

LG계약학과 제어시스템전공 2025년 동계 집중교육 특강

# AI 기반 SW 설계를 위한 이론 및 실무 I

안상태 교수

2026년 2월 10일 (화)

# 1. 머신러닝 기초

# 1.1 인간을 닮은 기계를 만들기 위한 노력

- 머신러닝: 1959년에 아서 새뮤얼이 체커스 프로그램을 이용한 실험 결과를 발표하면서 처음 사용



## 1.1 인간을 닮은 기계를 만들기 위한 노력

- 1950년대에 IBM 의 아서 새뮤얼 Arthur Samuel 은 최초의 상업용 컴퓨터 IBM 701 로 체커스 checkers 라는 소프트웨어를 개발
  - 인간의 전유물이었던 규칙에 기반한 놀이를 컴퓨터도 할 수 있음을 보임

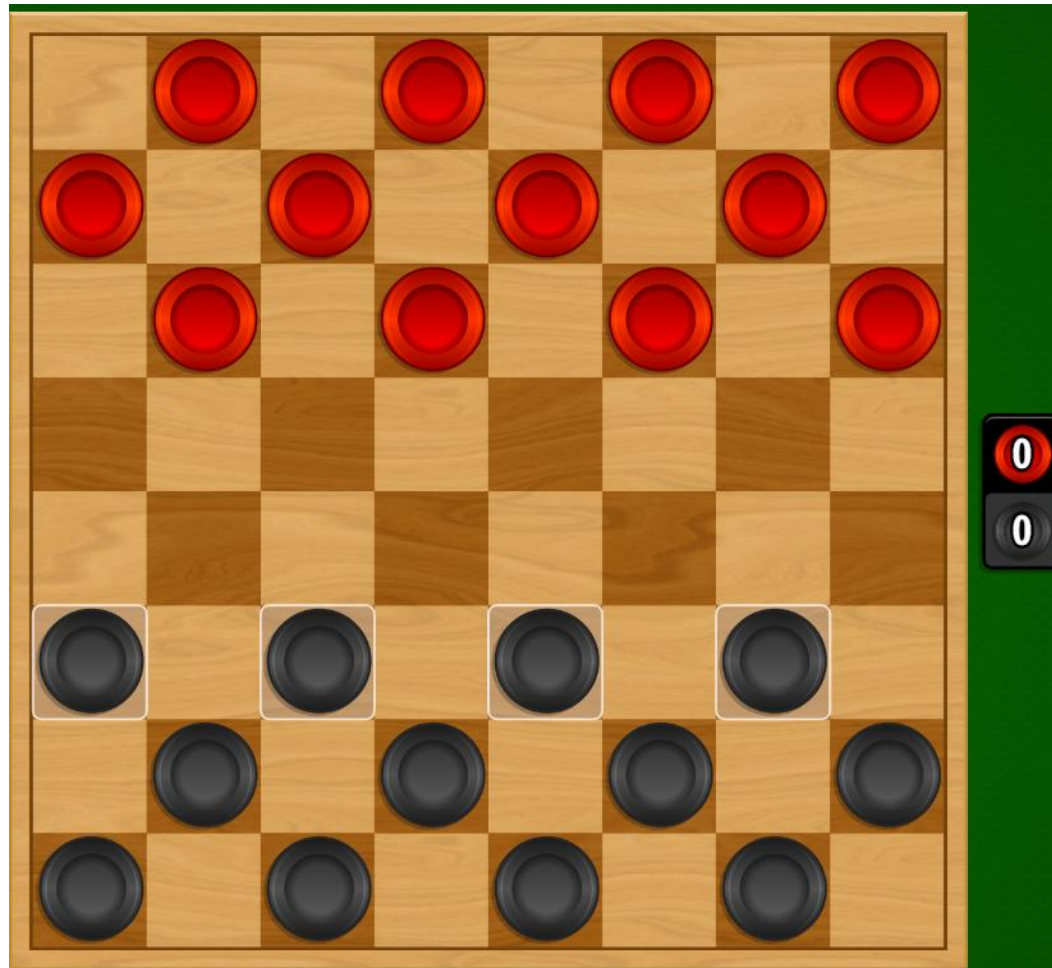


## 1.1 인간을 닮은 기계를 만들기 위한 노력

- <https://www.247checkers.com/>



## 1.1 인간을 닮은 기계를 만들기 위한 노력



## 1.2 생각하는 기계

- 튜링이 “생각하는 기계”를 정의한 이후에도 인공지능에 대한 합의된 정의는 아직 없다. 하지만 한 가지 공통적으로 요구되는 것이 있다.

→ 기계가 문제를 해결하는 방법을 스스로 찾는 것.

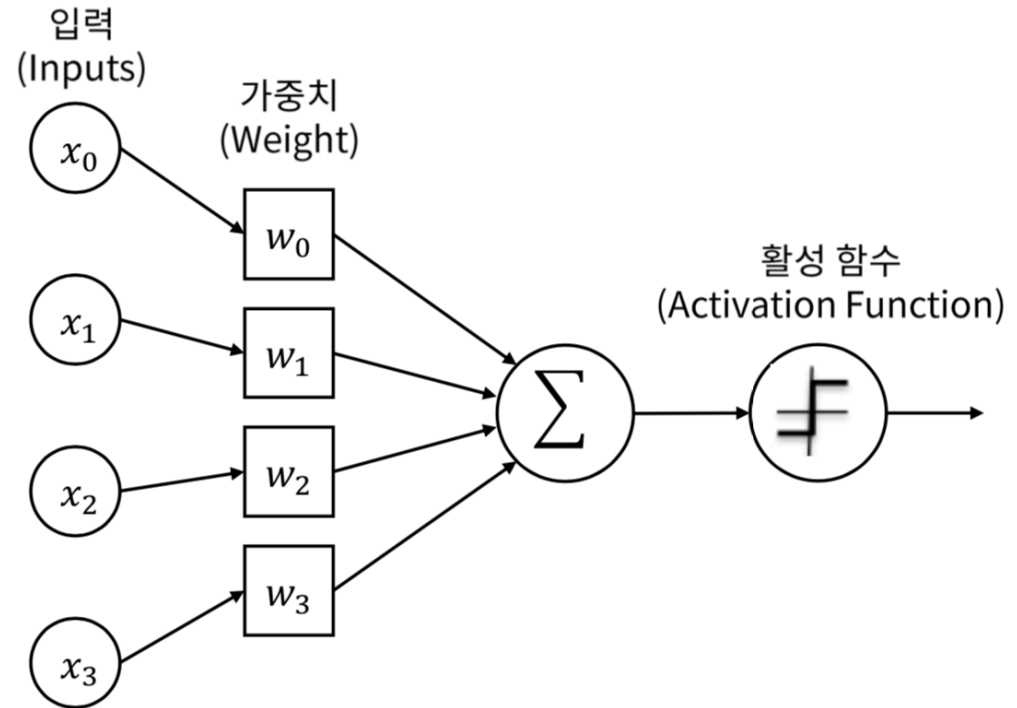
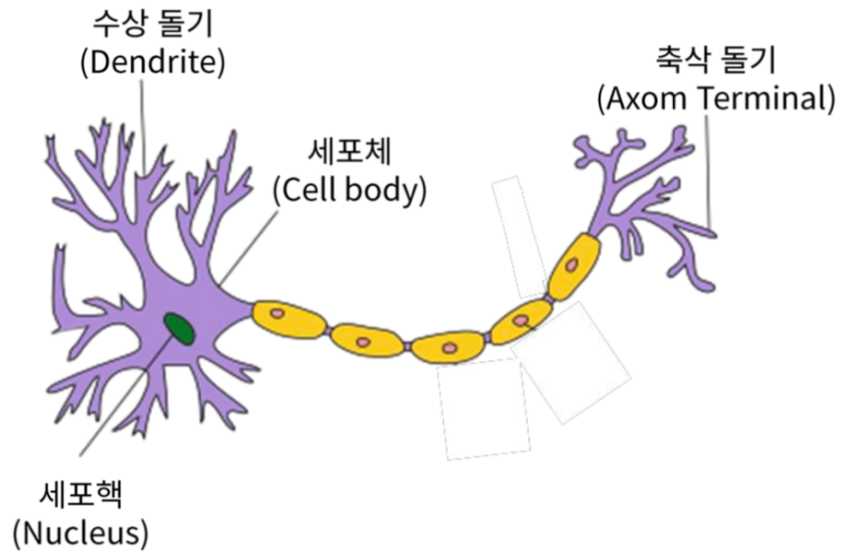
- 컴퓨터 프로그램이 많은 데이터를 경험하고 이 경험을 바탕으로 목표를 달성할 수 있도록 하는 알고리즘이 인공지능의 핵심

## 1.2 생각하는 시작하는 기계

- 인공지능 연구의 첫 번째 봄은 1960년대
- 1962년 이전 인공지능 연구에서는 여러 가지 시행착오를 탐색해 휴리스틱<sup>heuristics</sup>을 찾는 것을 가장 중요하게 여김

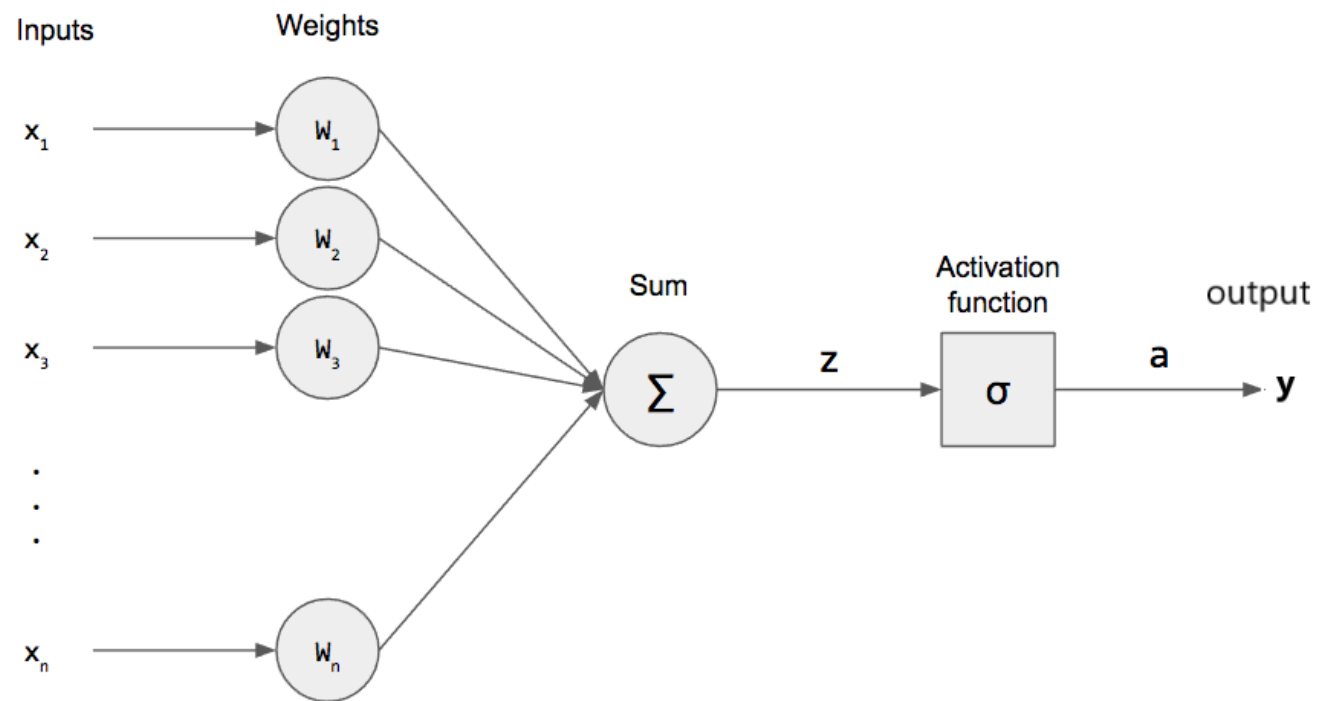
## 1.2 생각하는 시작하는 기계

- 1958년 Frank Rosenblatt : Perceptron (퍼셉트론)



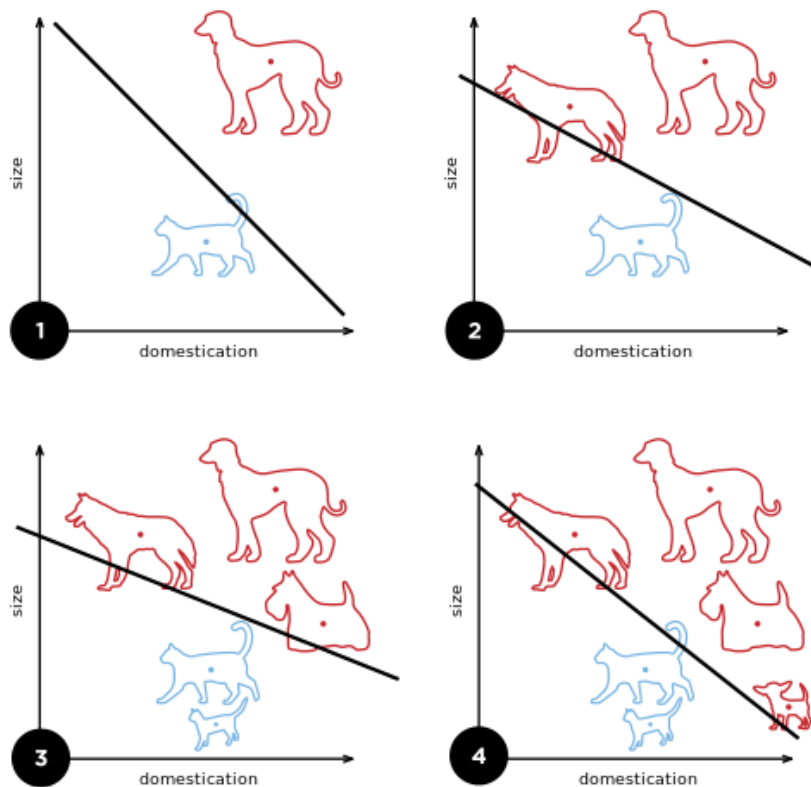
## 1.2 생각하는 시작하는 기계

- 단층 퍼셉트론



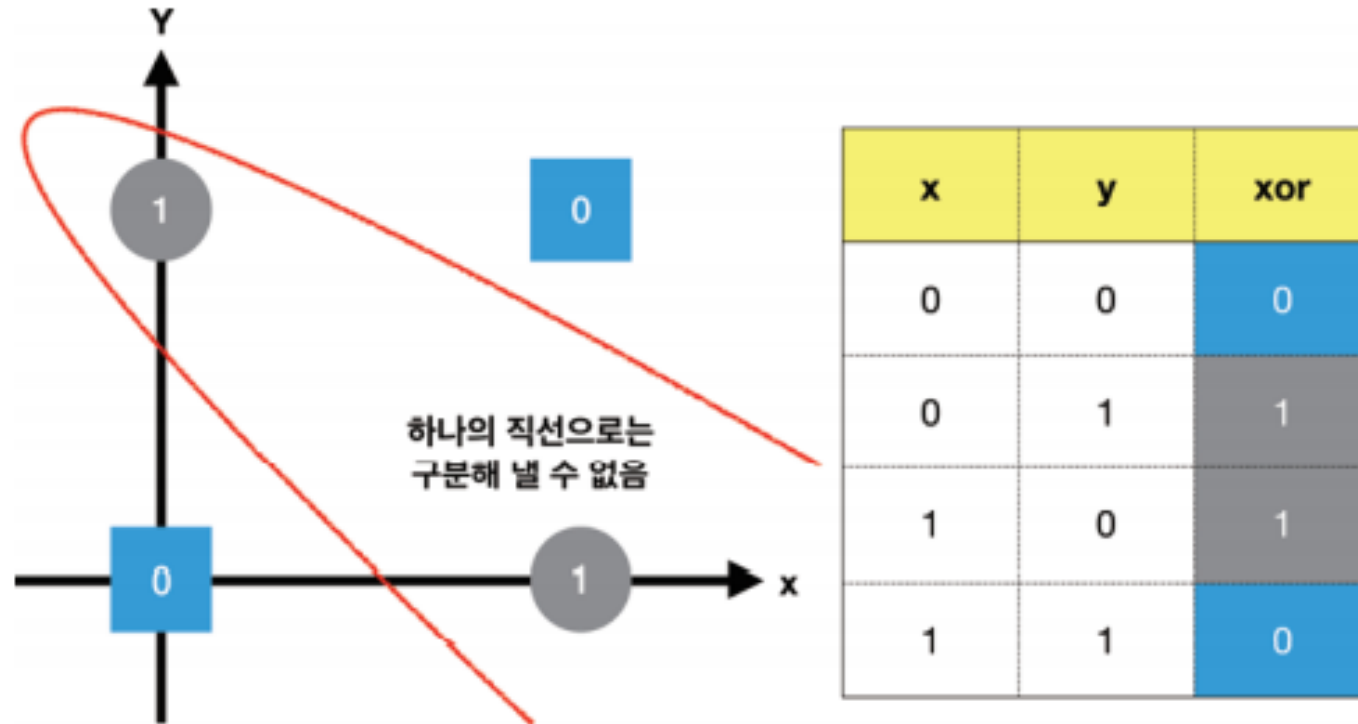
## 1.2 생각하는 시작하는 기계

- 분류 문제 : 데이터 추가에 따른 새로운 결정경계 (Decision Boundary) 생성



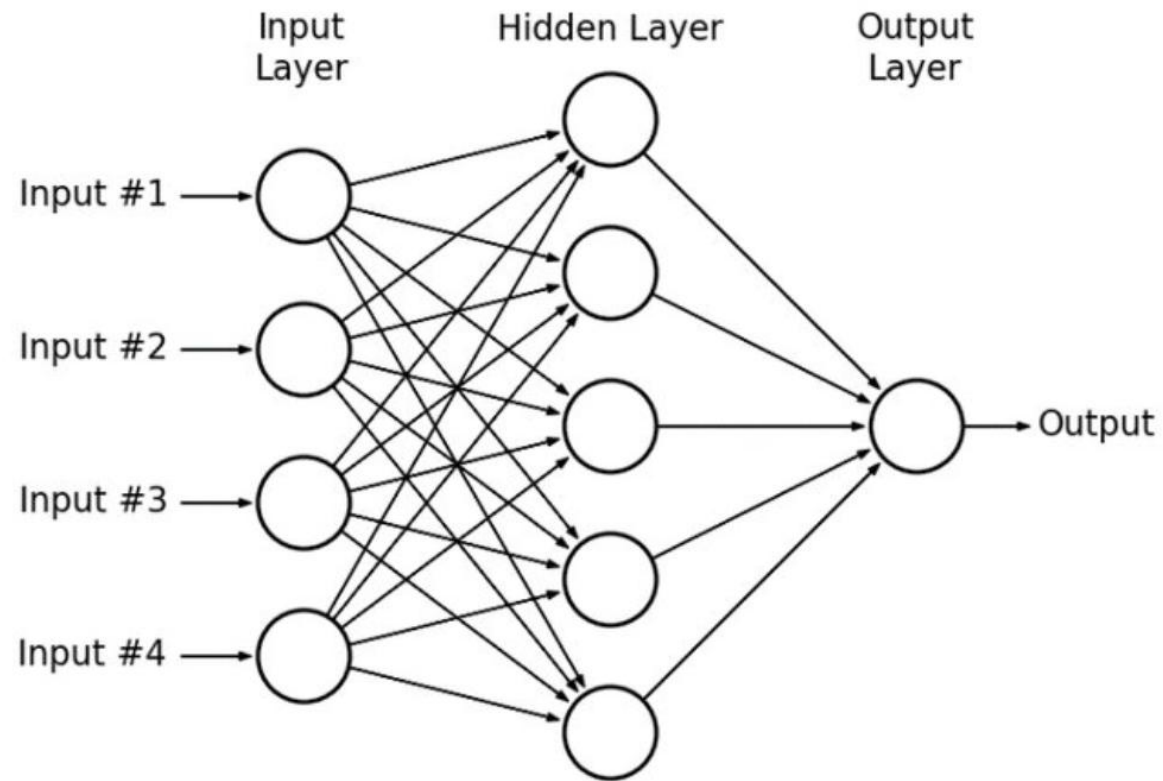
## 1.3 머신러닝의 겨울과 봄

- 1969년 Marvin Minsky : XOR 문제 제기



## 1.3 머신러닝의 겨울과 봄

- 다층퍼셉트론이 XOR문제를 해결할 수 있으나 학습 방법 부재

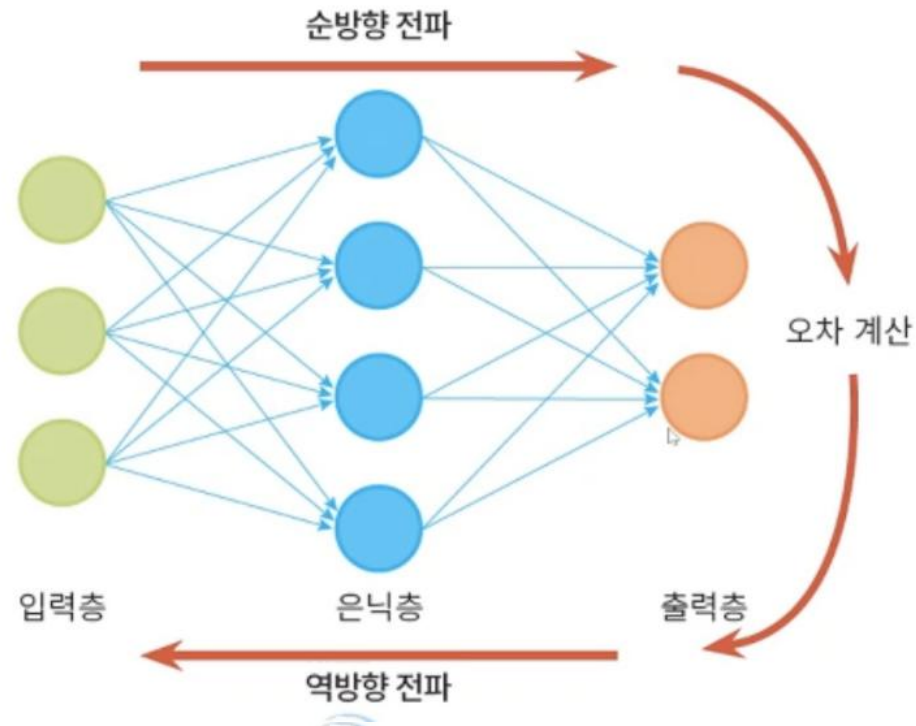


## 1.3 머신러닝의 겨울과 봄

- 1970년대 이후
- **기호주의**symbolism : 문제와 논리, 탐색을 사람이 이해할 수 있는 기호로 표현하여 답을 찾으려고 하는 인공지능의 흐름  
ex) 규칙 기반 인공지능
- **연결주의**connectionism : 퍼셉트론처럼 연결된 요소가 학습을 통해 해법을 스스로 찾도록 만들려는 방식을 이라고 부른다.  
ex) 데이터 기반 인공지능

