

AI 프로그래밍

- 4주차



이하 공전 컴퓨터 정보과
민 정혜

- 지난 주 내용 복습
- Numpy 실습
- Matplotlib 실습

선형 회귀 예제 실습 - 당뇨병 예제

인하공전 컴퓨터 정보과

특징(10개)											
데이터 개수 (442)	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	혈당

bmi

Bmi와 혈당간의 관계 예측

```
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
```

```
print('diabetes_X',diabetes_X.shape )
```

하나의 특징(BMI)만 추려내서 2차원 배열로 만든다. BMI 특징의 인덱스가 2이다.

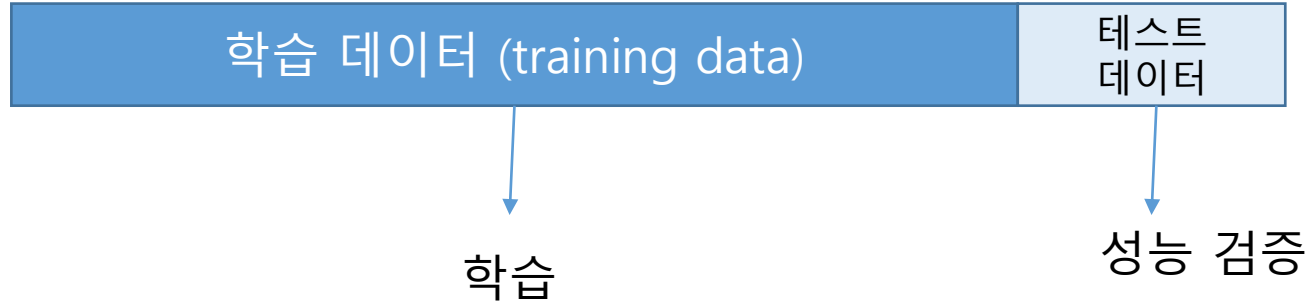
```
diabetes_X_new0 = diabetes_X[:, 2]
```

```
print('diabetes_X_new0',diabetes_X_new0.shape )
```

```
diabetes_X_new = diabetes_X_new0[:, np.newaxis]
```

```
print('diabetes_X_new',diabetes_X_new.shape )
```

```
diabetes_X (442, 10)
diabetes_X_new0 (442,)
diabetes_X_new (442, 1)
```



442개

```
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)
print('X_train',X_train.shape )
print('X_test',X_test.shape )
print('Y_train',y_train.shape )
print('Y_test',y_test.shape )
```

```
X_train (397, 1)
X_test (45, 1)
Y_train (397,)
Y_test (45,)
```

선형 회귀 예제 실습 - 당뇨병 예제

인하공전 컴퓨터 정보과

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

```
# 당뇨병 데이터 세트를 적재한다.
```

```
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
```

```
print('diabetes_X',diabetes_X.shape )
# 하나의 특징(BMI)만 추려내서 2차원 배열로 만든다. BMI 특징의 인덱스가 2이다.
diabetes_X_new0 = diabetes_X[:, 2]
print('diabetes_X_new0',diabetes_X_new0.shape )
diabetes_X_new = diabetes_X[:, np.newaxis, 2]
print('diabetes_X_new',diabetes_X_new.shape )
```

```
# 학습 데이터와 테스트 데이터를 분리한다.
```

```
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)
print('X_train',X_train.shape)
print('X_test',X_test.shape)
print('y_train',y_train.shape)
print('y_test',y_test.shape)
```

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

```
# 테스트 데이터로 예측해보자.
```

```
y_pred = regr.predict(X_test)
print(regr.predict([[0.01]])) # bmi가 0.01일때 혈당 예측값
```

```
# 실제 데이터와 예측 데이터를 비교해보자.
```

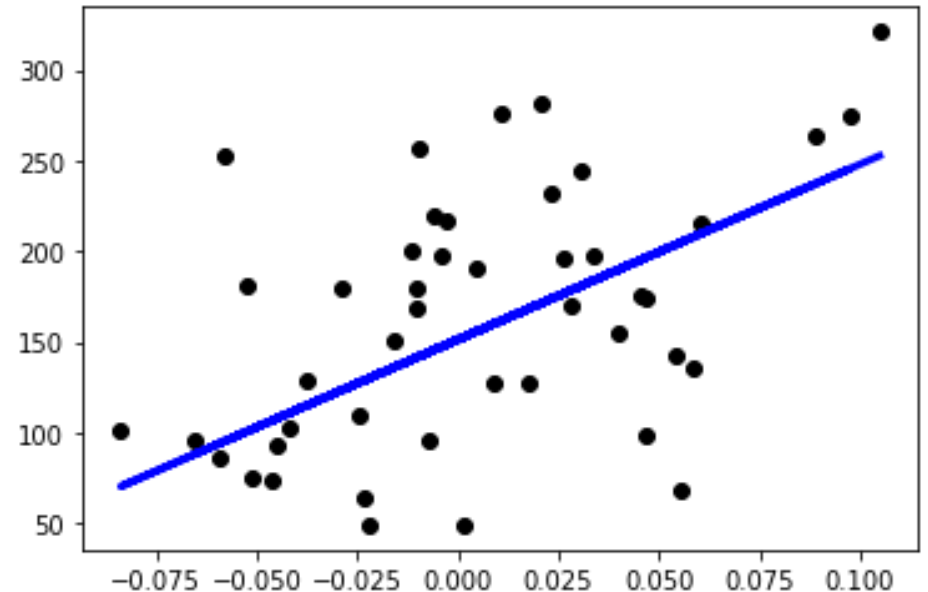
```
# plt.plot(y_test, y_pred, '.')

```

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

자료실의 diabetes_exe.ipynb

Bmi가 0.025일때의 혈당의 예측 값을 채팅으로 보내주세요



오차 계산하기 (2)

- 평균 제곱 오차(Mean Squared Error, MSE) :

오차의 합에 이어 각 x 값의 평균 오차를 이용함

위에서 구한 값을 n으로 나누면 오차 합의 평균을 구할 수 있음

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n} \sum (\hat{y}_i - y_i)^2$$

- 선형 회귀란 :

임의의 직선을 그어 이에 대한 평균 제곱 오차를 구하고, 이 값을 가장 작게 만들어 주는 a와 b 값을 찾아가는 작업임

오차 수정하기 : 경사 하강법

- $Y=ax$
- 오차가 가장 작은 지점은?

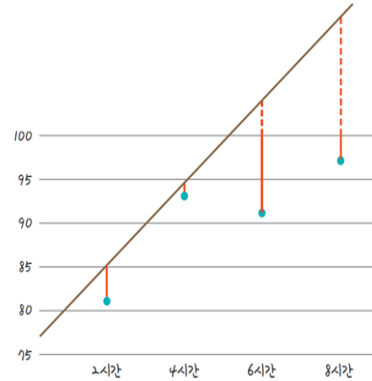


그림 3-7 기울기를 너무 크게 잡았을 때의 오차

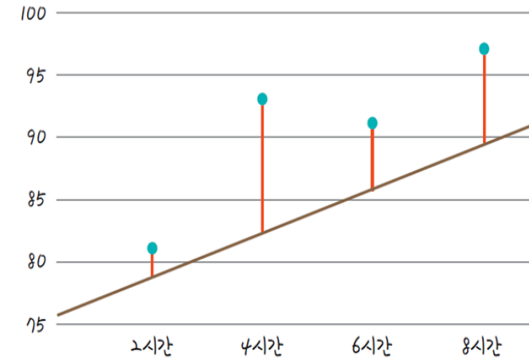
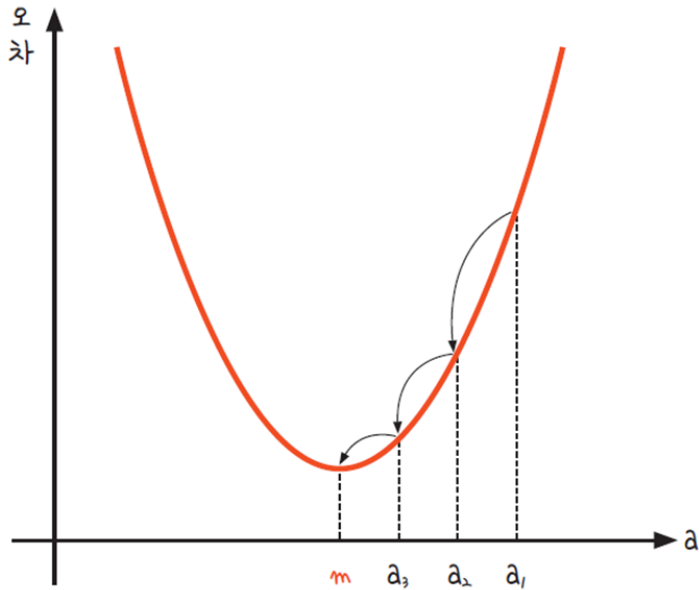


그림 3-8 기울기를 너무 작게 잡았을 때의 오차

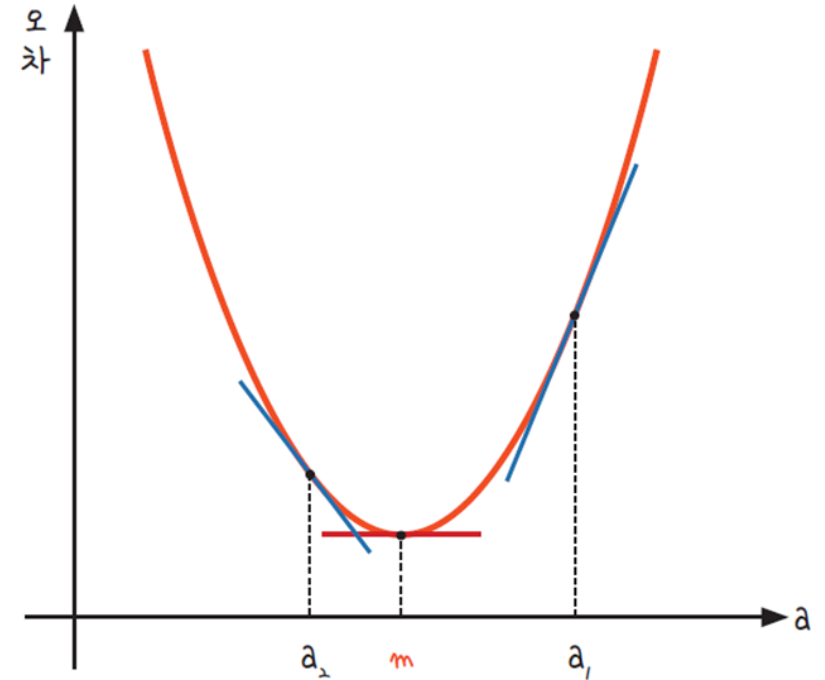


- 컴퓨터를 이용해 m 의 값을 구하려면 임의의 한 점(a_1)을 찍고 이 점을 m 에 가까운 쪽으로 점점 이동($a_1 \rightarrow a_2 \rightarrow a_3$)시키는 과정이 필요함
- 경사 하강법 (gradient descent):
그래프에서 오차를 비교하여 가장 작은 방향으로 이동시키는 방법이 있는데 바로 미분 기울기를 이용



