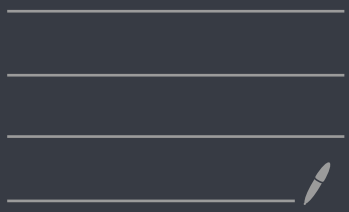
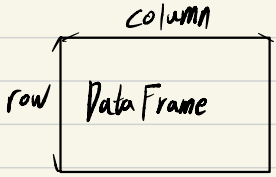


Pandas: DataFrame 객체를 다루기 위한 패키지

DataFrame

DataFrame: row와 column으로 이루어진 객체



row(관측치): 하나의 구성요소에 대한 데이터
column(변수): 하나의 특성에 대한 데이터

각각을 지칭하는 index를 갖고있다. column을 나타내는 문자로된 index → 열 이름

정의

```
DataFrame_Name = pd.DataFrame({ Key : Value , .... })
```

↳ Dictionary 형식

Key → Column 명, Value → 각 row에 저장되는 값

ex)
```
df = pd.DataFrame(
    { 'name' : ['철수', '영희', '지원'],
      'Age' : [20, 15, 23],
      'sex' : ['M', 'F', 'M'] } )
```

	name	Age	sex
0	철수	20	M
1	영희	15	F
2	지원	23	M

df

Type

Type(df) → pandas.core.frame.DataFrame

↳ DataFrame 객체이다.

Slicing & indexing

Column slicing

- 데이터프레임 [변수명] 으로 slicing한다.

ex) df["name"] ⇒

0	철수
1	영희
2	지원

df.name으로 가능

df[["name", "sex"]] ⇒

0	철수	M
1	영희	F
2	지원	M

row slicing

- index로 슬라이싱

ex) df[1:2] ⇒

	name	Age	sex
1	영희	15	F

- Boolean vector로 슬라이싱

ex) df["Age"] > 19 ⇒

0	True
1	False
2	True

df[df["Age"] > 19] ⇒

0	철수	20	M
2	지원	23	M

df[(df["Age"] == 23) & (df["sex"] = "M")]

⇒ 2 지원 23 M

row와 column을 동시에 indexing & slicing

loc method : boolean 및 이름을 매개변수로 사용
iloc method : index number를 매개변수로 사용

중괄호를 사용한다!

데이터프레임.loc[row 조건 or 이름 , column 이름]
.iloc[row index , column index]

ex) df.loc[(df["Age"] > 19), "name"]

⇒ 0 철수
2 지원

df.iloc[0,0] ⇒ 철수
df.iloc[:,0] ⇒ 0 철수
1 영희
2 지원

Series

DataFrame을 slicing한 결과는 Series 객체로 반환된다. ★
↳ indexing은 그냥 그 요소의 type으로 반환

ex) type(df.iloc[0,0]) ⇒ str

DataFrame의 메서드 != Series의 메서드

Array VS DataFrame

Vector : 1차원 array, Series
Matrix : 2차원 array, DataFrame

Array VS DataFrame
1. method가 다르다.
2. Array : 연산을 위한 것, DataFrame : 데이터를 다루기 위한 것
↳ Type 밑김

외부데이터 읽기

Working directory

```
import os
os.getcwd()
os.chdir("../")
```

⇒ current working directory의 path를 반환

⇒ 부모 directory로 이동

데이터 읽기

```
pd.read_excel('path')
```

⇒ 데이터프레임으로 반환한다.

ex) df = pd.read_excel("삼성전자_데이터.xlsx")

→ colums: stock, date, 종가, 거래량

df.head() ⇒ 상위 5개 row 보여줌

df.tail() ⇒ 하위 5개 row 보여줌

df.info() ⇒ 각각의 column의 결측치, type 보여줌

정리가 필요한 데이터

위의 데이터는 정리가 되어있는 dataset이다.

kospi = pd.read_excel("코스피.xlsx")

	Unnamed: 0	Last Refresh: 2023-08-07 15:20:36	Unnamed: 2
0	Calendar Basis	NaN	NaN
1	Portfolio		Default
2	Item		
3	Frequency	일간	KRW
4	Non-Trading Day	Exclusive	Asc
5	Include Weekend	ALL	
6	Term	20120101	20230131
7	Symbol	I.001	I.101
8	Symbol Name	코스피	코스피 200
9	Kind	IDX	IDX
10	Item	I31000040F	I31000040F
11	Item Name	종가지수(포인트)	종가지수(포인트)
12	Frequency	DAILY	DAILY
13	2012-01-02 00:00:00	1826.37	238.7
14	2012-01-03 00:00:00	1875.41	245.82
15	2012-01-04 00:00:00	1866.22	244.24