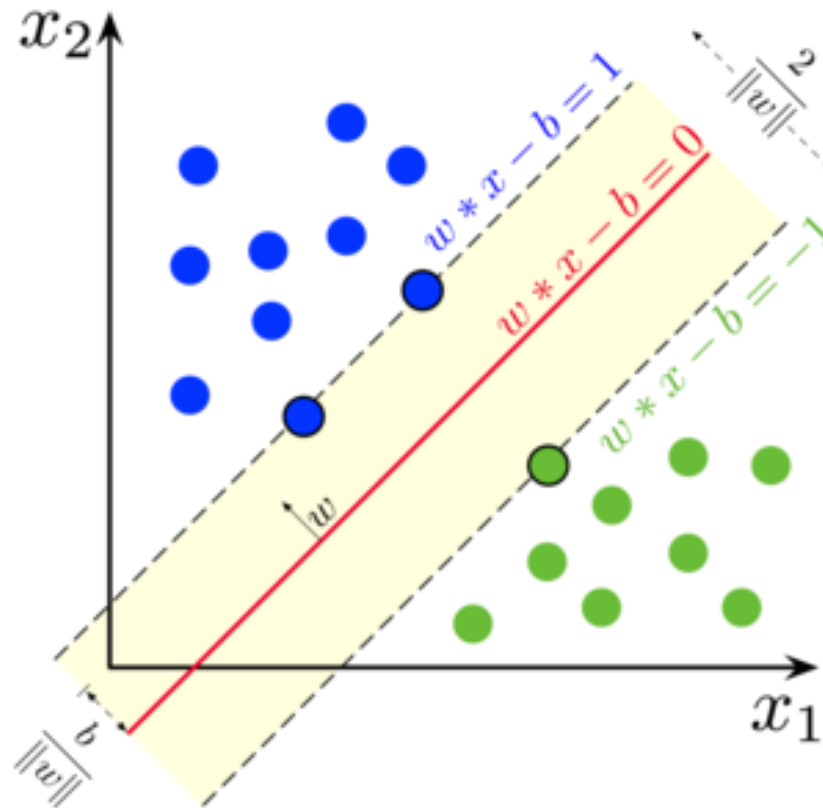
## 1.3 머신러닝의 겨울과 봄

- 1970년대 연결주의 쇠퇴기
- 1986년 Geoffrey Hinton  
다층퍼셉트론을 학습할 수 있는 알고리즘 개발 : 오류역전파



## 1.3 머신러닝의 겨울과 봄

- 1995년 Vladimir Vapnik : Support Vector Machine (SVM) 알고리즘 개발



## 1.3 머신러닝의 겨울과 봄

- 1997년: IBM 의 Deep Blue 가 체스 세계 챔피언을 이김
- 2000년대 이후 : 방대한 데이터와 하드웨어의 발전, 다양한 알고리즘 개발
- 2012년 : ImageNet에서 딥러닝 기반 AlexNet 우승



## 1.3 머신러닝의 겨울과 봄

- 2010년 이후 Python 기반 딥러닝 프레임워크 개발 및 배포



TensorFlow

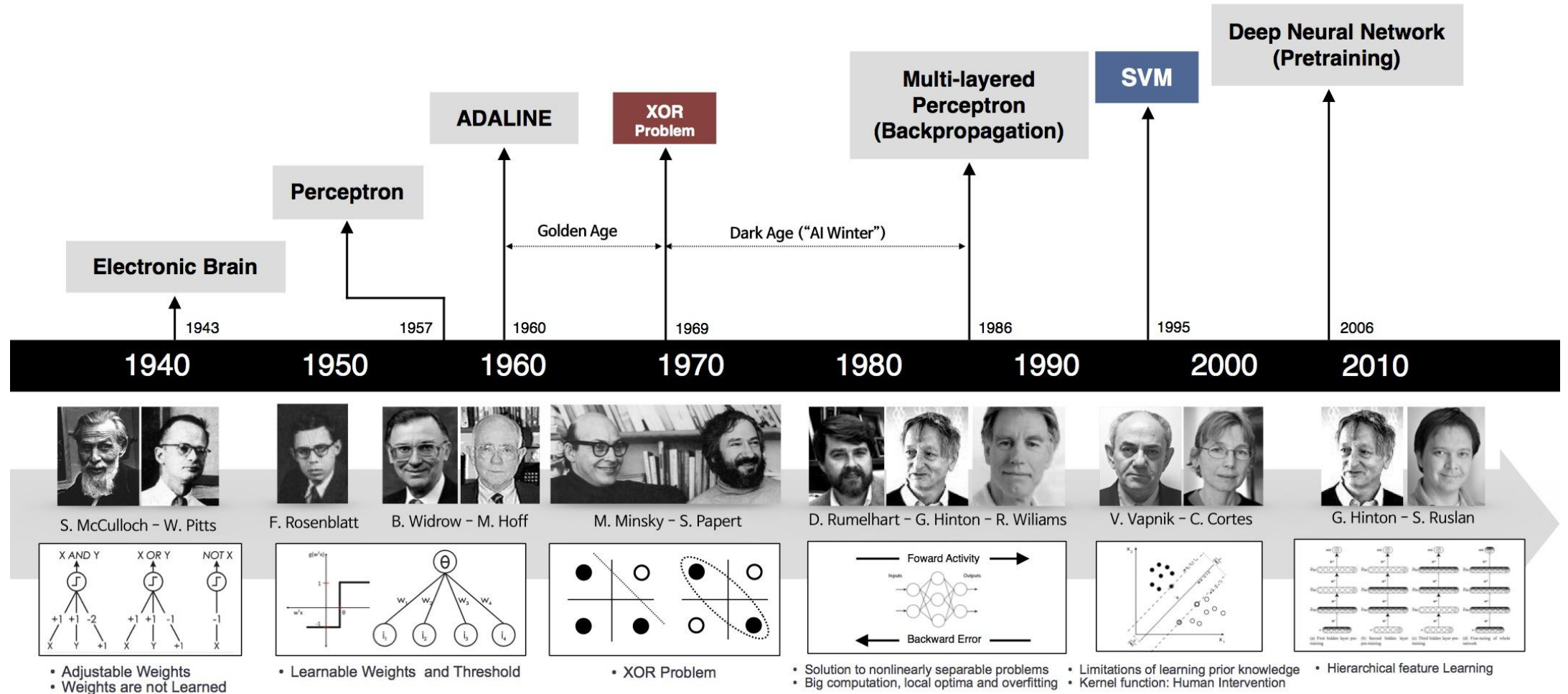


Keras



PyTorch

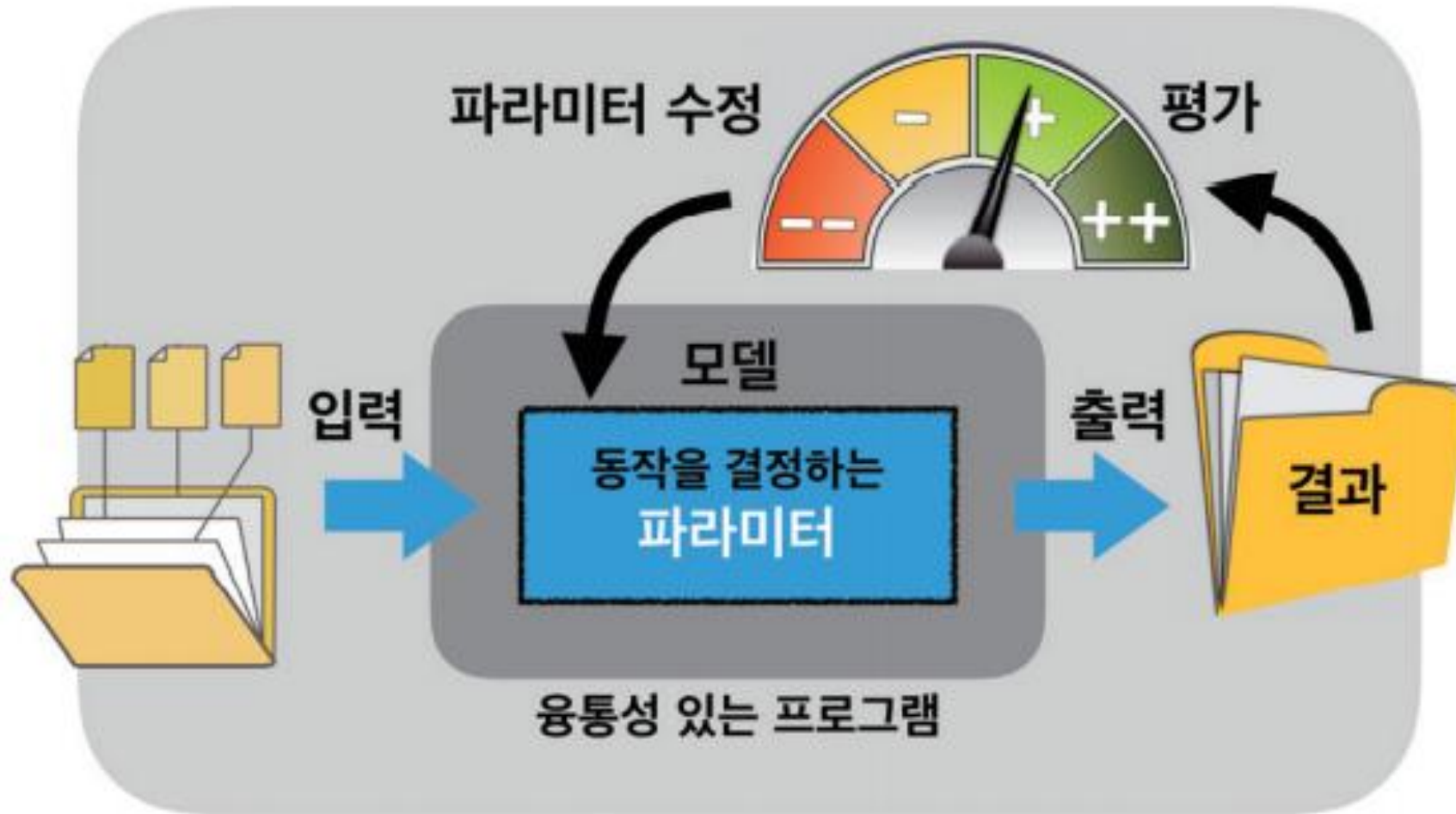
# 1.3 머신러닝의 겨울과 봄



## 1.4 머신러닝은 무엇을 하려는 것인가?

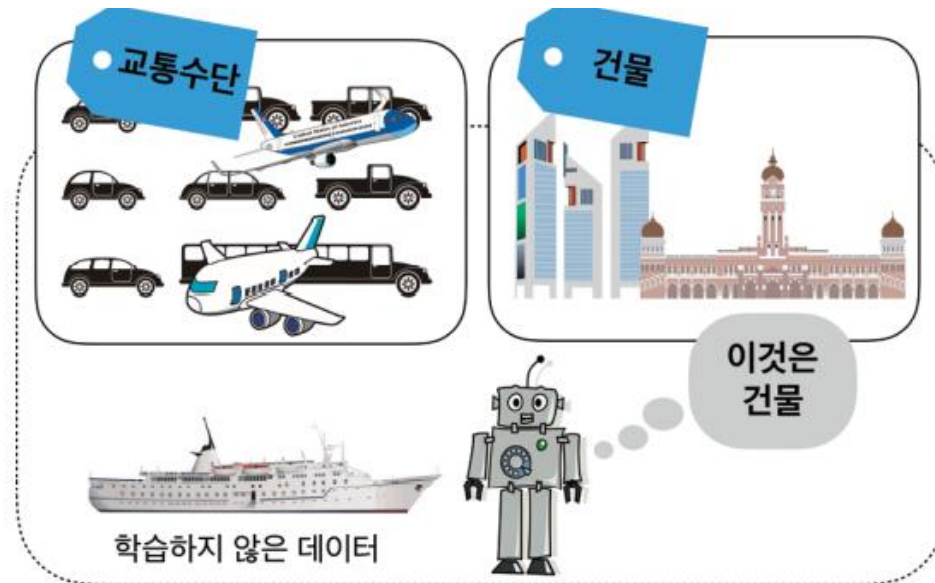
- **모델**<sup>model</sup> : 변경 가능한 **파라미터**<sup>parameter</sup>에 의해 동작이 결정되는 **프로그램**
  - 파라미터가 바뀌면 동작도 바뀜
  - 데이터를 다양하게 제공하여 프로그램이 이 데이터를 얼마나 잘 처리하는지 확인
- **학습**<sup>learning</sup> : 좋은 동작이 나오도록 파라미터를 변경하는 일

## 1.4 머신러닝은 무엇을 하려는 것인가?



## 1.5 머신러닝, 무엇이 문제일까?

- 머신러닝 : 파라미터에 따라 동작하는 알고리즘algorithm을 선택하고, 이 알고리즘에 데이터를 제공하여 알고리즘이 더 나은 동작을 하도록 파라미터를 수정하는 것
- 머신러닝의 핵심적인 문제는 알고리즘과 데이터

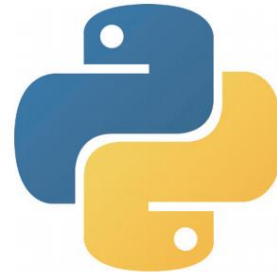


## 1.5 머신러닝, 무엇이 문제일까?

- **데이터 편향**<sup>data bias</sup> - 확보된 데이터가 대표하는 모집단의 분포를 제대로 반영하지 못하고 일부의 특성만을 가지고 있는 경우
  - 편향의 원인은 두 가지
    - 너무 적은 수의 표본을 추출한 경우
    - 표집 방법이 잘못되어 모집단에 속한 대상을 골고루 추출하지 못 하는 경우.
- **부정확성**<sup>inaccuracy</sup> - 데이터의 품질이 낮아 많은 오류와 이상치, 잡음을 포함하고 있는 경우
- **무관함**<sup>irrelavance</sup> - 데이터는 많이 확보했지만, 이 데이터가 담고 있는 특성들이 학습하려고 하는 문제와는 무관한 데이터

## 1.6 Python

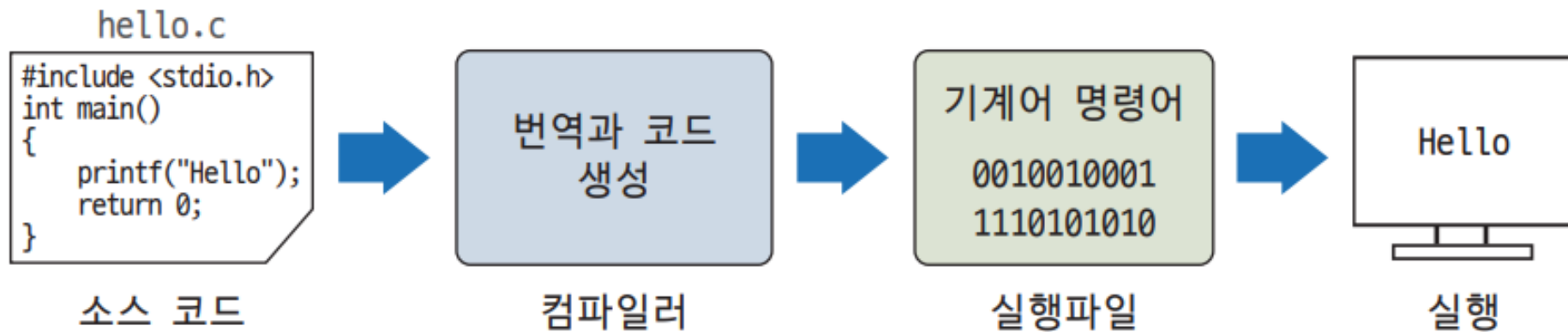
- 파이썬 Python은 귀도 반 로섬 Guido van Rossum이 1991년에 개발한 대화형 프로그래밍 언어



- 파이썬의 특징 중에 하나는 정수, 부동소수점, 문자와 같은 전통적인 자료형 data type 뿐만 아니라, 데이터 묶음을 처리하기에 편리한 리스트, 튜플, 딕셔너리, 집합과 같은 자료형을 기본으로 제공
- 파이썬은 '객체지향 프로그래밍 언어'이며, 파이썬이 다루는 모든 자료형, 함수, 모듈은 객체