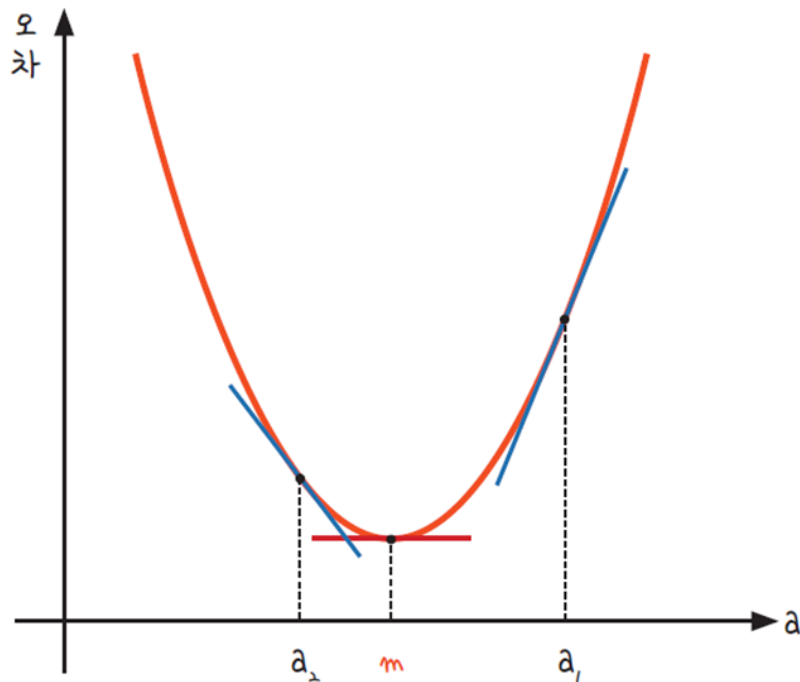
경사 하강법

- $y = x^2$ 그래프에서 x 에 다음과 같이 a_1, a_2 그리고 m 을 대입하여 그 자리에서 미분하면 그림 4-2처럼 각 점에서의 순간 기울기가 그려짐
- 여기서 눈여겨 봐야 할 것은 우리가 찾는 최솟값 m 에서의 순간 기울기임
- 그래프가 이차 함수 포물선이므로 꼭짓점의 기울기는 x 축과 평행한 선이 됨
- 즉, 기울기가 0임
- 우리가 할 일은 '미분 값이 0인 지점'을 찾는 것이 됨



경사 하강법

- 1 | a_1 에서 미분을 구함
- 2 | 구해진 기울기의 반대 방향(기울기가 +면 음의 방향, -면 양의 방향)으로 얼마간 이동시킨 a_2 에서 미분을 구함(그림 4-3 참조).
- 3 | 위에서 구한 미분 값이 0이 아니면 위 과정을 반복함



경사 하강법 실습

- 손실 함수 $y = (x - 3)^2 + 10$

- 그래디언트: $y' = 2x - 6$

- 학습률 : 0.2

- $X=10, y'=14, 0.2*14=2.8,$

Gradient의 반대 방향 => -2.8

$10 - 2.8 = 7.2$

- $X=7.2, Y'=8.4, 0.2*8.4=$

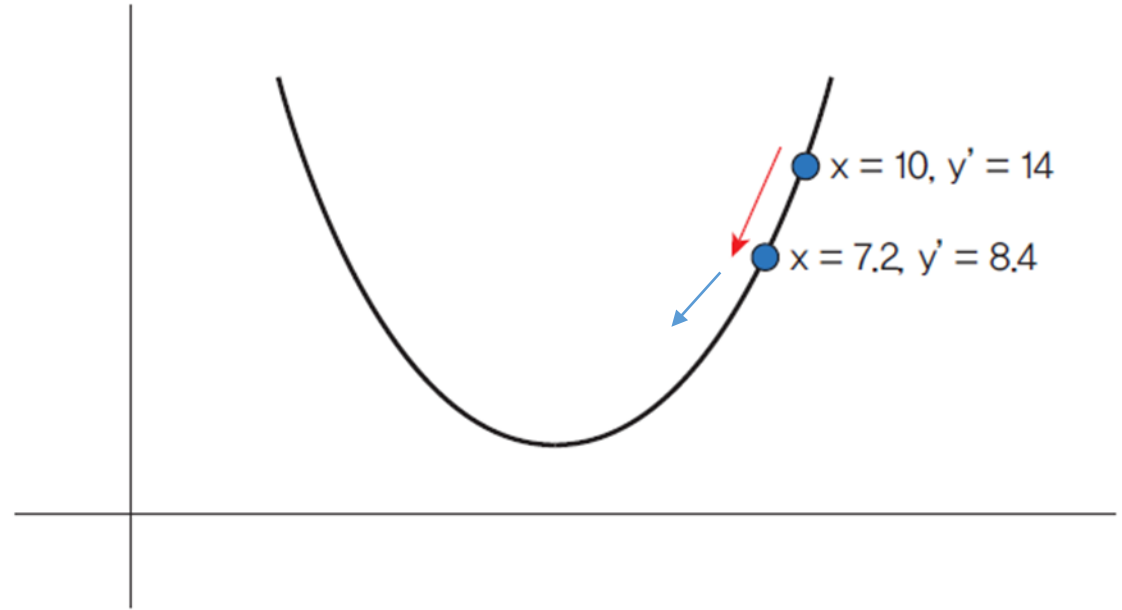


그림 6-12 그래디언트의 계산

경서 하강법 실습

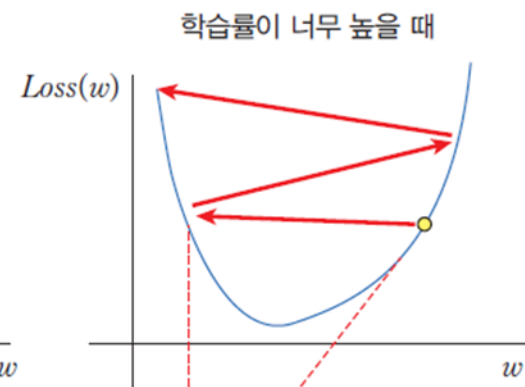
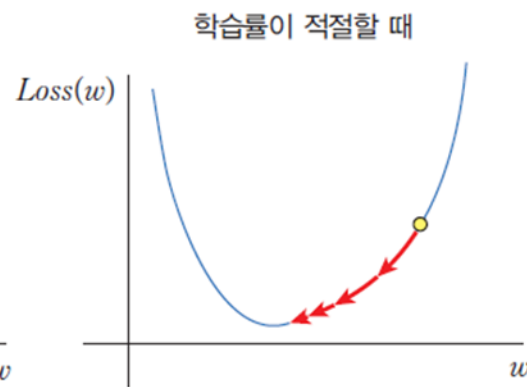
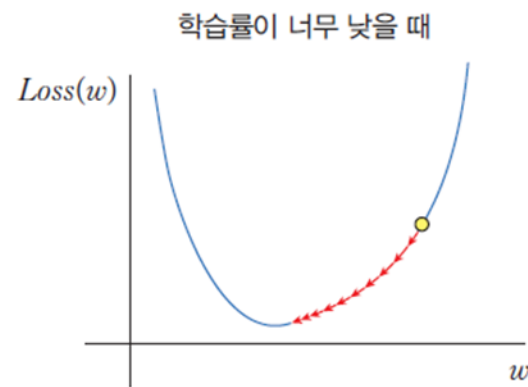
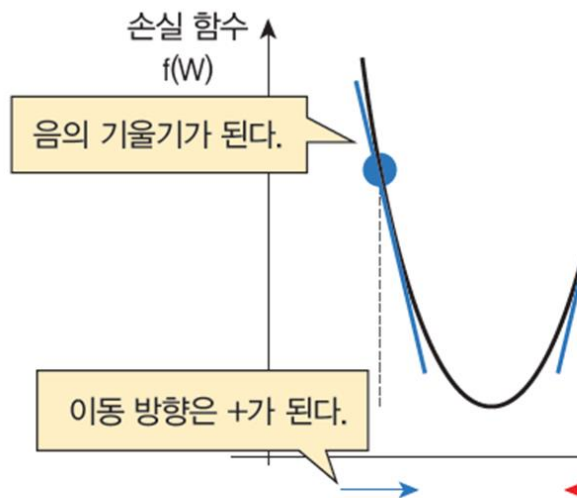


그림 4-9 학습률



이것은 마치 산에서 내려오는 것과 유사합니다. 현재 위치에서 산의 기울기를 계산하여서 기울기의 반대 방향으로 이동하면 산에서 내려오게 됩니다.

