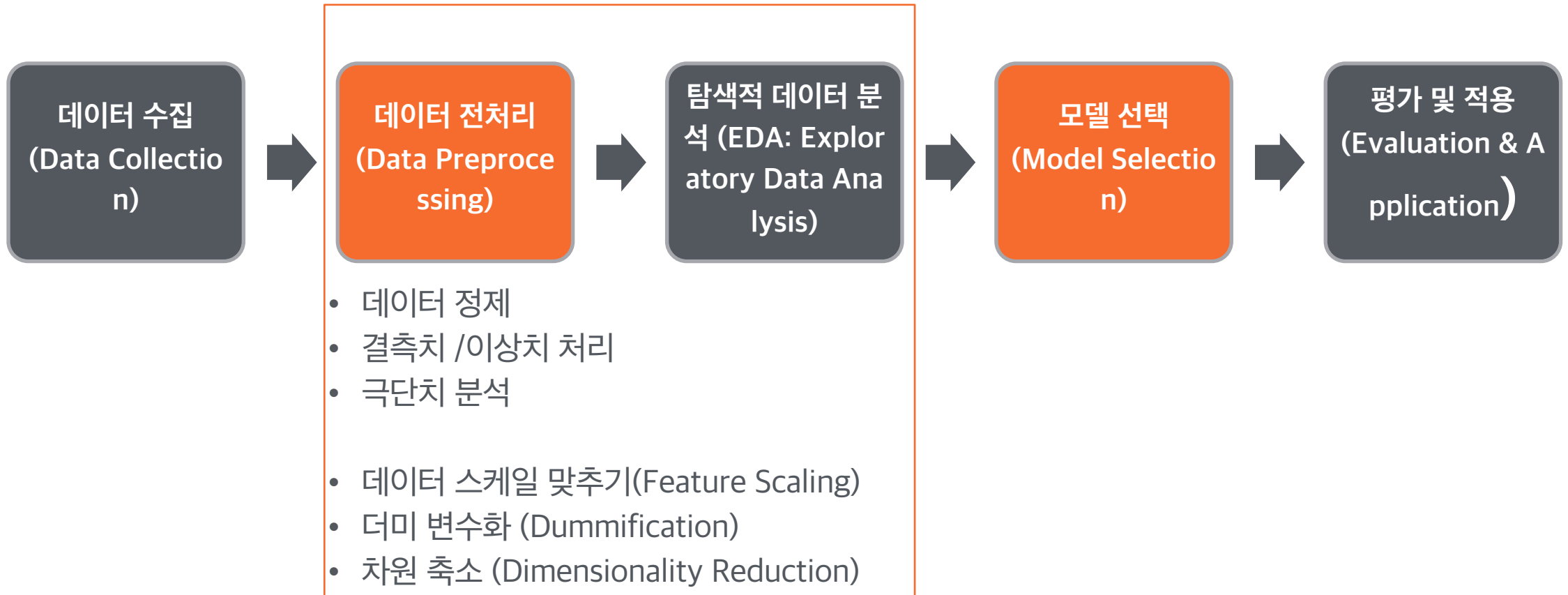


BIGDATA & AI ANALYTICS
EXPERT COMPANY

pandas 데이터프레임

빅데이터 분석 절차



Python 데이터 분석 기본 패키지

Python package

1. NumPy



2 pandas



3 matplotlib



4 Seaborn

BIGDATA & AI ANALYTICS
EXPERT COMPANY

1. pandas 개요

Pandas 개요

1.1 pandas 개요

- 금융회사에 다니고 있던 Wes Mckinney가 처음에 금융 데이터 분석을 위해 2008년 설계
- Pandas: 계량 경제학 용어인 panel data와 analysis의 합성어
- 구조화된 데이터를 빠르고 쉬우면서 다양한 형식으로 가공할 수 있는 풍부한 자료 구조와 함수를 제공
- Pandas의 기본 구조는 numpy

1.2 pandas 특징

- 빅데이터 분석에 최적화 된 필수 패키지
- 데이터는 시계열(series)이나 표(table)의 형태
- 표 데이터를 다루기 위한 시리즈(series) 클래스 변환
- 데이터프레임(dataframe) 클래스 변환

Pandas 개요

1.3 pandas 패키지 설치

cmd, 터미널에서 설치 시

```
pip install pandas
```

Jupyter Notebook에서 설치시

```
!pip install pandas
```

1.4 pandas 패키지 import

데이터프레임을 사용하기 위해 pandas 패키지를 임포트 해야한다.
Pandas는 pd라는 축약어 사용이 관례이다.

```
import pandas as pd
```

BIGDATA & AI ANALYTICS
EXPERT COMPANY

2. pandas 시리즈

시리즈 정의

시리즈 클래스 (series class)

시리즈 클래스는 NumPy에서 제공하는 1차원 배열과 비슷하지만 각 데이터의 의미를 표시하는 인덱스(index)를 붙일 수 있다. 데이터 자체는 값(value)라고 한다.

