데이터 분석을 위한 라이브러리

03 데이터 조작 및 분석을 위한 Pandas 기본

/* elice */

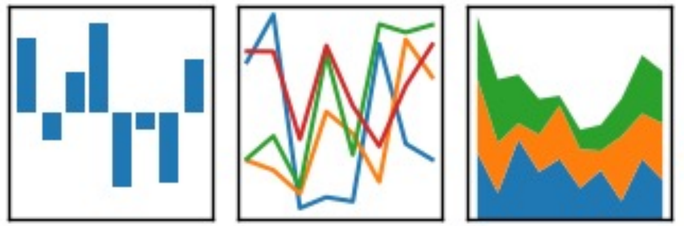
01

Series 데이터

Pandas?

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



: 파이썬 라이브러리

: 구조화된 데이터를 효과적으로 처리하고 저장

: Array 계산에 특화된 NumPy를 기반으로 설계

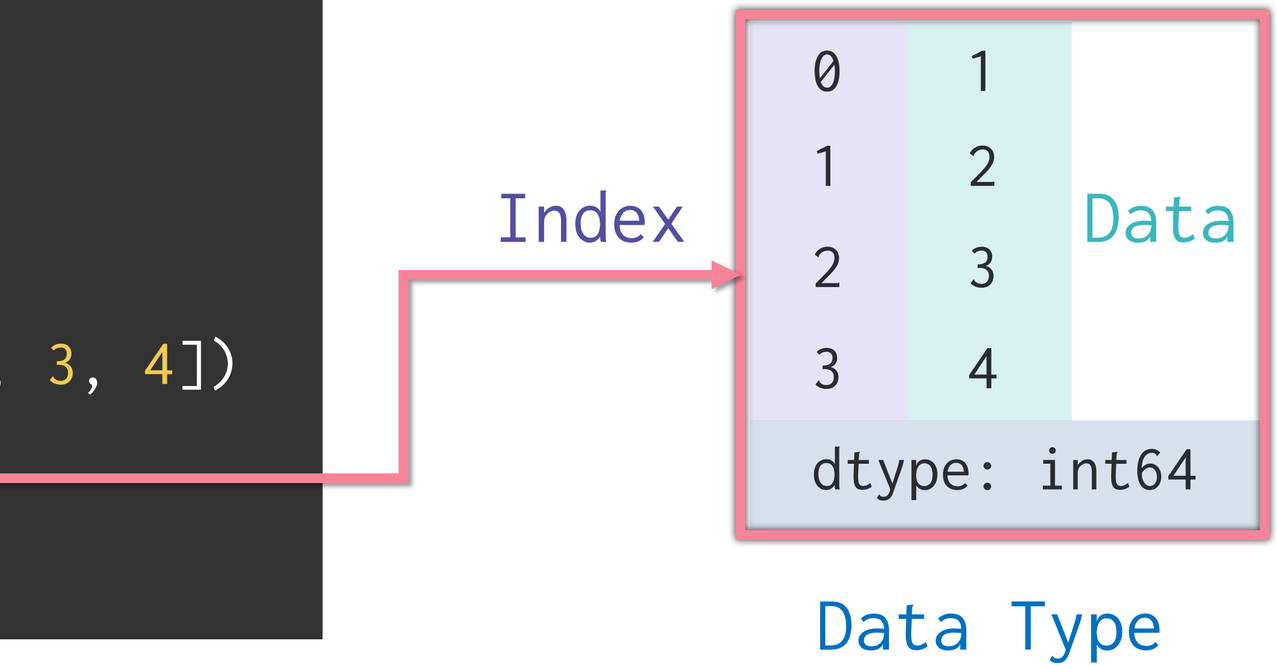
Series

Numpy의 array가 보강된 형태

Data와 **Index**를 가지고 있음

```
import pandas as pd


data = pd.Series([1, 2, 3, 4])
print(data)
```



Series

Series는 값(values)을 ndarray형태로 가지고 있음

```
import pandas as pd

data = pd.Series([1, 2, 3, 4])

print(data)
#  0    1
#  1    2
#  2    3
#  3    4
#  dtype: int64
```

```
print(type(data))
# <class 'pandas.core.series.Series'>

print(data.values)
# [1 2 3 4]

print(type(data.values))
# <class 'numpy.ndarray'>
```