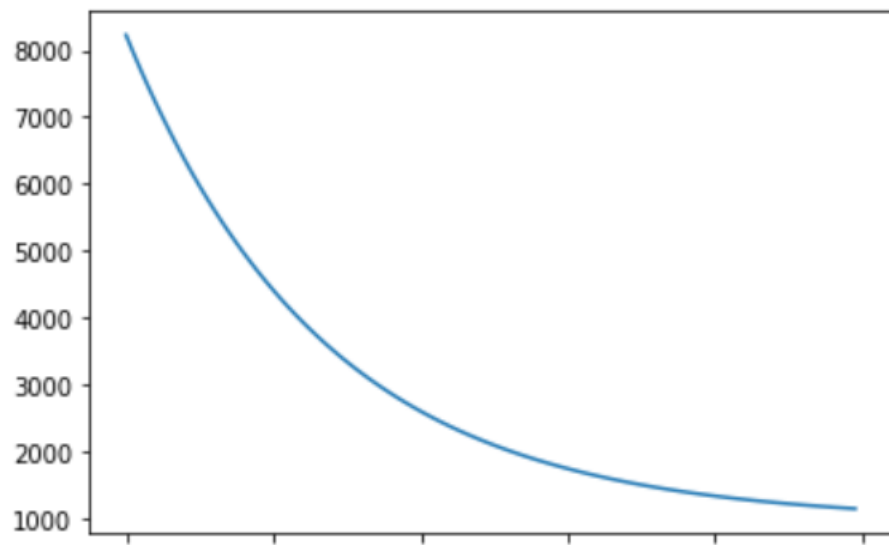
경사 하강법 실습

- `colab_03_LinearRegression.ipynb` 실행
- 학습률 $lr=0.0003$ 일때의 첫번째와 두번째 `loss` => 채팅으로 보내기
- 학습률 $lr=0.0009$ 일때의 첫번째와 두번째 `loss` => 채팅으로 보내기

경사 하강법 실습

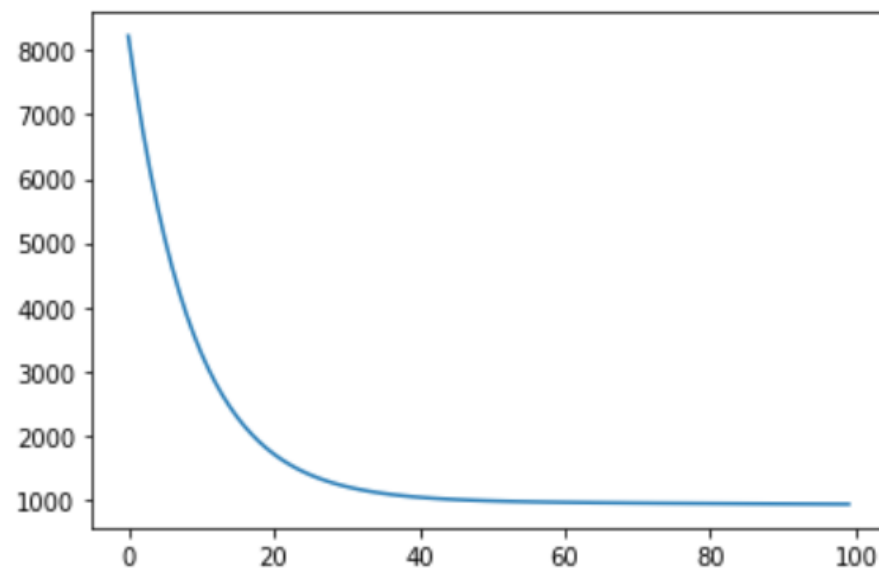
■ 학습률=0.0003

```
epoch=0, 기울기=0.2784, 절편=0.0543 loss =8225.00000  
epoch=10, 기울기=2.7931, 절편=0.5521 loss =5965.81863  
epoch=20, 기울기=4.8772, 절편=0.9777 loss =4410.22195  
epoch=30, 기울기=6.6042, 절편=1.3434 loss =3338.90852  
epoch=40, 기울기=8.0349, 절편=1.6594 loss =2600.93165  
epoch=50, 기울기=9.2197, 절편=1.9341 loss =2092.39373  
epoch=60, 기울기=10.2006, 절편=2.1745 loss =1741.78132  
epoch=70, 기울기=11.0123, 절편=2.3865 loss =1499.87125  
epoch=80, 기울기=11.6836, 절편=2.5749 loss =1332.78285  
epoch=90, 기울기=12.2385, 절편=2.7438 loss =1217.19585  
[<matplotlib.lines.Line2D at 0x7f140432c8d0>]
```



■ 학습률=0.0009

```
epoch=0, 기울기=0.8352, 절편=0.1629 loss =8225.00000  
epoch=10, 기울기=7.0087, 절편=1.4265 loss =3288.43886  
epoch=20, 기울기=10.4800, 절편=2.2366 loss =1709.21635  
epoch=30, 기울기=12.4249, 절편=2.7902 loss =1201.43535  
epoch=40, 기울기=13.5076, 절편=3.1987 loss =1035.60302  
epoch=50, 기울기=14.1032, 절편=3.5249 loss =978.91795  
epoch=60, 기울기=14.4237, 절편=3.8044 loss =957.08669  
epoch=70, 기울기=14.5889, 절편=4.0572 loss =946.40002  
epoch=80, 기울기=14.6663, 절편=4.2946 loss =939.29155  
epoch=90, 기울기=14.6943, 절편=4.5230 loss =933.34650  
[<matplotlib.lines.Line2D at 0x7f1404a82190>]
```



다중 선형 회귀

- 더 정확한 예측을 하려면 추가 정보를 입력해야 하며, 정보를 추가해 새로운 예측값을 구하려면 변수의 개수를 늘려 다중 선형 회귀를 만들어 주어야 함

공부한 시간(x_1)	2	4	6	8
과외 수업 횟수(x_2)	0	4	2	3
성적(y)	81	93	91	97

표 4-1 공부한 시간, 과외 수업 횟수에 따른 성적 데이터

다중 선형 회귀

- 그럼 지금부터 두 개의 독립 변수 x_1 과 x_2 가 생긴 것임
- 이를 사용해 종속 변수 y 를 만들 경우 기울기를 두 개 구해야 하므로 다음과 같은 식이 나옴

$$y = a_1x_1 + a_2x_2 + b$$

- 이번에는 x 의 값이 두 개이므로 다음과 같이 data 리스트를 만들고 x_1 과 x_2 라는 두 개의 독립 변수 리스트를 만들어 줌

```
data = [[2, 0, 81], [4, 4, 93], [6, 2, 91], [8, 3, 97]]  
x1 = [i[0] for i in data]  
x2 = [i[1] for i in data]  
y = [i[2] for i in data]
```

다중 선형 회귀

- data가 그래프로 어떻게 보이는지를 확인해 보자
- 먼저 x, y 두 개의 축이던 이전과 달리 x_1, x_2, y 이렇게 세 개의 축이 필요함
- 3D 그래프를 그려주는 라이브러리를 아래와 같이 불러옴

```
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d # 3D 그래프 그리는 라이브러리 가져오기

ax = plt.axes(projection='3d') # 그래프 유형 정하기
ax.set_xlabel('study_hours')
ax.set_ylabel('private_class')
ax.set_zlabel('Score')
ax.scatter(x1, x2, y)
plt.show()
```


다중 선형 회귀

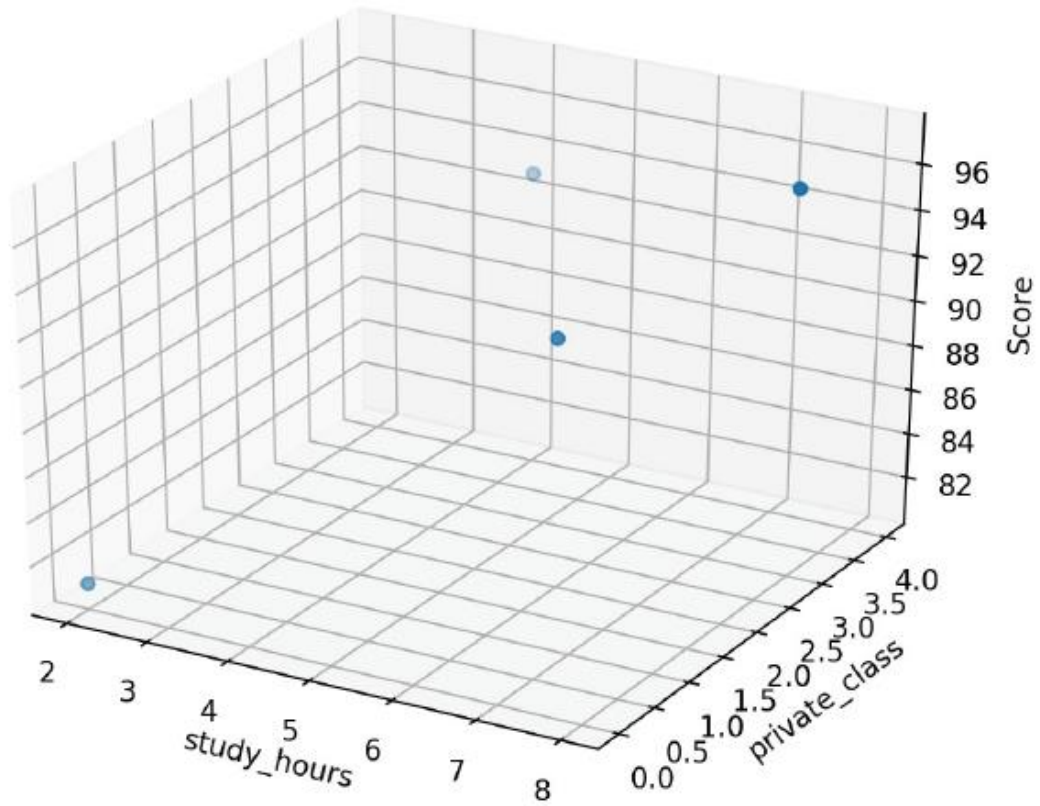


그림 4-6 축이 하나 더 늘어 3D로 배치된 모습

다중 선형 회귀

- 이제 x 가 두 개가 되었으므로 x 과 x_2 두 가지의 변수를 지정함
- 각각의 값에 기울기 a 값이 다르므로 기울기도 a_1 과 a_2 이렇게 두 가지를 만들
- 각각 앞서 했던 방법과 같은 방법으로 경사 하강법을 적용하고 학습률을 곱해 기존의 값을 업데이트 함

```
y_pred = a1 * x1_data + a2 * x2_data + b # y를 구하는 식을 세우기
error = y_data - y_pred # 오차를 구하는 식
a1_diff = -(1/len(x1_data)) * sum(x1_data * (error)) # 오차 함수를 a1로 미분한 값
a2_diff = -(1/len(x2_data)) * sum(x2_data * (error)) # 오차 함수를 a2로 미분한 값

b_new = -(1/len(x1_data)) * sum(y_data - y_pred) # 오차 함수를 b로 미분한 값
a1 = a1 - lr * a1_diff # 학습률을 곱해 기존의 a1 값 업데이트
a2 = a2 - lr * a2_diff # 학습률을 곱해 기존의 a2 값 업데이트
b = b - lr * b_diff # 학습률을 곱해 기존의 b 값 업데이트
```

다중 선형 회귀

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

# 공부 시간 X와 성적 Y의 리스트 만들기
data = [[2, 0, 81], [4, 4, 93], [6, 2, 91], [8, 3, 97]]
x1 = [i[0] for i in data]
x2 = [i[1] for i in data]
y = [i[2] for i in data]
```

다중 선형 회귀

그래프로 확인

