데이터분석을위한 PANDAS

ABOUT PANDAS

PANDAS 살펴보기

◆ 소개

- 계량 경제학 용어인 '**PAN**el **DA**ta' 앞 글자
- **금융 데이터에 대한 계량적 분석 위해 2008년부터 개발 작업 시작**
- **BSD 라이선스 : 소스코드 공개 의무 없음. 상업적 사용 가능**
- **https://pandas.pydata.org**

PANDAS 살펴보기

◆ 설치

- **CONDA 기반**

  conda install -c conda-forge pandas

  conda install pandas

- **CONDA 저장소**

  conda-forge : 커뮤니티로 최신 버전 많음

  default : 안전성 중심. 최신 버전 없음

PANDAS 살펴보기

◆ 설치

- **가상환경 생성 – Anaconda S/W**

  (1) Anaconda Powrshell 관리자모드로 실행

  (2) 현재 가상환경 리스트

```
(base) PS C:\Users\kwon> conda env list

# conda environments:
#
                      C:\ProgramData\anaconda3
base                * C:\Users\kwon\anaconda3
PY311                 C:\Users\kwon\anaconda3\envs\PY311
```

PANDAS 살펴보기

◆ 설치

- **가상환경 생성 – Anaconda S/W**

(3) Anaconda Powrshell 관리자모드로 실행

```
(base) PS C:\Users\kwon> conda create -n DATA_311 python=3.11
```

(4) 가상환경 리스트에서 생성된 가상환경 확인

```
(base) PS C:\Users\kwon> conda env list

# conda environments:
#
                      C:\ProgramData\anaconda3
base                * C:\Users\kwon\anaconda3
DATA_311              C:\Users\kwon\anaconda3\envs\DATA_311
PY311                 C:\Users\kwon\anaconda3\envs\PY311
```

PANDAS 살펴보기

◆ 설치

- **가상환경 생성 – Anaconda S/W**

  (5) 새로운 가상환경 실행

```
(base) PS C:\Users\kwon> conda activate DATY_311
```

```
(DATA_311) PS C:\Users\kwon>
```

PANDAS 살펴보기

◆ 설치

- **패키지 설치 – Anaconda S/W**

(6) pandas 설치

```
(DATA_311) PS C:\Users\kwon> conda install pandas
```

(7) 설치 확인 – python 모드에서 체크

```
(DATA_311) PS C:\Users\kwon> python
Python 3.11.14 | packaged by conda-forge | (main, Oct 13 2025, 14:00:26)
[MSC v.1944 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas
>>> exit( )
(DATA_311) PS C:\Users\kwon>
```

python 명령어 모드

PANDAS 살펴보기

◆ 설치

- **패키지 삭제 후 다시 설치 – Anaconda S/W**

(1) pandas 삭제

```
(DATA_311) PS C:\Users\kwon> conda  uninstall  pandas
## Package Plan ##
  environment location: C:\Users\kwon\anaconda3\envs\DATA_311
  removed specs:
    - pandas
```