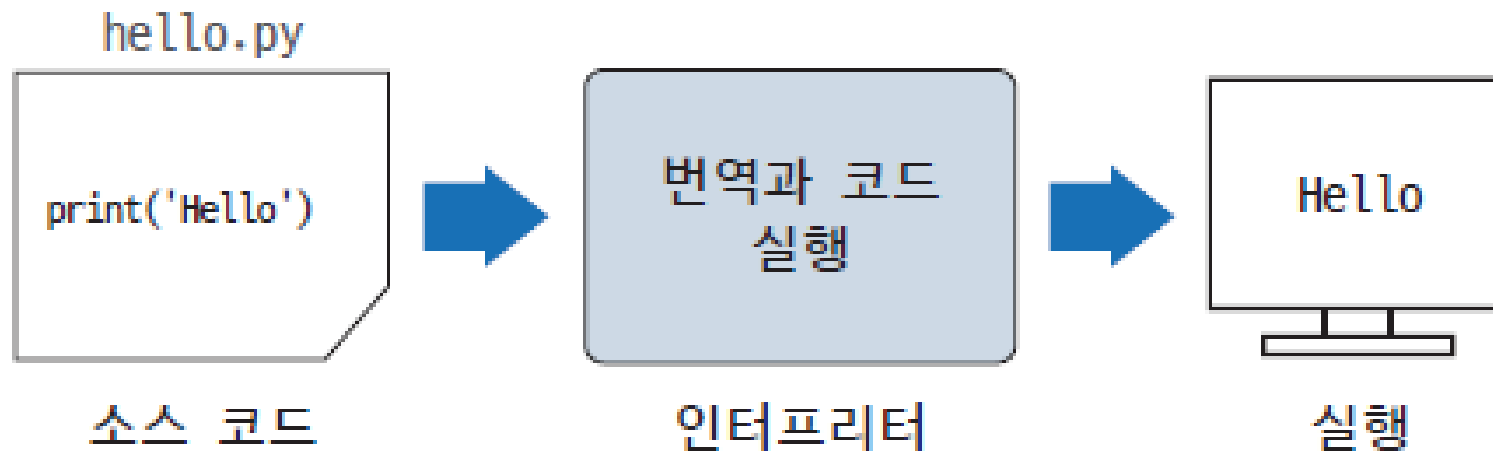
## 1.6 Python

- 컴파일 방식
  - 프로그램 명령어를 기계어로 번역한 후 이 기계어를 실행하는 방식
  - C, C++, 파스칼 언어등이 있음



## 1.6 Python

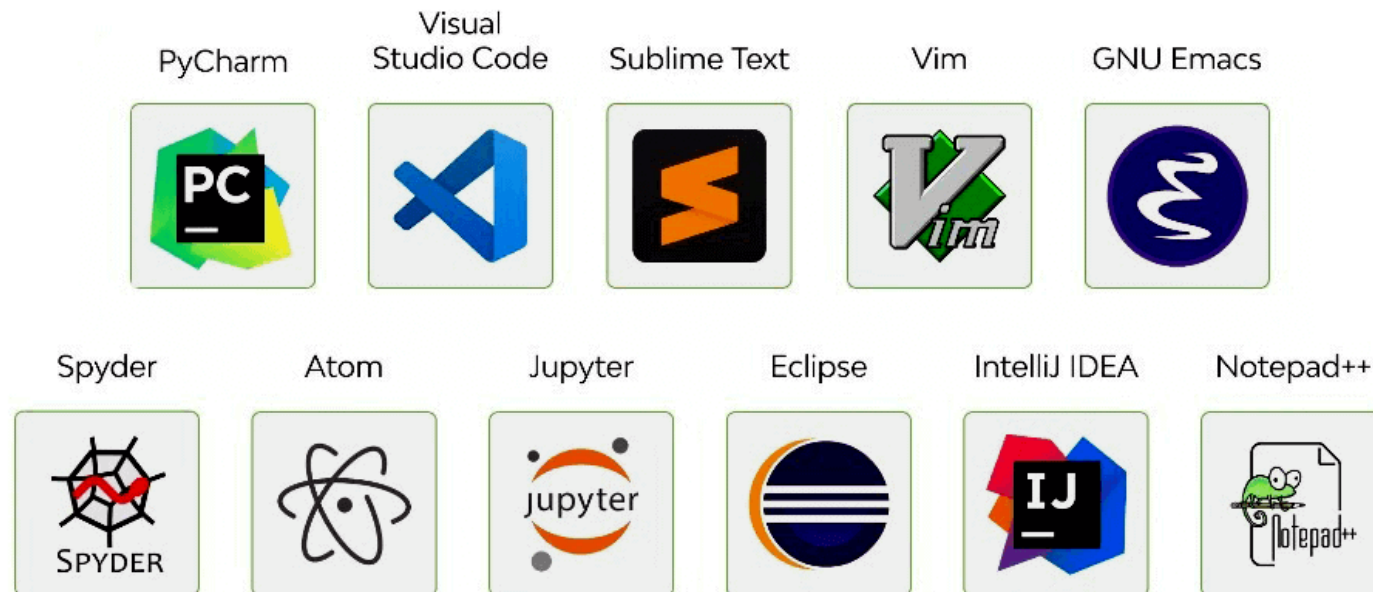
- 인터프리터 방식
  - 프로그램 명령어를 한 번에 한 줄씩 읽어 번역한 후 바로 실행
  - Python, JavaScript, Ruby, Matlab, BASIC 등의 언어가 있음



## 1.7 환경설정

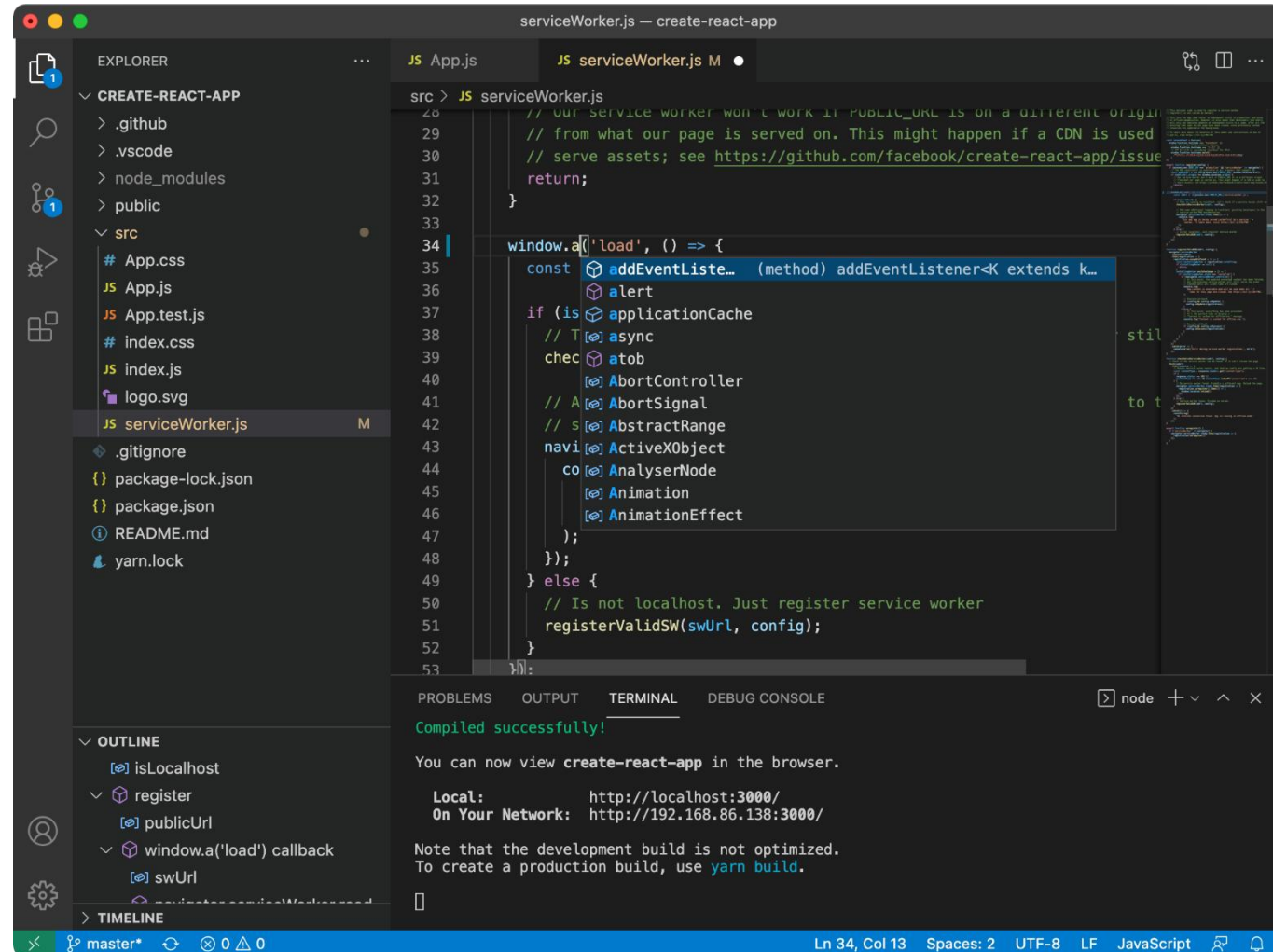
- IDE (Integrated Development Environment)

**통합개발환경:** 효율적으로 소프트웨어를 개발하기 위한 통합개발환경  
소프트웨어 어플리케이션 인터페이스



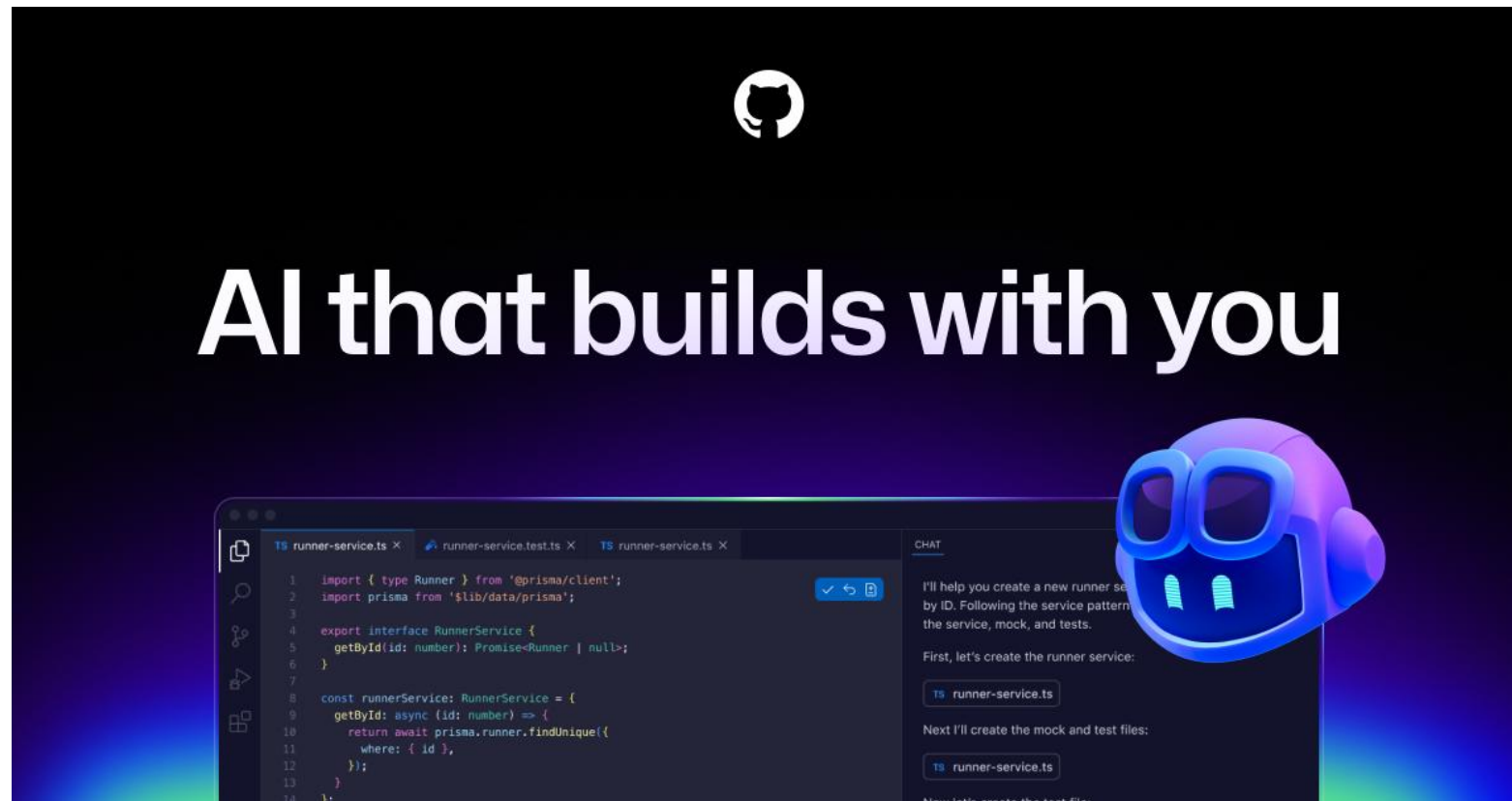
## 1.7 환경설정

- Visual Studio Code (VS Code)



