0	1	4	7
1	1	2	5	8
2	2	3	6	9

	c1	c2	c3
c0			
0	1	4	7
1	2	5	8
2	3	6	9

<실행결과>

데이터 저장하기

■ CSV 파일로 저장하기

데이터프레임 객체.to_csv('파일경로(이름)')

■ JSON 파일로 저장하기

데이터프레임 객체.to_json('파일경로(이름)')

■ Excel 파일로 저장하기

데이터프레임 객체.to_excel('파일경로(이름)')

```
data = {'name' : [ 'Jerry', 'Riah', 'Paul'],  
        'algol' : [ "A", "A+", "B"],  
        'basic' : [ "C", "B", "B+"],  
        'c++' : [ "B+", "C", "C+"],  
        }
```

```
df = pd.DataFrame(data)  
df.set_index('name', inplace=True)    #name 열을 인덱스  
로 지정  
print(df)
```

```
# to_csv() 메소드를 사용하여 csv 파일로 내보내기.  
df.to_csv("data/df_sample.csv")
```

```
# to_json() 메소드를 사용하여 json 파일로 내보내기.  
df.to_json("data/df_sample.json")
```

```
# to_excel() 메소드를 사용하여 excel 파일로 내보내기.  
df.to_excel("data/df_sample.xlsx")
```

Pandas 활용 실습 예제(titanic data set)

```
import pandas as pd
```

```
# titanic 데이터셋 가져오기
```

```
df= pd.read_csv('data/titanic.csv')
```

```
df
```

- Survived - 생존 여부 (0 = 사망, 1 = 생존)
- Pclass - 티켓 클래스 (1 = 1등석, 2 = 2등석, 3 = 3등석)
- Sex - 성별
- Age - 나이
- SibSp - 함께 탑승한 자녀 / 배우자 의 수
- Parch - 함께 탑승한 부모님 / 아이들 의 수
- Ticket - 티켓 번호
- Fare - 탑승 요금
- Cabin - 수하물 번호
- Embarked - 선착장 (C = Cherbourg, Q = Queenstown, S = Southampton)

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

Pandas 활용 실습 예제(titanic data set)

head, tail 함수

- 데이터 전체가 아닌, 일부(처음부터, 혹은 마지막부터)를 간단히 보기 위한 함수

```
1 df.head(n=3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

```
1 df.tail(n=10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257	7.8958	NaN	S
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.5167	NaN	S
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.5000	NaN	S
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500	NaN	S
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250	NaN	Q

Pandas 활용 실습 예제(titanic data set)

dataframe 데이터 파악하기

- shape 속성 (row, column)
- describe 함수 - 숫자형 데이터의 통계치 계산
- info 함수 - 데이터 타입, 각 아이템의 개수 등 출력

```
1 df.shape
```

```
(891, 12)
```

```
1 df.describe()
```

```
:
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   PassengerId           891 non-null    int64  
1   Survived              891 non-null    int64  
2   Pclass                891 non-null    int64  
3   Name                  891 non-null    object  
4   Sex                   891 non-null    object  
5   Age                   714 non-null    float64 
6   SibSp                 891 non-null    int64  
7   Parch                 891 non-null    int64  
8   Ticket                891 non-null    object  
9   Fare                  891 non-null    float64 
10  Cabin                 204 non-null    object  
11  Embarked              889 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Pandas 활용 실습 예제(titanic data set)

변수(column) 사이의 상관관계수(correlation)

- corr함수를 통해 상관관계수 연산 (-1, 1 사이의 결과)
 - 연속성(숫자형)데이터에 대해서만 연산
 - 인과관계를 의미하진 않음

```
1 df.corr()
```

	PassengerId	Survived	Pclass	Fare10	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.012658	0.036847	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	0.257307	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.549500	-0.369226	0.083081	0.018443	-0.549500
Fare10	0.012658	0.257307	-0.549500	1.000000	0.096067	0.159651	0.216225	1.000000
Age	0.036847	-0.077221	-0.369226	0.096067	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.057527	-0.035322	0.083081	0.159651	-0.308247	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	0.216225	-0.189119	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	1.000000	0.096067	0.159651	0.216225	1.000000