시리즈 = 값(value) + 인덱스(index)

Series			Series			DataFrame	
	apples			oranges			
0	3	+	0	0	=	0	3
1	2		1	3		1	2
2	0		2	7		2	0
3	1		3	2		3	1
							oranges
							0
							3
							7
							2

시리즈 개요

시리즈 (series) 개요

데이터를 리스트나 1차원 배열 형식으로 series 클래스 생성자에 넣어주면 시리즈 클래스 객체를 만들 수 있다.

```
# Series 정의하기
```

```
obj = pd.Series([4, 5, -2, 8])  
obj
```

```
> 0 4  
1 5  
2 -2  
3 8  
dtype: int64
```

시리즈 확인

시리즈 (series) 확인하기

```
# Series의 값만 확인하기  
obj.values
```

```
> array([ 4, 5, -2, 8])
```

```
# Series의 인덱스만 확인하기  
obj.index
```

```
> RangeIndex(start=0, stop=4, step=1)
```

```
# Series의 자료형 확인하기  
obj.dtypes
```

```
> dtype('int64')
```

시리즈 인덱스

시리즈 (series) 인덱스

인덱스의 길이는 데이터의 길이와 같아야 한다. 인덱스의 값을 인덱스 라벨(label)이라고도 한다. 인덱스 라벨은 문자열 뿐 아니라 날짜, 시간, 정수 등도 가능하다.

인덱스를 바꿀 수 있다.

```
obj2 = pd.Series([4, 5, -2, 8], index=["a", "b", "c", "d"])
obj2
```

```
> a 4
b 5
c -2
d 8
dtype: int64
```

DATAKUBWA

시리즈와 딕셔너리 자료형

시리즈와 딕셔너리 자료형

- python의 dictionary 자료형을 series data로 만들 수 있다.
- dictionary의 key가 series의 index가 된다

```
data = {"Kim": 35000, "Park": 67000, "Joon": 12000, "Choi": 4000}
obj3 = pd.Series(data)
obj3
```

```
> Kim 35000
Park 67000
Joon 12000
Choi 4000
dtype: int64
```

시리즈와 딕셔너리 자료형

```
# 시리즈 이름 지정 및 index name 지정
obj3.name = "Salary"
obj3.index.name = "Names"
obj3
```

```
> Names
Kim 35000
Park 67000
Joon 12000
Choi 4000
Name: Salary, dtype: int64
```

```
# index 변경
obj3.index = ["A", "B", "C", "D"]
obj3
```

```
> A 35000
B 67000
C 12000
D 4000
Name: Salary, dtype: int64
```

DATAKUBWA

시리즈 연산

시리즈 연산

Numpy 배열처럼 시리즈도 벡터화 연산 가능
시리즈의 값에만 적용되며 인덱스 값은 변하지 않는다.

```
# 스칼라 연산 곱하기
```

```
obj * 10
```

```
> 0 40
```

```
1 50
```

```
2 -20
```

```
3 80
```

```
dtype: int64
```

DATAKUBWA

시리즈 연산

```
# 인덱싱 끼리 연산하기
obj * obj
```

```
> 0 16
1 25
2 4
3 64
dtype: int64
```

```
# values 값끼리 연산, 출력은 array
obj.values + obj.values
```

```
> array([ 8, 10, -4, 16])
```

시리즈 인덱싱

시리즈 인덱싱

- 시리즈는 numpy 배열의 인덱스 방법처럼 사용 외에 인덱스 라벨을 이용한 인덱싱
- 시리즈는 numpy 배열의 인덱스 방법처럼 사용 외에 인덱스 라벨을 이용한 인덱싱
- 배열 인덱싱은 자료의 순서를 바꾸거나 특정한 자료만 선택 가능
- 라벨 값이 영문 문자열인 경우에는 마치 속성인것처럼 점(.)을 이용하여 접근 값이 결과에 포함

시리즈 슬라이싱

- 배열 인덱싱이나 인덱스 라벨을 이용한 슬라이싱(slicing)도 가능
- 문자열 라벨을 이용한 슬라이싱은 콜론(:) 기호 뒤에 오는 인덱스에 해당하는 값이 결과에 포함

