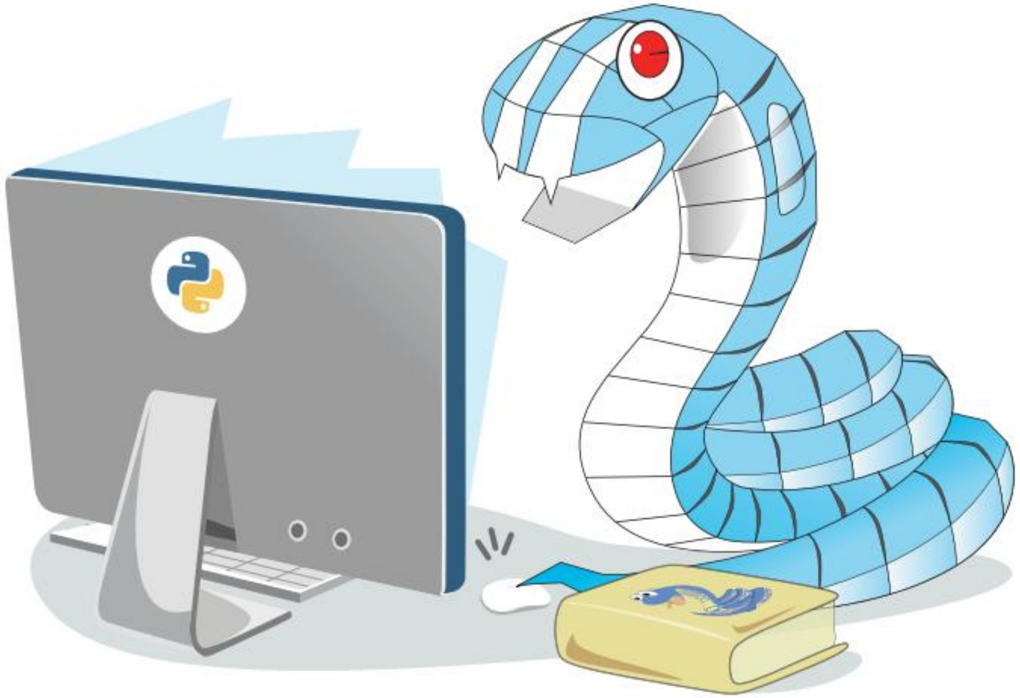


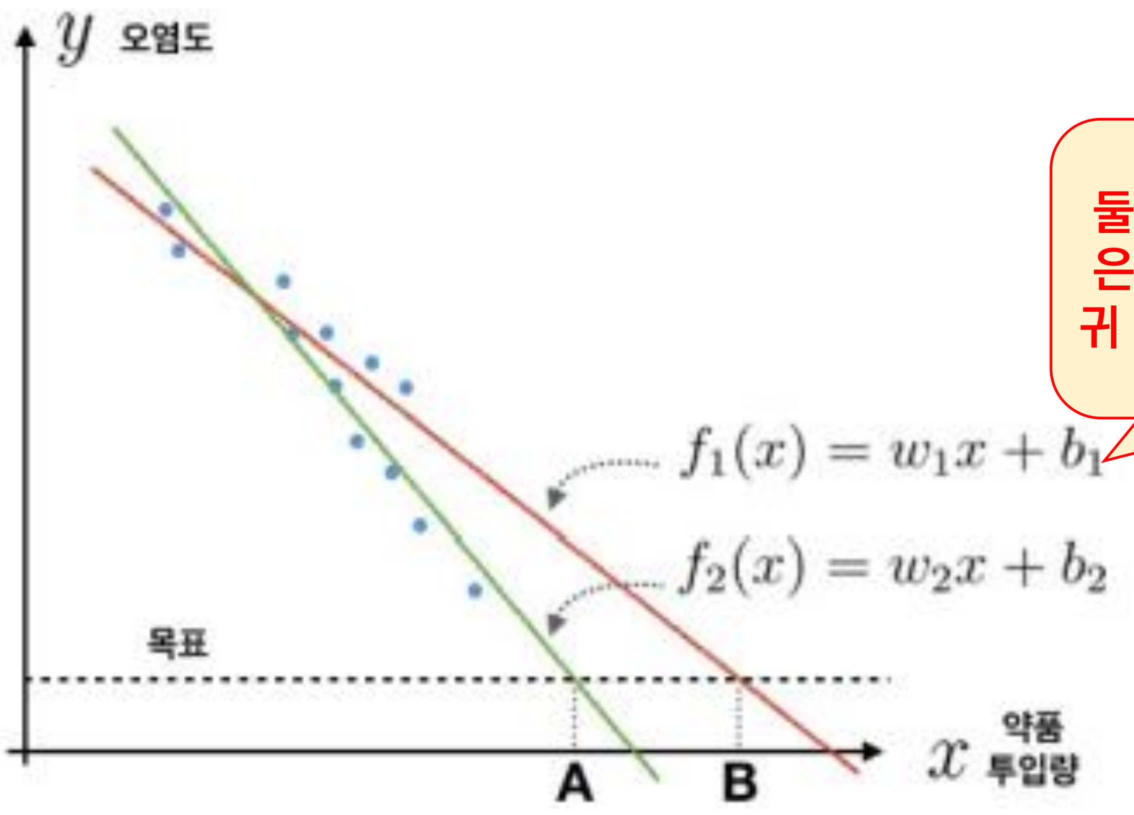
선형 회귀 종합실습
– 다변량 선형회귀까지

회귀 모델

선형회귀 돌아보기

- 오차의 이해 / 평균 제곱 오차
- 모델과 파라미터, 그리고 좋은 파라미터
- 경사 하강법을 통한 "학습"
- 다변량 선형회귀

- 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.

오차의 이해

```
[ ] import numpy as np
```

예측치(prediction) $\hat{y}$와 정답(label) y

```
[ ] prediction = np.array([1, 2, 3, 4, 5, 6, 7])
    label = np.array([1.1, 2.2, 3.3, 4.1, 5.5, 6.5, 7.4])
```

평균 제곱 오차를 계산해 보자

단순 오차: $E = \hat{y} - y$

평균 제곱 오차 (mean squared error) : E^2

```
[ ] E = prediction - label
    E
```

```
array([-0.1, -0.2, -0.3, -0.1, -0.5, -0.5, -0.4])
```

```
[ ] E_Square = E**2
    E_Square
```

```
array([0.01, 0.04, 0.09, 0.01, 0.25, 0.25, 0.16])
```

```
[ ] E_Square.sum()
```

```
np.float64(0.8100000000000002)
```

```
[ ] my_mse = E_Square.sum() / len(E)
    my_mse
```

- **데이터 다루기**

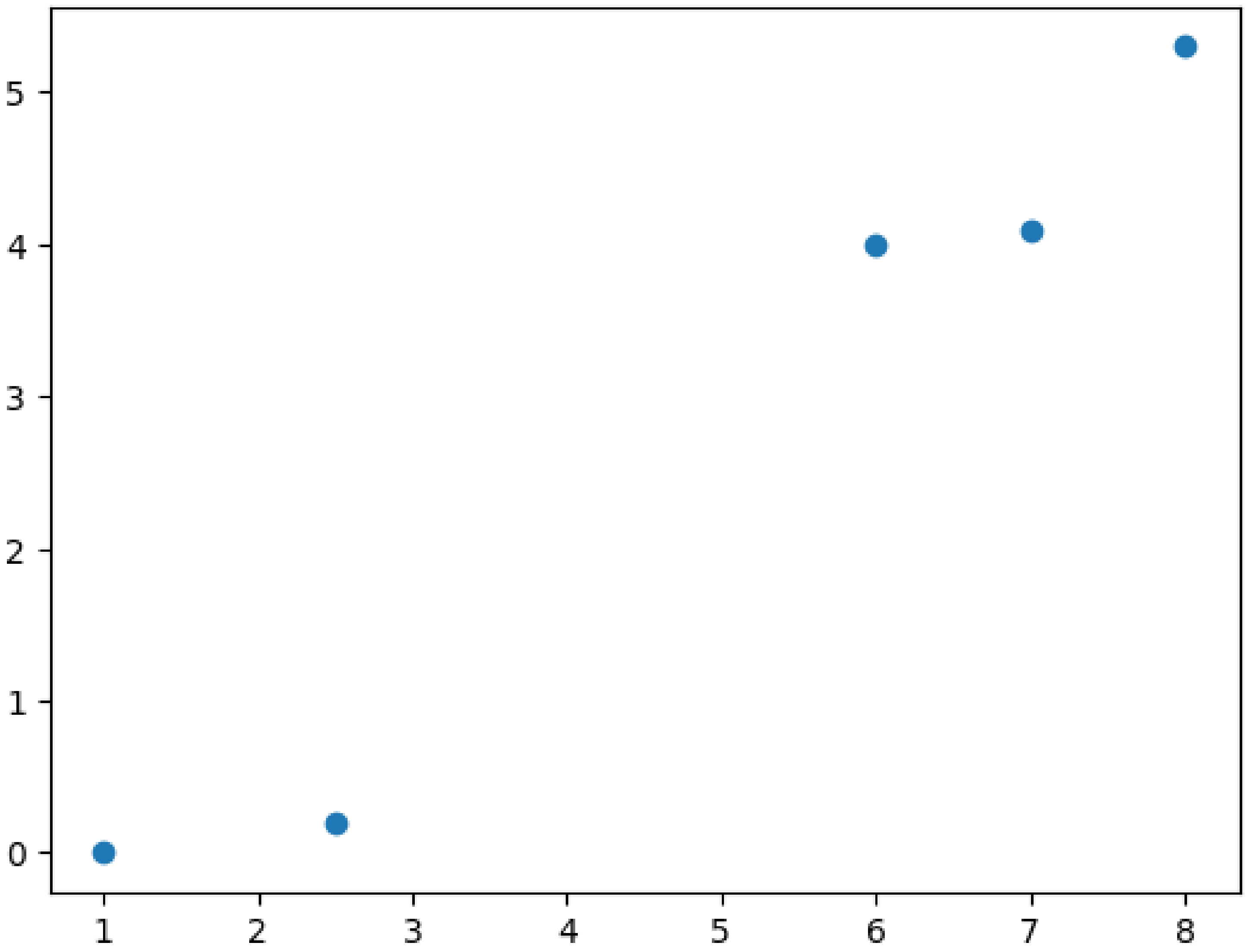
$y = f(x)$인데 우리의 목표는 데이터를 통해 $f()$를 추정하는 것

```
x = np.array([1, 2.5, 6, 7, 8])       # domain
y = np.array([0, 0.2, 4.0, 4.1, 5.3])  # range
```

눈으로 봐야 이해하지

```
[ ]  import matplotlib.pyplot as plt
     plt.scatter(x, y)
```

- **데이터를 살펴보자**



여기에는 어떤 관계가 있을까?

