

파이썬 데이터분석

- Pandas

강사 : KAIST 김동훈

I . Pandas 시작

- What is Pandas?
- Pandas 특징
- Series & DataFrame
- Object Creation
- Data Viewing
- Baseball.csv

I. Pandas 시작

- What is Pandas ?

판다스는 금융 데이터를 계량 분석하기 위해서 개발되었다. 패널 데이터(panel data) 구조를 제공하기 위해 Numpy 위에 구성되도록 개발하였으므로 넘파이에서 쉽게 사용할 수 있다. CSV, Excel, SQL 등 여러 형식의 데이터를 분석하고 처리할 수 있으며 데이터 행과 열의 라벨로 데이터를 분석하고 처리할 수 있다. *

- Timeline:

- 2008: Development of pandas started

- 2009: pandas becomes open source

- 2012: First edition of Python for Data Analysis is published

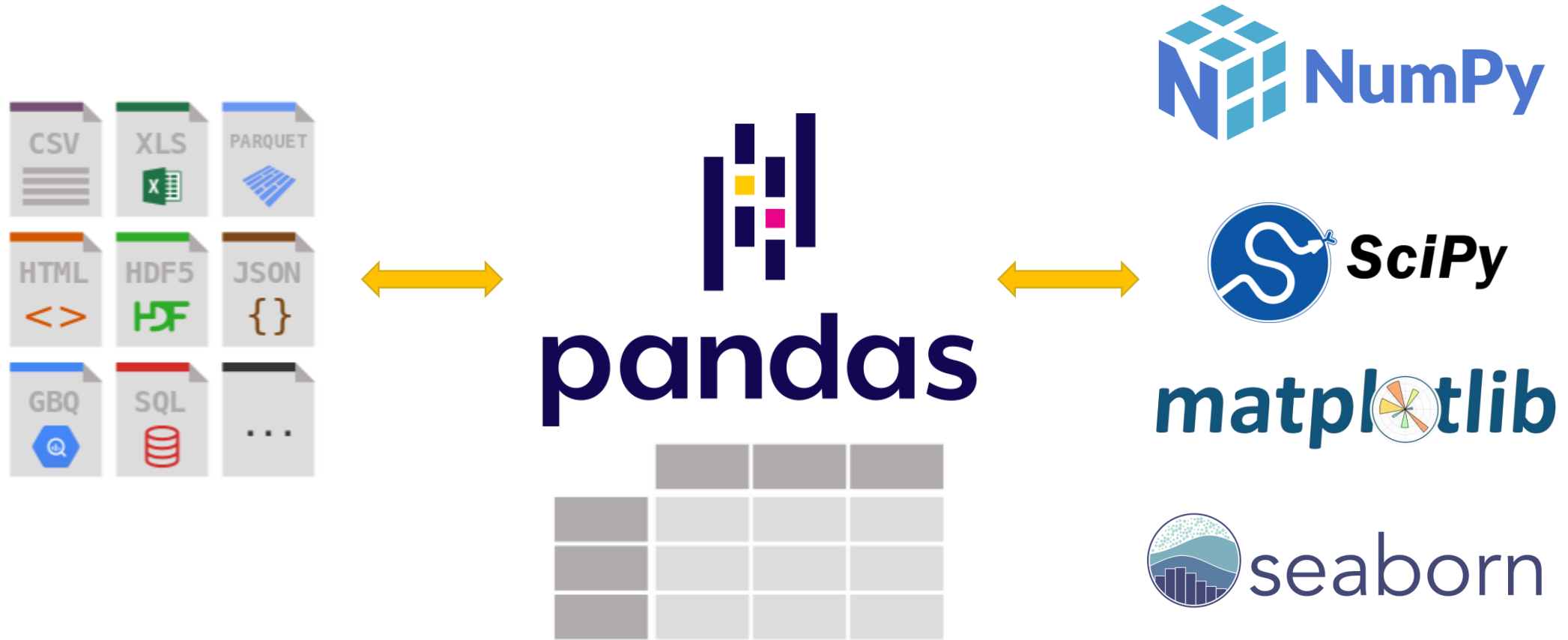
- 2015: pandas becomes a NumFOCUS sponsored project

- 2018: First in-person core developer sprint

I. Pandas 시작

- 판다스 라이브러리

- MS Excel 과 유사하지만, 다른 파이썬 라이브러리들과 결합해서 강력한 데이터 분석 도구가 된다.



I . Pandas 시작

- 판다스 주요 기능

- A fast and efficient **DataFrame** object
- Tools for reading and writing data between different formats: **CSV**, **Excel**, SQL DB
- Intelligent data **alignment** and integrated handling of **missing data**
- Flexible reshaping and pivoting
- Label-based **slicing**, fancy indexing
- A powerful **group by** engine
- High performance **Merging** and **Joining**
- Hierarchical axis indexing (= **Multiindex**)
- **Time series**-functionality
- Highly optimized for performance
- Wide variety of applications

I . Pandas 시작

- Pandas Cheat Sheet

https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

I. Pandas 시작

- Series 와 DataFrame 개념
 - Series : 1차원 array (라벨 있는), \approx ndarray
 - DataFrame : 2차원 data structure (라벨 있는), \approx a dictionary of Series Object

Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

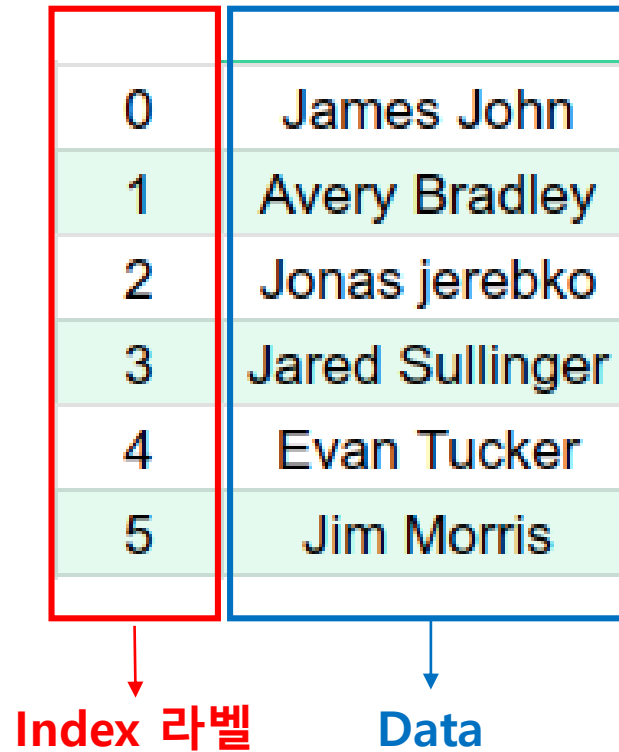
DataFrame

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

I. Pandas 시작

- Series : 1차원 array (라벨 있는), \approx ndarray

0	James John
1	Avery Bradley
2	Jonas jerebko
3	Jared Sullinger
4	Evan Tucker
5	Jim Morris



Index 라벨 Data

I. Pandas 시작

- DataFrame : 2차원 data structure (라벨 있는), \approx a dictionary of Series Object

Column 라벨

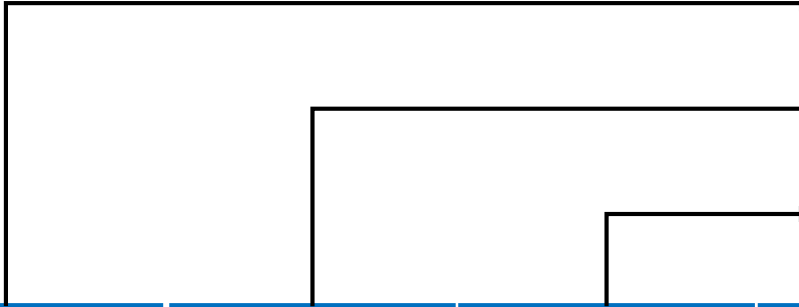
	Name	Korean	English	Math.	Average
0	James John	80	98	85	87.7
1	Avery Bradley	85	98	70	84.3
2	Jonas jerebko	40	94	54	62.7
3	Jared Sullinger	70	90	68	76.0
4	Evan Tucker	100	80	85	88.3
5	Jim Morris	95	85	92	90.7

Data

Index 라벨

I. Pandas 시작

- **DataFrame** 의 특정 Column 을 선택하면 → **Series** 객체가 된다.



The diagram illustrates the process of selecting a specific column from a DataFrame to create a Series. It features a DataFrame table with five columns: Name, Korean, English, Math., and Average. Three arrows originate from the top of the 'Name', 'Korean', and 'English' columns and point to the corresponding assignment statements on the right: `ser_name = df['Name']`, `ser_kor = df['Korean']`, and `ser_eng = df['English']`.

