[3-5. 피벗테이블과 그룹화(pivot_table, groupby)]

1. pivot_table 함수

pivot_table 함수는 pandas에서 복잡한 데이터를 집계하고, 데이터를 재구성하는데 사용됩니다. 이 함수를 사용하면 데이터를 특정 키에 따라 그룹화하고, 각 그룹에 대해 다양한 집계 함수를 적용할 수 있습니다.

1-1. 기본 사용법

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

df = pd.DataFrame({
    'A': ['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'foo'],
    'B': ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
    'C': np.random.randn(8),
    'D': np.random.randn(8)
})

# pivot_table 생성
pivot_table = pd.pivot_table(
    df,
    values='D',
    index=['A'],
    columns=['B'],
    aggfunc='sum'
)

print(pivot_table)
```

```
B         one     three       two
A
bar -0.508109  2.758640 -0.102677
foo -0.095130  0.458155  0.416187
```

1-2. 주요 매개변수

- data: DataFrame 객체. pivot_table을 생성할 데이터프레임입니다.
- values: 재구성된 테이블에서 집계할 열 이름을 나타내는 스칼라 또는 리스트. 지정하지 않으면, 모든 수치형 데이터를 집계합니다.
- index: 반환된 피벗 테이블의 행으로 들어갈 DataFrame 열 이름이나 다른 그룹 키를 나타내는 스칼라 또는 리스트입니다. 이는 결과 피벗 테이블에서 그룹화의 기준이 됩니다.
- columns: 반환된 피벗 테이블의 열로 들어갈 DataFrame 열 이름이나 다른 그룹 키를 나타내는 스칼라 또는 리스트입니다. 이는 선택적 매개변수이며, 지정하지 않으면 최종 결과는 MultiIndex를 가진 DataFrame이 됩니다.
- aggfunc: 집계 함수나 함수의 리스트입니다. 기본값은 numpy.mean입니다. 집계 함수는 'sum', 'mean', 'count', 'min', 'max' 등 numpy 함수나 사용자 정의 함수를 사용할 수 있습니다. values에 여러 항목이 있으면, 딕셔너리를 사용하여 열마다 다른 집계 함수를 지정할 수 있습니다.
- fill_value: 결과 테이블에서 나타날 결측값(missing value)을 대체할 값입니다. 기본값은 None입니다. 이 매개변수는 집계 과정에서 발생할 수 있는 NaN 값을 원하는 값으로 채우는 데 사용됩니다.
- margins: 부분합계와 총합을 추가할지 여부를 결정하는 불리언 값입니다. 기본값은 False입니다. True로 설정하면, 모든 행과 열에 대한 총합계가 'All' 라벨과 함께 추가됩니다.
- dropna: 결측값을 가진 열을 결과에서 제외할지 여부를 결정하는 불리언 값입니다. 기본값은 True로, 결측값을 가진 모든 열을 포함하지 않습니다.
- margins_name: margins=True일 때, 마진 열이나 행의 이름을 지정합니다. 기본값은 'All'입니다.

In [ ]:
```
df = pd.DataFrame({
    "A": ["foo", "foo", "foo", "foo", "foo", "bar", "bar", "bar", "bar"],
    "B": ["one", "one", "one", "two", "two", "one", "one", "two", "two"],
    "C": ["small", "large", "large", "small", "small", "large", "small",
          "small", "large"],
    "D": [1, 2, 2, 3, 3, 4, 5, 6, 7],
    "E": [2, 4, 5, 5, 6, 6, 8, 9, 9]
})

pivot_table = pd.pivot_table(
    df, values="D", index=["A", "B"], columns=["C"],
    aggfunc='sum', fill_value=0,
    margins=True, margins_name="Total"
)

print(pivot_table)
```

```
C          large  small  Total
A     B
bar   one      4      5      9
      two      7      6     13
foo   one      4      1      5
      two      0      6      6
Total         15     18     33
```

1-3. aggfunc 매개변수 사용 예시

pivot_table에서 aggfunc 매개변수는 데이터를 집계하는 방법을 지정하는 데 사용됩니다. aggfunc 매개변수에는 단일 함수를 사용할 수도 있고, 여러 함수를 리스트나 딕셔너리 형태로 제공하여 다양한 집계를 한 번에 수행할 수도 있습니다.

1-3-1. 단일 집계 함수 사용

In [ ]:
```
df = DataFrame({
    'Category': ['A', 'A', 'B', 'B'],
    'Values': [10, 15, 10, 20]
})

