

NUMPY

Numpy

■ Numpy 개요

- 배열의 대한 처리에 특화된 파이썬 라이브러리
- Numerical Python의 줄임말
- 빠른 배열 연산 처리를 가능하게 해준다
- 통상적으로 `import numpy as np`를 통해 사용한다

■ 주요기능

- 유니버설 함수(universal function)
- 수학/선형대수/통계처리 함수 등 다양한 함수
- 다른 언어에 대한 인터페이스



Numpy

■ Numpy 필요성

- 파이썬은 기본적으로 배열을 리스트나 튜플로 저장
- 그러나 이는 대규모 데이터 처리 시 C 등의 언어에 비해 현저히 느리다
- Numpy는 다차원 배열 객체 **ndarray**를 사용하여 고속 데이터 처리를 가능하게 한다

■ ndarray의 작동 방식

- Numpy의 빠른 데이터 처리 속도는 다차원 배열 객체인 ndarray를 기본 데이터 저장 포맷으로 사용하고 있기 때문
- 파이썬의 기본 객체인 list는 각 요소가 메모리 이곳저곳에 배치되는 데 반해 ndarray는 RAM에 빈틈없이 배치됨
- 데이터가 분산되어 배치되면 데이터에 접근할 때 오버헤드가 발생한다
- 따라서 배열 연산을 C와 같은 속도가 빠른 low-level language로 구현할 수 있다

Numpy 데이터타입

■ Numpy – detailed data type

- Numpy가 아닌 기본 파이썬 코딩에서는 변수를 선언할 때 세부적인 데이터 타입을 지정할 필요가 없다

```
# python에서 잘못된 변수 선언(데이터 타입을 지정함)  
int a = 2  
# python에서 올바른 변수 선언  
a = 2
```

- 이로 인해 코드 작성이 간편해지지만 메모리를 낭비할 가능성이 존재한다
- Numpy를 사용하면 메모리의 사용 절감을 위해 비트 수 등을 지정할 수 있다

Numpy의 데이터 타입

- Numpy – detailed data type
 - 다양하게 사용할 수 있는 내장 데이터 타입이 존재한다

데이터 타입명	설명
bool_, bool8	파이썬의 부울 타입과 호환되는 부울 타입
byte/short/intc/longlong	C의 char/short/int/long long과 호환
int8/int16/int32/int64	8/16/32/64 비트 부호 있는 정수 타입
float_	파이썬의 float과 호환되는 타입
float16/float32/float64	16/32/64비트 부동 소수점 타입
object_	파이썬 객체
str_	고정길이 문자열 타입
...	...

Numpy - ndarray

■ 다차원 배열 객체 ndarray

- np.arange는 파이썬 자체 내장 함수인 range와 비슷하다

```
# start로 지정한 값부터 stop 사이 값 중 step 간격을 갖는 수열을 생성한다
np.arange([start,] stop[, step,][, dtype])
```

```
# range로 생성
for i in range(10,20,2):
    print(i)
```

```
10
12
14
16
18
```

```
# np.arange()로 생성
np.arange(10,20,2)
Out[11]: array([10, 12, 14, 16, 18])
```

Numpy - ndarray

- ndarray의 데이터 타입 지정
 - Generic Type으로 지정
 - 파이썬 내장 데이터 타입에 대응하는 타입을 지정
 - Numpy의 내장 데이터 타입으로 지정
 - 문자열 코드로 지정

```
# dtype 객체로 지정  
dt = np.dtype( ' float ' )
```

```
# ndarray를 생성할 때 dtype으로 지정(dtype 객체 사용)  
x = np.array([1.1, 2.1, 3.1], dtype=dt)
```

```
# ndarray를 생성할 때 dtype으로 지정(직접 문자열로 지정)  
x = np.array([1.1, 2.1, 3.1], dtype='float')
```


Numpy - ndarray

■ 파이썬 리스트와 배열의 차이점

```
# 3x5 배열 d 정의
d = np.array([[1,2,3,4,5], [2,4,5,6,7], [5,7,8,9,9]])
# 배열 d와 똑같이 생긴 리스트를 만들어 d_list객체에 저장
d_list = [[1,2,3,4,5],
           [2,4,5,6,7],
           [5,7,8,9,9]]
type(d_list)
```

list

■ 생김새가 비슷하지만 같은 자료형은 아님

```
# d_list의 두 번째 원소까지 슬라이싱을 해 0을 저장하라고 명령하면 오류가 발생
d_list[:2] = 0
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-28-7779e178cd5c> in <module>()
----> 1 d_list[:2] = 0
```