> $f(x) = y$
>
> 함수 f는 무엇일까? 데이터를 보고 추측해 보자.
>
> $f(x) = y = wx + b$ 인데, w는 반드시 양수이고, b는 0 근처

우리의 예측 모델을 만들어 보자

```python
def f(x, w = 1.0, b= 0.0) :
    return w * x + b
```

```python
# x - > y 예측
y_pred = f(x, w=1, b=-0.0)
x, y_pred
```

```
(array([1. , 2.5, 6. , 7. , 8. ]), array([1. , 2.5, 6. , 7. , 8. ]))
```

- **결과 비교**

예측 결과를 정답과 비교하자

정답 y와 추측값 $\hat{y}$을 비교해 보자

```
[ ] mse(y_pred, y)
```

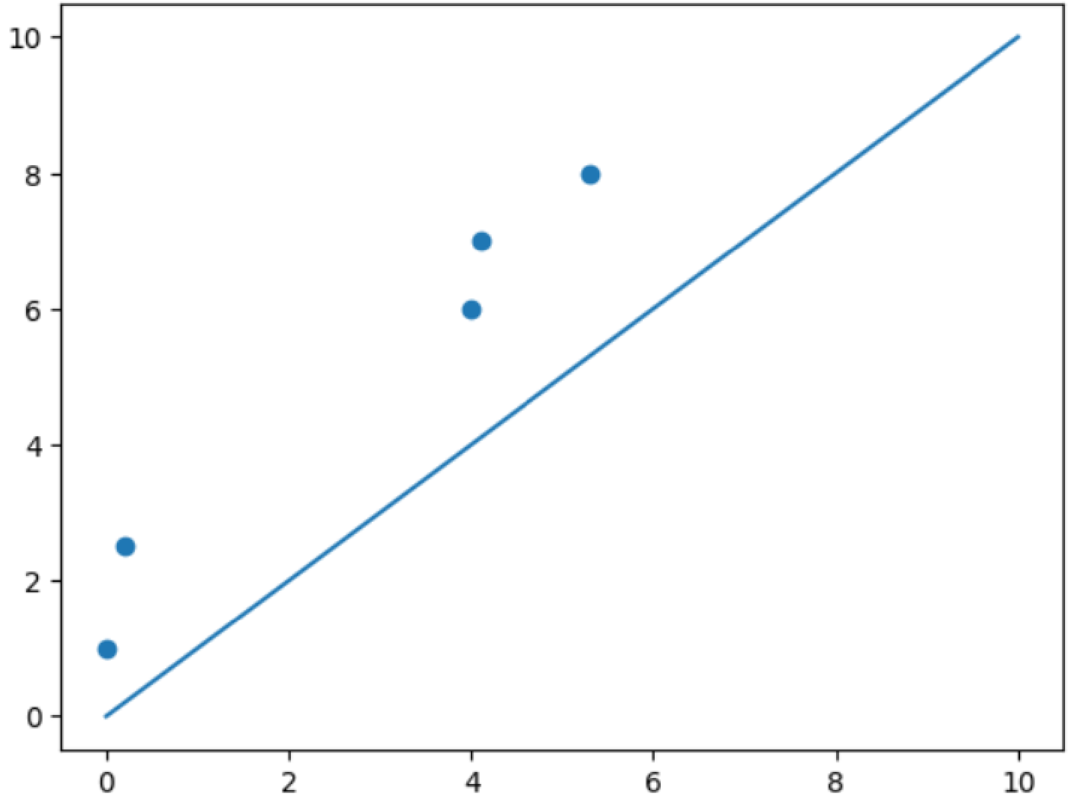
```
5.198
```

- **결과 비교 – 일치도**

데이터와 우리가 고안한 함수 $f(x)$를 비교하면 이렇게 차이가 난다

```
plt.scatter(y, y_pred)
plt.plot([0, 10], [0, 10])
```

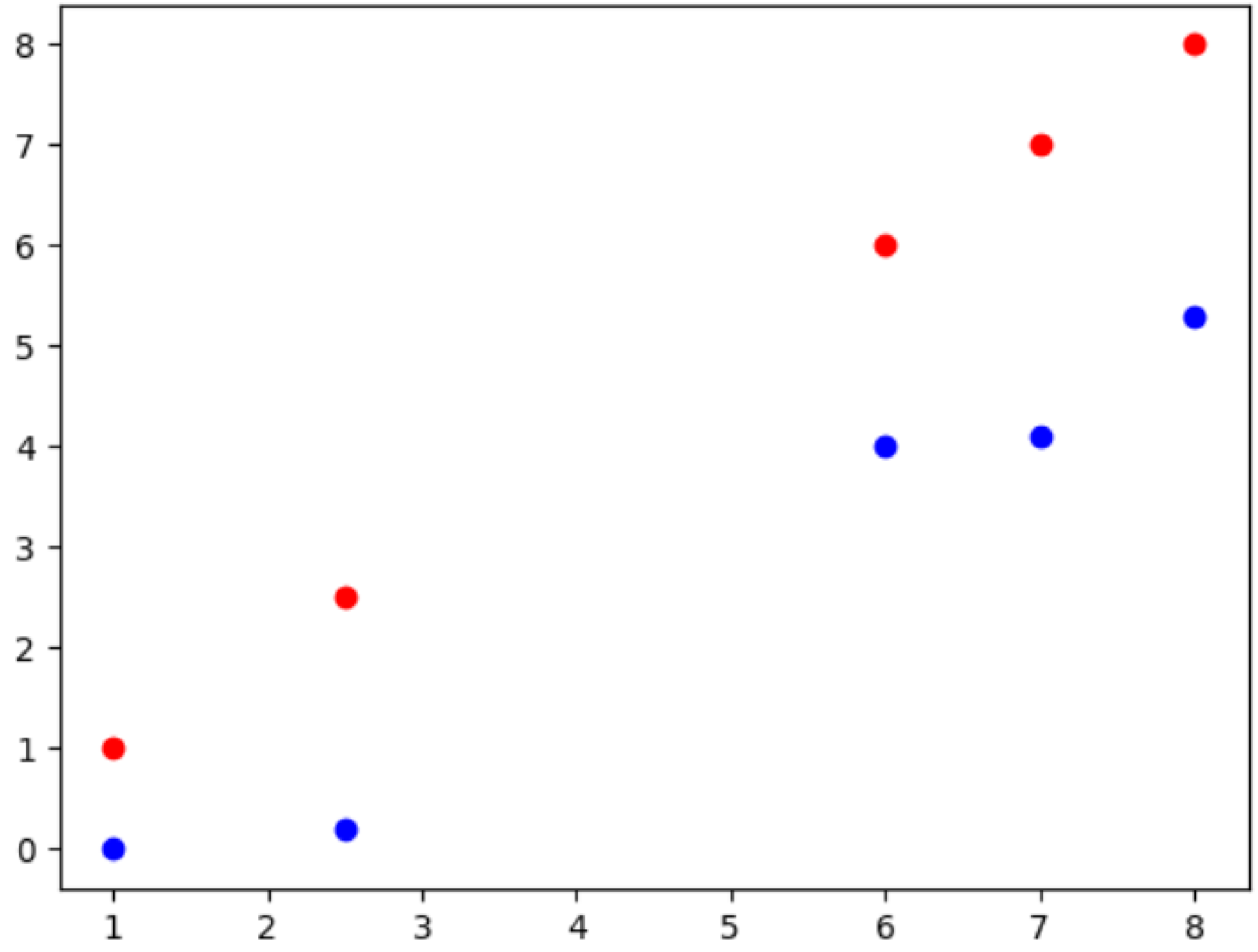
```
[<matplotlib.lines.Line2D at 0x7ee235d7df90>]
```



- **결과 비교 – 예측과 정답**

```
plt.scatter(x,y, color='blue')
plt.scatter(x, y_pred, color='red')
```

```
<matplotlib.collections.PathCollection at 0x7ee235aa57d0>
```



- **파라미터의 조작**

파라미터: 모델의 동작을 변경하는 변수

파라미터를 변경함으로써 더 좋은 함수 $f(x)$를 찾을 수 있을까?

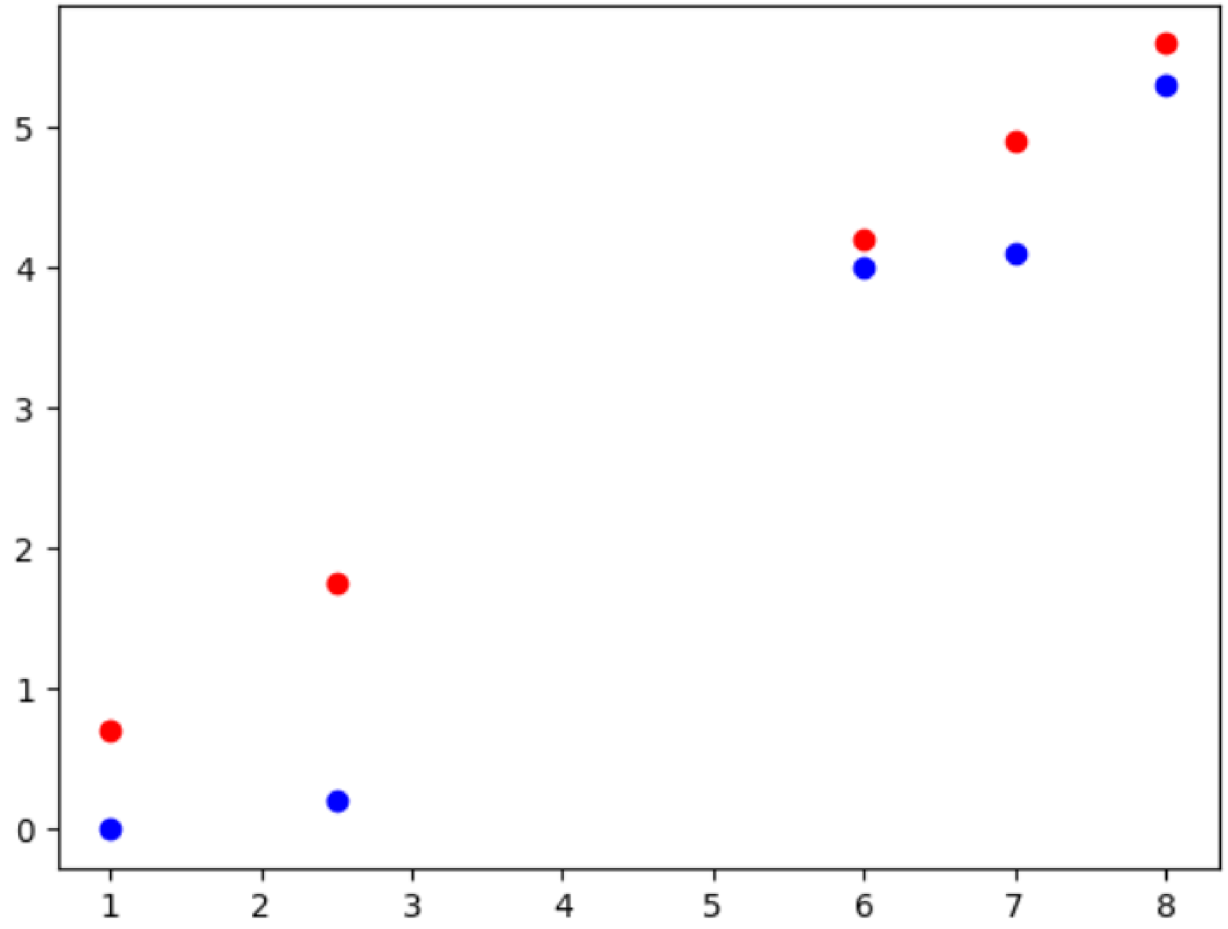
```python
# x - > y 예측
y_pred = f(x, w=0.7, b=-0.0)
x, y_pred
```

```
(array([1. , 2.5, 6. , 7. , 8. ]), array([0.7 , 1.75, 4.2 , 4.9 , 5.6 ]))
```

