

[그룹빅데이터스쿨]

데이터 분석 in 모빌리티 산업

※ 부제 : HDAT-DA 인증대비반

● 2024.09.25 ~ 09.27

01 파이썬 기본 문법 - 데이터 편

학습목표 : 파이썬 기본 문법에 대해서 익힌다.

01 큰 그림

- 파이썬 기본 문법 공부
 - 크게 2가지!
 - 명령을 내릴 대상(데이터, 객체)에 관해서
 - 대상에 어떻게 명령을 내릴지



02 파이썬의 데이터

- 명령을 내릴 대상
- 객체(object)라 함
- 처음 공부할 때는 익숙하지 않은 용어는 배제하고 직관적으로 '데이터' 라고 하자!

02 파이썬의 데이터

- 총 4가지를 공부 함
 - 기본 데이터 타입들
 - 변수
 - 각 데이터 타입이 가지고 있는 기능이나 속성
 - 자료구조

03 기본 데이터 타입 - 숫자 데이터 타입

- 정수, 실수
 - (복소수 등 더 다양한 데이터 타입이 있지만 처음부터 굳이 그런것들을 모두 알 필요가 없다)
- 쉽다! => 그냥 엑셀이나 계산기 사용하 듯이 사용하면 됨
- 데이터 생성 : 키보드의 숫자키를 입력
- 연산 : 사칙연산 (우리가 알고 있는)

3

3

3.0

3.0

3 + 4

7

3.0 + 4.0

7.0

03 기본 데이터 타입 - 문자 데이터 타입

- 문자열이라고 부름
- 처음 공부할 때 실수가 가장 많이 나오는 곳이니 정확히 학습!
- 생성 : ' ', " " 안에 들어가면 무조건 문자열

'python'

'python'

'파이썬'

'파이썬'

':)'

':)'

03 기본 데이터 타입 - 문자 데이터 타입

- 문자열
- 연산 : 더하기 하나만 정의되어 있음
 - 문자열 + 문자열 => 합쳐진 새로운 문자열 생성

```
'파이썬' + ' ' + '안녕!'
```

```
'파이썬 안녕!'
```


03 기본 데이터 타입 - 문자 데이터 타입

• 문자열

• 숫자와 연산?! => 기본적으로 다른 데이터 타입과 연산 X

• 예외 : 정수 곱하기 => 정수만큼 문자열 반복

```
'3' + 4
```

```
'파이썬' * 3
```

```
'파이썬파이썬파이썬'
```

TypeError

Traceback (most recent call last)

Cell In[10], line 1

----> 1 '3' + 4

TypeError: can only concatenate str (not "int") to str

```
'파이썬' * 3
```

```
'파이썬파이썬파이썬'
```

04 기본 데이터 타입 - bool 데이터 타입

- True, False
- 참과 거짓을 나타냄
- 대소문자 주의
- 파이썬에서 True는 정수 1, False는 정수 0과 같음

```
True
```

```
True
```

```
False
```

```
False
```

```
3 > 2
```

```
True
```

```
3 < 2
```

```
False
```

05 기본 데이터 타입 - None 타입

- 데이터가 없음을 명시적으로 나타냄

None

06 변수

- 지금까지 배운 기본 데이터만으로는 생성한 데이터를 재사용할 수가 없음

3 + 4

7

???? * 2

06 변수

- 그래서 변수라는 개념이 필요!
- **변수(variable)**은 데이터를 재사용하기 위해서 데이터에 이름을 붙인 것
- 이름이 있어야 우리는 그 데이터를 호출(call)할 수 있고, 그 데이터를 호출할 수 있어야 명령을 내릴 수 있음

06 변수

- 항상 데이터를 생성 했으면 변수부터 만들어 주자!
- 데이터는 생성하는 데 변수를 안 만드는 실수를 정말 많이 함

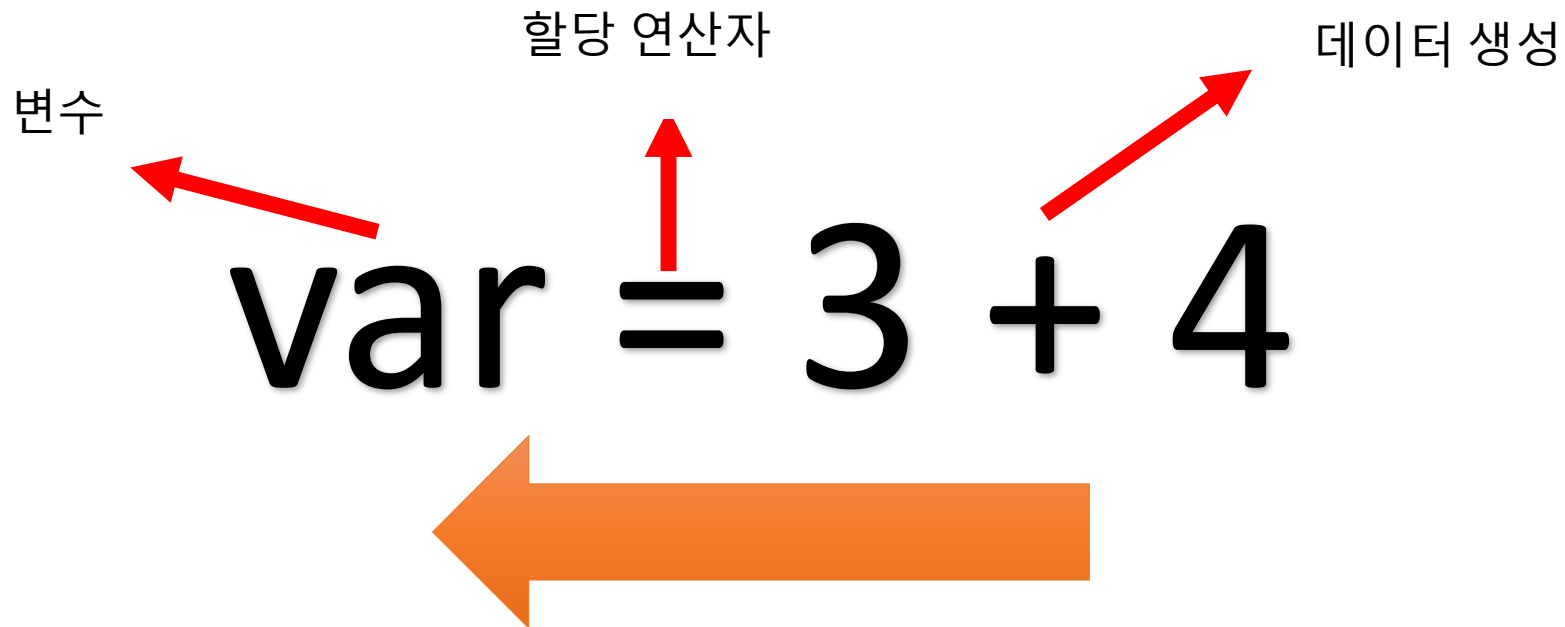
```
3 + 4
```

```
7
```

```
var = 3 + 4
```

06 변수

- 데이터를 변수에 할당하기



06 변수

- 변수명은 길어도 좋은 길고 어떤 데이터가 할당이 되어 있는지 명확히 알 수 있게 만들자
- 변수명 잘 만드는 것이 능력!
- 여러 단어를 조합할 시 띄어쓰기는 쓸 수 없고 파이썬은 보통 _로 연결
ex) data + number => data_number

07 데이터의 기능 및 속성

- 파이썬의 데이터는 눈에 보이는 값만 있는 것이 아니라 **데이터 타입에 따라 고유한 기능 및 속성**이 있다.
- 문자열은 문자열 만의 기능이나 속성, 숫자는 숫자 데이터만의 기능이나 속성...
- A라는 데이터 타입은 A만의 기능이나 속성값이 있음

07 데이터의 기능 및 속성

- .(dot) 연산자를 이용해서 꺼내 쓴다
- 주피터 노트북에서 'tab' 버튼을 누르면 자동으로 어떤 속성이나 기능을 꺼내서 사용할 수 있는지 알려 줌

```
var = 3
```

var.

f	as_integer_ratio	function
f	bit_length	function
f	conjugate	function
p	denominator	property
f	from_bytes	function
p	imag	property
p	numerator	property
p	real	property
f	to_bytes	function

```
var = '3'
```

var.

f	capitalize	function
f	casefold	function
f	center	function
f	count	function
f	encode	function
f	endswith	function
f	expandtabs	function
f	find	function
f	format	function
f	format_map	function

07 데이터의 기능 및 속성

- 내용물은 같은 3 이지만 어떤 데이터 타입으로 사용하느냐에 따라 사용할 수 있는 기능이나 속성이 다르다!
- 어떤 기능이나 속성이 있는지 공부하는 것이 아니라 그때, 그때 검색해서 사용!

08 자료구조

- 데이터는 한번에 하나씩 생성되거나 사용되는 것이 아니라 한번에 여러 개가 생성되거나 사용된다.
- 예 : A회사 주식의 1주일 종가를 가지고 데이터 분석을 한다고 가정해 보자.
- 1주일이면 영업일이 5일이고 종가 데이터는 5개가 생성된다 => 변수 5개 필요
- 일년으로 확장하면?! => 대략 영업일이 250일, 데이터가 250개 => 변수 250개 필요?!

08 자료구조

- 한번에 여러 개의 데이터를 효율적으로 묶어서 관리하는 도구가 필요
- 이것이 자료구조!
- 파이썬을 더 많이 사용하면 정말 다양한 자료구조를 만나게 됨
- 일단 파이썬의 기본 자료구조만 살펴봐도 충분하다.

08 자료구조

- 파이썬 기본 자료구조
- **리스트, 딕셔너리**, 튜플, 셋
- 모두 중요하지만 일단 처음 공부할 때는 리스트, 딕셔너리에 집중
- 이 둘은 정말 자주 사용되므로 정확히 공부하자!

09 리스트(List)

- 파이썬의 대표적인 **순서**있는 자료구조
- 생성 : []
- 데이터 꺼내기 : 순서(번호, 인덱스)
- 프로그래밍에서 순서는 일반적으로 0부터 시작
- 음수도 가능

```
ex_list = ['떡볶이', '라면', '돈까스', '짜장면']
```

```
ex_list[2]
```

'돈까스'

```
ex_list[0]
```

'떡볶이'

09 리스트(List)

- 범위로 가져오기(슬라이싱) : 순서있는 자료구조이기 때문에 범위로도 가져올 수 있음
- []안에 ':' (콜론)을 사용하면 슬라이싱을 사용한다는 의미
- ':'을 기준으로 앞에는 시작, 뒤에는 끝나는 번호
- 숫자 범위를 사용할 때는 앞에는 포함, 뒤에는 불포함
- 앞을 비우면 처음부터, 뒤를 비우면 마지막까지

```
ex_list[ 0 : 3]
```

```
['떡볶이', '라면', '돈까스']
```

```
ex_list[ 2 : 4]
```

```
['돈까스', '짜장면']
```

```
ex_list[ : 2 ]
```

```
['떡볶이', '라면']
```


09 리스트(List)

- 리스트의 기능들 => 여러가지 있음 ☺
- 앞에서도 말했듯이 공부할 필요 없음!
- 그때 그때 검색하거나 ChatGPT에게 물어봐서 사용
- append 하나만 써보고 넘어가자
 - append는 굉장히 특이한 기능인데 무엇이 특이한지 수업시간에 배운 것을 다시 생각해 보자

ex_list.

f	append	function
f	clear	function
f	copy	function
f	count	function
f	extend	function
f	index	function

```
ex_list.append('새로운 메뉴')
```

```
ex_list
```

```
['떡볶이', '라면', '돈까스', '짜장면', '새로운 메뉴']
```

10 딕셔너리(Dictionary)

- 마찬가지로 여러 데이터를 묶어 놓은 개념인데 데이터끼리 '순서'라는 개념이 없음
- 대신 데이터마다 '**키(key)**'라는 개념을 통하여 서로 구분하고 접근 함
- 생성 : { }
- 데이터 꺼내기 : 키값

```
ex_dict = {'math':90, 'eng':100, 'kor':92}
```

```
ex_dict['math']
```

90

10 딕셔너리(Dictionary)

- 슬라이싱 => 순서가 없으므로 딕셔너리에서는 슬라이싱 x
- 데이터 추가하기
`dict[추가하고 싶은 키] = 추가하고 싶은 데이터`

```
ex_dict['music'] = 100
```

```
ex_dict
```

```
{'math': 90, 'eng': 100, 'kor': 92, 'music': 100}
```

02 파이썬 기본 문법 - 명령 내리기 편

학습목표 : 파이썬 기본 문법에 대해서 익힌다.

11 파이썬 데이터에 명령 내리기

- 대상(데이터)에 명령을 내려서 일을 시켜 보자!
- 크게 3가지
 - 연산
 - 함수
 - 제어문(반복문, 조건문)

파이썬 데이터에 명령 내리기

- 대상(데이터)에 명령을 내려서 일을 시켜 보자!
- 크게 3가지
 - ~~연산~~ - 데이터 타입마다 다르므로 그때, 그때 공부
 - 함수
 - 제어문(반복문, 조건문)

12 함수(사용자 관점)

- 기능, 동작
- 파이썬 함수는 이름만 쓰면 동작하지 않음
- 이런 함수가 존재한다고 값으로 나옴(데이터, 객체)

```
print
```

```
<function print(*args, sep=' ', end='\n', file=None, flush=False)>
```

12 함수(사용자 관점)

- 함수() <= 동작버튼
- ()가 있어야 동작함

```
print
```

```
<function print(*args, sep=' ', end='\n', file=None, flush=False)>
```

```
print( 3, 4, 5 )
```

```
3 4 5
```


12 함수(사용자 관점)

- ()
- 동작 버튼
- 필요에 따라 **인풋(매개변수라 함)**들의 입구가 됨
- 인풋을 정해진 규칙에 맞게 잘 넣어야 동작함

```
sum(1, 2, 3)
```

```
-----  
TypeError  
Cell In[39], line 1  
----> 1 sum(1, 2, 3)  
  
TypeError: sum() takes a
```

```
sum([1, 2, 3])
```

12 함수(사용자 관점)

- 함수의 '시그니처'를 잘 확인하자!
- 공부하거나 외울 필요 없음
- 그때, 그때 검색해서 사용!
(자주 사용하는 것은 자연스레 외워짐)

sum?

Signature: sum(iterable, /, start=0)

12 함수(사용자 관점)

- 함수의 인풋을 정확히 입력하였다면
- **결과값(정확히 리턴값)**을 확인!
- 이 함수를 사용했을 때 결과 값이 있는지 없는지 명확하게 확인하자!

```
print( 6 )
```

6

```
sum([1, 2, 3])
```


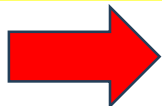
6

- DRY : Do not Repeat Yourself
- 반복적인 작업을 컴퓨터에게 시켜야지 우리가 하고 있으면 안된다!
- 똑같은 '패턴' 의 코드가 1번 이상 반복되면 반복문으로 바꿔주자!
- 파이썬에는 for문과 while문 2가지 반복문이 있다.

- DRY : Do not Repeat Yourself
- 반복적인 작업을 컴퓨터에게 시켜야지 우리가 하고 있으면 안된다!
- 똑같은 '패턴'의 코드가 1번 이상 반복되면 반복문으로 바꿔주자!
- 파이썬에는 for문과 while문 2가지 반복문이 있다.

14 for문

- 순서있는 자료구조에서 데이터를 순차적으로 가져와 같은 작업을 반복

`for` 변수 `in` 순서있는 자료구조 
 반복할 코드

14 for문

- 순서있는 자료구조에서 데이터를 순차적으로 가져와 같은 작업을 반복

```
var = '철수'  
print(var + '입니다')
```

```
var = '수현'  
print(var + '입니다')
```

```
var = '민재'  
print(var + '입니다')
```

철수입니다
수현입니다
민재입니다

자동으로 데이터를 순차적으로
꺼내서 변수에 할당

```
for var in ['철수', '수현', '민재']:  
    print(var + '입니다')
```

변수를 가지고 똑같은
코드를 반복

철수입니다
수현입니다
민재입니다

14 for문

- 파이썬 for문은 리스트와 궁합이 좋다!
- 리스트가 주어지면 이 리스트를 가지고 for문을 사용해 보자

```
num_list = [10, 11, 12, 13, 7, 8, 9]

for num in num_list:
    print(num)
```

```
10
11
12
13
7
8
9
```


14 for문

- 자주 사용하는 for문 패턴

```
num_list = [10, 11, 12, 13, 7, 8, 9]

result = 0
for num in num_list:
    result = result + num
```

- for문을 사용하기 전에 for문 돌면서 계산될 결과를 저장하기 위해 result라는 변수를 초기화
- result = result + num 연산. 여기서 오른쪽에서 왼쪽으로 해석해야 함 (더하고 다시 덮어쓰고!)
- 계속 반복되면 result 에 num_list의 값들이 모두 더해지게 됨
- 헛갈리는 사람들은 result 가 어떻게 변하는지 순차적으로 종이에 써보자!

14 for문

- range(시작, 끝) 함수
- 정수로 이루어진 리스트를 만들어 줌(정확히 리스트는 아님)
- for문 사용시 정수로 이루어진 리스트가 필요할 경우가 많은데 range를 이용하면 쉽게 만들어서 사용 가능

15 while문

- 조건이 '참' 일 동안만 반복

while 조건:
반복할 코드

True
False



15 while문

- 조건이 '참' 일 동안만 반복

조건 초기화

True
False

while 조건:

반복할 코드

조건 업데이트

A diagram illustrating the components of a while loop. The text '조건 초기화' (Initialize condition) is in a red box. To its right, 'True' and 'False' are listed in red, with a red arrow pointing from 'True' to the '조건' (condition) part of the 'while' statement. The 'while' statement is shown as 'while 조건:', with '조건' in a red box. Below it is '반복할 코드' (Code to be repeated). At the bottom, '조건 업데이트' (Update condition) is in a red box.

