

[3-2. Series, DataFrame 생성 및 기본 조작]

1. Series 생성하기

1-1. 리스트를 이용한 Series 생성

- 인덱스는 기본적으로 0부터 시작하는 정수로 자동 생성됩니다.

```
In [ ]: import pandas as pd

# 기본 인덱스 사용
s1 = pd.Series([1, 2, 3, 4])
print(s1)

0    1
1    2
2    3
3    4
dtype: int64
```

1-2. 넘파이 배열(numpy array)를 이용한 Series 생성

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

array = np.array([1, 2, 3, 4])
s2 = pd.Series(array)
print(s2)

0    1
1    2
2    3
3    4
dtype: int64
```

1-3. 딕셔너리를 이용한 Series 생성

- 딕셔너리의 키가 인덱스가 됩니다.

```
In [ ]: from pandas import Series

s3 = Series({'a': 1, 'b': 2, 'c': 3, 'd': 4})
print(s3)
```

```
a    1
b    2
c    3
d    4
dtype: int64
```

1-4. 인덱스 지정

- Series를 생성할 때 index 매개변수를 사용하면, 데이터 각각에 해당하는 인덱스를 수동으로 지정할 수 있습니다.

```
In [ ]: s4 = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
print(s4)
```

```
a    10
b    20
c    30
d    40
dtype: int64
```

2. Series 객체 조회 및 값 수정

2-1. Series 객체 값 조회

```
In [ ]: from pandas import Series

s = Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'])
print(s)
```

```
a    10
b    20
c    30
d    40
e    50
dtype: int64
```

```
In [ ]: # 인덱스를 사용한 접근
print(s['a']) # 출력: 10

# 위치를 이용한 접근
print(s[0])   # 출력: 10

# 슬라이스를 이용한 접근
print(s[:3])  # 출력: a, b, c 인덱스의 값

# 불리언 인덱싱
print(s[s > 30]) # 30보다 큰 값들을 가진 항목들을 출력
```

```
10
10
a    10
b    20
c    30
dtype: int64
d    40
e    50
dtype: int64
```

2-2. Series 객체 값 수정

```
In [ ]: # 인덱스를 사용하여 개별 값 수정
s['a'] = 100
print(s)

# 여러 값 동시 수정 <= 여러 값을 지정할 때는 리스트나 튜플로 지정
s[['b', 'c']] = [200, 300]
print(s)

# 슬라이스를 이용한 범위의 값 수정
s[:2] = 0
print(s)

# 조건을 만족하는 항목 수정
s[s > 100] = 999
print(s)
```

```
a    100
b     0
c    999
d     40
e     50
dtype: int64
a    100
b    200
c    300
d     40
e     50
dtype: int64
a     0
b     0
c    300
d     40
e     50
dtype: int64
a     0
b     0
c    999
d     40
e     50
dtype: int64
```

3. DataFrame 생성하기

3-1. 리스트의 리스트를 이용한 DataFrame 생성

```
In [ ]: import pandas as pd

data = [
    [1, 'John Smith', 32],
    [2, 'Diana Green', 28],
    [3, 'Sarah Brown', 45]
]

df = pd.DataFrame(data, columns=['ID', 'Name', 'Age'])
print(df)
```

	ID	Name	Age
0	1	John Smith	32
1	2	Diana Green	28
2	3	Sarah Brown	45

3-2. 딕셔너리를 이용한 DataFrame 생성

- 각 키가 열의 이름이 되고, 값의 리스트가 열의 데이터가 됩니다.

```
In [ ]: data = {
    'ID': [1, 2, 3],
    'Name': ['John Smith', 'Diana Green', 'Sarah Brown'],
    'Age': [32, 28, 45]
}

df = pd.DataFrame(data)
print(df)
```

	ID	Name	Age
0	1	John Smith	32
1	2	Diana Green	28
2	3	Sarah Brown	45

3-3. 딕셔너리의 리스트를 이용한 DataFrame 생성

- 리스트의 각 항목이 행을 나타내는 딕셔너리입니다.

```
In [ ]: from pandas import DataFrame

data = [
    {'ID': 1, 'Name': 'John Smith', 'Age': 32},
    {'ID': 2, 'Name': 'Diana Green', 'Age': 28},
    {'ID': 3, 'Name': 'Sarah Brown', 'Age': 45}
]

df = DataFrame(data)
print(df)
```

	ID	Name	Age
0	1	John Smith	32
1	2	Diana Green	28
2	3	Sarah Brown	45

3-4. NumPy 배열을 이용한 DataFrame 생성

```
In [ ]: import numpy as np

# 4x3 크기의 무작위 배열 생성
array = np.random.rand(4, 3)

df = DataFrame(array, columns=['Column1', 'Column2', 'Column3'])
print(df)
```

	Column1	Column2	Column3
0	0.781182	0.139754	0.255914
1	0.977990	0.557837	0.410401
2	0.674401	0.419973	0.492563
3	0.969080	0.501805	0.821683

3-5. Series 객체를 이용한 DataFrame 생성

```
In [ ]: s1 = pd.Series([1, 2, 3])
s2 = pd.Series(['John Smith', 'Diana Green', 'Sarah Brown'])
s3 = pd.Series([32, 28, 45])

df = DataFrame({'ID': s1, 'Name': s2, 'Age': s3})
print(df)
```

	ID	Name	Age
0	1	John Smith	32
1	2	Diana Green	28
2	3	Sarah Brown	45

4. DataFrame 객체 조회 및 값 수정

4-1. 데이터 조회하기

4-1-1. 열 선택하기

