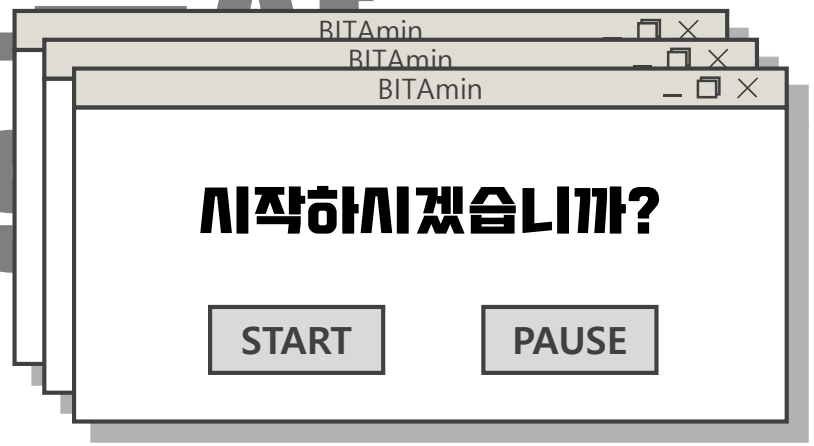
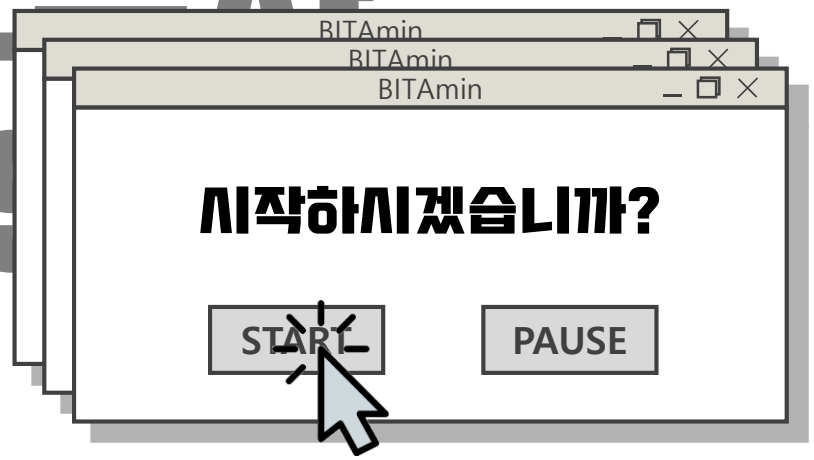
Gram

2주차
SESSION

2주차
SES



2주차
SES



START

목차

1. 판다스 입문(판다스 자료구조)

2. 데이터프레임 핸들링

3. 데이터프레임 살펴보기 및 필터링

4. 데이터프레임 합치기

5. 데이터프레임 연산

1. 판다스 입문
- 데이터 과학자가 판다스를 배우는 이유

Pandas

데이터과학자가 하는 일

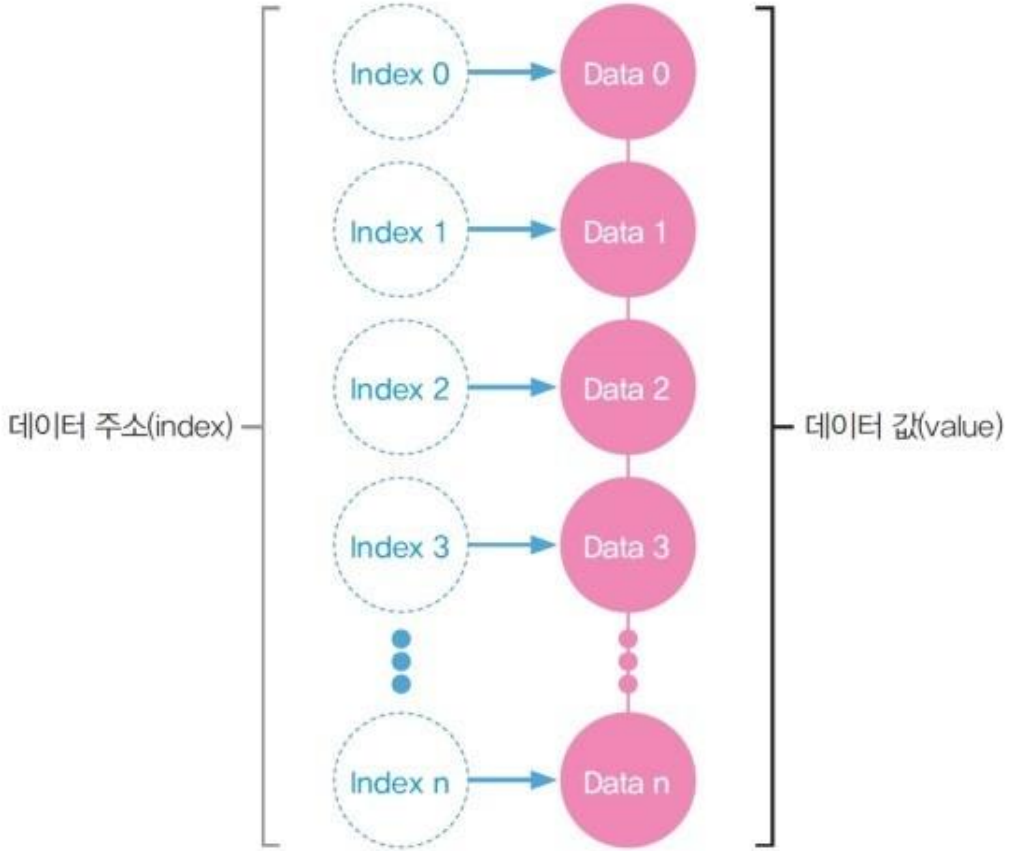
=> 데이터를 수집하고 정리

판다스

=> 데이터를 수집하고 정리하는 데 최적화된 도구, 오픈소스

1. 판다스 입문
- 판다스 자료구조(시리즈)

" 데이터가 순차적으로 나열된 1차원 배열의 형태 "



1. 판다스 입문
- 판다스 자료구조(시리즈)

" 딕셔너리와 구조가 비슷하기 때문에 딕셔너리를 시리즈로 변환하는 방법을 많이 사용. "

```
# pandas 불러오기
import pandas as pd

# k:v 구조를 갖는 딕셔너리를 만들고, 변수 dict_data에 저장
dict_data = {'a': 1, 'b': 2, 'c': 3}

# 판다스 Series() 함수로 딕셔너리(dict_data)를 시리즈로 변환. 변수 sr에 저장
sr = pd.Series(dict_data)

# 변수 sr의 자료형 출력
print(type(sr))
print('₩n')

# 변수 sr에 저장되어 있는 시리즈 객체를 출력
print(sr)
```

```
<class 'pandas.core.series.Series'>


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

- 판다스 내장 함수인 Series()를 이용
- 키는 시리즈의 인덱스에 대응
- 값은 시리즈의 데이터 값으로 변환

1. 판다스 입문
- 시리즈(인덱스 구조)

```
# 리스트를 시리즈로 변환하여 변수 sr에 저장
list_data = ['2019-01-02', 3.14, 'ABC', 100, True]
sr = pd.Series(list_data)
print(sr)
```

```
0    2019-01-02
1          3.14
2           ABC
3           100
4          True
dtype: object
```

- 리스트를 시리즈로 변환 시 인덱스 값이 없다.
- 디폴트로 정수형 위치 인덱스(integer position) 지정
- 왼쪽 예제는 0~4 범위의 정수값이 인덱스로 지정
- 직접 인덱스 이름을 리스트 형태로 전달 가능

  => 인덱스 이름(index name) or 인덱스 라벨(index label)

```
# 인덱스 배열은 변수 idx에 저장. 데이터 값 배열은 변수 val에 저장
idx = sr.index
val = sr.values
print(idx)
print('₩n')
print(val)
```

```
RangeIndex(start=0, stop=5, step=1)


['2019-01-02' 3.14 'ABC' 100 True]
```

- 인덱스 배열 : Series객체.index
- 데이터 값 배열 : Series객체.values

1. 판다스 입문
- 시리즈(원소 선택)

```
# 투플을 시리즈로 변환(index 옵션에 인덱스 이름을 지정)
tup_data = ('영인', '2010-05-01', '여', True)
sr = pd.Series(tup_data, index=['이름', '생년월일', '성별', '학생여부'])
print(sr)
print('₩n')
```

```
이름              영인
생년월일    2010-05-01
성별                여
학생여부          True
dtype: object
```

- 투플을 시리즈로 변환
- Index 옵션에 인덱스 이름을 지정
- 데이터 값의 자료형(dtype)은 문자열