	Name	Korean	English	Math.	Average
0	James John	80	98	85	87.7
1	Avery Bradley	85	98	70	84.3
2	Jonas jerebko	40	94	54	62.7
3	Jared Sullinger	70	90	68	76.0
4	Evan Tucker	100	80	85	88.3
5	Jim Morris	95	85	92	90.7

I . Pandas 시작

- Series (only) Members

```
argmax, argmin, argsort, array, autocorr, between, cat,  
divmod, dt, dtype, factorize, hasnans, item, map,  
name, nbytes, ravel, rdivmod, repeat, searchsorted, str,  
tolist, unique, view,
```

- DataFrame (only) Members

```
applymap, assign, boxplot, columns, corrwith, eval, info,  
insert, iterrows, itertuples, join, lookup, melt, merge,  
pivot, query, stack, style,
```

- Both Members

```
T, abs, add, agg, aggregate, align, all,  
any, append, apply, asfreq, asof, astype, at,  
attrs, axes, backfill, bfill, bool, clip, combine,  
compare, copy, corr, count, cov, cummax, cummin,  
cumprod, cumsum, describe, diff, div, divide, dot,  
drop, droplevel, dropna, dtypes, duplicated, empty, eq,  
equals, ewm, expanding, explode, ffill, fillna, filter,  
first, floordiv, ge, get, groupby, gt, head,  
hist, iat, idxmax, idxmin, iloc, index, interpolate,  
isin, isna, isnull, items, iteritems, keys, kurt,  
kurtosis, last, le, loc, lt, mad, mask,  
max, mean, median, min, mod, mode, mul,  
multiply, ndim, ne, nlargest, notna, notnull, nsmallest,
```

I. Pandas 시작

- Series 생성

- Series 를 생성하기 위해서는 데이터가 필요: (예) [10, 20, 30]
- pd.Series(data=None, index=None) 를 사용하여 생성

① data와 index 를 인자로 구분해서 생성

```
In [16]: se1 = pd.Series(np.random.randn(4)) # 인덱스 미지정  
se2 = pd.Series(data=np.random.randn(4), index=['a', 'b', 'c', 'd']) # 인덱스 지정  
se3 = pd.Series(data=[1, 2, 3, 4], index=['a', 'b', 'c', 'd']) # 인덱스 지정
```

출력값:

```
se1:  
0    0.694259  
1    0.021438  
2    0.729406  
3    3.160829  
dtype: float64
```

```
se2:  
a   -0.525557  
b   -0.391010  
c    0.501442  
d    0.262316  
dtype: float64
```

```
se3:  
a     1  
b     2  
c     3  
d     4  
dtype: int64
```

I. Pandas 시작

- Series 생성

- Series 를 생성하기 위해서는 데이터가 필요: (예) [10, 20, 30]
- `pd.Series(data=None, index=None)` 를 사용하여 생성

② dictionary 로 부터 (key=index, value=data)

```
In [18]: a_dict = {'a':1, 'b':2, 'c':3, 'd':4}
         se4 = pd.Series(a_dict)
```

출력값:

```
se3:
a    1
b    2
c    3
d    4
dtype: int64
```

I. Pandas 시작

- DataFrame 생성

- DataFrame 은 Series 여러개를 dictionary 에 담아서 생성하는 형태

① 여러 Series 객체를 조합해서, ※ index 지정하는 경우 각 Series 들의 인덱스와 일치여부가 중요

```
In [27]: se1 = pd.Series([1,2,3,4])  
se2 = pd.Series([10,20,30,40])  
df1 = pd.DataFrame({'col1':se1, 'col2':se2})
```

출력값:

df1:		
	col1	col2
0	1	10
1	2	20
2	3	30
3	4	40

I. Pandas 시작

- DataFrame 생성

- DataFrame 은 Series 여러개를 dictionary 에 담아서 생성하는 형태

- ② dictionary 로부터 (key=컬럼, value=해당 컬럼의 값)

```
In [36]: df2 = pd.DataFrame(data={
            'col1': [1, 2, 3, 4],
            'col2': [10, 20, 30, 40],
            'col3': [100, 200, 300, 400]},
            index = range(1, 5))
```

출력값:

	col1	col2	col3
1	1	10	100
2	2	20	200
3	3	30	300
4	4	40	400

I. Pandas 시작

- DataFrame 생성

- DataFrame 은 Series 여러개를 dictionary 에 담아서 생성하는 형태

※ 중요 : DataFrame 의 한 개 컬럼만 선택하면 Series 객체가 된다.

```
In [18]: a = df3["Name"]  
         type(a)
```

```
Out[18]: pandas.core.series.Series
```

```
In [19]: a
```

```
Out[19]: 1      James John  
         2      Avery Bradley  
         3      Jonas jerebko  
         4      Jared Sullinger  
         5      Evan Tucker  
         6      Jim Morris  
         Name: Name, dtype: object
```


I. Pandas 시작

- DataFrame 생성

- DataFrame 은 다른 데이터 소스 (.csv, excel, SQL 등) 로 부터 불러오기해서 생성 가능

from .csv file

```
In [3]: df = pd.read_csv('./data_raw/baseball.csv')
```

from excel file

```
In [24]: df = pd.read_excel('./data_raw/performance.xlsx')
```

행, 열이 많을 경우 가운데 생략해서 보여주는데 이를 회피하려면 아래와 같이 세팅하면 된다.

```
pd.set_option("display.max_rows", 보여질 숫자)  
pd.set_option("display.max_columns", 보여질 숫자)
```

I. Pandas 시작

- Encoding 문제

※ 한글 encoding : 'utf-8', 'euc-kr', 'cp949'

'euc-kr' : 한글 2350자만 지원

'cp949' : 한국어 MS Windows 용 인코딩, 'euc-kr' 의 확장형, '확장 완성형' 이라고 불림

'utf-8' : 맥과 리눅스용 Unicode 인코딩

```
In [20]: df = pd.read_csv('./data_raw/population.csv', encoding='euc-kr')  
df
```

I. Pandas 시작

- Data 훑어보기(Viewing)

- `df.head()` : DataFrame 젤 처음 행부터 일부를 보여줌.
- `df.tail()` : DataFrame 젤 뒷 행부터 일부를 보여줌.
- `df.index` : DataFrame 의 행 라벨(index) 정보를 보여줌.
- `df.columns` : DataFrame 의 컬럼 라벨(columns) 정보를 보여줌.
- `df.describe()`: DataFrame 데이터의 기초 통계 정보를 보여줌.
- `df.info()` : DataFrame 의 index, column, datatype 등의 정보를 보여줌.
- `df['column'].value_counts()` : 특정 컬럼의 개수를 세어서 보여줌.
- `df.nlargest()` / `df.nsmallest()` : 특정 컬럼의 가장 큰 순서(작은 순서) 대로 가져 옴.

I. Pandas 시작

- Data 훑어보기(Viewing)

- `df['column'].unique()` : 특정 컬럼의 유일한 값들을 보여줌.
- `df.hist()` : DataFrame 의 각 컬럼들의 데이터를 히스토그램으로 나타내 줌.
- `df['column'].hist()` : 특정 컬럼을 데이터를 히스토그램으로 나타내 줌.
- `df.sample()` : DataFrame 의 데이터 일부(행) 을 sampling 해서 가져 옴.
- `df.filter()` : DataFrame 을 조건을 걸어 일부 데이터를 가져 옴.

