

# 인공지능

## 03. 선형 회귀

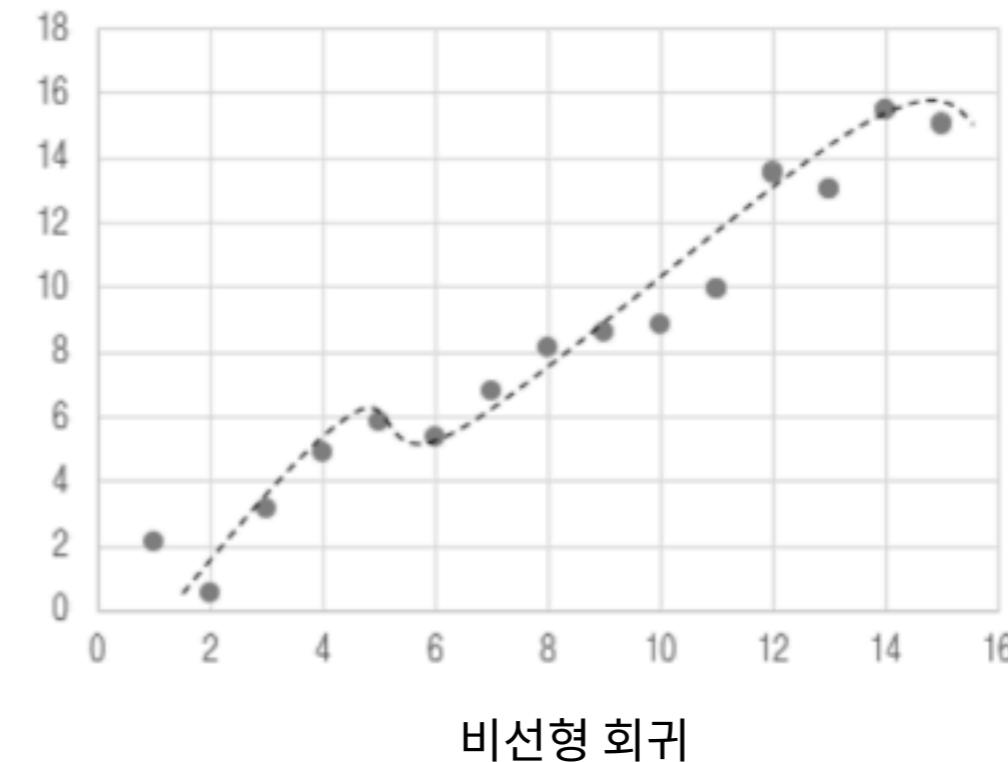
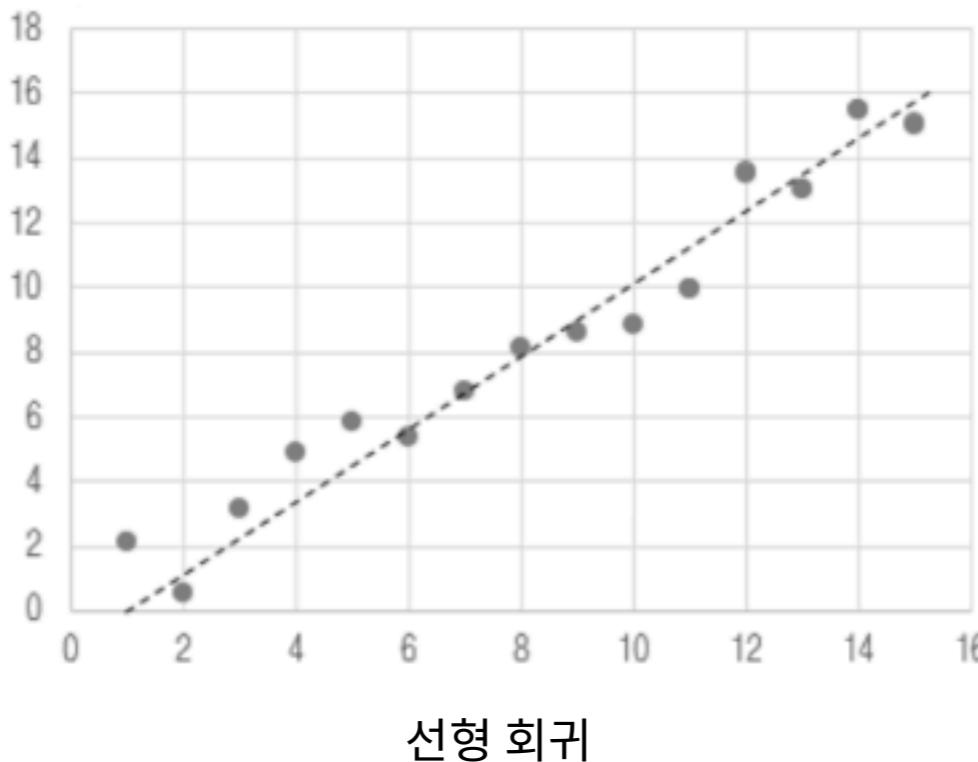
Dept. of Digital Contents



