2022/08/20

1주차 정규세션

데이터 분석을 위한 패키지

10기 교육부 이소현, 유근태

목차

01

배열 다루기

numpy

numpy란?

파이썬에서 배열을 사용하기 위한 표준 패키지

수치 해석용 파이썬 패키지로, 벡터/행렬 사용하는 선형대수 계산에 주로 사용

numpy의 사용

```
1 import numpy as np
```

numpy 패키지를 np로 줄여서 사용

배열이란?

같은 타입의 변수들로 이루어진 유한 집합

배열 요소: 배열을 구성하는 각각의 요소

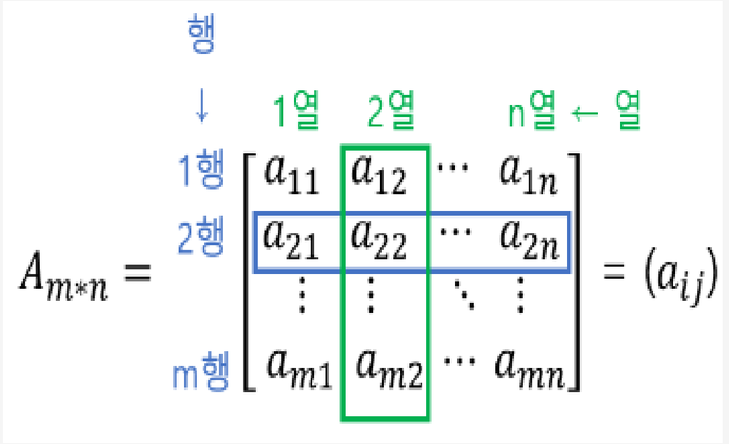
인덱스: 배열에서의 위치를 나타내는 숫자 항상 0부터 시작하며, 0을 포함한 양의 정수만을 가짐

배열의 생성

```
ar = np.array([0, 1, 2, 3, 4, 5])
ar # 출력 결과 array([0, 1, 2, 3, 4, 5])
```

행렬(matrix) == 다차원 배열

수 또는 다항식을 직사각형 모양으로 배열한 것

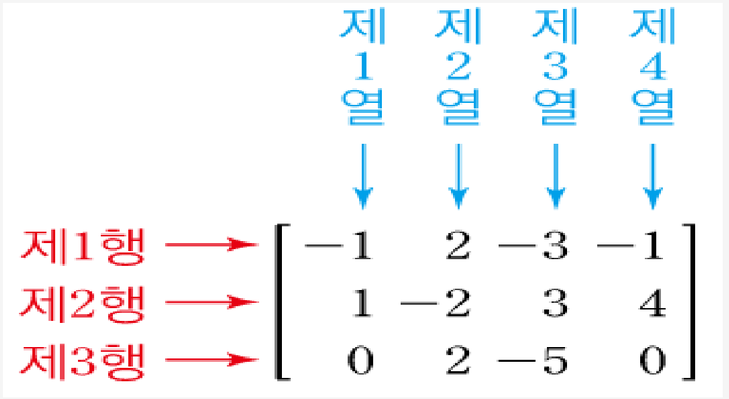


ex)

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

행(row)의 개수: len(c) # 출력 결과 3

열(column)의 개수: len(c[0]) # 출력 결과 4



1차원 배열 인덱싱

인덱스 번호가 0부터 시작함

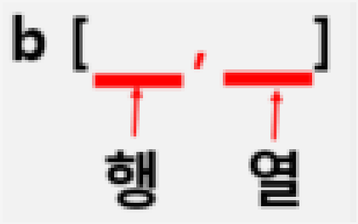
뒤에서부터 거꾸로 셀 때는 -1부터 시작함

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

다차원 배열 인덱싱

콤마(,)를 사용하여 접근

콤마 앞에는 행, 뒤는 열을 나타냄




```
b[0, 0] # 첫번째 행의 첫번째 열
b[0, -1] # 첫번째 행의 마지막 열
b[-1, -1] # 마지막 행의 마지막 열
```

* 인덱싱(indexing) : 데이터에 빠르게 접근할 수 있도록 인덱스를 사용하는 것

fancy 인덱싱

boolean 값을 가지는 다른 numpy 배열로 배열을 인덱싱
boolean 값을 가진 배열을 사용하여 직관적으로 인덱싱 가능

---

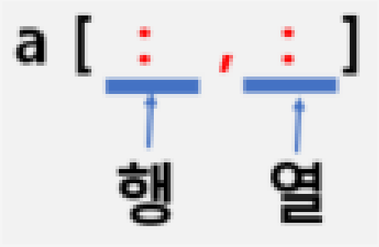
ex)

```
array = np.array([i for i in range(5)])  # array([0, 1, 2, 3, 4])
index = np.array([True, False, True, False, True])
print(array[index]) # 출력 결과 [0, 2, 4]
```

배열 슬라이싱

배열의 원소 중 복수개를 접근

ex)




```
a[0, :]   # 첫번째 행 전체
a[:, 1]   # 두번째 열 전체
a[1, 1:]  # 두번째 행의 두번째 열부터 마지막 열까지
```

zeros

크기를 선언해주면, 모든 값이 0인 배열 생성

```
a = np.zeros(5)
a  # 출력 결과 array([0., 0., 0., 0., 0.]]
```

ones

크기를 선언해주면, 모든 값이 1인 배열 생성

```
b = np.ones((5,2), dtype = "i")
b  # 출력 결과
   array([0, 0],
         [0,0],
         [0,0],
         [0,0], dtype = int32)
```

zeros_like

다른 배열과 같은 크기, 모든 값이 0인 배열 생성

```
az = np.zeros_like(a, dtype = "f")
az # 출력 결과 array([0., 0., 0., 0., 0.], dtype=float32]
```

ones_like

다른 배열과 같은 크기, 모든 값이 1인 배열 생성

```
ao = np.ones_like((a, dtype = "i")
ao  # 출력 결과 array([1, 1, 1, 1, 1], dtype = int32)
```

empty

배열 생성만 하고, 특정 값으로 초기화하지 않음

```
a = np.empty((3,2))
a   # 출력 결과
array([[9.34577196e-307, 9.34598246e-307],
       [1.60218491e-306, 1.69119873e-306],
       [1.24611673e-306, 1.11262266e-307]])
```

arange

특정 수열을 만들려고 할 때 사용

```
np.arange(9)   # 출력 결과 array([0, 1, 2, 3, 4, 5, 6, 7, 8])
np.arange(3, 18, 3) # 출력 결과 array([ 3, 6, 9, 12, 15])
```

