데이터셋 읽어오기와 사용하기

한국기술교육대학교
온라인평생교육원

학습목표

- Pandas 라이브러리의 필요성을 이해하고, 데이터셋을 만들 수 있다.
- 파이썬에서 제공하는 기본타입들을 Pandas 데이터셋으로 전환하고 원하는 방식으로 데이터를 제어할 수 있다.

학습내용

- Pandas가 제공하는 데이터셋
- Pandas가 제공하는 데이터셋의 연산

Pandas가 제공하는 데이터셋

1 Pandas 라이브러리

1 데이터셋

데이터 수집	▪ 다양한 소스로부터 데이터를 수집하는 방법 ▪ 수집된 데이터는 각자 동일하거나 다른 형태로 구성됨
데이터셋	▪ 다양한 형태의 데이터를 하나의 구조로 만듦 ➡ 관리 및 분석 용이

Pandas가 제공하는 데이터셋

1 Pandas 라이브러리

1 데이터셋

- 파이썬에서 데이터셋 관리를 위해 Pandas 라이브러리를 제공함
- 데이터 분석을 위해 파이썬에서 기존에 사용하던 타입인 list, dict 등을 그냥 사용할 수 없음

머신러닝 또는 딥러닝은 2차원 구조의 데이터를 이용해 스스로 학습하고 미래의 상황을 예측할 수 있음

✓ 2차원 구조의 Numpy의 Ndarray, Pandas의 DataFrame 객체를 이용

➡ 파이썬이 제공하는 기본 타입을 데이터 분석에 사용하기 쉽도록 함

Pandas가 제공하는 데이터셋

1 Pandas 라이브러리

2 Pandas 라이브러리란?

Pandas 라이브러리

- 데이터 분석을 위해 널리 사용되는 파이썬 라이브러리 패키지
- 클래스를 별도로 만들지 않고도 데이터를 읽고 쓸 수 있음
- 파이썬에서 제공하는 통계 관련 패키지에서도 자주 사용

아나콘다(Anaconda)를 이용해 파이썬을 설치한 경우	이미 Pandas 패키지가 설치되어 있음
파이썬만 설치되어 있는 경우	Pandas 별도 설치 필요

Pandas가 제공하는 데이터셋

1 Pandas 라이브러리

3 설치

- 콘솔 창에서 pip install Pandas라고 하면 설치가 됨

설치가 되지 않는 경우	▪ 파이썬의 path 설정이 잘못되어 있거나, 권한이 부족한 경우

- pip 명령이 작동되지 않는 경우 : 환경변수에 경로를 지정
- 권한이 부족하여 설치가 되지 않는 경우 : 콘솔 창의 마우스 오른쪽을 눌러서 관리자 권한으로 실행
- 처음부터 아나콘다 패키지를 이용해 파이썬을 설치했으므로 바로 사용 가능

Pandas가 제공하는 데이터셋

1 Pandas 라이브러리

4 구성 및 내용

- 클래스와 여러 가지 내장함수로 구성

- list나 dict 타입 등을 Pandas에서 사용하는 데이터 구조로 변환할 수 있음
- 데이터 분석에서 사용하는 데이터 타입은 파이썬에서 제공하는 list나 dict 타입을 그냥 사용하기에는 무리가 있음
- csv 파일을 읽고 쓰거나, Excel 파일을 읽고 쓰기 쉽도록 구성

Pandas가 제공하는 데이터셋

1 Pandas 라이브러리

5 Pandas 사용법

- Pandas를 사용하려면 Pandas 라이브러리를 임포트해야 함

- 임포트 시 별명을 붙여서 사용하는 경우가 많음
- 'import Pandas as pd'
- 기존의 언어와 달리 각 데이터 구조 차원별로 별도의 이름을 붙임