```
# 원소를 1개 선택
print(sr[0])       # sr의 1 번째 원소를 선택 (정수형 위치 인덱스를 활용)
print(sr['이름'])  # '이름' 라벨을 가진 원소를 선택 (인덱스 이름을 활용)
```

```
영인
영인
```

- 원소를 1개 선택하는 두 가지 방법

1. 판다스 입문
- 시리즈(원소 선택)

```
# 여러 개의 원소를 선택 (인덱스 리스트 활용)
print(sr[[1, 2]])
print('₩n')
print(sr[['생년월일', '성별']])
```

```
생년월일    2010-05-01
성별             여
dtype: object


생년월일    2010-05-01
성별             여
dtype: object
```

- 여러 개의 원소를 선택
- 인덱스 리스트 활용

```
# 여러 개의 원소를 선택 (인덱스 범위 지정)
print(sr[1 : 2])
print('₩n')
print(sr['생년월일' : '성별'])
```

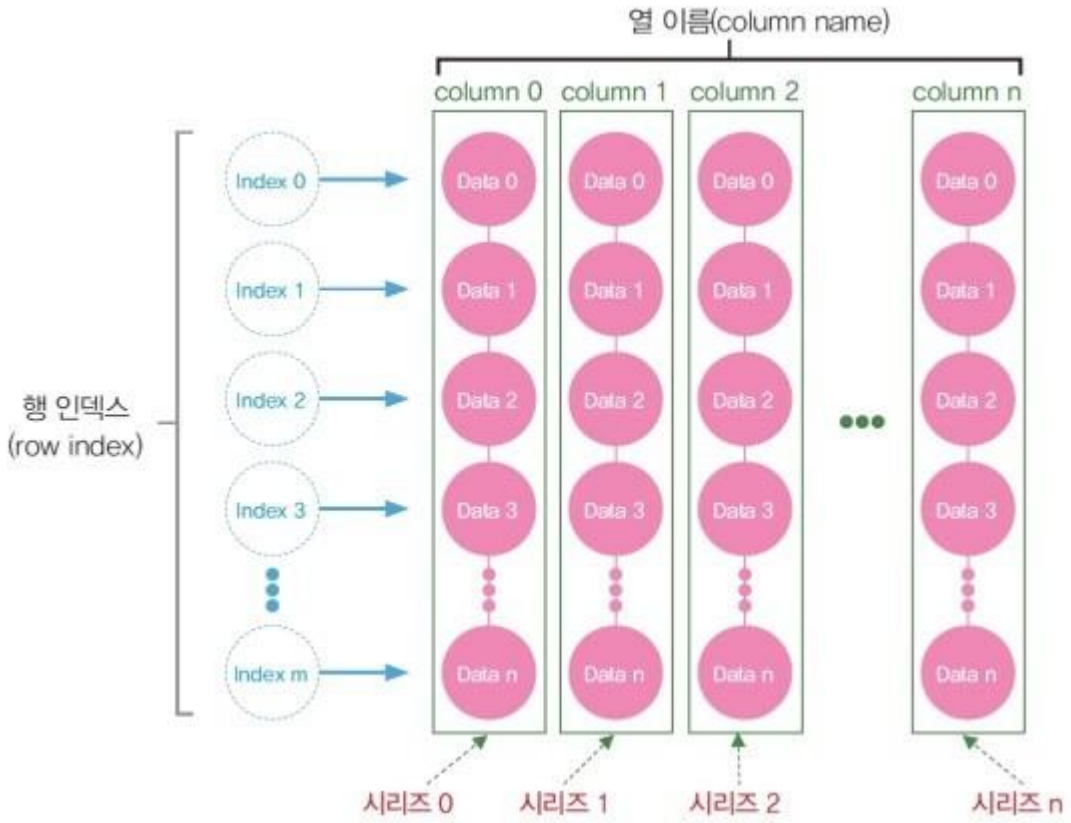
```
생년월일    2010-05-01
dtype: object


생년월일    2010-05-01
성별             여
dtype: object
```

- 인덱스 범위 지정
- [1:2] => 2는 포함하지 않음!

1. 판다스 입문
- 판다스 자료구조(데이터프레임)

" 데이터가 행과 열로 만들어진 2차원 배열의 형태 "



1. 판다스 입문
- 판다스 자료구조(데이터프레임)

" 여러 개의 리스트를 원소로 갖는 딕셔너리를 함수의 인자로 전달하는 방법을 많이 사용. "

```
# 열이름을 key로 하고, 리스트를 value로 갖는 딕셔너리 정의(2차원 배열)
dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}

# 판다스 DataFrame() 함수로 딕셔너리를 데이터프레임으로 변환. 변수 df에 저장.
df = pd.DataFrame(dict_data)

# df의 자료형 출력
print(type(df))
print('₩n')
# 변수 df에 저장되어 있는 데이터프레임 객체를 출력
print(df)
```

```
<class 'pandas.core.frame.DataFrame'>


   c0  c1  c2  c3  c4
0   1   4   7  10  13
1   2   5   8  11  14
2   3   6   9  12  15
```

- 판다스 함수 DataFrame()를 이용
- 키는 열 이름이 됨
- 값에 해당하는 각 리스트가 열이 됨
- 행 인덱스는 정수형 위치 인덱스가 자동 지정됨

2. 데이터프레임 핸들링
- 데이터프레임(행 인덱스/열 이름 설정)

```
# 행 인덱스/열 이름 지정하여, 데이터프레임 만들기
df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']],
                  index=['준서', '예은'],
                  columns=['나이', '성별', '학교'])
# 행 인덱스, 열 이름 확인하기
print(df)           #데이터프레임
print('₩n')
print(df.index)     #행 인덱스
print('₩n')
print(df.columns)   #열 이름
```

```
    나이 성별   학교
준서  15  남  덕영중
예은  17  여  수리중


Index(['준서', '예은'], dtype='object')


Index(['나이', '성별', '학교'], dtype='object')
```

- pd.DataFrame(2차원 배열, index = 행 인덱스 배열, columns = 열 이름 배열)
- 실행결과 리스트가 행으로 변환
- 행 인덱스 접근 => df.index
- 열 인덱스 접근 => df.columns

2. 데이터프레임 핸들링
- 데이터프레임(행 인덱스/열 이름 변경)

```
# 행 인덱스, 열 이름 변경하기
df.index=['학생1', '학생2']
df.columns=['연령', '남녀', '소속']
print(df)            #데이터프레임
print('₩n')
print(df.index)      #행 인덱스
print('₩n')
print(df.columns)    #열 이름
```

```
    연령 남녀   소속
학생1  15  남  덕영중
학생2  17  여  수리중


Index(['학생1', '학생2'], dtype='object')


Index(['연령', '남녀', '소속'], dtype='object')
```

```
# 열 이름 중, '나이'를 '연령'으로, '성별'을 '남녀'로, '학교'를 '소속'으로 바꾸기
df.rename(columns={'나이':'연령', '성별':'남녀', '학교':'소속'}, inplace=True)

# df의 행 인덱스 중에서, '준서'를 '학생1'로, '예은'을 '학생2'로 바꾸기
df.rename(index={'준서':'학생1', '예은':'학생2' }, inplace=True)

# df 출력(변경 후)
print(df)
```

```
    나이 성별   학교
준서  15  남  덕영중
예은  17  여  수리중


    연령 남녀   소속
학생1  15  남  덕영중
학생2  17  여  수리중
```

“ 두 방법 다 변경할 수 있으나 ,
오른쪽 경우는 일부분의 행 인덱스와 열 이름 변경도 가능”

2. 데이터프레임 핸들링
- 데이터프레임(행 삭제)

```
# DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
exam_data = {'수학' : [ 90, 80, 70], '영어' : [ 98, 89, 95],
             '음악' : [ 85, 95, 100], '체육' : [ 100, 90, 90]}

df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])
print(df)
print('₩n')

# 데이터프레임 df를 복제하여 변수 df2에 저장. df2의 1개 행(row)을 삭제
df2 = df[:]
df2.drop('우현', inplace=True)
print(df2)
print('₩n')

# 데이터프레임 df를 복제하여 변수 df3에 저장. df3의 2개 행(row)을 삭제
df3 = df[:]
df3.drop(['우현', '인아'], axis=0, inplace=True)
print(df3)
```

```
    수학  영어   음악   체육
서준  90  98   85  100
우현  80  89   95   90
인아  70  95  100   90


    수학  영어   음악   체육
서준  90  98   85  100
인아  70  95  100   90


    수학  영어  음악   체육
서준  90  98  85  100
```

- 행 삭제 : DataFrame 객체.drop(행 인덱스 또는 배열, axis=0)
- drop() 메소드는 기존 객체를 변경하지 않음
- 원본 객체를 변경하기 위해서는
  => inplace=True 옵션을 추가

2. 데이터프레임 핸들링
- 데이터프레임(열 삭제)

