데이터 분석 입문

Category	
Difficulty	Lv.5
Status	In progress
Created	@2023년 12월 7일 오후 7:47
복습	☐
유형	강의

데이터 분석 입문

```
import csv
import matplotlib.pyplot as plt

f = open('gender.csv', encoding='cp949')
data = csv.reader(f)

header = next(data)

male = []
```

```
female = []

name = input('찾고 싶은 지역의 이름을 알려주세요: ')
# 제주특별자치도

for row in data :
    if name in row[0] :
        for i in row[3:104] :
            i = i.replace(',', '')
            male.append(-int(i))
        for i in row[106:] :
            i = i.replace(',', '') # replace() 함수 -> 특정 문자
            female.append(int(i))

f.close()
plt.rc('font', family = 'Malgun Gothic')
plt.rc('axes', unicode_minus = False)

plt.style.use('ggplot')
# plt.figure(figsize=(5, 10))

plt.title(name + ' 지역의 남녀 성별 인구 분포')

plt.barh(range(len(male)), male, color='b', label='male')
plt.barh(range(len(female)), female, color='r', label='female
plt.xlabel('Population')
plt.ylabel('Age')
plt.grid(True)
plt.legend()
plt.show()
```

replace () 함수 활용시 특정문자 제거 가능

replace('특정문자' , "") → 특정 문자를 제거

파일 형식은 **PNG Image**가 기본

PNG(Portable Network Graphics)는 비손실 그래픽 파일 포맷의 하나

```
import matplotlib.pyplot as plt

x = [0, 2, 4, 5]
y = [5, 10, 15, 20]

plt.bar(x, y)
plt.savefig('막대그래프_기본.png')
plt.savefig('막대그래프_DPI_50.png', dpi=50)
plt.savefig('막대그래프_DPI_200.png', dpi=200)
plt.show()
```

DPI(Dots per inch)는 디스플레이 해상도의 측정 단위이며,
1 제곱 인치 공간 안에 배치되는 픽셀(Pixel)의 수를 의미

→ DPI 값을 크게 지정하면 이미지가 선명해지는 대신 , 이미지 파일 크기가 커짐

```
import matplotlib.pyplot as plt

x = [0, 2, 4, 5]
y = [5, 10, 15, 20]

plt.bar(x, y)
plt.savefig('막대그래프_facecolor.png', facecolor='skyblue')
plt.show()
```

savefig : 함숫값으로 저장할 이미지 파일 명 작성

facecolor : 배경색 지정

```
# CSV 파일을 읽기 위해 csv 모듈을 import합니다.
import csv

# 'subwayfee.csv' 파일을 'cp949' 인코딩으로 엽니다.
f = open('subwayfee.csv', encoding='cp949')

# CSV 파일의 내용을 읽기 위해 csv.reader를 사용합니다.
data = csv.reader(f)

# 첫 번째 행을 건너뛰고 실제 데이터를 읽습니다.
next(data)

# 각각의 방법으로 초기화된 리스트를 생성합니다.

# 1. 빈 문자열로 초기화된 1차원 리스트를 생성합니다.
max_station = ['', '', '', '']
print(max_station)

# 2. 빈 문자열로 초기화된 1차원 리스트를 생성하는 다른 방법입니다.
max_station = [''] * 4
print(max_station)

# 3. 빈 문자열로 초기화된 1차원 리스트를 생성하는 또 다른 방법입니다.
max_station = []
for i in range(4):
    max_station.append('')
print(max_station)
```

```
# 4. 리스트 컴프리헨션을 사용하여 빈 문자열로 초기화된 1차원 리스트를 생성
max_station = ['' for _ in range(4)]
print(max_station)

# 파일을 닫습니다.
f.close()
```

`max_station`은 코드에서 사용된 네 가지 방법 중 하나로 **초기화된 1차원 리스트입니다. 이 리스트는 역별 유임승차인원이 가장 많은 역을 기록하기 위한 변수로 사용됩니다**. 코드의 실행 중에 이 리스트는 각 방법으로 초기화되고, 이후에는 데이터를 처리하면서 유임승차인원이 최대인 역의 정보를 저장하는 데 사용될 것입니다.

```
# CSV 파일을 읽기 위해 csv 모듈을 import합니다.
import csv

# 'subwayfee.csv' 파일을 'cp949' 인코딩으로 엽니다.
f = open('subwayfee.csv', encoding='cp949')

# CSV 파일의 내용을 읽기 위해 csv.reader를 사용합니다.
data = csv.reader(f)

# 첫 번째 행을 헤더로 저장합니다.
header = next(data)

# 최대 승차 인원과 해당 역을 기록할 리스트를 초기화합니다.
max_station = ['', '', '', '']
max_people = [0 for _ in range(4)]

# 유임승차, 유임하차, 무임승차, 무임하차를 나타내는 라벨을 정의합니다.
label = ['유임승차', '유임하차', '무임승차', '무임하차']

# 데이터를 반복하면서 최대 승차 인원을 찾습니다.
```

```
for row in data:
    # 각 열에 대한 처리를 수행합니다. 인덱스 4, 5, 6, 7은 유임승차, 유임
    for i in range(4, 8):
        # 문자열로 표현된 숫자에서 쉼표를 제거하고 정수로 변환합니다.
        row[i] = int(row[i].replace(',', ''))
        # 최대값을 비교하여 업데이트합니다.
        if row[i] > max_people[i - 4]:
            max_people[i - 4] = row[i]
            max_station[i - 4] = row[3] + ' ' + row[1]

# 파일을 닫습니다.
f.close()

# 최대 승차 인원과 해당 역을 출력합니다.
for i in range(4):
    print(label[i] + ' : ' + max_station[i] + ':', max_people
```

`max_people = [0 for _ in range(4)]`는 리스트 컴프리헨션을 사용하여 0으로 초기화된 길이가 4인 리스트를 생성하는 코드입니다.

여기서 사용된 각 구성 요소의 의미는 다음과 같습니다:

- `[0 for _ in range(4)]`: 0으로 초기화된 값을 가지는 리스트를 생성합니다.
- `for _ in range(4)`: 0부터 3까지의 인덱스를 가지는 범위를 반복합니다. `_`는 변수를 사용하지 않을 경우에 관례적으로 사용되는 이름으로, 실제로 사용되지 않는 변수를 나타냅니다. 이 경우에는 반복 횟수에만 관심이 있기 때문에 변수의 값을 사용하지 않습니다.