```
ax = plt.axes(projection='3d')
ax.set_xlabel('study_hours')
ax.set_ylabel('private_class')
ax.set_zlabel('Score')
ax.dist = 11
ax.scatter(x1, x2, y)
plt.show()
```

리스트로 되어 있는 x와 y 값을 넘파이 배열로 바꾸기(인덱스로 하나씩 불러와 계산할 수 있도록 하기 위함)

```
x1_data = np.array(x1)
x2_data = np.array(x2)
y_data = np.array(y)
```

다중 선형 회귀

기울기 a와 절편 b의 값 초기화

a1 = 0

a2 = 0

b = 0

학습률

lr = 0.05

몇 번 반복할지 설정(0부터 세므로 원하는 반복 횟수에 +1)

epochs = 2001

다중 선형 회귀

경사 하강법 시작

for i in range(epochs): # epoch 수 만큼 반복

y_pred = a1 * x1_data + a2 * x2_data + b # y를 구하는 식 세우기

error = y_data - y_pred # 오차를 구하는 식

오차 함수를 a1로 미분한 값

a1_diff = -(1/len(x1_data)) * sum(x1_data * (error))

오차 함수를 a2로 미분한 값

a2_diff = -(1/len(x2_data)) * sum(x2_data * (error))

오차 함수를 b로 미분한 값

b_new = -(1/len(x1_data)) * sum(y_data - y_pred)

a1 = a1 - lr * a1_diff # 학습률을 곱해 기존의 a1 값 업데이트

a2 = a2 - lr * a2_diff # 학습률을 곱해 기존의 a2 값 업데이트

b = b - lr * b_diff # 학습률을 곱해 기존의 b값 업데이트

if i % 100 == 0: # 100번 반복될 때마다 현재의 a1, a2, b 값 출력

print("epoch=%.f, 기울기1=%.04f, 기울기2=%.04f, 절편=%.04f"

% (i, a1, a2, b))

다중 선형 회귀

실행
결과



epoch=0, 기울기 1=23.2000, 기울기 2=10.5625, 절편=4.5250

epoch=100, 기울기 1=6.4348, 기울기 2=3.9893, 절편=43.9757

epoch=200, 기울기 1=3.7255, 기울기 2=3.0541, 절편=62.5766

epoch=300, 기울기 1=2.5037, 기울기 2=2.6323, 절편=70.9656

epoch=400, 기울기 1=1.9527, 기울기 2=2.4420, 절편=74.7491

epoch=500, 기울기 1=1.7042, 기울기 2=2.3562, 절편=76.4554

(중략)

epoch=1500, 기울기 1=1.5001, 기울기 2=2.2857, 절편=77.8567

epoch=1600, 기울기 1=1.5000, 기울기 2=2.2857, 절편=77.8569

epoch=1700, 기울기 1=1.5000, 기울기 2=2.2857, 절편=77.8570

epoch=1800, 기울기 1=1.5000, 기울기 2=2.2857, 절편=77.8571

epoch=1900, 기울기 1=1.5000, 기울기 2=2.2857, 절편=77.8571

epoch=2000, 기울기 1=1.5000, 기울기 2=2.2857, 절편=77.8571

다중 선형 회귀

- 다중 선형 회귀 문제에서의 기울기 a_1 , a_2 와 절편 b 의 값을 찾아 확인할 수 있음

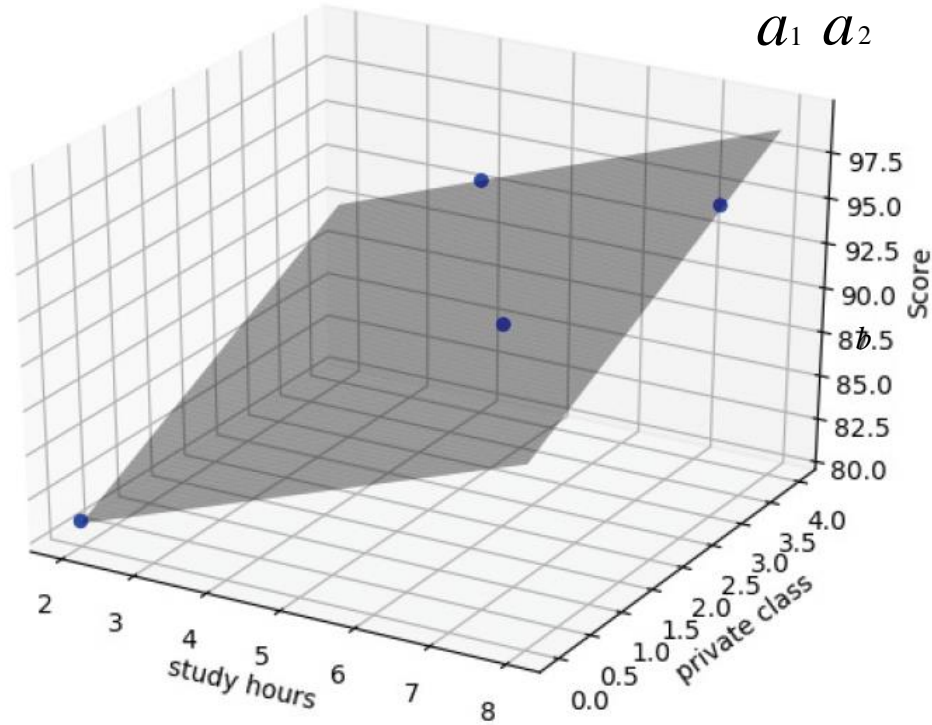


그림 4-7 다중 선형 회귀의 그래프: 차원이 하나 더 늘어난 모습

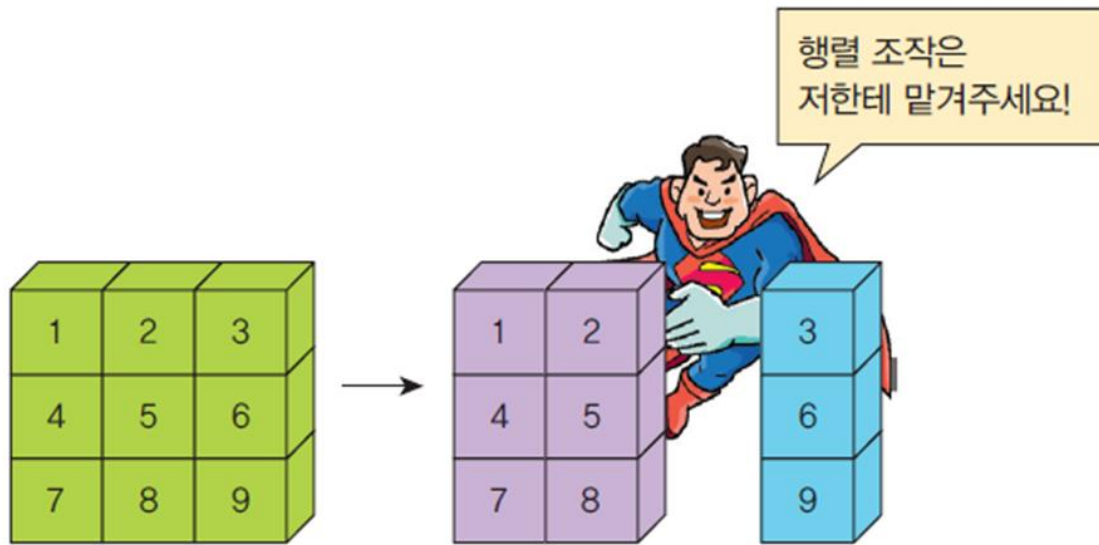
자료실의 `colab_04_Multi_Linear_Regression_loss.ipynb`을 실행 한 후, 첫번째와 두번째 loss 확인.

다중 선형 회귀

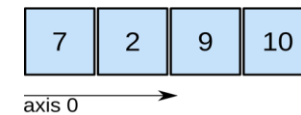
- 1차원 예측 직선이 3차원 '예측 평면'으로 바뀜
- 과외 수업 횟수(privateclass)라는 새로운 변수가 추가됨
- 1차원 직선에서만 움직이던 예측 결과가 더 넓은 평면 범위 안에서 움직이게 됨
- 이로 인해 좀 더 정밀한 예측을 할 수 있게 된 것임

Numpy

- 고성능의 수치 계산을 위한 라이브러리
- 넘파이 라이브러리에는 다차원 배열 데이터 구조가 포함되어 있다.
- 넘파이는 데이터 전처리 및 다양한 수학적 행렬 연산을 수행하는 데 사용할 수 있다.
- `Import numpy as np`

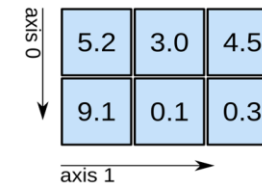


1D array



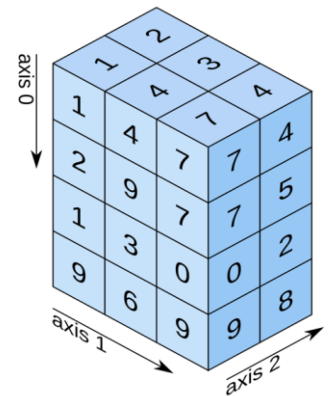
shape: (4,)

2D array



shape: (2, 3)

3D array



shape: (4, 3, 2)

Numpy

딥러닝에서 넘파이가 중요한 이유

- 학습 데이터는 2차원 행렬이나 3차원 행렬에 저장된다.

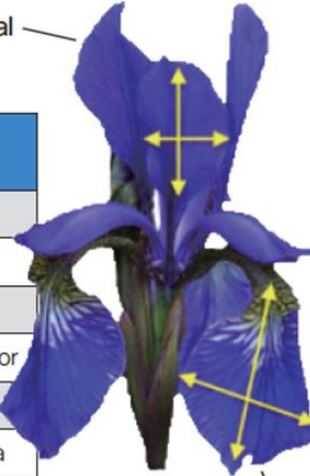
	특징 #1	특징 #2	특징 #3	특징 #4	...		
샘플 #1							
샘플 #2							
샘플 #3							
...							

샘플

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
50	6.4	3.5	4.5	1.2	Versicolor
150	5.9	3.0	5.0	1.8	Virginica

특징

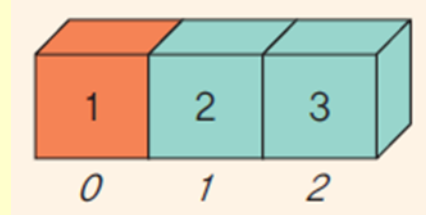
레이블



The diagram shows a blue Iris flower. Yellow arrows indicate measurements: a vertical arrow for 'Petal' length and a horizontal arrow for 'Sepal' width. The labels 'Petal' and 'Sepal' are placed next to their respective arrows.

Numpy

