판다스

파이썬에서 데이터를 조작/분석하기 위한 라이브러리

- 행/열로 이루어진 테이블 형태의 데이터를 다룬다.
- 엑셀로 다루지 못하는 대용량 데이터를 다룰 수 있다.
- **Matplotlib, Seaborn** 등 다양한 시각화 도구와 함께 사용할 수 있다.

판다스에서는 **Series**와 **DataFrame**으로 데이터를 만들 수 있다.

1. Pandas 라이브러리 import

- **pandas**는 외부 패키지이지만 아나콘다에 기본적으로 설치되어 제공되기 때문에 별도 설치할 필요가 없다.
- **pandas**는 관용적으로 **pd**라는 별칭을 사용한다.

In [1]:

```
import pandas as pd
```

2. 데이터프레임 만들기

- 데이터프레임 **:** 엑셀의 시트**(sheet)**와 동일한 개념이다.

DataFrame : 행 **/** 열로 만들어진 **2**차원 표 형태

- 행**(row) : index** 로 구분
- 열**(series, column) : column_name**으로 구분

2.1 리스트로 만들기

`pd.DataFrame(2차원 리스트, columns=컬럼 리스트, index=인덱스 리스트)`

- 행 단위로 데이터프레임이 만들어진다.
- 컬럼, 인덱스를 지정하지 않으면 디폴트로부터 **0**으로 시작하는 숫자가 지정된다.

In [9]:

```
list = [
    ['james', 30, 'programmer'],
    ['amy', 20, 'student'],
    ['david', 25, 'designer']
]
df1 = pd.DataFrame(list, columns = ['name', 'age', 'job'], index=['a', 'b', 'c'])
df1
```

Out[9]:

	name	age	job
a	james	30	programmer
b	amy	20	student
c	david	25	designer

2.2 딕셔너리로 만들기

`pd.DataFrame(딕셔너리, index=인덱스 리스트)`

- 딕셔너리는 **{컬렴명1 : 컬럼리스트, 컬럼명2 : 컬럼값리스트 ,,, }**
- 컬럼 단위로 데이터프레임이 생성된다.
- 딕셔너리의 키가 컬럼명이 된다.
- 인덱스를 지정하지 않으면 디폴트로 **0**부터 시작하는 숫자가 지정된다.

In [11]:

```
dic = {
    'name' : ['james', 'amy', 'david'],
    'age' : [30, 20, 25],
    'job' : ['programmer', 'student', 'designer']
}
df2 = pd.DataFrame(dic, index=['name', 'age', 'job'])
```

```
df2
```

Out[11]:

	name	age	job
name	james	30	programmer
age	amy	20	student
job	david	25	designer

2.3 csv 파일에서 데이터를 읽어와 만들기

`pd.read_csv('파일 경로 및 파일명')`

- **UTF-8** 형식의 **csv** 파일을 준비해야 한다 **:** 예시 **) ./data/scores.csv**
- 메모장에 정리한 후**, UTF-8** 형식의 인코딩 방식으로 저장한다**.**
- **csv** 파일은 무조건 **utf-8** 형식이어야 한다**.**

In [13]:

```
score = pd.read_csv('./data/scores.csv')
score
```

Out[13]:

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
4	Henry	NaN	35.0	60.0
5	Ian	90.0	100.0	90.0
6	James	70.0	75.0	65.0
7	Julian	80.0	90.0	55.0
8	Justin	50.0	60.0	100.0
9	Kevin	100.0	100.0	90.0

	name	kor	eng	math
10	Lee	90.0	95.0	70.0
11	Oliver	70.0	75.0	65.0
12	Peter	100.0	95.0	100.0
13	Amy	90.0	75.0	90.0
14	Chloe	95.0	100.0	95.0
15	Danna	100.0	100.0	100.0
16	Ellen	NaN	60.0	NaN
17	Emma	70.0	65.0	70.0
18	Jennifer	80.0	55.0	80.0
19	Kate	50.0	NaN	50.0
20	Linda	100.0	90.0	100.0
21	Olivia	90.0	70.0	90.0
22	Rose	70.0	65.0	70.0
23	Sofia	100.0	100.0	100.0
24	Tiffany	90.0	NaN	90.0
25	Vanessa	95.0	70.0	95.0
26	Viviana	100.0	80.0	100.0
27	Vikkie	NaN	50.0	100.0
28	Winnie	70.0	100.0	70.0
29	Zuly	80.0	90.0	95.0

3. 데이터 미리보기

3.1 가장 앞의 n행 보기

데이터프레임.head(n)
시리즈.head(n)

- n을 생략하면 5개의 행을 출력한다.

In [15]:

```
# score = pd.read_csv('./data/scores.csv')
n = 3
score.head(n)
# score.head()
```

Out[15]:

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
1	Charles	90.0	80.0	75.0
2	Danial	95.0	100.0	100.0

3.2 가장 뒤의 n행 보기

데이터프레임.tail(n)

- n을 생략하면 5개의 행을 보여준다.

In [16]:

In [16]:

```
score.tail(n)
```

Out[16]:

	name	kor	eng	math
27	Vikkie	NaN	50.0	100.0
28	Winnie	70.0	100.0	70.0
29	Zuly	80.0	90.0	95.0

3.3 랜덤 n개 데이터 보기

`데이터프레임.sample(n)`

- **n**을 생략하면 **1**개의 샘플을 보여준다.