시리즈 데이터 갱신, 추가, 삭제

2.10 시리즈의 데이터 갱신, 추가, 삭제

- 인덱싱을 이용하여 딕셔너리처럼 데이터를 갱신(update)하거나 추가(add)
- 데이터 삭제 시 딕셔너리처럼 **del** 명령 사용

BIGDATA & AI ANALYTICS
EXPERT COMPANY

3. pandas 데이터프레임

데이터프레임 개요

데이터프레임(DataFrame) 개요

- 시리즈가 1차원 벡터 데이터에 행 방향 인덱스(row index)이라면,
- 데이터프레임(data-frame) 클래스는 2차원 행렬 데이터에 합친 것으로
- 행 인덱스(row index)와 열 인덱스(column index)를 지정

Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

데이터프레임 특성, 생성

데이터프레임 특성

- 데이터프레임은 공통 인덱스를 가지는 열 시리즈(column series)를 딕셔너리로 묶어놓은 것
- 데이터프레임은 numpy의 모든 2차원 배열 속성이나 메서드를 지원

데이터프레임 생성

1. 우선 하나의 열이 되는 데이터를 리스트나 일차원 배열을 준비
2. 각 열에 대한 이름(label)의 키(key)를 갖는 딕셔너리를 생성
3. pandas의 DataFrame 클래스로 생성
4. 열방향 인덱스는 columns 인수로, 행방향 인덱스는 index 인수로 지정

DATAKUBWA

데이터프레임 생성

```
# Data Frame은 python의 dictionary 또는 numpy의 array로 정의
data = {'name': ["Choi ", "Choi", "Choi", "Kim", "Park"],
        'year': [2013, 2014, 2015, 2016, 2017],
        'points': [1.5, 1.7, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data)
df
```

	name	year	points
0	Choi	2013	1.5
1	Choi	2014	1.7
2	Choi	2015	3.6
3	Kim	2016	2.4
4	Park	2017	2.9

데이터프레임 확인

행 방향의 index

df.index

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

열 방향의 index

df.columns

> Index(['name', 'year', 'points'], dtype='object')

값 얻기

df.values

> array([['Choi', 2013, 1.5],
['Choi', 2014, 1.7],
['Choi', 2015, 3.6],
['Kim', 2016, 2.4],
['Park', 2015, 2.9]], dtype=object)

데이터프레임 열(Columns)

데이터프레임 열 갱신 추가

- 데이터프레임은 열 시리즈의 딕셔너리로 볼 수 있으므로 열 단위로 데이터를 갱신하거나 추가, 삭제
- data에 포함되어 있지 않은 값은 nan(not a number)으로 나타내는 null과 같은 개념

DataFrame을 만들면서 columns와 index를 설정

```
df = pd.DataFrame(data, columns=["year", "name", "points", "penalty"],
                  index=["one", "two", "three", "four", "five"])
```

df

	year	name	points	penalty
one	2013	Choi	1.5	NAN
two	2014	Choi	1.7	NAN
three	2015	Choi	3.6	NAN
four	2016	Kim	2.4	NAN
five	2017	Park	2.9	NAN

DATAKUBWA

데이터프레임 열(Columns)

```
df[["year", "points"]]
```

	year	points
one	2013	1.5
two	2014	1.7
three	2015	3.6
four	2016	2.4
five	2017	2.9

특정 열에 선택하고, 값을 대입

```
df["penalty"] = 0.5
```

```
df
```

	year	names	points	penalty
one	2013	Choi	1.5	0.5
two	2014	Choi	1.7	0.5
three	2015	Choi	3.6	0.5
four	2016	Kim	2.4	0.5
five	2017	Park	2.9	0.5

또는 # python의 List / numpy의 array

```
df["penalty"] = [0.1, 0.2, 0.3, 0.4, 0.5]
```

```
df
```

	year	names	points	penalty
one	2013	Choi	1.5	0.1
two	2014	Choi	1.7	0.2
three	2015	Choi	3.6	0.3
four	2016	Kim	2.4	0.4
five	2017	Park	2.9	0.5

DATAKUBWA

데이터프레임 열(Columns)

새로운 열을 추가하기

```
df["zeros"] = np.arange(5)
df
```

	year	names	points	penalty	zeros
one	2013	Choi	1.5	0.1	0
two	2014	Choi	1.7	0.2	1
three	2015	Choi	3.6	0.3	2
four	2016	Kim	2.4	0.4	3
five	2017	Park	2.9	0.5	4