- **다른 파라미터의 결과**

```
plt.scatter(x,y, color='blue')
plt.scatter(x, y_pred, color='red')
```

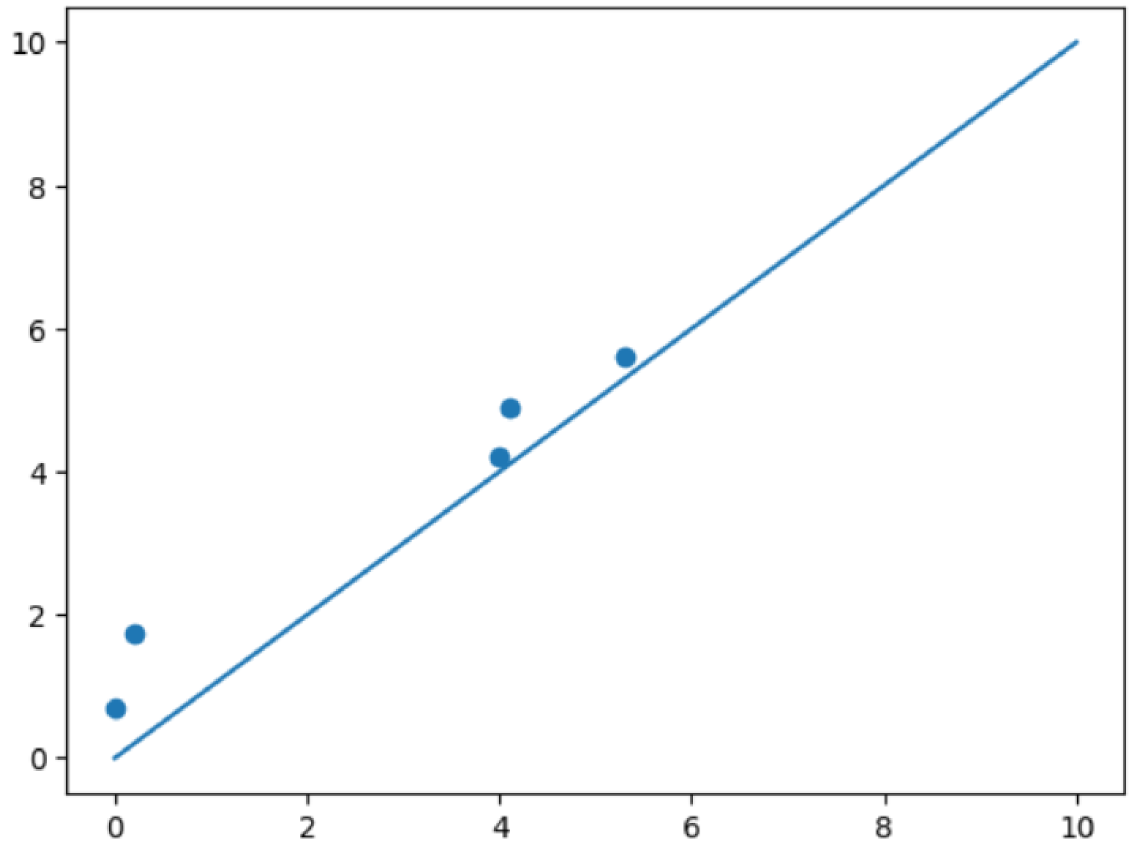
```
<matplotlib.collections.PathCollection at 0x7ee235b28790>
```



다른 파라미터에 의한 일치도

```
plt.scatter(y, y_pred)
plt.plot([0, 10], [0, 10])
```

```
[<matplotlib.lines.Line2D at 0x7ee235b75f90>]
```



- **오차는 어떻게 바뀌나 – 파라미터 준비**

함수를 바꾸어 가며 예측을 하고, 오차(mse)를 계산해 보자.

함수는 예측 **모델**, 모델의 동작을 결정하는 w, b는 **파라미터**

다양한 w 값을 적용하여 오차 측정

b는 0으로 고정

```python
w_list = np.arange(-2, 2, 0.2)
w_list
```

```
array([-2.0000000e+00, -1.8000000e+00, -1.6000000e+00, -1.4000000e+00,
       -1.2000000e+00, -1.0000000e+00, -8.0000000e-01, -6.0000000e-01,
       -4.0000000e-01, -2.0000000e-01, -4.4408921e-16,  2.0000000e-01,
        4.0000000e-01,  6.0000000e-01,  8.0000000e-01,  1.0000000e+00,
        1.2000000e+00,  1.4000000e+00,  1.6000000e+00,  1.8000000e+00])
```

- **오차는 어떻게 바뀌나 – 파라미터에 따른 오차 변화**

w에 따라 변화하는 오차 제곱을 확인해 보자

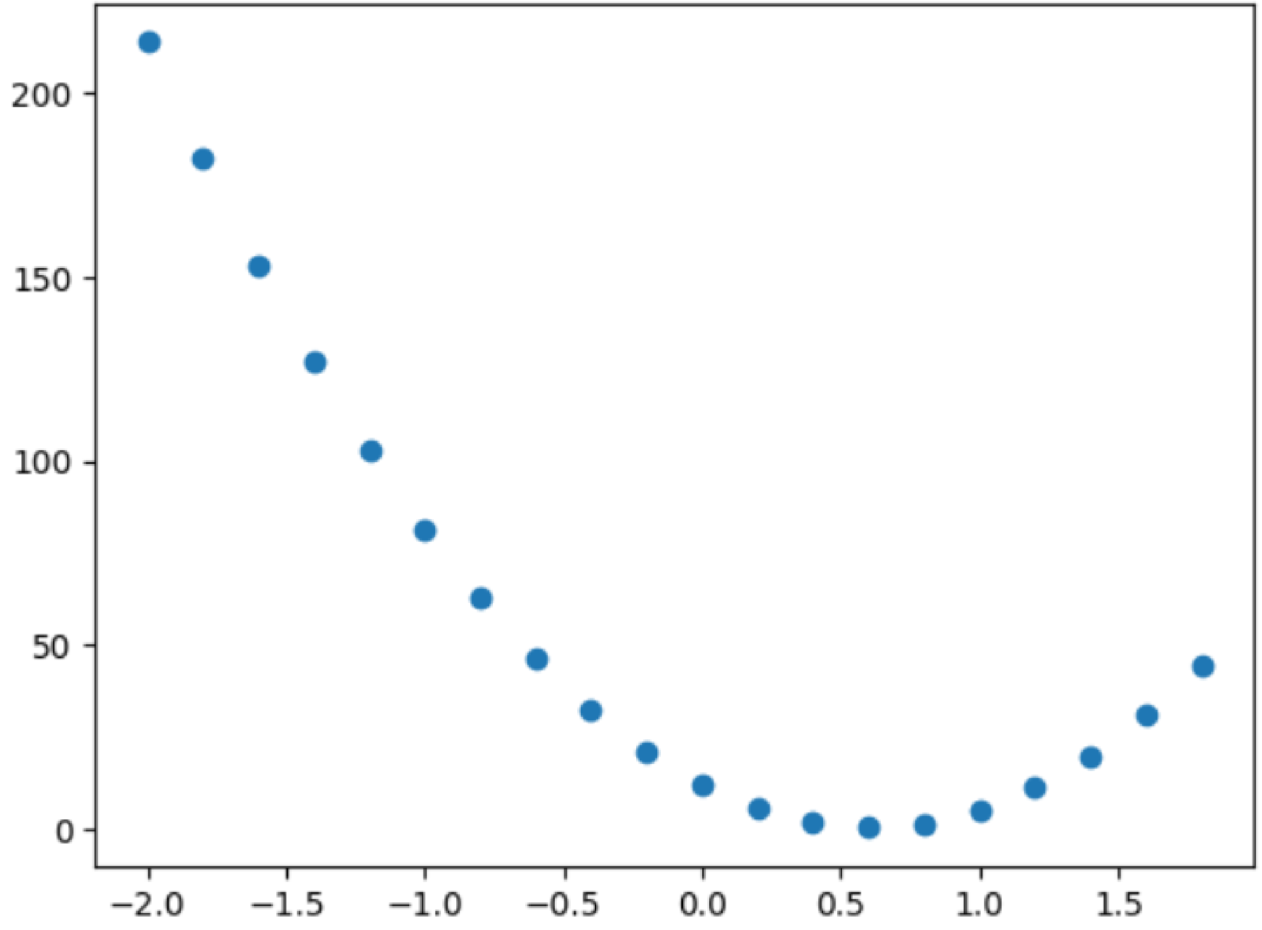
```
[ ]  mse_list = []
     for w in w_list:
         y_pred = f(x, w=w, b=0)
         mse_list.append(mse(y_pred, y))

     mse_list
```

```
[213.66800000000003,
 182.26999999999998,
 153.372,
 126.974,
 103.07600000000005,
 81.67800000000003,
 62.780000000000015,
 46.38200000000002,
 32.48400000000002,
 21.08600000000002
```

- 눈으로 봐야…

```
plt.scatter(w_list, mse_list)
```

```
<matplotlib.collections.PathCollection at 0x7ee2359fea90>
```



- **잘 보고 조정해 보자**

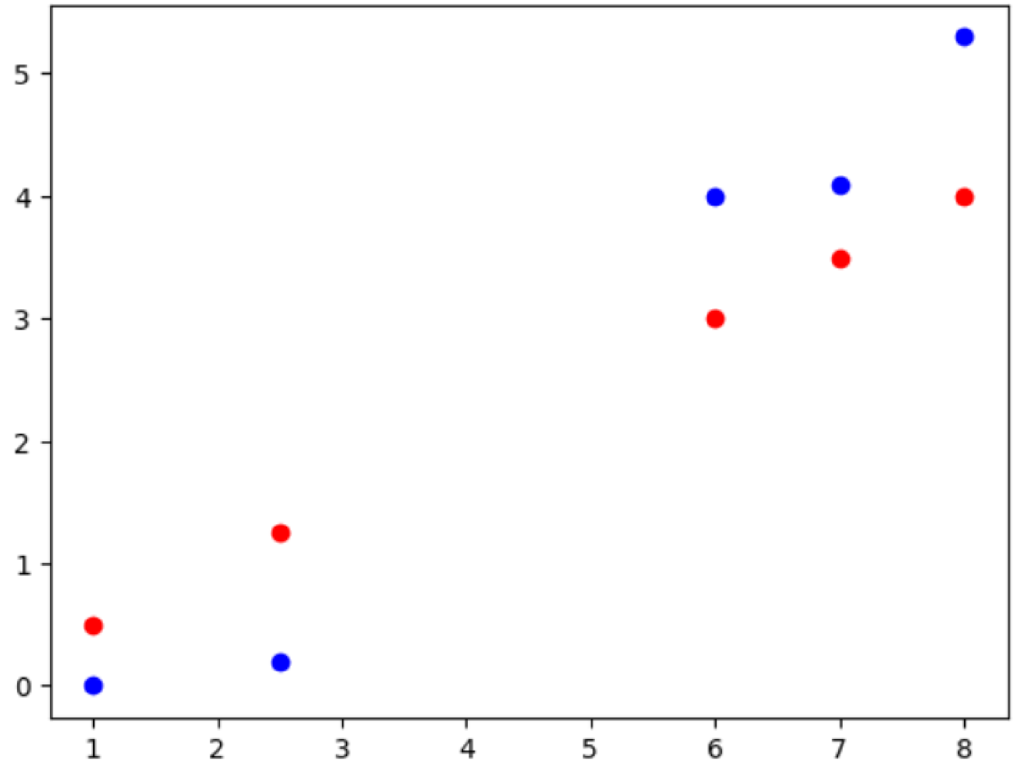
눈으로 볼 때 가장 작은 오차를 가지는 지점은 w가 0.5 부근

```
y_pred = f(x, w=0.5, b=0)
mse(y_pred, y)
```

```
0.8804999999999998
```

```
plt.scatter(x,y, color='blue')
plt.scatter(x, y_pred, color='red')
```

```
<matplotlib.collections.PathCollection at 0x7ee235a4fad0>
```



- 좀 더 전문가답게 바꾸려면?

