CHAPTER

05

Pandas 라이브러리

Pandas Library

컴퓨터소프트웨어공학과
김대영

개 요

- Pandas 자료구조
- Pandas 필수 기능
- 통계

- **인덱싱, 선택, 필터링**(Indexing, Selection, Filtering)
  - Series 인덱싱은 NumPy 배열 인덱싱과 유사하게 동작
  - 인덱싱을 위해 정수값 대신 Series의 인덱스 사용 가능

```
obj = pd.Series(np.arange(4.), index=['a','b','c','d’])
print(obj)
>       a    0.0
        b    1.0
        c    2.0
        d    3.0
print(obj['b’])
> 1.0
print(obj[2:4])
>       c    2.0
        d    3.0
print(obj[[1,3]])
>       b    1.0
        d    3.0
print(obj['b':'c’])
>       b    1.0
        c    2.0
```

- **인덱싱, 선택, 필터링**(Indexing, Selection, Filtering)

```
obj = pd.Series(np.arange(4.), index=['a','b','c','d’])
print(obj)
>       a    0.0
        b    1.0
        c    2.0
        d    3.0
print(obj['b':'c’])
>       b    1.0
        c    2.0

obj['b':'c'] = 5
print(obj)
>       a    0.0
        b    5.0
        c    5.0
        d    3.0
        dtype: float64
```

- **인덱싱, 선택, 필터링**(Indexing, Selection, Filtering)

```
data = pd.DataFrame(np.arange(16).reshape((4,4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four’])
print(data)
>                          one  two  three  four
        Ohio          0    1      2      3
        Colorado      4    5      6      7
        Utah          8    9     10     11
        New York     12   13     14     15
print(data['two’])
>       Ohio          1
        Colorado      5
        Utah          9
        New York     13
        Name: two, dtype: int64
print(data[:2])
>                          one  two  three  four
        Ohio          0    1      2      3
        Colorado      4    5      6      7
```

- **인덱싱, 선택, 필터링**(Indexing, Selection, Filtering)
  - 축 레이블(loc) 또는 정수(iloc)를 사용하여 NumPy 같은 표기법으로 DataFrame 객체의 로우와 컬럼 하위 집합을 선택할 수 있음

```
data = pd.DataFrame(np.arange(16).reshape((4,4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four’])

d = data.loc['Colorado', ['two', 'three’]]
print(d)
>       two      5
        three    6
        Name: Colorado, dtype: int64
d = data.iloc[2, [3, 0, 1]]
print(d)
>       four    11
        one      8
        two      9
        Name: Utah, dtype: int64
```

- **인덱싱, 선택, 필터링**(Indexing, Selection, Filtering)
  - DataFrame 객체의 인덱싱 옵션

df[val]	DataFrame 으로부터 컬럼 선택
df.loc[val]	레이블 값에 의해 DataFrame 객체로부터 로우 선택
df.loc[:, val]	레이블 값에 의해 DataFrame 객체로부터 컬럼의 부분집합 선택
df.loc[val1, val2]	레이블 값에 의해 DataFrame 객체로부터 로우와 컬럼 부분집합 선택
df.iloc[where]	정수 인덱스 값에 의해 DataFrame 객체로부터 로우 선택
df.iloc[:, where]	정수 인덱스 값에 의해 DataFrame 객체로부터 컬럼 선택
df.iloc[where_i, where_j]	정수 인덱스 값에 의해 DataFrame 객체로부터 로우와 컬럼 부분 집합 선택
df.at[label_i, label_j]	로우와 컬럼 레이블 값에 의해 단일 값 선택
df.iat[i, j]	로우와 컬럼 정수 인덱스 값에 의해 단일 값 선택
reindex method	레이블 값에 의해 로우 및 컬럽에 대한 축 선택
get_value, set_value method	로우와 컬럼 레이블 값에 의해 DataFrame 값 선택

- **산술연산과 정렬**(Arithmetic & Data Alignment)
  - Pandas의 중요한 특징은 서로 다른 인덱스를 가진 객체들 사이에서 산술연산 수행이 가능한 것임

```
s1 = pd.Series([7.3,-2.5,3.4,1.5], index=['a','c','d','e’])
s2 = pd.Series([-2.1,3.6,-1.5,4,3.1], index=['a','c','e','f','g’])
s = s1 + s2
print(s)
>       a    5.2
        c    1.1
        d    NaN
        e    0.0
        f    NaN
        g    NaN
        dtype: float64
```