* 수열 : 수 또는 다른 대상의 순서 있는 나열

linspace

선형 구간 지정한 구간 수만큼 분할

```
np.linspace(0, 100, 5)
# 출력 결과 array([ 0., 25., 50., 75., 100.])
```

logspace

로그 구간 지정한 구간 수만큼 분할

```
np.logspace(0.1, 1, 10)

# 출력 결과
array([ 1.25892541, 1.58489319, 1.99526231, 2.51188643, 3.16227766, 3.98107171, 5.01187234, 6.30957344, 7.94328235, 10. ])
```

2차원 배열의 전치 작업

전치(Transpose): 행과 열을 바꾸는 작업

ex)

A = np.array([1, 2, 3], [4, 5, 6])

A.T # 출력 결과 array([1, 4],
[2, 5],
[3, 6])

reshape()

만들어진 배열 내부 데이터를 보존한 채로 형태만 변경

```
a = np.array([0, 1, 2, 3])
b = a.reshpae(2,2) # 출력 결과 array([0, 1],
                                [2, 3]])
```

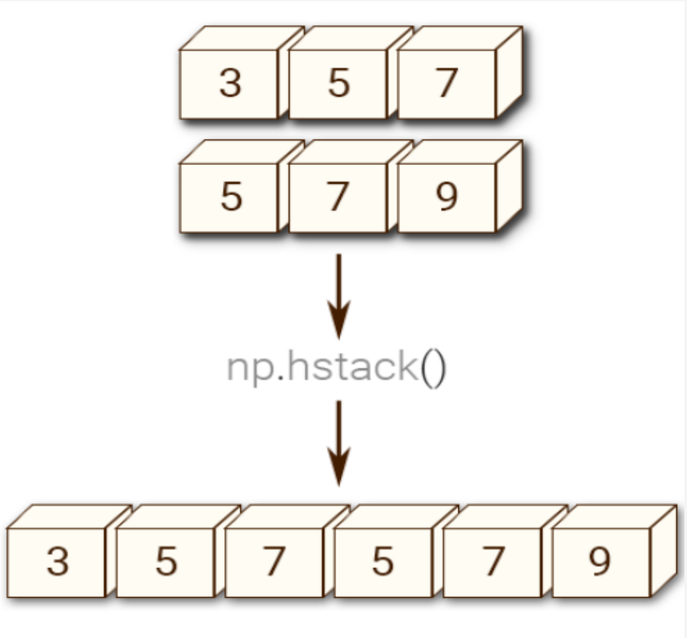
flatten()

다차원 배열을 1차원으로 변경

```
c = b.flatten() # 출력 결과 array([0, 1, 2, 3])
```

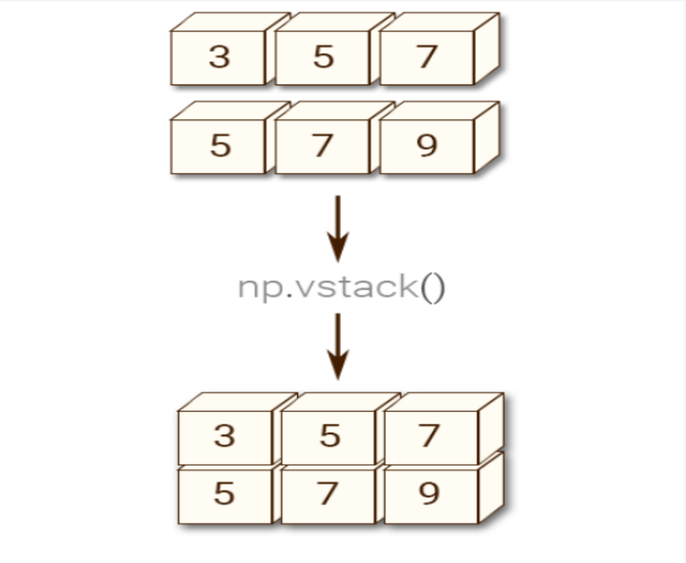
hstack()

horizontal, 수평으로 행렬 연결
행의 수가 같은 2개 이상의 배열을 옆으로 연결



vstack()

vertical, 수직으로 행렬 연결
열의 수가 같은 2개 이상의 배열 아래로 연결



배열의 사칙연산

기본적으로 행렬의 shape가 같을 때 배열 간의 사칙연산을 지원함

A = np.array([[1, 2, 3], [4, 5, 6]])

ex)

A + A

```
# 출력결과
array([[2, 4, 6],
       [8, 10, 12]])
```

A - A

```
# 출력결과
array([[0, 0, 0],
       [0, 0, 0]])
```

A * A

```
# 출력결과
array([[1, 4, 9],
       [16, 25, 36]])
```

A / A

```
# 출력결과
array([[1., 1., 1.],
       [1., 1., 1.]])
```

행렬 곱 (Dot product)

조건) 행렬 A의 열의 개수와 B의 행의 개수가 같을 때

행렬 A의 제i행의 각 성분과 행렬 B의 제j행 열의 각 성분을 순서대로 곱하여 더한 값

np.dot()

---

ex)

$$A \times B = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

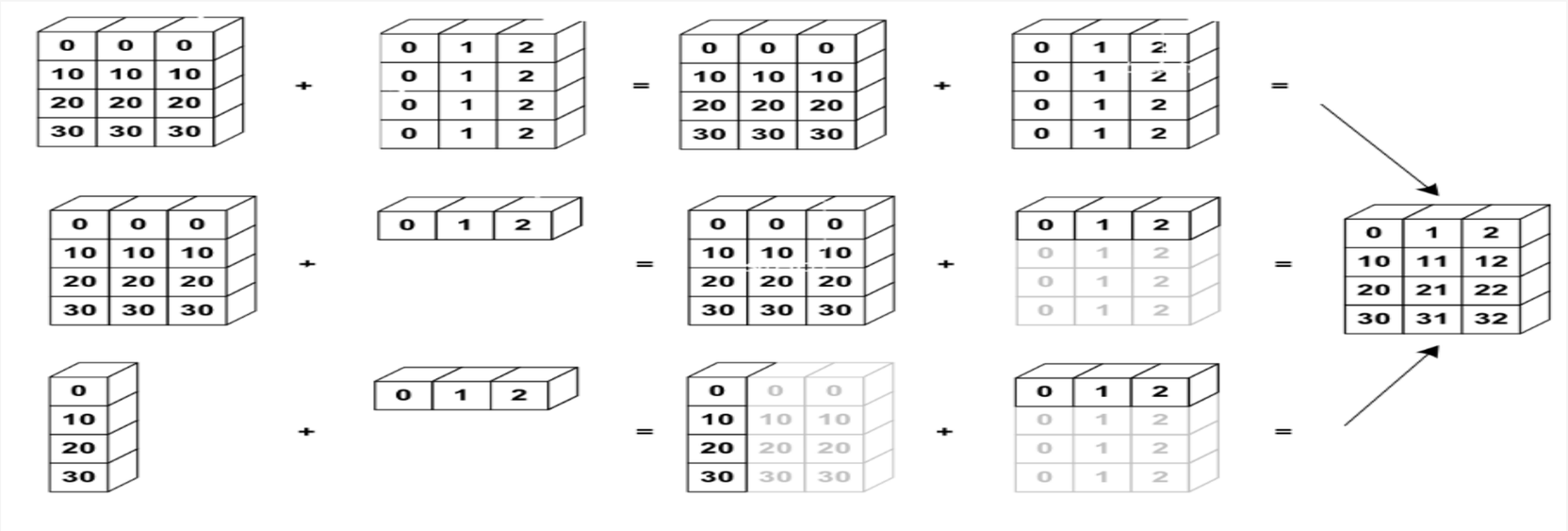
$$= \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{pmatrix}$$

브로드캐스팅(broadcasting)

t서로 다른 크기를 가진 두 배열의 사칙연산 지원

크기가 작은 배열을 자동으로 반복 확장하여 큰 배열에 맞춤

ex)



02

데이터 분석

pandas

pandas란?