15 while문

- while 문 vs for 문
- 둘 다 같은 작업을 할 수 있다.
- 반복의 범위가 정해진 경우 => for문
- 반복의 범위가 정해지지 않고 바뀔 경우 => while문

```
var = 1
while var < 10:
    print(var)
    var = var + 1
```

1
2
3
4
5
6
7
8
9

16 조건문

- 조건에 따라 실행되는 코드를 나누어 줌
- 무조건 **if로 시작**하기 때문에 if문이라고도 함

if 조건:

조건이 참일 시 실행되는 코드

16 조건문

- if 문 하나만 썼을 시 경우의 수는 2개
- 참이라 실행이 되거나
- 참이 아니라 아무것도 실행이 안되고 다음 코드로 넘어 감

```
data = 5
if data < 10:
    print('data < 10')
```

data < 10

```
data = 15
if data < 10:
    print('data < 10')
```

16 조건문

- if 문 하나만 썼을 시 경우의 수는 2개
- 참이라 실행이 되거나
- 참이 아니라 아무것도 실행이 안되고 다음 코드로 넘어 감

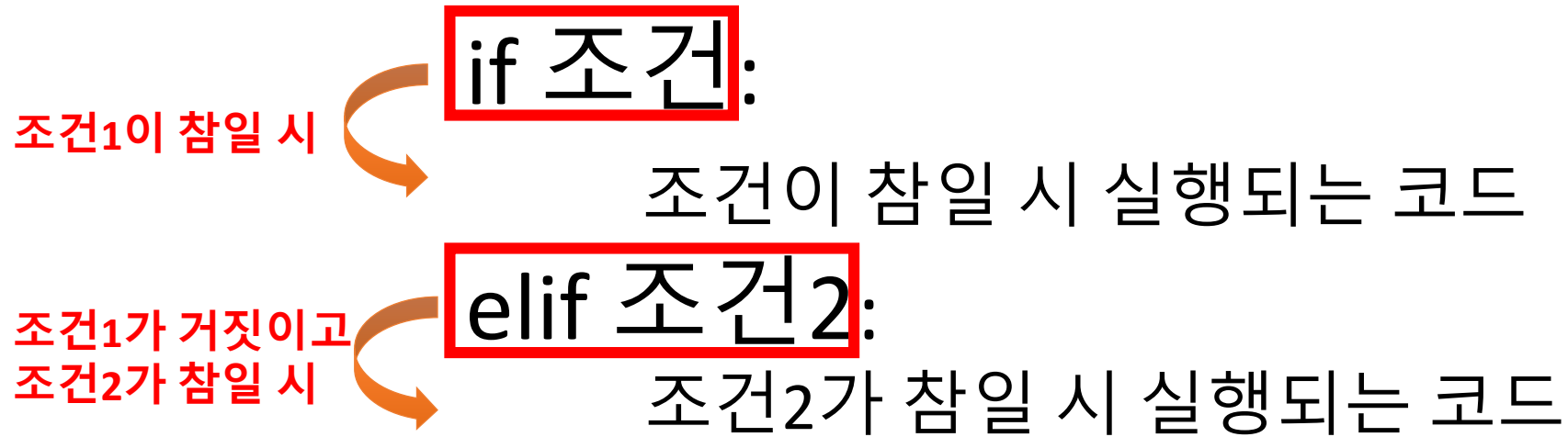
```
data = 5
if data < 10:
    print('data < 10')
```

data < 10

```
data = 15
if data < 10:
    print('data < 10')
```

결과 없음

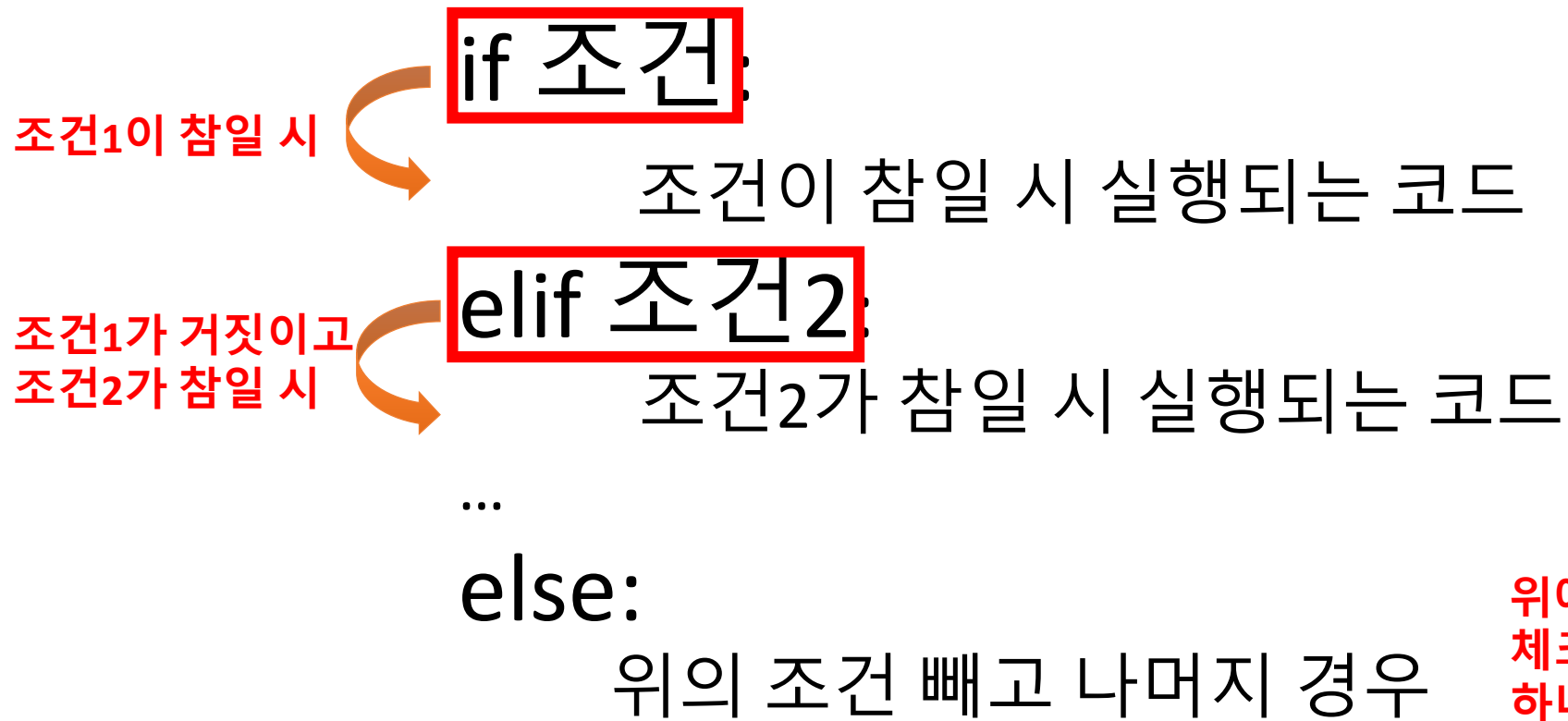
- if 문에서 참이 아니면 이어서 다른 조건을 체크 가능



위에서 아래로 순차적으로
체크한다.
하나의 결과가 나오면 해당
조건문은 종료 됨.

16 조건문

- else => 맨 마지막에서 여집합일 경우 실행



위에서 아래로 순차적으로
체크한다.
하나의 결과가 나오면 해당
조건문은 종료 됨.

03 파이썬 기본 문법 - 함수, 모듈

학습목표 : 파이썬 기본 문법에 대해서 익힌다.

17 함수

- 함수를 사용하는 이유
- 코드를 재사용하기 위해서 코드에 이름을 붙여 줌!
- 함수를 단순히 복사 붙여 넣기하여 사용시 꼭 발생하는 문제점
 - 변수명 중복
 - 변수명 통일 안됨

17 함수

- 함수 정의문 문법

```
def 함수이름(매개변수들):  
    재사용하고 싶은 코드  
    return 데이터
```

아직 값이 확정되지 않은 변수들
함수 호출 시 입력해 주어야 함

함수 사용 후 가지고 와서 사용할
값

17 함수

- 많이들 하는 착각!
- 함수를 정의하는 것은 설계도를 만드는 것 뿐이지 코드를 동작 시킨 것은 아님!
- 정의한 함수를 호출해야 그때서야 코드가 동작 함!

함수 정의!
코드가 동작한 것이 아님

```
def get_average( number_list ):  
    result = 0  
    for number in number_list:  
        result = result + number  
    avg = result / len(number_list)  
    return avg
```

```
get_average([3, 2, 1, 4])
```

2.5

함수 호출!
number_list=[3, 2, 1, 4]가 되어
함수 내부의 코드가 동작함

18 모듈

- 함수를 모아 놓은 **도구 상자**

(사실 함수 말고 다른 것도 있지만 처음 공부할 때는 함수를 모아 놓았다고 생각해도 무방하다.)

- import 모듈

- 특정 모듈을 통째로 가져옴
- 매번 코드에 '모듈.함수'와 같이 그때, 그때 꺼내서 사용

- from 모듈 import 함수

- 특정 모듈에서 특정 함수 하나를 꼭 짚어서 가져옴
- 사용 시 함수 이름만 호출하면 됨

18 모듈

```
import random
```

```
random.randrange(1, 11)
```

10

random 모듈 통째로 가져옴
randrange 함수 사용시 매번
random.randrange라고 꺼내야 함

```
from random import randrange
```

```
randrange(1, 11)
```

7

random 모듈에서 randrange 함수
하나만 꺼내옴.
randrange 함수를 가져 온 개념이니
함수만 단독 사용가능

04 파이썬과 ChatGPT

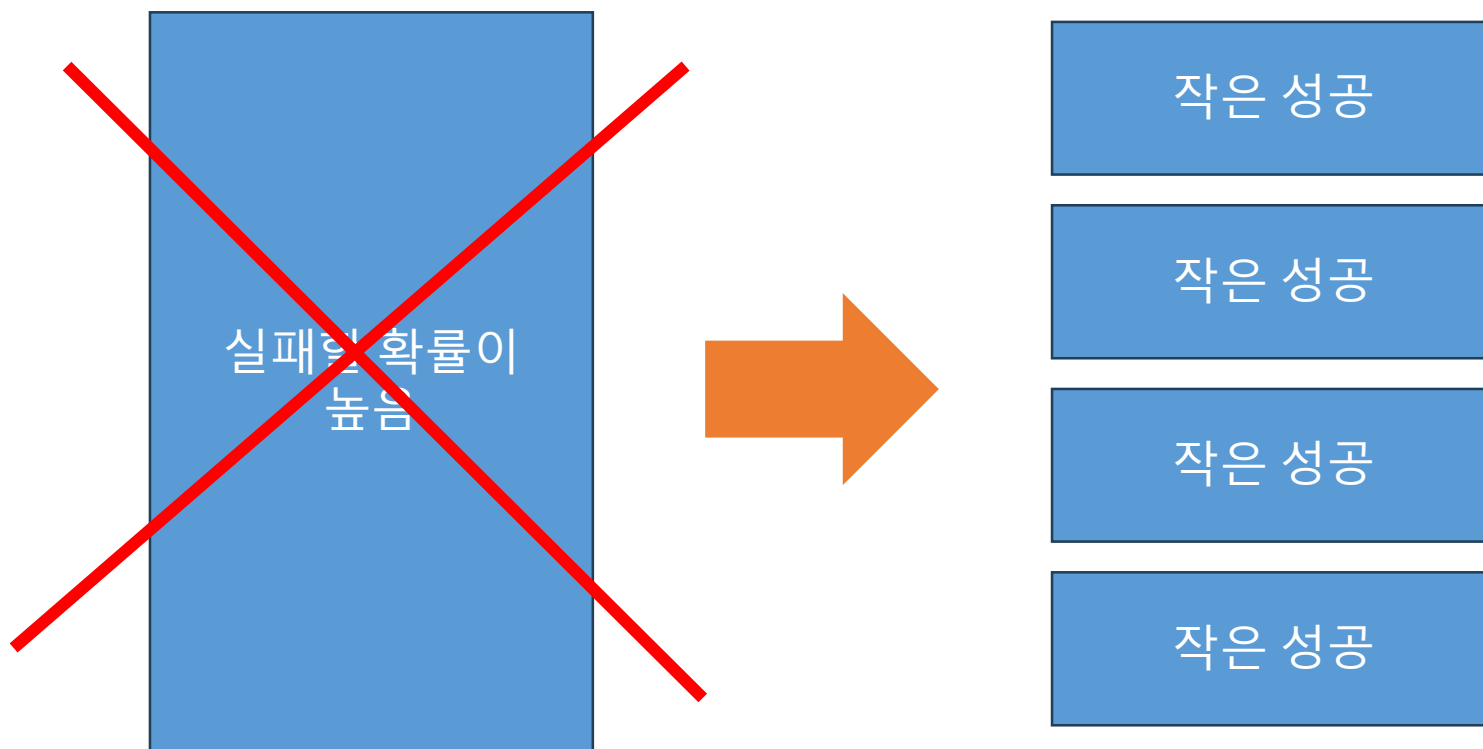
학습목표 : ChatGPT를 이용하여 파이썬을 활용한다.

19 ChatGPT의 일반적인 활용법

- 구체적으로
 - 문제를 명확하게
 - 작게 세분화하여 부분 => 전체 정복
- 문맥 제공
 - 룰(role) 지정
 - 관점이나 배경지식 제공
- 반복적으로
 - 솔루션을 점차 발전 시켜 간다는 느낌으로

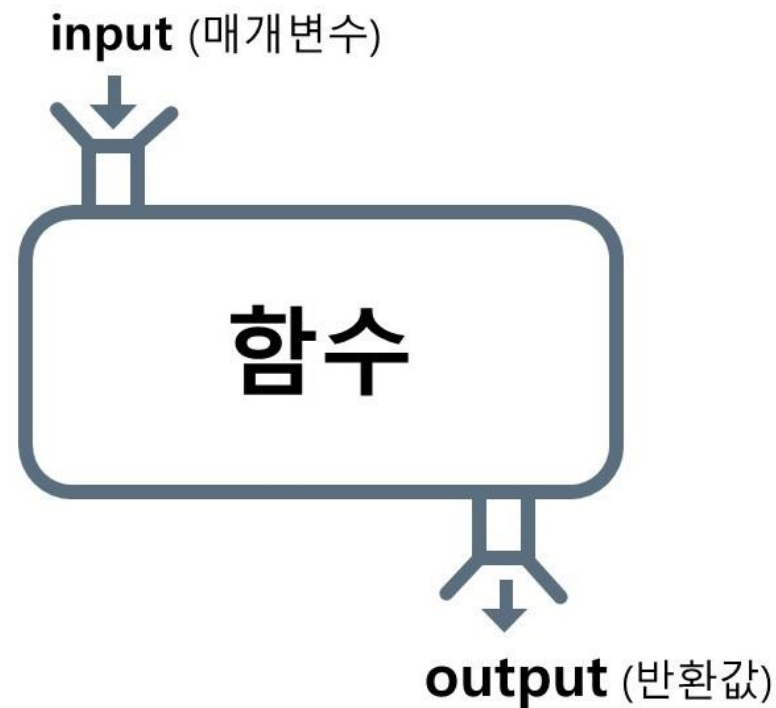
20 ChatGPT with Python

- 구체적으로
- 문제를 작게 세분화 하자! (가장 중요)



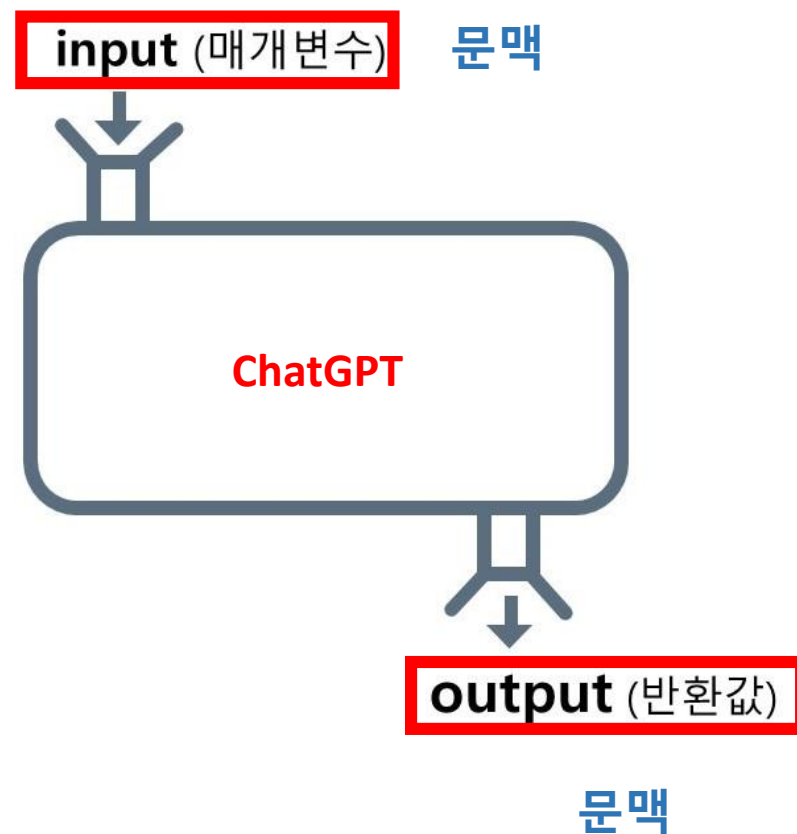
20 ChatGPT with Python

- 문맥 제공
- 앞 뒤 코드(정보)를 제공
- 가장 좋은 방법은 함수의 형태로



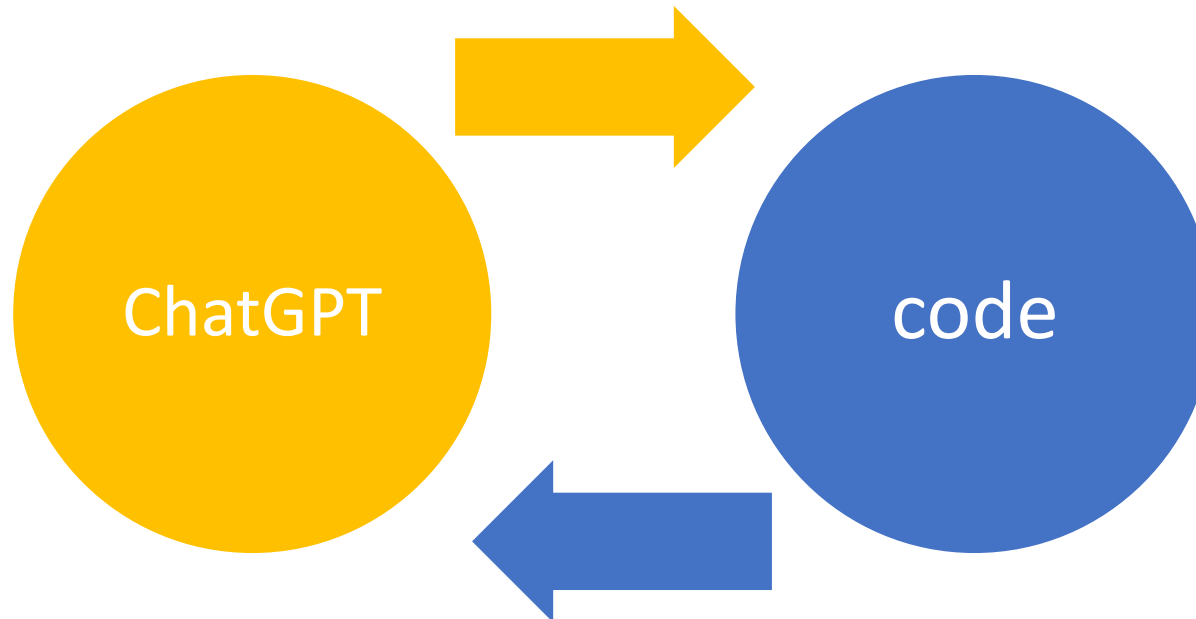
20 ChatGPT with Python

- 문맥 제공
- 앞 뒤 코드(정보)를 제공
- 가장 좋은 방법은 함수의 형태로



20 ChatGPT with Python

- 반복적으로
- ChatGPT 생성 코드를 실행해 보고 계속해서 조정해 나간다
(이것은 신탁이 아니다! 믿을 것이 아니라 계속 의심해 봐야 함. 대졸 신입 인턴과 일하는 느낌)



20 ChatGPT with Python

- 반복적으로
- 코드를 그대로 올려서 에러나 잘못된 부분 체크
- 코드에 주석달기
- 복잡한 코드를 풀어서 작성

05 파이썬과 데이터분석 입문

학습목표 : 파이썬 데이터 분석의 기본을 익힌다.