Series로 추가

```
val = pd.Series([-1.2, -1.5, -1.7], index=["two", "four", "five"])
df["debt"] = val
df
```

	year	names	points	penalty	zeros	debt
one	2013	Choi	1.5	0.1	0	nan
two	2014	Choi	1.7	0.2	1	-1.2
three	2015	Choi	3.6	0.3	2	nan
four	2016	Kim	2.4	0.4	3	-1.5
five	2017	Park	2.9	0.5	4	-1.7

DATAKUBWA

데이터프레임 열(Columns)

연산 후 새로운 열을 추가하기

```
df["net_points"] = df["points"] - df["penalty"]
```

조건 추가

```
df["high_points"] = df["net_points"] > 2.0
```

```
df
```

	year	names	points	penalty	zeros	debt	net_points	high_points
one	2013	Choi	1.5	0.1	0	nan	1.4	False
two	2014	Choi	1.7	0.2	1	-1.2	1.5	False
three	2015	Choi	3.6	0.3	2	nan	3.3	True
four	2016	Kim	2.4	0.4	3	-1.5	2.0	False
five	2017	Park	2.9	0.5	4	-1.7	2.4	True

DATAKUBWA

데이터프레임 열(Columns)

데이터프레임 열 삭제하기

```
# 열 삭제하기
del df["high_points"]
del df["net_points"]
del df["zeros"]
df
```

	year	names	points	penalty	debt
one	2014	Choi	1.5	0.1	NaN
two	2015	Choi	1.7	0.2	-1.2
three	2016	Choi	3.6	0.3	NaN
four	2015	Kim	2.4	0.4	-1.5
five	2016	Park	2.9	0.5	-1.7

데이터프레임 인덱싱

데이터프레임 인덱스 지정

df.columns

> Index(['year', 'names', 'points', 'penalty', 'debt'], dtype='object')

index와 columns 이름 지정

df.index.name = 'Order'

df.columns.name = 'Info'

df

info	year	names	points	penalty	debt
order					
one	2014	Choi	1.5	0.1	nan
two	2015	Choi	1.7	0.2	-1.2
three	2016	Choi	3.6	0.3	nan
four	kim	Charles	2.4	0.4	-1.5
five	park	Charles	2.9	0.5	-1.7

데이터프레임 인덱싱

데이터프레임 인덱싱

- 데이터프레임을 인덱싱을 할 때도 열 라벨(column label)을 키 값으로 생각하여 인덱싱
- 인덱스로 라벨 값을 하나만 넣으면 시리즈 객체가 반환되고 라벨의 배열 또는 리스트를 넣으면 부분적인 데이터프레임이 반환
- 하나의 열만 빼내면서 데이터프레임 자료형을 유지하고 싶다면 원소가 하나인 리스트를 써서 인덱싱

```
df["year"]
```

```
> Order
one 2013
two 2014
three 2015
four 2016
five 2017
Name: year, dtype: int64
```

```
# 동일한 의미를 갖는, 다른 방법
df.year
```

```
> Order
one 2013
two 2014
three 2015
four 2016
five 2017
Name: year, dtype: int64
```

데이터프레임 인덱싱

행 인덱싱

- 행 단위로 인덱싱을 하고자 하면 항상 슬라이싱(slicing)을 해야 한다.
- 인덱스의 값이 문자 라벨이면 라벨 슬라이싱

0번째 부터 2(3-1) 번째까지 가져온다. # 뒤에 써준 숫자번째의 행은 뺀다.

`df[0:3]`

info	year	names	points	penalty	debt
order					
one	2014	Choi	1.5	0.1	nan
two	2015	Choi	1.7	0.2	-1.2
three	2016	Choi	3.6	0.3	nan

DATAKUBWA

데이터프레임 인덱싱

loc 인덱싱

라벨값 기반의 2차원 인덱싱

인덱싱 값	가능	결과	자료형	추가사항
행 인덱스값(정수)	o	행	시리즈	
행 인덱스값(정수) 슬라이스	o	행	데이터프레임	loc가 없는 경우와 같음
행 인덱스값(정수) 리스트	o	행	데이터프레임	
불리언 시리즈	o	행	데이터프레임	시리즈의 인덱스가 데이터프레임의 행 인덱스와 같아야 한다.
불리언 시리즈를 반환하는 함수	o	행	데이터프레임	
열 라벨	x		loc가 없는 경우에만 쓸 수 있다.	
열 라벨 리스트	x		loc가 없는 경우에만 쓸 수 있다.	

