Index

Contents

- Index
- Index 활용
- Index 생성 및 추출

데이터 처리에 중요한 역활을 하는 Index 에 대하여 알아보겠습니다. Index 는 우리말로 색인이라고 할 수 있을 것 같은데요. 색인은 무엇을 빨리 찾기 위해 순서대로 정리되어 있는 목록입니다. Index 는 색인처럼 어떤 값을 빨리 찾을 때도 필요하지만, 두 데이터를 어떤 값을 기준으로 결합하는데도 유용하게 쓰입니다. Index 는 Series 와 DataFrame 에 주로 활용됩니다. ss2 는 바로 이전 장에서 만든 Series 입니다. 출력을 해 보면 맨 왼쪽에 0 ~ 4 까지 값이 보이는데요. 이게 Index 입니다. 특별하게 지정하지 않으면 숫자 0 부터서 순서대로 들어가게 됩니다. 다음은 알파벳 Index 를 넣어서 ss3 를 생성하고 출력 해보겠습니다. 맨 왼쪽 index 값이 숫자가 아니라 알파벳으로 바뀌었습니다.

```
import pandas as pd

ss1 = [11,12,13,14,15]
ss2 = pd.Series(ss1)
print(ss2)

ss3 = pd.Series(ss1, index=['a', 'b', 'c', 'd', 'e'])
print(ss3)
```

```
0    11
1    12
2    13
3    14
4    15
dtype: int64
a    11
b    12
c    13
d    14
e    15
dtype: int64
```

Index 의 본연의 기능은 찾기입니다. ss3.loc[인덱스값] 를 이용하여 원하는 값을 찾을 수 있습니다. 인덱스 'c' 에 해당하는 값은 13입니다. ss3.loc['c'] 를 하면 13이 출력됩니다. 만약, 인덱스 'a' 와 'c' 를 다 찾고 싶으면 ['a', 'c'] 와 같이 List 로 넣어주면 됩니다. loc 를 하지 않아도 같은 결과를 얻으시겠지만, loc 를 넣으면 'a','c' 를 column 이 아니라 index 에서 찾는다는 것을 명확하게 해 줍니다.

```
print(ss3.loc['a'], ss3['c'])
print(ss3.loc[['a','c']])
```

```
11 13
a    11
c    13
dtype: int64
```

DataFrame 에서도 동일하게 활용가능합니다. 먼저 df1 이라는 DataFrame 을 생성하고 출력합니다. Default Index 인 숫자 0 ~ 4 로 되어 있음을 확인할 수 있습니다. 이제 원하는 인덱스 s1 ~ s5 를 할당하고 df2 에 저장합니다. 출력 결과를 보니 df2 의 인덱스가 바뀌었습니다.

이번에는 원하는 값을 찾아보겠습니다. df2 의 index 가 's3' 인 c1 컬럼값을 알고 싶다면 df2.loc['s3']['c1'] 이라고 하면 됩니다. 만약, c1 과 c2 둘다 출력하고 싶으면 df2.loc['s3'][['c1','c2']] 형태로 리스트로 입력합니다. 실수로 df2.loc['s3']['c1','c2'] 로 입력을 하면 Pandas 패키지는 'c1,'c2' 가 하나의 column 이름이라고 착각하게 되어 에러가 발생합니다.

```
# DataFrame 생성
c1_list = [11,12,13,14,15]
c2_list = ['a','b','c','d','e']
df1 = pd.DataFrame({'c1': c1_list, 'c2': c2_list})
print(df1)

print('\n')
df2 = pd.DataFrame({'c1': c1_list, 'c2': c2_list}, index=['s1','s2','s3','s4','s4'])
print(df2)

print('\n')
print(df2.loc['s3']['c1']) # 13 출력
print(df2.loc['s3'][['c1','c2']]) # 13 과 c 출력
```

```
   c1 c2
0  11  a
1  12  b
2  13  c
3  14  d
4  15  e


    c1 c2
s1  11  a
s2  12  b
s3  13  c
s4  14  d
s4  15  e


13
c1    13
c2     c
Name: s3, dtype: object
```

set_index 메소드로 기존의 column 을 index 로 만들 수 있습니다. set_index('c2') 처리 후, df2 를 출력하시면 df1 의 'c2' 컬럼이 index 로 되어 있음을 확인할 수 있습니다. 이제 df2 의 index 값을 변경해 보겠습니다. 아래와 같이 DataFrame 의 Index를 호출하여 원하는 Index 로 교체도 가능합니다. 참고로 아래 df2 는 column 하나지만 현재 Series 가 아닌 DataFrame 입니다.

```
c1_list = [11,12,13,14,15]
c2_list = ['a','b','c','d','e']
df1 = pd.DataFrame({'c1': c1_list, 'c2': c2_list})
print(df1)

df2 = df1.set_index('c2') # c1 를 index 로 변경
print(df2)

print('\n')
df2.index = ['ss1', 'ss2', 'ss3', 'ss4', 'ss5']
print(df2, type(df2))
```

```
   c1 c2
0  11  a
1  12  b
2  13  c
3  14  d
4  15  e
    c1
c2
a   11
b   12
c   13
d   14
e   15


     c1
ss1  11
ss2  12
ss3  13
ss4  14
ss5  15 <class 'pandas.core.frame.DataFrame'>
```

항상 두 데이터셋을 index 로 병합할 때는 index 에 중복이 있는지 확인을 할 필요가 있습니다. index 가 중복 여부를 체크하는 인수는 verify_integriry 입니다. 아래는 중복이 있는 경우 에러를 발생시킵니다.

```
c1_list = [11,12,13,14,15]
c2_list = ['a','a','b','c','d'] # 값에 중복이 있음
df = pd.DataFrame({'c1': c1_list, 'c2': c2_list})
df.set_index('c2', verify_integrity=True) # index 중복여부를 체크
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_3484\2078573242.py in <module>
      2 c2_list = ['a','a','b','c','d'] # 값에 중복이 있음
      3 df = pd.DataFrame({'c1': c1_list, 'c2': c2_list})
----> 4 df.set_index('c2', verify_integrity=True) # index 중복여부를 체크

~\Anaconda3\lib\site-packages\pandas\util\_decorators.py in wrapper(*args, **kwargs)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
    312
    313         return wrapper

~\Anaconda3\lib\site-packages\pandas\core\frame.py in set_index(self, keys, drop,
append, inplace, verify_integrity)
   5508         if verify_integrity and not index.is_unique:
   5509             duplicates = index[index.duplicated()].unique()
-> 5510             raise ValueError(f"Index has duplicate keys: {duplicates}")
   5511
   5512         # use set to handle duplicate column names gracefully in case of drop

ValueError: Index has duplicate keys: Index(['a'], dtype='object', name='c2')
```
