코딩으로 확인하는 평균 제곱근 오차(Root Mean Squared Error, RMSE)

In [1]:

```
import numpy as np

#가상의 기울기 a와 y 절편 b
fake_a_b=[3,76]  #임의의 값을 a,b 값으로 설정

# x 값과 y값
data = [[2, 81], [4, 93], [6, 91], [8, 97]]

print(data[0],  data[1], data[2], data[3])
print(data[0][0],  data[1][0], data[2][0], data[3][0])
print(data[0][1],  data[1][1], data[2][1], data[3][1])
```

```
[2, 81] [4, 93] [6, 91] [8, 97]
2 4 6 8
81 93 91 97
```

In [2]:

```
x = [i[0] for i in data] #data리스트에서 0번째 요소를 분리
y = [i[1] for i in data] #data리스트에서 1번째 요소를 분리

print("fake_a_b :", fake_a_b)
print("x        :", x)
print("y        :", y)
```

```
fake_a_b : [3, 76]
x        : [2, 4, 6, 8]
y        : [81, 93, 91, 97]
```

In [3]:

```
fake_a_b[0]
```

Out[3]:

```
3
```

In [4]:

```
fake_a_b[1]
```

Out[4]:

```
76
```

In [5]:

```
a = fake_a_b[0]
b = fake_a_b[1]
print("a :", a)
print("b :", b)
```

```
a : 3
b : 76
```

● predict( ) 함수 정의 : 일차 방정식 y = ax + b

In [6]:

```
#a : fake_a_b[0], b : fake_a_b[1]

def predict(x):
    return a*x + b
```

● predict()함수 호출 : 일차 방정식 y = ax + b

In [7]:

```
# 예측값이 들어갈 빈 리스트
predict_result = []
print("공부한 시간 : ",x)
print("실제 성적   : ",y)
print('-'*50)

# 모든 x값을 한 번씩 대입하여 predict_result 리스트완성
for i in range(len(x)):
    predict_result.append(predict(x[i]))    #predict()함수 호출 후, 리턴값을 predict_result 리스
    print("공부시간=%.f, 실제점수=%.f, 예측점수=%.f" % (x[i], y[i], predict_result[i]))

print("예측한 성적 : ", predict_result)
```

```
공부한 시간 :  [2, 4, 6, 8]
실제 성적   :  [81, 93, 91, 97]
--------------------------------------------------
공부시간=2, 실제점수=81, 예측점수=82
공부시간=4, 실제점수=93, 예측점수=88
공부시간=6, 실제점수=91, 예측점수=94
공부시간=8, 실제점수=97, 예측점수=100
예측한 성적 :  [82, 88, 94, 100]
```

● mse() 함수 정의 : 평균 제곱 오차 (MSE ; Mean Squared Error)

In [8]:

```
#y → real y, y_hat → predicted y

def mse(y, y_hat):
    return ((y - y_hat) ** 2).mean()
```

● mse_numpy()함수 정의 : 위에서 정의한 mse()함수 호출
실제 y 와 예측된 y에 해당하는 predict_result를 넘파이(numpy)타입으로 mse()함수

에게 전달

In [9]:

```
#실제 y 와 예측된 값 y에 해당하는 값을 numpy 형식으로 교체해서 mse()에게 전달
#y → real y, predict_result → predicted y

def mse_numpy(y, predict_result):
    #위에서 정의한 평균제곱오차를 구하는 mse()함수 호출 시, numpy형식으로 y, predict_result를 전
    return mse(np.array(y), np.array(predict_result))
```

참고

● list와 numpy 비교

In [10]:

```
#참고 < 리스트와 넘파이 활용 비교 >
print(y)
print(type(y))
print(y[0])

print('-'*50)

print(np.array(y))
print(type(np.array(y)))
print(np.array(y[0]))
```

```
[81, 93, 91, 97]
<class 'list'>
81
--------------------------------------------------
[81 93 91 97]
<class 'numpy.ndarray'>
81
```

In [11]:

```
#참고 < 리스트와 넘파이 활용 비교 >
test = [1,2,3,4]
np_test = np.array(test)

print(type(test), test)
print(type(np_test), np_test)

print(test[0:4])
print(np_test[0:4])
```

```
<class 'list'> [1, 2, 3, 4]
<class 'numpy.ndarray'> [1 2 3 4]
[1, 2, 3, 4]
[1 2 3 4]
```

In [12]:

```
#참고 < 리스트와 넘파이 활용 비교 >
print(test[0], test[1], test[2], test[3])
print(np_test[0], np_test[1], np_test[2], np_test[3])
```

```
1 2 3 4
1 2 3 4
```

In [13]:

```
#참고 < 리스트와 넘파이 활용 비교 >
result_test = []   # 리스트 활용

for i in test :
    k = i+100
    result_test.append(k)
print("리스트 활용해서 연산 후 새로운 리스트에 할당 : ",result_test)
#print("리스트 타입의 데이터에 직접 연산 : ", test+100)  # 리스트 타입의 데이터에 직접 연산하면


result_np_test = [] # numpy 활용 후 리스트에 담기 위해서
result_np_test.append(np_test+100)
print("넘파이 활용해서 연산 후 새로운 리스트에 할당 : ", result_np_test)

print("넘파이 타입의 데이터에 직접 연산 : ", np_test + 500)
```

```
리스트 활용해서 연산 후 새로운 리스트에 할당 :  [101, 102, 103, 104]
넘파이 활용해서 연산 후 새로운 리스트에 할당 :  [array([101, 102, 103, 104])]
넘파이 타입의 데이터에 직접 연산 :  [501 502 503 504]
```

● mse_numpy()함수 호출 : 실제 y 와 예측된 값 y에 해당하는 predict_result를 넘파이(numpy)타입으로 변환하여 다시 mse()함수를 호출

In [14]:

```
print("MSE 최종값: " , mse_numpy(y, predict_result))
```

```
MSE 최종값:  11.0
```

● 평균 제곱근 오차(Root Mean Squared Error, RMSE) 구하기

In [15]:

```
print("RMSE 최종값: " , np.sqrt(mse_numpy(predict_result,y)))  # numpy에서 제공하는 sqrt함수는
```

```
RMSE 최종값:  3.3166247903554
```

이를 통해 우리가 처음 가정한 a = 3, b = 76은 오차가 약 3.3166이라는 것을 알게 됨
이제 남은 것은 이 오차를 줄이면서 새로운 선을 긋는 것
이를 위해서는 a와 b의 값을 적절히 조절하면서 오차의 변화를 살펴보고
그 오차가 최소화되는 a와 b의 값을 구해야 함

In [ ]:

```

```