Pandas_basic

류영표

Pandas Basic

Pandas 시작하기

Indexing & Slicing

산술연산

Pandas 시작하기

Pandas(판다스)

- Pandas라
 - 데이터 처리, 분석용 라이브러리
 - 표 형식 데이터, 시계열 데이터 등 다양한 형태의 데이터를 다루는데에 초 점

(CSV, text files, Microsoft Excel, SQL databases, ...)

- numpy의 배열 기반 계산 스타일을 많이 차용
- Pandas 특징
 - missing data 처리가 용이
 - 축의 이름에 따라 데이터를 정렬할 수 있는 자료구조 제공
 - 일반 데이터베이스처럼 데이터를 합치고 관계연산을 수행하는 기능
 - 시계열

____series

	Date	kospi	kosdaq	gold_fut_132030	Bond_273130
0	2020. 1. 2 오후 3:30:00	2175.17	674.02	10845	108215
1	2020. 1. 3 오후 3:30:00	2176.46	669.93	11000	108565
2	2020. 1. 6 오후 3:30:00	2155.07	655.31	11245	108745
3	2020. 1. 7 오후 3:30:00	2175.54	663.44	11180	108400
4	2020. 1. 8 오후 3:30:00	2151.31	640.94	11360	108270
5	2020. 1. 9 오후 3:30:00	2186.45	666.09	11055	107980
6	2020. 1. 10 오후 3:30:00	2206.39	673.03	11035	107760
7	2020. 1. 13 오후 3:30:00	2229.26	679.22	11080	107695
8	2020. 1. 14 오후 3:30:00	2238.88	678.71	10975	107860

DataFrame

- Series
 - 시리즈는 1차원 배열 같은 자료구조
 - 1) Series 생성하기
 - list, numpy, dictionary 등을 이용하여 생성가능

pd.Series(data, index, dtype, name)

- index: 각 row의 이름, 기본적으로는 숫자로 되어있음
- dtype: numpy의 dtype과 동일
- name: 각 column의 이름, 기본값이 없음

- DataFrame
 - 표 같은 스프레드시트 형식의 자료구조
 - index가 같은 series가 여러개 모여있는 형태
 - 1) DataFrame 생성하기
 - dictionary나 numpy배열을 이용하여 생성

pd.DataFrame(data, index, dtype, columns)

- index : 각 row의 이름, 기본적으로는 숫자로 되어있음
- dtype: numpy의 dtype과 동일
- columns : 각 column의 이름, 기본값이 없음

데이터 다루기

• 데이터 로딩

pandas file 파싱 함수	설명
read_csv	파일, url, 파일과 유사한 객체로부터 구분된 데이터 읽어옴. 데이터 구분자는 쉼표(,)가 기본
read_excel	엑셀(xls, xlsx)에서 표 형식의 데이터를 읽어옴.
read_json	JSON 문자열에서 데이터를 읽어옴.
read_pickle	파이썬 피클 포맷으로 저장된 객체를 읽어옴

• 데이터 저장

pandas file 저장 함수	설명
to_csv	csv 형식으로 저장
to_json	JSON 형식으로 저장
to_pickle	파이썬 피클 포맷으로 저장

(<u>참고</u>) 더 많은 데이터 파싱, 저장 함수

Indexing & Scling

Indexing & Scling

- Series
- DataFrame

- 1. integer-location based
 - numpy indexing과 유사하게 동작함
- 2. label-location based
 - label(index)를 기준으로 동작함
- 3. Series.iloc
 - integer-location based property
- 4. Series.loc
 - label-location based property

Series 생성

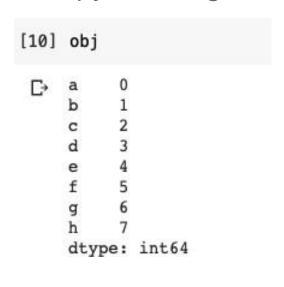
```
import pandas as pd
import numpy as np

obj = pd.Series([0,1,2,3,4,5,6,7], index=['a','b','c','d','e','f','g','h'], dtype = 'int64')
obj
```

```
# index
obj2 = pd.Series(['a', 'b', 'c'], index = ['i1','i2','i3'])
obj2
i1    a
i2    b
```