# 1. 선형 회귀

## ○ 회귀(Regression)

- 데이터 추세(상관관계)를 예측하는 학습 방법
- $Y = f(X)$ 에 대하여 입력 변수( $X$ )와 출력 변수( $Y$ )의 관계를 모델링



# 1. 선형 회귀

---

## ○ 선형 회귀(Linear Regression)

- 직선의 방정식으로 모델링

$$f(x) = mx + b$$

- $m$  : 기울기(slope),  $b$  : 절편(intercept)

- $\rightarrow$  머신 러닝 ::  $m$  : 가중치(weight),  $b$  : 바이어스(bias)

- 선형 회귀의 종류

- 단순 선형 회귀  $f(x) = wx + b$

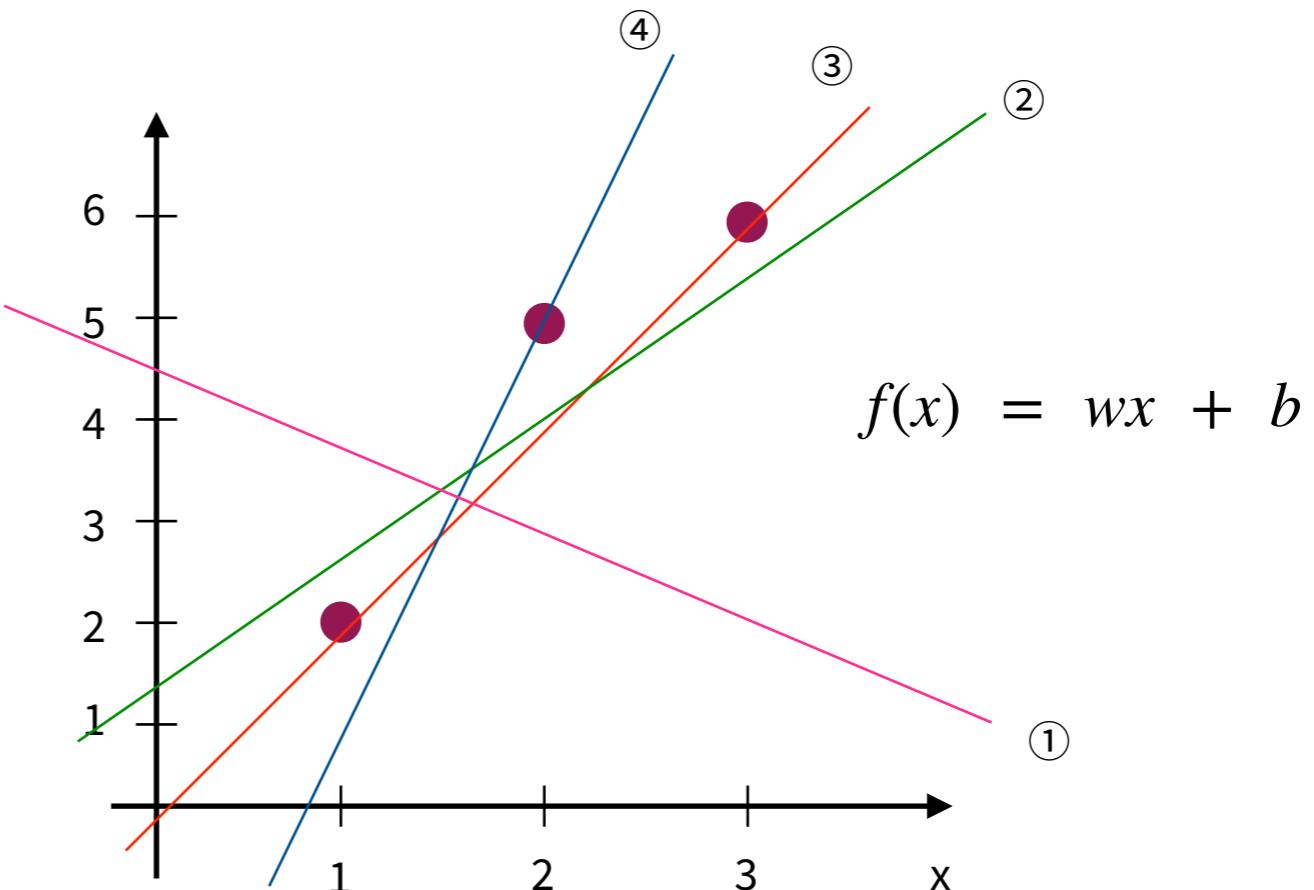
- 다중 선형 회귀  $f(x_1, x_2, x_3) = w_0 + w_1x_1 + w_2x_2 + w_3x_3$

