pandas 기초

열(column) 중심 데이터 분석 API(Application Programming Interface)입니다.
입력 데이터를 처리하고 분석하는 데 효과적인 도구이며,
여러 ML 프레임워크에서도 Pandas 데이터 구조를 입력으로 지원합니다.

https://pandas.pydata.org/pandas-docs/stable/index.html (https://pandas.pydata.org/pandas-docs/stable/index.html)

판다스 설치 C:\Users\tina>conda install pandas

● pandas 라이브러리의 Series 및 DataFrame 구조에 관한 소개

Pandas에서 제공하는 Series 부분 코딩

In [1]:
```
1 import pandas as pd
2 pd.__version__
```

Out[1]: '1.5.3'

Use Guide
https://pandas.pydata.org/docs/user_guide/dsintro.html#series (https://pandas.pydata.org/docs/user_guide/dsintro.html#series)

위 사이트를 참고로 코딩

In [2]:
```
1 s = pd.Series(data, index=index)
2 # data 부분에 실제 의미있는 값이 들어가지 않아서 에러 발생
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 s = pd.Series(data, index=index)

NameError: name 'data' is not defined
```

In [3]:
```
#numpy에서 제공하는 랜덤함수를 활용하여 데이터를 생성

import numpy as np
s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
```

In [4]:
```
s
```

Out[4]:
```
a    0.335998
b   -0.091687
c   -1.313730
d    1.232747
e   -1.095451
dtype: float64
```

In [5]:
```
 s.index
```

Out[5]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

In [6]:
```
s = pd.Series(np.random.randn(5)) # index 생략가능
```

In [7]:
```
s
```

Out[7]:
```
0    0.704178
1    0.003439
2    0.675634
3    0.981088
4   -1.005553
dtype: float64
```

In [8]:
```
s.index
```

Out[8]:
```
RangeIndex(start=0, stop=5, step=1)
```

In [9]:
```
d = {"b": 1, "a": 0, "c": 2}  # 딕셔너리 타입으로 d 데이터 생성
```

In [10]:
```
print(type(d))
print(d)
```

```
<class 'dict'>
{'b': 1, 'a': 0, 'c': 2}
```

In [11]:
```
pd.Series(d)  # 판단스 시리즈에 딕셔너리 타입을 전달

# 딕셔너리에서 key 부분이 index, value 부분이 시리즈에서 데이터 값으로 생성
```

Out[11]:
```
b    1
a    0
c    2
dtype: int64
```

In [12]:
```
d = {"a": 0.0, "b": 1.0, "c": 2.0}
```

In [13]:
```
pd.Series(d)
```

Out[13]:
```
a    0.0
b    1.0
c    2.0
dtype: float64
```

In [14]:
```
pd.Series(d, index=["b", "c", "d", "a"])  # index값을 기준으로 데이터가 출력됨
```

Out[14]:
```
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
```

In [15]:
```
# 참고
dd = {"a": 0.0, "b": 1.0, "c": 2.0}  # 딕셔너리 생성
ddd = pd.Series(dd)  # 판다스 시리즈 생성
print(ddd)
```

```
a    0.0
b    1.0
c    2.0
dtype: float64
```

In [16]:
```
# 참고
print(d)
dddd = pd.Series(d, index=["b", "c", "d", "a"]) # index 값을 기준으로 데이터 순서 변경
print(dddd)
print("● type(dddd)", type(dddd))
```

```
{'a': 0.0, 'b': 1.0, 'c': 2.0}
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
● type(dddd) <class 'pandas.core.series.Series'>
```

위와 같이 하나씩 코딩하면서 공부해야 함!!

Pandas에서 제공하는 DataFrame 부분 코딩

In [17]:
```
d = {'one': pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
     'two': pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}
```

In [18]:
```
type(d)  # 딕셔너리 타입
```

Out[18]:
```
dict
```

In [19]:
```
d  # 딕셔너리 타입으로 key : index value 가 있는 상태
```

Out[19]:
```
{'one': a    1.0
 b    2.0
 c    3.0
 dtype: float64,
 'two': a    1.0
 b    2.0
 c    3.0
 d    4.0
 dtype: float64}
```

In [20]:
```
df = pd.DataFrame(d)  # key : 열(columne)이름,  index : 행(row)이름으로 연결됨
```

In [21]:
```
df  # key : 열(columne)이름,  index : 행(row)이름으로 연결됨
```

Out[21]:

	one	two
a	1.0	1.0
b	2.0	2.0
c	3.0	3.0
d	NaN	4.0

In [22]:
```
print(d)
pd.DataFrame(d, index=['d', 'b', 'a'])
```

```
{'one': a    1.0
b    2.0
c    3.0
dtype: float64, 'two': a    1.0
b    2.0
c    3.0
d    4.0
dtype: float64}
```