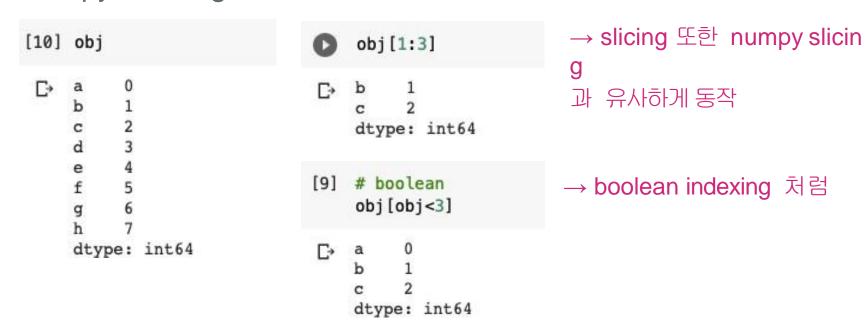
dtype: object

- 1. integer-location based
 - numpy indexing과 유사하게 동작함

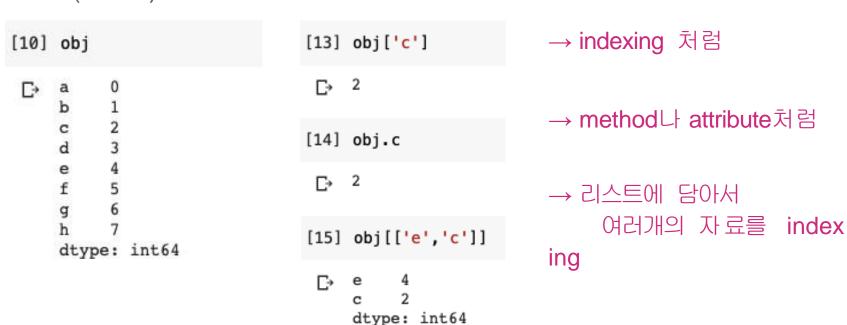


```
→ numpy의 indexing과 유사 하게 동
[5] obj[3]
[→ 3
[6] obj[-1]
₽
[7] obj[[1,3,5]] → 리스트에 위치(int)를 넘겨주면
                 그위치에 해당되는 자료들을
₽
                  index(label) 값과 함께 반환
   dtype: int64
```

- 1. integer-location based
 - numpy indexing과 유사하게 동작함



- 2. label-location based
 - label(index)를 기준으로 동작함



- 2. label-location based
 - label(index)를 기준으로 동작함

```
[10] obj
[16] obj['a':'c']

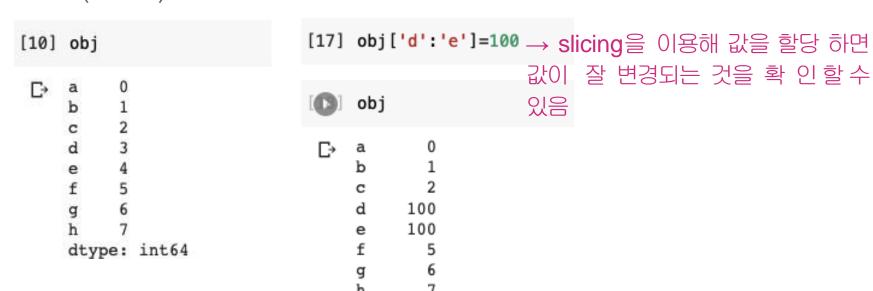
C a 0
b 1
c 2
d 3
e 4
f 5
g 6
h 7
dtype: int64

[16] obj['a':'c']

C a 0
b 1
c 2
dtype: int64

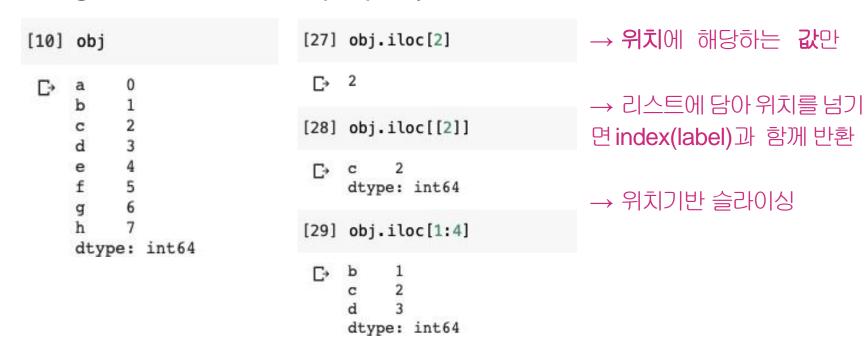
*주의*
라벨값을 이용해 슬라이싱을
하면 끝점까지 모두 포함
```