고수준의 자료 구조와 빠르고 쉬운 데이터 분석 도구를 제공하는 파이썬 라이브러리
pandas의 자료구조에는 series, dataframe이 있다.

pandas의 사용

```
1 import pandas as pd
```

pandas 패키지를 pd로 줄여서 사용

series 생성

일련의 객체를 담을 수 있는 1차원 배열 같은 구조

```
s = pd.Series([9904312, 3448737, 2890451],
              index = ["서울", "부산", "인천"])
s  # 출력 결과 서울 9904312
           부산 3448737
           인천 2890451
           dtype: int64
```

series의 속성

index : 인덱스 접근

values : 값 접근

name : 데이터 / 인덱스 이름 붙이기 기능

```
s.index  # 출력 결과 Index(['서울', '부산', '인천'], dtype='object')
s.values
# 출력 결과 array([9904312, 3448737, 2890451], dtype=int64)
```

series 연산

연산은 series 값에만 적용, 인덱스 값은 적용 X

ex)

```
s = pd.Series([9904312, 3448737, 2890451],
              index = ["서울", "부산", "인천"])

s / 100000 # 출력 결과 서울 9.904312
                       부산 3.448737
                       인천 2.890451
                       dtype: float64
```

series 인덱싱

numpy 배열에서 사용한 인덱싱/슬라이실 방법 사용 가능

인덱스 라벨을 사용하여 인덱싱 슬라이싱 가능 (콜론 뒤 값 포함)

---

ex)

```
s[1:3]  # 출력 결과 부산 3448737
                  인천 2890451
                  dtype: int64
```

```
s['서울':'인천']  # 출력 결과 서울 9904312
                          부산 3448737
                          인천 2890451
                          dtype: int64
```

데이터의 갱신, 추가, 삭제

기존

```
rs = pd.Series([1.636984, 2.832690, 9.818107],
               index = ["서울", "부산", "인천"])

# 출력 결과 서울 1.636984
            부산 2.832690
            인천 9.818107
```

갱신

```
rs['부산'] = 1.63 # 출력 결과 서울 1.636984
                            부산 1.630000
                            인천 9.818107
                            dtype: float64
```

추가

```
rs['대구'] = 1.41 # 출력 결과 서울 1.636984
                            부산 1.630000
                            인천 9.818107
                            대구 1.410000
```

삭제

```
del rs['부산']      # 출력 결과 서울 1.636984
                            인천 9.818107
                            대구 1.410000
```

dataframe

2차원 자료 구조

STEP1. 딕서너리 형태로 데이터를 정의해준다.
(키 값의 개수가 열의 개수가 되고, 리스트의 성분의 개수가 행의 개수가 된다.)

STEP2. 만들어진 딕셔너리를 pd.Dataframe() 인자에 넣는다.

ex)

```
data = {"2015" : [9904312, 3448737, 2890451, 2466052],
        "2010" : [9631482, 3393191, 2632035, 2431774],
        "2005" : [9762546, 3512547, 2517680, 2456016],
        "2000" : [9853972, 3655437, 2466338, 2473990],
        "지역" : ["수도권", "경상권", "수도권", "경상권"],
        "2010-2015 증가율" : [0.0283, 0.0163, 0.0982, 0.0141]}
columns = ["지역", "2015", "2010", "2005", "2000", "2010-2015 증가율"]
index = ["서울", "부산", "인천", "대구"]
df = pd.DataFrame(data, index = index, columns = columns)
df
```

	지역	2015	2010	2005	2000	2010-2015 증가율
서울	수도권	9904312	9631482	9762546	9853972	0.0283
부산	경상권	3448737	3393191	3512547	3655437	0.0163
인천	수도권	2890451	2632035	2517680	2466338	0.0982
대구	경상권	2466052	2431774	2456016	2473990	0.0141

데이터의 갱신, 추가, 삭제

기존

	지역	2015	2010	2005	2000	2010-2015 증가율
서울	수도권	9904312	9631482	9762546	9853972	0.0283
부산	경상권	3448737	3393191	3512547	3655437	0.0163
인천	수도권	2890451	2632035	2517680	2466338	0.0982
대구	경상권	2466052	2431774	2456016	2473990	0.0141

갱신

```
df["2010-2015 증가율"] = df["2010-2015 증가율"] * 100
df
```

	지역	2015	2010	2005	2000	2010-2015 증가율
서울	수도권	9904312	9631482	9762546	9853972	2.83
부산	경상권	3448737	3393191	3512547	3655437	1.63
인천	수도권	2890451	2632035	2517680	2466338	9.82
대구	경상권	2466052	2431774	2456016	2473990	1.41

추가

```
df["2005-2010 증가율"] = ((df["2010"] - df["2005"]) / df["2005"] * 100)
df
```

	지역	2015	2010	2005	2000	2010-2015 증가율	2005-2010 증가율
서울	수도권	9904312	9631482	9762546	9853972	2.83	-1.342519
부산	경상권	3448737	3393191	3512547	3655437	1.63	-3.397990
인천	수도권	2890451	2632035	2517680	2466338	9.82	4.542078
대구	경상권	2466052	2431774	2456016	2473990	1.41	-0.987046

삭제

```
del df["2000"]
df
```

	지역	2015	2010	2005	2010-2015 증가율	2005-2010 증가율
서울	수도권	9904312	9631482	9762546	2.83	-1.342519
부산	경상권	3448737	3393191	3512547	1.63	-3.397990
인천	수도권	2890451	2632035	2517680	9.82	4.542078
대구	경상권	2466052	2431774	2456016	1.41	-0.987046

열 인덱싱

```
1 df["지역"]
```

```
서울      수도권
부산      경상권
인천      수도권
대구      경상권
Name: 지역, dtype: object
```

하나의 열만 인덱싱하면,
시리즈 반환

```
1 df[["2010", "2015"]]
```

	2010	2015
서울	9631482	9904312
부산	3393191	3448737
인천	2632035	2890451
대구	2431774	2466052