더욱 멋지게 이 지점을 찾을 수 없을까? - 경사하강법 (Gradient Descent)

b는 0으로 고정하고 w만 찾아 보자.

오차 $E = \hat{y} - y = wx - y$

제곱 $E^2 = (wx - y)^2$

기울기 $\nabla_w E^2$: 변수 w 공간에서 정의되는 오차 E^2의 기울기

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

w는 기울기를 따라 오차가 줄어드는 방향으로 가야하므로 $-\nabla_w E^2$ 방향으로 이동해야 함 = 경사하강법

$w = w - \eta \cdot Ex$

- **경사하강법에 따른 변경 = 학습**

```
# 여러번 공부를 반복한다
공부횟수 = 1000
학습률 = 0.001



w = 5.0

for i in range(공부횟수):
    y_pred = f(x, w=w, b=0.0)
    # 오차계산
    E = y_pred - y
    # 오차를 줄이는 방향으로 공부 실시
    기울기_반대방향 = - (E * x).sum() / len(x)
    w = w + 학습률 * 기울기_반대방향

print(w)
```

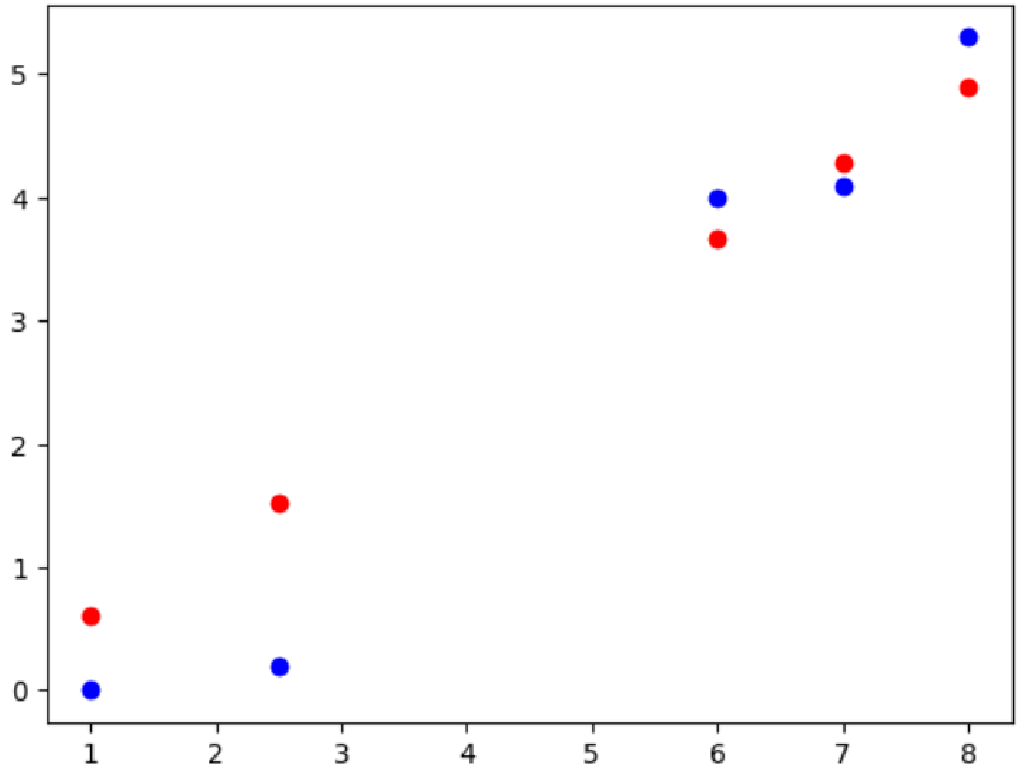
```
0.6118400000000715
```

찾은 결과를 확인해 보자

```
0.6118400000000715
```

```python
y_pred = f(x, w=w, b=0.0)
plt.scatter(x, y, color='blue')
plt.scatter(x, y_pred, color='red')
```

```
<matplotlib.collections.PathCollection at 0x7ee2358df290>
```



• 학습률에 따른 오차 변화

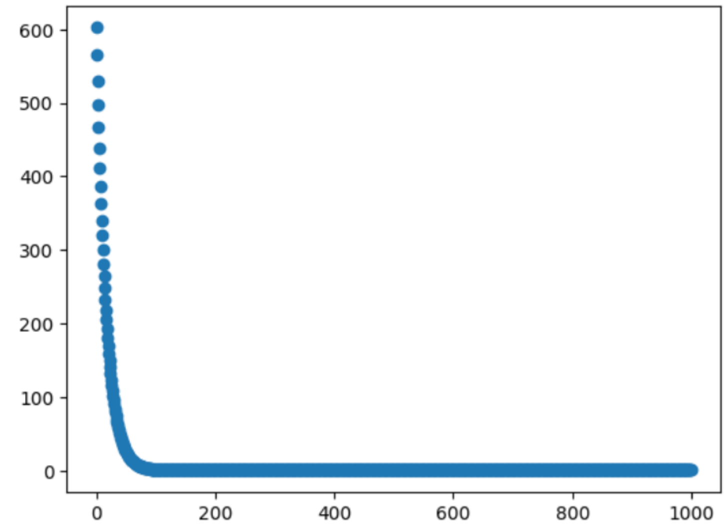
오차의 변화를 추적해 보자

```
[ ] # 여러번 공부를 반복한다
    공부횟수 = 1000
    학습률 = 0.001
    mse_list = []

    w = 5.0

    for i in range(공부횟수):
        y_pred = f(x, w=w, b=0.0)
        # 오차계산
        E = y_pred - y
        mse_list.append( mse(y_pred, y) )
        # 오차를 줄이는 방향으로 공부 실시
        기울기_반대방향 = - (E * x).sum() / len(x)
        w = w + 학습률 * 기울기_반대방향

    mse_list
```

```
0.49438076448912893,
0.49408781667387985,
0.4938128920152329З,
0.4935548816666394,
```

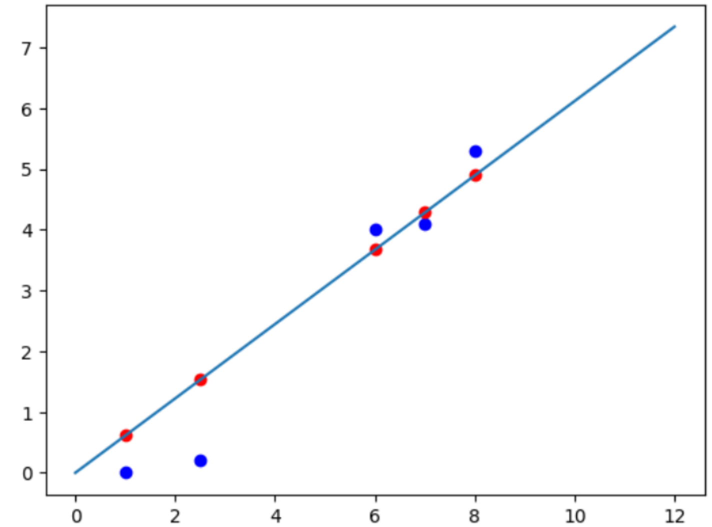
```
plt.scatter(range(공부횟수), mse_list)
```

```
<matplotlib.collections.PathCollection at 0x7ee2356e2210>
```




```
plt.scatter(x, y, color='blue')
plt.scatter(x, f(x, w, 0), color='red')
plt.plot([0,12], [f(0, w, 0), f(12, w, 0) ])
print(w)
```

```
0.6118400000000715
```



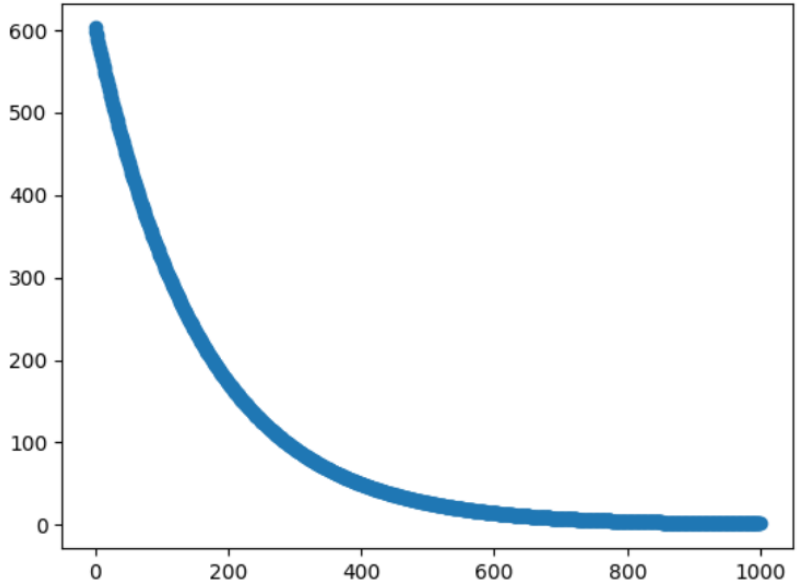
• 학습률에 따른 오차 변화

학습률 (하이퍼 파라미터 = 학습을 조절하는 파라미터)

변경

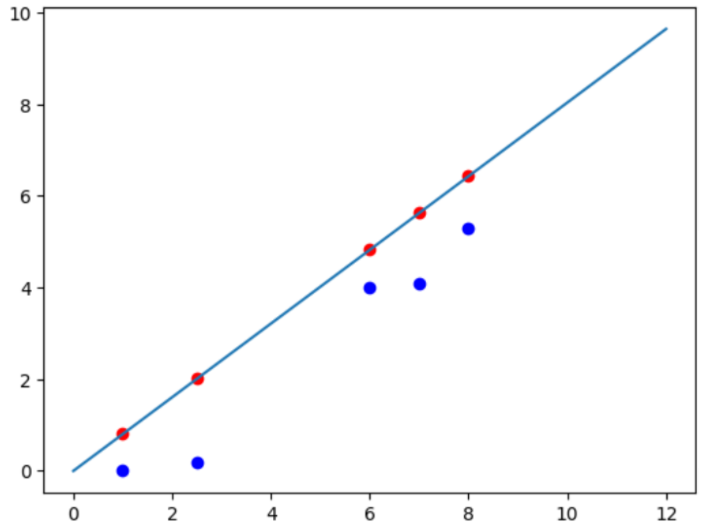
```
# 여러번 공부를 반복한다
공부횟수 = 1000
학습률 = 0.0001
mse_list = []

w = 5.0

for i in range(공부횟수):
    y_pred = f(x, w=w, b=0.0)
    # 오차계산
    E = y_pred - y
    mse_list.append( mse(y_pred, y) )
    # 오차를 줄이는 방향으로 공부 실시
    기울기_반대방향 = - (E * x).sum() / len(x)
    w = w + 학습률 * 기울기_반대방향

print(w)
plt.scatter(range(공부횟수), mse_list)
```

```
0.8037012155317347
<matplotlib.collections.PathCollection at 0x7ee2355b6190>
```




```
plt.scatter(x, y, color='blue')
plt.scatter(x, f(x, w, 0), color='red')
plt.plot([0,12], [f(0, w, 0), f(12, w, 0) ])
print(w)
```

```
0.8037012155317347
```