데이터프레임 인덱싱

.loc 또는 .iloc 함수를 사용하여 Series 반환
df.loc["two"]

> Info year 2014
names Choi
points 1.7
penalty 0.2
debt -1.2
Name: two, dtype: object

.loc 또는 .iloc 함수를 사용하여 데이터프레임으로 인덱싱
df.loc["two":"four"]

info	year	names	points	penalty	debt
order					
two	2015	Choi	1.7	0.2	-1.2
three	2016	Choi	3.6	0.3	nan
four	2015	Kim	2.4	0.4	-1.5

DATAKUBWA

데이터프레임 .loc 인덱싱

```
df.loc['two':'four', 'points']
```

```
> Order
two 1.7
three 3.6
four 2.4
Name: points, dtype: float64
```

```
df.loc[:, 'year'] # == df['year']
```

```
> Order
one 2013
two 2014
three 2015
four 2016
five 2017
Name: year, dtype: int64
```

DATAKUBWA

데이터프레임 .loc 인덱싱

```
# == df['year']
df.loc[:, 'year']
```

```
> Order
one 2013
two 2014
three 2015
four 2016
five 2017
Name: year, dtype: int64
```

```
df.loc['three':'five', 'year':'penalty']
```

info	year	names	points	penalty
order				
three	2016	Choi	3.6	0.3
four	2015	Kim	2.4	0.4
five	2016	Kim	2.9	0.5

데이터프레임 .iloc 인덱싱

3.10 iloc 인덱싱

정수값 기반의 2차원 인덱싱

```
# 새로운 행 삽입하기
df.loc["six",:] = [2013,"Jun", 4.0, 0.1, 2.1]
df
```

info	year	names	points	penalty	debt
order					
one	2014.0	choi	1.5	0.1	nan
two	2015.0	choi	1.7	0.2	-1.2
three	2016.0	choi	3.6	0.3	nan
four	2015.0	kim	2.4	0.4	-1.5
five	2016.0	park	2.9	0.5	-1.7
six	2013.0	jun	4.0	0.1	2.1

```
# .iloc 사용:: index 번호를 사용한다.
df.iloc[3] # 4번째 행을 가져온다.
```

```
> Info
year 2016
names Kim
points 2.4
penalty 0.4
debt -1.5
Name: four, dtype: object
```

데이터프레임 .iloc 인덱싱

슬라이싱으로 지정하여 반환
df.iloc[3:5, 0:2]

info	year	names
order		
four	2016.0	Kim
five	2017.0	Park

각각의 행과 열을 지정하여 반환하기
df.iloc[[0, 1, 3], [1, 2]]

info	names	points
order		
one	Choi	1.5
two	Choi	1.7
four	Kim	2.4

데이터프레임 .iloc 인덱싱

행을 전체, 열은 두번째열부터 마지막까지 슬라이싱으로 지정하여 반환
 df.iloc[:, 1:4]

info	names	points	penalty
order			
one	Choi	1.5	0.1
two	Choi	1.7	0.2
three	Choi	3.6	0.3
four	Kim	2.4	0.4
five	Park	2.9	0.5
six	Jun	4.0	0.1

df.iloc[1,1]

> 'Choi'

DATAKUBWA

데이터프레임 불린 인덱싱

Boolean 인덱싱

True, False 논리연산 기반의 인덱싱

df

info	year	names	points	penalty	debt
order					
one	2014.0	Choi	1.5	0.1	nan
two	2015.0	Choi	1.7	0.2	-1.2
three	2016.0	Choi	3.6	0.3	nan
four	2015.0	Kim	2.4	0.4	-1.5
five	2016.0	Park	2.9	0.5	-1.7
six	2013.0	Jun	4.0	0.1	2.1

year가 2014보다 큰 boolean data
df["year"] > 2014

> Order
one False
two True
three True
four True
five True
six False
Name: year, dtype: bool

데이터프레임 불린 인덱싱

```
df.loc[df['name'] == "Choi", ['name', 'points']]
```

info	names	points
order		
one	Choi	1.5
two	Choi	1.7
three	Choi	3.6

numpy에서와 같이 논리연산을 응용할 수 있다.

```
df.loc[(df["points"]>2) & (df["points"]<3),:]
```

info	year	names	points	penalty	debt
order					
four	2015.0	Kim	2.4	0.4	-1.5
five	2016.0	Park	2.9	0.5	-1.7

