[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
```

```
/var/folders/01/c04y5bhn61l8kzg4jnpy_8dw0000gp/T/ipykernel_70231/1532506328.py:5: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  print(s[0])    # 출력: 10
```

2-2. Series 객체 값 수정

In [ ]:
```python
# 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     20
c     30
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 딕셔너리 입니다.

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  0.603613  0.781750  0.802971
1  0.347847  0.250743  0.576425
2  0.820680  0.451111  0.938709
3  0.176985  0.826091  0.552387
```

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
```

3-6. CSV 파일로부터 DataFrame 생성(read_csv 메서드)

CSV 파일을 읽어들여서 DataFrame으로 전환하기 위해서는 pandas 라이브러리의 read_csv 메서드를 사용합니다.

이 메서드는 다양한 매개변수를 제공하여 CSV 파일 읽기 과정을 상세하게 제어할 수 있습니다.

3-6-1. 기본적인 CSV 파일 읽어들이는 방법

In [ ]:
```
import pandas as pd

# sample_data.csv 파일을 읽어 DataFrame 생성
df = pd.read_csv('sample_data.csv')
print(df)
```

```
           날짜    제품     가격  판매량
0  2023-01-01    사과  1000    50
1  2023-01-01   바나나   1500    40
2  2023-01-02    사과  1000    60
3  2023-01-02   바나나   1500    45
```

3-6-2. 주요 매개변수

- sep 또는 delimiter : 필드를 구분하는 데 사용되는 문자를 지정합니다. 기본값은 ',' 입니다.
- header : 열 이름으로 사용할 행의 번호를 지정합니다. 기본값은 '0'이며, 파일의 첫 번째 행을 열 이름으로 사용합니다. 헤더가 없는 경우 'None'으로 설정할 수 있습니다.
- index_col : 인덱스로 사용할 열 번호나 이름을 지정합니다.
- usecols : 데이터를 읽어올 열의 이름이나 번호를 지정하는 리스트입니다. 이를 사용하면 파일에서 필요한 열만 선택적으로 읽어올 수 있습니다.
- dtype : 열의 데이터 타입을 지정하는 딕셔너리 입니다. 예를 들어 'dtype={'날짜': str, '제품': str, '가격': int, '판매량': int}와 같이 사용할 수 있습니다.
- skiprows : 무시할 행의 수나 무시할 행 번호의 리스트를 지정합니다.
- nrows : 파일에서 읽을 행의 수입니다.

3-6-3. 주요 매개변수 사용 예시

In [ ]:
```
# 구분자 지정하기
df = pd.read_csv('sample_data.csv', sep=',')

print(df)
```

```
           날짜    제품     가격  판매량
0  2023-01-01    사과   1000    50
1  2023-01-01   바나나   1500    40
2  2023-01-02    사과   1000    60
3  2023-01-02   바나나   1500    45
```

In [ ]:
```
# 첫 번째 행을 열 이름으로 사용하지 않고, 첫 번째 열을 인덱스로 사용, 첫 번째 행은 skip
df = pd.read_csv(
    'sample_data.csv', header=None, index_col=0, skiprows=[0]
)

print(df)
```

```
              1     2   3
0
2023-01-01   사과  1000  50
2023-01-01  바나나   1500  40
2023-01-02   사과  1000  60
2023-01-02  바나나   1500  45
```

In [ ]:
```
# 특정 열만 읽어오기
df = pd.read_csv('sample_data.csv', usecols=['제품', '가격', '판매량'])

print(df)
```

```
    제품    가격  판매량
0   사과  1000   50
1  바나나  1500   40
2   사과  1000   60
3  바나나  1500   45
```

In [ ]:
```
# 처음 2행만 읽어오기
df = pd.read_csv('sample_data.csv', nrows=2)

print(df)
```

```
         날짜    제품    가격  판매량
0  2023-01-01   사과  1000   50
1  2023-01-01  바나나  1500   40
```

In [ ]:
```
# 열 타입 지정하기
df = pd.read_csv(
    'sample_data.csv',
    dtype={'날짜':str, '제품': str, '가격': int, '판매량': int}
)

print(df)
```

```
         날짜    제품    가격  판매량
0  2023-01-01   사과  1000   50
1  2023-01-01  바나나  1500   40
2  2023-01-02   사과  1000   60
3  2023-01-02  바나나  1500   45
```

In [ ]:
```
# 특정 행을 건너뛰고 읽기
df = pd.read_csv('sample_data.csv', skiprows=[1, 2])

print(df)
```

```
         날짜    제품    가격  판매량
0  2023-01-02   사과  1000   60
1  2023-01-02  바나나  1500   45
```

3-7. Excel 파일로부터 DataFrame 생성(read_excel 메서드)

엑셀 파일을 읽어들여서 DataFrame으로 전환하기 위해서는 pandas 라이브러리의 read_excel 메서드를 사용합니다.

read_excel 메서드는 엑셀 파일을 읽어 들이기 위해 별도의 엑셀파일을 다루는 라이브러리를 사용합니다. 주로 사용되는 라이브러리는 'openpyxl' 라이브러리 입니다.

만약 'openpyxl' 라이브러리가 설치되어있지 않다면, 추가로 설치가 필요합니다.

3-7-1. 기본적인 Excel 파일 읽어들이는 방법

In [ ]:
```
# output.xlsx 파일을 읽어 DataFrame 생성
df = pd.read_excel('output.xlsx')
print(df)
```

```
      Name  Age  Gender
0     John   25    male
1    Emily   30  female
2  Michael   35    male
3   Celine   24  female
```

3-7-2. 엑셀 리딩 엔진 설정

먼저 사용했던 read_excel 메서드에 엑셀 리딩 엔진을 별도로 설정할 수도 있습니다.

In [ ]:
```
# output.xlsx 파일을 읽어 DataFrame 생성
df = pd.read_excel('output.xlsx', engine='openpyxl')
print(df)
```

```
      Name  Age  Gender
0     John   25    male
1    Emily   30  female
2  Michael   35    male
3   Celine   24  female
```

3-7-3. sheet_name 매개변수

읽어들이고 싶은 시트의 이름이나 숫자를 지정합니다. 기본값은 0이며, 이는 첫 번째 시트를 읽어옵니다.

In [ ]:
```
df_sheet = pd.read_excel('output.xlsx', sheet_name='Sheet')
print(df_sheet)
```

```
      Name  Age  Gender
0     John   25    male
1    Emily   30  female
2  Michael   35    male
3   Celine   24  female
```

3-7-4. header 매개변수

열 이름으로 사용할 행의 번호를 지정합니다. 기본값은 0이며, 첫 번째 행을 열 이름으로 사용합니다.

In [ ]:
```
df_header = pd.read_excel('output.xlsx', header=0)
print(df_header)
```

```
      Name  Age  Gender
0     John   25    male
1    Emily   30  female
2  Michael   35    male
3   Celine   24  female
```

3-7-5. usecols 매개변수

파일에서 읽을 열을 지정하는데 사용됩니다. 열 이름 또는 열 번호의 리스트, 또는 엑셀에서 사용하는 열 범위를 나타내는 문자열을 사용할 수 있습니다.

In [ ]:
```
df_columns1 = pd.read_excel('output.xlsx', usecols='A,C')
print(df_columns1)
```

```
      Name  Gender
0     John    male
1    Emily  female
2  Michael    male
3   Celine  female
```

In [ ]:
```
df_columns1 = pd.read_excel('output.xlsx', usecols='B:C')
print(df_columns1)
```

```
   Age  Gender
0   25    male
1   30  female
2   35    male
3   24  female
```

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 [ ]:
```python
# '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.