TypeError: can only assign an iterable

Numpy - ndarray

- 파이썬 리스트와 배열의 차이점

```
# 배열 d의 두 번째 원소까지 슬라이싱을 해 0을 저장하라고 명령  
d[:2] = 0  
d # 두 번째 리스트까지 모든 원소에 0을 저장
```

```
array([[0, 0, 0, 0, 0],  
       [0, 0, 0, 0, 0],  
       [5, 7, 8, 9, 9]])
```

- 이러한 기능은 대량의 데이터를 한꺼번에 처리할 때 리스트보다 배열이 경쟁력 있음을 보여줌

Numpy - ndarray

- 데이터와 메모리의 관계
 - Numpy는 같은 타입의 데이터를 메모리상에 모여있도록 배치하여 빠른 처리가 가능하다
 - 이런 방법에는 "행방향 우선" 배치 방법과 "열방향 우선" 방식이 있다
 - 두 가지 배치 방법은 ndarray를 생성하는 함수의 옵션 인자로 지정 가능하다

```
# Fortran과 같은 방식으로 열방향 우선 메모리 배치 지정  
nda = np.arange(12).reshape(4,3,order='F')
```

```
# C와 같이 행방향 우선 메모리 배치 지정  
ndb = np.zeros((3,3), order = 'C')
```

Order='C'
행 방향 우선
(파이썬의 기본 방법)

0	1	2
3	4	5



메모리상의 데이터 배치

0	1	2	3	4	5
---	---	---	---	---	---

Order='F'
열 방향 우선

0	1	2
3	4	5



메모리상의 데이터 배치

0	3	1	4	2	5
---	---	---	---	---	---

PANDAS

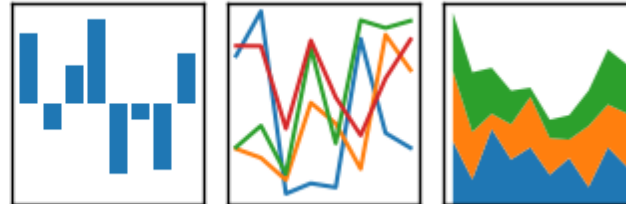
Pandas

- Pandas

- Pandas는 **Open source project**로써, Python 데이터 분석을 돕기 위한 library

pandas

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



Read data

■ Read Data

- Pandas는 데이터 입력을 지원하며, CSV 파일뿐만 아니라 EXCEL 파일도 지원한다.
- CSV 파일은 아래와 같이 Pandas로 읽어 들일 수 있다.

```
# input_file은 불러오려는 CSV 파일의 경로  
data_frame = pd.read_csv(input_file)
```

- 이 때 pd.read_csv 함수의 출력값은 DataFrame이다.
- DataFrame은 Pandas에서 제공하는 2차원의 데이터 구조를 말하며, 라벨을 포함한 열과 행으로 구성되어 있다. (0행, 0열로 시작)
- DataFrame의 보다 자세한 정보는 아래에서 확인 가능하다.
<https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html>

Create data

■ Create Data

- Pandas의 DataFrame은 CSV, EXCEL에서 불러오기 뿐만 아니라, 직접 생성도 가능하다.

```
# 라이브러리를 임포트합니다.  
import pandas as pd  
  
# 데이터프레임을 만듭니다.  
dataframe = pd.DataFrame()  
  
# 열을 추가합니다.  
dataframe['Name'] = ['Jacky Jackson', 'Steven Stevenson']  
dataframe['Age'] = [38, 25]  
dataframe['Driver'] = [True, False]  
  
# 데이터프레임을 확인합니다.  
dataframe
```



