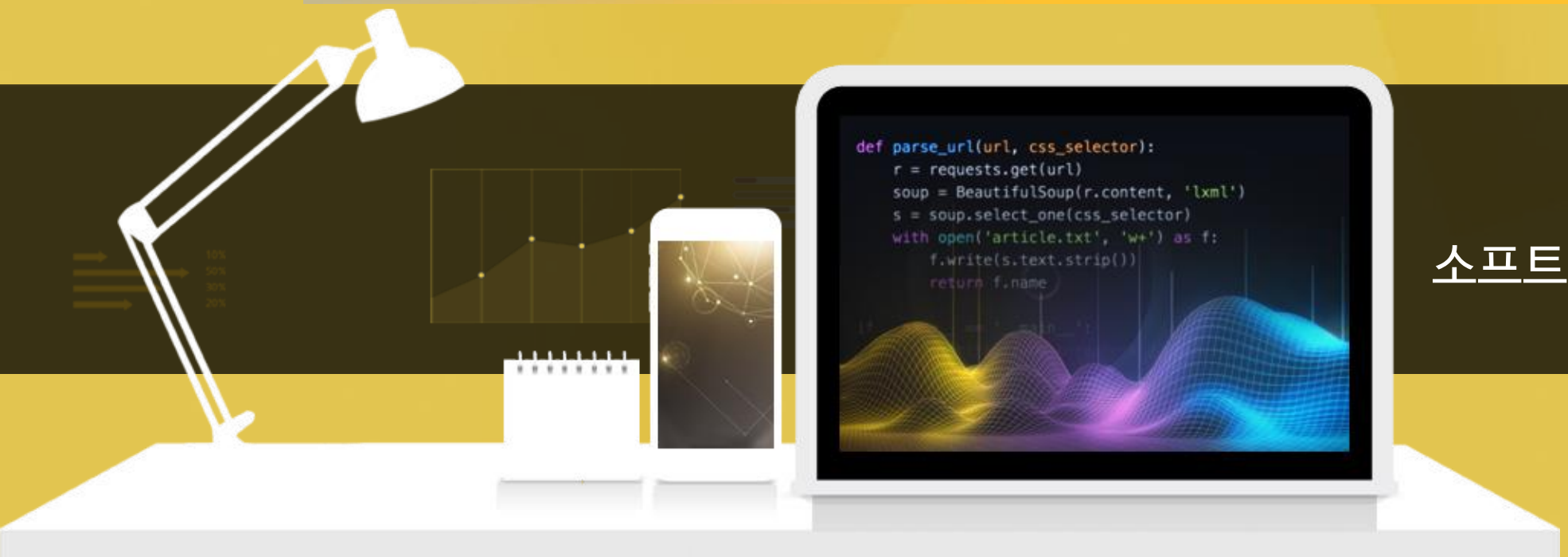
판다스(Pandas)



소프트웨어융합대학원
진혜진

목차

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?

- 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

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

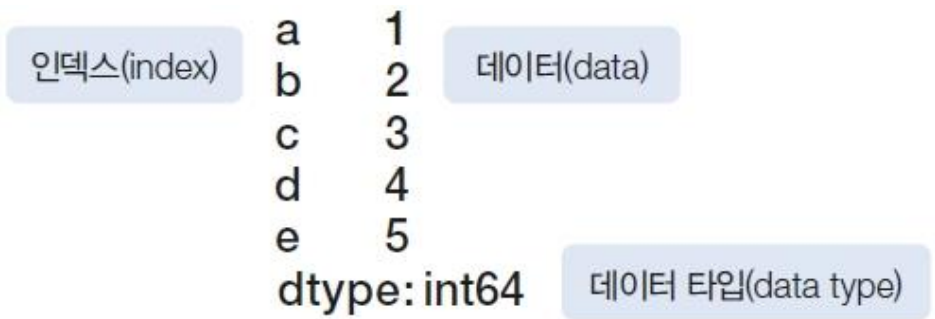


그림 4-2 시리즈 객체 예시

01 판다스란?

- 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?

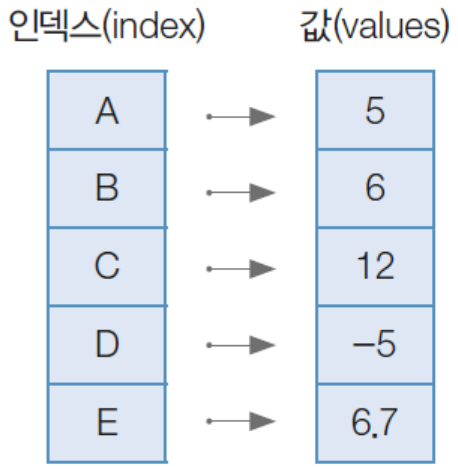


그림 4-3 시리즈 객체와 인덱스

- 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?

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

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

01 판다스란?

