[3-3. Series, DataFrame 다루기]

1. Series 다루기

1-1. Series의 주요 속성

- index : Series의 인덱스 객체를 반환합니다. 인덱스는 Series의 각 데이터 포인트에 할당된 라벨입니다.
- values : Series의 데이터를 numpy 배열 형태로 반환합니다.
- dtype : Series에 저장된 데이터의 타입을 반환합니다. 예를 들어, 'float64', 'int64', 'object' 등 입니다.
- shape : Series의 형태를 나타내며, (n,) 형태의 튜플을 반환합니다. 여기서 'n'은 데이터의 수입니다.
- size : Series에 있는 데이터 항목의 총 개수를 반환합니다.
- name : Series의 이름을 반환하거나 설정합니다. Series 객체를 생성할 때 name 매개변수를 통해 이름을 할당할 수 있습니다.

1-2. Series의 주요 속성 사용 예시

In [ ]:
```
from pandas import Series
data = Series(
    [10, 20, 30, 40, 50],
    index=['a', 'b', 'c', 'd', 'e'],
    name='numbers'
)
```

In [ ]:
```
print(data.index)
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

In [ ]:
```
print(data.values)
```

```
[10 20 30 40 50]
```

In [ ]:
```
print(data.dtype)
```

```
int64
```

In [ ]:
```
print(data.shape)
```

```
(5,)
```

In [ ]:
```
print(data.size)
```

```
5
```

In [ ]:
```
print(data.name)
```

```
numbers
```

In [ ]:
```
data.name = 'new_numbers'
print(data.name)
```

```
new_numbers
```

1-3. Series의 주요 메서드

- head(n=5) : Series의 처음부터 'n'개의 항목을 반환합니다. 'n'을 지정하지 않으면 기본값은 5입니다.
- tail(n=5) : Series의 끝에서부터 'n'개의 항목을 반환합니다. 'n'을 지정하지 않으면 기본값은 5입니다.
- describe() : Series의 요약 통계를 반환합니다. 평균, 표준편차, 최소값, 최대값 등의 통계 정보를 제공합니다.
- unique() : Series에서 고유한 값들의 배열을 반환합니다.
- value_counts() : Series에서 각 값의 출현 빈도를 계산하여 반환합니다.
- apply(func) : Series의 각 요소에 함수 func을 적용합니다.
- sort_values(ascending=True) : Series의 값을 기준으로 오름차순(또는 내림차순)으로 정렬합니다.
- sort_index(ascending=True) : Series의 인덱스를 기준으로 오름차순(또는 내림차순)으로 정렬합니다.
- isnull() : Series의 각 요소가 NaN(Not a Number)인지 아닌지를 나타내는 불리언 배열을 반환합니다. 즉, 개별 값에 대하여 NaN이면 True, NaN이 아니면 False를 도출하여 최종적으로 True와 False로 이루어진 배열을 반환합니다.
- notnull() : isnull()과는 반대로, Series의 각 요소가 NaN이 아닌 경우 True를 반환합니다.

1-4. Series의 주요 메서드 사용 예시

In [ ]:
```
import numpy as np
from pandas import Series

data = Series([1, 2, np.nan, 4, 5, np.nan, 7, 8, 9])

print(data)
```

```
0    1.0
1    2.0
2    NaN
3    4.0
4    5.0
5    NaN
6    7.0
7    8.0
8    9.0
dtype: float64
```

In [ ]:
```
print(data.head(3))
```

```
0    1.0
1    2.0
2    NaN
dtype: float64
```

In [ ]:
```
print(data.tail(3))
```

```
6    7.0
7    8.0
8    9.0
dtype: float64
```

In [ ]:
```
print(data.describe())
```

```
count    7.000000
mean     5.142857
std      3.023716
min      1.000000
25%      3.000000
50%      5.000000
75%      7.500000
max      9.000000
dtype: float64
```

In [ ]:
```
print(data.unique())
```

```
[ 1.  2. nan  4.  5.  7.  8.  9.]
```

In [ ]:
```
print(data.value_counts())
```

