[3-6. 데이터 재구성 및 피벗(stack, unstack, melt 등]

1. stack() 과 unstack() 메서드

1-1. stack() 메서드

stack() 메서드는 DataFrame의 컬럼을 인덱스 레벨로 "쌓아 올리는" 작업을 수행합니다. 이로 인해, DataFrame이 더 길고 좁은 형태로 변환됩니다. 기본적으로, 가장 안쪽 레벨의 컬럼 인덱스가 새로운 인덱스 레벨로 이동합니다.

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

df = pd.DataFrame(np.random.randn(4, 2),
                  index=['A', 'B', 'C', 'D'],
                  columns=['One', 'Two'])

# stack() 사용
stacked = df.stack()

print("Original DataFrame:")
print(df)
print("\nStacked DataFrame:")
print(stacked)
```

```
Original DataFrame:
        One       Two
A -0.170347  0.269697
B -0.830369 -0.091330
C  0.829074 -0.234975
D  0.030879  0.269165

Stacked DataFrame:
A  One   -0.170347
   Two    0.269697
B  One   -0.830369
   Two   -0.091330
C  One    0.829074
   Two   -0.234975
D  One    0.030879
   Two    0.269165
dtype: float64
```

1-2. unstack() 메서드

unstack() 메서드는 stack() 메서드의 반대 작업을 수행하여, 인덱스 레벨을 컬럼 레벨로 "펼치는" 작업을 수행합니다. 이를 통해 데이터가 더 넓고 짧은 형태로 변환됩니다.

In [ ]:
```
# unstack() 사용
unstacked = stacked.unstack()

print("Unstacked DataFrame:")
print(unstacked)
```

```
Unstacked DataFrame:
        One       Two
A -0.170347  0.269697
B -0.830369 -0.091330
C  0.829074 -0.234975
D  0.030879  0.269165
```

1-3. stack()과 unstack()의 레벨 지정

stack()과 unstack()은 level 매개변수를 통해 작업할 인덱스 또는 컬럼의 레벨을 지정할 수 있습니다. 이는 멀티레벨 인덱스 또는 컬럼을 가진 복잡한 DataFrame에서 사용됩니다.

In [ ]:
```
# 멀티레벨 컬럼을 가진 DataFrame 예시
multi_df = pd.DataFrame(
    np.random.randn(4, 4),
    index=['A', 'B', 'C', 'D'],
    columns=[
        ['One', 'One', 'Two', 'Two'],
        ['First', 'Second', 'First', 'Second']
    ]
)

print(multi_df)
```

```
        One                   Two
      First    Second     First    Second
A  0.298290 -1.362875  0.118668  0.662039
B -1.526280  0.361817 -0.968303 -1.631330
C -0.199253  0.807268  1.024909  0.319309
D -0.153495 -0.495558  1.876642 -0.256473
```

In [ ]:
```
# stack()에서 레벨 지정
stacked_multi = multi_df.stack(level=1)

print("Multi-level Stacked DataFrame:")
print(stacked_multi)
```

```
Multi-level Stacked DataFrame:
                One       Two
A First    0.298290  0.118668
  Second -1.362875  0.662039
B First   -1.526280 -0.968303
  Second   0.361817 -1.631330
C First   -0.199253  1.024909
  Second   0.807268  0.319309
D First   -0.153495  1.876642
  Second -0.495558 -0.256473
```

In [ ]:
```python
# 한번 더 stack() 메서드 적용
stacked_double = stacked_multi.stack()
print("Double Stacked DataFrame:")
print(stacked_double)
```

```
Double Stacked DataFrame:
A  First   One    0.298290
           Two    0.118668
   Second  One   -1.362875
           Two    0.662039
B  First   One   -1.526280
           Two   -0.968303
   Second  One    0.361817
           Two   -1.631330
C  First   One   -0.199253
           Two    1.024909
   Second  One    0.807268
           Two    0.319309
D  First   One   -0.153495
           Two    1.876642
   Second  One   -0.495558
           Two   -0.256473
dtype: float64
```

In [ ]:
```python
# unstack()에서 레벨 지정
unstacked = stacked_double.unstack(0)
print(unstacked)
```

```
                   A         B         C         D
First  One  0.298290 -1.526280 -0.199253 -0.153495
       Two  0.118668 -0.968303  1.024909  1.876642
Second One -1.362875  0.361817  0.807268 -0.495558
       Two  0.662039 -1.631330  0.319309 -0.256473
```

In [ ]:
```python
unstacked_double = unstacked.unstack(0)
print(unstacked_double)
```