# 평균을 집계 함수로 사용
pivot = pd.pivot_table(
    df, values='Values', index='Category', aggfunc='mean'
)
print(pivot)
```

```
          Values
Category
A           12.5
B           15.0
```

1-3-2. 여러 집계 함수 사용

여러 함수를 리스트로 제공하여 Values에 대해 각각의 집계를 수행할 수 있습니다.

In [ ]:
```
# 평균, 최소값, 최대값을 집계 함수로 사용
pivot = pd.pivot_table(
    df, values='Values', index='Category',
    aggfunc=['mean', 'min', 'max']
)
print(pivot)
```

```
           mean    min    max
         Values Values Values
Category
A          12.5     10     15
B          15.0     10     20
```

1-3-3. 딕셔너리를 사용하여 열별로 다른 집계 함수 적용

aggfunc에 딕셔너리를 사용하여 서로 다른 열에 대해 다른 집계 함수를 적용할 수 있습니다.

In [ ]:
```
# 샘플 데이터 확장
df['Quantity'] = [1, 2, 3, 4]

# Values는 평균을, Quantity는 합계를 집계 함수로 사용
pivot = pd.pivot_table(
    df, values=['Values', 'Quantity'], index='Category',
    aggfunc={'Values': 'mean', 'Quantity': 'sum'}
)
print(pivot)
```

```
          Quantity  Values
Category
A                3    12.5
B                7    15.0
```

1-3-4. 주요 집계 함수(aggfunc)

- 'mean' : 평균 값을 계산합니다.
- 'median' : 중앙값을 계산합니다.
- 'sum' : 합계를 계산합니다.
- 'std' : 표준 편차를 계산합니다.
- 'var' : 분산을 계산합니다.
- 'min' : 최소값을 찾습니다.
- 'max' : 최대값을 찾습니다.
- 'count' : 0이 아닌 값의 개수를 세거나, 특정 조건을 만족하는 항목의 수를 계산합니다.
- 'nunique' : 고유값의 개수를 세어줍니다.
- 'first' 또는 'last' : 각 그룹의 첫번째 또는 마지막 값에 접근합니다.

In [ ]:
```
pivot = pd.pivot_table(
    df, values='Values',
    index='Category', aggfunc='mean'
)
print(pivot)
```

```
          Values
Category
A           12.5
B           15.0
```

In [ ]:
```
pivot = pd.pivot_table(
    df, values='Values', index='Category', aggfunc='std'
)
print(pivot)
```

```
            Values
Category
A         3.535534
B         7.071068
```

1-3-5. 사용자 정의 집계 함수

aggfunc 매개변수는 사용자 정의 함수도 받을 수 있습니다. 이를 통해 더 복잡한 집계 로직을 구현할 수 있습니다.

사용자 정의 함수는 DataFrame의 각 컬럼(또는 values에 지정된 컬럼)에 대해 호출되며, 해당 컬럼의 데이터를 인자로 받아 집계된 값을 반환해야 합니다.

In [ ]:
```
def range_func(series):
    return series.max() - series.min()

pivot_table = pd.pivot_table(
    df, values='Values', index='Category',
    aggfunc=range_func
)

