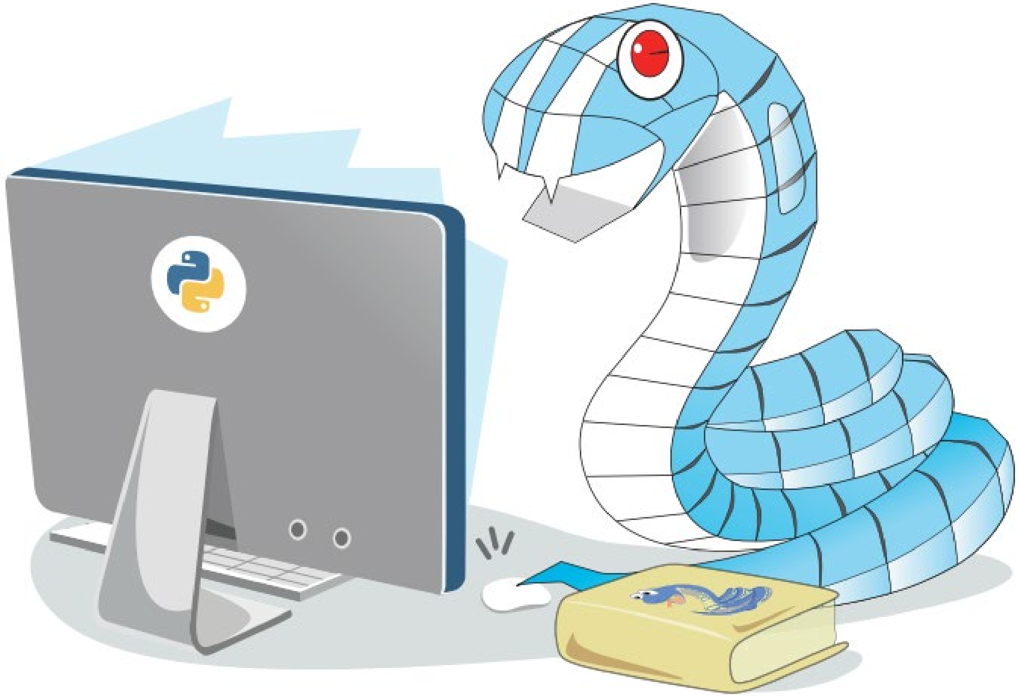


4장 선형 회귀로 이해하는 지도학습

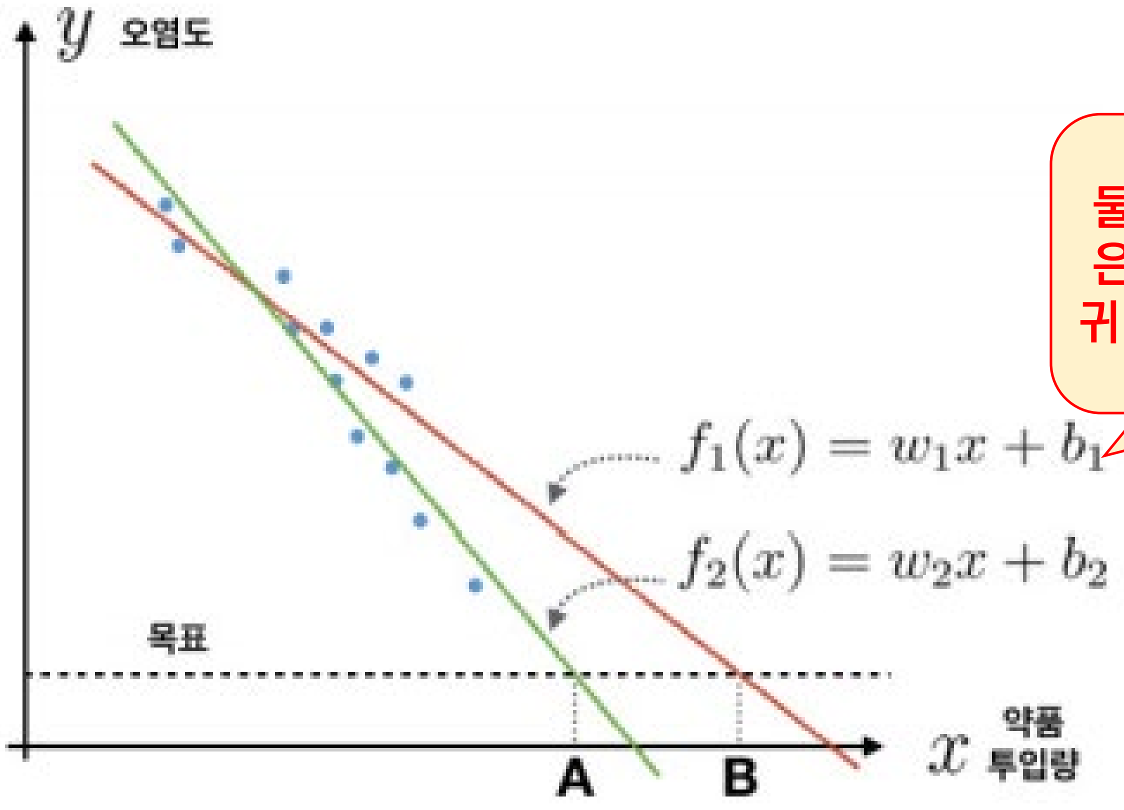
이장에서 배울 것들

- 회귀 분석이란 무엇인가.
- 회귀 분석은 머신러닝의 범주에 들어가는가.
- 회귀 분석은 어떻게 동작하며 구현 방법은 어떤 것이 있는가.
- 학습에 사용되는 데이터는 어떻게 정제할 수 있는가.

4.1 회귀 모델

- 회귀regression : 회귀 분석은 대표적인 지도학습 알고리즘
  - 관측된 데이터를 통해 독립변수와 종속변수 사이의 숨어있는 관계를 추정하는 것이다
- 선형 회귀linear regression는 $y = f(x)$에서 입력 x에 대응되는 실수 y들이 주어지고 추정치 $f(x)$가 가진 오차를 측정
  - 이 오차를 줄이는 방향으로 함수의 계수를 최적화하는 일
- 이때 작업 T는 독립변수에 대응하는 종속변수를 추정하는 일이며, 주어진 데이터가 경험 E에 해당함
  - 성능 척도 P는 예측한 값 $\hat{y}$과 데이터로 제공되는 목표값 y의 차이가 적을수록 높은 점수를 부여함

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

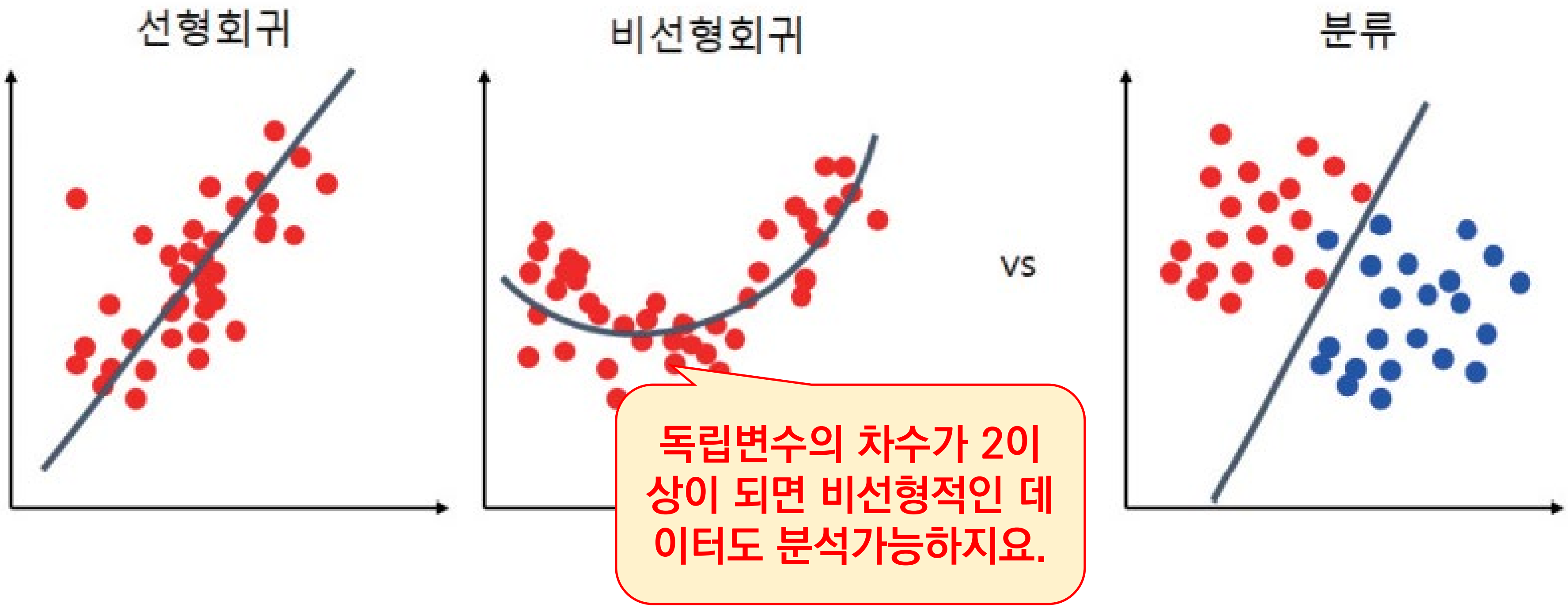


둘 중에서 오차가 더 작은 가설을 찾는 일이 회귀 분석이 하는 일이지요.

4.2 선형 회귀와 지도학습

- 데이터에 제시된 목표값을 정답값 혹은 **레이블**label이라고 부름
  - 지도 학습은 주어진 입력-출력 쌍을 학습한 후에 새로운 입력값이 들어왔을 때, 합리적인 출력값을 예측하는 것.
- 전통적 프로그래밍 : 사람이 이 함수 $f(x)$를 고안하여 구현한 뒤, 입력 x를 넣어 답 y를 얻는 것
- 기계학습: 데이터 $(x,\ y)$를 주면 함수 $f(x)$를 만들어내는 일

- 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
- 회귀 분석 : 데이터를 설명하는 직선을 찾는 선형linear 회귀와 곡선을 찾는 비선형 nonlinear 회귀
  - 이진 분류binary classification는 데이터를 양분하는 경계 직선 혹은 곡선을 찾는 것.



- 데이터를 학습시킬 때, 데이터에서 중요한 일부 정보만을 추출하여 이것으로 학습시키고 테스트할 수도 있음
  - 특징feature : 특징이란 관찰되는 현상에서 측정할 수 있는 개별적인 속성attribute을 의미
  - $y = f(x)$의 함수를 찾는다고 할 때, 입력 데이터로 사용되는 x가 바로 특징이다.
- 다음은 기계학습에서 다룰 수 있는 특징의 예
  - 사람의 키와 몸무게
  - 개의 몸통 길이와 높이
  - 주택 가격에 영향을 주는 주택의 특징들

특징에 대한 분석이 선행되어야 기계학습이 효과적으로 이루어 집니다.

4.3 실제 데이터를 읽고 가설 만들어 보기

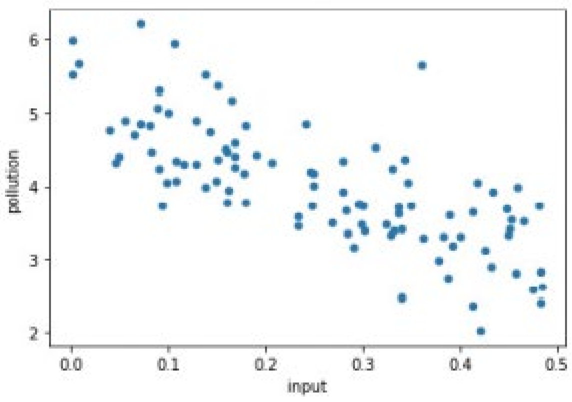
- 아래와 같이 판다스를 이용하여 CSV 파일를 읽어 lin_data라는 데이터프레임을 만듬
- 투입량에 따른 오염도 측정 결과 100건이 담겨 있음
  - 데이터프레임의 input 열은 오염도를 줄이기 위한 약품의 투입량
  - pollution 열은 실체 측정된 오염도 수치

```python
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]
```

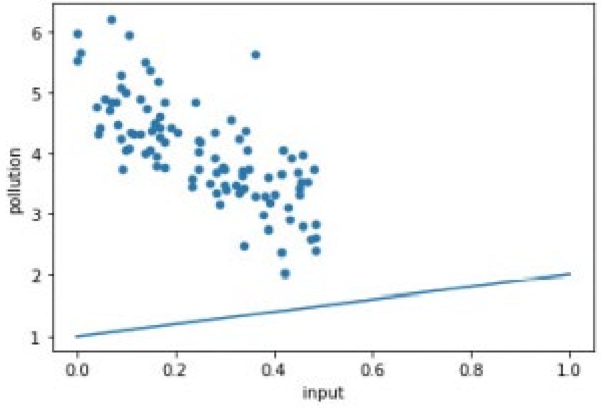
- 데이터를 시각적으로 확인하기 위하여 plot() 메소드를 사용.
    - x축으로는 input 열을 사용
    - y축으로 pollution 열을 사용
    - 투입량을 늘이면 오염도가 줄어드는 경향이 있는 것을 확인할 수 있음

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



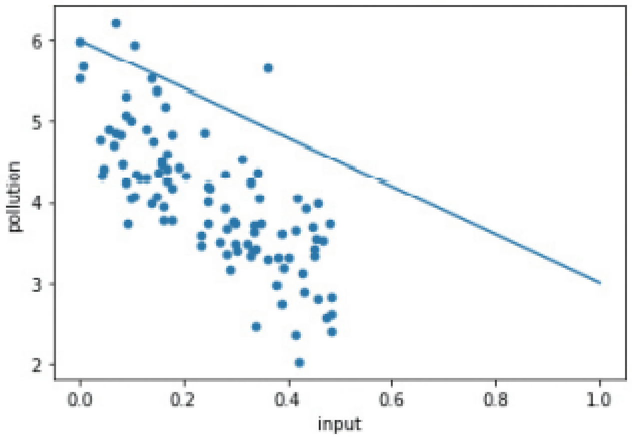
- 두 변수들이 input을 독립변수 x로, pollution을 종속변수 y로 하는 $y = wx + b$라는 직선으로 표현하면, 데이터가 이 함수를 따를 것이라는 가설hypothesis을 제시한 것
  - pyplot의 plot() 함수는 독립변수의 리스트와 종속변수의 리스트를 주면, 이들을 연결한 선을 그려낸다.

```
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?

