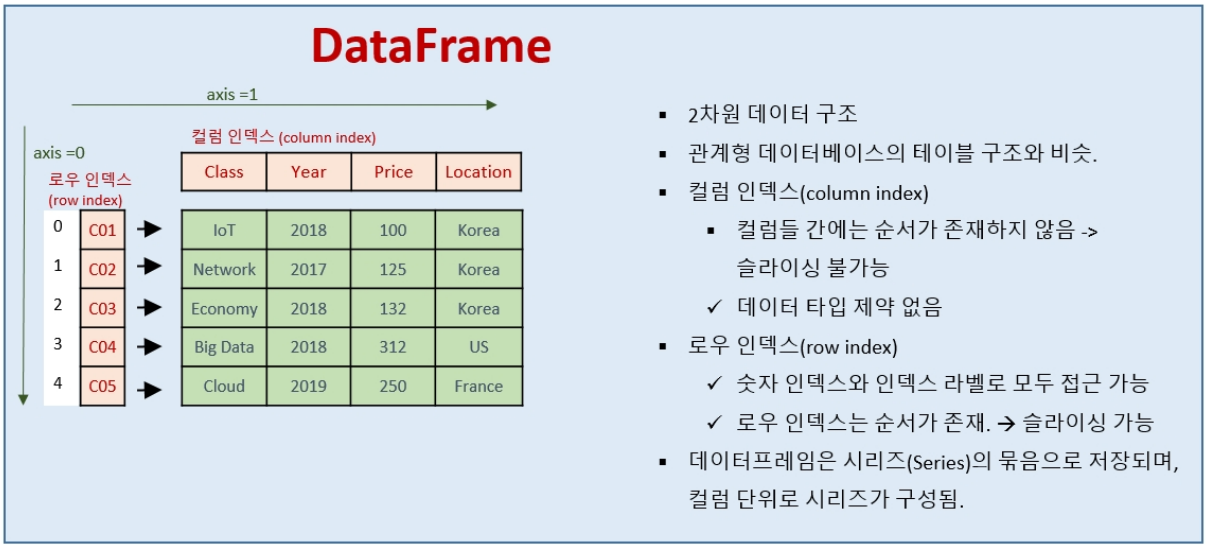
Pandas 주요 데이터 타입

DataFrame (데이터프레임)



In [1]:
```
import os

# 경로 확인
os.getcmd()

# 경로 변경
os.chdir('')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-1-1a41aabcb3f9> in <module>
      1 import os
      2
----> 3 os.getcmd()

AttributeError: module 'os' has no attribute 'getcmd'
```

In [2]:
```
# Pandas 라이브러리 임포트
import pandas as pd
```

2.2.1 데이터프레임 생성

생성자 인자

1) data : dataFrame에 저장할 데이터

2) index : 행(row) 이름, 기본값 = 0부터 1씩 증가하는 정수

3) columns : 열(column) 이름, 기본값 = 0부터 1씩 증가하는 정수

4) dtype : 데이터 형태(type), 만약 지정하지 않으면 파이썬이 자동으로 값의 형태를 보고 결정

5) copy : 입력 데이터를 복사할지 지정. 디폴트는 False 임. (복사할 거 아니면 메모리 관리 차원에서 디폴트인 False 설정 사용하면 됨)

In [5]:
```
# 1. 2차원 데이터 생성
data = pd.DataFrame([[1, 'kim', 26], [2, 'lee', 27]])
data
```

Out[5]:

	0	1	2
0	1	kim	26
1	2	lee	27

In [6]:
```
# 2. row index 설정 -> 0부터 2까지 1 간격으로 출력
data.index
```

Out[6]:
```
RangeIndex(start=0, stop=2, step=1)
```

In [7]:
```
# index 명칭 바꾸기
data.index = ['A', 'B']
data
```

Out[7]:

	0	1	2
A	1	kim	26
B	2	lee	27

In [8]:
```
# 3. column index 설정
data.columns = ['순서', '이름', '나이']
data
```

Out[8]:

	순서	이름	나이
A	1	kim	26
B	2	lee	27

In [9]:
```
# 1,2,3 한꺼번에 하기
pd.DataFrame(data = [[1, 'kim', 26], [2, 'lee', 27]],
             index = ['A', 'B'],
             columns = ['order', 'name', 'age'])
```

Out[9]:

	order	name	age
A	1	kim	26
B	2	lee	27

In [10]:
```
# 사전 타입 데이터를 이용하여 데이터 프레임 생성하기
population_dic = {  '서울': [1053.5, 1023, 987],
                    '경기': [1023, 1067, 1123],
                    '충청': [512, 489, 487],
                    '경상': [897, 872, 811],
                    '전라': [451, 421, 399]}
```

In [12]:
```
data2 = pd.DataFrame(population_dic)
data2
```

Out[12]:

	서울	경기	충청	경상	전라
0	1053.5	1023	512	897	451
1	1023.0	1067	489	872	421
2	987.0	1123	487	811	399

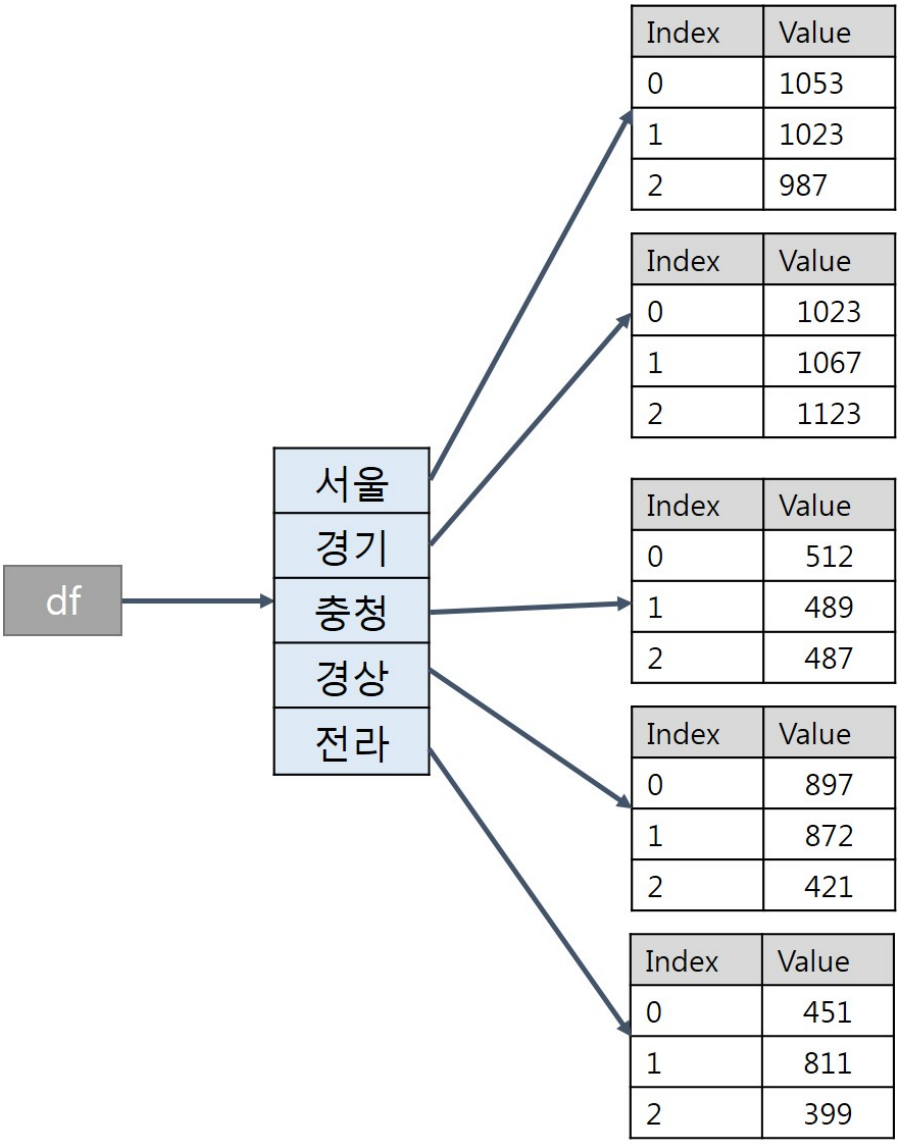
In [14]:
```
# row index를 2015, 2016, 2017로 만들기
data2.index = [x for x in range(2015, 2018)]
data2
```