1차원 데이터 구조 Series	2차원 데이터 구조 DataFrame	3차원 데이터 구조 Panel

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

1 Series

1 Series 구조

- Pandas가 제공하는 1차원 데이터 구조의 객체 또는 클래스
- 배열/리스트와 같은 일련의 시퀀스(연속된) 데이터를 저장

list 타입 → Series 구조

데이터를 식별하기 위해 인덱스가 지정됨

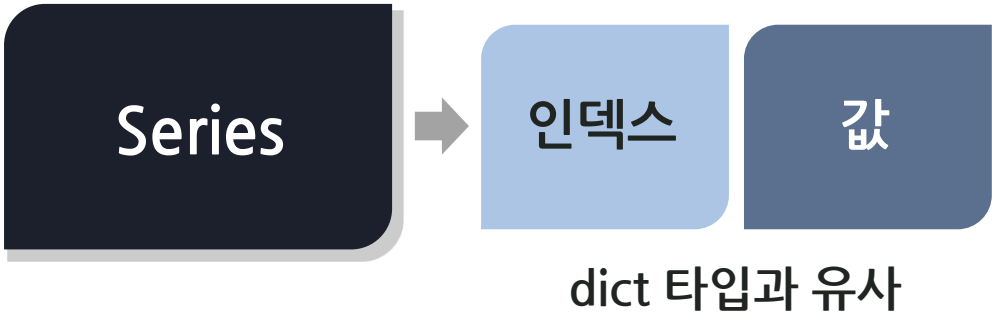
- 별도의 레이블 인덱스를 지정하지 않으면 0부터 시작하는 인덱스가 자동으로 부여됨

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

1 Series

1 Series 구조



dict 타입과 유사

- 파이썬이 제공하는 dict 타입을 이용해서 인덱스를 강제로 부여하는 방법으로 Series를 만들 수 있음

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

1 Series

2 인덱스 구조

- 자기와 짝을 이루는 데이터의 값과 주소를 저장
- 데이터 값의 검색, 정렬, 선택, 결합 등을 쉽게 할 수 있도록 도와주는 구조
- 데이터를 빠르게 식별하도록 도와주는 구조

Index	Value
0	10
1	20
2	30
3	40

- Series 구조의 데이터에 접근하기
- 변수명[인덱스] = 값

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

1 Series

3 Series 예제

```
#파일명 : exam9_1.py
import Pandas as pd
```

- Pandas 라이브러리를 사용하기 위해 임포트
- 임포트 시 pd라는 별도의 이름을 부여함

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

1 Series

3 Series 예제

- 파이썬의 list를 사용하여 Series 만들기

```
data = [10,20,30,40,50]
series = pd.Series(data)
print(type(series))
print(series)
```

```
<class 'pandas.core.series.Series'>
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

- 자동으로 인덱스가 부여됨

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

1 Series

3 Series 예제

- 직접 index 부여해보기(dict 타입 사용)

```
data2 = {'a':1, 'b':2, 'c':3, 'd':4, 'e':5}
series2 = pd.Series(data2)
print(series2)
```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

2 DataFrame

1 DataFrame 구조

- Pandas가 제공하는 2차원 데이터 구조의 객체 또는 클래스

R 언어

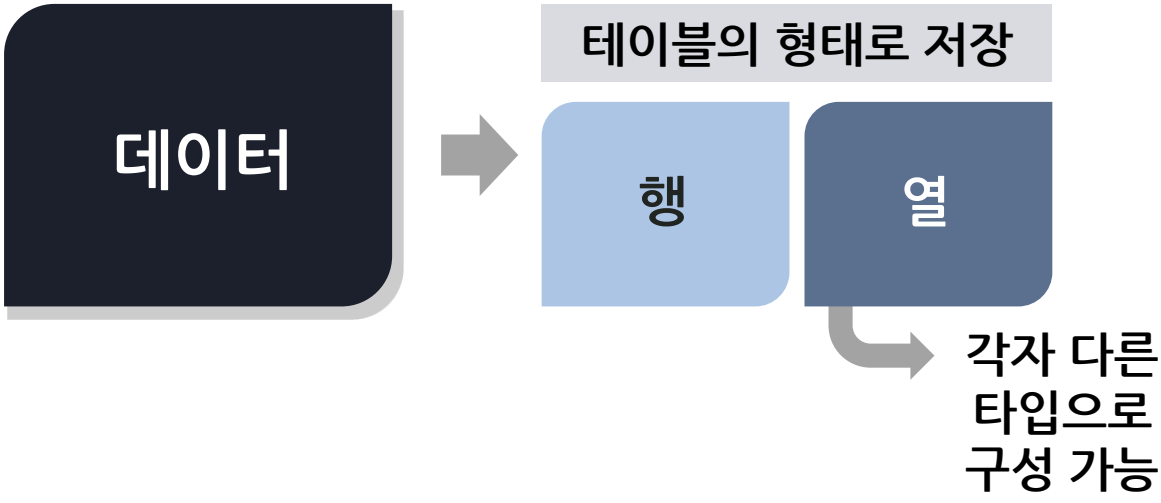
- Pandas의 DataFrame은 R이라는 통계 언어에서 비롯됨
- 다른 언어의 기능이 파이썬으로 구현되어 R 언어보다 기능이 떨어지지만, 데이터를 다루기에 충분함

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

2 DataFrame

1 DataFrame 구조



- 데이터를 저장하는 전통적인 방법 : **테이블 형태**

- 테이블 구조 형태의 데이터 : 관계형 데이터베이스, Excel, csv 형식의 자료 등

  ➡ DataFrame은 이러한 유형의 데이터를 다루는데 최적화된 도구

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

2 DataFrame

1 DataFrame 구조