```python
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)])
```



4.4 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

- 대표적인 오차 척도는 평균 제곱 오차
- 예측치 $\hat{y}$와 정답 레이블 y사이의 차이를 제곱하여 모두 더한 뒤에 전체 데이터의 개수 m으로 나누는 것
  - 평균 제곱 오차[mean square error:MSE]라고 하며, 다음과 같은 식

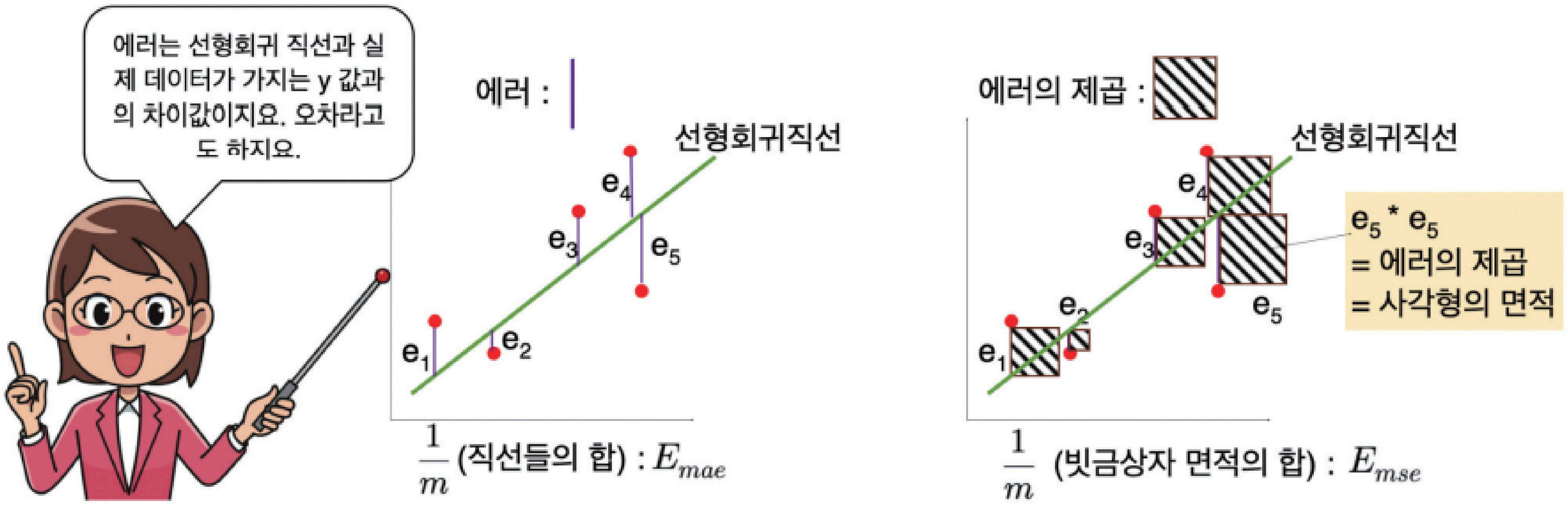
$$E_{mse} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

- 예측 결과가 y_hat, 정답 레이블이 y에 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
```

```
0.06099999999999996
```

- 오차를 제곱하는 이유
  - 빨간색 점으로 표시된 레이블과 가설 사이에 차이가 있음
  - e_1에서 e_5까지 전체 에러의 합이 최소가 되는 모델이 바람직한 모델
  - 오차합 곡면(뒤에 배웁니다)의 기울기를 따라 내려가 최소 오차에 접근하기 위하여



- 평균 제곱 오차는 기계학습에서 가장 흔히 사용되는 오차 척도
- 기계학습의 대표적인 패키지 중 하나인 사이킷런 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: 0.06099999999999996
```

- 제곱을 하지 않고 오차를 더하고 싶다면 평균 절대 오차mean absolute error:MAE라는 것을 사용할 수 있음
  - 오차를 제곱하지 않고 절대값을 취해 더하는 것

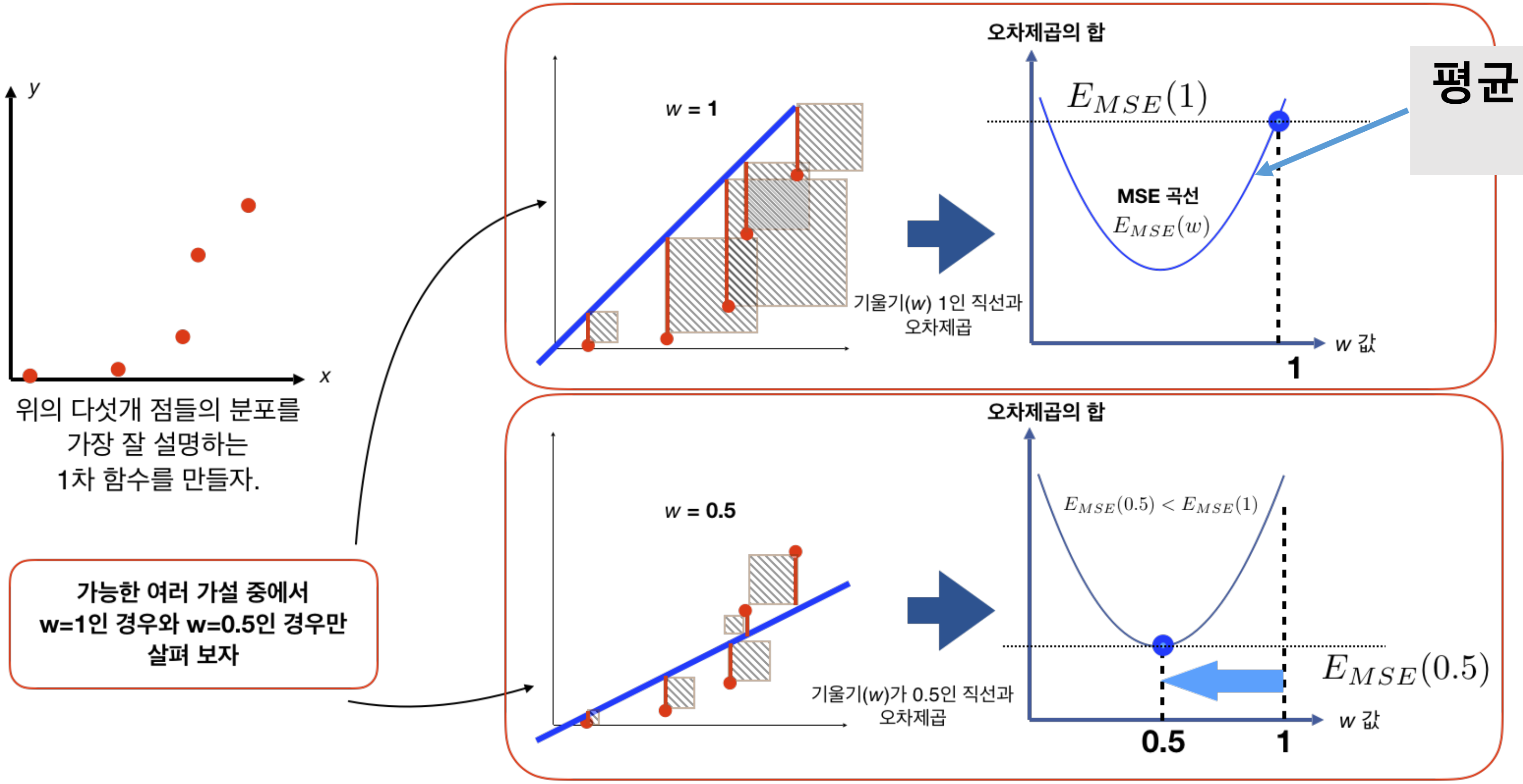
$$E_{mae} = \frac{1}{m}\sum_{i=1}^{m}|\hat{y}_i - y_i|$$

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

```
Mean absolute error: 0.2099999999999988
```

4.5 데이터의 관계를 설명하는 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 y절편이 0인 경우를 가정하고 최적의 w값을 찾는 과정



평균 제곱 오차 곡선

오차 함수로 평균 제곱 오차를 널리 사용한답니다.

- 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 = 1.0으로 하는 $\hat{y}$에서 w = 0.3인 $\hat{y}$까지 0.1씩 값을 감소시켜가며 w 와 평균제곱오차를 출력해 보도록 하자

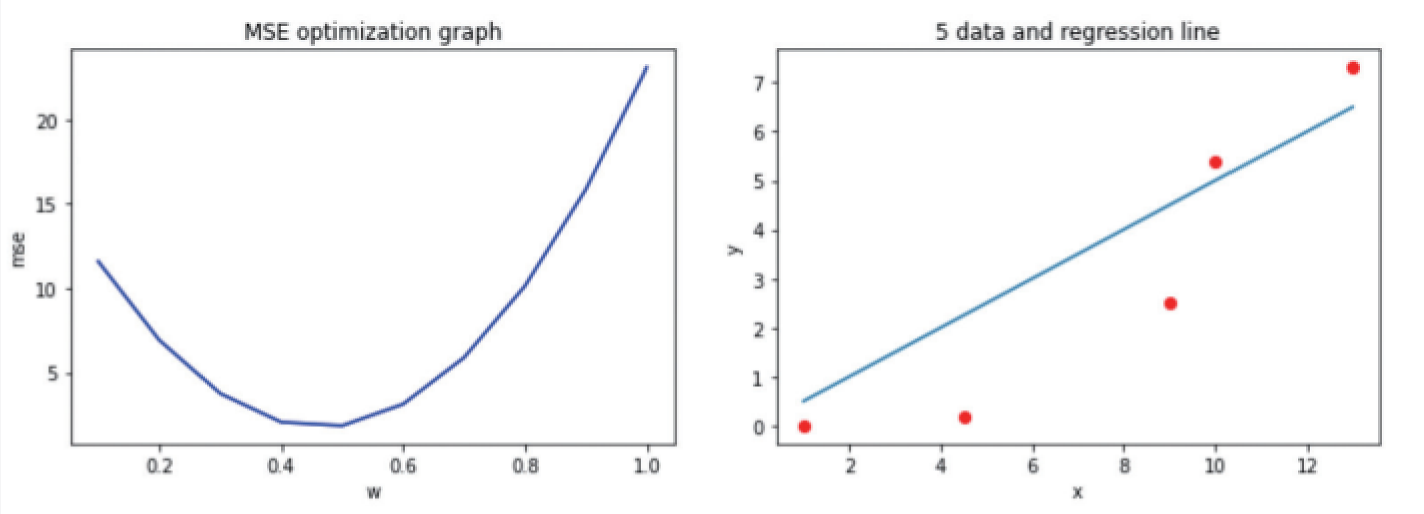
```python
import numpy as np
from sklearn.metrics import mean_squared_error as mse