	Name	Age	Driver
0	Jacky Jackson	38	True
1	Steven Stevenson	25	False

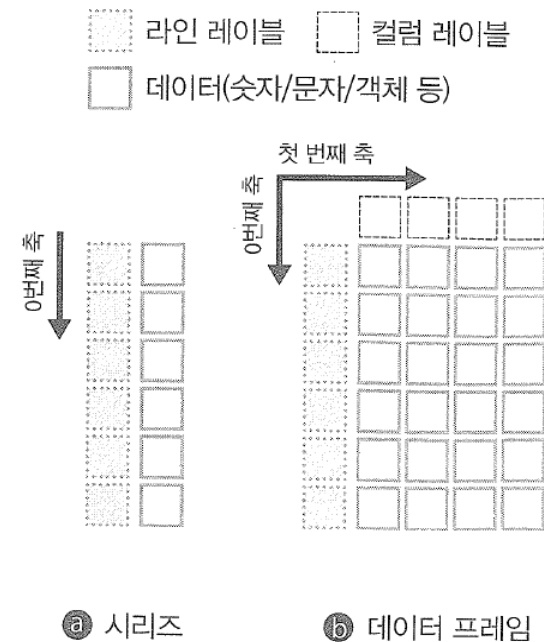
Pandas의 Data Type

■ 기본 데이터 타입

- Pandas는 Numpy를 기반으로 만들어져 있으며, Numpy의 ndarray와 함께 쓰기 적합한 데이터 타입으로 데이터를 저장한다.

① Series (1차원에 적합)

② DataFrame (2차원에 적합)

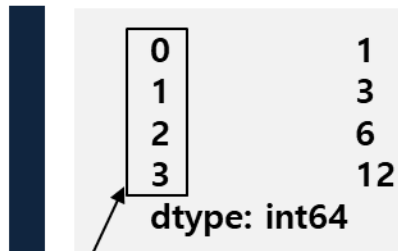


Pandas의 Data Type – Series

■ Series

- Series는 1차원 데이터를 저장하기 위한 데이터 구조이다.
- 정수, 문자열, 부동 소수, 파이썬 객체 등으로 임의의 타입에 대한 1차원 배열이라고 봐도 무방하다.

```
import pandas as pd
dat = pd.Series([1,3,6,12])
dat
```



0	1
1	3
2	6
3	12

dtype: int64

레이블을 별도로 지정하지 않았을 때,
자동으로 0부터 시작하는 정수가 라인
레이블로 할당된다.

Pandas의 Data Type – Series

■ Series

- Series로 저장된 데이터의 내용은 다음과 같이 values로 확인할 수 있다.

```
dat.values
```

```
array([ 1, 3, 6, 12], dtype=int64)
```

- 앞서 설명했듯, Series는 다양한 객체를 데이터로 저장할 수 있다. 또한, 모든 요소가 같은 데이터 타입일 필요도 없다.

```
import numpy as np
dat2 = pd.Series(np.array([1,3,np.nan,12]))
dat3 = pd.Series(['aa', 'bb', 'c', 'd'])
dat4 = pd.Series([1, 'aa', 2.34, 'd'])
```

Pandas의 Data Type – Series

■ Series

- 아래와 같이 다양한 방법으로 레이블을 지정해줄 수 있다.

```
dat5 = pd.Series([1,3,6,12], index=[1, 10, 20, 33])
dat6 = pd.Series([1,3,6,12], index=['a','b','c','a'])
dat7 = pd.Series({'a': 1, 'b':3, 'c':6, 'd':12})
print(dat5)
print(dat6)
print(dat7)
```

1	1
10	3
20	6
33	12
dtype: int64	

a	1
b	3
c	6
a	12
dtype: int64	

a	1
b	3
c	6
d	12
dtype: int64	

- 다음과 같이 레이블 값을 변경하는 것도 가능하다.

```
dat2.index = ['un', 'due', 'trois', 'quatre']
```

Pandas의 Data Type – Data Frame

■ DataFrame

- DataFrame은 레이블을 갖는 2차원 데이터 구조이다.
- NumPy로부터 2차원 ndarray를 그대로 입력하는 것이 가능하다.

```
df=pd.DataFrame(np.random.randn(6,2),  
                 index=[[1,1,2,2,3,3],[1,2,1,2,1,2]],  
                 columns=['item1','item2'])
```