BIGDATA & AI ANALYTICS
EXPERT COMPANY

4. pandas 데이터프레임 다루기

DATAKUBWA

데이터프레임 생성

numpy randn 데이터프레임 생성

DataFrame을 만들때 index, column을 설정하지 않으면 기본값으로 0부터 시작하는 정수형 숫자로 입력된다.

```
df = pd.DataFrame(np.random.randn(6,4))
```

df

	0	1	2	3
0	0.682000	-0.570393	-1.602829	-1.316469
1	-1.176203	0.171527	0.387018	1.027615
2	-0.263178	-0.212049	1.006856	0.096111
3	2.679378	0.347145	0.548144	0.826258
4	-0.249877	0.018721	-0.393715	0.302361
5	0.851420	-0.440360	-0.345895	1.055936

DATA KUBWA

시계열 데이터프레임 생성

시계열 데이트 함수 date_range

#pandas에서 제공하는 date_range 함수는 datetime 자료형으로 구성된, 날짜/시간 함수
df.columns = ['A', 'B', 'C', 'D']
df.index = pd.date_range('20160701', periods=6)
df.index

> DatetimeIndex(['2016-07-01', '2016-07-02', '2016-07-03', '2016-07-04',
 '2016-07-05', '2016-07-06'],
 dtype='datetime64[ns]', freq='D')

DATAKUBWA

시계열 데이터프레임 생성

df

	A	B	C	D
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469
2016-07-02	-1.176203	0.171527	0.387018	1.027615
2016-07-03	-0.263178	-0.212049	1.006856	0.096111
2016-07-04	2.679378	0.347145	0.548144	0.826258
2016-07-05	-0.249877	0.018721	-0.393715	0.302361
2016-07-06	0.851420	-0.440360	-0.345895	1.055936

DATAKUBWA

결측치 다루기

numpy로 데이터프레임 결측치 다루기

np.nan은 NaN값을 의미

df['F'] = [1.0, np.nan, 3.5, 6.1, np.nan, 7.0]

df

	A	B	C	D	F
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469	1.0
2016-07-02	-1.176203	0.171527	0.387018	1.027615	NAN
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-04	2.679378	0.347145	0.548144	0.826258	6.1
2016-07-05	-0.249877	0.018721	-0.393715	0.302361	NAN
2016-07-06	0.851420	-0.440360	-0.345895	1.055936	7.0

DATAKUBWA

결측치 다루기

행의 값중 하나라도 nan인 경우 그 행을 없앤다.
df.dropna(how='any')

	A	B	C	D	F
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469	1.0
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-04	2.679378	0.347145	0.548144	0.826258	6.1
2016-07-06	0.851420	-0.440360	-0.345895	1.055936	7.0

DATAKUBWA

결측치 다루기

행의 값의 모든 값이 nan인 경우 그 행을 없앤다.
df.dropna(how='all')

	A	B	C	D	F
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469	1.0
2016-07-02	-1.176203	0.171527	0.387018	1.027615	NAN
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-04	2.679378	0.347145			
2016-07-05	-0.249877	0.018721			
2016-07-06	0.851420	-0.440360			

nan 값에 값 넣기
df.fillna(value=0.5)

	A	B	C	D	F
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469	1.0
2016-07-02	-1.176203	0.171527	0.387018	1.027615	0.5
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-04	2.679378	0.347145	0.548144	0.826258	6.1
2016-07-05	-0.249877	0.018721	-0.393715	0.302361	0.5
2016-07-06	0.851420	-0.440360	-0.345895	1.055936	7.0

DATAKUBWA

drop 명령어

특정 행 drop하기

```
df.drop(pd.to_datetime('20160701'))
```

	A	B	C	D	F
2016-07-02	-1.176203	0.171527	0.387018	1.027615	NAN
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-04	2.679378	0.347145	0.548144	0.826258	6.1
2016-07-05	-0.249877	0.018721	-0.393715	0.302361	NAN
2016-07-06	0.851420	-0.440360	-0.345895	1.055936	7.0

2개 이상도 가능

```
df.drop([pd.to_datetime('20160702'),pd.to_datetime('20160704')])
```

	A	B	C	D	F
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469	1.0
2016-07-03	-0.263178	-0.212049	1.006856	0.096111	3.5
2016-07-05	-0.249877	0.018721	-0.393715	0.302361	NAN
2016-07-06	0.851420	-0.440360	-0.345895	1.055936	7.0

DATA KUBWA

drop 명령어

특정 열 삭제하기

df.drop('F', axis = 1)