따라서 `max_people`은 길이가 4이고 모든 요소가 0으로 초기화된 리스트가 됩니다. 이 리스트는 이후에 최대 승차 인원을 저장하기 위한 용도로 사용됩니다.

```
# CSV 파일을 읽기 위해 csv 모듈을 import합니다.
import csv

# 'subwaytime.csv' 파일을 'cp949' 인코딩으로 엽니다.
f = open('subwaytime.csv', encoding='cp949')
```

```
# CSV 파일의 내용을 읽기 위해 csv.reader를 사용합니다.
data = csv.reader(f)

# 헤더를 읽고 다음 행을 건너뜁니다.
header = next(data)
next(data)

# 결과를 저장할 빈 리스트를 생성합니다.
result = []

# 데이터를 처리하면서 승차 인원 정보를 추출합니다.
for row in data:
    # 4번 인덱스부터 51번 인덱스까지의 데이터에 있는 쉼표를 제거합니다.
    for i in range(4, 52):
        row[i] = row[i].replace(',', '')

    # 문자열로 되어 있는 숫자를 정수로 변환합니다.
    row[4:] = map(int, row[4:])

    # 승차 인원 정보 중 10번 인덱스의 데이터를 결과 리스트에 추가합니다.
    result.append(row[10])

# 파일을 닫습니다.
f.close()

# 결과 리스트를 오름차순으로 정렬합니다.
result.sort()
# 내림차순으로 정렬하려면 주석을 해제하고 위의 코드를 주석 처리합니다.
# result.sort(reverse=True)

# 시각화를 위해 필요한 라이브러리를 import합니다.
import matplotlib.pyplot as plt

# 한글 폰트를 설정합니다.
plt.rc('font', family='Malgun Gothic')

# 그래프의 크기를 설정합니다.
plt.figure(figsize=(10, 3))
```

```
# 막대 그래프를 생성합니다.
plt.bar(range(len(result)), result)

# x축과 y축에 라벨을 추가합니다.
plt.xlabel('역')
plt.ylabel('승차 인원수')

# 그리드를 표시합니다.
plt.grid(True)

# 그래프를 출력합니다.
plt.show()
```

- sort 함수를 이용하면 데이터의 순서를 오름차순으로 정렬할 수 있습니다.
- 내림차순으로 정렬하고 싶다면 sort(reverse=True)라고 적으면 됩니다.

```
# CSV 파일을 읽기 위해 csv 모듈을 import합니다.
import csv

# 시각화를 위해 필요한 라이브러리를 import합니다.
import matplotlib.pyplot as plt

# 'subwaytime.csv' 파일을 'cp949' 인코딩으로 엽니다.
f = open('subwaytime.csv', encoding='cp949')

# CSV 파일의 내용을 읽기 위해 csv.reader를 사용합니다.
data = csv.reader(f)

# 헤더를 읽고 다음 행을 건너뜁니다.
header = next(data)
next(data)

# 최대 승차 인원과 해당 역을 기록할 변수를 초기화합니다.
max_people = 0
max_station = ''
```

```
# 데이터를 처리하면서 최대 승차 인원을 찾습니다.
for row in data:
    # 4번 인덱스부터 51번 인덱스까지의 데이터에 있는 쉼표를 제거합니다.
    for i in range(4, 52):
        row[i] = row[i].replace(',', '')

    # 문자열로 되어 있는 숫자를 정수로 변환합니다.
    row[4:] = map(int, row[4:])

    # 10번 인덱스부터 14번 인덱스까지의 짝수 번째 열의 값을 합하여 승차 인
    if sum(row[10:15:2]) > max_people:
        max_people = sum(row[10:15:2])
        max_station = row[3] + '(' + row[1] + ')'

# 파일을 닫습니다.
f.close()

# 최대 승차 인원과 해당 역을 출력합니다.
print(max_station, ':', max_people, '명')
```

```
row[10] + row[12] + row[14]
② sum([row[10], row[12], row[14]])
③ sum(row[10:15:2])

=

# 위 코드에서 사용된 부분
if sum(row[10:15:2]) > max_people:
    max_people = sum(row[10:15:2])
    max_station = row[3] + '(' + row[1] + ')'
```

이 부분에서 `sum(row[10:15:2])`은 리스트 슬라이싱을 활용하여 `row` 리스트에서 10번 인덱스부터 14번 인덱스까지의 홀수 번째 열의 값을 추출한 후, 그 값을 모두 합하는 역할을 합니다.

해석해보면:

- `row[10:15:2]`: 10, 12, 14번 인덱스의 값을 추출합니다. (슬라이스의 세 번째 매개변수는 증가값을 나타냅니다. 여기서는 2로 설정하여 짝수 번째 인덱스만 추출합니다.)
- `sum(row[10:15:2])`: 추출한 값들의 합을 계산합니다.

따라서 `sum(row[10:15:2])`는 `row[10] + row[12] + row[14]`와 동일한 결과를 갖습니다. 이것은 주로 코드를 간결하게 표현하기 위해 사용된 표현 방식 중 하나입니다.

```
# CSV 파일을 읽기 위해 csv 모듈을 import합니다.
import csv

# 시각화를 위해 필요한 라이브러리를 import합니다.
import matplotlib.pyplot as plt

# 'subwaytime.csv' 파일을 'cp949' 인코딩으로 엽니다.
f = open('subwaytime.csv', encoding='cp949')

# CSV 파일의 내용을 읽기 위해 csv.reader를 사용합니다.
data = csv.reader(f)

# 헤더를 읽고 다음 행을 건너뜁니다.
header = next(data)
next(data)

# 각 시간대별 최대 승차 인원과 해당 역을 기록할 리스트를 초기화합니다.
max_people = [0] * 24
max_station = [''] * 24

# 데이터를 처리하면서 최대 승차 인원과 해당 역을 찾습니다.
for row in data:
    # 4번 인덱스부터 51번 인덱스까지의 데이터에 있는 쉼표를 제거합니다.
    for i in range(4, 52):
        row[i] = row[i].replace(',', '')

    # 문자열로 되어 있는 숫자를 정수로 변환합니다.
    row[4:] = map(int, row[4:])
```

```
    # 각 시간대별로 최대 승차 인원을 찾습니다.
    for j in range(24):
        people = row[2 * j + 4]
        if people > max_people[j]:
            max_people[j] = people
            max_station[j] = row[3] + '(' + str((j + 4) % 24)

# 파일을 닫습니다.
f.close()

# 한글 폰트를 설정합니다.
plt.rc('font', family='Malgun Gothic')

# 막대 그래프를 생성합니다.
plt.bar(range(24), max_people)

# x축에 시간대별 역 이름을 표시합니다.
plt.xticks(range(24), max_station, rotation=90)

# 그리드를 표시합니다.
plt.grid(True)

# 그래프를 출력합니다.
plt.show()
```

- `max_people = [0] * 24`: 24개의 원소를 가진 리스트를 생성하고, 모든 원소를 0으로 초기화합니다. 이 리스트는 각 시간대별 최대 승차 인원을 기록합니다.
- `max_station = [''] * 24`: 24개의 원소를 가진 리스트를 생성하고, 모든 원소를 빈 문자열로 초기화합니다. 이 리스트는 각 시간대별 최대 승차 역을 기록합니다.