```
#파일명 : exam9_2.py
import Pandas as pd

data = {
'name':['홍길동', '임꺽정', '장길산', '홍경래'],
'kor':[90, 80, 70, 70],
'eng':[99, 98, 97, 46],
'mat':[90, 70, 70, 60],
}
```

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

2 DataFrame

2 DataFrame 예제

```
df = pd.DataFrame(data)
print("타입 : ", type(df))
print(df)
```

```
타입 :  <class 'pandas.core.frame.DataFrame'>
  name  kor  eng  mat
0  홍길동   90   99   90
1  임꺽정   80   98   70
2  장길산   70   97   70
3  홍경래   70   46   60
```

- dict 타입의 데이터를 정의
- dict 타입은 키와 값 쌍으로 이루어짐
- 특정키 값에 데이터를 여러 개 넣을 경우의 키 : list 형태로 저장 가능

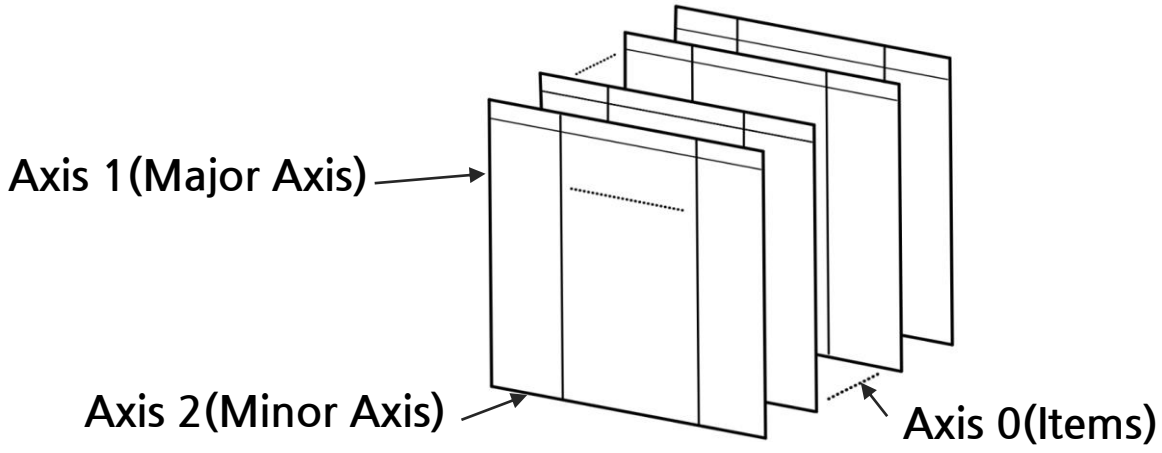
Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

3 Panel

1 Panel 구조

- Pandas가 제공하는 3차원 데이터 구조의 객체 또는 클래스



Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

3 Panel

1 Panel 구조

Axis 0	▪ 그 한 요소가 2차원의 DataFrame에 해당됨
Axis 1	▪ DataFrame의 행(Row)에 해당
Axis 2	▪ DataFrame의 열(Column)에 해당

- 파이썬의 3차원 배열을 이용해 Panel 객체를 만들 수 있음

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

3 Panel

2 Panel 예제

