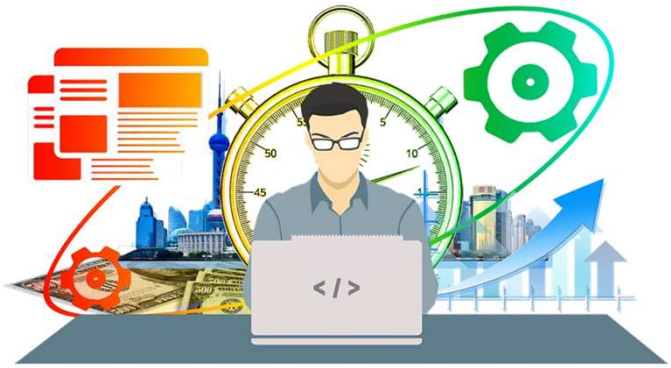
파이썬 라이브러리(전처리)



“In a day, when you don’t come across any problems,
you can be sure that you are traveling in a wrong way!”

파이썬 라이브러리(전처리)

1. **넘파이**(Numpy)

1. 개요

- 리스트 장점과 한계, 그리고 Numpy
- 목표
- 라이브러리 불러오기
- 용어정의

리스트 장점과 한계, 그리고 Numpy

- 리스트
  - 값의 집합(collection)
  - 다른 타입의 데이터도 한꺼번에 저장 가능
  - 요소 변경, 추가, 제거가 용이함
- 그러나, 데이터 분석에서의 필요는...
  - 단순히 값의 집합 개념을 넘어선 수학적 계산이 가능해야 함
  - 대량의 데이터를 처리하는데 처리 속도가 빨라야 함
- Numpy
  - Numerical Python의 줄임말
  - 빠른 수치 계산을 위해 C언어로 만들어진 Python 라이브러리
  - 벡터와 행렬 연산에 위해 매우 편리한 기능들을 제공
  - 데이터분석용 라이브러리인 Pandas와 Matplotlib의 기반으로 사용되기도 함
  - Array(행렬 개념)라는 단위로 데이터를 관리하고 연산을 수행

라이브러리 불러오기

- Numpy 라이브러리를 불러와(Import) 사용해야 함

Python
```
# 별칭 없이 라이브러리 불러오기
import numpy

a = numpy.array([1, 2, 3, 4, 5])
```

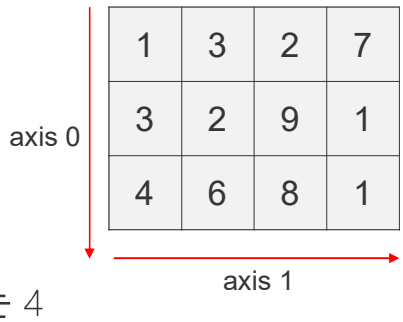
Python
```
# 별칭을 주고 라이브러리 불러오기
import numpy as np

a = np.array([1, 2, 3, 4, 5])
```

Python
```
# 특정 함수만 불러오기
from numpy import array

a = array([1, 2, 3, 4, 5])
```

용어 정의

- Axis: 배열의 각 축
- Rank: 축의 개수
- Shape: 축의 길이

- [3 x 4 형태인 배열의 경우]
- axis 0 과 axis 1 을 갖는 2차원 배열
- Rank 2 Array
- 첫번째 축의 길이는 3, 두번째 축의 길이는 4
- Shape는 (3, 4)



2. 배열 만들기

- 1, 2, 3차원 배열 만들기
- 배열에 대한 이해
- Reshape
- 배열을 만드는 여러 함수들

1, 2, 3차원 배열 만들기 - 1, 2차원 배열

- 1차원 배열 만들기
  - [...] 형대
  - 대괄호 한 겹
- 2차원 배열 만들기
  - [[...], [...]] 형태
  - 대괄호 두 겹
- 다른 자료형에서 변환
  - 리스트, 튜플: 직접 변환
  - 문자열, 딕셔너리, 집합: 자료형 변환 후 변환 가능

Python

```
a = np.array([1, 2, 3, 4, 5])

print(a)          # [1 2 3 4 5]
print(type(a))    # <class 'numpy.ndarray'>
print(a.ndim)     # 1      <-- 차원
print(a.shape)    # (5, )  <-- 모양
print(a.dtype)    # int32  <-- 데이터 형식
print(a[0], a[1], a[2]) # 1 2 3
```

Python

```
a = np.array([[1.5, 2.5, 3.2],
              [4.2, 5.7, 6.4]])

print(a)
print(a.ndim)     # 2
print(a.shape)    # (2, 3)
print(a.dtype)    # float64
```

1, 2, 3차원 배열 만들기 - 자료형 변환

- 리스트, 튜플: 직접 배열로 변환 가능
- 문자열, 딕셔너리, 집합: 다른 자료형으로 변환 후 배열로 변환 가능

Python

```
# 문자열 --> 자료형 변환 --> 2차원 배열
a = np.array([list('python'), list('flower')])

# 딕셔너리 --> 키, 값 분해 --> 자료형 변환 --> 2차원 배열
dic = {'a':1, 'b':2, 'c':3}
a = np.array([list(dic.keys()), list(dic.values())])

# 집합 --> 자료형 변환 --> 2차원 배열
st = {1, 2, 3}
a = np.array([list(st), list(st)])
```

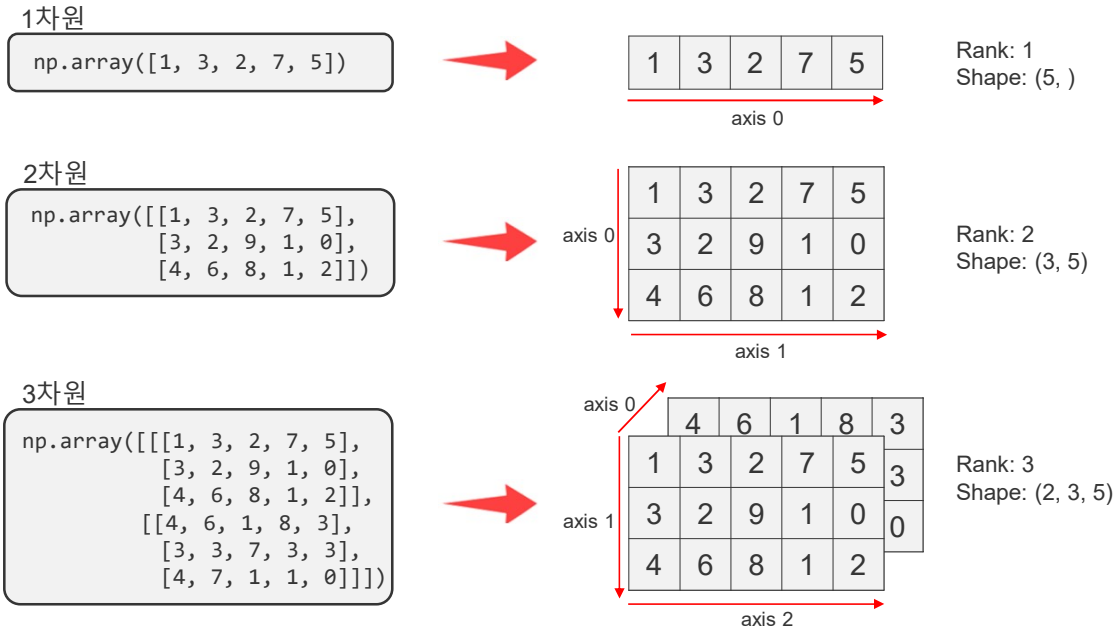
1, 2, 3차원 배열 만들기 - 3차원 배열

- 3차원 배열 만들기
  - [[[...],[...]].[[...],[...]]] 형태
  - 대괄호 세 겹

Python

```
a = np.array([[[1, 3, 1],
               [4, 7, 6],
               [8, 3, 4]],
              [[6, 2, 4],
               [8, 1, 5],
               [3, 5, 9]]])

print(a)
print(a.ndim)      # 3
print(a.shape)     # (2, 3, 3)
print(a.dtype)     # int64
```

배열에 대한 이해



Reshape

- 기존 배열을 새로운 형태(Shape)의 배열로 다시 구성
- 배열 요소가 사라지지 않는 형태라면 자유롭게 변환 가능
- 3 x 4 → 2 x 6 → 4 X 3 → 12 x 1 → 6 x 2

Python

```
# 2 x 3 형태의 2차원 배열 만들기
a = np.array([[1, 2, 3],
              [4, 5, 6]])

# 함수를 사용해 6 x 1 형태의 2차원 배열로 Reshape
b = np.reshape(a, (6, 1))

# 메소드를 사용해 3 x 2 형태의 2차원 배열로 Reshape
c = a.reshape(3, 2)
```

배열 a

1	2	3
4	5	6

배열 b

1
2
3
4
5
6

배열 c

1	2
3	4
5	6

- a.reshape(3, -1) → a를 3행으로 된 배열로 변환
- a.reshape(-1, 2) → a를 2열로 된 배열로 변환

Reshape - MNIST

- MNIST Sample 사용 예
  - 100 x 784 형태의 2차원 배열을
  - 100 x 28 x 28 형태의 3차원 배열로 Reshape
  - 즉, 100개의 28 x 28 픽셀을 갖는 이미지 정보가 됨
  - matplotlib 라이브러리를 사용해 이미지 출력

```
(100, 784)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

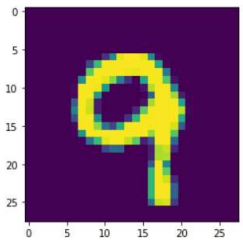
```
(100, 28, 28)
[[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0
```

- 이미지 번호 99(숫자 9) 출력 예

Python

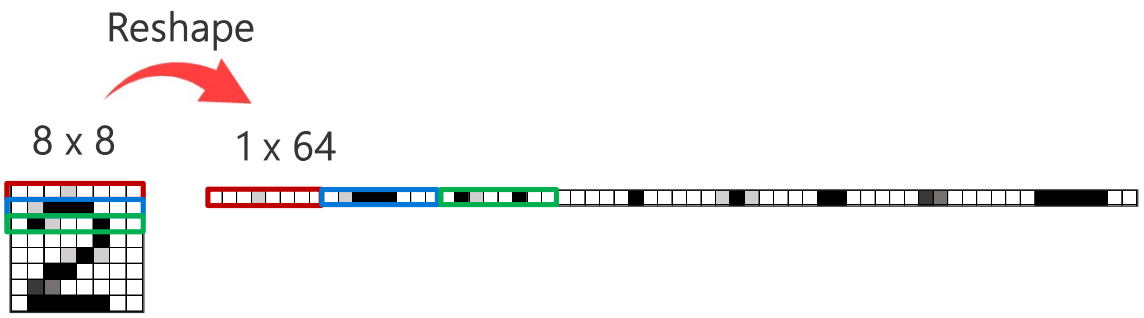
```
import matplotlib.pylab as plt

image_no = 99
plt.imshow(mnist_pixels2[image_no, : , : ])
```

```
<matplotlib.image.AxesImage at 0x7fec27a0c390>
```



Reshape - 계속

- 이미지 분석



배열을 만드는 여러 함수들

- np.zeros( )
  - 0으로 채워진 배열
- np.ones( )
  - 1로 채워진 배열
- np.full( )
  - 특정 값으로 채워진 배열
- np.eye( )
  - 정방향 행렬
- np.random.random( )
  - 랜덤 값으로 채운 배열

Python

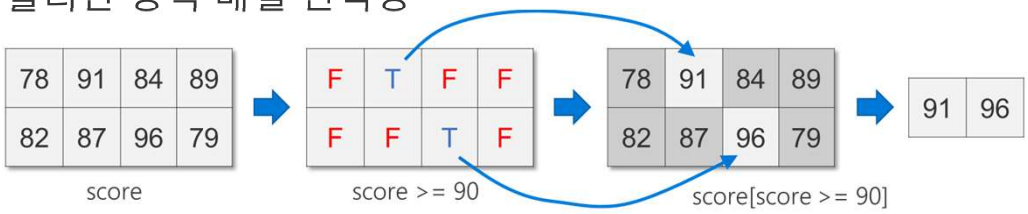
```
# 0으로 채워진 배열
a = np.zeros((2, 2))
print(a)                # [[ 0.  0.]
                        #  [ 0.  0.]]
# 1로 채워진 배열
b = np.ones((1, 2))
print(b)                # [[ 1.  1.]]