# 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
```

w와 오차값을 그림으로 그려보면 이전 페이지의 그림이 되겠지요.

도전문제 4.1

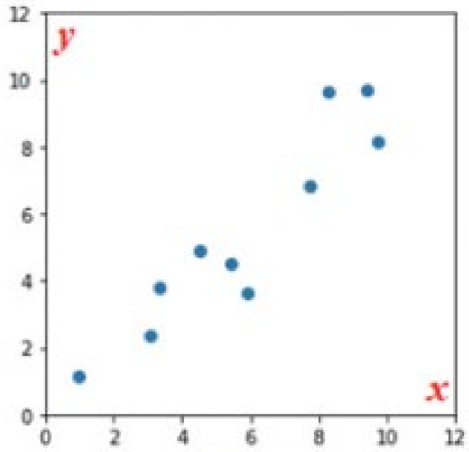
1. w를 1.0에서 0.0까지 0.1씩 감소시켜가며 평균 제곱 오차값의 변화를 그래프로 그리도록 하자.
2. 평균 제곱 오차값이 가장 작을 때의 w를 기울기로 가지는 직선을 주어진 점과 함께 그려보도록 하자.



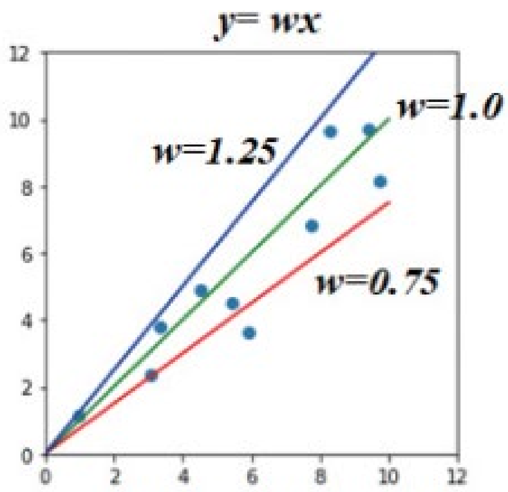
matplotlib을 이용해서 그려보세요
matplotlib.org 사이트에 그래프를 그리는 기능에 대한 상세한 설명이 나옵니다.

4.6 오차의 종류에 따른 오차 곡면의 모습

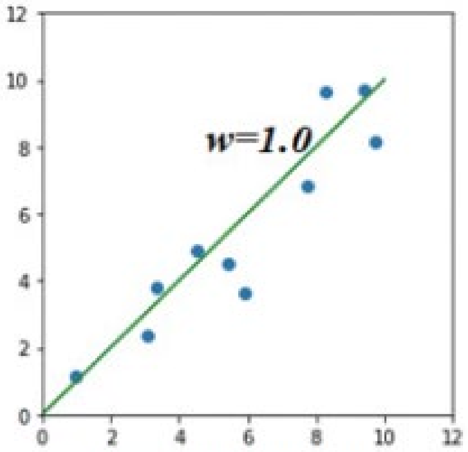
- 모델의 오차를 계산하는 방법에 따른 오차 곡면의 모습을 살펴보고, 이 오차 곡면을 이용하여 최적의 모델을 찾는 방법에 대해 살펴보도록 하자
- 목표: 데이터 x와 y의 관계를 가장 잘 설명하는 모델 $y = wx$를 찾는 것



데이터



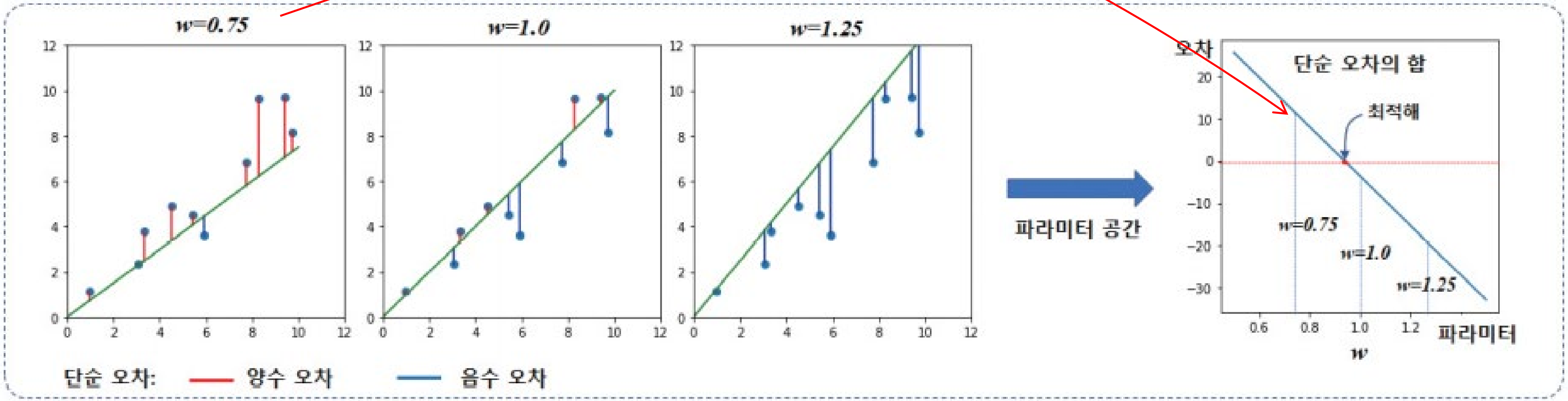
데이터를 설명하는 모델



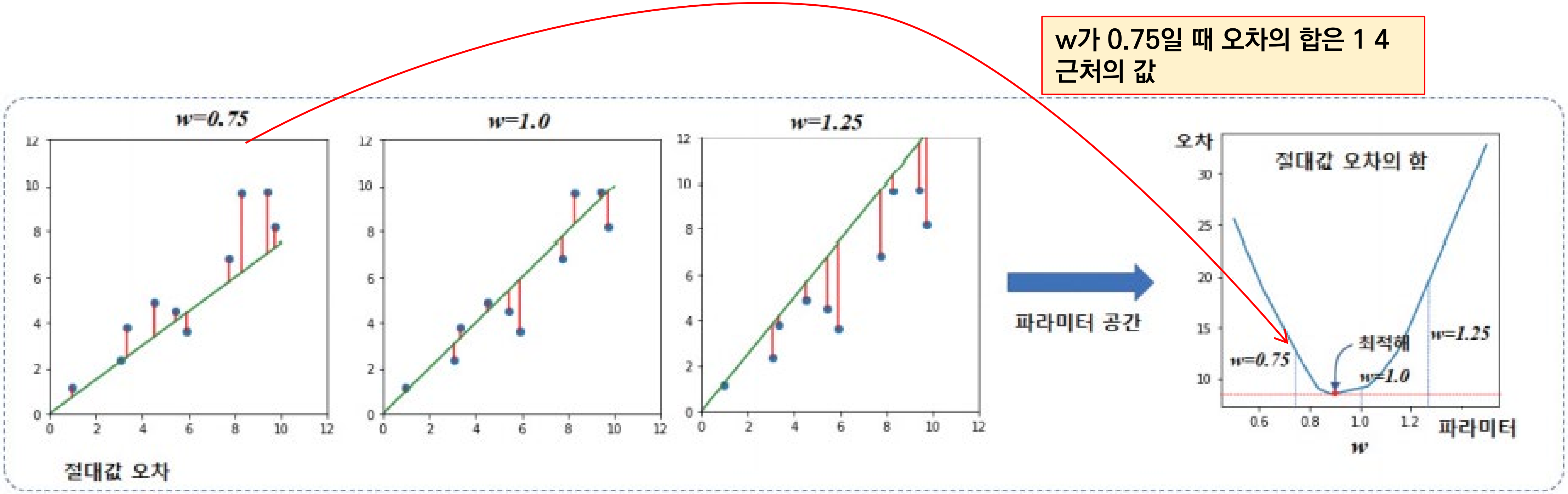
좋은 모델의 파라미터 w는?

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

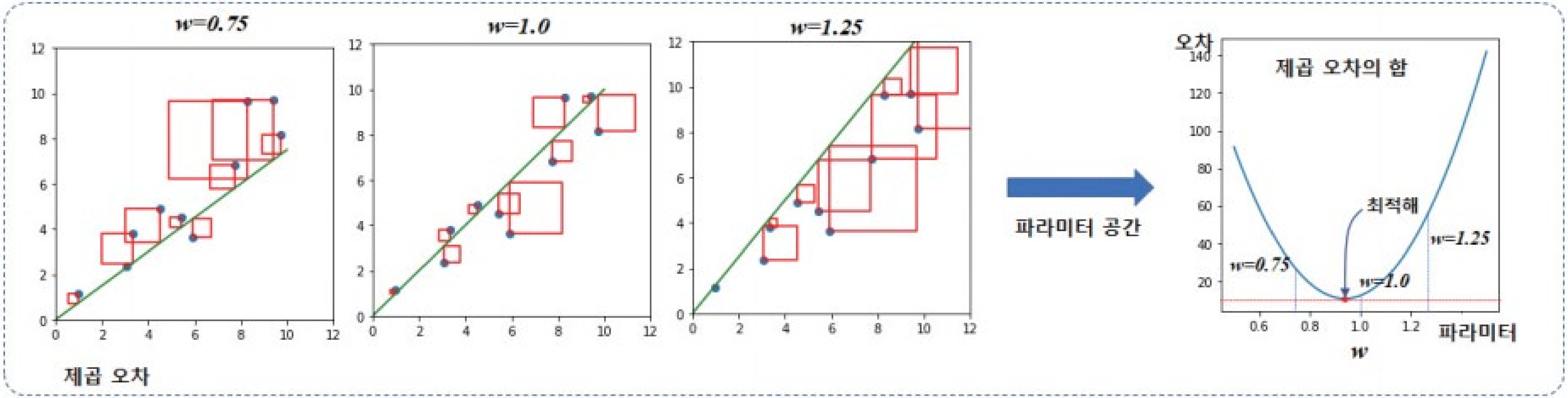
w가 0.75일 때 오차의 합은 10 근처의 값



- **평균 절대 오차**MAE **정답과 예측치의 차이를 절대값으로 구해 평균값을 구하는 것**
- **교차지점을 구할 필요없이 오차 곡선의 극소만 구하면 된다.**



• 정답과 예측의 차이를 제곱하는 평균 제곱 오차MSE



4.7 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 오차error라고 부른다.
  - 추정치는 일반적으로 종속변수 y의 위에 ∧기호를 씌운 $\hat{y}$ ('^'기호는 햇으로 읽는다)로 표기
  - 오차 E는 다음과 같은 식으로 계산할 수 있다.

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

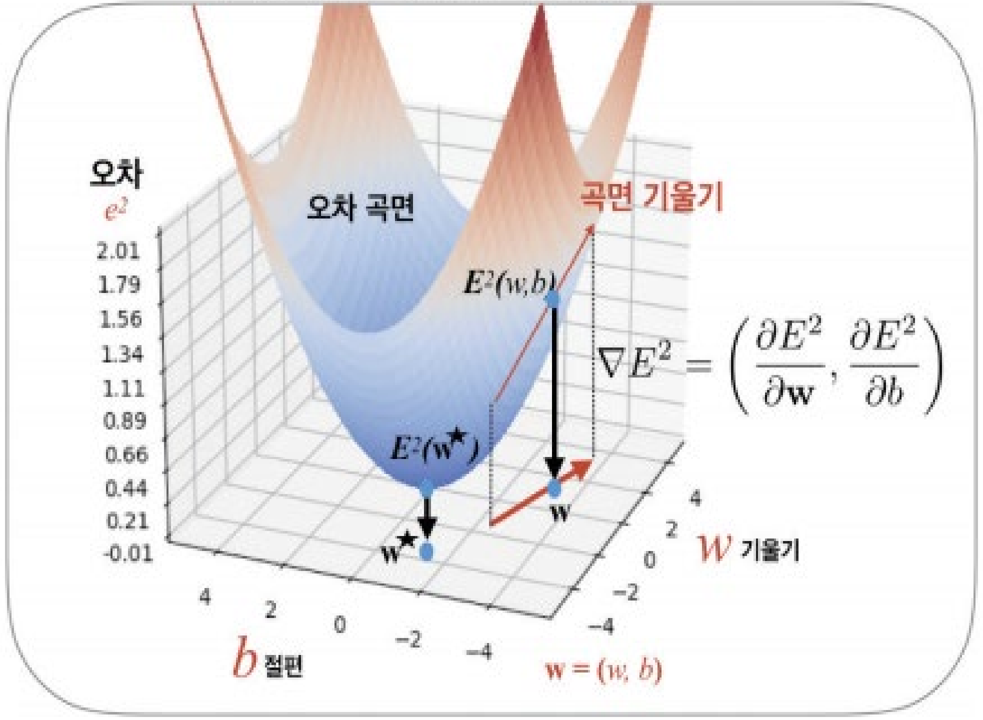
- **최소 제곱법**least squares approximation은 오차를 제곱하여 오차 곡면의 기울기를 따라 내려가 기울기가 0인 극소 지점을 찾는 것
- 오차의 제곱 $E^2(w, b)$이 그림과 같은 곡면이라면, 최적의 w와 b를 찾기 위한 오차 곡면의 기울기 방향은 다음과 같이 구할 수 있음

