Python Data Analysis Library - Pandas

TEAMLAB director
최성철

boostcamp AI Tech

pandas

구조화된 데이터의 처리를 지원하는 Python 라이브러리

Python계의 엑셀!

pandas

- 구조화된 데이터의 처리를 지원하는 Python 라이브러리
- panel data → pandas
- 고성능 array 계산 라이브러리인 numpy와 통합하여,
  강력한 "스프레드시트" 처리 기능을 제공
- 인덱싱, 연산용 함수, 전처리 함수 등을 제공함
- 데이터 처리 및 통계 분석을 위해 사용

Data table, Sample

attribute, field, **feature**, column

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	CAT. MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	0
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6	0
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	1
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	1
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2	1
0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7	0
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9	0
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1	0
0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5	0
0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9	0
0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15	0
0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.9	13.27	18.9	0
0.09378	12.5	7.87	0	0.524	5.889	39	5.4509	5	311	15.2	390.5	15.71	21.7	0
0.62976	0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21	396.9	8.26	20.4	0
0.63796	0	8.14	0	0.538	6.096	84.5	4.4619	4	307	21	380.02	10.26	18.2	0

instance, **tuple**, row

Feature vector

data

pandas 설치


```
conda create -n ml python=3.8 # 가상환경생성
activate ml     # 가상환경실행
conda install pandas    #  pandas 설치
```

```
jupyter notebook # 주피터 실행하기
```

데이터 로딩

In [1]:
```
import pandas as pd #라이브러리 호출
```

In [2]:
```
data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data' #Data URL
df_data = pd.read_csv(data_url, sep='₩s+', header = None) #csv 타입 데이터 로드, separate는 빈공간으로 지정하고, Column은 없음
```

In [3]:
```
df_data.head() #처음 다섯줄 출력
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

데이터 로딩

In [4]:
```
df_data.columns = ['CRIM','ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO' ,'B', 'LSTAT', 'MEDV']
# Column Header 이름 지정
df_data.head()
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

기본적인 것 부터 해보자

series

pandas의 구성

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	weight_0
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	1
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	1

Series

DataFrame 중 하나의 Column에 해당하는 데이터의 모음 Object

DataFrame

DataTable 전체를 포함하는 Object

일반적인 pandas의 활용


```
In [1]: import pandas as pd #라이브러리 호출

In [2]: data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data' #Data URL
        df_data = pd.read_csv(data_url, sep='₩s+', header = None) #csv 타입 데이터 로드, separate는 빈공간으로 지정하고, Column은 없음

In [3]: df_data.head() #처음 다섯줄 출력
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

기존데이터를불러와서
DataFrame을생성

Series

- column vector를 표현하는 object

```
In [1]: from pandas import Series, DataFrame
        import pandas as pd

In [ ]: example_obj = Series()
```

```
Init signature: Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)
Docstring:
One-dimensional ndarray with axis labels (including time series).
```

Shift + TAB

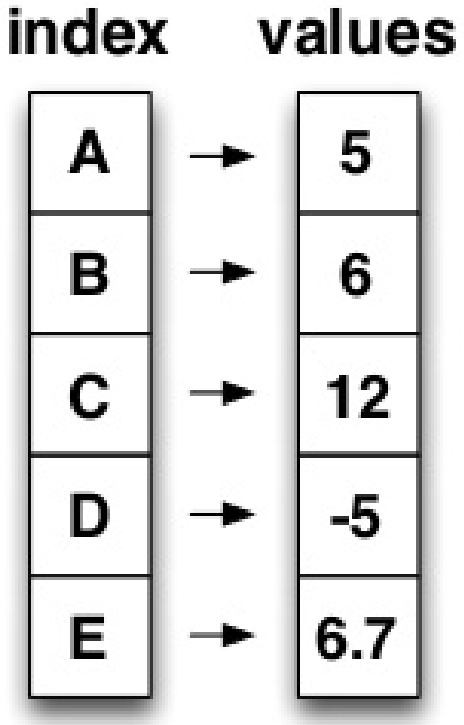
Series

Index

data

Data type

Series



- Subclass of `numpy.ndarray`
- Data: any type
- Index labels need not be ordered
- Duplicates are possible (but result in reduced functionality)

https://www.slideshare.net/wesm/pandas-powerful-data-analysis-tools-for-python

Series

Series

data와 index 이름을 지정

```
In [4]: dict_data = {"a":1, "b":2, "c":3, "d":4, "e":5}
        example_obj = Series(dict_data, dtype=np.float32, name="example_data")
        example_obj
```

data type 설정

series 이름 설정