I. Pandas 시작

- baseball.csv 예제를 통해 기초적인 Pandas 사용법을 익혀 보자.

```
[3]: df = pd.read_csv('./data_raw/baseball.csv')
```

```
[4]: df
```

```
[4]:
```

	id	player	year	stint	team	lg	g	ab	r	h	...	rbi	sb	cs	bb	so	ibb	hbp	sh	sf	gidp
0	88641	womacto01	2006	2	CHN	NL	19	50	6	14	...	2.0	1.0	1.0	4	4.0	0.0	0.0	3.0	0.0	0.0
1	88643	schilcu01	2006	1	BOS	AL	31	2	0	1	...	0.0	0.0	0.0	0	1.0	0.0	0.0	0.0	0.0	0.0
2	88645	myersmi01	2006	1	NYA	AL	62	0	0	0	...	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0
3	88649	helliri01	2006	1	MIL	NL	20	3	0	0	...	0.0	0.0	0.0	0	2.0	0.0	0.0	0.0	0.0	0.0
4	88650	johnsra05	2006	1	NYA	AL	33	6	0	1	...	0.0	0.0	0.0	0	4.0	0.0	0.0	0.0	0.0	0.0
...
95	89525	benitar01	2007	2	FLO	NL	34	0	0	0	...	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0
96	89526	benitar01	2007	1	SFN	NL	19	0	0	0	...	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0
97	89530	ausmubr01	2007	1	HOU	NL	117	349	38	82	...	25.0	6.0	1.0	37	74.0	3.0	6.0	4.0	1.0	11.0
98	89533	aloumo01	2007	1	NYN	NL	87	328	51	112	...	49.0	3.0	0.0	27	30.0	5.0	2.0	0.0	3.0	13.0
99	89534	alomasa02	2007	1	NYN	NL	8	22	1	3	...	0.0	0.0	0.0	0	3.0	0.0	0.0	0.0	0.0	0.0

100 rows x 23 columns

II. 결측치 처리 & 컬럼 연산

- 기초 통계량 확인
- 컬럼 추출 및 추가
- drop 함수
- 인덱스 재설정
- 결측치 확인
- 결측치 처리

II. 결측치 처리 & 컬럼 연산

- 기초 통계량 확인
 - `df.sum()`: 값들의 합을 반환
 - `df.count()`: 결측치를 제외한 값들의 수 반환
 - `df.max()`: 최대값을 반환
 - `df.min()`: 최소값을 반환
 - `df.mean()`: 평균값을 반환
 - `df.median()`: 중간값을 반환
 - `df.std()`: 표준 편차를 반환

II. 결측치 처리 & 컬럼 연산

- 컬럼 추출

- 컬럼명 사용하여 추출 → 한 개 컬럼 선택시 Series 객체 반환

df (DataFrame)

	날짜	전력소모량	태양광	풍력
0	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
3	2024-08-13	122	30	27
4	2024-08-14	127	29	22



df['태양광']

0 34

1 34

2 32

3 30

4 29

Name: 태양광, dtype: int64

II. 결측치 처리 & 컬럼 연산

- 컬럼 추출
 - 복수개 컬럼 선택 → DataFrame 반환

df (DataFrame)

	날짜	전력소모량	태양광	풍력
0	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
3	2024-08-13	122	30	27
4	2024-08-14	127	29	22



```
df[['전력소모량', '태양광', '풍력']]
```

	전력소모량	태양광	풍력
0	112	34	21
1	108	34	19
2	134	32	24
3	122	30	27
4	127	29	22

특정 컬럼을 선택하거나 복수 컬럼을 선택하는 방법은 다음 시간에 자세히 다룰 예정

II. 결측치 처리 & 컬럼 연산

- 컬럼 추가
 - 컬럼명과 함께 차원이 맞는 데이터 추가
 - DataFrame 을 다룰 때 원하는 data 를 가공 후 별도의 컬럼으로 추가하는 일이 많이 일어난다.

df (DataFrame)

	날짜	전력소모량	태양광	풍력
0	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
3	2024-08-13	122	30	27
4	2024-08-14	127	29	22



```
df['태양광+풍력'] = df['태양광'] + df['풍력']  
df
```

	날짜	전력소모량	태양광	풍력	태양광+풍력
0	2024-08-10	112	34	21	55
1	2024-08-11	108	34	19	53
2	2024-08-12	134	32	24	56
3	2024-08-13	122	30	27	57
4	2024-08-14	127	29	22	51

II. 결측치 처리 & 컬럼 연산

- 컬럼/행 삭제
 - 행 삭제, `df.drop(0)` or `df.drop(0, axis=0)`
 - 컬럼 삭제, `df.drop('a', axis=1)`

df (DataFrame)

	날짜	전력소모량	태양광	풍력	태양광+풍력
0	2024-08-10	112	34	21	55
1	2024-08-11	108	34	19	53
2	2024-08-12	134	32	24	56
3	2024-08-13	122	30	27	57
4	2024-08-14	127	29	22	51



삭제할, index label 을 적으면 된다.

```
df.drop(0)
```

	날짜	전력소모량	태양광	풍력	태양광+풍력
1	2024-08-11	108	34	19	53
2	2024-08-12	134	32	24	56
3	2024-08-13	122	30	27	57
4	2024-08-14	127	29	22	51

II. 결측치 처리 & 컬럼 연산

- 컬럼/행 삭제
 - 행 삭제, `df.drop(0)` or `df.drop(0, axis=0)`
 - 컬럼 삭제, `df.drop('a', axis=1)`

df (DataFrame)

	날짜	전력소모량	태양광	풍력	태양광+풍력
0	2024-08-10	112	34	21	55
1	2024-08-11	108	34	19	53
2	2024-08-12	134	32	24	56
3	2024-08-13	122	30	27	57
4	2024-08-14	127	29	22	51



```
df.drop('태양광+풍력', axis=1)
```

	날짜	전력소모량	태양광	풍력
0	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
3	2024-08-13	122	30	27
4	2024-08-14	127	29	22

II. 결측치 처리 & 컬럼 연산

- 컬럼/행 삭제
 - 행 삭제, `df.drop(0)` or `df.drop(0, axis=0)`
 - 여러 컬럼 삭제, `df.drop(['a','b'], axis=1)`

df (DataFrame)

	날짜	전력소모량	태양광	풍력	태양광+풍력
0	2024-08-10	112	34	21	55
1	2024-08-11	108	34	19	53
2	2024-08-12	134	32	24	56
3	2024-08-13	122	30	27	57
4	2024-08-14	127	29	22	51



```
df.drop(['태양광', '풍력', '태양광+풍력'], axis=1)
```

	날짜	전력소모량
0	2024-08-10	112
1	2024-08-11	108
2	2024-08-12	134
3	2024-08-13	122
4	2024-08-14	127

II. 결측치 처리 & 컬럼 연산

- 컬럼명 변경
 - `df.columns = ['a', 'b']`, 한꺼번에 변경하고 싶을 때
 - `df.rename(columns = {'old_name' : 'new_name'}, inplace=True)`

```
df.columns = ['날짜', '전력소모량', '태양광', '풍력']  
df
```

df (DataFrame)

	date	consum.	solar.	wind.
0	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
3	2024-08-13	122	30	27
4	2024-08-14	127	29	22



	날짜	전력소모량	태양광	풍력
0	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
3	2024-08-13	122	30	27
4	2024-08-14	127	29	22

II. 결측치 처리 & 컬럼 연산

- 컬럼명 변경

- `df.columns = ['a', 'b']`, 한꺼번에 변경하고 싶을 때

- `df.rename(columns = {'old_name' : 'new_name'}, inplace=True)`

```
df.rename( columns= { 'solar.' : '태양광',  
                      'wind.' : '풍력' })  
df
```

df (DataFrame)

	date	consum.	solar.	wind.
0	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
3	2024-08-13	122	30	27
4	2024-08-14	127	29	22



	date	consum.	태양광	풍력
0	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
3	2024-08-13	122	30	27
4	2024-08-14	127	29	22

II. 결측치 처리 & 컬럼 연산

- 인덱스 라벨 변경

- `df.index = ['a', 'b']`, 한꺼번에 변경하고 싶을 때

- `df.rename(index = { 'old_name' : 'new_name' }, inplace=True)`

```
df.index = [ 'AA', 'BB', 'CC', 'DD', 'EE' ]  
df
```

df (DataFrame)

	날짜	전력소모량	태양광	풍력
0	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
3	2024-08-13	122	30	27
4	2024-08-14	127	29	22



	날짜	전력소모량	태양광	풍력
AA	2024-08-10	112	34	21
BB	2024-08-11	108	34	19
CC	2024-08-12	134	32	24
DD	2024-08-13	122	30	27
EE	2024-08-14	127	29	22

II. 결측치 처리 & 컬럼 연산

- 인덱스 라벨 변경

- `df.index = ['a', 'b']`, 한꺼번에 변경하고 싶을 때

- `df.rename(index = {'old_name' : 'new_name'}, inplace=True)`

```
df.rename(index={0: 'AA', 3: 'BB'})  
df
```

df (DataFrame)

	날짜	전력소모량	태양광	풍력
0	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
3	2024-08-13	122	30	27
4	2024-08-14	127	29	22



	날짜	전력소모량	태양광	풍력
AA	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
BB	2024-08-13	122	30	27
4	2024-08-14	127	29	22

II. 결측치 처리 & 컬럼 연산

- 인덱스 재설정

- `df.set_index('Date')`

```
df.set_index('날짜', inplace=True)  
df
```

df (DataFrame)