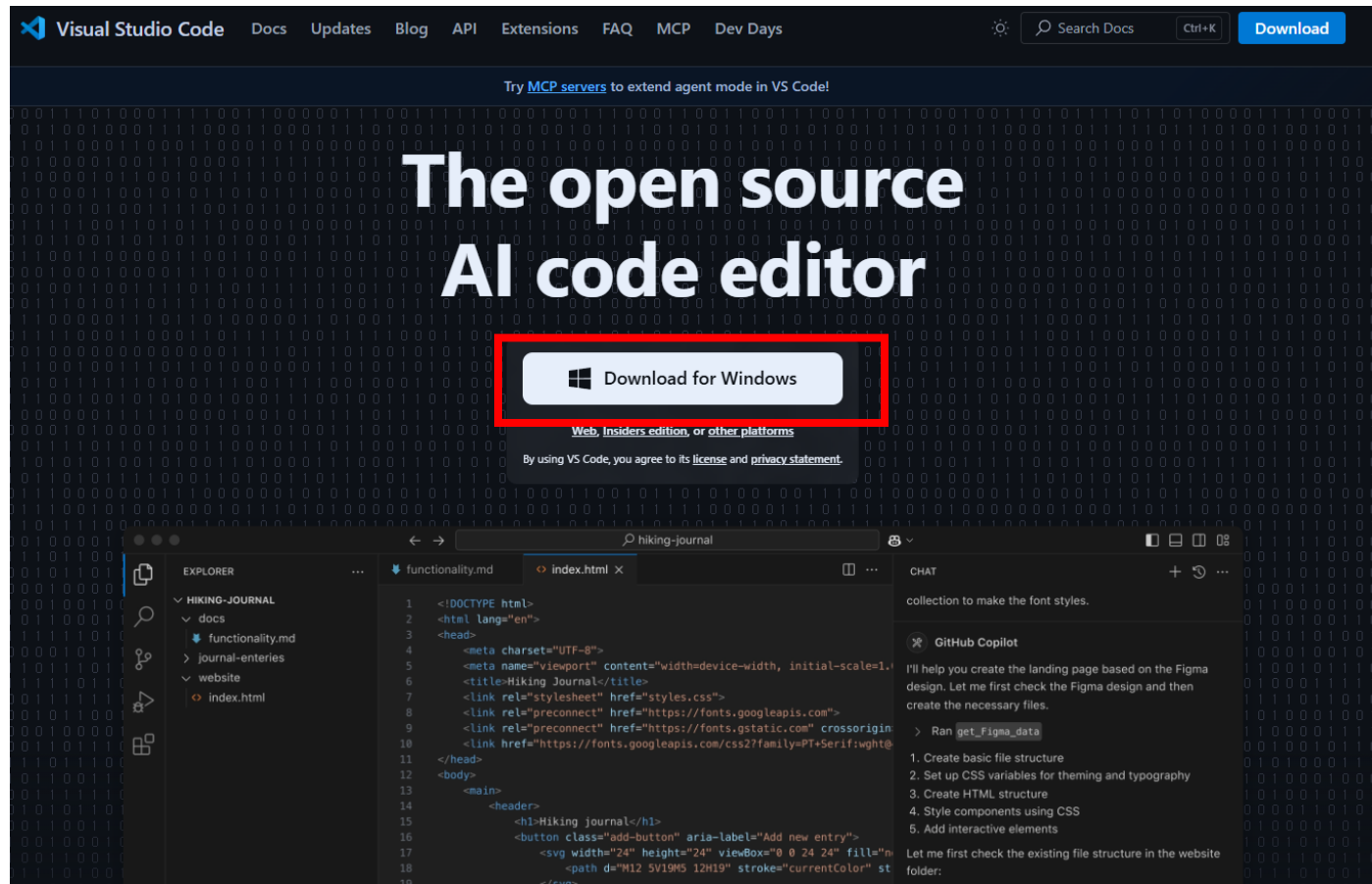
## 1.7 환경설정

- Github Copilot

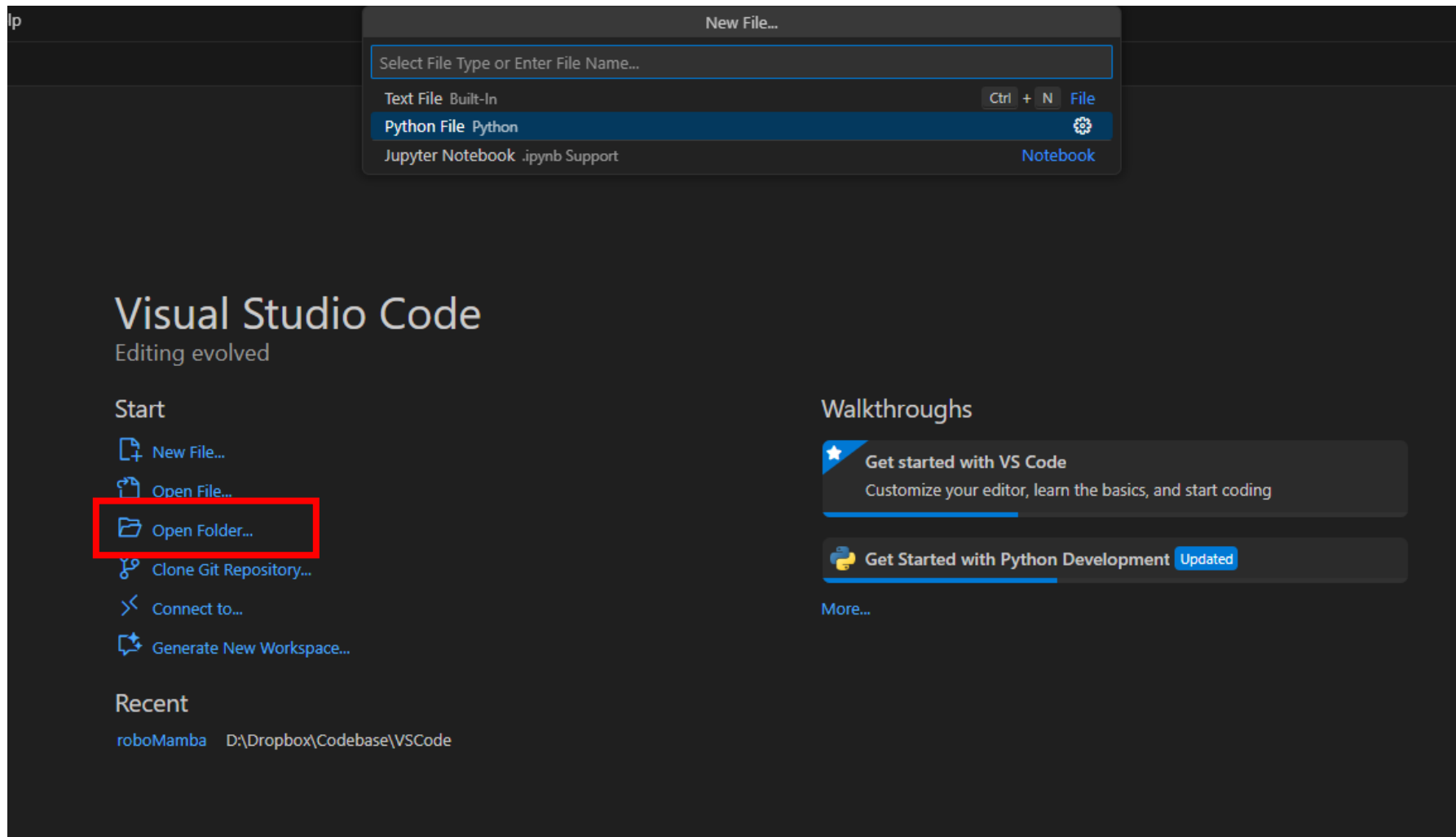


# 1.7 환경설정

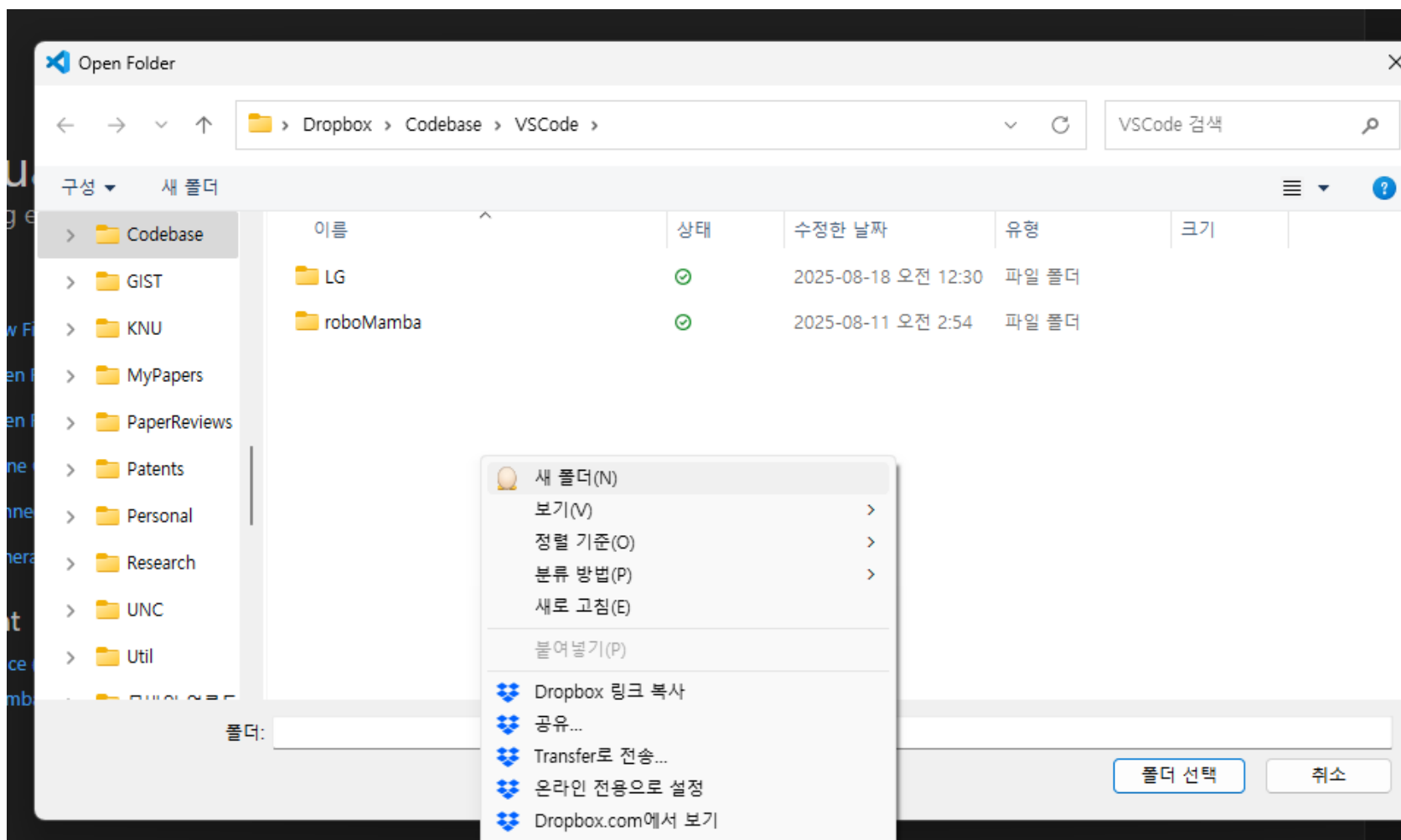
- <https://code.visualstudio.com/>



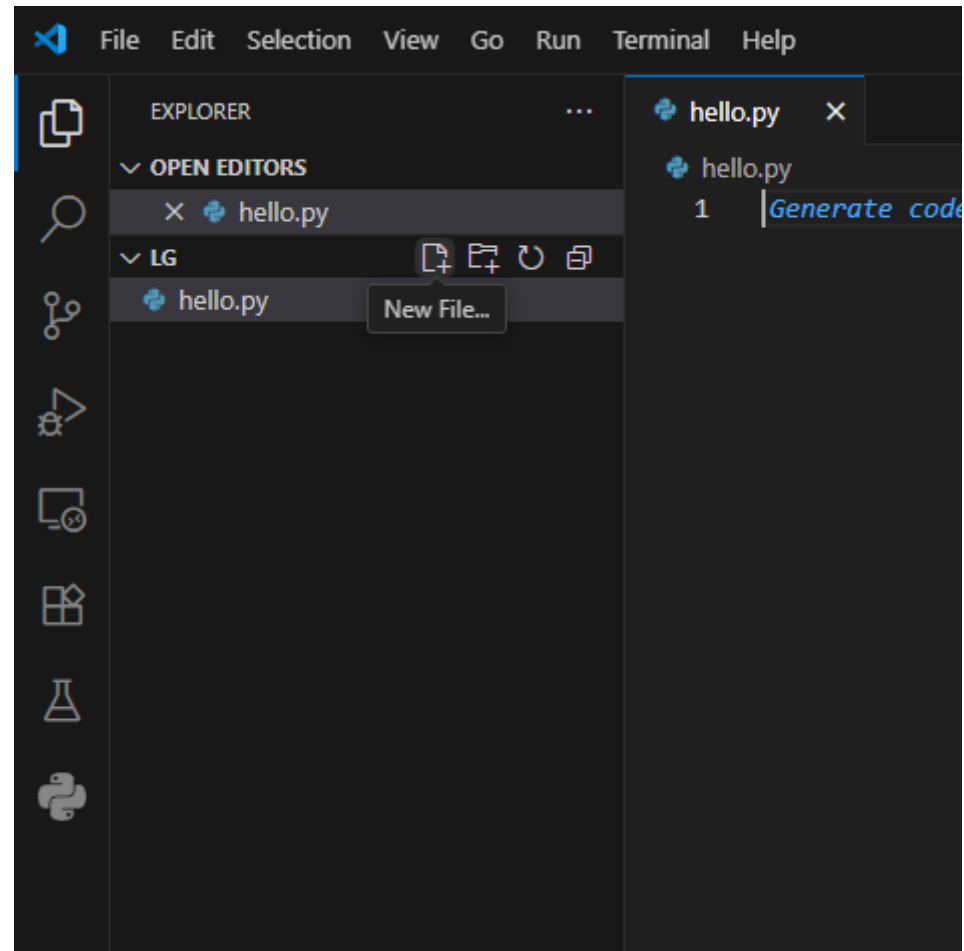
## 1.7 환경설정



## 1.7 환경설정

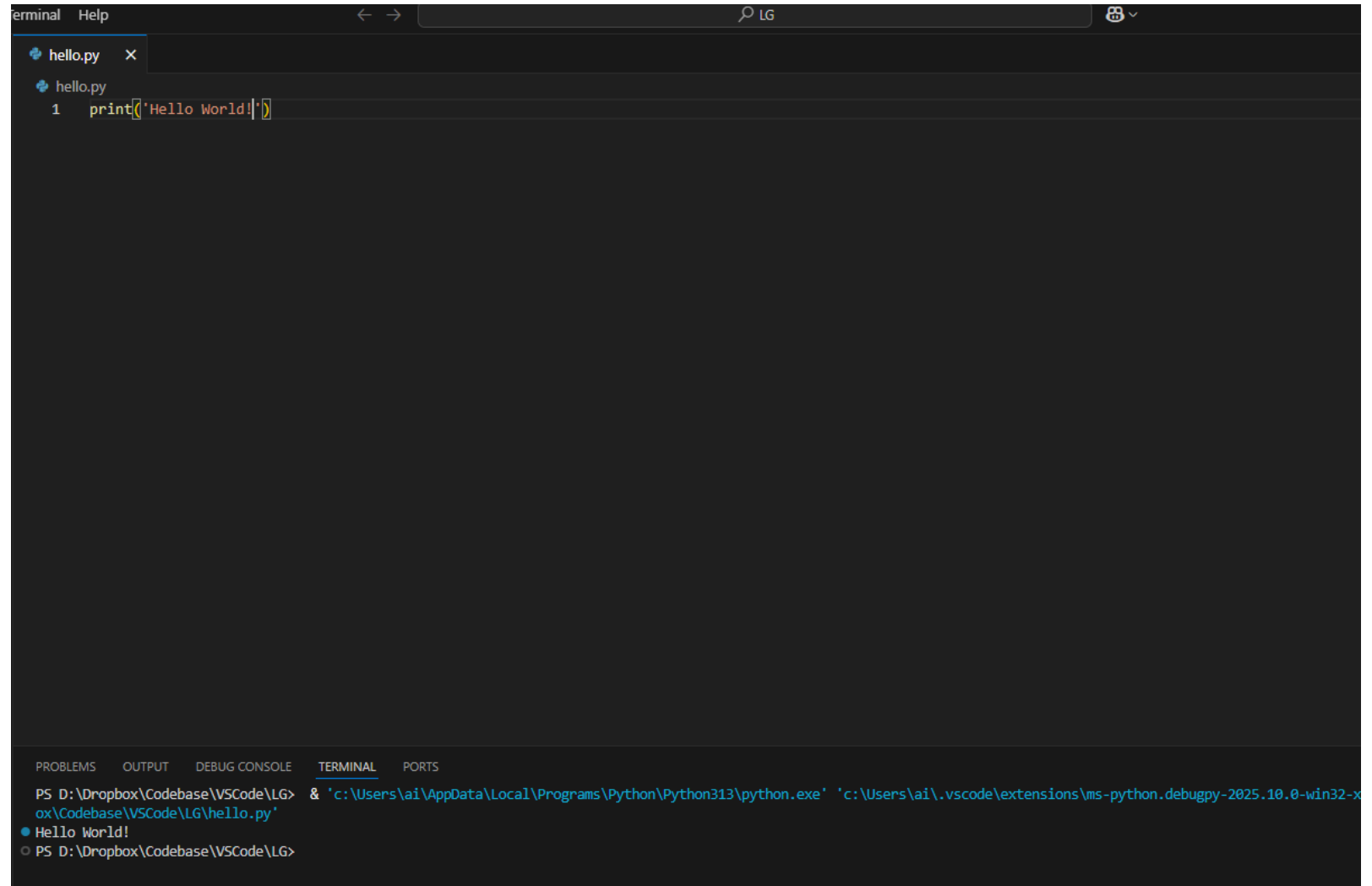


## 1.7 환경설정



# 1.7 환경설정

- Run : F5

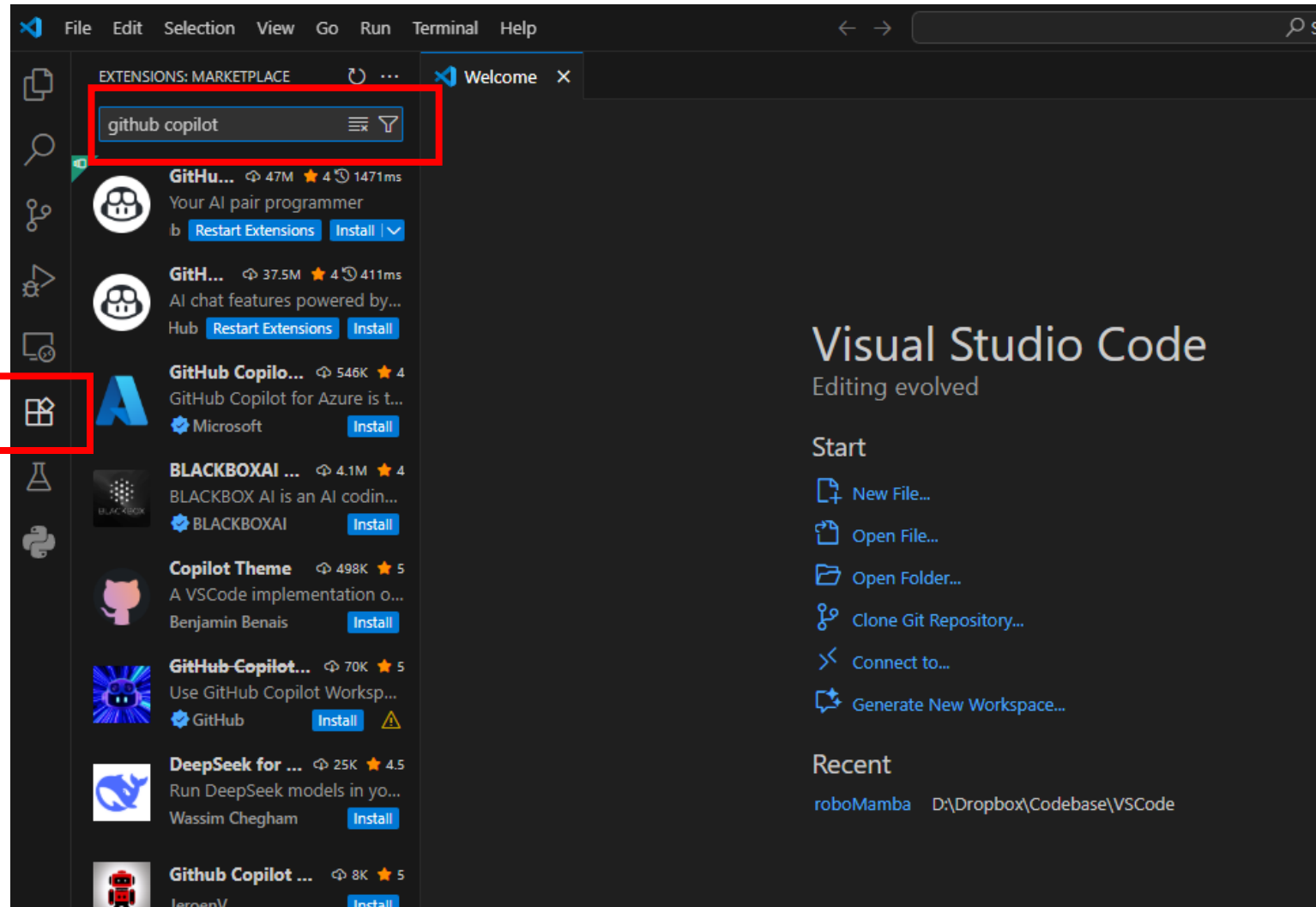


```
terminal Help
hello.py x
hello.py
1 print('Hello World!')
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Dropbox\Codebase\VSCode\LG> & 'c:\Users\ai\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ai\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x
ox\Codebase\VSCode\LG\hello.py'
● Hello World!
○ PS D:\Dropbox\Codebase\VSCode\LG>
```

