



4. 판다스 및 데이터 시각화

2022.10.17

Chung-Ang University
AI/ML Innovation Research Center
Hyun-soon Lee



<판다스>

목차

01 판다스란?

02 데이터 추출

03 그룹별 집계

04 병합과 연결

[강의 PPT 이용 안내]

1. 본 강의 PPT에 사용된 [데이터 과학을 위한 파이썬 머신러닝]의 내용에 관한 저작권은 한빛아카데미 (주) 있습니다.
2. [데이터 과학을 위한 파이썬 머신러닝]과 관련된 자료를 무단으로 전제하거나 배포할 경우 저작권법 136조에 의거하여 처벌을 받을 수 있습니다.
3. 강의에 사용된 교재 이외에 사용된 이미지 데이터 등도 강사명의로의 논문 또는 특허 등록 또는 특허 출원 중인 자료들로 무단 사용을 금합니다.

01

판다스란?

01 판다스란?

1. 판다스의 개념

- 판다스(pandas) : 파이썬의 데이터 분석 라이브러리
 - 데이터 테이블(data table)을 다루는 도구
- 기본적으로 넘파이를 사용
 - 넘파이 : 파이썬에서 배열을 다루는 최적의 라이브러리
 - 판다스는 넘파이를 효율적으로 사용하기 위해 인덱싱, 연산, 전처리 등 다양한 함수 제공

01 판다스란?

1. 판다스의 개념

- 데이터프레임(DataFrame) : 데이터 테이블 전체 객체
- 시리즈(Series) : 각 열 데이터를 다루는 객체

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	weight_0
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.9	4.98	1
1	0.02731	0.0	7.07	0	0.469	6.421	48.9	4.9671	2	242.0	17.8	396.9	9.14	1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.9	5.33	1

시리즈(Series)

데이터프레임 중 하나의 열에
해당하는 데이터의 모음 객체

데이터프레임(DataFrame)

데이터 테이블 전체를
포함하는 객체

그림 4-1 데이터프레임 객체와 시리즈 객체

[TIP] 시리즈 : 리스트, 튜플, 넘파이 배열의 1차원 자료구조로, 데이터프레임의 한 행이나 열을 의미한다.

01 판다스란?

2. 시리즈 객체

- 시리즈 객체 : 피쳐 벡터(feature vector)와 같은 개념
 - 일반적으로 하나의 피쳐 데이터를 포함하는 형태
 - 생성된 데이터프레임(DataFrame) 안에 포함될 수 있음
 - list, dict, ndarray 등 다양한 데이터 타입이 시리즈 객체 형태로 변환되기도 함

인덱스(index)	a b c d e	1 2 3 4 5	데이터(data)
	dtype: int64		데이터 타입(data type)

그림 4-2 시리즈 객체 예시

01 판다스란?

2. 시리즈 객체

- 시리즈 객체를 생성하면 세 가지 요소(property) 생성
 - 데이터(data) : 기존 다른 객체처럼 값을 저장하는 요소
 - 인덱스(index) : 항상 0부터 시작하고, 숫자로만 할당하는 값
 - 시리즈 객체에서는 숫자, 문자열, 0 외의 값으로 시작하는 숫자, 순서가 일정하지 않은 숫자를 입력할 수도 있음
 - 시리즈 객체에서는 인덱스 값의 중복을 허용
 - 데이터 타입(data type) : 넘파이의 데이터 타입과 일치
 - 판다스는 넘파이의 래퍼(wrapper) 라이브러리
 - 넘파이의 모든 기능 지원하고 데이터 타입도 그대로 적용

01 판다스란?

2. 시리즈 객체

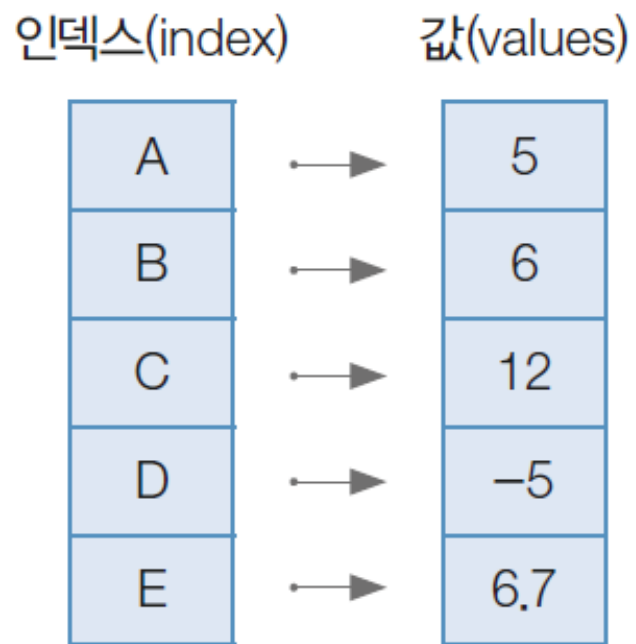


그림 4-3 시리즈 객체와 인덱스

- 시리즈(Series) 객체는 넘파이 배열(ndarray)의 하위 클래스
- 넘파이가 지원하는 어떠한 데이터 타입도 지원
- 인덱스와 반드시 정렬되어 있을 필요는 없음
- 인덱스 값은 중복을 허용

01 판다스란?

2. 시리즈 객체

```
In [1]: import pandas as pd      # pandas 모듈 호출
import numpy as np              # numpy 모듈 호출

from pandas import Series, DataFrame

list_data = [1,2,3,4,5]
list_name = ["a","b","c","d","e"]
example_obj = Series(data = list_data, index=list_name)
example_obj
```

```
Out [1]: a    1
         b    2
         c    3
         d    4
         e    5
         dtype: int64
```

01 판다스란?

2. 시리즈 객체

In [2]:	<code>example_obj.index</code>
Out [2]:	<code>Index(['a', 'b', 'c', 'd', 'e'], dtype='object')</code>

- index 값에 `In [1]`에서 입력한 `list_name` 객체의 값이 있음

In [3]:	<code>example_obj.values</code>
Out [3]:	<code>array([1, 2, 3, 4, 5], dtype=int64)</code>
In [4]:	<code>type(example_obj.values)</code>
Out [4]:	<code>numpy.ndarray</code>

- 데이터 값을 보기 위해서는 `values`를 사용
- 실제 생성된 `values`는 넘파이 배열(`numpy.ndarray`) 타입

01 판다스란?

2. 시리즈 객체

In [5]:	example_obj.dtype
Out [5]:	dtype('int64')

- dtype은 데이터의 타입을 나타냄
- 넘파이의 데이터 타입과 동일

01 판다스란?

2. 시리즈 객체

- 시리즈 객체는 객체의 이름을 변경할 수 있음
 - 열의 이름을 지정해주는 방식
 - 인덱스 이름도 추가로 지정 가능

In [6]:	<pre>example_obj.name = "number" example_obj.index.name = "id" example_obj</pre>
Out [6]:	<pre>id a 1 b 2 c 3 d 4 e 5 Name: number, dtype: int64</pre>

01 판다스란?

2. 시리즈 객체

- 시리즈 객체 생성하기
 - 데이터프레임 객체를 먼저 생성하고 각 열에서 시리즈 객체를 뽑는 것이 일반적인 방법
 - 다양한 시퀀스형 데이터 타입으로 저장 가능

In [7]:	<pre>dict_data = {"a":1, "b":2, "c":3, "d":4, "e":5} example_obj = Series(dict_data, dtype=np.float32, name="example_data") example_obj</pre>
Out [7]:	<pre>a 1.0 b 2.0 c 3.0 d 4.0 e 5.0 Name: example_data, dtype: float32</pre>

01 판다스란?

2. 시리즈 객체

- 판다스의 모든 객체는 인덱스 값을 기준으로 생성
 - 기존 데이터에
인덱스 값을 추가
하면 **NaN 값**이
출력 됨

In [8]:	<pre>dict_data_1 = {"a":1, "b":2, "c":3, "d":4, "e":5} indexes = ["a","b","c","d","e","f","g","h"] series_obj_1 = Series(dict_data_1, index=indexes) series_obj_1</pre>
Out [8]:	<pre>a 1.0 b 2.0 c 3.0 d 4.0 e 5.0 f NaN g NaN h NaN dtype: float64</pre>

*NaN(not a number): 'Null',
파이썬에서는 'None'을 의미함.

01 판다스란?

3. 데이터프레임 객체

- 데이터 테이블 전체를 지칭하는 객체
- 넘파이 배열의 특성을 그대로 가짐
- 인덱싱 : 열과 행 각각 사용하여 하나의 데이터에 접근

		열			
		foo	bar	baz	qux
인덱스	A	0	x	2.7	True
	B	4	y	6	True
	C	8	z	10	False
	D	-12	w	NA	False
	E	16	a	18	False

그림 4-4 데이터프레임 객체

- 넘파이 배열과 같음
- 각 열은 다른 타입을 가질 수 있음
- 행과 열 인덱스
- 변할 수 있는 크기(추가/삭제 열)

01 판다스란?

3. 데이터프레임 객체

3.1 데이터프레임의 생성

- 'read_확장자' 함수로 데이터 바로 로딩
 - .csv나 .xlsx 등 스프레드시트형 확장자 파일에서 데이터 로딩

```
In [9]: data_url = 'https://archive.ics.uci.edu/ml/machine-learning-  
databases/housing/housing.data'  
# 데이터 URL을 변수 data_url에 넣기  
df_data = pd.read_csv(data_url, sep='\s+', header = None) # csv 데이터 로드  
  
df = pd.DataFrame(df_data)  
df
```

01 판다스란?

3. 데이터프레임 객체

3.1 데이터프레임의 생성

Out [9]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2
...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	7.88	11.9

*데이터 개수:
506개
(0~505)

03 데이터를 모델에 대입하기 (2주차 강의자료, In [9]와 비교)

2. 판다스로 데이터 다루기

2.1 데이터 불러오기

```
In [1]: import pandas as pd                # (1) pandas 모듈 호출

data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data' # (2)
# 데이터 URL을 변수 data_url에 넣기

df_data = pd.read_csv(data_url, sep='\s+', header = None) # (3) csv 데이터 로드
df_data.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B',
                  'LSTAT', 'MEDV'] # (4) 데이터의 열 이름 지정
df_data.head()                # (5) 데이터 출력, 위에서부터 5개
```

Out [1]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.9	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	48.9	4.9671	2	242.0	17.8	396.9	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.9	5.33	36.2

01 판다스란?

3. 데이터프레임 객체

3.1 데이터프레임의 생성

- 데이터프레임을 직접 생성
 - 딕셔너리 타입 데이터에서 키(key)는 열 이름, 값(value)은 시퀀스형 데이터 타입을 넣어 각 열의 데이터로 만듦

```
In [10]: from pandas import Series, DataFrame

raw_data = {'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
            'last_name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze'],
            'age': [42, 52, 36, 24, 73],
            'city': ['San Francisco', 'Baltimore', 'Miami', 'Douglas', 'Boston']}

df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age', 'city'])

df
```

01 판다스란?

3. 데이터프레임 객체

3.1 데이터프레임의 생성

Out [10]:

	first_name	last_name	age	city
0	Jason	Miller	42	San Francisco
1	Molly	Jacobson	52	Baltimore
2	Tina	Ali	36	Miami
3	Jake	Milner	24	Douglas
4	Amy	Cooze	73	Boston

01 판다스란?

3. 데이터프레임 객체

3.2 데이터프레임의 열 다루기

- 데이터 생성시, 열 이름을 한정하면 해당 열만 추출

In [11]:	DataFrame(raw_data, columns = ["age", "city"])																		
Out [11]:	<table><tr><th></th><th>age</th><th>city</th></tr><tr><td>0</td><td>42</td><td>San Francisco</td></tr><tr><td>1</td><td>52</td><td>Baltimore</td></tr><tr><td>2</td><td>36</td><td>Miami</td></tr><tr><td>3</td><td>24</td><td>Douglas</td></tr><tr><td>4</td><td>73</td><td>Boston</td></tr></table>		age	city	0	42	San Francisco	1	52	Baltimore	2	36	Miami	3	24	Douglas	4	73	Boston
	age	city																	
0	42	San Francisco																	
1	52	Baltimore																	
2	36	Miami																	
3	24	Douglas																	
4	73	Boston																	

01 판다스란?

3. 데이터프레임 객체

3.2 데이터프레임의 열 다루기

- 데이터가 존재하지 않는 열을 추가하면 해당 열에는 NaN 값들 추가

```
In [12]: DataFrame(raw_data,  
                  columns = ["first_name","last_name","age", "city", "debt"]  
                  )
```

Out [12]:

	first_name	last_name	age	city	debt
0	Jason	Miller	42	San Francisco	NaN
1	Molly	Jacobson	52	Baltimore	NaN
2	Tina	Ali	36	Miami	NaN
3	Jake	Milner	24	Douglas	NaN
4	Amy	Cooze	73	Boston	NaN

02

데이터 추출

02 데이터 추출

1. 데이터 로딩

- excel-comp-data.xlsx 데이터로 실습 진행

account	name	street	city	state	postal-code	Jan	Feb	Mar
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000
320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000
648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000
109996	D'Amore, Gleichner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000
121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000
132971	Williamson, Schumm and Hettinger	89403 Casimer Spring	Jeremieburgh	Arkansas	62785	150000	120000	35000
145068	Casper LLC	340 Consuela Bridge Apt. 400	Lake Gabriellaton	Mississippi	18008	62000	120000	70000
205217	Kovacek-Johnston	91971 Cronin Vista Suite 601	Deronville	Rhodelsland	53461	145000	95000	35000
209744	Champlin-Morar	26739 Grant Lock	Lake Juliannton	Pennsylvania	64415	70000	95000	35000
212303	Gerhold-Maggio	366 Maggio Grove Apt. 998	North Ras	Idaho	46308	70000	120000	35000
214098	Goodwin, Homenick and Jerde	649 Cierra Forks Apt. 078	Rosaberg	Tennessee	47743	45000	120000	55000
231907	Hahn-Moore	18115 Olivine Throughway	Norbertomouth	NorthDakota	31415	150000	10000	162000
242368	Frami, Anderson and Donnelly	182 Bertie Road	East Davian	Iowa	72686	162000	120000	35000
268755	Walsh-Haley	2624 Beatty Parkways	Goodwinmouth	Rhodelsland	31919	55000	120000	35000

그림 4-5 excel-comp-data.xlsx

02 데이터 추출

1. 데이터 로딩

- xlsx 형태 데이터를 호출
 - openpyxl 모듈을 설치

```
In [1]: !conda install --y openpyxl
```

[TIP] 콘솔 명령어는 앞에 '!'를 붙여 실행시킨다.

- 설치 완료 또는 이미 설치된 경우에 아래와 같은 메시지가 나타나면 성공적으로 설치된 것임.

```
: !conda install --y openpyxl
```

```
Collecting package metadata (current_repodata.json): ...working... done
```

```
Solving environment: ...working... done
```

```
# All requested packages already installed.
```

```
Retrieving notices: ...working... done
```

02 데이터 추출

1. 데이터 로딩

- `xlsx` 형태 데이터를 호출
 - `read_excel` 함수로 엑셀 데이터 호출
 - 파일이 C 드라이브 `[source]-[ch04]` 폴더에 있다고 가정
(개인이 저장한 엑셀 파일에 대한 경로를 입력할 것.
가능한 폴더명들은 영어를 사용하고 짧은 경로를 추천함.)

```
In [2]: import pandas as pd # pandas 모듈 호출
import numpy as np # numpy 모듈 호출
df = pd.read_excel("c:/source/ch04/excel-comp-data.xlsx")
```

02 데이터 추출

2. 열 이름을 사용한 데이터 추출

- head와 tail 함수 : 처음 n개 행이나 마지막 n개 행 호출

In [3]: df.head(5)

Out [3]:

	account	name	street	city	state	postal-code	Jan	Feb	Mar
0	211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000
1	320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000
2	648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lillianland	Iowa	76517	91000	120000	35000
3	109996	D'Amore, Gleichner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000
4	121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000

02 데이터 추출

2. 열 이름을 사용한 데이터 추출

In [4]: `df.head(3).T`

Out [4]:

	0	1	2
account	211829	320563	648336
name	Kerluke, Koepp and Hilpert	Walter-Trantow	Bashirian, Kunde and Price
street	34456 Sean Highway	1311 Alvis Tunnel	62184 Schamberger Underpass Apt. 231
city	New Jaycob	Port Khadijah	New Lillianland
state	Texas	NorthCarolina	Iowa
postal-code	28752	38365	76517
Jan	10000	95000	91000
Feb	62000	45000	120000
Mar	35000	35000	35000

02 데이터 추출

2. 열 이름을 사용한 데이터 추출

- 열 이름을 리스트 형태로 넣어 호출
 - 가장 일반적인 호출 방법
 - 문자형 열 이름을 하나만 넣으면 값이 시리즈 객체로 반환됨
 - 열 이름을 여러 개 넣으면 데이터프레임 객체로 반환됨

In [5]:	df[["account", "street", "state"]].head(3)																		
Out [5]:	<table><thead><tr><th></th><th>account</th><th>street</th><th>state</th></tr></thead><tbody><tr><td>0</td><td>211829</td><td>34456 Sean Highway</td><td>Texas</td></tr><tr><td>1</td><td>320563</td><td>1311 Alvis Tunnel</td><td>NorthCarolina</td></tr><tr><td>2</td><td>648336</td><td>62184 Schamberger Underpass Apt. 231</td><td>Iowa</td></tr></tbody></table>				account	street	state	0	211829	34456 Sean Highway	Texas	1	320563	1311 Alvis Tunnel	NorthCarolina	2	648336	62184 Schamberger Underpass Apt. 231	Iowa
	account	street	state																
0	211829	34456 Sean Highway	Texas																
1	320563	1311 Alvis Tunnel	NorthCarolina																
2	648336	62184 Schamberger Underpass Apt. 231	Iowa																

02 데이터 추출

3. 행 번호를 사용한 데이터 추출

- 인덱스 번호로 호출
 - 기존의 리스트나 넘파이 배열(ndarray) 인덱싱과 동일

In [6]:	df[:3]																																								
Out [6]:	<table><tr><th></th><th>account</th><th>name</th><th>street</th><th>city</th><th>state</th><th>postal-code</th><th>Jan</th><th>Feb</th><th>Mar</th></tr><tr><td>0</td><td>211829</td><td>Kerluke, Koepp and Hilpert</td><td>34456 Sean Highway</td><td>New Jaycob</td><td>Texas</td><td>28752</td><td>10000</td><td>62000</td><td>35000</td></tr><tr><td>1</td><td>320563</td><td>Walter-Trantow</td><td>1311 Alvis Tunnel</td><td>Port Khadijah</td><td>NorthCarolina</td><td>38365</td><td>95000</td><td>45000</td><td>35000</td></tr><tr><td>2</td><td>648336</td><td>Bashirian, Kunde and Price</td><td>62184 Schamberger Underpass Apt. 231</td><td>New Lilianland</td><td>Iowa</td><td>76517</td><td>91000</td><td>120000</td><td>35000</td></tr></table>		account	name	street	city	state	postal-code	Jan	Feb	Mar	0	211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000	1	320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000	2	648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000
	account	name	street	city	state	postal-code	Jan	Feb	Mar																																
0	211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000																																
1	320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000																																
2	648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000																																

02 데이터 추출

4. 행과 열을 모두 사용한 데이터 추출

- 위의 두 가지 방법(열 이름과 행 번호) 함께 사용
- 데이터의 일정 부분을 사각형 형태로 잘라냄

In [7]:	df[["name","street"]][:2]										
Out [7]:	<table><thead><tr><th></th><th>name</th><th>street</th></tr></thead><tbody><tr><td>0</td><td>Kerluke, Koepp and Hilpert</td><td>34456 Sean Highway</td></tr><tr><td>1</td><td>Walter-Trantow</td><td>1311 Alvis Tunnel</td></tr></tbody></table>			name	street	0	Kerluke, Koepp and Hilpert	34456 Sean Highway	1	Walter-Trantow	1311 Alvis Tunnel
	name	street									
0	Kerluke, Koepp and Hilpert	34456 Sean Highway									
1	Walter-Trantow	1311 Alvis Tunnel									

02 데이터 추출

4. 행과 열을 모두 사용한 데이터 추출

- loc 함수 : 인덱스 이름과 열 이름으로 데이터 추출
 - 인덱스를 0부터 시작하는 숫자 아닌 다른 값으로 변경 가능

```
In [8]: df.index = df["account"]  
del df["account"]  
df.head()
```

Out [8]:

	name	street	city	state	postal-code	Jan	Feb	Mar
account								
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000
320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000
648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000
109996	D'Amore, Gleichner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000
121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000

02 데이터 추출

4. 행과 열을 모두 사용한 데이터 추출

- 인덱스 대신 특정 account 번호를 넣어 해당 번호의 값을 나타냄

In [9]:	df.loc[[211829,320563],["name","street"]]														
Out [9]:	<table><tr><th></th><th>name</th><th>street</th></tr><tr><td>account</td><td></td><td></td></tr><tr><td>211829</td><td>Kerluke, Koepp and Hilpert</td><td>34456 Sean Highway</td></tr><tr><td>320563</td><td>Walter-Trantow</td><td>1311 Alvis Tunnel</td></tr></table>				name	street	account			211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	320563	Walter-Trantow	1311 Alvis Tunnel
	name	street													
account															
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway													
320563	Walter-Trantow	1311 Alvis Tunnel													

02 데이터 추출

4. 행과 열을 모두 사용한 데이터 추출

- 인덱스 번호가 항상 정렬되어 있지 않아 처음 저장된 순서대로 출력

In [10]:	df.loc[205217:,"name","street"]																																
Out [10]:	<table><tr><th></th><th>name</th><th>street</th></tr><tr><td>account</td><td></td><td></td></tr><tr><td>205217</td><td>Kovacek-Johnston</td><td>91971 Cronin Vista Suite 601</td></tr><tr><td>209744</td><td>Champlin-Morar</td><td>26739 Grant Lock</td></tr><tr><td>212303</td><td>Gerhold-Maggio</td><td>366 Maggio Grove Apt. 998</td></tr><tr><td>214098</td><td>Goodwin, Homenick and Jerde</td><td>649 Cierra Forks Apt. 078</td></tr><tr><td>231907</td><td>Hahn-Moore</td><td>18115 Olivine Throughway</td></tr><tr><td>242368</td><td>Frami, Anderson and Donnelly</td><td>182 Bertie Road</td></tr><tr><td>268755</td><td>Walsh-Haley</td><td>2624 Beatty Parkways</td></tr><tr><td>273274</td><td>McDermott PLC</td><td>8917 Bergstrom Meadow</td></tr></table>				name	street	account			205217	Kovacek-Johnston	91971 Cronin Vista Suite 601	209744	Champlin-Morar	26739 Grant Lock	212303	Gerhold-Maggio	366 Maggio Grove Apt. 998	214098	Goodwin, Homenick and Jerde	649 Cierra Forks Apt. 078	231907	Hahn-Moore	18115 Olivine Throughway	242368	Frami, Anderson and Donnelly	182 Bertie Road	268755	Walsh-Haley	2624 Beatty Parkways	273274	McDermott PLC	8917 Bergstrom Meadow
	name	street																															
account																																	
205217	Kovacek-Johnston	91971 Cronin Vista Suite 601																															
209744	Champlin-Morar	26739 Grant Lock																															
212303	Gerhold-Maggio	366 Maggio Grove Apt. 998																															
214098	Goodwin, Homenick and Jerde	649 Cierra Forks Apt. 078																															
231907	Hahn-Moore	18115 Olivine Throughway																															
242368	Frami, Anderson and Donnelly	182 Bertie Road																															
268755	Walsh-Haley	2624 Beatty Parkways																															
273274	McDermott PLC	8917 Bergstrom Meadow																															