```
# 데이터프레임 df를 복제하여 변수 df4에 저장. df4의 1개 열(column)을 삭제
df4 = df[:]
df4.drop('수학', axis=1, inplace=True)
print(df4)
print('₩n')

# 데이터프레임 df를 복제하여 변수 df5에 저장. df5의 2개 열(column)을 삭제
df5 = df[:]
df5.drop(['영어', '음악'], axis=1, inplace=True)
print(df5)
```

```
    영어   음악   체육
서준  98   85  100
우현  89   95   90
인아  95  100   90


    수학   체육
서준  90  100
우현  80   90
인아  70   90
```

- 열 삭제 : DataFrame 객체.drop(열 이름 또는 배열, axis=1)
- drop() 메소드는 기존 객체를 변경하지 않음
- 원본 객체를 변경하기 위해서는
  => inplace=True 옵션을 추가

2. 데이터프레임 핸들링
- 데이터프레임(행 선택)

```python
# 행 인덱스를 사용하여 행 1개를 선택
label1 = df.loc['서준']     # loc 인덱서 활용
position1 = df.iloc[0]      # iloc 인덱서 활용
print(label1)
print('₩n')
print(position1)
```

```
수학     90
영어     98
음악     85
체육    100
Name: 서준, dtype: int64


수학     90
영어     98
음악     85
체육    100
Name: 서준, dtype: int64
```

```python
# 행 인덱스를 사용하여 2개 이상의 행 선택
label2 = df.loc[['서준', '우현']]
position2 = df.iloc[[0, 1]]
print(label2)
print('₩n')
print(position2)
```

```
    수학  영어  음악   체육
서준  90  98  85  100
우현  80  89  95   90


    수학  영어  음악   체육
서준  90  98  85  100
우현  80  89  95   90
```

```python
# 행 인덱스의 범위를 지정하여 행 선택
label3 = df.loc['서준':'우현']
position3 = df.iloc[0:1]
print(label3)
print('₩n')
print(position3)
```

```
    수학  영어  음악   체육
서준  90  98  85  100
우현  80  89  95   90


    수학  영어  음악   체육
서준  90  98  85  100
```

구 분	loc	iloc
탐색 대상	인덱스 이름(index label)	정수형 위치 인덱스(integer position)
범위 지정	가능(범위의 끝 포함) 예) ['a' : 'c'] -> 'a', 'b', 'c'	가능(범위의 끝 제외) 예) [3 : 7] -> 3, 4, 5, 6(7 제외)

2. 데이터프레임 핸들링
- 데이터프레임(열 선택)

```
# '수학' 점수 데이터만 선택, 변수 math1에 저장
math1 = df['수학']
print(math1)
print(type(math1))
print('₩n')
```

```
서준    90
우현    80
인아    70
Name: 수학, dtype: int64
<class 'pandas.core.series.Series'>
```

- 열 1개 선택 : df[열1] or df.열1
- 시리즈 객체로 추출

```
# '음악', '체육' 점수 데이터를 선택, 변수 music_gym 에 저장
music_gym = df[['음악', '체육']]
print(music_gym)
print(type(music_gym))
print('₩n')
```

```
    음악   체육
서준   85  100
우현   95   90
인아  100   90
<class 'pandas.core.frame.DataFrame'>
```

- 열 n개 선택 : df[[열1, 열2, … , 열n]]
- 데이터프레임 객체로 추출

```
# '수학' 점수 데이터만 선택, 변수 math2에 저장
math2 = df[['수학']]
print(math2)
print(type(math2))
```

```
    수학
서준  90
우현  80
인아  70
<class 'pandas.core.frame.DataFrame'>
```

- 열 1개를 데이터프레임으로 추출
  : df[[‘열1’]]

2. 데이터프레임 핸들링
- 데이터프레임(원소 선택)

	수학	영어	음악	체육
서준	90	98	85	100
우현	80	89	95	90
인아	70	95	100	90

```
# 데이터프레임 df의 특정 원소 1개 선택 ('서준'의 '음악' 점수)
a = df.loc['서준', '음악']
print(a)
b = df.iloc[0, 2]
print(b)
```

```
85
85
```

```
# 데이터프레임 df의 특정 원소 2개 이상 선택 ('서준'의 '음악', '체육' 점수)
c = df.loc['서준', ['음악', '체육']]
print(c)
d = df.iloc[0, [2, 3]]
print(d)
e = df.loc['서준', '음악':'체육']
print(e)
f = df.iloc[0, 2:]
print(f)
```

```
음악     85
체육    100
Name: 서준, dtype: int64
음악     85
체육    100
Name: 서준, dtype: int64
음악     85
체육    100
Name: 서준, dtype: int64
음악     85
체육    100
Name: 서준, dtype: int64
```

“ 시리즈 형 ”

```
# df의 2개 이상의 행과 열로부터 원소 선택 ('서준', '우현'의 '음악', '체육' 점수)
g = df.loc[['서준', '우현'], ['음악', '체육']]
print(g)
h = df.iloc[[0, 1], [2, 3]]
print(h)
i = df.loc['서준':'우현', '음악':'체육']
print(i)
j = df.iloc[0:2, 2:]
print(j)
```

```
    음악   체육
서준  85  100
우현  95   90
    음악   체육
서준  85  100
우현  95   90
    음악   체육
서준  85  100
우현  95   90
    음악   체육
서준  85  100
우현  95   90
```

“ 데이터프레임 형 ”

2. 데이터프레임 핸들링
- 데이터프레임(열 추가)

```
# 데이터프레임 df에 '국어' 점수 열(column)을 추가. 데이터 값은 80 지정
df['국어'] = 80
df
```

	수학	영어	음악	체육	국어
서준	90	98	85	100	80
우현	80	89	95	90	80
인아	70	95	100	90	80

- 열 추가 : DataFrame 객체[‘추가하려는 열 이름‘] = 데이터 값
- 모든 행에 동일한 값이 입력된다.
- 각자 다른 값을 넣을 때는 리스트를 입력하면 된다.

2. 데이터프레임 핸들링
- 데이터프레임(행 추가)

```
# 앞에서 만들었던 데이터프레임 다시 만들기!!
exam_data = {'이름' : ['서준', '우현', '인아'],
             '수학' : [ 90, 80, 70],
             '영어' : [ 98, 89, 95],
             '음악' : [ 85, 95, 100],
             '체육' : [ 100, 90, 90]}
df = pd.DataFrame(exam_data)
df
```

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90

‘행 추가하기 전에 앞에서 만들었던 데이터프레임 다시 만들어주세요!’

2. 데이터프레임 핸들링
- 데이터프레임(행 추가)

```
# 새로운 행(row)을 추가 - 같은 원소 값을 입력
df.loc[3] = 0
df
```

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90
3	0	0	0	0	0

```
# 새로운 행(row)을 추가 - 원소 값 여러 개의 배열 입력
df.loc[4] = ['동규', 90, 80, 70, 60]
df
```

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90
3	0	0	0	0	0
4	동규	90	80	70	60

- 행 추가 : DataFrame.loc[‘새로운 행 이름‘] = 데이터 값 (또는 배열)
- 데이터 값 입력 시 동일한 값이 입력
- 다른 값을 넣을 때 여러 개의 배열 입력
- df.loc[‘새로운 행 이름’] = df.loc[‘기존 행 이름’] : 새로운 행에 기존 행 복사

2. 데이터프레임 핸들링
- 데이터프레임(원소 값 변경)

" 아까 부른 데이터프레임 다시 불러주세요.^^ "

```
# 데이터프레임 df의 특정 원소를 변경하는 방법: '서준'의 '체육' 점수
df.iloc[0][3] = 80
df
```

	수학	영어	음악	체육
서준	90	98	85	80
우현	80	89	95	90
인아	70	95	100	90

```
df.loc['서준']['체육'] = 90
df
```

	수학	영어	음악	체육
서준	90	98	85	90
우현	80	89	95	90
인아	70	95	100	90

```
df.loc['서준', '체육'] = 100
df
```

	수학	영어	음악	체육
서준	90	98	85	100
우현	80	89	95	90
인아	70	95	100	90

- 3가지 방법으로 한 개의 원소 값 변경 가능
- df.iloc[m][n] = 변경할 데이터 값
- df.loc['행 이름']['열 이름'] = 변경할 데이터 값
- df.loc['행 이름', '열 이름'] = 변경할 데이터 값

2. 데이터프레임 핸들링
- 데이터프레임(여러 개 원소 값 변경)