```
1.0    1
2.0    1
4.0    1
5.0    1
7.0    1
8.0    1
9.0    1
Name: count, dtype: int64
```

In [ ]:
```
print(data.apply(lambda x: x * 2))
```

```
0     2.0
1     4.0
2     NaN
3     8.0
4    10.0
5     NaN
6    14.0
7    16.0
8    18.0
dtype: float64
```

In [ ]:
```
print(data.sort_values())
```

```
0    1.0
1    2.0
3    4.0
4    5.0
6    7.0
7    8.0
8    9.0
2    NaN
5    NaN
dtype: float64
```

In [ ]:
```
print(data.sort_values(ascending=False))
```

```
8    9.0
7    8.0
6    7.0
4    5.0
3    4.0
1    2.0
0    1.0
2    NaN
5    NaN
dtype: float64
```

In [ ]:
```
print(data.sort_index(ascending=False))
```

```
8    9.0
7    8.0
6    7.0
5    NaN
4    5.0
3    4.0
2    NaN
1    2.0
0    1.0
dtype: float64
```

In [ ]:
```
print(data.isnull())
```

```
0    False
1    False
2     True
3    False
4    False
5     True
6    False
7    False
8    False
dtype: bool
```

In [ ]:
```
print(data.notnull())
```

```
0     True
1     True
2    False
3     True
4     True
5    False
6     True
7     True
8     True
dtype: bool
```

1-5. Series 객체들 사이에서의 연산

Series 객체들 사이에서 연산을 수행하는 것은 매우 직관적이며, Python의 기본 연산자를 사용하여 쉽게 할 수 있습니다. Series 객체들 사이의 연산은 기본적으로 인덱스에 맞춰서 이루어집니다. 이는 각 연산에서 대응되는 인덱스의 값끼리 연산이 수행된다는 것을 의미합니다. 만약 일치하는 인덱스가 없는 경우, 결과는 'NaN' (Not a Number, 즉, 결측치를 의미)을 표시됩니다.

1-5-1. 기본 연산 예시

In [ ]:
```python
import pandas as pd

# Series 객체 생성
s1 = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])
s2 = pd.Series([4, 3, 2, 1], index=['d', 'c', 'b', 'a'])

# 더하기
add_result = s1 + s2

# 빼기
sub_result = s1 - s2

# 곱하기
mul_result = s1 * s2

# 나누기
div_result = s1 / s2

print("더하기 결과:\n", add_result)
print("빼기 결과:\n", sub_result)
print("곱하기 결과:\n", mul_result)
print("나누기 결과:\n", div_result)
```

```
더하기 결과:
 a    2
b    4
c    6
d    8
dtype: int64
빼기 결과:
 a    0
b    0
c    0
d    0
dtype: int64
곱하기 결과:
 a     1
b     4
c     9
d    16
dtype: int64
나누기 결과:
 a    1.0
b    1.0
c    1.0
d    1.0
dtype: float64
```

1-5-2. 브로드캐스팅

Series와 스칼라 값(단일 숫자 값) 사이의 연산도 가능합니다. 이 경우 스칼라 값은 Series의 모든 요소에 대해 연산이 적용됩니다. 이를 브로드캐스팅이라고 합니다.

In [ ]:
```
# 스칼라 값과의 연산
scalar_add = s1 + 5
scalar_mul = s1 * 2

print("스칼라 더하기 결과:\n", scalar_add)
print("스칼라 곱하기 결과:\n", scalar_mul)
```

```
스칼라 더하기 결과:
 a    6
b    7
c    8
d    9
dtype: int64
스칼라 곱하기 결과:
 a    2
b    4
c    6
d    8
dtype: int64
```

1-5-3. 불리언 인덱싱

불리언 배열을 사용하여 Series에서 특정 조건에 맞는 값을 추출할 수 있습니다. 불리언 인덱싱은 조건을 만족하는(True) 데이터만 선택적으로 추출할 수 있도록 해줍니다.

In [ ]:
```
s = pd.Series(range(10))
print(s)
```