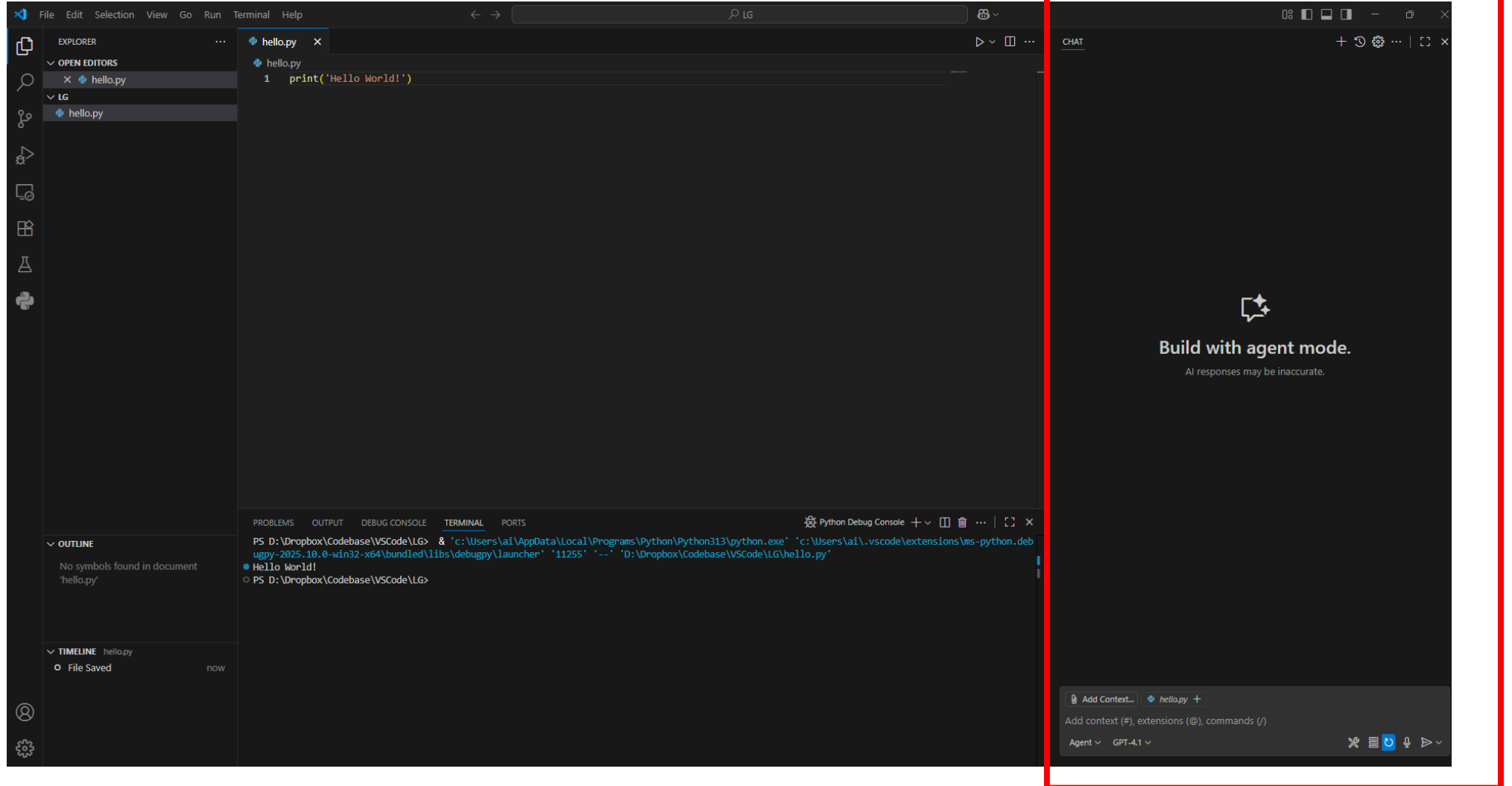
## 1.7 환경설정

Ctrl+Shift+X

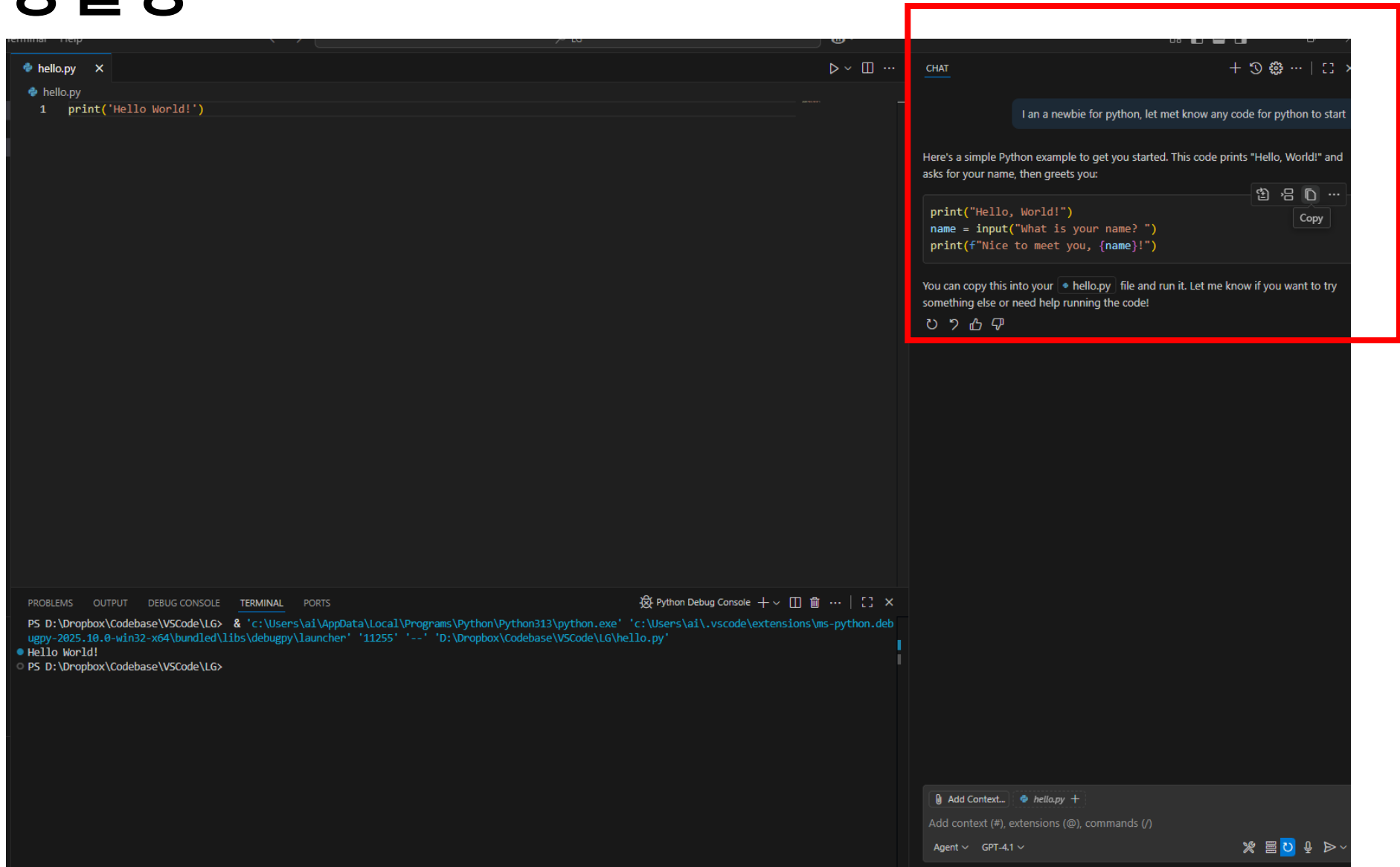


# 1.7 환경설정

Ctrl+Alt+I

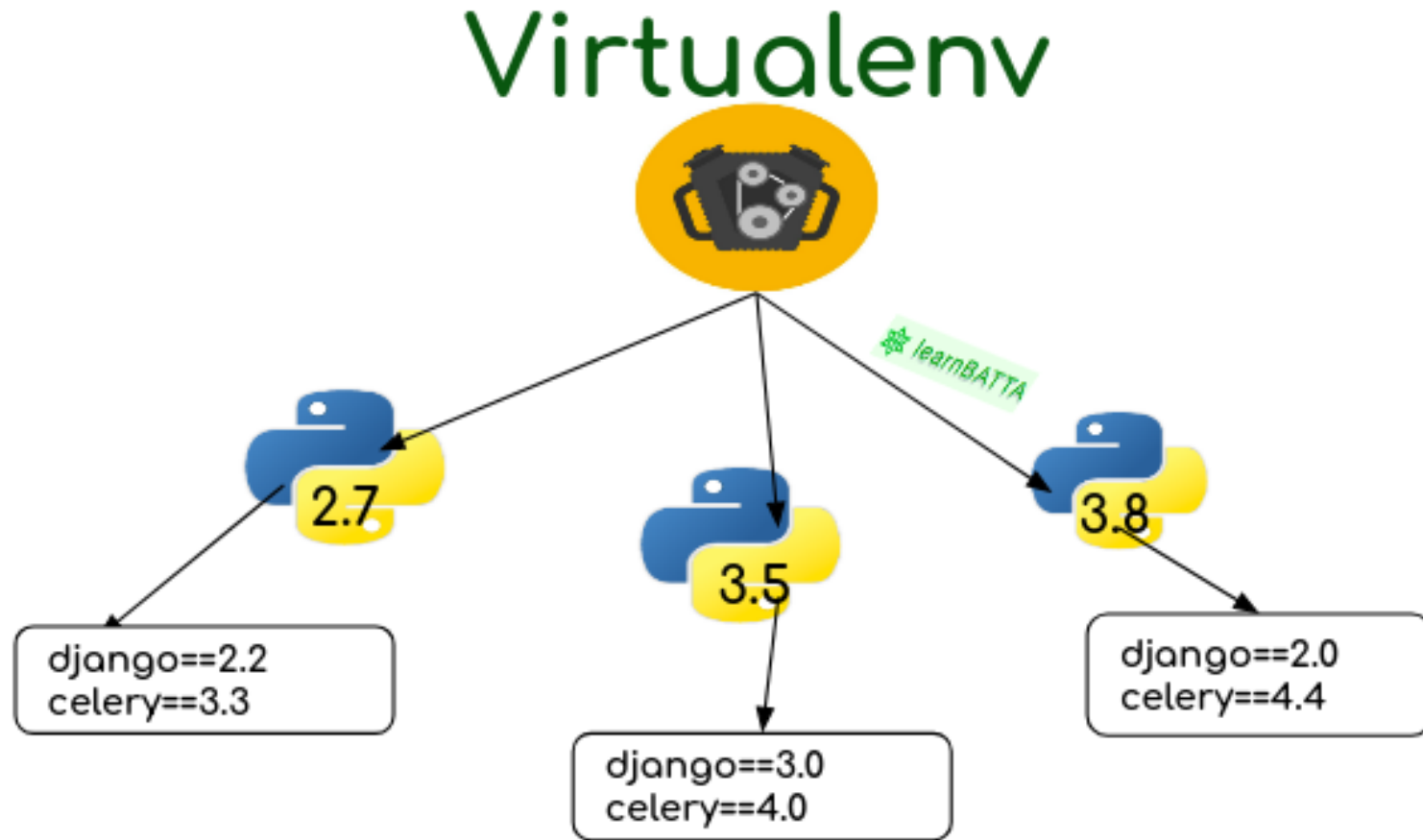


## 1.7 환경설정



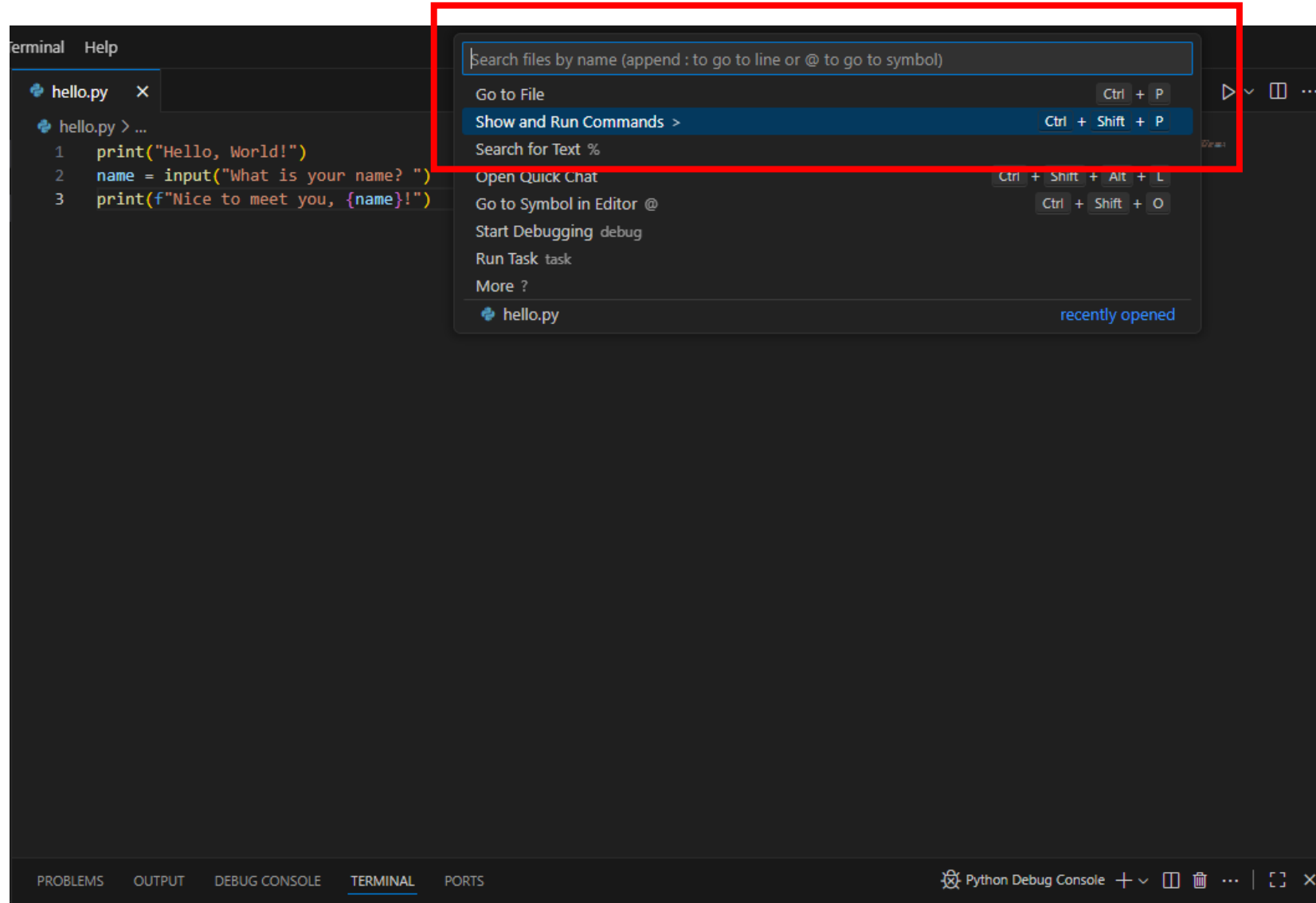
## 1.7 환경설정

가상환경

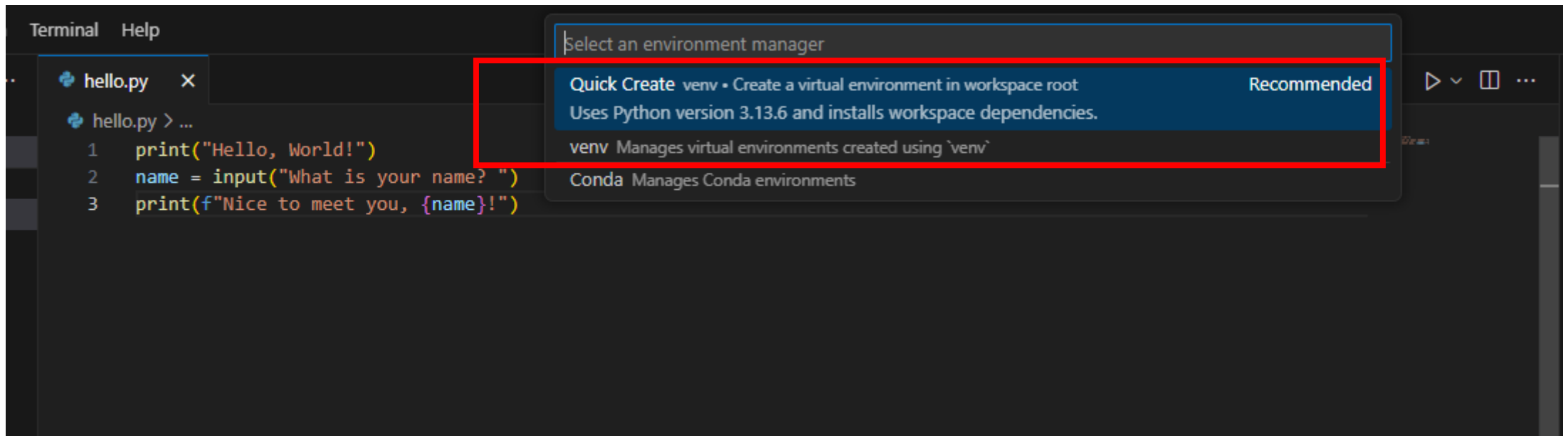
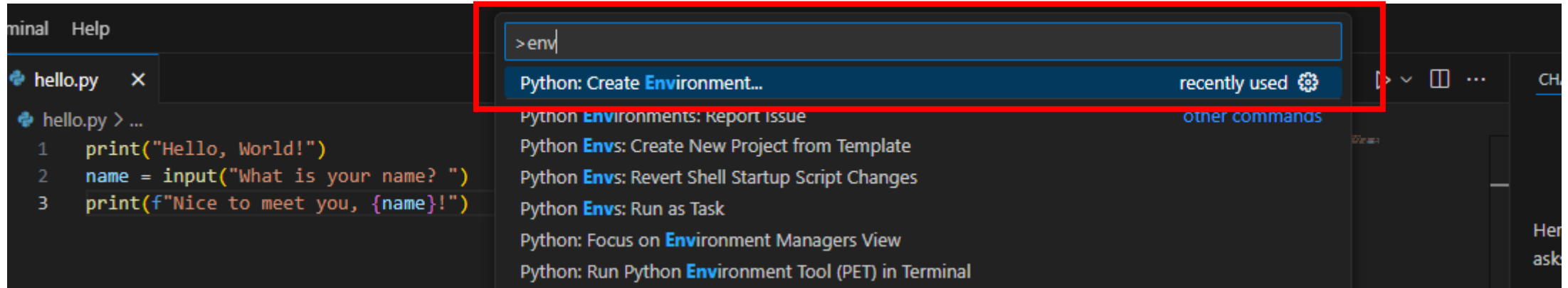


## 1.7 환경설정

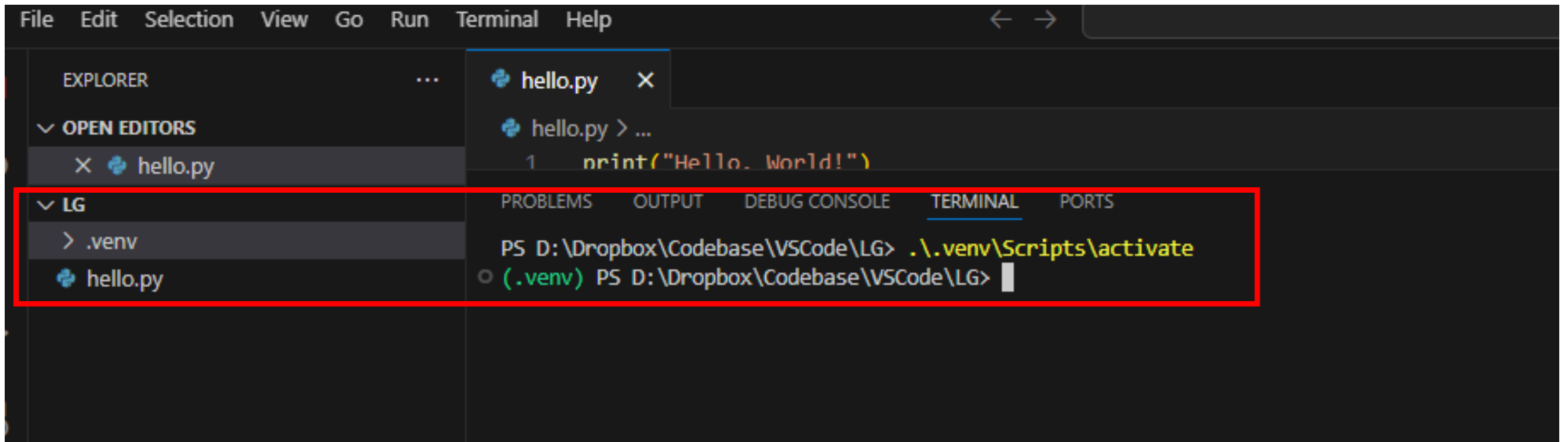
**Ctrl+Shift+P**



## 1.7 환경설정



## 1.7 환경설정



## 1.7 환경설정

- 설치된 패키지 리스트 확인

pip list

```
(.venv) PS D:\Dropbox\Codebase\VSCode\LG> pip list
Package Version
-----
pip      25.2
```

## 1.7 환경설정

- (주의) 패키지 설치 시 본인 가상환경에 설치
- 패키지 설치 시 터미널에서 pip install 사용

> pip install numpy

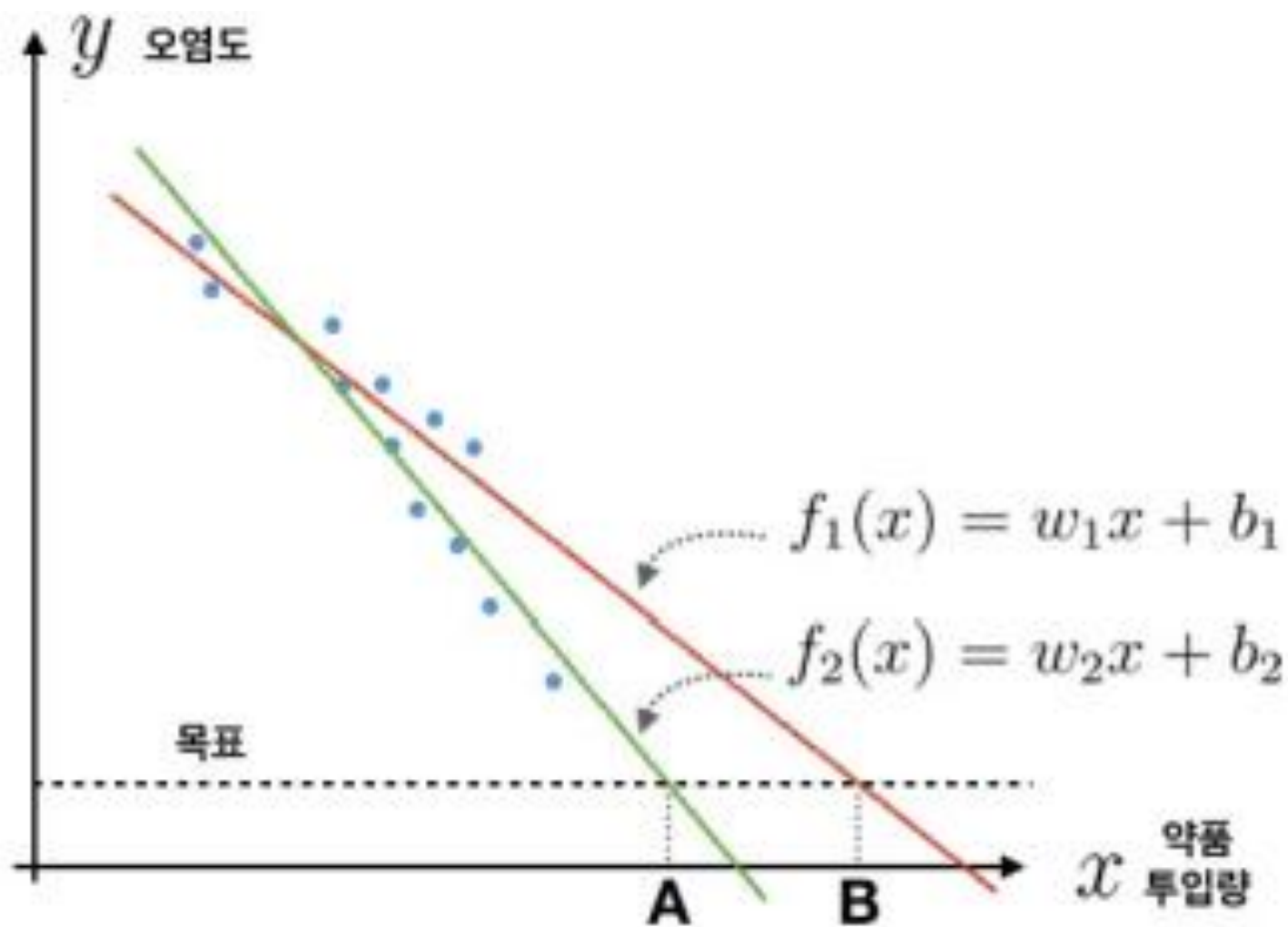
```
● (.venv) PS D:\Dropbox\Codebase\VSCode\LG> pip install numpy
Collecting numpy
  Using cached numpy-2.3.2-cp313-cp313-win_amd64.whl.metadata (60 kB)
Using cached numpy-2.3.2-cp313-cp313-win_amd64.whl (12.8 MB)
Installing collected packages: numpy
Successfully installed numpy-2.3.2
○ (.venv) PS D:\Dropbox\Codebase\VSCode\LG> 
```

## 2. 회귀 모델

## 2.1 회귀 모델

- 회귀 regression : 회귀 분석은 대표적인 지도학습 알고리즘
  - 관측된 데이터를 통해 독립변수와 종속변수 사이의 숨어있는 관계를 추정하는 것
- 선형 회귀 linear regression 는  $y = f(x)$ 에서 입력  $x$ 에 대응되는 실수  $y$ 들이 주어지고 추정치  $f(x)$ 가 가진 오차를 측정
  - 이 오차를 줄이는 방향으로 함수의 계수를 최적화하는 일

## 2.1 회귀 모델

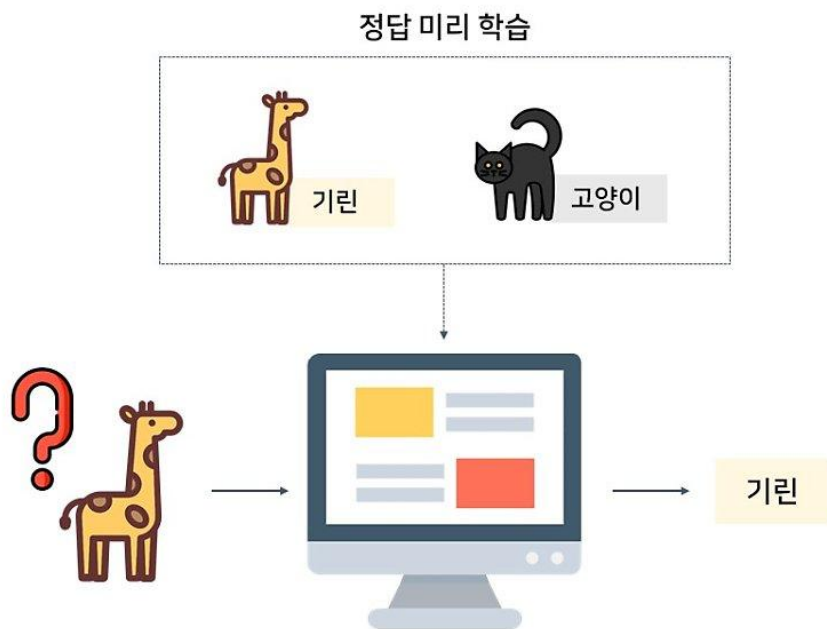


## 2.1 회귀 모델

- 데이터에 숨겨진 관계를 표현하고, 약품 투입량과 같은 독립변수에 대해 오염도라는 종속 변수가 어떤 값을 가질지 예측하는  $f_a(x)$ 와  $f_b(x)$ 를 가설hypothesis이라고 부름
- 좋은 가설은 오차error가 작은 가설
  - 회귀 분석은 데이터를 설명하는 좋은 가설을 찾는 것