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

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

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

최적의 기울기와 절편은 어디에 있나?



- (w, b)벡터를 $-(E_x, E)$방향으로 옮겨주면 최적의 w와 b에 가까워질 것
- 아래 코드는 "조금"의 정도를 learning_rate라는 하이퍼파라미터로 제어하고 있다. n개의 데이터 x_i에 대한 예측 오차가 E_i라고 할 때 다음과 같이 기울기 w와 절편 b를 수정하는 것

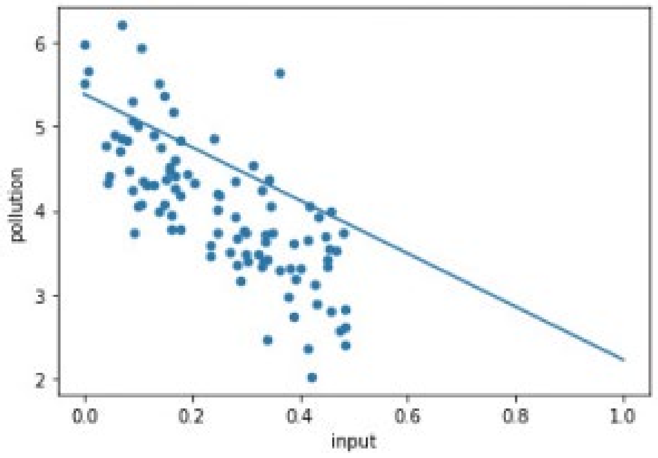
$$w \leftarrow w - \eta \sum_{i=1}^{n} E_i x_i, \quad b \leftarrow b - \eta \sum_{i=1}^{n} E_i$$

- 벡터화 연산을 사용하여 아래와 같이 직선의 기울기와 절편을 갱신할 수 있음

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

- 수정된 기울기 w와 절편 b를 이용하여 직선을 그려보자.
- 직선이 데이터를 잘 표현하도록 옮겨진 것을 확인할 수 있을 것

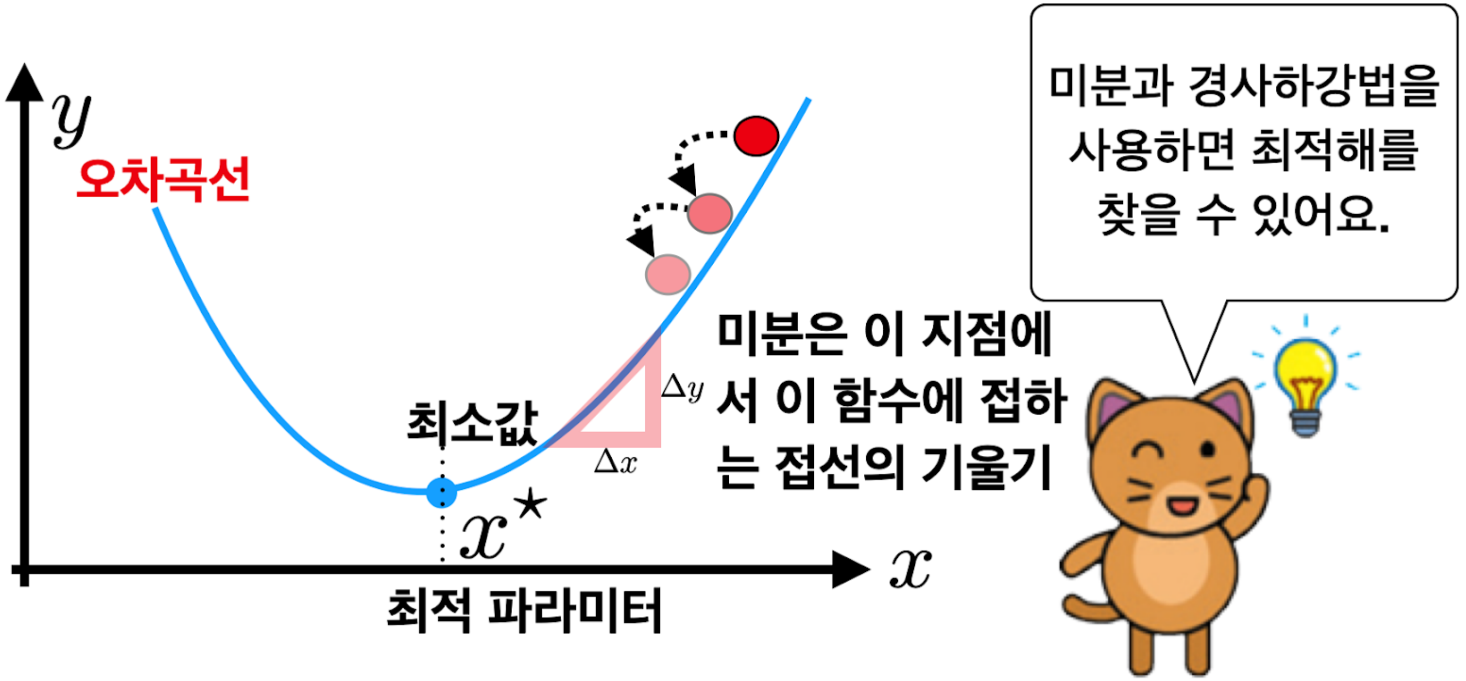
```
lin_data.plot(kind = 'scatter', x = 'input', y = 'pollution')
plt.plot([x0, x1], [h(x0, w, b), h(x1, w, b)])
```



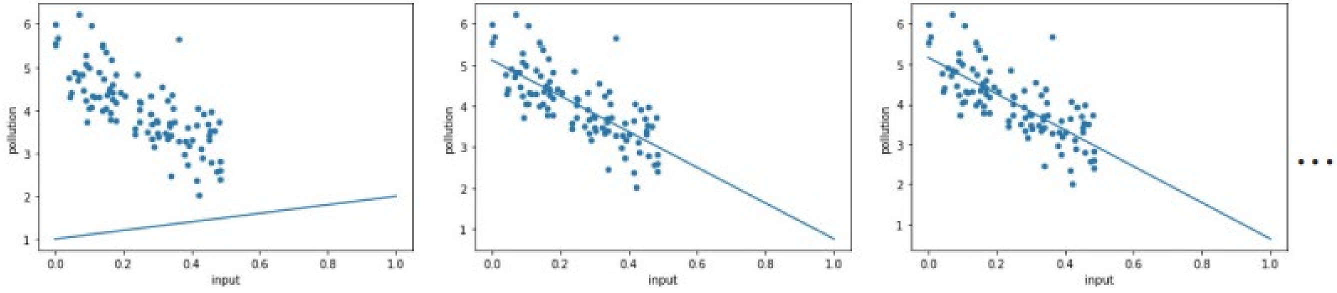
4.8 기계 학습의 개념으로 해석하는 선형 회귀

- 선형 회귀는 독립변수가 주어졌을 때 대응하는 종속변수를 찾는 최적의 가설hypothesis을 찾는 것
- 톰 미첼Tom Mitchell의 기계 학습 정의에 따른 선형 회귀
  - 독립변수에 대응하는 종속변수를 계산하는 일이 작업 T이다.
  - 작업의 성능은 종속변수의 추정치와 실제 값의 차이로 정의할 수 있고, 이것이 바로 성능척도 P
  - 선형 회귀는 좋은 가설을 찾기 위해 (독립변수, 종속변수) 쌍으로 이루어진 다수의 데이터의 독립변수에 대해 현재의 가설로 추정치를 계산하고, 이 값이 데이터와 일치하도록 변화한다.
  - 이때 주어지는 데이터가 바로 경험 E

- 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 : 경사하강법(gradient descent)라고 합니다.

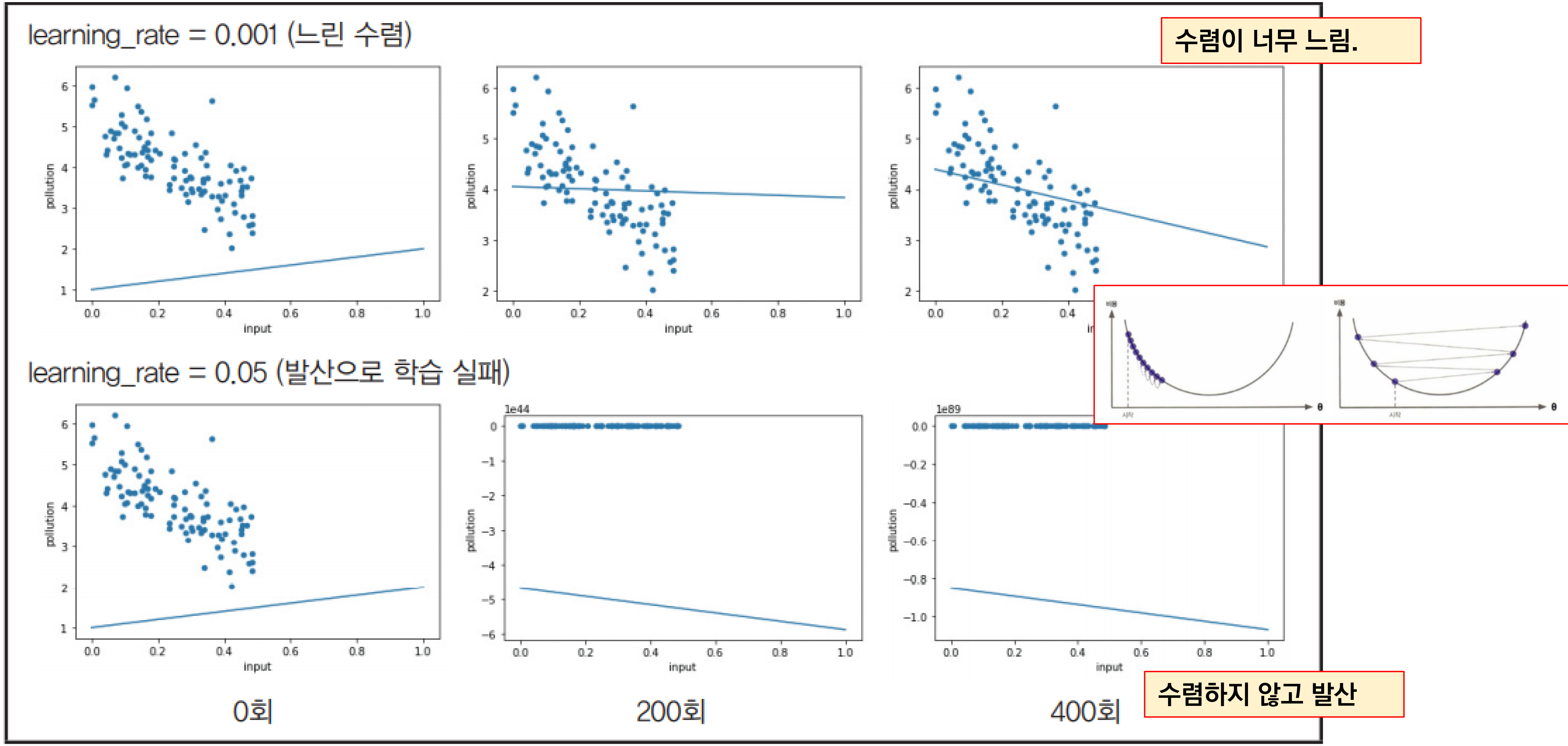