21 Python 데이터 분석 3형제

- Pandas

- ✓데이터 분석에 적합한 새로운 자료구조 2개

- Numpy

- ✓선형대수 연산

- Matplotlib(+seaborn)


- ✓데이터 시각화

22 Pandas - Series

- pandas => 새로운 자료구조 2개 제공
(list, dict은 데이터 분석에는 적합하지 않음 <= 왜 그런지 고민해 보자)
- Series
- DataFrame

22 Pandas - Series

- 1차원 자료구조
- 구성요소 2개
 - index
 - values
- 순서 있음



index	values
A	90
B	92
C	91
D	89
E	88

dtype: int64

22 Pandas - Series

- 데이터 꺼내기
- 인덱스
- 순서(경고가 나올 수 있지만 현시점에서는 무시)
- 슬라이싱

```
math_sr['A']
```

90

```
math_sr[0]
```

90

```
math_sr[:3]
```

A 90

B 92

C 91

dtype: int64

22 Pandas - Series

- 시리즈의 기능들
- 무수히 많음 => 그때, 그때 검색해서 사용
- 예시 : 정렬하기 (sort_values)

```
math_sr.sort_values()
```

```
E      88  
D      89  
A      90  
C      91  
B      92  
dtype: int64
```

```
math_sr.sort_values(ascending=False)
```

```
B      92  
C      91  
A      90  
D      89  
E      88  
dtype: int64
```

22 Pandas - Series

- 필터링

- 조건에 맞는 데이터만 가져오기

- `series[조건]`

- `series[(조건1) & (조건2)]`

- `series[(조건1) | (조건2)]`

```
math_sr[ math_sr > 90 ]
```

```
B      92
```

```
C      91
```

```
dtype: int64
```

조건은 꼭 컬럼(열, 시리즈)를
기준으로!!!!

22 Pandas - Series

- 시리즈의 연산1

- 같은 인덱스의 데이터끼리 알아서 맞춰서 연산 됨

- 집합적 연산

- 굉장히 강력한 기능이다!

- 오른쪽 예제는 각 시리즈의 데이터가 숫자이기 때문에 사칙연산이 가능

math_sr

A	90
B	92
C	91
D	89
E	88

dtype: int64

eng_sr

A	88
B	89
C	90
D	92
E	89

dtype: int64

math_sr + eng_sr

A	178
B	181
C	181
D	181
E	177

dtype: int64

22 Pandas - Series

- 시리즈의 연산2
- 시리즈와 값 하나 연산
- 시리즈의 모든 값이 개별적으로 값 하나와 연산 됨

```
math_sr + 100
```

A 190

B 192

C 191

D 189

E 188

dtype: int64

23 Pandas - DataFrame

- 2차원 자료구조
- 구성요소 3개 : columns, indexs, values
- 순서 있음 (기본적으로 행방향)

grade_df

	math	kor	eng
A	90	90	88
B	92	91	89
C	91	89	90
D	89	90	92
E	88	91	89

indexs

values

23 Pandas - DataFrame

- 데이터 꺼내기
- 기본적으로 column 기준
- 컬럼 하나 => 열 한줄 = Series
- 컬럼 여러 개 => 2차원 = DataFrame

```
grade_df['math']
```

```
A    90  
B    92  
C    91  
D    89  
E    88
```

```
Name: math, dtype: int64
```

```
grade_df[['math', 'kor']]
```

	math	kor
A	90	90
B	92	91
C	91	89
D	89	90
E	88	91

23 Pandas - DataFrame

- 데이터 꺼내기
- 인덱스 기준으로 가져오기 => .loc[인덱스]

```
grade_df.loc['A']
```

```
math    90  
kor     90  
eng     88  
Name: A, dtype: int64
```

```
grade_df.loc[['A', 'C']]
```

	math	kor	eng
A	90	90	88
C	91	89	90

23 Pandas - DataFrame

- 데이터 꺼내기
- 순서를 기준으로 가져오기 => .iloc[행순서]

```
grade_df.iloc[0]
```

```
math    90  
kor     90  
eng     88  
Name: A, dtype: int64
```

```
grade_df.iloc[[0, 2]]
```

	math	kor	eng
A	90	90	88
C	91	89	90

23 Pandas - DataFrame

- 슬라이싱
- `.iloc[행시작:행마지막+1, 열시작:열마지막+1]`

첫행부터 2행
전까지

```
grade_df.iloc[:2, 1:]
```

2열부터 마지막
열까지

	kor	eng
A	90	88
B	91	89

23 Pandas - DataFrame

- DataFrame의 기능들
- 무수히 많음 => 그때, 그때 검색해서 사용
- 예시 : 정렬하기 (sort_values)
- 이제 어떤 컬럼을 기준으로 할 것인지 입력!!

```
grade_df.sort_values('kor', ascending=False)
```

	math	kor	eng
B	92	91	89
E	88	91	89
A	90	90	88
D	89	90	92
C	91	89	90

23 Pandas - DataFrame

- 필터링

- 조건에 맞는 데이터만 가져오기

```
grade_df[ (grade_df['kor']>=90) & (grade_df['eng']>=90) ]
```

	math	kor	eng
D	89	90	92

- dataframe[조건]

- dataframe[(조건1) & (조건2)]

- dataframe[(조건1) | (조건2)]

조건은 꼭 컬럼(열, 시리즈)를
기준으로!!!!

23 Pandas - DataFrame

• 컬럼(속성) 추가하기

- dataframe의 속성을 이용하여 새로운 속성을 계산하고 이를 추가하기

시리즈의 연산 => 시리즈

```
grade_df['total'] = grade_df['math'] + grade_df['eng'] + grade_df['kor']
```

total이란 컬럼 만들고 저장

```
grade_df['total']
```

컬럼 => 시리즈

컬럼 => 시리즈

컬럼 => 시리즈

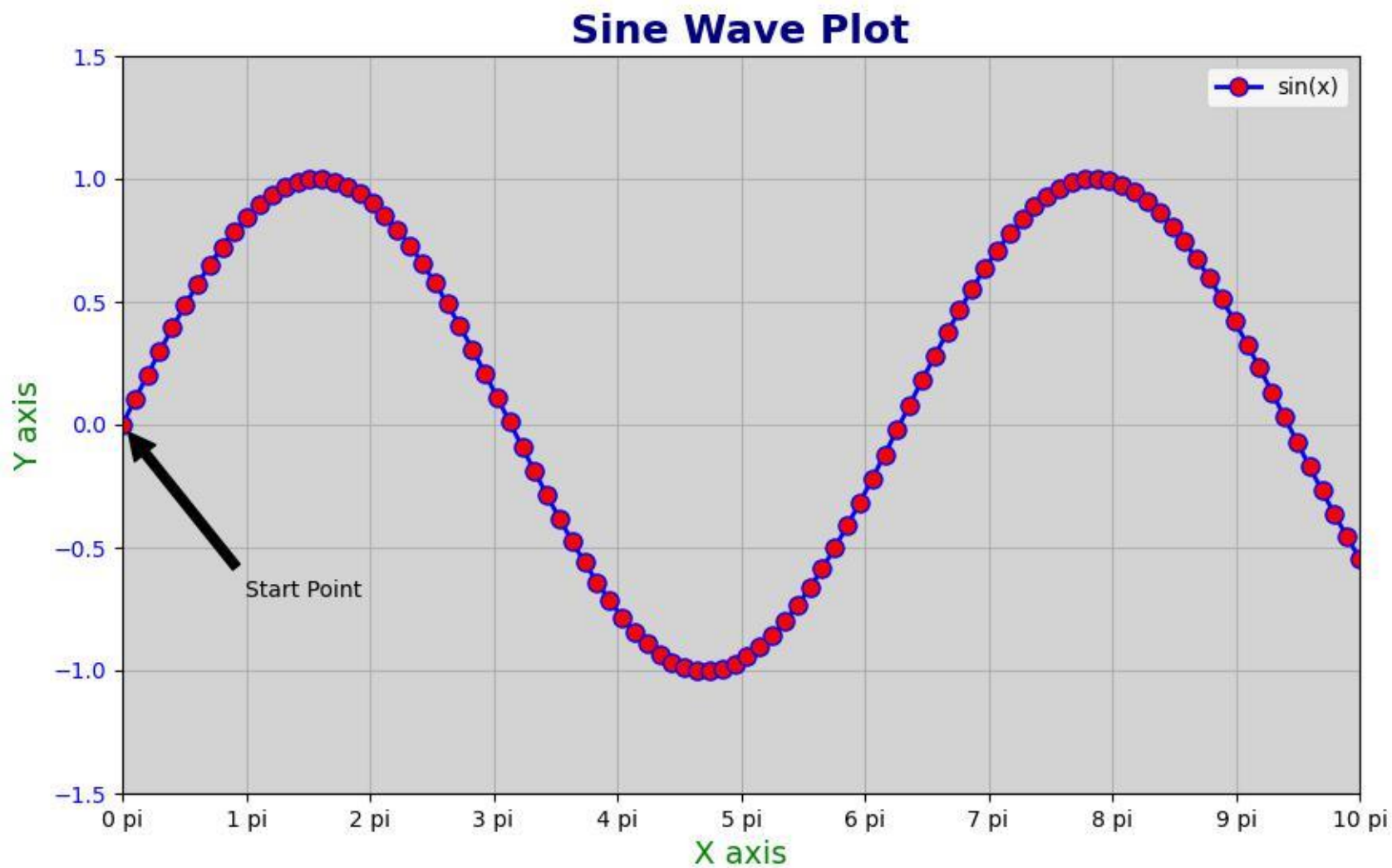
```
A    268
B    272
C    270
D    271
E    268
```

```
Name: total, dtype: int64
```


06 파이썬과 데이터 시각화

학습목표 : 파이썬을 이용하여 적절한 데이터 시각화를 수행한다.

24 데이터 시각화 기본



```

import matplotlib.pyplot as plt

# 그래프 생성
plt.figure(figsize=(10, 6)) # 그래프 크기 지정
plt.plot(x, y, label='sin(x)', color='blue', linestyle='--', linewidth=2, marker='o', markerfacecolor='red', markersize=8)

# 제목 및 라벨 추가
plt.title('Sine Wave Plot', fontsize=18, fontweight='bold', color='navy')
plt.xlabel('X axis', fontsize=14, color='green')
plt.ylabel('Y axis', fontsize=14, color='green')

# 범례 표시
plt.legend()

# 그리드 추가
plt.grid(True)

# 축의 범위 지정
plt.xlim(0, 10)
plt.ylim(-1.5, 1.5)

# 주석 추가
plt.annotate('Start Point', xy=(0, 0), xytext=(1, -0.7),
            arrowprops=dict(facecolor='black', shrink=0.05))

# 축의 배경 색상 변경
ax = plt.gca() # 현재 축 가져오기
ax.set_facecolor('lightgray') # 배경 색상 설정

# 틱 라벨 설정
plt.xticks(ticks=np.arange(0, 11, 1), labels=[f'{i} pi' for i in range(11)])
plt.yticks(color='blue')

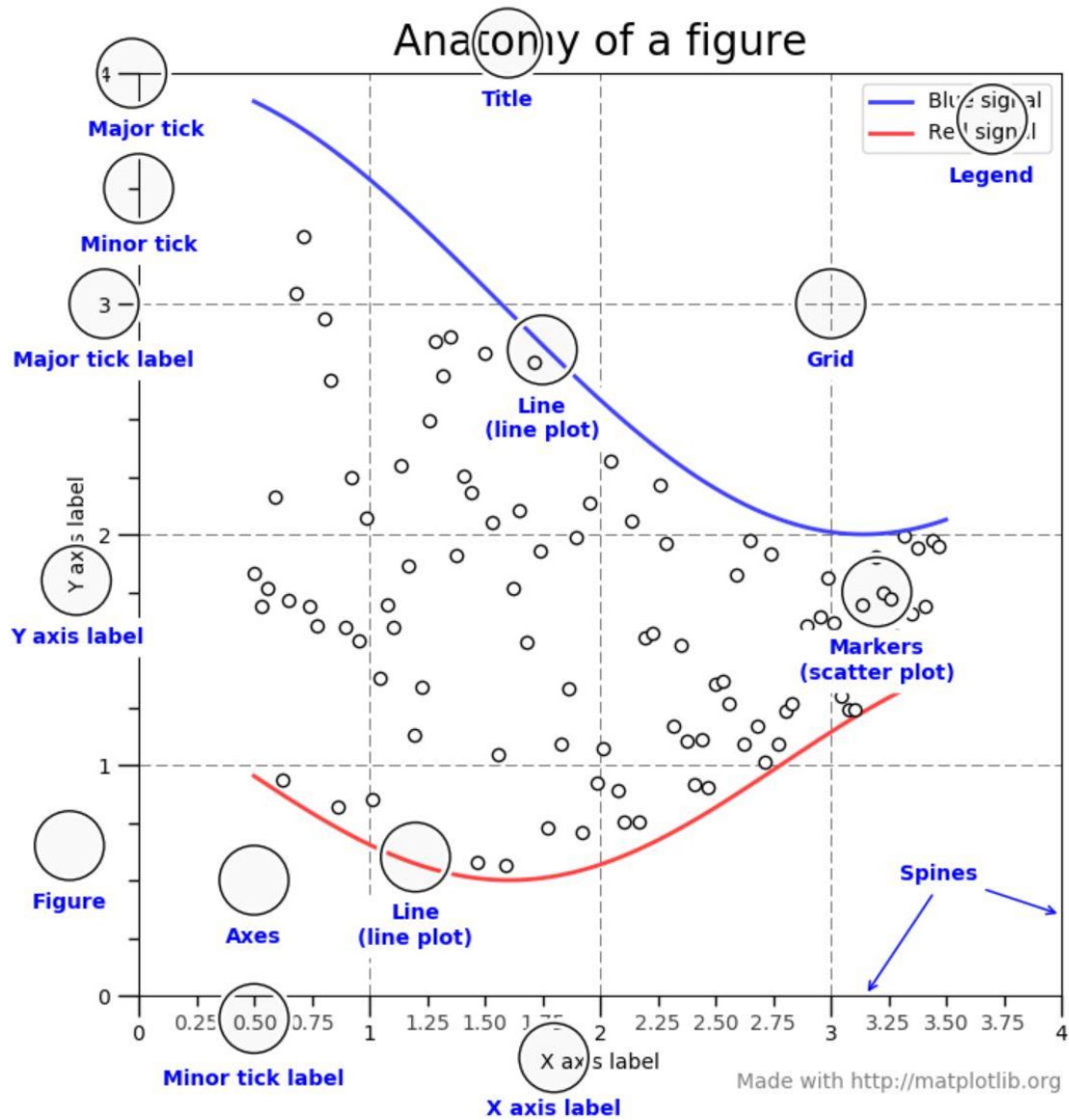
# 플롯 표시
plt.show()

```



24 데이터 시각화 기본

**일일이 다 코드로
넣어줘야 함!**



24 데이터 시각화 기본

- 기본 폼과 몇 가지 자주 사용하는 코드만 공부
- 더욱 이빠지게 하는 옵션들은 chatGPT를 이용!

```
plt.figure()
```

시작, 그림 크기 등을 조정

선, 막대 그래프 등 그래프 그리는 코드

```
plt.show()
```

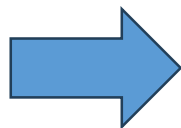
마침표, 모든 옵션이 여기서 반영되고 끝남

25 데이터 시각화의 목적

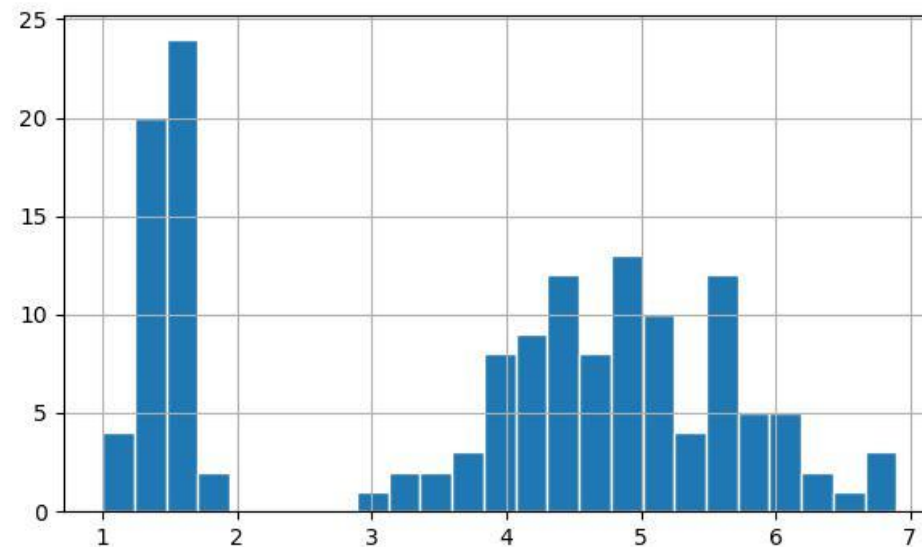
- 우리는 왜 데이터 시각화를 하는가?!
- 우리 스스로 데이터를 이해하기 위해서!

단순 숫자의 나열
이해할 수 없음

0	1.4
1	1.4
2	1.3
3	1.5
4	1.4
5	1.7
6	1.4
7	1.5
8	1.4
9	1.5
10	1.5
11	1.6
12	1.4
13	1.1
14	1.2
15	1.5
16	1.3
17	1.4
18	1.7
19	1.5
20	1.7
21	1.5
22	1.0
23	1.7



시각화를 해보면 어떤 분포(특징)을
가지고 있는지 단번에 이해 됨



25 데이터 시각화의 목적

- 데이터 타입에 따른 적절한 시각화가 필요!
- 데이터 타입은 단순히 딱 2가지만 생각해 보자.

연속된 숫자 데이터

0	22.0
1	38.0
2	26.0
3	35.0
4	35.0
	...
886	27.0
887	19.0
888	NaN
889	26.0
890	32.0

카테고리 데이터

0	man
1	woman
2	woman
3	woman
4	man
	...
886	man
887	woman
888	woman
889	man
890	man

26 데이터에 따른 적절한 시각화

- 연속된 숫자 데이터
- 히스토그램 => 데이터의 분포를 본다
- 히스토그램은 연속된 숫자 데이터를 동일 구간으로 n 개 나누어서 그 구간 안에 데이터가 몇 개가 있는지 막대그래프로 보여준다.
- 이를 통하여 숫자 데이터가 어디에 몰려있는지, 어떻게 퍼져 있는지 등을 알 수 있다.

26 데이터에 따른 적절한 시각화

- 연속된 숫자 데이터
- 히스토그램 => 데이터의 분포를 본다

```
plt.figure()
```

분포를 보고 싶은 연속된 숫자 데이터(1차원)

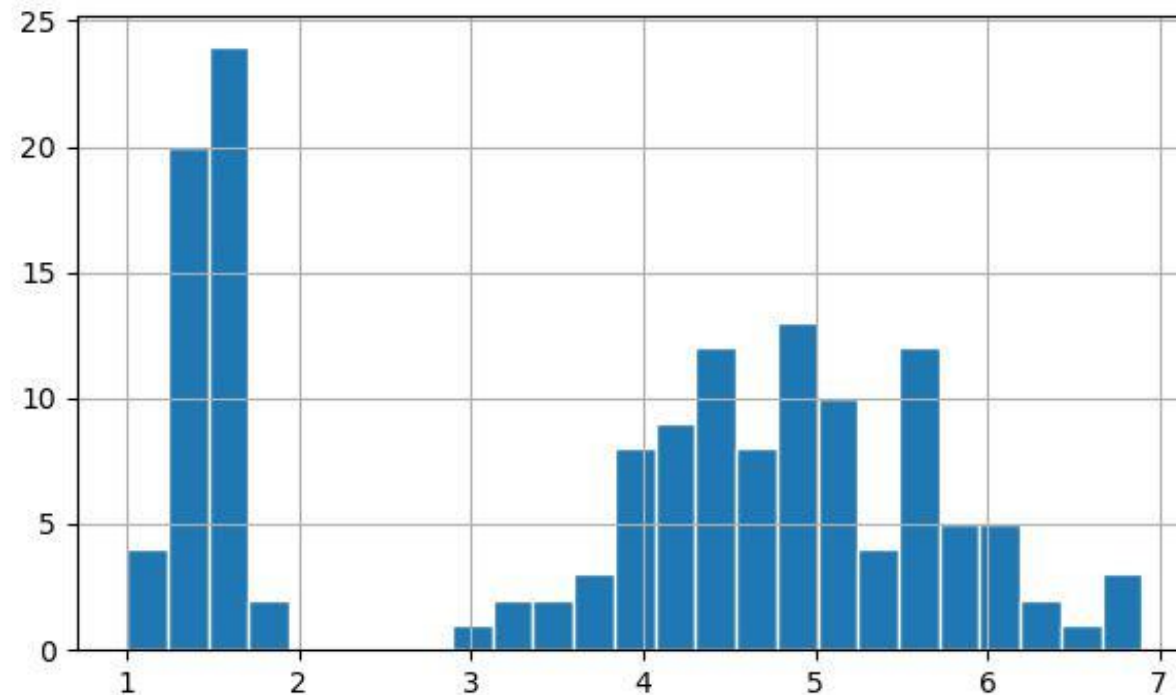
```
plt.hist( 1차원 데이터, bins=25 )
```

숫자 데이터의 구간을 몇 개로 나눌 것인가

```
plt.show()
```

26 데이터에 따른 적절한 시각화

- 연속된 숫자 데이터



26 데이터에 따른 적절한 시각화

- 카테고리 데이터
- countplot => 카테고리별 데이터의 개수를 본다
- seaborn의 countplot을 이용하면 카테고리별로 데이터의 개수를 막대그 래프로 비교를 해 준다.