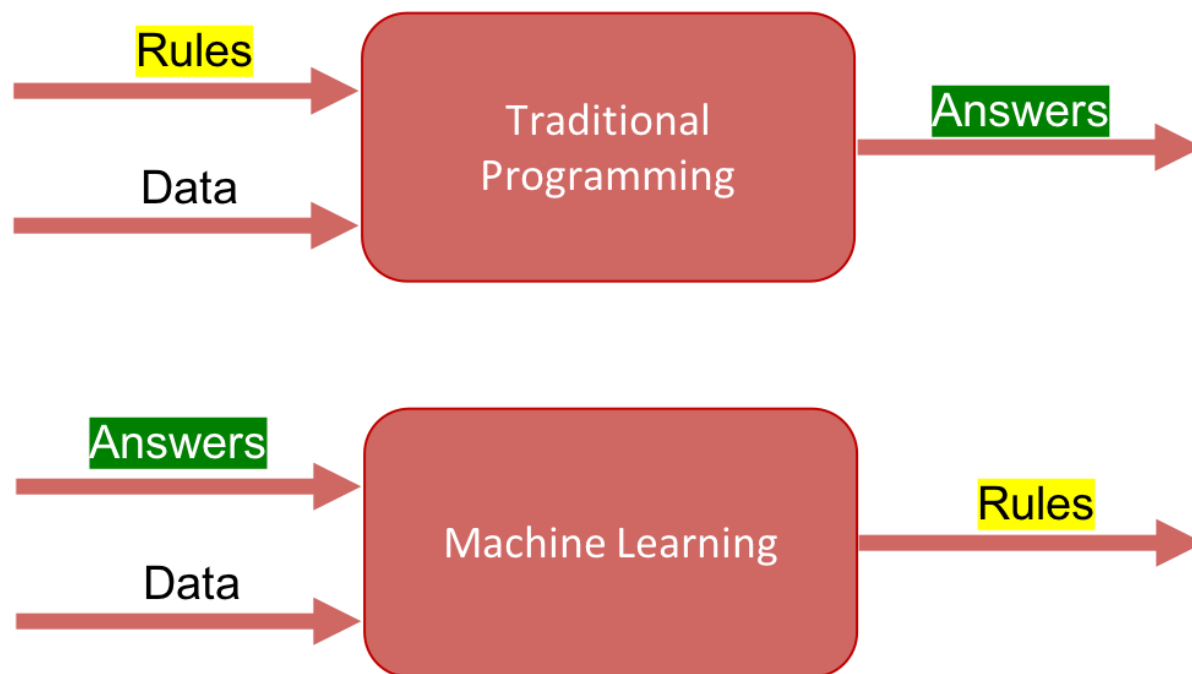
## 2.2 선형 회귀와 지도 학습

- 데이터에 제시된 목표값을 정답값 혹은 레이블<sup>label</sup>이라고 부름
  - 지도 학습은 주어진 입력-출력 쌍을 학습한 후에 새로운 입력값이 들어왔을 때, 합리적인 출력값을 예측하는 것.



## 2.2 선형 회귀와 지도학습

- 전통적 프로그래밍 : 사람이 이 함수  $f(x)$ 를 고안하여 구현한 뒤, 입력  $x$ 를 넣어 답  $y$ 를 얻는 것
- 머신러닝: 데이터  $(x, y)$ 를 주면 함수  $f(x)$ 를 만들어내는 일

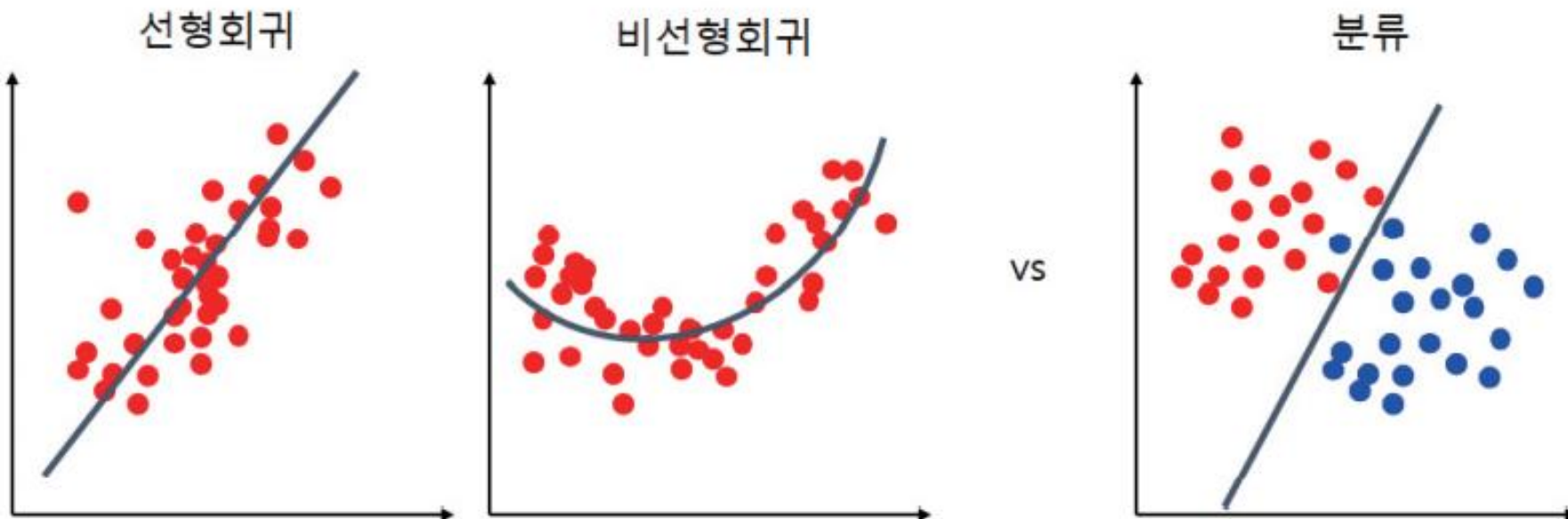


## 2.2 선형 회귀와 지도 학습

- 지도 학습 알고리즘의 대표적인 두 유형은 회귀 분석과 분류classification
  - 회귀는 입력 데이터 하나 하나에 대응하는 출력값을 예측
  - 분류는 입력 데이터를 몇 가지 이산적인 범주category 중의 하나로 대응

## 2.2 선형 회귀와 지도학습

- 회귀 분석 : 데이터를 설명하는 직선을 찾는 선형<sup>linear</sup> 회귀와 곡선을 찾는 비선형<sup>nonlinear</sup> 회귀
  - 이진 분류<sup>binary classification</sup>는 데이터를 양분하는 경계 직선 혹은 곡선을 찾는 것.



## 2.2 선형 회귀와 지도학습

- 데이터를 학습시킬 때, 데이터에서 중요한 일부 정보만을 추출하여 이것으로 학습시키고 테스트할 수도 있음
  - **특징** feature : 특징이란 관찰되는 현상에서 측정할 수 있는 개별적인 **속성** attribute을 의미
  - $y = f(x)$ 의 함수를 찾는다고 할 때, 입력 데이터로 사용되는  $x$ 가 특징
- 기계학습에서 다룰 수 있는 Feature의 예
  - **사람의 키와 몸무게**
  - **개의 몸통 길이와 높이**
  - **주택 가격에 영향을 주는 주택의 특징들**

## 2.3 실제 데이터를 읽고 가설 설정

- Pandas를 이용하여 CSV 파일을 읽어 lin\_data라는 데이터프레임을 만듦
- 투입량에 따른 오염도 측정 결과 100건이 담겨 있음
  - 데이터프레임의 input 열은 오염도를 줄이기 위한 약품의 투입량
  - pollution 열은 실제 측정된 오염도 수치



```
import matplotlib.pyplot as plt
import pandas as pd
# 데이터 저장 위치
data_home = 'https://github.com/dknife/ML/raw/main/data/'
lin_data = pd.read_csv(data_home+'pollution.csv') # 데이터 파일 이름
print(lin_data)
```

	input	pollution
0	0.240557	4.858750
1	0.159731	4.471091
..	...	...
99	0.290294	3.169049

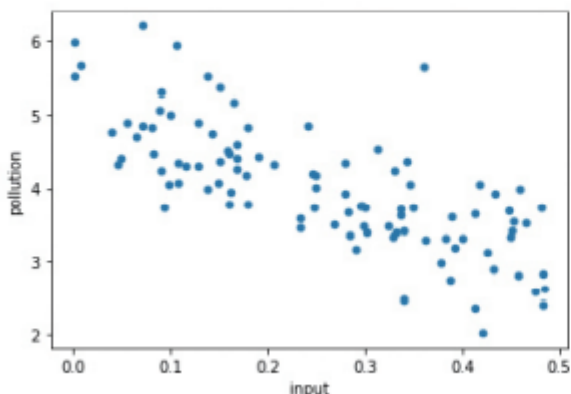
[100 rows x 2 columns]

## 2.3 실제 데이터를 읽고 가설 설정

- 데이터를 시각적으로 확인하기 위하여 plot() 메소드를 사용.
  - x축으로는 input 열을 사용
  - y축으로 pollution 열을 사용
  - 투입량을 늘이면 오염도가 줄어드는 경향이 있는 것을 확인할 수 있음



```
lin_data.plot(kind = 'scatter', x = 'input', y = 'pollution')
```

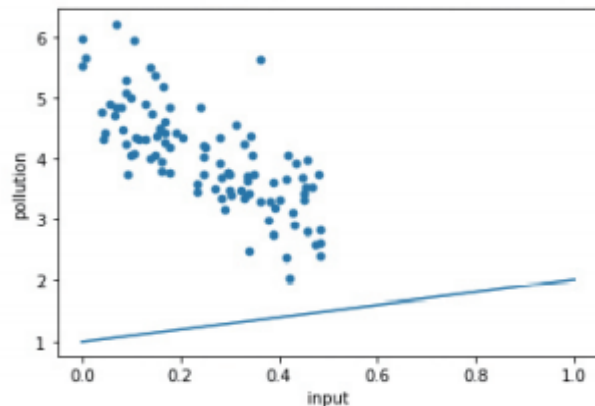


- input을 독립변수  $x$ 로, pollution을 종속변수  $y$ 로 하는  $y = wx + b$ 라는 직선으로 표현하면, 데이터가 이 함수를 따를 것이라는 **가설** hypothesis을 제시
  - pyplot의 plot() 함수는 독립변수의 리스트와 종속변수의 리스트를 주면, 이들을 연결한 선을 그려낸다.



```
w, b = 1, 1
x0, x1 = 0.0, 1.0
def h(x, w, b):          # 가설에 따라 값을 계산하는 함수
    return w*x + b

# 데이터(산포도)와 가설(직선)을 비교
lin_data.plot(kind = 'scatter', x = 'input', y = 'pollution')
plt.plot([x0, x1], [h(x0, w, b), h(x1, w, b)])
```

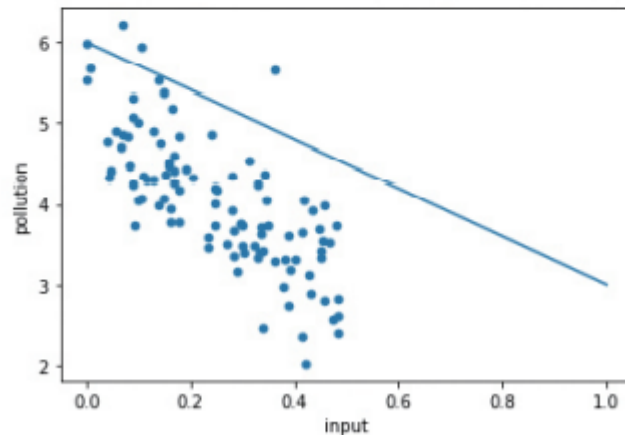


- 데이터와 일치시키기 위해서는 음수 기울기가 필요할 것
  - 최적의  $w$  와  $b$  를 찾을 수 있는 방법은?



```
w, b = -3, 6  
x0, x1 = 0.0, 1.0
```

```
# 새로운 파라미터로 가설(직선)과 데이터(산포도) 비교  
lin_data.plot(kind = 'scatter', x = 'input', y = 'pollution')  
plt.plot([x0, x1], [h(x0, w, b), h(x1, w, b)])
```



## 2.4 좋은 가설과 모델의 오차

- 데이터를 추정하는 가설이 얼마나 정확한지를 평가하는 방법
  - 가설이 훌륭한 모델이라면 데이터는 가설이 나타내는 직선 위에 모두 놓이게 될 것
  - 좋은 가설이라면 데이터가 이 직선들 근처에 있을 것

## 2.4 좋은 가설과 모델의 오차

- 대표적인 오차 척도는 평균 제곱 오차
- 예측치  $\hat{y}$ 와 정답 레이블  $y$ 사이의 차이를 제곱하여 모두 더한 뒤에 전체 데이터의 개수  $m$ 으로 나누는 것
- 평균 제곱 오차 mean square error: MSE라고 하며, 다음과 같은 식

$$E_{mse} = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2$$

## 2.4 좋은 가설과 모델의 오차

- 예측 결과가  $y_{\text{hat}}$ , 정답 레이블이  $y$ 에 저장되어 있다고 할 때, 평균 제곱 오차는 아래와 같이 구할 수 있음

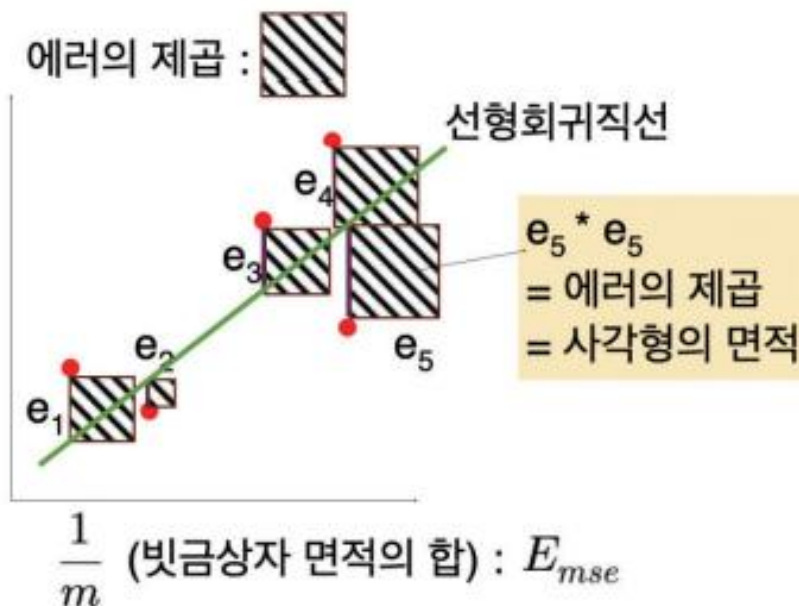
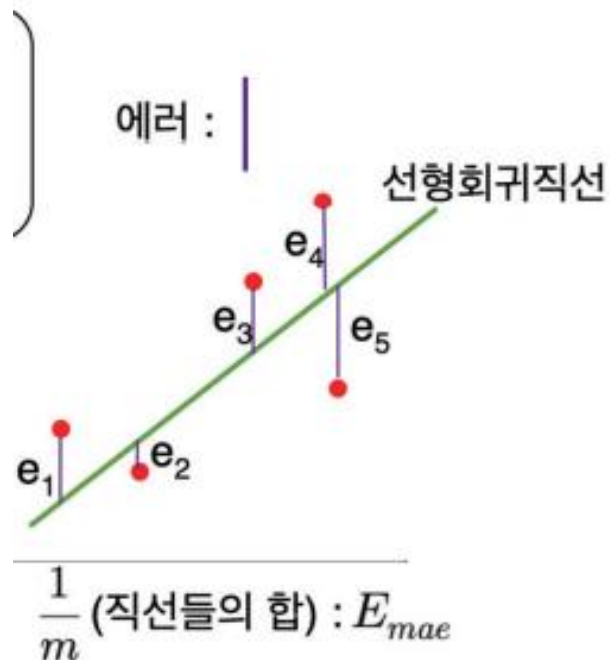


```
import numpy as np
y_hat = np.array([1.2, 2.1, 2.9, 4.1, 4.7, 6.3, 7.1, 7.7, 8.5, 10.1])
y = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
diff_square = (y_hat - y)**2
e_mse = diff_square.sum() / len(y)
e_mse
```

```
0.060999999999999996
```

## 2.4 좋은 가설과 모델의 오차

- 오차를 제공하는 이유
  - 빨간색 점으로 표시된 레이블과 가설 사이에 차이가 있음
  - $e_1$ 에서  $e_5$ 까지 전체 에러의 합이 최소가 되는 모델이 바람직한 모델
  - 오차합 곡면의 기울기를 따라 내려가 최소 오차에 접근하기 위하여



## 2.4 좋은 가설과 모델의 오차

- 평균 제곱 오차는 머신러닝에서 가장 흔히 사용되는 오차 척도
- 머신러닝의 대표적인 패키지 중 하나인 scikit-learn 역시 이러한 오차 계산을 지원한다 : `mean_squared_error()` 함수




```
from sklearn.metrics import mean_squared_error  
print('Mean squared error:', mean_squared_error(y_hat, y))
```

```
Mean squared error: 0.060999999999999996
```

## 2.4 좋은 가설과 모델의 오차

- <https://scikit-learn.org/stable/>

 [Install](#) [User Guide](#) [API](#) [Examples](#) [Community](#) [More](#)

# scikit-learn

Machine Learning in Python

[Getting Started](#) [Release Highlights for 1.2](#) [GitHub](#)

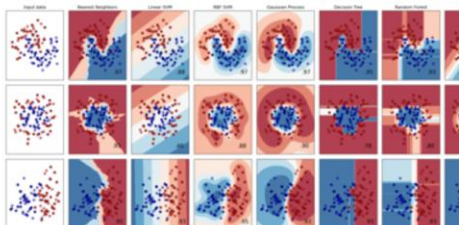
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

### Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, and more...

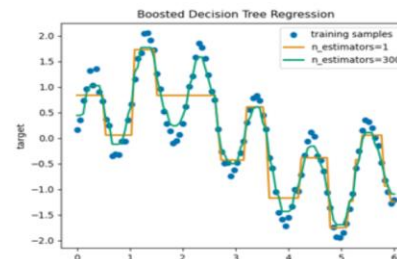


### Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, nearest neighbors, random forest, and more...

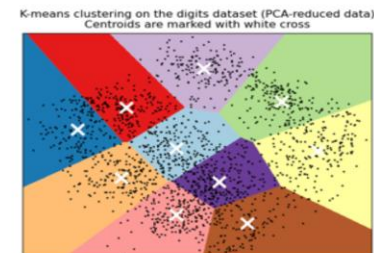


### Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, and more...