1. `for j in range(24):`: 0부터 23까지의 값을 순회하는 반복문입니다. 이는 24시간 동안의 시간대를 나타냅니다.
2. `people = row[2 * j + 4]`: 현재 시간대 `j`에 대한 승차 인원을 `row` 리스트에서 가져와서 `people` 변수에 저장합니다. 시간대마다 두 개의 열에 해당하는 데이터를 가져오기 때문에 `2 * j + 4`를 사용합니다.

3. `if people > max_people[j]:`: 현재 시간대의 승차 인원이 기존 최대 승차 인원보다 큰지 비교합니다. 만약 크다면 아래 코드를 실행합니다.
4. `max_people[j] = people`: 현재 시간대의 승차 인원이 기존 최대 승차 인원보다 크다면, `max_people` 리스트의 해당 시간대 인덱스에 현재 승차 인원을 업데이트합니다.
5. `max_station[j] = row[3] + '(' + str((j + 4) % 24) + ')'`: 최대 승차 인원이 갱신되면, `max_station` 리스트의 해당 시간대 인덱스에 현재 역의 정보를 저장합니다. `row[3]`은 역 이름을 나타내고, `(j + 4) % 24`는 현재 시간대를 나타냅니다. `% 24`를 사용하여 24를 초과하는 값이 나올 경우 24로 나눈 나머지를 구합니다. 이렇게 함으로써 시간대가 24를 넘어가더라도 리스트의 범위를 넘어가지 않도록 합니다.

```
# CSV 파일을 읽기 위해 csv 모듈을 import합니다.
import csv

# 시각화를 위해 필요한 라이브러리를 import합니다.
import matplotlib.pyplot as plt

# 'subwaytime.csv' 파일을 'cp949' 인코딩으로 엽니다.
f = open('subwaytime.csv', encoding='cp949')

# CSV 파일의 내용을 읽기 위해 csv.reader를 사용합니다.
data = csv.reader(f)

# 헤더를 읽고 다음 행을 건너뜁니다.
header = next(data)
next(data)

# 시간대별 승차와 하차 인원을 저장할 리스트를 초기화합니다.
s_in = [0] * 24
s_out = [0] * 24

# 데이터를 처리하면서 시간대별 승차와 하차 인원을 누적합니다.
for row in data:
    # 4번 인덱스부터 51번 인덱스까지의 데이터에 있는 쉼표를 제거합니다.
```

```
    for i in range(4, 52):
        row[i] = row[i].replace(',', '')

    # 문자열로 되어 있는 숫자를 정수로 변환합니다.
    row[4:] = map(int, row[4:])

    # 각 시간대별 승차와 하차 인원을 누적합니다.
#이 코드 부분은 한 행의 데이터에서 각 시간대별로 승차와 하차 인원을 추출하여
#s_in과 s_out 리스트에 누적하는 역할을 합니다
    for i in range(24):
        s_in[i] += row[4 + i * 2]
        s_out[i] += row[5 + i * 2]

# 파일을 닫습니다.
f.close()

# 한글 폰트를 설정합니다.
plt.rc('font', family='Malgun Gothic')

# 그래프의 제목을 설정합니다.
plt.title('지하철 시간대별 승하차 인원 추이')

# 승차와 하차 인원을 선 그래프로 표현합니다.
plt.plot(s_in, label='승차')
plt.plot(s_out, label='하차')

# 범례를 표시합니다.
plt.legend()

# 그리드를 표시합니다.
plt.grid(True)

# x축에 시간대를 표시합니다.
timelst = map(lambda x: x % 24, range(4, 28))
plt.xticks(range(24), timelst)

# 그래프를 출력합니다.
plt.show()
```

```
# 총 승차와 하차 인원을 출력합니다.
print('총 승차 인원:', sum(s_in))
print('총 하차 인원:', sum(s_out))
```

1. `for i in range(24):`: 0부터 23까지의 값을 순회하는 반복문입니다. 이는 24시간 동안의 시간대를 나타냅니다.
2. `s_in[i] += row[4 + i * 2]`: 현재 시간대 `i`에 대한 승차 인원을 `row` 리스트에서 가져와서 `s_in` 리스트의 해당 시간대 인덱스에 누적합니다. 시간대마다 두 개의 열에 해당하는 데이터를 가져오기 때문에 `4 + i * 2`를 사용합니다.
3. `s_out[i] += row[5 + i * 2]`: 현재 시간대 `i`에 대한 하차 인원을 `row` 리스트에서 가져와서 `s_out` 리스트의 해당 시간대 인덱스에 누적합니다. 승차와 마찬가지로 두 개의 열에 해당하는 데이터를 가져오기 때문에 `5 + i * 2`를 사용합니다.

→

- `s_in[i] += row[4 + i * 2]`: 승차 인원 정보를 가져오기 위해 `4`를 더하고, 시간대 `i`에 대한 데이터를 추출하기 위해 `i * 2`를 더합니다.
- `s_out[i] += row[5 + i * 2]`: 하차 인원 정보를 가져오기 위해 `5`를 더하고, 시간대 `i`에 대한 데이터를 추출하기 위해 `i * 2`를 더합니다.

```
import numpy
print(numpy.sqrt(2))
```

`numpy.sqrt(2)`는 2의 제곱근을 계산하는 NumPy 함수입니다. 제곱근은 해당 숫자를 제곱했을 때 원래의 숫자가 되는 값을 의미합니다. 따라서 `numpy.sqrt(2)`는 2의 제곱근을 계산한 결과를 반환합니다.

```
import numpy as np

a = np.random.rand(6) # 5개의 [0, 1) 중의 난수
print(a)
print(type(a))
```

`umpy.random.rand(6)`는 [0, 1) 범위에서 균일한 분포를 따르는 난수를 6개 생성하는 NumPy 함수입니다. 각 난수는 독립적으로 생성되며, 생성된 난수들은 1차원 배열로 반환됩니다.

여기서 코드를 간단히 설명하면:

- `np.random.rand(6)`: 0과 1 사이에서 균일한 분포를 따르는 난수를 6개 생성합니다.
- `print(a)`: 생성된 난수들을 출력합니다.
- `print(type(a))`: `a`의 데이터 타입을 출력합니다.

따라서 `a`는 0과 1 사이의 난수로 이루어진 1차원 NumPy 배열이 됩니다. 출력 결과는 예를 들어 다음과 같을 것입니다:

np.random.rand()

`numpy.random.rand()` 함수는 [0, 1) 범위에서 균일한 분포를 따르는 난수를 생성하는 NumPy 함수입니다. 함수의 형태는 다음과 같습니다

→

```
numpy.random.rand(d0, d1, ..., dn)
```

여기서 `d0, d1, ..., dn`은 생성하고자 하는 난수의 차원을 나타냅니다. 각 차원에 대해 난수가 생성됩니다.

예를 들어:

- `numpy.random.rand(5)`: 길이가 5인 1차원 배열을 생성하고, 배열의 각 요소는 [0, 1) 범위에서 균일한 분포를 따르는 난수입니다.