```
>>> import numpy as np  
  
>>> a = np.array([1, 2, 3])  
>>> a  
array([1, 2, 3])  
  
>>> a[0]
```



1

`a = np.array([1, 2, 3])`

배열
객체

생성자 함수

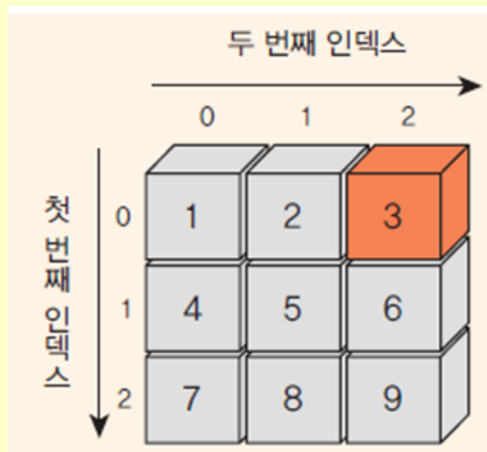
파이썬 리스트

Numpy

```
>>> b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
>>> b  
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

```
>>> b[0][2]  
3
```



Numpy

- 배열의 차원 및 항목 수는 형상(shape)에 의해 정의된다. 배열의 형상은 각 차원의 크기를 지정하는 정수의 튜플이다.
- 넘파이에서는 차원을 축(axis)이라고 한다

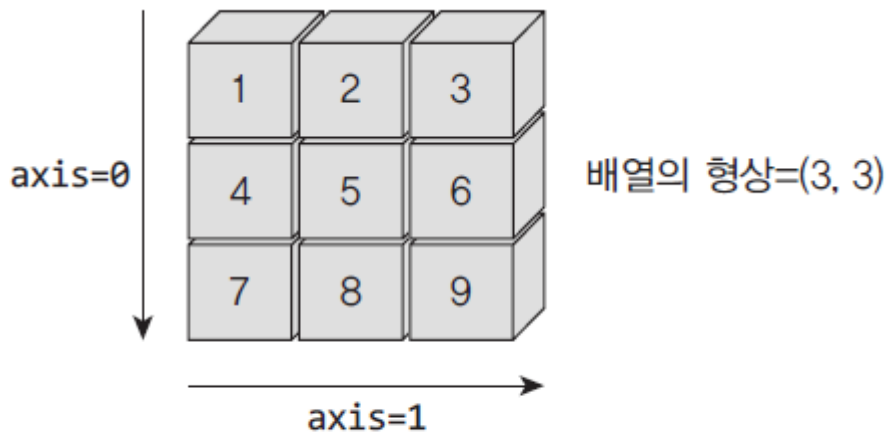
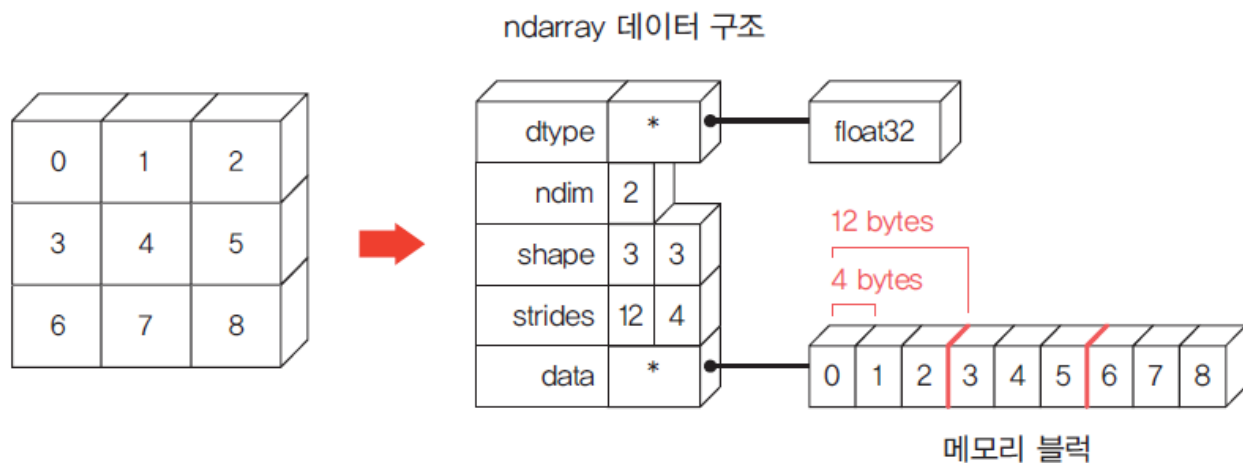


그림 2-4 넘파이 배열의 형상

Numpy- 배열의 속성

속성	설명
ndim	축의 개수. 2차원 배열이면 ndim은 2이다.
shape	배열의 형상. 형상은 정수 튜플로 나타낸다. 예를 들어서 n개의 행과 m개의 열이 있는 행렬의 경우, shape는 (n, m)이다.
size	배열 안에 있는 요소들의 총 개수
dtype	배열 요소의 자료형, numpy.int32, numpy.int16, numpy.float64가 그 예이다.
itemsize	배열을 이루는 요소의 크기로서 단위는 바이트이다. 예를 들어, float64은 itemsize가 8이다.
data	실제 데이터가 저장되는 메모리 블록의 주소



Numpy

```
>>> import numpy as np
>>> a = np.array([[ 0, 1, 2],
                  [ 3, 4, 5],
                  [ 6, 7, 8]])
>>> a.shape           # 배열의 형상
(3, 3)
>>> a.ndim            # 배열의 차원 개수
2
>>> a.dtype           # 요소의 자료형
dtype('int32')
>>> a.itemsize        # 요소 한개의 크기
4                      # 오타
>>> a.size            # 전체 요소의 개수
9
```


Numpy

```
>>> np.zeros( (3, 4) )  
array([[ 0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.]])
```

(3, 4)는 배열의 형상(행의 개수, 열의 개수)

```
>>> np.ones((3, 4))  
array([[1, 1, 1, 1],  
       [1, 1, 1, 1],  
       [1, 1, 1, 1]])
```

```
>>> np.eye(3)  
array([[1., 0., 0.],  
       [0., 1., 0.],  
       [0., 0., 1.]])
```

Numpy-arange

```
>>> np.arange(5)  
array([0, 1, 2, 3, 4])
```

시작값을 지정하려면 다음과 같이 한다.

```
>>> np.arange(1, 6)  
array([1, 2, 3, 4, 5])
```

증가되는 간격을 지정하려면 다음과 같이 한다.

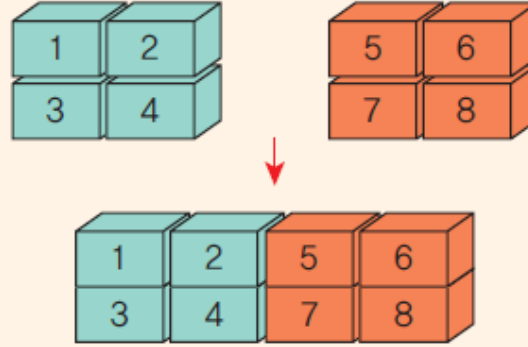
```
>>> np.arange(1, 10, 2)  
array([1, 3, 5, 7, 9])
```

np.arange(start, stop, step)

시작값 종료값 간격

Numpy-배열 합치기

```
>>> x = np.array([[1, 2], [3, 4]])  
>>> y = np.array([[5, 6], [7, 8]])  
  
>>> np.concatenate((x, y), axis=1)  
array([[1, 2, 5, 6],  
       [3, 4, 7, 8]])
```



```
>>> np.vstack((x, y))  
array([[1, 2],  
       [3, 4],  
       [5, 6],  
       [7, 8]])
```



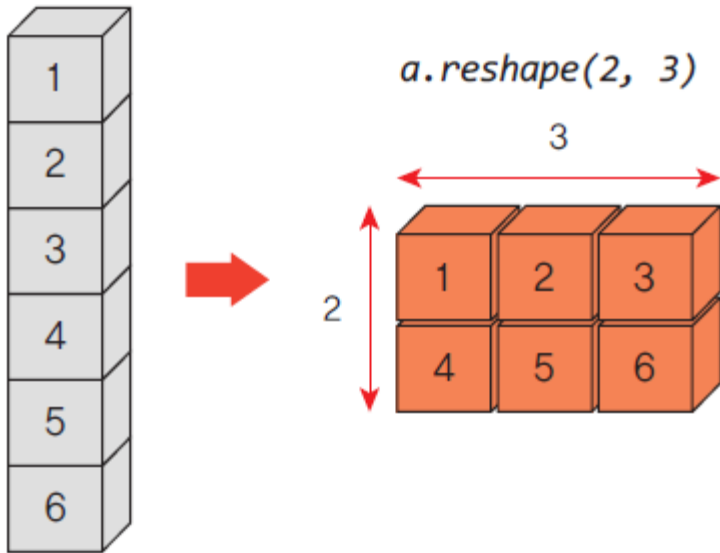
Numpy-reshape

```
new_array = old_array.reshape((2, 3))
```

새로운 배열

원래의 배열

새로운 배열의 형상



Numpy

```
>>> a = np.arange(12)
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

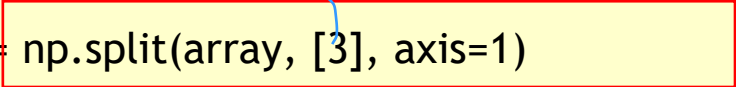
# a에 대하여 reshape(3, 4)를 호출하면 1차원 배열이 2차원 배열로 바뀌게 된다.
>>> a.reshape(3, 4)
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>> a.reshape(6, -1)
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
```

Numpy

```
>>> array = np.arange(30).reshape(-1, 10)
>>> array
Array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]])

>>> arr1, arr2 = np.split(array, [3], axis=1)
>>> arr1
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22]])
>>> arr2
array([[ 3,  4,  5,  6,  7,  8,  9],
       [13, 14, 15, 16, 17, 18, 19],
       [23, 24, 25, 26, 27, 28, 29]])
```



Numpy

```
>>> a = np.array([1, 2, 3, 4, 5, 6])
```

```
>>> a.shape
```

```
(6,)
```

```
>>> a1 = a[np.newaxis, :]
```

```
>>> a1
```

```
array([[1, 2, 3, 4, 5, 6]])
```

```
>>> a1.shape
```

```
(1, 6)
```

```
>>> a2 = a[:, np.newaxis]
```

```
array([[1],
```

```
       [2],
```

```
       [3],
```

```
       [4],
```

```
       [5],
```

```
       [6]])
```

```
>>> a2.shape
```