- **산술연산과 정렬**(Arithmetic & Data Alignment)
  - DataFrame객체에서는 로우와 컬럼에 대한 정렬이 가능

```
df1 = pd.DataFrame(np.arange(9.).reshape((3,3)),
                                columns=list('bcd'),
                                index=['Ohio','Texas','Colorado’])
df2 = pd.DataFrame(np.arange(12.).reshape((4,3)),
                                columns=list('bde’),
                                index=['Utah','Ohio','Texas','Oregon’])

df = df1 + df2
print(df1)
>                b    c    d
        Ohio     0.0  1.0  2.0
        Texas    3.0  4.0  5.0
        Colorado 6.0  7.0  8.0
print(df2)
>                b     d     e
        Utah    0.0   1.0   2.0
        Ohio    3.0   4.0   5.0
        Texas   6.0   7.0   8.0
        Oregon  9.0  10.0  11.0
```

```
#If you add DataFrame objects with
#no column or row labels in common,
#the result will contain all nulls
 print(df)
 >                b   c     d   e
        Colorado  NaN NaN   NaN NaN
        Ohio      3.0 NaN   6.0 NaN
        Oregon    NaN NaN   NaN NaN
        Texas     9.0 NaN  12.0 NaN
        Utah      NaN NaN   NaN NaN
```

- **산술연산 메소드로 값 채우기**
  - 특정 값으로 채우기
    - 로우 인덱스와 컬럼 이름을 사용하여 지정된 위치에 값 변경
    - 산술연산 메소드를 사용하여 객체의 데이터 연산

```
df1 = pd.DataFrame(np.arange(12.).reshape((3,4)), columns=list('abcd'))
df2 = pd.DataFrame(np.arange(20.).reshape((4,5)), columns=list('abcde'))
df2.loc[1,'b'] = np.nan
print(df1)
>            a    b     c     d
        0  0.0  1.0   2.0   3.0
        1  4.0  5.0   6.0   7.0
        2  8.0  9.0  10.0  11.0
print(df2)
>            a     b     c     d     e
        0   0.0   1.0   2.0   3.0   4.0
        1   5.0   NaN   7.0   8.0   9.0
        2  10.0  11.0  12.0  13.0  14.0
        3  15.0  16.0  17.0  18.0  19.0
```

- 산술연산 메소드로 값 채우기

```
df = df1.add(df2, fill_value=0)
print(df)
>              a     b     c     d     e
         0   0.0   2.0   4.0   6.0   4.0
         1   9.0   5.0  13.0  15.0   9.0
         2  18.0  20.0  22.0  24.0  14.0
         3  15.0  16.0  17.0  18.0  19.0
df = 1/df1
print(df)
>              a         b         c         d
         0    inf  1.000000  0.500000  0.333333
         1  0.250  0.200000  0.166667  0.142857
         2  0.125  0.111111  0.100000  0.090909
df = df1.rdiv(1)
print(df)
>              a         b         c         d
         0    inf  1.000000  0.500000  0.333333
         1  0.250  0.200000  0.166667  0.142857
         2  0.125  0.111111  0.100000  0.090909
```

- **산술연산 메소드**

add, radd	덧셈 메소드 df1.add(df2): df1 + df2, df1.radd(df2): df2 + df1
sub, rsub	뺄셈 메소드 df1.sub(df2): df1 – df2, df1.rsub(df2): df2 – df1
div, rdiv	나눗셈 메소드 df1.div(df2): df1 / df2, df1.rdiv(df2): df2 / df1
floordiv, rfloordiv	소수점 내림 연산 메소드
mul, rmul	곱셈 메소드 df1.mul(df2) = df1 * df2, df1.rmul(df2) = df2 * df1
pow, rpow	거듭제곱 메소드 df1.pow(df2) = df1^df2, df1.rpow(df2) = df2^df1

- **DataFrame과 Series 사이 연산**
  - DataFrame과 Series 사이의 산술연산은 DataFrame의 컬럼에 있는 Series의 인덱스와 일치시키고, 그 로우에 대한 값을 브로드캐스팅

```
frame = pd.DataFrame(np.arange(12.).reshape((4,3)),
                                    columns=list('bde’),
                                    index=['Utah','Ohio','Texas','Oregon'])
series = frame.iloc[0]
print(frame)
>                b     d     e
        Utah    0.0   1.0   2.0
        Ohio    3.0   4.0   5.0
        Texas   6.0   7.0   8.0
        Oregon  9.0  10.0  11.0
        Name: Utah, dtype: float64
print(series)
>       b    0.0
        d    1.0
        e    2.0
val = frame - series
```

```
print(val)
>                b     d     e
        Utah    0.0   0.0   0.0
        Ohio    3.0   3.0   3.0
        Texas   6.0   6.0   6.0
        Oregon  9.0   9.0   9.0
```