## 2.4 좋은 가설과 모델의 오차

- 제곱을 하지 않고 오차를 더하고 싶다면 **평균 절대 오차** mean absolute error: MAE 라는 것을 사용할 수 있음
  - 오차를 제곱하지 않고 절대값을 취해 더하는 것

$$E_{mae} = \frac{1}{m} \sum_{i=1}^m |\hat{y}_i - y_i|$$

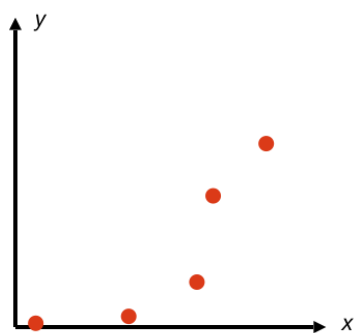


```
from sklearn.metrics import mean_absolute_error
print('Mean absolute error:', mean_absolute_error(y_hat, y))
```

```
Mean absolute error: 0.209999999999999988
```

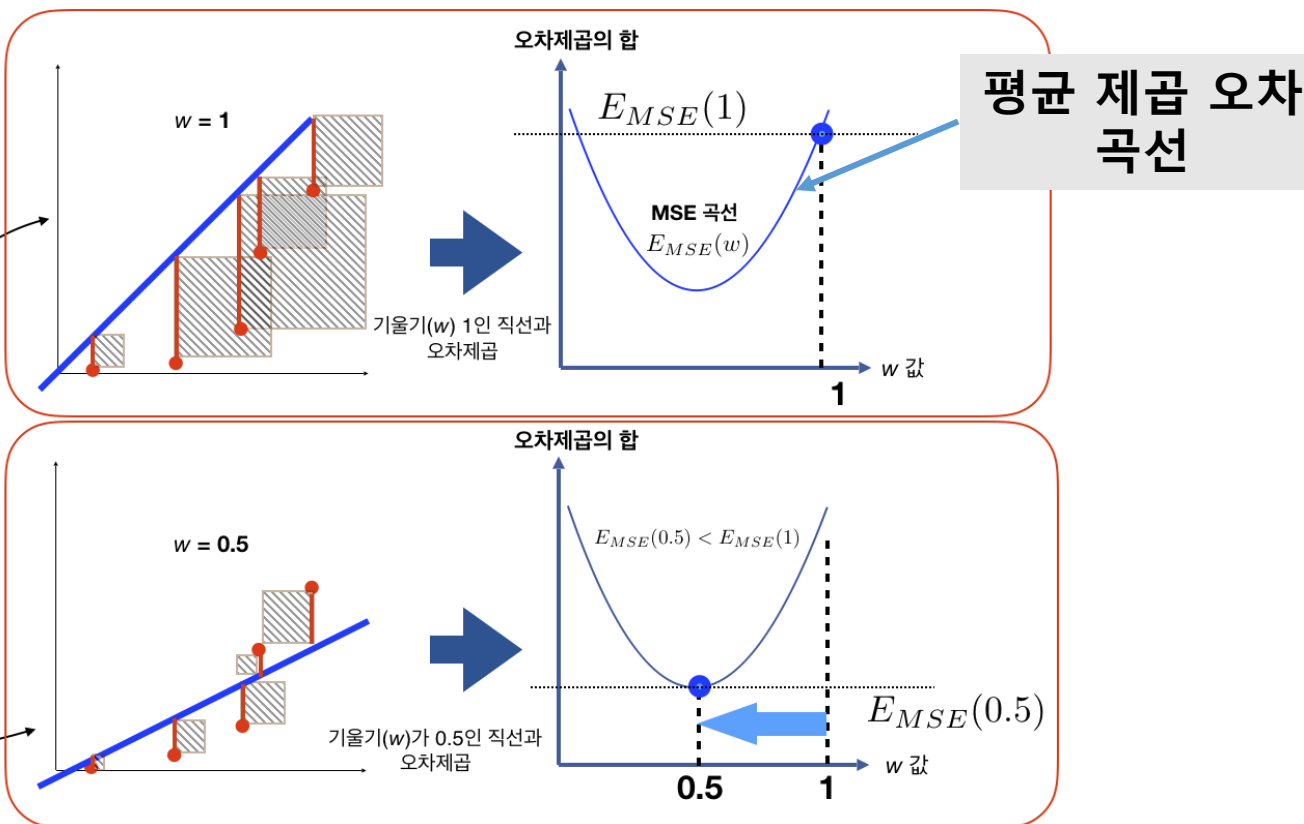
## 2.5 선형 회귀 함수의 시각적 이해

- 5 개의 점이 분포하고 있는데 이 분포를 잘 설명하는  $y = f(x)$ 의 함수를 찾는 것이 바로 선형 회귀의 목적
  - $y = wx$ 와 같이  $y$ 절편이 0인 경우를 가정하고 최적의  $w$ 값을 찾는 과정



위의 다섯개 점들의 분포를 가장 잘 설명하는 1차 함수를 만들자.

가능한 여러 가설 중에서  $w=1$ 인 경우와  $w=0.5$ 인 경우만 살펴 보자



## 2.5 선형 회귀 함수의 시각적 이해

- 오차 제곱의 합  $E_{mse}^{0.5}$ 는  $E_{mse}^1$ 와 비교하여 작은 값
- 기울기 0.5가 이 점들의 분포를 설명하는 함수의 최적 기울기 값이라고 한다면, 오른쪽 하단과 같은 오목한 곡선의 가장 낮은 지점이 될 것
- 이러한 정보를 바탕으로 다음과 같은 코드를 작성해 보도록 하자
- 우선 실제 값을 가지는 5개의  $x, y$  데이터를 생성하고  $w = 1.0$ 으로 하는  $\hat{y}$ 에서  $w = 0.3$ 인  $\hat{y}$ 까지 0.1씩 값을 감소시켜가며  $w$ 와 평균제곱오차를 출력해 보도록 하자



```
import numpy as np
from sklearn.metrics import mean_squared_error as mse

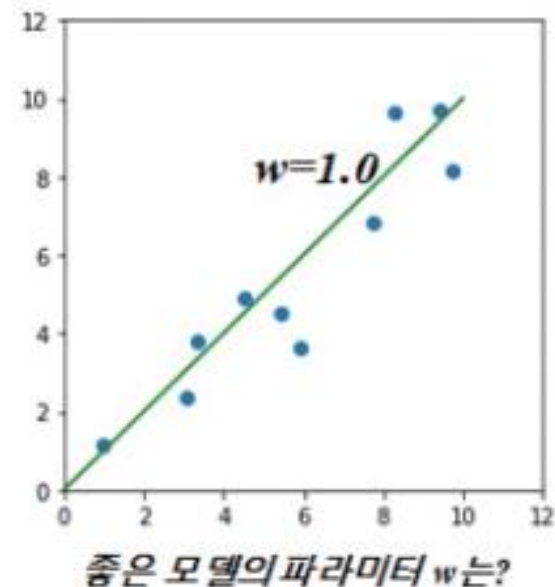
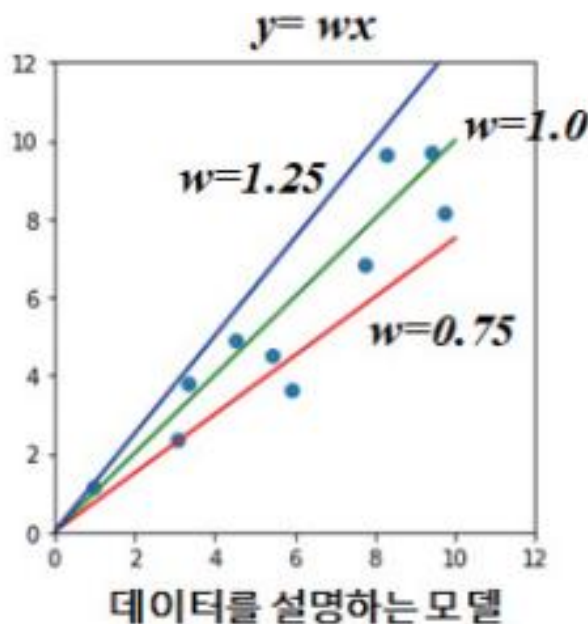
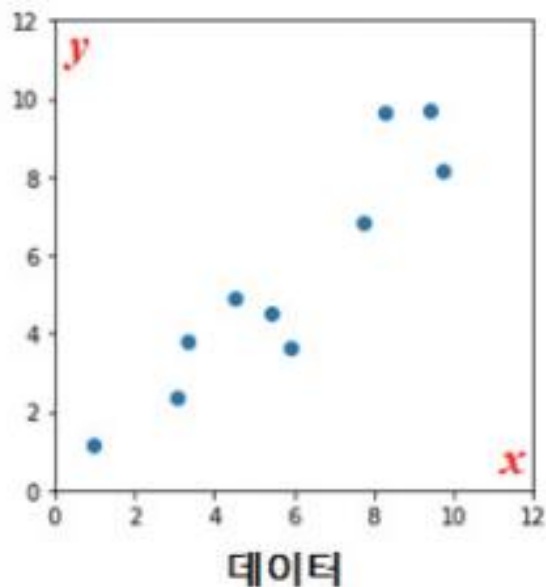
# 5개 점의 x, y 좌표값
x = np.array([1, 4.5, 9, 10, 13])
y = np.array([0, 0.2, 2.5, 5.4, 7.3])

w_list = np.arange(1.0, 0.2, -0.1)
for w in list(w_list):      # w를 바꾸어가며 예측치와 정답의 오차 비교
    y_hat = w * x
    print('w = {:.1f}, 평균제곱 오차: {:.2f}'.format(w, mse(y_hat, y)))
```

```
w = 1.0, 평균제곱 오차: 23.08
w = 0.9, 평균제곱 오차: 15.86
w = 0.8, 평균제곱 오차: 10.13
w = 0.7, 평균제곱 오차: 5.89
w = 0.6, 평균제곱 오차: 3.13
w = 0.5, 평균제곱 오차: 1.85
w = 0.4, 평균제곱 오차: 2.06
w = 0.3, 평균제곱 오차: 3.75
```

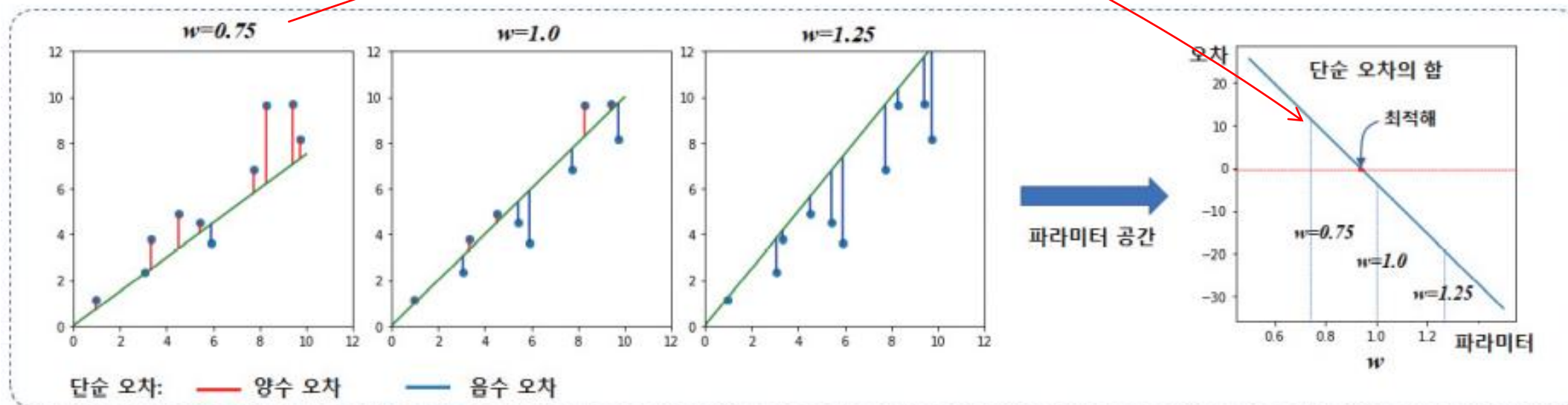
## 2.6 오차의 종류에 따른 오차 곡면의 모습

- 모델의 오차를 계산하는 방법에 따른 오차 곡면의 모습을 살펴보고, 이 오차 곡면을 이용하여 최적의 모델을 찾는 방법에 대해 살펴보도록 하자
- 목표: 데이터  $x$ 와  $y$ 의 관계를 가장 잘 설명하는 모델  $y = wx$ 를 찾는 것



## 2.6 오차의 종류에 따른 오차 곡면의 모습

- 예측과 정답의 차이로 계산하는 단순한 오차를 살펴 보자.
- 그림의 왼쪽에는 파라미터가 0.75, 1.0, 1.25일 때 모델과 데이터의 오차를 보이고 있다.

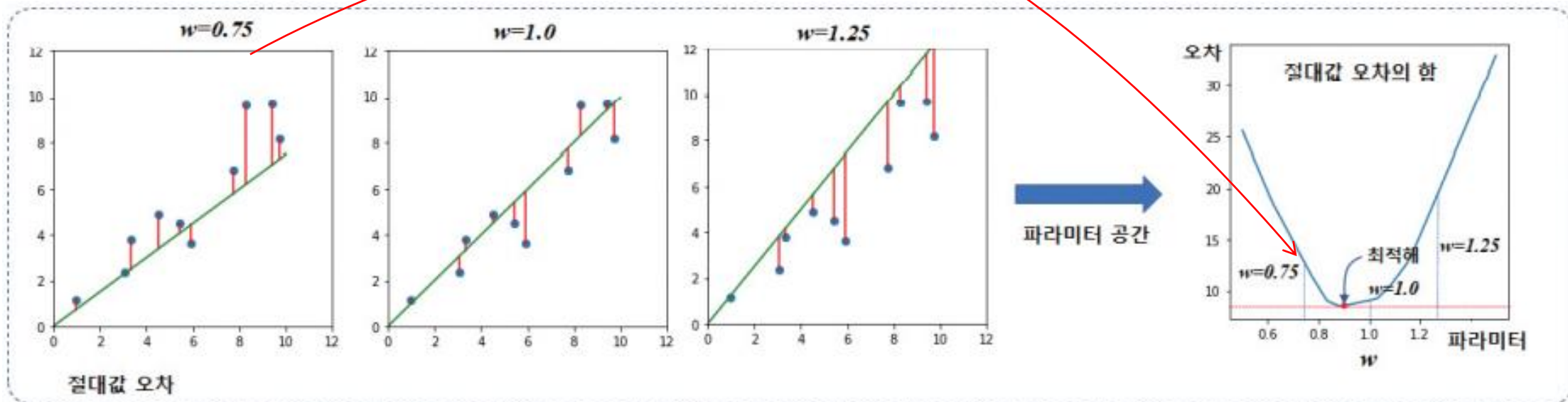


$w$ 가 0.75일 때 오차의 합은 10 근처의 값

## 2.6 오차의 종류에 따른 오차 곡면의 모습

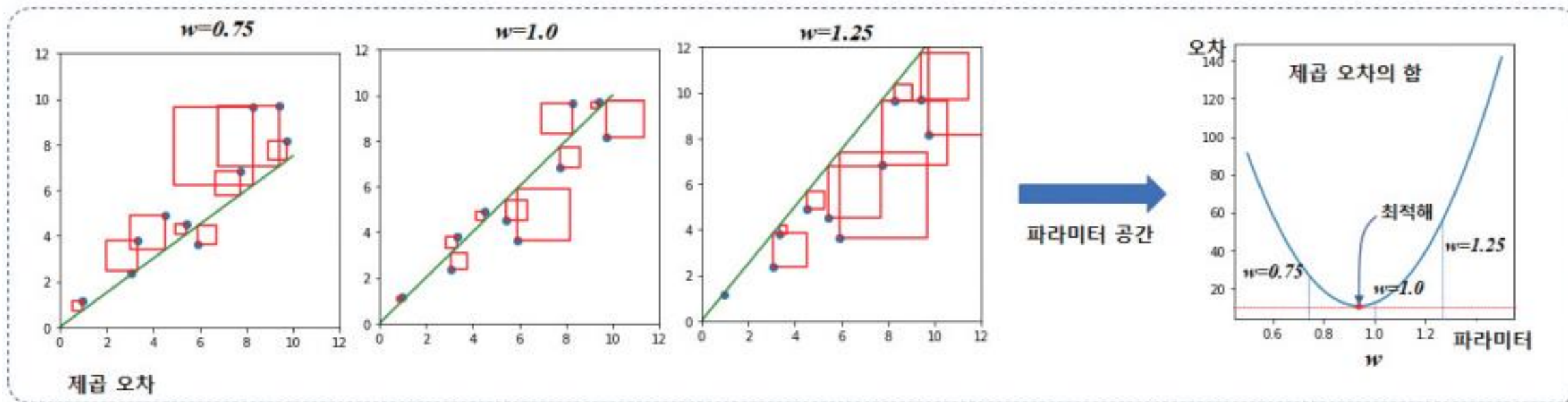
- 평균 절대 오차<sup>MAE</sup>
- 교차지점을 구할 필요없이 오차 곡선의 극소만 구하면 된다.

w가 0.75일 때 오차의 합은 14근처의 값



## 2.6 오차의 종류에 따른 오차 곡면의 모습

- 정답과 예측의 차이를 제공하는 평균 제곱 오차<sup>MSE</sup>



## 2.7 오차로 가설을 평가하고 좋은 가설 찾기

- 여러가지 가설이 존재할 경우 가설이 추정한 종속변수와 실제 데이터의 종속변수의 차이가 적을수록 좋은 것
- 가설에 의한 추정치 값과 실제 값의 차이를 오차error
  - 추정치는 일반적으로 종속변수  $y$ 의 위에  $\wedge$ 기호를 씌운  $\hat{y}$ ('^'기호는 hat으로 읽는다)로 표기
  - 오차  $E$ 는 다음과 같은 식으로 계산할 수 있다.

$$E = \hat{y} - y = wx + b - y$$

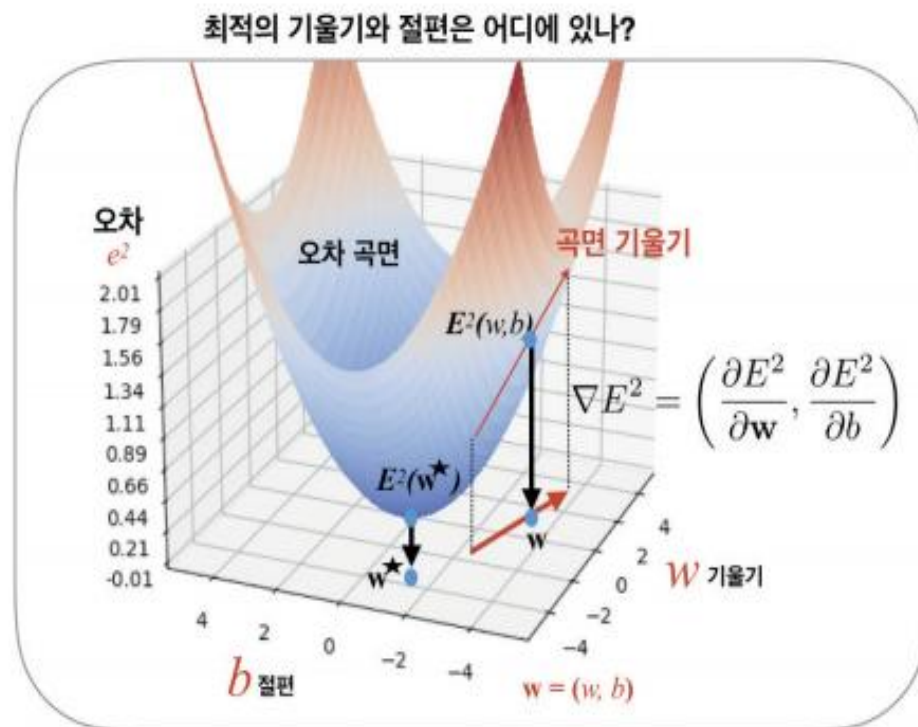
## 2.7 오차로 가설을 평가하고 좋은 가설 찾기

- **최소 제곱법** least squares approximation은 오차를 제공하여 오차 곡면의 기울기를 따라 내려가 기울기가 0인 극소 지점을 찾는 것
- 오차의 제곱  $E^2(w, b)$ 이 그림과 같은 곡면이라면, 최적의  $w$ 와  $b$ 를 찾기 위한 오차 곡면의 기울기 방향은 다음과 같이 구할 수 있음

$$\nabla E^2 = \left( \frac{\partial E^2}{\partial w}, \frac{\partial E^2}{\partial b} \right)$$

$$\frac{\partial E^2}{\partial w} = \frac{\partial (wx + b - y)^2}{\partial w} = 2(wx + b - y)x = 2Ex$$

$$\frac{\partial E^2}{\partial b} = \frac{\partial (wx + b - y)^2}{\partial b} = 2(wx + b - y) \cdot 1 = 2E$$



## 2.7 오차로 가설을 평가하고 좋은 가설 찾기

- $(w, b)$  벡터를  $-(E_x, E)$  방향으로 옮겨주면 최적의  $w$ 와  $b$ 에 가까워질 것
- 아래 코드는 "조금"의 정도를 learning rate(학습률)라는 하이퍼파라미터로 제어하고 있다.  $n$ 개의 데이터  $x_i$ 에 대한 예측 오차가  $E_i$ 라고 할 때 다음과 같이 기울기  $w$ 와 절편  $b$ 를 수정하는 것

$$w \leftarrow w - \eta \sum_{i=1}^n E_i x_i, \quad b \leftarrow b - \eta \sum_{i=1}^n E_i$$