- `numpy.random.rand(3, 4)`: 3행 4열의 2차원 배열을 생성하고, 배열의 각 요소는 [0, 1) 범위에서 균일한 분포를 따르는 난수입니다.

함수가 생성하는 난수는 0 이상 1 미만의 값이며, 1은 포함되지 않습니다. 이 함수는 NumPy 배열(ndarray)을 반환합니다.

```
np.random.choice(6, 10, p=[0.1, 0.1, 0.4, 0.2, 0.1, 0.1])
```

`numpy.random.choice(6, 10, p=[0.1, 0.1, 0.4, 0.2, 0.1, 0.1])`는 0부터 5까지의 숫자 중에서 10번을 선택하는데, 각 숫자가 선택될 확률이 주어진 확률 분포에 따라 결정되도록 하는 NumPy 함수입니다.

여기서 사용된 매개변수는 다음과 같습니다:

- `6`: 생성할 난수의 범위를 나타냅니다. 0부터 5까지의 숫자를 의미합니다.
- `10`: 생성할 난수의 개수를 나타냅니다. 여기서는 10개의 난수를 생성합니다.
- `p=[0.1, 0.1, 0.4, 0.2, 0.1, 0.1]`: 각 숫자가 선택될 확률을 나타내는 확률 분포를 지정합니다. 여기서는 0부터 5까지의 숫자가 선택될 확률이 `[0.1, 0.1, 0.4, 0.2, 0.1, 0.1]`으로 주어져 있습니다.

이 함수는 주어진 확률 분포에 따라 무작위로 숫자를 선택하고, 선택된 숫자들을 배열로 반환합니다. 반환된 배열은 예를 들어 `[2, 4, 3, 2, 0, 3, 2, 1, 5, 2]`와 같은 형태가 될 것입니다. 이때, 숫자 2가 상대적으로 더 자주 선택될 것이고, 숫자 0이나 5가 더 드물게 선택될 것입니다.

```
import numpy as np

# 파이썬 리스트를 NumPy 배열로 변환합니다.
a = np.array([1, 2, 3, 4])
```

```
# NumPy 배열을 출력합니다.
print(a)
```

1. `import numpy as np`: NumPy 라이브러리를 `np`라는 이름으로 임포트합니다. 이는 NumPy 함수를 사용할 때 `np`라는 접두어를 사용하기 위함입니다.
2. `np.array([1, 2, 3, 4])`: 파이썬 리스트 `[1, 2, 3, 4]`를 NumPy 배열로 변환합니다. NumPy 배열은 다차원 배열로 데이터를 저장하는 데 효율적인 자료 구조입니다.
3. `print(a)`: NumPy 배열 `a`를 출력합니다.

따라서 실행 결과는 `[1 2 3 4]`와 같이 NumPy 배열이 출력될 것입니다. NumPy 배열은 파이썬 리스트와 비슷하지만 더 많은 기능을 제공하며 벡터화 연산 등을 효율적으로 수행할 수 있습니다.

```
import numpy as np

# NumPy 배열 생성
a = np.array([1, 2, 3, 4])

# 배열의 특정 위치에 접근하여 값 출력
print(a[1], a[-1])

# 배열의 일부를 슬라이싱하여 출력
print(a[1:])
```

1. `print(a[1], a[-1])`: NumPy 배열 `a`에서 인덱스 1에 해당하는 요소와 마지막 요소(인덱스 -1)를 출력합니다. 따라서 결과는 `2 4`가 됩니다.
2. `print(a[1:])`: NumPy 배열 `a`에서 인덱스 1부터 마지막 요소까지를 슬라이싱하여 출력합니다. 따라서 결과는 `[2, 3, 4]`가 됩니다.

numpy → arange() , linspace()

`numpy.arange(start, stop, step, dtype=None)`

- `start`: 시작 값.
- `stop`: 끝 값 (이 값은 생성된 배열에 포함되지 않음). —> **미만**
- `step`: 간격(증가값). 기본값은 1.0.
- `dtype`: 배열의 데이터 타입. 기본값은 None으로, 자동으로 적절한 데이터 타입이 선택됩니다.

`arange` 함수는 `start`부터 `stop`까지 `step` 간격으로 숫자 배열을 생성합니다. `arange` 함수는 범위에 해당하는 값들을 간격만큼 증가시키면서 배열을 생성합니다.

```
import numpy as np

a = np.arange(1, 5, 0.5)
print(a)
# 출력: [1.  1.5 2.  2.5 3.  3.5 4.  4.5]
```

`numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)`

- `start`: 시작 값.
- `stop`: 끝 값.
- `num`: 생성할 숫자의 개수. 기본값은 50.
- `endpoint`: `stop` 값을 포함할지 여부. 기본값은 True로, 포함됩니다.
- `retstep`: 반환된 간격 값을 함께 반환할지 여부. 기본값은 False로, 반환되지 않습니다.
- `dtype`: 배열의 데이터 타입. 기본값은 None으로, 자동으로 적절한 데이터 타입이 선택됩니다.

`linspace` 함수는 `start`부터 `stop`까지 `num` 개의 동일한 간격으로 숫자 배열을 생성합니다. `endpoint`가 True인 경우에는 `stop` 값이 배열에 포함되고, False인 경우에는 포함되지 않습니다.

```
import numpy as np

b = np.linspace(1, 5, 10)
print(b)
# 출력: [1.         1.44444444 1.88888889 2.33333333 2.7777777
```

`linspace` 함수의 인자들은 다음과 같습니다:

- `start`: 시작 값. 여기서는 1입니다.
- `stop`: 끝 값. 여기서는 5입니다.
- `num`: 생성할 숫자의 개수. 여기서는 10입니다.

따라서 `np.linspace(1, 5, 10)`은 1부터 5까지의 범위를 10개의 동일한 간격으로 나눈 숫자 배열을 생성하는 것을 의미합니다. 이는 1에서 시작하여 5에서 끝나며, 총 10개의 숫자가 생성됩니다.

생성된 배열의 값은 간격이 균등하게 나눠져 있습니다. 위의 예시에서는 첫 번째 값이 1이고, 마지막 값이 5이며, 중간 값들은 등간격으로 배치됩니다. 즉, 1에서 5까지를 10개의 구간으로 균등하게 나눈 값들이 배열에 담겨 있습니다.