```
# 데이터프레임 df의 원소 여러 개를 변경하는 방법: '서준'의 '음악', '체육' 점수
df.loc['서준', ['음악', '체육']] = 50
df
```

	수학	영어	음악	체육
서준	90	98	50	50
우현	80	89	95	90
인아	70	95	100	90

```
df.loc['서준', ['음악', '체육']] = 100, 50
df
```

	수학	영어	음악	체육
서준	90	98	100	50
우현	80	89	95	90
인아	70	95	100	90

- 여러 개의 원소 값 변경
- 데이터 값만 입력 시 동일한 값으로 변경
- 다른 값으로 변경 시 입력 순서대로 변경

2. 데이터프레임 핸들링

- 데이터프레임(행, 열의 위치 바꾸기) " 아까 부른 데이터프레임 다시 불러주세요.^^ "

```
df=df.transpose()
df
```

	서준	우현	인아
수학	90	80	70
영어	98	89	95
음악	100	95	100
체육	50	90	90

```
df=df.T
df
```

	서준	우현	인아
수학	90	80	70
영어	98	89	95
음악	100	95	100
체육	50	90	90

- 행, 열 바꾸기 : DataFrame 객체.transpose() or DataFrame 객체.T

2. 데이터프레임 핸들링
- 인덱스 활용(특정 열을 행 인덱스로 설정)

```
# DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장
exam_data = {'이름' : [ '서준', '우현', '인아'],
             '수학' : [ 90, 80, 70],
             '영어' : [ 98, 89, 95],
             '음악' : [ 85, 95, 100],
             '체육' : [ 100, 90, 90]}
df = pd.DataFrame(exam_data)
df
```

	이름	수학	영어	음악	체육
0	서준	90	98	85	100
1	우현	80	89	95	90
2	인아	70	95	100	90

```
# 특정 열(column)을 데이터프레임의 행 인덱스(index)로 설정
ndf = df.set_index(['이름'])
ndf
```

이름	수학	영어	음악	체육
서준	90	98	85	100
우현	80	89	95	90
인아	70	95	100	90

- 특정 열을 행 인덱스로 설정

  : DataFrame 객체.set_index(['열 이름'] or '열 이름')

2. 데이터프레임 핸들링
- 인덱스 활용(특정 열을 행 인덱스로 설정)

```
# 딕셔너리를 정의
dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}

# 딕셔너리를 데이터프레임으로 변환. 인덱스를 [r0, r1, r2]로 지정
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df)
print('₩n')

# 인덱스를 [r0, r1, r2, r3, r4]로 재지정
new_index = ['r0', 'r1', 'r2', 'r3', 'r4']
ndf = df.reindex(new_index)
print(ndf)
print('₩n')

# reindex로 발생한 NaN값을 숫자 0으로 채우기
new_index = ['r0', 'r1', 'r2', 'r3', 'r4']
ndf2 = df.reindex(new_index, fill_value=0)
print(ndf2)
```

```
    c0  c1  c2  c3  c4
r0   1   4   7  10  13
r1   2   5   8  11  14
r2   3   6   9  12  15
```

```
     c0   c1   c2    c3    c4
r0  1.0  4.0  7.0  10.0  13.0
r1  2.0  5.0  8.0  11.0  14.0
r2  3.0  6.0  9.0  12.0  15.0
r3  NaN  NaN  NaN   NaN   NaN
r4  NaN  NaN  NaN   NaN   NaN
```

```
    c0  c1  c2  c3  c4
r0   1   4   7  10  13
r1   2   5   8  11  14
r2   3   6   9  12  15
r3   0   0   0   0   0
r4   0   0   0   0   0
```

2. 데이터프레임 핸들링
- 인덱스 활용(행 인덱스 초기화/행 인덱스 기준으로 데이터프레임 정렬)

```
ndf = df.reset_index()
print(ndf)
```

```
  index  c0  c1  c2  c3  c4
0    r0   1   4   7  10  13
1    r1   2   5   8  11  14
2    r2   3   6   9  12  15
```

- 정수형 위치 인덱스로 초기화

  : DataFrame 객체.reset_index()

```
# 내림차순으로 행 인덱스 정렬
ndf = df.sort_index(ascending=False)
print(ndf)
```

```
    c0  c1  c2  c3  c4
r2   3   6   9  12  15
r1   2   5   8  11  14
r0   1   4   7  10  13
```

- 행 인덱스 기준으로 정렬

  : DataFrame 객체.sort_index()

```
# c1 열을 기준으로 내림차순 정렬
ndf = df.sort_values(by='c1', ascending=False)
print(ndf)
```

```
    c0  c1  c2  c3  c4
r2   3   6   9  12  15
r1   2   5   8  11  14
r0   1   4   7  10  13
```

- 특정 열 기준으로 정렬

  : DataFrame 객체.sort_values(by='열 이름')

*ascending=False 는 내림차순, 디폴트 값은 오름차순

2. 데이터프레임 핸들링
- 열 순서 변경(리스트 이용)

```
# 라이브러리 불러오기
import seaborn as sns

# titanic 데이터셋의 부분을 선택하여 데이터프레임 만들기
titanic=sns.load_dataset('titanic')
df=titanic.loc[0:4, 'survived':'age']
df
```

	survived	pclass	sex	age
0	0	3	male	22.0
1	1	1	female	38.0
2	1	3	female	26.0
3	1	1	female	35.0
4	0	3	male	35.0

```
# 열 이름의 리스트 만들기
columns=list(df.columns.values) #기존 열 이름
print(columns)
```

```
['survived', 'pclass', 'sex', 'age']
```

- STEP 1. 열 이름을 원하는 순서대로 정리 => 리스트 생성

- STEP 2. 데이터프레임에서 열을 다시 선택
  - 1) sorted() : 알파벳 순
  - 2) reversed() : 기존의 역순
  - 3) List 생성 : 사용자의 임의

2. 데이터프레임 핸들링
- 열 순서 변경(리스트 이용)

```
# sorted() : 알파벳
columns_sorted=sorted(columns)
df_sorted=df[columns_sorted]
df_sorted
```

	age	pclass	sex	survived
0	22.0	3	male	0
1	38.0	1	female	1
2	26.0	3	female	1
3	35.0	1	female	1
4	35.0	3	male	0

```
# reversed() : 기존 순서의 역순
columns_reversed=list(reversed(columns))
df_reversed=df[columns_reversed]
df_reversed
```

	age	sex	pclass	survived
0	22.0	male	3	0
1	38.0	female	1	1
2	26.0	female	3	1
3	35.0	female	1	1
4	35.0	male	3	0

```
# List 생성 : 사용자의 임의
columns_customed=['pclass', 'sex', 'age', 'survived']
df_customed=df[columns_customed]
df_customed
```

	pclass	sex	age	survived
0	3	male	22.0	0
1	1	female	38.0	1
2	3	female	26.0	1
3	1	female	35.0	1
4	3	male	35.0	0

3. 데이터프레임 살펴보기 및 필터링
- 데이터프레임 메소드 활용법

" DataFrame 클래스에 대해 메소드를 활용할 때"

DataFrame.메소드()

DataFrame['변수'].메소드()

DataFrame[['변수1','변수2',…]].메소드()

1. 변수는 열 이름
2. 다수의 열 일 때 [[ ]] 사용, 실수 조심!!
3. 모든 메소드가 다 적용 X

3. 데이터프레임 살펴보기 및 필터링
- 데이터 살펴보기(head(), tail())

```
# read_csv() 함수로 df 생성
df = pd.read_csv('./auto-mpg.csv', header=None)

# 열 이름을 지정
df.columns = ['mpg','cylinders','displacement','horsepower','weight',
              'acceleration','model year','origin','name']

# 데이터프레임 df의 내용을 일부 확인
print(df.head())     # 처음 5개의 행
print('₩n')
print(df.tail())     # 마지막 5개의 행
```

```
    mpg  cylinders  displacement horsepower  weight  acceleration  model year
0  18.0          8         307.0      130.0  3504.0          12.0          70
1  15.0          8         350.0      165.0  3693.0          11.5          70
2  18.0          8         318.0      150.0  3436.0          11.0          70
3  16.0          8         304.0      150.0  3433.0          12.0          70
4  17.0          8         302.0      140.0  3449.0          10.5          70

   origin                       name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino


      mpg  cylinders  displacement horsepower  weight  acceleration  ₩
393  27.0          4         140.0      86.00  2790.0          15.6
394  44.0          4          97.0      52.00  2130.0          24.6
395  32.0          4         135.0      84.00  2295.0          11.6
396  28.0          4         120.0      79.00  2625.0          18.6
397  31.0          4         119.0      82.00  2720.0          19.4

     model year  origin            name
393          82       1  ford mustang gl
394          82       2         vw pickup
395          82       1     dodge rampage
396          82       1       ford ranger
397          82       1        chevy s-10
```

앞부분 미리보기 : DataFrame 객체.head(n)
뒷부분 미리보기 : DataFrame 객체.tail(n)

3. 데이터프레임 살펴보기 및 필터링
- 데이터 살펴보기(shape, info())