26 데이터에 따른 적절한 시각화

- 카테고리 데이터

```
import seaborn as sns
```

```
plt.figure()
```

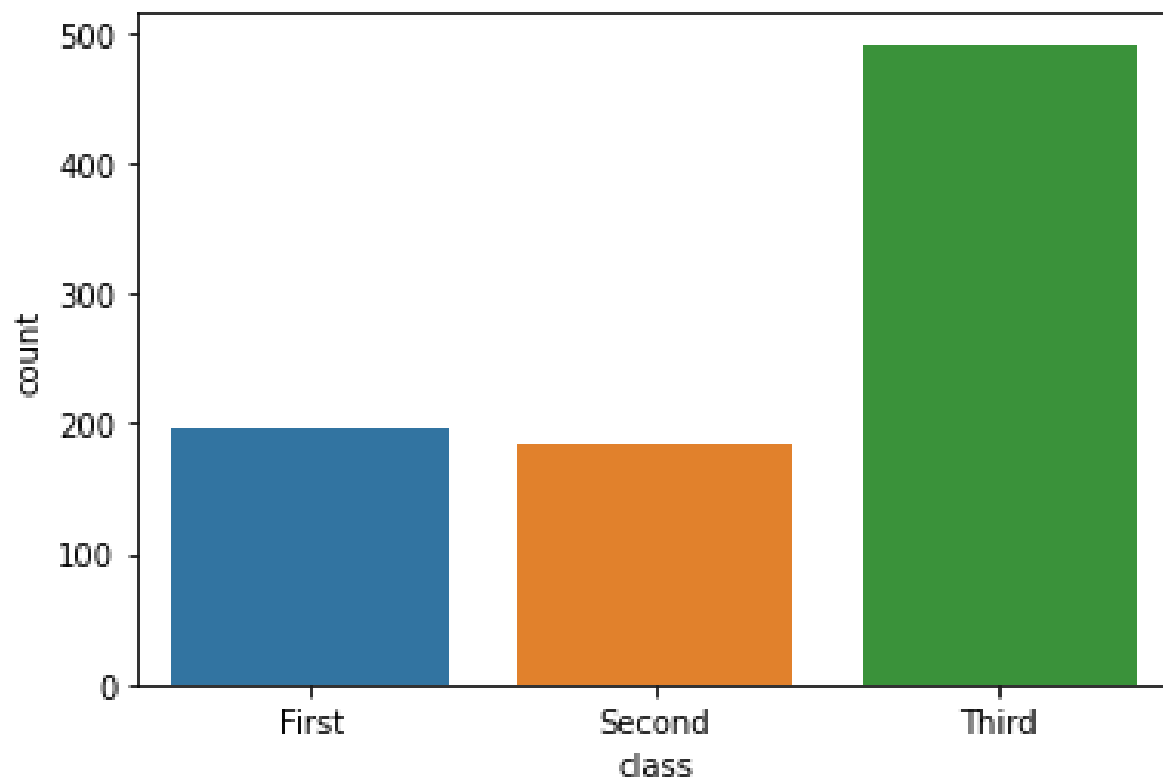
```
sns.countplot( data=데이터프레임 x=카테고리 데이터 컬럼명 )
```

```
plt.show()
```

seaborn은 df를 통째로 넣고 그 중에서 시각화 할 컬럼을 고르는 구조

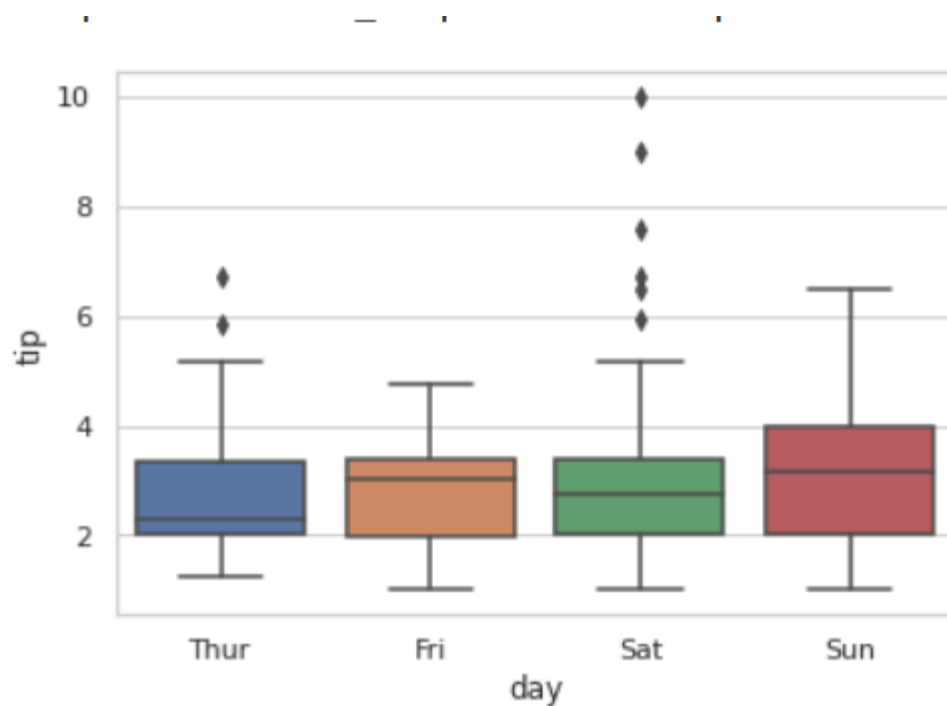
26 데이터에 따른 적절한 시각화

- 카테고리 데이터



26 데이터에 따른 적절한 시각화

- 카테고리 * 연속된 숫자 데이터
- boxplot, violinplot => 카테고리별 연속된 숫자 데이터의 분포를 본다



26 데이터에 따른 적절한 시각화

- 카테고리 * 연속된 숫자 데이터

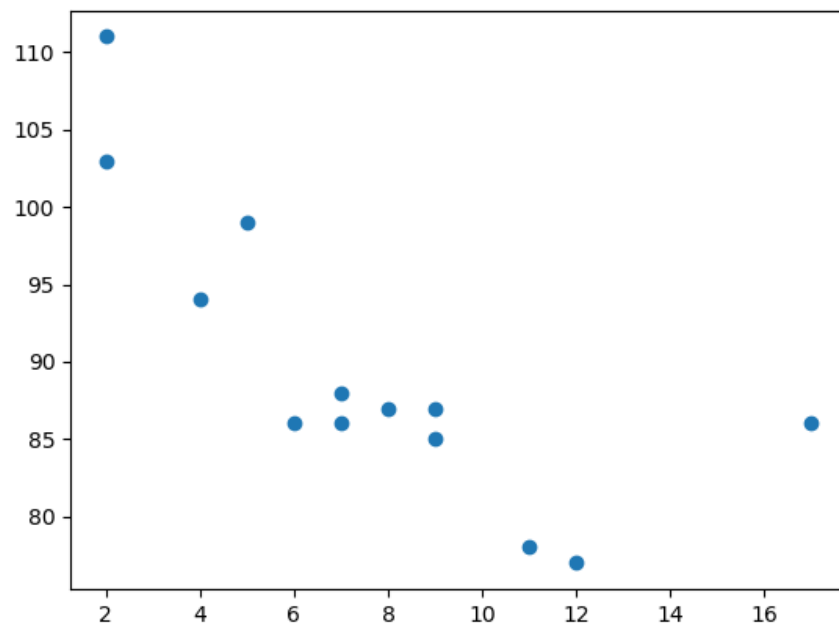
```
plt.figure()
```

```
sns.boxplot( data=데이터프레임, x=카테고리 데이터 컬럼명, y=연속된 숫자 데이터 컬럼명 )
```

```
plt.show()
```

26 데이터에 따른 적절한 시각화

- 연속된 숫자 데이터 * 연속된 숫자 데이터
- scatter=> 산포도. 두 데이터가 만나는 지점에 표시를 하여 두 데이터가 어떻게 퍼져있는지 살펴본다



26 데이터에 따른 적절한 시각화

- 연속된 숫자 데이터 * 연속된 숫자 데이터

```
plt.figure()

plt.scatter(x=x축에 들어갈 1차원 데이터, y=y축에 들어갈 1차원 데이터)

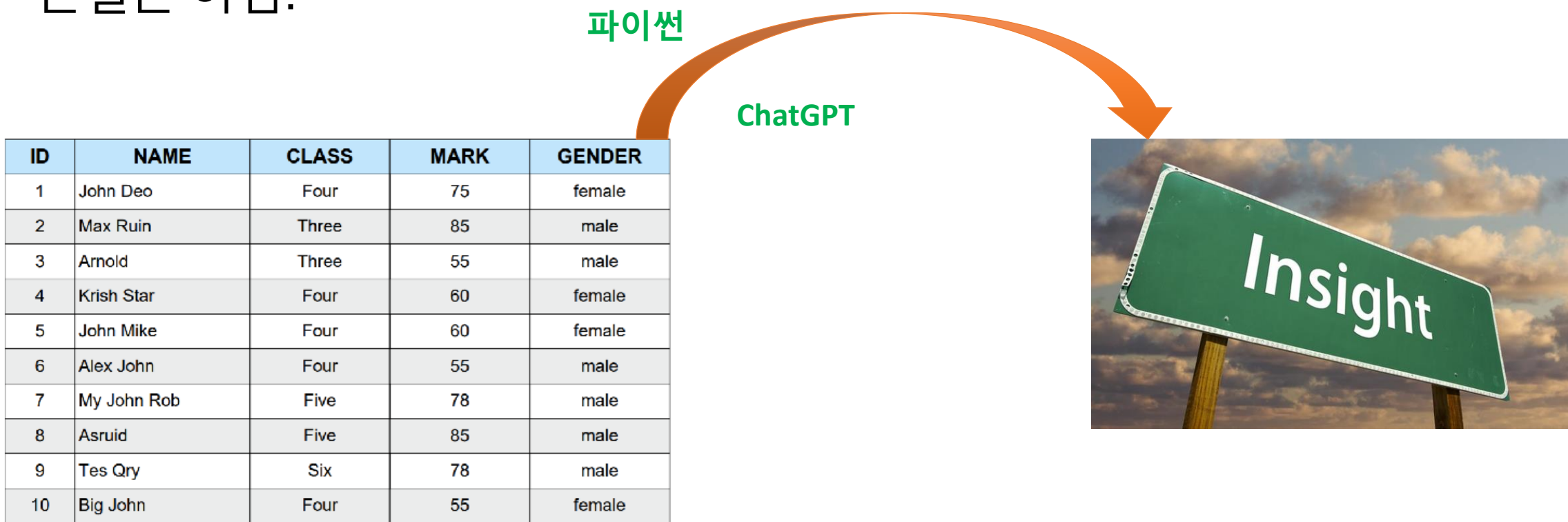
plt.show()
```

07 데이터 분석과 ChatGPT

학습목표 : ChatGPT를 활용하여 데이터 분석을 수행한다.

27 데이터 분석과 ChatGPT

- 데이터 분석이란 데이터에 숨겨진 패턴과 인사이트를 추론해내는 과정이다. 파이썬과 ChatGPT는 추론해내는 과정에서 사용하는 도구(tool)일 뿐이지 본질은 아님!



27 데이터 분석과 ChatGPT

- 파이썬 => table등을 조정할 수 있는 tool
- ChatGPT => 파이썬, table을 조정할 수 있는 tool (**meta tool**)
- 본질은 앞의 파이썬 + ChatGPT와 다를것이 없음

27 데이터 분석과 ChatGPT

- 데이터 분석에서 추가적인 Tip
- 문맥 전달 => 내가 지금 분석하는 table의 형태를 알려줘야 함

**dtypes를 이용하여
컬럼명과 컬럼 별
데이터 타입을 전달**

```
titanic_df.dtypes
```

survived	int64
pclass	int64
sex	object
age	float64
sibsp	int64
parch	int64
fare	float64
embarked	object
class	category
who	object
adult_male	bool
deck	category
embark_town	object
alive	object
alone	bool

27 데이터 분석과 ChatGPT

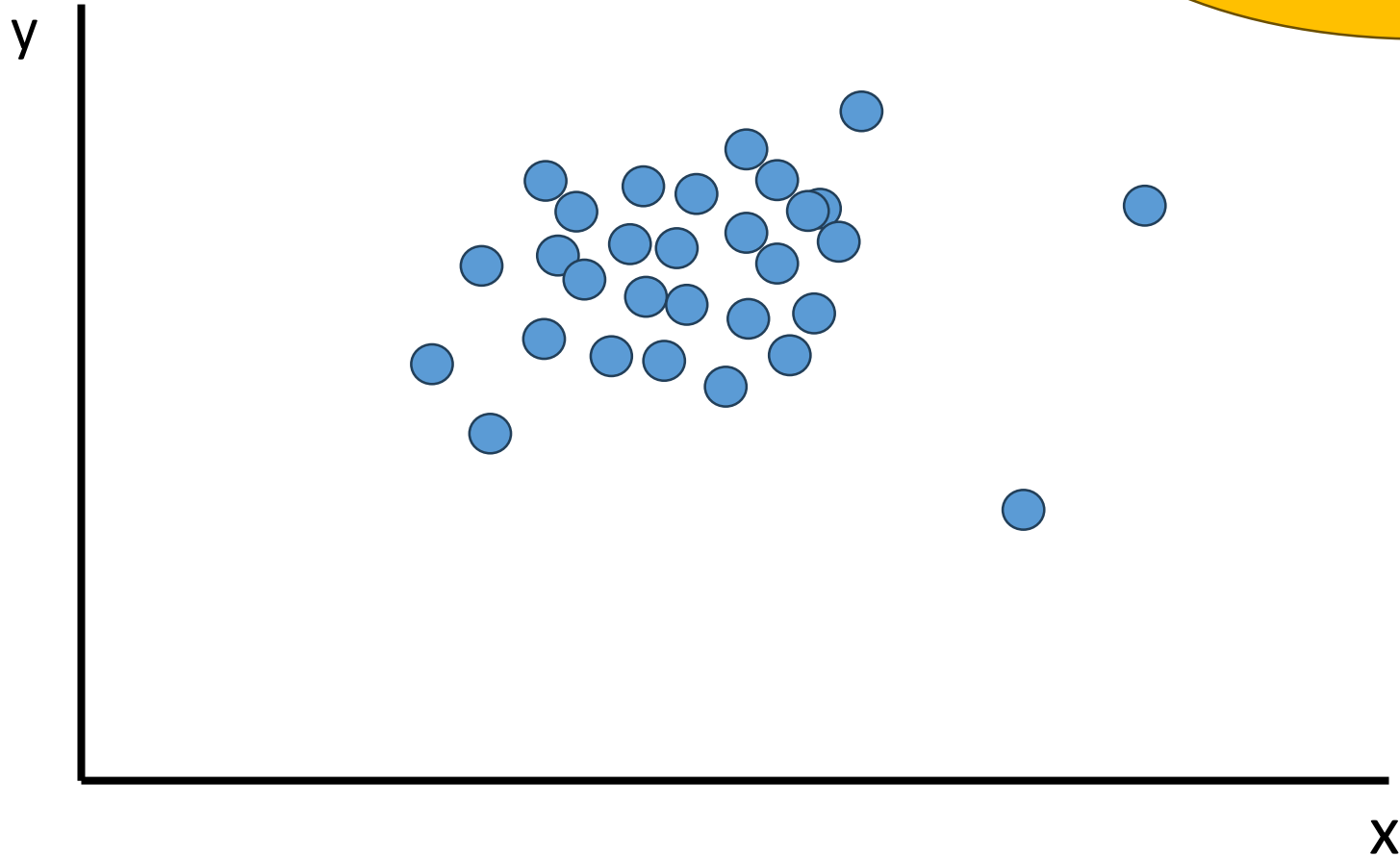
- 데이터 분석에서 추가적인 Tip
- 데이터 시각화는 뼈대만 만들고 나머지 꾸미는 것은 ChatGPT에게! (뼈대는 건드리지 말고 꾸미는 코드만 추가해 달라 한다)

08 머신러닝과 파이썬 기초

학습목표 : 머신러닝의 기초에 대해서 학습한다.