실행 결과 설명 → 여기서 각각의 값은 일정한 간격으로 증가하며, 첫 번째 값은 1이고, 마지막 값은 5입니다. 이 숫자들은 등간격으로 배치되어 있으며, 등간격의 크기는 (5 - 1) / 9 ≈ 0.44444444입니다.

mask

- "마스크(Mask)"는 조건에 따라 원본 배열의 각 요소에 대해 `True` 또는 `False`로 이루어진 배열을 생성하는 것을 의미합니다. 이렇게 생성된 불리언 배열은 주로 조건을 만족하는 요소를 선택하거나 걸러내는데 사용됩니다.
- 예를 들어, **배열의 각 요소에 대해 특정 조건을 검사하여 그 결과를 `True` 또는 `False`로 나타낸 불리언 배열이 마스크입니다.** 이 마스크를 활용하면 배열에서 특정 조건을 만족하는 요소를 선택할 수 있습니다.

```
import numpy as np

a = np.array([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4])

# 마스크 생성
mask = abs(a) > 3 # abs는 절댓값을 의미함

print(mask)
# 출력: [ True  True False False False False False False False False

# 마스크를 사용하여 조건을 만족하는 요소 선택
selected_elements = a[mask]

print(selected_elements)
# 출력: [-5 -4  4]
```

`abs`는 "절댓값(absolute value)"을 계산하는 함수입니다. 절댓값은 어떤 수의 부호를 제거한 값으로, 그 수가 양수이면 자기 자신이 되고, 음수이면 그 수의 부호를 바꾼 양수가 됩니다. 예를 들어, `abs(-3)`은 3이고, `abs(5)`는 5입니다.

주어진 코드에서 `abs(a)`는 배열 `a`의 각 요소에 대해 절댓값을 취하는 연산입니다. 따라서 `abs(a)`의 결과는 배열 `a`의 각 요소에 대한 절댓값으로 이루어진 배열이 됩니다. 이것을 통해 원래 배열의 부호를 무시하고 값만을 가지는 새로운 배열을 얻을 수 있습니다.

```
import csv
import numpy as np
import matplotlib.pyplot as plt

# CSV 파일 열기
f = open('age.csv', encoding='cp949')
data = csv.reader(f)
header = next(data)
```

```
# 데이터를 리스트로 변환
data = list(data)

# 입력받은 지역의 인구 구조를 저장할 배열 초기화
home = np.array([])

# 입력 받은 지역 이름
name = input('인구 구조가 알고 싶은 지역의 이름(읍면동 단위)을 입력해주세

# 입력 받은 지역의 인구 구조를 home 배열에 저장
for row in data:
    for i in range(102):
        row[i + 2] = int(row[i + 2].replace(',', ''))
    if name in row[0]:
        home = np.array(row[3:], dtype=int) / row[2]

# 최소값 초기화
min_val = 1
result_name = ''
result = np.array([])

# 입력 받은 지역과 가장 비슷한 인구 구조를 가진 지역 찾기
for row in data:
    if name not in row[0]:
        if row[2] == 0:
            continue

        # 현재 지역과의 유클리디안 거리 계산
        away = np.array(row[3:], dtype=int) / row[2]
        distance = np.sum((home - away) ** 2)

        # 최소 거리 갱신
        if distance < min_val:
            min_val = distance
            result_name = row[0]
            result = away
```

```
# CSV 파일 닫기
f.close()

# 시각화
plt.rc('font', family='Malgun Gothic')
plt.title(name + ' 지역과 가장 비슷한 인구 구조를 가진 지역: ' + resu
plt.plot(home, label=name)
plt.plot(result, label=result_name)
plt.xlabel('나이')
plt.ylabel('인구수')
plt.grid(True)
plt.legend()
plt.show()
```

1. `for row in data:`: 데이터에서 각 행을 하나씩 가져옵니다.
2. `for i in range(102):`: 각 행의 3번째 열부터 104번째 열까지(총 102개의 나이 그룹) 반복합니다.
    - `row[i + 2] = int(row[i + 2].replace(',', ''))`: 각 열의 값을 정수로 변환하고, 문자열에서 나온 쉼표(,)를 제거합니다. 이렇게 처리된 값을 다시 해당 위치에 할당합니다.
3. `if name in row[0]:`: 현재 행의 첫 번째 열에 입력받은 지역의 이름이 포함되어 있다면:
    - `home = np.array(row[3:], dtype=int) / row[2]`: 해당 행의 3번째 열부터 마지막 열까지의 값을 정수로 변환하고, 지역의 총 인구 수로 나누어서 `home` 배열에 저장합니다.
4. `min_val = 1`: 최소 거리를 나타내는 변수를 1로 초기화합니다.
5. `result_name = ''`: 가장 비슷한 인구 구조를 가진 지역의 이름을 저장할 변수를 빈 문자열로 초기화합니다.
6. `result = np.array([])`: 가장 비슷한 인구 구조를 가진 지역의 인구 구조를 저장할 배열을 빈 배열로 초기화합니다.
7. `for row in data:`: 데이터의 각 행을 다시 반복합니다.
8. `if name not in row[0]:`: 현재 행의 첫 번째 열에 입력받은 지역의 이름이 포함되어 있지 않다면:
    - `if row[2] == 0: continue`: 현재 행의 총 인구 수가 0이라면 다음 행으로 넘어갑니다.

- `away = np.array(row[3:], dtype=int) / row[2]`: 현재 행의 3번째 열부터 마지막 열까지의 값을 정수로 변환하고, 지역의 총 인구 수로 나누어서 `away` 배열에 저장합니다.
- `distance = np.sum((home - away) ** 2)`: 현재 지역(`home`)과 대상 지역(`away`) 간의 유클리디안 거리를 계산합니다. 거리는 각 요소 간의 차이를 제곱하고 모두 더한 값입니다.
- `if distance < min_val:`: 현재 계산한 거리가 현재까지의 최소 거리보다 작다면:
  - `min_val = distance`: 최소 거리를 현재 거리로 갱신합니다.
  - `result_name = row[0]`: 최소 거리를 가진 지역의 이름을 저장합니다.
  - `result = away`: 최소 거리를 가진 지역의 인구 구조를 저장합니다.

dtype

`dtype`**는 NumPy에서 배열의 데이터 타입을 나타내는 속성입니다.** NumPy 배열은 동일한 데이터 타입의 요소들로 이루어져야 하며, 이 데이터 타입은 배열이 메모리에 저장되는 방식을 결정합니다. `dtype` **은 데이터 타입을 지정하는 데 사용되며, 배열을 생성할 때 지정하거나 배열의 데이터 타입을 확인할 때 사용됩니다.**

```
import numpy as np

# 배열 생성 시 데이터 타입 명시
arr_int = np.array([1, 2, 3], dtype=int)
arr_float = np.array([1.1, 2.2, 3.3], dtype=float)

# 기존 배열의 데이터 타입 변경
arr_change_dtype = arr_float.astype(int)
```

리스트로 ndarray(N–Dimensional Array) 만들기

```
import numpy as np

# 리스트 생성
my_list = [1, 2, 3, 4, 5]

# 리스트를 NumPy ndarray로 변환
my_array = np.array(my_list)

# 결과 확인
print(my_array)
```

NumPy의 `array` 함수를 사용하여 리스트를 NumPy의 ndarray로 만들 수 있습니다. **이 함수는 리스트나 튜플 등의 iterable 객체를 배열로 변환합니다.**

numpy array의 인덱싱(Indexing), 슬라이싱(Slicing)

NumPy 배열의 인덱싱과 슬라이싱은 파이썬 리스트의 인덱싱과 슬라이싱과 유사하지만, 다차원 배열의 경우에도 강력하게 지원됩니다.