```
# df의 모양과 크기 확인: (행의 개수, 열의 개수)를 튜플로 반환
print(df.shape)
```

```
(398, 9)
```

```
# 데이터프레임 df의 내용 확인
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
mpg             398 non-null float64
cylinders       398 non-null int64
displacement    398 non-null float64
horsepower      398 non-null object
weight          398 non-null float64
acceleration    398 non-null float64
model year      398 non-null int64
origin          398 non-null int64
name            398 non-null object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
None
```

- 데이터프레임의 크기 확인
  : DataFrame 객체.shape

=> 행의 개수와 열의 개수를 투플로 반환

- 데이터프레임의 기본 정보 출력
  : DataFrame 객체.info()

⇒ 첫 행에 df의 클래스 유형

⇒ 행 인덱스와 열에 대한 정보

⇒ 각 열의 이름과 데이터 개수, 자료형

⇒ 자료형과 메모리 사용량 표시

* 번외
- 판다스 자료형(data type)

판다스 자료형	파이썬 자료형	설명
object	string	문자열
int64	int	정수
float64	float	소수점을 가진 숫자
datetime64	datetime	파이썬 표준 라이브러리인 datetime이 반환하는 자료형

```
# 데이터프레임 df의 자료형 확인
print(df.dtypes)
print('₩n')

# 시리즈(mog 열)의 자료형 확인
print(df.mpg.dtypes)
```

```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight          float64
acceleration    float64
model year        int64
origin            int64
name             object
dtype: object


float64
```

info() 메소드 외 데이터프레임 클래스의 dtypes 속성을 활용하여 각 열의 자료형을 확인할 수 있다!

3. 데이터프레임 살펴보기 및 필터링
- 데이터 살펴보기(describe())

```
# 데이터프레임 df의 기술통계 정보 확인
print(df.describe())
print('₩n')
print(df.describe(include='all'))
```

- 데이터프레임의 기술 통계 정보 요약
  : DataFrame 객체.describe()

- 산술 데이터가 아닌 열에 대한 정보를 포함할 때는 include='all' 옵션 추가

- 고유값 개수, 최빈값, 빈도수에는 추가, 산술 데이터에는 NaN 값이 표시

```
              mpg   cylinders  displacement        weight  acceleration
count  398.000000  398.000000    398.000000    398.000000    398.000000
mean    23.514573    5.454774    193.425879   2970.424623     15.568090
std      7.815984    1.701004    104.269838    846.841774      2.757689
min      9.000000    3.000000     68.000000   1613.000000      8.000000
25%     17.500000    4.000000    104.250000   2223.750000     13.825000
50%     23.000000    4.000000    148.500000   2803.500000     15.500000
75%     29.000000    8.000000    262.000000   3608.000000     17.175000
max     46.600000    8.000000    455.000000   5140.000000     24.800000

       model year      origin
count  398.000000  398.000000
mean    76.010050    1.572864
std      3.697627    0.802055
min     70.000000    1.000000
25%     73.000000    1.000000
50%     76.000000    1.000000
75%     79.000000    2.000000
max     82.000000    3.000000

              mpg   cylinders  displacement horsepower       weight
count  398.000000  398.000000    398.000000        398   398.000000
unique        NaN         NaN           NaN         94          NaN
top           NaN         NaN           NaN      150.0          NaN
freq          NaN         NaN           NaN         22          NaN
mean    23.514573    5.454774    193.425879        NaN  2970.424623
std      7.815984    1.701004    104.269838        NaN   846.841774
min      9.000000    3.000000     68.000000        NaN  1613.000000
25%     17.500000    4.000000    104.250000        NaN  2223.750000
50%     23.000000    4.000000    148.500000        NaN  2803.500000
75%     29.000000    8.000000    262.000000        NaN  3608.000000
max     46.600000    8.000000    455.000000        NaN  5140.000000

       acceleration  model year      origin        name
count    398.000000  398.000000  398.000000         398
unique          NaN         NaN         NaN         305
top             NaN         NaN         NaN  ford pinto
freq            NaN         NaN         NaN           6
mean      15.568090   76.010050    1.572864         NaN
std        2.757689    3.697627    0.802055         NaN
min        8.000000   70.000000    1.000000         NaN
25%       13.825000   73.000000    1.000000         NaN
50%       15.500000   76.000000    1.000000         NaN
75%       17.175000   79.000000    2.000000         NaN
max       24.800000   82.000000    3.000000         NaN
```

3. 데이터프레임 살펴보기 및 필터링
- 데이터 살펴보기(describe())

```
df.describe(percentiles=[.1,.2,.3])
```

	mpg	cylinders	displacement	weight	acceleration	model year	origin
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	2970.424623	15.568090	76.010050	1.572864
std	7.815984	1.701004	104.269838	846.841774	2.757689	3.697627	0.802055
min	9.000000	3.000000	68.000000	1613.000000	8.000000	70.000000	1.000000
10%	14.000000	4.000000	90.000000	1988.500000	12.000000	71.000000	1.000000
20%	16.000000	4.000000	98.000000	2155.000000	13.500000	72.000000	1.000000
30%	18.000000	4.000000	112.000000	2301.000000	14.200000	73.000000	1.000000
50%	23.000000	4.000000	148.500000	2803.500000	15.500000	76.000000	1.000000
max	46.600000	8.000000	455.000000	5140.000000	24.800000	82.000000	3.000000

- 백분위구간 지정도 설정 가능(예시 참고)
  : df.describe(percentiles=[.1,.2,.3])

```
df.describe().astype('int')
```

	mpg	cylinders	displacement	weight	acceleration	model year	origin
count	398	398	398	398	398	398	398
mean	23	5	193	2970	15	76	1
std	7	1	104	846	2	3	0
min	9	3	68	1613	8	70	1
25%	17	4	104	2223	13	73	1
50%	23	4	148	2803	15	76	1
75%	29	8	262	3608	17	79	2
max	46	8	455	5140	24	82	3

- astype 메소드를 사용하여 정수형
  (또는 문자형)으로 간략히 표현 가능
  : df.describe().astype('int'(또는 'str'))

3. 데이터프레임 살펴보기 및 필터링
- 데이터 살펴보기(count(), value_counts())

```
# 데이터프레임 df의 각 열이 가지고 있는 원소 개수 확인
print(df.count())
```

```
mpg             398
cylinders       398
displacement    398
horsepower      398
weight          398
acceleration    398
model year      398
origin          398
name            398
dtype: int64
```

- 열 데이터 개수 확인

  : DataFrame 객체.count()

```
# 데이터프레임 df의 특정 열이 가지고 있는 고유값 확인
unique_values = df['origin'].value_counts()
print(unique_values)
```

```
1    249
3     79
2     70
Name: origin, dtype: int64
```

- 열 데이터의 고유값 개수

  : DataFrame 객체['열 이름'].value_counts()

3. 데이터프레임 살펴보기 및 필터링
- 통계 함수 적용(평균값)

```
# 평균값
print(df.mean())
print('₩n')
print(df['mpg'].mean())
print(df.mpg.mean())
print('₩n')
print(df[['mpg','weight']].mean())
```

```
mpg              23.514573
cylinders         5.454774
displacement    193.425879
weight         2970.424623
acceleration     15.568090
model year       76.010050
origin            1.572864
dtype: float64


23.514572864321615
23.514572864321615


mpg         23.514573
weight    2970.424623
dtype: float64
```

- 모든 열의 평균값
  : DataFrame 객체.mean()

- 특정 열의 평균값
  : DataFrame 객체['열 이름'].mean()

- 다른 통계 함수도 똑같이 적용한다.

3. 데이터프레임 살펴보기 및 필터링
- 통계 함수 적용(중간값, 최대값, 최소값, 표준편차, 상관계수)

```
# 중간값
print(df.median())
print('₩n')
print(df['mpg'].median())

# 최대값
print(df.max())
print('₩n')
print(df['mpg'].max())

# 최소값
print(df.min())
print('₩n')
print(df['mpg'].min())

# 표준편차
print(df.std())
print('₩n')
print(df['mpg'].std())

# 상관계수
print(df.corr())
print('₩n')
print(df[['mpg','weight']].corr())
```

```
mpg            23.0
cylinders       4.0
displacement  148.5
weight       2803.5
acceleration   15.5
model year     76.0
origin          1.0
dtype: float64