Out[14]:

	서울	경기	충청	경상	전라
2015	1053.5	1023	512	897	451
2016	1023.0	1067	489	872	421
2017	987.0	1123	487	811	399

df →

DataFrame					
	Series('서울')	Series('경기')	Series('충청')	Series('경상')	Series('전라')
0	1053	1023	512	897	451
1	1023	1067	489	872	811
2	987	1123	487	421	399



DataFrame 속성 조회하기 (속성을 조회할 때에는 ()를 사용하지 않으니 유의하기 바람.)

In [15]:
```
#1. T (Transpose) : 행과 열을 바꾸기.
# R 언어는 소문자 t
data2.T
```

Out[15]:

	2015	2016	2017
서울	1053.5	1023.0	987.0
경기	1023.0	1067.0	1123.0
충청	512.0	489.0	487.0
경상	897.0	872.0	811.0
전라	451.0	421.0	399.0

In [16]:
```
#2. axes : 행과 열 이름을 리스트로 반환
data2.axes
```

Out[16]:
```
[Int64Index([2015, 2016, 2017], dtype='int64'),
 Index(['서울', '경기', '충청', '경상', '전라'], dtype='object')]
```

In [17]:
```
#3. dtypes : 데이터 형태 반환
data2.dtypes
```

Out[17]:
```
서울    float64
경기      int64
충청      int64
경상      int64
전라      int64
dtype: object
```

In [19]:
```
# 참고
# float64 -> int64 data type 변경
data2['서울'].astype('int64')
```

Out[19]:
```
2015    1053
2016    1023
2017     987
Name: 서울, dtype: int64
```

In [20]:
```
data2['서울'] = data2['서울'].astype('int64')
```

In [21]:
```
#4. shape : 행과 열의 개수(차원)을 튜플로 반환
# data2.shape = data2.shape[0] : 행
# data2.shape[1] : 열
data2.shape
```

Out[21]:
```
(3, 5)
```

In [23]:
```
def plus(a, b):
    result = a + b
    return result
```

In [24]:
```
plus(20, 30)
```

Out[24]:
```
50
```

In [25]:
```
def data_shape(data):
    row = data.shape[0]
    col = data.shape[1]
    print('현재 데이터는', row, '행', col, '열 입니다')
```

In [26]:
```
data_shape(data2)
```

```
현재 데이터는 3 행 5 열 입니다
```

In [27]:
```
#5. size : DataFrame의 원소의 개수를 반환
data2.size
```

Out[27]:
```
15
```

In [28]:
```
#6. index : 데이터프레임의 인덱스를 리스트로 반환
data.index
```

Out[28]:
```
Index(['A', 'B'], dtype='object')
```

In [29]:
```
#7. columns : 데이터프레임의 컬럼을 리스트로 반환
data2.columns
```

Out[29]:
```
Index(['서울', '경기', '충청', '경상', '전라'], dtype='object')
```

In [31]:
```
#8. values : 데이터프레임의 값들을 반환
data2.values
```

Out[31]:
```
array([[1053, 1023,  512,  897,  451],
       [1023, 1067,  489,  872,  421],
       [ 987, 1123,  487,  811,  399]])
```