같은 횟수로 공부해도 성적이 좋지 못함

```
[ ] mse(f(x, w, 0), y)
```

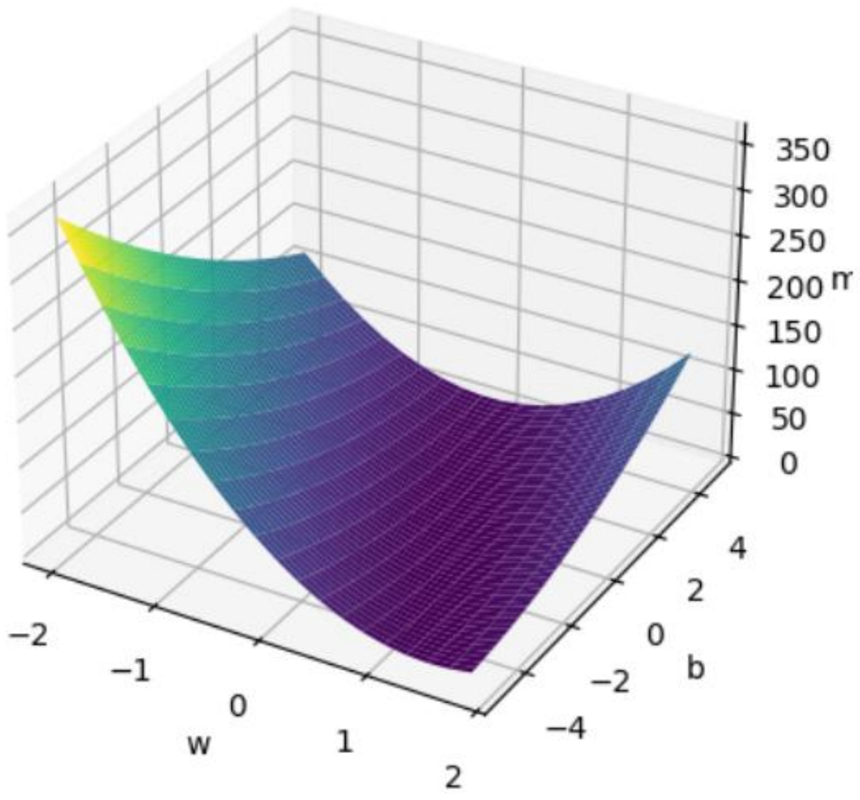
```
1.6399543882910863
```

• 두 파라미터를 함께 추적

```python
w_list = np.arange(-2, 2, 0.2)
b_list = np.arange(-5, 5, 0.2)

mse_list = []
for b in b_list:
    for w in w_list:
        y_pred = f(x, w=w, b=b)
        mse_list.append( mse(y_pred, y))
```

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

X, Y = np.meshgrid(w_list, b_list)

fig = plt.figure()
ax = plt.axes(projection='3d')

mse_list = np.array(mse_list).reshape(len(b_list), len(w_list))
print(mse_list.shape)
ax.plot_surface(X, Y, mse_list, cmap='viridis')

ax.set_xlabel('w')
ax.set_ylabel('b')
ax.set_zlabel('mse')

plt.show()
```

```
(50, 20)
```



- **경사하강법 다시 보기**

경사하강법 (Gradient Descent)

오차 $E = \hat{y} - y = wx - y$

제곱 $E^2 = (wx + b - y)^2$

기울기 $\nabla_{w,b}E^2$: (w, b) 공간에서 정의되는 오차 E^2의 기울기

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

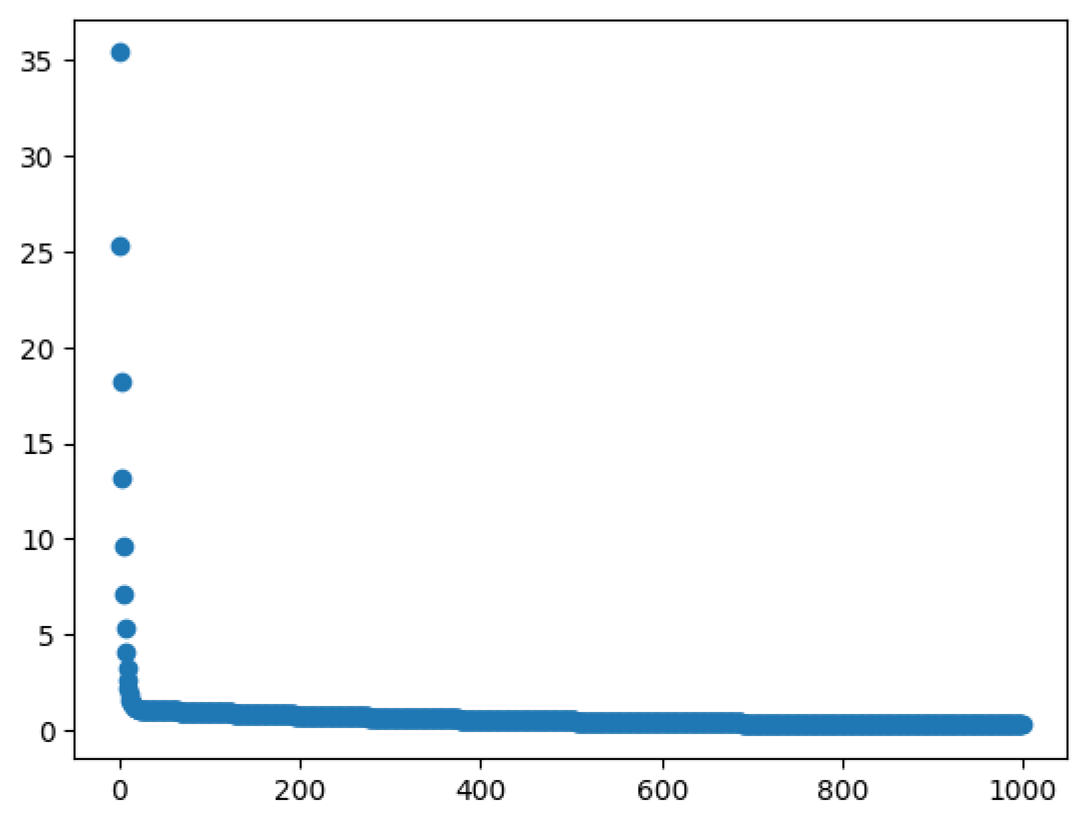
w와 b는 기울기를 따라 오차가 줄어드는 방향으로 가야하므로 $-\nabla_{w,b}E^2$ 방향으로 이동해야 함 = 경사하강법

$w = w - \eta \cdot Ex$

$b = b - \eta \cdot E$

학습 실시

```python
# 여러번 공부를 반복한다
공부횟수 = 1000
학습률 = 0.005
mse_list = []

w = 1.5
b = 1.

for i in range(공부횟수):
    y_pred = f(x, w=w, b=b)
    # 오차계산
    E = y_pred - y
    mse_list.append( mse(y_pred, y) )
    # 오차를 줄이는 방향으로 공부 실시
    w기울기_반대방향 = - (E * x).sum() / len(x)
    b기울기_반대방향 = - E.sum() / len(x)
    w = w + 학습률 * w기울기_반대방향
    b = b + 학습률 * b기울기_반대방향

print(w, b)
plt.scatter(range(공부횟수), mse_list)
```



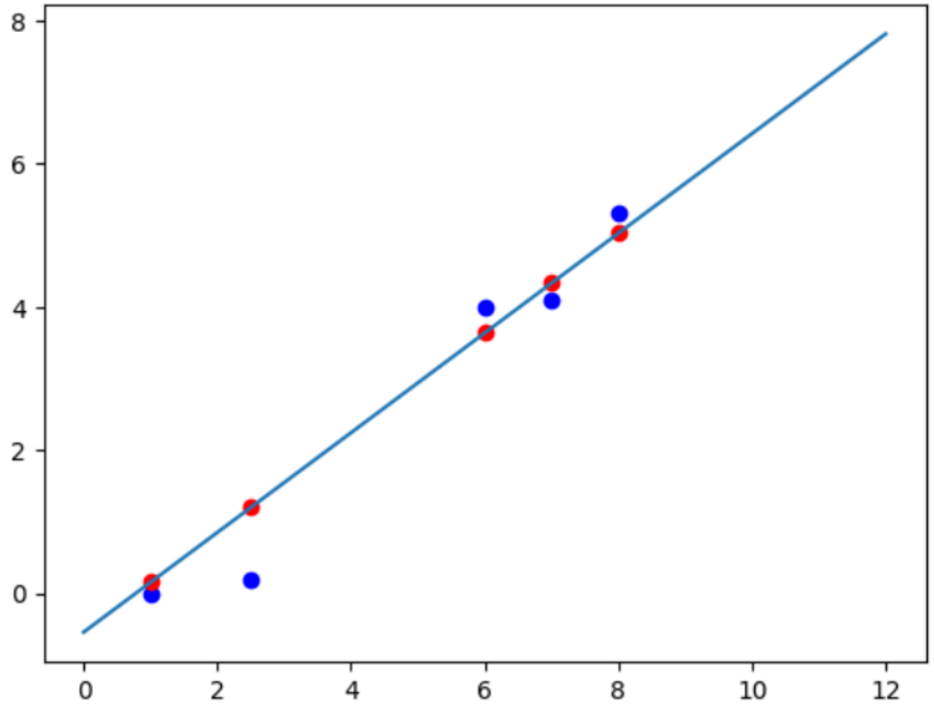
- **학습 결과**

```
[ ] y_pred = f(x, w=w, b=b)
    mse(y_pred, y)
```

```
0.2562790700131511
```

```
plt.scatter(x, y, color='blue')
plt.scatter(x, f(x, w, b), color='red')
plt.plot([0,12], [f(0, w, b), f(12, w, b) ])
print(w, b)
mse(f(x, w=w, b=b), y)
```

```
0.696098441937211 -0.5421564244674564
0.2562790700131511
```



- **sklearn**

```python
from sklearn import linear_model
X = x[:, np.newaxis]

x, X
```

```
(array([1. , 2.5, 6. , 7. , 8. ]),
 array([[1. ],
        [2.5],
        [6. ],
        [7. ],
        [8. ]]))
```