02 데이터 추출

4. 행과 열을 모두 사용한 데이터 추출

- `iloc` 함수 : 인덱스 번호로만 데이터 호출
 - ‘index location’의 약자

In [11]:

df.iloc[:10, :3]

Out [11]:

account	name	street	city
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob
320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah
648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lillianland
109996	D'Amore, Gleichner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh
121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester
132971	Williamson, Schumm and Hettinger	89403 Casimer Spring	Jeremieburgh
145068	Casper LLC	340 Consuela Bridge Apt. 400	Lake Gabriellaton
205217	Kovacek-Johnston	91971 Cronin Vista Suite 601	Deronville
209744	Champlin-Morar	26739 Grant Lock	Lake Juliannton
212303	Gerhold-Maggio	366 Maggio Grove Apt. 998	North Ras

02 데이터 추출

5. loc, iloc 함수를 사용한 데이터 추출

- reset_index 함수로 새로운 인덱스 할당된 객체 생성
 - 인덱스 이름이나 인덱스 중 편한 방법을 사용

In [12]: `df_new = df.reset_index()`
`df_new`

Out [12]:

	street	city	state	postal-code	Jan	Feb	Mar
0	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000
2	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000
3	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000
4	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000
5	89403 Casimer Spring	Jeremieburgh	Arkansas	62785	150000	120000	35000
6	340 Consuela Bridge Apt. 400	Lake Gabriellaton	Mississippi	18008	62000	120000	70000
7	91971 Cronin Vista Suite 601	Deronville	Rhodelsland	53461	145000	95000	35000
8	26739 Grant Lock	Lake Juliannton	Pennsylvania	64415	70000	95000	35000
9	366 Maggio Grove Apt. 998	North Ras	Idaho	46308	70000	120000	35000
10	649 Cierra Forks Apt. 078	Rosaberg	Tennessee	47743	45000	120000	55000
11	18115 Olivine Throughway	Norbertomouth	NorthDakota	31415	150000	10000	162000
12	182 Bertie Road	East Davian	Iowa	72686	162000	120000	35000
13	2624 Beatty Parkways	Goodwinmouth	Rhodelsland	31919	55000	120000	35000
14	8917 Bergstrom Meadow	Kathryneborough	Delaware	27933	150000	120000	70000

02 데이터 추출

6. drop 함수

- drop 함수 : 특정 열이나 행을 삭제한 객체를 반환, 해당 값이 실제로 사라지는 것이 아님, 원본 객체는 영향을 받지 않음

In [13]: `df_new.drop(1).head()`

Out [13]:

	account	name	street	city	state	postal-code	Jan	Feb	Mar
0	211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000
2	648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000
3	109996	D'Amore, Gleichner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000
4	121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000
5	132971	Williamson, Schumm and Hettinger	89403 Casimer Spring	Jeremieburgh	Arkansas	62785	150000	120000	35000

02 데이터 추출

6. drop 함수

```
In [14]: df_drop = df_new.drop(1)
```

```
In [15]: df_new.drop(1, inplace=True)
```

```
In [16]: df_new.drop("account", axis=1) # account 열 제거  
df_new.drop(["account", "name"], axis=1) # account, name 열 제거
```

Out [16]:

	street	city	state	postal-code	Jan	Feb	Mar
0	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000
2	62184 Schamberger Underpass Apt. 231	New Lillianland	Iowa	76517	91000	120000	35000
3	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000
4	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000
5	89403 Casimer Spring	Jeremieburgh	Arkansas	62785	150000	120000	35000
6	340 Consuela Bridge Apt. 400	Lake Gabriellaton	Mississippi	18008	62000	120000	70000
7	91971 Cronin Vista Suite 601	Deronville	Rhodelsland	53461	145000	95000	35000
8	26739 Grant Lock	Lake Juliannton	Pennsylvania	64415	70000	95000	35000
9	366 Maggio Grove Apt. 998	North Ras	Idaho	46308	70000	120000	35000
10	649 Cierra Forks Apt. 078	Rosaberg	Tennessee	47743	45000	120000	55000
11	18115 Olivine Throughway	Norbertomouth	NorthDakota	31415	150000	10000	162000
12	182 Bertie Road	East Davian	Iowa	72686	162000	120000	35000
13	2624 Beatty Parkways	Goodwinmouth	Rhodelsland	31919	55000	120000	35000
14	8917 Bergstrom Meadow	Kathryneborough	Delaware	27933	150000	120000	70000

03

그룹별 집계

03 그룹별 집계

1. 그룹별 집계의 개념

- 그룹별 집계(**groupby**) : 데이터로부터 동일한 객체를 가진 데이터만 따로 뽑아 기술통계 데이터를 추출
 - 엑셀의 피벗테이블(pivot table) 기능과 비슷
 - 예) A반 수학 점수의 원본 데이터(raw data)를 가지고 있을 때 해당 데이터에서
 - 같은 성별을 가진 학생들의 평균 점수를 구하거나
 - 50점 이상을 받은 학생의 수를 구함

03 그룹별 집계

1. 그룹별 집계의 개념

- groupby 명령어는 분할→적용→결합 과정을 거침
 - 분할(split) : 같은 종류의 데이터끼리 나누는 기능
 - 적용(apply) : 데이터 블록마다 sum, count, mean 등 연산 적용
 - 결합(combine) : 연산 함수가 적용된 각 블록들을 합침

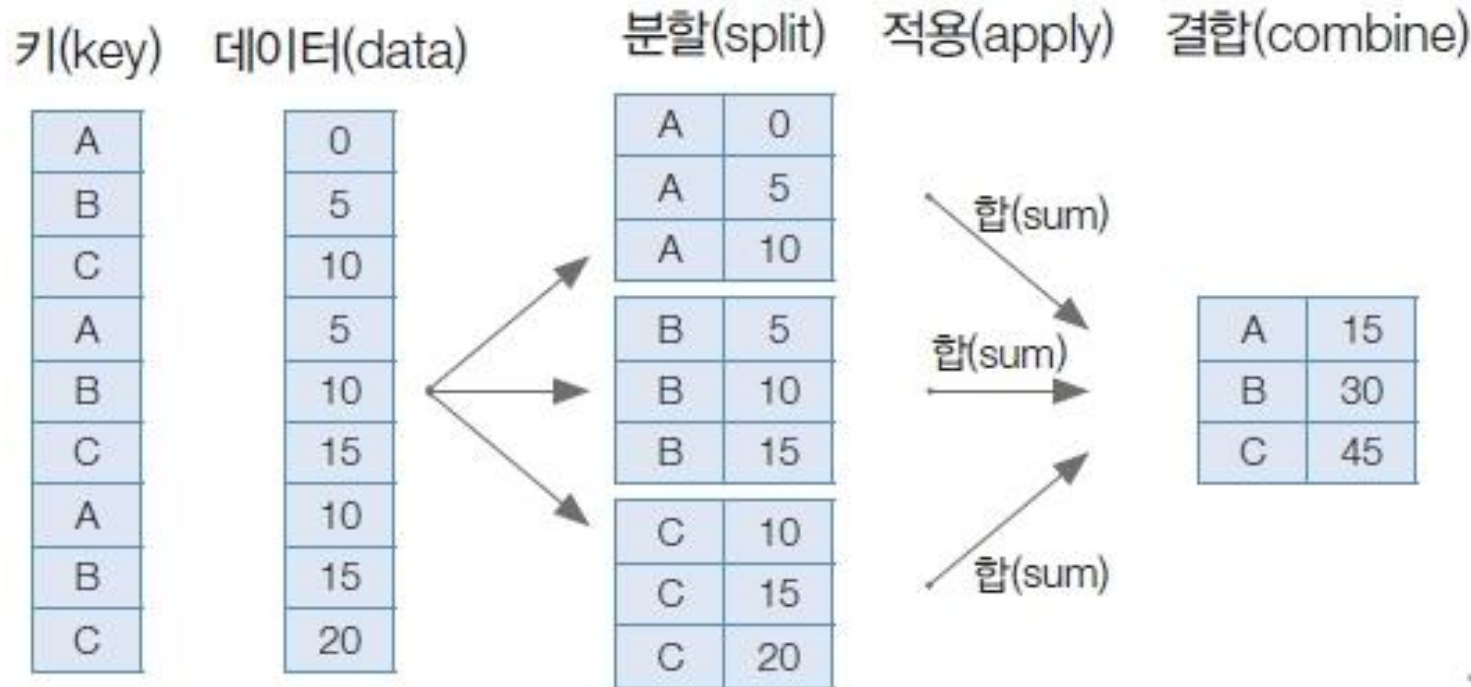


그림 4-6 그룹별 집계의 과정

03 그룹별 집계

2. 그룹별 집계 사용하기

2.1 그룹별 집계의 기본형

- 해당 데이터는 레이싱 팀이 매년 경기에 참가하여 얻는 순위와 획득 점수를 정리한 데이터 테이블

```
In [1]: import pandas as pd # pandas 모듈 호출
import numpy as np # numpy 모듈 호출

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings', 'kings', 'Kings', 'Kings',
'Riders', 'Royals', 'Royals', 'Riders'],
'Rank': [1, 2, 2, 3, 3, 4, 1, 1, 2, 4, 1, 2],
'Year': [2014, 2015, 2014, 2015, 2014, 2015, 2016, 2017, 2016, 2014, 2015, 2017],
'Points': [876, 789, 863, 673, 741, 812, 756, 788, 694, 701, 804, 690]}

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

03 그룹별 집계

2. 그룹별 집계 사용하기

2.1 그룹별 집계의 기본형

Out [1]:

	Team	Rank	Year	Points
0	Riders	1	2014	876
1	Riders	2	2015	789
2	Devils	2	2014	863
3	Devils	3	2015	673
4	Kings	3	2014	741
5	kings	4	2015	812
6	Kings	1	2016	756
7	Kings	1	2017	788
8	Riders	2	2016	694
9	Royals	4	2014	701
10	Royals	1	2015	804
11	Riders	2	2017	690

03 그룹별 집계

2. 그룹별 집계 사용하기

2.1 그룹별 집계의 기본형

In [2]:	<code>df.groupby("Team")["Points"].sum()</code>
Out [2]:	<pre>Team Devils 1536 Kings 2285 Riders 3049 Royals 1505 kings 812 Name: Points, dtype: int64</pre>

`df.groupby("Team")["Points"].sum()`

묶음의 기준이 되는 열

적용받는 열

적용받는 연산

그림 4-7 코드 설명

03 그룹별 집계

2. 그룹별 집계 사용하기

2.2 멀티 인덱스 그룹별 집계

- 한 개 이상의 열을 기준으로 그룹별 집계를 실행
 - 리스트를 사용하여 여러 개의 열 이름을 기준으로 넣으면 여러 열이 키 값이 되어 결과 출력
 - 계층적 인덱스(hierarchical index) 형태

03 그룹별 집계

2. 그룹별 집계 사용하기

2.2 멀티 인덱스 그룹별 집계

In [3]:	multi_groupby = df.groupby(["Team", "Year"])["Points"].sum() multi_groupby		
Out [3]:	Team	Year	
	Devils	2014	863
		2015	679
	Kings	2014	741
		2016	756
		2017	788
	Riders	2014	876
		2015	789
		2016	694
		2017	690
	Royals	2014	701
		2015	804
	kings	2015	812
	Name: Points, dtype: int64		

03 그룹별 집계

2. 그룹별 집계 사용하기

2.3 멀티 인덱스

- 한 개 이상의 열로 그룹별 집계 수행하면 여러 열이 모두 인덱스로 반환됨

In [4]:	<pre>multi_groupby = df.groupby(["Team", "Year"])["Points"].sum() multi_groupby.index</pre>
Out [4]:	<pre>MultiIndex([('Devils', 2014), ('Devils', 2015), ('Kings', 2014), ('Kings', 2016), ('Kings', 2017), ('Riders', 2014), ('Riders', 2015), ('Riders', 2016), ('Riders', 2017), ('Royals', 2014), ('Royals', 2015), ('kings', 2015)], names=['Team', 'Year'])</pre>

03 그룹별 집계

2. 그룹별 집계 사용하기

2.3 멀티 인덱스

In [5]:	multi_groupby["Devils":"Kings"]		
Out [5]:	Team	Year	
	Devils	2014	863
		2015	673
	Kings	2014	741
		2016	756
		2017	788
	Name: Points, dtype: int64		

03 그룹별 집계

2. 그룹별 집계 사용하기

2.3 멀티 인덱스

- `Unstack` 함수를 사용하여 기존 인덱스를 기준으로 묶인 데이터에서 **두 번째 인덱스를 열로 변화**시켜 엑셀의 피벗테이블과 유사한 형태로 데이터를 보여줌

In [6]:	multi_groupby.unstack()																																							
Out [6]:	<table><tr><th>Year</th><th>2014</th><th>2015</th><th>2016</th><th>2017</th></tr><tr><th>Team</th><td></td><td></td><td></td><td></td></tr><tr><td>Devils</td><td>863.0</td><td>673.0</td><td>NaN</td><td>NaN</td></tr><tr><td>Kings</td><td>741.0</td><td>NaN</td><td>756.0</td><td>788.0</td></tr><tr><td>Riders</td><td>876.0</td><td>789.0</td><td>694.0</td><td>690.0</td></tr><tr><td>Royals</td><td>701.0</td><td>804.0</td><td>NaN</td><td>NaN</td></tr><tr><td>kings</td><td>NaN</td><td>812.0</td><td>NaN</td><td>NaN</td></tr></table>					Year	2014	2015	2016	2017	Team					Devils	863.0	673.0	NaN	NaN	Kings	741.0	NaN	756.0	788.0	Riders	876.0	789.0	694.0	690.0	Royals	701.0	804.0	NaN	NaN	kings	NaN	812.0	NaN	NaN
Year	2014	2015	2016	2017																																				
Team																																								
Devils	863.0	673.0	NaN	NaN																																				
Kings	741.0	NaN	756.0	788.0																																				
Riders	876.0	789.0	694.0	690.0																																				
Royals	701.0	804.0	NaN	NaN																																				
kings	NaN	812.0	NaN	NaN																																				

03 그룹별 집계

2. 그룹별 집계 사용하기

2.3 멀티 인덱스

- `swaplevel` 함수로 인덱스 간 레벨을 변경
- `sort_index` 함수로 첫 번째 인덱스를 기준으로 데이터 재정렬

In [7]:	<code>multi_groupby.swaplevel().sort_index()</code>
Out [7]:	<pre>Year Team 2014 Devils 863 Kings 741 Riders 876 Royals 701 2015 Devils 673 Riders 789 Royals 804 kings 812 2016 Kings 756 Riders 694 2017 Kings 788 Riders 690 Name: Points, dtype: int64</pre>

03 그룹별 집계

2. 그룹별 집계 사용하기

2.3 멀티 인덱스

- 각 레벨에 별도의 연산함수를 적용할 수 있음

In [8]:	<code>multi_groupby.sum(level=0)</code>
Out [8]:	Team Devils 1536 Kings 2285 Riders 3049 Royals 1505 kings 812 Name: Points, dtype: int64
In [9]:	<code>multi_groupby.sum(level=1)</code>
Out [9]:	Year 2014 3181 2015 3078 2016 1450 2017 1478 Name: Points, dtype: int64

03 그룹별 집계

3. 그룹화된 상태

- 그룹화된(grouped) 상태 : 분할→적용→결합 중에서 분할까지만 이루어진 상태
- `get_group` 함수 : 해당 키 값을 기준으로 분할된 데이터프레임 객체를 확인