⇒ Type 지정 어려움

→ 원하는 데이터는 14번째 부터

kospi.info() ⇒ type이 모두 object로 보여진다.
↳ 문자형

⚠ info 메서드는 0번째 row는 무시, 1부터 읽어가며 타입을 설정한다.

kospi DataFrame은 타입이 인식하지 않다. ⇒ 문자열(object)로 인식

원하는 데이터 slicing

skiprows 와 header 파라미터
↳ 해당 row를 column 명으로 한다.
↳ row를 skip한다.

⚠ skiprows 와 header를 동시에 파라미터로 지정 시,
skiprows 가 우선 먼저 지정된다.

ex) pd.read_excel("코스피.xlsx", skiprows = 13, header = 9)

	2012-01-12 00:00:00	1864.57	243.46
0	2012-01-13	1875.68	245.28
1	2012-01-16	1859.27	243.10
2	2012-01-17	1892.74	247.81
3	2012-01-18	1892.39	247.82
4	2012-01-19	1914.97	251.27
...	...	...	...
2715	2023-01-25	2428.57	320.76
2716	2023-01-26	2468.65	325.57

∴ skiprows로 column 명이 나오기 전까지 skip ⇒ iloc으로 필요한 데이터 slicing

ex)
```
kospi = pd.read_excel("코스피.xlsx", skiprows = 9)
kospi.iloc[4:, :]
kospi.columns = ['date', '코스피', '코스피 200']
```

Type 변환

원하는 data만 slicing해도 타입은 바뀌지 않는다.
→ astype으로 데이터 타입 변경

```
kospi["코스피200"] = kospi["코스피200"].astype(float)
kospi.코스피 = kospi.코스피.astype(float)

kospi.date = pd.to_datetime(kospi.date, format="%Y-%m-%d")
```

index 조정

```
DataFrame_name.reset_index([drop])
```

↳ default : false

```
final = kospi.reset_index()
```

	index	date	코스피	코스피200
0	4	2012-01-02	1826.37	238.70
1	5	2012-01-03	1875.41	245.82
2	6	2012-01-04	1866.22	244.24
3	7	2012-01-05	1863.74	243.56
4	8	2012-01-06	1843.14	240.57

→ 기존 index 남아있음

```
final = kospi.reset_index(drop=True)
```

→ 기존 index 없어짐

외부데이터로 저장

```
DataFrame_Name.to_excel("파일 이름" [, index])
```

↳ default : True
False이면 row index 저장 안됨

그룹화

```
ex_dat = pd.DataFrame({
        "school" : ["A", "A", ···, "B"],
        "sex"    : ["M", "M", ···, "F"],
        "score"  : [100, 98, ···, 65], })
```

	school	sex	score
0	A	M	100
1	A	M	98
2	A	F	34
3	A	F	83
4	B	M	56
5	B	F	90
6	B	F	65

학교, 성별에 따른 score 평균 계산

⇒ `mean_dat = ex_dat.groupby(by = ["school", "sex"]).mean()`
`mean_dat.reset_index()`

통계량

`df.groupby([column1, column2, ···]).메소드`

↳ 범주형이어야 한다.

↳ 나머지 모든 column에 적용
⇒ 나머지는 수치형이어야 할

	cate_1	cate_2	value_1	value_2
0	B	X	19	95
1	B	Y	85	97
2	A	X	12	87
3	A	Y	29	14
4	B	Y	30	10
5	B	X	15	8
6	B	X	51	64
7	B	Y	69	62
8	B	X	88	23
9	A	X	88	58

ex) `df.groupby("cate_1").mean()` ⇒ Error!
`df.groupby("cate_1")["value_1", "value_2"].mean()` ⇒ 가능

여러 통계량 적용

그룹화된 dataframe에 여러 통계량을 적용하고 싶다.

⇒ agg 메소드 사용

```
sum_dat = ex_dat.groupby(["cate_1", "cate_2"]).agg({"value_1": "mean", "value_2": "sum"})
```

→ value_1에는 평균계산
value_2에는 합계산

```
sum_dat = ex_dat.groupby(["cate_1", "cate_2"]).agg({"value_1": ["mean", "sum"]})
```