인덱싱(indexing)

```
import numpy as np

# 1차원 배열
arr_1d = np.array([1, 2, 3, 4, 5])

# 인덱스를 사용하여 특정 요소에 접근
print(arr_1d[0])  # 출력: 1
print(arr_1d[3])  # 출력: 4
```

```
# 음수 인덱스를 사용하여 끝에서부터 접근
print(arr_1d[-1])  # 출력: 5

# 2차원 배열
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# 인덱스를 사용하여 특정 요소에 접근
print(arr_2d[0, 0])  # 출력: 1
print(arr_2d[1, 2])  # 출력: 6
```

슬라이싱(slicing)

```
import numpy as np

# 1차원 배열
arr_1d = np.array([1, 2, 3, 4, 5])

# 슬라이싱을 사용하여 부분 배열 생성
sub_arr_1d = arr_1d[1:4]  # 인덱스 1부터 3까지의 요소를 포함한 부분
print(sub_arr_1d)  # 출력: [2, 3, 4]

# 2차원 배열
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# 슬라이싱을 사용하여 부분 배열 생성
sub_arr_2d = arr_2d[0:2, 1:3]  # 행은 0부터 1까지, 열은 1부터 2까지
print(sub_arr_2d)
# 출력:
# [[2 3]
#  [5 6]]
```

zeros( ), ones( ), eye( ) 함수로 numpy array 만들기

NumPy의 `zeros()`, `ones()`, `eye()` 함수를 사용하여 각각 0으로 채워진 배열, 1로 채워진 배열, 대각선 요소가 1이고 나머지는 0인 행렬(단위 행렬)을 생성할 수 있습니다

zeros() - 0으로 채워진 배열

```
import numpy as np

# 1차원 배열 생성 (5개의 요소를 0으로 초기화)
arr_zeros_1d = np.zeros(5)
print(arr_zeros_1d)  # 출력: [0. 0. 0. 0. 0.]

# 2차원 배열 생성 (3x4 배열을 0으로 초기화)
arr_zeros_2d = np.zeros((3, 4))
print(arr_zeros_2d)
# 출력:
# [[0. 0. 0. 0.]
#  [0. 0. 0. 0.]
#  [0. 0. 0. 0.]]
```

ones() - 1로 채워진 배열

```
import numpy as np

# 1차원 배열 생성 (3개의 요소를 1로 초기화)
arr_ones_1d = np.ones(3)
print(arr_ones_1d)  # 출력: [1. 1. 1.]

# 2차원 배열 생성 (2x3 배열을 1로 초기화)
arr_ones_2d = np.ones((2, 3))
```

```
print(arr_ones_2d)
# 출력:
# [[1. 1. 1.]
#  [1. 1. 1.]]
```

eye() - 대각선 요소는 1이고 나머지는 0인 배열

```
import numpy as np

# 3x3의 단위 행렬 생성
arr_eye = np.eye(3)
print(arr_eye)
# 출력:
# [[1. 0. 0.]
#  [0. 1. 0.]
#  [0. 0. 1.]]
```

`eye()` 함수는 단위 행렬(Identity Matrix)을 생성하는 함수입니다. 단위 행렬은 주 대각선 요소가 1이고 나머지 요소는 0인 정사각 행렬을 말합니다

13주차 -pandas

pandas 라이브러리

- panel datas 의 약자로 파이썬을 활용한 데이터 분석에서 많이 활용되고 있음
- numpy를 기반으로 만들어졌으며 데이터 분석을 위한 효율적인 데이터 구조를 제공하고 있음

Series (시리즈) - 1차원 배열 형태의 테이블 구조

- 1차원 배열 구조: 한 열(column)을 나타냅니다.
- 동일한 데이터 타입: 시리즈 내의 데이터는 동일한 데이터 타입을 가집니다.
- 레이블링: 각 데이터에 인덱스가 부여되어 있습니다.
- 크기 변경 불가능: 시리즈의 크기는 변경할 수 없습니다.
- 데이터 조작 및 분석: 시리즈를 이용해 데이터를 조작하고 분석할 수 있습니다

Data Frame (데이터 프레임) - 2차원 배열 형태의 테이블 구조

- 2차원 배열 구조: 행과 열로 이루어져 있습니다.
- 다양한 데이터 타입: 각 열은 서로 다른 데이터 타입을 가질 수 있습니다.
- 레이블링: 행과 열에 이름이나 숫자로 레이블을 부여할 수 있습니다.
- 크기 변경 가능: 행이나 열을 추가, 삭제, 수정할 수 있습니다.
- 데이터 조작 및 분석: 데이터프레임을 이용해 다양한 데이터 조작과 분석 작업을 수행할 수 있습니다.
- reshape → 2차원 배열 형태의 테이블 구조 → 기말고사에 나옴

```
import pandas as pd

# 데이터프레임 생성
df = pd.DataFrame({
    'Name': ['John', 'Alice', 'Bob'],
    'Age': [25, 30, 22],
    'Gender': ['Male', 'Female', 'Male']
})

# 시리즈 생성 (데이터프레임의 한 열)
ages = df['Age']

# 출력
print("데이터프레임:")
print(df)
print("\n시리즈:")
print(ages)
```

- 데이터프레임은 표 형태로 출력되며, 'Name', 'Age', 'Gender' 열로 구성되어 있습니다.
- 시리즈는 'Age' 열에 해당하는 데이터로, 인덱스와 함께 출력됩니다. 데이터 타입은 `int64`입니다.

행과 열 바꾸기

```
df2.T # T는 행과 열을 바꾼다는 의미의 단어인 Transpose를 의미합니다.
```

판다스 데이터프레임에서 데이터 부분 추출

```
import pandas as pd
import numpy as np

# 1에서 10 사이의 무작위 정수로 이루어진 3x4 행렬 생성
data = np.random.randint(1, 11, (3, 4))

# 데이터프레임 생성
df = pd.DataFrame(data=data, index=pd.date_range('2024-01-01'
                  columns=list('ABCD'))
# periods = 3 -> 생성할 날짜의 갯수
```

```
A  B  C  D
2024-01-01   5  7  3  6
2024-01-02   4  3  6  2
2024-01-03   5  4  9  8
```

- 날짜 인덱스는 '2024-01-01', '2024-01-02', '2024-01-03'입니다.
- 열(column)은 'A', 'B', 'C', 'D'입니다.
- 각 셀에는 1에서 10 사이의 무작위 정수 데이터가 들어있습니다.

```
df.loc['2024-01-01'] # 행 참조,
```

`loc` 메서드는 레이블을 기반으로 행이나 열을 참조하는 메서드입니다.

```
df.iloc[2, :] # index==2 인 행 모두 참조
```

`iloc` 메서드는 정수 기반의 행과 열의 인덱싱을 사용하여 데이터를 참조합니다.