```
#파일명 : exam9_3.py
import Pandas as pd
#파이썬은 언어의 구조상 3차원 배열이라는 개념이 없음
3차원 배열을 만듦 - list of list of list  [ [ [ ], [], []]]

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

array= pd.Panel(array)
print("타입 : ", type(array))
print(array)
```

Pandas가 제공하는 데이터셋

2 Pandas 데이터셋

3 Panel

2 Panel 예제

- 실행하면 경고 메시지가 나옴
  ➡ 앞으로는 Panel 함수가 사라진다는 의미

```
타입 :  <class 'pandas.core.panel.Panel'>
<class 'pandas.core.panel.Panel'>
Dimensions: 3 (items) x 2 (major_axis) x 3 (minor_axis)
Items axis: 0 to 2
Major_axis axis: 0 to 1
Minor_axis axis: 0 to 2
```

```
Panel is deprecated and will be removed in a future version.
The recommended way to represent these types of 3-dimensional data are with a MultiIndex on a DataFrame, via the Panel.
to_frame() method
Alternatively, you can use the xarray package http://xarray.pydata.org/en/stable/.
Pandas provides a `.to_xarray()` method to help automate this conversion.
```

Pandas가 제공하는 데이터셋

3 데이터 액세스

1 Series 데이터 액세스

- Series 타입의 데이터에 접근하기 위해 인덱스를 활용
- 각각의 요소에 대한 접근 방법 : **변수명[인덱스]**

예시

- print(mydata1[0])
  : Series의 첫 번째 저장된 데이터 값을 읽어서 출력
- mydata1[0]=100
  : Series의 첫 번째 저장된 값을 변경

Pandas가 제공하는 데이터셋

3 데이터 액세스

1 Series 데이터 액세스

- 슬라이싱을 이용해 출력 가능

예시

- print(mydata1[0: 3])
  0번 방부터 3번 방 이전까지 마지막 방에 있는 데이터는 제외
- print(mydata1[:3])
  0번 방부터 3번 방 이전까지 출력
- print(mydata1[3:])
  3번 방 이후로 마지막까지 출력

- 인덱스에 숫자가 아닌 라벨이 있을 경우 라벨을 이용해 접근할 수 있음

예시

- print(mydata2['one'])

Pandas가 제공하는 데이터셋

3 데이터 액세스

1 Series 데이터 액세스

- 인덱스를 이용한 데이터 액세스 예제

```
#파일명 : exam9_4.py
import Pandas as pd
data = [10,20,30,40,50,60]
s = pd.Series(data)
print(s[0])   #0번째 해당 데이터 출력
s[1]=200    #1번째 데이터 200으로 수정
print(s)
print(s[:5])              #5번째 데이터부터 마지막까지 출력
print(s[2:5])             #2번 데이터부터 4번 데이터까지 출력
print(s[3:])              #3번 이후로 출력
```

Pandas가 제공하는 데이터셋

3 데이터 액세스

1 Series 데이터 액세스

- 라벨을 이용한 데이터 액세스 예제

```
data = {'one':'일', 'two':'이', 'three':'삼', 'four':'사',
'five':'오'}
series = pd.Series(data)
print(series)
print(series['one'])
print(series[0:3])
print(series['one':'three'])
print(series['two':])
```

```
one      일
two      이
three    삼
four     사
five     오
dtype: object
```

Pandas가 제공하는 데이터셋

3 데이터 액세스

2 DataFrame 데이터 액세스

- Pandas 라이브러리에서 가장 빈번하게 사용되는 데이터 타입
- iloc, loc라는 두 개의 함수를 이용해 데이터에 접근할 수 있음

iloc 함수	▪ 행과 열의 인덱스를 이용해 접근
loc 함수	▪ 행의 인덱스와 열의 컬럼명(필드명)을 통해 데이터를 접근

Pandas가 제공하는 데이터셋

3 데이터 액세스

2 DataFrame 데이터 액세스

Index	Name	Kor	Eng
0	홍길동	90	85
1	장길산	80	75
2	임꺽정	70	65
3	홍경래	60	55