(2) 다시 설치

```
(DATA_311) PS C:\Users\kwon> conda install pandas
```

PANDAS 살펴보기

◆ 의존성 패키지들

- Pandas 동작에 연관된 패키지들
- 패키지별 가상환경 필요한 이유

  ➔ 다른 패키지들과 중복되는 경우 버전 충돌 발생 원인
- **반드시 설치**되어야 함

  → NumPy : 다차원 배열 처리 패키지

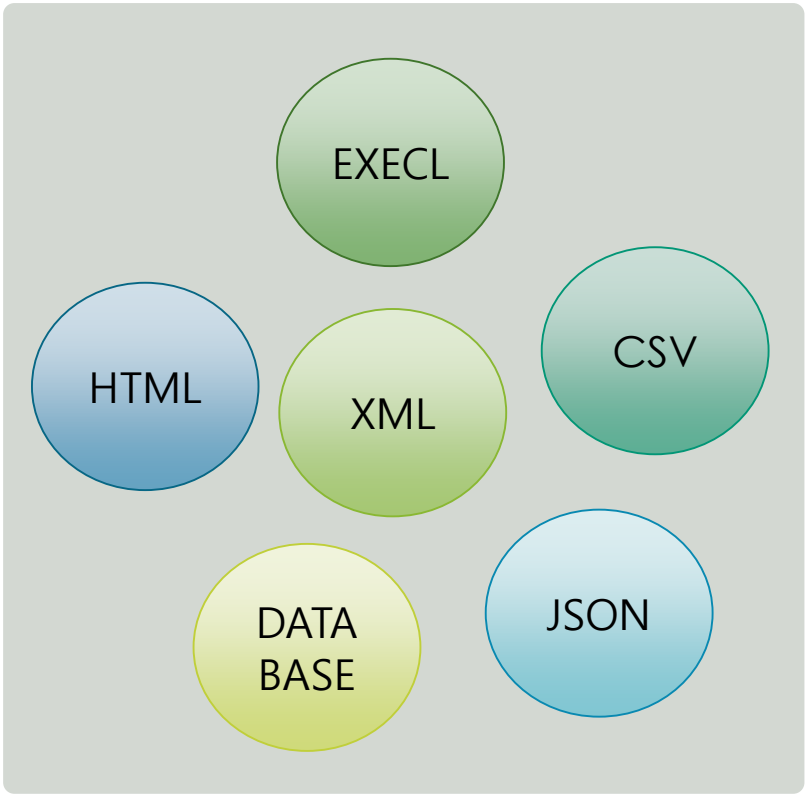
  → python-dateutil : date관련 패키지

  → pytz : 세계 시간 Timezone 패키지

PANDAS 살펴보기

◆ Pandas 자료구조

[다양한 형태 데이터]



변 환

동일형태
동일포맷

번호

번호	이름	학교

PANDAS 살펴보기

◆ 자료구조

- **시리즈 (Series)**

  데이터를 1차원 배열에 저장하는 구조화 데이터

9	홍길동	대구

번호
9
11
21

- **데이터프레임 (DataFrame)**

  데이터를 2차원 배열에 저장하는 구조화 데이터

번호	이름	학교
9	홍길동	대구
11	마징가	부산
21	베트맨	부산

PANDAS 살펴보기

◆ 자료구조

PANDAS 살펴보기

◆ Pandas 데이터 타입 - dType

Pandas	Python	NumPy	사용
object	str or mixed	string, unicode, mixed type	Text or mixed numeric non-numeric values
int64	int	int, int8, int16, int32, int64, uint8, uint16, uint32, uint64	Integer numbers
float64	float	float, float16, float32, float64	Floating point numbers
bool	bool	bool	True/False values
datetime64	NA	datetime64[ns]	Date and time values
timedelta[ns]	NA	NA	Differences between two datetimes
category	NA	NA	Finitelist of text values