2.2.3. 데이터프레임 조회하기

In [55]:
```
# 테스트 데이터프레임 생성
data3 = pd.DataFrame(  [['IoT', 2018, 100, 'Korea'],
                        ['Network', 2017, 125, 'Korea'],
                        ['Economy', 2018, 132, 'Korea'],
                        ['Big Data', 2018, 312, 'US'],
                        ['Cloud', 2019, 250, 'France']],
            index = ['C01', 'C02', 'C03', 'C04', 'C05'],
            columns = ['Class', 'Year', 'Price', 'Location'])
data3
```

Out[55]:

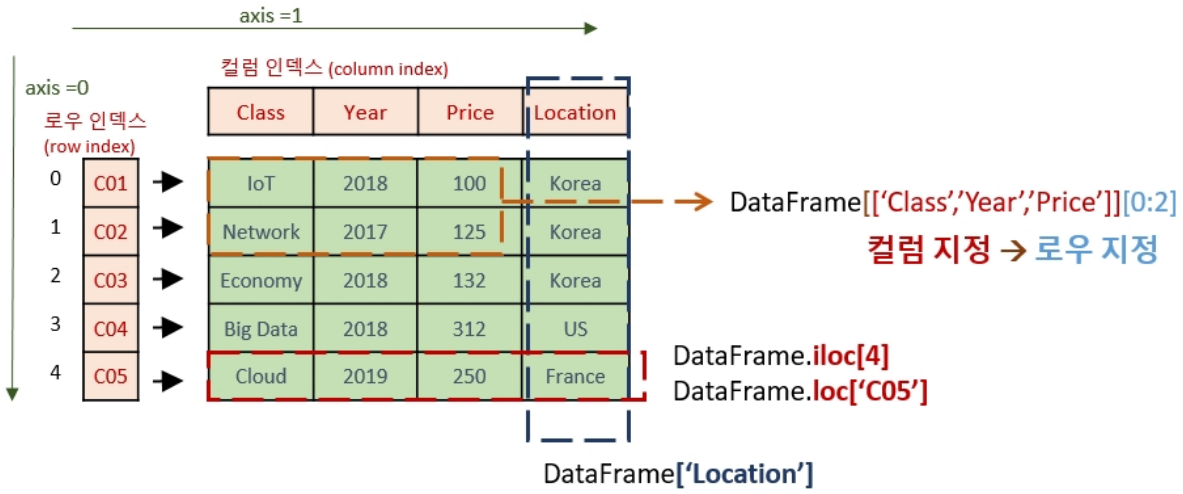
	Class	Year	Price	Location
C01	IoT	2018	100	Korea
C02	Network	2017	125	Korea
C03	Economy	2018	132	Korea
C04	Big Data	2018	312	US
C05	Cloud	2019	250	France

In [46]:
```
# 모범답안: dictionary 자료구조로 테스트 데이터프레임 생성
df = pd.DataFrame({'Class': ['Iot', 'Network', 'Economy', 'Big Data', 'Cloud'],
                   'Year': [2018, 2017, 2018, 2018, 2019],
                   'Price': [100, 125, 132, 312, 250],
                   'Location': ['Korea', 'Korea', 'Korea', 'US', 'France']},
                  index = ['C01', 'C02', 'C03', 'C04', 'C05'])
df
```

Out[46]:

	Class	Year	Price	Location
C01	Iot	2018	100	Korea
C02	Network	2017	125	Korea
C03	Economy	2018	132	Korea
C04	Big Data	2018	312	US
C05	Cloud	2019	250	France

데이터프레임 조회 규칙



data frame에서 바로 접근

1. 행

In [56]:
```
# 숫자 인덱스 하나를 쓰면 접근 x
df[2]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~/opt/anaconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in ge
t_loc(self, key, method, tolerance)
   2896             try:
-> 2897                 return self._engine.get_loc(key)
   2898             except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

KeyError: 2
```

In [57]:
```
# index slicing은 가능
df[:2]
```

Out[57]:

	Class	Year	Price	Location
C01	Iot	2018	100	Korea
C02	Network	2017	125	Korea

In [58]:
```
# index label 하나로 접근 x
df['C01']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~/opt/anaconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in ge
t_loc(self, key, method, tolerance)
   2896             try:
-> 2897                 return self._engine.get_loc(key)
   2898             except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

KeyError: 'C01'
```

In [59]:
```
# index label slicing은 가능
# 되도록이면 사용 안하는 것을 추천
df['C01':'C03']
```

Out[59]:

	Class	Year	Price	Location
C01	Iot	2018	100	Korea
C02	Network	2017	125	Korea
C03	Economy	2018	132	Korea

2. 열

In [ ]:
```
# 열로 접근하고 싶으면 column명을 직접 입력
# Price 정보만 조회하기
# 행은 안되고, 열은 된다.
df['Price']
```

In [60]:
```
# Class와 Year 컬럼만 조회
# 여러 개의 컬럼들을 조회하기 위해서는 컬럼명들을 리스트로 선언해야 함.
df[['Class', 'Year']]
```

Out[60]:

	Class	Year
C01	Iot	2018
C02	Network	2017
C03	Economy	2018
C04	Big Data	2018
C05	Cloud	2019

loc를 통합 접근

loc는 특정 name을 지정해줘야함

1. 행

In [61]:
```
# 행을 row로 접근하여 조회하기
# row index가 'C04'인 경우만 조회하기
df.loc['C01']
```

Out[61]:
```
Class         Iot
Year         2018
Price         100
Location    Korea
Name: C01, dtype: object
```

In [63]:
```
# row label로 slicing (start index와 end index 모두 포함)
# :로 접근
df.loc['C01':'C03']
```

Out[63]:

	Class	Year	Price	Location
C01	Iot	2018	100	Korea
C02	Network	2017	125	Korea
C03	Economy	2018	132	Korea

In [64]:
```
# C01, C03 추출
#loc를 생략하면 KeyError 발생
df.loc[['C01', 'C03']]
```

Out[64]:

	Class	Year	Price	Location
C01	Iot	2018	100	Korea
C03	Economy	2018	132	Korea

2. 열

In [67]:
```
# Price 정보만 조회하기
# x값(:): 모든 행, y값('Price'): 어떤 열
# 이 때 loc를 잘 쓰지는 않음
df.loc[:, 'Price']
```

Out[67]:
```
C01    100
C02    125
C03    132
C04    312
C05    250
Name: Price, dtype: int64
```

In [68]:
```
df['Price']
```

Out[68]:
```
C01    100
C02    125
C03    132
C04    312
C05    250
Name: Price, dtype: int64
```

In [69]:
```
# Class와 Year 컬럼만 조회
df.loc[:, ['Class', 'Year']]
```

Out[69]:

	Class	Year
C01	Iot	2018
C02	Network	2017
C03	Economy	2018
C04	Big Data	2018
C05	Cloud	2019

In [70]:
```
df[['Class', 'Year']]
```

Out[70]:

	Class	Year
C01	Iot	2018
C02	Network	2017
C03	Economy	2018
C04	Big Data	2018
C05	Cloud	2019

In [71]:
```
# 특정 컬럼과 로우를 동시에 인덱싱하기
# C02, C03 강의의 Class와 Year, Price 정보만 조회
# 문자일 경우 loc, 숫자일 경우 iloc로 접근
df.loc[['C02', 'C03'], ['Class', 'Year', 'Price']]
```

Out[71]:

	Class	Year	Price
C02	Network	2017	125
C03	Economy	2018	132

iloc를 통한 접근

i는 index를 의미하며 숫자로 통해 접근

1. 행

In [72]:
```
# 행을 숫자 index로 접근하여 조회하기 (iloc)
df.iloc[2]
```

Out[72]:
```
Class       Economy
Year           2018
Price           132
Location      Korea
Name: C03, dtype: object
```

