

학습 내용

3부. 데이터 분석 라이브러리 활용

11장. N차원 배열 다루기



12장. 데이터프레임과 시리즈

13장. 데이터 시각화

14장. 웹 데이터 수집

- 1절. 판다스 패키지
- 2절. 데이터프레임 만들기
- 3절. 이름 지정하기
- 4절. 부분 데이터 조회
- 5절. 데이터 삭제 및 추가
- 6절. 정렬
- 7절. 기초 통계 분석
- 8절. 데이터 그룹화 및 집계
- 9절. 데이터 구조 변경
- 10절. 데이터프레임 병합과 연결
- 11절. 데이터 구조 변경
- 12절. 데이터프레임에 함수 적용하기
- 13절. 값 일괄 변경하기
- 14절. 시리즈

1.1. 판다스 소개

1절. 판다스 패키지

- 1차원 구조를 갖는 시리즈(Series)와 2차원 구조를 갖는 데이터프레임(DataFrame)을 제공
- 데이터프레임은 테이블 형식이고 이종모음들로 구조화 된 데이터를 말하는데 엑셀의 시트 또는 스프레드시트 형식의 데이터
- 시리즈는 시계열 데이터를 표현하기 위한 데이터 구조.
 - 시리즈는 데이터프레임에서 열(Column) 하나를 의미
- 판다스의 데이터프레임과 시리즈 데이터 구조는 재무, 통계, 사회 과학 등 다양한 분야의 데이터를 처리하기 위해 사용
- 판다스는 데이터의 부분집합 조회, 열 추가 및 제거, 병합, 데이터 구조 변경 등 데이터 전처리를 위한 많은 기능을 제공
- 다음처럼 다양한 종류의 데이터를 처리하기에 적합
 - SQL 테이블 또는 Excel 스프레드시트에서와 같이 열(column) 단위로 데이터의 타입이 지정된 테이블 형식
 - 순서가 있고 정렬되지 않은 시계열 데이터.
 - 행 및 열 레이블이 포함 된 임의의 행렬 데이터(동종 유형 또는 이종 유형)

DataFrame

1절. 판다스 패키지 > 1.1. 판다스 소개

- 2차원(행, 열) 구조(엑셀 시트 구조)

- 행(Row)

- 1개 행은 각각 다른 데이터를 갖는 튜플
- 행의 이름과 인덱스(위치)를 가짐
- 행의 이름은 인덱스와 같을 경우가 많음

- 열(Column)

- 열 내의 모든 데이터는 같은 타입
- 열의 이름과 인덱스(위치)를 가짐

axis=1 →

열(Column)

		0	1	2	3	← 인덱스
	행(Row)	EMPNO	ENAME	SALARY	DEPTNO	← 이름
axis=0 ↓	0	0	7369	SMITH	5500.0	10
	1	1	7499	ALLEN	8900.0	30
	2	2	7521	WARD	NaN	10
	3	3	7566	JONES	9500.0	20

↑ 인덱스 ↑ 이름

1.2. 판다스 장점

1절. 판다스 패키지

- 결측치(Missing Value) 처리 : 부동 소수점 데이터뿐만 아니라 **누락 된 데이터(NaN으로 표시됨)**를 **손쉽게 처리**할 수 있음
- 크기 변경 : 데이터프레임 및 상위 차원 개체에서 **열을 삽입하고 삭제**할 수 있음
- 데이터 정렬 : 개체를 레이블 세트에 **명시적으로 정렬**하거나 사용자가 레이블을 무시하고 시리즈, 데이터프레임 등으로 **자동으로 데이터를 정렬**에 사용할 수 있음
- 데이터 분할 및 병합 : 데이터를 집계 및 변환하기 위해 **데이터 세트에 분할 및 병합 작업을 수행**할 수 있는 강력하고 유연한 그룹 별 기능을 제공합니다.
- 데이터프레임 생성 : 다른 파이썬 및 넘파이 데이터 구조의 비정형 색인 데이터를 **데이터프레임 객체로 쉽게 변환**할 수 있습니다.
- 부분 데이터 섹트 추출 : 지능형 **레이블 기반 슬라이싱, 고급 인덱싱 및 대용량 데이터 세트의 하위 집합**을 사용할 수 있습니다.
- 피벗과 언피벗 : 데이터 세트의 **피벗 및 언피벗 기능을 제공**합니다.
- 레이블링 : 축의 **계층적 레이블링**(다중 레이블을 가질 수 있음)을 제공합니다.
- 파일 입출력 : **CSV** 파일 또는 구분자에 의한 플랫폼 파일, **Excel** 파일, 데이터베이스 및 초고속 **HDF5** 형식의 데이터 저장/로드를 위한 **입출력 도구를 제공**합니다.
- 시계열 관련 기능 : 날짜 범위 생성 및 빈도 변환, 통계, 선형 회귀 등을 사용할 수 있습니다.

2.1. 딕셔너리를 이용해서 데이터프레임 만들기

2절. 데이터프레임 만들기

- 딕셔너리를 이용해 데이터프레임을 만들면 **키가 열 이름**이 됨

```
DataFrame(data=None, index=None, columns=None, dtype=None,
           copy=False)
```

```
1 import pandas as pd
```

```
1 d = {'col1': [1, 2], 'col2': [3, 4]}
2 df = pd.DataFrame(data=d)
3 df
```

	col1	col2
0	1	3
1	2	4

```
1 d = [{'col1': 1, 'col2': 3}, {'col1': 2, 'col2': 4}]
2 df = pd.DataFrame(data=d)
3 df
```

	col1	col2
0	1	3
1	2	4

```
1 d = [{'col1': 1, 'col2': 3}, {'col1': 2, 'col2': 4},
2      {'col1': 2}]
3 df = pd.DataFrame(data=d)
4 df
```

	col1	col2
0	1	3.0
1	2	4.0
2	2	NaN

2.2. 리스트를 이용해 데이터프레임 만들기

2절. 데이터프레임 만들기

- 리스트를 이용하려면 딕셔너리의 값으로 지정하거나

```
1 a = [1,2,3,4,5]
2 b = [6,7,8,9,10]
3 df = pd.DataFrame({"col1": a, "col2": b})
4 df
```

	col1	col2
0	1	6
1	2	7
2	3	8
3	4	9
4	5	10

- np.c_로 열 단위 결합 후 열 이름 지정

```
1 import numpy as np
2 a = [1,2,3,4,5]
3 b = [6,7,8,9,10]
4 df = pd.DataFrame(np.c_[a, b], columns=["col1", "col2"])
5 df
```

	col1	col2
0	1	6
1	2	7
2	3	8
3	4	9
4	5	10

1) CSV 파일 불러오기

2절. 데이터프레임 만들기 > 2.3. read_csv()

● read_csv()

```
pandas.read_csv(filepath_or_buffer, sep=',', delimiter=None,
                  header='infer', ...)
```

```
1 import pandas as pd
2 member_df = pd.read_csv("member_data.csv", sep=",")
```

```
1 member_df
```