PANDAS PROGRAMMING

구조화 데이터

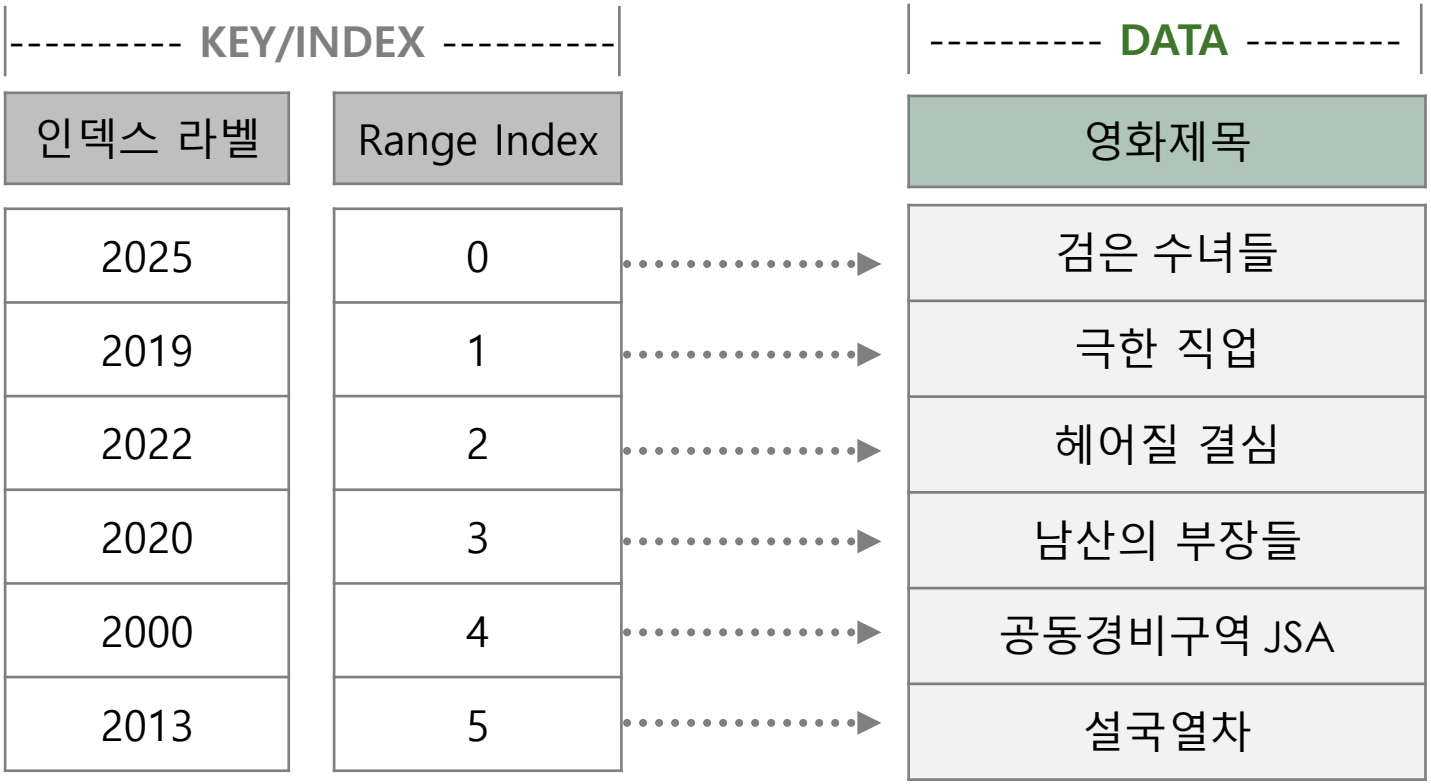
◆ 시리즈(Series)

- 순차적으로 나열된 1차원 배열 형태
- 키(k)와 값(v)로 구성된 {K, V} 딕셔너리와 비슷

- 형식 : Index + Data

구조화 데이터

◆ 시리즈(Series)



구조화 데이터

◆ 시리즈(Series)

- **인스턴스/객체 생성**

```
pandas.Series( data=None,     → 데이터
               index=None,    → 데이터 식별 인덱스
               dtype=None,    → 데이터 타입/자료형
               name=None,     → 메타정보. 시리즈 이름
               copy=None )    → 독립 메모리 저장 여부
```

구조화 데이터

◆ 시리즈(Series)

▪ 속성들 – 기본 속성

분 류	속성명	설 명
라벨	index	• 시리즈의 인덱스(pd.Index) 객체
	name	• 시리즈 이름(메타데이터, DataFrame 변환 시 열 이름으로 사용)
	axes	• 축의 리스트([index] 반환)
	T	• 전치(Transpose). Series는 1차원이므로 원본과 동일
타입 및 메모리	dtype	• 데이터 타입(int64, float64, string, category, datetime64[ns] 등)
	array	• 내부 데이터를 담은 ExtensionArray 객체
	values	• 넘파이 배열 또는 ExtensionArray 뷰 (to_numpy() 권장)
	nbytes	• 데이터(값) 부분이 차지하는 메모리 바이트 수
	attrs	• 사용자가 임의로 부착 가능한 메타데이터 딕셔너리

