데이터분석을위한 PANDAS

그룹연산

인덱스활용

◆ 그룹연산

- **특정 기준 적용하여 몇 개의 그룹으로 분할하여 처리하는 것**
- **데이터 집계, 변환, 필터링에 효율적**

- **속성 (Attribute)**
  - → names : 멀티인덱스의 레벨명들
  - → nleves : 멀티인덱스의 레벨 수
  - → levels : 멀티인덱스 레벨들
  - → levshape : 각 레벨의 길이
  - → dtypes : 타입 정보

인덱스활용

◆ 그룹연산

▪ 처리 프로세스

① 분할(Split) : 데이터를 특정 조건에 의해 분할

② 적용(apply) : 데이터 집계, 변환, 필터링에 필요한 메서드 적용

③ 결합(Combine) : 2단계의 처리 결과 하나로 결합

인덱스활용

◆ 그룹연산

▪ GroupBy Object

→ DataFrame/Series.groupby() 의해서 반환되는 반복 가능한 객체

→ 그룹 키와 데이터로 Dict형태로 구성

→ 다양한 연산 메서드 사용

		age	fare	survived
class	sex			
First	female	34.611765	106.125798	0.968085
	male	41.281386	67.226127	0.368852
Second	female	28.722973	21.970121	0.921053
	male	30.740707	19.741782	0.157407
Third	female	21.750000	16.118810	0.500000

인덱스활용

◆ 그룹연산

▪ GroupBy Object Attribute

obj	속 성	• GroupBy가 생성된 원본 DataFrame/Series 반환	g.obj
groups	속 성	• 각 그룹 이름(키)과 해당 인덱스 목록 딕셔너리 반환	g.groups
ngroups	속 성	• 그룹의 개수 반환	g.ngroups
size()	메서드	• 각 그룹의 행 개수 반환 (Series 형태)	g.size()
indices	속 성	• 각 그룹의 인덱스 위치 정보를 dict로 반환	g.indices
dtypes	속 성	• 그룹 내 열들의 dtype 정보 반환	g.dtypes
group_keys	속 성	• 그룹 키 표시 여부(True/False)	g.group_keys
as_index	속 성	• 그룹 키를 인덱스로 쓸지 여부	g.as_index

인덱스활용

◆ 그룹연산

```
## 모듈 로딩
import pandas as pd

## DataFrame 인스턴스 생성
df = pd.DataFrame({
    '지점': ['서울','서울','서울','부산','부산','대구','대구','대구'],
    '상품': ['A','A','B','A','B','A','B','B'],
    '매출': [100,150,200,120,180,90,210,240],
    '수량': [10,15,20,12,18,9,21,24]
})

## 구조 확인
display(df)
```

	지점	상품	매출	수량
0	서울	A	100	10
1	서울	A	150	15
2	서울	B	200	20
3	부산	A	120	12
4	부산	B	180	18
5	대구	A	90	9
6	대구	B	210	21
7	대구	B	240	24

인덱스활용

◆ 그룹연산

```
## 지점별 그룹화
g = df.groupby('지점')

## 그룹 속성 확인
print( f"-> 그룹수 : { g.ngroups}개" )        # 그룹 수
print( f"-> 그룹별 : {g.groups}"  )           # 각 그룹의 인덱스
print( f"-> 그룹별 행 수₩n{g.size()}" )       # 그룹별 행 수
```

```
-> 그룹수 : 3개
-> 그룹별 : {'대구': [5, 6, 7], '부산': [3, 4], '서울': [0, 1, 2]}
-> 그룹별 행 수
지점
대구    3
부산    2
서울    3
dtype: int64
```

인덱스활용

◆ 그룹연산

```
## 그룹별 데이터 추출
for k in g.groups.keys():
    print(f'₩n[{k}]======')
    print(g.get_group(k))
```

	지점	상품	매출	수량
0	서울	A	100	10
1	서울	A	150	15
2	서울	B	200	20
3	부산	A	120	12
4	부산	B	180	18
5	대구	A	90	9
6	대구	B	210	21
7	대구	B	240	24