	날짜	전력소모량	태양광	풍력
0	2024-08-10	112	34	21
1	2024-08-11	108	34	19
2	2024-08-12	134	32	24
3	2024-08-13	122	30	27
4	2024-08-14	127	29	22



	전력소모량	태양광	풍력
날짜			
2024-08-10	112	34	21
2024-08-11	108	34	19
2024-08-12	134	32	24
2024-08-13	122	30	27
2024-08-14	127	29	22

II. 결측치 처리 & 컬럼 연산

■ 결측치

- 현실의 데이터를 다루다 보면 일부 값이 누락되는 경우가 있다.
ex) 기상 기기 고장으로 14-16시의 습도 데이터가 측정되지 않음
- 일부 결측치가 전체 데이터셋의 품질에 영향을 미칠 수 있기 때문에 적절한 처리가 필요하다.

player	year	stint	team	lg	g	ab	r	h	X2b	X3b	hr
womacto01	2006	2	CHN	NL	19	50	6	14	1	0	1
schilcu01	2006	1	BOS	AL	31	2	0	1	0	0	0
myersmi01	2006	1	NYA		62	0	0	0			
helliri01	2006	1	MIL	NL	20	3	0	0	0	0	0
johnsra05	2006	1	NYA	AL	33	6	0	1	0	0	0
finlest01	2006	1	SFN	NL	139	426	66	105			
gonzalu01	2006	1	ARI	NL	153	586	93	159	52	2	15
seleaa01	2006	1	LAN	NL	28	26	2	5	1	0	0
francju01	2007	2	ATL	NL	15		1		3	0	0

II. 결측치 처리 & 컬럼 연산

- 결측치를 확인하는 다양한 방법

- **df.info()**
- **isnull()**
- **isnull().sum()**
- **isnull().any()**
- **pd.isna(df)**

```
df.info()
```

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

```
RangeIndex: 10 entries, 0 to 9
```

```
Data columns (total 13 columns):
```

#	Column	Non-Null Count	Dtype
0	id	10 non-null	int64
1	player	10 non-null	object
2	year	9 non-null	float64
3	stint	10 non-null	int64
4	team	10 non-null	object
5	lg	10 non-null	object
6	g	10 non-null	int64
7	ab	10 non-null	int64
8	r	10 non-null	int64
9	h	9 non-null	float64
10	X2b	4 non-null	float64
11	X3b	7 non-null	float64
12	hr	7 non-null	float64

```
dtypes: float64(5), int64(5), object(3)
```

```
memory usage: 1.1+ KB
```

II. 결측치 처리 & 컬럼 연산

- 결측치를 확인하는 다양한 방법

- `df.info()`
- **`isnull()`**
- `isnull().sum()`
- `isnull().any()`
- `pd.isna(df)`

```
df.isnull()
```

	id	player	year	stint	team	lg	g	ab	r	h	X2b	X3b	hr
0	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	True	False	True
2	False	False	False	False	False	False	False	False	False	False	True	True	True
3	False	False	False	False	False	False	False	False	False	True	False	False	True
4	False	False	False	False	False	False	False	False	False	False	True	True	False
5	False	False	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	True	True	False
8	False	False	True	False	False	False	False	False	False	False	True	False	False
9	False	False	False	False	False	False	False	False	False	False	True	False	False

II. 결측치 처리 & 컬럼 연산

- 결측치를 확인하는 다양한 방법

- df.info()
- isnull()
- **isnull().sum()**
- isnull().any()
- pd.isna(df)

```
df.isnull().sum()
```

id	0
player	0
year	1
stint	0
team	0
lg	0
g	0
ab	0
r	0
h	1
X2b	6
X3b	3
hr	3
dtype:	int64

II. 결측치 처리 & 컬럼 연산

- 결측치를 확인하는 다양한 방법

- df.info()
- isnull()
- isnull().sum()
- **isnull().any()**
- pd.isna(df)

```
df.isnull().any()
```

id	False
player	False
year	True
stint	False
team	False
lg	False
g	False
ab	False
r	False
h	True
X2b	True
X3b	True
hr	True
dtype:	bool

II. 결측치 처리 & 컬럼 연산

▪ 결측치를 확인하는 다양한 방법

- df.info()
- isnull()
- isnull().sum()
- isnull().any()
- **pd.isna(df)**

```
pd.isna(df)
```

	id	player	year	stint	team	lg	g	ab	r	h	X2b	X3b	hr
0	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	True	False	True
2	False	False	False	False	False	False	False	False	False	False	True	True	True
3	False	False	False	False	False	False	False	False	False	True	False	False	True
4	False	False	False	False	False	False	False	False	False	False	True	True	False
5	False	False	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	True	True	False
8	False	False	True	False	False	False	False	False	False	False	True	False	False
9	False	False	False	False	False	False	False	False	False	False	True	False	False

※ 주의 : pd.isna 는 pandas 의 메서드

II. 결측치 처리 & 컬럼 연산

- 결측치 처리
 - `df.fillna(value)`: 결측치를 특정 값으로 모두 채움

```
df.fillna(value=0)
```

	id	player	year	stint	team	lg	g	ab	r	h	X2b	X3b	hr
0	88641	womacto01	2006.0	2	CHN	NL	19	50	6	14.0	1.0	0.0	1.0
1	88643	schilcu01	2006.0	1	BOS	AL	31	2	0	1.0	0.0	0.0	0.0
2	88645	myersmi01	2006.0	1	NYA	AL	62	0	0	0.0	0.0	0.0	0.0
3	88649	helliri01	2006.0	1	MIL	20	3	0	0	0.0	0.0	0.0	0.0
4	88650	johnsra05	2006.0	1	NYA	AL	33	6	0	1.0	0.0	0.0	0.0

II. 결측치 처리 & 컬럼 연산

- 결측치 처리
 - df.bfill / ffill
 - ffill = forwardfill: 앞에 오는 마지막 유효한 값으로 모두 채움

```
df[['id', 'h', 'X2b', 'X3b', 'hr']]
```

	id	h	X2b	X3b	hr
0	88641	14.0	1.0	0.0	1.0
1	88643	1.0	NaN	0.0	NaN
2	88645	0.0	NaN	NaN	NaN
3	88649	NaN	0.0	0.0	NaN
4	88650	1.0	NaN	NaN	0.0
5	88652	105.0	21.0	12.0	6.0



```
df.ffill()[['id', 'h', 'X2b', 'X3b', 'hr']]
```

	id	h	X2b	X3b	hr
0	88641	14.0	1.0	0.0	1.0
1	88643	1.0	1.0	0.0	1.0
2	88645	0.0	1.0	0.0	1.0
3	88649	0.0	0.0	0.0	1.0
4	88650	1.0	0.0	0.0	0.0
5	88652	105.0	21.0	12.0	6.0

II. 결측치 처리 & 컬럼 연산

- 결측치 처리
 - df.bfill / ffill
 - bfill = backfill: 뒤에 오는 최초의 유효한 값으로 모두 채움

```
df[['id', 'h', 'X2b', 'X3b', 'hr']]
```

	id	h	X2b	X3b	hr
0	88641	14.0	1.0	0.0	1.0
1	88643	1.0	NaN	0.0	NaN
2	88645	0.0	NaN	NaN	NaN
3	88649	NaN	0.0	0.0	NaN
4	88650	1.0	NaN	NaN	0.0
5	88652	105.0	21.0	12.0	6.0



```
df.bfill()[['id', 'h', 'X2b', 'X3b', 'hr']]
```

	id	h	X2b	X3b	hr
0	88641	14.0	1.0	0.0	1.0
1	88643	1.0	0.0	0.0	0.0
2	88645	0.0	0.0	0.0	0.0
3	88649	1.0	0.0	0.0	0.0
4	88650	1.0	21.0	12.0	0.0
5	88652	105.0	21.0	12.0	6.0

II. 결측치 처리 & 컬럼 연산

- 결측치 처리
 - **df.dropna()**: 결측치를 포함한 행(열) 삭제
 - 결측치 있는 행 삭제 : `df.dropna(axis=0)`
 - 결측치 있는 열 삭제 : `df.dropna(axis=1)`

Ⅲ. 인덱싱

- Series 인덱싱
- DataFrame 인덱싱
- Boolean 인덱싱

Ⅲ. 인덱싱

- Indexing 의 삼총사

- [] (대괄호) : ser [index label] , df [**column name**]
- .loc (라벨 기준) : ser.loc [indexer], df.loc [row_indexer, column_indexer]
- .iloc (포지션 기준) : ser.iloc [indexer], df.iloc [row_indexer, column_indexer]