- **DataFrame과 Series 사이 연산**
  - 컬럼이 아닌 로우에 대한 인덱스를 일치 시키고, 컬럼 값을 브로드캐스팅하기 위해 다른 메소드를 사용해야 함

```
series3 = frame['d']
print(frame)
>                b     d     e
        Utah    0.0   1.0   2.0
        Ohio    3.0   4.0   5.0
        Texas   6.0   7.0   8.0
        Oregon  9.0  10.0  11.0
print(series3)
>       Utah       1.0
        Ohio       4.0
        Texas      7.0
        Oregon    10.0
        Name: d, dtype: float64

val = frame.sub(series3, axis='index')
```

```
print(val)
>                 b    d    e
        Utah   -1.0  0.0  1.0
        Ohio   -1.0  0.0  1.0
        Texas  -1.0  0.0  1.0
        Oregon -1.0  0.0  1.0
```

- **함수 응용과 매핑**(Func. Application & Mapping)
  - NumPy의 유니버셜 함수(원소 단위의 배열 처리함수)와 Pandas 객체는 함께 동작할 수 있음

```
frame = pd.DataFrame(np.random.randn(4,3),
                     columns=list('bde'),
                     index = ['Utah', 'Ohio', 'Texas', 'Oregon'])
print(frame)
>                   b         d         e
        Utah    0.059607 -0.138955  0.657033
        Ohio   -0.664228  2.886051 -0.277190
        Texas   0.337992 -0.170650 -1.292136
        Oregon -0.619456 -0.334069 -0.937778
print(np.abs(frame))
>                   b         d         e
        Utah    0.059607  0.138955  0.657033
        Ohio    0.664228  2.886051  0.277190
        Texas   0.337992  0.170650  1.292136
        Oregon  0.619456  0.334069  0.937778
```

- **함수 응용과 매핑**(Func. Application & Mapping)
  - 1차원 배열에 대한 함수를 각 컬럼과 로우에 적용하는 동작도 자주 활용됨

```
frame = pd.DataFrame(np.random.randn(4,3),
                     columns=list('bde'),
                     index = ['Utah', 'Ohio', 'Texas', 'Oregon'])
print(frame)
>                    b         d         e
        Utah   -0.660786  0.962111  0.539039
        Ohio    0.092493  0.255853 -0.794341
        Texas  -0.272489  0.987850  2.378761
        Oregon -2.253532 -0.125087  0.155372

f = lambda x : x.max() - x.min()
v = frame.apply(f)
print(v)
>       b    2.346025
        d    1.112937
        e    3.173102
        dtype: float64
```

- **함수 응용과 매핑**(Func. Application & Mapping)
  - Apply메소드에 axis='column'를 사용하면 주어진 함수를 각 로우 마다 한번씩 적용할 수 있음

```
frame = pd.DataFrame(np.random.randn(4,3),
                          columns=list('bde'),
                          index = ['Utah', 'Ohio', 'Texas', 'Oregon'])
print(frame)
>                               b         d         e
         Utah    0.401577  0.572950  2.275922
         Ohio    0.979988  1.826116  0.483472
         Texas   1.144329  0.930770  0.166007
         Oregon  0.075902  0.659843  0.267665
f = lambda x : x.max() - x.min()
v = frame.apply(f, axis='columns')
print(v)
>        Utah      1.874345
         Ohio      2.806103
         Texas     2.075100
         Oregon    0.927508
         dtype: float64
```

- **함수 응용과 매핑**(Func. Application & Mapping)
  - 원소 단위의 Python 함수도 Pandas와 함께 사용될 수 있음

```
frame = pd.DataFrame(np.random.randn(4,3),
                          columns=list('bde'),
                          index = ['Utah', 'Ohio', 'Texas', 'Oregon’])
def f(x):
        return pd.Series([x.min(), x.max()], index=['min','max’])

v = frame.apply(f)
print(v)
>                 b         d         e
        min -0.720619 -0.755647 -1.110628
        max  0.688569  0.591289  1.553278
```

- **함수 응용과 매핑**(Func. Application & Mapping)
  - 실수 값에 대해서 형식문자열을 적용하고자 할 때, applymap 메소드를 사용할 수 있음

```
format = lambda x: '%.2f' % x
v = frame.applymap(format)
print(v)
>                b      d      e
        Utah    -1.03  -1.02  -0.48
        Ohio    -0.58   0.20   0.19
        Texas   -0.53  -1.73  -0.68
        Oregon   0.97  -1.51  -0.74
```