```
Out[4]: a    1.0
        b    2.0
        c    3.0
        d    4.0
        e    5.0
        Name: example_data, dtype: float32
```

data index에 접근하기

data index에 값 할당하기

Series


```
example_obj.values
```

값 리스트만

```
array([ 3.20000005,  2.        ,  3.        ,  4.        ,  5.        ],
 dtype=float32)
```

```
example_obj.index
```

Index 리스트만

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

```
example_obj.name = "number"
example_obj.index.name = "alphabet"
example_obj
```

Data에 대한 정보를 저장

```
alphabet
a    3.2
b    2.0
c    3.0
d    4.0
e    5.0
Name: number, dtype: float32
```

Series


```
dict_data_1 = {"a":1, "b":2, "c":3, "d":4, "e":5}
indexes = ["a","b","c","d","e","f","g","h"]
series_obj_1 = Series(dict_data_1, index=indexes)
series_obj_1
```

index 값을 기준으로 series 생성

```
a    1.0
b    2.0
c    3.0
d    4.0
e    5.0
f    NaN
g    NaN
h    NaN
dtype: float64
```

dataframe

pandas의 구성

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	weight_0
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	1
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	1

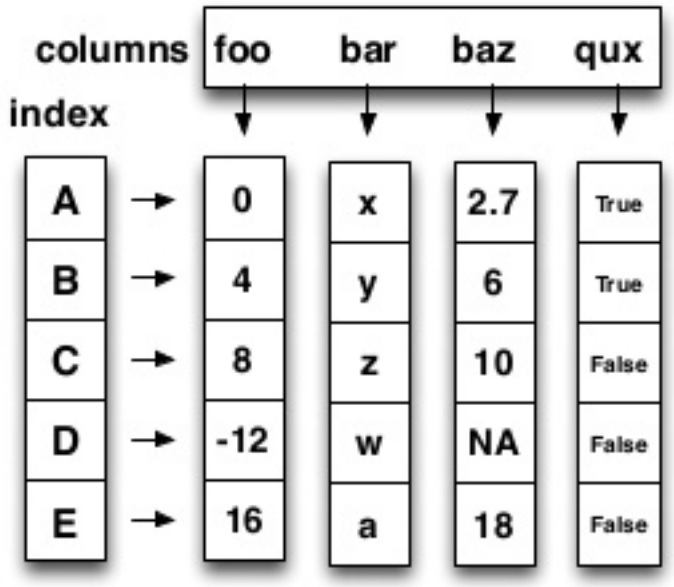
Series

DataFrame 중 하나의 Column에 해당하는 데이터의 모음 Object

DataFrame

Data Table 전체를 포함하는 Object

dataframe memory



- NumPy array-like
- Each column can have a different type
- Row and column index
- Size mutable: insert and delete columns

https://www.slideshare.net/wesm/pandas-powerful-data-analysis-tools-for-python

dataframe overview

Series를 모아서 만든 Data Table = 기본 2차원

```
In [4]: from pandas import Series, DataFrame
        import pandas as pd
        import numpy as np

In [ ]: DataFrame()
```

Shift + TAB

```
In [3]: Init signature: DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)
        Docstring:
        Two-dimensional size-mutable, potentially heterogeneous tabular data
        structure with labeled axes (rows and columns). Arithmetic operations
```

dataframe 생성

In [1]:
```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

column_name : data

In [2]:
```
# Example from - https://chrisalbon.com/python/pandas_map_values_to_values.html
raw_data = {'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
        'last_name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze'],
        'age': [42, 52, 36, 24, 73],
        'city': ['San Francisco', 'Baltimore', 'Miami', 'Douglas', 'Boston']}
df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age', 'city'])
df
```

Out[2]:

	first_name	last_name	age	city
0	Jason	Miller	42	San Francisco
1	Molly	Jacobson	52	Baltimore
2	Tina	Ali	36	Miami
3	Jake	Milner	24	Douglas
4	Amy	Cooze	73	Boston

dataframe 생성


```
In [3]: DataFrame(raw_data, columns = ["age", "city"])
```

Out[3]:

column 선택

	age	city
0	42	San Francisco
1	52	Baltimore
2	36	Miami
3	24	Douglas
4	73	Boston

```
In [4]: DataFrame(raw_data, columns = ["first_name","last_name","age", "city", "debt"])
```

Out[4]:

새로운 column 추가

	first_name	last_name	age	city	debt
0	Jason	Miller	42	San Francisco	NaN
1	Molly	Jacobson	52	Baltimore	NaN
2	Tina	Ali	36	Miami	NaN
3	Jake	Milner	24	Douglas	NaN
4	Amy	Cooze	73	Boston	NaN

dataframe 생성


```
df = DataFrame(raw_data, columns = ["first_name","last_name","age", "city", "debt"])
df.first_name
```

```
0    Jason
1    Molly
2     Tina
3     Jake
4      Amy
Name: first_name, dtype: object
```

column 선택 – series 추출

```
df["first_name"]
```

```
0    Jason
1    Molly
2     Tina
3     Jake
4      Amy
Name: first_name, dtype: object
```

column 선택 – series 추출

dataframe indexing

loc – index location

iloc – index position

dataframe indexing

loc은 index 이름, iloc은 index number

```
# Example from - https://stackoverflow.com/questions/31593201/pandas-iloc-vs-ix-vs-loc-explanation
s = pd.Series(np.nan, index=[49,48,47,46,45, 1, 2, 3, 4, 5])
s.loc[:3]
```

```
49   NaN
48   NaN
47   NaN
46   NaN
45   NaN
1    NaN
2    NaN
3    NaN
dtype: float64
```

```
s.iloc[:3]
```

```
49   NaN
48   NaN
47   NaN
dtype: float64
```

column에 새로운 데이터 할당

dataframe handling


```
In [13]: df.T
```

Out[13]:

	0	1	2	3	4
first_name	Jason	Molly	Tina	Jake	Amy
last_name	Miller	Jacobson	Ali	Milner	Cooze
age	42	52	36	24	73
city	San Francisco	Baltimore	Miami	Douglas	Boston
debt	True	True	False	False	True

transpose

```
In [14]: df.values
```

```
Out[14]: array([['Jason', 'Miller', 42, 'San Francisco', True],
                ['Molly', 'Jacobson', 52, 'Baltimore', True],
                ['Tina', 'Ali', 36, 'Miami', False],
                ['Jake', 'Milner', 24, 'Douglas', False],
                ['Amy', 'Cooze', 73, 'Boston', True]], dtype=object)
```

값 출력

```
In [20]: df.to_csv()
```

```
Out[20]: ',first_name,last_name,age,city,debt₩n0,Jason,Miller,42,San Francisco,True₩n1,Molly,Jacobson,52,Baltimore,True₩n2,Tina,Ali,36,Miami,False₩n
3,Jake,Milner,24,Douglas,False₩n4,Amy,Cooze,73,Boston,True₩n'
```

csv 변환

dataframe handling

column을 삭제함

```
del df["debt"]
```

```
df
```

	first_name	last_name	age	city
0	Jason	Miller	42	San Francisco
1	Molly	Jacobson	52	Baltimore
2	Tina	Ali	36	Miami
3	Jake	Milner	24	Douglas
4	Amy	Cooze	73	Boston

selection & drop

Selection with column names


```
df["account"].head(3)
```

한개의 column 선택시

```
0    211829
1    320563
2    648336
Name: account, dtype: int64
```

1개 이상의 column 선택

```
df[["account", "street", "state"]].head(3)
```

	account	street	state
0	211829	34456 Sean Highway	Texas
1	320563	1311 Alvis Tunnel	NorthCarolina
2	648336	62184 Schamberger Underpass Apt. 231	Iowa

Selection with index number


```
df[:3]
```

column 이름 없이 사용하는 index number는 row 기준 표시

	account	name	street	city	state	postal-code	Jan	Feb	Mar
0	211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000
1	320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000
2	648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000

```
df["account"][:3]
```

column이름과 함께 row index 사용시, 해당 column만

```
0    211829
1    320563
2    648336
Name: account, dtype: int64
```

Selection with index number


```
account_serires = df["account"]
account_serires[:3]
```

```
0    211829
1    320563
2    648336
Name: account, dtype: int64
```

```
account_serires[[0,1,2]]
```

```
0    211829
1    320563
2    648336
Name: account, dtype: int64
```

1개 이상의 index

```
account_serires[account_serires<250000]
```

```
0      211829
3      109996
4      121213
5      132971
6      145068
7      205217
8      209744
9      212303
10     214098
11     231907
12     242368
Name: account, dtype: int64
```