	Name	Age	Email	Address
0	홍길동	20	kildong@hong.com	서울시 강동구
1	홍길서	25	kilseo@hong.com	서울시 강서구
2	홍길남	26	south@hong.com	서울시 강남구
3	홍길북	27	book@hong.com	서울시 강북구

샘플 데이터(member_data.csv)

```
Name, Age, Email, Address
홍길동, 20, kildong@hong.com, 서울시 강동구
홍길서, 25, kilseo@hong.com, 서울시 강서구
홍길남, 26, south@hong.com, 서울시 강남구
홍길북, 27, book@hong.com, 서울시 강북구
```

2) 행 제외하기

2절. 데이터프레임 만들기 > 2.3. read_csv()

● skiprows=[제외할 행들]

```
1 member_df = pd.read_csv("member_data.csv", skiprows=[0,2])
```

```
1 member_df
```

	홍길동	20	kildong@hong.com	서울시 강동구
--	-----	----	------------------	---------

0	홍길남	26	south@hong.com	서울시 강남구
---	-----	----	----------------	---------

1	홍길북	27	book@hong.com	서울시 강북구
---	-----	----	---------------	---------

```
1 member_df = pd.read_csv("member_data.csv", skiprows=[1,3])
```

```
2 member_df
```

	Name	Age	Email	Address
--	------	-----	-------	---------

0	홍길서	25	kilseo@hong.com	서울시 강서구
---	-----	----	-----------------	---------

1	홍길북	27	book@hong.com	서울시 강북구
---	-----	----	---------------	---------

3) 주석행 지정하기

2절. 데이터프레임 만들기 > 2.3. read_csv()

● comment='#'

샘플 데이터(member_data.csv)

Name, Age, Email, Address

홍길동, 20, kildong@hong.com, 서울시 강동구

홍길서, 25, kilseo@hong.com, 서울시 강서구

홍길남, 26, south@hong.com, 서울시 강남구

홍길북, 27, book@hong.com, 서울시 강북구

#홍홍동, 30, bad@hong.com, 주소 없음

```
1 member_df = pd.read_csv("member_data.csv")
2 member_df
```

	Name	Age	Email	Address
0	홍길동	20	kildong@hong.com	서울시 강동구

1	홍길서	25	kilseo@hong.com	
---	-----	----	-----------------	--

2	홍길남	26	south@hong.com	
---	-----	----	----------------	--

3	홍길북	27	book@hong.com	
---	-----	----	---------------	--

4	#홍홍동	30	bad@hong.com	
---	------	----	--------------	--

```
1 member_df = pd.read_csv("member_data.csv", comment='#')
2 member_df
```

	Name	Age	Email	Address
0	홍길동	20	kildong@hong.com	서울시 강동구

1	홍길서	25	kilseo@hong.com	서울시 강서구
---	-----	----	-----------------	---------

2	홍길남	26	south@hong.com	서울시 강남구
---	-----	----	----------------	---------

3	홍길북	27	book@hong.com	서울시 강북구
---	-----	----	---------------	---------

2.4. sklearn.datasets 모듈 데이터를 데이터프레임으로 변환하기

2절. 데이터프레임 만들기

- Scikit-learn 패키지에는 **학습**을 위한 많은 **데이터셋**이 **제공**
- Scikit-learn에서 제공하는 데이터셋은 **딕셔너리 형식**

```
1 import numpy as np
2 import pandas as pd
3 from sklearn import datasets
```

```
1 iris = datasets.load_iris()
2 iris
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2],
                [4.6, 3.1, 1.5, 0.2],
                [5. , 3.6, 1.4, 0.2],
```

```
1 x = pd.DataFrame(iris.data, columns=iris.feature_names)
2 y = pd.DataFrame(iris['target_names'][iris['target']],
3                  columns=["species"])
```

```
1 iris_df = pd.concat([x,y], axis=1)
```

```
1 iris_df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

종속변수 지정하기

2절. 데이터프레임 만들기

```
1 y = pd.DataFrame(iris.target, columns=["species"])
```

```
1 iris_df = pd.concat([x,y], axis=1)
```

```
1 iris_df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

데이터프레임의 열 이름 변경하기

2절. 데이터프레임 만들기

- columns 속성 이용 열 이름 지정

```
1 iris_df.columns = ["sepal_length", "sepal_width", "petal_length",  
2                      "petal_width", "species"]
```

```
1 iris_df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

3.1. 열 이름 지정하기

3절. 이름 지정하기

● columns 속성 이용 열 이름 지정

```
1 member_df.columns = ["이름", "나이", "이메일", "주소"]
2 member_df
```

	이름	나이	이메일	주소
0	홍길동	20	kildong@hong.com	서울시 강동구
1	홍길서	25	kilseo@hong.com	서울시 강서구
2	홍길남	26	south@hong.com	서울시 강남구
3	홍길북	27	book@hong.com	서울시 강북구

```
1 member_df.columns
```

```
Index(['이름', '나이', '이메일', '주소'], dtype='object')
```

4.1. 단일 열 조회

4절. 부분 데이터 조회

- 데이터프레임 *.열이름*
- 데이터프레임 *["열이름"]*
 - 열 이름에 . 또는 공백 등이 포함되어 있을 경우 사용

1	member_df.Name
0	홍길동
1	홍길서
2	홍길남
3	홍길북

Name: Name, dtype: object

1	member_df["Name"]
0	홍길동
1	홍길서
2	홍길남
3	홍길북

Name: Name, dtype: object

4.2. loc를 이용한 이름으로 조회

4절. 부분 데이터 조회

- loc[행이름, 열이름]
- 열이름 생략 가능

슬라이싱으로 찾기

```
1 member_df.loc[0:2]
```

열 이름을 생략하면 행 이름으로 찾음

	Name	Age	Email	Address
0	홍길동	20	kildong@hong.com	서울시 강동구
1	홍길서	25	kilseo@hong.com	서울시 강서구
2	홍길남	26	south@hong.com	서울시 강남구

```
1 member_df.loc["Name":"Email"] # nothing
```

열 이름을 생략하면 행 이름으로 찾음

	Name	Age	Email	Address
--	------	-----	-------	---------

```
1 member_df.loc[0:2, "Name":"Email"]
```

	Name	Age	Email
0	홍길동	20	kildong@hong.com
1	홍길서	25	kilseo@hong.com
2	홍길남	26	south@hong.com

```
1 member_df.loc[[0,2], ["Name","Email"]]
```

리스트로 찾기

	Name	Email
0	홍길동	kildong@hong.com
2	홍길남	south@hong.com

4.3. iloc를 이용한 인덱스로 조회

4절. 부분 데이터 조회

● iloc[행인덱스, 열인덱스]

```
1 member_df.iloc[1:3, 1:3]
```

	Age	Email
1	25	kilseo@hong.com
2	26	south@hong.com