In [73]:
```
# row index로 slicing (start index와 end index 모두 포함)
df.iloc[:2]
```

Out[73]:

	Class	Year	Price	Location
C01	Iot	2018	100	Korea
C02	Network	2017	125	Korea

2. 열

In [74]:
```
# Class와 Year, Price 정보만 조회
df.iloc[:, :3]
```

Out[74]:

	Class	Year	Price
C01	Iot	2018	100
C02	Network	2017	125
C03	Economy	2018	132
C04	Big Data	2018	312
C05	Cloud	2019	250

In [75]:
```
# C02, C03 강의의 Class와 Year, Price 정보만 조회
df.iloc[1:3, :3]
```

Out[75]:

	Class	Year	Price
C02	Network	2017	125
C03	Economy	2018	132

조건 indexing

In [76]:
```
# Price의 값이 200보다 큰 경우만 조회하기
df[df['Price'] > 200]

# 동일한 표현
df[df.Price > 200]
```

Out[76]:

	Class	Year	Price	Location
C04	Big Data	2018	312	US
C05	Cloud	2019	250	France

In [ ]:
```
# 아래 결과들이 출력되도록 코드를 완성하시오.
# 실습 1. 'Location' 컬럼만 조회
```

In [77]:
```
# 실습 1-1. 그냥 접근
# 가장 간단한 형태
df['Location']
```

Out[77]:
```
C01     Korea
C02     Korea
C03     Korea
C04        US
C05    France
Name: Location, dtype: object
```

In [96]:
```
# 실습 1-2. loc를 통해 접근
df.loc[:, 'Location']
```

Out[96]:
```
C01     Korea
C02     Korea
C03     Korea
C04        US
C05    France
Name: Location, dtype: object
```

In [80]:
```
# 실습 1-3. iloc를 통해 접근
df.iloc[:, 3]
```

Out[80]:
```
C01     Korea
C02     Korea
C03     Korea
C04        US
C05    France
Name: Location, dtype: object
```

In [ ]:
```
# 실습 2. 'Class'와 'Price' 컬럼만 조회
```

In [81]:
```
# 실습 2-1. 그냥 접근
df[['Class', 'Price']]
```

Out[81]:

	Class	Price
C01	Iot	100
C02	Network	125
C03	Economy	132
C04	Big Data	312
C05	Cloud	250

In [83]:
```
# 실습 2-2. loc를 통해 접근
df.loc[:, ['Class', 'Price']]
```

Out[83]:

	Class	Price
C01	Iot	100
C02	Network	125
C03	Economy	132
C04	Big Data	312
C05	Cloud	250

In [85]:
```
# 실습 2-3. iloc를 통해 접근
df.iloc[:, [0, 2]]
```

Out[85]:

	Class	Price
C01	Iot	100
C02	Network	125
C03	Economy	132
C04	Big Data	312
C05	Cloud	250

In [ ]:
```
# 실습 3. C01과 C03 강의의 모든 컬럼 조회
```

In [87]:
```
# 실습 3-1. loc를 통해 접근
df.loc[['C01', 'C03']]
```

Out[87]:

	Class	Year	Price	Location
C01	Iot	2018	100	Korea
C03	Economy	2018	132	Korea

In [88]:
```
# 실습 3-2. iloc를 통해 접근
df.iloc[[0, 2], :]
```

Out[88]:

	Class	Year	Price	Location
C01	Iot	2018	100	Korea
C03	Economy	2018	132	Korea

In [ ]:
```
# 실습 4. C01~C03 강의의 Class와 Price만 조회
```

In [98]:
```
# 실습 4-1. loc를 통해 접근
# 오류: df.loc[['C01':'C03'], ['Class', 'Price']]
df.loc['C01':'C03', ['Class', 'Price']]
```

Out[98]:

	Class	Price
C01	Iot	100
C02	Network	125
C03	Economy	132

In [99]:
```
# 실습 4-2. iloc를 통해 접근
df.iloc[:3, [0, 2]]
```

Out[99]:

	Class	Price
C01	Iot	100
C02	Network	125
C03	Economy	132

In [101]:
```
# 실습 5. Year가 2018이고, Price가 200 미만인 강의만 조회
df[(df.Year == 2018) & (df.Price < 200)]
```

Out[101]:

	Class	Year	Price	Location
C01	Iot	2018	100	Korea
C03	Economy	2018	132	Korea

데이터프레임에 새로운 컬럼 추가하기

In [102]:
```
# 컬럼 추가하기 1.
# numStudent 컬럼 추가하고, 값을 10으로 저장
df['numStudent'] = 10
df
```

Out[102]:

	Class	Year	Price	Location	numStudent
C01	Iot	2018	100	Korea	10
C02	Network	2017	125	Korea	10
C03	Economy	2018	132	Korea	10
C04	Big Data	2018	312	US	10
C05	Cloud	2019	250	France	10

In [112]:
```
# numStudent 의 값을 15, 30, 26, 32, 50으로 변경
df['numStudent'] = [15, 30, 26, 32, 50]
df
```

Out[112]:

	Class	Year	Price	Location	numStudent
C01	Iot	2018	100	Korea	15
C02	Network	2017	125	Korea	30
C03	Economy	2018	132	Korea	26
C04	Big Data	2018	312	US	32
C05	Cloud	2019	250	France	50

In [104]:
```
# 컬럼 추가하기 2 - 기존 컬럼을 이용하여 새 컬럼 추가하기
# Price과 numStudent의 값을 곱한 값을 'Income'라는 새 컬럼으로 추가
df['income'] = df['Price'] * df['numStudent']
df
```

Out[104]:

	Class	Year	Price	Location	numStudent	income
C01	Iot	2018	100	Korea	15	1500
C02	Network	2017	125	Korea	30	3750
C03	Economy	2018	132	Korea	26	3432
C04	Big Data	2018	312	US	32	9984
C05	Cloud	2019	250	France	50	12500

2.2.5.데이터프레임 로우, 컬럼 삭제

In [105]:
```
# drop() 함수 사용 (원본 변경 X)
## 첫번째 인자 - 지우고자 하는 인덱스명 (로우 인덱스, 컬럼 인덱스 모두 가능)
## 두번째 인자 (axis)- 0 혹은 1. 0 = 로우 삭제, 1 = 컬럼 삭제
df.drop(['numStudent'], axis = 1)
```

Out[105]:

	Class	Year	Price	Location	income
C01	Iot	2018	100	Korea	1500
C02	Network	2017	125	Korea	3750
C03	Economy	2018	132	Korea	3432
C04	Big Data	2018	312	US	9984
C05	Cloud	2019	250	France	12500

In [ ]:
```
# 'Income' 컬럼 삭제
df.drop(['income'], axis = 1)
```

In [106]:
```
# 원본 변경을 위해서는 다시 변수에 할당해야 함.
df = df.drop(['income'], axis = 1)
df
```

Out[106]:

	Class	Year	Price	Location	numStudent
C01	Iot	2018	100	Korea	15
C02	Network	2017	125	Korea	30
C03	Economy	2018	132	Korea	26
C04	Big Data	2018	312	US	32
C05	Cloud	2019	250	France	50

In [113]:
```
# inplace=True option 부여
df.drop(['numStudent'], axis = 1, inplace = True)
df
```

Out[113]:

	Class	Year	Price	Location
C01	Iot	2018	100	Korea
C02	Network	2017	125	Korea
C03	Economy	2018	132	Korea
C04	Big Data	2018	312	US
C05	Cloud	2019	250	France

In [114]:
```
# 여러 index를 한 번에 지우기 위해서는 list로 column name을 인자로 넘김
df.drop(['Class', 'Year'], axis = 1, inplace = True)
df
```

Out[114]:

	Price	Location
C01	100	Korea
C02	125	Korea
C03	132	Korea
C04	312	US
C05	250	France