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

- 데이터 값을 보기 위해서는 values를 사용
- 실제 생성된 values는 넘파이 배열(numpy.ndarray) 타입

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

- dtype은 데이터의 타입을 나타냄
- 넘파이의 데이터 타입과 동일

01 판다스란?

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

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

01 판다스란?

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

In [7]:	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
Out [7]:	a 1.0 b 2.0 c 3.0 d 4.0 e 5.0 Name: example_data, dtype: float32

01 판다스란?

- 판다스의 모든 객체는 인덱스 값을 기준으로 생성
  - 기존 데이터에 인덱스 값을 추가하면 NaN 값이 출력됨

In [8]:
```
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
```

Out [8]:
```
a    1.0
b    2.0
c    3.0
d    4.0
e    5.0
f    NaN
g    NaN
h    NaN
dtype: float64
```

01 판다스란?

3. 데이터프레임 객체

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

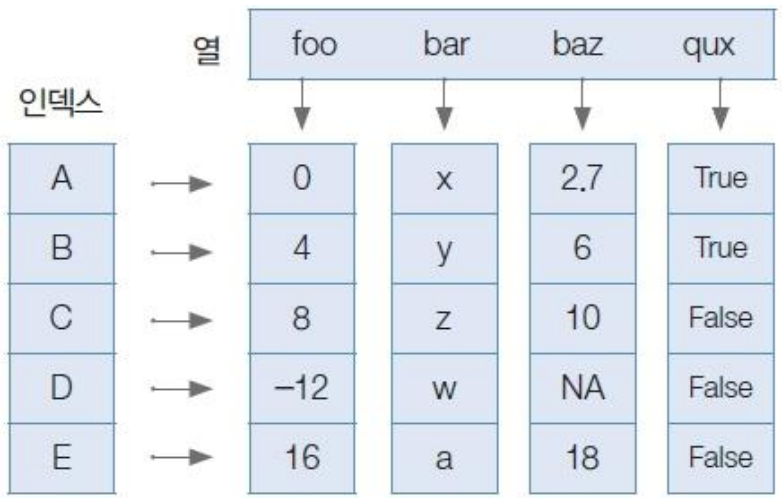


그림 4-4 데이터프레임 객체

01 판다스란?

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?

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

01 판다스란?

- 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?

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?

3.2 데이터프레임의 열 다루기

- 데이터 생성시, 열 이름을 한정하면 해당 열만 추출

In [11]:
```
DataFrame(raw_data, columns = ["age", "city"])
```

Out [11]:

	age	city
0	42	San Francisco
1	52	Baltimore
2	36	Miami
3	24	Douglas
4	73	Boston

01 판다스란?

- 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

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	Mississipi	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	Tenessee	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.

  - read_excel 함수로 엑셀 데이터 호출
    - 파일이 C 드라이브 [source]-[ch04] 폴더에 있다고 가정

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 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

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 Lilianland
state	Texas	NorthCarolina	Iowa
postal-code	28752	38365	76517
Jan	10000	95000	91000
Feb	62000	45000	120000
Mar	35000	35000	35000

02 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]:

	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]:

	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]:

	name	street
0	Kerluke, Koepp and Hilpert	34456 Sean Highway
1	Walter-Trantow	1311 Alvis Tunnel

02 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]:

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

02 데이터 추출

In [9]:
```
df.loc[[211829,320563],["name","street"]]
```

Out [9]:

account	name	street
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway
320563	Walter-Trantow	1311 Alvis Tunnel

- 인덱스 대신 특정 account 번호를 넣어 해당 번호의 값을 나타냄

In [10]:
```
df.loc[205217:,["name","street"]]
```

Out [10]:

account	name	street
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

- 인덱스 번호가 항상 정렬되어 있지 않아 처음 저장된 순서대로 출력

02 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 Lilianland
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

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	Mississipi	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	Tenessee	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