```
1 member_df.iloc[0:3, 0:3]
```

	Name	Age	Email
0	홍길동	20	kildong@hong.com
1	홍길서	25	kilseo@hong.com
2	홍길남	26	south@hong.com

```
1 member_df.iloc[:, -1]
```

	Name	Age	Email	Address
3	홍길북	27	book@hong.com	서울시 강북구
2	홍길남	26	south@hong.com	서울시 강남구
1	홍길서	25	kilseo@hong.com	서울시 강서구
0	홍길동	20	kildong@hong.com	서울시 강동구

```
1 member_df.iloc[0::2, [1,3]]
```

	Age	Address
0	20	서울시 강동구
2	26	서울시 강남구

4.4 조건으로 조회하기

4절. 부분 데이터 조회

- 예제에 사용할 데이터
- statsmodels 패키지의 get_rdataset() 함수 이용

```
1 import numpy as np
2 import pandas as pd
```

```
1 import statsmodels.api as sm
2 iris = sm.datasets.get_rdataset("iris", package="datasets")
```

```
1 iris
```

```
<class 'statsmodels.datasets.utils.Dataset'>
```

4.4 조건으로 조회하기

4절. 부분 데이터 조회

- `get_rdataset()` 함수로 불러온 데이터의 `data` 속성은 데이터를 담고 있는 데이터프레임

```
1 iris_df = iris.data
2 iris_df.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

4.4 조건으로 조회하기

4절. 부분 데이터 조회

- `loc[행조건]` : 행 조건에 맞는 모든 열을 반환

```
1 iris_df.loc[iris_df['Species']=='versicolor'].head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor
52	6.9	3.1	4.9	1.5	versicolor
53	5.5	2.3	4.0	1.3	versicolor
54	6.5	2.8	4.6	1.5	versicolor

4.4 조건으로 조회하기

4절. 부분 데이터 조회

- `loc[행조건, 열리스트]` : 행 조건에 맞는 지정한 열을 반환

```
1 iris_df.loc[iris_df['Species']=='versicolor',  
2             ['Sepal.Length', 'Species']].head()
```

	Sepal.Length	Species
50	7.0	versicolor
51	6.4	versicolor
52	6.9	versicolor
53	5.5	versicolor
54	6.5	versicolor

4.4 조건으로 조회하기

4절. 부분 데이터 조회

● loc[중복조건]

```
1 iris_df.loc[(iris_df['Species']=='versicolor') &
2             (iris_df['Sepal.Length'].astype(float) > 6.5)].head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
50	7.0	3.2	4.7	1.4	versicolor
52	6.9	3.1	4.9	1.5	versicolor
58	6.6	2.9	4.6	1.3	versicolor
65	6.7	3.1	4.4	1.4	versicolor
75	6.6	3.0	4.4	1.4	versicolor

5.1. 데이터프레임의 항목 삭제

5절. 데이터 추가 및 삭제

```
DataFrame.drop(labels=None, axis=0, inplace=False)
```

구문에서...

- *labels* : 삭제할 index 또는 컬럼의 이름을 지정합니다.
- *axis* : int 타입 또는 축의 이름입니다. (0 또는 'index') 와 (1 또는 'columns') 중 하나를 갖습니다. 1이면 열을 삭제합니다.
- *inplace* : bool 타입이며, False(기본값)이면 삭제된 결과 데이터프레임을 리턴하며, True 이면 현재 데이터프레임에서 데이터를 삭제하고 None을 반환합니다.

예제에 사용할 데이터

```
1 member_df = pd.read_csv("member_data.csv", comment='#')
2 member_df.columns = ["이름", "나이", "이메일", "주소"]
3 member_df.index = ["동", "서", "남", "북"]
4 member_df
```

1) 단일 행 삭제하기

5절. 데이터 추가 및 삭제 > 5.1. 데이터프레임의 항목 삭제

● axis=0

```
1 member_df = member_df.drop('북') #axis=0(기본값)이면 행에서 찾아 삭제
2 member_df
```

	이름	나이	이메일	주소
동	홍길동	20	kildong@hong.com	서울시 강동구
서	홍길서	25	kilseo@hong.com	서울시 강서구
남	홍길남	26	south@hong.com	서울시 강남구

2) 단일 열 삭제하기

5절. 데이터 추가 및 삭제 > 5.1. 데이터프레임의 항목 삭제

● axis=1

```
1 member_df = member_df.drop('주소', axis=1) #axis=1이면 열에서 찾아 삭제
2 member_df
```

	이름	나이	이메일
동	홍길동	20	kildong@hong.com
서	홍길서	25	kilseo@hong.com
남	홍길남	26	south@hong.com

3) 복수일 열 삭제하기

5절. 데이터 추가 및 삭제 > 5.1. 데이터프레임의 항목 삭제

● labels = [삭제할_열_리스트,]

```
1 member_df.drop(labels=["Email", "Address"], axis=1)
```

	Name	Age
0	홍길동	20
1	홍길서	25
2	홍길남	26
3	홍길북	27

axis=1과 axis='columns'와 동일

```
1 member_df.drop(labels=["Email", "Address"], axis="columns")
```

	Name	Age
0	홍길동	20
1	홍길서	25
2	홍길남	26
3	홍길북	27

4) 열 삭제와 재 할당

5절. 데이터 추가 및 삭제 > 5.1. 데이터프레임의 항목 삭제

● inplace=True

```
1 member_df.drop("Address", axis=1, inplace=True)  
2 member_df
```

	Name	Age	Email
0	홍길동	20	kildong@hong.com
1	홍길서	25	kilseo@hong.com
2	홍길남	26	south@hong.com
3	홍길북	27	book@hong.com

1) 열 추가

5절. 데이터 추가 및 삭제 > 5.2. 데이터프레임의 항목 추가

● 데이터프레임["새로운_열_이름"] = 값

예제에 사용할 데이터

```
1 member_df = pd.read_csv("member_data.csv", comment='#')
2 member_df["BirthYear"] = 2000
3 member_df
```

	Name	Age	Email	Address	BirthYear
0	홍길동	20	kildong@hong.com	서울시 강동구	2000
1	홍길서	25	kilseo@hong.com		
2	홍길남	26	south@hong.com		
3	홍길북	27	book@hong.com		

```
1 member_df = pd.read_csv("member_data.csv", comment='#')
2 member_df["BirthYear"] = [2001, 2002, 2003, None]
3 member_df
```

	Name	Age	Email	Address	BirthYear
0	홍길동	20	kildong@hong.com	서울시 강동구	2001.0
1	홍길서	25	kilseo@hong.com	서울시 강서구	2002.0
2	홍길남	26	south@hong.com	서울시 강남구	2003.0
3	홍길북	27	book@hong.com	서울시 강북구	NaN