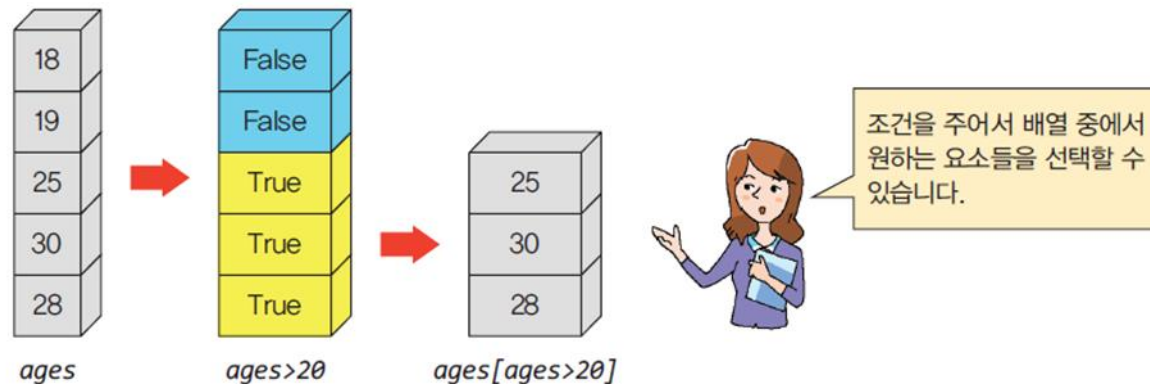
```
(6, 1)
```

Numpy-인덱싱과 슬라이싱

```
>>> ages = np.array([18, 19, 25, 30, 28])
>>> ages[1:3] # 인덱스 1에서 인덱스 2까지
array([19, 25])
>>> ages[:2] # 인덱스 0에서 인덱스 1까지
array([18, 19])
```

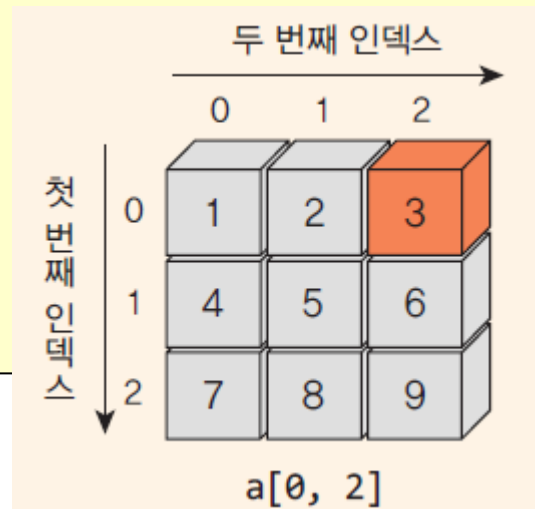
논리적인 인덱싱(logical indexing)

```
>>> y = ages > 20
>>> y
array([False, False,  True,  True,  True])
>>> ages[ ages > 20 ]
array([25, 30, 28])
```



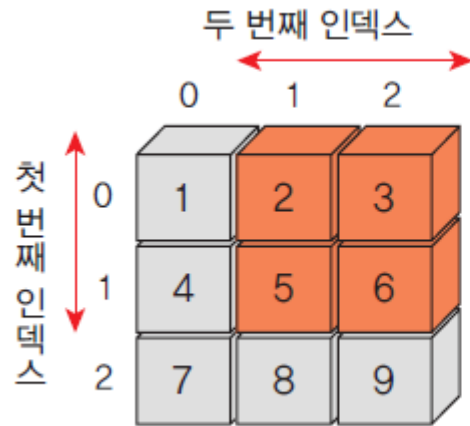
Numpy-2차원 배열의 인덱싱

```
>>> a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> a[0, 2]
3
>>> a[0, 0] = 12
>>> a
array([[12, 2, 3],
       [ 4, 5, 6],
       [ 7, 8, 9]])
```



Numpy-2차원 배열의 슬라이싱

```
>>> a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
>>> a[0:2, 1:3]  
array([[2, 3],  
       [5, 6]])
```



`a[0:2, 1:3]`

Numpy-2차원 배열의 슬라이싱

data

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[0]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[1,:]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[:,2]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[0:2,0:2]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[0:2,2:4]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[:,2,::2]

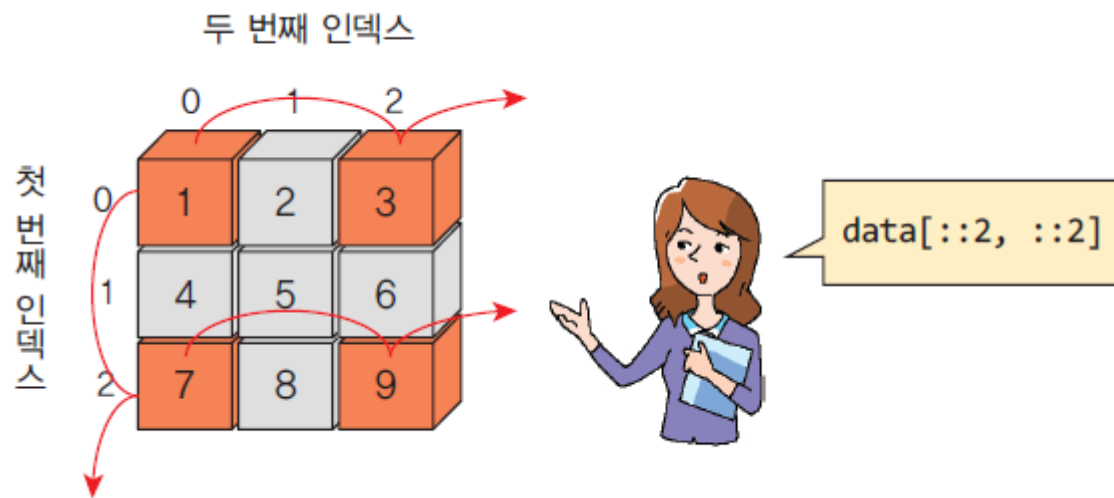
(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[1::2,1::2]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

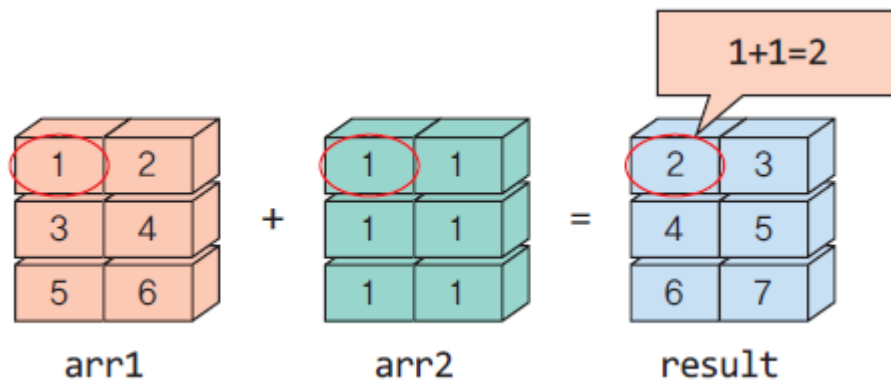
Numpy

```
a[::2, ::2]
```



Numpy-배열의 연산

```
>>> arr1 = np.array([[1, 2], [3, 4], [5, 6]])  
>>> arr2 = np.array([[1, 1], [1, 1], [1, 1]])  
>>> result = arr1 + arr2 # 넘파이 배열에 + 연산이 적용된다.  
>>> result  
array([[2, 3],  
       [4, 5],  
       [6, 7]])
```



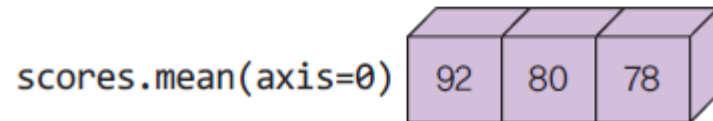
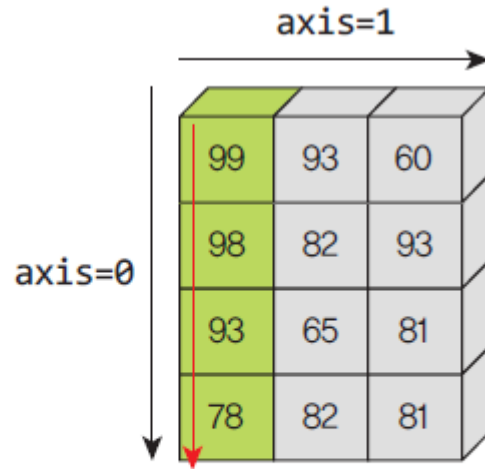
Numpy

```
>>> A = np.array([0, 1, 2, 3])  
>>> 10 * np.sin(A)  
array([0.          , 8.41470985, 9.09297427, 1.41120008])
```

넘파이의 `sin()` 함수를 적용하면 배열의 요소에 모두 `sin()` 함수가 적용된다.

Numpy-특정한 행과 열을 이용한 연산

```
>>> scores = np.array([[99, 93, 60], [98, 82, 93],  
...:                   [93, 65, 81], [78, 82, 81]])  
>>> scores.mean(axis=0)  
array([92. , 80.5 , 78.75])
```