```
In [10]: grouped = df.groupby("Team")  
grouped.get_group("Riders")
```

Out [10]:

	Team	Rank	Year	Points
0	Riders	1	2014	876
1	Riders	2	2015	789
8	Riders	2	2016	694
11	Riders	2	2017	690

03 그룹별 집계

3. 그룹화된 상태

3.1 집계

- 집계(aggregation) : 요약된 통계 정보를 추출
- `agg` 함수 : `min`, `넘파이` `mean` 등 기존 함수 그대로 적용

In [11]:	<code>grouped.agg(min)</code>	In [12]:	<code>grouped.agg(np.mean)</code>																																																								
Out [11]:	<table><thead><tr><th></th><th>Rank</th><th>Year</th><th>Points</th></tr></thead><tbody><tr><td>Team</td><td></td><td></td><td></td></tr><tr><td>Devils</td><td>2</td><td>2014</td><td>673</td></tr><tr><td>Kings</td><td>1</td><td>2014</td><td>741</td></tr><tr><td>Riders</td><td>1</td><td>2014</td><td>690</td></tr><tr><td>Royals</td><td>1</td><td>2014</td><td>701</td></tr><tr><td>kings</td><td>4</td><td>2015</td><td>812</td></tr></tbody></table>		Rank	Year	Points	Team				Devils	2	2014	673	Kings	1	2014	741	Riders	1	2014	690	Royals	1	2014	701	kings	4	2015	812	Out [12]:	<table><thead><tr><th></th><th>Rank</th><th>Year</th><th>Points</th></tr></thead><tbody><tr><td>Team</td><td></td><td></td><td></td></tr><tr><td>Devils</td><td>2.500000</td><td>2014.500000</td><td>768.000000</td></tr><tr><td>Kings</td><td>1.666667</td><td>2015.666667</td><td>761.666667</td></tr><tr><td>Riders</td><td>1.750000</td><td>2015.500000</td><td>762.250000</td></tr><tr><td>Royals</td><td>2.500000</td><td>2014.500000</td><td>752.500000</td></tr><tr><td>kings</td><td>4.000000</td><td>2015.000000</td><td>812.000000</td></tr></tbody></table>		Rank	Year	Points	Team				Devils	2.500000	2014.500000	768.000000	Kings	1.666667	2015.666667	761.666667	Riders	1.750000	2015.500000	762.250000	Royals	2.500000	2014.500000	752.500000	kings	4.000000	2015.000000	812.000000
	Rank	Year	Points																																																								
Team																																																											
Devils	2	2014	673																																																								
Kings	1	2014	741																																																								
Riders	1	2014	690																																																								
Royals	1	2014	701																																																								
kings	4	2015	812																																																								
	Rank	Year	Points																																																								
Team																																																											
Devils	2.500000	2014.500000	768.000000																																																								
Kings	1.666667	2015.666667	761.666667																																																								
Riders	1.750000	2015.500000	762.250000																																																								
Royals	2.500000	2014.500000	752.500000																																																								
kings	4.000000	2015.000000	812.000000																																																								

03 그룹별 집계

3. 그룹화된 상태

3.2 변환

- 변환(transformation) : 해당 정보를 변환
- 키 값별로 요약된 정보가 아닌 개별 데이터 변환 지원
- 적용 시점에서는 그룹화된 상태의 값으로 적용

03 그룹별 집계

3. 그룹화된 상태

3.2 변환

In [13]: `grouped.transform(max)`

Out [13]:

	Rank	Year	Points
0	2	2017	876
1	2	2017	876
2	3	2015	863
3	3	2015	863
4	3	2017	788
5	4	2015	812
6	3	2017	788
7	3	2017	788
8	2	2017	876
9	4	2015	804
10	4	2015	804
11	2	2017	876

03 그룹별 집계

3. 그룹화된 상태

3.2 변환

In [14]: `score = lambda x: (x - x.mean()) / x.std()
grouped.transform(score)`

Out [14]:

	Rank	Year	Points
0	-1.500000	-1.161895	1.284327
1	0.500000	-0.387298	0.302029
2	-0.707107	-0.707107	0.707107
3	0.707107	0.707107	-0.707107
4	1.154701	-1.091089	-0.860862
5	NaN	NaN	NaN
6	-0.577350	0.218218	-0.236043
7	-0.577350	0.872872	1.096905
8	0.500000	0.387298	-0.770596
9	0.707107	-0.707107	-0.707107
10	-0.707107	0.707107	0.707107
11	0.500000	1.161895	-0.815759

03 그룹별 집계

3. 그룹화된 상태

3.3 필터

- 필터(filter) : 특정 조건으로 데이터를 검색
 - 주로 filter 함수 사용
 - x는 분할된 상태에서 각각의 그룹화된 데이터프레임

In [15]:	df.groupby('Team').filter(lambda x: len(x) >= 3)																																								
Out [15]:	<table><tr><th></th><th>Team</th><th>Rank</th><th>Year</th><th>Points</th></tr><tr><td>0</td><td>Riders</td><td>1</td><td>2014</td><td>876</td></tr><tr><td>1</td><td>Riders</td><td>2</td><td>2015</td><td>789</td></tr><tr><td>4</td><td>Kings</td><td>3</td><td>2014</td><td>741</td></tr><tr><td>6</td><td>Kings</td><td>1</td><td>2016</td><td>756</td></tr><tr><td>7</td><td>Kings</td><td>1</td><td>2017</td><td>788</td></tr><tr><td>8</td><td>Riders</td><td>2</td><td>2016</td><td>694</td></tr><tr><td>11</td><td>Riders</td><td>2</td><td>2017</td><td>690</td></tr></table>		Team	Rank	Year	Points	0	Riders	1	2014	876	1	Riders	2	2015	789	4	Kings	3	2014	741	6	Kings	1	2016	756	7	Kings	1	2017	788	8	Riders	2	2016	694	11	Riders	2	2017	690
	Team	Rank	Year	Points																																					
0	Riders	1	2014	876																																					
1	Riders	2	2015	789																																					
4	Kings	3	2014	741																																					
6	Kings	1	2016	756																																					
7	Kings	1	2017	788																																					
8	Riders	2	2016	694																																					
11	Riders	2	2017	690																																					

03 그룹별 집계

3. 그룹화된 상태

3.3 필터

- lambda 함수는 분할된 데이터프레임 전체를 매개변수로 받음
- Points 열을 추출

In [16]:	<code>df.groupby('Team').filter(lambda x: x["Points"].max() > 800)</code>																																																					
Out [16]:	<table><thead><tr><th></th><th>Team</th><th>Rank</th><th>Year</th><th>Points</th></tr></thead><tbody><tr><td>0</td><td>Riders</td><td>1</td><td>2014</td><td>876</td></tr><tr><td>1</td><td>Riders</td><td>2</td><td>2015</td><td>789</td></tr><tr><td>2</td><td>Devils</td><td>2</td><td>2014</td><td>863</td></tr><tr><td>3</td><td>Devils</td><td>3</td><td>2015</td><td>673</td></tr><tr><td>5</td><td>kings</td><td>4</td><td>2015</td><td>812</td></tr><tr><td>8</td><td>Riders</td><td>2</td><td>2016</td><td>694</td></tr><tr><td>9</td><td>Royals</td><td>4</td><td>2014</td><td>701</td></tr><tr><td>10</td><td>Royals</td><td>1</td><td>2015</td><td>804</td></tr><tr><td>11</td><td>Riders</td><td>2</td><td>2017</td><td>690</td></tr></tbody></table>					Team	Rank	Year	Points	0	Riders	1	2014	876	1	Riders	2	2015	789	2	Devils	2	2014	863	3	Devils	3	2015	673	5	kings	4	2015	812	8	Riders	2	2016	694	9	Royals	4	2014	701	10	Royals	1	2015	804	11	Riders	2	2017	690
	Team	Rank	Year	Points																																																		
0	Riders	1	2014	876																																																		
1	Riders	2	2015	789																																																		
2	Devils	2	2014	863																																																		
3	Devils	3	2015	673																																																		
5	kings	4	2015	812																																																		
8	Riders	2	2016	694																																																		
9	Royals	4	2014	701																																																		
10	Royals	1	2015	804																																																		
11	Riders	2	2017	690																																																		

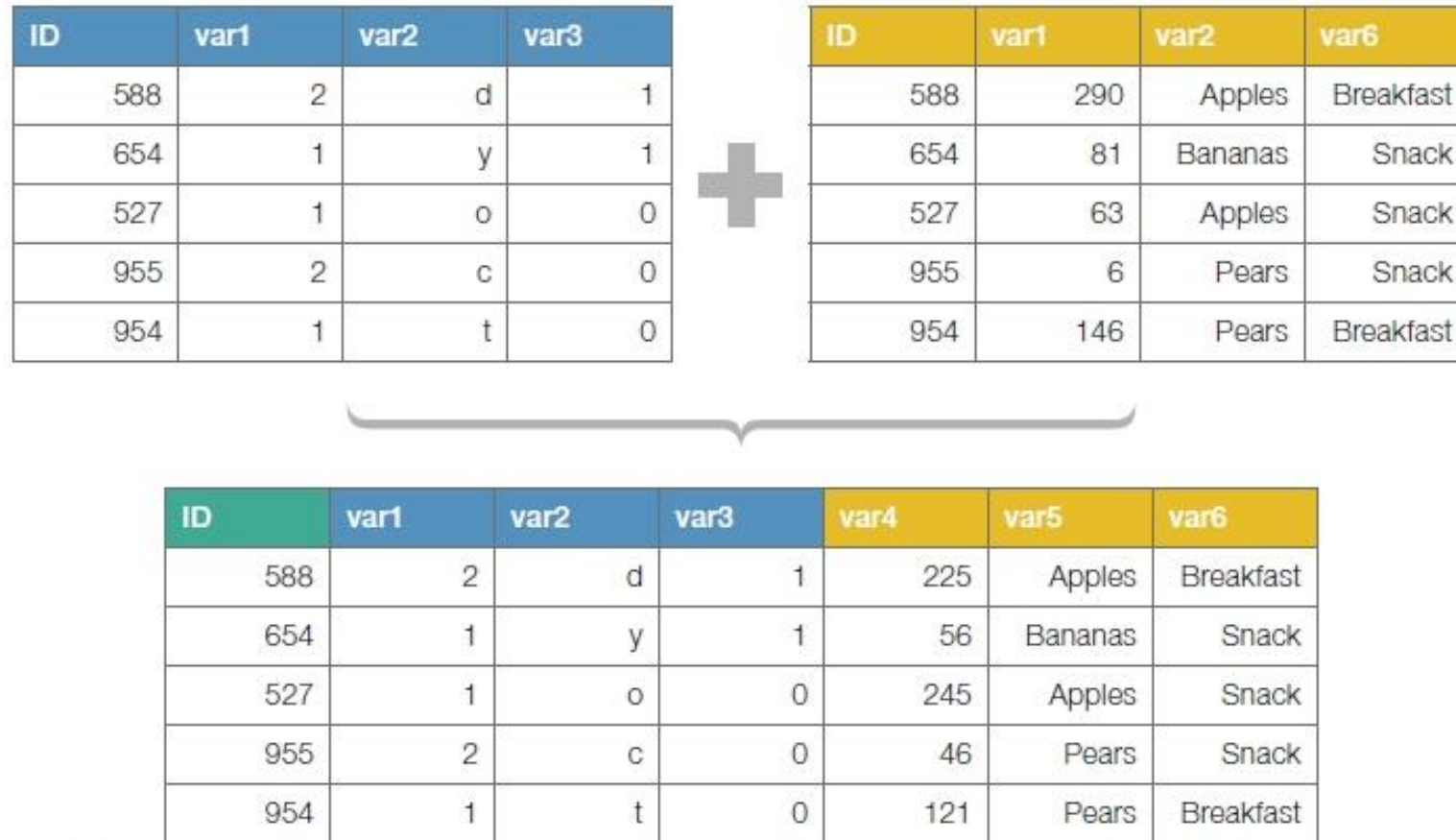
04

병합과 연결

04 병합과 연결

1. 병합

- 병합(merge) : 두 개의 데이터를 특정 기준한 기준을 가지고 하나로 통합하는 작업



ID	var1	var2	var3
588	2	d	1
654	1	y	1
527	1	o	0
955	2	c	0
954	1	t	0

ID	var1	var2	var6
588	290	Apples	Breakfast
654	81	Bananas	Snack
527	63	Apples	Snack
955	6	Pears	Snack
954	146	Pears	Breakfast

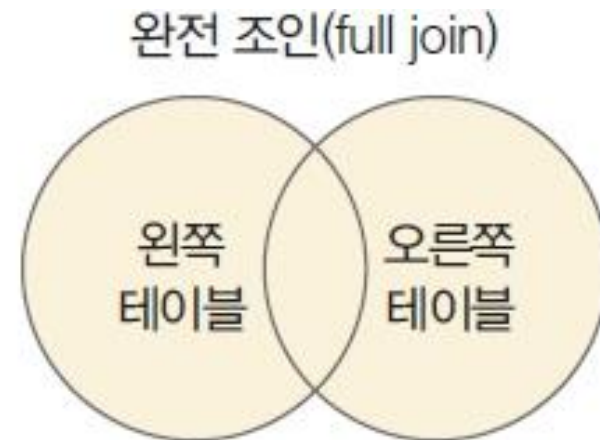
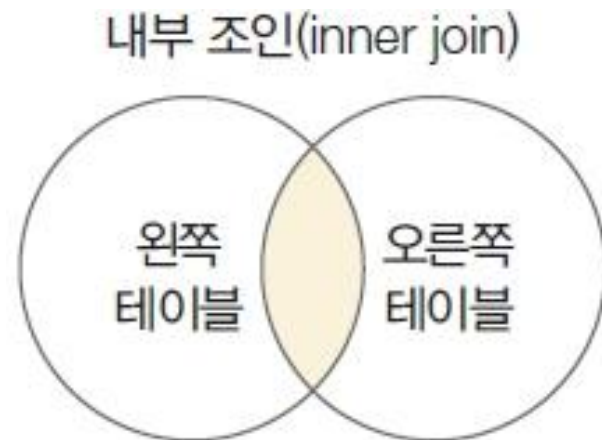
ID	var1	var2	var3	var4	var5	var6
588	2	d	1	225	Apples	Breakfast
654	1	y	1	56	Bananas	Snack
527	1	o	0	245	Apples	Snack
955	2	c	0	46	Pears	Snack
954	1	t	0	121	Pears	Breakfast

그림 4-8 데이터 테이블 간 병합의 예시

04 병합과 연결

1. 병합

- SQL에서는 조인(join)이라는 표현을 더 많이 사용
 - 내부 조인(inner join) : 키 값을 기준으로 두 테이블에 모두 존재하는 키 값의 행끼리 병합
 - 완전 조인(full join) : 두 개의 테이블에서 각각의 행을 병합
두 테이블에서 동일한 키 값을 가진 행은 통합하고, 두 테이블 중 하나라도 키 값이 존재하지 않는다면 존재하는 쪽의 데이터만 남겨둠



04 병합과 연결

1. 병합

- 왼쪽 조인(left join) : 왼쪽 테이블의 값을 기준으로 같은 키 값을 소유하고 있는 행을 병합하고, 오른쪽 테이블에 해당 키 값이 존재하지 않는다면 해당 행은 삭제
- 오른쪽 조인(right join) : 오른쪽 테이블의 값을 기준으로 같은 키 값을 소유하고 있는 행을 병합하고, 왼쪽 테이블에 해당 키 값이 존재하지 않는다면 해당 행은 삭제

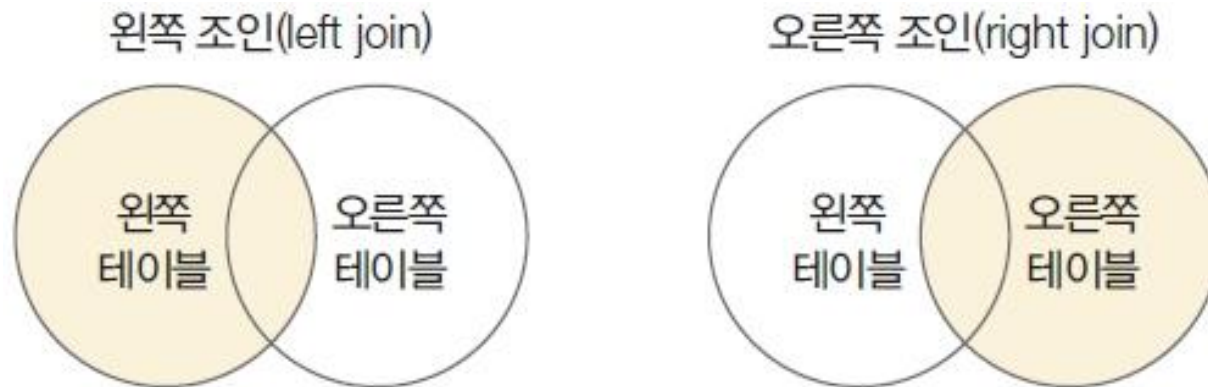


그림 4-9 병합, 조인(join)의 종류

04 병합과 연결

1. 병합

1.1 내부 조인

- 내부 조인(inner join) : 가장 기본적인 조인
- 집합으로 보면 양쪽의 교집합 데이터를 통합

04 병합과 연결

1. 병합

1.1 내부 조인

```
In [1]: import pandas as pd # pandas 모듈 호출
raw_data = {
    'subject_id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
    'test_score': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]}

df_left = pd.DataFrame(raw_data, columns = ['subject_id', 'test_score'])
df_left
```

04 병합과 연결

1. 병합

1.1 내부 조인

Out [1]:

	subject_id	test_score
0	1	51
1	2	15
2	3	15
3	4	61
4	5	16
5	7	14
6	8	15
7	9	1
8	10	61
9	11	16

04 병합과 연결

1. 병합

1.1 내부 조인

```
In [2]: raw_data = {  
        'subject_id': ['4', '5', '6', '7', '8'],  
        'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],  
        'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}  
  
        df_right = pd.DataFrame(raw_data, columns = ['subject_id', 'first_name',  
        'last_name'])  
        df_right
```

04 병합과 연결

1. 병합

1.1 내부 조인

Out [2]:

	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

04 병합과 연결

1. 병합

1.1 내부 조인

- subject_id를 기준으로 내부 조인을 수행
 - 키 값 subject_id 열의 값이 두 테이블 모두 존재해야 병합됨
 - left, right 매개변수에 각 위치에 해당하는 데이터프레임 객체를 입력
 - **how**에 **조인 방법** "inner"를 문자열 타입으로 입력
 - **on**에 **병합의 기준이 되는 열 이름**을 입력

In [3]: `pd.merge(left=df_left, right=df_right, how="inner", on='subject_id')`

Out [3]:

	subject_id	test_score	first_name	last_name
0	4	61	Billy	Bonder
1	5	16	Brian	Black
2	7	14	Bryce	Brice
3	8	15	Betty	Btisan

04 병합과 연결

1. 병합

1.1 내부 조인

[하나 더 알기] left_on과 right_on 매개변수

- 왼쪽 테이블과 오른쪽 테이블의 키 값이 다른 경우
left_on과 right_on 매개변수를 사용하여 각 테이블 키 값을 입력

```
pd.merge(left=df_left, right=df_right,  
         left_on='subject_id', right_on='subject_id')
```

04 병합과 연결

1. 병합

1.2 왼쪽 조인, 오른쪽 조인

- 왼쪽 조인(left join) : 왼쪽 테이블을 기준으로 데이터를 병합
 - 오른쪽 테이블에 왼쪽 테이블에 있는 키 값이 존재하지 않는다면 NaN으로 출력
- 오른쪽 조인(right join) : 오른쪽 테이블 기준으로 데이터를 병합

04 병합과 연결

1. 병합

1.2 왼쪽 조인, 오른쪽 조인

In [4]: `pd.merge(df_left, df_right, on='subject_id', how='left')`

Out [4]:

	subject_id	test_score	first_name	last_name
0	1	51	NaN	NaN
1	2	15	NaN	NaN
2	3	15	NaN	NaN
3	4	61	Billy	Bonder
4	5	16	Brian	Black
5	7	14	Bryce	Brice
6	8	15	Betty	Btisan
7	9	1	NaN	NaN
8	10	61	NaN	NaN
9	11	16	NaN	NaN

04 병합과 연결

1. 병합

1.2 왼쪽 조인, 오른쪽 조인

In [5]: `pd.merge(df_left, df_right, on='subject_id', how='right')`

Out [5]:

	subject_id	test_score	first_name	last_name
0	4	61.0	Billy	Bonder
1	5	16.0	Brian	Black
2	6	NaN	Bran	Balwner
3	7	14.0	Bryce	Brice
4	8	15.0	Betty	Btisan

04 병합과 연결

1. 병합

1.3 완전 조인

- 완전 조인(full join): 두 테이블의 합집합을 의미
 - 양쪽에 같은 키 값이 있는 데이터는 합치고 나머지는 NaN

In [6]: `pd.merge(df_left, df_right, on='subject_id', how='outer')`

Out [6]:

	subject_id	test_score	first_name	last_name
0	1	51.0	NaN	NaN
1	2	15.0	NaN	NaN
2	3	15.0	NaN	NaN
3	4	61.0	Billy	Bonder
4	5	16.0	Brian	Black
5	7	14.0	Bryce	Brice
6	8	15.0	Betty	Btisan
7	9	1.0	NaN	NaN
8	10	61.0	NaN	NaN
9	11	16.0	NaN	NaN
10	6	NaN	Bran	Balwner

04 병합과 연결

1. 병합

1.3 완전 조인

[하나 더 알기] 인덱스에 의한 병합

- 인덱스 값을 키 값으로 하여 두 테이블을 병합할 수 있음
- 인덱스가 의미 있는 열로 지정되어 있거나, 두 데이터가 모두 순서대로 들어가 있는 경우에 사용
- `right_index`나 `left_index` 매개변수

```
In [7]: df_left.index = df_left.subject_id
del df_left["subject_id"]
df_right.index = df_right.subject_id
del df_right["subject_id"]

pd.merge(df_left, df_right, on='subject_id', how='inner')
```

Out [7]:

	test_score	first_name	last_name
subject_id			
4	61	Billy	Bonder
5	16	Brian	Black
7	14	Bryce	Brice
8	15	Betty	Btisan

04 병합과 연결

2. 연결

- 연결(concatenate) : 두 테이블을 그대로 붙임
- 데이터의 스키마가 동일할 때 그대로 연결
- 주로 세로로 데이터를 연결
 - concat 함수 : 두 개의 서로 다른 테이블을 하나로 합침
 - append 함수 : 기존 테이블 하나에 다른 테이블을 붙임

[TIP] append 함수는 파일을 한 개씩 합치기 때문에 두 개 이상의 데이터프레임을 합칠 때에는 concat 함수를 쓰는 것이 좋다.

04 병합과 연결

2. 연결

	account	number	name	sku	quantity	unit price	ext price	date
0		163416	Purdy-Kunde	S1-30248	19	65.03	1235.57	2014-03-01 16:07:40
1		527099	Sanford and Sons	S2-82423	3	76.21	228.63	2014-03-01 17:18:01
2		527099	Sanford and Sons	B1-50809	8	70.78	566.24	2014-03-01 18:53:09
3		737550	Fritsch, Russel and Anderson	B1-50809	20	50.11	1002.20	2014-03-01 23:47:17
4		688981	Keeling LLC	B1-86481	-1	97.16	-97.16	2014-03-02 01:46:44
...	
137		737550	Fritsch, Russel and Anderson	B1-65551	12	56.24	674.88	2014-03-31 08:43:24
138		642753	Pollich LLC	S1-93683	21	92.57	1943.97	2014-03-31 11:37:34
139		412290	Jerde-Hilpert	B1-20000	30	22.38	671.40	2014-03-31 21:41:31
140		307599	Kassulke, Ondricka and Metz	S2-16558	46	56.04	2577.84	2014-03-31 22:11:22
141		672390	Kuhn-Gusikowski	B1-04202	19	27.86	529.34	2014-03-31 23:13:14

그림 4-10 데이터 테이블

04 병합과 연결

2. 연결

- In[8]에서 경로(path)는 설정이 동일한 경우에는 그대로 사용하고 다른 경우에는 본인이 엑셀을 저장한 경로로 바꾼다.

In [8]:	<pre>import os filenames = [os.path.join("c:/source/ch04", filename) for filename in os.listdir("c:/source/ch04") if "sales" in filename] print(filenames)</pre>
Out [8]:	<pre>['c:/source/ch04\\sales-feb-2014.xlsx', 'c:/source/ch04\\sales-jan-2014.xlsx', 'c:/source/ch04\\sales-mar-2014.xlsx']</pre>

04 병합과 연결

2. 연결

In [9]:	!pip install --user --upgrade openpyxl
In [10]:	<pre>df_list = [pd.read_excel(filename, engine="openpyxl") for filename in filenames] for df in df_list: print(type(df), len(df))</pre>
Out [10]:	<pre><class 'pandas.core.frame.DataFrame'> 108 <class 'pandas.core.frame.DataFrame'> 134 <class 'pandas.core.frame.DataFrame'> 142</pre>

04 병합과 연결

2. 연결

- axis=0으로 세로로 연결
- reset_index(drop=True) 함수 사용하여 중복된 인덱스를 제거

```
In [11]: df = pd.concat(df_list, axis=0)
print(len(df)) # 384
df.reset_index(drop=True)
```

Out [11]:

	account number		name	sku	quantity	unit price	ext price	date
0	383080		Will LLC	B1-20000	7	33.69	235.83	2014-02-01 09:04:59
1	412290		Jerde-Hilpert	S1-27722	11	21.12	232.32	2014-02-01 11:51:46
2	412290		Jerde-Hilpert	B1-86481	3	35.99	107.97	2014-02-01 17:24:32
3	412290		Jerde-Hilpert	B1-20000	23	78.90	1814.70	2014-02-01 19:56:48
4	672390		Kuhn-Gusikowski	S1-06532	48	55.82	2679.36	2014-02-02 03:45:20
...
379	737550	Fritsch, Russel and Anderson		B1-65551	12	56.24	674.88	2014-03-31 08:43:24
380	642753		Pollich LLC	S1-93683	21	92.57	1943.97	2014-03-31 11:37:34
381	412290		Jerde-Hilpert	B1-20000	30	22.38	671.40	2014-03-31 21:41:31
382	307599	Kassulke, Ondricka and Metz		S2-16558	46	56.04	2577.84	2014-03-31 22:11:22
383	672390		Kuhn-Gusikowski	B1-04202	19	27.86	529.34	2014-03-31 23:13:14

384 rows X 7 columns

04 병합과 연결

2. 연결

```
In [12]: df_1, df_2, df_3 = [pd.read_excel(filename, engine="openpyxl") for
filename in filenames]
df = df_1.append(df_2)
df = df.append(df_3)
df
```

Out [12]:

FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use **pandas.concat** instead.

```
df = df_1.append(df_2)
```

The frame.append method is deprecated and will be removed from pandas in a future version. Use **pandas.concat** instead.