◆ 시리즈(Series)

▪ 속성들 – 기본 속성

분 류	속성명	설 명
형태 및 크기	shape	• (n,) 형태의 크기 튜플
	size	• 원소의 개수
	ndim	• 차원 수 (항상 1)
	empty	• 비어 있으면 True
인덱싱 및 접근자	loc	• 라벨 기반 인덱서
	iloc	• 위치 기반 인덱서
	at	• 단일 라벨 원소 접근(빠르고 엄격)
	iat	• 단일 위치 원소 접근(빠르고 엄격)

구조화 데이터

◆ 시리즈(Series)

딕셔너리 ➔ 시리즈

```
##- 모듈로딩
import pandas as pd

##- 데이터
dict_data = {'a': 1, 'b': 2, 'c': 3}

##- 딕셔너리(dict_data) 데이터 => 시리즈로 변환
sr = pd.Series(dict_data)

##- 시리즈 타입 출력
print(type(sr))

##- 시리즈 데이터 출력
print(sr)
```

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

구조화 데이터

◆ 시리즈(Series)

▪ 속성 활용 / 값 출력

속성 읽기 => Series객체.속성명

속성 변경 => Series객체.속성명 = 새로운값

[예시]

```
sr = pd.Series([10, 20, 30])

print( "인덱스", sr.index )
print( "데이터", sr.values )
```

구조화 데이터

◆ 시리즈(Series)

▪ 원소/요소 선택

Series객체[정수인덱스] 또는 **Series객체[인덱스라벨]**

[예시]

```
sr = pd.Series([10, 20, 30])
print( "0번 원소 : ", sr[0]  )

sr = pd.Series([10, 20, 30], index=['n01', 'n02', 'n03'])
print( "n01 원소 : ", sr["n01"] )
```

구조화 데이터

◆ 시리즈(Series)

- 여러 개 원소/요소 선택

[방법1] 리스트 기반

Series객체[[정수인덱스, 정수인덱스, ..., 정수인덱스]]

Series객체[[인덱스라벨, 인덱스라벡, ... , 인덱스라벨]]

[방법2] 슬라이싱 기반

Series객체[시작인덱스 : 끝인덱스: 간격]

구조화 데이터

◆ 시리즈(Series)

리스트 ➔ 시리즈

```
##- 모듈로딩
import pandas as pd

##- 데이터
list_data = ['2019-01-02', 3.14, 'ABC', 100, True]

##- 리스트 데이터 => 시리즈로 변환
sr = pd.Series(list_data)

##- 시리즈 타입 출력
print(f'sr 타입 : {type(sr)}')

##- 시리즈 속성 출력
print(f'sr 인스턴스===₩n{sr}')
print(f'series => {sr}')
print(f'index  => {sr.index}')
print(f'values => {sr.values}')
```

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

구조화 데이터

◆ 시리즈(Series)

튜플 ➔ 시리즈

```
##- 모듈로딩
import pandas as pd

##- 데이터
tup_data = ('영인', '2010-05-01', '여', True)

##- 투플 => 시리즈로 변환(index 옵션에 인덱스 이름 지정)
sr = pd.Series(tup_data, index=['이름', '생년월일', '성별', '학생여부'])

##- 타입 출력
print(f'sr 타입 : {type(sr)}')

##- 속성 출력
print(f'sr 인스턴스===₩n{sr}')
print(f'series => {sr}')
print(f'index  => {sr.index}')
print(f'values => {sr.values}')
```

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

구조화 데이터

◆ 시리즈(Series)