```
           A                   B                   C                   D
\
       First    Second     First    Second     First    Second     First
One  0.298290 -1.362875 -1.526280  0.361817 -0.199253  0.807268 -0.153495
Two  0.118668  0.662039 -0.968303 -1.631330  1.024909  0.319309  1.876642


       Second
One -0.495558
Two -0.256473
```

1-4. stack 메서드와 unstack 메서드를 사용하는 주요 목적

두 메서드를 사용하는 주요 목적은 DataFrame의 데이터 구조를 재구성(reshaping)하는 것입니다. 두 메서드를 사용함으로써, 데이터 분석과 시각화 과정에서 필요한 형태로 데이터를 쉽게 변환할 수 있습니다.

[stack() 메서드의 목적]

- 데이터의 차원 축소 : stack() 메서드는 DataFrame의 컬럼을 인덱스로 이동시켜, 데이터를 더 길고 좁은 형태로 만듭니다. 이 과정에서 데이터의 차원이 축소될 수 있으며, 이는 특히 멀티레벨 컬럼을 가진 DataFrame에서 유용합니다.
- 계층적 인덱싱의 활용 : 멀티레벨 컬럼을 가진 DataFrame을 다룰 때, stack()을 사용하면 계층적 인덱싱을 통해 데이터에 대한 세밀한 조회와 분석이 가능해집니다.
- 시계열 데이터 분석 용이성 증가 : 시계열 데이터를 다루는 경우, stack() 메서드를 사용하여 시간에 따른 데이터의 변화를 더 쉽게 분석할 수 있도록 데이터 구조를 조정할 수 있습니다.

[unstack() 메서드의 목적]

- 데이터의 차원 확장 : unstack() 메서드는 인덱스 레벨을 컬럼 레벨로 이동시켜, 데이터를 더 넓고 짧은 형태로 만듭니다. 이는 데이터를 요약하거나 특정 인덱스 기준으로 데이터를 재구성할 때 유용합니다.
- 피벗 테이블 생성과 유사한 효과 : unstack() 메서드를 사용하면 멀티인덱스 데이터를 피벗 테이블 형태로 쉽게 변환할 수 있습니다.
- 데이터 비교와 시각화 용이성 증가 : unstack() 메서드를 사용해 데이터를 구성하면, 여러 그룹의 데이터를 동시에 비교하고 시각화하기가 더 쉬워집니다.

2. melt() 메서드

melt() 메서드는 DataFrame의 데이터를 "길고 좁은" 형태로 재구성하는데 사용됩니다. 이 함수는 여러 컬럼을 하나의 컬럼으로 합치면서, 각각의 값에 대해 두 개의 새로운 컬럼(변수 이름과 값)을 생성합니다.

이는 다양한 데이터 분석 상황에서 유용하게 사용될 수 있으며, 특히 여러 변수가 컬럼으로 분포된 "넓은" 데이터를 "긴" 형태로 변환할 때 매우 유용합니다.

stack() 메서드는 단순히 컬럼을 인덱스로 이동시켜서 멀티인덱스를 가진 결과물을 반환합니다. melt() 메서드는 여러 변수가 각각 다른 컬럼에 저장된 경우에 유용하며, 이러한 변수들을 단일 컬럼으로 "녹여서" 변수명과 값의 쌍을 생성합니다.

사용 방법 : melt() 메서드는 id_vars로 지정된 하나 이상의 컬럼을 기준으로 나머지 컬럼을 변수와 값의 쌍으로 변환합니다. var_name과 value_name 인자를 통해 새로운 변수 이름과 값의 이름을 지정할 수 있습니다.

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

# 샘플 데이터 생성
df = pd.DataFrame({
    'Day': ['Monday', 'Tuesday', 'Wednesday'],
    'Apple': [3, 2, 1],
    'Banana': [1, 3, 5],
    'Cherry': [2, 4, 6]
})
print(df)
```

```
         Day  Apple  Banana  Cherry
0     Monday      3       1       2
1    Tuesday      2       3       4
2  Wednesday      1       5       6
```