```
print(mydata.iloc[0, 0] )
print(mydata.loc[0, 'name'])
```

```
print(mydata.iloc[1, 2] )
print(mydata.loc[1, 'kor'] )
```

Pandas가 제공하는 데이터셋

3 데이터 액세스

2 DataFrame 데이터 액세스

```
#파일명 : exam9_5.py
import Pandas as pd
data = {
 ' name ' :[ ' 홍길동 ' , ' 임꺽정 ' , ' 장길산 ' , ' 홍경래 ' ,
 ' 이상민 ' , ' 김수경 ' ],
 ' kor ' :[90, 80, 70, 70, 60, 70],
 ' eng ' :[99, 98, 97, 46, 77, 56],
 ' mat ' :[90, 70, 70, 60, 88, 99],
}
df = pd.DataFrame(data)

print(df.head() ) #앞의 다섯 명에 대한 데이터만 나옴

print( " 지정한 열만 출력 " )
print(df[ ' name ' ])
print(df[ ' kor ' ])
print(df.columns) #컬럼 이름만 출력

#iloc 함수 : 배열에서의 위치 값으로 데이터를 접근할 수 있음
print("iloc 함수 사용 ----------")
print(df.iloc[0,0])     #0,0번에 해당하는 데이터 출력
print(df.iloc[3,2])     #3번째 행의 2번째 열
print(df.iloc[2:4,2])
```

(계속)

Pandas가 제공하는 데이터셋

3 데이터 액세스

2 DataFrame 데이터 액세스

```
print(df.iloc[2:4,2:4])

print("loc함수 사용 ----------")
#loc 함수는 필드명으로 데이터를 출력할 수 있음
print(df.loc[0, 'name'])   #0번째 행의 name 필드값 출력
print(df.loc[3, 'eng'])    #0번째 행의 eng 필드값 출력
print(df.loc[:, 'name':'eng'])
```

Pandas가 제공하는 데이터셋의 연산

1 Series 연산

Pandas 객체의 산술연산 프로세스

1. 행, 열 인덱스를 기준으로 모든 원소 정렬
2. 동일한 위치에 있는 원소끼리 일대일 대응
3. 일대일로 대응되어 있는 원소끼리 연산 처리
   (대응되는 원소가 없을 경우 NaN으로 처리)

- Pandas는 연산 시 벡터 연산 수행
- Series 객체에 어떤 숫자를 더하거나 빼는 등의 연산을 진행하면 모든 원소에 대응됨

Pandas가 제공하는 데이터셋의 연산

1 Series 연산

1 예제 1

```
#파일명 : exam9_7.py
import Pandas as pd

data1 = {'kor':90, 'eng':70, 'mat':80}
data2 = {'kor':90, 'eng':70, 'mat':80}
data3 = {'kor':90, 'eng':70, 'mat':80}
data4 = {'eng':90, 'mat':70, 'kor':80
series1 = pd.Series( data1 )
series2 = pd.Series( data2 )
series3 = pd.Series( data3 )
series4 = pd.Series( data4 )

result1 = series1 + series2 + series3 + series4
result2 = result1/4

print("총점 ----------")
print(result1 )
print("평균 ----------")
print(result2 )
```

```
총점 ----------
eng    300
kor    350
mat    310
dtype: int64
평균 ----------
eng    75.0
kor    87.5
mat    77.5
dtype: float64
```

- 인덱스의 순서가 바뀌어도 정렬을 진행하기 때문에 알아서 대응

Pandas가 제공하는 데이터셋의 연산

1 Series 연산

2 예제 2