28 머신러닝의 직관

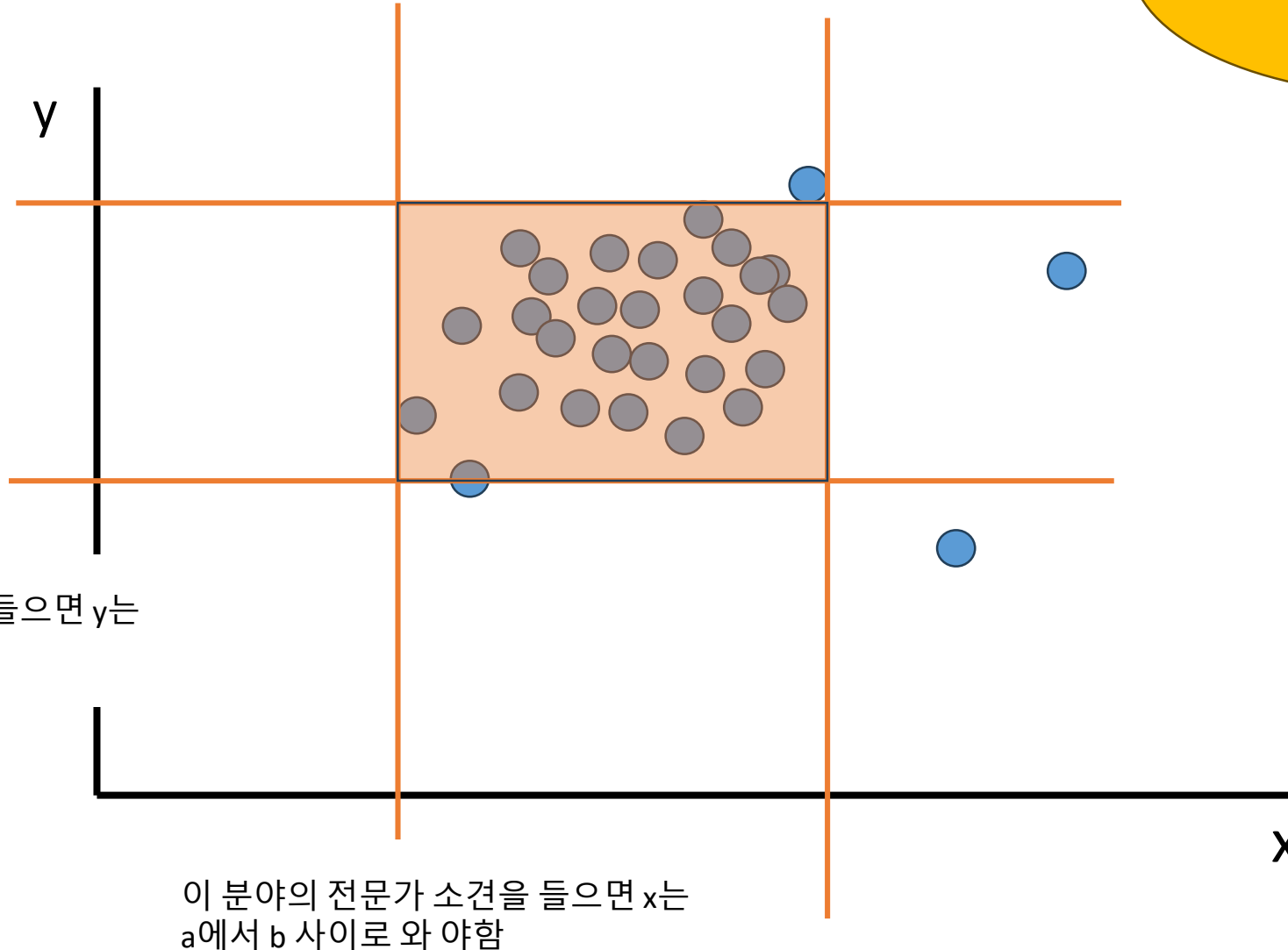
x, y축 어디에 점이
찍히는지 알아내야
한다고 가정해 보자



28 머신러닝의 직관

지식/규칙 기반
전문가 시스템

이 분야의 전문가 소견을 들으면 y 는
 c 에서 d 사이로 와야 함



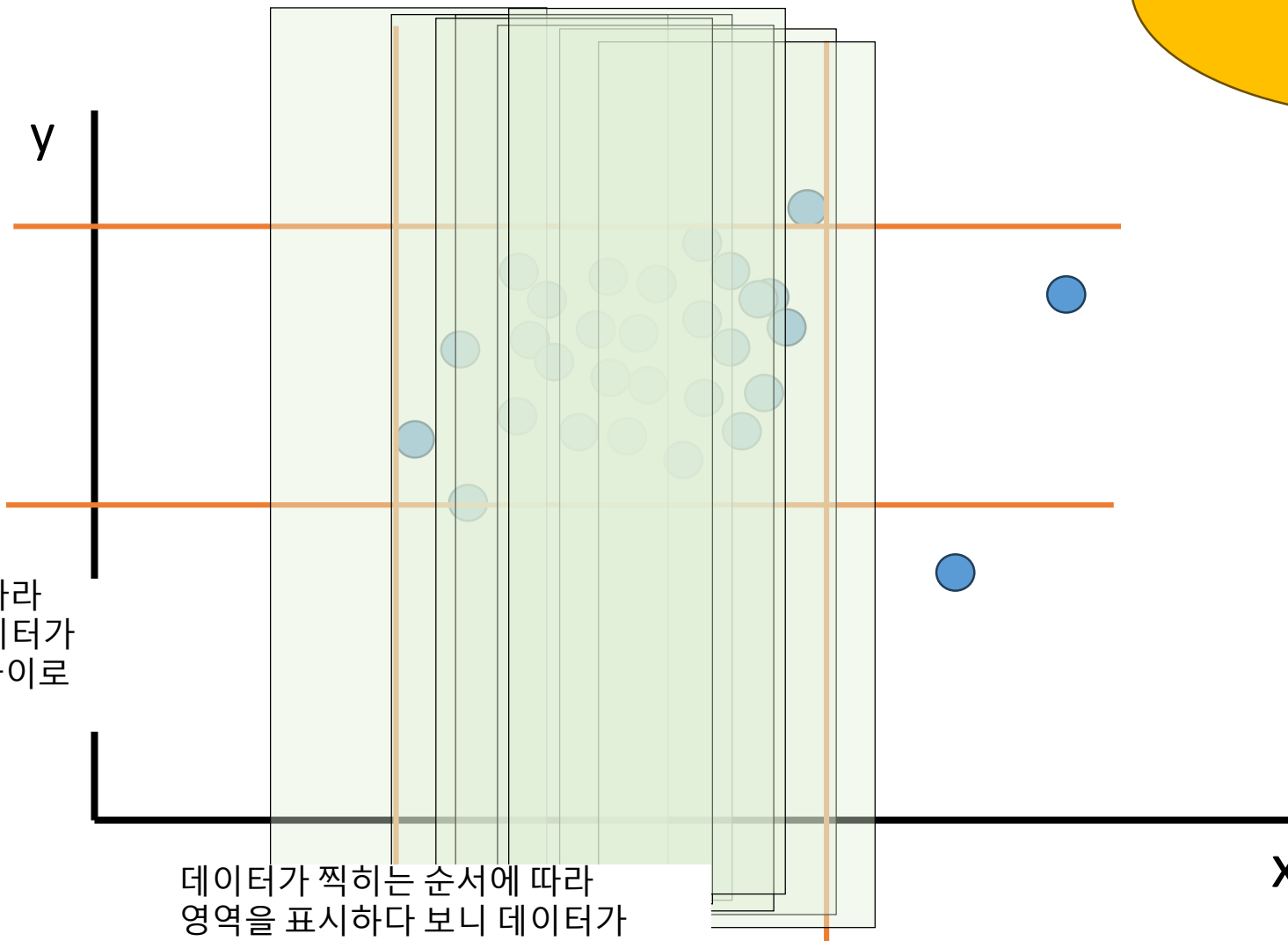
이 분야의 전문가 소견을 들으면 x 는
 a 에서 b 사이로 와야 함

28 머신러닝의 직관

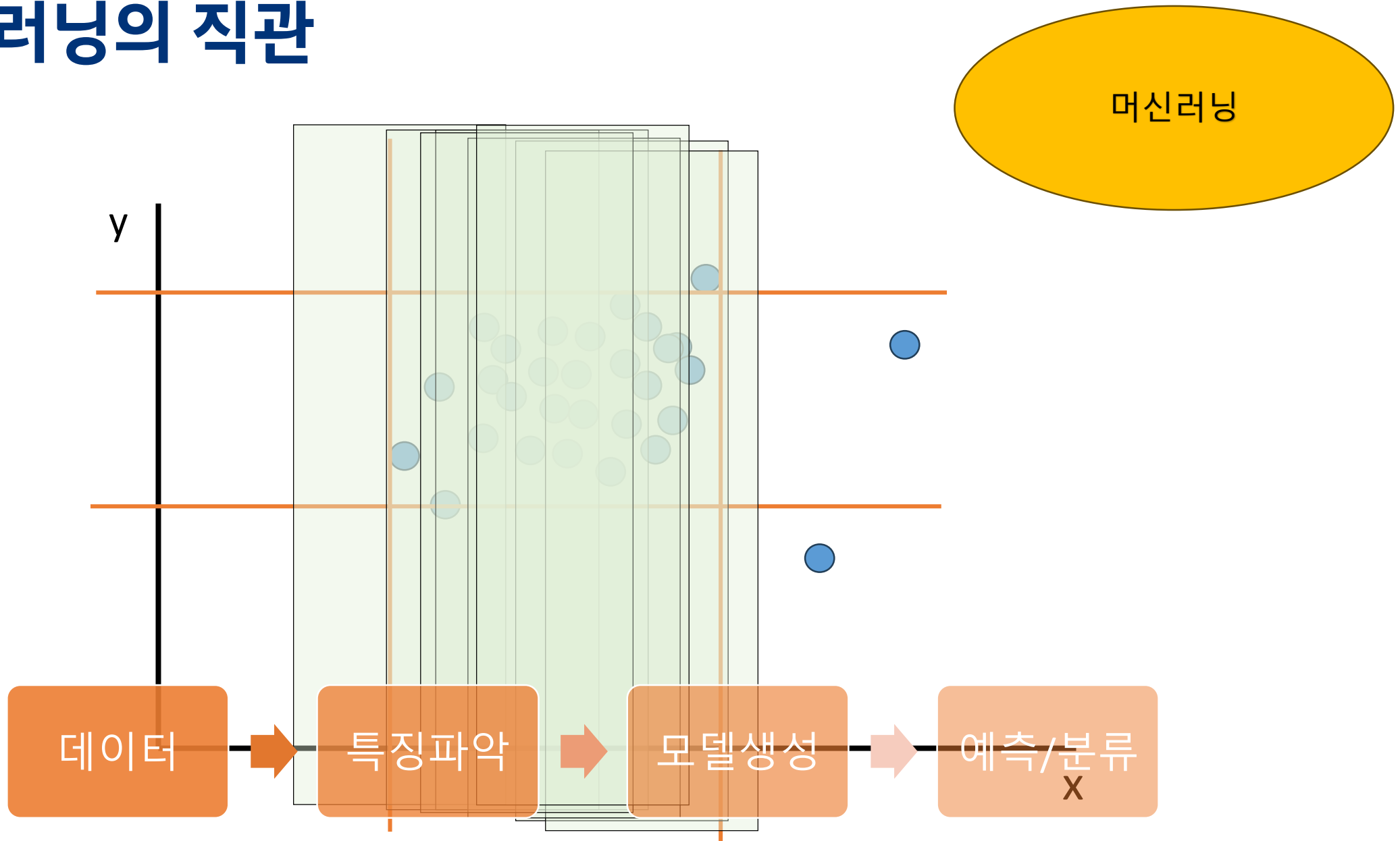
머신러닝

데이터가 찍히는 순서에 따라
영역을 표시하다 보니 데이터가
찍히는 y축은 결국 c와 d 사이로
수렴

데이터가 찍히는 순서에 따라
영역을 표시하다 보니 데이터가
찍히는 x축은 결국 a와 b 사이로
수렴



28 머신러닝의 직관

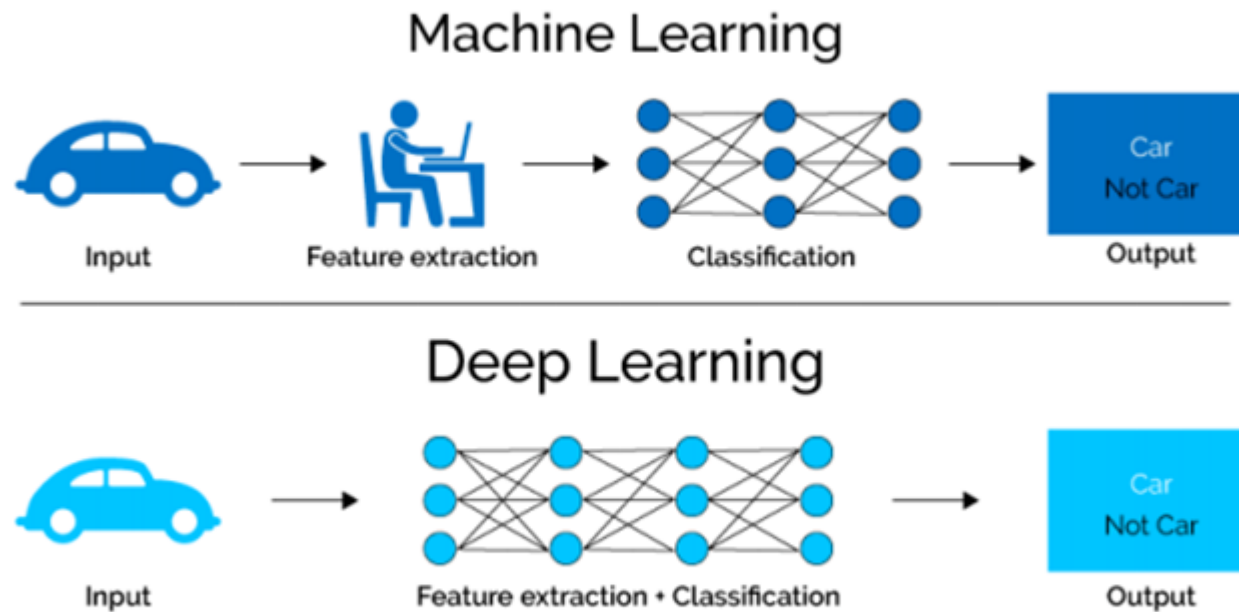


28 머신러닝의 직관

- 머신러닝의 예측이라는 것은 주어진 데이터를 일반화하는 것과 같다.
- 주어진 데이터에서 어떤 규칙을 찾아내고 이를 일반화하여 새로운 데이터에 적용하여 결과값을 얻는 것이다.

28 머신러닝 vs 딥러닝

그림4 머신러닝 VS 딥러닝



자료: Towards Data Science, 메리츠증권증권 리서치센터

28 머신러닝 모델의 종류

- 지도학습(Supervised Learning)
 - 훈련데이터(train data set)에 타깃(target)또는 레이블(label)이라고 불리는 정답이 포함된 학습방식
 - 훈련이 끝나면 정답이 포함되지 않은 새로운 데이터 셋(test data set)을 사용하여 정답을 예측
- 비지도학습(Unsupervised Learning)
 - 훈련데이터(train data set)가 타깃(target)또는 레이블(label)이라고 불리는 정답이 포함하지 않음
 - 훈련데이터에 별다른 가이드라인이 없어 기계학습모델이 스스로 학습하여 결과를 만듦(=> 이 결과가 무엇인지 해석의 문제가 존재)

28 머신러닝 모델의 종류

- 지도학습(Supervised Learning)
 - 분류 (classification) - 어떤 집단에 속하는 지 판별해 내는 것
 - 정답이 카테고리 데이터
 - 회귀 (regression) - 주어진 값에 대한 결과값을 예측하는 것
 - 정답이 연속된 숫자 데이터

29 파이썬으로 머신러닝 실습하기 - scikit-learn

- 파이썬 머신러닝 모듈인 사이킷런(scikit-learn)을 이용한다.
- 매우 잘 만들어진 모듈로 어떤 머신러닝 모델을 사용하든지 구조가 같다!
- **모델 생성**(모델 객체를 만든다. 이 때 만들어진 모델은 아직 데이터에 독립적인 순수한 백지같은 모델)
- **데이터에 훈련**(모든 모델에는 fit 함수가 있어서 이를 이용하여 모델을 데이터에 훈련 시킨다)
- **예측 및 검증**(fit함수를 이용하여 특정 데이터를 가지고 훈련된 모델은 predict 함수를 이용하여 예측하거나 score 함수를 이용하여 검증할 수 있다)

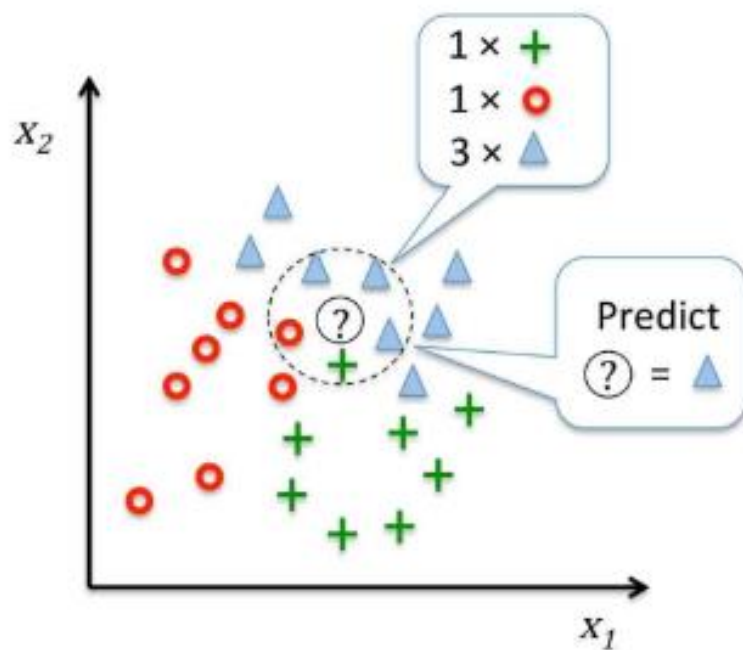
모듈	설명
<code>sklearn.datasets</code>	내장된 예제 데이터 세트
<code>sklearn.preprocessing</code>	다양한 데이터 전처리 기능 제공 (변환, 정규화, 스케일링 등)
<code>sklearn.feature_selection</code>	특징(feature)을 선택할 수 있는 기능 제공
<code>sklearn.feature_extraction</code>	특징(feature) 추출에 사용
<code>sklearn.decomposition</code>	차원 축소 관련 알고리즘 지원 (PCA, NMF, Truncated SVD 등)
<code>sklearn.model_selection</code>	교차 검증을 위해 데이터를 학습/테스트용으로 분리, 최적 파라미터를 추출하는 API 제공 (GridSearch 등)
<code>sklearn.metrics</code>	분류, 회귀, 클러스터링, Pairwise에 대한 다양한 성능 측정 방법 제공 (Accuracy, Precision, Recall, ROC-AUC, RMSE 등)
<code>sklearn.pipeline</code>	특징 처리 등의 변환과 ML 알고리즘 학습, 예측 등을 묶어서 실행할 수 있는 유틸리티 제공
<code>sklearn.linear_model</code>	선형 회귀, 릿지(Ridge), 라쏘(Lasso), 로지스틱 회귀 등 회귀 관련 알고리즘과 SGD(Stochastic Gradient Descent) 알고리즘 제공
<code>sklearn.svm</code>	서포트 벡터 머신 알고리즘 제공
<code>sklearn.neighbors</code>	최근접 이웃 알고리즘 제공 (k-NN 등)
<code>sklearn.naive_bayes</code>	나이브 베이즈 알고리즘 제공 (가우시안 NB, 다항 분포 NB 등)
<code>sklearn.tree</code>	의사 결정 트리 알고리즘 제공
<code>sklearn.ensemble</code>	앙상블 알고리즘 제공 (Random Forest, AdaBoost, GradientBoost 등)
<code>sklearn.cluster</code>	비지도 클러스터링 알고리즘 제공 (k-Means, 계층형 클러스터링, DBSCAN 등)

09 분류모델

학습목표 : 머신러닝 분류모델에 대해서 학습한다

30 KNN 모델

- 분류하고자 하는 샘플에 대한 k개의 근접한 이웃(가장 가까운 데이터)을 찾음
- 다수결 투표 방식으로 분류 레이블을 할당함



30 KNN 모델 장점

- 이해하기 쉬움: KNN은 매우 직관적인 모델로, 새로운 데이터 포인트에 대한 예측을 이해하고 설명하기가 비교적 쉬움
- 훈련 단계가 필요 없음: KNN은 훈련 데이터를 직접 사용하기 때문에 별도의 훈련 과정이 필요 없음. 이를 '게으른 학습기(lazy learner)' 라고 함
- 비선형 데이터에도 효과적: KNN은 데이터의 분포에 대한 가정이 없기 때문에 비선형 데이터에도 잘 작동