# 1. 선형 회귀

## ○ 선형 회귀의 원리

- 간단 데이터

x	y
1	2
2	5
3	6

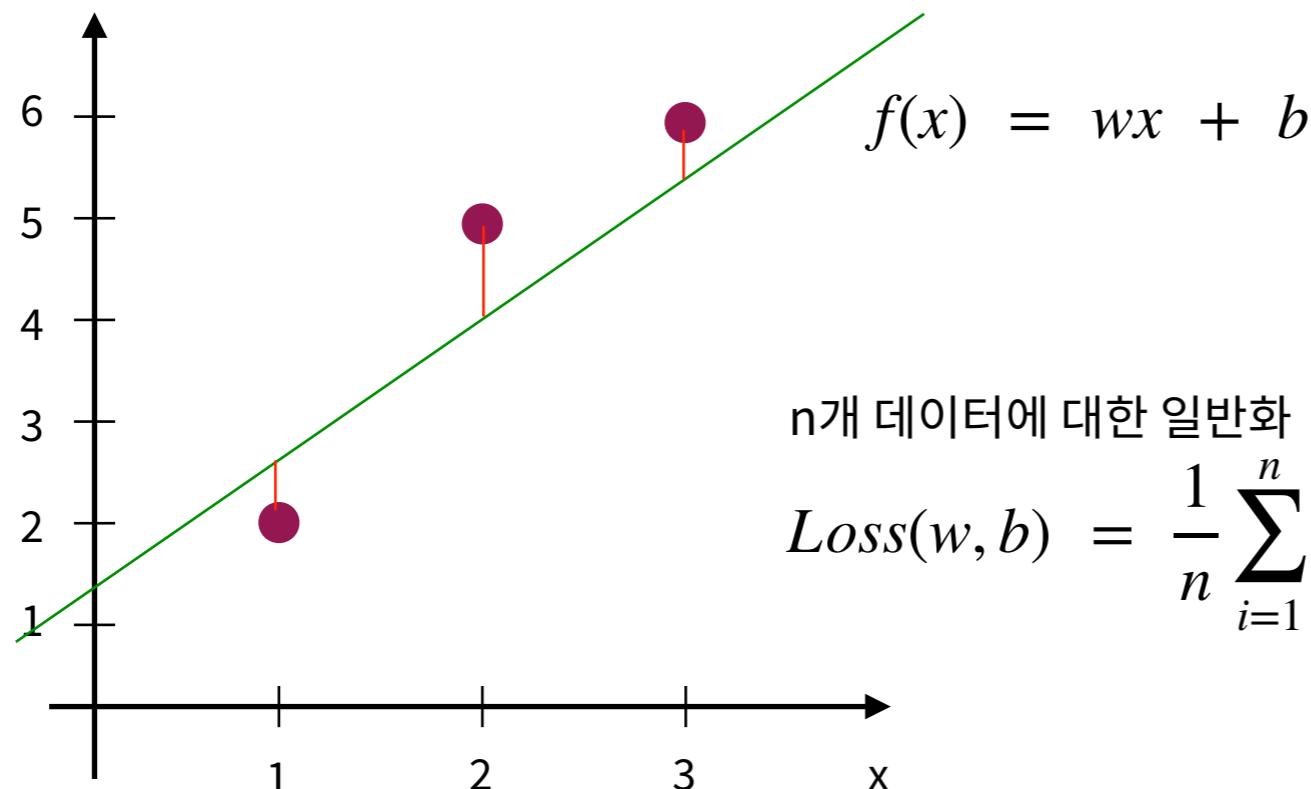


# 1. 선형 회귀

## ○ 선형 회귀의 원리

### ■ 손실 함수(Loss function)

$$Loss = \frac{1}{3} ((f(x_1) - y_1)^2 + (f(x_2) - y_2)^2 + (f(x_3) - y_3)^2)$$



# 1. 선형 회귀

---

## ○ 선형 회귀의 원리

### ■ 학습과 손실

- 손실 : 특정 샘플에서 모델의 예측이 잘못된 크기를 나타냄
- 훈련 : 손실함수를 최소화 시키는  $w, b$  값을 찾는 것

$$\underset{w,b}{\operatorname{argmin}} \ Loss(w,b)$$

## 2. 선형 회귀에서 손실함수 최소화 방법

---

### ○ 분석적인 방법

- 변수가 하나인 선형 회귀에서 사용(단순)

- 최소 제곱법(Ordinary Least Square)

- 특징이 1개이고 1차항인 선형 회귀 모델:  $y = b + wx$

- i 번째 데이터:  $x_i, y_i$

- $\hat{y}_i$  :  $x_i$ 의 추정치(예측값),  $\hat{y}_i = \hat{b} + \hat{w}x_i$

- 오차:  $e_i = y_i - \hat{y}_i$

- 손실함수:  $J(b, w) = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - \hat{b} - \hat{w}x_i)^2$

- $y$ 의 평균:  $\bar{y} = \frac{\sum_{i=1}^n y_i}{n}$        $\frac{\partial J}{\partial b} = 0 \rightarrow b = \bar{y} - w\bar{x}$