2) 시리즈를 이용한 열 추가

5절. 데이터 추가 및 삭제 > 5.2. 데이터프레임의 항목 추가

- 인덱스를 포함하는 **시리즈 객체를 이용해** 추가

```
1 member_df = pd.read_csv("member_data.csv", comment='#')
2 member_df["BirthYear"] = pd.Series([2001, 2002, 2004], index=[0,1,3])
3 member_df
```

	Name	Age	Email	Address	BirthYear
0	홍길동	20	kildong@hong.com	서울시 강동구	2001.0
1	홍길서	25	kilseo@hong.com	서울시 강서구	2002.0
2	홍길남	26	south@hong.com	서울시 강남구	NaN
3	홍길북	27	book@hong.com	서울시 강북구	2004.0

3) 딕셔너리로 행 추가

5절. 데이터 추가 및 삭제 > 5.2. 데이터프레임의 항목 추가

- 데이터프레임 추가가 아닐때 `ignore_index=True` 파라미터 지정

```
1 new_member = {"Name" : "이순신",
2               "Age"  : 30,
3               "Email" : "hong@abc.com",
4               "Address": "서울시 마포구"}
```

```
1 new_df = member_df.append(new_member, ignore_index=True)
2 new_df
```

	Name	Age	Email	Address
0	홍길동	20	kildong@hong.com	서울시 강동구
1	홍길서	25	kilseo@hong.com	서울시 강서구
2	홍길남	26	south@hong.com	서울시 강남구
3	홍길북	27	book@hong.com	서울시 강북구
4	이순신	30	hong@abc.com	서울시 마포구

4) 시리즈를 이용한 리스트 데이터의 행 추가

- 리스트는 시리즈 객체로 만들어 행 단위로 추가

```

1 new_list = ['파이썬', 23, 'python@hong.com', '서울시 마포구']
2 new_series = pd.Series(new_list, index=member_df.columns)
3 new_df = member_df.append(new_series, ignore_index=True)
4 new_df

```

	Name	Age	Email	Address
0	홍길동	20	kildong@hong.com	서울시 강동구
1	홍길서	25	kilseo@hong.com	서울시 강서구
2	홍길남	26	south@hong.com	서울시 강남구
3	홍길북	27	book@hong.com	서울시 강북구
4	파이썬	23	python@hong.com	서울시 마포구

6절. 정렬

6절. 정렬

행 또는 이름으로 정렬

```
DataFrame.sort_index(axis=0, level=None, ascending=True,
                      inplace=False, kind='quicksort',
                      na_position='last', sort_remaining=True,
                      by=None)
```

값으로 정렬

```
DataFrame.sort_values(by, axis=0, ascending=True, inplace=False,
                      kind='quicksort', na_position='last')
```

```
1 member_df = pd.read_csv("member_data.csv", comment='#')
2 member_df.index = ["동", "서", "남", "북"]
3 member_df
```

	Name	Age	Email	Address
동	홍길동	20	kildong@hong.com	서울시 강동구
서	홍길서	25	kilseo@hong.com	서울시 강서구
남	홍길남	26	south@hong.com	서울시 강남구
북	홍길북	27	book@hong.com	서울시 강북구

예제에 사용할 데이터

6.1. 행 이름으로 정렬

6절. 정렬

- `sort_index()` 함수는 데이터프레임의 **행 이름을 이용해서 정렬**

1	<code>member_df.sort_index()</code>
---	-------------------------------------

	Name	Age	Email	Address
남	홍길남	26	south@hong.com	서울시 강남구
동	홍길동	20	kildong@hong.com	서울시 강동구
북	홍길북	27	book@hong.com	서울시 강북구
서	홍길서	25	kilseo@hong.com	서울시 강서구

6.2. 열 이름으로 열 순서 바꾸기

6절. 정렬

- `axis=1`
- 열의 이름순으로 열의 순서를 바꿈

```
1 member_df.sort_index(axis=1)
```

	Address	Age	Email	Name
동	서울시 강동구	20	kildong@hong.com	홍길동
서	서울시 강서구	25	kilseo@hong.com	홍길서
남	서울시 강남구	26	south@hong.com	홍길남
북	서울시 강북구	27	book@hong.com	홍길북

6.3. 값으로 정렬

6절. 정렬

- 데이터프레임의 **값을 기준으로 정렬**하려면 `sort_values()`를 이용

```
1 member_df.sort_values(by=["Email"])
```

	Name	Age	Email	Address
북	홍길북	27	book@hong.com	서울시 강북구
동	홍길동	20	kildong@hong.com	서울시 강동구
서	홍길서	25	kilseo@hong.com	서울시 강서구
남	홍길남	26	south@hong.com	서울시 강남구

6.4. 레벨로 정렬

6절. 정렬

- **level 매개변수** 이용하면 레벨이 지정되어 있을 경우 **레벨로 정렬**

```
1 member_df = pd.read_csv("member_data.csv", comment='#')
```

```
1 member_df.columns = ["기본정보", "기본정보", "추가정보", "추가정보"],
2                       ["이름", "나이", "이메일", "주소"]
3 member_df.columns.names = ["정보구분", "상세정보"]
```

```
1 member_df.index = ["좌우", "좌우", "상하", "상하"],
2                   ["동", "서", "남", "북"]
3 member_df.index.names = ["위치구분", "상세위치"]
```

```
1 member_df
```

정보구분		기본정보		추가정보	
상세정보		이름	나이	이메일	주소
위치구분	상세위치	정렬 전			
좌우	동	홍길동	20	kildong@hong.com	서울시 강동구
	서	홍길서	25	kilseo@hong.com	서울시 강서구
상하	남	홍길남	26	south@hong.com	서울시 강남구
	북	홍길북	27	book@hong.com	서울시 강북구

```
1 member_df.sort_index(level=["위치구분"])
```

정보구분		기본정보		추가정보	
상세정보		이름	나이	이메일	주소
위치구분	상세위치	정렬 후			
상하	남	홍길남	26	south@hong.com	서울시 강남구
	북	홍길북	27	book@hong.com	서울시 강북구
좌우	동	홍길동	20	kildong@hong.com	서울시 강동구
	서	홍길서	25	kilseo@hong.com	서울시 강서구

7절. 기초 통계 분석

7절. 기초 통계 분석

- 판다스에서 제공하는 통계분석은 기본적인 기술통계 및 데이터 요약

함수	설명
count	NA를 제외한 개수
min	최소값
max	최대값
sum	합
cumprod	누적합
mean	평균
median	중앙값
quantile	분위수
corr	상관관계
var	표본분산
std	표본 정규분산

예제에 사용할 데이터

```

1 import statsmodels.api as sm
2 iris = sm.datasets.get_rdataset("iris", package="datasets")
3 iris_df = iris.data
4 iris_df.head()

```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

7.1. 최소값, 최대값, 평균, 중위수

7절. 기초 통계 분석

● min(), max(), mean(), median()

1	iris_df.min()
Sepal.Length	4.3
Sepal.Width	2
Petal.Length	1
Petal.Width	0.1
Species	setosa
dtype:	object

1	iris_df.mean()
Sepal.Length	5.843333
Sepal.Width	3.057333
Petal.Length	3.758000
Petal.Width	1.199333
dtype:	float64

1	iris_df.max()
Sepal.Length	7.9
Sepal.Width	4.4
Petal.Length	6.9
Petal.Width	2.5
Species	virginica
dtype:	object

1	iris_df.median()
Sepal.Length	5.80
Sepal.Width	3.00
Petal.Length	4.35
Petal.Width	1.30
dtype:	float64

7.2. 요약 통계량

7절. 기초 통계 분석

```
DataFrame.describe(percentiles=None, include=None, exclude=None)
```

● 구문에서...