# 특정 값으로 채워진 배열
c = np.full((2, 2), 7.)
print(c)                # [[ 7.  7.]
                        #  [ 7.  7.]]
# 2x2 단위 행렬(identity matrix)
d = np.eye(2)
print(d)                # [[ 1.  0.]
                        #  [ 0.  1.]]
# 랜덤값으로 채운 배열
e = np.random.random((2, 2))
print(e)
```

3.배열 데이터 조회

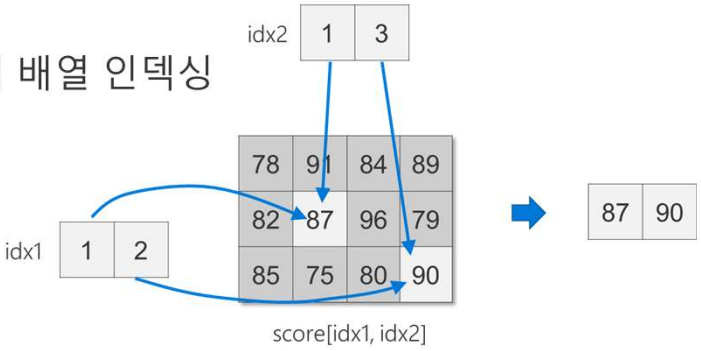
- 팬시 인덱싱
- 불리안 방식 배열 인덱싱
- 정수 방식 배열 인덱싱
- 배열 인덱싱과 슬라이싱 정리

팬시 인덱싱

- 불리안 방식 배열 인덱싱



- 정수 방식 배열 인덱싱



불리안 방식 배열 인덱싱

- 1차원 배열

Python
```
Score = np.array([78, 91, 84, 89, 93, 65])

Score >= 90
# array([False,True,False,False,True,False], dtype=bool)

Score[Score >= 90]                  # array([91, 93])
```

- 2차원 배열

Python
```
Score_2d = np.array([[78, 91, 84, 89, 93, 65],
                     [82, 87, 96, 79, 91, 73]])

score_2d[0] >= 90
# array([False,  True, False, False,  True, False])

score_2d[0][score_2d[0] >= 90]   # array([91, 93])

score_2d[1][score_2d[1] >= 90]   # array([96, 91])
```

정수 방식 배열 인덱싱

Python
```
# 2 x 3 형태의 2차원 배열 만들기
a = np.array([[1, 2],
              [3, 4],
              [5, 6]])

# 0, 1, 2행 가져와 2차원 배열 만들기
b = a[[0, 1, 2]]

# 0, 2행 가져와 2차원 배열 만들기
c = a[[0, 2]]

# 0행 가져와 2차원 배열 만들기
d = a[[0]]

# 다음 결과는?
f = a[[0, 1, 2], [0, 1, 0]]

# 같은 위치 데이터 여러 번 사용가능
g = a[[0, 0], [1, 1]]
```

배열 a

1	2
3	4
5	6

배열 b

1	2
3	4
5	6

배열 c

1	2
5	6

배열 d

1	2

배열 f

1	4	5

배열 g

2	2

정수 방식 배열 인덱싱 - 계속

Python

```
# 4 x 3 형태의 2차원 배열 만들기
a = np.array([[ 1,  2,  3],
              [ 4,  5,  6],
              [ 7,  8,  9],
              [10, 11, 12]])

# 두 개의 1차원 배열 만들기
b = np.arange(4)
c = np.array([0, 2, 0, 1])

# 다음 코드의 실행 결과는?
d = a[b, c]

# 위 코드는 다음과 같은 의미
d = a[[0, 1, 2, 3], [0, 2, 0, 1]]

# 선택된 모든 값을 변경
a[b, c] += 10
```

배열 인덱싱과 슬라이싱 정리

- 인덱스 번호로 가져오기
  - np_array[#, #]
    - 하나의 값을 가져옴
  - np_array[[#, #]]
    - 원래 차원의 배열이 됨
- 범위로 가져오기
  - np_array[0:2, 2:4]
  - 원래 차원 배열이 됨
- 인덱스와 범위로 가져오기
  - np_array[1, 2:4]
  - 더 낮은 차원의 배열이 됨

Python

```
a = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])


# 인덱스로 가져오기
b = a[1, 1]
print(b)                        # 6

c = a[[0, 1, 2], [0, 1, 1]]
print(b)                        # [[1 6 9]]

# 범위를 사용하면 원래 차원 배열을 가져옴
d = a[1:2, :]                   # Rank 2
Print(d)                        # [[5 6 7 8]]

# 인덱스와 범위를 섞으면 더 낮은 차원의 배열
e = a[1, :]                     # Rank 1
print(e)                        # [5 6 7 8]
```

4. 배열 변환과 연산

- 배열 변환
- 기본 연산(사칙연산)
- 리스트와 다른 점
- 행렬 연산
- Shuffle, Sampling, Split

배열 변환

- 숫자간 변환
  - 실수가 하나라도 포함되면 실수
  - 필요시 배열 만들 때 dtype 지정
  - 기존 배열의 데이터 형식은 astype 메소드로 변경

Python

```
a = np.array([1, 2])
print(a.dtype)    # int64

a = np.array([1.0, 2.0])
print(a.dtype)    # float64

a = np.array([1.9, 8.0], dtype=np.int32)
print(a.dtype)    # int32
```

- 배열 → 리스트 변환
  - .tolist() 메소드 사용을 권고
  - 1차원 배열은 list() 함수 사용 가능
  - 2차원 배열은 tolist() 메소드와 list 함수의 결과가 다름

Python

```
# 배열 만들기
x = np.array([1, 2])
print(type(x))    # <class 'numpy.ndarray'>

# 배열 --> 리스트
y = x.tolist()    # 종종 사용됨
print(y)          # [1, 2]
Print(type(y))    # <class 'list'>
```

기본 연산(사칙연산)

더하기: + 또는 np.add

1	2	+	5	6	=	6	8
3	4		7	8		10	12

빼기: - 또는 np.subtract

1	2	-	5	6	=	-4	-4
3	4		7	8		-4	-4

곱하기: * 또는 np.multiply

1	2	*	5	6	=	5	12
3	4		7	8		21	32

나누기: / 또는 np.divide

1	2	/	5	6	=	0.2	0.333
3	4		7	8		0.428	0.5

제곱: ** 또는 np.power

1	2	**	5	6	=	1	64
3	4		7	8		2187	65536

제곱근: np.sqrt

1	2	제곱근 =	1	1.414
3	4		1.732	2

리스트와 다른 점

- 리스트 사이에는 연산이 안되는 경우가 많음

Python

```
height = [1.73, 1.68, 1.71, 1.89, 1.79]
weight = [65.4, 59.2, 63.6, 88.4, 68.7]
weight / height ** 2              # TypeError: unsupported operand type(s) for **

np_height = np.array(height)
np_weight = np.array(weight)
np_weight / np_height ** 2        # array([ 21.852, 20.975, 21.75 , 24.747, 21.441])
```

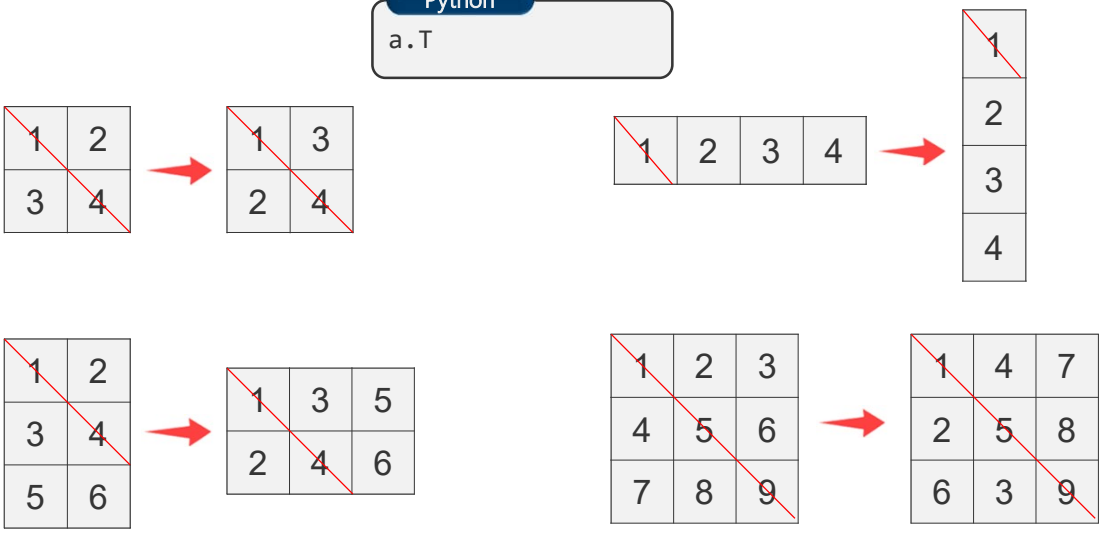
- 동일한 연산자에 대한 작동방식이 다름

Python