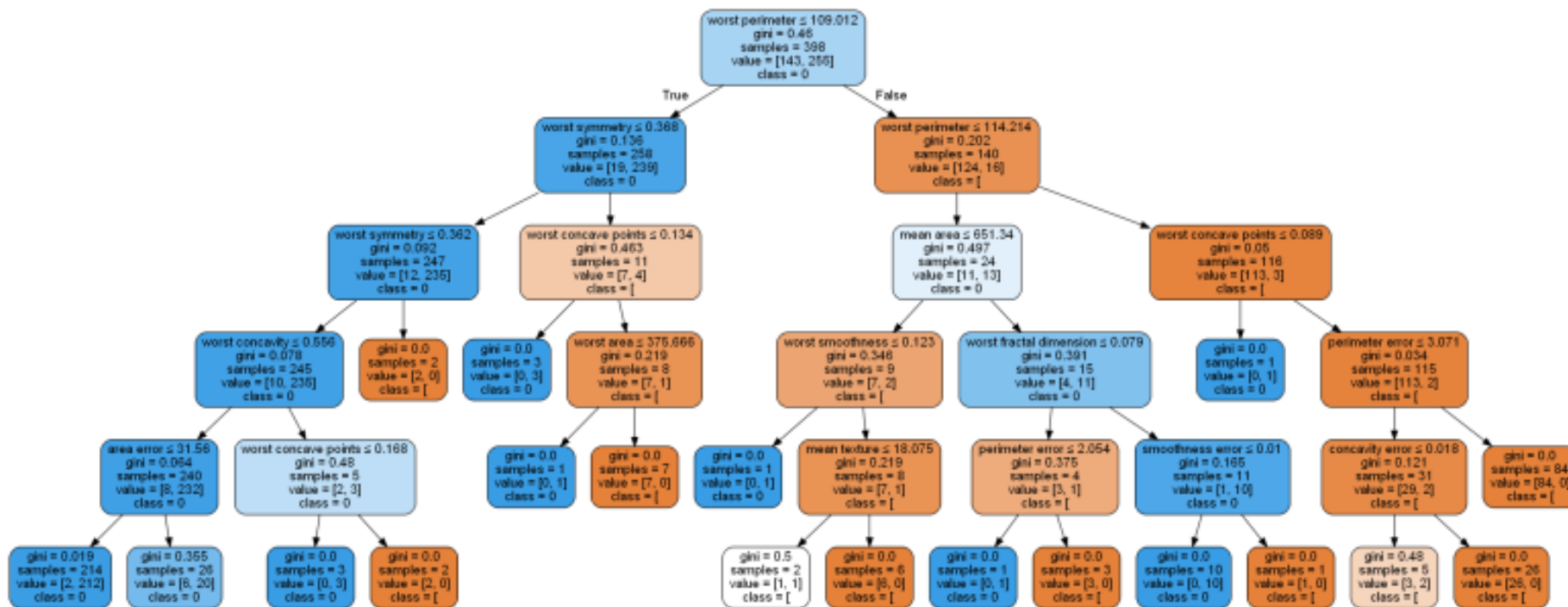
30 KNN 모델 단점

- 규모가 큰 데이터셋에 비효율적: KNN은 예측을 위해 전체 데이터셋을 검색하므로 데이터셋의 크기가 클수록 계산 비용이 크게 증가
- 차원의 저주: 차원이 높아질수록 각 차원에서의 데이터 포인트 간 거리가 증가하여, KNN의 성능이 저하될 수 있음 (feature 개수 컨트롤 중요!)
- 이상치에 민감: KNN은 이웃 데이터 포인트에 크게 의존하기 때문에, 이상치(outlier)가 포함된 데이터셋에서는 성능이 저하될 수 있음
- 특성의 스케일에 민감: 서로 다른 스케일을 가진 특성이 존재하면, 거리 계산에 영향을 미쳐 결과적으로 모델의 성능에 부정적인 영향을 줄 수 있음

- 스케일링: 모든 특성이 동일한 스케일을 가지도록 정규화나 표준화를 수행해야 함
- 불필요한 특성 제거: 관련성이 낮은 특성은 모델의 성능을 저하시킬 수 있으므로 주의 깊게 선택하거나 제거
- 차원 축소: 데이터의 차원이 매우 높은 경우 차원 축소 기법을 적용하여 차원의 저주를 완화할 수 있음

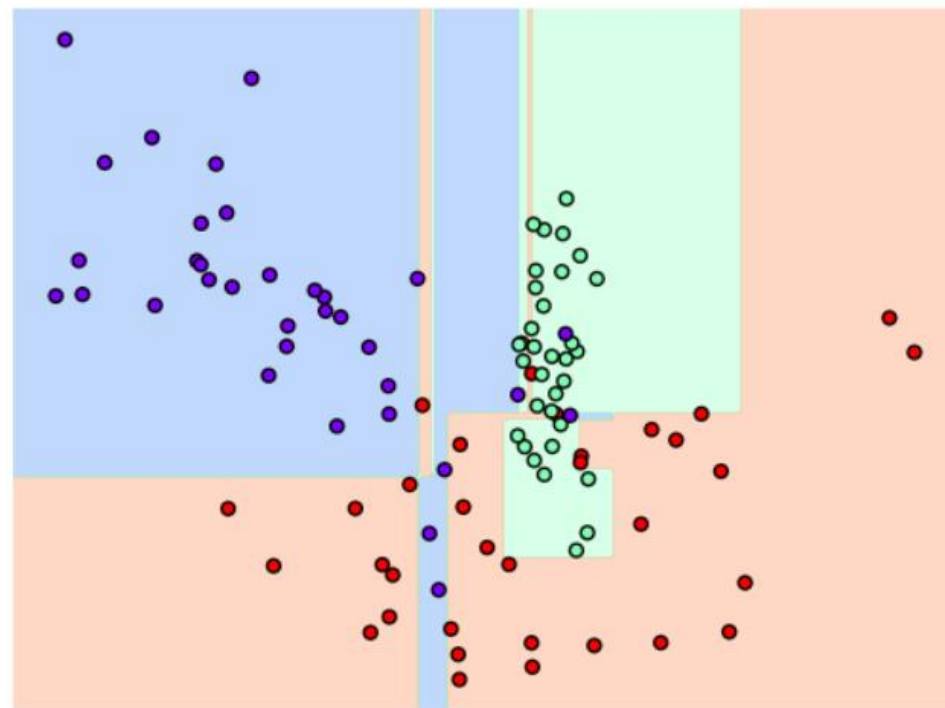
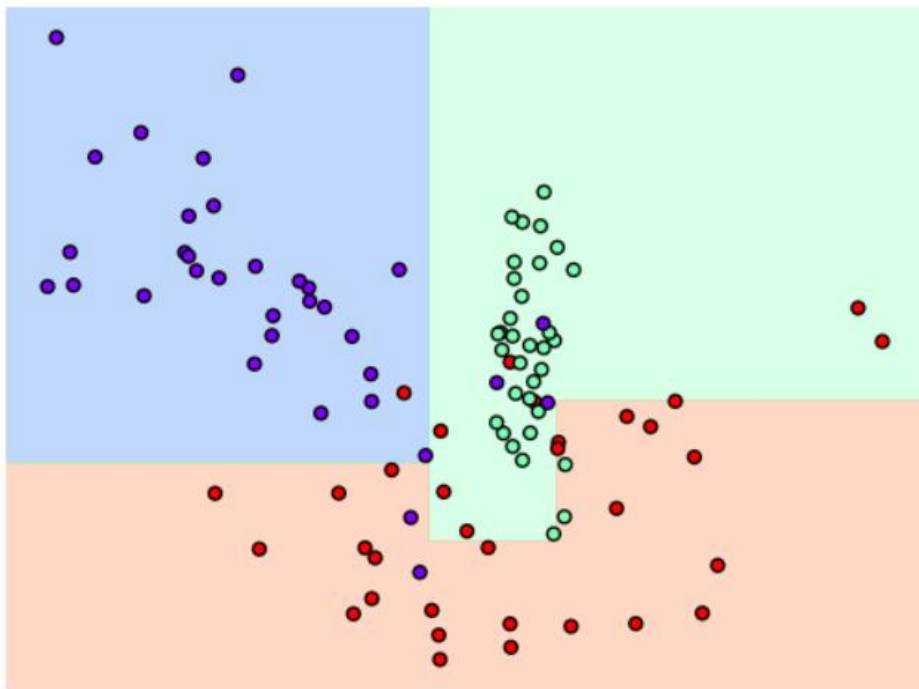
31 결정 트리

- 결정트리는 매우 쉽고, 스케일링이나 정규화 등 사전 데이터 가공의 영향이 적음
- 예측 성능을 향상시키기 위해 복잡한 규칙 구조를 가져야 하고 이로 인한 과적합이 발생, 예측 성능 떨어질수도 있음



31 결정 트리

- 복잡하고 엄격한 학습 규칙으로 인한 과적합 사례
- 결정트리의 max depth를 제한하거나, 말단 노드의 데이터 수를 완화하는 방식으로 해결



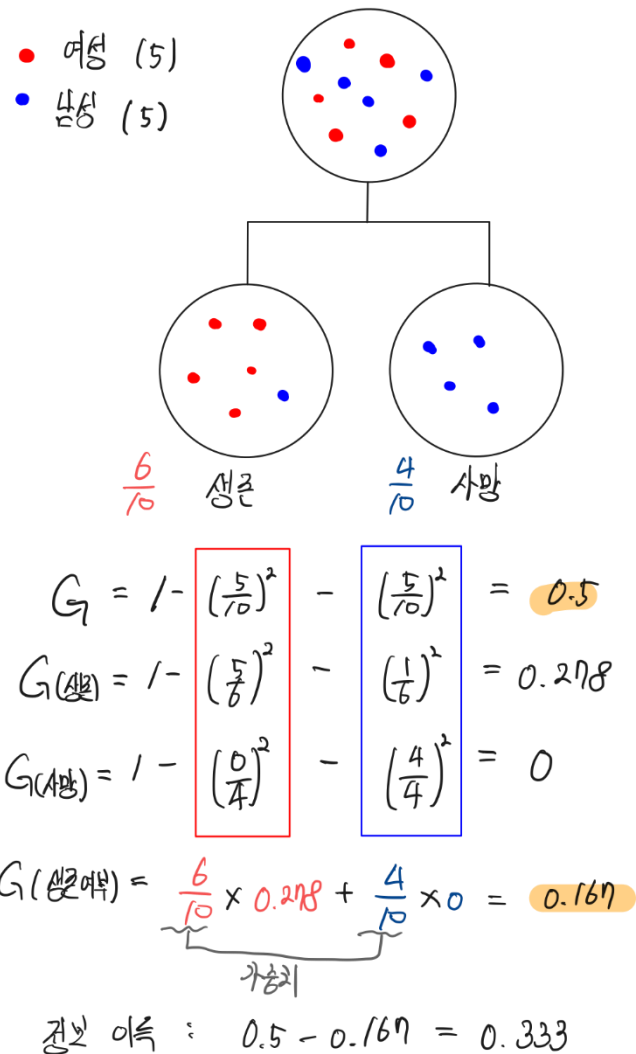
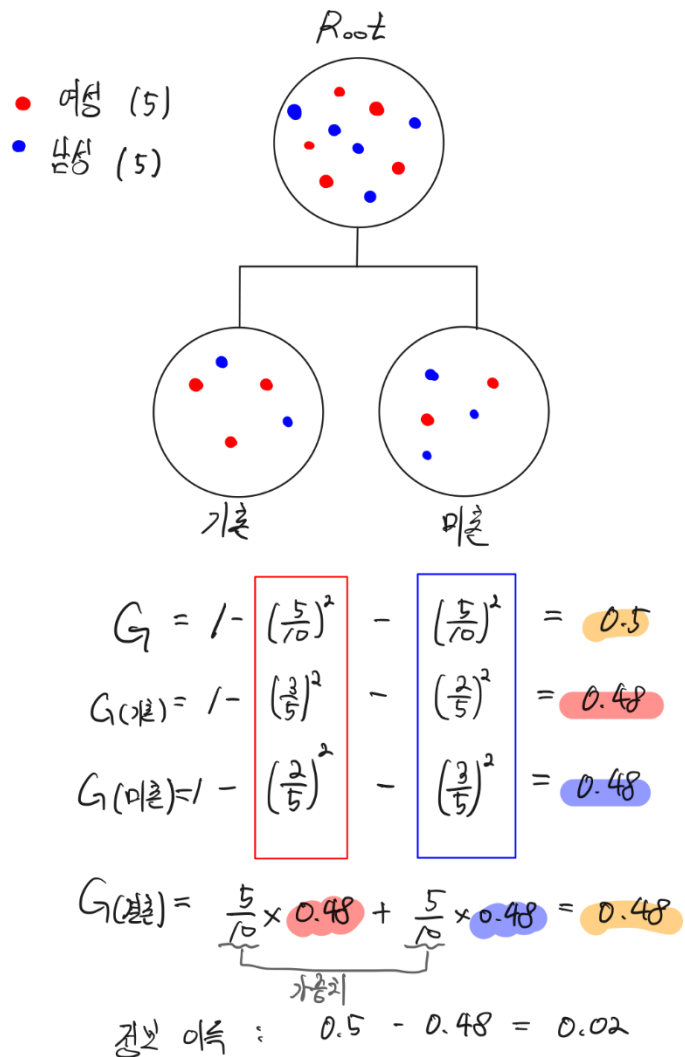
31 결정 트리

- 지니 불순도
- 주어진 데이터 집합의 불확실성을 측정
- 지니 불순도가 낮을수록 분할 후의 집합이 더 순수해진다는 의미

$$Gini = G_{(A)} = 1 - \sum_{i=1}^N (p_i)^2$$

- p 는 특정 카테고리에 속할 확률, N 은 카테고리의 갯수

31 결정 트리



- 이해와 해석이 쉬움: 결정 트리는 규칙 기반의 접근 방식을 사용하여 결과를 쉽게 이해하고 해석할 수 있음
- 데이터 전처리 요구가 적음: 결정 트리는 다른 많은 머신러닝 모델과 달리, 특성 스케일링이나 정규화가 필요하지 않음
- 비선형 관계 모델링 가능: 결정 트리는 데이터의 비선형 패턴을 포착할 수 있어 복잡한 데이터 구조를 모델링하는 데 유용
- 결측치 처리 용이: 결정 트리는 데이터 내의 결측치를 자체적으로 다룰 수 있는 기능을 가지고 있음

31 결정 트리 단점

- 과적합(Overfitting) 경향: 결정 트리는 훈련 데이터에 대해 너무 자세하게 학습하면서, 새로운 데이터에 대한 일반화 성능이 떨어질 수 있음
- 안정성 부족: 작은 데이터 변화에도 트리 구조가 크게 바뀔 수 있어, 모델의 예측이 불안정
- 균형 잡히지 않은 트리 생성: 일부 클래스가 다른 클래스보다 많은 경우 불균형한 트리가 만들어지고, 일방적인 결정 경로가 형성될 수 있음

31 결정 트리 주의점

- 가지치기(Pruning) 사용: 과적합을 방지하기 위해 트리의 깊이나 리프 노드의 최소 데이터 수를 제한하는 가지치기 기법을 사용. 사이킷런에서는 max_depth, min_samples_split, min_samples_leaf 등의 매개변수로 이를 조절할 수 있음
- 데이터 균형 조절: 불균형한 데이터셋에서는 모델의 성능이 저하될 수 있으므로, 적절한 샘플링 기법을 사용하여 균형을 맞춰주는 것이 좋음
- 앙상블 기법 고려: 단일 결정 트리의 불안정성과 과적합 문제를 완화하기 위해 랜덤 포레스트나 그래디언트 부스팅 같은 앙상블 기법을 사용할 수 있음

32 앙상블 학습 (Ensemble Learning)

- 여러 개의 분류기를 생성하고 이를 결합하여 보다 정확한 최종 예측을 도출하는 기법
- 집단지성 느낌
- 크게 보팅(voting), 배깅(bagging), 부스팅(boosting)의 방법이 있음

32 앙상블 학습 (Ensemble Learning)

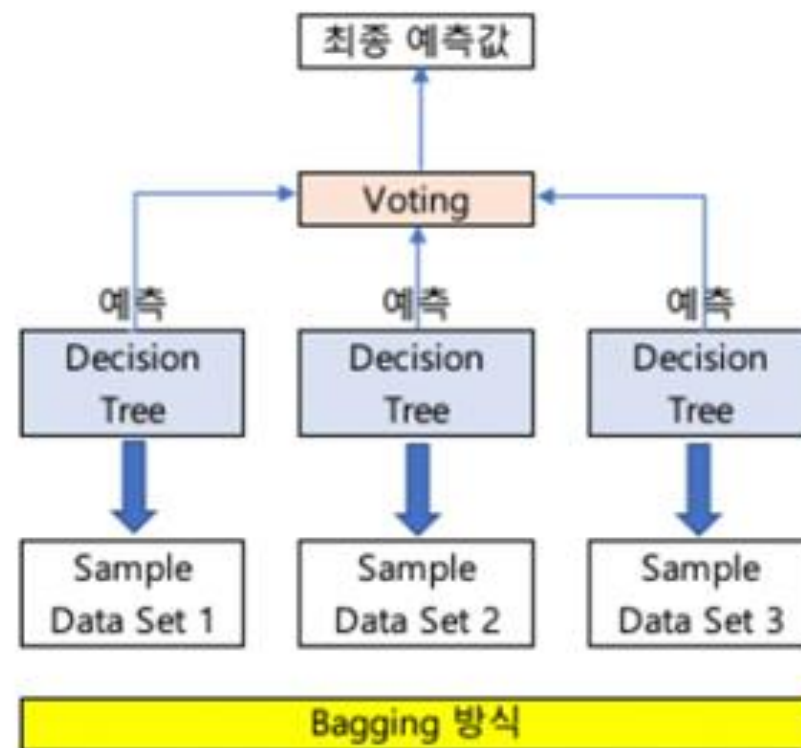
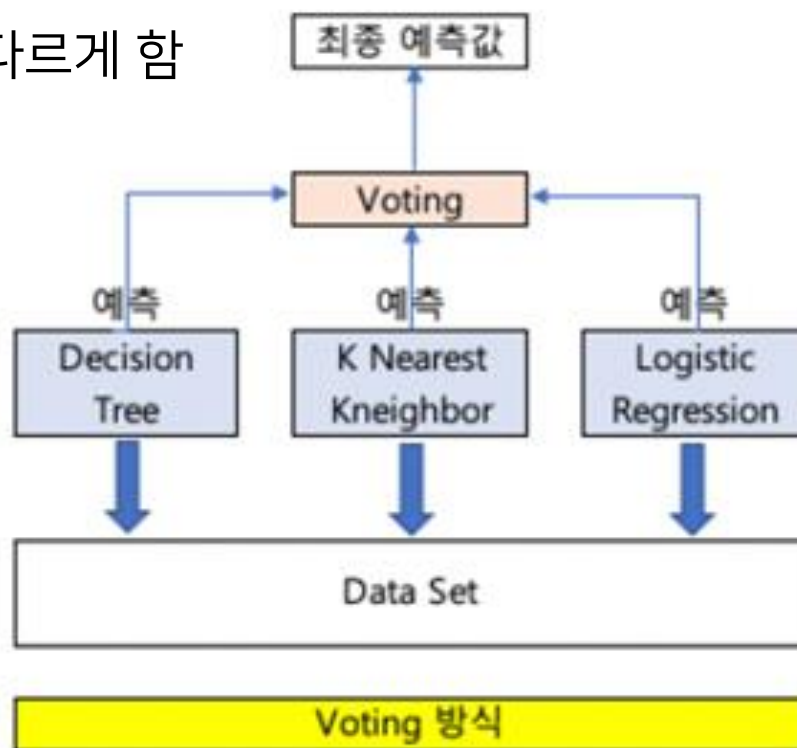
- 여러 개의 분류기를 생성하고 이를 결합하여 보다 정확한 최종 예측을 도출하는 기법
- 집단지성 느낌
- 크게 보팅(voting), 배깅(bagging), 부스팅(boosting)의 방법이 있음

32 앙상블 학습 (Ensemble Learning)

- 단일 모델의 약점을 다수의 모델로 보완
- 단일 모델로는 성능이 떨어질지는 모르나 다양한 모델을 섞어서 전체 성능의 향상을 도모
- 많은 경우 결정 트리 모델을 이용하여 앙상블 모델을 구성
- 결정 트리의 단점인 오버피팅 문제를 해결하며 장점인 직관적 분류 기준은 강화 됨