- *percentiles* : 출력에 포함될 백분위 수를 0~1사이의 값으로 지정. 기본 값은 [.25, .5, .75]. 25%, 50%, 75% 위치 데이터를 출력
- *include* : 출력에 포함될 데이터의 유형을 지정합니다. None(기본값) 이면 모든 숫자 타입 열들을 출력에 포함시킵니다. “all”이면 모든 열을 포함합니다. 정수형이면 “int64”, 논리형이면 “bool”, 실수형이면 “float64” 등으로 지정합니다.
- *exclude* : 출력에서 제외할 데이터의 유형을 지정합니다. None(기본값) 이면 아무것도 제외시키지 않습니다.

7.2. 요약 통계량

7절. 기초 통계 분석 > 7.2. 요약 통계량

- 숫자 데이터
 - 결과의 인덱스에는 `count`, `mean`, `std`, `min`, `max` 및 하위 백분위 수, 상위 백분위 수 및 상위 백분율이 포함
 - 기본적으로 하위 백분위 수는 25이고 상위 백분위 수는 75입니다. 50 백분위 수는 중앙값과 같음
- 객체 데이터(예 : 문자열 또는 타임 스탬프)
 - 결과 색인에 `count`, `unique`, `top` 그리고 `freq`가 포함
 - `top`가 가장 일반적인 값.
 - 여러 오브젝트 값이 가장 높은 `count`를 갖는 경우, `count`와 `top` 결과는 가장 높은 `count`를 갖는 오브젝트 값 중에서 임의로 선택.
- DataFrame을 통해 제공되는 혼합 데이터 유형
 - 기본값은 숫자 열의 분석만 반환
 - 데이터프레임이 숫자 열이 없는 개체 및 범주 데이터로만 구성된 경우 기본값은 개체 열과 범주 형 열 모두의 분석을 반환
 - `include='all'` 매개변수가 제공되면 결과에는 각 유형의 속성이 결합됨

1) 기본 요약 통계량

7절. 기초 통계 분석 > 7.2. 요약 통계량

- iris 데이터의 요약 통계량에는 종(Species) 정보는 출력되지 않음
- 기본적으로 숫자 데이터의 요약 통계량이 출력

```
1 iris_df.describe()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
1 iris_df.Species.describe()
```

```
count          150
unique           3
top    virginica
freq           50
Name: Species, dtype: object
```


2) include와 exclude

7절. 기초 통계 분석 > 7.2. 요약 통계량

- **include** 및 **exclude** 매개 변수를 사용하여 DataFrame에서 출력용으로 분석되는 열을 포함 또는 제한

```
1 import pandas as pd
2 df = pd.DataFrame({'a': [1, 2] * 3,
3                     'b': [True, False] * 3,
4                     'c': [2.0, 4.0] * 3})
```

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

	a	c
count	6.000000	6.000000
mean	1.500000	3.000000
std	0.547723	1.095445
min	1.000000	2.000000
25%	1.000000	2.000000
50%	1.500000	3.000000
75%	2.000000	4.000000
max	2.000000	4.000000

```
1 df.describe(include=["int64"])
```

	a
count	6.000000
mean	1.500000
std	0.547723
min	1.000000
25%	1.000000
50%	1.500000
75%	2.000000
max	2.000000

```
1 df.describe(exclude=["bool", "float64"])
```

```
1 df.describe(include='all')
```

	a	b	c
count	6.000000	6	6.000000
unique	NaN	2	NaN
top	NaN	True	NaN
freq	NaN	3	NaN
mean	1.500000	NaN	3.000000
std	0.547723	NaN	1.095445
min	1.000000	NaN	2.000000
25%	1.000000	NaN	2.000000
50%	1.500000	NaN	3.000000
75%	2.000000	NaN	4.000000
max	2.000000	NaN	4.000000

7.3. 분산, 표준편차

7절. 기초 통계 분석

```
DataFrame.var(axis=None, skipna=None, level=None, ddof=1,
               numeric_only=None, **kwargs)
```

```
DataFrame.std(axis=None, skipna=None, level=None, ddof=1,
               numeric_only=None, **kwargs)
```

1	iris_df.var()
---	---------------

```
Sepal.Length    0.685694
Sepal.Width     0.189979
Petal.Length    3.116278
Petal.Width     0.581006
dtype: float64
```

1	iris_df.std()
---	---------------

```
Sepal.Length    0.828066
Sepal.Width     0.435866
Petal.Length    1.765298
Petal.Width     0.762238
dtype: float64
```

7.4. 공분산, 상관계수

7절. 기초 통계 분석

```
DataFrame.cov(min_periods=None)
```

```
DataFrame.corr(method='pearson', min_periods=1)
```

```
1 iris_df.cov()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	0.685694	-0.042434	1.274315	0.516271
Sepal.Width	-0.042434	0.189979	-0.329656	-0.121639
Petal.Length	1.274315	-0.329656	3.116278	1.295609
Petal.Width	0.516271	-0.121639	1.295609	0.581006

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	0.685694	-0.042434	1.274315	0.516271
Sepal.Width	-0.042434	0.189979	-0.329656	-0.121639
Petal.Length	1.274315	-0.329656	3.116278	1.295609
Petal.Width	0.516271	-0.121639	1.295609	0.581006

```
1 iris_df.corr()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	1.000000	-0.117570	0.871754	0.817941
Sepal.Width	-0.117570	1.000000	-0.428440	-0.366126
Petal.Length	0.871754	-0.428440	1.000000	0.962865
Petal.Width	0.817941	-0.366126	0.962865	1.000000

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	1.000000	-0.117570	0.871754	0.817941
Sepal.Width	-0.117570	1.000000	-0.428440	-0.366126
Petal.Length	0.871754	-0.428440	1.000000	0.962865
Petal.Width	0.817941	-0.366126	0.962865	1.000000

8.1. groupby

8절. 데이터 그룹화 및 집계

- `groupby()`는 데이터를 구분 할 수 있는 열(column)의 값들을 이용하여 데이터를 여러 기준에 의해 구분하여 그룹화 한 후 기초 통계 함수 등을 적용 할 수 있도록 함

```
DataFrame.groupby(by=None, axis=0, level=None, as_index=True,
                    sort=True, group_keys=True, squeeze=False,
                    observed=False, **kwargs)
```

```
1 import statsmodels.api as sm
2 iris = sm.datasets.get_rdataset("iris", package="datasets")
3 iris_df = iris.data
4 iris_df.head()
```