```
python_list = [1, 2, 3]
numpy_array = np.array([1, 2, 3])

python_list + python_list         # [1, 2, 3, 1, 2, 3]

numpy_array + numpy_array         # array([2, 4, 6])
```

전치 행렬

Python
```
a.T
```



※ 전치행렬: 주 대각선을 축으로 하는 반사 대칭을 가하여 얻는 행렬(위키백과)

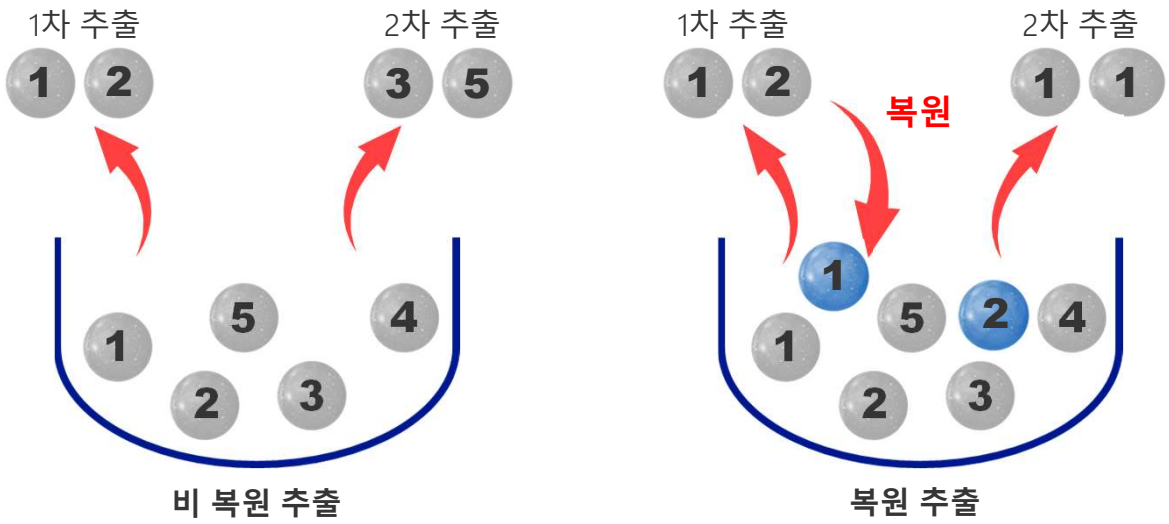
Shuffle, Sampling, Split

- Shuffle: 기존 데이터들 무작위로 섞기
- Sampling: 임의의 데이터 추출하기(복원, 비복원 추출)
- Split: 데이터 분할하기

Python
```
data = np.genfromtxt('./Graduate_apply.csv', delimiter=',', names=True)

# 데이터 섞기
np.random.shuffle(data)

# 임의의 4개 추출(비 복원 예)
np.random.choice(data, 4, replace = False)

# 분할하기
s1, s2, s3, s4 = data[:100], data[100:200], data[200:300], data[300:]
```

복원 추출과 비 복원 추출

- 임으로 꺼낸 값을 다시 넣을지 말지에 대한 선택



파이썬 라이브러리(전처리)

2. **판다스**(Pandas)

- 개요
- Pandas 시리즈
- Pandas 데이터프레임

1. 개요

- NumPy의 한계와 Pandas의 장

NumPy의 한계와 Pandas의 장점

- Numpy
  - 오직 하나의 데이터 형식
- Pandas
  - 열(Column)별 데이터 형식 지정 → 테이블 형태(데이터프레임)의 데이터 구조
  - Missing Value 처리 가능
  - 조인과 같은 Relational operations 지원
  - Time Series 기능

2. Pandas **시리즈**

- 시리즈 이해
- 시리즈 만들기
- CSV파일에서 데이터 불러오기
- 인덱스 지정
- 시리즈 인덱싱과 슬라이싱
- 시리즈 연산
- 차트 그리기

시리즈와 데이터프레임

- 시리즈는 하나의 값(=변수)을, 데이터프레임은 여러 값을 가짐

	A
0	10
1	15
2	21
3	20
4	19

Series

+

	B
0	20
1	28
2	35
3	26
4	20

Series

=

	A	B
0	10	20
1	15	28
2	21	35
3	20	26
4	19	20

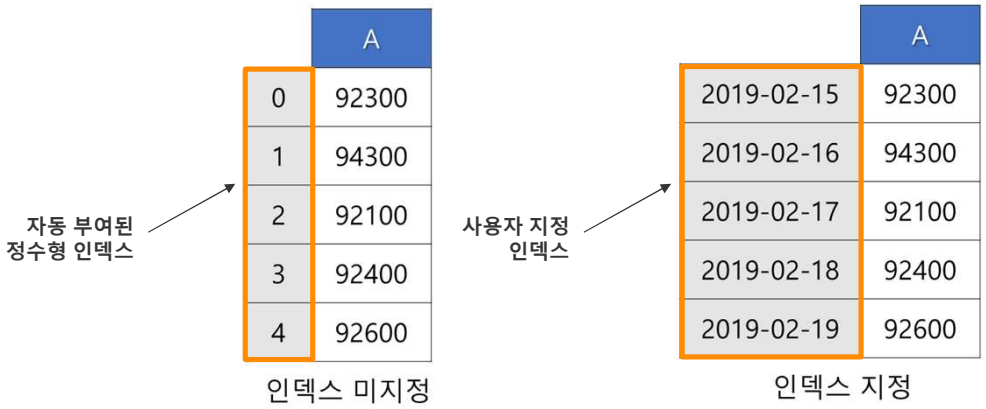
DataFrame

- 시리즈와 데이터프레임 사용을 위해 Pandas 라이브러리가 필요함
- 라이브러리를 불러올 때 일반적으로 pd 별칭을 사용함

Python
```
import pandas as pd
```

시리즈 이해

- 데이터프레임과 더불어 Pandas가 제공하는 자료구조 중 하나
- 인덱스와 값으로 구성되어 키와 값으로 구성되는 딕셔너리와 유사
- 객체 정보, 일부 데이터, 집계 결과가 시리즈로 저장되는 경우 많음



시리즈 이해

- 배열(Array), 리스트(List)와 비교

Python

```
# Pandas Series
stock = pd.Series([92300, 94300, 92100, 92400, 92600])
print(stock1[1:])

# Numpy Rank 1 Array
np_stock = np.array([92300, 94300, 92100, 92400, 92600])
print(np_stock[1:])

# Python List
list_stock = [92300, 94300, 92100, 92400, 92600]
print(list_stock[1:])
```

```
0    92300
1    94300
2    92100
3    92400
4    92600
dtype: int64
```

시리즈 만들기

- 스칼라 값, 리스트, 튜플, 딕셔너리, 1차원 배열 → 시리즈
- 집합, 다차원 배열 → 변환 → 리스트
- 리스트 → 시리즈

```
0    92300
1    94300
2    92100
3    92400
4    92600
dtype: int64
```

Python
```
list_stock = [92300, 94300, 92100, 92400, 92600]
stock = pd.Series(list_stock)
```

- 딕셔너리 → 시리즈

```
2019-02-15    92300
2019-02-16    94300
2019-02-17    92100
2019-02-18    92400
2019-02-19    92600
dtype: int64
```

Python
```
dict_stock = {'2019-02-15': 92300,
              '2019-02-16': 94300,
              '2019-02-17': 92100,
              '2019-02-18': 92400,
              '2019-02-19': 92600}
stock = pd.Series(dict_stock)
```

인덱스 지정하기

```
2019-02-15    92300
2019-02-16    94300
2019-02-17    92100
2019-02-18    92400
2019-02-19    92600
dtype: int64
```

- 만들 때 인덱스 지정하기

Python
```
list_stock = [92300, 94300, 92100, 92400, 92600]
dates = ['2019-02-15', '2019-02-16', '2019-02-17', '2019-02-18', '2019-02-19']
stock = pd.Series(list_stock, index=dates)
```

- 기존 인덱스 변경하기

Python
```
stock = pd.Series([92300, 94300, 92100, 92400, 92600])
dates = ['2019-02-15', '2019-02-16', '2019-02-17', '2019-02-18', '2019-02-19']
stock.index=dates
```

- 인덱스 초기화

Python
```
stock.reset_index(drop=True, inplace=True)
```

CSV파일에서 데이터 불러오기

- CSV 파일에서 데이터를 읽어오면 즉시 데이터프레임이 됨
- 데이터프레임에서 ['열이름'] 형태로 조회하면 시리즈가 됨
- [['열이름']] 형태(대괄호를 두 겹)로 조회하면 데이터프레임이 됨

Python
```
# 파일 읽어와 데이터프레임 만들기
f_path = 'csv/Graduate_apply.csv'
df = pd.read_csv(f_path, sep = ',')

# 열 하나를 읽어와 시리즈 만들기
s = df['gre']
s.head()
```

```
0    380
1    660
2    800
3    640
4    520
Name: gre, dtype: int64
```

Python
```
f_path = 'https://raw.githubusercontent.com/Jangrae/files/master/Graduate_apply.csv'
df = pd.read_csv(file_path, sep = ',')
```

시리즈 정보 확인

- 값을 확인하기 전에 우선 시리즈에 대한 정보를 확인해야 함
- 인덱스, 값, 자료형, 크기 확인 필요

Python
```
stock.index
```

Python
```
stock.values
```

Python
```
stock.dtype
```

Python
```
stock.shape
```

- NaN 값(=결측치) 확인
  - NaN 값은 데이터 분석 전에 제거되거나 또는 다른 값으로 채워져야 함
  - 이러한 처리를 위해 NaN 값이 있는지 필히 확인을 해야 함
  - isnull() 메소드 → NaN 값을 True로 표시
  - notnull() 메소드 → NaN 값을 False로 표시

Python
```
stock.isnull()
```