Series

dtype 인자로 데이터 타입을 지정할 수 있음

```
data = pd.Series([1, 2, 3, 4], dtype = "float")
print(data.dtype)      # float64
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

data

0	1
1	2
2	3
3	4

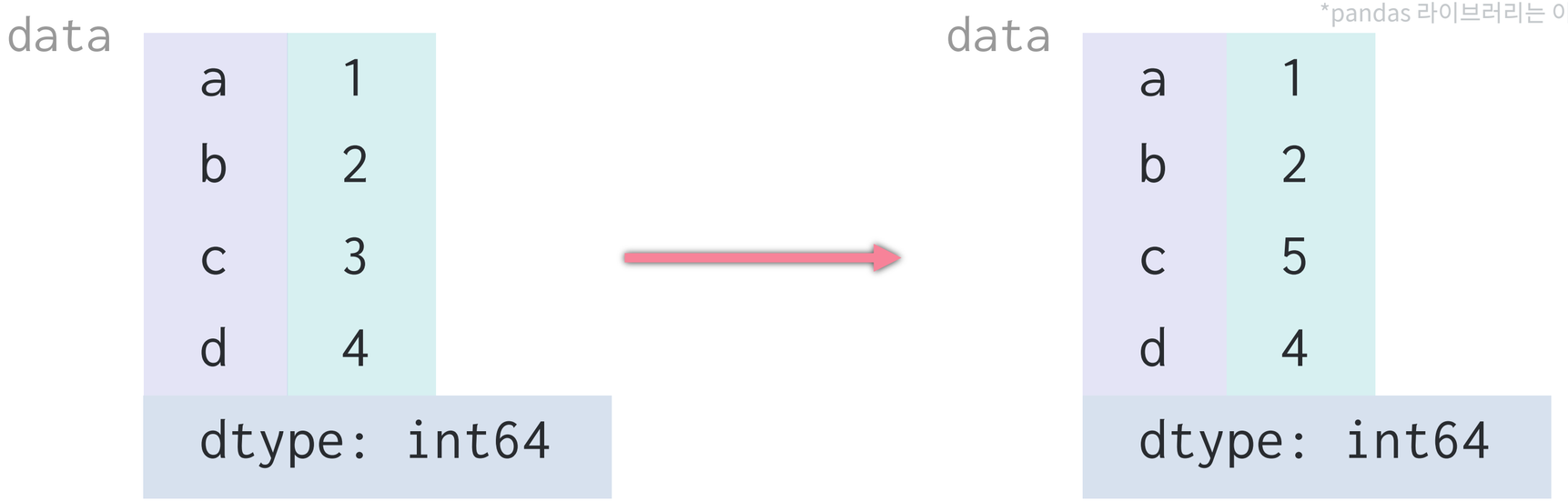
dtype: float64

Series

인덱스를 지정할 수 있고 인덱스로 접근 가능

```
data = pd.Series([1, 2, 3, 4], index = ['a', 'b', 'c', 'd'])
data['c'] = 5    # 인덱스로 접근하여 요소 변경 가능
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정



Series

Dictionary를 활용하여 Series 생성 가능

```
population_dict = {
    'china': 141500,
    'japan': 12718,
    'korea': 5180,
    'usa': 32676
}
population = pd.Series(population_dict)
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

```
population
china     141500
japan      12718
korea       5180
usa        32676
dtype: int64
```

02

데이터프레임

DataFrame

여러 개의 Series가 모여서 행과 열을 이룬 데이터

```
…
gdp_dict = {
    'china': 1409250000,
    'japan': 516700000,
    'korea': 169320000,
    'usa': 2041280000,
}
gdp = pd.Series(gdp_dict)
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

```
country = pd.DataFrame({
    'gdp': gdp,
    'population': population
})
```

country	gdp	population
china	1409250000	141500
japan	516700000	12718
korea	169320000	5180
usa	2041280000	32676

DataFrame

Dictionary를 활용하여 DataFrame 생성 가능

```
data = {
    'country': ['china', 'japan', 'korea', 'usa'],
    'gdp': [1409250000, 516700000, 169320000, 2041280000],
    'population': [141500, 12718, 5180, 32676]
}
country = pd.DataFrame(data)
country = country.set_index('country')
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

country

country	gdp	population
china	1409250000	141500
japan	516700000	12718
korea	169320000	5180
usa	2041280000	32676

DataFrame – 속성

DataFrame 속성을 확인하는 방법

```
…
print(country.shape)    # (4, 2)
print(country.size)     # 8
print(country.ndim)     # 2
print(country.values)   # [[1409250000     141500]
                        #  [ 516700000      12718]
                        #  [ 169320000       5180]
                        #  [2041280000      32676]]
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

country

country	gdp	population
china	1409250000	141500
japan	516700000	12718
korea	169320000	5180
usa	2041280000	32676

DataFrame – index, columns 이름 지정

DataFrame의 index와 column에 이름 지정

```
country.index.name = "Country"  # 인덱스에 이름 지정
country.columns.name = "Info"   # 컬럼에 이름 지정


print(country.index)
# Index(['china', 'japan', 'korea', 'usa'], dtype='object', name='Country’)
print(country.columns)
# Index(['gdp', 'population'], dtype='object', name='Info')
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

DataFrame – 저장 & 로드

데이터 프레임 **저장** 및 **불러오기** 가능

```
country.to_csv("./country.csv")
country.to_excel("country.xlsx")

country = pd.read_csv("./country.csv")
country = pd.read_excel("country.xlsx")
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

country.csv/country.xlsx

	A	B	C
1	Country	gdp	population
2	china	1409250000	141500
3	japan	516700000	12718
4	korea	169320000	5180
5	usa	2041280000	32676