- 시리즈와 마찬가지로 레이블 입력이 가능하며, 2차원이므로 라인 레이블과 컬럼 레이블을 지정할 수 있다.
- DataFrame이 가장 많이 사용

(code) Use – DataFrame

■ 데이터 설명하기

- head메서드로 처음 몇 개 행 확인
- tail 메서드로 마지막 몇 개의 행 확인

```
dataframe.head(2)
```

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.0	female	1	1
1	Allison, Miss Helen Loraine	1st	2.0	female	0	1

```
dataframe.tail(3)
```

	Name	PClass	Age	Sex	Survived	SexCode
1310	Zenni, Mr Philip	3rd	22.0	male	0	0
1311	Lievens, Mr Rene	3rd	24.0	male	0	0
1312	Zimmerman, Leo	3rd	29.0	male	0	0

Use – DataFrame

■ 데이터 설명하기

- shape 메서드로 행과 열의 수 확인
- describe 메서드로 수치형 열의 기본 통계값 확인

```
# 차원을 확인합니다.  
dataframe.shape
```

(1313, 6)

```
dataframe.describe()
```

	Age	Survived	SexCode
count	756.000000	1313.000000	1313.000000
mean	30.397989	0.342727	0.351866
std	14.259049	0.474802	0.477734
min	0.170000	0.000000	0.000000
25%	21.000000	0.000000	0.000000
50%	28.000000	0.000000	0.000000
75%	39.000000	1.000000	1.000000
max	71.000000	1.000000	1.000000

Use – DataFrame

■ 탐색하기

- loc나 iloc 메서드를 사용해 하나 이상의 행이나 값을 선택
- loc는 데이터프레임의 인덱스가 레이블(예를 들어 문자열)일 때 사용. 마지막 인덱스 포함
- iloc는 데이터프레임의 위치를 참조. 예를 들어 iloc[0]은 정수 혹은 문자열 인덱스에 상관없이 첫번째 행을 반환
- 데이터 정제 단계에서 자주 등장하므로 loc메서드나 iloc메서드에 익숙해지는 것이 좋음

Use – DataFrame

■ 탐색하기

- loc나 iloc 메서드를 사용해 하나 이상의 행이나 값을 선택
- loc는 데이터프레임의 인덱스가 레이블(예를 들어 문자열)일 때 사용. 마지막 인덱스 포함
- iloc는 데이터프레임의 위치를 참조. 예를 들어 iloc[0]은 정수 혹은 문자열 인덱스에 상관없이 첫번째 행을 반환
- 데이터 정제 단계에서 자주 등장하므로 loc메서드나 iloc메서드에 익숙해지는 것이 좋음

Use – DataFrame

■ 탐색하기

- loc나 iloc 메서드를 사용해 하나 이상의 행이나 값을 선택

```
# 첫 번째 행을 선택합니다.
```

```
dataframe.iloc[0]
```

```
Name      Allen, Miss Elisabeth Walton
PClass      1st
Age         29
Sex         female
Survived      1
SexCode      1
Name: 0, dtype: object
```

```
# 세 개의 행을 선택합니다.
```

```
dataframe.iloc[1:4]
```

	Name	PClass	Age	Sex	Survived	SexCode
1	Allison, Miss Helen Loraine	1st	2.0	female	0	1
2	Allison, Mr Hudson Joshua Creighton	1st	30.0	male	0	0
3	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.0	female	0	1

```
# 인덱스를 설정합니다.
```

```
dataframe = dataframe.set_index(dataframe['Name'])
```

```
# 행을 확인합니다.
```

```
dataframe.loc['Allen, Miss Elisabeth Walton']
```

```
Name      Allen, Miss Elisabeth Walton
PClass      1st
Age         29
Sex         female
Survived      1
SexCode      1
Name: Allen, Miss Elisabeth Walton, dtype: object
```

Use – DataFrame

■ 탐색하기

- loc는 마지막 인덱스 포함

```
# 네 개의 행을 선택합니다.  
dataframe.loc[1:4]
```

	Name	PClass	Age	Sex	Survived	SexCode
1	Allison, Miss Helen Loraine	1st	2.00	female	0	1
2	Allison, Mr Hudson Joshua Creighton	1st	30.00	male	0	0
3	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	female	0	1
4	Allison, Master Hudson Trevor	1st	0.92	male	1	0