Boolean index

index 변경


```
df.index = df["account"]
del df["account"]
df.head()
```

	name	street	city	state	postal-code	Jan	Feb	Mar
account								
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000
320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000
648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000
109996	D'Amore, Gleichner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000
121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000

basic, loc, iloc selection

Column 과 index number

```
df[["name","street"]][:2]
```

	name	street
account		
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway
320563	Walter-Trantow	1311 Alvis Tunnel

Column number와 index number

```
df.iloc[:2,:2]
```

	name	street
account		
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway
320563	Walter-Trantow	1311 Alvis Tunnel

```
df.loc[[211829,320563],["name","street"]]
```

Column 과 index name

	name	street
account		
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway
320563	Walter-Trantow	1311 Alvis Tunnel

index 재설정


```
df.index = list(range(0,15))
df.head()
```

	name	street	city	state	postal-code
0	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752
1	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365
2	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517

data drop


```
df.drop(1)
```

index number로 drop

	name	street	city	s
0	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	T
2	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Ic

data drop


```
df.drop([0,1,2,3])
```

한개 이상의 Index number로 drop

	account	name	street	city	state
4	121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California
5	132971	Williamson, Schumm and Hettinger	89403 Casimer Spring	Jeremieburgh	Arkansas
6	145068	Casper LLC	340 Consuela Bridge Apt. 400	Lake Gabriellaton	Mississipi

data drop

axis 지정으로 축을 기준으로 drop ➔ column 중에 “city”

```
df.drop("city",axis=1) # df.drop(["city", "state"],axis=1)
```

	name	street	state	postal-code
0	Kerluke, Koepp and Hilpert	34456 Sean Highway	Texas	28752
1	Walter-Trantow	1311 Alvis Tunnel	NorthCarolina	38365
2	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	Iowa	76517

dataframe operations

series operation


```
s1 = Series(
    range(1,6), index=list("abced"))
s1
```

```
a    1
b    2
c    3
e    4
d    5
dtype: int64
```

```
s2 = Series(
    range(5,11), index=list("bcedef"))
s2
```

```
b     5
c     6
e     7
d     8
e     9
f    10
dtype: int64
```

```
s1.add(s2)
```

```
a     NaN
b     7.0
c     9.0
d    13.0
e    11.0
e    13.0
f     NaN
dtype: float64
```

```
s1 + s2
```

```
a     NaN
b     7.0
c     9.0
d    13.0
e    11.0
e    13.0
f     NaN
dtype: float64
```

index 으로 기준으로 연산수행

겹치는 index가 없을 경우
NaN값으로 반환

dataframe operation


```
df1 = DataFrame(
    np.arange(9).reshape(3,3),
    columns=list("abc"))
df1
```

	a	b	c
0	0	1	2
1	3	4	5
2	6	7	8

```
df2 = DataFrame(
    np.arange(16).reshape(4,4),
    columns=list("abcd"))
df2
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

```
df1 + df2
```

	a	b	c	d
0	0.0	2.0	4.0	NaN
1	7.0	9.0	11.0	NaN
2	14.0	16.0	18.0	NaN
3	NaN	NaN	NaN	NaN

```
df1.add(df2,fill_value=0)
```

	a	b	c	d
0	0.0	2.0	4.0	3.0
1	7.0	9.0	11.0	7.0
2	14.0	16.0	18.0	11.0
3	12.0	13.0	14.0	15.0

df는 column과 index를 모두 고려

add operation을 쓰면 NaN값 0으로 변환
Operation types: add, sub, div, mul

series + dataframe


```
df = DataFrame(
    np.arange(16).reshape(4,4),
    columns=list("abcd"))
df
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

```
s2 = Series(np.arange(10,14))
s2
```

```
0    10
1    11
2    12
3    13
dtype: int64
```

```
df + s2
```

	a	b	c	d	0	1	2	3
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
df.add(s2, axis=0)
```

	a	b	c	d
0	10	11	12	13
1	15	16	17	18
2	20	21	22	23
3	25	26	27	28

axis를 기준으로
row broadcasting 실행

lambda, map, apply

map for series

- pandas의 series type의 데이터에도 map 함수 사용가능
- function 대신 dict, sequence형 자료등으로 대체 가능

```
s1 = Series(np.arange(10))
s1.head(5)
```

```
0    0
1    1
2    2
3    3
4    4
dtype: int64
```

```
s1.map(lambda x: x**2).head(5)
```

```
0     0
1     1
2     4
3     9
4    16
dtype: int64
```

map for series


```
z = {1: 'A', 2: 'B', 3: 'C'}
s1.map(z).head(5)
```

```
0    NaN
1      A
2      B
3      C
4    NaN
dtype: object
```

dict type으로
데이터 교체

없는 값은 NaN