```
2019-02-15    False
2019-02-16    False
2019-02-17    False
2019-02-18    False
2019-02-19    False
2019-02-20     True
dtype: bool
```

시리즈 인덱싱과 슬라이싱

- 다양한 인덱싱과 슬라이싱 방법이 있고 선택은 자유
- 하지만 일관된 .iloc[ ]과 .loc[ ] 사용을 적극 권고함
- 특정 위치의 행 조회할 때 → .iloc[ ] 사용
- 특정 인덱스 값(=인덱스 이름)을 갖는 행 조회 → .loc[ ] 사용
- 데이터프레임 인덱싱과 슬라이싱도 같은 방법을 사용함

조회	행 번호로 찾기	인덱스 이름으로 찾기
특정 값	.iloc[정수]	.loc['열이름']
특정 행	.iloc[[정수]]	.loc[['열이름']]
특정 범위	.iloc[정수:정수]	.loc['열이름' : '열이름']

시리즈 인덱싱과 슬라이싱 예

- 인덱싱
  - [인덱스] 형태로 값 하나만 조회하면 결과가 시리즈가 아닌 스칼라 값이 됨

Python
```
print(stock.iloc[0])
print(stock.loc['2019-02-15'])
```

Python
```
print(stock.iloc[[0, 2]])
print(stock.loc[['2019-02-15', '2019-02-19']])
```

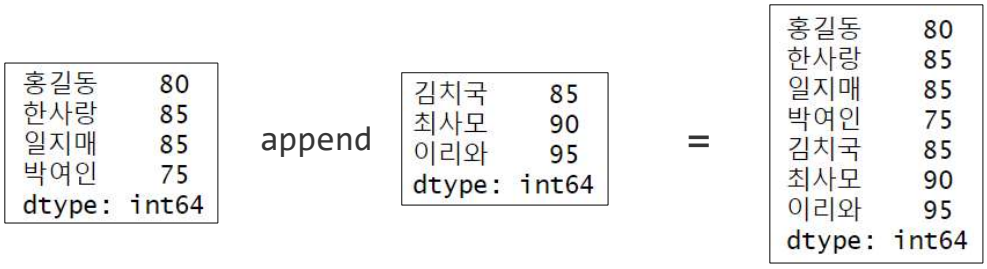
- 슬라이싱

Python
```
print(stock.iloc[0:3])
print(stock.loc['2019-02-15':'2019-02-17'])
```

시리즈 결합

- append() 메소드를 사용해 시리즈를 결합할 수 있음
- ignore_index=True를 지정하면 위치에 따른 정수 인덱스로 초기화

Python
```
score = score.append(score_new)
```

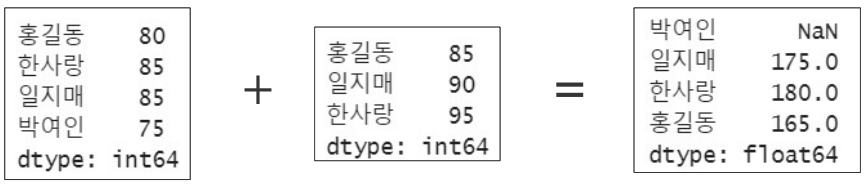


시리즈 연산

- 인덱스 위치가 달라도 인덱스의 값을 기준으로 연산 수행

Python
```
score_math1 = pd.Series([90, 80, 85, 75],
                        index = ['홍길동', '한사랑', '일지매', '박여인'])
score_math2 = pd.Series([85, 90, 95],
                        index = ['홍길동', '일지매', '한사랑'])
```

Python
```
print(score_math1 + score_math2)
```

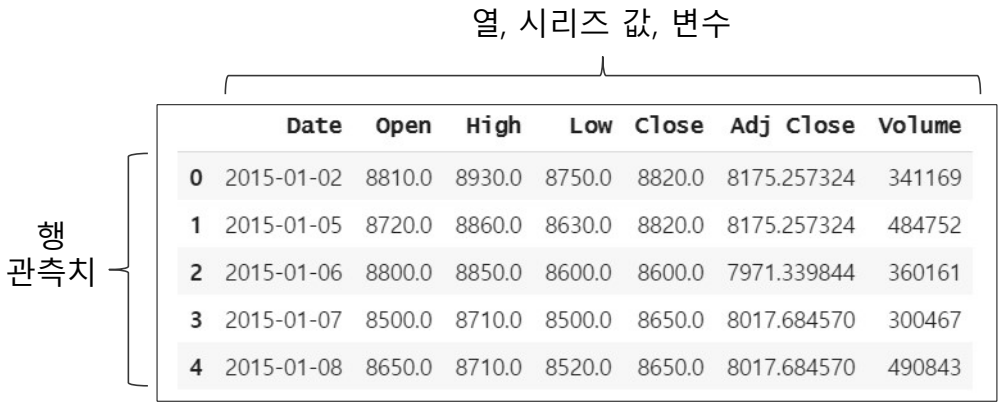


3. Pandas **데이터프레임**

- 데이터프레임 만들기
- CSV 파일에서 데이터 불러오기
- 데이터 살펴보기
- 데이터 조회
- 데이터프레임 변경
- Group By, Join, Rolling & Shift
- NaN 값 처리

데이터프레임 이해

- 데이터프레임(Dataframe)이란?
  - 데이터 분석에서 가장 중요한 데이터 구조
  - 관계형 데이터베이스의 테이블 또는 엑셀 시트와 같은 형태
  - 변수들의 집합 → 각 열을 변수라고 부름



	Date	Open	High	Low	Close	Adj Close	Volume
0	2015-01-02	8810.0	8930.0	8750.0	8820.0	8175.257324	341169
1	2015-01-05	8720.0	8860.0	8630.0	8820.0	8175.257324	484752
2	2015-01-06	8800.0	8850.0	8600.0	8600.0	7971.339844	360161
3	2015-01-07	8500.0	8710.0	8500.0	8650.0	8017.684570	300467
4	2015-01-08	8650.0	8710.0	8520.0	8650.0	8017.684570	490843

데이터프레임 만들기

- 데이터 프레임을 만들 때 다음 세 가지를 우선 지정해야 함
  - 데이터(필수), 인덱스(생략 가능), 열 이름(생략 가능)
- 인덱스, 열 이름을 지정하지 않으면 위치에 기반한 정수가 설정 됨
- 의미 있는 값(날짜가 대표적임)이 인덱스가 되면 분석이 용이

	Name	Level	Score
0	Gildong	Gold	56000
1	Sarang	Bronze	23000
2	Jiemae	Silver	44000
3	Yeoin	Gold	52000

인덱스 미지정

	Name	Level	Score
Std01	Gildong	Gold	56000
Std02	Sarang	Bronze	23000
Std03	Jiemae	Silver	44000
Std04	Yeoin	Gold	52000

인덱스 지정

	Level	Score
Gildong	Gold	56000
Sarang	Bronze	23000
Jiemae	Silver	44000
Yeoin	Gold	52000

일반 열을 인덱스로 지정

데이터프레임 만들기

- DataFrame 함수 사용

Python

```
dict = {
    '이름': ['홍길동', '한사랑', '일지매', '박여인'],
    '등급': ['Gold', 'Bronze', 'Silver', 'Gold'],
    '점수': [56000, 23000, 44000, 52000]
}

df = pd.DataFrame(dict)
df.head()
```

	이름	등급	점수
0	홍길동	Gold	56000
1	한사랑	Bronze	23000
2	일지매	Silver	44000
3	박여인	Gold	52000

- 인덱스를 지정하지 않으면 위치 기반 정수 값이 인덱스로 지정됨
- 딕셔너리에서 만들면 딕셔너리 키가 데이터프레임 열 이름이 됨

데이터프레임 만들기 - 인덱스 지정

- 만들 때 인덱스와 열 이름 지정할 수 있음

Python

```
lst = [['홍길동', 'Gold', 56000],
       ['한사람', 'Bronze', 23000],
       ['일지매', 'Silver', 44000],
       ['박여인', 'Gold', 52000]]

df = pd.DataFrame(lst, index = [1, 2, 3, 4],
                       columns = ['이름', '등급', '점수'])
df.head()
```

	이름	등급	점수
1	홍길동	Gold	56000
2	한사람	Bronze	23000
3	일지매	Silver	44000
4	박여인	Gold	52000

- 나중에 일반 열을 인덱스로 지정할 수 있음

Python

```
df.set_index(['이름'], inplace = True)
df.head()
```

이름	등급	점수
홍길동	Gold	56000
한사람	Bronze	23000
일지매	Silver	44000
박여인	Gold	52000

CSV파일에서 데이터 불러오기

- 많은 데이터를 갖는 데이터프레임을 만들 때는 일반적으로
  - 데이터베이스에 연결에 직접 가져오거나
  - CSV 파일을 읽어서 데이터를 가져옴
- Github에서 CSV 파일을 읽어오는 예

Python

```
f_path
='https://raw.githubusercontent.com/Jangrae/files/master/Graduate_apply.csv'
df = pd.read_csv(f_path, sep = ',', skipinitialspace = True)
df.head()
```

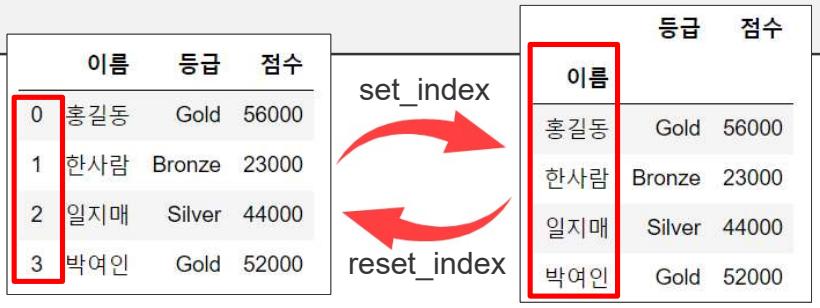
	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4

- 파일 읽기: pd.read_csv( ), pd.read_excel( )
- 파일 쓰기: pd.to_csv( ), pd.to_excel( )

인덱스 재설정

- 일반 열을 인덱스 열로, 또는 기존 인덱스를 위치 인덱스로 초기화

Python
```
# 기존 열 중에서 인덱스열 선택
df.set_index(['이름'], inplace = True)
df.head()

df.reset_index(inplace = True)
df.head()
```



데이터 살펴 보기 – 상위, 하위 조회

- head, tail 메소드 사용, 기본 5개 행 조회

Python
```
# 첫 5개 행 데이터
df.head()
```

	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4

Python
```
# 마지막 3개 행 데이터
df.tail(3)
```

	admit	gre	gpa	rank
397	0	460	2.63	2
398	0	700	3.65	2
399	0	600	3.89	3

- shape 메소드: 데이터프레임 모양 확인 → (행수, 열수) 형태
- columns 속성: 열 이름 확인

Python
```
print(df.columns) # Index(['admit', 'gre', 'gpa', 'rank'], dtype='object')
```

데이터 살펴 보기 – 기초 통계량

- describe 메소드 사용

Python
```
df.describe()
```

- 다음 내용을 확인할 수 있음
  - 개수(Count)
  - 평균(Mean)
  - 표준편차(Std)
  - 최솟값(Min)
  - 사분위값(25%, 50%, 75%)
  - 최댓값(Max) 정렬

	admit	gre	gpa	rank
count	400.000000	400.000000	400.000000	400.00000
mean	0.317500	587.700000	3.389900	2.48500
std	0.466087	115.516536	0.380567	0.94446
min	0.000000	220.000000	2.260000	1.00000
25%	0.000000	520.000000	3.130000	2.00000
50%	0.000000	580.000000	3.395000	2.00000
75%	1.000000	660.000000	3.670000	3.00000
max	1.000000	800.000000	4.000000	4.00000

데이터 살펴 보기 - 정렬

- 정렬 #1: df.sort_index(axis = 0...) → 인덱스로 정렬

Python
```
df.sort_index(axis = 0, ascending = False).head()
```

- 정렬 #2 df.sort_index(axis = 0...) → 열 이름으로 정렬

Python
```
df.sort_index(axis = 1, ascending = True).head()
```

- 정렬 #3 df.sort_values(...) → 열 값으로 정렬

Python
```
df.sort_values(by = ['admit', 'gpa'], ascending = [False, True]).head()
```

데이터 조회 - 열 조회

- 열 이름 지정해서 조회
  - df['열 이름']: 결과가 시리즈로 출력
  - df[['열 이름']]: 결과가 데이터프레임으로 출력

Python

```
# 열 이름으로 조회 #1 --> 결과: 시리즈
df['Close'].head()

# 열 이름으로 조회 #2 --> 데이터프레임
df[['Close']].head()

# 중복 제거 --> 배열
df['admit'].unique()

# 세 개의 열을 동시에 조회
df[[Open', 'Close', 'Volume']].head()
```

```
Date
2015-01-02    8820.0
2015-01-05    8820.0
2015-01-06    8600.0
2015-01-07    8650.0
2015-01-08    8650.0
Name: Close, dtype: float64
```

	Close
Date	
2015-01-02	8820.0
2015-01-05	8820.0
2015-01-06	8600.0
2015-01-07	8650.0
2015-01-08	8650.0

데이터 조회 - 행 조회

- iloc., loc 메소드 사용
  - iloc: 정수형 위치 인덱스, 즉 0부터 시작하는 인덱스로 조회
  - loc: 인덱스 이름(= 인덱스 값)으로 조회(다소 인간적인 방법)

Python