```
df = df.append(df_3)
```

Out [12]:

	account number	name	sku	quantity	unit price	ext price	date
0	383080	Will LLC	B1-20000	7	33.69	235.83	2014-02-01 09:04:59
1	412290	Jerde-Hilpert	S1-27722	11	21.12	232.32	2014-02-01 11:51:46
2	412290	Jerde-Hilpert	B1-86481	3	35.99	107.97	2014-02-01 17:24:32
3	412290	Jerde-Hilpert	B1-20000	23	78.90	1814.70	2014-02-01 19:56:48
4	672390	Kuhn-Gusikowski	S1-06532	48	55.82	2679.36	2014-02-02 03:45:20
...
137	737550	Fritsch, Russel and Anderson	B1-65551	12	56.24	674.88	2014-03-31 08:43:24
138	642753	Pollich LLC	S1-93683	21	92.57	1943.97	2014-03-31 11:37:34
139	412290	Jerde-Hilpert	B1-20000	30	22.38	671.40	2014-03-31 21:41:31
140	307599	Kassulke, Ondricka and Metz	S2-16558	46	56.04	2577.84	2014-03-31 22:11:22
141	672390	Kuhn-Gusikowski	B1-04202	19	27.86	529.34	2014-03-31 23:13:14

384 rows × 7 columns



<데이터 시각화>

목차

01 맷플롯립

02 시본

03 플롯리

01

맷플롯립

01 맷플롯립

- 데이터 시각화(data visualization) : 데이터 분석 결과를 쉽게 이해할 수 있도록 시각적으로 표현하고 전달

1. 맷플롯립의 구조

- 맷플롯립(matplotlib) : 매트랩(matlab) 기능을 파이썬에서 그대로 사용하도록 하는 시각화 모듈
 - 엑셀의 정형화된 차트나 그래프 작성, 다양한 함수 지원
 - 매트랩을 포장(wrapping)해서 맷플롯립을 지원

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

01 맷플롯립

1. 맷플롯립의 구조

1.1 파이플롯

- 맷플롯립을 이용할 때 가장 기본이 되는 객체
- 파이플롯(pyplot) 위에 그림 (figure) 객체를 올리고 그 위에 그래프에 해당하는 축(axes)을 올림
- 그림 위에 축을 여러 장 올리면 여러 개의 그래프 작성

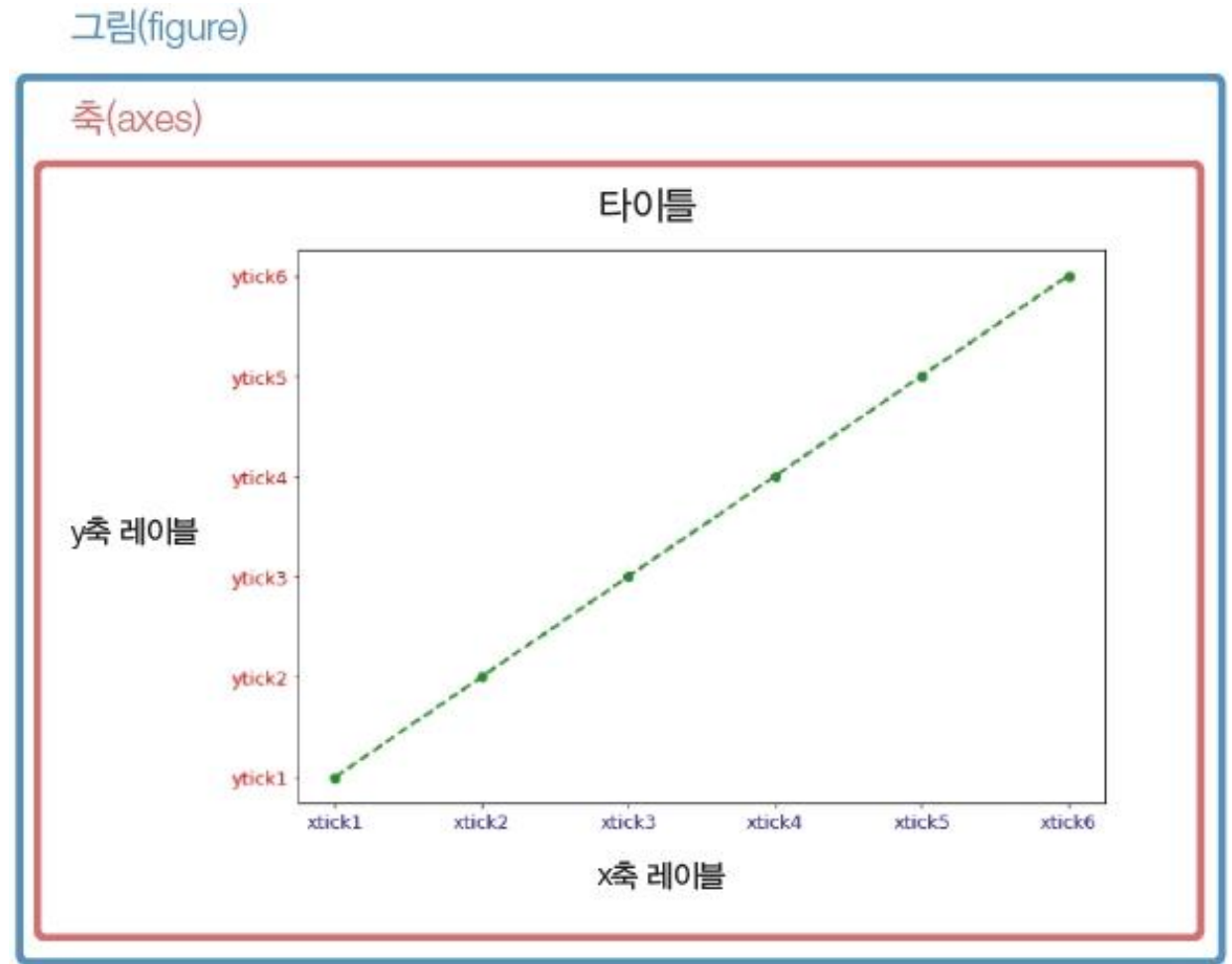


그림 5-1 파이플롯(pyplot), 그림(figure), 축(axes)의 개념

01 맷플롯립

1. 맷플롯립의 구조

1.1 파이플롯

- X 객체와 Y 객체 값 쌍으로 좌표평면 위에 점을 찍음
- plot 함수로 점들을 연결

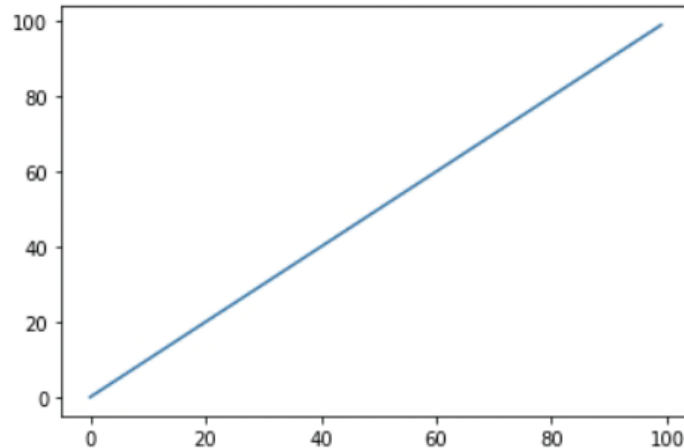
In [1]: `import matplotlib.pyplot as plt # matplotlib 모듈 호출`

`X = range(100)`

`Y = range(100)`

`plt.plot(X, Y)`

Out [1]:



01 맷플롯립

1. 맷플롯립의 구조

1.1 파이플롯

- pyplot 객체 내부에 있는 하나의 그림 객체 위에 코사인 그래프와 사인 그래프를 그림

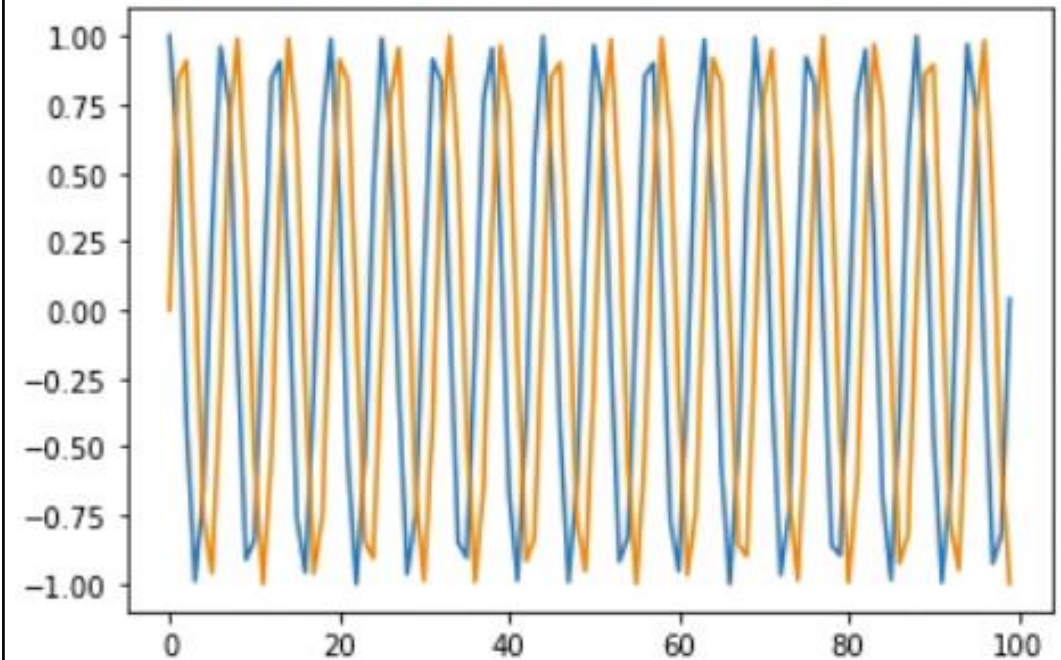
```
In [2]: import numpy as np
# numpy 모듈 호출

X_1 = range(100)
Y_1 = [np.cos(value) for value in X]

X_2 = range(100)
Y_2 = [np.sin(value) for value in X]

plt.plot(X_1, Y_1)
plt.plot(X_2, Y_2)
plt.show()
```

Out [2]:



01 맷플롯립

1. 맷플롯립의 구조

1.2 그림과 축

- 그림은 그래프를 작성하는 밑바탕이 됨
- 축은 실제로 그래프를 작성하는 공간

```
In [3]: fig, ax = plt.subplots() # (1) figure와 axes 객체 할당

X_1 = range(100)
Y_1 = [np.cos(value)
        for value in X]

ax.plot(X_1, Y_1) # (2) plot 함수를 사용하여 그래프 생성
ax.set(title='cos graph', # (3) 그래프 제목, x축 라벨, y축 라벨 설정
        xlabel='X',
        ylabel='Y');

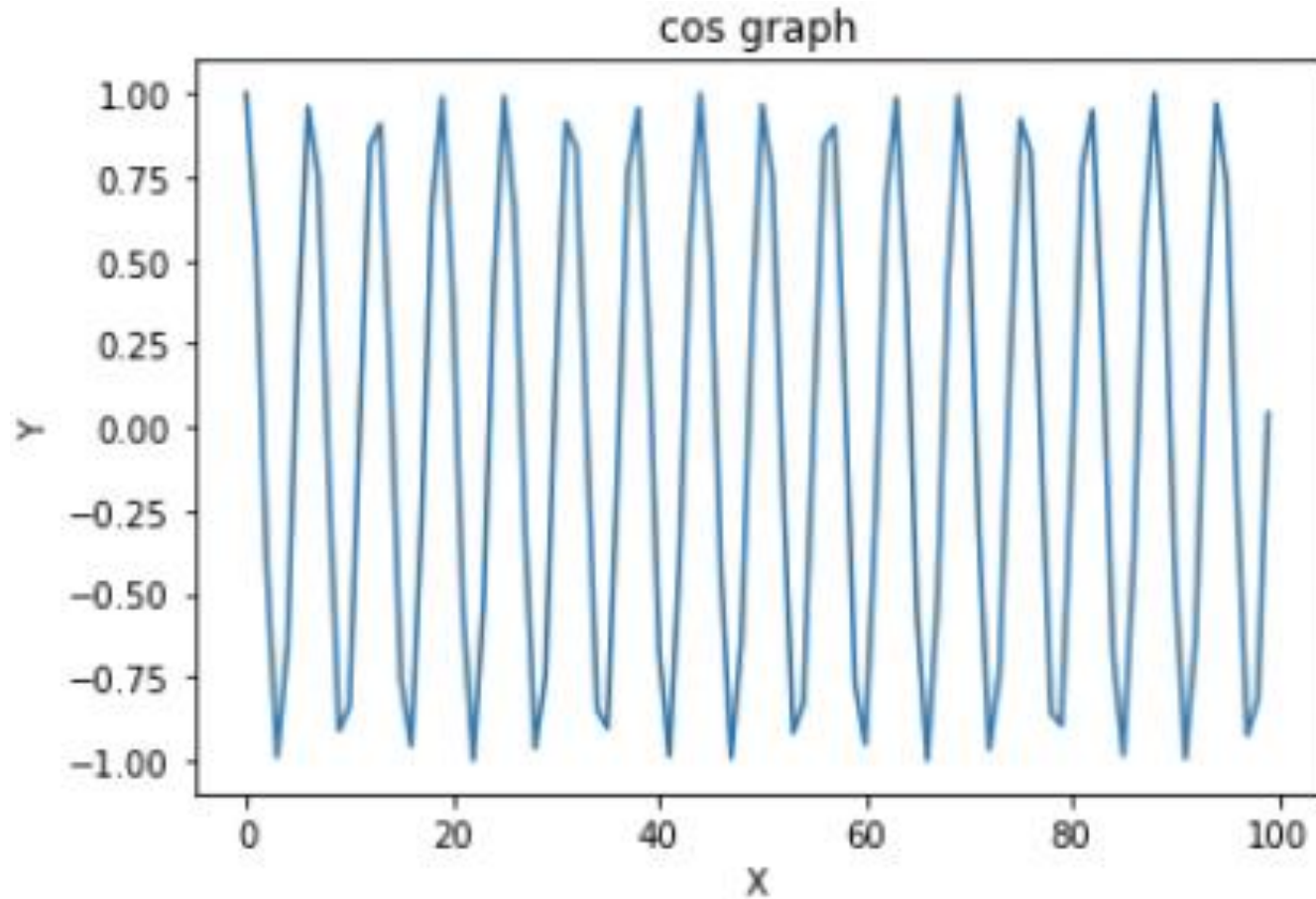
plt.show() # (4) 그래프 출력
```

01 맷플롯립

1. 맷플롯립의 구조

1.2 그림과 축

Out[3]:



01 맷플롯립

1. 맷플롯립의 구조

1.2 그림과 축

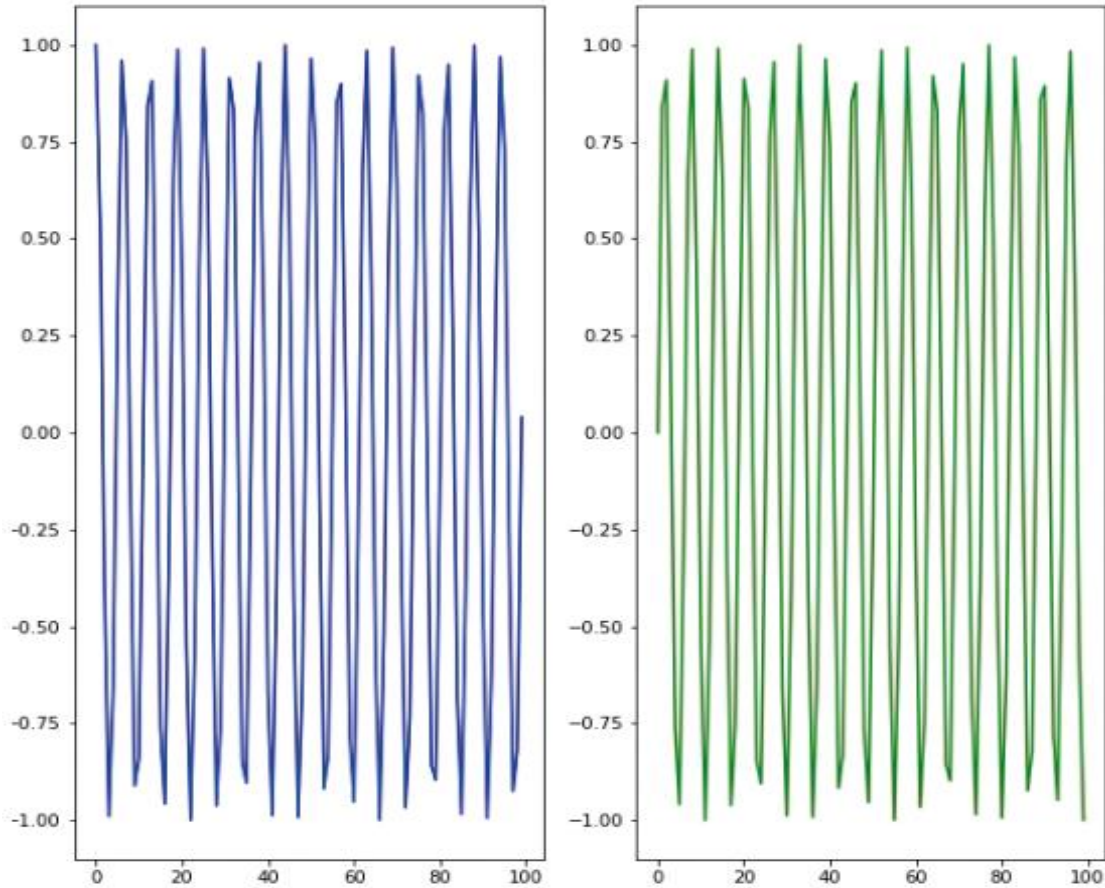
In [4]:	<pre>fig = plt.figure() fig.set_size_inches(10,10) ax_1 = fig.add_subplot(1,2,1) ax_2 = fig.add_subplot(1,2,2) ax_1.plot(X_1, Y_1, c="b") ax_2.plot(X_2, Y_2, c="g") plt.show()</pre>	<pre># (1) figure 반환 # (2) figure의 크기 지정 # (3) 첫 번째 그래프 생성 # (4) 두 번째 그래프 생성 # (5) 첫 번째 그래프 설정 # (6) 두 번째 그래프 설정 # (7) 그래프 출력</pre>
---------	---	---

01 맷플롯립

1. 맷플롯립의 구조

1.2 그림과 축

Out[4]:



01 맷플롯립

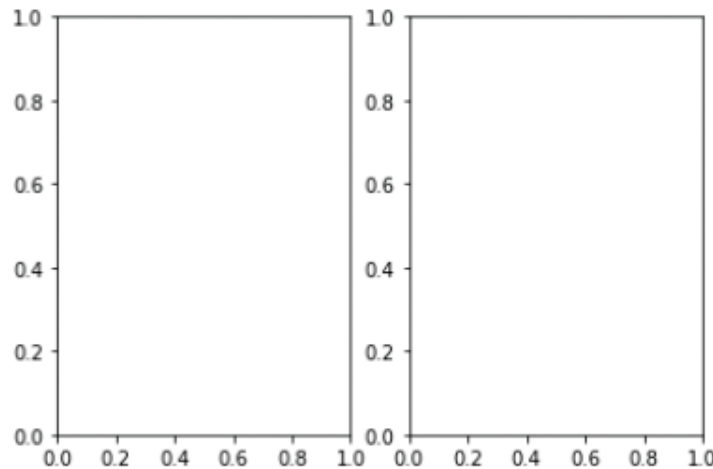
1. 맷플롯립의 구조

1.3 서브플롯 행렬

- 축을 여러 개 만들 때 서브플롯으로 축 객체 공간 확보
 - 그림 객체에서 `add_subplot` 함수 사용
 - 또는 `plot` 객체에서 `subplots` 함수 사용

```
In [5]: fig, ax = plt.subplots(nrows=1, ncols=2)  
print(ax)
```

Out [5]:



01 맷플롯립

1. 맷플롯립의 구조

1.3 서브플롯 행렬

- ax 변수에 축 객체가 넘파이 배열 타입으로 생성됨

In [6]:	<code>print(type(ax))</code>
Out [6]:	<code><class 'numpy.ndarray'></code>

01 맷플롯립

1. 맷플롯립의 구조 1.3 서브플롯 행렬

```
In [7]: import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1,1,100) # (1) x 값과 y_n 값 생성
y_1 = np.sin(x)
y_2 = np.cos(x)
y_3 = np.tan(x)
y_4 = np.exp(x)

fig, ax = plt.subplots(2, 2) # (2) 2×2 figure 객체를 생성

ax[0, 0].plot(x, y_1)      # (3) 첫 번째 그래프 생성
ax[0, 1].plot(x, y_2)      # (4) 두 번째 그래프 생성
ax[1, 0].plot(x, y_3)      # (5) 세 번째 그래프 생성
ax[1, 1].plot(x, y_4)      # (6) 네 번째 그래프 생성