지점별

```
[대구]=====
   지점 상품   매출  수량
5  대구  A    90    9
6  대구  B   210   21
7  대구  B   240   24

[부산]=====
   지점 상품   매출  수량
3  부산  A   120   12
4  부산  B   180   18

[서울]=====
   지점 상품   매출  수량
0  서울  A   100   10
1  서울  A   150   15
2  서울  B   200   20
```

인덱스활용

◆ 그룹연산

- **GroupBy Object Method – 기본 집계(Aggregation)**

sum()	• 그룹별 합계	g.sum()
mean()	• 그룹별 평균	g.mean()
median()	• 그룹별 중앙값	g.median()
min() / max()	• 그룹별 최소/최대값	g.max()
count()	• 각 그룹의 개수	g.count()
size()	• 그룹의 전체 행 수	g.size()
prod()	• 그룹별 곱(product)	g.prod()
std() / var()	• 그룹별 표준편차/분산	g.std()
describe()	• 그룹별 통계 요약	g.describe()
agg() / aggregate()	• 여러 함수로 동시에 집계	g.agg({'col1':'sum', 'col2':['mean','max']})

인덱스활용

◆ 그룹연산

```
## (1) 그룹별 합계
print("합계===>₩n", g.sum(numeric_only=True))
```

```
합계===>
        매출   수량
지점
대구   540   54
부산   300   30
서울   450   45
```

```
## (2) 평균
print("평균===>₩n", g.mean(numeric_only=True))
```

```
평균===>
          매출    수량
지점
대구   180.0  18.0
부산   150.0  15.0
서울   150.0  15.0
```

인덱스활용

◆ 그룹연산

```
## (3) 여러 집계 동시 적용
print( g.agg( {'매출':['sum','mean'], '수량':['max','min']} ) )
```

```
[대구]=====
   지점 상품   매출  수량
5  대구  A    90   9
6  대구  B   210  21
7  대구  B   240  24

[부산]=====
   지점 상품   매출  수량
3  부산  A   120  12
4  부산  B   180  18

[서울]=====
   지점 상품   매출  수량
0  서울  A   100  10
1  서울  A   150  15
2  서울  B   200  20
```

지점별

	매출		수량	
	sum	mean	max	min
지점				
대구	540	180.0	24	9
부산	300	150.0	18	12
서울	450	150.0	20	10

인덱스활용

◆ 그룹연산

```
## (4)지점의 상품별 매출 합계와 평균
for key in g.groups.keys():
    print(f'₩n[{key}]=====')
    localDF = g.get_group(key).drop(columns='지점').reset_index(drop=True)

    localDF=localDF.groupby('상품')
    display( localDF.agg({'매출':['sum','mean']}) )
```

[대구]=====

	매출	
	sum	mean
상품		
A	90	90.0
B	450	225.0

[부산]=====

	매출	
	sum	mean
상품		
A	120	120.0
B	180	180.0

[서울]=====

	매출	
	sum	mean
상품		
A	250	125.0
B	200	200.0

인덱스활용

◆ 그룹연산

```
## 지점별 상품에 따른 매출 합계와 평균 집계
g = df.groupby(['지점','상품'])

## 그룹별 데이터 추출
for k in g.groups.keys():
    print(f'\n[{k}]=====')
    print(g.get_group(k))
```

```
[('대구', 'A')]=====
   지점 상품  매출  수량
5  대구  A  90   9

[('대구', 'B')]=====
   지점 상품   매출  수량
6  대구  B  210  21
7  대구  B  240  24

[('부산', 'A')]=====
   지점 상품   매출  수량
3  부산  A  120  12

[('부산', 'B')]=====
   지점 상품   매출  수량
4  부산  B  180  18
```

```
[('서울', 'A')]=====
   지점 상품   매출  수량
0  서울  A  100  10
1  서울  A  150  15

[('서울', 'B')]=====
   지점 상품   매출  수량
2  서울  B  200  20
```