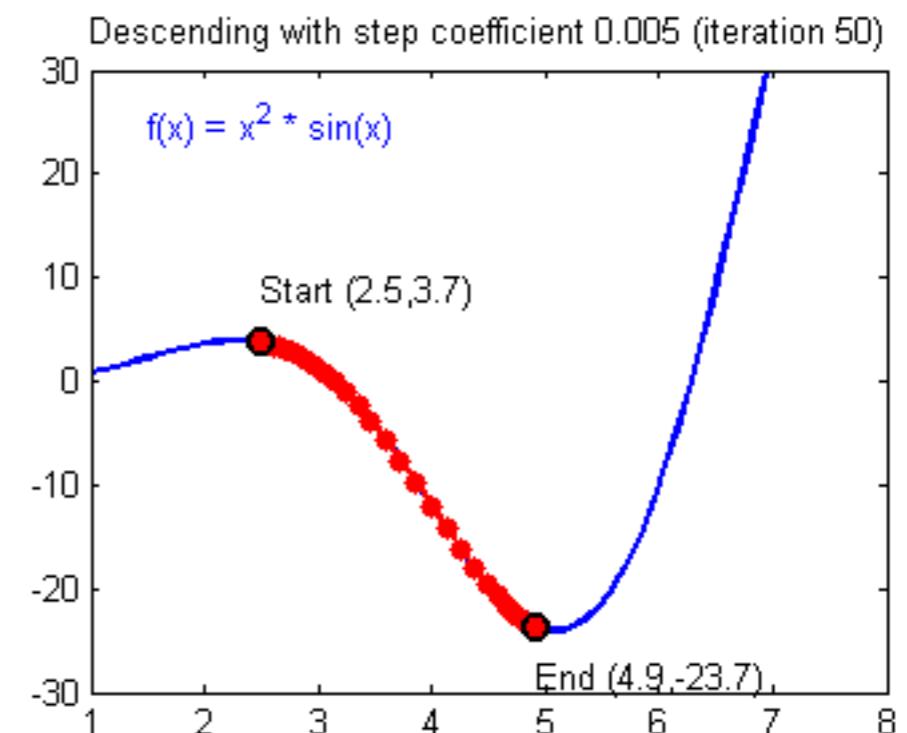
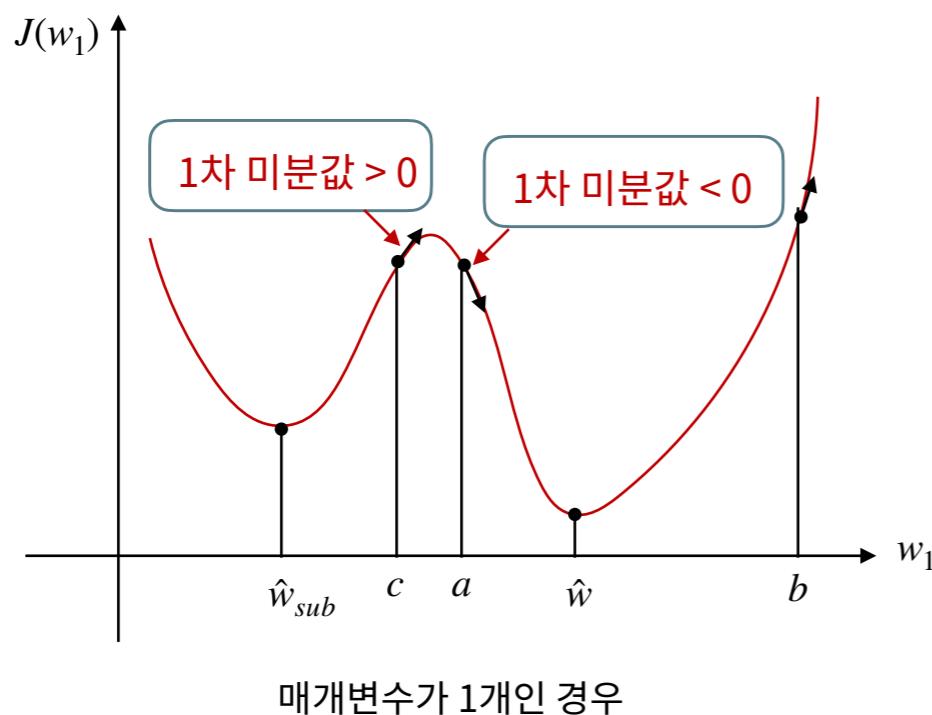
- $x$ 의 평균:  $\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$        $\frac{\partial J}{\partial w} = 0 \rightarrow w = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$

## 2. 선형 회귀에서 손실함수 최소화 방법

### ○ 경사하강법(Gradient Descent Method)

- 손실함수 최소값을 찾기 위해 1차 미분 계수(현재 위치에서의 기울기)를 이용
- 반복적으로 최소값 방향으로 이동 → 학습률이 step size가 됨

$$w_1 = w_1 + \rho \left( -\frac{\partial J}{\partial w_1} \right) \quad \rho : \text{학습률(learning rate)}$$



## 2. 선형 회귀에서 손실함수 최소화 방법

---

### ○ 경사하강법(Gradient Descent Method)

#### ■ 손실 함수

$$Loss(w, b) = \frac{1}{n} \sum_{i=1}^n (f(x_i) - y_i)^2 = \frac{1}{n} \sum_{i=1}^n ((wx_i + b) - y_i)^2$$

$$\frac{\partial Loss(w, b)}{\partial w} = \frac{1}{n} \sum_{i=1}^n 2((wx_i + b) - y_i)(x_i) = \frac{2}{n} \sum_{i=1}^n x_i((wx_i + b) - y_i)$$

$$\frac{\partial Loss(w, b)}{\partial b} = \frac{1}{n} \sum_{i=1}^n 2((wx_i + b) - y_i)(1) = \frac{2}{n} \sum_{i=1}^n ((wx_i + b) - y_i)$$