Use – DataFrame

■ 탐색하기

- 데이터프레임 객체에 슬라이싱 사용하면 행을 선택. 인덱싱을 사용하면 열을 선택

```
# dataframe[:2]와 동일합니다.  
dataframe[:'Allison, Miss Helen Loraine']
```

	Name	PClass	Age	Sex	Survived	SexCode
	Name					
	Allen, Miss Elisabeth Walton	Allen, Miss Elisabeth Walton	1st	29.0	female	1
	Allison, Miss Helen Loraine	Allison, Miss Helen Loraine	1st	2.0	female	0

```
dataframe[['Age', 'Sex']].head(2)
```

	Age	Sex
	Name	
	Allen, Miss Elisabeth Walton	29.0 female
	Allison, Miss Helen Loraine	2.0 female

Use – DataFrame

■ 조건에 따라 행 선택

■ 여성 승객만 선택

```
# 'sex' 열이 'female' 인 행 중 처음 두 개를 출력합니다.  
dataframe[dataframe['Sex'] == 'female'].head(2)
```

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.0	female	1	1
1	Allison, Miss Helen Loraine	1st	2.0	female	0	1

■ 여성이면서 65세 이상인 승객만 선택

```
# 행을 필터링합니다.  
dataframe[(dataframe['Sex'] == 'female') & (dataframe['Age'] >= 65)]
```

	Name	PClass	Age	Sex	Survived	SexCode
73	Crosby, Mrs Edward Gifford (Catherine Elizabet...	1st	69.0	female	1	1

Use – DataFrame

- 조건에 따라 행 선택
 - 문자열 다룰 때는 str 메서드 이용
 - str.find는 특정 문자 포함하는 열 추출.
문자열 포함되어있으면 열 인덱스 반환.
포함되어 있지 않을 경우 -1 반환

```
# Name 열에 Allison이 포함된 행만 찾기  
dataframe['Name'].str.find('Allison')
```

```
0      -1  
1       0  
2       0  
3       0  
4       0
```

```
..  
1308    -1  
1309    -1  
1310    -1  
1311    -1  
1312    -1
```

```
Name: Name, Length: 1313, dtype: int64
```

Use – DataFrame

- 조건에 따라 행 선택
 - 다음과 같은 방식으로 특정 문자열 포함한 행 선택 가능

```
dataframe[dataframe['Name'].str.find('Allison')>-1].head(3)
```

	Name	PClass	Age	Sex	Survived	SexCode
1	Allison, Miss Helen Loraine	1st	2.0	female	0	1
2	Allison, Mr Hudson Joshua Creighton	1st	30.0	male	0	0
3	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.0	female	0	1

Use – DataFrame

■ 기초통계 구하기

■ 최댓값, 최솟값, 평균, 합 계산 및 개수 세기

```
# 통계를 계산합니다.  
print('최댓값:', dataframe['Age'].max())  
print('최솟값:', dataframe['Age'].min())  
print('평균:', dataframe['Age'].mean())  
print('합:', dataframe['Age'].sum())  
print('카운트:', dataframe['Age'].count())
```

최댓값: 71.0
최솟값: 0.17
평균: 30.397989417989415
합: 22980.88
카운트: 756

- 이 외에도 분산(var), 표준편차(std), 첨도(kurt), 비대칭도(왜도)(skew), 평균의 표준오차(sem), 최빈값(mode), 중간값(median)을 포함하여 많은 메서드 제공

Use – DataFrame

- 기초통계 구하기
 - 데이터프레임 전체에 대해서도 적용가능

```
# 카운트를 출력합니다.  
dataframe.count()
```

```
Name      1313  
PClass     1313  
Age        756  
Sex        1313  
Survived   1313  
SexCode    1313  
dtype: int64
```


Dataframe 정렬하기

- `sort_index`: 인덱스(행) 따라 정렬
- `sort_values`: 값을 기준으로 정렬
- `groupby()`: 같은 값을 하나로 묶어 통계 결과(평균, max, min, ...) 를 얻기 위해 사용

```
df = pd.DataFrame(np.round(np.random.randn(7, 3) * 10),  
                  columns=["AAA", "BBB", "CCC"],  
                  index=list("defcabg"))  
df
```