```
1 df[['Survived', 'Fare10']].corr()
```

	Survived	Fare10
Survived	1.000000	0.257307
Fare10	0.257307	1.000000

Pandas 활용 실습 예제(titanic data set)

NaN 값 확인

- info함수를 통하여 개수 확인
- isna함수를 통해 boolean 타입으로 확인

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 100 to 990
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype  
---  -
 0   PassengerId   891 non-null    int64  
 1   Survived      891 non-null    int64  
 2   Pclass        891 non-null    int64  
 3   Fare10        891 non-null    float64 
 4   Name          891 non-null    object  
 5   Sex           891 non-null    object  
 6   Age           714 non-null    float64 
 7   SibSp         891 non-null    int64  
 8   Parch         891 non-null    int64  
 9   Ticket        891 non-null    object  
10   Fare          891 non-null    float64 
11   Cabin         204 non-null    object  
12   Embarked      889 non-null    object  
dtypes: float64(3), int64(5), object(5)
memory usage: 137.5+ KB
```

```
1 df.isnull()
```

	PassengerId	Survived	Pclass	Fare10	Name	Sex	Age	SibSp
100	False	False	False	False	False	False	False	False
101	False	False	False	False	False	False	False	False
102	False	False	False	False	False	False	False	False
103	False	False	False	False	False	False	False	False
104	False	False	False	False	False	False	False	False
...
986	False	False	False	False	False	False	False	False

```
1 df.isna()
```

	PassengerId	Survived	Pclass	Fare10	Name	Sex	Age	SibSp
100	False	False	False	False	False	False	False	False
101	False	False	False	False	False	False	False	False
102	False	False	False	False	False	False	False	False
103	False	False	False	False	False	False	False	False
104	False	False	False	False	False	False	False	False
...
986	False	False	False	False	False	False	False	False

```
1 df['Age'].isna()
```

```
100    False
101    False
102    False
103    False
104    False
...
986    False
987    False
988     True
989    False
990    False
```

Name: Age, Length: 891, dtype: bool

Pandas 활용 실습 예제(titanic data set)

NaN 처리 방법

- 데이터에서 삭제
 - dropna 함수
- 다른 값으로 치환
 - fillna 함수
- NaN 데이터 삭제하기

```
1 df.dropna()
```

	PassengerId	Survived	Pclass	Fare10	Name
101	2	1	1	7.12833	Cumings, Mrs. John Bradley (Florence Briggs Th...
103	4	1	1	5.31000	Futrelle, Mrs. Jacques Heath (Lily May Peel)
106	7	0	1	5.18625	McCarthy, Mr. Timothy J
110	11	1	3	1.67000	Sandstrom, Miss. Marguerite Rut
111	12	1	1	2.65500	Bonnell, Miss. Elizabeth

```
1 df.dropna(subset=['Age', 'Cabin'])
```

	PassengerId	Survived	Pclass	Fare10	Name
101	2	1	1	7.12833	Cumings, Mrs. John Bradley (Florence Briggs Th...
103	4	1	1	5.31000	Futrelle, Mrs. Jacques Heath (Lily May Peel)
106	7	0	1	5.18625	McCarthy, Mr. Timothy J
110	11	1	3	1.67000	Sandstrom, Miss. Marguerite Rut
111	12	1	1	2.65500	Bonnell, Miss. Elizabeth

Pandas 활용 실습 예제(titanic data set)

```
1 df.dropna(axis=1)
```

	PassengerId	Survived	Pclass	Fare10	Name	Sex	SibSp	Parch	Ticket	Fare
100	1	0	3	0.72500	Braund, Mr. Owen Harris	male	1	0	A/5 21171	7.2500
101	2	1	1	7.12833	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	1	0	PC 17599	71.2833
102	3	1	3	0.79250	Heikkinen, Miss. Laina	female	0	0	STON/O2. 3101282	7.9250
103	4	1	1	5.31000	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	1	0	113803	53.1000
104	5	0	3	0.80500	Allen, Mr. William Henry	male	0	0	373450	8.0500
...
986	887	0	2	1.30000	Montvila, Rev. Juozas	male	0	0	211536	13.0000
987	888	1	1	3.00000	Graham, Miss. Margaret Edith	female	0	0	112053	30.0000
988	889	0	3	2.34500	Johnston, Miss. Catherine Helen "Carrie"	female	1	2	W./C. 6607	23.4500
989	890	1	1	3.00000	Behr, Mr. Karl Howell	male	0	0	111369	30.0000
990	891	0	3	0.77500	Dooley, Mr. Patrick	male	0	0	370376	7.7500