여러개 열 인덱싱하면,
부분적인 데이터프레임 반환

```
1 df[["2010"]]
```

	2010
서울	9631482
부산	3393191
인천	2632035
대구	2431774

하나의 열 인덱싱하면서,
데이터프레임 반환

행 인덱싱

numpy 배열에서 사용한 인덱싱/슬라이실 방법 사용 가능

인덱스 라벨을 사용하여 인덱싱 슬라이싱 가능 (콜론 뒤 값 포함)

```
1 df[1:2]
```

	지역	2015	2010	2005	2010-2015 증가율	2005-2010 증가율
부산	경상권	3448737	3393191	3512547	1.63	-3.39799

```
1 df["서울":"부산"]
```

	지역	2015	2010	2005	2010-2015 증가율	2005-2010 증가율
서울	수도권	9904312	9631482	9762546	2.83	-1.342519
부산	경상권	3448737	3393191	3512547	1.63	-3.397990

loc

라벨 값 기반의 2차원 인덱싱

df2

	A	B	C	D
a	10	11	12	13
b	14	15	16	17
c	18	19	20	21

ex)

df.loc[행 인덱싱 값]

df.loc[행 인덱싱 값, 열 인덱싱 값]

```
1  df2.loc[df2.A > 15]
```

	A	B	C	D
c	18	19	20	21

```
1  df2.loc[["a", "b"], ["B", "D"]]
```

	B	D
a	11	13
b	15	17

iloc

순서를 나타내는 정수 기반의 2차원 인덱싱

df2

	A	B	C	D
a	10	11	12	13
b	14	15	16	17
c	18	19	20	21

ex)

df.iloc[행 인덱싱 값]

df.iloc[행 인덱싱 값, 열 인덱싱 값]

```
1  df2.iloc[-1]
```

```
A     18
B     19
C     20
D     21
Name: c, dtype: int64
```

```
1  df2.iloc[0:2, 1:4]
```

	B	C	D
a	11	12	13
b	15	16	17

count()

데이터 개수 셀 때 사용

```
1  df.count()
```

```
지역                    4
2015                    4
2010                    4
2005                    4
2000                    4
2010-2015 증가율        4
dtype: int64
```

value_count()

카테고리 값 셀 때 사용

데이터프레임에서는 각 열마다 별도로 적용해야 한다.

```
1  df["지역"].value_counts()
```

```
수도권    2
경상권    2
Name: 지역, dtype: int64
```

sort_index()

인덱스 값을 기준으로 정렬

```
1 df.sort_index()
```

	지역	2015	2010	2005	2000	2010-2015 증가율
대구	경상권	2466052	2431774	2456016	2473990	0.0141
부산	경상권	3448737	3393191	3512547	3655437	0.0163
서울	수도권	9904312	9631482	9762546	9853972	0.0283
인천	수도권	2890451	2632035	2517680	2466338	0.0982

sort_values()

데이터 값을 기준으로 정렬

NaN 값이 있는 경우, NaN 값이 가장 마지막으로 간다.

```
1 df["2015"].sort_values()
```

```
대구      2466052
인천      2890451
부산      3448737
서울      9904312
Name: 2015, dtype: int64
```

03

데이터 시각화

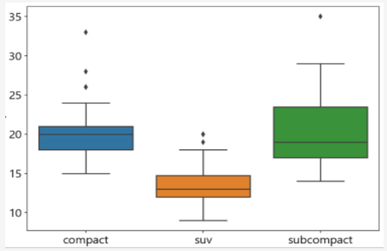
matplotlib

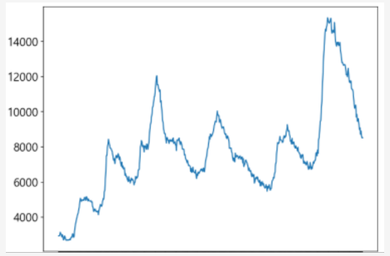
데이터 시각화

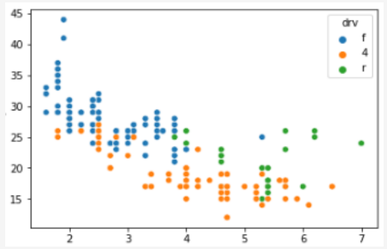
시각적 이미지를 사용하여 데이터를 화면에 표시하는 것

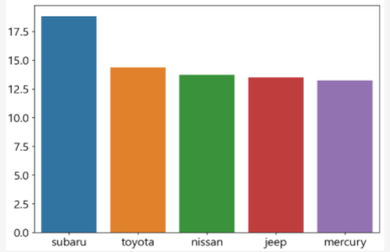
PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings, Mrs. John Bradley (Florence	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925		S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	0	3	Moran, Mr. James	male		0	0	330877	8.4583		Q
7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.075		S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhe	female	27	0	2	347742	11.1333		S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708		C
11	1	3	Sandstrom, Miss. Marguerite Rut	female	4	1	1	PP 9549	16.7	G6	S
12	1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.55	C103	S
13	0	3	Saundercock, Mr. William Henry	male	20	0	0	A/5. 2151	8.05		S
14	0	3	Andersson, Mr. Anders Johan	male	39	1	5	347082	31.275		S
15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14	0	0	350406	7.8542		S
16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55	0	0	248706	16		S
17	0	3	Rice, Master. Eugene	male	2	4	1	382652	29.125		Q
18	1	2	Williams, Mr. Charles Eugene	male		0	0	244373	13		S
19	0	3	Vander Planke, Mrs. Julius (Emelia Mari	female	31	1	0	345763	18		S
20	1	3	Masselmani, Mrs. Fatima	female		0	0	2649	7.225		C
21	0	2	Fynney, Mr. Joseph J	male	35	0	0	239865	26		S

데이터 원자료나 통계표는 수많은 문자로 구성되어 있어서 내용을 파악하는 것이 어려움









추세와 경향성이 드러나기 때문에 특징을 쉽게 이해할 수 있고, 새로운 패턴을 발견할 수 있음

matplotlib란?