print(pivot_table)
```

```
          Values
Category
A              5
B             10
```

2. groupby 함수

pandas의 groupby 함수를 사용하면 데이터를 그룹화하여 집계, 변환, 필터링 하는 작업을 쉽게 수행할 수 있습니다.

이 함수는 SQL의 'GROUP BY' 명령어와 유사한 방식으로 작동하며, 하나 이상의 컬럼을 기준으로 데이터를 그룹화하고, 각 그룹에 대해 집계 함수를 적용합니다.

2-1. 기본 작동 원리

groupby는 크게 세 단계로 작동합니다. 분할(Split), 적용(Apply), 결합(Combine).

- 분할(Split) : 데이터를 특정 기준에 따라 여러 그룹으로 분할합니다.
- 적용(Apply) : 각 그룹에 대해 집계, 변환, 필터링 등의 연산을 적용합니다.
- 결합(Combine) : 연산의 결과를 하나의 데이터 구조로 결합합니다.

In [ ]:
```
df.groupby(
    by=None, axis=0, level=None, as_index=True, sort=True,
    group_keys=True, dropna=True
)
```

[주요 매개변수]

- by : 그룹화할 기준을 설정합니다. 컬럼 이름이나, 컬럼을 선택하는 함수, 컬럼 이름의 리스트 등을 사용할 수 있습니다.
- axis : 0은 행을 기준으로 그룹화하고, 1은 열을 기준으로 그룹화합니다.
- level : 멀티인덱스인 경우, 인덱스의 라벨을 기준으로 그룹화합니다.
- as_index : True일 경우, 그룹 라벨을 인덱스로 사용합니다. False일 경우, 그룹 라벨이 인덱스로 사용되지 않고, 데이터 컬럼 중 하나로 남습니다.
- sort : 그룹 키에 따라 정렬할지 여부를 결정합니다.
- group_keys : 기본값은 True입니다. 이를 False로 설정하면, groupby에 의해 반환된 객체에서 그룹 키가 인덱스에 추가되지 않습니다. 그룹화 작업 후, 결과 DataFrame에서 그룹화를 위해 사용된 컬럼이 멀티인덱스의 일부로 포함되지 않게 합니다.
- dropna : 기본값은 True입니다. False로 설정하면, 그룹화하는 과정에서 NA(null) 값을 하나의 그룹으로 간주합니다. 이는 데이터 분석 시 NA 값을 따로 분류하고 싶을 때 유용할 수 있습니다.

2-2. 기본 사용 예시

2-2-1. 데이터 준비

In [ ]:
```
# 샘플 데이터 생성
data = {
    'City': ['Seoul', 'Seoul', 'Seoul', 'Busan', 'Busan', 'Daegu'],
    'Year': [2020, 2021, 2022, 2020, 2021, 2022],
    'Population': [9904312, 9853972, 9836486, 3448737, 3404423, 3428568],
    'Temperature': [11.0, 11.5, 12.0, 14.0, 14.5, 15.0]
}
df = DataFrame(data)
```

In [ ]:
```
df
```

Out[ ]:

	City	Year	Population	Temperature
0	Seoul	2020	9904312	11.0
1	Seoul	2021	9853972	11.5
2	Seoul	2022	9836486	12.0
3	Busan	2020	3448737	14.0
4	Busan	2021	3404423	14.5
5	Daegu	2022	3428568	15.0

2-2-2. grouped 객체 생성

In [ ]:
```
grouped = df.groupby('City')
```

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

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x104b7d010>
```

groupby 메서드를 통해서 그룹화된 객체를 일반적으로 'grouped' 객체라고 합니다. 현재 grouped 변수에 대입된 grouped 객체는 실제 데이터를 그룹화된 형태로 포함하지 않지만, 각 그룹에 대한 정보와 그룹별로 연산을 수행할 수 있는 메커니즘을 제공합니다.

2-2-3. 단일 열을 기준으로 그룹화

각 도시의 인구('Population')의 평균 계산

In [ ]:
```
grouped = df.groupby('City')
average_population = grouped['Population'].mean()
print(average_population)
```

```
City
Busan    3.426580e+06
Daegu    3.428568e+06
Seoul    9.864923e+06
Name: Population, dtype: float64
```

2-2-4. 여러 열을 기준으로 그룹화

'City'와 'Year' 열을 기준으로 데이터를 그룹화하고, 각 그룹의 최대 온도('Temperature')를 계산합니다.

In [ ]:
```
grouped = df.groupby(['City', 'Year'])
max_temperature = grouped['Temperature'].max()
print(max_temperature)
```

```
City   Year
Busan  2020    14.0
       2021    14.5
Daegu  2022    15.0
Seoul  2020    11.0
       2021    11.5
       2022    12.0
Name: Temperature, dtype: float64
```

In [ ]:
```
df
```

Out[ ]:

	City	Year	Population	Temperature
0	Seoul	2020	9904312	11.0
1	Seoul	2021	9853972	11.5
2	Seoul	2022	9836486	12.0
3	Busan	2020	3448737	14.0
4	Busan	2021	3404423	14.5
5	Daegu	2022	3428568	15.0

2-2-5. 집계 함수 사용

agg() 함수를 사용하여 각 그룹에 여러 집계 함수를 적용할 수 있습니다. 예를 들어, 도시별 최대 인구와 평균 온도를 계산할 수 있습니다.

In [ ]:
```
aggregated = df.groupby('City').agg({
    'Population': 'max',
    'Temperature': 'mean'
})
print(aggregated)
```

```
       Population  Temperature
City
Busan     3448737        14.25
Daegu     3428568        15.00
Seoul     9904312        11.50
```

2-2-6. 사용자 정의 함수 적용

agg() 함수에 사용자 정의 함수를 전달하여 그룹별로 적용할 수 있습니다. 예를 들어, 각 도시의 인구 변화를 계산하는 함수를 적용해 봅니다.