## 2. 선형 회귀 구현

### ○ 파이썬 구현 1

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Linear\_Regression.ipynb
- Toolbar:** 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말, 모든 변경사항이 저장됨
- Code Cell:** The code cell contains the following Python code:

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

X = np.array([0.0, 1.0, 2.0])
y = np.array([3.0, 3.5, 5.5])

w = 0      # 기울기
b = 0      # 절편

lrate = 0.01    # 학습률
epochs = 1000   # 반복 횟수

n = float(len(X))    # 입력 데이터 개수
```

## 2. 선형 회귀 구현

### ○ 파이썬 구현 1

```
# 경사하강법
for i in range(epochs):
    y_pred = w*X + b
    dw = (2/n) * sum(X * (y_pred - y))
    db = (2/n) * sum(y_pred - y)
    w = w - lrate * dw
    b = b - lrate * db

# 기울기와 절편 출력
print(w, b)

# 예측
y_pred = w * X + b

# 그래프
plt.scatter(X, y)

# 회귀선
plt.plot([min(X), max(X)], [min(y_pred), max(y_pred)], color='red')
plt.show()
```

## 2. 선형 회귀 구현

### ○ 파이썬 구현 2

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Linear\_Regression2.ipynb
- Toolbar:** 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말, 모든 변경사항이 저장됨
- Code Cell:** The cell contains the following Python code:

```
import matplotlib.pyplot as plt
from sklearn import linear_model

# 선형 회귀 모델 생성
reg = linear_model.LinearRegression()

X = [[0], [1], [2]]
y = [3, 3.5, 5.5]

# 학습
reg.fit(X, y)

# 학습 결과 출력
print("기울기: ", reg.coef_)
print("절편: ", reg.intercept_)
print("Score: ", reg.score(X,y))
```

## 2. 선형 회귀 구현

---

### ○ 파이썬 구현 2

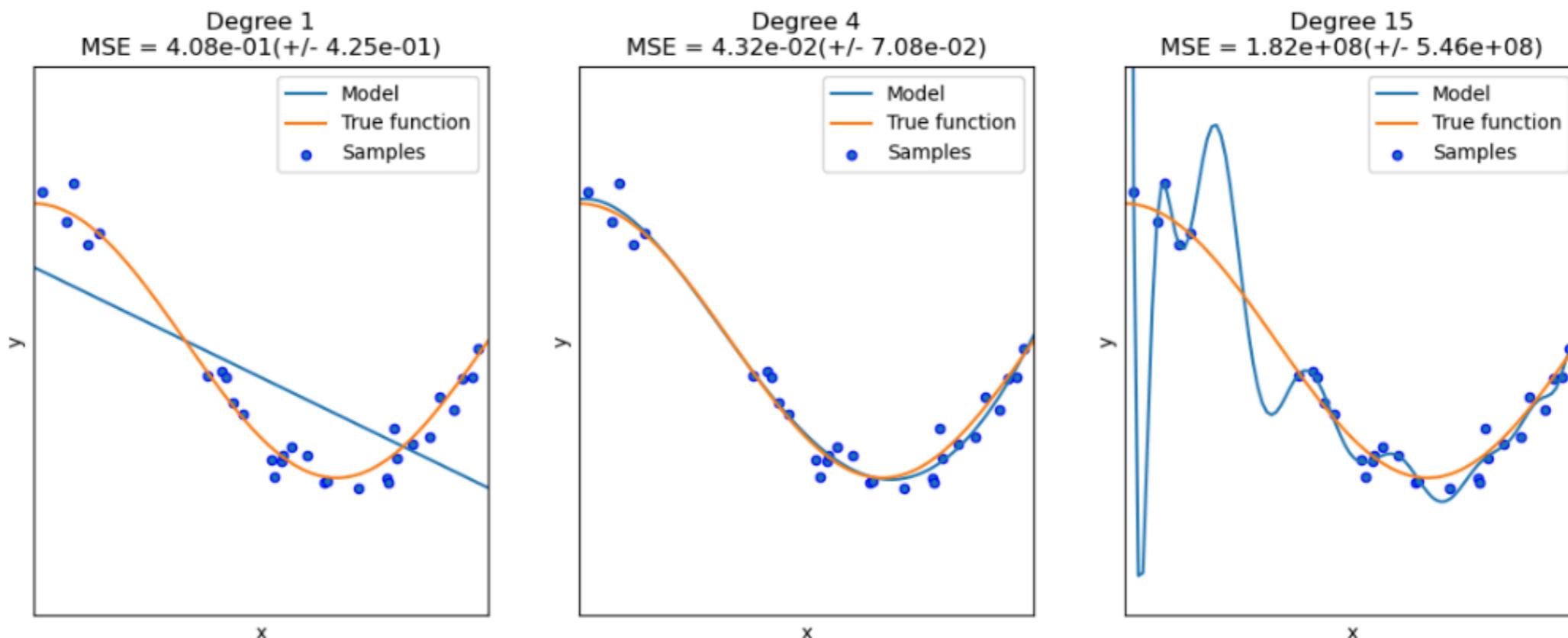
```
# 예측
print(reg.predict([[5]]))