Numpy-난수 생성

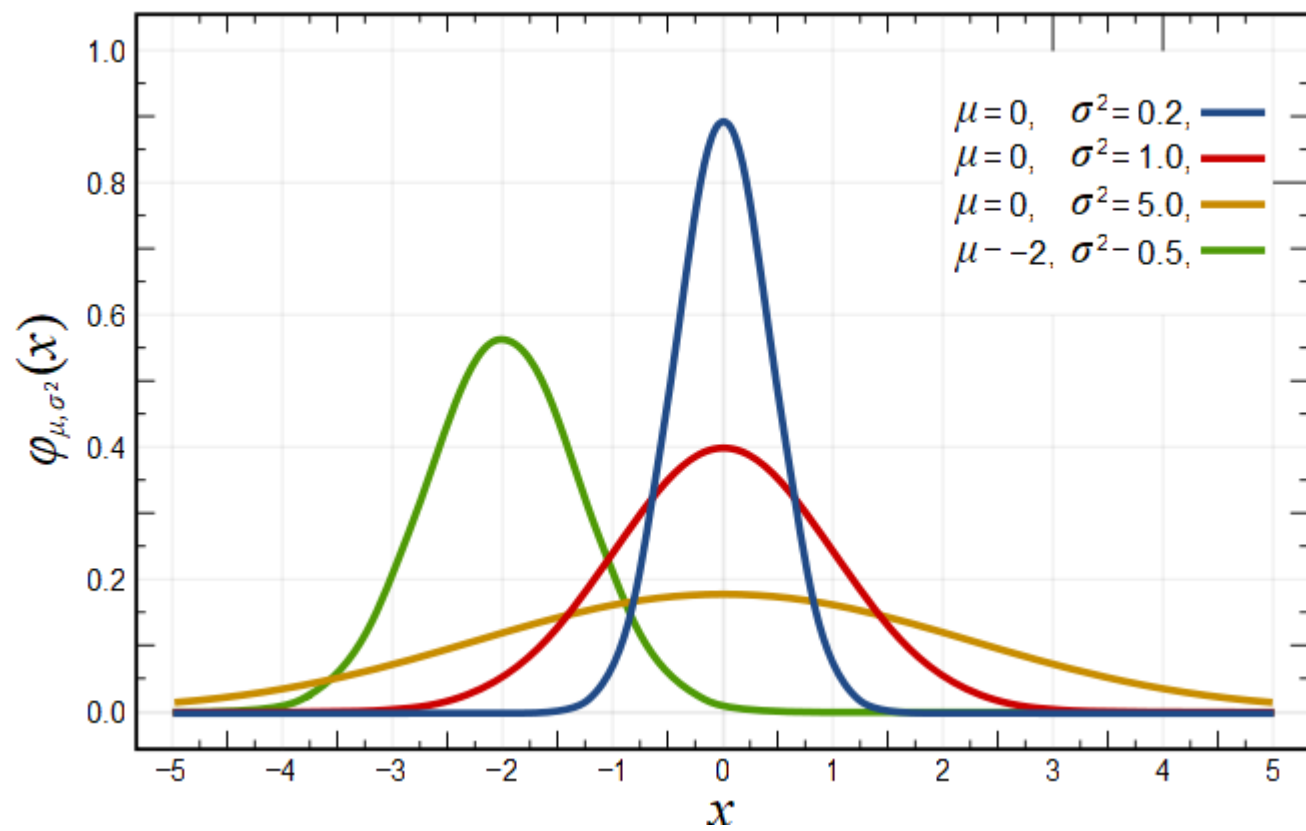
`a = np.random.rand(5, 3)`

새로운 배열 행의 개수 열의 개수

```
>>> np.random.seed(100)
>>> np.random.rand(5)
array([0.54340494, 0.27836939, 0.42451759, 0.84477613, 0.00471886])

>>> np.random.rand(5, 3)
array([[0.12156912, 0.67074908, 0.82585276],
       [0.13670659, 0.57509333, 0.89132195],
       [0.20920212, 0.18532822, 0.10837689],
       [0.21969749, 0.97862378, 0.81168315],
       [0.17194101, 0.81622475, 0.27407375]])
```


Numpy



Numpy

```
>>> np.random.randn(5)
array([ 0.78148842, -0.65438103,  0.04117247, -0.20191691, -0.87081315])

>>> np.random.randn(5, 4)
array([[ 0.22893207, -0.40803994, -0.10392514,  1.56717879],
       [ 0.49702472,  1.15587233,  1.83861168,  1.53572662],
       [ 0.25499773, -0.84415725, -0.98294346, -0.30609783],
       [ 0.83850061, -1.69084816,  1.15117366, -1.02933685],
       [-0.51099219, -2.36027053,  0.10359513,  1.73881773]])

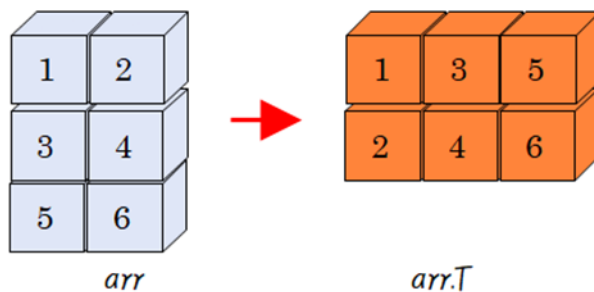
>>> m, sigma = 10, 2
>>> m + sigma*np.random.randn(5)
array([ 8.56778091, 10.84543531,  9.77559704,  9.09052469,  9.48651379])
```

Numpy – 전치 행렬

```
import numpy as np
```

```
arr = np.array([[1, 2], [3, 4], [5, 6]])  
print(arr.T)
```

```
[[1 3 5]  
 [2 4 6]]
```



Numpy= 평탄화

```
>>> x = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])  
>>> x.flatten()  
array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
```

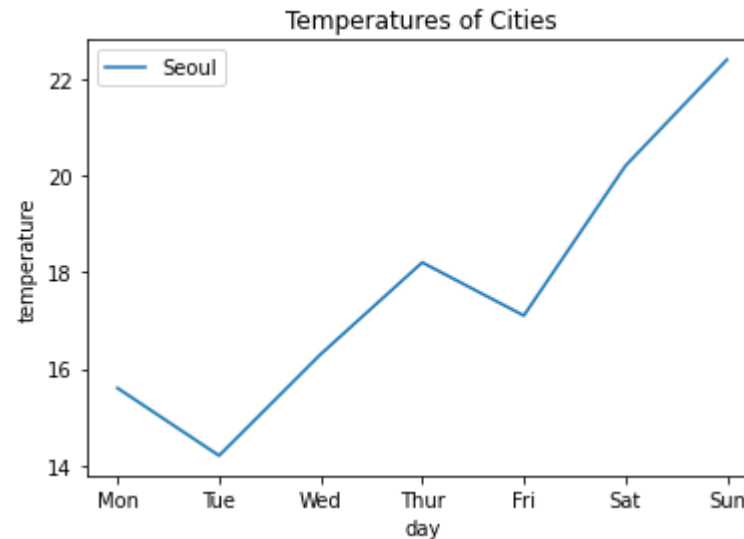
매트플롯

- 그래프를 그리는 라이브러리이다.

```
import matplotlib.pyplot as plt
```

```
X = [ "Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun" ]  
Y1 = [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4]
```

```
plt.plot(X, Y1, label="Seoul") # 분리시켜서 그려도 됨  
plt.xlabel("day")  
plt.ylabel("temperature")  
plt.legend(loc="upper left")  
plt.title("Temperatures of Cities")  
plt.show()
```



매트플롯

- 그래프를 그리는 라이브러리이다.

```
import matplotlib.pyplot as plt
```

```
X = [ "Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun" ]
```

```
Y1 = [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4]
```

```
Y2 = [20.1, 23.1, 23.8, 25.9, 23.4, 25.1, 26.3]
```

```
plt.plot(X, Y1, label="Seoul")
```

```
plt.plot(X, Y2, label="Busan")
```

```
plt.xlabel("day")
```

```
plt.ylabel("temperature")
```

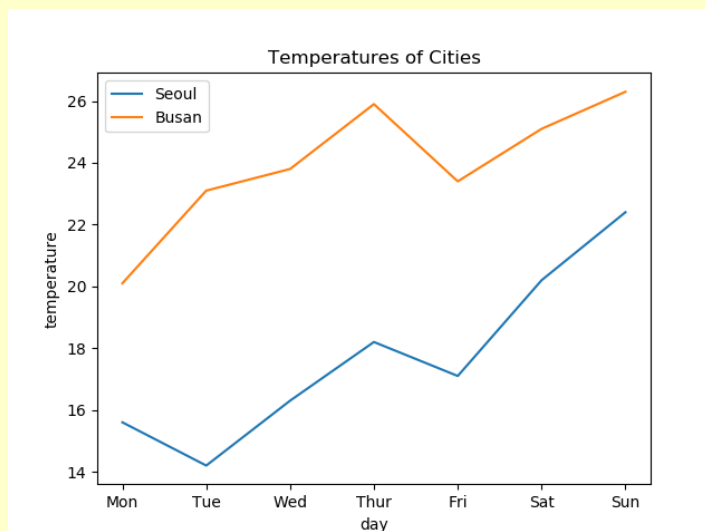
```
plt.legend(loc="upper left")
```

```
plt.title("Temperatures of Cities")
```

```
plt.show()
```

분리시켜서 그려도 됨

분리시켜서 그려도 됨



매트플롯

마커속성

character	description
'_'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond

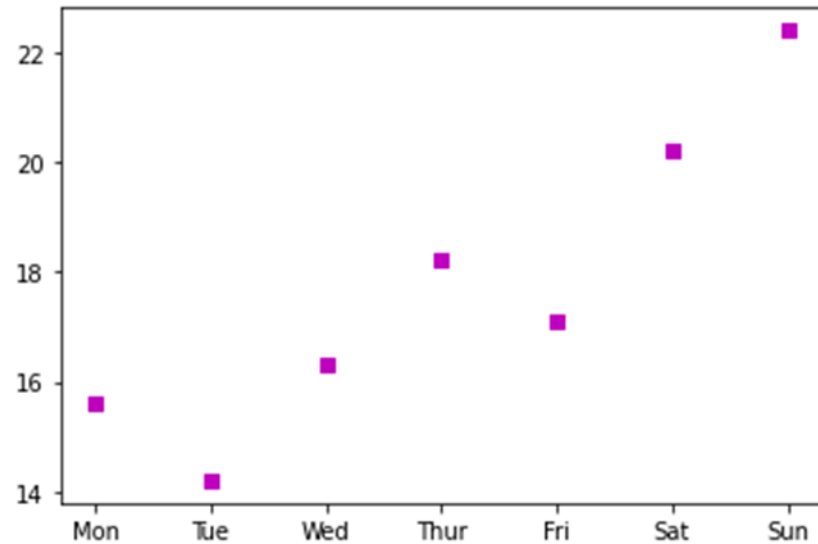
character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

"_4c"