891 rows × 10 columns

Pandas 활용 실습 예제(titanic data set)

- NaN 값 대체하기
 - 평균으로 대체하기
 - 생존자/사망자 별 평균으로 대체하기

```
1 df['Age'].fillna(df['Age'].mean())
```

100	22.000000
101	38.000000
102	26.000000
103	35.000000
104	35.000000
...	...
986	27.000000
987	19.000000
988	29.699118
989	26.000000
990	32.000000

Name: Age, Length: 891, dtype: float64

```
1 # 생존자 나이 평균
2 mean1 = df[df['Survived'] == 1]['Age'].mean()
3
4 # 사망자 나이 평균
5 mean0 = df[df['Survived'] == 0]['Age'].mean()
6
7 print(mean1, mean0)
```

28.343689655172415 30.62617924528302

```
1 df[df['Survived'] == 1]['Age'].fillna(mean1)
2 df[df['Survived'] == 0]['Age'].fillna(mean0)
```

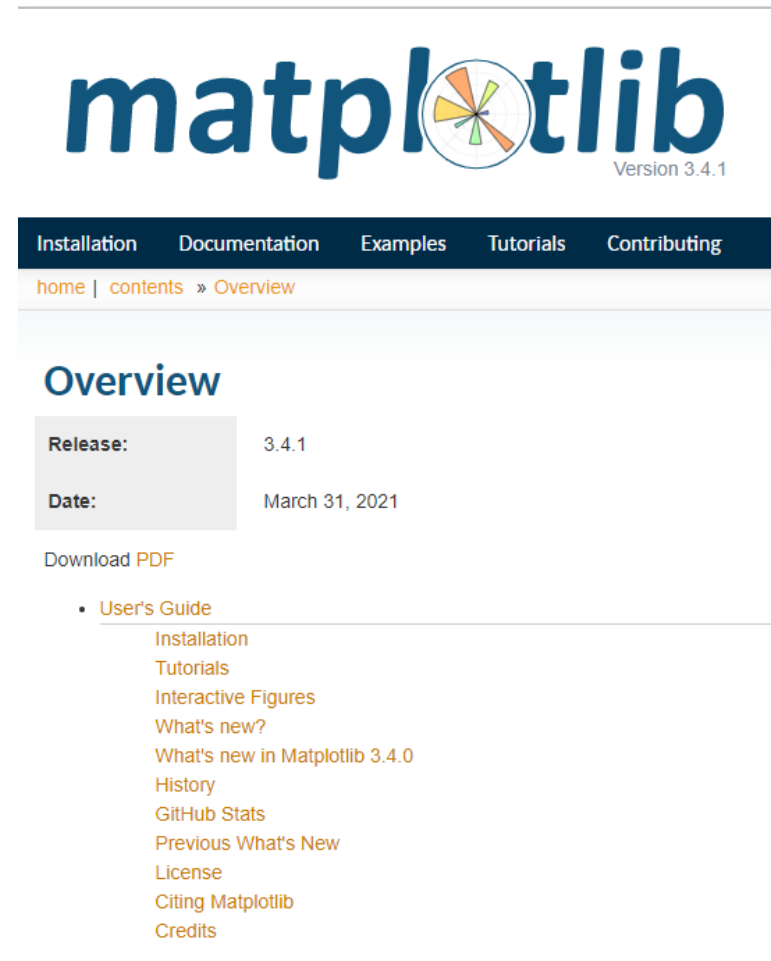
100	22.000000
104	35.000000
105	30.626179
106	54.000000
107	2.000000
...	...
984	25.000000
985	39.000000
986	27.000000
988	30.626179
990	32.000000