23.0
mpg                      46.6
cylinders                   8
displacement              455
horsepower                  ?
weight                   5140
acceleration             24.8
model year                 82
origin                      3
name           vw rabbit custom
dtype: object


7.815984312565782
                mpg  cylinders  displacement    weight  acceleration
mpg        1.000000  -0.775396     -0.804203 -0.831741      0.420289
cylinders -0.775396   1.000000      0.950721  0.896017     -0.505419
displacement -0.804203   0.950721      1.000000  0.932824     -0.543684
weight     -0.831741   0.896017      0.932824  1.000000     -0.417457
acceleration  0.420289  -0.505419     -0.543684 -0.417457      1.000000
model year   0.579267  -0.348746     -0.370164 -0.306564      0.288137
origin       0.563450  -0.562543     -0.609409 -0.581024      0.205873

              model year    origin
mpg             0.579267  0.563450
cylinders      -0.348746 -0.562543
displacement   -0.370164 -0.609409
weight         -0.306564 -0.581024
acceleration    0.288137  0.205873
model year      1.000000  0.180662
origin          0.180662  1.000000


              mpg    weight
mpg      1.000000 -0.831741
weight  -0.831741  1.000000
```

```
46.6
mpg                          9
cylinders                    3
displacement                68
horsepower               100.0
weight                    1613
acceleration                 8
model year                  70
origin                       1
name           amc ambassador brougham
dtype: object


9.0
mpg             7.815984
cylinders       1.701004
displacement  104.269838
weight        846.841774
acceleration    2.757689
model year      3.697627
origin          0.802055
dtype: float64
```

3. 데이터프레임 살펴보기 및 필터링
- 불린 인덱싱

" 매우 편리한 데이터 필터링 방식 "

```
# 라이브러리 불러오기
import seaborn as sns

# titanic 데이터셋의 부분을 선택하여 데이터프레임 만들기
titanic=sns.load_dataset('titanic')

# age가 60 이상인 승객만 추출
titanic_boolean = titanic[titanic['age'] > 60]
print(type(titanic_boolean))
titanic_boolean.head()
```

- 새롭게 타이타닉 데이터 세트를 DataFrame으로 로드한 뒤, 승객 중 나이가 60세 이상인 데이터를 추출

- [] 연산자 내에 불린 조건을 입력하면 불린 인덱싱이 진행

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
33	0	2	male	66.0	0	0	10.5000	S	Second	man	True	NaN	Southampton	no	True
54	0	1	male	65.0	0	1	61.9792	C	First	man	True	B	Cherbourg	no	False
96	0	1	male	71.0	0	0	34.6542	C	First	man	True	A	Cherbourg	no	True
116	0	3	male	70.5	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True
170	0	1	male	61.0	0	0	33.5000	S	First	man	True	B	Southampton	no	True

3. 데이터프레임 살펴보기 및 필터링
- 불린 인덱싱(예시, AND)

```
titanic[titanic['age'] > 60][['sex','age']].head(3)
```

	sex	age
33	male	66.0
54	male	65.0
96	male	71.0

```
titanic[ (titanic['age'] > 60) & (titanic['pclass']==1) & (titanic['sex']=='female')]
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
275	1	1	female	63.0	1	0	77.9583	S	First	woman	False	D	Southampton	yes	False
829	1	1	female	62.0	0	0	80.0000	NaN	First	woman	False	B	NaN	yes	True

```
cond1 = titanic['age'] > 60
cond2 = titanic['pclass']==1
cond3 = titanic['sex']=='female'
titanic[ cond1 & cond2 & cond3]
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
275	1	1	female	63.0	1	0	77.9583	S	First	woman	False	D	Southampton	yes	False
829	1	1	female	62.0	0	0	80.0000	NaN	First	woman	False	B	NaN	yes	True

3. 데이터프레임 살펴보기 및 필터링
- 불린 인덱싱(예시, OR)

```
titanic[cond1|cond2].head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no	True
11	1	1	female	58.0	0	0	26.5500	S	First	woman	False	C	Southampton	yes	True
23	1	1	male	28.0	0	0	35.5000	S	First	man	True	A	Southampton	yes	True

3. 데이터프레임 살펴보기 및 필터링
- isin() 활용

“ 특정 값을 지닌 행들을 추출 ”

```
# isin() 메서드 활용
isin_filter = titanic['sibsp'].isin([3,4,5])
df_isin=titanic[isin_filter]
df_isin.head()
```

- ‘sibsp’값이 3, 4, 5인 행들을 추출
- 추출하려는 값들로 만든 리스트를 전달

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
7	0	3	male	2.0	3	1	21.0750	S	Third	child	False	NaN	Southampton	no	False
16	0	3	male	2.0	4	1	29.1250	Q	Third	child	False	NaN	Queenstown	no	False
24	0	3	female	8.0	3	1	21.0750	S	Third	child	False	NaN	Southampton	no	False
27	0	1	male	19.0	3	2	263.0000	S	First	man	True	C	Southampton	no	False
50	0	3	male	7.0	4	1	39.6875	S	Third	child	False	NaN	Southampton	no	False

3. 데이터프레임 살펴보기 및 필터링
- 불린 인덱싱과 isin활용() 비교

```
# 불린 인덱싱과 비교
mask3 = titanic['sibsp'] == 3
mask4 = titanic['sibsp'] == 4
mask5 = titanic['sibsp'] == 5
df_boolean = titanic[mask3|mask4|mask5]
df_boolean.head()
```

- 똑같은 결과가 나오는 것을 확인!

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
7	0	3	male	2.0	3	1	21.0750	S	Third	child	False	NaN	Southampton	no	False
16	0	3	male	2.0	4	1	29.1250	Q	Third	child	False	NaN	Queenstown	no	False
24	0	3	female	8.0	3	1	21.0750	S	Third	child	False	NaN	Southampton	no	False
27	0	1	male	19.0	3	2	263.0000	S	First	man	True	C	Southampton	no	False
50	0	3	male	7.0	4	1	39.6875	S	Third	child	False	NaN	Southampton	no	False

3. 데이터프레임 살펴보기 및 필터링
- Query 함수

“ 조건에 맞는 데이터를 추출 ”

- 장점 : 가독성과 편의성이 뛰어남
- 단점 : .loc[]로 구현한 것보다 속도가 느림

“ Query 함수의 여러 기능 ”

- 비교 연산자(==, >, >=, <, <=, !=)
- in 연산자(in, ==, not in, !=)
- 논리 연산자(and, or, not)
- 외부 변수(또는 함수) 참조 연산
- 인덱스 검색

3. 데이터프레임 살펴보기 및 필터링
- Query 함수(비교 연산자(==, >, >=, <, <=, !=))

```
# titanic 데이터셋의 부분을 선택하여 데이터프레임 만들기
titanic=sns.load_dataset('titanic')
str_expr = "age == 65"    # 나이가 65 이다 (비교연산자 ==)
titanic_q = titanic.query(str_expr) # 조건 부합 데이터 추출
titanic_q
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
54	0	1	male	65.0	0	1	61.9792	C	First	man	True	B	Cherbourg	no	False
280	0	3	male	65.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True
456	0	1	male	65.0	0	0	26.5500	S	First	man	True	E	Southampton	no	True

```
str_expr = "age >= 65"    # 나이가 65 이다 (비교연산자 ==)
titanic_q = titanic.query(str_expr) # 조건 부합 데이터 추출
titanic_q.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
33	0	2	male	66.0	0	0	10.5000	S	Second	man	True	NaN	Southampton	no	True
54	0	1	male	65.0	0	1	61.9792	C	First	man	True	B	Cherbourg	no	False
96	0	1	male	71.0	0	0	34.6542	C	First	man	True	A	Cherbourg	no	True
116	0	3	male	70.5	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True
280	0	3	male	65.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

- 칼럼명과 비교연산자 사용
- 칼럼의 값이 같거나 크거나 작거나 다름을 비교

3. 데이터프레임 살펴보기 및 필터링
- Query 함수(in 연산자(in, ==, not in, !=))

```
str_expr = "age in [65, 66]"  # 나이가 65 또는 66이다 (소문자 in 연산자)
# str_expr = "age == [65, 66]" # 위와 동일한 표현식이다
titanic_q = titanic.query(str_expr) # 조건 부합 데이터 추출
titanic_q
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
33	0	2	male	66.0	0	0	10.5000	S	Second	man	True	NaN	Southampton	no	True
54	0	1	male	65.0	0	1	61.9792	C	First	man	True	B	Cherbourg	no	False
280	0	3	male	65.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True
456	0	1	male	65.0	0	0	26.5500	S	First	man	True	E	Southampton	no	True