```python
reg = linear_model.LinearRegression()
reg.fit(X, y)
```

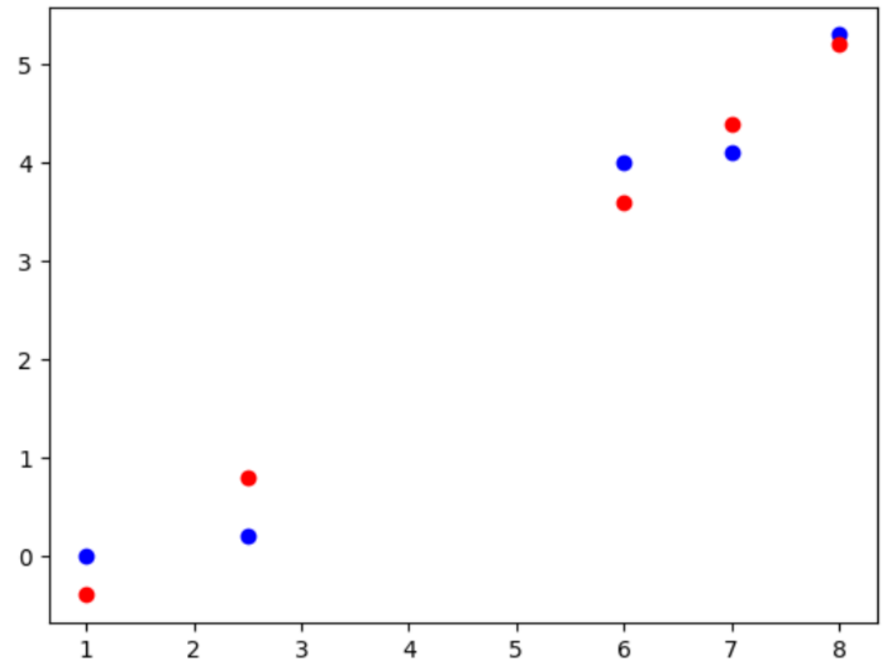
```
▾ LinearRegression
LinearRegression()
```

```python
y_pred = reg.predict(X)
y_pred
```

```
array([-0.4,  0.8,  3.6,  4.4,  5.2])
```

```python
plt.scatter(x, y, color='blue')
plt.scatter(x, y_pred, color='red')
```

```
<matplotlib.collections.PathCollection at 0x7ee233ae2d90>
```




```python
print("Coefficients (w):", reg.coef_)
print("Intercept (b):", reg.intercept_)
```

```
Coefficients (w): [0.8]
Intercept (b): -1.2000000000000002
```

mse(y_pred, y)

```python
mse(y_pred, y)
```

```
0.15599999999999997
```

- **다변량 회귀**

다변량 회귀분석

예측 모델

$$y = f(x_1, x_2, x_3, \cdots, x_n) = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + b$$

$$x_0 = 1$$

$$b = w_0x_0$$

$$\theta = (w_0, w_1, w_2, w_3, \cdots, w_n)^T$$

$$\hat{y} = \theta_0x_0 + \theta_1x_1 + \theta_2x_2 + \theta_3x_3 + \theta_4x_4 + \theta_5x_5 + \cdots \theta_nx_n = \theta^T\mathbf{x}$$

오차는

m: 데이터 개수

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

데이터 준비

```
[ ] import pandas as pd

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

	Country	Year	Status	Life expectancy	Adult mortality	Infant deaths	Alcohol	Percentage expenditure	Hepatitis B	Measles	...	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Po
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	...	6.0	8.16	65.0	0.1	584.259210	33
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	...	58.0	8.18	62.0	0.1	612.696514	
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	...	62.0	8.13	64.0	0.1	631.744976	31
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	...	67.0	8.52	67.0	0.1	669.959000	3
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	...	68.0	7.87	68.0	0.1	63.537231	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2933	Zimbabwe	2004	Developing	44.3	723.0	27	4.36	0.000000	68.0	31	...	67.0	7.13	65.0	33.6	454.366654	12
2934	Zimbabwe	2003	Developing	44.5	715.0	26	4.06	0.000000	7.0	998	...	7.0	6.52	68.0	36.7	453.351155	12
2935	Zimbabwe	2002	Developing	44.8	73.0	25	4.43	0.000000	73.0	304	...	73.0	6.53	71.0	39.8	57.348340	
2936	Zimbabwe	2001	Developing	45.3	686.0	25	1.72	0.000000	76.0	529	...	76.0	6.16	75.0	42.1	548.587312	12
2937	Zimbabwe	2000	Developing	46.0	665.0	24	1.68	0.000000	79.0	1483	...	78.0	7.10	78.0	43.5	547.358878	12

2938 rows × 22 columns

- 일부 데이터만 사용하기 (어떤 데이터만 사용할까?)

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

	Life expectancy	Alcohol	Percentage expenditure	Total expenditure	Hepatitis B	Measles	Polio	BMI	GDP	Thinness 1-19 years	Thinness 5-9 years
0	65.0	0.01	71.279624	8.16	65.0	1154	6.0	19.1	584.259210	17.2	17.3
1	59.9	0.01	73.523582	8.18	62.0	492	58.0	18.6	612.696514	17.5	17.5
2	59.9	0.01	73.219243	8.13	64.0	430	62.0	18.1	631.744976	17.7	17.7
3	59.5	0.01	78.184215	8.52	67.0	2787	67.0	17.6	669.959000	17.9	18.0
4	59.2	0.01	7.097109	7.87	68.0	3013	68.0	17.2	63.537231	18.2	18.2
...	...	...	...	...	...	...	...	...	...	...	...
2933	44.3	4.36	0.000000	7.13	68.0	31	67.0	27.1	454.366654	9.4	9.4
2934	44.5	4.06	0.000000	6.52	7.0	998	7.0	26.7	453.351155	9.8	9.9
2935	44.8	4.43	0.000000	6.53	73.0	304	73.0	26.3	57.348340	1.2	1.3
2936	45.3	1.72	0.000000	6.16	76.0	529	76.0	25.9	548.587312	1.6	1.7
2937	46.0	1.68	0.000000	7.10	79.0	1483	78.0	25.5	547.358878	11.0	11.2

2938 rows × 11 columns

- **데이터 확인**

```python
life.shape
```

```
(2938, 11)
```

```python
life.isnull().sum()
```

	0
Life expectancy	10
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

dtype: int64

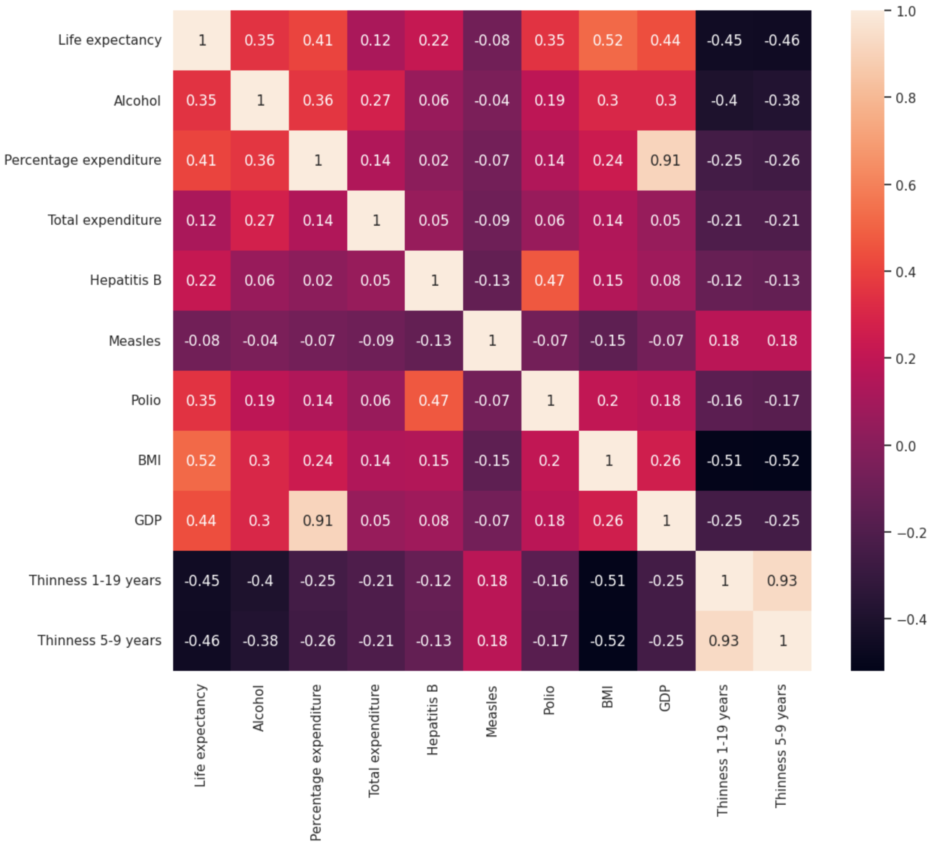
```python
life = life.dropna()
life.shape
```

```
(1853, 11)
```

- # 상관관계 분석

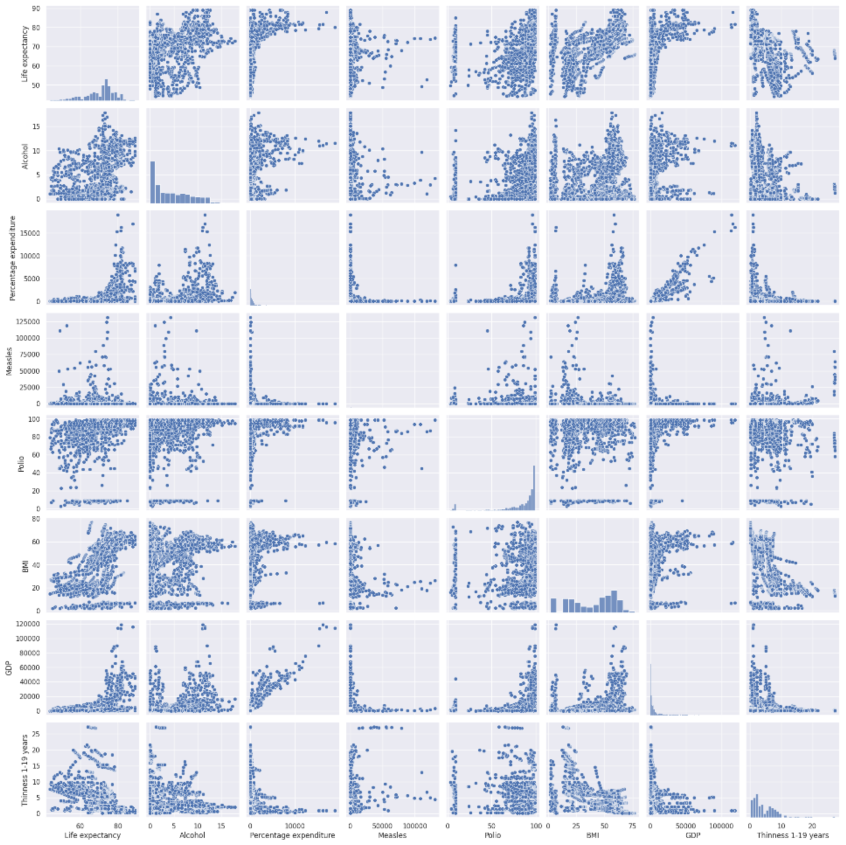
```python
import seaborn as sns
```

```python
sns.set(rc = {'figure.figsize':(12,10)})
correlation_matrix = life.corr().round(2)
sns.heatmap( data = correlation_matrix, annot = True)
```



pairplot

```python
sns.pairplot(life[['Life expectancy', 'Alcohol', 'Percentage expenditure', 'Measles', 'Polio', 'BMI', 'GDP', 'Thinness 1-19 years']])
plt.show()
```