Name: Age, Length: 549, dtype: float64

데이터 시각화

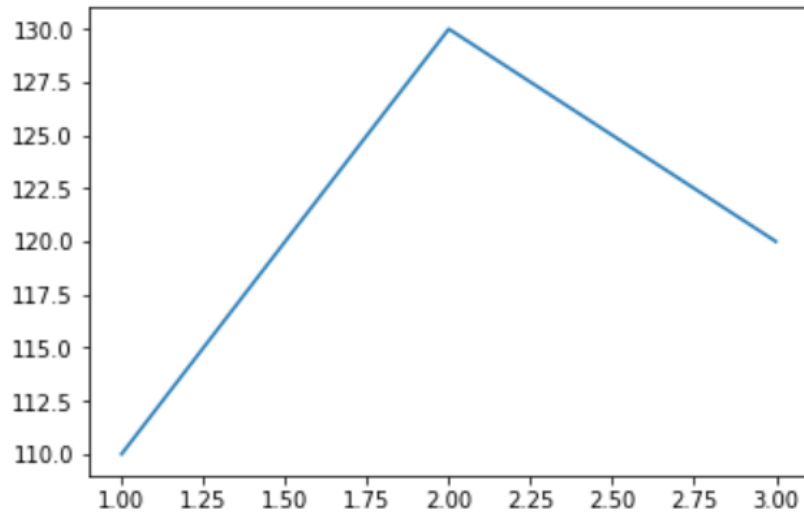
Matplotlib 개요

- 파이썬에서 데이터를 차트나 플롯(Plot)으로 그려주는 라이브러리 패키지
- 가장 많이 사용되는 데이터 시각화(Data Visualization) 패키지
- 라인 플롯, 바 차트, 파이차트, 히스토그램, Box Plot, Scatter Plot 등을 비롯하여 다양한 차트와 플롯 스타일을 지원



Matplotlib 기본 사용법

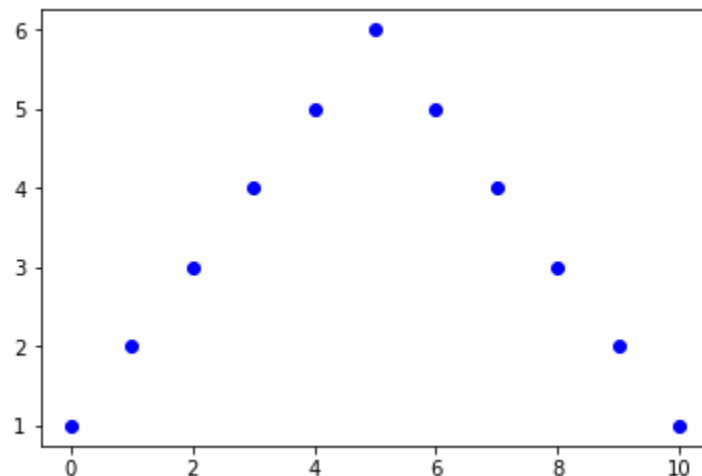
```
plt.plot([1,2,3], [110,130,120])  
plt.show()
```



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

```
%matplotlib inline
```

```
plt.figure()  
plt.plot([1,2,3,4,5,6,5,4,3,2,1], "ob")  
plt.show()
```

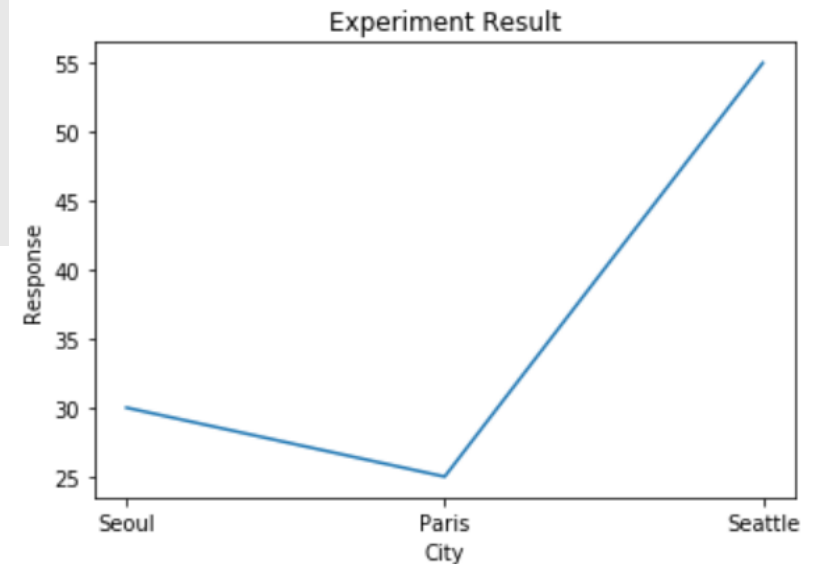


Matplotlib 기본 사용법

■ 제목과 축 레이블

- 플롯에 X,Y 축 레이블이나 제목을 붙이기 위해서는 plt.xlabel(축이름), plt.ylabel(축이름), plt.title(제목) 등의 함수를 사용

```
plt.plot(["Seoul", "Paris", "Seattle"], [30, 25, 55])  
plt.xlabel('City')  
plt.ylabel('Response')  
plt.title('Experiment Result')  
plt.show()
```