```
str_expr = "age not in [65, 66]" # 나이가 65 또는 66 아니다 (소문자 not in 연산자)
#str_expr = "age != [65, 66]"    # 위와 동일한 표현식이다
titanic_q = titanic.query(str_expr) # 조건 부합 데이터 추출
titanic_q.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

- 칼럼에 리스트 값이 하나라도 있으면 참
- 리스트와 튜플 형태 모두 가능

3. 데이터프레임 살펴보기 및 필터링
- Query 함수(논리 연산자(and, or, not))

```
str_expr = "(age == 65) and (pclass >= 1)" # 나이가 65이고 pclass 는 1 이상이다.
titanic_q = titanic.query(str_expr) # 조건 부합 데이터 추출
titanic_q.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
54	0	1	male	65.0	0	1	61.9792	C	First	man	True	B	Cherbourg	no	False
280	0	3	male	65.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True
456	0	1	male	65.0	0	0	26.5500	S	First	man	True	E	Southampton	no	True

```
str_expr = "(age == 65) or (pclass >= 1)" # 나이가 65이고 pclass 는 1 이상이다.
titanic_q = titanic.query(str_expr) # 조건 부합 데이터 추출
titanic_q.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

- and : 좌우 모두 참
- or : 좌우 중 하나
- not : 값의 반대

3. 데이터프레임 살펴보기 및 필터링
- Query 함수(외부 변수(또는 함수) 참조 연산)

```
num_age=35
str_expr = "age == @num_age"
titanic_q = titanic.query(str_expr) # 조건 부합 데이터 추출
titanic_q.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
3	1	1	female	35.0	1	0	53.100	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.050	S	Third	man	True	NaN	Southampton	no	True
20	0	2	male	35.0	0	0	26.000	S	Second	man	True	NaN	Southampton	no	True
211	1	2	female	35.0	0	0	21.000	S	Second	woman	False	NaN	Southampton	yes	True
230	1	1	female	35.0	1	0	83.475	S	First	woman	False	C	Southampton	yes	False

```
num_age    = 35
num_pclass = 2
str_expr = f"(age == {num_age}) and (pclass >= {num_pclass})"
titanic_q = titanic.query(str_expr) # 조건 부합 데이터 추출
titanic_q.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
4	0	3	male	35.0	0	0	8.05	S	Third	man	True	NaN	Southampton	no	True
20	0	2	male	35.0	0	0	26.00	S	Second	man	True	NaN	Southampton	no	True
211	1	2	female	35.0	0	0	21.00	S	Second	woman	False	NaN	Southampton	yes	True
279	1	3	female	35.0	1	1	20.25	S	Third	woman	False	NaN	Southampton	yes	False
363	0	3	male	35.0	0	0	7.05	S	Third	man	True	NaN	Southampton	no	True

- 외부 변수명 또는 함수명 앞에 @를 붙여 사용

- f-String를 이용하여 str_expr를 만들 때 외부변수를 미리 참조

3. 데이터프레임 살펴보기 및 필터링
- Query 함수(인덱스 검색)

```
str_expr = "index >= 2"     # 인덱스 2이상인 데이터
titanic_q = titanic.query(str_expr) # 조건 부합 데이터 추출
titanic_q.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
5	0	3	male	NaN	0	0	8.4583	Q	Third	man	True	NaN	Queenstown	no	True
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no	True

- 인덱스이름이 있다면 index대신 인덱스 이름을 기입
- 인덱스명이 칼럼 명과 겹친다면 칼럼명으로 간주되어 칼럼으로 연산

4. 데이터프레임 합치기
- 3가지 방법(연결, 병합, 결합)

```
# 데이터프레임 만들기
df1=pd.DataFrame({'a':['a0','a1','a2'],
                  'b':['b0','b1','b2'],
                  'c':['c0','c1','c2']},
                 index=[0,1,2])
df2=pd.DataFrame({'a':['a1','a2','a3'],
                  'b':['b1','b2','b3'],
                  'c':['c1','c2','c3'],
                  'd':['d1','d2','d3']},
                 index=[1,2,3])

print(df1)
print('₩n')
print(df2)
```

```
    a   b   c
0  a0  b0  c0
1  a1  b1  c1
2  a2  b2  c2


    a   b   c   d
1  a1  b1  c1  d1
2  a2  b2  c2  d2
3  a3  b3  c3  d3
```

BITAmin

- 연결(concat)
- 병합(merge)
- 결합(join)

4. 데이터프레임 합치기
- 연결(concat)

```
result1=pd.concat([df1,df2])
result1
```

	a	b	c	d
0	a0	b0	c0	NaN
1	a1	b1	c1	NaN
2	a2	b2	c2	NaN
1	a1	b1	c1	d1
2	a2	b2	c2	d2
3	a3	b3	c3	d3

- 데이터프레임을 행 방향으로 위 아래로 연결
- 축 방향을 지정하지 않으면 디폴트 값이 행 방향
- 행 인덱스는 본래 형태를 유지

4. 데이터프레임 합치기
- 연결(concat)

```
result2=pd.concat([df1,df2], ignore_index=True)
result2
```

	a	b	c	d
0	a0	b0	c0	NaN
1	a1	b1	c1	NaN
2	a2	b2	c2	NaN
3	a1	b1	c1	d1
4	a2	b2	c2	d2
5	a3	b3	c3	d3

- Ignore_index=True 옵션을 사용

- 기존 행 인덱스 무시하고 새로운 행 인덱스 설정

4. 데이터프레임 합치기
- 연결(concat)

```
result3=pd.concat([df1,df2],axis=1)
result3
```

	a	b	c	a	b	c	d
0	a0	b0	c0	NaN	NaN	NaN	NaN
1	a1	b1	c1	a1	b1	c1	d1
2	a2	b2	c2	a2	b2	c2	d2
3	NaN	NaN	NaN	a3	b3	c3	d3

- axis=1 옵션으로 좌우 열 방향으로 연결

- join='outer'은 디폴트 값으로 행 인덱스 합집합으로 구성

4. 데이터프레임 합치기
- 연결(concat)

```
result3_in=pd.concat([df1,df2],axis=1,join='inner')
result3_in
```

	a	b	c	a	b	c	d
1	a1	b1	c1	a1	b1	c1	d1
2	a2	b2	c2	a2	b2	c2	d2

- axis=1 옵션으로 좌우 열 방향으로 연결

- join='inner'은 디폴트 값으로 행 인덱스 교집합으로 기준

4. 데이터프레임 합치기
- 병합(merge)

```
merge_inner=pd.merge(df1, df2)
merge_inner
```

	a	b	c	d
0	a1	b1	c1	d1
1	a2	b2	c2	d2

- merge()함수 기본 값 : on=None, how='inner'

- on=None 옵션 : 공통으로 속하는 모든 열을 기준으로 병합

- how='inner' 옵션(교집합) : 기준이 되는 열의 데이터가 공통으로 존재하는 교집합일 경우에만 추출

4. 데이터프레임 합치기
- 병합(merge)