- 2. label-location based
 - label(index)를 기준으로 동작함

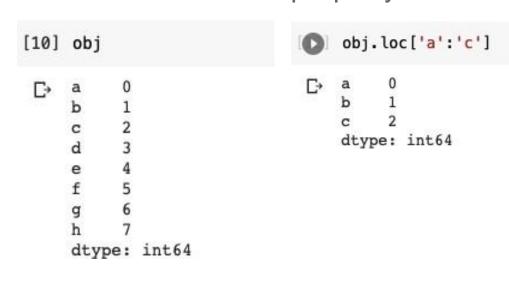


dtype: int64

- 3. Series.iloc
 - integer-location based property



- 3. Series.loc
 - label-location based property



→ **라벨**로 슬라이싱을 하기 때 문에 **끝**점까지 포함

- 1. indexing
 - 컬럼을 기준으로
- 2. slicing
 - 인덱스(라벨)을 기준으로
- 3. DataFrame.iloc
 - integer-location based property
- 4. DataFrame.loc
 - label-location based property

- 1. indexing
 - 컬럼을 기준으로

```
[7] frame['c3']
   r1
    r2
          14
    r4
          20
    Name: c3, dtype: int64
[8] frame.c3
   r1
    r2
          14
    r4
          20
    Name: c3, dtype: int64
```

- 1. indexing
 - 컬럼을 기준으로

frame									
₽		c1	c2	с3	c4	с5	с6		
	r1	0	1	2	3	4	5		
	r2	6	7	8	9	10	11		
	r3	12	13	14	15	16	17		
	r4	18	19	20	21	22	23		

⑤ frame[['c1', 'c2']]

C c1 c2
 r1 0 1
 r2 6 7
 r3 12 13
 r4 18 19

Q) C1,C2 으로 접근하려면?

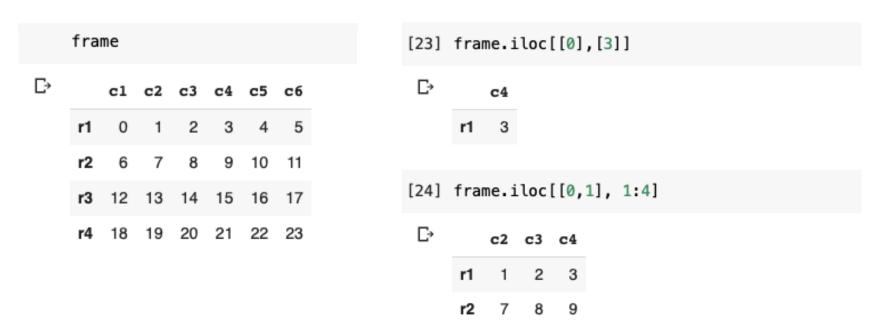
- 2. slicing
 - 라벨(row)을 기준으로

	fra	me					
₽		c1	c2	с3	c4	с5	c 6
	r1	0	1	2	3	4	5
	r2	6	7	8	9	10	11
	r3	12	13	14	15	16	17
	r4	18	19	20	21	22	23



df.iloc(row, column)

- 3. DataFrame.iloc
 - integer-location based property



df.loc(row, column)

- 4. DataFrame.loc
 - label-location based property

	fra	me					
₽		c1	c2	с3	c4	с5	c 6
	r1	0	1	2	3	4	5
	r2	6	7	8	9	10	11
	r3	12	13	14	15	16	17
	r4	18	19	20	21	22	23

```
[25] frame.loc[['r1'],['c4']]
₽
[26] frame.loc['r1':'r2',['c2','c3','c4']]
Ľ÷
                   → label-based는 슬라이싱을
                   하면 끝점까지 포함
```

산술연산

산술 연산

- 산술 연산
- Series
- DataFrame
- Series와 DataFrame

산술 연산

- 산술 연산자 혹은 산술 연산 메소드를 사용하여 연 산

메소드	설명
add, radd	덧셈(+)을 위한 메소드
sub, rsub	뺄셈(-)을 위한 메소드
mul, rmul	곱셈(*)을 위한 메소드
div, rdiv	나눗셈(/)을 위한 메소드
pow, rpow	거듭제곱(**)을 위한 메소드
floordiv, rfloordiv	소수점 내림(//)을 위한 메소드