```
# 0 행 조회
df.iloc[[0]]

# 1, 3, 6 행 조회
df.iloc[[1, 3]]

# 1 ~ 3 행 조회
df.iloc[1:4]
```

Python

```
# 0 행 조회
df.loc[['2015-01-02']]

# Date가 2015-01-02, 2015-01-06 행 조회
df.loc[['2015-01-02', '2015-01-06']]

# Date가 2015-01-02 ~ 2015-01-08 행 조회
df.loc['2015-01-02':'2015-01-08']
```

데이터 조회 - 일부 행, 열 조회

- iloc., loc 메소드 사용

Python
```
# 1번 행의 1번 열
df.iloc[1 , 1]

# 0 ~ 3번 행의 0 ~ 2번 열
df.iloc[0:4, 0:3]
```

Python
```
# Date가 '2015-01-02'인 행의 Close 열
df.loc['2015-01-02', 'Close']

# Date가 '2015-01-02' ~ '2015-01-08' 인 행의 Open ~ Close 열
df.loc['2015-01-02':'2015-01-08', 'Open':'Close']
```

데이터 조회 - 조건 조회

- df[조건] 형태로 사용
  - ==, !=, >=, <=, >, <
  - .isin([val1, val2, ...])
  - .between(val1, val2)
  - &(and), |(or)
  - .str.contains(문자열)

Python
```
# 조건을 비교한 결과 True 인 값 조회
df[df['gpa'] > 3.5].head()

# 여러 값을 한꺼번에 비교할 때 .isin() 사용
df[df['rank'].isin([1, 2])].head()
```

Python
```
# and, or 조건으로 여러 조건 함께 조회
df[(df['gpa'] > 3.0) & (df['rank'] == 3)].head()
df[(df['gpa'] > 3.0) | (df['rank'] == 3)].head()

# 이름에 '사'가 들어간 값 조회
df[df['이름'].str.contains('사')]
```

데이터 조회 - 샘플링

- Sampling 메소드 사용

Python
```
# 5개 행 샘플링
df.sample(5)
```

Python
```
# 2% 샘플링
df.sample(frac = 0.02)
```

Python
```
# 전체 샘플링(frac = 1)
df.sample(frac = 1)
```

	admit	gre	gpa	rank
28	1	780	3.22	2
46	1	580	3.46	2
299	0	720	3.40	3
79	1	620	4.00	1
322	0	500	3.01	4
47	0	500	2.97	4
262	1	520	3.19	3
30	0	540	3.78	4

데이터프레임 변경 - 값 변경

- 방법1: loc[인덱스 이름, 열 이름] 으로 조회 & 변경
- 방법2: iloc[정수형 위치 인덱스, 열 번호] 으로 조회 & 변경

Python
```
# 2번 이름을 '한사랑'으로 변경
df.loc['2번', '이름'] = '한사람'

# 2행 이름을 0열을 '일등급'으로 변경
df.iloc[2, 0] = '일등급'

# 1번 ~ 3번 모두 등급은 'NA', 점수는 0으로 변경
df.loc['1번':'3번', ['등급','점수']] = [np.NaN, 0]

# 점수 50000 이상은 등급을 'Gold'로 변경
df.loc[df['점수'] >= 50000, '등급'] = 'Gold'
```

	이름	등급	점수
1번	홍길동	Gold	56000
2번	한사랑	Bronze	23000
3번	일지매	Silver	44000
4번	박여인	Gold	52000

데이터프레임 변경 - 열 이름 변경

- rename 메소드 사용

Python
```
df.rename(columns = {'이름': 'Name',
                     '등급': 'Level',
                     '점수': 'Score',}, inplace = True)
```

	이름	등급	점수
1번	홍길동	Gold	56000
2번	한사랑	Bronze	23000
3번	일지매	Silver	44000
4번	박여인	Gold	52000

	Name	Level	Score
1번	홍길동	Gold	56000
2번	한사랑	Bronze	23000
3번	일지매	Silver	44000
4번	박여인	Gold	52000

데이터프레임 변경 - 열 삭제

- drop 메소드 사용
  - axis = 1: 열을 삭제
  - inplace = True: df에서 직접 삭제

Python
```
# 열 하나 삭제
df.drop('성별', axis = 1, inplace = True)

# 열 두 개 삭제
df.drop(['나이', '등급'], axis = 1, inplace = True)
```

	이름	성별	나이	등급	점수
1번	홍길동	남	25	Gold	56000
2번	한사랑	여	27	Bronze	23000
3번	일지매	남	31	Silver	44000
4번	박여인	여	26	Gold	52000

	이름	점수
1번	홍길동	56000
2번	한사랑	23000
3번	일지매	44000
4번	박여인	52000

데이터프레임 변경 - 가변수

- get_dummies 함수 사용
  - 가변수(Dummy Variable): 범주형 데이터를 숫자로 변환

Python

```
# rank 열은 범주형이므로 더미 변수로 변환
df_rank = pd.get_dummies(df['rank'])
```

	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4

	1	2	3	4
0	0	0	1	0
1	0	0	1	0
2	1	0	0	0
3	0	0	0	1
4	0	0	0	1

데이터프레임 변경 - 가변수

- get_dummies 함수 사용
  - 가변수(Dummy Variable): 범주형 데이터를 숫자로 변환

Python

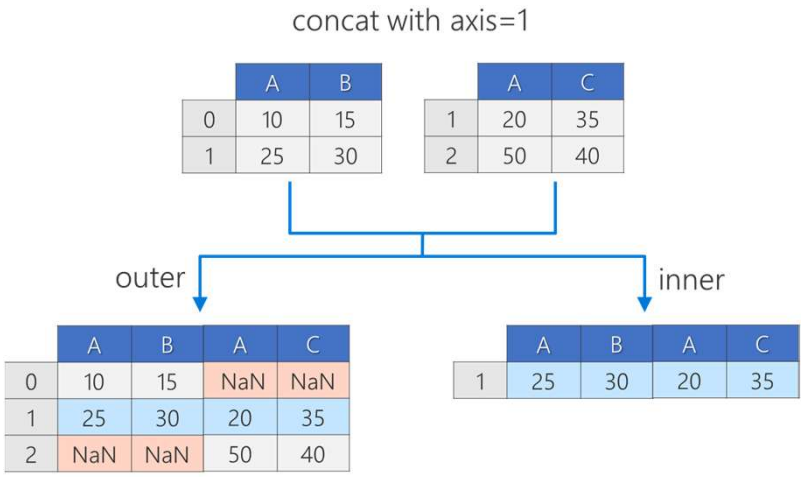
```
# rank 열은 범주형이므로 더미 변수로 변환
df_rank = pd.get_dummies(df['rank'])
```

	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4

	1	2	3	4
0	0	0	1	0
1	0	0	1	0
2	1	0	0	0
3	0	0	0	1
4	0	0	0	1

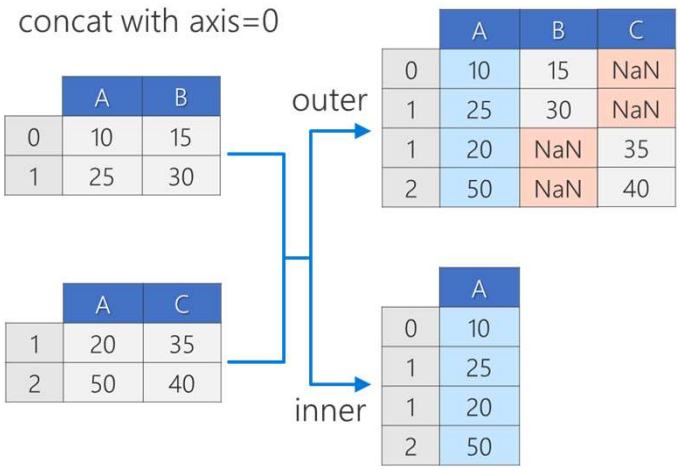
데이터프레임 변경 - 열 합치기

- concat 함수 개념 – 행으로 합치기
  - 두 데이터프레임을 합쳐 새로운 데이터프레임을 만듦



데이터프레임 변경 - 열 합치기

- concat 함수 개념 – 열로 합치기
  - 두 데이터프레임을 합쳐 새로운 데이터프레임을 만듦



데이터프레임 변경 - 열 합치기

- concat 함수 사용

Python

```
# 두 데이터프레임 합치기
df_new = pd.concat([df, df_rank], axis = 1)
```

	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4

+

	1	2	3	4
0	0	0	1	0
1	0	0	1	0
2	1	0	0	0
3	0	0	0	1
4	0	0	0	1

=

	admit	gre	gpa	rank	1	2	3	4
0	0	380	3.61	3	0	0	1	0
1	1	660	3.67	3	0	0	1	0
2	1	800	4.00	1	1	0	0	0
3	1	640	3.19	4	0	0	0	1
4	0	520	2.93	4	0	0	0	1

Group By

- groupby 메소드 사용: 특정 열 기준으로 연속형 값 집계

Python

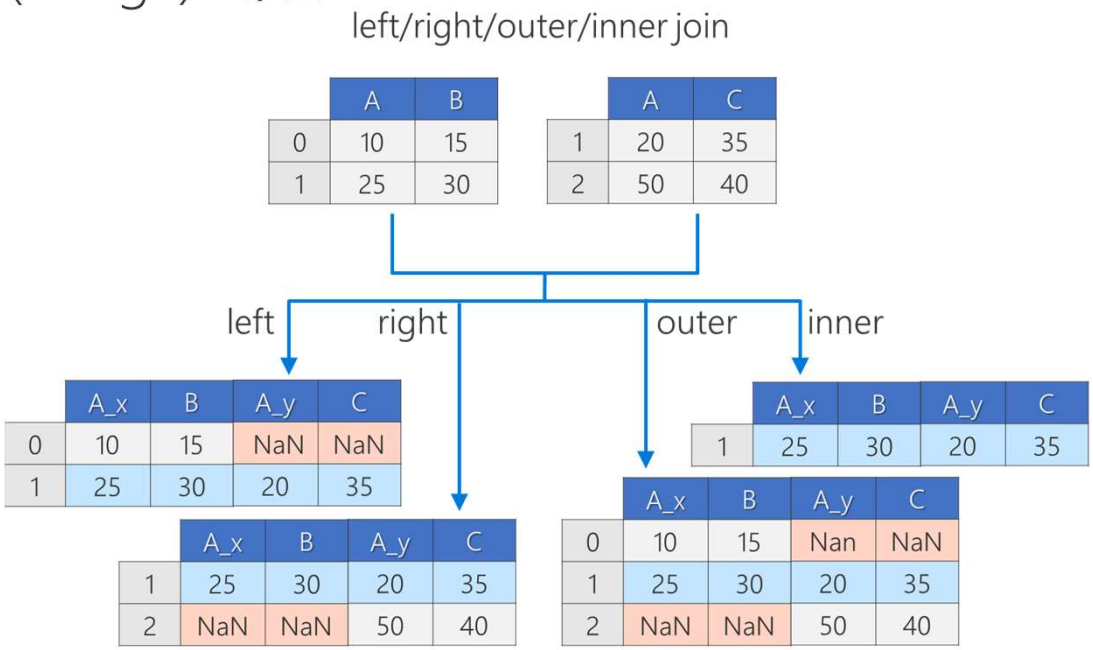
```
# rank별 gre 평균 점수 조회, as_index = True이면 결과가 시리즈가 됨
df.groupby(by='rank', as_index=False)['gre'].mean()