- **정렬과 랭킹**(Sorting & Ranking)
  - sort_index 메소드는 정렬된 값을 새로운 객체에 할당하여 리턴

```
obj = pd.Series(range(4), index=['d','a','b','c'])
print(obj.sort_index())
>       a    1
        b    2
        c    3
        d    0
        dtype: int64
frame = pd.DataFrame(np.arange(8).reshape((2,4)),
                                index=['three', 'one'],
                                columns=['d','a','b','c'])
print(frame.sort_index())
>                       d  a  b  c
        one     4  5  6  7
        three   0  1  2  3
print(frame.sort_index(axis=1))
>                       a  b  c  d
        three   1  2  3  0
        one     5  6  7  4
```

- **정렬과 랭킹**(Sorting & Ranking)
  - 랭킹은 배열 객체의 원소들에 대하여 차례로 1부터 유효한 데이터 포인트(점수 값)를 할당하는 것

```
obj = pd.Series([7,-5,7,4,2,0,4])
print(obj.rank())
>       0    6.5
        1    1.0
        2    6.5
        3    4.5
        4    3.0
        5    2.0
        6    4.5
        dtype: float64
```

- **정렬과 랭킹**(Sorting & Ranking)
  - rank 메소드의 활용
    - 'first' 적용시 가장 작은 원소에서 부터 포인트 1씩 할당
    - 동일한 원소값에 대해서는 우선적으로 나타난 값부터 포인트 할당

```
obj = pd.Series([7,-5,7,4,2,0,4])
print(obj.rank(method='first'))
>       0    6.0
        1    1.0
        2    7.0
        3    4.0
        4    3.0
        5    2.0
        6    5.0
        dtype: float64
```

- **정렬과 랭킹**(Sorting & Ranking)
  - rank 메소드의 활용

average	같은 값을 가진는 원소들의 평균을 순위로 지정
min	같은 값을 가지는 그룹을 낮은 순위로 지정
max	같은 값을 가지는 그룹을 높은 순위로 지정
first	원소의 위치 순서에 따라서 순위 지정
dense	'method=min' 과 같지만 같은 그룹 내에서 같은 순위를 지정하지 않고 1씩 증가시킴

- **중첩된 레이블을 가진 축 인덱스**
  - 모든 Pandas 함수들이 독립적인 레이블을 가지도록 하지만, 경우에 따라서는 중첩된 레이블을 가질 수 있음

```
obj = pd.Series(range(5), index=['a', 'a', 'b', 'b', 'c'])
print(obj)
>       a    0
        a    1
        b    2
        b    3
        c    4
        dtype: int64
print(obj.index.is_unique)
> False
print(obj['a'])
>       a    0
        a    1
        dtype: int64
```

- **통계**(Statistics)
  - Pandas 객체는 수학 및 통계 메소드를 포함하고 있음
  - 대부분의 통계 메소드는 Series나 DataFrame에 대한 축소와 요약의 통계 범주에 속함

```
df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],
                   [np.nan, np.nan],[0.75, -1.3]],
                   index=['a','b','c','d'],
                   columns=['one','two'])

print(df)
print(df.sum())
print(df.sum(axis='columns'))
print(df.mean(axis='columns',skipna=False))
```

- **통계**(Statistics)
  - Pandas 축소 메소드

axis	연산을 수행하기 위한 축 지정 DataFrame: row (0), column (1)
skipna	누락된 값을 제외할 것인지 지정
level	연산하려는 축이 계층적 인덱스로 구성되었으면 레벨에 따라 그룹화하여 계산

- **통계**(Statistics)

```
df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],
                   [np.nan, np.nan],[0.75, -1.3]],
                   index=['a','b','c','d'],
                   columns=['one','two'])

print(df.idxmax())                  # min/max index
print(df.cumsum())                  # cumulative sum

# set of summary statistics for Series or DF column
print(df.describe())
```

- **통계**(Statistics)
  - Pandas 통계 메소드

count	Non-NA값 계산
describe	Series 또는 DataFrame 컬럼에 대한 요약 통계 계산
min, max	최소값 및 최대값 계산
argmin, argmax	최소값 및 최대값의 인덱스 위치(정수) 반환
idxmin, idxmax	최소값과 최대값의 인덱스의 값 반환
quantile	0과 1사이의 범위에서 표본 분위수 계산
sum	합을 계산
mean	평균을 계산
median	중간값(50% 분위) 반환
mad	평균에서 평균절대편차 계산

- **통계**(Statistics)
  - Pandas 통계 메소드

prod	모든 원소에 대한 곱 연산
var	분산 값 계산
std	편차 값 계산
skew	표본비대칭도(3차 적률) 계산
kurt	표본첨도(4차 적률) 계산
cumsum	누적합 계산
cummin, cummax	누적 최소값 및 누적 최대값 계산
cumprod	누적 곱 계산
diff	1차 산술 차이 값 계산
pct_change	퍼센트 변화율 계산