인덱스활용

◆ 그룹연산

- **GroupBy Object Method – 구조/정보 확인(Inspection)**

get_group(name)	• 특정 그룹만 추출	g.get_group('서울')
first() / last()	• 각 그룹의 첫/마지막 행	g.first()
nth(n)	• 각 그룹의 n번째 행	g.nth(1)
head(n) / tail(n)	• 각 그룹의 앞/뒤 n개 행	g.head(2)
indices	• 그룹별 행 인덱스 딕셔너리 / 속성	g.indices

인덱스활용

◆ 그룹연산

- **GroupBy Object Method – 그룹 통계 (Advanced)**

corr() / cov()	• 그룹별 상관관계/공분산 계산	g.corr()
quantile(q)	• 그룹별 분위수	g.quantile(0.25)
value_counts()	• 각 그룹 내 값 빈도 계산	g['컬럼'].value_counts()
sample(n)	• 그룹별 표본 추출	g.sample(2)

인덱스활용

◆ 그룹연산

- 속성 Attribute
  - ✓ **GroupBy.groups** : dict 타입 그룹별 행 인덱스
  - ✓ **GroupBy.indices** : 그룹별 인덱스 배열 정보

- 그룹별 DataFrame
  - ✓ **GroupBy.get_group(name)** : 반환값 : 지정된 그룹의 DF

인덱스활용

◆ 그룹연산

▪ 집계(Aggregation)

- **그룹 객체에 다양한 연산 적용하는 것**
- GroupBy 객체 메서드

  → mean(), sum(), max(), min(), count(), size()

  → var(), std(), describe(), info(), last(), agg()

- 매핑함수 **agg(함수 or 함수명)**

  → 그룹별 연산 가능 컬럼 데이터만 연산 실행

인덱스활용

◆ 그룹연산

```
## 지점별 상품에 따른 매출 합계와 평균 집계
g.agg(['sum','mean'])
```

		매출		수량	
		sum	mean	sum	mean
지점	상품				
대구	A	90	90.0	9	9.0
	B	450	225.0	45	22.5
부산	A	120	120.0	12	12.0
	B	180	180.0	18	18.0
서울	A	250	125.0	25	12.5
	B	200	200.0	20	20.0

인덱스활용

◆ 그룹연산

▪ 변환(Transformation)

- 그룹별로 구분하여 각 원소에 함수 적용
- 계산한 결과를 원래 데이터의 행 개수 그대로 반환
- 용 도 : 그룹 평균 대비 비율, 표준화, 순위 계산, z-score

- 매핑함수 **transform(함수 or 함수명)**

  → 그룹별 연산 가능 컬럼 데이터만 연산 실행

인덱스활용

◆ 그룹연산

```
## DataFrame 인스턴스 생성
df = pd.DataFrame({
    '지점': ['서울','서울','서울','부산','부산','부산','대구','대구','대구'],
    '매출': [100,150,200,120,180,160,90,210,240]
})

## 확인
display(df)
```

	지점	매출
0	서울	100
1	서울	150
2	서울	200
3	부산	120
4	부산	180
5	부산	160
6	대구	90
7	대구	210
8	대구	240

인덱스활용

◆ 그룹연산

```
## ----------------------------------------
## (1) 그룹별 평균 매출 대비 비율
## ----------------------------------------
df['매출비율'] = df.groupby('지점')['매출']
                   .transform(lambda x: x / x.mean())

## 같은 지점 내에서 매출이
## 평균보다 높은지/낮은지 한눈에 파악
display(df)
```

	지점	매출	매출비율
0	서울	100	0.666667
1	서울	150	1.000000
2	서울	200	1.333333
3	부산	120	0.782609
4	부산	180	1.173913
5	부산	160	1.043478
6	대구	90	0.500000
7	대구	210	1.166667
8	대구	240	1.333333

인덱스활용

◆ 그룹연산

```
## ============================================================
## (2) 그룹별 표준화 (z-score)
##     =>데이터의 상대적인 크기 맞추어 공정 비교 할 수 있음
## ============================================================
## - 같은 지점 내에서 평균보다 높은 값은 +, 낮은 값은 -
## - 각 지점별 데이터의 상대적 위치(편차)를 확인할 수 있음
## - 모든 지점별로 평균은 0, 표준편차는 1로 맞춰짐
df['표준화'] = df.groupby('지점')['매출']
                 .transform(lambda x: (x - x.mean()) / x.std())
display(df)
```