	AAA	BBB	CCC
d	-6.0	-4.0	-1.0
e	5.0	-4.0	-5.0
f	4.0	7.0	-1.0
c	-9.0	-10.0	13.0
a	-6.0	11.0	-9.0
b	-17.0	16.0	24.0
g	14.0	-1.0	-3.0



```
df.sort_index() #기본: 오름차순
```

	AAA	BBB	CCC
a	5.0	-16.0	3.0
b	-18.0	2.0	-8.0
c	5.0	-4.0	-9.0
d	0.0	0.0	13.0
e	10.0	10.0	-12.0
f	11.0	-3.0	-6.0
g	4.0	2.0	11.0

열따라 정렬, 내림차순

```
df.sort_index(axis=1, ascending=False)
```

	CCC	BBB	AAA
d	13.0	0.0	0.0
e	-12.0	10.0	10.0
f	-6.0	-3.0	11.0
c	-9.0	-4.0	5.0
a	3.0	-16.0	5.0
b	-8.0	2.0	-18.0
g	11.0	2.0	4.0

Dataframe 정렬하기

- `sort_index`: 인덱스(행) 따라 정렬
- `sort_values`: 값을 기준으로 정렬
- `groupby()`: 같은 값을 하나로 묶어 통계 결과(평균, max, min, ...) 를 얻기 위해 사용

```
df = pd.DataFrame(np.round(np.random.randn(7, 3) * 10),  
                  columns=["AAA", "BBB", "CCC"],  
                  index=list("defcabg"))  
df
```

	AAA	BBB	CCC
d	-6.0	-4.0	-1.0
e	5.0	-4.0	-5.0
f	4.0	7.0	-1.0
c	-9.0	-10.0	13.0
a	-6.0	11.0	-9.0
b	-17.0	16.0	24.0
g	14.0	-1.0	-3.0



```
df.sort_values(by='AAA')
```

	AAA	BBB	CCC
b	-18.0	2.0	-8.0
d	0.0	0.0	13.0
g	4.0	2.0	11.0
c	5.0	-4.0	-9.0
a	5.0	-16.0	3.0
e	10.0	10.0	-12.0
f	11.0	-3.0	-6.0

```
df.sort_values(by=['BBB', 'CCC'])
```

	AAA	BBB	CCC
a	5.0	-16.0	3.0
c	5.0	-4.0	-9.0
f	11.0	-3.0	-6.0
d	0.0	0.0	13.0
b	-18.0	2.0	-8.0
g	4.0	2.0	11.0
e	10.0	10.0	-12.0

Dataframe 정렬하기

- `sort_index`: 인덱스(행) 따라 정렬
- `sort_values`: 값을 기준으로 정렬
- `groupby()`: 같은 값을 하나로 묶어 통계 결과(평균, max, min, ...) 를 얻기 위해 사용
 - `groupby` 자체 인덱스를 사용하고 싶은 않은 경우에는 `as_index=False` 를 설정하면 됨

	city	fruits	price	quantity
0	부산	apple	100	1
1	부산	orange	200	2
2	부산	banana	250	3
3	부산	banana	300	4
4	서울	apple	150	5
5	서울	apple	200	6
6	서울	banana	400	7



```
df.groupby('city', as_index=False).mean()
```

	city	price	quantity
0	부산	212.5	2.5
1	서울	250.0	6.0

```
df.groupby(['city', 'fruits'], as_index=False).mean()
```

	city	fruits	price	quantity
0	부산	apple	100.0	1.0
1	부산	banana	275.0	3.5
2	부산	orange	200.0	2.0
3	서울	apple	175.0	5.5
4	서울	banana	400.0	7.0

```
df.groupby(['city', 'fruits'], as_index=False).mean()
```

	city	fruits	price	quantity
0	부산	apple	100.0	1.0
1	부산	banana	275.0	3.5
2	부산	orange	200.0	2.0
3	서울	apple	175.0	5.5
4	서울	banana	400.0	7.0