그래프 생성을 통한 데이터 시각화에 사용되는 파이썬 패키지

그래프 구성 요소를 자유자재로 커스터마이즈 가능

matplotlib의 사용

```
1 import matplotlib.pyplot as plt
```

matplotlib 패키지의 pyplot 함수를 plt로 줄여서 사용

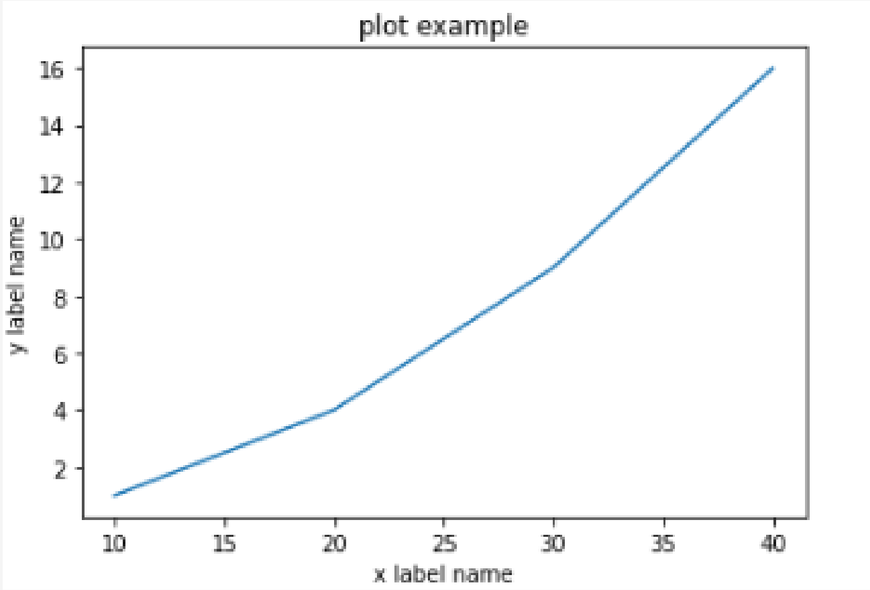
lineplot

선 그래프 만드는 함수

시간, 순서에 따라 데이터가 어떻게 변화하는지 보여주기 위해 사용

* title(): 그래프 제목 표시
* xlabel(): x축 레이블 표시
* ylabel(): y축 레이블 표시
* show(): 그래프를 화면에 나타나게 함

```
1 plt.title('plot example')
2 plt.xlabel('x label name')
3 plt.ylabel('y label name')
4 plt.plot([10, 20, 30, 40], [1, 4, 9, 16])
5 plt.show()
```

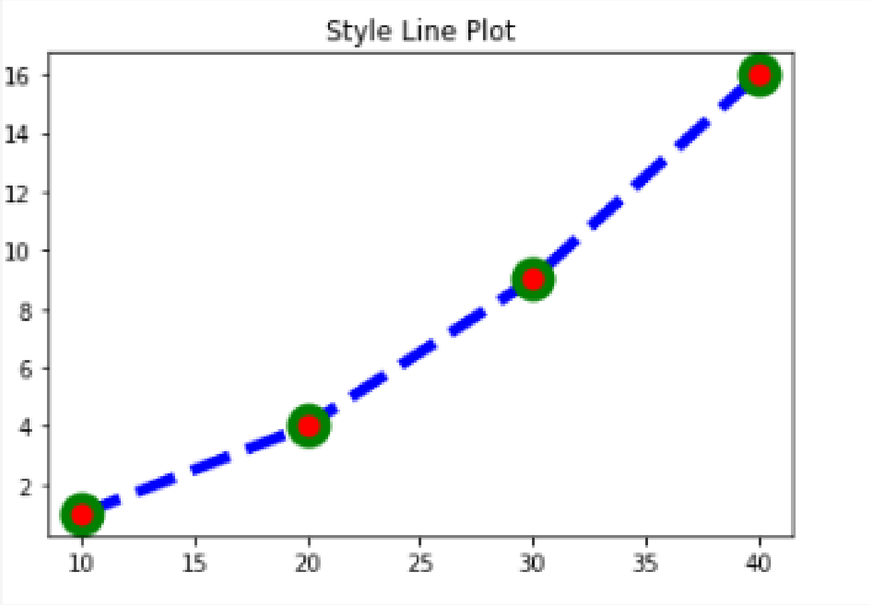


plt.plot([10, 20, 30, 40], [1, 4, 9, 16])

lineplot

* c : 선 색깔 지정
* ls: 선 스타일 (line style)
* lw: 선 굵기 (line width)
* marker: 마커 지정하기
* ms: 마커 크기(marker size)
* mec: 마커 선 색깔
* mew: 마커 선 굵기
* mfc: 마커 내부 색깔

```
plt.plot([10, 20, 30, 40], [1, 4, 9, 16], c = 'b', lw = 5, ls = '--',
         marker = 'o', ms = 15, mec = 'g', mew = 5, mfc = 'r')
plt.title('Style Line Plot')
plt.show()
```



plt.plot([10, 20, 30, 40], [1, 4, 9, 16], c = 'b', lw = 5, ls = '--', marker = 'o', ms = 15, mec = 'g', mew = 5, mfc = 'r')

Colors

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Line Styles

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

Markers

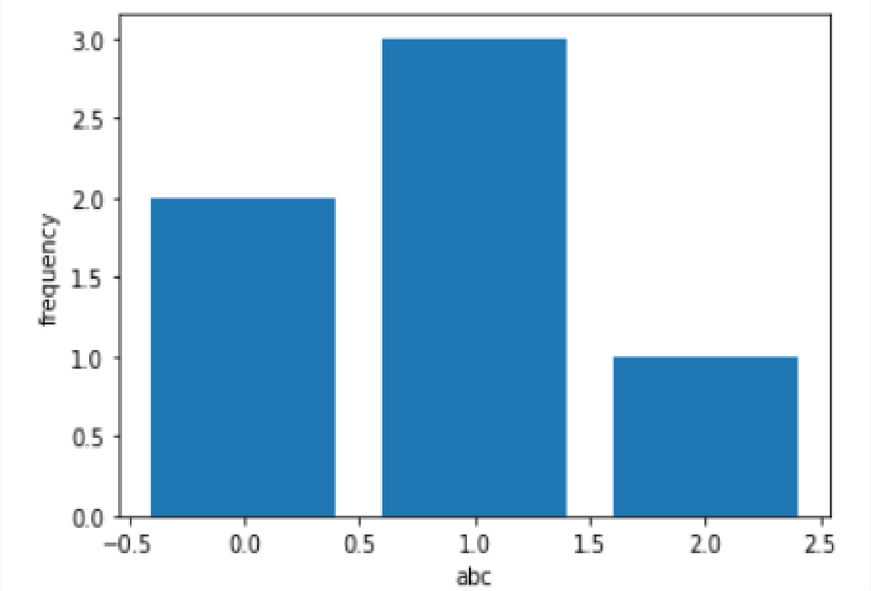
character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

barplot

막대 그래프 만드는 함수

집단 간 차이를 표현할 때 주로 사용

```
1  x = np.arange(len(y))
2  y = [2,3,1]
3  plt.bar(x,y)
4  plt.xlabel('abc')
5  plt.ylabel('frequency')
6  plt.show()
```