plt.show()
```

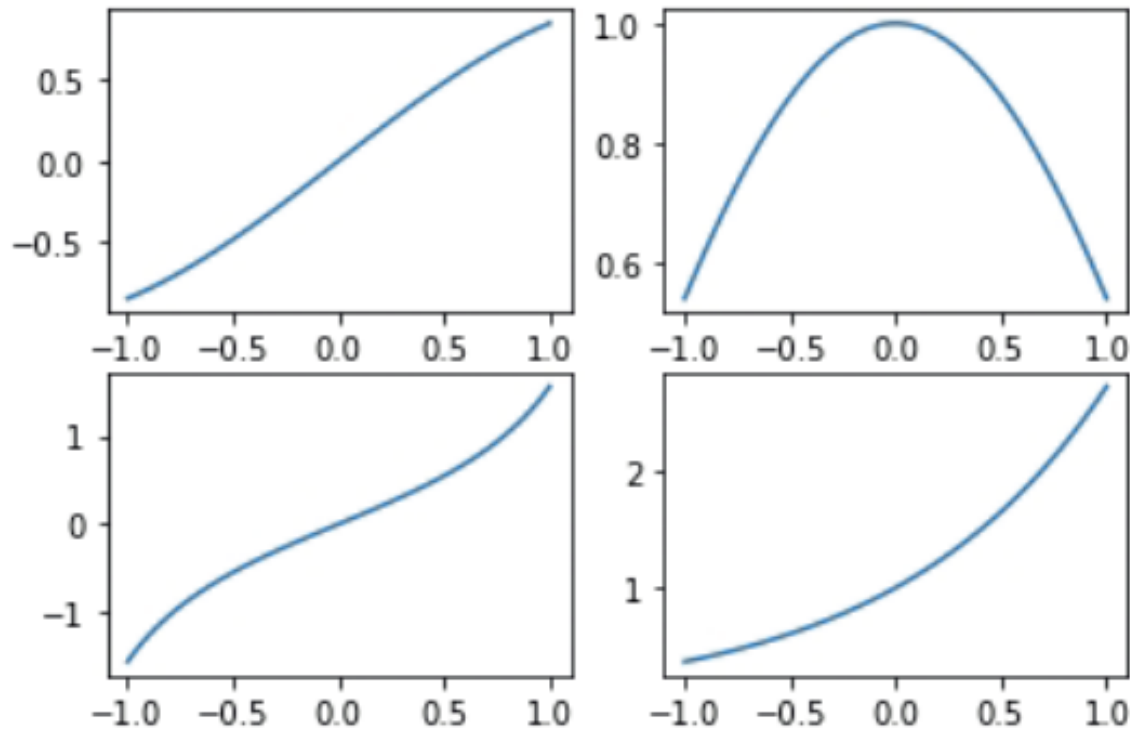
01 맷플롯립

1. 맷플롯립의 구조

1.3 서브플롯 행렬

- # (2) subplots 함수에서 2×2 행렬 그림 객체가 생성되어 ax 변수에 4개의 축 객체가 2×2 넘파이 배열 형태로 들어가 있음
- 넘파이 배열의 인덱스로 각 축 객체에 접근하여 그래프를 생성

Out[7]:



01 맷플롯립

1. 맷플롯립의 구조

1.3 서브플롯 행렬

- 행과 열을 지정하고, 세 번째 숫자는 축의 위치

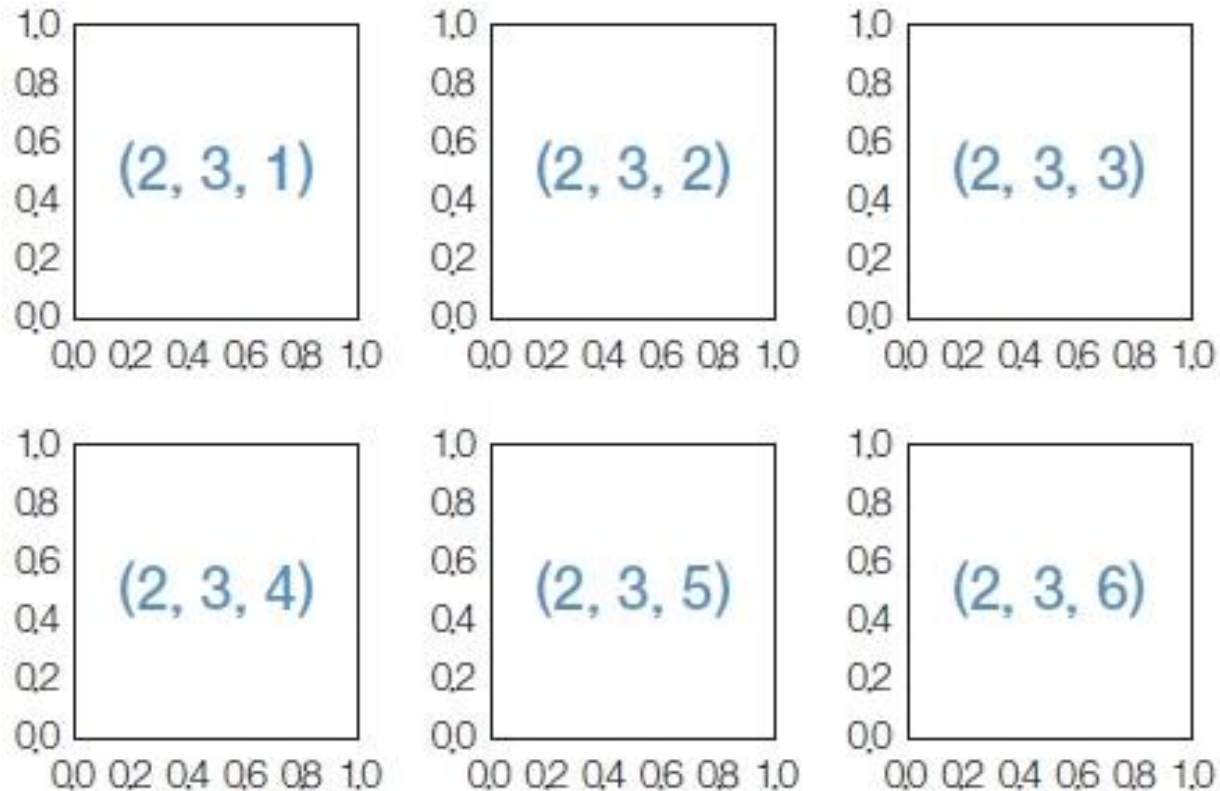


그림 5-2 지정된 행렬에서 축(axes)의 위치 배열

01 맷플롯립

1. 맷플롯립의 구조

1.3 서브플롯 행렬

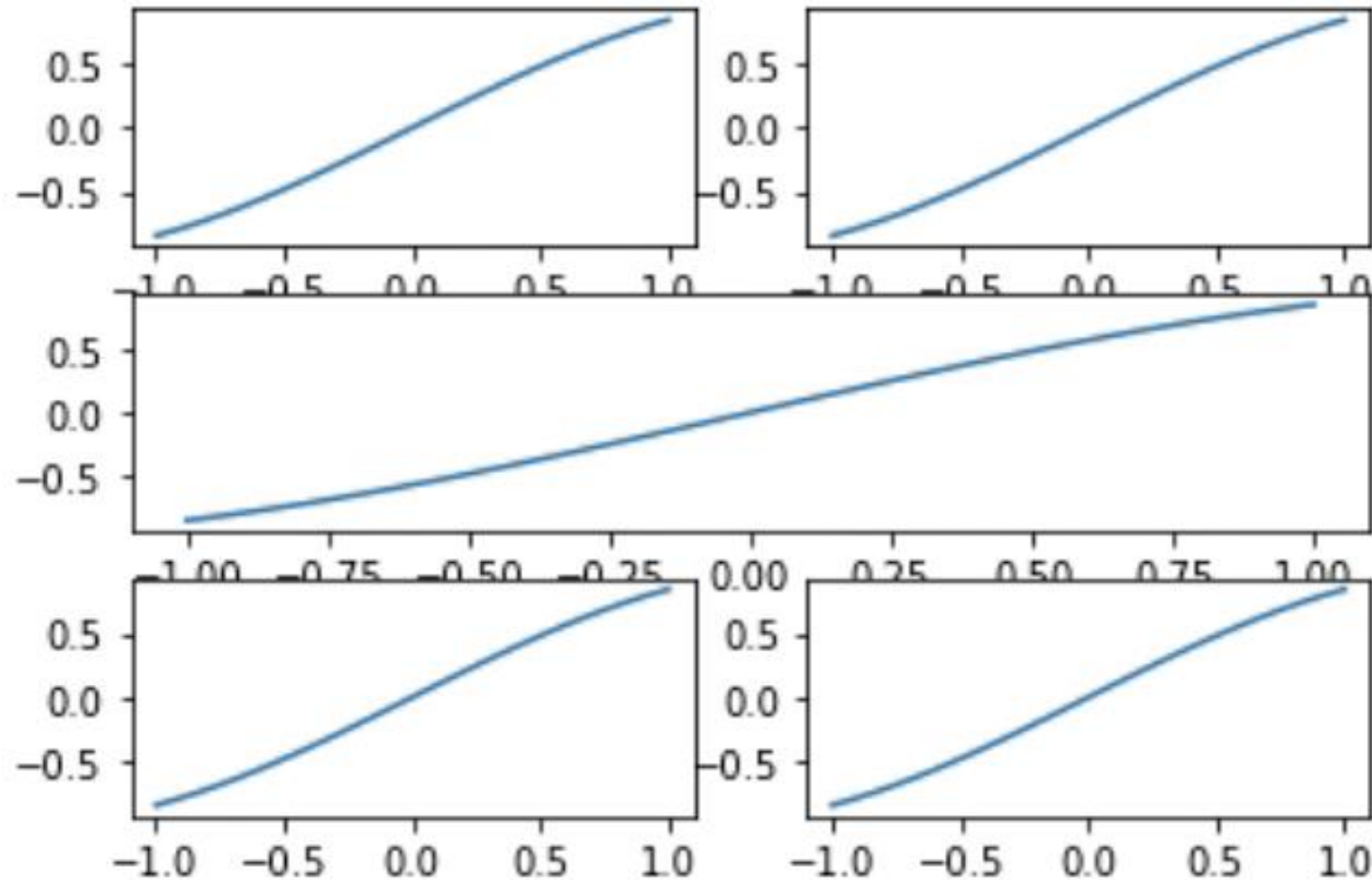
```
In [8]: ax1 = plt.subplot(321) # (1) 행 3, 열 2, 첫 번째 공간에 axes 생성
plt.plot(x, y_1)
ax2 = plt.subplot(322) # (2) 행 3, 열 2, 두 번째 공간에 axes 생성
plt.plot(x, y_1)
ax3 = plt.subplot(312) # (3) 행 3, 열 1, 두 번째 공간에 axes 생성
plt.plot(x, y_1)
ax4 = plt.subplot(325) # (4) 행 3, 열 2, 다섯 번째 공간에 axes 생성
plt.plot(x, y_1)
ax5 = plt.subplot(326) # (5) 행 3, 열 2, 여섯 번째 공간에 axes 생성
plt.plot(x, y_1)
plt.show()
```

01 맷플롯립

1. 맷플롯립의 구조

1.3 서브플롯 행렬

Out [8]:



01 맷플롯립

2. 맷플롯립으로 그래프 꾸미기

2.1 색상

- color 또는 c 매개변수로 색상 변경
 - RGB 값을 사용해서 #을 붙여 16진법으로 색상 표현
 - 또는 b, g, r, c, m, y, k, w 등 약어 입력

01 맷플롯립

2. 맷플롯립으로 그래프 꾸미기

2.1 색상

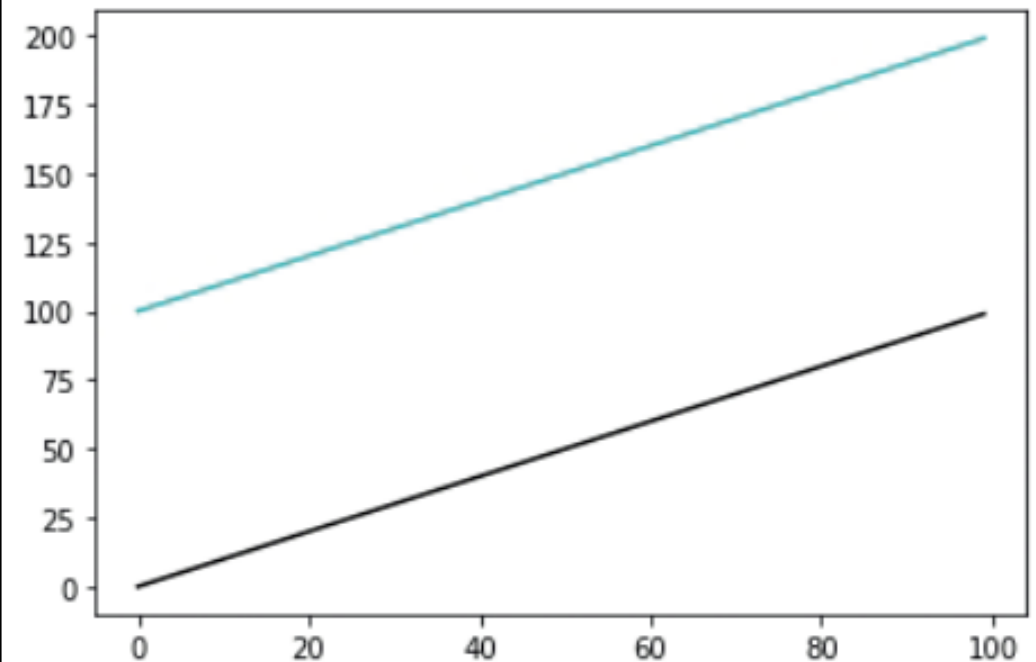
```
In [9]: X_1 = range(100)
        Y_1 = [value for value in X]

        X_2 = range(100)
        Y_2 = [value + 100 for value in X]

        plt.plot(X_1, Y_1, color="#000000")
        plt.plot(X_2, Y_2, c="c")

        plt.show()
```

Out [9]:



01 맷플롯립

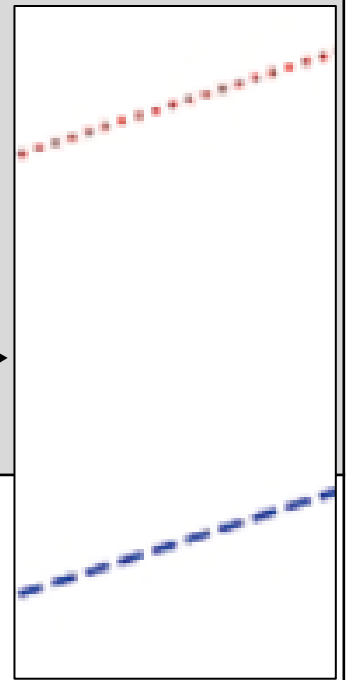
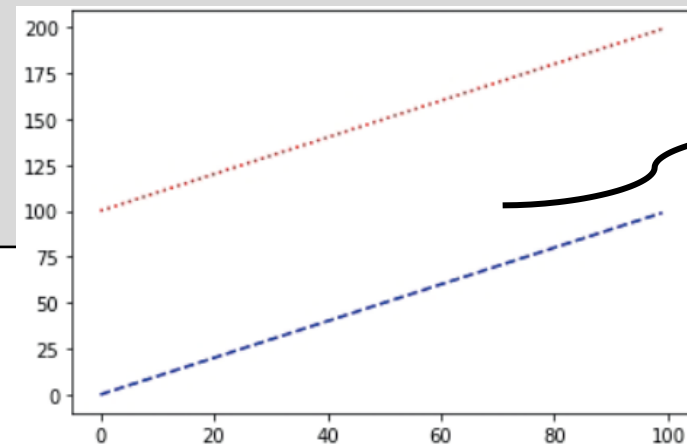
2. 맷플롯립으로 그래프 꾸미기

2.2 선의 형태

- linestyle 또는 ls로 선의 형태를 정의
 - dashed : 점선 형태, solid : 실선 형태

```
In [10]: plt.plot(X_1, Y_1, c="b",  
                linestyle="dashed")  
plt.plot(X_2, Y_2, c="r",  
                ls="dotted")  
  
plt.show()
```

Out [10]:



01 맷플롯립

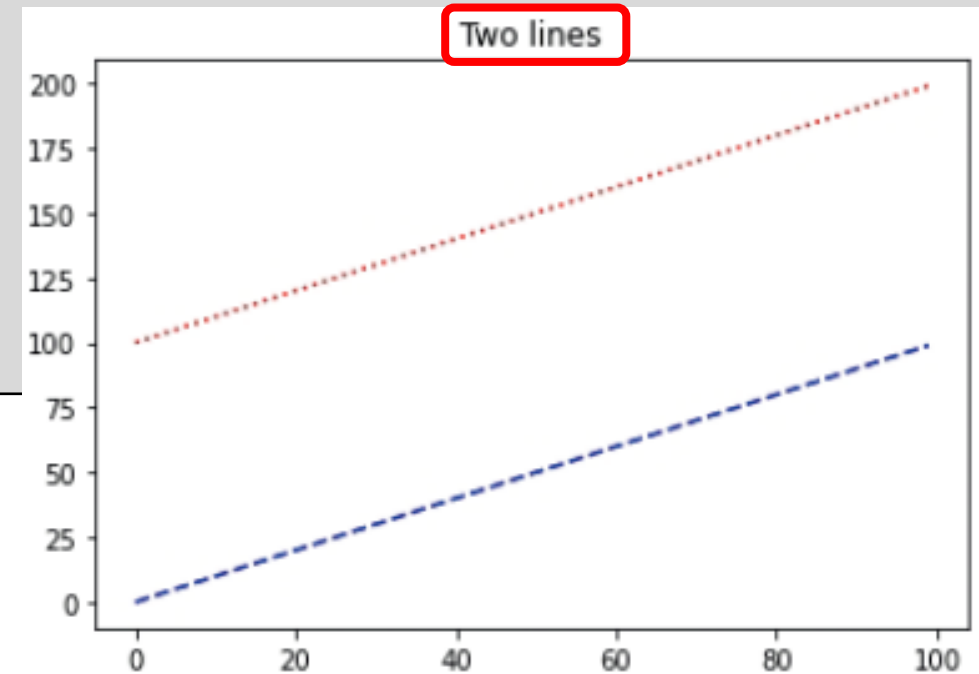
2. 맷플롯립으로 그래프 꾸미기

2.3 제목

- 축 객체마다 제목을 달 수 있음

```
In [11]: plt.plot(X_1, Y_1, c="b",  
                linestyle="dashed")  
         plt.plot(X_2, Y_2, c="r",  
                ls="dotted")  
  
         plt.title("Two lines")  
         plt.show()
```

Out [11]:



01 맷플롯립

2. 맷플롯립으로 그래프 꾸미기

2.3 제목

```
In [12]: fig = plt.figure()
fig.set_size_inches(10,10)

ax_1 = fig.add_subplot(1,2,1)
ax_2 = fig.add_subplot(1,2,2)

ax_1.plot(X_1, Y_1, c="b")
ax_1.set_title("Figure 1")
ax_2.plot(X_2, Y_2, c="g")
ax_2.set_title("Figure 2")

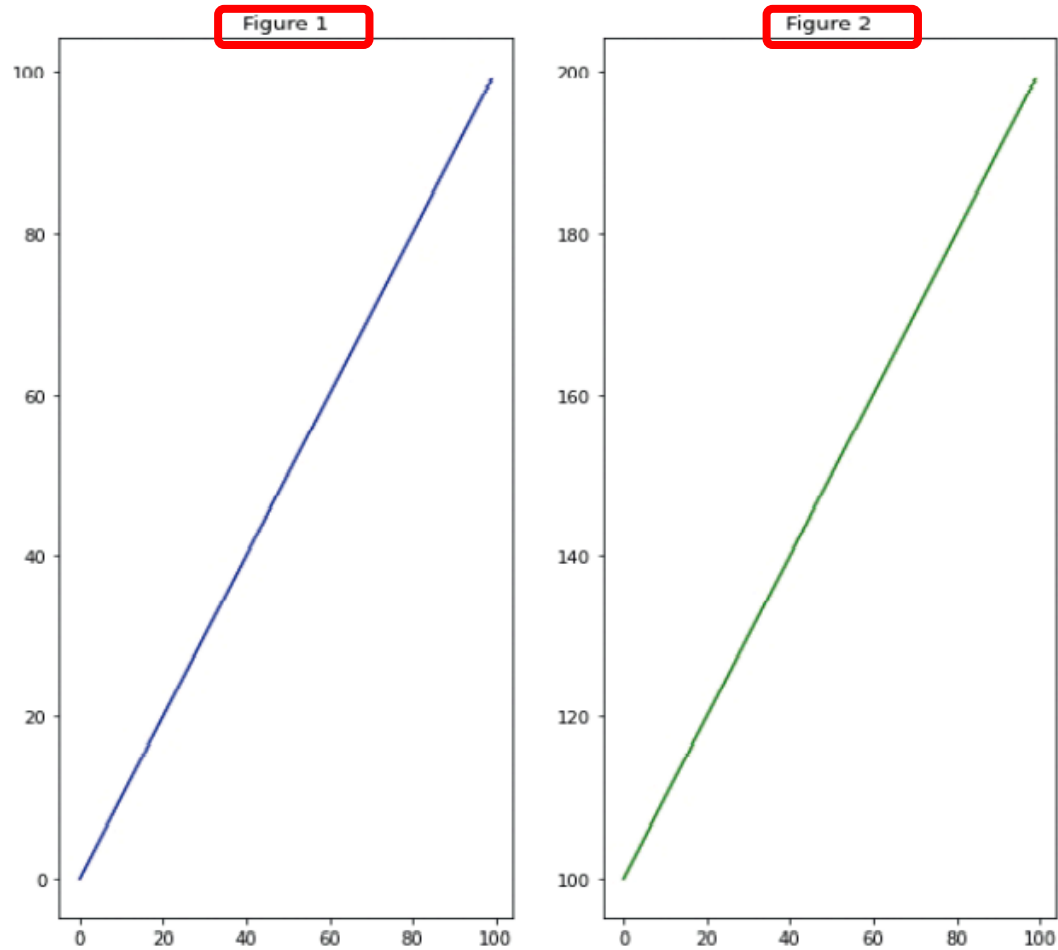
plt.show()
```


01 맷플롯립

2. 맷플롯립으로 그래프 꾸미기

2.3 제목

Out [12]:



01 맷플롯립

2. 맷플롯립으로 그래프 꾸미기

2.4 범례

- 축 객체마다 범례(legend)를 설정할 수 있음
- **legend 함수** 사용하여 생성
 - **shadow** 매개변수로 범례에 그림자 효과 추가
 - **loc** 매개변수로 범례의 위치 지정
 - 값은 center, upper right 등 총 11가지
 - best라고 지정하면 적절한 위치에 범례가 놓임

01 맷플롯립

2. 맷플롯립으로 그래프 꾸미기

2.4 범례

```
In [13]: plt.plot(X_1, Y_1,
                color="b",
                linestyle="dashed",
                label='line_1')
plt.plot(X_2, Y_2,
                color="r",
                linestyle="dotted",
                label='line_2')
plt.legend(
    shadow=True,
    fancybox=False, # (3) legend 백그라운드인 FancyboxPatch에 둥근모서리 사용여부
    loc="upper right")

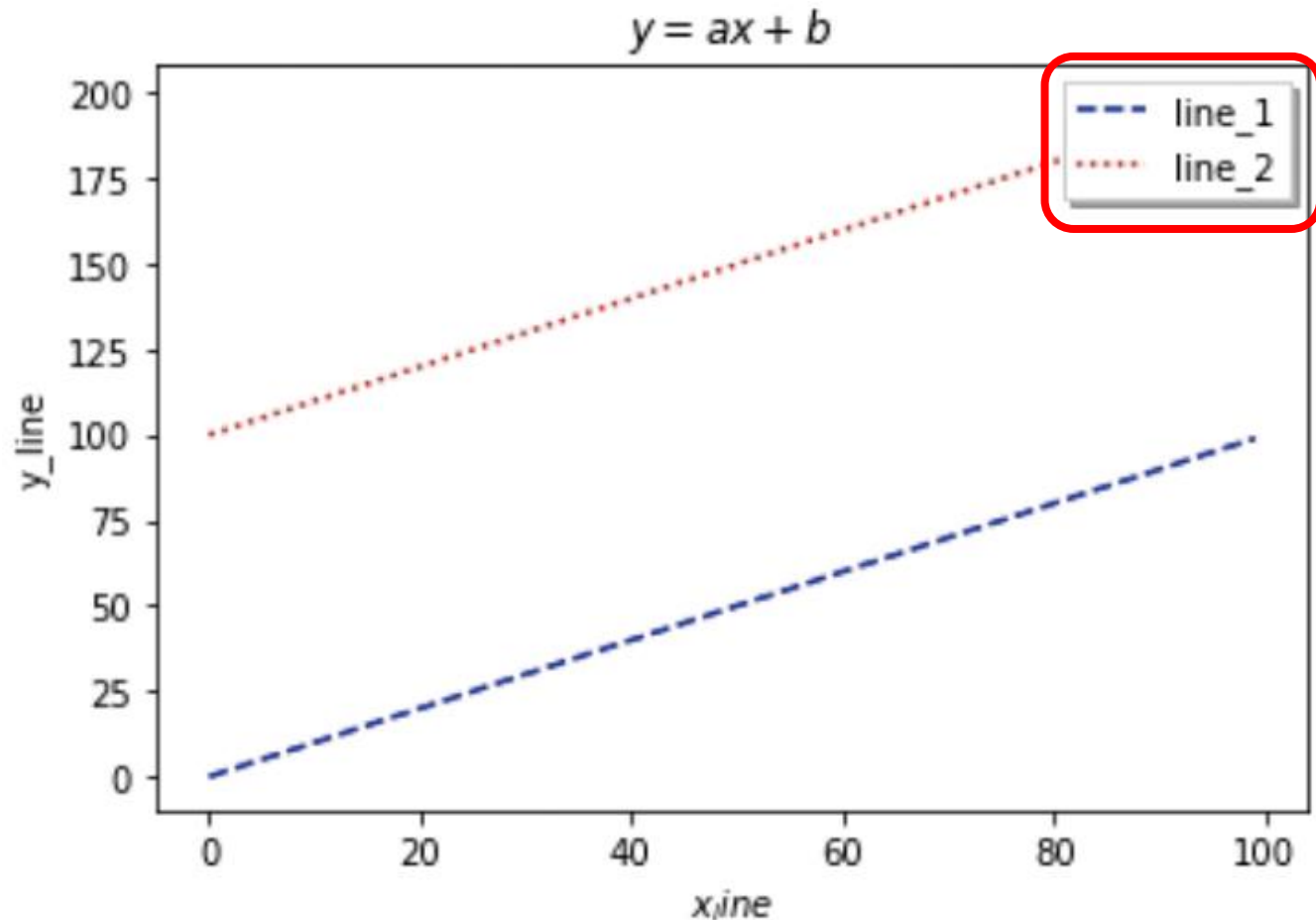
plt.title('$y = ax + b$')
plt.xlabel('$x_{line}$')
plt.ylabel('$y_{line}$')
```

01 맷플롯립

2. 맷플롯립으로 그래프 꾸미기

2.4 범례

Out [13]:



01 맷플롯립

3. 맷플롯립에서 사용하는 그래프

3.1 산점도

- 산점도(scatter plot) : 데이터 분포를 2차원 평면에 표현
 - 매개변수 `c`는 포인트 색상을 지정
 - `marker`는 포인트 모양을 지정
 - `size`는 포인트 크기를 지정
 - `alpha`는 포인트 불투명도를 지정

01 맷플롯립

3. 맷플롯립에서 사용하는 그래프

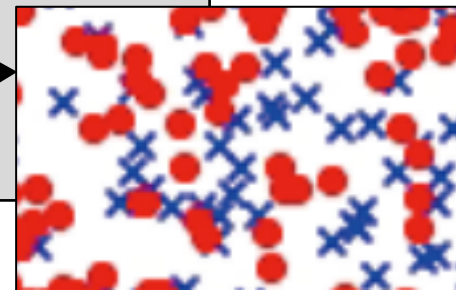
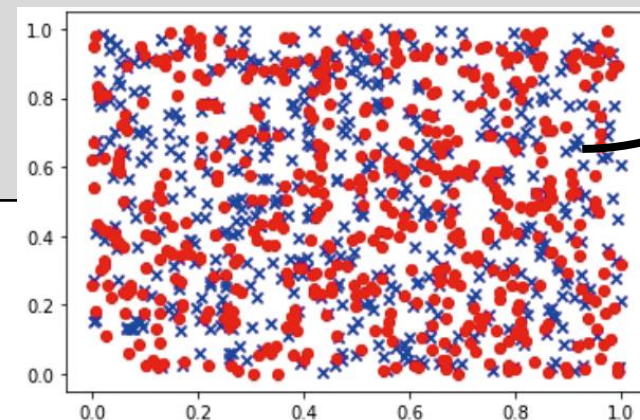
3.1 산점도