In [17]:

```
score.sample(n)
```

Out[17]:

	name	kor	eng	math
0	Aiden	100.0	90.0	95.0
15	Danna	100.0	100.0	100.0
2	Danial	95.0	100.0	100.0

랜덤 샘플 비율로 보기 : `데이터프레임.sample(frac=0.2)`

- 지정한 비율의 샘플을 보여준다.

In [18]:

```
score.sample(frac=0.2)
```

Out[18]:

	name	kor	eng	math
25	Vanessa	95.0	70.0	95.0
12	Peter	100.0	95.0	100.0
15	Danna	100.0	100.0	100.0
16	Ellen	NaN	60.0	NaN
5	Ian	90.0	100.0	90.0
1	Charles	90.0	80.0	75.0

3.4 높은 순/낮은 순 보기

`데이터프레임.nlargest(갯수, 컬럼명)`

- 컬럼의 데이터가 숫자형일 때 사용할 수 있다.

In [19]:

```
score.nlargest(5, 'eng')
```

Out[19]:

	name	kor	eng	math
2	Danial	95.0	100.0	100.0
3	Evan	100.0	100.0	100.0
5	Ian	90.0	100.0	90.0
9	Kevin	100.0	100.0	90.0
14	Chloe	95.0	100.0	95.0

낮은 순으로 보기:
`데이터프레임.nsmallest(갯수, 컬럼명)`

- 컬럼의 데이터가 숫자형일 때 사용할 수 있다.

In [20]:

```
# eng 낮은 순으로 보기
score.nsmallest(5, 'eng')
```

Out[20]:

	name	kor	eng	math
4	Henry	NaN	35.0	60.0
27	Vikkie	NaN	50.0	100.0
18	Jennifer	80.0	55.0	80.0
8	Justin	50.0	60.0	100.0
16	Ellen	NaN	60.0	NaN

4. 데이터 요약 보기

4.1 (행, 열)의 크기 보기

`df.shape`

In [21]:

```
score.shape
```

Out[21]:

```
(30, 4)
```

4.2 데이터의 개수 보기

`len(df)`

In [22]:

```
len(score)
```

Out[22]:

```
30
```

4.3 컬럼명 보기

`df.columns`

- 데이터프레임의 열 이름을 확인한다.

- **csv**의 첫 행에 컬럼명으로 들어갈 내용들이 들어가야 한다.

In [23]:

```
score.columns
```

Out[23]:

```
Index(['name', 'kor', 'eng', 'math'], dtype='object')
```

4.4 인덱스 보기

`df.index`

- 데이터프레임의 인덱스를 확인한다.

In [25]:

```
score.index
```

Out[25]:

```
RangeIndex(start=0, stop=30, step=1)
```

4.5 데이터의 자료형 보기

`df.dtypes`

- 데이터프레임을 구성하는 값의 자료형을 확인한다.
- 판다스에서는 ***문자열의 데이터타입이 object***이다.

In [26]:

```
score.dtypes
```

Out[26]:

```
name     object
kor     float64
eng     float64
math    float64
dtype: object
```

4.6 데이터프레임 정보 보기

`df.info()`

- 데이터프레임의 총 샘플 개수, 컬럼 수, 컬럼 별 정보 등을 확인한다.

In [27]:

```
score.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   name    30 non-null     object 
 1   kor     27 non-null     float64
 2   eng     28 non-null     float64
 3   math    29 non-null     float64
dtypes: float64(3), object(1)
memory usage: 1.1+ KB
```

위의 결과에서

- **rangeIndex :** 인덱스의 총 범위 = **entries(** 시작부터 끝까지 **)**
  - **0 to 29 : 0**부터 **29**까지이므로 총 범위가 **30**개
- **columns :** 총 열의 개수 **(** 각 항목의 개수 **)**

4.7 컬럼의 유니크한 정보 보기

`df['column'].unique()`

In [29]:

```
score['eng'].unique()
```

Out[29]:

```
array([ 90.,  80., 100.,  35.,  75.,  60.,  95.,  65.,  55.,  nan,  70.,
        50.])
```

4.8 컬럼의 유니크한 값의 개수 보기

`df['column'].value_counts()`

In [34]:

```
score['kor'].value_counts()
```

Out[34]:

```
100.0    8
90.0     6
70.0     5
95.0     3
80.0     3
50.0     2
Name: kor, dtype: int64
```

4.9 요약통계 보기

`df.describe()`

In [35]:

```
score.describe()
```

Out[35]:

	kor	eng	math
count	27.000000	28.000000	29.000000
mean	85.740741	80.892857	84.827586
std	15.045042	18.361270	15.950972
min	50.000000	35.000000	50.000000
25%	75.000000	68.750000	70.000000
50%	90.000000	85.000000	90.000000
75%	100.000000	100.000000	100.000000
max	100.000000	100.000000	100.000000

In [36]:

```
score['kor'].mean()
```

Out[36]:

Out[37]:

85.74074074074075

In [38]:

```
score['kor'].std()
```

Out[38]:

15.045041586957357