# 그래프
plt.scatter(X, y, color='black')
y_pred = reg.predict(X)
plt.plot(X, y_pred, color='blue', linewidth=3)
plt.show()
```

# 3. 과잉 적합 vs. 과소 적합

## ○ 과잉 적합과 과소 적합

- 과잉 적합(overfitting) : 훈련(학습) 데이터에서는 성능이 뛰어나지만 새로운 데이터(일반화)에 대해서는 성능이 잘 나오지 않는 현상
- 과소 적합(underfitting) : 훈련 데이터의 규칙을 찾지 못해 모델의 성능이 낮게 나오는 현상



### 3. 과잉 적합 vs. 과소 적합

---

#### ○ 과잉 적합의 발생원인

- 훈련 데이터가 충분하지 못한 경우
- 훈련 데이터에 노이즈가 심한 경우
- 모델의 복잡도가 큰 경우
- 과도하게 큰 epoch로 학습하는 경우

#### ○ 과소 적합의 발생원인

- 모델의 복잡도가 낮은 경우
- 모델에 너무 많은 규제가 적용된 경우
- 충분하지 못한 epoch로 학습하는 경우

### 3. 과잉 적합 vs. 과소 적합

---

#### ○ 과잉적합 해결 방법

- 훈련 데이터가 충분하지 못한 경우 → 데이터 증대(data augmentation)
- 훈련 데이터에 노이즈가 심한 경우 → 데이터 전처리(preprocessing)
- 모델의 복잡도가 큰 경우 → 규제(regulation)를 이용하여 복잡도를 낮춤
- 과도하게 큰 epoch로 학습하는 경우 → 조기종료(early-stopping)

#### ○ 과소적합 해결 방법

- 모델의 복잡도가 낮은 경우 → 모델의 복잡도 증대
- 모델에 너무 많은 규제가 적용된 경우 → 규제 완화
- 충분하지 못한 epoch로 학습하는 경우 → epoch 수 키우기

# 4. BMI와 혈당 관계 분석

## ○ sklearn의 diabetes 데이터

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** BMI.ipynb
- Toolbar:** 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말, 모든 변경사항이 저장됨
- Code Cells:** + 코드 + 텍스트
- Code Content:** The code imports matplotlib, numpy, and LinearRegression from sklearn, and datasets from sklearn. It then loads the diabetes dataset, prints its shape, creates a new feature array with only the BMI column, and splits the data into training and testing sets.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import datasets

# 당뇨병 데이터 세트 가져오기
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

print(diabetes_X.data.shape)

# BMI 특징만 따로 떼기
diabetes_X_new = diabetes_X[:, np.newaxis, 2]

# 학습 데이터(90%)와 테스트 데이터(10%) 분리
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    diabetes_X_new, diabetes_y, test_size=0.1, random_state=0)
```

## 4. BMI와 혈당 관계 분석

---

### ○ sklearn의 diabetes 데이터

```
# 학습
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)

# 예측
y_pred = regr.predict(X_test)

# 그래프
plt.scatter(X_test, y_test, color='black')
plt.plot(X_test, y_pred, color='blue', linewidth=3)
plt.show()
```