```
In [ ]: from pandas import DataFrame

# 예제 DataFrame 생성
data = {'Name': ['John', 'Anna', 'Peter', 'Linda'],
        'Age': [28, 34, 29, 32],
        'City': ['New York', 'Paris', 'Berlin', 'London']}
df = DataFrame(data)

# 'Name' 열 선택
print(df['Name'])

0    John
1    Anna
2    Peter
3    Linda
Name: Name, dtype: object
```

```
In [ ]: # 여러 열 선택하기 : 리스트를 사용하여 선택하고자 하는 열 이름 나열
print(df[['Name', 'Age']])
```

	Name	Age
0	John	28
1	Anna	34
2	Peter	29
3	Linda	32

4-1-2. loc, iloc 사용하기

DataFrame의 데이터를 조회하거나 수정할 때 'loc'와 'iloc' 속성을 사용합니다.

'loc'는 라벨 기반으로 데이터를 선택하며, 'iloc'는 정수 위치 기반으로 데이터를 선택합니다.

[loc]

- 라벨 기반 선택 : 'loc'를 사용하면 DataFrame에서 라벨을 기반으로 행이나 열을 선택할 수 있습니다. 이 때 라벨은 DataFrame의 인덱스 값(행 라벨)과 컬럼명(열 라벨)을 의미합니다.
- 구문 : df.loc[행 라벨, 열 라벨]
- 슬라이싱 지원 : 'loc'를 사용하여 슬라이싱을 할 때는 시작 라벨과 끝 라벨 모두 포함됩니다.

```
In [ ]: # 'Anna'의 전체 정보 조회
print(df.loc[1])

# 'Anna'와 'Peter'의 이름과 나이 조회
print(df.loc[1:2, ['Name', 'Age']])

# 마지막 행 조회
print(df.loc[df.index[-1]])
```

```
Name      Anna
Age        34
City      Paris
Name: 1, dtype: object

      Name  Age
1  Anna   34
2  Peter  29
Name      Linda
Age        32
City      London
Name: 3, dtype: object
```

[iloc]

- 위치 기반 선택 : 'iloc'를 사용하면 Python과 같은 방식으로 정수 위치를 기반으로 행이나 열을 선택할 수 있습니다.
- 구문 : df.iloc[행 위치, 열 위치]
- 슬라이싱 지원 : 'iloc'를 사용하여 슬라이싱을 할 때는 Python의 표준 슬라이싱과 같이 시작 위치는 포함되지만, 끝 위치는 포함되지 않습니다.

```
In [ ]: # 첫 번째 행의 첫 번째 열(John) 조회
print(df.iloc[0, 0])

# 첫 번째와 두 번째 행의 첫 번째와 두 번째 열 조회
print(df.iloc[0:2, 0:2])

# 마지막 행 조회
print(df.iloc[-1])
```

```
John
      Name  Age
0  John   28
1  Anna   34
Name      Linda
Age        32
City      London
Name: 3, dtype: object
```

4-2. 데이터 수정하기

4-2-1. 특정 값 수정하기

```
In [ ]: # 첫 번째 사람의 이름을 'Jon'으로 변경
df.loc[0, 'Name'] = 'Jon'

print(df)
```

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

4-2-2. 조건에 따른 값 수정하기

```
In [ ]: # 나이가 30보다 큰 사람의 도시를 'Tokyo'로 변경
df.loc[df['Age'] > 30, 'City'] = 'Tokyo'

print(df)
```

	Name	Age	City
0	Jon	28	New York
1	Anna	34	Tokyo
2	Peter	29	Berlin
3	Linda	32	Tokyo

4-2-3. 열에 함수 적용하기

```
In [ ]: # 모든 이름을 대문자로 변경
df['Name'] = df['Name'].apply(lambda x: x.upper())

print(df)
```

	Name	Age	City
0	JON	28	New York
1	ANNA	34	Tokyo
2	PETER	29	Berlin
3	LINDA	32	Tokyo

4-2-4. 열 추가하기

- 새로운 키값에 대해서 데이터 셋(리스트 등)을 설정하면 신규 데이터가 반영됨.

```
In [ ]: # 'Employee'라는 새 열을 추가하고, 값을 리스트로 설정함
df['Employee'] = ['Yes', 'Yes', 'No', 'Yes']

print(df)
```

	Name	Age	City	Employee
0	JON	28	New York	Yes
1	ANNA	34	Tokyo	Yes
2	PETER	29	Berlin	No
3	LINDA	32	Tokyo	Yes

4-2-5. 열 삭제하기

```
In [ ]: # 'City' 열 삭제
df.drop('City', axis=1, inplace=True)

print(df)
```

	Name	Age	Employee
0	JON	28	Yes
1	ANNA	34	Yes
2	PETER	29	No
3	LINDA	32	Yes

- axis 매개변수가 0이면 행에, 1이면 열에 drop 메서드가 적용됩니다.
- inplace 매개변수가 True면 원본 DataFrame이 직접 수정됩니다. 즉, 메서드가 적용된 후 DataFrame에서 지정된 행이나 열이 제거됩니다. 이 경우 반환 값은 'None' 입니다. inplace 매개변수가 False면 수정된 DataFrame을 반환하고, 원본 DataFrame은 수정되지 않습니다.