예제에 사용할 데이터

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

1) 단일 열로 그룹화

8절. 데이터 그룹화 및 집계 > 8.1. groupby

● groupby 함수의 인수로 그룹화 할 열을 지정

```
1 iris_grouped = iris_df.groupby(iris_df.Species)
2 iris_grouped
```

<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x0000014F76F0BEF0>

```
1 iris_grouped.mean()
```

<https://stackoverflow.com/questions/30490740/move-legend-outside-figure-in-seaborn-tsplot>

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Species				
setosa	5.006	3.428	1.462	0.246
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026

2) 다중 열로 그룹화

8절. 데이터 그룹화 및 집계 > 8.1. groupby

- groupby 함수의 인수로 **그룹화 할 열을 리스트 형식으로 지정**

```
1 iris_grouped2 = iris_df.groupby([iris_df.Species, iris_df["Sepal.Length"]])
2 iris_grouped2
```

<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x0000014F7712C710>

```
1 iris_grouped2.mean().head()
```

		Sepal.Width	Petal.Length	Petal.Width
Species	Sepal.Length			
setosa	4.3	3.000000	1.100000	0.100
	4.4	3.033333	1.333333	0.200
	4.5	2.300000	1.300000	0.300
	4.6	3.325000	1.325000	0.225
	4.7	3.200000	1.450000	0.200

8.2. 그룹간 반복 처리

8절. 데이터 그룹화 및 집계

- 그룹화 된 데이터에서 그룹의 **타입**과 **그룹** 객체를 반복문을 이용해 처리 가능

```
1 for type, group in iris_grouped:
2     print(type, 'Wn', group.head())
```

setosa

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

versicolor

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor
52	6.9	3.1	4.9	1.5	versicolor
53	5.5	2.3	4.0	1.3	versicolor
54	6.5	2.8	4.6	1.5	versicolor

virginica

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
100	6.3	3.3	6.0	2.5	virginica

9.1. 와이드 포맷과 롱 포맷

9절. 데이터 구조 변경

와이드 포맷(wide format)	롱 포맷(long format)	비고
가로로 긴 형식의 데이터 구조	세로로 긴 형식의 데이터 구조	
열 단위 데이터 구조	행 단위 데이터 구조	
피벗테이블(pivot table)	언피벗테이블(unpivot table)	몰튼(molten) 테이블

와이드 포맷(wide format)

```

Ozone Solar.R wind Temp Month Day
1      41      190  7.4   67     5    1
2      36      118  8.0   72     5    2
3      12      149 12.6   74     5    3
4      18      313 11.5   62     5    4
5      NA       NA 14.3   56     5    5
6      28       NA 14.9   66     5    6
7      23      299  8.6   65     5    7
8      19       99 13.8   59     5    8
9       8       19 20.1   61     5    9
10     NA      194  8.6   69     5   10

```

롱 포맷(long format)

```

Month Day variable value
1      5    1   Ozone    41
2      5    2   Ozone    36
3      5    3   Ozone    12
4      5    4   Ozone    18
5      5    6   Ozone    28
6      5    7   Ozone    23
7      5    8   Ozone    19
8      5    9   Ozone     8
9      5   11   Ozone     7
10     5   12   Ozone    16

```


샘플 데이터

9절. 데이터 구조 변경

```

1  import statsmodels.api as sm
2  airquality_data = sm.datasets.get_rdataset("airquality")
3  airquality = airquality_data.data
4  airquality.head()

```

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67	5	1
1	36.0	118.0	8.0	72	5	2
2	12.0	149.0	12.6	74	5	3
3	18.0	313.0	11.5	62	5	4
4	NaN	NaN	14.3	56	5	5

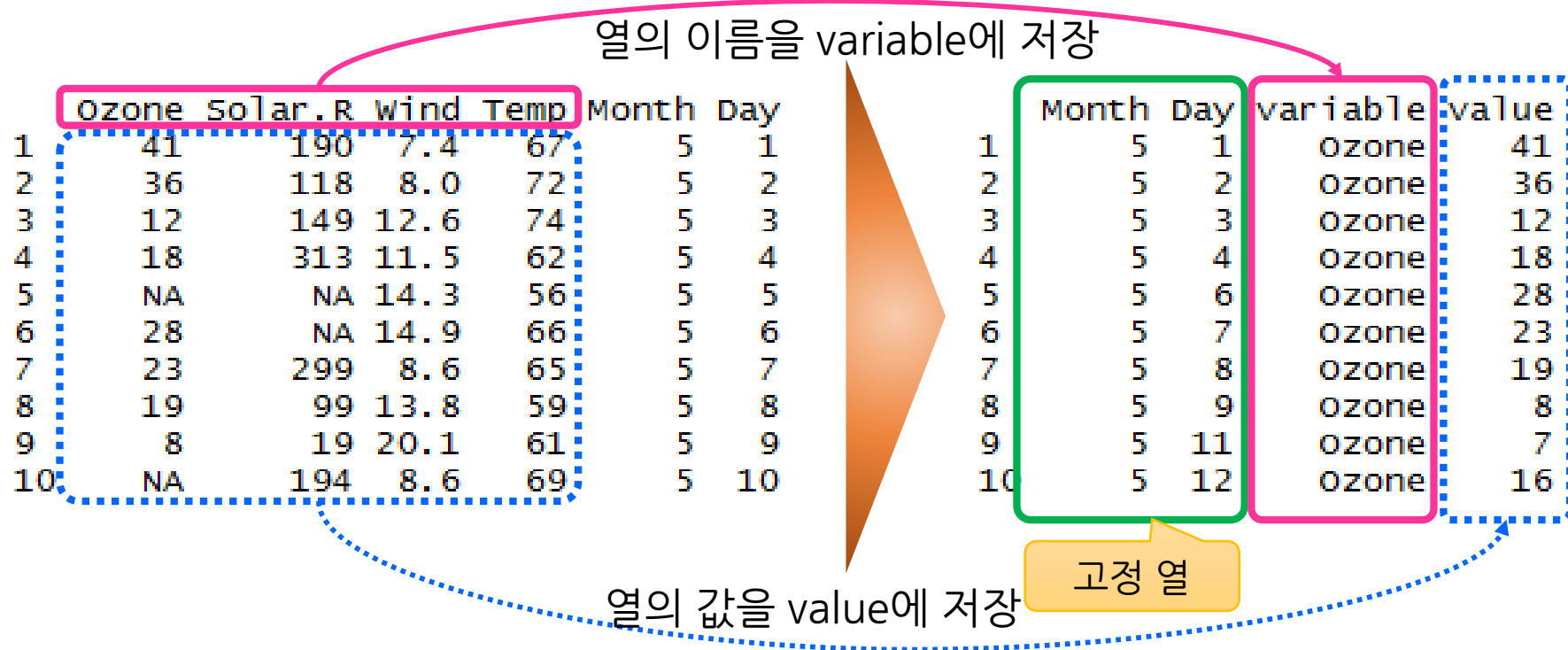
뉴욕이 대기 질을 측정한 데이터 셋

9.2. melt()를 이용한 언피벤팅

https://pandas.pydata.org/docs/user_guide/reshaping.html#reshaping-by-melt

[Tidy Data 란?](<https://vita.had.co.nz/papers/tidy-data.pdf>) 7page

```
pandas.melt(frame, id_vars=None, value_vars=None,
             var_name=None, value_name='value', col_level=None)
```