```
def h(x, param):
    return param[0]*x + param[1]

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 모델링하지요.

- learning_iteration과 learning_rate가 변경되면 학습으로 얻는 모델의 파라미터가 달라질 것
  - 학습을 제어하는 변수 : 하이퍼파라미터hyperparameter라고 함



4.9 Scikit-Learn을 이용한 선형 회귀

- 파이썬에서 가장 많이 사용되는 기계학습 라이브러리를 활용하여 선형 회귀를 구현
  - 사이킷런scikit-learn: 선형 회귀, k-NN 알고리즘, 서포트 벡터머신, k-means 등 다양한 기계학습 알고리즘을 쉽게 구현할 수 있게 해줌

```python
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')
```

- 사이킷런의 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.

```python
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]]
```

• 선형회귀 모델과 fit() 메소드의 역할?

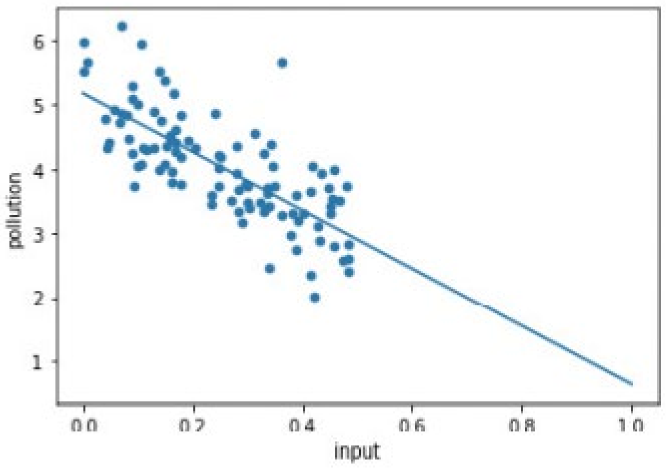
```
regr = linear_model.LinearRegression()
regr.fit(x, y)    # 선형 회귀 모델에 데이터를 넣어 학습을 진행함
```

데이터를 기반으로 최적의
선형회귀 모델 생성

이전의 수렴 과정을 모두 자
동화하는 메소드가 바로 fit()

- 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
```

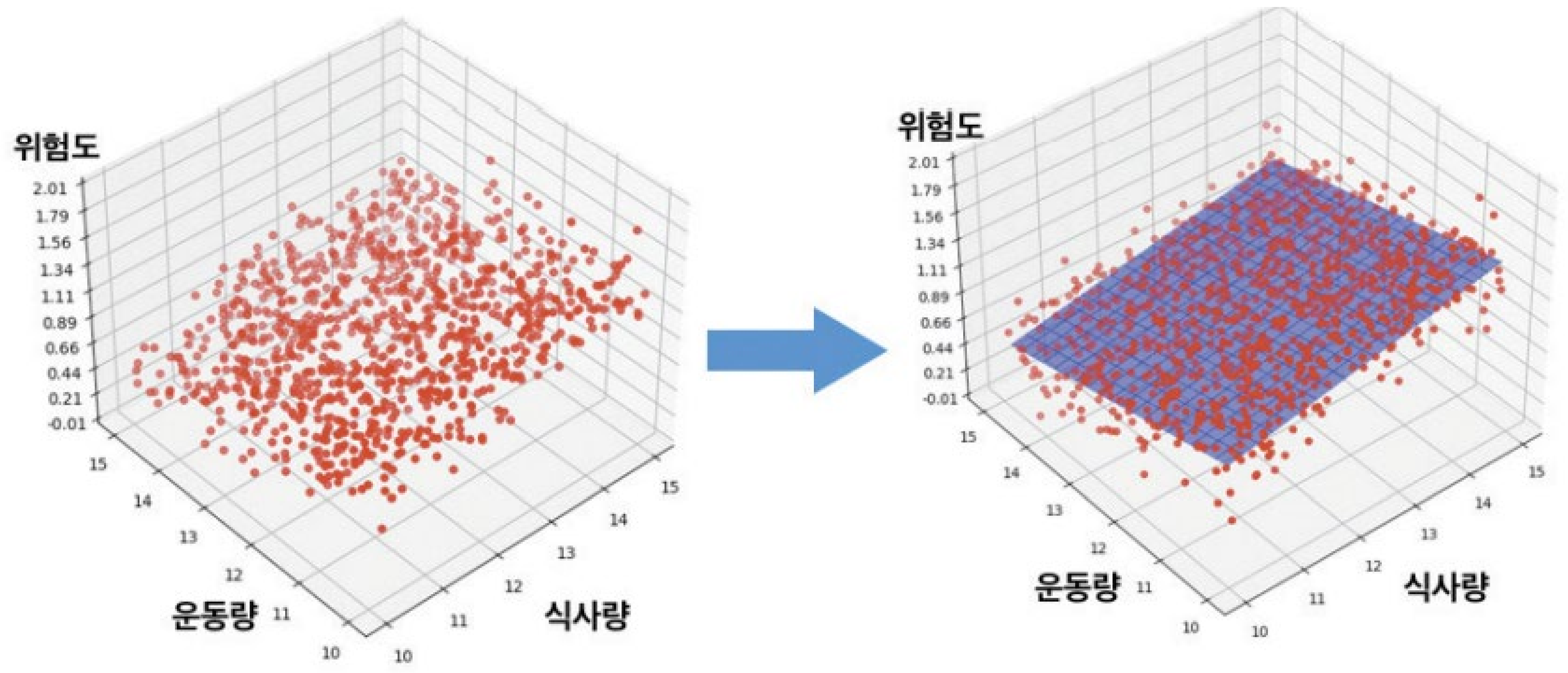


4.10 다변량 회귀분석 – 수학적 모델

- 입력에 사용되는 특징이 하나가 아니라 n개라면 선형 회귀 모델은 다음과 같은 식으로 표현할 수 있으며 다변량 multivariate 회귀분석이라 함

$$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

- 이들에 대한 건강검진을 통해 건강위험도 정보를 측정할 수 있었다면 아래 그림 왼쪽과 같이 운동량과 식사량에 따라 건강위험도가 결정되는 어떤 관계가 있을 것이라고 추측
- 이때 이 관계가 선형 관계라는 가설을 세웠다면 선형 회귀는 이 데이터를 설명할 수 있는 평면을 찾는 것



- 수식으로 표현하면 식사량을 x_1, 운동량을 x_2라고 할 수 있고, 위험도 y가 다음과 같은 식으로 설명될 수 있다는 것

$$\hat{y} = w_1 x_1 + w_2 x_2 + b = \mathbf{w}^{\mathrm{T}} \mathbf{x} + b$$

- 이때 w는 w_1, w_2를 원소로 하는 벡터이고, x 는 입력 특징 x_1, x_2를 원소로 하는 벡터
- 다변량 회귀분석은 이때 특징 벡터 x가 임의의 차원을 가질 수 있는 것을 의미하며, 찾아야 하는 것은 n차원 공간의 초평면hyperplane이 되는 것

- 이것을 파라미터parameter 벡터로 표현할 수도 있음
  - 이 모델의 동작을 결정하는 변수는 w 와 b
  - 이 둘을 따로 구분하지 않고 하나의 파라미터 θ로 표현한다면 w_i는 θ_i로 표현할 수 있고, b는 θ_0라고 하면 될 것
  - 그러면 선형 회귀는 다음과 같은 식으로 표현할 수 있음

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = \theta^T \mathbf{x}$$

- 가설은 파라미터 θ에 의해 동작이 결정되므로, m개의 데이터 인스턴스에 대해 현재 가설의 평균 제곱 오차mean square error를 다음과 같이 구할 수 있음
  - 각 데이터 인스턴스에 대해 구한 오차를 제곱하여 평균한 값
  - 다변량 선형 회귀 분석은 결국 이 오차가 최소가 되는 파라미터 θ를 찾는 문제

$$E_{mse}(\mathbf{x}, \theta) = \frac{1}{m}\sum_{i=1}^{n}(\theta^T \mathbf{x}^{(i)} - y^{(i)})^2$$

4.11 회귀분석의 학습, 혹은 최적화 방법 – 정규 방정식

- 다변량 선형 회귀에서 가설과 레이블의 평균 제곱 오차를 구했다면, 오차를 줄이는 방향으로 파라미터를 수정하는 것이 학습
  - 특정 파라미터 θ_j에 대해 이 오차를 편미분하면 다음을 얻을 수 있음.

$$\frac{\partial E_{mse}(\mathbf{x}, \theta)}{\partial \theta_j} = \frac{2}{m} \sum_{i=1}^{n} (\theta^T \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$$

- 오차 곡면이 파라미터에 대해 가지는 기울기 $\nabla_\theta E_{mse}$의 j번째 원소가 됨
  - 파라미터는 다음과 같이 변경할 수 있음

$$\theta \leftarrow \theta - \eta \nabla_\theta E_{mse}$$

- 오차의 기울기를 따라 최적의 파라미터를 찾아가는 방법을 사용하지 않고, 정규 방정식normal equation이라고 부르는 최적의 파라미터를 찾는 수식
- 훈련에 사용되는 각 데이터 인스턴스 $x^{(\mathrm{i})}$를 i번째 행으로 하는 행렬을 X라고 정의하자. 그러면 최적의 파라미터 $\hat{\theta}$는 다음과 같음

$$\hat{\theta} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}y$$

- 편향bias를 파라미터 θ_0로 다루기 위해 넘파이의 c_()를 이용하여 100개의 행을 가지고 각 행마다 1을 원소로 하는 벡터를 생성해 입력 데이터에 병합

```python
data_home = 'https://github.com/dknife/ML/raw/main/data/'
lin_data = pd.read_csv(data_home+'pollution.csv')
x = lin_data['input'].to_numpy()
y = lin_data['pollution'].to_numpy()
x = x[:, np.newaxis]
X = np.c_[np.ones((100, 1)), x]
print(X)
```

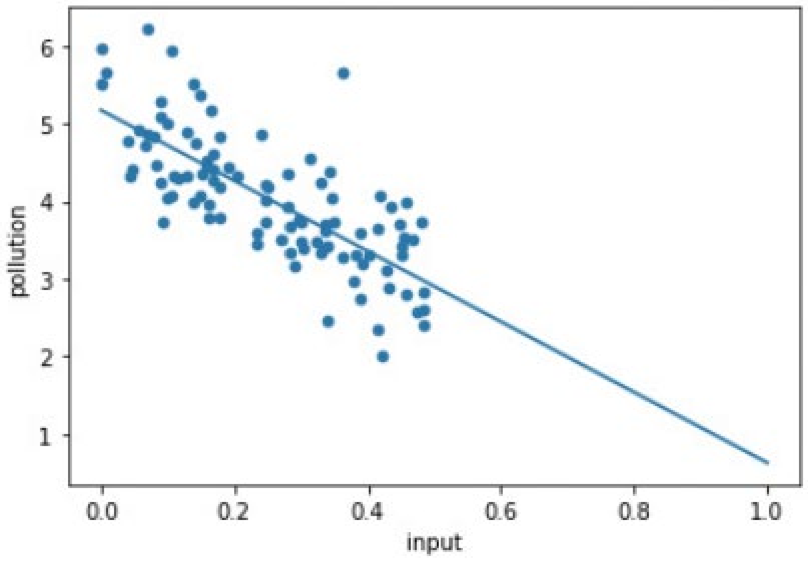
```
[[1.00000000e+00 2.40557071e-01]
 [1.00000000e+00 1.59730598e-01]
 ...
 [1.00000000e+00 7.20485998e-03]
 [1.00000000e+00 2.90293677e-01]]
```

- 정규 방정식을 푸는 것은 넘파이의 선형대수 서브 모듈인 linalg를 이용

```
theta = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
theta
```

```
array([ 5.17550358, -4.54449866])
```

```python
def h(x, theta):          # 가설 함수
    return x*theta[1] + theta[0]