튜플 ➔ 시리즈

```
## ----------------------------------------
## 원소 선택
## ----------------------------------------
# 원소를 1개 선택
print(sr[0])        ## sr의 1 번째 원소 선택 (정수형 위치 인덱스 활용)
print(sr['이름'])   ## '이름' 라벨 가진 원소 선택 (인덱스 이름 활용)
print('\n')

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

구조화 데이터

◆ 시리즈(Series)

튜플 ➔ 시리즈

```
## ------------------------------------
## 여러 개의 원소를 선택 (인덱스 범위 지정)
## ------------------------------------
## 시작인덱스 : 끝인덱스, 끝인덱스 미포함
print( sr[ 1 : 2 ] )
print('₩n')

## 시작인덱스명: 끝인덱스명 , 모두포함
print( sr[  '생년월일' : '성별'  ] )
print( sr[  ['생년월일', '성별']  ] )
print('₩n')

# 여러 개의 원소를 선택 (인덱스 범위 지정)
## 시작인덱스 : 끝인덱스, 끝인덱스 미포함
print( sr[ 1 : 2 ] )

## 시작인덱스명: 끝인덱스명, 모두포함
print( sr[  '생년월일' : '성별'  ] )
print('₩n')
```

구조화 데이터

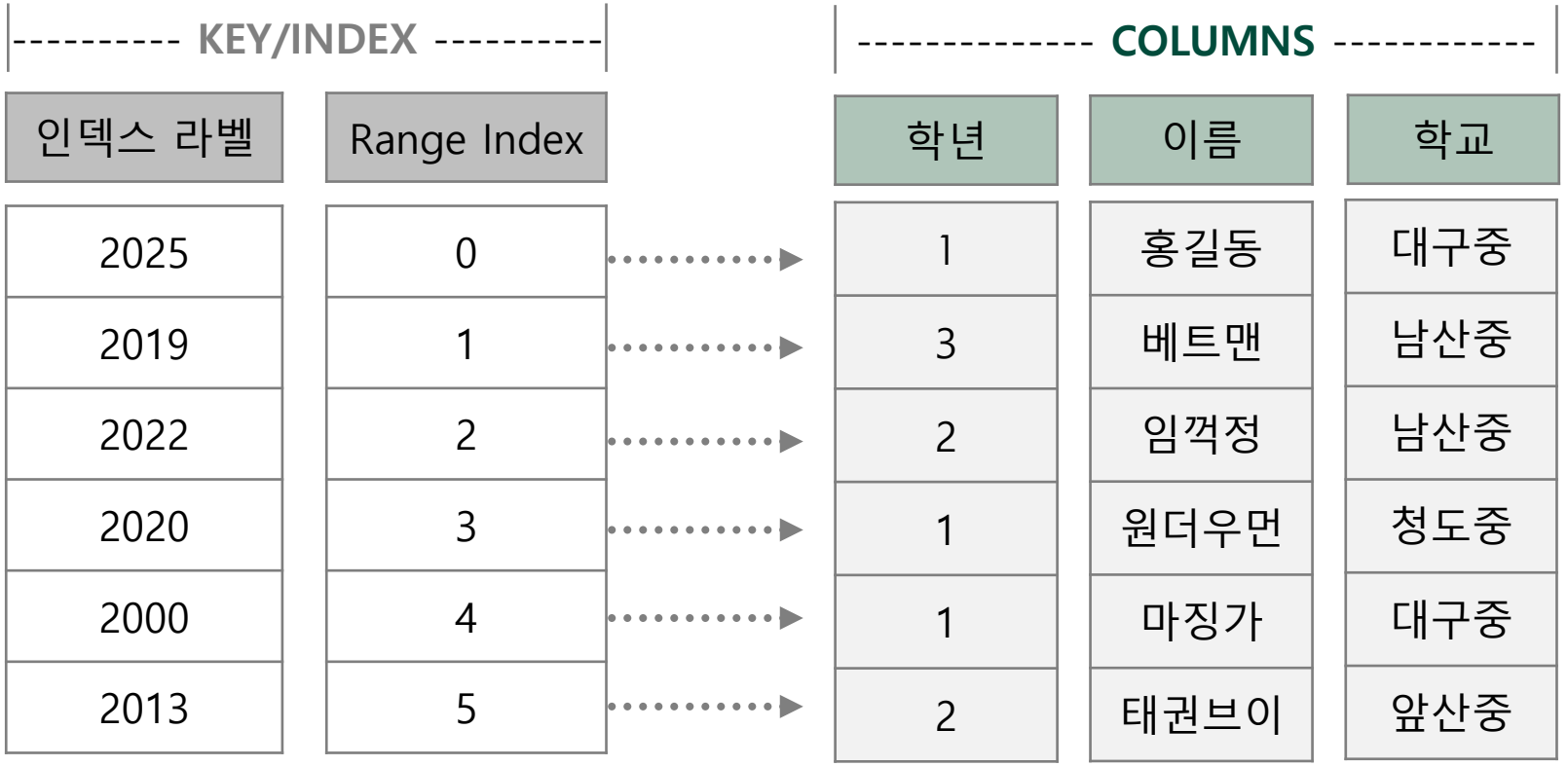
◆ 데이터프레임(DataFrame)

- **행과 열(Series)**로 만들어진 **2차원 배열 구조**
- 통계 패키지 R의 데이터프레임에서 유래
- **여러 개의 시리즈(Series) 묶음**

- 형식 : Index + Columns + Datas

구조화 데이터

◆ 데이터프레임(DataFrame)



구조화 데이터

◆ 데이터프레임(DataFrame)

- **인스턴스/객체 생성**