9.2. melt()를 이용한 언피벤팅

9절. 데이터 구조 변경

```
1 airquality_melted = airquality.melt(id_vars=["Month", "Day"])
```

```
1 airquality_melted.head()
```

	Month	Day	variable	value
0	5	1	Ozone	41.0
1	5	2	Ozone	36.0
2	5	3	Ozone	12.0
3	5	4	Ozone	18.0
4	5	5	Ozone	NaN

9.3. pivot_table()을 이용한 피벗팅

9절. 데이터 구조 변경

```
DataFrame.pivot_table(values=None, index=None,
                        columns=None, aggfunc='mean', fill_value=None,
                        margins=False, dropna=True, margins_name='All')
```

열의 값을 열 이름으로 지정

	Month	Day	variable	value
1	5	1	Ozone	41
2	5	2	Ozone	36
3	5	3	Ozone	12
4	5	4	Ozone	18
5	5	6	Ozone	28
6	5	7	Ozone	23
7	5	8	Ozone	19
8	5	9	Ozone	8
9	5	11	Ozone	7
10	5	12	Ozone	16

	Month	Day	Ozone	solar.R	wind	Temp
1	5	1	41	190	7.4	67
2	5	2	36	118	8.0	72
3	5	3	12	149	12.6	74
4	5	4	18	313	11.5	62
5	5	5	NA	NA	14.3	56
6	5	6	28	NA	14.9	66
7	5	7	23	299	8.6	65
8	5	8	19	99	13.8	59
9	5	9	8	19	20.1	61
10	5	10	NA	194	8.6	69

고정 열

value 열의 값을 저장

9.3. pivot_table()을 이용한 피벗팅

9절. 데이터 구조 변경

```
1 airquality2 = airquality_melted.pivot_table(index=["Month", "Day"],
2 columns=["variable"], values=["value"])
```

다중 인덱스를 갖는 데이터 프레임

```
1 airquality2.head()
```

		value				
		variable	Ozone	Solar.R	Temp	Wind
Month	Day					
5	1		41.0	190.0	67.0	7.4
	2		36.0	118.0	72.0	8.0
	3		12.0	149.0	74.0	12.6
	4		18.0	313.0	62.0	11.5
	5		NaN	NaN	56.0	14.3

```
1 airquality2 = airquality2.reset_index(level=["Month", "Day"], col_level=1)
2 airquality2.columns = airquality2.columns.droplevel(level=0)
```

```
1 airquality2.head()
```

결과 데이터프레임에서 인덱스를 제거하고 출력

variable	Month	Day	Ozone	Solar.R	Temp	Wind
0	5	1	41.0	190.0	67.0	7.4
1	5	2	36.0	118.0	72.0	8.0
2	5	3	12.0	149.0	74.0	12.6
3	5	4	18.0	313.0	62.0	11.5
4	5	5	NaN	NaN	56.0	14.3

예제

- 다음과 같은 데이터프레임데이터를 정의하고 melt와 피버팅을 하시오

	year	mon	latte	americano	mocha
0	2020	1	410	500	350
1	2020	2	401	483	299
2	2020	3	402	484	300
3	2021	1	400	470	301
4	2021	2	404	486	302
5	2021	3	405	488	300

10.1. apply()

10절. 데이터프레임에 함수 적용하기

```
DataFrame.apply(func, axis=0, raw=False,  
                result_type=None, args=(), **kwds)
```

- 구문에서...

- *func*: 각 열 또는 행에 적용할 함수
- *axis*: 함수가 적용될 축.
 - 기본값(0 또는 'index')이면 각 열 별로 함수가 적용
 - 1 또는 'columns'이면 각 행 별로 함수가 적용

10.1. apply()

10절. 데이터프레임에 함수 적용하기

● 샘플 데이터

```
1 import statsmodels.api as sm
2 iris = sm.datasets.get_rdataset("iris", package="datasets")
3 iris_df = iris.data
```

```
1 iris_df.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

10.1. apply()

10절. 데이터프레임에 함수 적용하기

● 항목 별 동작과 축 별 동작

```
1 import numpy as np
2 iris_df.iloc[:, :-1].apply(np.round).head()
```

넘파이의 round 함수
는 항목 별로 동작하는
함수

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0	5.0	4.0	1.0	0.0
1	5.0	3.0	1.0	0.0
2	5.0	3.0	1.0	0.0
3	5.0	3.0	2.0	0.0
4	5.0	4.0	1.0	0.0

```
1 iris_df.iloc[:, :-1].apply(np.sum)
```

넘파이의 sum 함수
는 축 별로 동작하는
함수

```
Sepal.Length    876.5
Sepal.Width      458.6
Petal.Length     563.7
Petal.Width      179.9
dtype: float64
```

10.1. apply()

10절. 데이터프레임에 함수 적용하기

● 시리즈와 람다식의 사용

```
1 iris_avg = iris_df.iloc[:, :-1].apply(np.average)
2 iris_avg
```

```
Sepal.Length    5.843333
Sepal.Width     3.057333
Petal.Length    3.758000
Petal.Width     1.199333
dtype: float64
```

변수 별 평균값을 갖는
시리즈 객체

```
1 iris_df2.apply(lambda x : x-iris_avg, axis=1).head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0	-0.743333	0.442667	-2.358	-0.999333
1	-0.943333	-0.057333	-2.358	-0.999333
2	-1.143333	0.142667	-2.458	-0.999333
3	-1.243333	0.042667	-2.258	-0.999333
4	-0.843333	0.542667	-2.358	-0.999333

람다식과 시리즈의 사
용 예

10.1. apply()