	A	B	C	D
2016-07-01	0.682000	-0.570393	-1.602829	-1.316469
2016-07-02	-1.176203	0.171527	0.387018	1.027615
2016-07-03	-0.263178	-0.212049	1.006856	0.096111
2016-07-04	2.679378	0.347145	0.548144	0.826258
2016-07-05	-0.249877	0.018721	-0.393715	0.302361
2016-07-06	0.851420	-0.440360	-0.345895	1.055936

2개 이상의 열도 가능

df.drop(['B','D'], axis = 1)

	A	C	F
2016-07-01	0.682000	-1.602829	1.0
2016-07-02	-1.176203	0.387018	NAN
2016-07-03	-0.263178	1.006856	3.5
2016-07-04	2.679378	0.548144	6.1
2016-07-05	-0.249877	-0.393715	NAN
2016-07-06	0.851420	-0.345895	7.0

BIGDATA & AI ANALYTICS
EXPERT COMPANY

5. pandas 데이터 입출력

Pandas 데이터 불러오기

pandas 데이터 불러오기

- pandas는 데이터 분석을 위해 여러 포맷의 데이터 파일을 읽을 수 있다.
- csv, excel, html, json, hdf5, sas, stata, sql

```
pd.read_csv('data/sample1.csv')
```

	c 1	c 2	c 3
0	1	1.11	one
1	2	2.22	two
2	3	3.33	three

'c1'을 인덱스로 불러오기

```
pd.read_csv('data/sample1.csv', index_col= 'c1')
```

	c 2	c 3
c 1		
1	1.11	one
2	2.22	two
3	3.33	three

Pandas 데이터 쓰기

pandas 데이터 쓰기

pandas는 데이터프레임을 출력하는데 여러가지 포맷을 지원

```
df.to_csv('sample6.csv')
```

index, header 인수를 지정하여 인덱스 및 헤더 출력 여부를 지정

```
df.to_csv('sample9.csv', index=False, header=False)
```

4.8 pandas 웹에서 데이터 불러오기

```
# 인터넷 링크의 데이터 불러오기
```

```
titanic = pd.read_excel("http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls")
```

```
titanic.head()
```

```
titanic.head(10)
```

```
titanic.tail
```

BIGDATA & AI ANALYTICS
EXPERT COMPANY

5. pandas 데이터 처리

DATAKUBWA

데이터프레임 정렬

정렬 (Sort)

- 데이터를 정렬로 `sort_index`는 인덱스 값을 기준으로,
- `sort_values`는 데이터 값을 기준으로 정렬

```
# np.random으로 시리즈 생성
s = pd.Series(np.random.randint(6, size=100))
s.head()
```

```
> 0 0
1 4
2 3
3 4
4 3
dtype: int64
```

데이터프레임 정렬

정렬 (Sort)

- 데이터를 정렬로 sort_index는 인덱스 값을 기준으로,
- sort_values는 데이터 값을 기준으로 정렬

s.value_counts().sort_index()

```
> 0 18
1 22
2 13
3 14
4 17
5 16
dtype: int64
```

참고: NaN값이 있는 경우에는 정렬하면 NaN값이 가장 나중에 반환

s1.sort_values()

```
> 0 0.0
1 1.0
2 2.0
4 4.0
5 5.0
6 6.0
7 7.0
8 8.0
9 9.0
3 NaN
dtype: float64
```

DATAKUBWA

내림차순, 오름차순 정렬

내림차순 정렬

큰 수에서 작은 수로 반대 방향 정렬하려면 'ascending=false' 인수를 지정

```
# ascending=False 인자로 내림차순 정렬
s.sort_values(ascending=False).head()
```

```
> 0 5
10 5
44 5
53 5
57 5 dtype: int64
```

```
# 데이터프레임에서 by 인수로 정렬 기준이 되는 열을 지정
df.sort_values(by=1)
```

	0	1	2	3
0	0.0	0.0	3.0	2.0
1	3.0	0.0	2.0	1.0
2	3.0	2.0	4.0	NAN
3	4.0	3.0	4.0	2.0

DATAKUBWA

apply 함수

apply 변환

- 행이나 열 단위로 더 복잡한 처리를 하고 싶을 때는 apply 메서드를 사용
- 인수로 행 또는 열을 받는 함수를 apply 메서드의 인수로 넣으면 각 열(또는 행)을 반복하여 수행

```
df = pd.DataFrame({
    'A': [1, 3, 4, 3, 4],
    'B': [2, 3, 1, 2, 3],
    'C': [1, 5, 2, 4, 4]
})
df
```