```
0    0
1    1
2    2
3    3
4    4
5    5
6    6
7    7
8    8
9    9
dtype: int64
```

In [ ]:
```
s > 5
```

Out[ ]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
7     True
8     True
9     True
dtype: bool
```

In [ ]:
```
s[s > 5]
```

Out[ ]:
```
6    6
7    7
8    8
9    9
dtype: int64
```

2. DataFrame 다루기

2-1. DataFrame의 주요 속성

- index : DataFrame의 행 레이블(인덱스)입니다. 기본적으로, 이는 0부터 시작하는 정수 인덱스이지만, 시계열 데이터와 같이 다른 레이블을 가질 수도 있습니다.
- columns : DataFrame의 열 이름을 나타냅니다.
- dtypes : DataFrame 내 각 열의 데이터 타입을 나타내는 Series입니다.
- shape : DataFrame의 형태를 나타내는 튜플입니다.(행의 수, 열의 수)
- size : DataFrame에 있는 전체 요소의 수입니다. 행의 수 x 열의 수로 계산됩니다.
- values : DataFrame의 데이터를 NumPy 배열로 반환합니다. 이를 통해 DataFrame 데이터를 NumPy 라이브러리를 사용하여 처리할 수 있습니다.
- T : DataFrame의 전치를 반환합니다. 즉, 행과 열을 바꾼 DataFrame을 생성합니다.
- axes : DataFrame의 행과 열 레이블을 리스트로 반환합니다. 첫 번째 요소는 행 레이블(index), 두 번째 요소는 열 레이블(columns) 입니다.

2-2. DataFrame의 주요 속성 사용 예시

In [ ]:
```
from pandas import DataFrame

data = {'Name': ['John', 'Anna', 'Peter', 'Linda'],
        'Age': [28, 34, 29, 32],
        'City': ['New York', 'Paris', 'Berlin', 'London']}

df = DataFrame(data)

print(df)
```

```
    Name  Age      City
0   John   28  New York
1   Anna   34     Paris
2  Peter   29    Berlin
3  Linda   32    London
```

In [ ]:
```
print(df.index)
# RangeIndex 객체(Generator) 반환
```

```
RangeIndex(start=0, stop=4, step=1)
```

In [ ]:
```
print(df.columns)
```

```
Index(['Name', 'Age', 'City'], dtype='object')
```

In [ ]:
```
print(df.dtypes)
```

```
Name    object
Age      int64
City    object
dtype: object
```

In [ ]:
```
print(df.shape)
```

```
(4, 3)
```

In [ ]:
```
print(df.size)
```

```
12
```

In [ ]:
```
print(df.values)
```

```
[['John' 28 'New York']
 ['Anna' 34 'Paris']
 ['Peter' 29 'Berlin']
 ['Linda' 32 'London']]
```

In [ ]:
```
print(df.T)
```

```
             0      1       2       3
Name      John   Anna   Peter   Linda
Age         28     34      29      32
City  New York  Paris  Berlin  London
```

In [ ]:
```
print(df)
```

```
    Name  Age      City
0   John   28  New York
1   Anna   34     Paris
2  Peter   29    Berlin
3  Linda   32    London
```

In [ ]:
```
print(df.axes)
```

```
[RangeIndex(start=0, stop=4, step=1), Index(['Name', 'Age', 'City'], dtyp
e='object')]
```