# rank별 admit별 gre와 gpa 평균 조회
df1 = df.groupby(by=['rank','admit'], as_index=False)['gre', 'gpa'].mean()

# rank별 admit 별 count 조회
df2 = df.groupby(by=['rank', 'admit'], as_index=False)['gre'].count()
```

	rank	gre
0	1	611.803279
1	2	596.026490
2	3	574.876033
3	4	570.149254

	rank	admit	gre	gpa
0	1	0	582.857143	3.345714
1	1	1	636.363636	3.544242
2	2	0	586.597938	3.316598
3	2	1	612.963...	

	rank	admit	gre
0	1	0	28
1	1	1	33
2	2	0	97
3	2		

Join(Merge) 개념



	A	B
0	10	15
1	25	30

	A	C
1	20	35
2	50	40

	A_x	B	A_y	C
0	10	15	NaN	NaN
1	25	30	20	35

	A_x	B	A_y	C
1	25	30	20	35
2	NaN	NaN	50	40

	A_x	B	A_y	C
0	10	15	Nan	NaN
1	25	30	20	35
2	NaN	NaN	50	40

	A_x	B	A_y	C
1	25	30	20	35

Join(Merge) 사용

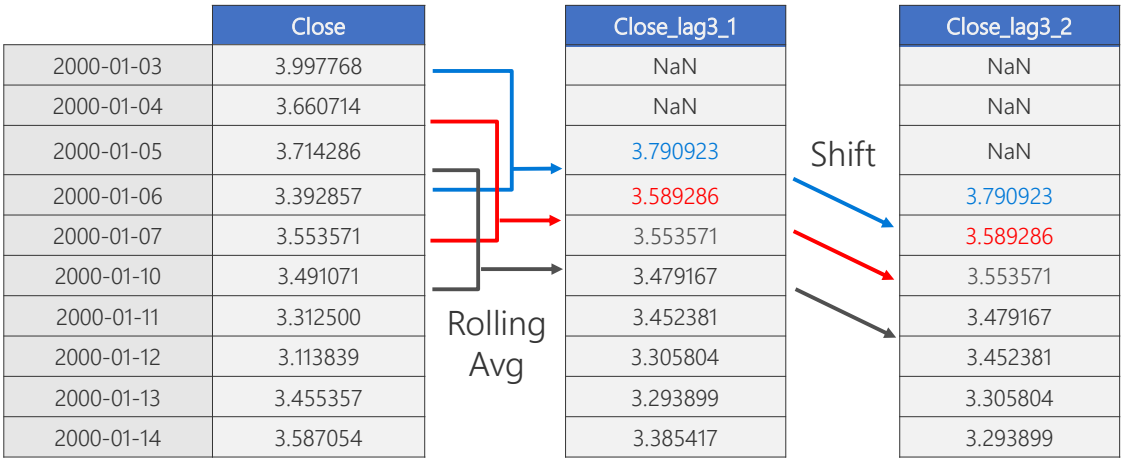
- merge 함수 사용
  - 특정 열(key) 기준으로 두 데이터프레임 합치기
  - 공통된 열이 있으면 그 열을 key로 사용

Python

```
mean_by_rank = df.groupby(by=['rank'], as_index=False)['gre', 'gpa'].mean()
mean_by_rank.rename(columns={'gre':'gre_mean', 'gpa':'gpa_mean'}, inplace=True)

cnt_by_rank = df.groupby(by=['rank'], as_index=False)['gre'].count()
cnt_by_rank.rename(columns={'gre':'count'}, inplace=True)

# inner join
pd.merge(mean_by_rank, cnt_by_rank)

# outer join
pd.merge(mean_by_rank, cnt_by_rank, on='rank',
         how='left')
```

	rank	gre_mean	gpa_mean	count
0	1	611.803279	3.453115	61
1	2	596.026490	3.361656	151
2	3	574.876033	3.432893	121
3	4	570.149254	3.318358	67

Rolling & Shift 개념



	Close	Close_lag3_1	Close_lag3_2
2000-01-03	3.997768	NaN	NaN
2000-01-04	3.660714	NaN	NaN
2000-01-05	3.714286	3.790923	NaN
2000-01-06	3.392857	3.589286	3.790923
2000-01-07	3.553571	3.553571	3.589286
2000-01-10	3.491071	3.479167	3.553571
2000-01-11	3.312500	3.452381	3.479167
2000-01-12	3.113839	3.305804	3.452381
2000-01-13	3.455357	3.293899	3.305804
2000-01-14	3.587054	3.385417	3.293899

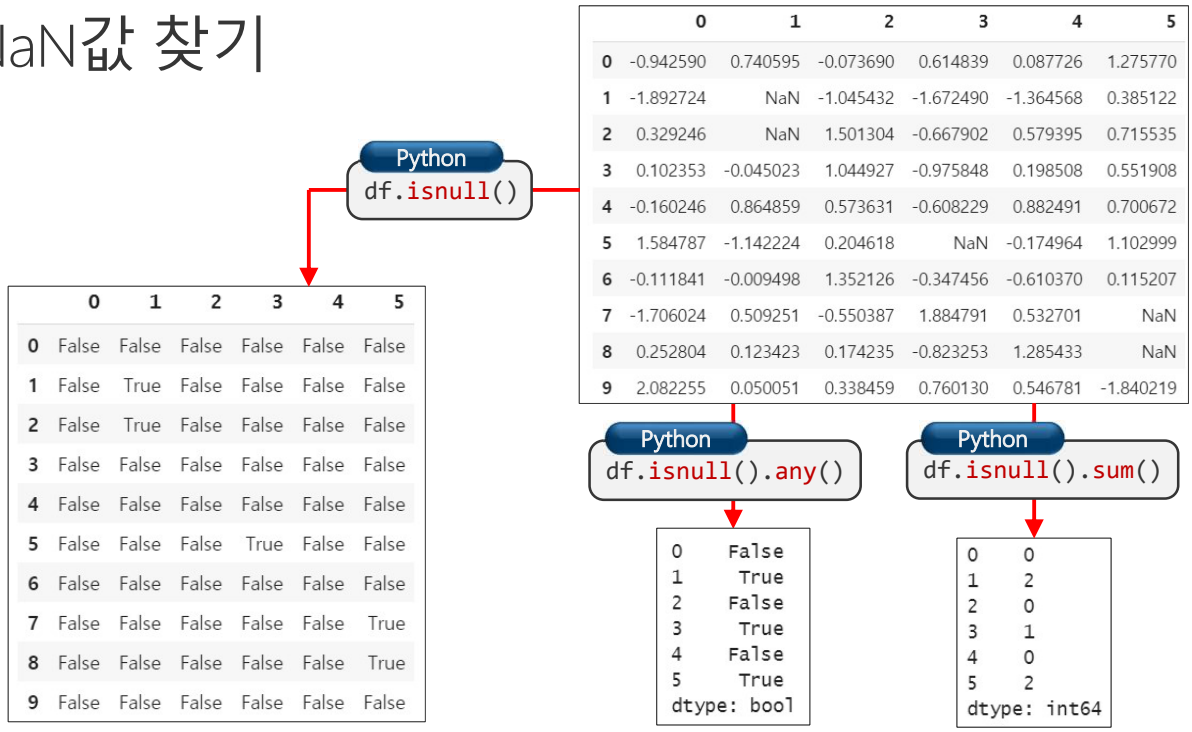
Rolling & Shift 사용

- rolling, shift 메소드 사용
  - Sliding window로 이동하며 값 계산

Python

```
# 기준일 포함하여 과거 3일의 평균을 데이터프레임에 붙이기
stock['Close_lag3_1'] = stock['Close'].rolling(3).mean()

# shift()안에 숫자를 변경하면서 기능 확인, default = 1
stock['Close_lag3_2'] = stock['Close_lag3_1'].shift(1)

# 이동평균(Rolling)과 하루 뒤로 미루는(Shift) 작업을 한꺼번에 하려면
stock['Close_lag3_3'] = stock['Close'].rolling(3).mean().shift(1)

# 값이 몇 개이면 계산을 할 지 지정, min_perios = 1 --> 값이 하나라도 있으면 계산
stock['Close_lag3_4'] = stock['Close'].rolling(3, min_periods = 1).mean()
```

NaN값 찾기



	0	1	2	3	4	5
0	-0.942590	0.740595	-0.073690	0.614839	0.087726	1.275770
1	-1.892724	NaN	-1.045432	-1.672490	-1.364568	0.385122
2	0.329246	NaN	1.501304	-0.667902	0.579395	0.715535
3	0.102353	-0.045023	1.044927	-0.975848	0.198508	0.551908
4	-0.160246	0.864859	0.573631	-0.608229	0.882491	0.700672
5	1.584787	-1.142224	0.204618	NaN	-0.174964	1.102999
6	-0.111841	-0.009498	1.352126	-0.347456	-0.610370	0.115207
7	-1.706024	0.509251	-0.550387	1.884791	0.532701	NaN
8	0.252804	0.123423	0.174235	-0.823253	1.285433	NaN
9	2.082255	0.050051	0.338459	0.760130	0.546781	-1.840219

	0	1	2	3	4	5
0	False	False	False	False	False	False
1	False	True	False	False	False	False
2	False	True	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
5	False	False	False	True	False	False
6	False	False	False	False	False	False
7	False	False	False	False	False	True
8	False	False	False	False	False	True
9	False	False	False	False	False	False

NaN값 채우기

- 단일 값으로 채우기: .fillna(0)
- 열 별로 채울 값을 딕셔너리 형태로 만들고 채우기

```
Python
df.fillna(0)
```

	0	1	2	3	4	5
0	-0.942590	0.740595	-0.073690	0.614839	0.087726	1.275770
1	-1.892724	0.000000	-1.045432	-1.672490	-1.364568	0.385122
2	0.329246	0.000000	1.501304	-0.667902	0.579395	0.715535
3	0.102353	-0.045023	1.044927	-0.975848	0.198508	0.551908
4	-0.160246	0.864859	0.573631	-0.608229	0.882491	0.700672
5	1.584787	-1.142224	0.204618	0.000000	-0.174964	1.102999
6	-0.111841	-0.009498	1.352126	-0.347456	-0.610370	0.115207
7	-1.706024	0.509251	-0.550387	1.884791	0.532701	0.000000
8	0.252804	0.123423	0.174235	-0.823253	1.285433	0.000000
9	2.082255	0.050051	0.338459	0.760130	0.546781	-1.840219