```
In [14]: data_1 = np.random.rand(512, 2)
data_2 = np.random.rand(512, 2)

plt.scatter(data_1[:,0],
            data_1[:,1],
            c="b", marker="x")
plt.scatter(data_2[:,0],
            data_2[:,1],
            c="r", marker="o")

plt.show()
```

Out [14]:



01 맷플롯립

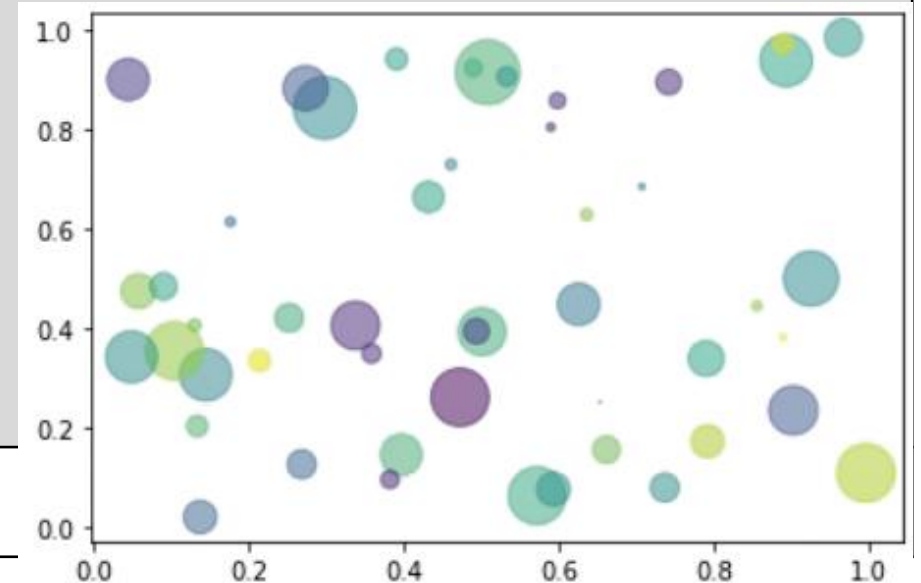
3. 맷플롯립에서 사용하는 그래프

3.1 산점도

```
In [15]: N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (
    15 * np.random.rand(N))**2

plt.scatter(x, y,
            s=area, c=colors,
            alpha=0.5)
plt.show()
```

Out [15]:



01 맷플롯립

3. 맷플롯립에서 사용하는 그래프

3.2 막대그래프

- 막대그래프(bar graph) : 데이터의 개수나 크기를 비교

```
In [16]: data = [[5., 25., 50., 20.],  
               [4., 23., 51., 17],  
               [6., 22., 52., 19]] # (1) 데이터 생성  
  
X = np.arange(0,8,2) # (2) X 좌표 시작점  
  
plt.bar(X + 0.00, data[0], color = 'b', width = 0.50) # (3) 3개의 막대그래프 생성  
plt.bar(X + 0.50, data[1], color = 'g', width = 0.50)  
plt.bar(X + 1.0, data[2], color = 'r', width = 0.50)  
  
plt.xticks(X+0.50, ("A","B","C", "D")) # (4) X축에 표시될 이름과 위치 설정  
  
plt.show() # (5) 막대그래프 출력
```

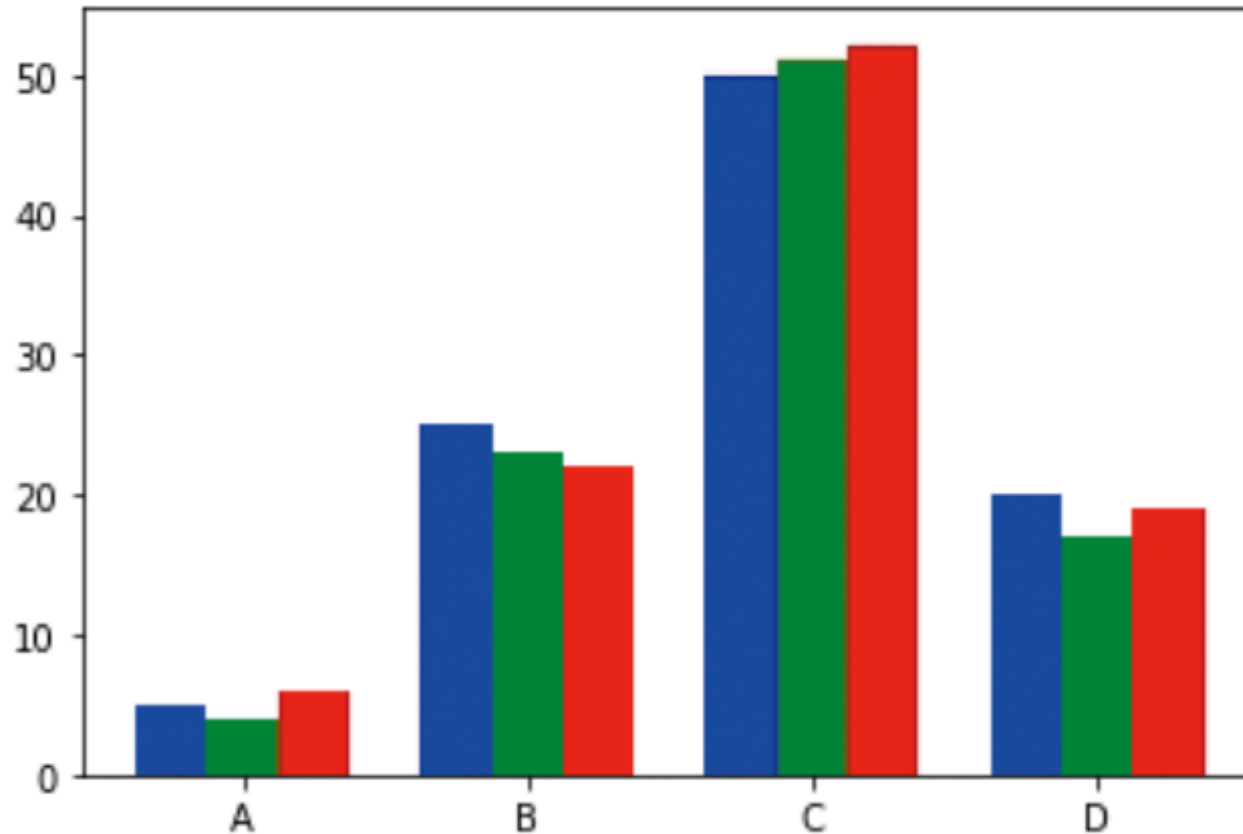

01 맷플롯립

3. 맷플롯립에서 사용하는 그래프

3.2 막대그래프

- 막대그래프(bar graph) : 데이터의 개수나 크기를 비교

Out [16]:



01 맷플롯립

3. 맷플롯립에서 사용하는 그래프

3.3 누적 막대그래프

- 누적 막대그래프(stacked bar graph) : 데이터를 밑에서부터 쌓아올려 데이터를 표현

In [17]: (뒷장 연결 코드 (code) 더 있음)	<pre>data = np.array([[5., 25., 50., 20.], [4., 23., 51., 17], [6., 22., 52., 19]]) color_list = ['b', 'g', 'r'] data_label = ["A","B","C"] X = np.arange(data.shape[1]) data = np.array([[5., 5., 5., 5.], [4., 23., 51., 17], [6., 22., 52., 19]])</pre>
---------------------------------------	--

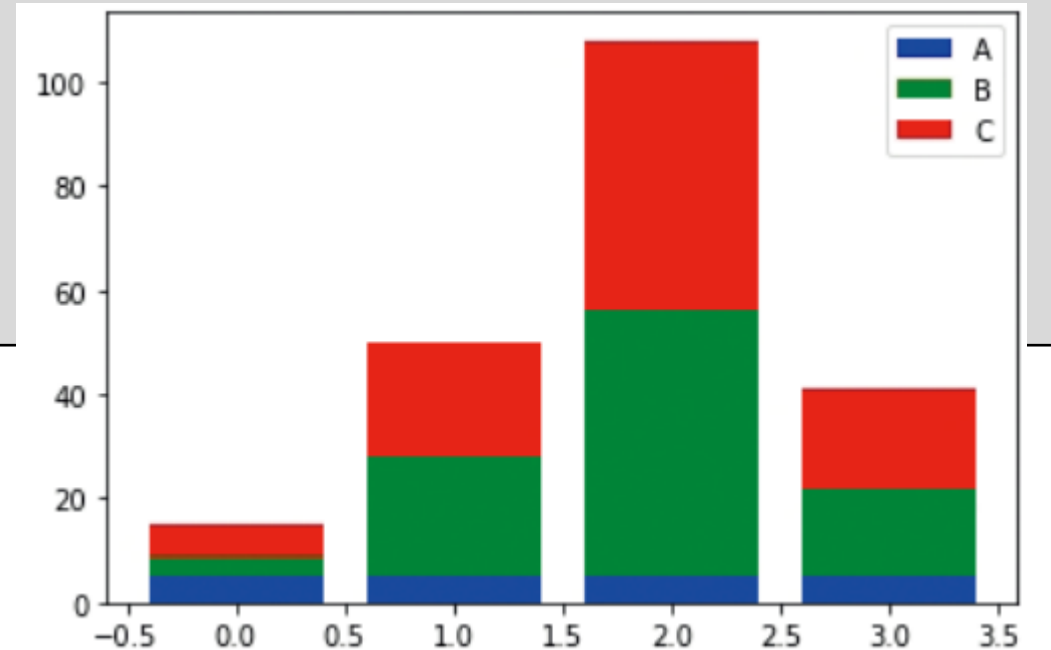
01 맷플롯립

3. 맷플롯립에서 사용하는 그래프

3.3 누적 막대그래프

```
In [17]: for i in range(3):  
          plt.bar(X, data[i],  
                  bottom = np.sum(  
                      data[:i], axis=0),  
                  color = color_list[i],  
                  label=data_label[i])  
  
          plt.legend()  
          plt.show()
```

Out[17]:



01 맷플롯립

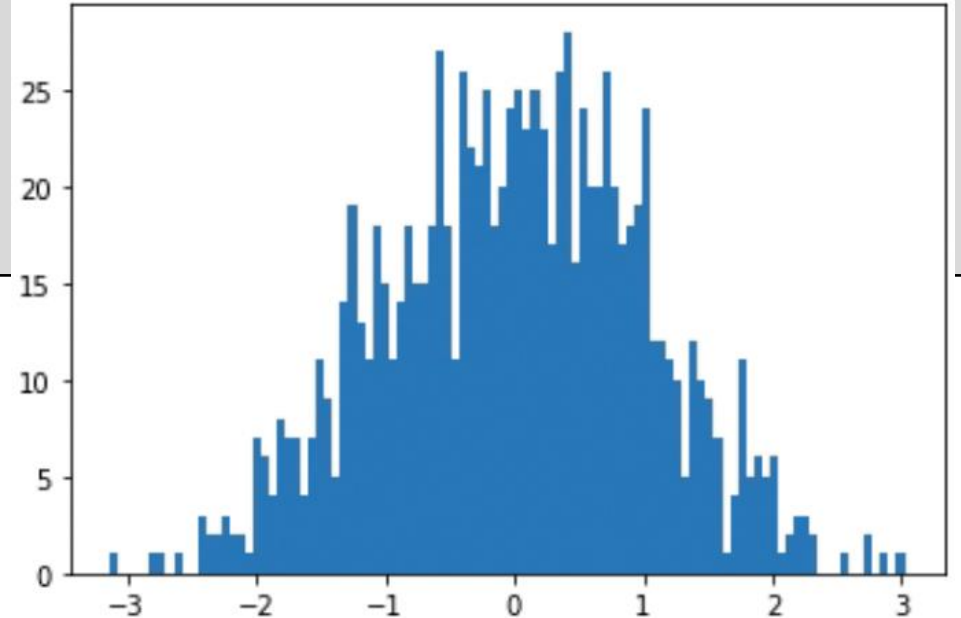
3. 맷플롯립에서 사용하는 그래프

3.4 히스토그램

- 히스토그램(histogram) : 데이터의 분포를 표현
 - hist 함수로 히스토그램 생성, 매개변수 bins로 막대 개수 지정

```
In [18]: N = 1000  
X = np.random.normal(size=N)  
  
plt.hist(X, bins=100)  
plt.show()
```

Out[18]:



01 맷플롯립

3. 맷플롯립에서 사용하는 그래프

3.5 상자그림

- 상자그림(boxplot) : 사분위수를 시각화하여 데이터의 분포와 밀집 정도를 표현

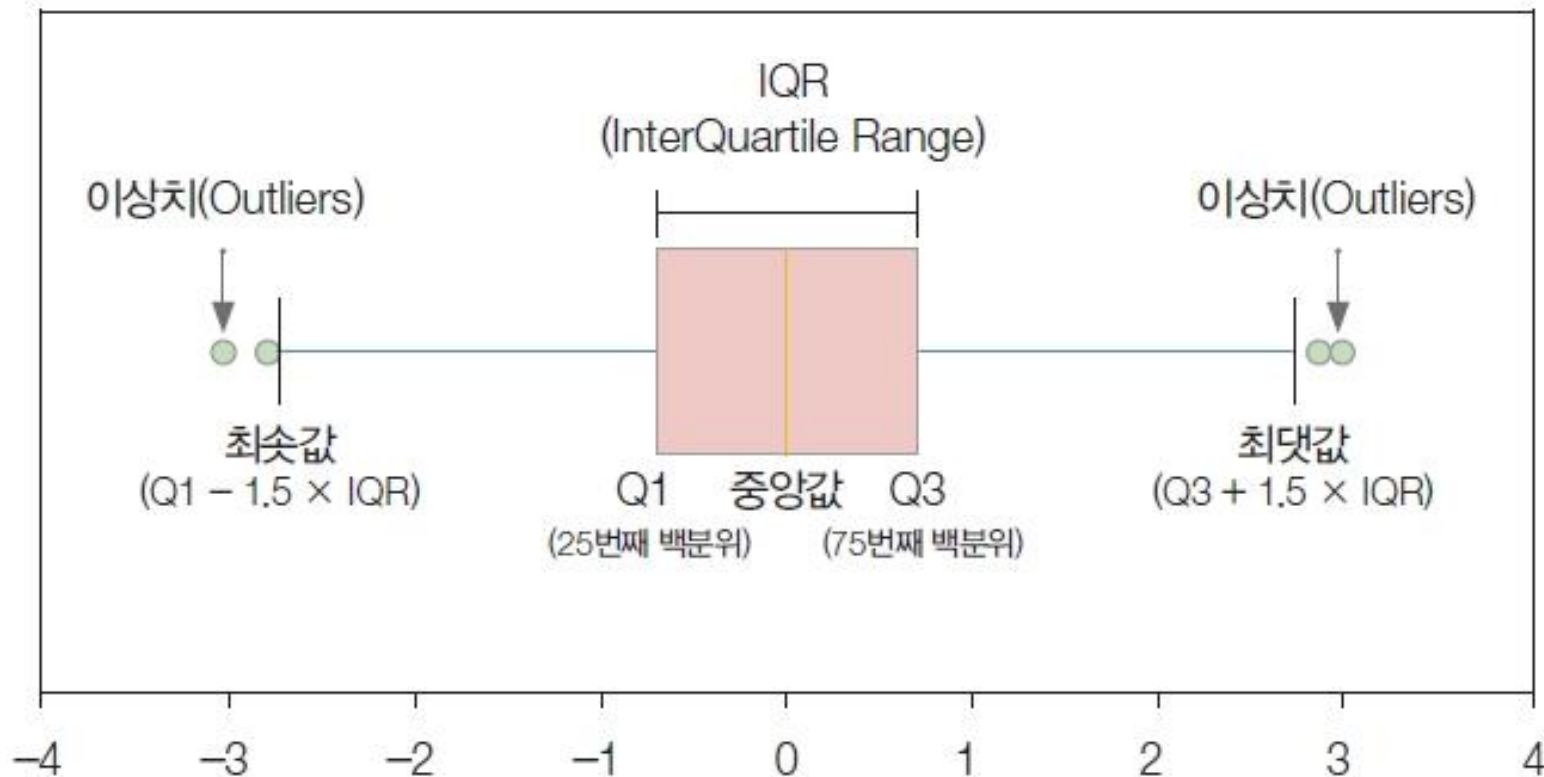


그림 5-3 상자그림의 구성 요소

01 맷플롯립

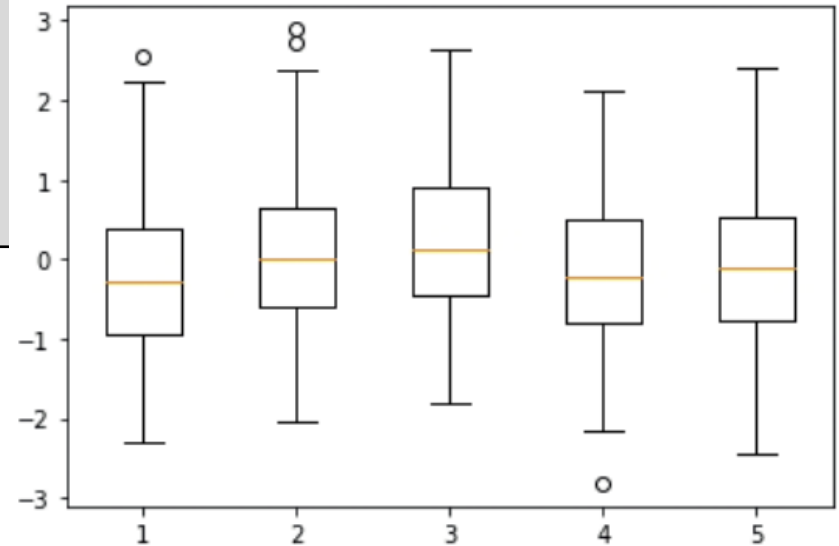
3. 맷플롯립에서 사용하는 그래프

3.5 상자그림

- 데이터를 작은 데이터부터 큰 데이터까지 정렬
- Q1(25%)부터 Q3(75%)까지 박스 형태로 위치시킴
- IQR(InterQuatile Range) : Q1 - Q3
- $Q1 - 1.5 \times IQR$ 을 하단 값으로, $Q3 + 1.5 \times IQR$ 을 상단 값으로
- **이상치(outlier)** : 상단 값과 하단 값을 넘어가는 값들

```
In [19]: data = np.random.randn(100,5)
plt.boxplot(data)
plt.show()
```

Out[19]:



02

시본

02 시본

1. 시본의 기본

- 시본(seaborn) : 맷플롯립을 바탕으로 다양한 함수 사용을 돕는 일종의 래퍼(wrapper) 모듈

```
import seaborn as sns
```

- 맷플롯립과 동일한 결과물이 나오며, 작성 과정이 간단
 - 그림 객체나 축 객체 같은 복잡한 개념이 없음
 - xticks 설정하지 않아도 각 축에 라벨 자동으로 생성
 - 데이터프레임과 x, y에 해당하는 열 이름만 지정하면 됨

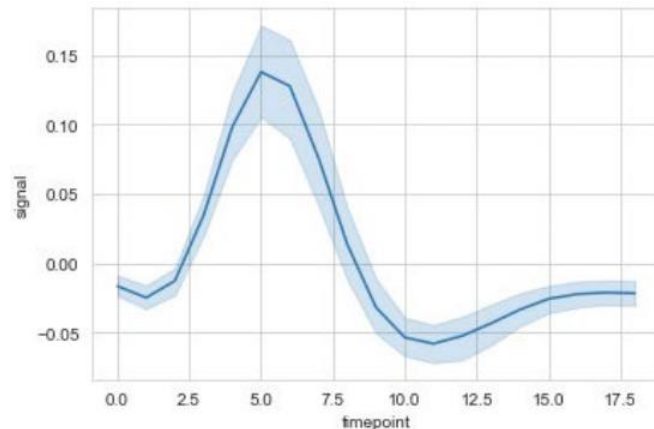
02 시본

1. 시본의 기본