# 데이터와 가설을 비교
lin_data.plot(kind = 'scatter', x = 'input', y = 'pollution')  # 데이터
plt.plot([0, 1], [h(0, theta), h(1, theta)])                   # 가설
```



4.12 정규 방정식의 유도와 시간 복잡도

- 정규 방정식은 개별 데이터 인스턴스의 특징값을 이용하여 정의된 평균 제곱 오차를 행렬로 다시 써서 유도

$$E_{mse}(\mathbf{X}, \theta) = \frac{1}{m}\sum_{i=1}^{n}(\theta^T \mathbf{x}^{(i)} - y^{(i)})^2$$

$$\downarrow$$

$$E_{mse}(\mathbf{X}, \theta) = \frac{1}{m}(\mathbf{X}\theta - \mathbf{y})^{\mathrm{T}}(\mathbf{X}\theta - \mathbf{y})$$

- 이 오차를 파라미터 벡터 θ로 편미분

$$\frac{\partial E_{mse}(\mathbf{X}, \theta)}{\partial \theta} = \frac{2}{m}(\mathbf{X}^{\mathrm{T}}\mathbf{X}\theta - \mathbf{X}^{\mathrm{T}}\mathbf{y})$$

- 최적의 오차는 이 미분치가 0인 극소
- 최적 파라미터는 이 미분을 0으로 만드는 $\hat{\theta}$일 때 다음 식을 만족

$$(\mathbf{X}^{\mathrm{T}}\mathbf{X}\hat{\theta} - \mathbf{X}^{\mathrm{T}}\mathbf{y}) = 0 \Rightarrow \mathbf{X}^{\mathrm{T}}\mathbf{X}\hat{\theta} = \mathbf{X}^{\mathrm{T}}\mathbf{y}$$

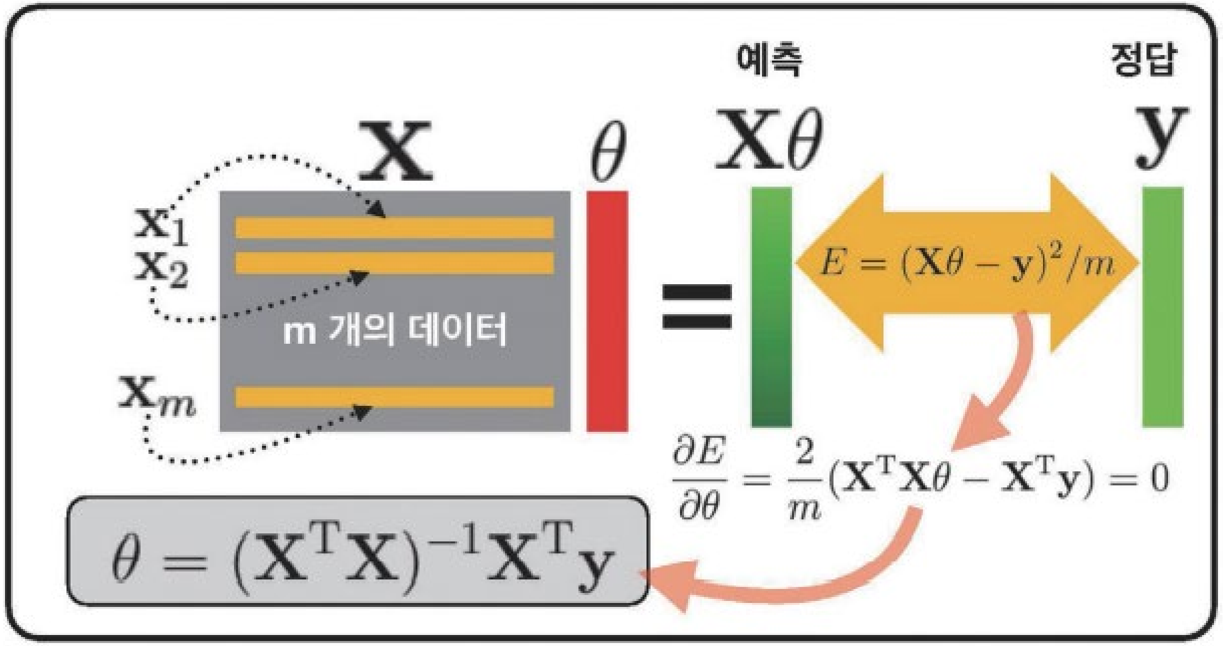
- 최적의 파라미터를 구하기 위해 이 파라미터 앞에 곱해진 행렬의 역행렬을 곱함

$$(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}(\mathbf{X}^{\mathrm{T}}\mathbf{X})\hat{\theta} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

- 최적의 파라미터는 다음과 같은 식

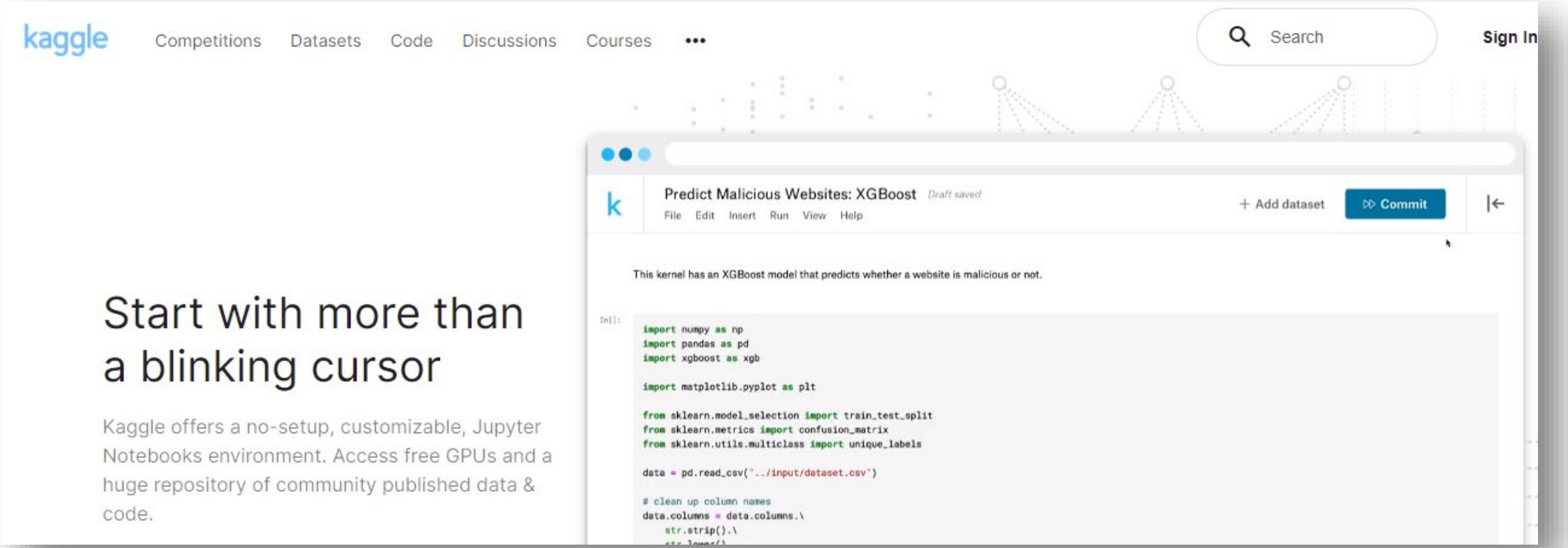
$$\hat{\theta} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

- 이 행렬의 각 원소는 m차원 벡터를 내적하여 얻을 수 있고, 벡터화 연산을 통해 효율적으로 처리될 것
  - 연산이 총 $(n+1) \times (n+1)$번 필요하다.
  - 행렬의 역행렬을 구해야 한다.
  - 역행렬 구하기의 복잡도를 결정하는 것은 특징의 수 n
- 특징값 벡터의 원소가 매우 많은 입력에 대해서 정규 방정식이 나쁜 성능을 보일 수 있음



4.13 다변량 선형 회귀 분석을 위한 데이터 확보하기

- 사이킷런을 활용하여 다변량 회귀분석을 공개된 큰 데이터에 적용
- 사용할 데이터는 캐글kaggle 사이트에서 가져옴
  - 캐글은 2010년 설립된 데이터 분석 플랫폼으로 다양한 데이터를 공개하고 있다. 사용할 데이터는 세계보건 기구WHO에서 내어 놓은 각 나라별 기대수명 데이터



```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns    # 시각화를 위하여 Seaborn 라이브러리를 이용함

data_loc = 'https://github.com/dknife/ML/raw/main/data/'
life = pd.read_csv(data_loc + 'life_expectancy.csv')
life.head()
```

	Country	Year	Status	Life expectancy	Adult mortality	Infant deaths	Alcohol	Percentage expenditure	Hepatitis B	Measles	BMI
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	19.1
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	18.6
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	18.1
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	17.6
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	17.2

Under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	Thinness 1-19 years	Thinness 5-9 years	Income composition of resources	Schooling
83	6.0	8.16	65.0	0.1	584.259210	33736494.0	17.2	17.3	0.479	10.1
86	58.0	8.18	62.0	0.1	612.696514	327582.0	17.5	17.5	0.476	10.0
89	62.0	8.13	64.0	0.1	631.744976	31731688.0	17.7	17.7	0.470	9.9
93	67.0	8.52	67.0	0.1	669.959000	3696958.0	17.9	18.0	0.463	9.8
97	68.0	7.87	68.0	0.1	63.537231	2978599.0	18.2	18.2	0.454	9.5

- **Adult mortality**: 15세에서 60세 사이의 성인 1000명당 사망자 수
- **Infant deaths**: 유아 1000명당 사망자 수
- **Alcohol**: 1인당 알콜 소비량
- **Percentage expenditure**: GDP 대비 보건 예산 지출 비율 (%)
- **Hepatitis B**: 1세 아동의 B형 간염 예방접종률 (%)
- **Measles**: 인구 1000명당 홍역 예방접종률 (%)
- **BMI**: 전인구의 평균 체질량 지수
- **Under-five deaths**: 5세 이하 아동 1000명당 사망자 수
- **Polio**: 1세 아동의 소아마비 면역률 (%)
- **Total expenditure**: 정부 총 예산 대비 보건분야 예산 비율 (%)
- **Diphtheria**: 1세 아동의 디프테리아 예방접종률 (%)
- **HIV/AIDS**: HIV/AIDS 감염 상태로 태어난 0-4세 인구 1000명 당 사망자 수
- **GDP**: 1인당 GDP
- **Population**: 국가 총 인구
- **Thinness 1-19 years**: 10세에서 19세 청소년 중 저체중 비율
- **Thinness 5-9 years**: 5세에서 9세 사이 아동의 저체중 비율
- **Income composition of resources**: 소득 구성에 따른 인간개발 지수
- **Schooling**: 학교 재학 연수

- 데이터에서 우리가 관심을 가진 데이터만 정리해서 새로 만들고 싶으면 원하는 열만 지정하여 새로운 순서로 정리할 수 있음

```
life = life[['Life expectancy', 'Year', 'Alcohol',
             'Percentage expenditure', 'Total expenditure',
             'Hepatitis B', 'Measles', 'Polio', 'BMI', 'GDP',
             'Thinness 1-19 years', 'Thinness 5-9 years']]
print(life)
```

```
    Life expectancy  Year  ...  Thinness 1-19 years  Thinness 5-9 years
0              65.0  2015  ...                 17.2                17.3
...
```

4.14 다변량 데이터 특징들 사이의 상관 관계를 파악하기

```
print(life.shape)
print(life.isnull().sum())
```

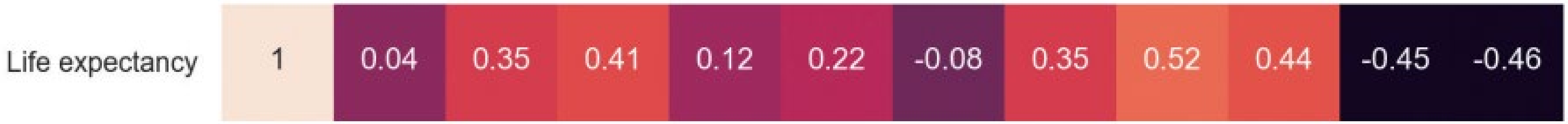
```
(2938, 12)
Life expectancy          10
Year                      0
Alcohol                 194
Percentage expenditure    0
Total expenditure       226
Hepatitis B             553
Measles                   0
Polio                    19
BMI                      34
GDP                     448
Thinness  1-19 years     34
Thinness 5-9 years       34
```

- 크기는 shape 속성으로 알 수 있고, 결손값은 isnull() 함수를 이용
  - 결손값이 있는 데이터는 모두 삭제