10절. 데이터프레임에 함수 적용하기

● 함수가 리스트 또는 스칼라를 반환할 경우

```
1 iris_df2.apply(lambda x : list(x-iris_avg), axis=1).head()
```

```
0    [-0.7433333333333341, 0.4426666666666663, -2.3...
1    [-0.9433333333333334, -0.05733333333333368, -2...
2    [-1.1433333333333335, 0.1426666666666665, -2.4...
3    [-1.2433333333333334, 0.04266666666666641, -2.2...
4    [-0.8433333333333337, 0.5426666666666664, -2.3...
dtype: object
```

```
1 iris_df2.apply(lambda x : list(x-iris_avg), axis=1,
2               result_type='broadcast').head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0	-0.743333	0.442667	-2.358	-0.999333
1	-0.943333	-0.057333	-2.358	-0.999333
2	-1.143333	0.142667	-2.458	-0.999333
3	-1.243333	0.042667	-2.258	-0.999333
4	-0.843333	0.542667	-2.358	-0.999333

result_type='broadcast' 매개변수를 설정하면 원본 데이터의 열 이름과 구조를 그대로 유지 함

10.2. applymap()

10절. 데이터프레임에 함수 적용하기

- applymap() 함수는 각 요소(element)별로 작동

```
1 iris_df2.applymap(lambda x : x**2).head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0	26.01	12.25	1.96	0.04
1	24.01	9.00	1.96	0.04
2	22.09	10.24	1.69	0.04
3	21.16	9.61	2.25	0.04
4	25.00	12.96	1.96	0.04

1) 사용자 정의 함수 사용

10절. 데이터프레임에 함수 적용하기 > 10.3. map()

- 함수를 적용하면 함수 인자에 시리즈의 항목 하나가 전달

```
1 import pandas as pd
2 x = pd.Series(['Hello', 'Python', 'World'], index=[1,2,3])
3 x
```

```
1 Hello
2 Python
3 World
dtype: object
```

map() 함수는 시리즈
(Series) 타입에서만 사용
할 수 있음

```
1 def my_func(data):
2     return (data, len(str(data)))
```

```
1 x.map(my_func)
```

```
1 (Hello, 5)
2 (Python, 6)
3 (World, 5)
dtype: object
```

2) 딕셔너리 사용

10절. 데이터프레임에 함수 적용하기 > 10.3. map()

- 시리즈에 딕셔너리를 적용하면 딕셔너리의 키별로 시리즈의 값에 적용

```
1 z = {"Hello": 'A', "Python": 'B', "World": 'C'}
```

```
1 x.map(z)
```

```
1    A
```

```
2    B
```

```
3    C
```

```
dtype: object
```

3) 시리즈 사용

10절. 데이터프레임에 함수 적용하기 > 10.3. map()

- 시리즈에 시리즈를 적용하면 원본 시리즈의 값에 적용할 시리즈(arg)의 인덱스 별로 적용

```
1 y = pd.Series(['foo', 'bar', 'baz'],
2               index=['Hello', 'Python', 'World'])
```

```
Hello    foo
Python   bar
World    baz
dtype: object
```

```
1 x.map(y)
```

```
1    foo
2    bar
3    baz
dtype: object
```

4) na_action

10절. 데이터프레임에 함수 적용하기 > 10.3. map()

- 시리즈가 NaN 값을 포함할 경우 na_action 인자의 값에 따라 결과는 달라짐

```
1 s = pd.Series([1, 2, 3, None])
2 s
```

```
0    1.0
1    2.0
2    3.0
3    NaN
dtype: float64
```

```
1 s.map(lambda x: (x, x**2))
```

```
0    (1.0, 1.0)
1    (2.0, 4.0)
2    (3.0, 9.0)
3    (nan, nan)
dtype: object
```

```
1 s.map(lambda x: (x, x**2), na_action='ignore')
```

```
0    (1.0, 1.0)
1    (2.0, 4.0)
2    (3.0, 9.0)
3          NaN
dtype: object
```


(예제)

❖ 1단계 : iris 데이터에서 결측치를 인위적으로 random하게

열 마다 20씩 결측치를 할당한다

❖ 2단계 : 결측치가 있는 iris데이터를 출력한다

❖ 3단계 : 결측치를 열평균으로 대체된 iris데이터를 출력한다.

3) pandas.DataFrame.plot()

2절. Matplotlib > 2.4. 그래프그리기

- 판다스의 데이터프레임 객체를 이용해서 그래프를 그릴 수 있음
- 데이터프레임의 plot() 함수는 데이터프레임의 데이터를 쉽게 시각화

```
DataFrame.plot(x=None, y=None, kind='line', ax=None, subplots=False,
               sharex=None, sharey=False, layout=None, figsize=None,
               use_index=True, title=None, grid=None, legend=True, style=None,
               logx=False, logy=False, loglog=False, xticks=None, yticks=None,
               xlim=None, ylim=None, rot=None, fontsize=None, colormap=None,
               table=False, yerr=None, xerr=None, secondary_y=False,
               sort_columns=False, **kwargs)
```

- 구문에서...
 - kind : 그래프의 종류('line', 'bar', 'barh', 'hist', 'box', 'kde', 'density', 'area', 'pie', 'scatter', 'hexbin')

연습문제 - 실습형

```
import seaborn as sns
iris = sns.load_dataset("iris")
type(iris)
```

```
pandas.core.frame.DataFrame
```

1. iris 데이터에서 처음 다섯개 행만 출력하세요
2. iris 데이터를 독립변수 X와 종속변수 y로 나누세요. 종속변수는 species 열입니다.
3. iris 데이터에서 처음 50개행을 빼내서 temp변수에 저장하세요
4. 3번에서 선택한 데이터프레임의 요약정보를 출력하세요. 모든 열에 대해 요약정보가 출력되어야 합니다.
5. versicolor종의 데이터만 iris_versicolor변수에 저장하세요
6. 2번의 X와 y변수를 합해서 iris_df데이터 프레임으로 만드세요

연습문제 - 실습형

7. iris 데이터의 각 열 평균값을 출력하세요.
8. iris 데이터의 각 열들 사이의 상관계수를 출력하세요.
9. iris 데이터에서 각 요소들과 변수별 평균과의 차이를 출력하세요. 다음 그림은 처음 다섯 개 행의 출력 결과입니다.

	sepal_length	sepal_width	petal_length	petal_width
0	-0.743333	0.442667	-2.358	-0.999333
1	-0.943333	-0.057333	-2.358	-0.999333
2	-1.143333	0.142667	-2.458	-0.999333
3	-1.243333	0.042667	-2.258	-0.999333
4	-0.843333	0.542667	-2.358	-0.999333

10. iris 데이터의 종별 평균을 출력하세요.

연습문제 - 실습형

11. iris 데이터에서 각 요소들과 종별 변수의 평균과의 차이를 출력하세요. 각 종별로 3개씩 출력하세요. 다음 그림은 출력 예입니다.

	petal_length	petal_width	sepal_length	sepal_width	species
0	-0.062	-0.046	0.094	0.072	NaN
1	-0.062	-0.046	-0.106	-0.428	NaN
2	-0.162	-0.046	-0.306	-0.228	NaN
50	0.440	0.074	1.064	0.430	NaN
51	0.240	0.174	0.464	0.430	NaN
52	0.640	0.174	0.964	0.330	NaN
100	0.448	0.474	-0.288	0.326	NaN
101	-0.452	-0.126	-0.788	-0.274	NaN
102	0.348	0.074	0.512	0.026	NaN