2-3. DataFrame의 주요 메서드

- head(n=5) : DataFrame의 처음 n행을 반환합니다. 기본값은 5입니다.
- tail(n=5) : DataFrame의 마지막 n행을 반환합니다. 기본값은 5입니다.
- describe() : 숫자형 열에 대한 주요 통계량(카운트, 평균, 표준편차, 최소값, 최대값 등)을 요약하여 보여줍니다.
- info() : DataFrame의 주요 정보를 출력합니다. 인덱스의 데이터 타입과 열, 비-결측치 값의 개수, 메모리 사용량 등이 포함됩니다.
- mean() : 숫자형 열의 평균값을 계산합니다.
- sum() : 숫자형 열의 합계를 계산합니다.
- max() : 각 열의 최대값을 찾습니다.
- min() : 각 열의 최소값을 찾습니다.
- drop(labels=None, axis=0, inplace=False) : 지정된 레이블의 행이나 열을 제거합니다. axis=0은 행을, axis=1은 열을 의미합니다. inplace=True로 설정하면 변경사항을 원본 DataFrame에 바로 적용합니다.
- apply(func, axis=0) : 함수 func를 DataFrame의 열 또는 행에 적용합니다. axis=0은 행 방향으로 함수를 적용합니다. 향 방향으로 이동하면서 각 열에 대해 지정된 함수를 적용합니다. axis=1은 열 방향으로 함수를 적용합니다. 즉, 열 방향으로 이동하면서 각 행 전체에 대해 함수를 적용합니다.

2-4. DataFrame의 주요 메서드 사용 예시

In [ ]:
```
import numpy as np
from pandas import DataFrame

data = {'Name': ['John', 'Anna', 'Peter', 'Linda'],
        'Age': [28, 34, 29, 32],
        'Salary': [50000, 62000, 58000, 74000]}
df = DataFrame(data)

print(df)
```

```
    Name  Age  Salary
0   John   28   50000
1   Anna   34   62000
2  Peter   29   58000
3  Linda   32   74000
```

In [ ]:
```
print(df.head(2))
```

```
   Name  Age  Salary
0  John   28   50000
1  Anna   34   62000
```

In [ ]:
```
print(df.tail(3))
```

```
    Name  Age  Salary
1   Anna   34   62000
2  Peter   29   58000
3  Linda   32   74000
```

In [ ]:
```
print(df.describe())
```

```
             Age   Salary
count   4.000000      4.0
mean   30.750000  61000.0
std     2.753785  10000.0
min    28.000000  50000.0
25%    28.750000  56000.0
50%    30.500000  60000.0
75%    32.500000  65000.0
max    34.000000  74000.0
```

In [ ]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Name    4 non-null      object
 1   Age     4 non-null      int64 
 2   Salary  4 non-null      int64 
dtypes: int64(2), object(1)
memory usage: 228.0+ bytes
None
```

In [ ]:
```
print(df[['Age', 'Salary']].mean())
```

```
Age          30.75
Salary    61000.00
dtype: float64
```

In [ ]:
```
print(df[['Age', 'Salary']].sum())
```

```
Age          123
Salary    244000
dtype: int64
```

In [ ]:
```
print(df[['Age', 'Salary']].max())
```

```
Age          34
Salary    74000
dtype: int64
```

In [ ]:
```
print(df[['Age', 'Salary']].min())
```

```
Age          28
Salary    50000
dtype: int64
```

In [ ]:
```
print(df.drop('Age', axis=1))
```

```
    Name  Salary
0   John   50000
1   Anna   62000
2  Peter   58000
3  Linda   74000
```

In [ ]:
```
print(df.drop(['Age', 'Salary'], axis=1))
```

```
    Name
0   John
1   Anna
2  Peter
3  Linda
```

In [ ]:
```
print(df.drop([1, 2], axis=0))
```

```
    Name  Age  Salary
0   John   28   50000
3  Linda   32   74000
```

In [ ]:
```
print(df.drop(range(1, 3), axis=0))
```

```
    Name  Age  Salary
0   John   28   50000
3  Linda   32   74000
```

In [ ]:
```
print(df[['Age', 'Salary']].apply(sum, axis=0))
```

```
Age          123
Salary    244000
dtype: int64
```

In [ ]:
```
print(df[['Age', 'Salary']].apply(np.mean, axis=0))
```

```
Age          30.75
Salary    61000.00
dtype: float64
```

In [ ]:
```
print(df[['Age', 'Salary']].apply(min, axis=0))
```