```
life.dropna(inplace = True)
print(life.shape)
```

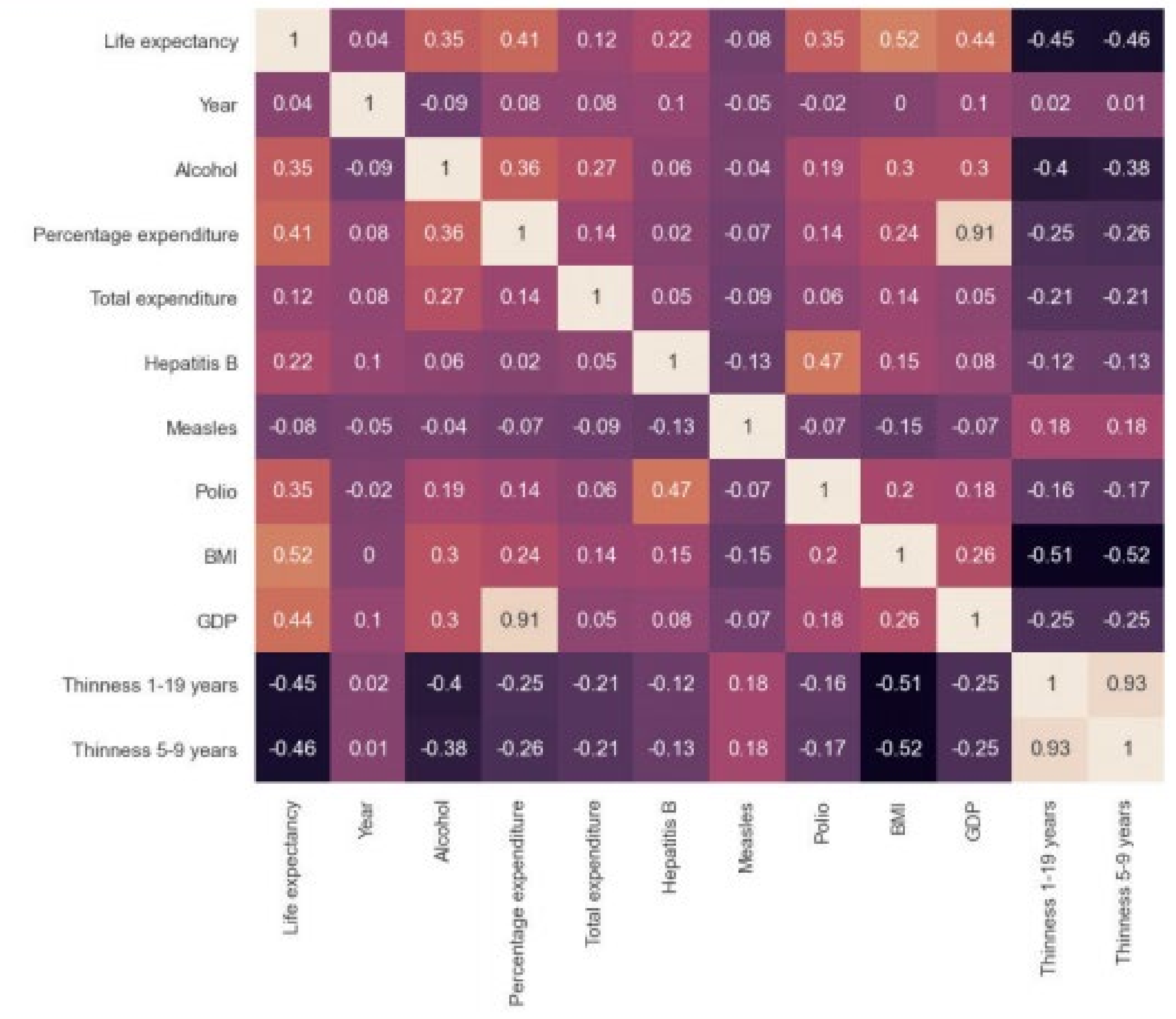
```
(1853, 12)
```

- 예측하려는 값과 상관관계가 있는 데이터만 입력으로 사용 하는 것이 바람직
- 이를 파악하기 위해 판다스 라이브러리의 corr() 함수를 사용하여 상관행렬correlation matrix을 형성
  - seaborn 라이브러리의 heatmap() 함수를 이용하여 그려보자.

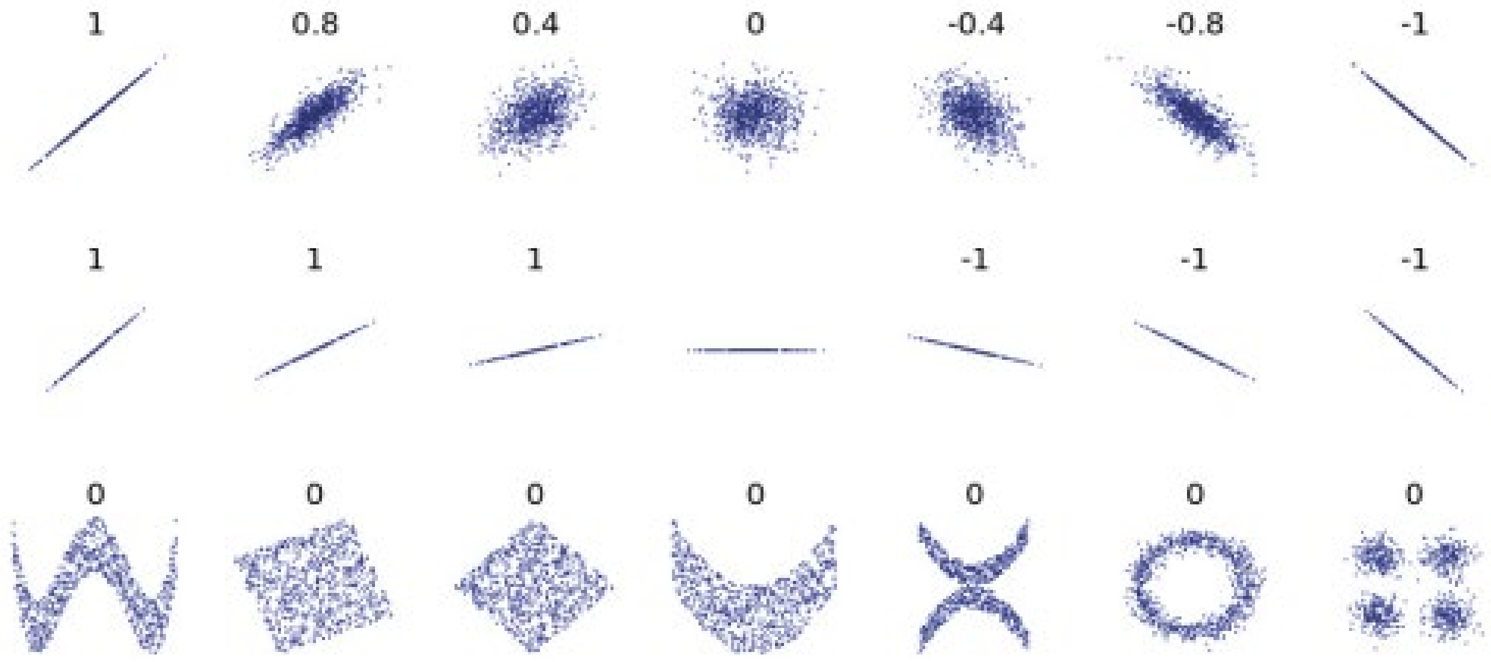


기대수명 상관계수값들

```python
sns.set(rc={'figure.figsize':(12,10)})    # 상관행렬 가시
correlation_matrix = life.corr().round(2) # 상관행렬 생성
sns.heatmap(data=correlation_matrix, annot=True)
plt.show()  # colab 등 노트북 환경에서는 필요없지만, 콘솔 환경 등에서는 필요
```

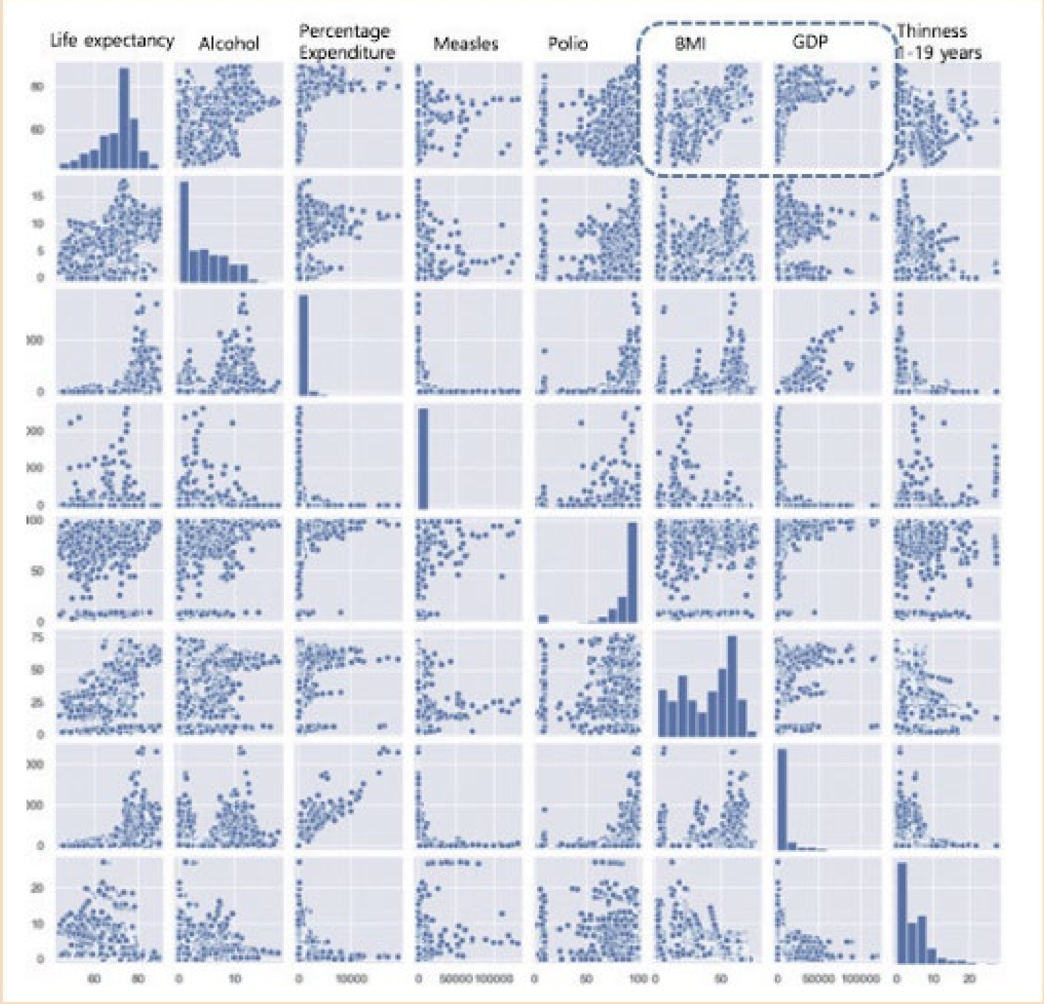


- 피어슨 상관계수Pearson correlation coefficient로서 단순히 두 데이터들을 선형관계로 가정했을 때 관찰되는 기울기가 아니라, 두 데이터의 공분산을 각각의 표준편차의 곱으로 나눈 값이다.
  - 데이터의 분포에 따른 피어슨 상관계수 값을 표시한 그림



4.15 특징들의 상관 쌍 그림을 확인하고 중요 특징 추출하기

```
sns.pairplot(life[['Life expectancy', 'Alcohol', 'Percentage expenditure',
'Measles', 'Polio', 'BMI', 'GDP', 'Thinness 1-19 years']])
plt.show()  # colab 등 노트북 환경에서는 필요없지만, 콘솔 환경 등에서는 필요
```



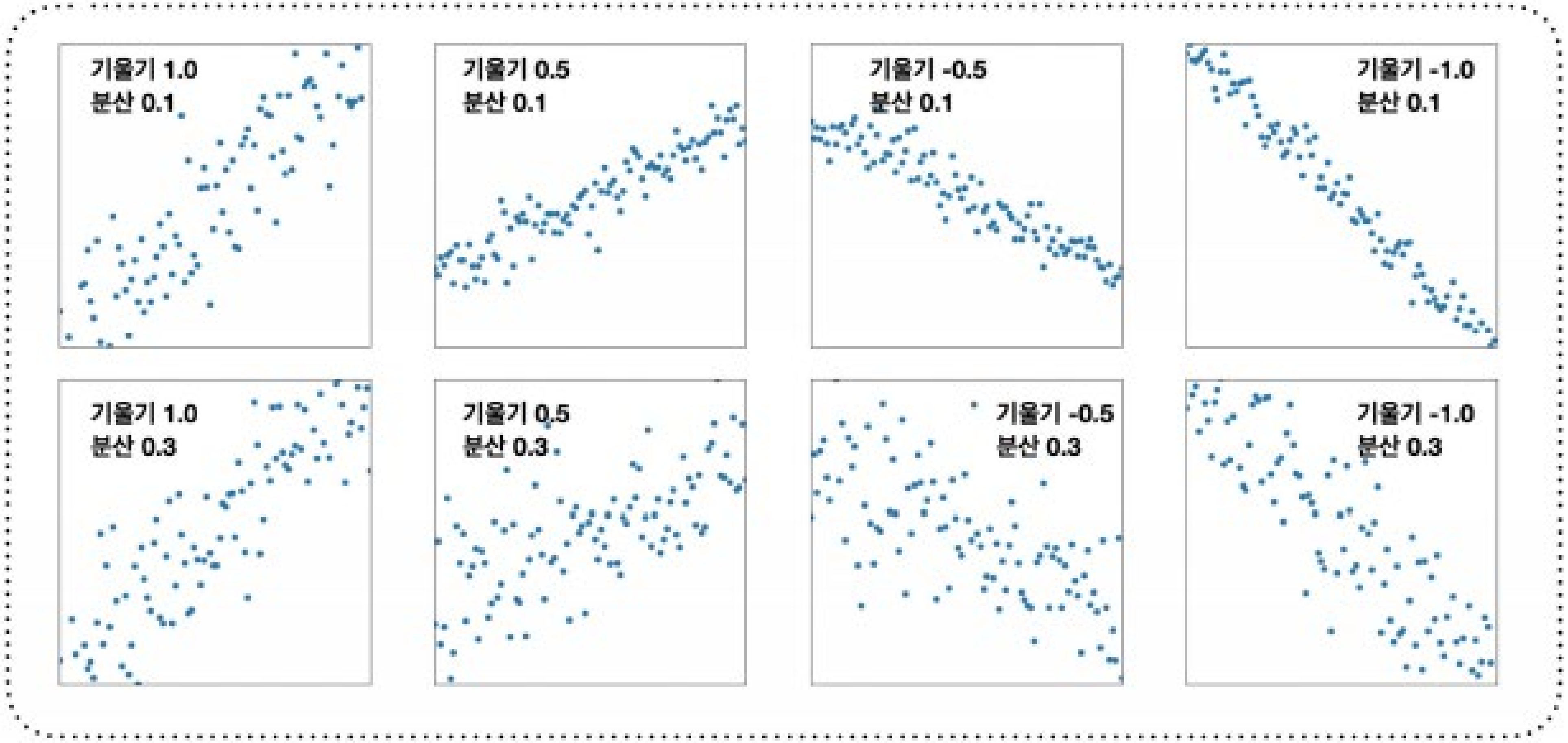
- 데이터의 상관관계correlation를 파악하기 위해서는 쌍 그림pair plot을 이용할 수도 있음
- 사망률에 크게 영향을 미치지 않는 특징들을 모두 포함하면 학습이 잘 되지 않거나, 예측의 신뢰도가 오히려 떨어질 수 있음
- 우리는 앞 절의 조사를 바탕으로 다음과 같이 알콜 소비, 보건 예산, 소아마비 접종률, BMI, GDP, 저체중 정보만을 추출하여서 데이터 프레임으로 재구성

- 기대 수명을 예측하는 데에 사용할 특징값 데이터들로 이루어진 행렬은 X로 저장하고, 이 데이터에 짝을 이루는 레이블, 기대수명은 y로 저장

```
X = life[['Alcohol', 'Percentage expenditure', 'Polio',
          'BMI', 'GDP', 'Thinness 1-19 years']]
y = life['Life expectancy']
print(X)
print(y)
```