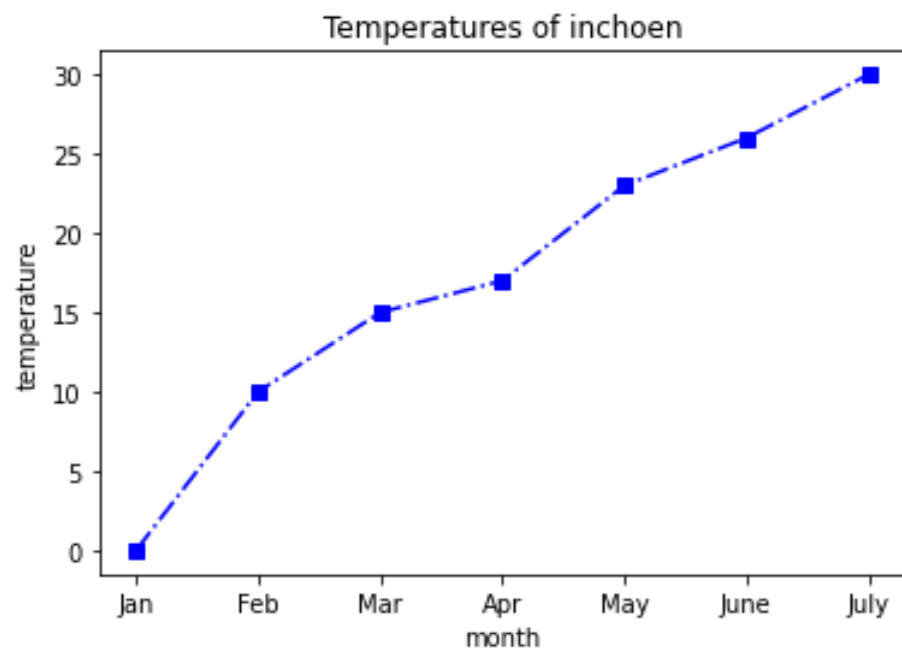
매트플롯

```
import matplotlib.pyplot as plt
%matplotlib inline

X = [ "Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun" ]
plt.plot(X, [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4], "sm")
plt.show()
```



매트플롯



매트플롯

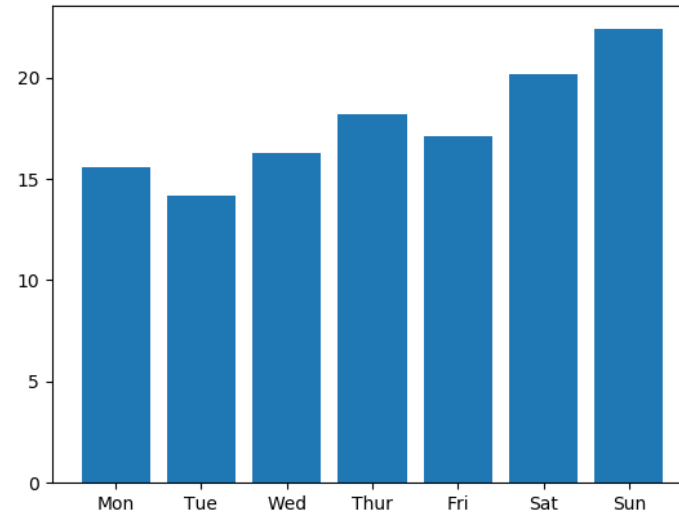
```
import matplotlib.pyplot as plt
```

```
X = [ "Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun" ]
```

```
Y = [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4]
```

```
plt.bar(X, Y)
```

```
plt.show()
```

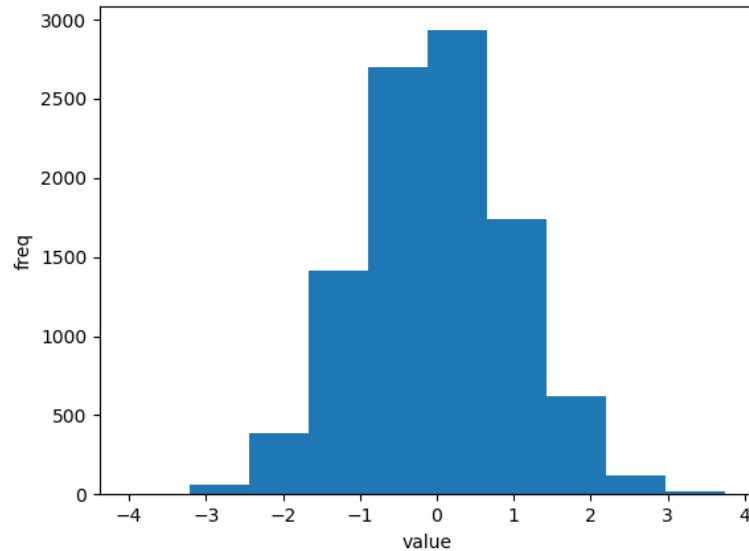


매트플롯

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

numbers = np.random.normal(size=10000)

plt.hist(numbers)
plt.xlabel("value")
plt.ylabel("freq")
plt.show()
```



Numpy+matplotlib

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

```
def sigmoid(x):
    s=1/(1+np.exp(-x))
    ds=s*(1-s)
    return s,ds
```

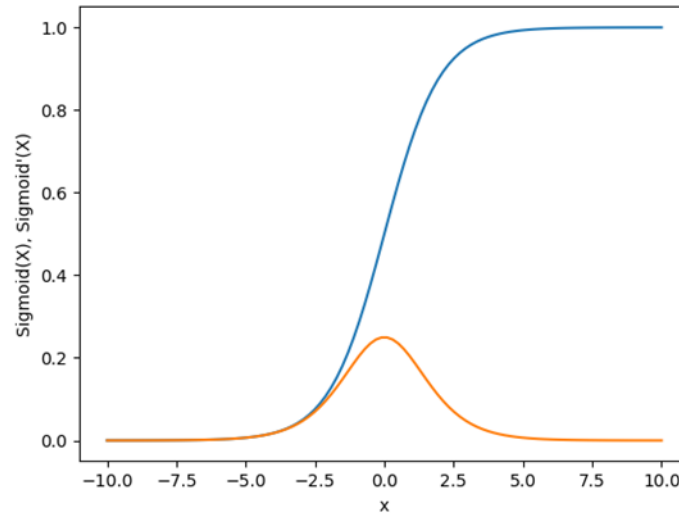
시그모이드 함수 1차 미분 함수

```
X = np.linspace(-10, 10, 100)
Y1, Y2 = sigmoid(X)
```

```
plt.plot(X, Y1, X, Y2)
plt.xlabel("x")
plt.ylabel("Sigmoid(X), Sigmoid'(X)")
plt.show()
```

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

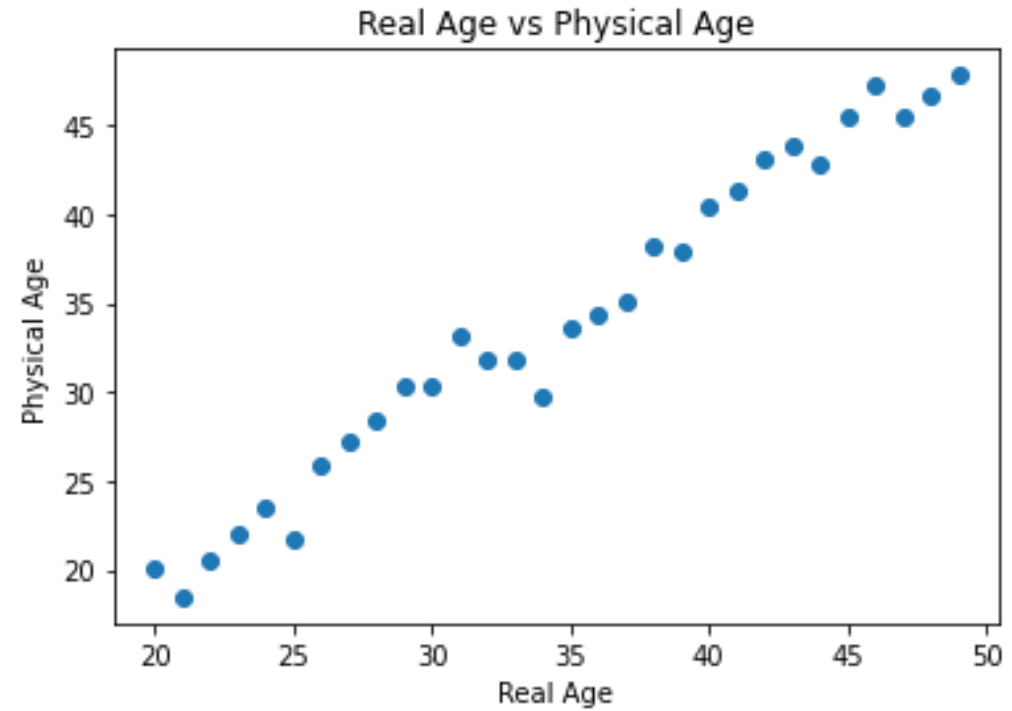


맷플롯 - 산포도

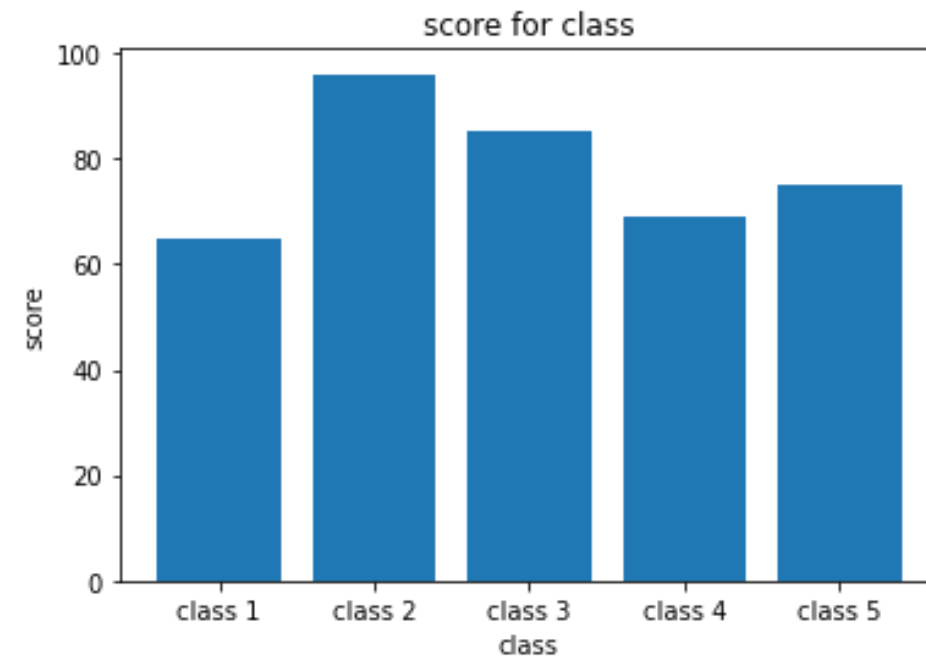
```
import matplotlib.pyplot as plt
import numpy as np

xData=np.arange(20,50)
yData=xData+2*np.random.randn(30)

plt.scatter(xData,yData)
plt.title('Real Age vs Physical Age')
plt.xlabel('Real Age')
plt.ylabel('Physical Age')
plt.show()
```



Numpy



수고 하셨습니다



jhmin@inhatec.ac.kr