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

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 제
거
```

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 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	Mississipi	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	Tenessee	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

그룹별 집계

03 그룹별 집계

1. 그룹별 집계의 개념

- 그룹별 집계(groupby) : 데이터로부터 동일한 객체를 가진 데이터만 따로 뽑아 기술통계 데이터를 추출
  - 엑셀의 피봇테이블(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

- 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

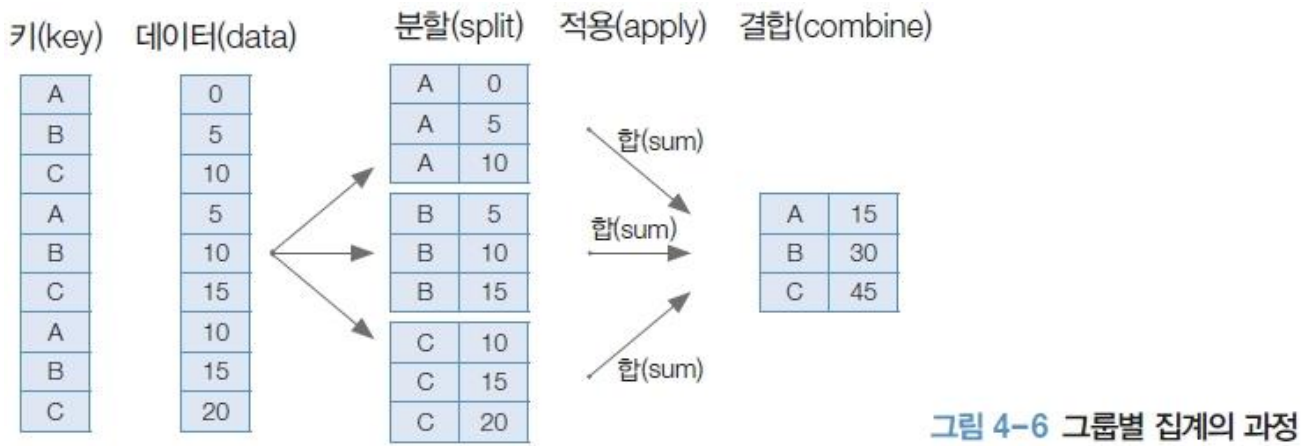


그림 4-6 그룹별 집계의 과정

2. 그룹별 집계 사용하기

2.1 그룹별 집계의 기본형

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

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

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

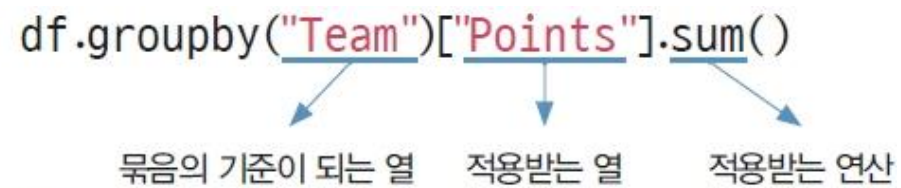


그림 4-7 코드 설명

03 그룹별 집계

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

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

2.3 멀티 인덱스

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

In [4]:
```
multi_groupby = df.groupby(["Team", "Year"])["Points"].sum()
multi_groupby.index
```

Out [4]:
```
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'])
```

03 그룹별 집계

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
```

In [6]:
```
multi_groupby.unstack()
```

Out [6]:

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

In [7]:	multi_groupby.swaplevel().sort_index()
Out [7]:	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

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

03 그룹별 집계

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

- 각 레벨에 별도의 연산함수를 적용할 수 있음

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

3.1 집계

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

03 그룹별 집계

In [11]:
```
grouped.agg(min)
```

Out [11]:

Team	Rank	Year	Points
Devils	2	2014	673
Kings	1	2014	741
Riders	1	2014	690
Royals	1	2014	701
kings	4	2015	812

In [12]:
```
grouped.agg(np.mean)
```

Out [12]:

Team	Rank	Year	Points
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

3.2 변환

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

03 그룹별 집계

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

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

3.3 필터

- 필터(filter) : 특정 조건으로 데이터를 검색
  - 주로 filter 함수 사용

In [15]:
```
df.groupby('Team').filter(
 lambda x: len(x) >= 3)
```

Out [15]:

	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

- x는 분할된 상태에서 각각의 그룹화된 데이터프레임

03 그룹별 집계

In [16]:
```
df.groupby('Team').filter(
 lambda x:
 x["Points"].max() > 800)
```

Out [16]:

	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

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

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

- 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

- 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

1.1 내부 조인

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

04 병합과 연결

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
```

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

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
```

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

- subject_id를 기준으로 내부 조인을 수행
  - 키 값 subject_id 열의 값이 두 테이블 모두 존재해야 병합됨

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

- left, right 매개변수에 각 위치에 해당하는 데이터프레임 객체를 입력
- how에 조인 방법 "inner"를 문자열 타입으로 입력
- on에 병합의 기준이 되는 열 이름을 입력

04 병합과 연결

1.2 왼쪽 조인, 오른쪽 조인

- 왼쪽 조인 : 왼쪽 테이블을 기준으로 데이터를 병합
  - 오른쪽 테이블에 왼쪽 테이블에 있는 키 값이 존재하지 않는다면 NaN으로 출력
- 오른쪽 조인 : 오른쪽 테이블 기준으로 데이터를 병합

04 병합과 연결

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

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

1.3 완전 조인

- 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

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

04 병합과 연결

In [8]:
```
import os
filenames = [os.path.join("c:/source/ch04", filename)
             for filename in os.listdir("c:/source/ch04") if "sales" in
filename]
print(filenames)
```

Out [8]:
```
['c:/source/ch04₩₩sales-feb-2014.xlsx',
'c:/source/ch04₩₩sales-jan-2014.xlsx',
'c:/source/ch04₩₩sales-mar-2014.xlsx']
```

	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

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

04 병합과 연결

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

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

04 병합과 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