※ **DataFrame** 은 대괄호 인덱싱 '[]' 할 때 **Column Name** 을 넣고 결과는 **Series**

Ⅲ. 인덱싱

position 은 내부 정보
index 정보는 외부 노출

column position

column label(name)

row position

index label

DataFrame

		0	1	2	3
		C1	C2	C3	C4
0	A	0.510287	-0.137837	-0.942603	0.145588
1	B	-1.099963	-0.708024	-0.606779	-2.416373
2	C	-0.748353	2.768246	0.400762	-0.389147
3	D	-0.104503	0.542986	0.178355	0.125238
4	E	0.140347	1.042130	0.454405	0.144571
5	F	0.401344	-1.020710	-1.455291	1.370276
6	G	-0.065459	1.164725	-0.017151	-0.389824
7	H	0.273478	1.147542	-0.657081	1.131138

Ⅲ. 인덱싱

- [] (대괄호) 인덱싱

Series : position 과 label 둘 다 사용 가능, 슬라이싱 Range 사용 가능

예)

ser[5] : position, 결과값 Scalar

ser['A'] : index label, 결과값 Scalar

ser[5:10] : position slicing, 결과값 Series

DataFrame : position 사용 불가, Column Name 사용가능, 슬라이싱 Range 사용 가능

예)

df[5] : column index position, 사용불가

df['C1'] : column index label, 결과값 Series

df[5:10] : row position slicing, 결과값 DataFrame

Ⅲ. 인덱싱

- [] (대괄호) 인덱싱

	인자 종류	인덱싱 방향	가능 여부	결과 값
Series	position	행	o	Scalar
	label	행	o	Scalar
	slicing range	행	o	Series
	label list	행	o	Series
DataFrame	position	-	x	Series
	column name	열	o	Series
	slicing range	행	o	DataFrame
	column name list	열	o	DataFrame

Ⅲ. 인덱싱

.loc : label 기준
.iloc : position 기준

column position

column label(name)

		0	1	2	3
		c1	c2	c3	c4
0	A	0.510287	-0.137837	-0.942603	0.145588
1	B	-1.099963	-0.708024	-0.606779	-2.416373
2	C	-0.748353	2.768246	0.400762	-0.389147
3	D	-0.104503	0.542986	0.178355	0.125238
4	E	0.140347	1.042130	0.454405	0.144571
5	F	0.401344	-1.020710	-1.455291	1.370276
6	G	-0.065459	1.164725	-0.017151	-0.389824
7	H	0.273478	1.147542	-0.657081	1.131138

row position

index label

DataFrame

Ⅲ. 인덱싱

- .loc 인덱싱 : Selection by label

Series : index label 사용, slicing 도 index label 로 가능

예) ser.loc['b'] : index label, 결과값 Scalar
ser.loc['c':'d'] : index label slicing, 결과값 Series
ser.loc[['a','c','d']] : index label list, 결과값 Series

DataFrame : index label 사용, slicing 도 index label 로 가능

예) df.loc['20130102'] : row index label 사용, 결과값 Series
df.loc[:, 'C1'] : column index label 사용, 결과값 Series
df.loc['20130102', 'C1'] : row index and column index label 동시 사용, 결과값 Scalar
df.loc['20130102':'20130104', :] = df.loc['20130102':'20130104'] : row index slicing, 결과 값 DataFrame
df.loc[:, 'A':'D'] : column index slicing, 결과 값 DataFrame, ※ slicing 끝 label 도 포함
df.loc[:, ['A', 'C', 'D']] : column index list, 결과 값 DataFrame

Ⅲ. 인덱싱

- .iloc 인덱싱 : Selection by position

Series : position 사용, slicing 도 가능

예) ser.iloc[0] : position, 결과값 Scalar
ser.iloc[0:4] : position slicing, 결과값 Series
ser.iloc[[0, 1, 3]] : position list, 결과값 Series

DataFrame : position 사용, slicing 도 가능

예) df.iloc[100] : row position, 결과값 Series
df.iloc[:, 5] : column position, 결과값 Series
df.iloc[100, 5] : row and column position 동시 사용, 결과값 Scalar
df.iloc[100:200,:] = df.iloc[100:200] : row position slicing, 결과 값 DataFrame
df.iloc[:, 2:5] : column position slicing, 결과 값 DataFrame, ※ 0-th index 사용
df.iloc[[1,3,5]] : row position list, 결과 값 DataFrame
df.iloc[:, [0,3,4,5]] : column position list, 결과 값 DataFrame

Ⅲ. 인덱싱 - Advanced

.loc only supports labels

.iloc only supports positions

But, we often want a combination of label and position

Ⅲ. 인덱싱 - Advanced

.loc : only supports labels

- **row index label** : `df.index[]`
- **column index label** : `df.columns[]`

	row index label	column index label
Single	<code>df.index[3]</code>	<code>df.columns[3]</code>
Slicing	<code>df.index[1:4]</code>	<code>df.columns[1:4]</code>
Fancy	<code>df.index[[1,3,4]]</code>	<code>df.columns[[1,3,4]]</code>

Ⅲ. 인덱싱 - Advanced

.iloc : only supports positions

- row position

- column position

	row position	column position
Single	<code>df.index.get_loc('A')</code>	<code>df.columns.get_loc('C1')</code>
Slicing	<code>df.index.slice_indexer('A', 'D')</code>	<code>df.columns.slice_indexer('C1','C4')</code>
Fancy	<code>df.index.get_indexer(['A',B',D'])</code>	<code>df.columns.get_indexer(['C1','C3','C4'])</code>

Ⅲ. 인덱싱

- Boolean Indexing

Numpy masking과 유사한 기법

Series와 DataFrame 구분 없이 **차원 맞으면** 적용이 가능함

filtered_data = data[boolean_filter]

- DataFrame data에 Series filter 적용 (O)
- DataFrame data에 DataFrame filter 적용 (O)
- Series data에 Series filter 적용 (O)
- Series data에 DataFrame filter 적용 (**X**)

TypeError: Indexing a Series with DataFrame is not supported)

III. 인덱싱

- Boolean Indexing - 조건 생성 및 적용 결과

```
df = pd.DataFrame(np.random.rand(6, 6), #  
                  columns=['A', 'B', 'C', 'D', 'E', 'F'])  
df
```

	A	B	C	D	E	F
0	0.490002	0.236682	0.620430	0.713736	0.146052	0.980088
1	0.644666	0.902812	0.911763	0.278795	0.946430	0.131201
2	0.688285	0.093478	0.782924	0.602689	0.314115	0.269043
3	0.254622	0.536576	0.486748	0.674103	0.656383	0.407015
4	0.243396	0.703640	0.328009	0.955008	0.104341	0.235750
5	0.560185	0.478441	0.831670	0.232693	0.771186	0.604760

DataFrame 레벨

```
df > 0.5
```

	A	B	C	D	E	F
0	False	False	True	True	False	True
1	True	True	True	False	True	False
2	True	False	True	True	False	False
3	False	True	False	True	True	False
4	False	True	False	True	False	False
5	True	False	True	False	True	True

Ⅲ. 인덱싱

- Boolean Indexing - 조건 생성 및 적용 결과

DataFrame 레벨

```
df > 0.5
```

	A	B	C	D	E	F
0	False	False	True	True	False	True
1	True	True	True	False	True	False
2	True	False	True	True	False	False
3	False	True	False	True	True	False
4	False	True	False	True	False	False
5	True	False	True	False	True	True



```
df[df > 0.5]
```

	A	B	C	D	E	F
0	NaN	0.922102	NaN	NaN	0.732188	NaN
1	NaN	NaN	0.538316	0.549558	NaN	0.766860
2	0.664970	NaN	NaN	0.586038	NaN	0.892312
3	NaN	NaN	NaN	NaN	0.596678	0.645663
4	0.814416	NaN	0.917453	0.548274	NaN	0.565958
5	NaN	NaN	NaN	NaN	NaN	0.639728

III. 인덱싱

- Boolean Indexing - 조건 생성 및 적용 결과

```
df = pd.DataFrame(np.random.rand(6, 6), #  
                  columns=['A', 'B', 'C', 'D', 'E', 'F'])  
df
```

	A	B	C	D	E	F
0	0.490002	0.236682	0.620430	0.713736	0.146052	0.980088
1	0.644666	0.902812	0.911763	0.278795	0.946430	0.131201
2	0.688285	0.093478	0.782924	0.602689	0.314115	0.269043
3	0.254622	0.536576	0.486748	0.674103	0.656383	0.407015
4	0.243396	0.703640	0.328009	0.955008	0.104341	0.235750
5	0.560185	0.478441	0.831670	0.232693	0.771186	0.604760