`:`은 모든 열을 선택하라는 의미입니다.

to_excel() 함수 → 작업했던 데이터프레임 엑셀파일로 저장

```
import pandas as pd

df = pd.read_html('https://ko.wikipedia.org/wiki/%EC%98%AC%EB%
df2 = df[0].set_index('국가 (IOC 코드)')

summer = df2.iloc[:, :5]
summer.columns = ['경기수', '금', '은', '동', '합계']
summer = summer.sort_values('금', ascending=False)
summer.to_excel('하계올림픽메달.xlsx')
```

알고리즘 설계하기

Step 1) 데이터를 읽어온다.
ⓐ 전체 데이터를 총 인구수로 나누어 비율로 변환한다.
ⓑ 총 인구수와 연령 구간 인구수를 삭제한다.

Step 2) 궁금한 지역의 이름을 입력 받는다.

Step 3) 궁금한 지역의 인구 구조를 저장한다.

Step 4) 궁금한 지역의 인구 구조와 가장 비슷한 인구 구조를 가진 지역을 찾는다.
ⓐ 전국의 모든 지역 중 한 곳(B)를 선택한다.

ⓑ 궁금한 지역의 이름을 입력 받는다.
ⓒ ⓑ를 100세 이상 인구수에 해당하는 값까지 반복한 후 차이의 제곱을 모두 더한다.
ⓓ 전국의 모든 지역에 대해 반복하며 그 차이가 가장 작은 지역을 찾는다.

Step 5) 가장 비슷한 곳의 인구 구조와 궁금한 지역의 인구 구조를 시각화한다.

14주차

pandas -2

regex() - regular expression

```
import pandas as pd

# 샘플 데이터프레임 생성
data = {'A_1': [1, 2, 3], 'A_2': [4, 5, 6], 'B_1': [7, 8, 9]}
df = pd.DataFrame(data)

# 'A'로 시작하는 열 이름을 가진 열 선택 (정규 표현식 사용)
selected_columns = df.filter(regex='^A')

print(selected_columns)
```

Pandas에서 `regex` 옵션은 문자열을 검색하거나 패턴에 맞는 문자열을 선택할 때 정규 표현식을 사용할 수 있도록 하는 옵션입니다. `regex` 옵션은 데이터프레임의 행이나 열을 필터링하거나 변경할 때 유용하게 사용됩니다.

실행결과 →

```
A_1  A_2
0    1    4
1    2    5
2    3    6
```

df.index.str.contains( ) 함수

데이터 프레임의 인덱스 문자열로부터, 원하는 문자열이 포함된 행을 찾아냄

pandas의 Series나 DataFrame은 plot( ) 메서드를 내장하고있긔

2023-2 기말고사 힌트

교수님 피셜 기말고사에 나올것들 정리,,,,,,

np.arange() , type(np.arange() , Series , np.linspace() , 난수 , pandas DataFrame , df.~

코딩 관련한 힌트

```
'인천' in '경기도 인천시 남구'
```

```
true 출력
```

```
'경기도 인천시 남구' '인천' in
false 출력
```

```
import numpy as np
import pandas as pd
np.arange(1,5) #array(1,1.5,2,2.5,3,3.5,4,4.5 출력
```

np.linspace(n,n,n) → 결과 array

중간고사에도 나왔지만 또 나올각 ;

난수

"난수"는 무작위로 생성된 수를 나타냅니다. "난수"는 우리가 예측할 수 없고, 규칙이 없는 수를 의미합니다. 이는 일반적으로 무작위성을 도입하거나 실험에서 확률적인 요소를 모델링할 때 유용

```
np.random.randint() # 결과 ㄴ
```

```
import numpy as np

# 0 이상 1 미만의 난수로 이루어진 1차원 배열 생성
random_array = np.random.rand(5)
print(random_array)

#numpy 로 난수 생성
```

```
import random

# 0 이상 1 미만의 난수 생성
random_number = random.random()
print(random_number)

#python 으로 난수 생성
```

reshape → 2차원을 만들 수 있음

Pandas DataFrame..

```
import pandas as pd
import numpy as np

# 우선'pandas' 와 'numpy' 모듈 가져와야함
```

```
df = pd.DataFrame(np.arange(12).reshape(3, 4))
```

이 부분에서는 NumPy의 `arange` 함수를 사용하여 0부터 11까지의 숫자로 이루어진 1차원 배열을 생성하고, `reshape(3, 4)`를 통해 이를 3행 4열의 형태로 재구성합니다. 그리고 이를 `pd.DataFrame`으로 감싸 DataFrame으로 만듭니다.

```
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
```

```
#결과
```

여기서 각 행은 DataFrame의 행을 나타내고, 각 열은 DataFrame의 열을 나타냅니다. **숫자는 0부터 11까지 순서대로 채워진 값입니다.**

데이터프레임 열 이름 지정 list(' ')

```
df.columns = list('ABCD')
df


# 결과
  A B   C   D
0   0   1   2   3
1   4   5   6   7
2   8   9   10  11
```

여기서 `df.columns`는 DataFrame의 열 이름을 나타내는 속성입니다. 이 속성에 새로운 열 이름을 할당하는 것으로 열 이름을 변경할 수 있습니다. `list('ABCD')`는 문자열 'ABCD'를 리스트로 변환하는 역할을 합니다. 그 결과, 열 이름이 ['A', 'B', 'C', 'D']로 변경됩니다.

panas dataframe 에서 특정 위치의 데이터 추출 - iloc

괄호 안 첫번ㅉㅐ 값은 행 , 두번째 값은 열 선택

```
type(df.iloc[1: , 1])
# type 이라 붙이면 결과값 시리즈임

# 결과
1    5
```

```
2    9
Name: B, dtype: int64
```

여기서 `1:`은 1번 행부터 끝까지의 모든 행을 선택하고, `,` 뒤에 있는 `1`은 1번 열을 선택합니다. 따라서 이 코드는 DataFrame에서 1번 행부터 끝까지의 모든 행을 가져와서 그 중 1번 열의 데이터를 추출합니다.

선택된 데이터 타입 반환 - type

```
type(df.iloc[1:, 1])

# 결과
pandas.core.series.Series
```

여기서 반환된 결과는 **Pandas Series로,** 1번 행부터 끝까지의 1번 열 데이터를 담고 있습니다. 이 **Series의 인덱스는 원래 DataFrame의 행 인덱스(1, 2)이며, 데이터는 해당 위치에 있는 값입니다.**

pandas data frame 다차원 배열로 반환 - df.to_numpy()

```
# 예시코드
import pandas as pd
import numpy as np

df = pd.DataFrame(np.arange(12).reshape(3, 4), columns=['A',
```

결과

```
array([[ 0,  1,  2,  3],
[ 4,  5,  6,  7],
[ 8,  9, 10, 11]])
```

이 NumPy 배열은 원래 DataFrame의 데이터를 그대로 포함하며, 따라서 DataFrame과 NumPy 배열 간의 데이터 이동이 가능합니다. `to_numpy()`를 사용하면 **Pandas DataFrame을 NumPy 배열로 효율적으로 변환할 수 있습니다.**

Pandas dataframe에 있는 데이터를 numpy 배열로 반확 속성

`df.to_numpy()`와 유사하게 DataFrame의 데이터를 NumPy 배열로 추출합니다.