- index를 기준으로 연산

```
[4] s1 = pd.Series([1, 2, 3, 4], index = ['a', 'b', 'c', 'd'])
[5] s2 = pd.Series([10, 20, 30, 40], index = ['a', 'b', 'c', 'd'])
[6] s1+s2
₽
       11
      22
       33
        44
    dtype: int64
```

- 짝이 맞지 않는 인덱스가 있는 경우, outer join과 유사하게 동작
 - → 결과 값에 두 인덱스의 값이 통합

```
[7] s1 = pd.Series([1, 2, 3, 4], index = ['a', 'b', 'c', 'd'])
[8] s3 = pd.Series([2, 2, 2, 2], index = ['b', 'c', 'd', 'e'])
[9] # a, e
   s1+s3
       NaN
               → 단, 서로 짝이 맞지 않는 인덱스는 결과값이 NaN(Not- a-Number)
       4.0
               을 의미하는 단어로 pandas에서는 누락된 값 혹은 NA로 취급)
       5.0
       6.0
       NaN
   dtype: float64
```

- 짝이 맞지 않는 인덱스가 있는 경우, 결측치의 기본값을 지정해줄 수 있음

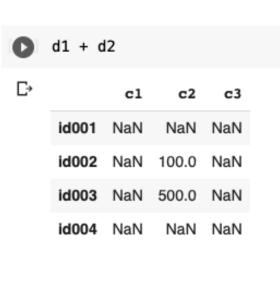
```
_ 메스트이 fill value 이자를 이요하며 견촉치를 채워줄 수 있음

▶ s1.add(s3, fill_value=0)

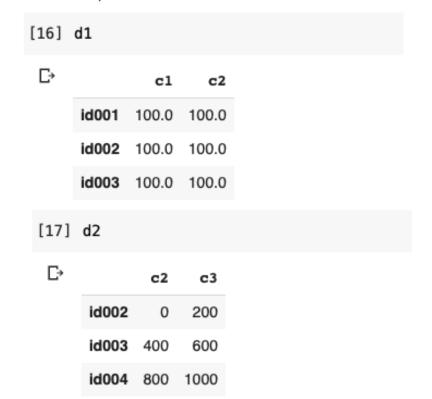
□ a 1.0
b 4.0
c 5.0
d 6.0
e 2.0
dtype: float64
```

- index, column을 기준으로 연산

[16]	d1		
₽		c1	c2
	id001	100.0	100.0
	id002	100.0	100.0
	id003	100.0	100.0
[17]	d2		
₽		c2	с3
	id002	0	200
	id003	400	600
	id004	800	1000



- index, column을 기준으로 연산

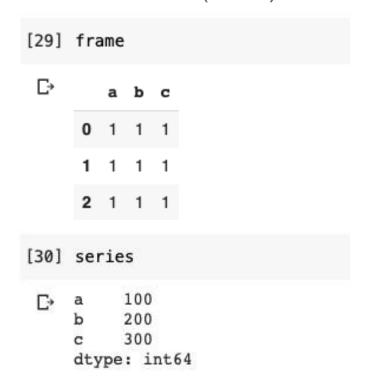


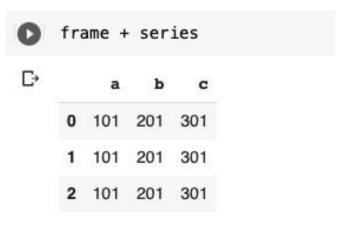
[19] # (id001, c3), (id004, c1) d1.add(d2, fill_value=0)

$\stackrel{\textstyle {}_{\!$		c1	c2	с3
	id001	100.0	100.0	NaN
	id002	100.0	100.0	200.0
	id003	100.0	500.0	600.0
	id004	NaN	800.0	1000.0

 \rightarrow 둘다 없는 경우는 NaN으로 채워짐

- Series의 index(label)을 frame의 column으로 맞추고 아래 row로 전파





- 메소드를 사용하면, axis 인자로 축을 지정해 column으로 전파할 수 있음
- axis = 0 또는 axis = 'index'