```
merge_outer=pd.merge(df1, df2, how='outer', on='c')
merge_outer
```

	a_x	b_x	c	a_y	b_y	d
0	a0	b0	c0	NaN	NaN	NaN
1	a1	b1	c1	a1	b1	d1
2	a2	b2	c2	a2	b2	d2
3	NaN	NaN	c3	a3	b3	d3

- on='c' 옵션 : 공통 열 중에서 'c'열을 키로 병합

- how='outer' 옵션(합집합) : 기준이 되는 'c'열의 데이터가 한 쪽에만 속하더라도 포함

4. 데이터프레임 합치기
- 병합(merge)

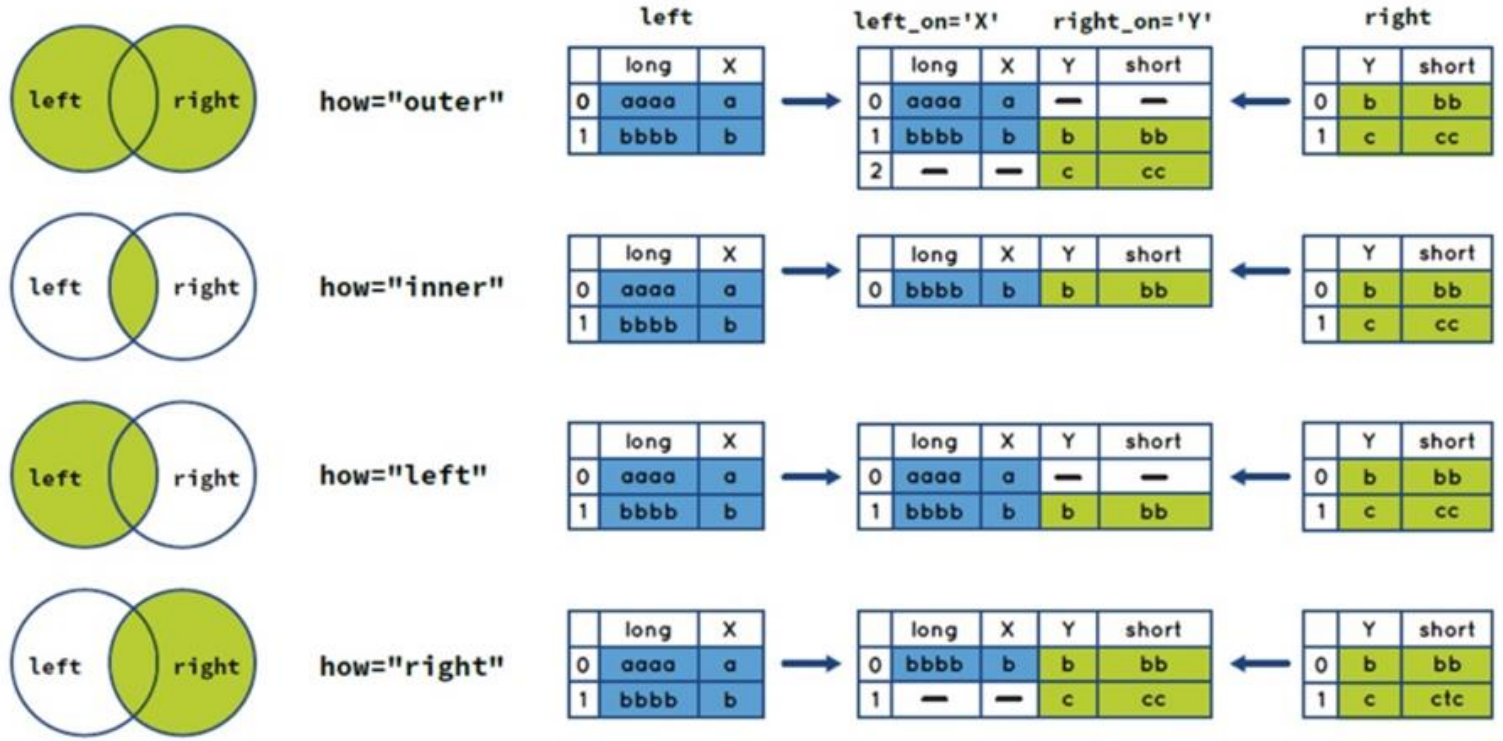
```
merge_left=pd.merge(df1, df2, how='left', on='c')
merge_left
```

	a_x	b_x	c	a_y	b_y	d
0	a0	b0	c0	NaN	NaN	NaN
1	a1	b1	c1	a1	b1	d1
2	a2	b2	c2	a2	b2	d2

- on='c' 옵션 : 공통 열 중에서 'c'열을 키로 병합

- how='left' 옵션 : 왼쪽 데이터프레임 키열에 속하는 데이터 값을 기준으로 병합

4. 데이터프레임 합치기
- 병합(merge)



- left_on, right_on 옵션을 사용하여 좌우 데이터 프레임에 각각 다르게 기준열 지정 가능
- ex) merge_left = pd.merge(df1, df2, how='left', left_on='X', right_on='Y')

4. 데이터프레임 합치기
- 연결(join)

```
# 데이터프레임 만들기
df1=pd.DataFrame({'a':['a0','a1','a2'],
                  'b':['b0','b1','b2'],
                  'c':['c0','c1','c2']},
                 index=[0,1,2])
df2=pd.DataFrame({'e':['e1','e2','e3'],
                  'f':['f1','f2','f3'],
                  'g':['g1','g2','g3'],
                  'h':['h1','h2','h3']},
                 index=[1,2,3])
df3=df1.join(df2)
df3
```

	a	b	c	e	f	g	h
0	a0	b0	c0	NaN	NaN	NaN	NaN
1	a1	b1	c1	e1	f1	g1	h1
2	a2	b2	c2	e2	f2	g2	h2

- merge()함수랑 비슷

- 행 인덱스를 기준으로 결합한다는 점에서 다름

5. 데이터프레임 연산
- 판다스 객체의 산술연산

“3단계 프로세스를 거친다.”

STEP 1. 행/열 인덱스를 기준으로 모든 원소를 정렬

STEP 2. 동일한 위치에 있는 원소끼리 일대일로 대응

STEP 3. 일대일 대응이 되는 원소끼리 연산을 처리
(단 대응되는 원소가 없으면 NaN으로 처리)

5. 데이터프레임 연산
- 데이터프레임 vs 숫자

```
titanic = sns.load_dataset('titanic')
df = titanic.loc[:, ['age','fare']]
print(df.head())   #첫 5행만 표시
print('₩n')
print(type(df))
print('₩n')

# 데이터프레임에 숫자 10 더하기
addition = df + 10
print(addition.head())   #첫 5행만 표시
print('₩n')
print(type(addition))
```

```
    age     fare
0  22.0   7.2500
1  38.0  71.2833
2  26.0   7.9250
3  35.0  53.1000
4  35.0   8.0500


<class 'pandas.core.frame.DataFrame'>


    age     fare
0  32.0  17.2500
1  48.0  81.2833
2  36.0  17.9250
3  45.0  63.1000
4  45.0  18.0500
```

- 기존 데이터프레임의 형태를 유지한 채 원소 값만 바뀜
- 새로운 데이터프레임 객체로 반환됨
- 데이터프레임과 숫자 연산

  : DataFrame 객체 + 연산자(+, -, *, /) + 숫자

5. 데이터프레임 연산
- 데이터프레임 vs 데이터프레임

```
# 데이터프레임끼리 연산하기 (additon - df)
subtraction = addition - df
print(subtraction.head())   #마지막 5행을 표시
```

	age	fare
0	10.0	10.0
1	10.0	10.0
2	10.0	10.0
3	10.0	10.0
4	10.0	10.0

- 같은 행, 같은 열 위치에 있는 원소끼리 계산
- 한쪽에 원소가 존재하지 않거나 NaN이면 결과는 NaN
- 데이터프레임의 연산자 활용

  : DataFrame1 + 연산자(+, -, *, /) + DataFrame2

5. 데이터프레임 연산
- 그룹 객체 만들기

```
titanic=sns.load_dataset('titanic')

titanic_groupby = titanic.groupby('pclass').count()
titanic_groupby
```

pclass	survived	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	216	216	186	216	216	216	214	216	216	216	175	214	216	216
2	184	184	173	184	184	184	184	184	184	184	16	184	184	184
3	491	491	355	491	491	491	491	491	491	491	12	491	491	491

- groupby()로 'pclass'열을 기준으로 그룹화한 다음 count()함수로 개수 반환
- 분석 작업에 매우 많이 활용된다!
- 적용 가능한 판다스 기본 집계 함수

  :mean()/max()/min()/sum()/count()/var()/std() 등이 있다.

5. 데이터프레임 연산
- 그룹 연산 메소드

```
titanic_groupby = titanic.groupby('pclass')[['sex', 'age']].count()
titanic_groupby
```

pclass	sex	age
1	216	186
2	184	173
3	491	355

```
titanic.groupby('pclass')['age'].agg([max, min])
```

pclass	max	min
1	80.0	0.92
2	70.0	0.67
3	74.0	0.42

```
agg_format={'age':'max', 'sibsp':'sum', 'fare':'mean'}
titanic.groupby('pclass').agg(agg_format)
```

pclass	age	sibsp	fare
1	80.0	90	84.154687
2	70.0	74	20.662183
3	74.0	302	13.675550

- agg() 내에 함수를 리스트로 입력해 적용될 칼럼에 다양한 함수를 적용 가능

- agg() 내에 입력값으로 딕셔너리 형태로 적용될 칼럼과 aggregation 함수를 입력해 각각 다른 함수를 적용

5. 데이터프레임 연산
- group 객체.transform(매핑 함수)

“ 그룹별 데이터의 원소에 함수를 적용 후,
기존의 행 인덱스와 열 이름을 유지하는 상태로 연산 결과를 반환 ”

```
def minusmean(x):
    return x - x.mean()

grouped=titanic.groupby(['pclass'])
age_minusmean=grouped.age.transform(minusmean)
print(age_minusmean.loc[0:4])
```

```
0   -3.140620
1   -0.233441
2    0.859380
3   -3.233441
4    9.859380
Name: age, dtype: float64
```

- 원본 데이터프레임과 같은 형태로 기존 행 인덱스와 열 이름으로 반환
- ‘age’열에 대한 minusmean 함수 값만 출력하므로 시리즈 형태로 출력된다.

Q & A

수고하셨습니다!

^_____^