Series (단일 컬럼) 레벨

```
df['A'] > 0.5
```

```
0    False  
1    False  
2     True  
3    False  
4     True  
5    False  
Name: A, dtype: bool
```

III. 인덱싱

- Boolean Indexing - 조건 생성 및 적용 결과

Series (단일 컬럼) 레벨

```
df['A'] > 0.5
```

```
0    False
1    False
2     True
3    False
4     True
5    False
Name: A, dtype: bool
```



```
df[df['A'] > 0.5]
```

	A	B	C	D	E	F
2	0.664970	0.013002	0.427867	0.586038	0.446110	0.892312
4	0.814416	0.429033	0.917453	0.548274	0.191362	0.565958

```
df['B'][df['A'] > 0.5]
```

```
2    0.013002
4    0.429033
Name: B, dtype: float64
```

Ⅲ. 인덱싱

- Boolean Indexing - 복합 조건 적용하기

조건 연산자

연산자	의미
	OR
&	AND
~	NOT

※ 반드시 조건별로 괄호를 사용할 것

Ⅲ. 인덱싱

- Boolean Indexing - 복합 조건 적용하기

	id	player	year	stint	team	lg	g	ab	r	h	...	rbi	sb	cs	bb	so	ibb	hbp	sh	sf	gidp
0	88641	womacto01	2006	2	CHN	NL	19	50	6	14	...	2.0	1.0	1.0	4	4.0	0.0	0.0	3.0	0.0	0.0
1	88643	schilcu01	2006	1	BOS	AL	31	2	0	1	...	0.0	0.0	0.0	0	1.0	0.0	0.0	0.0	0.0	0.0
2	88645	myersmi01	2006	1	NYA	AL	62	0	0	0	...	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0
3	88649	helliri01	2006	1	MIL	NL	20	3	0	0	...	0.0	0.0	0.0	0	2.0	0.0	0.0	0.0	0.0	0.0
4	88650	johnsra05	2006	1	NYA	AL	33	6	0	1	...	0.0	0.0	0.0	0	4.0	0.0	0.0	0.0	0.0	0.0
...
95	89525	benitar01	2007	2	FLO	NL	34	0	0	0	...	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0
96	89526	benitar01	2007	1	SFN	NL	19	0	0	0	...	0.0	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0
97	89530	ausmubr01	2007	1	HOU	NL	117	349	38	82	...	25.0	6.0	1.0	37	74.0	3.0	6.0	4.0	1.0	11.0
98	89533	aloumo01	2007	1	NYN	NL	87	328	51	112	...	49.0	3.0	0.0	27	30.0	5.0	2.0	0.0	3.0	13.0
99	89534	alomasa02	2007	1	NYN	NL	8	22	1	3	...	0.0	0.0	0.0	0	3.0	0.0	0.0	0.0	0.0	0.0

III. 인덱싱

- Boolean Indexing - 복합 조건 적용하기

```
# 타석(ab)에 100번 이상 들어서고,  
# 안타를 100개 이상 친 타자만  
# (ab>=100) & (h>=100)  
(df['ab']>=100) & (df['h']>=100)
```

```
0    False  
1    False  
2    False  
3    False  
4    False  
...  
95   False  
96   False  
97   False  
98    True  
99   False  
Length: 100, dtype: bool
```



```
df[(df['ab']>=100) & (df['h']>=100)]#  
[['id', 'player', 'year', 'team', 'ab', 'h']].head()
```

	id	player	year	team	ab	h
5	88652	finlest01	2006	SFN	426	105
6	88653	gonzalu01	2006	ARI	586	159
22	89347	vizquom01	2007	SFN	513	126
28	89360	thomeji01	2007	CHA	432	119
29	89361	thomafr04	2007	TOR	531	147

III. 인덱싱

- Boolean Indexing - 복합 조건 적용하기

```
# Boston RedSocks('BOS') 팀에서  
# 20홈런('hr') 이상을 친 선수  
(df['team']=='BOS') & (df['hr']>=20)
```

```
0      False  
1      False  
2      False  
3      False  
4      False  
...  
95     False  
96     False  
97     False  
98     False  
99     False  
Length: 100, dtype: bool
```



```
df[(df['team']=='BOS') & (df['hr']>=20)]#  
[['id', 'player', 'year', 'team', 'ab', 'h', 'X2b', 'hr']]
```

	id	player	year	team	ab	h	X2b	hr
48	89396	ramirma02	2007	BOS	483	143	33	20

IV. 문자열 다루기 & 데이터 합치기

- 문자열 데이터 다루기
- 데이터 합치기

IV. 문자열 다루기 & 데이터 합치기

- 문자열 데이터 다루기

- Series/DataFrame을 별도의 타입 표기 없이 생성하면 dtype=object로 설정됨
- 타입을 명시적으로 설정하기 위해서는 아래 방법 사용
 1. `s2 = pd.Series(["A", "B","dog", "cat"], dtype='string')`
 2. `s2 = s1.astype('string')`

* dtype를 string으로 명시하지 않아도 데이터를 다루는 데에는 크게 문제 없음!

IV. 문자열 다루기 & 데이터 합치기

- 스트링 접근자
 - string으로 된 Series를 조작할 때 스트링 접근자(**.str** accessor)를 사용함
 - 해당 접근자는 Series에 속하는 멤버로 DataFrame 전체에 적용할 수 없음
 - 필요하다면 특정 컬럼 추출해서 적용

```
df = pd.DataFrame([["A", "B", "C", "Aaba"], ["A", "B", "C", "Aaba"]])
df.str.len()
```

```
-----
AttributeError                                Traceback (most recent call last)
/var/folders/cy/pnx3rqp534d0ghwqtxn7lb2w0000gn/T/ipykernel_35663/1939693288.py in ?()
      1 df = pd.DataFrame([["A", "B", "C", "Aaba"], ["A", "B", "C", "Aaba"]])
----> 2 df.str.len()

~/anaconda3/envs/ita/lib/python3.11/site-packages/pandas/core/generic.py in ?(self, name)
    6295         and name not in self._accessors
    6296         and self._info_axis._can_hold_identifiers_and_holds_name(name)
    6297     ):
    6298         return self[name]
-> 6299     return object.__getattr__(self, name)

AttributeError: 'DataFrame' object has no attribute 'str'
```

IV. 문자열 다루기 & 데이터 합치기

- 스트링 접근자
 - python에서 제공하는 string 관련 함수들 지원
 - 대표적인 예시로
 - **str.split**: 특정 키워드를 기준 삼아 리스트로 분할 (예) '@'를 기준으로 이메일 주소 분할
 - **str.contains**: 특정 키워드를 포함하는지 확인 > Boolean Indexing에 활용
 - **str.removeprefix**: 반복되는 접두어를 한 번에 제거
 - **str.count**: 특정 패턴이 얼마나 반복되는지 count
 - **str.len**: 문자열의 길이를 반환

IV. 문자열 다루기 & 데이터 합치기

- Series 문자열 데이터 다루기

```
names = pd.Series(['Graham Chapman', 'John Cleese', 'Terry Gilliam',  
                  'Eric Idle', 'Terry Jones', 'Michael Palin'])
```

`names.str.len()`

```
0    14  
1    11  
2    13  
3     9  
4    11  
5    14  
dtype: int64
```

`names.str.lower()`

```
0    graham chapman  
1    john cleese  
2    terry gilliam  
3    eric idle  
4    terry jones  
5    michael palin  
dtype: object
```

`names.str.strip()`

```
0    Graham Chapman  
1    John Cleese  
2    Terry Gilliam  
3    Eric Idle  
4    Terry Jones  
5    Michael Palin  
dtype: object
```

`names.str.replace('Eric', 'Tom')`

```
0    Graham Chapman  
1    John Cleese  
2    Terry Gilliam  
3    Tom Idle  
4    Terry Jones  
5    Michael Palin  
dtype: object
```

IV. 문자열 다루기 & 데이터 합치기

- Series 문자열 데이터 다루기

```
names = pd.Series(['Graham Chapman', 'John Cleese', 'Terry Gilliam',  
                  'Eric Idle', 'Terry Jones', 'Michael Palin'])
```

※ split() 메서드

Split 한 첫번째 요소를
Series 객체로 반환하고 싶다면

```
names.str.split(' ')
```

```
0    [Graham, Chapman]  
1    [John, Cleese]  
2    [Terry, Gilliam]  
3    [Eric, Idle]  
4    [Terry, Jones]  
5    [Michael, Palin]  
dtype: object
```

```
names.str.split(' ').str[0]
```

```
0    Graham  
1    John  
2    Terry  
3    Eric  
4    Terry  
5    Michael  
dtype: object
```