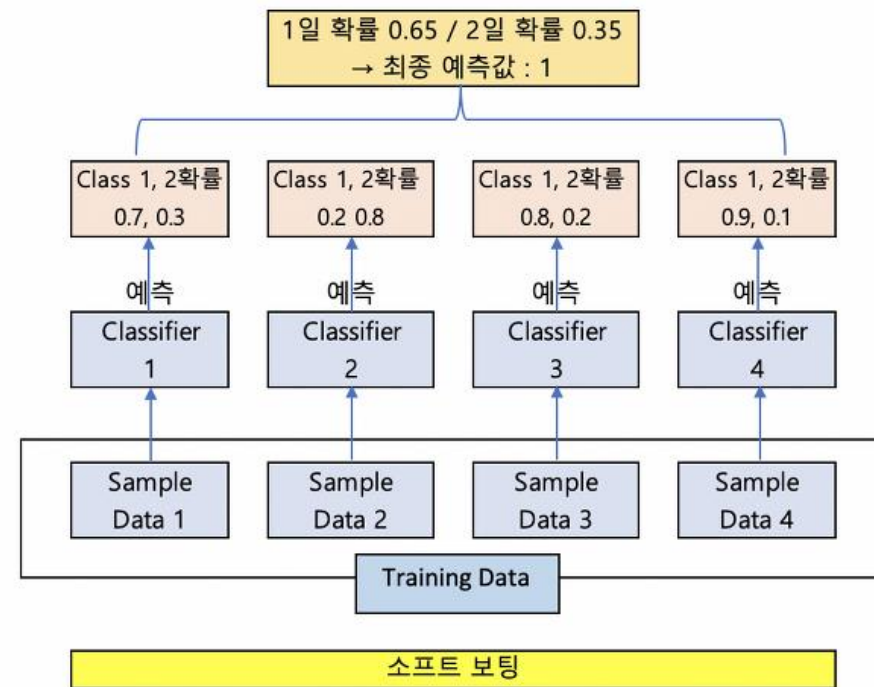
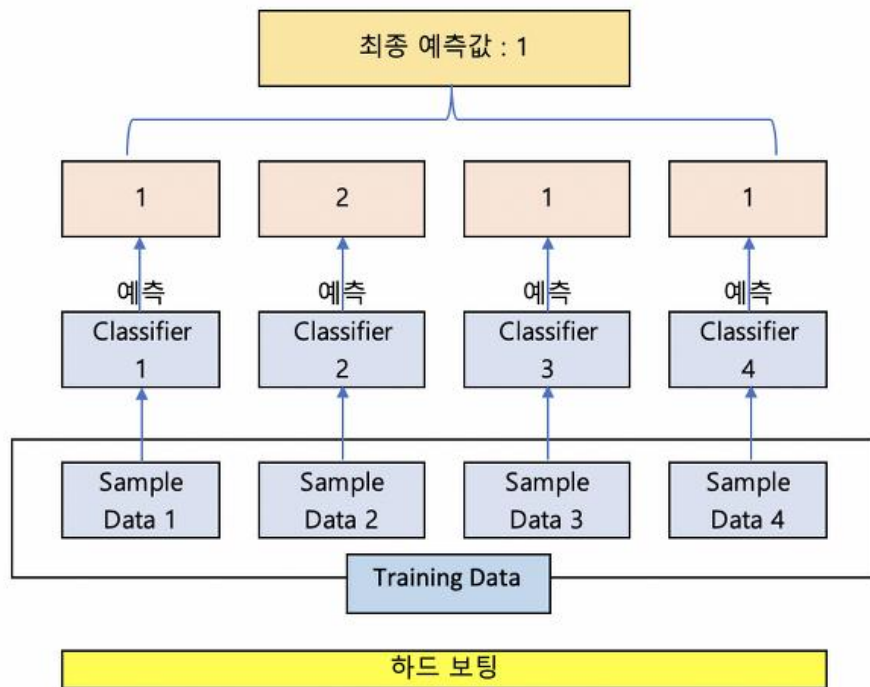
32 보팅과 배깅

- 여러 모델들의 투표를 통해 최종 예측 결과를 결정
- 보팅 => 서로 다른 분류 모델
- 배깅 => 데이터 샘플링을 다르게 함



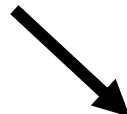
32 하드보팅 & 소프트 보팅

- 하드보팅 => 다수결
- 소프트 보팅 => 각 라벨이 될 확률 평균
- 일반적으로 소프트 보팅의 성능이 더 우수하다고 알려져 있음



33 랜덤 포레스트

				Y



			Y

			Y

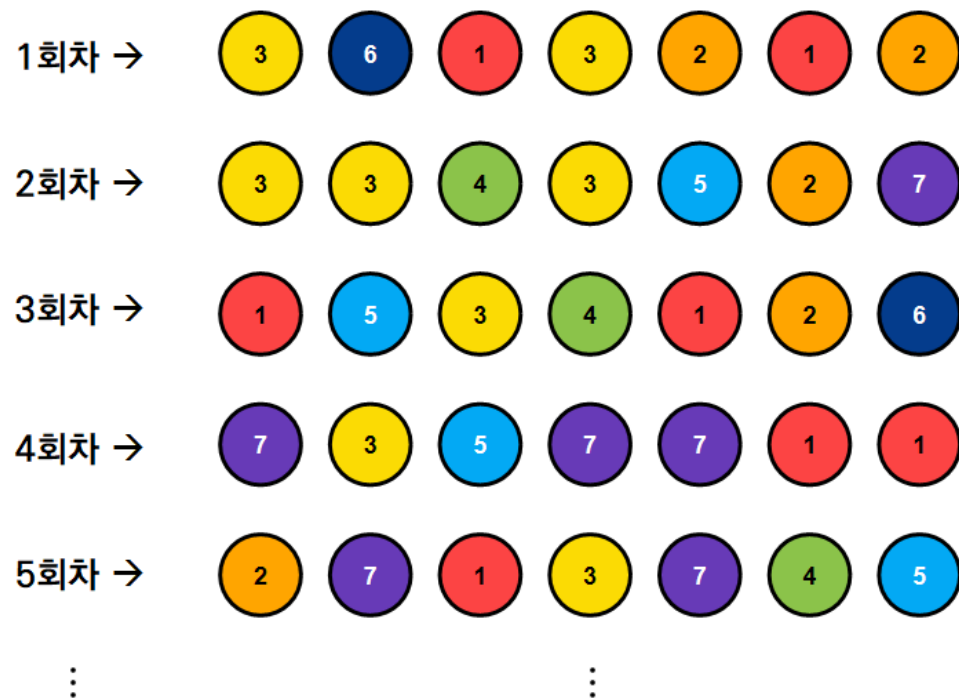
- 매 노드마다 사용할 피쳐를 랜덤하게 선택
=> 노드에 사용할 피쳐들을 무차별하게 선택함으로 써 오버피팅의 문제를 피해가고 다양한 정보를 탐험 가능

33 랜덤 포레스트

- 부트스트랩
- 주어진 표본에서 복원추출하여 표본을 재추출



resample을 통해 재구성한 데이터 셋

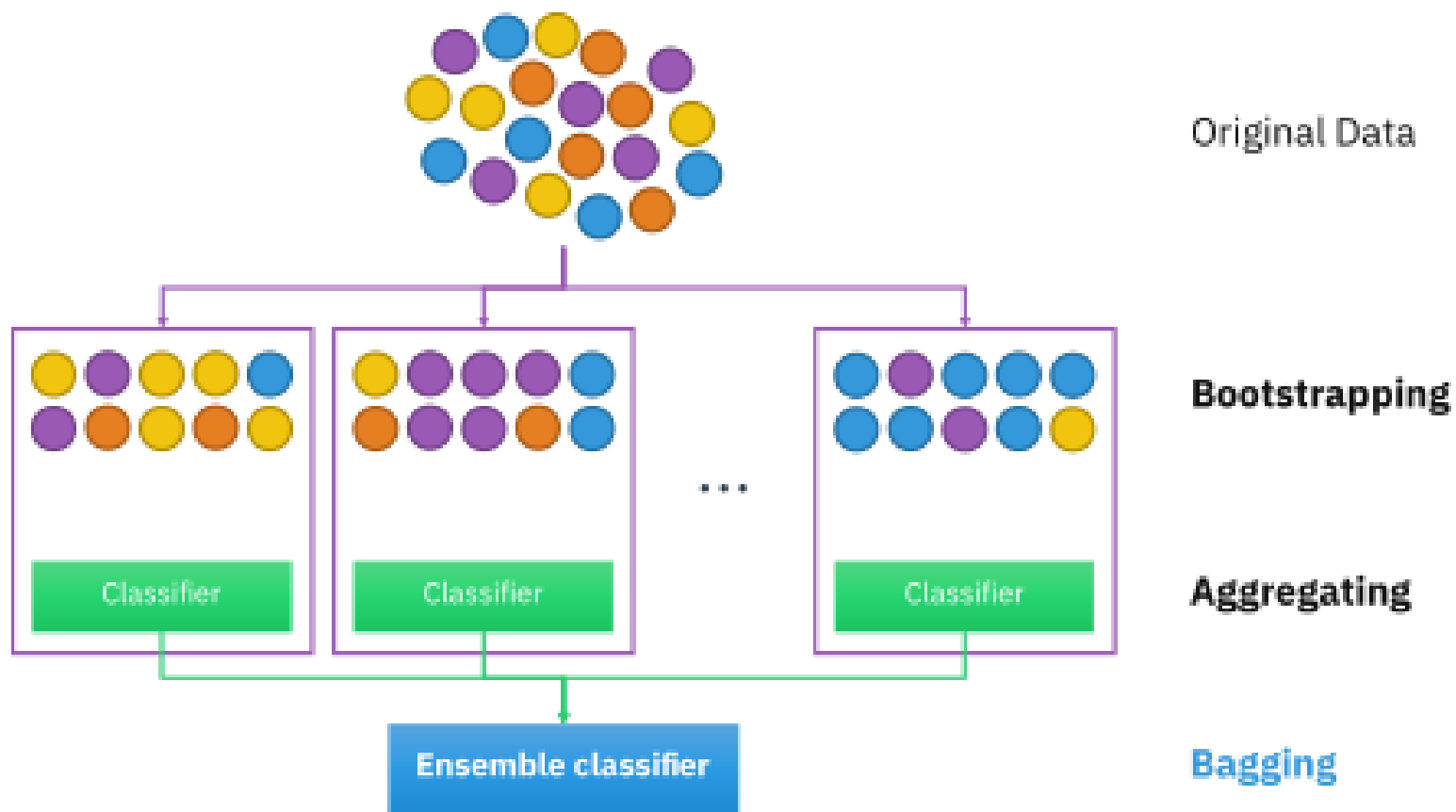


33 랜덤포레스트

- 샘플링을 여러 번 하지 못하는 현실에서 여러 번 추출할 수 있었을 샘플의 estimator값 분포가 어땠을지 논리적으로 추정할 수 있게 해준다
- 평균과 같이 잘 알려진 estimator들은 굳이 쓸 필요가 없지만 표본 median의 분포를 알고 싶다면?!
- 평균 이외의 estimator의 표준오차를 구하려면?!
- 부트스트래핑!!! => 연산력만 받쳐준다면. 요즘은 컴퓨터가 발달해서!

33 랜덤포레스트

- 랜덤 포레스트에서 부트스트래핑은 각 나무에 사용할 데이터를 똑같은 데이터가 아니라 부트스트래핑으로 재구성한다는 데 있다



33 랜덤포레스트

- 결국 랜덤 포레스트의 핵심은 다양성!
- 각 나무가 사용하는 feature와
- 각 나무에 사용되는 데이터 샘플을 다르게 함으로써
- 최대한 다양한(상관관계가 낮은) 나무를 많이 생성!

- 성능: 랜덤 포레스트는 일반적으로 결정 트리보다 높은 정확도를 제공. 여러 개의 결정 트리가 오버피팅의 영향을 상쇄하며, 다양한 데이터 샘플과 특성을 사용하여 학습하기 때문.
- 오버피팅 감소: 개별 결정 트리가 훈련 데이터에 과적합되는 경향이 있지만, 랜덤 포레스트는 여러 트리의 예측을 평균 내어 이 문제를 크게 줄임.
- 특성의 중요도 평가: 랜덤 포레스트는 각 특성의 중요도를 평가할 수 있어, 어떤 특성이 결과에 가장 큰 영향을 미치는지 이해하는 데 도움을 줌.
- 병렬 처리 용이: 각각의 트리는 독립적으로 학습되므로, 랜덤 포레스트는 병렬 처리를 통해 효율적으로 학습할 수 있음.

33 랜덤포레스트 단점

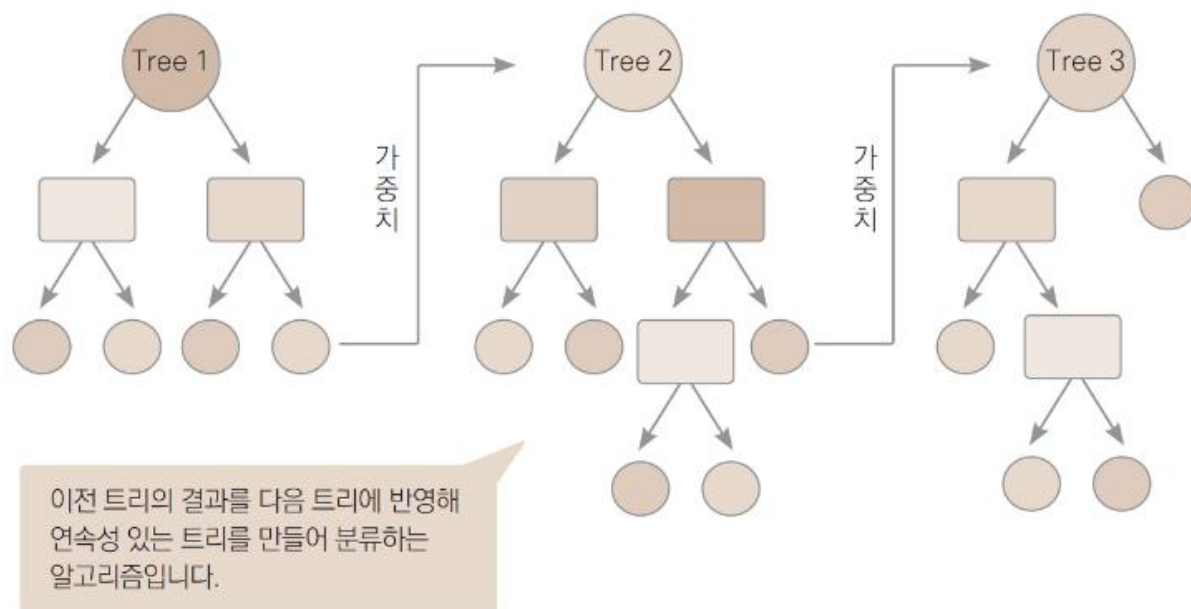
- 모델 해석의 어려움: 단일 결정 트리보다 해석하기 어려움. 수백 개의 트리로 구성되어 있기 때문.
- 훈련 시간: 큰 데이터셋에서는 많은 수의 트리를 생성하고 학습시키기 때문에 상대적으로 많은 계산 자원과 시간이 소요
- 메모리 사용: 많은 수의 트리를 저장해야 하기 때문에 상대적으로 많은 메모리를 사용

33 랜덤포레스트 주의점

- 트리의 수 설정: `n_estimators` 매개변수를 통해 트리의 수를 설정할 수 있는데, 트리 수가 많을수록 일반적으로 모델의 성능은 향상되지만, 계산 비용과 시간도 증가. 적절한 균형을 찾는 것이 중요.
- 과적합 방지: 랜덤 포레스트는 자체적으로 과적합에 강하지만, 매우 노이즈가 많은 데이터셋에서는 여전히 과적합이 발생할 수 있음. 적절한 매개변수 조정이 필요.
- 특성 선택: 모든 특성을 사용할 필요는 없으며, 불필요하거나 중복되는 특성은 제거하는 것이 좋음. 특히 특성의 수가 많을 경우, 랜덤 포레스트의 성능에 부정적인 영향을 미칠 수 있음

34 부스팅

- 여러 개의 모델이 순차적으로 학습을 수행 (랜덤 포레스트는 병렬적)
- 특이한 점은 앞에서 학습한 모델이 틀린 데이터에 더욱 가중치를 줘서 다음 모델이 이 부분을 더욱 집중하게 함
- 계속해서 모델에 가중치를 부스팅하면서 학습을 진행하기에 부스팅 방식으로 불림



- 성능: XGBoost는 높은 예측 성능을 제공하며, 많은 머신러닝 경진대회에서 우승 모델로 자주 사용 됨.
- 스케일러빌리티와 속도: XGBoost는 병렬 처리와 분산 컴퓨팅을 지원하여 대용량 데이터셋을 빠르게 처리할 수 있음
- 정규화: 내장된 정규화는 과적합을 방지하는 데 도움
- 유연성: 다양한 손실 함수를 지원하며, 사용자 정의 손실 함수도 추가할 수 있음
- 결측치 자동 처리: XGBoost는 내부적으로 결측치를 처리할 수 있는 기능을 가지고 있어 별도의 전처리가 필요하지 않음

34 부스팅 단점

- 복잡성: 매개변수가 많아 조정이 까다롭고, 모델의 튜닝에 시간이 많이 소요될 수 있음
- 과적합: 매개변수가 잘못 설정되면 쉽게 과적합될 수 있음
- 해석의 어려움: 부스팅 트리 모델은 해석하기 어렵고, 결정 트리나 선형 모델처럼 직관적이지 않음

34 부스팅 주의점

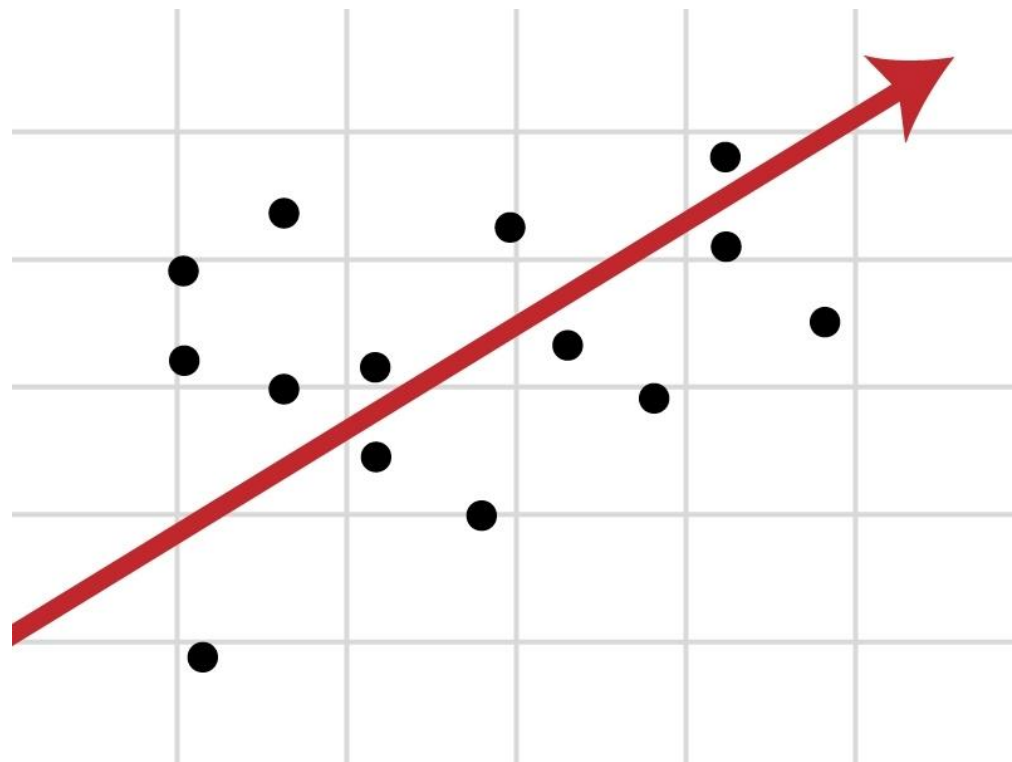
- 매개변수 조정: learning_rate, max_depth, n_estimators, subsample, colsample_bytree 등의 매개변수를 적절히 조정해야 함. 이들 각각은 학습 속도와 모델의 복잡성에 영향을 미치며, 잘못 설정하면 과적합이나 학습 부족이 발생할 수 있음
- 교차 검증 사용: 과적합을 피하기 위해 교차 검증을 사용하고, 다양한 데이터 분할에 대해 모델의 성능을 평가해야 함
- 성능 평가: 각 부스팅 스테이지 후 모델의 성능을 평가하고, 필요에 따라 조기 종료(early stopping)를 사용하여 더 이상의 성능향상이 없을 때 학습을 중단시키는 것이 좋음.