```
In [1]: import numpy as np          # numpy 모듈 호출
import pandas as pd              # pandas 모듈 호출
import matplotlib.pyplot as plt  # matplotlib 모듈 호출
import seaborn as sns           # (1)seaborn 모듈 호출

fmri = sns.load_dataset("fmri")  # (2)시본 자체 fmri 데이터셋 사용
sns.set_style("whitegrid")       # (3) 기본 스타일 적용
sns.lineplot(x="timepoint", y="signal", data=fmri) # (4) 선그래프 작성
```

Out[1]:



02 시본

1. 시본의 기본

- fmri 데이터는 연속형 값 외에도 다양한 범주형 값 가짐
 - 이럴 때 맷플롯립으로 표현하기는 상당히 복잡하고, 시본은 hue 매개변수만 추가하면 그래프 그릴 수 있음

In [2]: `fmri.sample(n=10, random_state=1)`

Out[2]:

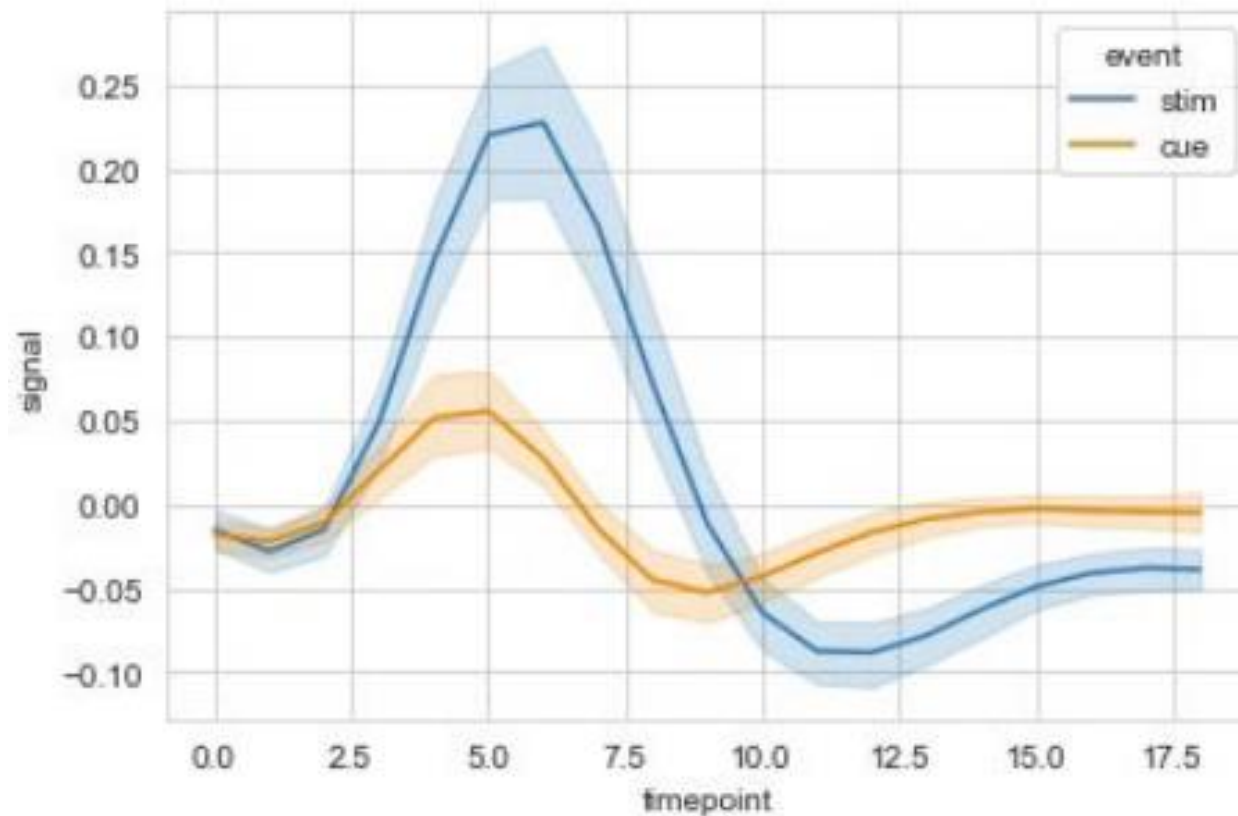
	subject	timepoint	event	region	signal
806	s6	18	cue	parietal	0.019532
691	s5	15	cue	frontal	-0.019507
148	s5	8	stim	parietal	0.006805
676	s13	0	cue	parietal	-0.018394
156	s11	7	stim	parietal	0.254042
27	s1	17	stim	parietal	-0.038021
200	s11	4	stim	parietal	0.087175
262	s3	0	stim	parietal	-0.008576
94	s4	12	stim	parietal	-0.090036
339	s4	5	stim	frontal	0.455575

02 시본

1. 시본의 기본

In [3]: `sns.lineplot(x="timepoint", y="signal", hue="event", data=fmri)`

Out[3]:



02 시본

2. 시본에서 사용하는 그래프

2.1 회귀 그래프

- 회귀 그래프(regression plot) : 회귀식을 적용하여 선형회귀 추세선을 그래프에 함께 작성
 - 선형회귀 추세선 : 데이터를 기반으로 데이터의 x 값 대비 y 값 변화를 예측하는 직선
- 함수 `regplot` 사용

02 시본

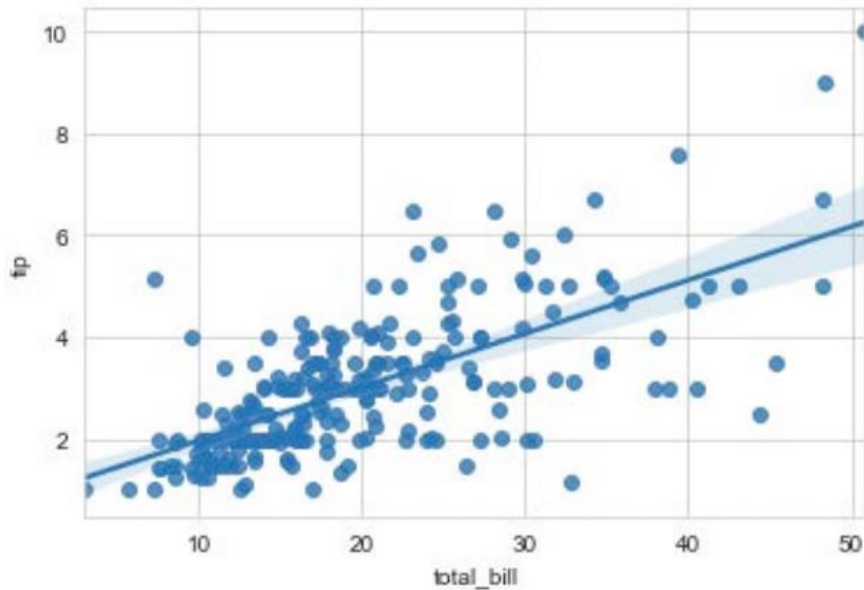
2. 시본에서 사용하는 그래프

2.1 회귀 그래프

- 매개변수 x_{ci} 는 신뢰구간의 비율을 나타냄

```
In [4]: tips = sns.load_dataset("tips")  
sns.regplot(x="total_bill", y="tip", data=tips, x_ci=95)
```

Out[4]:



02 시본

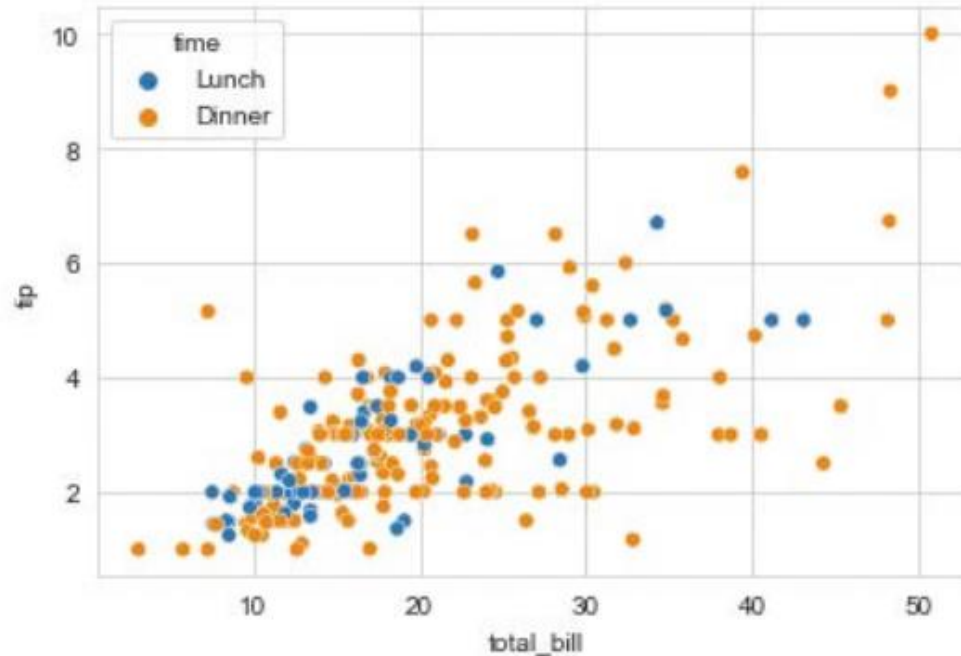
2. 시본에서 사용하는 그래프

2.2 산점도

- 산점도(scatter plot): x, y를 기준으로 데이터의 분포 표현
- 함수 scatterplot 사용

```
In [5]: tips = sns.load_dataset("tips")  
sns.scatterplot(x="total_bill", y="tip", hue="time", data=tips)
```

Out[5]:



02 시본

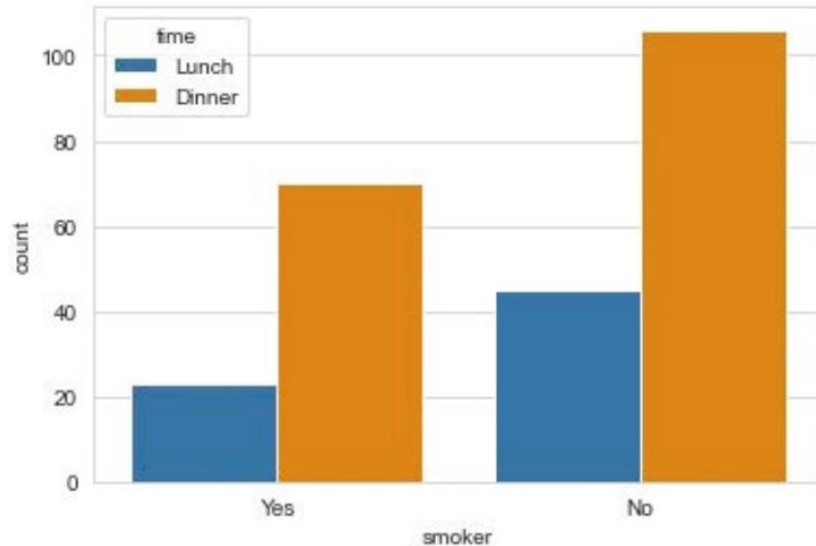
2. 시본에서 사용하는 그래프

2.3 비교 그래프

- 비교 그래프(counter plot) : 범주형 데이터의 항목별 개수

In [6]: `tips = sns.load_dataset("tips")`
`sns.countplot(x="smoker", hue="time", data=tips)`

Out[6]:



	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

그림 5-4 비교 그래프 작성을 위한 데이터

02 시본

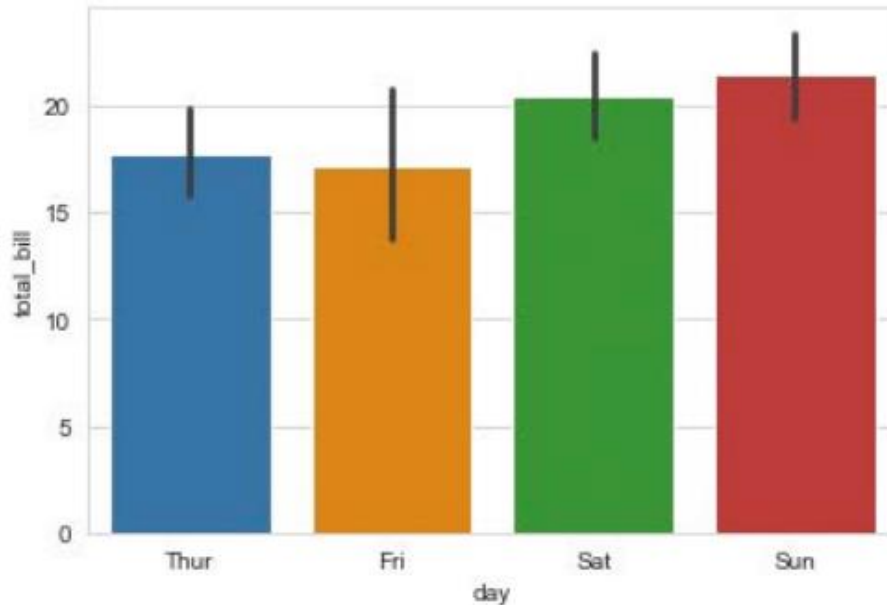
2. 시본에서 사용하는 그래프

2.4 막대그래프

- y 값이 연속형 값일 경우 해당 값들의 평균을 나타냄
- 데이터의 신뢰구간을 검은색 막대로 표현
- 함수 barplot 사용

In [7]: `sns.barplot(x="day", y="total_bill", data=tips)`

Out[7]:



02 시본

3. 사전 정의된 그래프

- 맷플롯립 관점에서 여러 그래프들을 합쳐 정보를 추출
- 특히 범주형 데이터에 유용

3.1 분포를 나타내는 그래프 : 바이올린 플롯과 스웜 플롯

- 바이올린 플롯(violin plot) : 상자그림과 분포도를 한 번에 나타낼 수 있음
 - x축에는 범주형 데이터, y축에는 연속형 데이터

02 시본

3. 사전 정의된 그래프

3.1 분포를 나타내는 그래프 : 바이올린 플롯과 스웜 플롯

In [8]: `sns.violinplot(x="day", y="total_bill", hue="smoker", data=tips, palette="muted")` # deep, muted, pastel, bright, dark, colorblind 팔레트제공[1]

Out[8]:

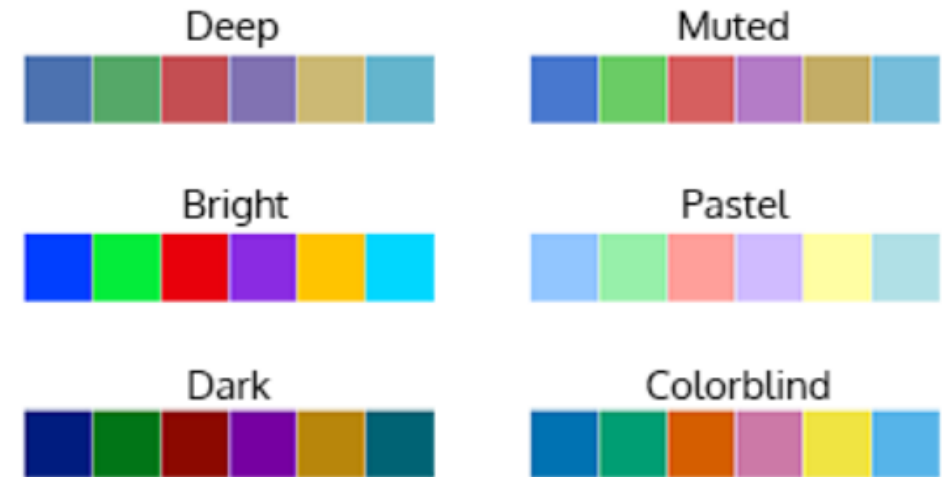
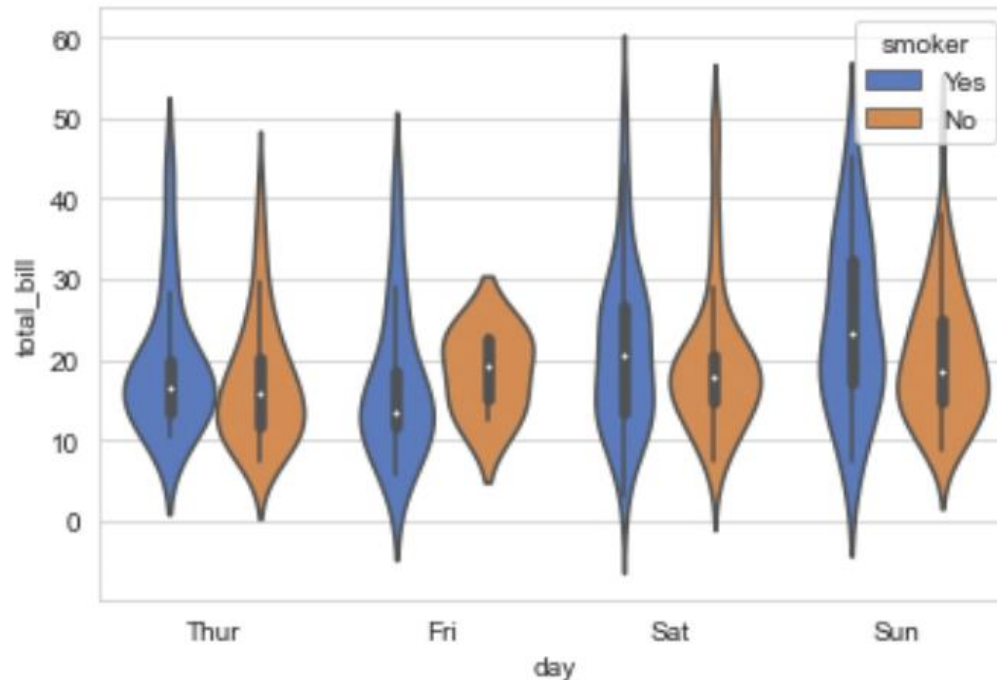


그림. Seaborn에서 제공하는 6개의 기본 팔레트[1]

[1] <https://seong6496.tistory.com/286>

02 시본

3. 사전 정의된 그래프

3.1 분포를 나타내는 그래프 : 바이올린 플롯과 스웜 플롯

- 스웜 플롯(swarm plot) : 바이올린 플롯과 같은 형태에 산점도로 데이터 분포를 나타냄
- 매개변수 hue로 두 개 이상의 범주형 데이터를 점이 겹치지 않게 정리
 - 영역별 데이터 양을 직관적으로 보여줌

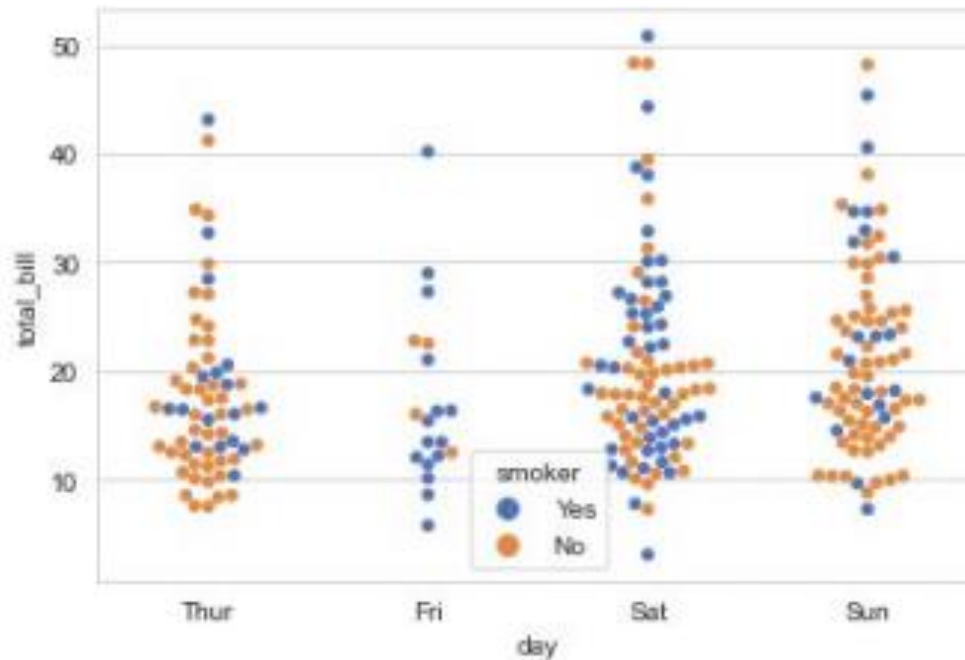
02 시본

3. 사전 정의된 그래프

3.1 분포를 나타내는 그래프 : 바이올린 플롯과 스웜 플롯

In [9]: `sns.swarmplot(x="day", y="total_bill", hue="smoker", data=tips, palette="muted")`

Out[9]:



02 시본

3. 사전 정의된 그래프

3.2 다양한 범주형 데이터를 나타내는 패싯그리드

- 패싯그리드(FacetGrid) : 그래프의 틀만 제공하여 적당한 그래프를 그려주는 클래스

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

그림 5-5 다음 코드에서 다룰 데이터

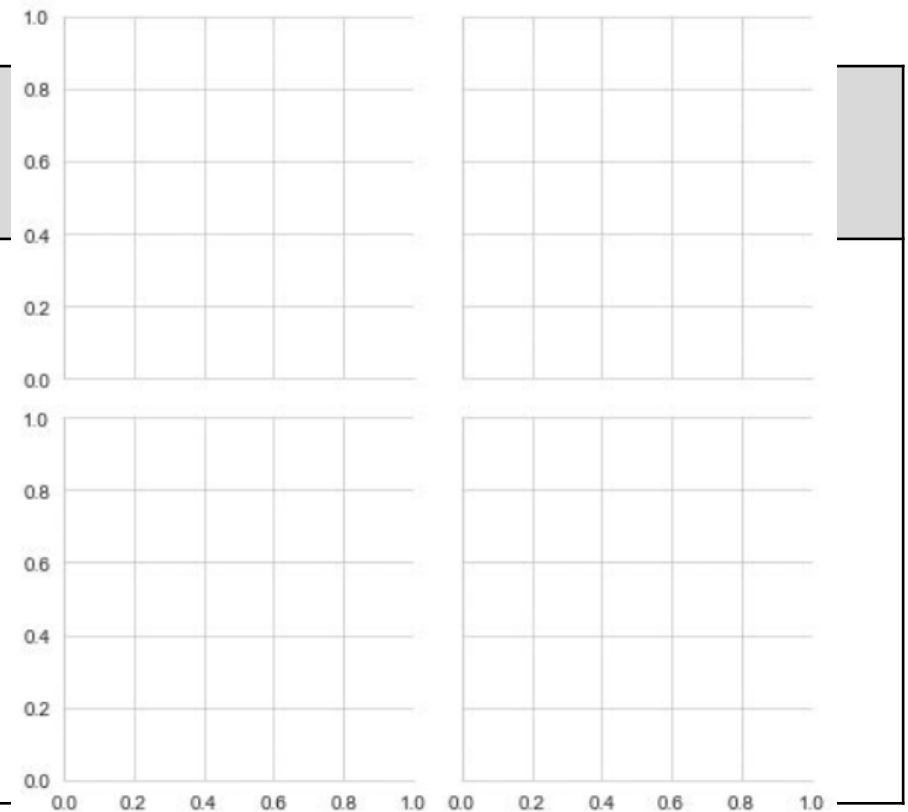
02 시본

3. 사전 정의된 그래프

3.2 다양한 범주형 데이터를 나타내는 패싯그리드

- 기본적인 데이터 표현 틀을 만듦
- 매개변수 col과 row에 범주형 데이터를 넣으면 데이터 종류만큼 'm×n'의 그래프 틀 생성

In [10]:	<code>g = sns.FacetGrid(tips, col="time", row="sex")</code>
Out[10]:	



02 시본

3. 사전 정의된 그래프

3.2 다양한 범주형 데이터를 나타내는 패싯그리드

- 그리드가 생성된 후 맵(map)을 사용하여 그래프 만듦
- 각 FacetGrid에 있는 개별 그래프 영역에 그래프를 집어넣는 구조
- 전체 데이터를 범주형 데이터의 다양한 관점에서 나눠서 볼 수 있음

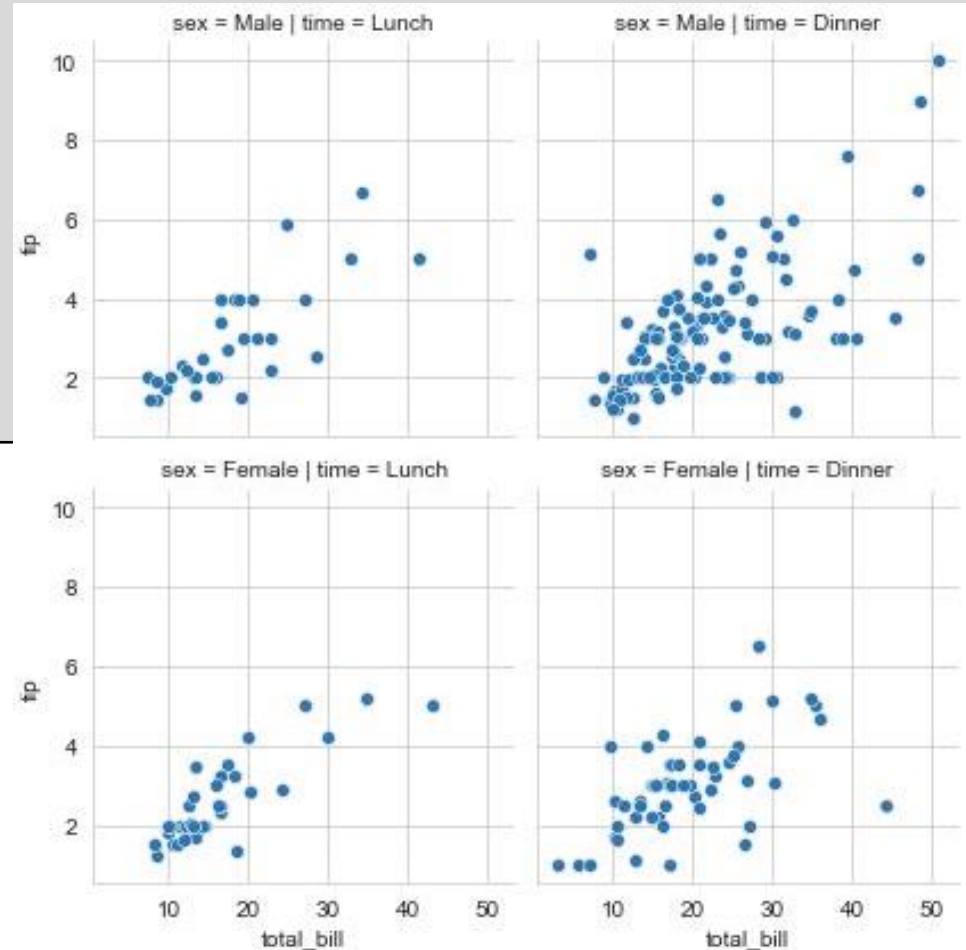
02 시본

3. 사전 정의된 그래프

3.2 다양한 범주형 데이터를 나타내는 패싯그리드