```
#파일명 : exam9_8.py
import Pandas as pd

#인덱스가 없을 경우 값이 NaN으로 들어가서 연산이 되지 않음

data1 = {'kor':90, 'mat':80}
data2 = {'kor':90, 'eng':70}
data3 = {'kor':90, 'eng':70, 'mat':80}

series1 = pd.Series( data1 )
series2 = pd.Series( data2 )
series3 = pd.Series( data3 )
result1 = series1 + series2 + series3

print(result1)
```

```
eng      NaN
kor    270.0
mat      NaN
dtype: float64
```

Pandas가 제공하는 데이터셋의 연산

2 DataFrame 연산

- DataFrame은 여러 개의 Series의 결합으로 볼 수 있으므로 연산 방법도 Series의 내용이 적용됨

- 산술 연산 모두 가능
- 새로운 행 추가 및 삭제 가능
- 새로운 필드를 손쉽게 붙일 수 있음

주의사항

- 새로운 행의 추가·삭제 시 **결과값**을 **반환**받아야 함
- 결과값을 반환받지 않으면 추가되거나 삭제된 행은 원래대로 보임

Pandas가 제공하는 데이터셋의 연산

2 DataFrame 연산

1 예제 1

```
#파일명 : exam9_10.py
#각각의 dict 타입의 데이터를 결합하여 DataFrame을 만들어 연산
import Pandas as pd
data1 = { ' kor ' :90,  ' eng ' :70,  ' mat ' :80}
data2 = { ' kor ' :90,  ' eng ' :70,  ' mat ' :80}
data3 = { ' kor ' :90,  ' eng ' :70,  ' mat ' :80}

data = pd.DataFrame()
data = data.append( data1 , ignore_index=True)
data = data.append( data2 , ignore_index=True)
data = data.append( data3 , ignore_index=True)

#각 필드값들을 더하여 새로 작성한 필드에 저장
  이미 있던 필드의 경우 data['kor'] 대신에 data.kor로 접근 가능

data['total'] = data.kor + data.eng + data.mat
data['avg'] = data.total/3

print(data)
```

	eng	kor	mat	total	avg
0	70.0	90.0	80.0	240.0	80.0
1	70.0	90.0	80.0	240.0	80.0
2	70.0	90.0	80.0	240.0	80.0

Pandas가 제공하는 데이터셋의 연산

2 DataFrame 연산

2 예제 2

```
#파일명 : exam9_11.py
#dict 타입을 DataFrame으로 전환 후 새로운 필드 추가
import Pandas as pd
data = {'name':['홍길동', '임꺽정', '장길산', '홍경래',
          '이상민', '김수경'],
     'work_time':[40, 30, 20, 42, 28, 35],
    'per_pay':[1000, 9000, 9700, 8500, 12000, 20000],
}

df = pd.DataFrame(data)

#새로운 필드 추가
df['pay'] = df['work_time'] * df['per_pay']
print(df)
```

```
   name  work_time  per_pay     pay
0  홍길동         40     1000   40000
1  임꺽정         30     9000  270000
2  장길산         20     9700  194000
3  홍경래         42     8500  357000
4  이상민         28    12000  336000
5  김수경         35    20000  700000
```

학습정리

1. Pandas가 제공하는 데이터셋

- Series 타입 : Pandas에서 1차원 데이터를 지원하기 위해 만든 데이터 타입
  - 데이터 접근 방법 : 인덱스 활용, 인덱스에 조건을 부여할 수도 있음
- DataFrame : Pandas에서 2차원 데이터를 지원하기 위해 만든 데이터 타입
  - iloc 함수 : 인덱스를 이용해 요소에 접근
  - loc 함수 : 필드명을 이용해 요소에 접근
  - csv파일이나 excel 파일을 다루기 용이

학습정리

2. Pandas가 제공하는 데이터셋의 연산

- Pandas가 제공하는 데이터셋 : 파이썬이 제공하는 기본 타입과 달리 스칼라 연산이 아닌 벡터 연산을 수행(for문 필요없음)
- DataFrame 객체에 원하는 필드를 손쉽게 추가할 수 있음
- DataFrame의 연산도 벡터 연산을 지원함
- DataFrame 클래스에서는 새로운 행을 추가하거나 삭제하는 등의 함수를 제공함
- 새로운 행을 추가하거나 삭제 시 결과값을 반환받아야만 하고, 결과값을 반환받지 않으면 추가되거나 삭제된 행은 반영되지 않음