03

데이터 선택 및 변경하기

데이터 선택 – Indexing/Slicing

.loc : 명시적인 인덱스를 참조하는 인덱싱/슬라이싱

```
country.loc['china']    # 인덱싱
country.loc['japan':'korea', :'population']   # 슬라이싱
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

country

	gdp	population
china	1409250000	141500
japan	516700000	12718
korea	169320000	5180
usa	2041280000	32676

인덱싱 결과값

gdp	1.409250e+09
population	1.415000e+05

Name: china, dtype: float64

슬라이싱 결과값

	gdp	population
japan	516700000	12718
korea	169320000	5180

데이터 선택 – Indexing/Slicing

.iloc : 파이썬 스타일의 정수 인덱스 인덱싱/슬라이싱

```
country.iloc[0]   # 인덱싱
country.iloc[1:3, :2]  # 슬라이싱
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

country

	gdp	population
china	1409250000	141500
japan	516700000	12718
korea	169320000	5180
usa	2041280000	32676

인덱싱 결과값

gdp	1.409250e+09
population	1.415000e+05

Name: china, dtype: float64

슬라이싱 결과값

	gdp	population
japan	516700000	12718
korea	169320000	5180

데이터 선택 – 컬럼 선택

컬럼명 활용하여 DataFrame에서 데이터 선택 가능

```
country
country['gdp']
country[['gdp']]
```

```
china    1409250000
japan     516700000
korea     169320000
usa      2041280000
Name: gdp, dtype: int64
```

= Series

country

	gdp	population
china	1409250000	141500
japan	516700000	12718
korea	169320000	5180
usa	2041280000	32676

	gdp
china	1409250000
japan	516700000
korea	169320000
usa	2041280000

= DataFrame

데이터 선택 – 조건 활용

Masking 연산이나 **query 함수**를 활용하여
조건에 맞는 DataFrame 행 추출 가능

```
country[country['population'] < 10000]   # masking 연산 활용
country.query("population > 100000")     # query 함수 활용
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

country

	gdp	population
china	1409250000	141500
japan	516700000	12718
korea	169320000	5180
usa	2041280000	32676

	gdp	population
korea	169320000	5180

masking 연산 활용

	gdp	population
china	1409250000	141500

query 함수 활용

데이터 변경 – 컬럼 추가

Series도 numpy array처럼 연산자 활용 가능

```
gdp_per_capita = country['gdp'] / country['population']
country['gdp per capita'] = gdp_per_capita
```

gdp_per_capita

china	9959.363958
japan	40627.457147
korea	32687.258687
usa	62470.314604

dtype: float64

*pandas 라이브러리는 이미 import 해둔 것으로 가정

country

	gdp	population	gdp per capita
china	1409250000	141500	9959.363958
japan	516700000	12718	40627.457147
korea	169320000	5180	32687.258687
usa	2041280000	32676	62470.314604

데이터 변경 – 데이터 추가/수정

리스트로 추가 or 딕셔너리로 추가

```
df = pd.DataFrame(columns = ['이름','나이','주소'])   # 데이터프레임 생성
df.loc[0] = ['길동', '26', '서울']       # 리스트로 데이터 추가
df.loc[1] = {'이름':'철수', '나이':'25', '주소':'인천'}  # 딕셔너리로 데이터 추가
df.loc[1, '이름'] = '영희'    # 명시적 인덱스 활용하여 데이터 수정
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

데이터프레임 생성

	이름	나이	주소

데이터 추가 결과

	이름	나이	주소
0	길동	26	서울
1	철수	25	인천

데이터 수정 결과

	이름	나이	주소
0	길동	26	서울
1	영희	25	인천

데이터 변경 – NaN 컬럼 추가

NaN값으로 초기화 한 새로운 컬럼 추가

```
df['전화번호'] = np.nan      # 새로운 컬럼 추가 후 초기화
df.loc[0, '전화번호'] = '01012341234'     # 명시적 인덱스 활용하여 데이터 수정
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

컬럼 추가 결과

	이름	나이	주소	전화번호
0	길동	26	서울	NaN
1	영희	25	인천	NaN

데이터 수정 결과

	이름	나이	주소	전화번호
0	길동	26	서울	01012341234
1	영희	25	인천	NaN

데이터 변경 – 컬럼 삭제

DataFrame에서 컬럼 삭제 후 원본 변경

```
df.drop('전화번호', axis = 1, inplace = True)      # 컬럼 삭제
# axis = 1 : 열 방향 / axis = 0 : 행 방향
# inplace = True : 원본 변경 / inplace = False : 원본 변경 X
```

*pandas 라이브러리는 이미 import 해둔 것으로 가정

컬럼 삭제 결과

	이름	나이	주소
0	길동	26	서울
1	영희	25	인천

크레딧

/* elice */

코스 매니저
하주희

콘텐츠 제작자
임원균, 하주희

강사
황지영

감수자
장석준

디자이너
강혜정

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io