```
In [11]: g = sns.FacetGrid(tips,  
                           col="time",  
                           row="sex")  
  
g.map(sns.scatterplot,  
      "total_bill", "tip")
```

Out[11]:



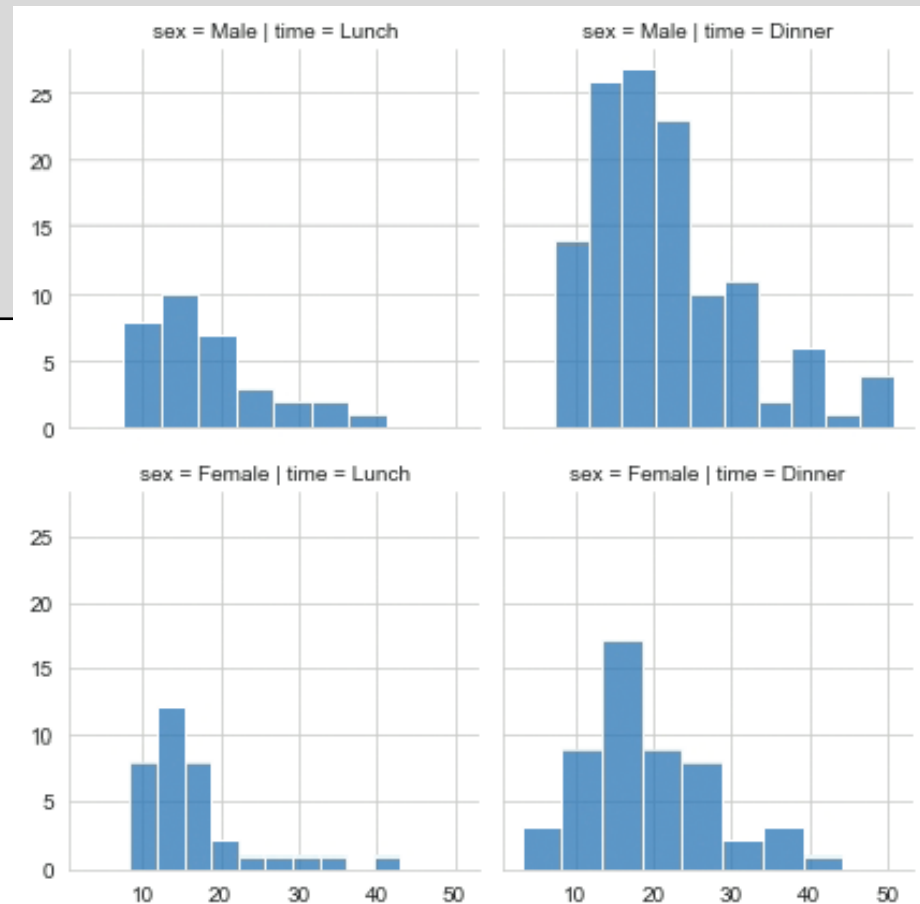
02 시본

3. 사전 정의된 그래프

3.2 다양한 범주형 데이터를 나타내는 패싯그리드

```
In [12]: g = sns.FacetGrid(tips, col="time",  
                             row="sex")  
  
g.map_dataframe(sns.histplot,  
                x="total_bill")
```

Out[12]:



03

플롯리

03 플롯리

1. 플롯리의 특징

- 플롯리(plotly) : 비즈니스 인텔리전스(Business Intelligence) 대시보드로 개발된 도구
 - 비즈니스 인텔리전스 : BI 도구라고도 불림
사내 여러 데이터들을 정리하여 의사결정을 도움
- 애플리케이션으로, 사용자에게 그래프를 제공
 - 맷플롯립이나 시본은 데이터 분석가들이 데이터의 형태나 분포를 살피기 위해 코드로 사용하는 도구
- 인터랙션 그래프를 지원
 - 인터랙션 그래프 : 그래프 생성 이후 사용자가 인터페이스를 통해 조절 가능

03 플롯리

2. 플롯리 사용하기

- 플롯리 설치
 - 터미널에 명령어 입력

```
(ml) C:\...>conda install -c plotly plotly
```

- 문법은 맷플롯립이나 시본과 유사

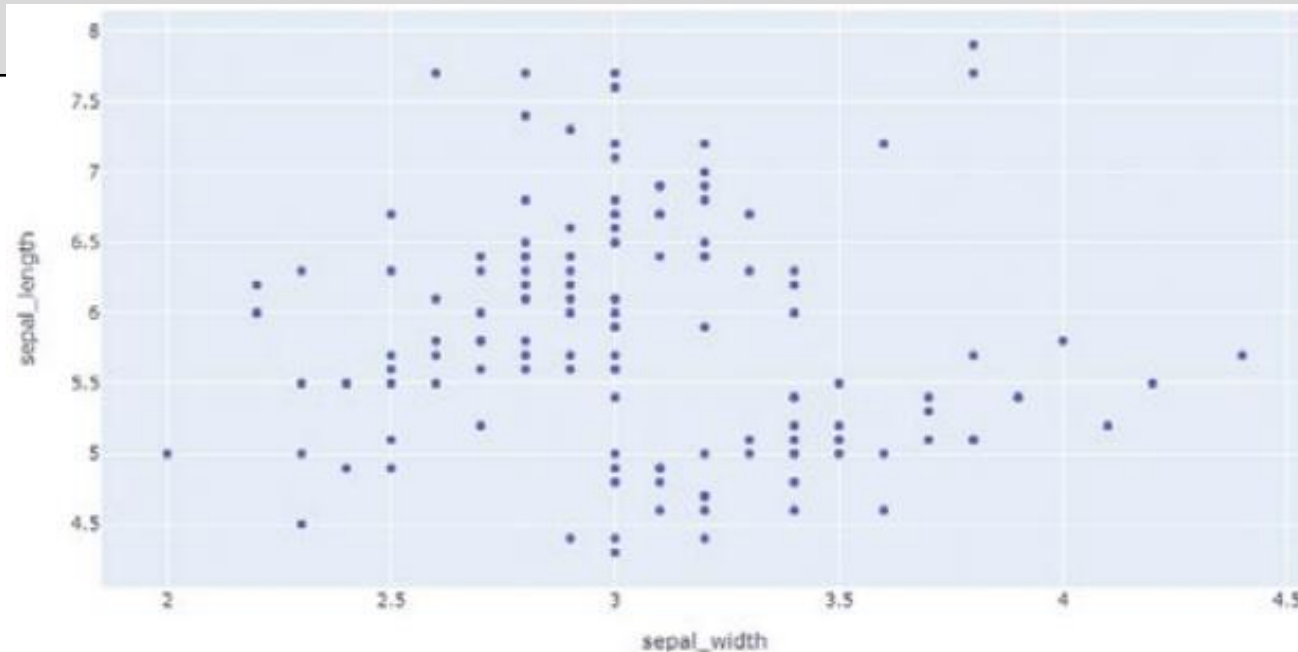
03 플롯리

2. 플롯리 사용하기

- iris 데이터셋을 호출하여 간단한 그래프를 생성
- 래퍼 모듈인 `express`를 호출한 뒤 산점도를 호출

```
In [1]: import plotly.express as px
df = px.data.iris() # iris는 판다스 데이터프레임
fig = px.scatter(df, x="sepal_width", y="sepal_length")
fig.show()
```

Out[1]:



03 플롯리

2. 플롯리 사용하기

- 생성된 그래프에 마우스 커서를 올리면 데이터를 볼 수 있음 (인터랙션 그래프)

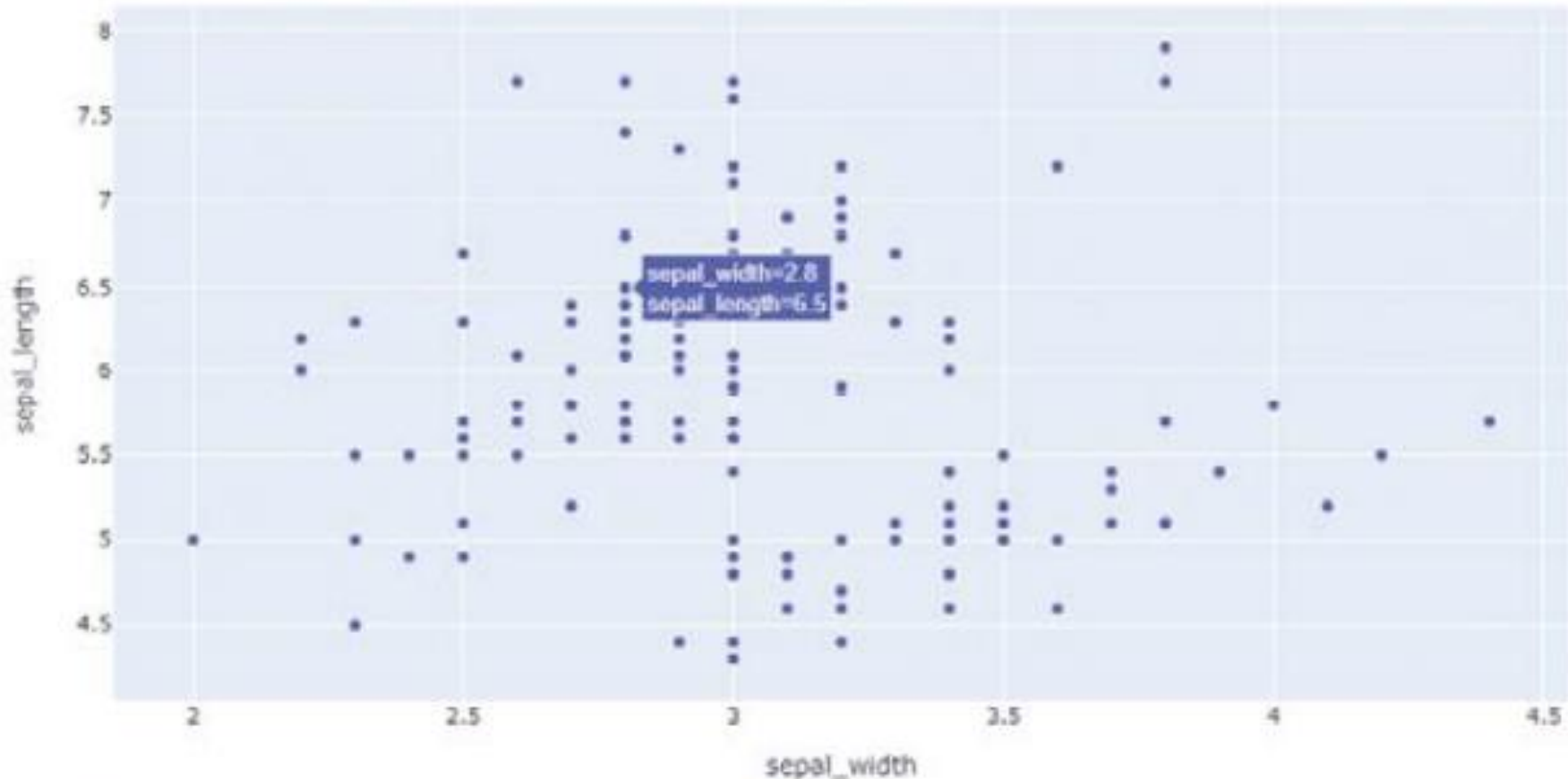


그림 5-6 플롯리로 호출한 그래프의 인터랙션 반응

03 플롯리

2. 플롯리 사용하기

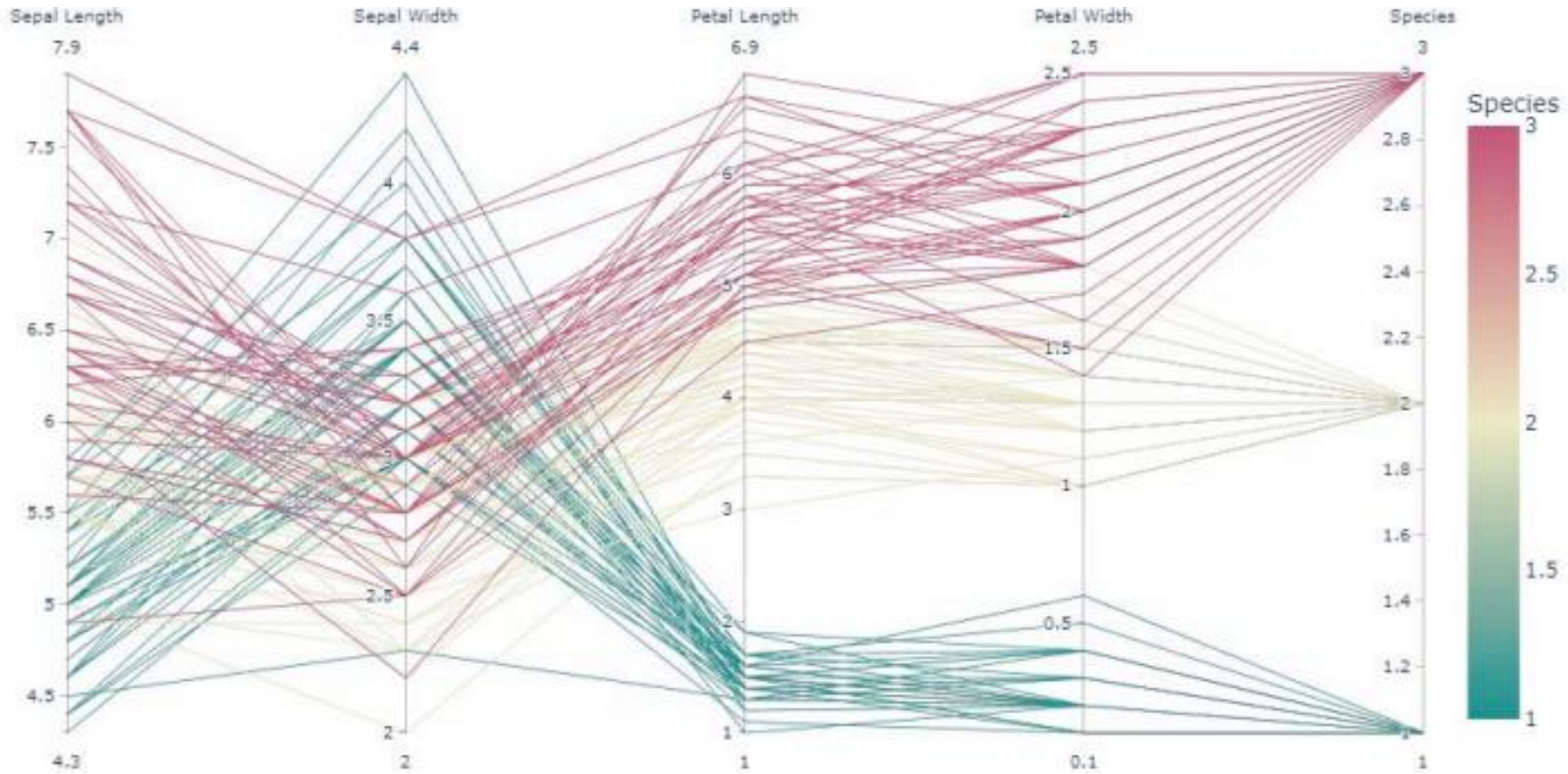
- 좌표 그래프(coordinates plot) : 데이터 간 관계를 표현
 - 시본은 제공하지 않지만 플롯리에서 제공하는 기능

```
In [2]: fig = px.parallel_coordinates(df, color="species_id",  
    labels={"species_id": "Species",  
    "sepal_width": "Sepal Width", "sepal_length": "Sepal Length",  
    "petal_width": "Petal Width", "petal_length": "Petal Length", },  
    color_continuous_scale=px.colors.diverging.Tealrose,  
    color_continuous_midpoint=2)  
fig.show()
```

03 플롯리

2. 플롯리 사용하기

Out [2]:



[TIP] 다양한 그래프들을 맷플롯립처럼 바닥부터 작성할 수도 있다.
대표적으로 `graph_objects`를 사용하면 다양한 그래프를 만들어 낼 수 있다.

Assignment

Assignment

■ 다음 코드를 실행할 경우 결과값은 무엇인가?

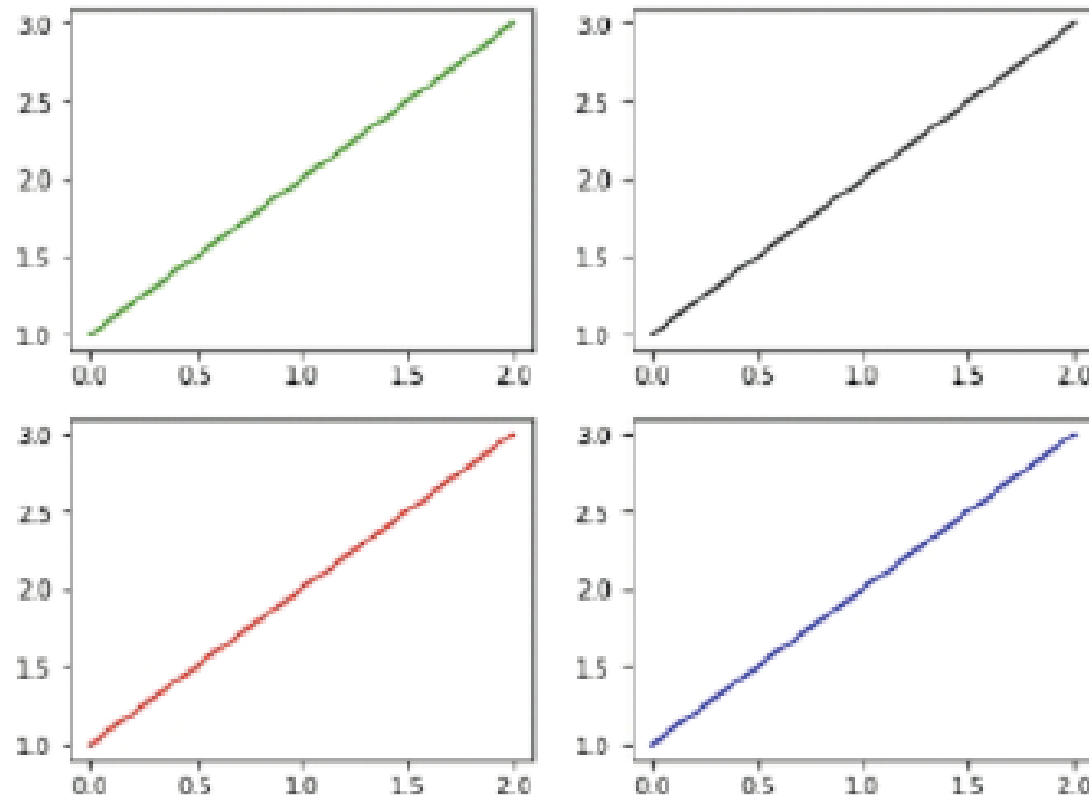
```
import pandas as pd
ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
                    'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
            'Rank': [1, 2, 2, 3, 3, 4, 1, 1, 2, 4, 1, 2],
            'Year': [2014, 2015, 2014, 2015, 2014, 2015, 2016, 2017, 2016, 2014, 2015, 2017],
            'Points': [876, 789, 863, 673, 741, 812, 756, 788, 694, 701, 804, 690]}

df = pd.DataFrame(ipl_data)

df.groupby("Team")["Points"].mean()
```

Assignment

■ 맷플롯립(matplotlib)을 사용하여 다음과 같이 여러 개의 그래프를 한 화면에 작성하고자 한다. 알맞은 코드를 작성하시오.



Thank You !