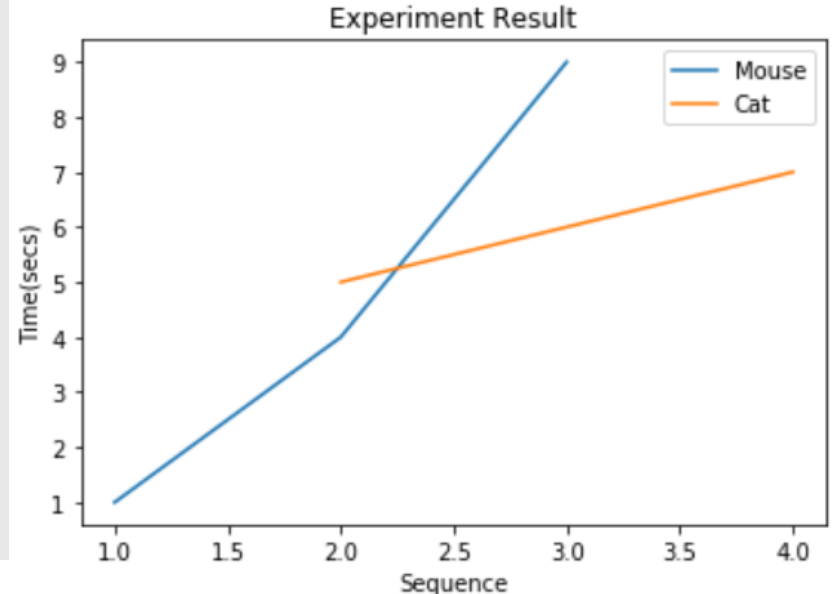


Matplotlib 기본 사용법

■ 범례 추가

- 플롯에 여러 개의 라인들을 추가하기 위해서는 plt.plot()을 plt.show() 이전에 여러 번 사용
- 각 라인에 대한 범례를 추가하기 위해서는 plt.legend([라인1범례, 라인2범례]) 함수를 사용하여 각 라인에 대한 범례를 순서대로 지정

```
plt.plot([1,2,3], [1,4,9])  
plt.plot([2,3,4],[5,6,7])  
plt.xlabel('Sequence')  
plt.ylabel('Time(secs)')  
plt.title('Experiment Result')  
plt.legend(['Mouse', 'Cat'])  
plt.show()
```

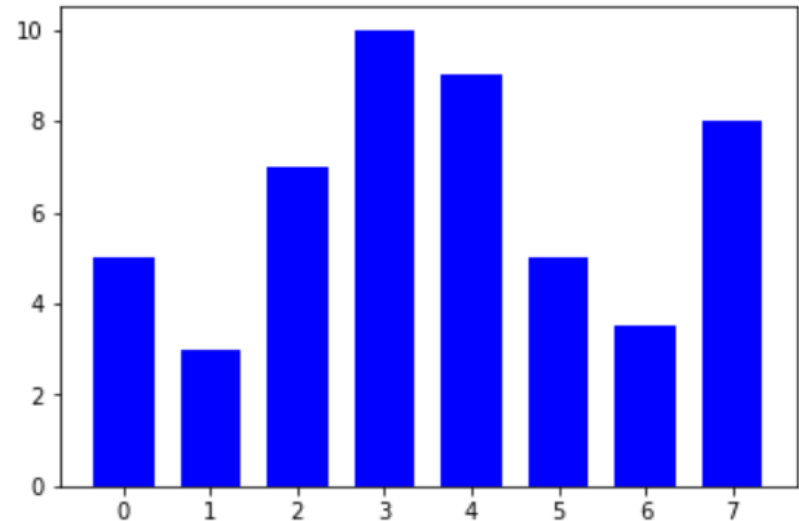


Matplotlib 기본 사용법

■ 다양한 차트 및 플롯

- Bar 차트를 그리기 위해서는 plt.bar() 함수
- Pie 차트를 그리기 위해서는 plt.pie() 함수
- 히스토그램을 그리기 위해선 plt.hist() 함수

```
y = [5, 3, 7, 10, 9, 5, 3.5, 8]
x = range(len(y))
plt.bar(x, y, width=0.7, color="blue")
plt.show()
```



감사합니다