```
Age          28
Salary    50000
dtype: int64
```

In [ ]:
```
print(df[['Age', 'Salary']].apply(max, axis=0))
```

```
Age          34
Salary    74000
dtype: int64
```

In [ ]:
```
print(df[['Age', 'Salary']].apply(lambda x: sum(x**2), axis=0))
```

```
Age              3805
Salary    15184000000
dtype: int64
```

2-5. Series 객체들 사이에서의 연산

pandas는 두 DataFrame 간의 연산을 지원하며 데이터 분석과정에서 다양한 수치 계산을 수행할 수 있습니다. 이러한 연산은 각 DataFrame의 동일한 위치에 있는 데이터끼리 수행되며, 기본적으로 더하기, 빼기, 곱하기, 나누기 등의 연산이 가능합니다.

2-5-1. 기본 연산 예시

In [ ]:
```
import numpy as np
from pandas import DataFrame

df1 = DataFrame(np.arange(9).reshape(3, 3), columns=list('ABC'))
df2 = DataFrame(np.arange(4, 13).reshape(3, 3), columns=list('ABC'))

print(df1)
print(df2)
```

```
   A  B  C
0  0  1  2
1  3  4  5
2  6  7  8
    A   B   C
0   4   5   6
1   7   8   9
2  10  11  12
```

In [ ]:
```
# DataFrame 더하기
result_add = df1 + df2

# DataFrame 빼기
result_sub = df1 - df2

# DataFrame 곱하기
result_mul = df1 * df2

# DataFrame 나누기
result_div = df1 / df2

print("더하기 결과:\n", result_add)
print("빼기 결과:\n", result_sub)
print("곱하기 결과:\n", result_mul)
print("나누기 결과:\n", result_div)
```

```
더하기 결과:
     A   B   C
0   4   6   8
1  10  12  14
2  16  18  20
빼기 결과:
    A  B  C
0 -4 -4 -4
1 -4 -4 -4
2 -4 -4 -4
곱하기 결과:
     A   B   C
0   0   5  12
1  21  32  45
2  60  77  96
나누기 결과:
           A         B         C
0  0.000000  0.200000  0.333333
1  0.428571  0.500000  0.555556
2  0.600000  0.636364  0.666667
```

2-5-2. 연산 메서드 사용

- add(), sub(), mul(), div() 등의 메서드를 사용하여 연산을 수행할 때, fill_value 매개변수를 통해 누락된 데이터에 대한 기본값을 지정할 수 있습니다.

In [ ]:
```
df3 = DataFrame(np.arange(4).reshape(2, 2), columns=['A', 'B'])
df4 = DataFrame(np.arange(6, 10).reshape(2, 2), columns=['B', 'C'])

print("df3:\n", df3)
print("\ndf4:\n", df4)
```

```
df3:
    A  B
0  0  1
1  2  3

df4:
    B  C
0  6  7
1  8  9
```

In [ ]:
```
print(df3 + df4)
```

```
    A   B   C
0 NaN   7 NaN
1 NaN  11 NaN
```

In [ ]:
```
# fill_value를 사용한 더하기 연산
result_add_fill = df3.add(df4, fill_value=0)

print("fill_value를 사용한 더하기 결과:\n", result_add_fill)
```

```
fill_value를 사용한 더하기 결과:
      A   B    C
0  0.0   7  7.0
1  2.0  11  9.0
```

2-5-3. 브로드캐스팅

DataFrame과 Series 간 연산을 수행할 때, pandas는 기본적으로 브로드캐스팅(broadcasting) 규칙을 따릅니다. 이는 Series의 각 요소가 DataFrame의 모든 행이나 열에 대해 연산을 수행한다는 의미입니다. 주로 DataFrame의 각 행이나 각 열과 Series의 연산에 사용되며, 연산의 기준은 Series의 인덱스와 DataFrame의 컬럼 또는 인덱스에 따라 달라집니다.

In [ ]:
```python
import numpy as np
from pandas import DataFrame, Series