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.arange(12).reshape(3, 4), columns=['A',
```

```
array([[ 0,  1,  2,  3],
[ 4,  5,  6,  7],
[ 8,  9, 10, 11]])
```

`df.values`와 `df.to_numpy()`는 대부분의 경우 동일한 결과를 제공하지만, 몇 가지 차이점이 있습니다. 주로 데이터 타입이나 복사 방식 등에서 차이가 발생할 수 있습니다. 일반적으로 두 방법 중 하나를 사용하여 Pandas DataFrame을 NumPy 배열로 변환할 수 있습니다.

pandas dataframe 다차원 배열 반환 → df.values() , df.to_numpy()

df.index - Dataframe 행 인덱스 확인

df.columns - Datafream 열 이름 나타내는 속성]

Numpy 라이브러리로 1차원 배열 생성

```
d = np.arange(10)
```

`np.arange(10)`은 0부터 9까지의 정수로 이루어진 1차원 배열을 생성합니다. 결과를 변수 `d`에 할당하므로, `d`는 다음과 같은 배열을 가지게 됩니다:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

이제 `d`를 출력하면 해당 배열이 표시됩니다 `print(d)`

결과 → `[0 1 2 3 4 5 6 7 8 9]`

numpy 배열 사용하여 조건을 만족하는 불리언(Boolean) 배열 생성 - cond

```
cond = d > 5
cond
```

여기서 `d > 5`는 각 원소가 5보다 큰지를 검사한 결과를 나타내는 불리언(Boolean) 배열을 생성합니다. 이 불리언 배열을 변수 `cond`에 할당합니다.

만약 d가 다음과 같은 배열이었다면

`array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])`

출력결과

`array([False, False, False, False, False, False, True, True, True, True])`

```
d[cond]
# 불리언 배열을 사용하여 numpy 배열을 필터링 함
# cond는 이 배열에 대한 조건을 나타내는 불리언(Boolean) 배열로, 각 원소
```

`d[cond]`는 이 불리언 배열을 사용하여 `True`에 해당하는 인덱스의 원소만을 선택하는 연산입니다. 따라서, 결과는 다음과 같은 배열이 됩니다:

매핑 (mapping) - map()

매핑 (mapping) 설명 : 매핑(mapping)은 하나의 집합(또는 범위)의 각 원소를 다른 집합(또는 도메인)의 원소와 연결시키는 일반적인 개념입니다.

프로그래밍에서 매핑은 주로 함수와 연관이 있습니다.

함수는 하나의 입력(도메인의 원소)을 받아서 출력(공역의 원소)을 반환하는 매핑의 한 예입니다.

매핑

ㅡ반복적인 개체의 각 항목에 변환 함수를 적용해 새로운 반복가능 개체로 변환해야 할 때 유용
ㅡ내장 함수 map() 사용

내장 함수 map()
ㅡ반복 구문을 사용하지 않고 반복가능(iterable) 컨테이너(container)의 모든 항목(item)을 특정 함수의 인자로 처리할 수 있는 함수

두개의 인자를 받아서 더한 값 반환 → 함수 addsome

```
def addsome(a, b):
    result = a + b
    return result
```

이 함수는 `a`와 `b`라는 두 매개변수(인자)를 받아서 덧셈 연산을 수행하고, 그 결과를 `result`라는 변수에 저장한 후 반환합니다.

이제, 다음과 같이 리스트를 만들고 `map()` 함수를 사용하여 `addsome` 함수를 리스트의 각 요소에 적용합니다:

```
# addsome 함수 정의
def addsome(a, b):
    result = a + b
    return result

# 리스트 생성
numbers1 = [1, 2, 3, 4]
numbers2 = [5, 6, 7, 8]

# map() 함수를 사용하여 addsome 함수를 각 요소에 적용
result = map(addsome, numbers1, numbers2)

# 결과 출력 (리스트로 변환하여 출력)
print(list(result))
```

위 코드에서 `map(addsome, numbers1, numbers2)`는 `addsome` 함수를 `numbers1`과 `numbers2` 리스트의 각 요소에 적용합니다. 결과는 두 리스트의 각 요소를 더한 값인 `[6, 8, 10, 12]`가 됩니다.

문자열에서 특정 문자를 다른 문자로 대체 - myreplace()

우선 myreplace() 함수 정의

```
def myreplace(input_str, target_char, replacement_char):
    """
    문자열에서 특정 문자를 다른 문자로 대체하는 함수
    """
    result_str = input_str.replace(target_char, replacement_c
    return result_str
```

이제, 다음과 같이 리스트를 만들고 `map()` 함수를 사용하여 `myreplace()` 함수를 리스트의 각 문자열에 적용합니다:

```
# myreplace 함수 정의
def myreplace(input_str, target_char, replacement_char):
    """
    문자열에서 특정 문자를 다른 문자로 대체하는 함수
    """
    result_str = input_str.replace(target_char, replacement_c
    return result_str

# 리스트 생성
string_list = ["apple", "banana", "cherry"]

# map() 함수를 사용하여 myreplace 함수를 각 문자열에 적용
result = map(myreplace, string_list, ["a", "a", "c"], ["@", "

# 결과 출력 (리스트로 변환하여 출력)
print(list(result))
```

위 코드에서 `map(myreplace, string_list, ["a", "a", "c"], ["@", "!", "&"])`는 `myreplace` 함수를 `string_list` 리스트의 각 문자열에 적용합니다. 대체할 대상 문자와 대체 문자는 각각 `["a", "a", "c"]`와 `["@", "!", "&"]`로 지정되어 있습니다. 결과는 각 문자열에 대체가 적용된 리스트 `['@pple', 'b@n@n@', '&herry']`가 됩니다.

내장함수 - int()

```
# 문자열의 리스트 생성
str_numbers = ["1", "2", "3", "4", "5"]

# map() 함수와 int() 함수를 사용하여 문자열을 정수로 변환
int_numbers = map(int, str_numbers)

# 결과 출력 (리스트로 변환하여 출력)
print(list(int_numbers))
```

위 코드에서 `map(int, str_numbers)`는 `str_numbers` 리스트의 각 문자열에 `int()` 함수를 적용하여 정수로 변환합니다. 결과는 `[1, 2, 3, 4, 5]`가 됩니다.

이렇게 `map()` 함수와 내장 함수 `int()`를 함께 사용하면, 반복 가능한(iterable) 데이터 구조에서 일괄적으로 데이터 타입을 변환하는 작업을 효율적으로 수행할 수 있습니다.

scatter()

`x = np.arange(1,5, .5`) → .5는 사이즈를 의미함