```
      Alcohol  Percentage expenditure  ...         GDP  Thinness 1-19 years
0        0.01               71.279624  ...  584.259210                 17.2
1        0.01               73.523582  ...  612.696514                 17.5
2        0.01               73.219243  ...  631.744976                 17.7
...       ...                     ...  ...         ...                  ...
2936     1.72                0.000000  ...  548.587312                  1.6
2937     1.68                0.000000  ...  547.358879                 11.0

[1853 rows x 6 columns]
0       65.0
1       59.9
        ...
2936    45.3
2937    46.0
Name: Life expectancy, Length: 1853, dtype: float64
```

- 두 데이터가 선형 관계를 이루고 있다고 할 때, 쌍그림은 선형관계의 기울기와 분산에 따라 폭이 달라짐
  - 기울기와 앞 절의 피어슨 상관계수는 동일하지 않다는 것에 유의하라.



4.16 다변량 선형 회귀에 사용할 데이터를 훈련용과 검증용으로 분리하기

- 학습 과정은 데이터가 알려주는 정답과 함수 출력의 오차를 가장 작게 만들려고 함
- 이미 살펴본 데이터에 대해서 이 함수는 당연히 좋은 결과를 얻을 것이고, 알고리즘의 성능을 검사하는데는 부적합
- 알고리즘의 성능을 검사할 때에 학습과정에서 한 번도 본 적이 없는 '새로운 데이터'로 테스트하는 것을 검증 validation 이라고 함

- 사이킷런의 model_selection 서브 모듈에 있는 train_test_split()이라는 함수를 이용하여 X, y에 담긴 데이터 가운데 80%는 훈련용, 나머지 20%는 검증용으로 나눔

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.2)
```

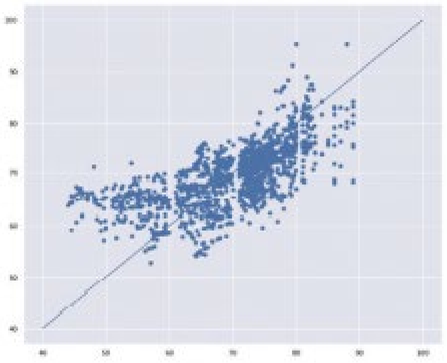
- 사이킷런의 LinearRegression 모델을 사용하여 학습을 진행

```
from sklearn.linear_model import LinearRegression

lin_model = LinearRegression()
lin_model.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

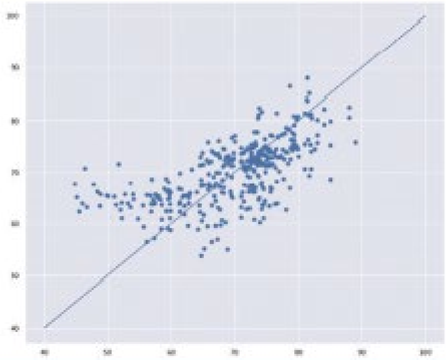
- 학습으로 얻은 함수가 내어 놓는 출력을 예측prediction
  - 입력에 대한 예측은 predict() 함수
- 훈련용 데이터에 대한 예측치를 y_hat_train이라고 하고, 이를 정답인 y_train과 비교 하여 화면에 그렸을 때 기울기 1인 직선에 가까울 수록 예측과 정답이 일치하는 것

```python
y_hat_train = lin_model.predict(X_train)
plt.scatter(y_train, y_hat_train)
xy_range = [40, 100]
plt.plot(xy_range, xy_range)
```



- 이 함수가 다른 데이터에도 잘 동작하는 지 확인하려면 남겨두었던 검증용 데이터 X_test를 사용

```
y_hat_test = lin_model.predict(X_test)

plt.scatter(y_test, y_hat_test)    # 검증 데이터와 예측 데이터의 산포도
plt.plot(xy_range, xy_range)       # (40,40)에서 (10,100)으로 직선
                                   #    - 데이터가 이 직선과 일치할수록 예측이 정확
```



- 검증 데이터에 대한 예측과 정답을 비교하여 평균 제곱오차를 구하는 것

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

```
Mean squared error: 44.01777478568859
```

4.17 데이터의 정규화를 통한 분석 성능 개선하기

- 알콜 섭취량은 10^2이내의 수치들이지만 GDP 항목은 10^5규모까지 근접한 값이 나타남
- 값의 범위가 크게 다른 특징들을 입력 변수로 사용할 경우 적절한 학습이 이루어지지 않을 수도 있음
- 각각의 특징들이 갖는 값들을 적당한 규모로 변경하는 작업이 필요
  - 각 데이터를 정규화normalization하는 방법
  - x데이터의 최소값이 x_{min}, 최대값이 x_{max}라고 하면 x를 정규화한 $\tilde{x}$는 다음과 같음

$$\tilde{x} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

- 입력 데이터를 모두 이러한 방식으로 정규화하면 모든 데이터는 0에서 1사이의 값을 갖게 됨
- 사이킷런에는 학습 전에 데이터를 정제하기 위한 정규화 함수가 제공
    - preprocessing 서브 모듈에 있는 normalize() 함수

```python
from sklearn.preprocessing import normalize

n_X = normalize(X, axis = 0)      # 정규화를 0번 축 기준으로 실시
```

- axis 파라미터는 디폴트 인자가 1로 설정되어 있는데, 이것은 배열을 행 단위로 정규화
    - 각 열에 저장된 특징의 값을 정규화
    - 각 특징들은 다른 특징의 최대, 최소에 영향을 받을 필요없기 때문

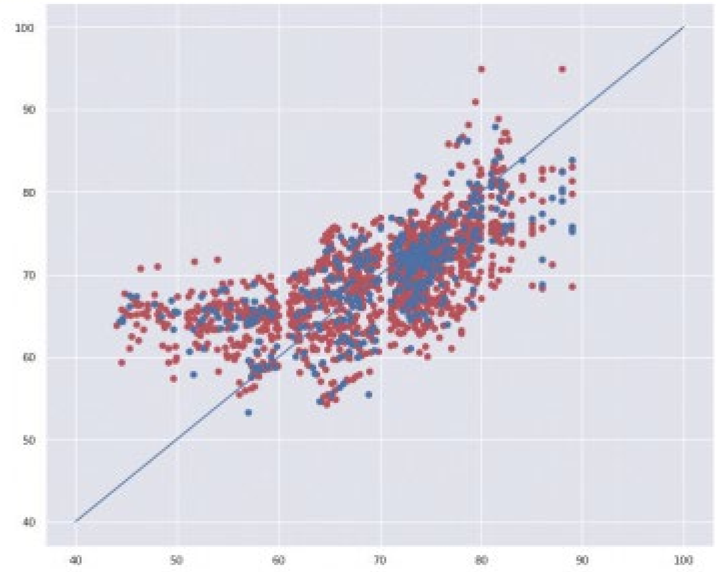
- 정규화된 데이터 n_X를 훈련용과 검증용으로 나누어 학습

```
nXtrain,nXtest,y_train,y_test = train_test_split(n_X, y, test_size=0.2)
lin_model.fit(nXtrain, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

- 학습의 결과로 얻은 선형 회귀 모델을 이용하여 훈련용 데이터와 검증용 데이터를 입력으로 예측치를 구해서 실제 정답과 비교
- 붉은 색은 훈련용 데이터, 파란 색은 검증용 데이터의 예측이 정답과 얼마나 일치하는지를 보임

```
y_hat_train = lin_model.predict(nXtrain)
y_hat_test = lin_model.predict(nXtest)
plt.scatter(y_train, y_hat_train, color='r')
plt.scatter(y_test, y_hat_test, color='b')
plt.plot(xy_range, xy_range)
```



- 정확한 오차를 확인하기 위해 평균 제곱 오차를 계산

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

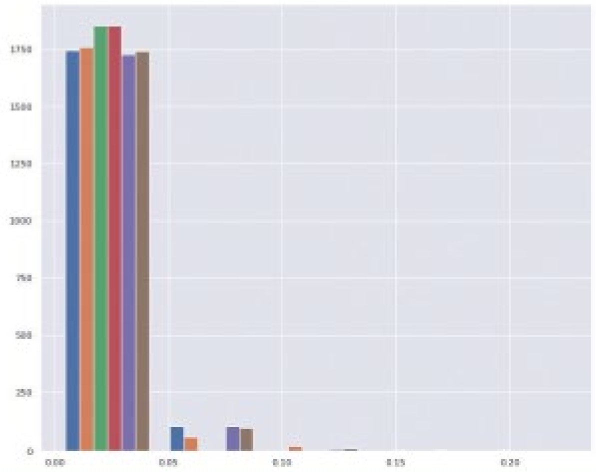
```
Mean squared error: 42.97092627692947
```

- 점진적으로 최적해를 찾아가는 다양한 기계학습 기법들에서 특징들마다 값의 크기가 크게 차이가 날 경우 학습이 제대로 이루어지지 않을 수 있어 정제가 매우 중요

4.18 데이터의 표준화를 통한 분석 성능 개선하기

- 정규화를 통해 얻은 n_X 데이터가 어떤 값을 갖는지 맷플롯립의 히스토그램histogram을 그려서 확인
- 그림에서 확인할 수 있는 바와 같이 6개의 특징값을 나타내는 대부분의 데이터가 매우 작은 값쪽에 몰려 있는데 이것은 극단적으로 큰 소수의 이상치outlier가 존재하기 때문

```
plt.hist(n_X, bins=5)
```



- 데이터에 대한 정제 기법이 필요
  - 방법은 **표준화**standardization
  - 표준화는 데이터를 정제할 때 전체의 **평균**mean과 **분산**variance을 사용
  - m개의 인스턴스를 가진 데이터 x_i의 평균 μ_x와 분산 σ_x^2는 다음과 같다.

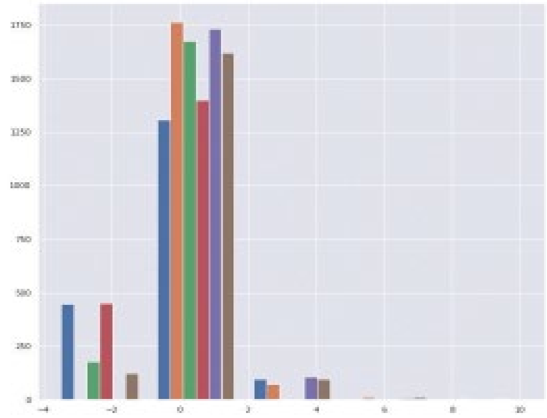
$$\mu_x = \frac{1}{m}\sum_{i=1}^{m} x_i, \quad \sigma_x^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_x)^2$$

- 표준편차standard deviation는 분산의 제곱근으로 σ_x로 나타냄
- 표준화는 이들 값을 구한 뒤에 x를 다음과 같이 x'로 바꾸는 일
- 변환이 이루어지면 x' 는 평균이 0이고 분산이 1 인 데이터가 됨

$$x' = \frac{x - \mu_x}{\sigma_x}$$

- 사이킷런을 이용하여 데이터를 표준화할 때는 preprocessing의StandardScaler 클래스를 사용
- 이 클래스의 fit_transform() 메소드에 표준화를 할 데이터를 주면 표준화

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()          # 표준화 스케일러 생성
s_X = scaler.fit_transform(X)     # 표준화 스케일러 적용

plt.hist(s_X, bins=5)             # 표준화 결과의 가시화
```



- 표준화를 통해 정제된 데이터는 0의 값 근처에 모여 있는 것을 확인
  - 데이터들의 분산은 1이다.

```
y_hat_train = lin_model.predict(sXtrain)
y_hat_test = lin_model.predict(sXtest)
plt.scatter(y_train, y_hat_train, color='r')
plt.scatter(y_test, y_hat_test, color='b')
plt.plot(xy_range, xy_range)
```

- 학습을 통해 얻은 모델의 X_test에 대한 예측이 가진 오차는 다음과 같음

```
print('Mean squared error:', mean_squared_error(y_test, y_hat_test))
```

```
Mean squared error: 37.61657220608388
```

핵심 정리

- 회귀 분석의 회귀라는 용어는 프랜시스 골턴이 통계학 분야에서 처음으로 사용한 용어에 기원을 두고 있 지만, 골턴의 방법과 현재의 회귀 분석에는 차이가 있다.
- 회귀 분석은 데이터를 설명하는 선형 함수를 찾는 선형 회귀와 비선형 함수를 찾는 비선형 회귀로 구분할 수 있다.
- 선형 회귀에서 데이터를 설명하기 위한 함수를 가설이라고 한다.
- 선형 회귀가 좋은 가설을 찾기 위해서는 입력에 대응하는 정답 출력을 레이블로 제공해야 한다.
- 회귀 모델은 가설의 출력과 레이블을 비교하여 얻은 오차를 줄이는 방식으로 가설을 개선해 나간다.
- 레이블에 근거하여 더 나은 가설을 찾아가는 방식의 학습을 지도학습이라고 한다.
- 최적의 파라미터를 찾기 위한 과정에서 오차를 제곱한 평균 제곱 오차가 흔히 사용된다.
- 가설의 동작을 결정하는 파라미터는 선형 방정식의 계수들이다.