In [ ]:
```python
# Day 컬럼을 기준으로 나머지 컬럼들을 변수와 값의 쌍으로 변환
melted_df = pd.melt(
    df,
    id_vars=['Day'],
    var_name='Fruit',
    value_name='Sales'
)

print(melted_df)
```

```
         Day   Fruit  Sales
0     Monday   Apple      3
1    Tuesday   Apple      2
2  Wednesday   Apple      1
3     Monday  Banana      1
4    Tuesday  Banana      3
5  Wednesday  Banana      5
6     Monday  Cherry      2
7    Tuesday  Cherry      4
8  Wednesday  Cherry      6
```

In [ ]:
```
# 동일한 DataFrame에 stack() 메서드를 적용한 경우
print(df.stack())
```

```
0  Day           Monday
   Apple              3
   Banana             1
   Cherry             2
1  Day          Tuesday
   Apple              2
   Banana             3
   Cherry             4
2  Day        Wednesday
   Apple              1
   Banana             5
   Cherry             6
dtype: object
```

In [ ]:
```
# 'Day' 컬럼을 인덱스로 변경하고, stack()메서드를 적용하여 melt() 메서드를 적용한 것과 유
modified_df = df.set_index('Day')
stacked_mdf = modified_df.stack()
print(stacked_mdf)
```

```
Day
Monday     Apple     3
           Banana    1
           Cherry    2
Tuesday    Apple     2
           Banana    3
           Cherry    4
Wednesday  Apple     1
           Banana    5
           Cherry    6
dtype: int64
```

3. pivot() 메서드

3-1. pivot() 메서드 소개 및 사용 예시

pivot() 메서드는 DataFrame의 데이터를 재구성하여, 특정 컬럼의 고유값들을 새로운 컬럼의 헤더로 사용하는 "넓은" 형태의 데이터를 생성합니다.

In [ ]:
```
# 샘플 데이터 생성
data = {
    'Date': ['2023-01-01', '2023-01-01', '2023-01-02',
             '2023-01-02', '2023-01-03', '2023-01-03'],
    'City': ['New York', 'Los Angeles', 'New York',
             'Los Angeles', 'New York', 'Los Angeles'],
    'Temperature': [59, 65, 57, 70, 60, 72],
    'Humidity': [56, 50, 54, 48, 55, 45]
}
df = pd.DataFrame(data)

print(df)
```

```
         Date         City  Temperature  Humidity
0  2023-01-01     New York           59        56
1  2023-01-01  Los Angeles           65        50
2  2023-01-02     New York           57        54
3  2023-01-02  Los Angeles           70        48
4  2023-01-03     New York           60        55
5  2023-01-03  Los Angeles           72        45
```

In [ ]:
```
pivot_df = df.pivot(index='Date', columns='City', values='Temperature')

print(pivot_df)
```

```
City        Los Angeles  New York
Date
2023-01-01           65        59
2023-01-02           70        57
2023-01-03           72        60
```

위 예시에서 :

- index는 새로운 DataFrame에서 인덱스로 사용될 컬럼을 지정합니다. 여기서는 'Date'가 사용되었습니다.
- columns는 새로운 컬럼으로 변환될, 고유값을 가진 컬럼을 지정합니다. 여기서는 'City'가 사용되었습니다.
- values는 새로운 피벗 테이블에서 데이터로 채워질 값을 가진 컬럼을 지정합니다. 여기서는 'Temperature'가 사용되었습니다.

3-2. pivot() 메서드와 pivot_table() 메서드 간의 차이점

- pivot() 메서드는 DataFrame의 형태를 변경하고자 할 때 사용되며, 단순한 데이터의 형태만 변환합니다. pivot은 주어진 DataFrame에서 중복된 인덱스/컬럼 쌍이 없어야 합니다. 중복 값이 있을 경우 에러를 발생시킵니다.
- pivot_table() 메서드는 pivot()과 유사하게 데이터를 재구조화하지만, 집계 함수를 사용하여 중복된 데이터를 요약할 수 있습니다. 이는 복잡한 데이터 집계와 요약에 적합합니다. 중복된 인덱스/컬럼 쌍이 있는 경우에도 사용할 수 있으며, aggfunc 매개변수를 통해 중복 값을 어떻게 처리할지 결정할 수 있습니다.
- 결과적으로 데이터에 중복 값이 있고 집계(aggfunc)가 필요한 상황에서는 'pivot_table'을 사용해야 하며, 단순히 형태를 변환하고자 하는 경우에는 'pivot'을 사용해도 괜찮습니다.