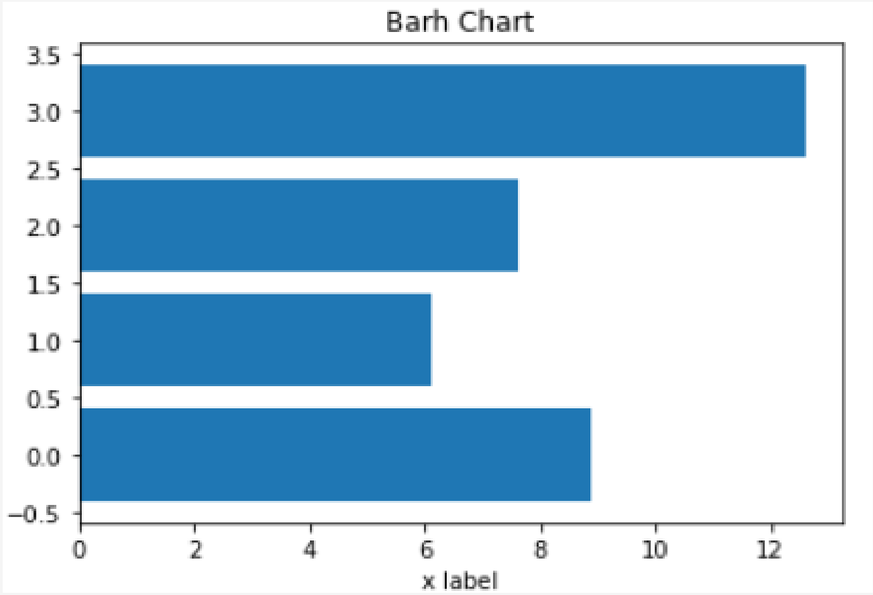
plt.bar(x, y)

barhplot

수평 막대 그래프 만드는 함수

집단 간 차이를 표현할 때 주로 사용

```
people = ['A', 'B', 'C', 'D']
y_pos = np.arange(len(people))
performance = 3 + 10 * np.random.rand(len(people))
plt.title('Barh Chart')
plt.barh(y_pos, performance)
plt.xlabel('x label')
plt.show()
```



plt.barh(y_pos, performance)

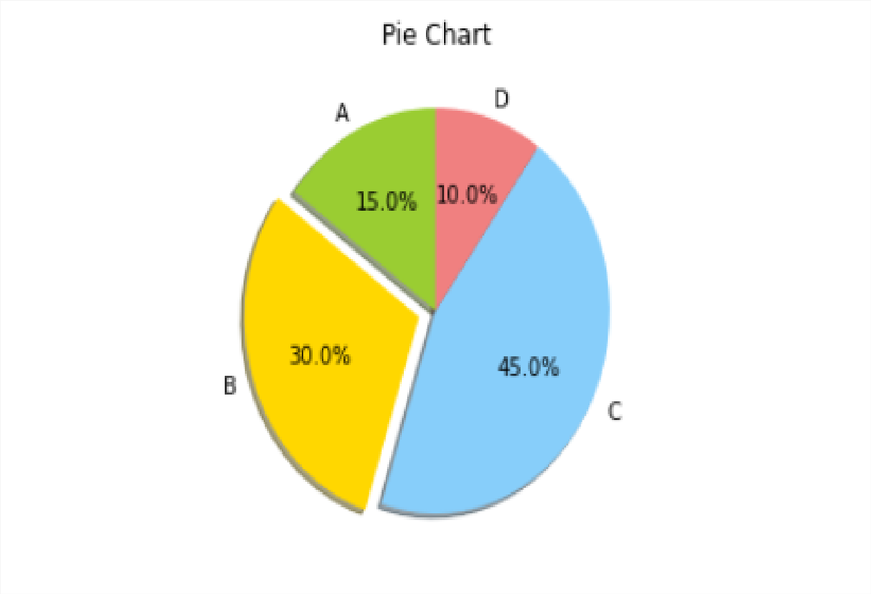
pie chart

원 그래프를 만드는 함수

카테고리 별, 상대적인 비율을 나타냄

* explode: 부채꼴이 파이 차트의 중심에서 벗어나는 정도
* autopct: 부채꼴 안에 표시될 숫자 형식 지정
* shadow: True 설정 시 파이 차트 그림자 표시
* startangle: 부채꼴 그려지는 시작 각도 설정

```
labels = ['A', 'B', 'C', 'D']
sizes = [15, 30, 45, 10]
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
explode = [0, 0.1, 0, 0]
plt.title('Pie Chart')
plt.pie(sizes, explode = explode, labels = labels, colors = colors,
        autopct = '%1.1f%%', shadow = True, startangle = 90)
plt.show()
```



plt.pie(sizes, explode = explode, labels = labels, colors = colors, autopct = '%1.1f%%', shadow = True, startangle = 90

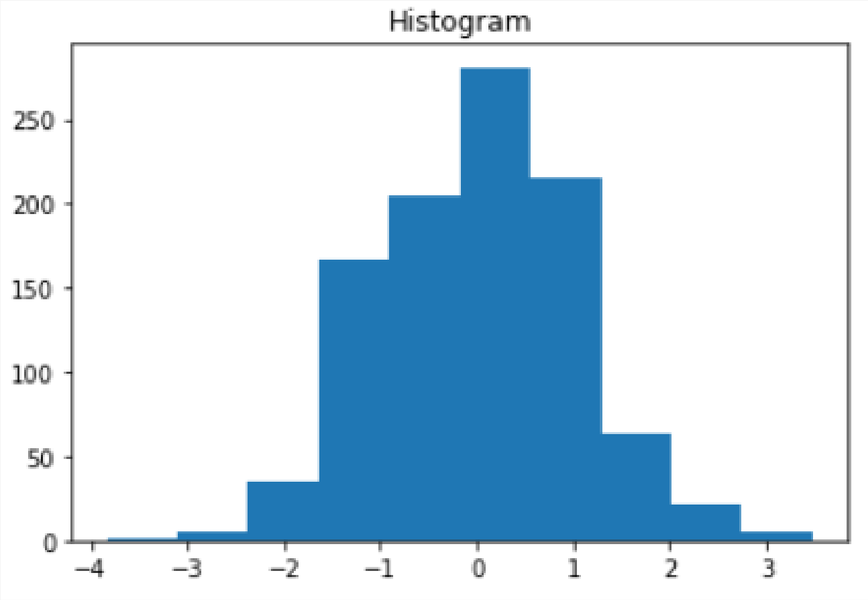
histogram

도수분포표를 그래프로 나타낸 것

가로축은 계급, 세로축은 도수를 나타냄

* bins: 히스토그램의 가로축 구간의 개수

```
1  x = np.random.randn(1000)
2  plt.title('Histogram')
3  plt.hist(x, bins = 10)
4  plt.show()
```