10 회귀모델

학습목표 : 머신러닝 회귀모델에 대해서 학습한다

35 회귀분석

- 회귀분석(regression analysis)은 하나의 변수와 다른 여러 변수간의 인과관계를 밝히기 위한 통계적 기법
- 아마도 무언가를 예측할 때 가장 많이 사용되는 모델일 것이다.
- 특히나 이렇게 인과관계를 대놓고 탐구하는 모델은 드물다!
- 잘 익히자 😊



35 회귀분석

- 선형 회귀분석은 두 (혹은 더 많은) 변수 간의 인과관계를 선형적으로 근사한 모델이다

$$Y = F(x_1, x_2, \dots, x_n)$$

Dependent

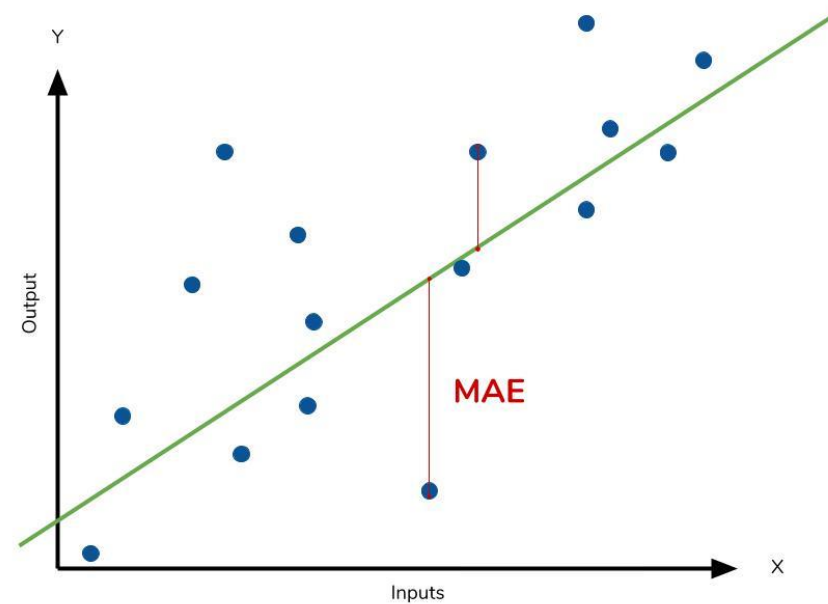
Independent

- 종속변수 Y 는 독립변수 x_n 의 함수이다.
- 선형회귀는 이 함수가 선형함수!

35 회귀분석

- 회귀 평가 지표
- MAE (Mean Absolute Error) 평균절대오차

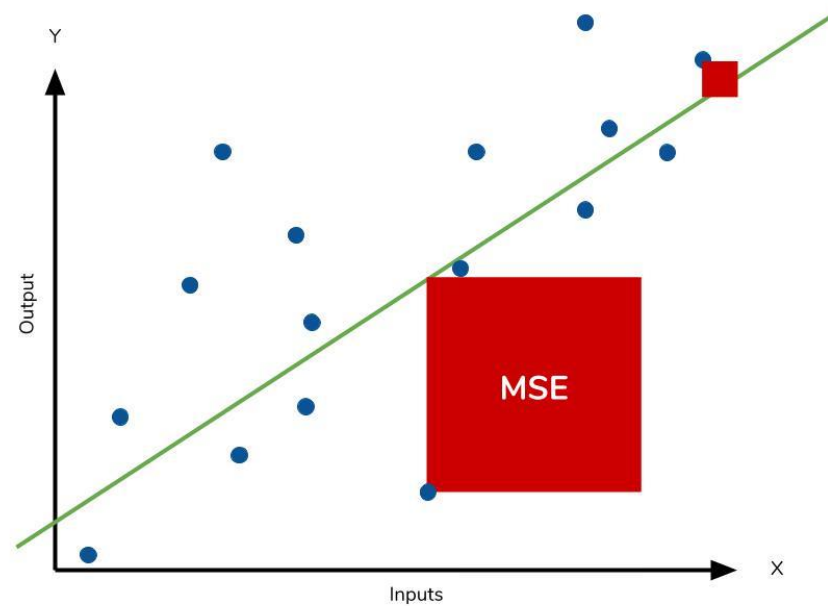
$$MAE = \frac{\sum |y - \hat{y}|}{n}$$



35 회귀분석

- 회귀 평가 지표
- MSE (Mean Squared Error) 평균제곱오차

$$MSE = \frac{\sum (y - \hat{y})^2}{n}$$



35 회귀분석

- 회귀 평가 지표
- RMSE (Root Mean Squared Error) 평균오차

$$RMSE = \sqrt{\frac{\sum (y - \hat{y})^2}{n}}$$

35 회귀분석

- 회귀 평가 지표
- 보통 RMSE나 adjusted R-square를 이용
- 사이킷런에는 MAE, MSE, R2이 있다
- RMSE는 MSE가지고 직접 계산해야 함

36 회귀분석의 규제

- 지금까지 회귀분석을 살펴보면 주어진 데이터를 SSR를 최대한 작게 fitting하는 모델을 찾는 것에 목적을 두었다.
- 즉, 실제 값과 예측 값의 차이를 최소화하는 것만 고려
- 하지만 이렇게 되면 독립변수가 많으면 무조건 좋아지거나, 주어진 데이터에만 맞추기 위하여 회귀계수가 무분별하게 커지는 문제가 발생
- 즉, 오버피팅의 문제가 발생!

36 회귀분석의 규제

- 이러한 문제를 해결하기 위해...
- 기존의 규제 => RSS
- 규제항을 추가 => $RSS + a * f(B)$
- 여기서 B는 회귀계수, f는 회귀계수 벡터의 길이를 구하는 함수 (절대값을 쓰거나 제곱을 하거나)

36 회귀분석의 규제

- 규제항 : $RSS + a * f(B)$
- 오차항 + 회귀계수 페널티
- 즉, 이제는 단순히 오차를 줄이는 것뿐만 아니라 회귀계수의 크기를 줄이거나 개수를 줄여야 함
- 여기서 a 는 위의 규제항에서 회귀계수 페널티에 어느정도 영향력을 줄 것인지 조정하는 매개변수

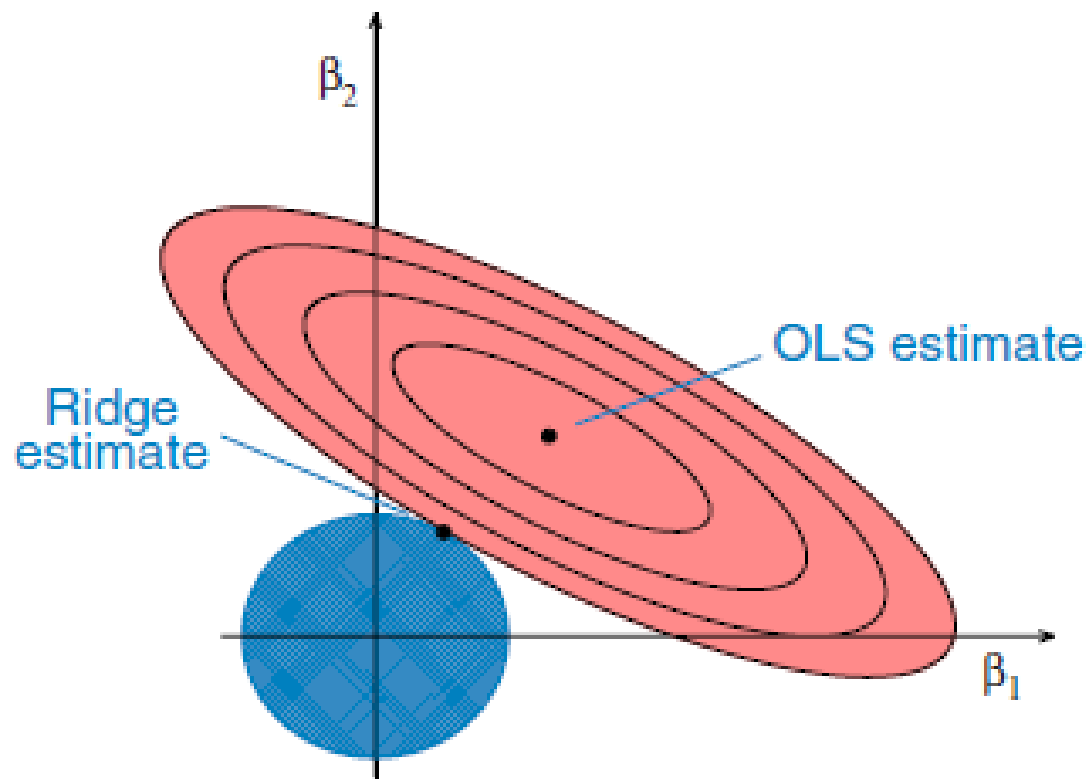
36 회귀분석의 규제

- 규제항 : $RSS + a * f(B)$
- 오차항 + 회귀계수 페널티
- a 가 0이면 => 일반 회귀분석과 같아짐 (회귀계수에 대한 페널티가 없음)
- a 가 매우 크면 => 회귀계수에 대한 페널티가 커져 거의 B 를 0에 가깝게 학습
- (예측값이 target값의 평균 정도밖에 안 됨)
- 적절한 a 를 선택 => 언더피팅과 오버피팅 조절

36 회귀분석의 규제

- 회귀계수에 대한 페널티를 회귀계수 벡터를 제공하는 방식으로 구함
- 아래 규제를 최소화하는 회귀 모델을 Ridge Regression이라 한다

$$\sum_{i=1}^n (y_i - \sum_{j=1}^p x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2$$



36 회귀분석의 규제

- 회귀계수에 대한 페널티를 회귀계수 벡터에 절대값을 취하는 방식으로 구함
- 아래 규제를 최소화하는 회귀 모델을 Lasso Regression이라 한다

$$\text{Minimize: } \sum_{i=1}^n (Y_i - \sum_{j=1}^p X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

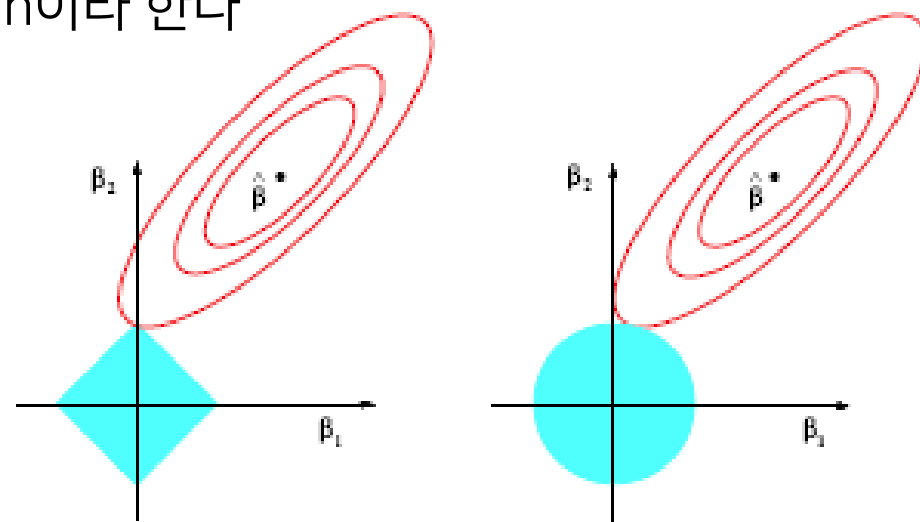


Figure 3.12: Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the least squares error function.

36 회귀분석의 규제

- L1 규제와 L2 규제를 조합하여 규제함 (Lasso와 Ridge의 하이브리드 모델)
- 아래 규제를 최소화하는 회귀 모델을 Elastic-Net 이라 한다

$$\text{minimize} \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda_1 \sum_{j=1}^p \beta_j^2 + \lambda_2 \sum_{j=1}^p |\beta_j|$$

36 회귀분석의 규제

- L1 규제와 L2 규제를 조합하여 규제함 (Lasso와 Ridge의 하이브리드 모델)
- 아무리 ML 커뮤니티가 섞는 것을 좋아한다지만 이렇게 섞는다고 좋은가?!
- Elastic Net은 확실히 좋아진다!
- 가장 큰 장점은 다중공선성 문제에 강력!

36 회귀분석의 규제

- Elastic-Net 은 단순한 두 모델을 섞은 모델이 아니라 이렇게 두 규제를 동시에 사용함으로써 변수간 grouping effect라는 엄청난 이점을 제공하게 된다
- 변수간 상관관계가 큰 변수들을 하나로 그룹핑하여, 중요도가 높은 그룹은 회귀계수를 높게 주고 그렇지 않은 그룹은 회귀계수를 0으로 수렴 시킨다
- Lasso는 변수간 상관관계가 크다고 하면 그 중 하나를 랜덤하게 선택하고 나머지 변수의 회귀계수를 0으로 수렴시킴 => 정보 소멸
- 변수간 상관관계가 크더라도 중요하지 않은 변수들을 모두 버리면서 중요한 변수들만 잘 골라내어 중요도와 상관관계에 따라 적합한 가중치를 적용할 수 있다.

36 회귀분석의 규제

- **릿지 회귀 (Ridge Regression)**

- 언제 사용? 데이터의 모든 변수가 예측에 중요하다고 생각될 때 사용
- 릿지 회귀는 변수 선택을 하지 않고, 모든 변수의 계수를 축소하여 모델의 복잡성을 줄임
- 주로 변수 간의 다중공선성이 있을 때 유용하며, 변수들이 서로 관련이 있고 중요한 경우에 적합
- 기준: 변수가 많지만 제거하고 싶지 않고, 변수 간의 상관관계가 높을 때 선택

- **라쏘 회귀 (Lasso Regression)**

- 언제 사용? 불필요하거나 중요하지 않은 변수를 데이터에서 제거하고 싶을 때 사용
- 라쏘 회귀는 일부 계수를 정확하게 0으로 만들어 해당 특성을 모델에서 제외
- 이는 모델의 해석을 용이하게 하며, 변수 선택의 효과를 가짐
- 기준: 변수가 많고, 그 중 일부만이 중요하다고 예상될 때 또는 변수 선택을 통해 모델의 복잡성을 줄이고자 할 때 유용

36 회귀분석의 규제

- 릿지 회귀 (Ridge Regression)
 - 언제 사용? 데이터의 모든 변수가 예측에 중요하다고 생각될 때 사용합니다. 릿지 회귀는 변수 선택을 하지 않고, 모든 변수의 계수를 축소하여 모델의 복잡성을 줄입니다. 주로 변수 간의 다중공선성이 있을 때 유용하며, 변수들이 서로 관련이 있고 중요한 경우에 적합합니다.
 - 기준: 변수가 많지만 제거하고 싶지 않고, 변수 간의 상관관계가 높을 때 선택합니다.
- 라쏘 회귀 (Lasso Regression)
 - 언제 사용? 불필요하거나 중요하지 않은 변수를 데이터에서 제거하고 싶을 때 사용합니다. 라쏘 회귀는 일부 계수를 정확하게 0으로 만들어 해당 특성을 모델에서 제외합니다. 이는 모델의 해석을 용이하게 하며, 변수 선택의 효과를 가집니다.
 - 기준: 변수가 많고, 그 중 일부만이 중요하다고 예상될 때 또는 변수 선택을 통해 모델의 복잡성을 줄이고자 할 때 유용합니다.
- 엘라스틱넷 회귀 (ElasticNet Regression)
 - 언제 사용? 릿지와 라쏘의 장점을 결합하고 싶을 때 사용합니다. 데이터에 많은 변수가 있으며, 이 중 일부는 완전히 제거하고 일부는 계수를 축소시키고 싶을 때 적합합니다. 엘라스틱넷은 릿지와 라쏘 회귀의 중간 형태로, 두 정규화 기법의 장점을 동시에 활용할 수 있습니다.
 - 기준: 변수들이 많고, 다중공선성 문제가 있으며, 변수 선택과 계수 축소가 동시에 필요할 때 선택합니다.
- 모델 선택 기준
 - 변수의 수와 중요성: 변수가 많고 일부만 중요하다면 라쏘나 엘라스틱넷을 고려해보세요.
 - 다중공선성의 유무: 변수들 간에 강한 상관관계가 있으면 릿지 또는 엘라스틱넷이 더 적합할 수 있습니다.
 - 모델의 해석 필요성: 모델 해석을 중요시하고 변수의 영향을 명확히 파악하고 싶다면 라쏘가 좋을 수 있습니다.
 - 성능과 정확성: 교차 검증을 통해 각 모델의 성능을 평가하여 가장 적합한 모델을 선택합니다.

36 회귀분석의 규제

- **엘라스틱넷 회귀 (ElasticNet Regression)**

- 언제 사용? 릿지와 라쏘의 장점을 결합하고 싶을 때 사용
- 데이터에 많은 변수가 있으며, 이 중 일부는 완전히 제거하고 일부는 계수를 축소시키고 싶을 때 적합
- 엘라스틱넷은 릿지와 라쏘 회귀의 중간 형태로, 두 정규화 기법의 장점을 동시에 활용할 수 있음
- 기준: 변수들이 많고, 다중공선성 문제가 있으며, 변수 선택과 계수 축소가 동시에 필요할 때 선택

- **모델 선택 기준**

- 변수의 수와 중요성: 변수가 많고 일부만 중요하다면 라쏘나 엘라스틱넷을 고려
- 다중공선성의 유무: 변수들 간에 강한 상관관계가 있으면 릿지 또는 엘라스틱넷이 더 적합할 수 있음
- 모델의 해석 필요성: 모델 해석을 중요시하고 변수의 영향을 명확히 파악하고 싶다면 라쏘가 좋을 수 있음
- 성능과 정확성: 교차 검증을 통해 각 모델의 성능을 평가하여 가장 적합한 모델을 선택

36 회귀분석의 규제

- **엘라스틱넷 회귀 (ElasticNet Regression)**
 - 언제 사용? 릿지와 라쏘의 장점을 결합하고 싶을 때 사용
 - 데이터에 많은 변수가 있으며, 이 중 일부는 완전히 제거하고 일부는 계수를 축소시키고 싶을 때 적합
 - 엘라스틱넷은 릿지와 라쏘 회귀의 중간 형태로, 두 정규화 기법의 장점을 동시에 활용할 수 있음
 - 기준: 변수들이 많고, 다중공선성 문제가 있으며, 변수 선택과 계수 축소가 동시에 필요할 때 선택
- **모델 선택 기준**
 - 변수의 수와 중요성: 변수가 많고 일부만 중요하다면 라쏘나 엘라스틱넷을 고려
 - 다중공선성의 유무: 변수들 간에 강한 상관관계가 있으면 릿지 또는 엘라스틱넷이 더 적합할 수 있음
 - 모델의 해석 필요성: 모델 해석을 중요시하고 변수의 영향을 명확히 파악하고 싶다면 라쏘가 좋을 수 있음
 - 성능과 정확성: 교차 검증을 통해 각 모델의 성능을 평가하여 가장 적합한 모델을 선택

END