df.apply

- 모든 열 원소에 함수 적용
 - apply를 사용하여 열의 모든 원소에 내장 함수나 사용자 정의 함수를 적용
 - apply와 유사한 map메서드도 존재 (map은 딕셔너리를 입력으로 넣을 수 있음)
 - apply 메서드는 매개변수를 지정할 수 있음

```
# 함수를 만듭니다.
```

```
def uppercase(x):  
    return x.upper()
```

```
# 함수를 적용하고 두 개의 행을 출력합니다.
```

```
dataframe['Name'].apply(uppercase)[0:2]
```

```
0    ALLEN, MISS ELISABETH WALTON  
1    ALLISON, MISS HELEN LORAINE  
Name: Name, dtype: object
```

```
# 함수의 매개변수(age)를 apply 메서드를 호출할 때 전달할 수 있습니다.  
dataframe['Age'].apply(lambda x, age: x < age, age=30)[:5]
```

```
0    True  
1    True  
2   False  
3    True  
4    True  
Name: Age, dtype: bool
```

Dataframe 연결하기

■ 데이터프레임 연결

- concat함수에 axis=0 매개변수를 설정하여 행의 축을 따라 연결
- axis=1을 사용하면 열의 축을 따라 연결

dataframe_a			
	id	first	last
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali

dataframe_b			
	id	first	last
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner



```
# 행 방향으로 데이터프레임을 연결합니다.  
pd.concat([dataframe_a, dataframe_b], axis=0)
```

	id	first	last
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner

```
# 열 방향으로 데이터프레임을 연결합니다.  
pd.concat([dataframe_a, dataframe_b], axis=1)
```

	id	first	last	id	first	last
0	1	Alex	Anderson	4	Billy	Bonder
1	2	Amy	Ackerman	5	Brian	Black
2	3	Allen	Ali	6	Bran	Balwner

dropna, fillna

■ 데이터프레임의 결측치(NaN, missing value)를 다룰 때 주로 사용

- Pandas는 NaN(누락값)에 대해 적절한 처리를 제공한다.
- NaN이 존재하는 모든 **행**을 삭제할 수 있다.
- NaN이 존재하는 모든 **열**을 삭제할 수도 있다.
- 특정 행이나 열에서 NaN을 제거하고 싶다면 아래와 같이 처리 가능하다.

```
#A, B열에 NaN이 포함된 행을 삭제  
data_frame.dropna(subset=['A', 'B'])
```

	A	B	C	D
1	3.0	4.0	NaN	1
3	3.0	4.0	NaN	1
4	3.0	4.0	0.0	1

```
#2행과 4행에 NaN이 포함된 열을 삭제  
data_frame.dropna(axis=1, subset=[2, 4])
```

	D
0	0
1	1
2	5
3	1
4	1

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5
3	3.0	4.0	NaN	1
4	3.0	4.0	0.0	1



```
data_frame.dropna()
```

	A	B	C	D
4	3.0	4.0	0.0	1

```
data_frame.dropna(axis=1)
```

	D
0	0
1	1
2	5
3	1
4	1

dropna, fillna

- NaN을 다른 값으로 모두 대체할 때에는 fillna 사용

```
#모든 NaN을 0으로 치환한다  
data_frame.fillna(0)
```

	A	B	C	D
0	0.0	2.0	0.0	0
1	3.0	4.0	0.0	1
2	0.0	0.0	0.0	5
3	3.0	4.0	0.0	1
4	3.0	4.0	0.0	1

- isnull()은 특정 위치의 값이 NaN이면 true, 아니면 false 반환

```
data_frame.isnull()
```

	A	B	C	D
0	True	False	True	False
1	False	False	True	False
2	True	True	True	False
3	False	False	True	False
4	False	False	False	False

*참고

```
#NaN을 같은 열의 바로 위의 행 값으로 대체  
data_frame.fillna(method='ffill')
```

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	3.0	4.0	NaN	5
3	3.0	4.0	NaN	1
4	3.0	4.0	0.0	1

```
#NaN을 같은 열의 바로 아래 행 값으로 대체  
data_frame.fillna(method='bfill')
```

	A	B	C	D
0	3.0	2.0	0.0	0
1	3.0	4.0	0.0	1
2	3.0	4.0	0.0	5
3	3.0	4.0	0.0	1
4	3.0	4.0	0.0	1