- f(X) → y: X와 y 준비

```
# f(X) --> y

y = life[['Life expectancy']]

X = life[['Alcohol', 'Percentage expenditure', 'Measles', 'Polio', 'BMI', 'GDP', 'Thinness 1-19 years']]

print(X), print(y)
```

```
      Alcohol  Percentage expenditure  Measles  Polio   BMI         GDP  \
0        0.01               71.279624     1154    6.0  19.1  584.259210
1        0.01               73.523582      492   58.0  18.6  612.696514
2        0.01               73.219243      430   62.0  18.1  631.744976
3        0.01               78.184215     2787   67.0  17.6  669.959000
4        0.01                7.097109     3013   68.0  17.2   63.537231
...       ...                     ...      ...    ...   ...         ...
2933     4.36                0.000000       31   67.0  27.1  454.366654
2934     4.06                0.000000      998    7.0  26.7  453.351155
2935     4.43                0.000000      304   73.0  26.3   57.348340
2936     1.72                0.000000      529   76.0  25.9  548.587312
2937     1.68                0.000000     1483   78.0  25.5  547.358878

[1853 rows x 7 columns]
      Life expectancy
0                65.0
1                59.9
2                59.9
3                59.5
4                59.2
...               ...
2933             44.3
2934             44.5
2935             44.8
2936             45.3
2937             46.0

[1853 rows x 1 columns]
(None, None)
```

X

y

• 선형회귀모델 준비 – 예측과 정답 비교

```python
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)
```

훈련용과 검증용 데이터 분리

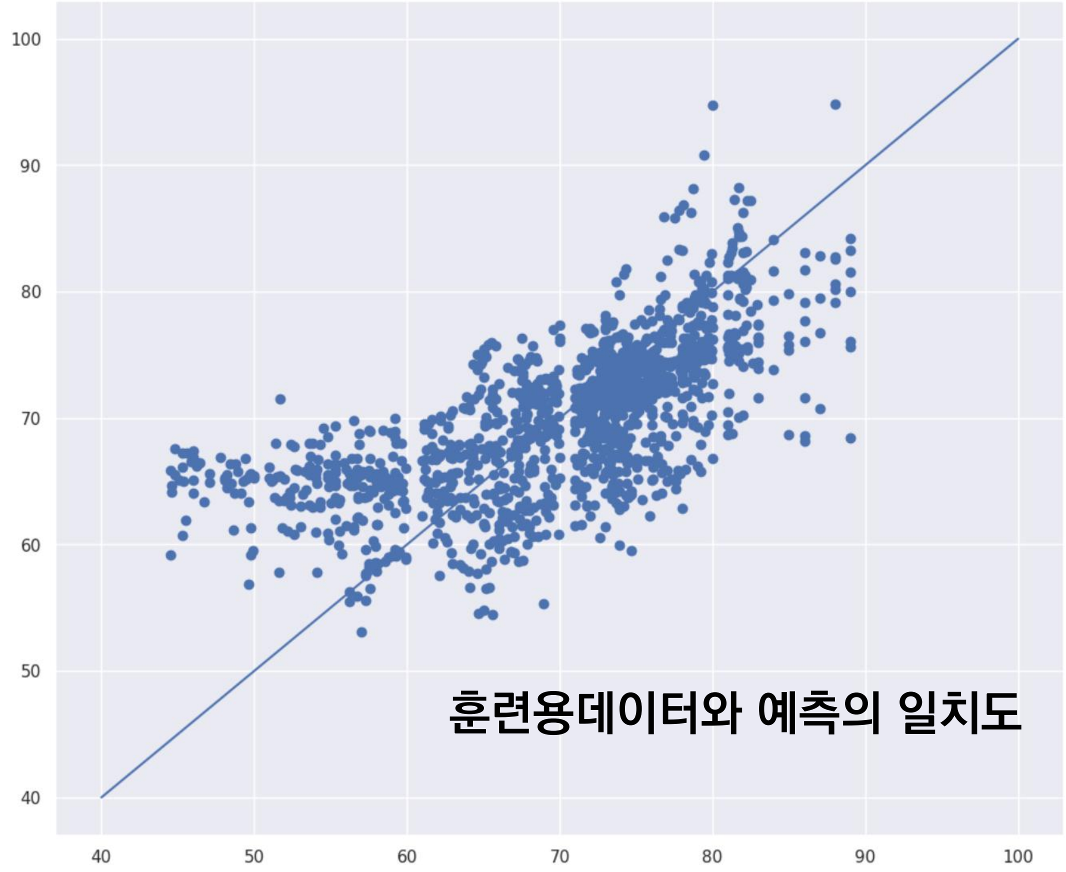
```python
from sklearn.linear_model import LinearRegression
lin_model = LinearRegression()
lin_model.fit (X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```python
y_train_predicted = lin_model.predict(X_train)
```

```python
plt.scatter(y_train, y_train_predicted)
xy_range = [40, 100]
plt.plot(xy_range, xy_range)
```

```
[<matplotlib.lines.Line2D at 0x7ee22efe0ad0>]
```



훈련용데이터와 예측의 일치도

• 선형회귀모델의 성능 – 훈련용 데이터로 검증

```
[ ] y_test_predicted = lin_model.predict(X_test)
```

```
[ ] plt.scatter(y_test, y_test_predicted)
    plt.plot(xy_range, xy_range)
```

학습데이터와 검증 데이터를 활용한 예측 결과의 오차 측정

```
[ ] mse(y_train, y_train_predicted)
```

```
40.54757544458645
```

```
[ ] mse(y_test, y_test_predicted)
```

```
43.4261297324397
```

- **일반화에 대한 이해**

만약 학습용 데이터에 대한 예측의 mse와 검증용 데이터에 대한 예측의 mse가 다음과 같다면 어떻게 해석해야 할까?

학습용 30.0 검증용 31.0

학습용 30.0 검증용 71.0

학습용 10.0 검증용 32341.0

학습용 40.0 검증용 35.0

• 데이터 정제

```
[ ] 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)
```

```
[ ] from sklearn.linear_model import LinearRegression
    lin_model = LinearRegression()
    lin_model.fit (X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
[ ] y_train_predicted = lin_model.predict(X_train)
```

```
plt.scatter(y_train, y_train_predicted)
xy_range = [40, 100]
plt.plot(xy_range, xy_range)
```

```
[<matplotlib.lines.Line2D at 0x7ee22efe0ad0>]
```

• 선형회귀모델 준비

```
[ ] 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)
```

```
[ ] from sklearn.linear_model import LinearRegression
    lin_model = LinearRegression()
    lin_model.fit (X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
[ ] y_train_predicted = lin_model.predict(X_train)
```

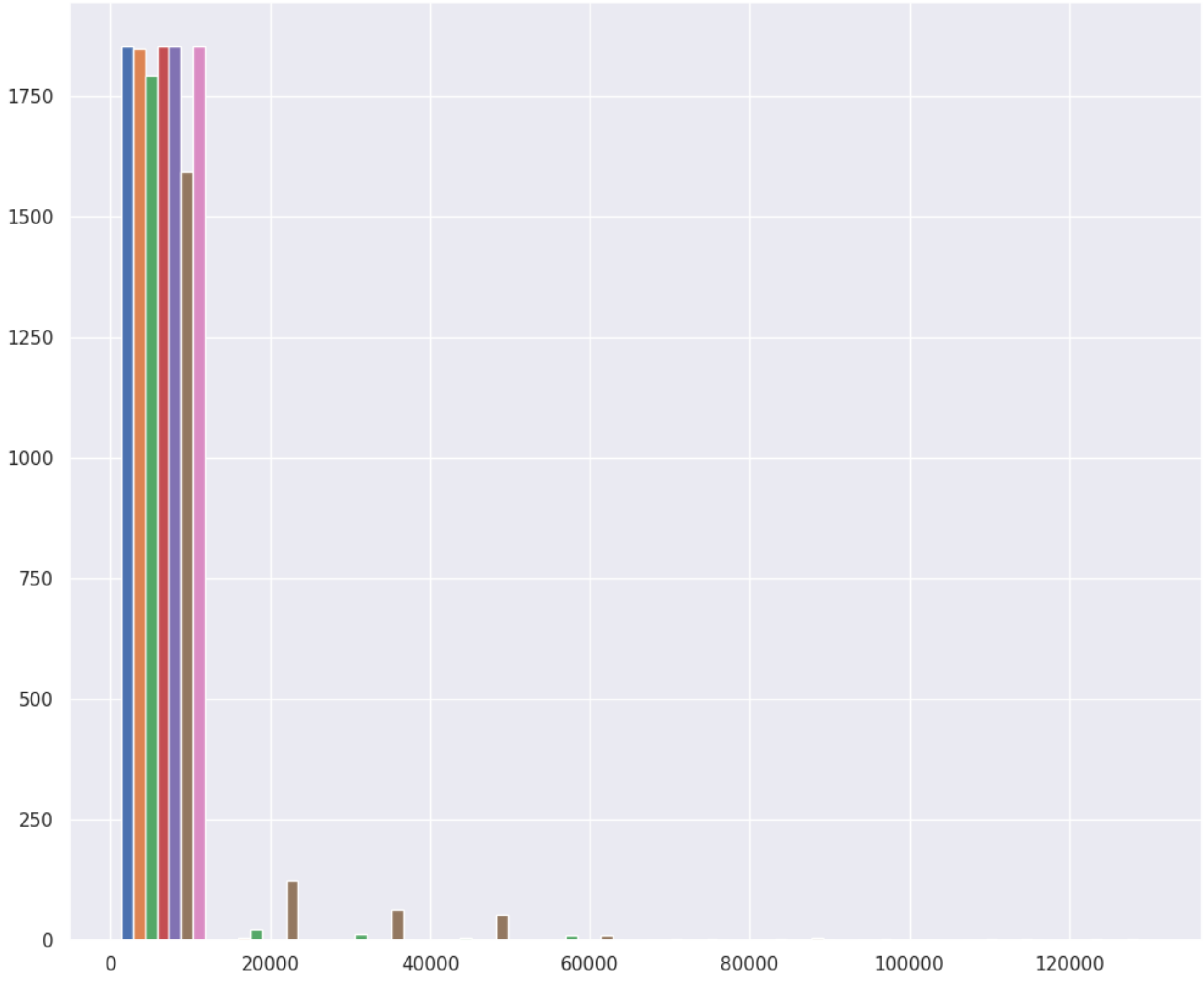
```
plt.scatter(y_train, y_train_predicted)
xy_range = [40, 100]
plt.plot(xy_range, xy_range)
```

```
[<matplotlib.lines.Line2D at 0x7ee22efe0ad0>]
```