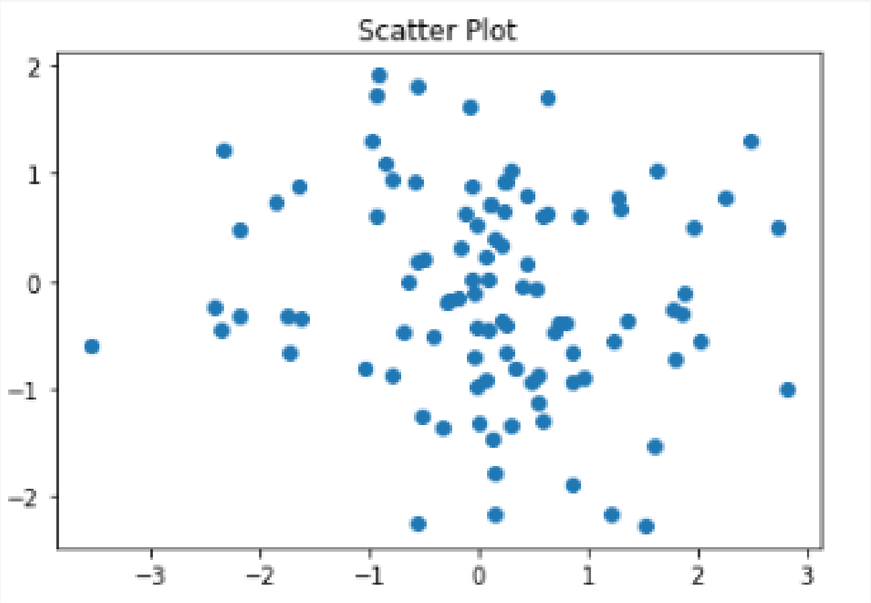
plt.hist(x, bins = 10)

scatterplot

산점도 만드는 함수

두 개의 실수 데이터 집합의 상관관계 살펴볼 때 사용

```
X = np.random.normal(0,1, 100)
Y = np.random.normal(0,1,100)
plt.title('Scatter Plot')
plt.scatter(X, Y)
plt.show()
```



plt.scatter(X, Y)

04

머신러닝 첫 걸음

생선 분류 문제

목표

도미와 빙어 **두 종류**의 생선을 구분하는 머신러닝 프로그램 설계

사용 데이터

도미와 빙어 데이터

(참고: 캐글 공개 데이터셋: https://www.kaggle.com/aungpyaeap/fish-market)

* 캐글(kaggle.com) : 2010년에 전 세계에서 가장 큰 머신러닝 경연 대회 사이트

1. 도미, 빙어 데이터 준비하기

```
bream_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0, 30.0, 30.7, 31.0, 31.0,
                31.5, 32.0, 32.0, 32.0, 33.0, 33.0, 33.5, 33.5, 34.0, 34.0, 34.5, 35.0,
                35.0, 35.0, 35.0, 36.0, 36.0, 37.0, 38.5, 38.5, 39.5, 41.0, 41.0]
bream_weight = [242.0, 290.0, 340.0, 363.0, 430.0, 450.0, 500.0, 390.0, 450.0, 500.0, 475.0, 500.0,
                500.0, 340.0, 600.0, 600.0, 700.0, 700.0, 610.0, 650.0, 575.0, 685.0, 620.0, 680.0,
                700.0, 725.0, 720.0, 714.0, 850.0, 1000.0, 920.0, 955.0, 925.0, 975.0, 950.0]
```

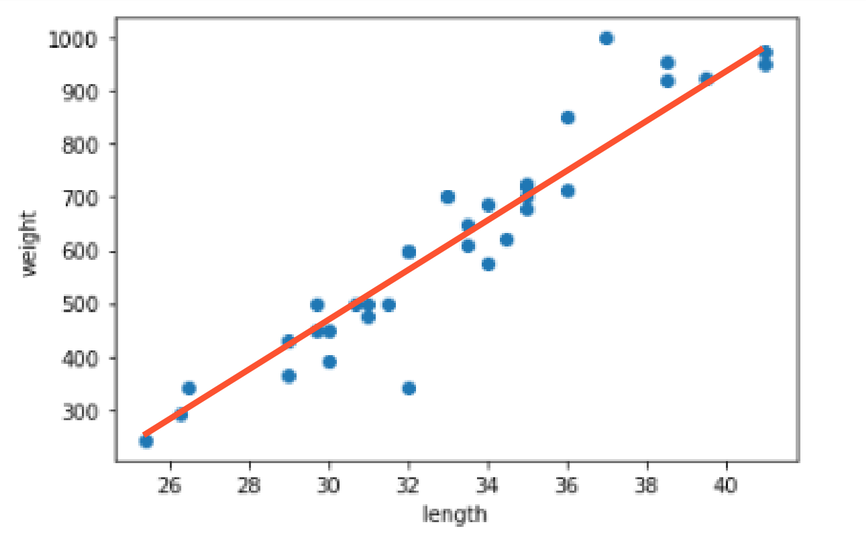
<35마리의 도미 데이터>

```
smelt_length = [9.8, 10.5, 10.6, 11.0, 11.2, 11.3, 11.8, 11.8, 12.0, 12.2, 12.4, 13.0, 14.3, 15.0]
smelt_weight = [6.7, 7.5, 7.0, 9.7, 9.8, 8.7, 10.0, 9.9, 9.8, 12.2, 13.4, 12.2, 19.7, 19.9]
```

<14마리의 빙어 데이터>

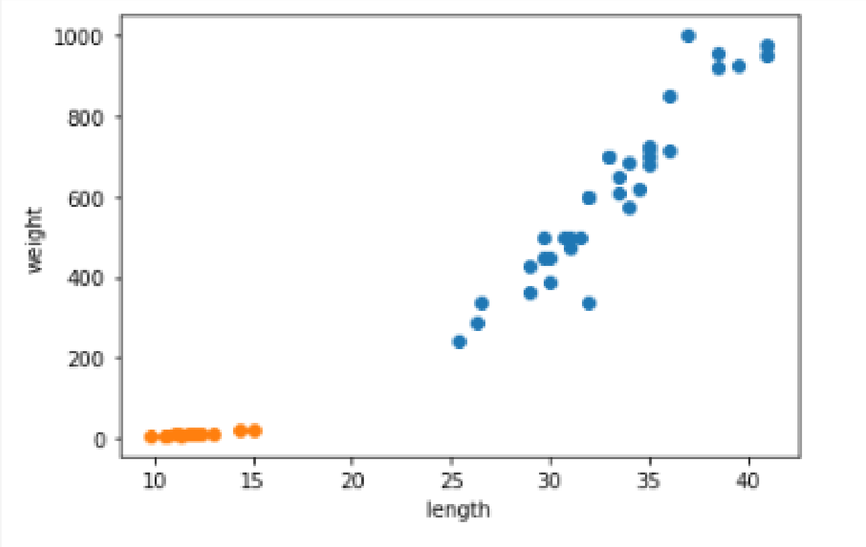
2. 산점도를 통한 관계 분석

```python
import matplotlib.pyplot as plt

plt.scatter(bream_length, bream_weight)
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



선형적 (길이가 길수록 무게가 많이 나간다)

```python
import matplotlib.pyplot as plt

plt.scatter(bream_length, bream_weight)
plt.scatter(smelt_length, smelt_weight)
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



무게가 길이의 영향을 덜 받는다.