Out[22]:

	one	two
d	NaN	4.0
b	2.0	2.0
a	1.0	1.0

In [23]:
```
# 열(column)이름 변경 시도 (기존에 없었던 'three'를 설정하면 새로 생성되고 값은 NaN)

pd.DataFrame(d, index=['d', 'b', 'a'], columns=['two', 'three'])
```

Out[23]:

	two	three
d	4.0	NaN
b	2.0	NaN
a	1.0	NaN

In [24]:
```
df # 위에서 생성한 df는 처음 값을 유지하고 있음
```

Out[24]:

	one	two
a	1.0	1.0
b	2.0	2.0
c	3.0	3.0
d	NaN	4.0

In [25]:
```
df.index  # df의 index 부분 출력
```

Out[25]:
```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

In [26]:
```
df.columns # df의 column 부분 출력
```

Out[26]:
```
Index(['one', 'two'], dtype='object')
```

Pandas의 기본 데이터 구조는 두 가지 클래스로 구현됩니다.
1. Series는 하나의 "열"입니다.
2. DataFrame은 행(row) 및 열(column)이 포함된 "관계형 데이터 테이블"이라고 생각할 수 있습니다.
   DataFrame에는 하나 이상의 "Series"와 "각 Series의 index"부분이 포함됩니다.

1) Series를 만드는 한 가지 방법은 Series 객체를 만드는 것입니다.

In [27]:
```
#샌프란시스코, 새너제이, 새크라멘토 도시 이름
pd.Series(['San Francisco', 'San Jose', 'Sacramento'])
```

Out[27]:
```
0    San Francisco
1         San Jose
2       Sacramento
dtype: object
```

In [28]:
```
#샌프란시스코, 새너제이, 새크라멘토 도시 이름
city_names = pd.Series(['San Francisco', 'San Jose', 'Sacramento'])

#샌프란시스코, 새너제이, 새크라멘토 도시에 해당하는 인구 수
population = pd.Series([852469, 1015785, 485199])
```

In [29]:
```
city_names
```

Out[29]:
```
0    San Francisco
1         San Jose
2       Sacramento
dtype: object
```

In [30]:
```
population
```

Out[30]:
```
0     852469
1    1015785
2     485199
dtype: int64
```

In [31]:
```
type(population)
```

Out[31]:
```
pandas.core.series.Series
```

In [32]:
```
population / 1000.  #Python의 기본 산술 연산이 Series 원소별 연산에 적용됨
```

Out[32]:
```
0     852.469
1    1015.785
2     485.199
dtype: float64
```

2) DataFrame 객체는 열(columbn) 이름과 매핑되는 문자열을 'dict'의 key에 전달하여 각각의 Series와 연결하여 만들 수 있다.
Series의 길이가 일치하지 않는 경우, 누락된 값은 특수 NA/NaN 값으로 채워진다.

In [33]:
```
 #딕셔너리 타입 활용 (주로 사용됨)

dic_df = pd.DataFrame({ 'City name': city_names,'Population': population })
```

In [34]:
```
1 dic_df
```

Out[34]:

	City name	Population
0	San Francisco	852469
1	San Jose	1015785
2	Sacramento	485199

DataFrame 객체는 Series를 'list' 타입에 전달하면 아래와 같이 행(row) 단위로 생성된다.

In [35]:
```
1 list_df = pd.DataFrame([city_names, population])  #리스트 타입 활용
```

In [36]:
```
1 list_df   # 행(row) 단위로 생성
```

Out[36]:

	0	1	2
0	San Francisco	San Jose	Sacramento
1	852469	1015785	485199

DataFrame 에서는 DataFrame.columns 이용하여 열(columne) 이름을 리스트 타입으로 설정할 수 있다.

In [37]:
```
1 #DataFrame에서는 DataFrame.columns 이용하여 열(column)이름 설정
2
3 list_df.columns = ['city_1','city_2','city_3']
```

In [38]:
```
1 list_df
```

Out[38]:

	city_1	city_2	city_3
0	San Francisco	San Jose	Sacramento
1	852469	1015785	485199

DataFrame 에서는 DataFrame.index 이용하여 index 이름을 리스트 타입으로 설정할 수 있다.

In [39]:
```
#DataFrame에서는 DataFrame.index 이용하여 index 이름 설정
list_df.index = ['City_name','Population']
```

In [40]:
```
list_df
```

Out[40]:

	city_1	city_2	city_3
City_name	San Francisco	San Jose	Sacramento
Population	852469	1015785	485199

In [ ]:
```

```