Split 한 마지막 요소를
Series 객체로 반환하고 싶다면

```
names.str.split(' ').str[-1]
```

```
0    Chapman  
1    Cleese  
2    Gilliam  
3    Idle  
4    Jones  
5    Palin  
dtype: object
```

IV. 문자열 다루기 & 데이터 합치기

- DataFrame 문자열 데이터 다루기

df (DataFrame)

	상영관	영화제목	상영시간	출연
0	1관	기생충	120분	송강호,이선균,조여정,
1	2관	미나리	115분	스티븐 연,윤여정,한예리
2	3관	오징어게임	200min.	이정재,박해수,오영수
3	4관	파묘	95분	최민식,유해진,김고은

- '상영시간' 에서 '분' 이나 'min.' 을 제거
- '출연' 을 ',' 으로 3개 값으로 분리

IV. 문자열 다루기 & 데이터 합치기

- DataFrame 문자열 데이터 다루기

- '상영시간' 에서 '분' 이나 'min.' 을 제거
- '출연' 을 ',' 으로 3개 값으로 분리

df (DataFrame)

	상영관	영화제목	상영시간	출연
0	1관	기생충	120분	송강호,이선균,조여정,
1	2관	미나리	115분	스티븐 연,윤여정,한예리
2	3관	오징어게임	200min.	이정재,박해수,오영수
3	4관	파묘	95분	최민식,유해진,김고은



	상영관	영화제목	상영시간	출연1	출연2	출연3
0	1관	기생충	120	송강호	이선균	조여정
1	2관	미나리	115	스티븐 연	윤여정	한예리
2	3관	오징어게임	200	이정재	박해수	오영수
3	4관	파묘	95	최민식	유해진	김고은

IV. 문자열 다루기 & 데이터 합치기

- 문자열 데이터 다루기
 - 스페이스만 있거나 공백 문자열을 결측치로 다루기

ex) ' ' or ''

② 빈 문자열("") 을 NaN 으로 변경

```
str.strip().replace("", np.nan)
```

① 문자열의 공백 없애기

IV. 문자열 다루기 & 데이터 합치기

▪ 데이터 합치기: `pd.merge` (JOIN)

- 공통되는 값을 기준으로 두 데이터를 병합
- **기준값**을 매개변수로 지정할 수 있음
 - 양쪽 df에 같은 이름의 컬럼이 있다면 `on`
 - 각기 다른 이름의 컬럼을 지정할 때는 `left_on`, `right_on`
 - 공통 column 대신 `index`를 기준으로 지정하고자 할 때는 `left_index=True`, `right_index=True`
- 기준값을 별도 명시하지 않으면 자동으로 같은 이름의 column을 감지

IV. 문자열 다루기 & 데이터 합치기

▪ 데이터 합치기: pd.merge (JOIN)

- 병합 방식을 how 매개변수로 지정
 - inner: **INNER JOIN**, merge 함수의 디폴트 설정
 - outer / left / right : **OUTER JOIN**
 - cross: **Cross JOIN**, Cartesian Product

※ 양쪽 DataFrame 의 key 값이 다를 경우
left_on, right_on 사용 가능

adf		bdf			
x1	x2	x1	x3		
A	1	A	T		
B	2	B	F		
C	3	D	T		

+

=

Standard Joins

x1	x2	x3	pd.merge(adf, bdf, how='left', on='x1')
A	1	T	Join matching rows from bdf to adf.
B	2	F	
C	3	NaN	

x1	x2	x3	pd.merge(adf, bdf, how='right', on='x1')
A	1.0	T	Join matching rows from adf to bdf.
B	2.0	F	
D	NaN	T	

x1	x2	x3	pd.merge(adf, bdf, how='inner', on='x1')
A	1	T	Join data. Retain only rows in both sets.
B	2	F	

x1	x2	x3	pd.merge(adf, bdf, how='outer', on='x1')
A	1	T	Join data. Retain all values, all rows.
B	2	F	
C	3	NaN	
D	NaN	T	

IV. 문자열 다루기 & 데이터 합치기

- 데이터 합치기: pd.merge (JOIN)

TABLE_A

NAME	Quantity
A	10
C	20
D	30
F	40
G	50
H	60

TABLE_B

NAME	COL1	COL2
B	123	TES
C	124	TES
D	456	ERT
E	789	FDK
F	101	VKV

- ① A table 들의 모든 행을 배치합니다.
- ② ON 조건을 만족하는 B table 을 오른쪽에 가져와 붙입니다.
- ③ 조건을 만족하지 않는 B table 행들은 NULL 값이 됩니다.

LEFT JOIN

LEFT (OUTER) JOIN
ON A.NAME = B.NAME

NAME	VALUE	NAME	COL1	COL2
A	10	NULL	NULL	NULL
C	20	C	123	TES
D	30	D	456	ERT
F	40	F	101	VKV
G	50	NULL	NULL	NULL
H	60	NULL	NULL	NULL

IV. 문자열 다루기 & 데이터 합치기

- 데이터 합치기: pd.merge (JOIN)

TABLE_A

NAME	Quantity
A	10
C	20
D	30
F	40
G	50
H	60

TABLE_B

NAME	COL1	COL2
B	123	TES
C	124	TES
D	456	ERT
E	789	FDK
F	101	VKV

INNER JOIN

INNER JOIN

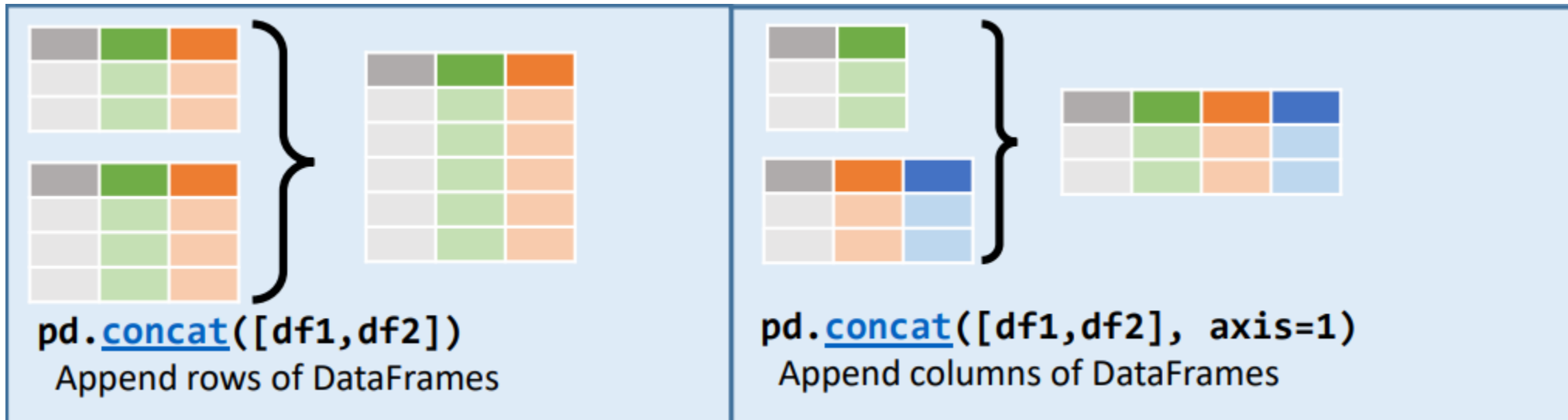
ON A.NAME = B.NAME

NAME	VALUE	NAME	COL1	COL2
C	30	C	123	TES
D	40	D	456	ERT
F	60	F	101	VKV

IV. 문자열 다루기 & 데이터 합치기

▪ 데이터 합치기: `pd.concat`

- 가로(`axis=0`) 혹은 세로(`axis=1`) 로 두 DataFrame 이어 붙이기
- Default 는 `axis=0`
- 기존 index 값을 버리고 새롭게 설정하고자 할 경우, `ignore_index=True`



V. 집계 연산 및 함수

- Group by
- Apply & Map

V. 집계 연산 및 함수

- 집계 연산 (Aggregation)

- sum(), min(), max() 같은 집계연산은 판다스 Series 나 DataFrame 에 적용 가능

```
ser = pd.Series(rng.rand(5))  
ser
```

```
0    0.155995  
1    0.058084  
2    0.866176  
3    0.601115  
4    0.708073  
dtype: float64
```

```
ser.sum()
```

```
2.389441867818592
```

```
ser.mean()
```

```
0.4778883735637184
```

```
df = pd.DataFrame({'A': rng.rand(5),  
                   'B': rng.rand(5)})  
df
```