```
s2 = Series(np.arange(10,20))
s1.map(s2).head(5)
```

```
0    10
1    11
2    12
3    13
4    14
dtype: int64
```

같은 위치의 데이터를s2로
전환

example - map for series


```
df = pd.read_csv("wages.csv")
df.head()
```

	earn	height	sex	race	ed	age
0	79571.299011	73.89	male	white	16	49
1	96396.988643	66.23	female	white	16	62
2	48710.666947	63.77	female	white	16	33
3	80478.096153	63.22	female	other	16	95
4	82089.345498	63.08	female	white	17	43

```
df.sex.unique()
```

```
array(['male', 'female'], dtype=object)
```

```
df["sex_code"] = df.sex.map({"male":0, "female":1})
df.head(5)
```

성별 str → 성별 code

	earn	height	sex	race	ed	age	sex_code
0	79571.299011	73.89	male	white	16	49	0
1	96396.988643	66.23	female	white	16	62	1
2	48710.666947	63.77	female	white	16	33	1
3	80478.096153	63.22	female	other	16	95	1
4	82089.345498	63.08	female	white	17	43	1

replace function

- Map 함수의 기능중 데이터 변환 기능만 담당
- 데이터 변환시 많이 사용하는 함수

```
df.sex.replace(
    {"male":0, "female":1}
).head()
```

```
0    0
1    1
2    1
3    1
4    1
Name: sex, dtype: int64
```

dict type 적용

Target list
Conversion list

```
df.sex.replace(
    ["male", "female"],
    [0,1], inplace=True)
df.head(5)
```

	earn	height	sex	race	ed	age	sex_code
0	79571.299011	73.89	0	white	16	49	0
1	96396.988643	66.23	1	white	16	62	1
2	48710.666947	63.77	1	white	16	33	1

inplace ← 데이터 변환결과를 적용

apply for dataframe

- map과 달리, series 전체(column)에 해당 함수를 적용
- 입력 값이 series 데이터로 입력 받아 handling 가능

```
df_info = df[["earn", "height","age"]]
df_info.head()
```

	earn	height	age
0	79571.299011	73.89	49
1	96396.988643	66.23	62
2	48710.666947	63.77	33
3	80478.096153	63.22	95
4	82089.345498	63.08	43

```
f = lambda x : x.max() - x.min()
df_info.apply(f)
```

```
earn      318047.708444
height        19.870000
age           73.000000
dtype: float64
```

각 column 별로 결과값 반환

apply for dataframe

- 내장 연산 함수를 사용할 때도 똑같은 효과를 거둘 수 있음
- mean, std 등 사용가능

```
df_info.sum()
```

```
earn      4.474344e+07
height    9.183125e+04
age       6.250800e+04
dtype: float64
```

```
df_info.apply(sum)
```

```
earn      4.474344e+07
height    9.183125e+04
age       6.250800e+04
dtype: float64
```

apply for dataframe

- scalar 값 이외에 series값의 반환도 가능함

```
def f(x):
    return Series([x.min(), x.max()], index=["min", "max"])
df_info.apply(f)
```

	earn	height	age
min	-98.580489	57.34	22
max	317949.127955	77.21	95

applymap for dataframe

- series 단위가 아닌 element 단위로 함수를 적용함
- series 단위에 apply를 적용시킬 때와 같은 효과

```
f = lambda x : -x
df_info.applymap(f).head(5)
```

	earn	height	age
0	-79571.299011	-73.89	-49
1	-96396.988643	-66.23	-62
2	-48710.666947	-63.77	-33
3	-80478.096153	-63.22	-95
4	-82089.345498	-63.08	-43

```
f = lambda x : -x
df_info["earn"].apply(f).head(5)
```

```
0    -79571.299011
1    -96396.988643
2    -48710.666947
3    -80478.096153
4    -82089.345498
Name: earn, dtype: float64
```

pandas built-in functions

describe

- Numeric type 데이터의 요약 정보를 보여줌

```
df = pd.read_csv("wages.csv")
df.head()
```

	earn	height	sex	race	ed	age
0	79571.299011	73.89	male	white	16	49
1	96396.988643	66.23	female	white	16	62
2	48710.666947	63.77	female	white	16	33
3	80478.096153	63.22	female	other	16	95
4	82089.345498	63.08	female	white	17	43