```
Python
values = {1: 0.5, 3: 1.5, 5: 2.5}
df.fillna(value = values)
```

	0	1	2	3	4	5
0	-0.942590	0.740595	-0.073690	0.614839	0.087726	1.275770
1	-1.892724	0.500000	-1.045432	-1.672490	-1.364568	0.385122
2	0.329246	0.500000	1.501304	-0.667902	0.579395	0.715535
3	0.102353	-0.045023	1.044927	-0.975848	0.198508	0.551908
4	-0.160246	0.864859	0.573631	-0.608229	0.882491	0.700672
5	1.584787	-1.142224	0.204618	1.500000	-0.174964	1.102999
6	-0.111841	-0.009498	1.352126	-0.347456	-0.610370	0.115207
7	-1.706024	0.509251	-0.550387	1.884791	0.532701	2.500000
8	0.252804	0.123423	0.174235	-0.823253	1.285433	2.500000
9	2.082255	0.050051	0.338459	0.760130	0.546781	-1.840219

파이썬 라이브러리(전처리)

4. 데이터 시각화

1. Pandas 자체 시각화 기능

Pandas 자체 시각화 기능

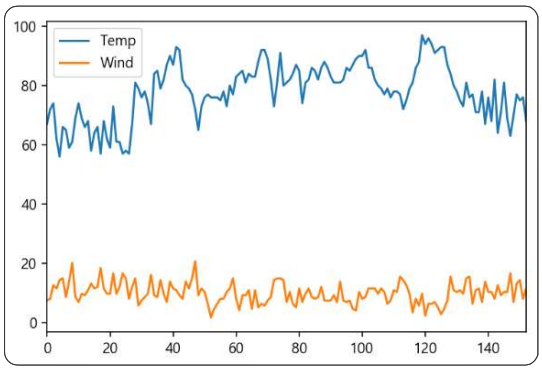
- Matplotlib 라이브러리의 일부 기능을 Pandas가 내장하고 있음
- Matplotlib 라이브러리를 불러오지 않아도 사용 가능

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67	5	1
1	36.0	118.0	8.0	72	5	2
2	12.0	149.0	12.6	74	5	3
3	18.0	313.0	11.5	62	5	4
4	NaN	NaN	14.3	56	5	5
...	...	...	...	...	...	...
148	30.0	193.0	6.9	70	9	26
149	NaN	145.0	13.2	77	9	27
150	14.0	191.0	14.3	75	9	28
151	18.0	131.0	8.0	76	9	29
152	20.0	223.0	11.5	68	9	30

153 rows × 6 columns

Python

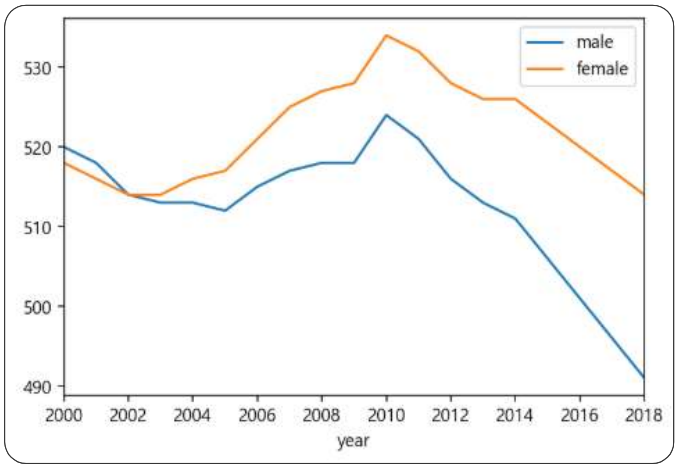
```
aq[['Temp', 'Wind']].plot()
```



Line Graph

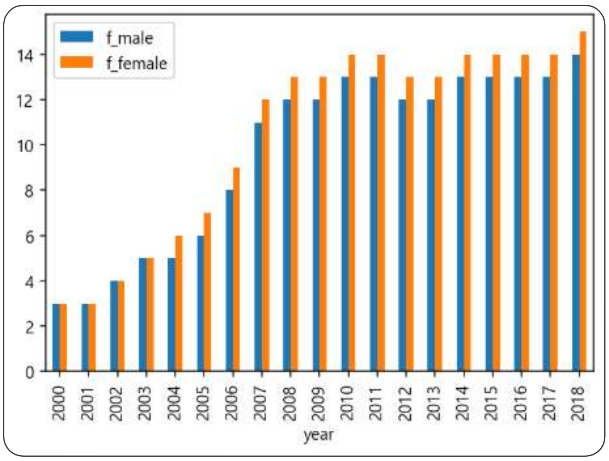
Python

```
seoul_pop.plot(kind='line', x='year', y=['male', 'female'])
plt.show()
```



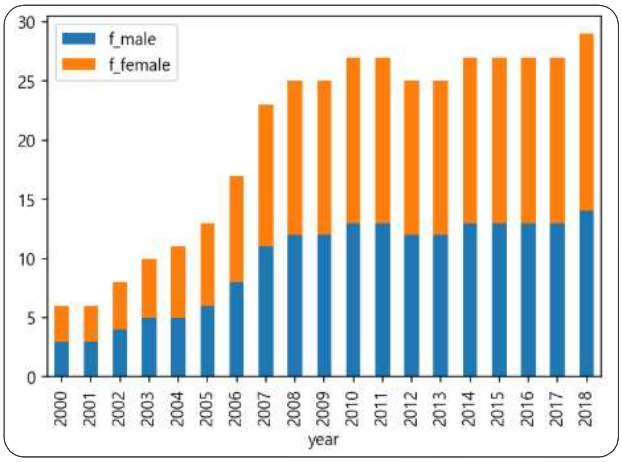
Bar Plot

Python
```
seoul_pop.plot(kind='bar', x='year', y=['f_male' ,'f_female'])
plt.show()
```



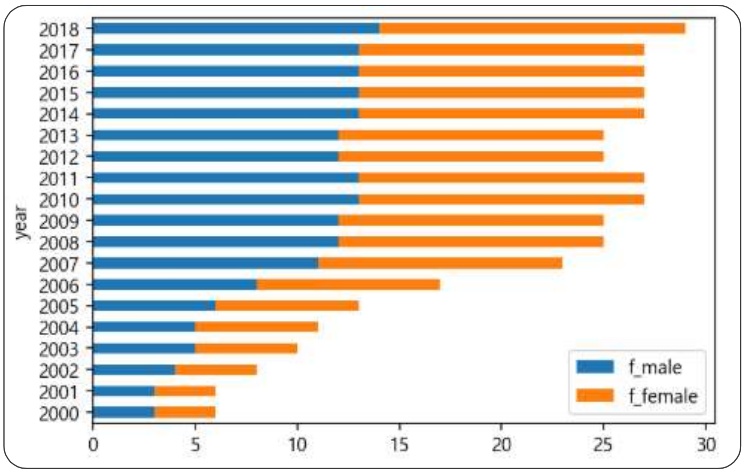
Bar Plot(누적)

Python
```
seoul_pop.plot(kind='bar', x='year', y=['f_male' ,'f_female'], stacked=True)
plt.show()
```



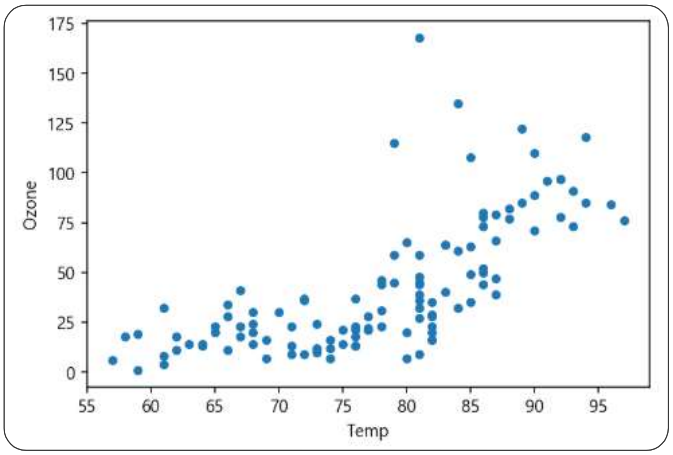
Barh Plot

Python
```
seoul_pop.plot(kind='barh', x='year', y=['f_male' ,'f_female'], stacked=True)
plt.show()
```



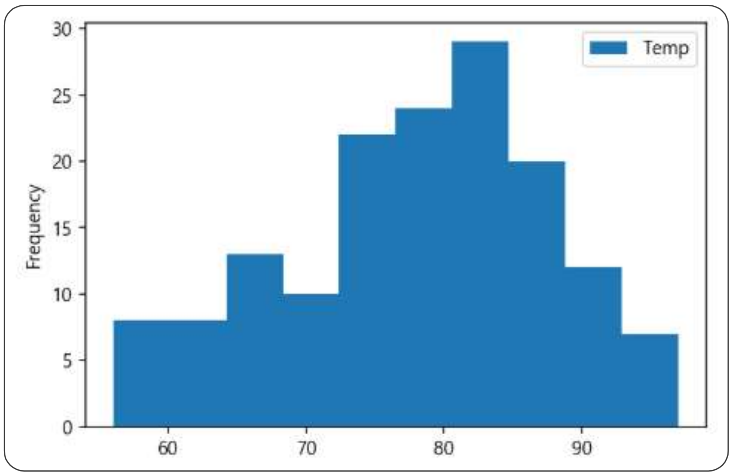
Scatter Plot

Python
```
airquality.plot(kind='scatter', x='Temp', y='Ozone')
plt.show()
```



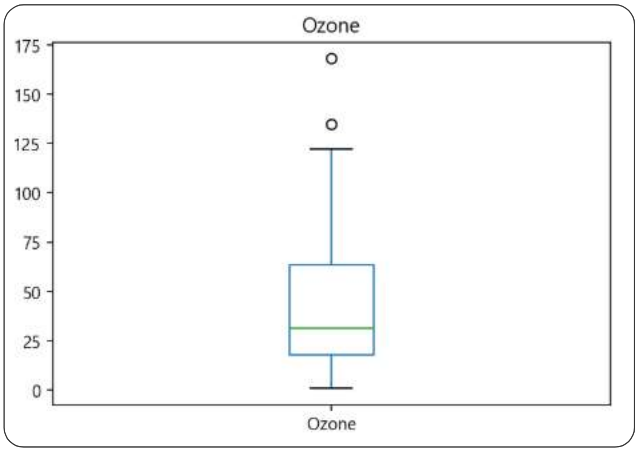
Histogram