	A	B
0	0.020584	0.183405
1	0.969910	0.304242
2	0.832443	0.524756
3	0.212339	0.431945
4	0.181825	0.291229

```
df.mean()
```

```
A    0.443420  
B    0.347115  
dtype: float64
```


V. 집계 연산 및 함수

▪ Group By

- DataFrame 의 data 를 groupby 하면 *DataFrameGroupBy* 객체가 됨.
- *DataFrameGroupBy* 객체는 컬럼 인덱싱을 지원하며,
인덱싱 결과는 *DataFrameGroupBy* (복수 컬럼 선택) 또는 *SeriesGroupBy* (단일 컬럼 선택) 객체가 됨.

scores (DataFrame)

	학년	성별	영어 성적	수학 성적
0	1학년	남	80	88
1	1학년	여	90	92
2	2학년	남	70	65
3	2학년	여	60	63
4	3학년	남	85	80

```
score.groupby('학년')
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002B0E4F86DF0>
```

'학년'으로 그룹화 된 객체를 '영어 성적'으로 인덱싱하면
Series 그룹이 반환된다. 하지만 아직까지 어떤 Aggregation 연산도 수행되지 않았다.

```
score.groupby('학년')['영어 성적']
```

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x000002B0E4F86760>
```

V. 집계 연산 및 함수

▪ Group By

- DataFrame 의 data 를 groupby 하면 *DataFrameGroupBy* 객체가 됨.
- *DataFrameGroupBy* 객체는 컬럼 인덱싱을 지원하며,
인덱싱 결과는 *DataFrameGroupBy* (복수 컬럼 선택) 또는 *SeriesGroupBy* (단일 컬럼 선택) 객체가 됨.
- *DataFrameGroupBy* 또는 *SeriesGroupBy* 객체는 Aggregation 을 수행할 수 있다.

scores (DataFrame)

	학년	성별	영어 성적	수학 성적
0	1학년	남	80	88
1	1학년	여	90	92
2	2학년	남	70	65
3	2학년	여	60	63
4	3학년	남	85	80

'학년'으로 그룹화 -> '영어 성적' 컬럼 특정화 -> median() 으로 aggregation 수행

```
score.groupby('학년')['영어 성적'].median()
```

```
학년
1학년    85
2학년    65
3학년    85
Name: 영어 성적, dtype: int64
```

V. 집계 연산 및 함수

- Group By

- GroupBy 객체는 그룹별로 순환하면서 Series 객체나 DataFrame 객체를 반환한다.

scores (DataFrame)

	학년	성별	영어 성적	수학 성적
0	1학년	남	80	88
1	1학년	여	90	92
2	2학년	남	70	65
3	2학년	여	60	63
4	3학년	남	85	80

```
for (grade, group) in score.groupby('학년'):
    print("{0:5s}, shape={1}, type={2}".format(grade, group.shape, type(group)))
```

```
1학년   , shape=(2, 4), type=<class 'pandas.core.frame.DataFrame'>
2학년   , shape=(2, 4), type=<class 'pandas.core.frame.DataFrame'>
3학년   , shape=(3, 4), type=<class 'pandas.core.frame.DataFrame'>
```

```
for (grade, group) in score.groupby('학년'):
    print("{0} group : \n{1}\n".format(grade, group))
```

```
1학년 group :
   학년 성별  영어 성적  수학 성적
0  1학년  남     80     88
1  1학년  여     90     92

2학년 group :
   학년 성별  영어 성적  수학 성적
2  2학년  남     70     65
3  2학년  여     60     63

3학년 group :
   학년 성별  영어 성적  수학 성적
4  3학년  남     85     80
5  3학년  여     95     75
6  3학년  여     75     92
```

V. 집계 연산 및 함수

- Group By
 - 단일 컬럼 Aggregation

scores (DataFrame)

	학년	성별	영어 성적	수학 성적
0	1학년	남	80	88
1	1학년	여	90	92
2	2학년	남	70	65
3	2학년	여	60	63
4	3학년	남	85	80

```
score.groupby('학년').aggregate(['min', np.median, max])
```

	영어 성적			수학 성적		
	min	median	max	min	median	max
학년						
1학년	80	85	90	88	90	92
2학년	60	65	70	63	64	65
3학년	75	85	95	75	80	92

V. 집계 연산 및 함수

- Group By
 - 복수 컬럼 Aggregation

scores (DataFrame)

	학년	성별	영어 성적	수학 성적
0	1학년	남	80	88
1	1학년	여	90	92
2	2학년	남	70	65
3	2학년	여	60	63
4	3학년	남	85	80

```
score.groupby('학년').aggregate({'영어 성적': 'min',  
                                '수학 성적': 'max'})
```

	영어 성적	수학 성적
학년		
1학년	80	92
2학년	60	65
3학년	75	92

V. 집계 연산 및 함수

- Group By
 - 복수 컬럼 Aggregation

scores (DataFrame)

	학년	성별	영어 성적	수학 성적
0	1학년	남	80	88
1	1학년	여	90	92
2	2학년	남	70	65
3	2학년	여	60	63
4	3학년	남	85	80

```
score.groupby('학년')[['영어 성적', '수학 성적']].aggregate(['min', 'max', 'mean'])
```

	영어 성적			수학 성적		
	min	max	mean	min	max	mean
학년						
1학년	80	90	85	88	92	90.000000
2학년	60	70	65	63	65	64.000000
3학년	75	95	85	75	92	82.333333

V. 집계 연산 및 함수

- Apply

- DataFrame 의 특정 축(axis) 를 따라 함수를 적용
- Numpy 함수, Pandas 함수 적용 가능
- 사용자 정의 함수도 가능

df

yyyy-mm-dd

0	2015-12-29
1	1996-08-21
2	1989-08-18

```
# 년도만 추출하는 함수 정의
```

```
def extract_year(column):  
    return column.split("-")[0]
```

```
# apply 수행
```

```
df['year'] = df['yyyy-mm-dd'].apply(extract_year)  
df
```

yyyy-mm-dd year

0	2015-12-29	2015
1	1996-08-21	1996
2	1989-08-18	1989

V. 집계 연산 및 함수

▪ Apply

- DataFrame 의 특정 축(axis) 를 따라 함수를 적용
- Numpy 함수, Pandas 함수 적용 가능
- 사용자 정의 함수도 가능

df

	yyyy-mm-dd	year
0	2015-12-29	2015
1	1996-08-21	1996
2	1989-08-18	1989

```
# 년도로부터 나이를 얻는 함수 정의
def get_age(year, current_year):
    return current_year - int(year) + 1
```

```
df['age'] = df['year'].apply(get_age, current_year = 2021)
df
```

	yyyy-mm-dd	year	age
0	2015-12-29	2015	7
1	1996-08-21	1996	26
2	1989-08-18	1989	33

V. 집계 연산 및 함수

- Apply

- DataFrame 의 특정 축(axis) 를 따라 함수를 적용
- Numpy 함수, Pandas 함수 적용 가능
- 사용자 정의 함수도 가능

df

	yyyy-mm-dd	year	age
0	2015-12-29	2015	7
1	1996-08-21	1996	26
2	1989-08-18	1989	33

```
def get_introduce(row):  
    return "I was born " + str(row.year) + ", my age is " + str(row.age)
```

```
df['introduce'] = df.apply(get_introduce, axis=1)  
df
```

	yyyy-mm-dd	year	age	introduce
0	2015-12-29	2015	7	I was born 2015, my age is 7
1	1996-08-21	1996	26	I was born 1996, my age is 26
2	1989-08-18	1989	33	I was born 1989, my age is 33

V. 집계 연산 및 함수

- Map

- apply 와 같이 컬럼을 함수를 통해 변경
- 딕셔너리를 직접 전달해 원하는 값으로 변환

df

	date
0	1989-08-18
1	2015-12-29
2	1996-08-21

```
def extract_year(date):  
    return date.split('-')[0]
```

```
df['year'] = df['date'].map(extract_year)  
df
```

	date	year
0	1989-08-18	1989
1	2015-12-29	2015
2	1996-08-21	1996

V. 집계 연산 및 함수

- Map

- apply 와 같이 컬럼을 함수를 통해 변경
- 딕셔너리를 직접 전달해 원하는 값으로 변환

df

	age	job
0	20	student
1	32	developer
2	23	student
3	35	teacher

```
df['job_number'] = df['job'].map({'student' : 1, 'developer' : 2, 'teacher' : 3 })  
df
```

	age	job	job_number
0	20	student	1
1	32	developer	2
2	23	student	1
3	35	teacher	3