```
df.describe()
```

	earn	height	ed	age
count	1379.000000	1379.000000	1379.000000	1379.000000
mean	32446.292622	66.592640	13.354605	45.328499
std	31257.070006	3.818108	2.438741	15.789715
min	-98.580489	57.340000	3.000000	22.000000
25%	10538.790721	63.720000	12.000000	33.000000
50%	26877.870178	66.050000	13.000000	42.000000
75%	44506.215336	69.315000	15.000000	55.000000
max	317949.127955	77.210000	18.000000	95.000000

unique

- series data의 유일한 값을 list를 반환함

```
df.race.unique()
```

유일한 인종의 값 list

```
array(['white', 'other', 'hispanic', 'black'], dtype=object)
```

```
np.array(dict(enumerate(df["race"].unique())))
```

dict type으로 index

```
array({0: 'white', 1: 'other', 2: 'hispanic', 3: 'black'}, dtype=object)
```

```
value = list(map(int, np.array(list(enumerate(df["race"].unique())))[:, 0].tolist()))
key = np.array(list(enumerate(df["race"].unique())), dtype=str)[:, 1].tolist()

value, key
```

label index 값과 label 값 각각 추출

```
([0, 1, 2, 3], ['white', 'other', 'hispanic', 'black'])
```

unique

- series data의 유일한 값을 list를 반환함

```
df["race"].replace(to_replace=key, value=value, inplace=True)
```

label str → index 값으로 변환

```
value = list(map(int, np.array(list(enumerate(df["sex"].unique())))[:, 0].tolist()))
key = np.array(list(enumerate(df["sex"].unique())), dtype=str)[:, 1].tolist()

value, key
```

성별에 대해서도 동일하게 적용

```
([0, 1], ['male', 'female'])
```

```
df["sex"].replace(to_replace=key, value=value, inplace=True)
df.head(5)
```

	earn	height	sex	race	ed	age
0	79571.299011	73.89	0	0	16	49
1	96396.988643	66.23	1	0	16	62

"sex"와 "race" column의 index labelling

sum

- 기본적인 column 또는 row 값의 연산을 지원
- sub, mean, min, max, count, median, mad, var 등

```
df.sum(axis=0)
```
column 별

```
earn      4.474344e+07
height    9.183125e+04
sex       8.590000e+02
race      5.610000e+02
ed        1.841600e+04
age       6.250800e+04
dtype: float64
```

```
df.sum(axis=1)
```
row 별

```
0        79710.189011
1        96542.218643
2        48824.436947
3        80654.316153
4        82213.425498
5        15423.882901
6        47231.711821
```

isnull

- column 또는 row 값의 NaN (null) 값의 index를 반환함

```
df.isnull()
```

	earn	height	sex	race	ed	age
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False

```
df.isnull().sum()
```

```
earn      0
height    0
sex       0
race      0
ed        0
age       0
dtype: int64
```

Null인 값의 합

sort_values

- column 값을 기준으로 데이터를 sorting

```
df.sort_values(["age", "earn"], ascending=True).head(10)
```

	earn	height	sex	race	ed	age
1038	-56.321979	67.81	0	2	10	22
800	-27.876819	72.29	0	0	12	22
963	-25.655260	68.90	0	0	12	22
1105	988.565070	64.71	1	0	12	22
801	1000.221504	64.09	1	0	12	22

ascending → 오름차순

Correlation & Covariance

- 상관계수와 공분산을 구하는 함수
- corr, cov, corrwith

```
df.age.corr(df.earn)
```

```
0.07400349177836057 5
```

```
df.age.cov(df.earn)
```

```
36523.6992104089
```

```
df.corrwith(df.earn)
```

```
earn        1.000000
height      0.291600
sex        -0.337328
race       -0.063977
ed          0.350374
age         0.074003
dtype: float64
```

Correlation & Covariance


```
df.corr()
```

	earn	height	sex	race	ed	age
earn	1.000000	0.291600	-0.337328	-0.063977	0.350374	0.074003
height	0.291600	1.000000	-0.703672	-0.045974	0.114047	-0.133727
sex	-0.337328	-0.703672	1.000000	0.000858	-0.061747	0.070036
race	-0.063977	-0.045974	0.000858	1.000000	-0.049487	-0.056879
ed	0.350374	0.114047	-0.061747	-0.049487	1.000000	-0.129802
age	0.074003	-0.133727	0.070036	-0.056879	-0.129802	1.000000

End of Document
Thank You.