- # 데이터 정제

```python
plt.hist(X, bins = 10)
```

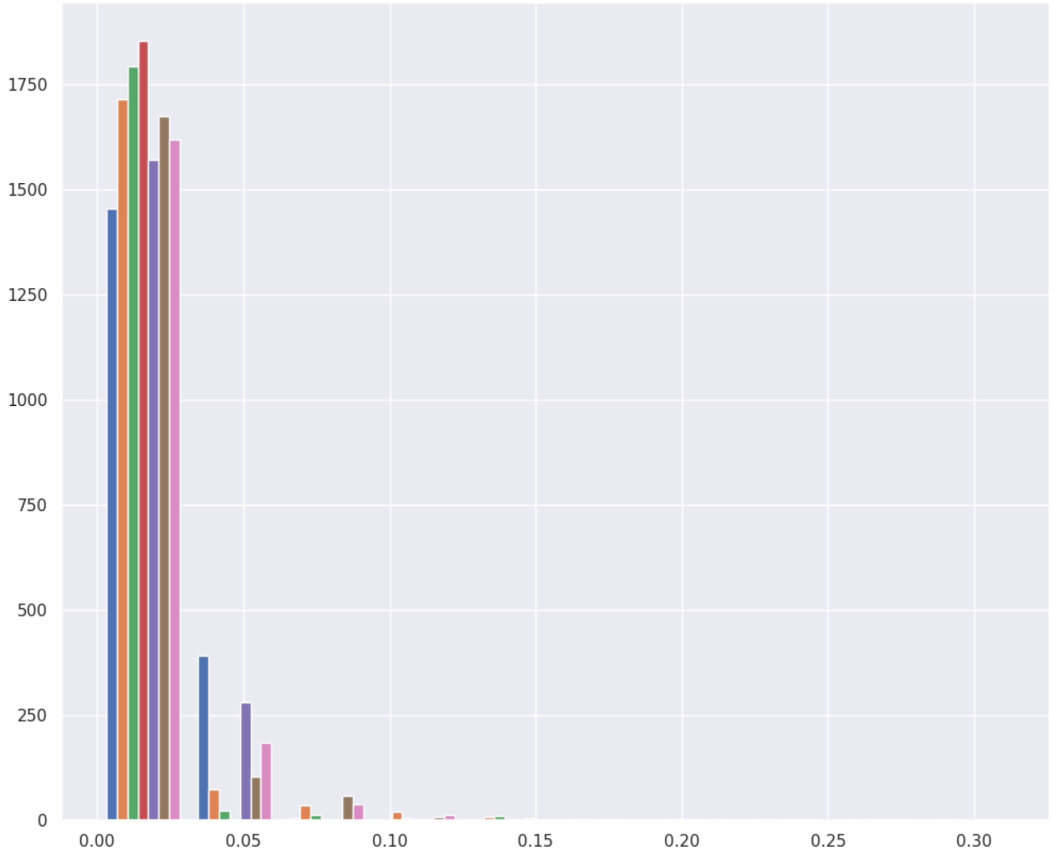
```
(array([[1.853e+03, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+
          0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00],
         [1.848e+03, 5.000e+00, 0.000e+00, 0.000e+00, 0.000e+
          0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00],
         [1.794e+03, 2.300e+01, 1.100e+01, 5.000e+00, 9.000e+
          2.000e+00, 1.000e+00, 2.000e+00, 3.000e+00],
         [1.853e+03, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+
          0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00],
         [1.853e+03, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+
          0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00],
         [1.593e+03, 1.240e+02, 6.200e+01, 5.200e+01, 1.000e+
          5.000e+00, 0.000e+00, 3.000e+00, 1.000e+00],
         [1.853e+03, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+
          0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00]]),
 array([     0. ,  13144.1,  26288.2,  39432.3,  52576.4,  6
          78864.6,  92008.7, 105152.8, 118296.9, 131441. ]),
 <a list of 7 BarContainer objects>)
```



- 정규화 – 데이터의 범위를 조정

```
[ ] from sklearn.preprocessing import normalize
    n_X = normalize(X, axis = 0)
```

```
plt.hist(n_X, bins = 10)
```



• 표준화 – 데이터의 평균과 분산을 조정

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
s_X = scaler.fit_transform(X)

plt.hist(s_X, bins = 10)
```

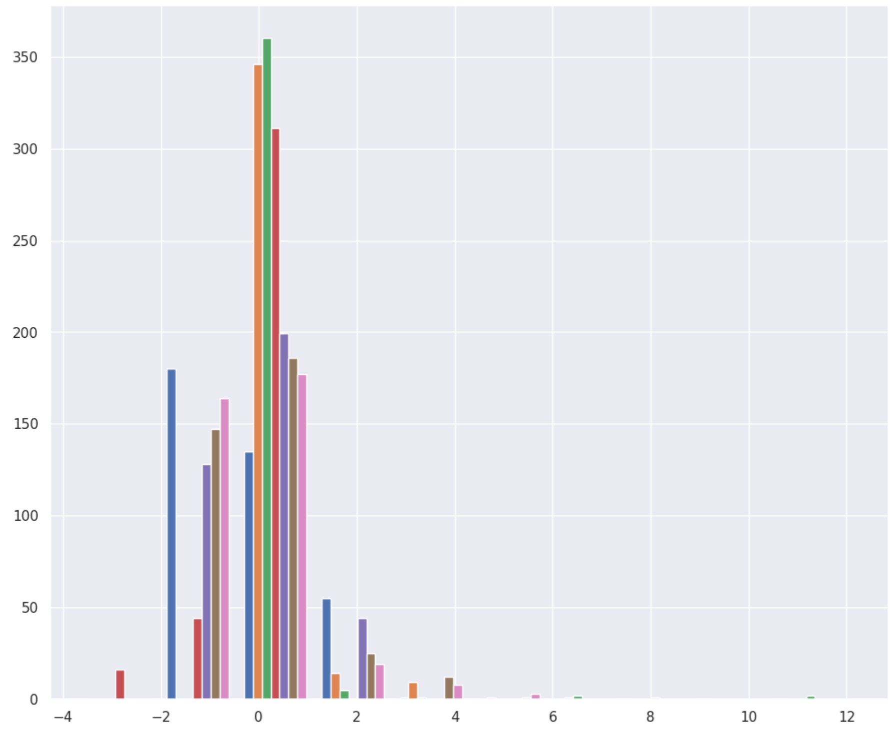
```
        [0.000e+00, 7.410e+02, 1.066e+03, 4.600e+01
         0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00
        [0.000e+00, 1.057e+03, 6.580e+02, 9.300e+01
         2.000e+00, 2.000e+00, 0.000e+00, 0.000e+00
        [0.000e+00, 9.410e+02, 7.730e+02, 1.050e+02
         0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00
 array([-3.77113931, -2.03651125, -0.30188319,  1.4
         4.90200099,  6.63662904,  8.3712571 , 10.1
        13.57514128]),
 <a list of 7 BarContainer objects>)
```

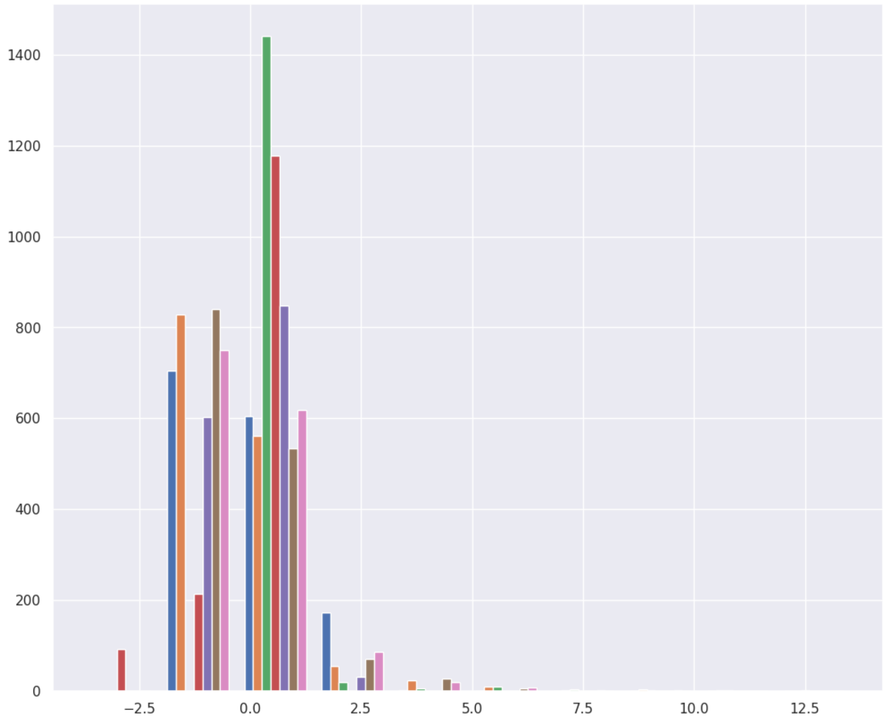
- **정제된 데이터로 예측**

```
[ ] sXtrain, sXtest, y_train, y_test = train_test_split(s_X, y, test_size = 0.2)
```

```
[ ] plt.hist(sXtrain, bins=10)
```

```
[ ] plt.hist(sXtest, bins=10)
```



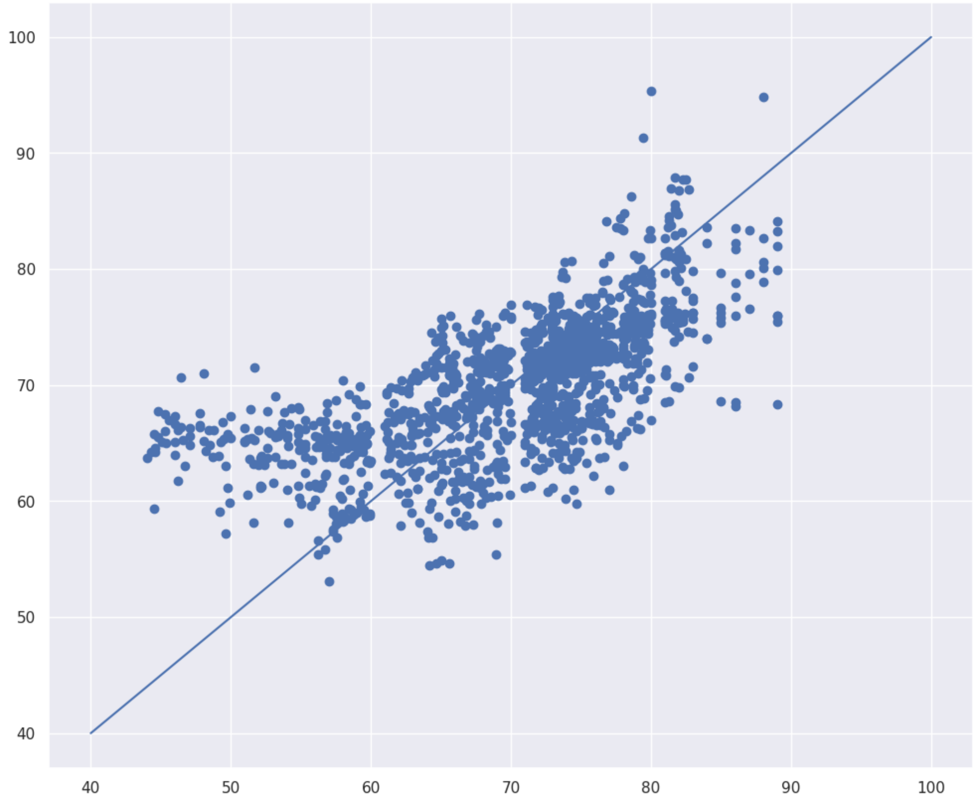


Sklearn 모델 준비 – 테스트 데이터로 훈련

```python
lin_model.fit(sXtrain, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

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




```python
[ ] mse(y_train, y_s_train_predicted)
```

```
41.84770040329405
```

Sklearn 모델 준비 – 검증용 데이터로 성능 평가

```
y_s_test_predicted = lin_model.predict(sXtest)
plt.scatter(y_test, y_s_test_predicted)
xy_range = [40, 100]
plt.plot(xy_range, xy_range)
```


```
[ ] mse(y_test, y_s_test_predicted)
```

```
38.30721475442583
```