- 벡터화 연산을 사용하여 아래와 같이 직선의 기울기와 절편을 갱신



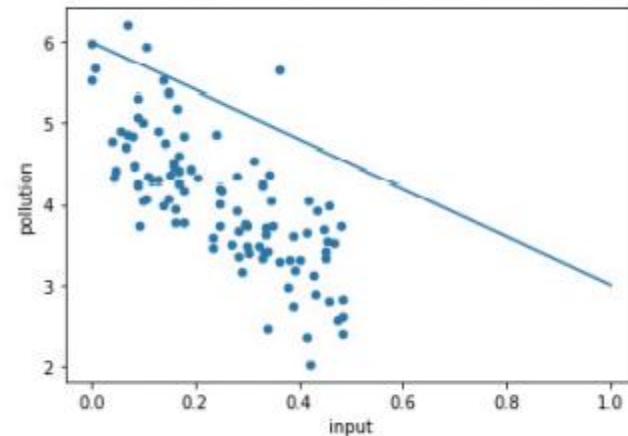
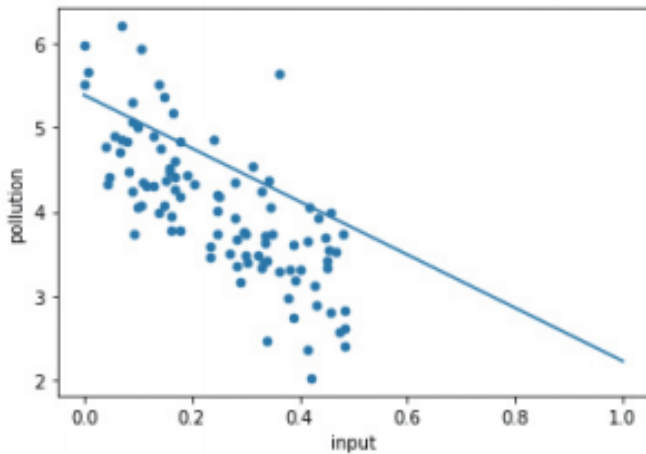
```
learning_rate = 0.005
w = w - learning_rate * (error * x).sum()
b = b - learning_rate * error.sum()
```

## 2.7 오차로 가설을 평가하고 좋은 가설 찾기

- 수정된 기울기  $w$ 와 절편  $b$ 를 이용하여 직선을 그려보자.
- 직선이 데이터를 잘 표현하도록 옮겨진 것을 확인할 수 있을 것

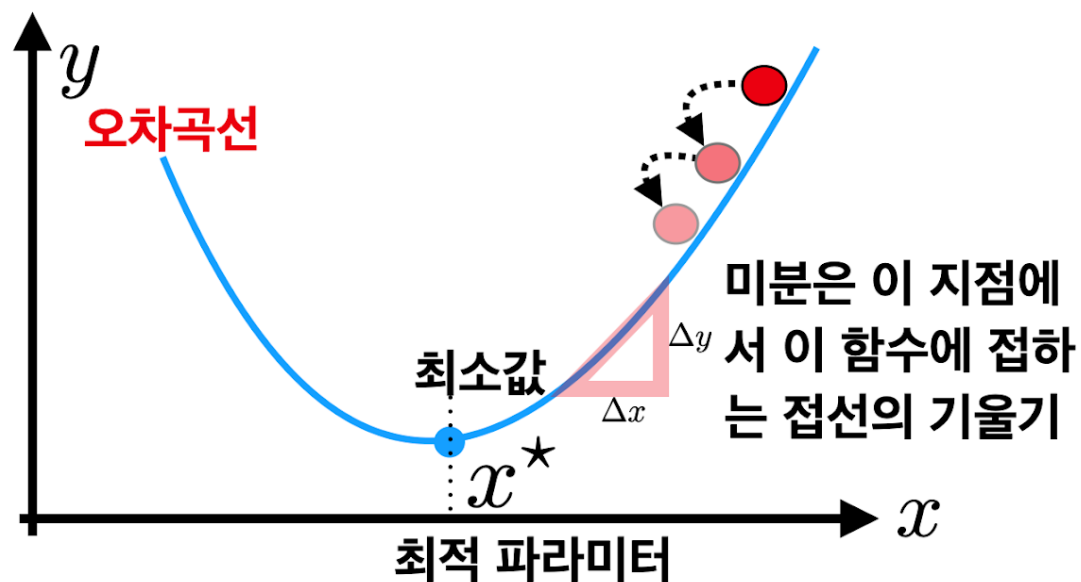


```
lin_data.plot(kind = 'scatter', x = 'input', y = 'pollution')  
plt.plot([x0, x1], [h(x0, w, b), h(x1, w, b)])
```



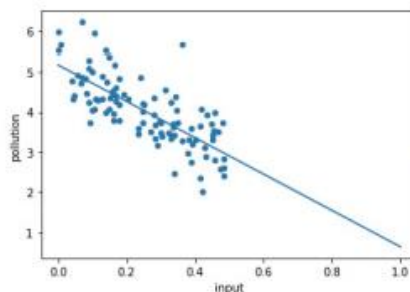
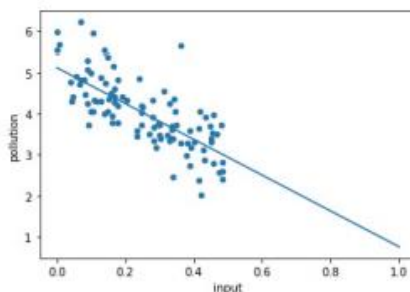
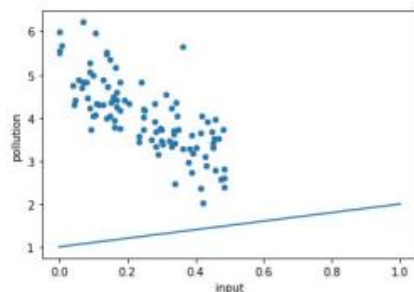
## 2.8 경사하강법

- 선형 회귀의 모델은 선형 방정식이고, 동작을 결정하는 파라미터는 직선의 기울기  $w$ 와 절편  $b$
- 벡터로 표현하면  $(w, b)$ 가 파라미터 벡터
- 학습 과정은 오차를 줄이도록 오차 곡면의 경사를 따라 내려가는 최적화 과정 : 경사 하강법(gradient descent)





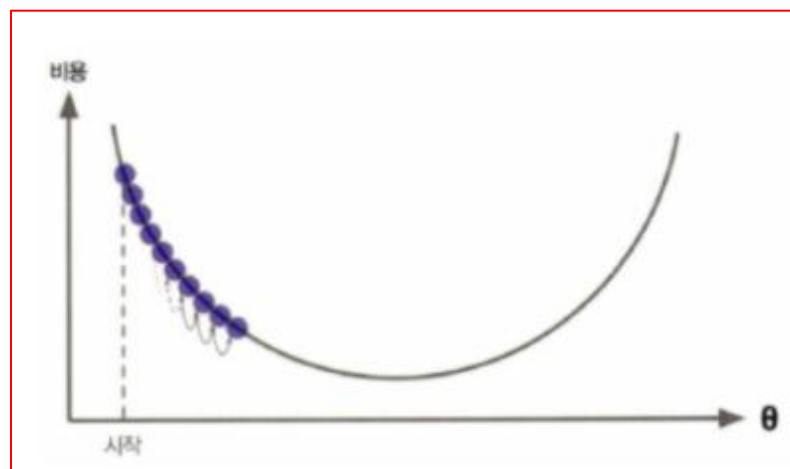
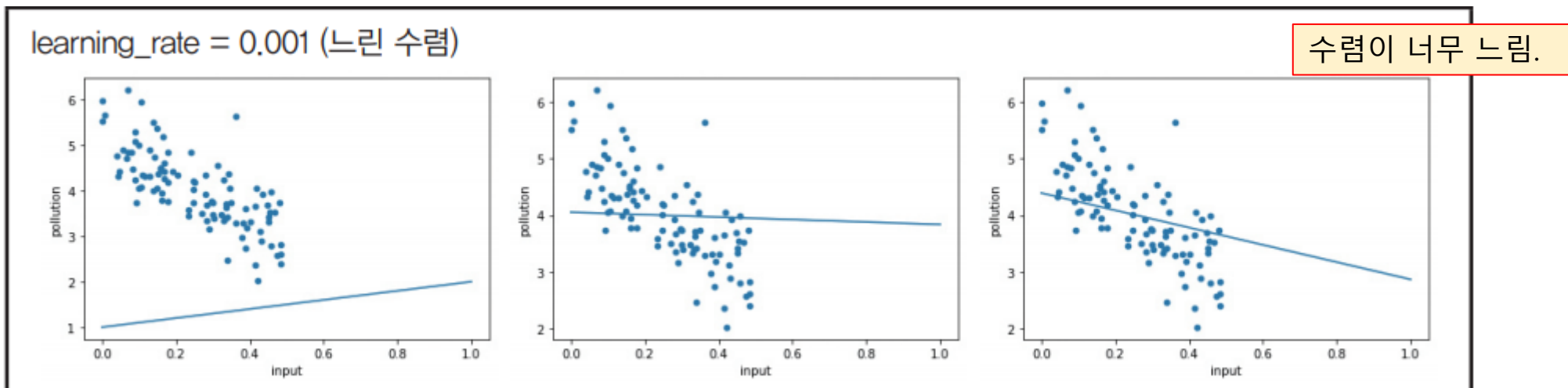
```
def h(x, param):  
    return param[0]*x + param[1]  
  
learning_iteration = 1000 # 하이퍼파라미터 : 학습반복 횟수  
learning_rate = 0.0025 # 하이퍼파라미터 : 학습율로 0.05, 0.001등이 가능  
  
param = [1, 1] # w, b를 하나의 변수로 함  
  
x = lin_data['input'].to_numpy()  
y = lin_data['pollution'].to_numpy()  
  
for i in range(learning_iteration):  
    if i % 200 == 0:  
        lin_data.plot(kind = 'scatter', x = 'input', y = 'pollution')  
  
        plt.plot([0, 1], [h(0, param), h(1, param)])  
        error = ( h(x, param) - y)  
        param[0] -= learning_rate * (error * x).sum()  
        param[1] -= learning_rate * error.sum()
```



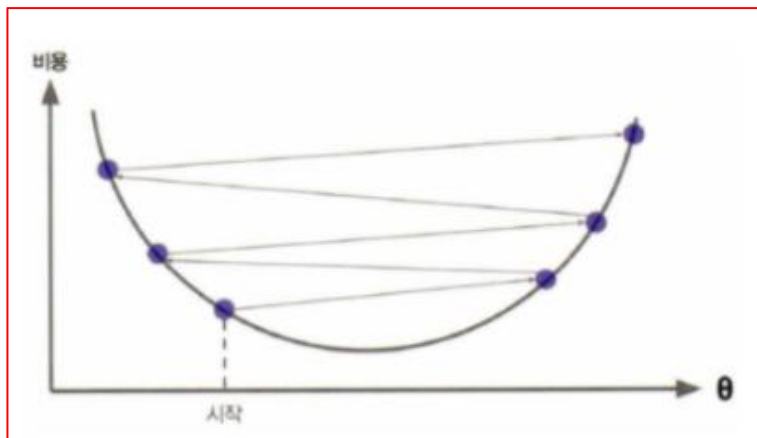
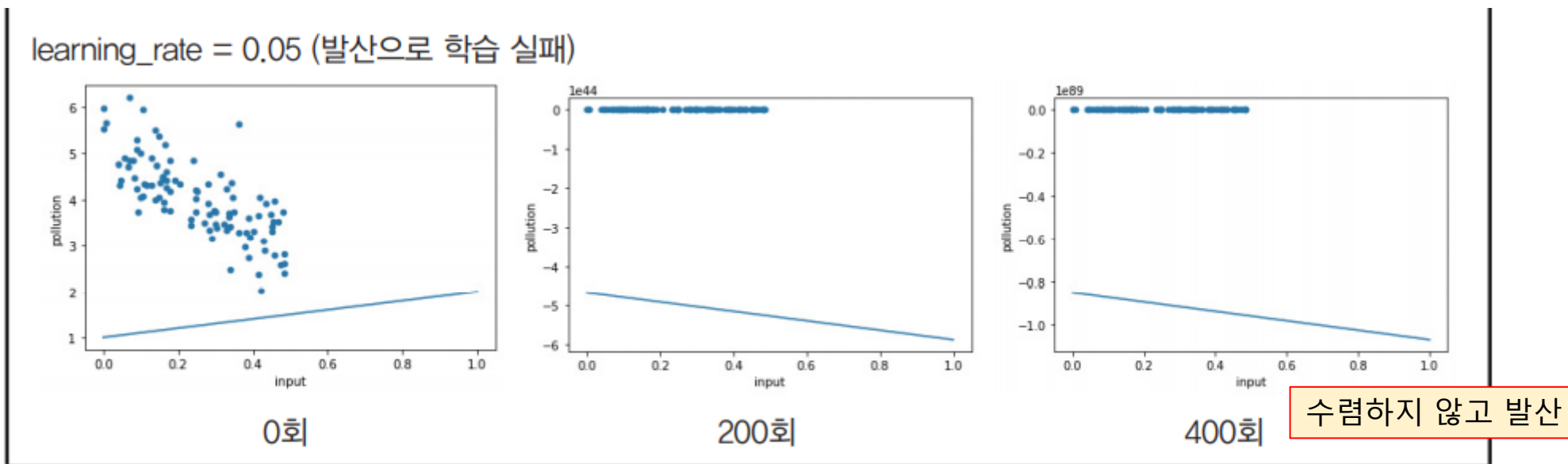
...

오차가 점점 줄어들면 회귀 직선이 데이터를 점점 더 정확하게 모델링

- learning\_iteration과 learning\_rate가 변경되면 학습으로 얻는 모델의 파라미터가 달라질 것
  - 학습을 제어하는 변수 : 하이퍼파라미터 hyperparameter 라고 함



- learning\_iteration과 learning\_rate가 변경되면 학습으로 얻는 모델의 파라미터가 달라질 것
  - 학습을 제어하는 변수 : 하이퍼파라미터 hyperparameter 라고 함



## 2.9 scikit-learn을 이용한 선형 회귀

- scikit-learn 라이브러리를 활용하여 선형 회귀를 구현
  - 사이킷런<sup>scikit-learn</sup>: 선형 회귀, k-NN 알고리즘, 서포트 벡터머신, k-means 등 다양한 머신러닝 알고리즘을 쉽게 구현할 수 있게 해줌



```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import linear_model  # scikit-learn 모듈을 가져온다

data_home = 'https://github.com/dknife/ML/raw/main/data/'
lin_data = pd.read_csv(data_home+'pollution.csv')
```

## 2.9 scikit-learn을 이용한 선형 회귀

- scikit-learn의 선형 회귀 모델의 입력 데이터는 2차원 배열로, 각 행이 데이터 인스턴스이며, 각 데이터 인스턴스가 여러 개의 특징값을 가질 수 있음
- 현재 우리는 하나의 특징값만을 사용하지만 이 경우에도 하나의 원소를 가진 벡터로 제공해야 함.



```
x = lin_data['input'].to_numpy()
y = lin_data['pollution'].to_numpy()
x = x[:, np.newaxis]      # 선형 회귀 모델의 입력형식에 맞게 차원을 증가시킴
print(x)
```

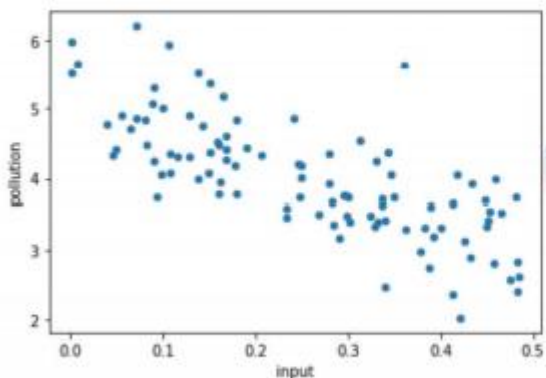
```
[[0.24055707]
 [0.1597306 ]
 ...
 [0.00720486]
 [0.29029368]]
```

## 2.9 scikit-learn을 이용한 선형 회귀

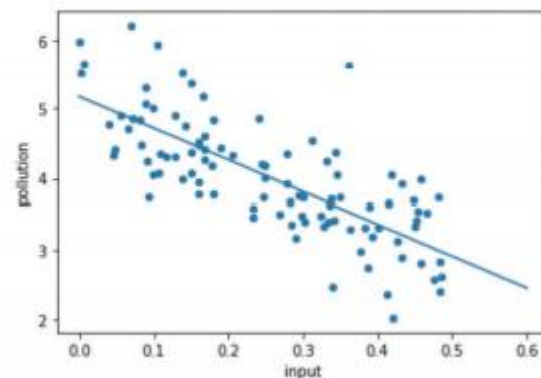
- 선형회귀 모델과 fit() 메소드의 역할?



```
regr = linear_model.LinearRegression()  
regr.fit(x, y) # 선형 회귀 모델에 데이터를 넣어 학습을 진행함
```



데이터를 기반으로 최적의  
선형회귀 모델 생성



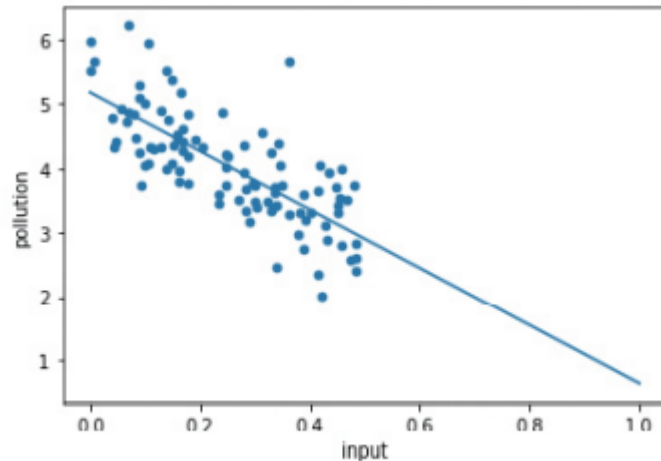
이전의 수렴 과정을 모  
두 자동화하는 메소드가  
바로 fit()

## 2.9 scikit-learn을 이용한 선형 회귀

- 입력으로 0과 1을 주고 이에 해당하는 출력값을 예측하도록 하면 됨

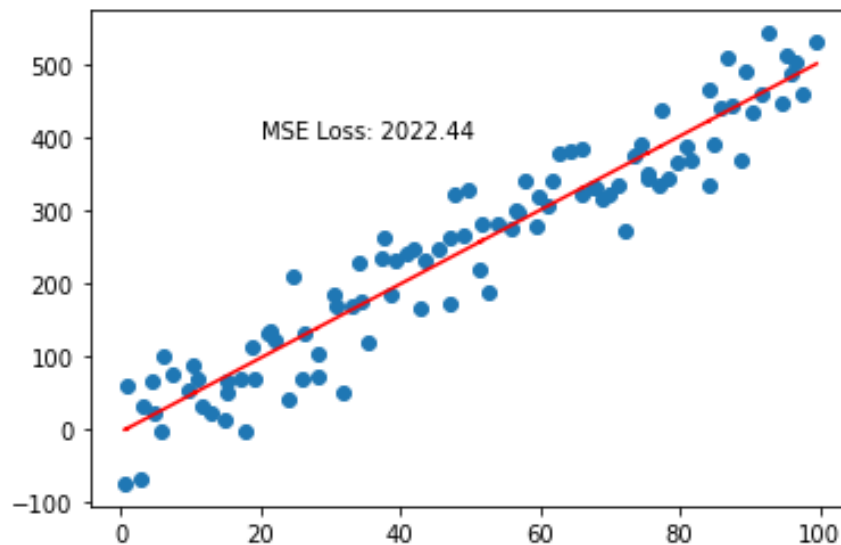


```
lin_data.plot(kind = 'scatter', x = 'input', y = 'pollution')  
y_pred = regr.predict([[0], [1]])  
plt.plot([0, 1], y_pred) # x 구간을 0에서 1 사이로 두자
```



# 실습문제

- 주어진 데이터셋(dataset1.csv)를 불러와서 아래 함수들을 정의하고, 평균 제곱오차(MSE)와 그래프를 그리세요.
  - read\_csv()함수를 이용하요 데이터 불러오기
  - $y = 5x + 50 \cdot N(0,1)$  로 y 데이터 생성
  - $E_{MSE} = \frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2$  로 평균제곱오차(MSE) 함수 정의
  - w와 b의 초기값은 0.5



감사합니다