	지점	매출	매출비율	표준화
0	서울	100	0.666667	-1.000000
1	서울	150	1.000000	0.000000
2	서울	200	1.333333	1.000000
3	부산	120	0.782609	-1.091089
4	부산	180	1.173913	0.872872
5	부산	160	1.043478	0.218218
6	대구	90	0.500000	-1.133893
7	대구	210	1.166667	0.377964
8	대구	240	1.333333	0.755929

인덱스활용

◆ 그룹연산

```
## ========================================================
## (3) 그룹별 순위(rank)
## ========================================================
df['순위'] = df.groupby('지점')['매출']
               .transform(lambda x: x.rank(ascending=False))

display(df)
```

	지점	매출	매출비율	표준화	순위
0	서울	100	0.666667	-1.000000	3.0
1	서울	150	1.000000	0.000000	2.0
2	서울	200	1.333333	1.000000	1.0
3	부산	120	0.782609	-1.091089	3.0
4	부산	180	1.173913	0.872872	1.0
5	부산	160	1.043478	0.218218	2.0
6	대구	90	0.500000	-1.133893	3.0
7	대구	210	1.166667	0.377964	2.0
8	대구	240	1.333333	0.755929	1.0

멀티인덱스

인덱스활용

◆ 멀티인덱스

- **2개 이상으로 구성된 행/열 인덱스**
- 행이나 열에 여러 계층 가지는 인덱스 => **계층적 데이터 적용**

- **속성 (Attribute)**
  - names : 멀티인덱스의 레벨명들
  - nleves : 멀티인덱스의 레벨 수
  - levels : 멀티인덱스 레벨들
  - levshape : 각 레벨의 길이
  - dtypes : 타입 정보

인덱스활용

◆ 멀티인덱스

- **Level : 인덱스를 식별하기 위한 것**

level 0	level 1	age	fare	survived
class	sex			
First	female	34.611765	106.125798	0.968085
	male	41.281386	67.226127	0.368852
Second	female	28.722973	21.970121	0.921053
	male	30.740707	19.741782	0.157407
Third	female	21.750000	16.118810	0.500000

인덱스활용

◆ 멀티인덱스

▪ 행 멀티인덱스 생성

```
values   = [ [1,2,3],     [4,5,6],     [7,8,9],
             [10,11,12],[13,14,15], [16,17,18] ]

index_   = [ ['row1','row1','row2','row2','row3','row3'],
             ['val1','val2','val1','val2','val2','val3'] ]

columns_ = ['col1', 'col2', 'col3']

## DF 생성
df = pd.DataFrame(values,
                  columns=columns_,
                  index=index_)

## MultiIndex 저장
mIndex = df.index
```

		col1	col2	col3
row1	val1	1	2	3
	val2	4	5	6
row2	val1	7	8	9
	val2	10	11	12
row3	val2	13	14	15
	val3	16	17	18

인덱스활용

◆ 멀티인덱스

▪ 행 멀티인덱스 생성

```
## 행 MultiIndex 속성들
print(f'levels      : {mIndex.levels}')
print(f'levshape    : {mIndex.levshape}')
print(f'nlevels     : {mIndex.nlevels}개')
print(f'name        : {mIndex.name}')
print(f'dtypes      ₩n{mIndex.dtypes}')
```

```
levels      : [['row1', 'row2', 'row3'], ['val1', 'val2', 'val3']]
levshape    : (3, 3)
nlevels     : 2
name        : None
dtypes
level_0    object
level_1    object
dtype: object
```

인덱스활용

◆ 멀티인덱스

▪ 행 멀티인덱스 - 행 선택

df.loc[(level0 , level1)] : 레벨0, 레벨1이 같은 1개 행 선택

df.loc[level0] : 레벨0이 같은 행들 선택

인덱스활용

◆ 멀티인덱스

▪ 행 멀티인덱스 - 행 선택

```
print( df.loc[("row1","val1")] )

print( df.loc["row1"] )
```

```
df.loc[("row1","val1")]
col1    1
col2    2
col3    3
Name: (row1, val1), dtype: int64
```

```
df.loc["row1"]
      col1  col2  col3
val1     1     2     3
val2     4     5     6
```