```python
airquality.plot(kind='hist', y='Temp')
plt.show()
```



Box Plot

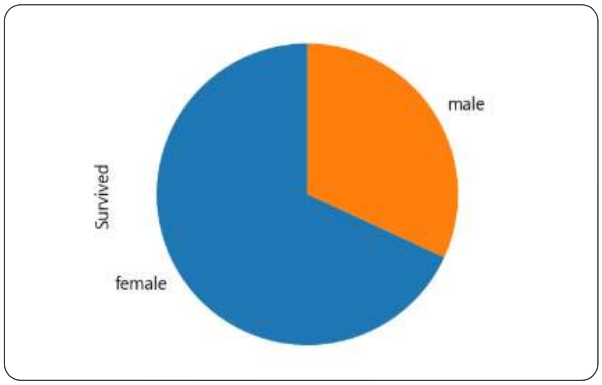
```python
airquality.plot(kind='box', y='Wind')
plt.show()
```



Pie Chart

Python

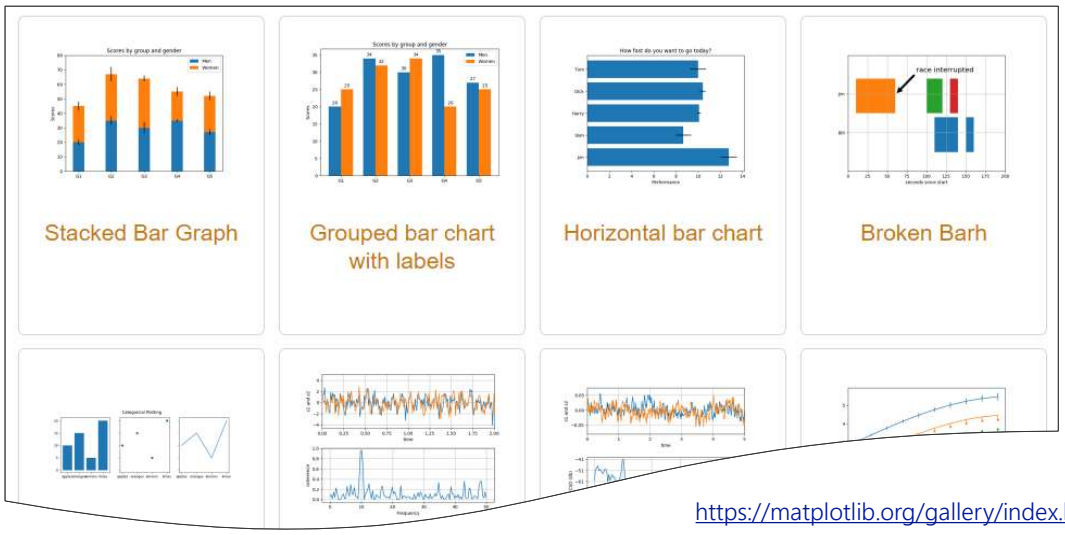
```
tit_survived = titanic.groupby(['Sex'])[['Survived']].sum()
tit_survived.plot.pie(y='Survived', startangle=90)
plt.legend().remove()
plt.show()
```



2. Maplotlib **라이브러리 사용**

Matplotlib?

- Python에서 데이터를 시각화 할 때 가장 많이 사용하는 패키지



https://matplotlib.org/gallery/index.html

Matplotlib 라이브러리 사용

- Matplotlib를 사용하기 위해서는 matplotlib.pyplot를 import 함
- pyplot을 일반적으로 plt 라는 별칭을 주어 사용
- Numpy와 Pandas도 같이 사용하는 경우가 대부분이니 같이 import 함
- Jupyter Notebook에서 원활한 사용을 위해 환경 설정도 같이 함

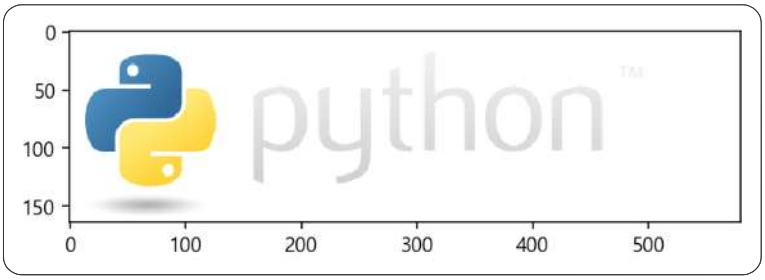
Python

```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import pandas as pd
```

이미지 불러오기

Python

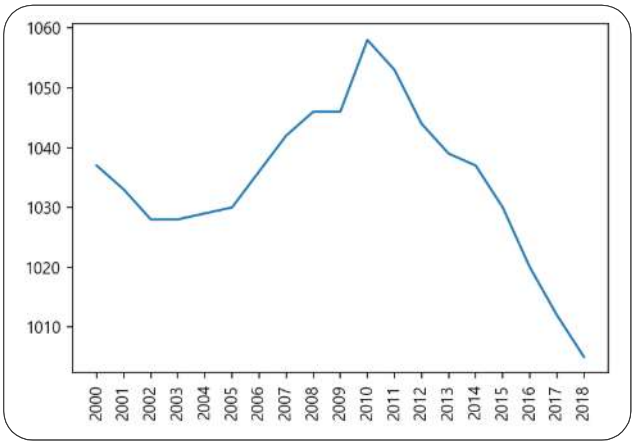
```
img = mpimg.imread('https://www.python.org/static/img/python-logo@2x.png')
imgplot = plt.imshow(img)
```



Line Plot

Python

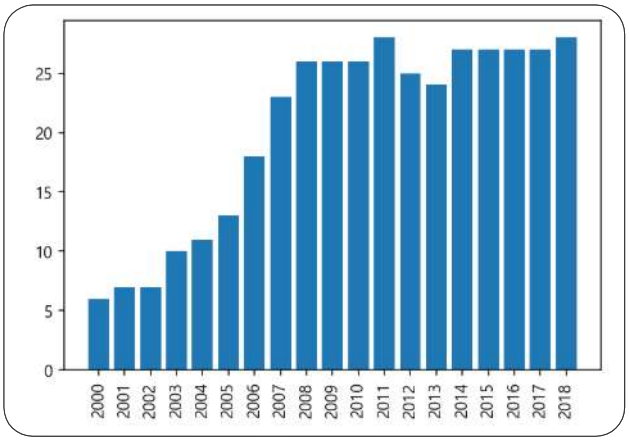
```
plt.plot(seoul_pop['year'], seoul_pop['total'])
plt.xticks(rotation='vertical')
plt.show()
```



Bar Plot

Python

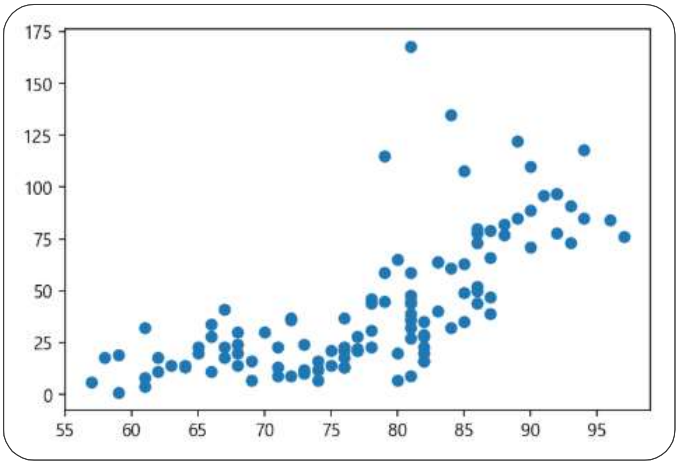
```
plt.bar(seoul_pop['year'], seoul_pop['f_total'])
plt.xticks(rotation='vertical')
plt.show()
```



Scatter Plot

Python

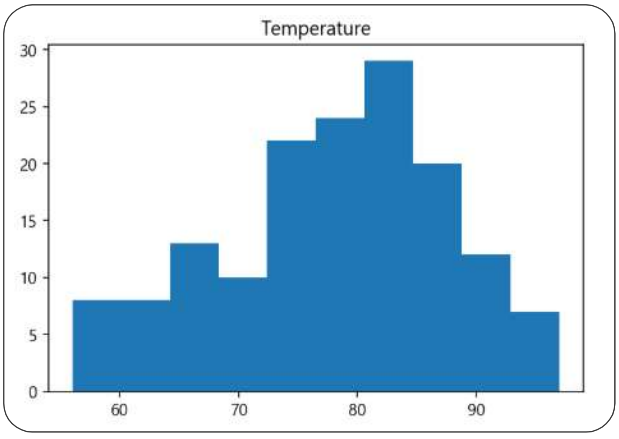
```
plt.scatter(airquality['Temp'], airquality['Ozone'])
plt.show()
```



Histogram

Python

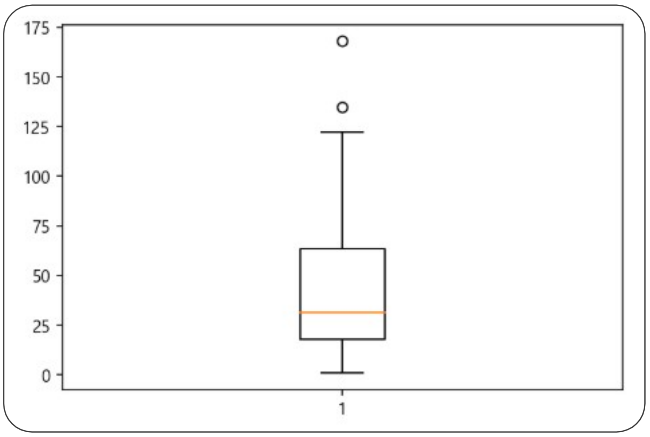
```
plt.hist(airquality['Temp'])
plt.title('Temperature')
plt.show()
```



Box Plot

Python

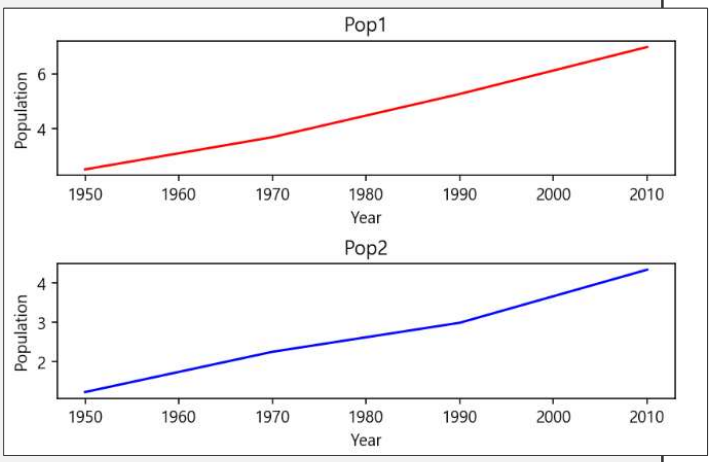
```
air_notnull = airquality[airquality['Ozone'].notnull()]
plt.boxplot(air_notnull['Ozone'])
plt.show()
```



Matplotlib 사용 예 - 커스터마이징

Python

```
year = [1950, 1970, 1990, 2010]
pop1 = [2.519, 3.692, 5.263, 6.972]
pop2 = [1.231, 2.252, 2.988, 4.334]

# plot1
plt.subplot(2, 1, 1)
plt.plot(year, pop1, 'red')
plt.xlabel('Year')
plt.ylabel('Population')
plt.title('Pop1')

# plot2
plt.subplot(2, 1, 2)
plt.plot(year, pop2, 'blue')
plt.xlabel('Year')
plt.ylabel('Population')
plt.title('Pop2')

plt.tight_layout()
plt.show()
```




E
N
D