In [ ]:
```
def population_change(series):
    return series.max() - series.min()

change = df.groupby('City')['Population'].agg(population_change)
print(change)
```

```
City
Busan    44314
Daegu        0
Seoul    67826
Name: Population, dtype: int64
```

2-3. 다양한 사용 방법 정리

2-3-1. 데이터 준비

In [ ]:
```
# 샘플 판매 데이터 생성
data = {
    'Date': ['2023-01-01', '2023-01-01', '2023-01-02',
             '2023-01-02', '2023-01-03', '2023-01-03'],
    'Product': ['Apple', 'Banana', 'Apple',
             'Banana', 'Apple', 'Banana'],
    'Category': ['Fruit', 'Fruit', 'Fruit',
             'Fruit', 'Fruit', 'Fruit'],
    'Quantity': [5, 7, 8, 4, 6, 10],
    'Price': [1.0, 0.5, 1.0, 0.5, 1.0, 0.5]
}
df = DataFrame(data)

print(df)
```

```
         Date Product Category  Quantity  Price
0  2023-01-01   Apple    Fruit         5    1.0
1  2023-01-01  Banana    Fruit         7    0.5
2  2023-01-02   Apple    Fruit         8    1.0
3  2023-01-02  Banana    Fruit         4    0.5
4  2023-01-03   Apple    Fruit         6    1.0
5  2023-01-03  Banana    Fruit        10    0.5
```

2-3-2. 단일 열 기준 그룹화 및 집계

In [ ]:
```
total_sales_by_date = df.groupby('Date')['Quantity'].sum()
print(total_sales_by_date)
```

```
Date
2023-01-01    12
2023-01-02    12
2023-01-03    16
Name: Quantity, dtype: int64
```

2-3-3. 여러 열 기준 그룹화 및 집계

In [ ]:
```
total_sales_by_date_product = df.groupby(
    ['Date', 'Product']
)['Quantity'].sum()
print(total_sales_by_date_product)
```

```
Date        Product
2023-01-01  Apple       5
            Banana      7
2023-01-02  Apple       8
            Banana      4
2023-01-03  Apple       6
            Banana     10
Name: Quantity, dtype: int64
```

2-3-4. 집계 함수 여러 개 적용

In [ ]:
```
stats_by_product = df.groupby(
    'Product'
).agg({'Quantity': 'sum', 'Price': 'mean'})
print(stats_by_product)
```

```
         Quantity  Price
Product
Apple          19    1.0
Banana         21    0.5
```

2-3-5. 사용자 정의 집계 함수 적용

In [ ]:
```
def max_min_diff(series):
    return series.max() - series.min()

diff_by_product = df.groupby('Product')['Quantity'].agg(max_min_diff)
print(diff_by_product)
```

```
Product
Apple     3
Banana    6
Name: Quantity, dtype: int64
```

2-3-6. transform 함수 사용

- transform 함수는 그룹화된 데이터에 함수를 적용하고, 원본 데이터프레임과 동일한 크기의 결과를 반환합니다.
- 이는 그룹별 계산 결과를 원 데이터프레임의 각 행에 추가 정보로 삽입하고 싶을 때 유용합니다.

In [ ]:
```
df['Total_Quantity'] = df.groupby('Product')['Quantity'].transform('sum')
df['Sale_Ratio'] = df['Quantity'] / df['Total_Quantity']
print(df)
```

```
         Date Product Category  Quantity  Price  Total_Quantity  Sale_Rat
io
0  2023-01-01   Apple    Fruit         5    1.0              19    0.2631
58
1  2023-01-01  Banana    Fruit         7    0.5              21    0.3333
33
2  2023-01-02   Apple    Fruit         8    1.0              19    0.4210
53
3  2023-01-02  Banana    Fruit         4    0.5              21    0.1904
76
4  2023-01-03   Apple    Fruit         6    1.0              19    0.3157
89
5  2023-01-03  Banana    Fruit        10    0.5              21    0.4761
90
```

2-3-7. 필터링을 통한 그룹 선택

- 그룹별로 조건을 설정하여 해당 조건을 통과한 그룹만 반환
- 예시 : 평균 판매량이 6 이상인 제품만 필터링

In [ ]:
```
df.groupby('Product')['Quantity'].mean()
```

Out[ ]:
```
Product
Apple     6.333333
Banana    7.000000
Name: Quantity, dtype: float64
```

In [ ]:
```
filtered_products = df.groupby(
    'Product'
).filter(
    lambda x: x['Quantity'].mean() > 6.5
)
print(filtered_products)
```

```
         Date Product Category  Quantity  Price  Total_Quantity  Sale_Rat
io
1  2023-01-01  Banana    Fruit         7    0.5              21    0.3333
33
3  2023-01-02  Banana    Fruit         4    0.5              21    0.1904
76
5  2023-01-03  Banana    Fruit        10    0.5              21    0.4761
90
```