# DataFrame 생성
df = DataFrame(
    np.arange(12.).reshape((4, 3)),
    columns=list('bde'),
    index=['Utah', 'Ohio', 'Texas', 'Oregon']
)

# Series 생성
sr = Series([1, 2, 3], index=['b', 'e', 'd'])

# DataFrame의 각 행과 Series의 연산 (브로드캐스팅)
result = df + sr

print("df:\n", df)
print("\nsr:\n", sr)
print("\nresult:\n", result)
```

```
df:
           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

sr:
 b    1
e    2
d    3
dtype: int64

result:
            b     d     e
Utah     1.0   4.0   4.0
Ohio     4.0   7.0   7.0
Texas    7.0  10.0  10.0
Oregon  10.0  13.0  13.0
```

DataFrame의 각 열과 Series를 연산하려면, DataFrame의 메서드를 사용하고 axis 매개변수를 적절히 설정해야 합니다.

In [ ]:
```
sr2 = pd.Series(
    [1, 2, 3, 4],
    index=['Utah', 'Ohio', 'Texas', 'Oregon']
)

# DataFrame의 각 열과 Series의 더하기 연산
result2 = df.sub(sr2, axis='index')  # axis=0

print("\nsr:\n", sr2)
print("\nresult:\n", result2)
```

```
sr:
 Utah      1
Ohio      2
Texas     3
Oregon    4
dtype: int64

result:
           b    d    e
Utah   -1.0  0.0  1.0
Ohio    1.0  2.0  3.0
Texas   3.0  4.0  5.0
Oregon  5.0  6.0  7.0
```

2-5-4. 불리언 인덱싱

Series와 마찬가지로 DataFrame도 불리언 인덱싱이 가능합니다.

불리언 인덱싱을 사용하기 위해서는, 각 행에 대해 평가되는 조건식을 작성합니다. 이 조건식은 DataFrame의 각 행에 대해 True 또는 False 값을 갖는 불리언 시리즈를 생성합니다.

이 시리즈를 DataFrame에 인덱싱으로 사용함으로써 True에 해당하는 행들만 필터링할 수 있습니다.

2-5-4-1. 단일 조건 사용

In [ ]:
```python
import pandas as pd

# 예제 DataFrame 생성
df = pd.DataFrame({
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Edward'],
    'age': [24, 27, 22, 32, 29],
    'salary': [70000, 54000, 50000, 120000, 77000]
})

# 'age'가 25보다 큰 행만 선택
filtered_df = df[df['age'] > 25]

print(filtered_df)
```

```
     name  age  salary
1     Bob   27   54000
3   David   32  120000
4  Edward   29   77000
```

2-5-4-2. 복수 조건 사용

복수의 조건을 함께 사용하려면 각 조건을 '()'로 감싸고, '&' (AND), '|' (OR) 연산자를 사용해 조합합니다.

In [ ]:
```python
# 'age'가 25보다 크고 'salary'가 75000보다 적은 행만 선택
filtered_df = df[(df['age'] > 25) & (df['salary'] < 75000)]

print(filtered_df)
```

```
  name  age  salary
1  Bob   27   54000
```

2-5-4-3. isin() 메서드 사용

특정 컬럼의 값이 주어진 리스트 안에 있는지 확인할 때 isin() 메서드를 사용할 수 있습니다.

In [ ]:
```python
# 'name'이 특정 리스트 안에 있는 행만 선택
names_to_select = ['Alice', 'David']
filtered_df = df[df['name'].isin(names_to_select)]

print(filtered_df)
```

```
    name  age  salary
0  Alice   24   70000
3  David   32  120000
```

2-5-4-4. str.contains() 메서드 사용

특정 문자열을 포함하는 행을 선택할 때는 str.contains() 메서드를 사용할 수 있습니다.

In [ ]:
```python
# 'name'에 'a'가 포함된 행만 선택
filtered_df = df[df['name'].str.contains('a')]

print(filtered_df)
```

```
      name  age  salary
2  Charlie   22   50000
3    David   32  120000
4   Edward   29   77000
```