```
pandas.DataFrame( data=None,      → 데이터
                  index=None,     → 데이터 식별 인덱스
                  columns=None,   → 컬럼 이름
                  dtype=None,     → 데이터 타입/자료형
                  copy=None )     → 독립 메모리 저장 여부
```

구조화 데이터

◆ 데이터프레임(DataFrame)

▪ 속성들 – 기본 속성

분 류	속성명	설 명
라벨	index	• 행 인덱스(pd.Index 객체)
	columns	• 열 인덱스(pd.Index 객체)
	axes	• [index, columns] 리스트 반환
	T	• 전치(Transpose). 행·열 교환된 DataFrame 반환
타입 및 메모리	dtypes	• 각 열의 데이터 타입(dtype) 시리즈
	array	• 내부 데이터를 담은 ExtensionArray 뷰 (Series 단위와 유사)
	values	• DataFrame 전체를 넘파이 배열 또는 ExtensionArray 형태로 반환
	nbytes	• 데이터(값) 부분이 차지하는 메모리 바이트 수
	attrs	• 사용자가 임의로 부착 가능한 메타데이터 딕셔너리

구조화 데이터

◆ 데이터프레임(DataFrame)

▪ 속성들 – 기본 속성

분 류	속성명	설 명
형태 및 크기	shape	• (행, 열) 형태의 크기 튜플
	size	• 전체 원소 수 (행 × 열)
	ndim	• 차원 수 (항상 2)
	empty	• 비어 있으면 True
인덱싱 및 접근자	loc	• 라벨 기반 인덱서 — 행·열 이름으로 접근
	iloc	• 위치 기반 인덱서 — 행·열 번호로 접근
	at	• 단일 라벨 원소 접근(빠르고 엄격)
	iat	• 단일 위치 원소 접근(빠르고 엄격)

구조화 데이터

◆ 데이터프레임(DataFrame)

딕셔너리 ➔ 데이터프레임

```
##- 모듈로딩
import pandas as pd

##- 열이름 key, 리스트 value로 갖는 딕셔너리 데이터
##- 2차원 배열
dict_data = {'c0':[1,2,3],  'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}

##- 딕셔너리 => 데이터프레임으로 변환
df = pd.DataFrame(dict_data)