인덱스활용

◆ 멀티인덱스

▪ 열 멀티인덱스 생성

```
## 데이터
values = [ [1,2,3], [4,5,6], [7,8,9], [10,11,12],
           [13,14,15], [16,17,18], [19,20,21] ]

## DataFrame 생성
df = pd.DataFrame(  values,
                    columns=[ [ 'A', 'B','B'],
                              ['col1', 'col1', 'col2'] ] )
## 출력
print(df)
```

	A	B	
	col1	col1	col2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12
4	13	14	15
5	16	17	18
6	19	20	21

인덱스활용

◆ 멀티인덱스

▪ 열 멀티인덱스 생성

```
## 열 MultiIndex 속성들
print(f'levels      : {mCols.levels}')
print(f'levshape    : {mCols.levshape}')
print(f'nlevels     : {mCols.nlevels}개')
print(f'name        : {mCols.name}')
print(f'dtypes       ₩n{mCols.dtypes}')
```

```
levels      : [['A', 'B'], ['col1', 'col2']]
levshape    : (2, 2)
nlevels     : 2개
name        : None
dtypes
level_0    object
level_1    object
dtype: object
```

인덱스활용

◆ 멀티인덱스

▪ 열 멀티인덱스 – 열/컬럼 선택

df[(level0, level1)] : 레벨0, 레벨1이 같은 열/컬럼 선택

df[level0] : 레벨0이 같은 열/컬럼들 선택

인덱스활용

◆ 멀티인덱스

▪ 열 멀티인덱스 – 열/컬럼 선택

```
print( df[("B","col1")] )

print( df["B"] )
```

```
df[("B","col1")]
0     2
1     5
2     8
3    11
4    14
5    17
6    20
Name: (B, col1), dtype: int64
```

```
df["B"]
    col1  col2
0      2     3
1      5     6
2      8     9
3     11    12
4     14    15
5     17    18
6     20    21
```

인덱스활용

◆ 멀티인덱스

▪ 레벨별 데이터 추출 : xs( ) 메서드

→ cross section(교차 섹션) 약자

→ 특정 수준(level)의 행 또는 열 쉽게 선택할 수 있게 함

→ 단!! 값 설정 불가함

```
DataFrame.xs(  key,                    # 기준값
               axis=0,                 # 0 - 행 , 1 -열
               level,                  # key가 포함된 레벨명
                 drop_level=True       # 선택 레벨 결과에서 제거
             )
```

인덱스활용

◆ 멀티인덱스

▪ 레벨별 데이터 추출 – 행 멀티인덱스

```
## 인덱서 생성
index = pd.MultiIndex.from_product( [['A', 'B'], [1, 2]],
                                    names=['letter', 'number'])
## DF 생성
df = pd.DataFrame({'value': [10, 20, 30, 40]}, index=index)


## 출력
print(df)
```

		value
letter	number	
A	1	10
	2	20
B	1	30
	2	40

인덱스활용

◆ 멀티인덱스

▪ 레벨별 데이터 추출 – 행 멀티인덱스

		value
letter	number	
A	1	10
	2	20
B	1	30
	2	40

```
## 행 선택
print( df.loc[ ('A', 1 ) ] )
```

```
value     10
Name: (A, 1), dtype: int64
```

```
## 행 선택
print( df.loc[ 'A' ] )
```

```
          value
number
1            10
2            20
```

인덱스활용

◆ 멀티인덱스

▪ 레벨별 데이터 추출 – 행 멀티인덱스

		value
letter	**number**	
A	1	10
	2	20
B	1	30
	2	40

```
## 행 선택
print( df.loc[ ('A',1) ] )
```

```
value    10
Name: (A, 1), dtype: int64
```

```
## level1이 1인 행 선택
## print( df.loc[  1  ] )   <== KeyError :1
print( df.xs( key=1, level="number" ) )
```

```
        value
letter
A          10
B          30
```