3. 2차원 리스트로 만들기

```
length = bream_length + smelt_length
weight = bream_weight + smelt_weight
```

<도미와 빙어 두 리스트 합치기>

```
fish_data = [[l, w] for l,w in zip(length, weight)]
print(fish_data)
```

```
[[25.4, 242.0], [26.3, 290.0], [26.5, 340.0], [29.0, 363.0], [29.0, 430.0], [29.7, 450.0], [29.7, 500.0], [30.0, 390.0], [30.0, 450.
0], [30.7, 500.0], [31.0, 475.0], [31.0, 500.0], [31.5, 500.0], [32.0, 340.0], [32.0, 600.0], [32.0, 600.0], [33.0, 700.0], [33.0, 70
0.0], [33.5, 610.0], [33.5, 650.0], [34.0, 575.0], [34.0, 685.0], [34.5, 620.0], [35.0, 680.0], [35.0, 700.0], [35.0, 725.0], [35.0, 7
20.0], [36.0, 714.0], [36.0, 850.0], [37.0, 1000.0], [38.5, 920.0], [38.5, 955.0], [39.5, 925.0], [41.0, 975.0], [41.0, 950.0], [9.8,
6.7], [10.5, 7.5], [10.6, 7.0], [11.0, 9.7], [11.2, 9.8], [11.3, 8.7], [11.8, 10.0], [11.8, 9.9], [12.0, 9.8], [12.2, 12.2], [12.4, 1
3.4], [13.0, 12.2], [14.3, 19.7], [15.0, 19.9]]
```

<2차원 리스트 만들기>

4. 정답 데이터 준비하기

```
1 fish_target = [1] * 35 + [0] * 14
2 print(fish_target)
```

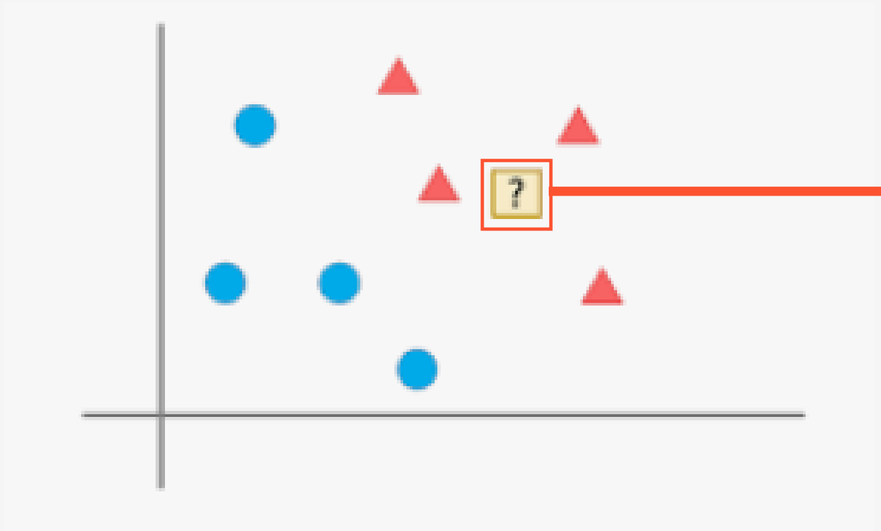
```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

도미를 숫자 1로, 빙어를 숫자 0으로 표현

5. 사용 머신러닝 모델

k-최근접 이웃 알고리즘

주변에서 가장 가까운 5개의 데이터를 보고, 다수결의 원칙에 따라
데이터를 예측하는 알고리즘



빨간색 세모에 가깝기 때문에
빨간색 세모로 예측

6. 머신러닝 모델 적용하기

```
1 from sklearn.neighbors import KNeighborsClassifier
2
3 kn = KNeighborsClassifier()
4 kn.fit(fish_data, fish_target)
5 kn.score(fish_data, fish_target)
```

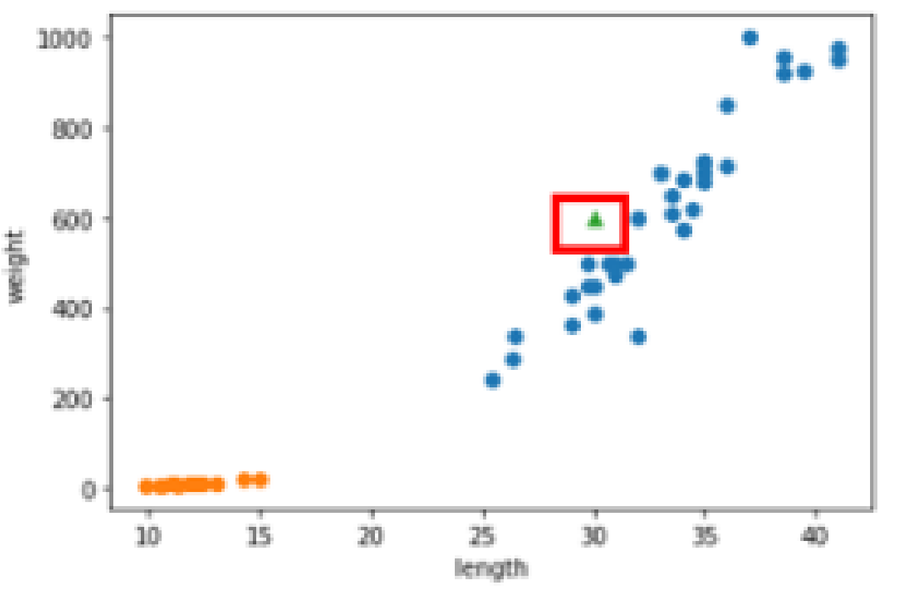
가장 가까운 5개의 데이터를 보고,
다수결의 원칙에 따라 데이터를 예측

```
1 from sklearn.neighbors import KNeighborsClassifier
2
3 kn = KNeighborsClassifier(n_neighbors = 49)
4 kn.fit(fish_data, fish_target)
5 kn.score(fish_data, fish_target)
```

가장 가까운 49개의 데이터를 보고,
다수결의 원칙에 따라 데이터를 예측

7. 새로운 데이터값 분류하기

```
1  import matplotlib.pyplot as plt
2
3  plt.scatter(bream_length, bream_weight)
4  plt.scatter(smelt_length, smelt_weight)
5  plt.xlabel('length')
6  plt.scatter(30, 600, marker = '^')
7  plt.ylabel('weight')
8  plt.show()
```




```
1  kn.predict([[30, 600]])
```

출력 결과)

```
array([1])
```

#도미로 분류

감사합니다 :)