	A	B	C
0	1	2	1
1	3	3	5
2	4	1	2
3	3	2	4
4	4	3	4

DATAKUBWA

apply-lambda

lambda 함수

파이썬에서 'lambda' 는 런타임에 생성해서 사용할 수 있는 익명 함수
 lambda는 쓰고 버리는 일시적인 함수로 생성된 곳에서만 적용

람다 함수 사용

```
df.apply(lambda x: x.max() - x.min())
```

```
> A 3
   B 2
   C 4
dtype: int64
```

만약 행에 대해 적용하고 싶으면 axis=1 인수 사용

```
df.apply(lambda x: x.max() - x.min(), axis=1)
```

```
> 0 1
   1 2
   2 3
   3 2
   4 1
dtype: int64
```

apply 데이터 다루기

4.14 apply로 데이터 다루기

- NaN 값은 fillna 메서드를 사용하여 원하는 값으로 변환 가능
- astype 메서드로 전체 데이터의 자료형을 바꾸는 것도 가능

df.apply(pd.value_counts)

	A	B	C
1	1.0	1.0	1.0
2	NAN	2.0	1.0
3	2.0	2.0	NAN
4	2.0	NAN	2.0
5	NAN	NAN	1.0

df.apply(pd.value_counts).fillna(0).astype(int)

	A	B	C
1	1	1	1
2	0	2	1
3	2	2	0
4	2	0	2
5	0	0	1

DATAKUBWA

describe 메서드

describe 메서드

describe() 함수는 DataFrame의 계산 가능한 값들의 기본 통계값을 반환

df.describe()

	A	B	C
count	5.000000	5.000000	5.000000
mean	3.000000	2.200000	3.200000
std	1.224745	0.836666	1.643168
min	1.000000	1.000000	1.000000
25%	3.000000	2.000000	2.000000
50%	3.000000	2.000000	4.000000
75%	4.000000	3.000000	4.000000
max	4.000000	3.000000	5.000000

BIGDATA & AI ANALYTICS
EXPERT COMPANY

7. pandas 시계열 분석

DATAKUBWA

pd.to_datetime

날짜/시간을 나타내는 문자열을 자동으로 datetime 자료형으로 바꾼 후
datetimeindex 자료형 인덱스를 생성

```
date_str = ["2018, 1, 1", "2018, 1, 4", "2018, 1, 5", "2018, 1, 6"]
idx = pd.to_datetime(date_str)
idx
```

```
DatetimeIndex(['2018-01-01', '2018-01-04', '2018-01-05', '2018-01-06'],
              dtype='datetime64[ns]', freq=None)
```

```
# 인덱스를 사용하여 시리즈나 데이터프레임을 생성
np.random.seed(0)
s = pd.Series(np.random.randn(4), index=idx)
s
```

```
2018-01-01 1.764052
2018-01-04 0.400157
2018-01-05 0.978738
2018-01-06 2.240893 dtype: float64
```

DATAKUBWA

pd.date_range

시작일과 종료일 또는 시작일과 기간을 입력하면 범위 내의 인덱스를 생성

```
pd.date_range("2018-4-1", "2018-4-5")
```

```
DatetimeIndex(['2018-04-01',
                '2018-04-02',
                '2018-04-03',
                '2018-04-04',
                '2018-04-05'], dtype='datetime64[ns]', freq='D')
```

```
pd.date_range(start="2018-4-1", periods=5)
```

```
DatetimeIndex(['2018-04-01',
                '2018-04-02',
                '2018-04-03',
                '2018-04-04',
                '2018-04-05'], dtype='datetime64[ns]', freq='D')
```

BIGDATA & AI ANALYTICS
EXPERT COMPANY

기말고사 과제

D A T A K U B W A

타이타닉 생존 데이터

Features 변수명 설명

- **Survival:** survival (0 = no; 1 = yes)
- **Pclass:** passenger class (1 = 1st; 2 = 2nd; 3 = 3rd)
- **Name:** name
- **Sex:** sex
- **Age:** age
- **Sibsp:** number of siblings/spouses aboard
- **Parch:** number of parents/children aboard
- **Ticket:** ticket number
- **Fare:** passenger fare
- **Cabin:** cabin
- **Embarked:** port of embarkation (C = cherbourg; Q = queenstown; S = southampton)

D A T A K U B W A

IMDB 영화 평점 데이터

Features 변수명 설명

전문가 및 사용자 평점 데이터