##- 데이터프레임 속성 출력
print(f'df 인스턴스===₩n{df}')
print(f'df 타   입 : {type(df)}')
```

구조화 데이터

◆ 데이터프레임(DataFrame)

▪ 속성 활용 / 값 출력

속성 읽기 => DataFrame객체.속성명

속성 변경 => DataFrame객체.속성명 = 새로운값

구조화 데이터

◆ 데이터프레임(DataFrame)

2차원 리스트 ➔ 데이터프레임

```
##- 모듈로딩
import pandas as pd

##- 2차원 리스트 => 데이터프레임으로 변환
df = pd.DataFrame( [ [15, '남', '덕영중'],
                     [17, '여', '수리중'] ],
                     index=['준서', '예은'],
                     columns=['나이', '성별', '학교'] )

##- 데이터프레임 속성 출력
print(f'df 인스턴스===₩n{df}')
print(f'df 타   입 : {type(df)}')
```

	나이	성별	학교
준서	15	남	덕영중
예은	17	여	수리중

구조화 데이터

◆ 데이터프레임(DataFrame)

차원 리스트 ➔ 데이터프레임

```
# 행 인덱스, 열 이름 변경하기
df.index=['학생1', '학생2']
df.columns=['연령', '남녀', '소속']


print(df)              #데이터프레임
print(df.index)        #행 인덱스
print(df.columns)      #열 이름
```

구조화 데이터

◆ 데이터프레임(DataFrame)

▪ 인덱스 & 열이름 제어 함수

```
DataFrame객체.rename( index={ 기존인덱스: 새 인덱스, ..... },
                      inplace )
```

```
DataFrame객체.rename( columns={ 기존인덱스: 새 인덱스, ..... },
                      inplace)
```

구조화 데이터

◆ 데이터프레임(DataFrame)

행인덱스, 열이름 변경

```
# 열 이름 중, '나이'를 '연령', '성별'을 '남녀', '학교'를 '소속'으로
df2=df.rename(columns={'나이':'연령', '성별':'남녀', '학교':'소속'})

# df의 행 인덱스 중에서, '준서'를 '학생1', '예은'을 '학생2'로 바꾸기
df3=df.rename(index={'준서':'학생1', '예은':'학생2' }, inplace=True)

# df 출력(변경 후)
print(f"df======{df3}₩n{df}")
print(f"df2======₩n{df2}")
```

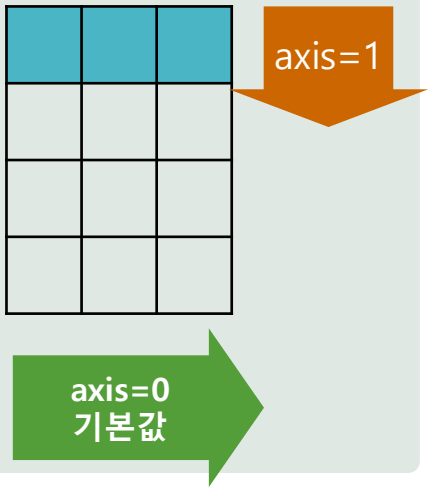
구조화 데이터

◆ 데이터프레임(DataFrame)

▪ 행 / 열 삭제

DataFrame객체.drop(행인덱스 또는 배열, axis=0)

DataFrame객체.drop(열이름 또는 배열 , axis=1)



** 기존 객체 변경하지 않고 새로운 객체 반환

inplace=True 옵션 : 기존 객체 데이터 삭제

구조화 데이터

◆ 데이터프레임(DataFrame)

행 삭제

```
# 모듈 로딩
import pandas as pd

# 데이터프레임 형식으로 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'])

# df 복제 => 변수 df2에 저장. df2의 1개 행(row) 삭제
df2 = df[:]
df2.drop('우현', inplace=True)

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

구조화 데이터

◆ 데이터프레임(DataFrame)

열삭제

```
# df 복제 => 변수 df4에 저장. df4의 1개 열(column) 삭제
df4 = df[:]
df4.drop('수학', axis=1, inplace=True)

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