인덱스활용

◆ 멀티인덱스

▪ 레벨별 데이터 추출 – 열 멀티인덱스

```
## 인덱서 생성
columns_ = pd.MultiIndex.from_product([['math', 'eng'], ['mid', 'final']],
                                      names=['subject', 'exam'])


## DF 생성
df = pd.DataFrame( [ [80, 90, 85, 95], [70, 88, 75, 93] ],
                   columns=columns_ )


## 출력
print(df)
```

subject	math		eng	
exam	mid	final	mid	final
0	80	90	85	95
1	70	88	75	93

인덱스활용

◆ 멀티인덱스

▪ 레벨별 데이터 추출 – 열 멀티인덱스

subject	math		eng	
exam	mid	final	mid	final
0	80	90	85	95
1	70	88	75	93

```
## 열 선택 : df[(level0, level1)]
print( df[("math","mid")] )
```

```
0    80
1    70
Name: (math, mid), dtype: int64
```

```
## 열 선택 : df[level0]
print( df["math"] )
```

```
exam  mid  final
0       80     90
1       70     88
```

인덱스활용

◆ 멀티인덱스

▪ 레벨별 데이터 추출 – 열 멀티인덱스

subject	math		eng	
exam	mid	final	mid	final
0	80	90	85	95
1	70	88	75	93

```
## 열 선택 : df[(level0, level1)]
print( df[("math","mid")] )
```

```
0    80
1    70
Name: (math, mid), dtype: int64
```

```
## 열 선택 : df[level0]
print( df["math"] )
```

exam	mid	final
0	80	90
1	70	88

인덱스활용

◆ 멀티인덱스

▪ 레벨별 데이터 추출 – 열 멀티인덱스

subject	math		eng	
exam	mid	final	mid	final
0	80	90	85	95
1	70	88	75	93

```
## level1인 열 선택 : df[level1]
print( df["mid"] )
```

```
KeyError: 'mid'

The above exception was the direct
```

```
## level1인 열 선택 : xs()
print( df.xs( key="mid",
              level='exam', axis=1) )
```

```
subject  math  eng
0           80   85
1           70   75
```

인덱스활용

◆ 멀티인덱스

- **슬라이싱 연산을 객체로 표현 : slice( ) 내장함수**

  → 일부 인덱스 수준 전체에 대해 선택 가능

slice(start, stop, step)

• [:]	slice(None)
• [1 :]	slice(1, None)
• [: 5]	slice(None, 5)
• [: : 2]	slice(None, None, 2)

인덱스활용

◆ 멀티인덱스

- **슬라이싱 연산을 객체로 표현**

```
# 컬럼 멀티인덱스
columns = pd.MultiIndex.from_product( [['math', 'eng'], ['mid', 'final']],
                                      names=['subject', 'exam'])
data = [ [80, 90, 85, 95],
         [70, 88, 75, 93],
         [60, 84, 65, 90] ]

# 행 멀티인덱스
index = pd.MultiIndex.from_tuples( [('A', 1), ('A', 2), ('B', 1)],
                                   names=['class', 'number'])
# 데이터프레임 생성
df = pd.DataFrame(data, index=index, columns=columns)
```

인덱스활용

◆ 멀티인덱스

- **슬라이싱 연산을 객체로 표현**

```
# 출력
display( df )
```

	subject	math		eng	
	exam	mid	final	mid	final
class	number				
A	1	80	90	85	95
	2	70	88	75	93
B	1	60	84	65	90

인덱스활용

◆ 멀티인덱스

▪ 슬라이싱 연산을 객체로 표현

```
## 행: ('A', 모든 번호), 열: 전체 슬라이싱
result = df.loc[('A', slice( None))]
print(result)
```

```
subject math        eng
exam      mid final mid final
number
1          80    90  85    95
2          70    88  75    93
```

```
## 행: ('A', 2 이상), 열: 전체
result = df.loc[('A', slice(2, None)), :]
print(result)
```

```
subject       math        eng
exam           mid final mid final
class number
A     2         70    88  75    93
```