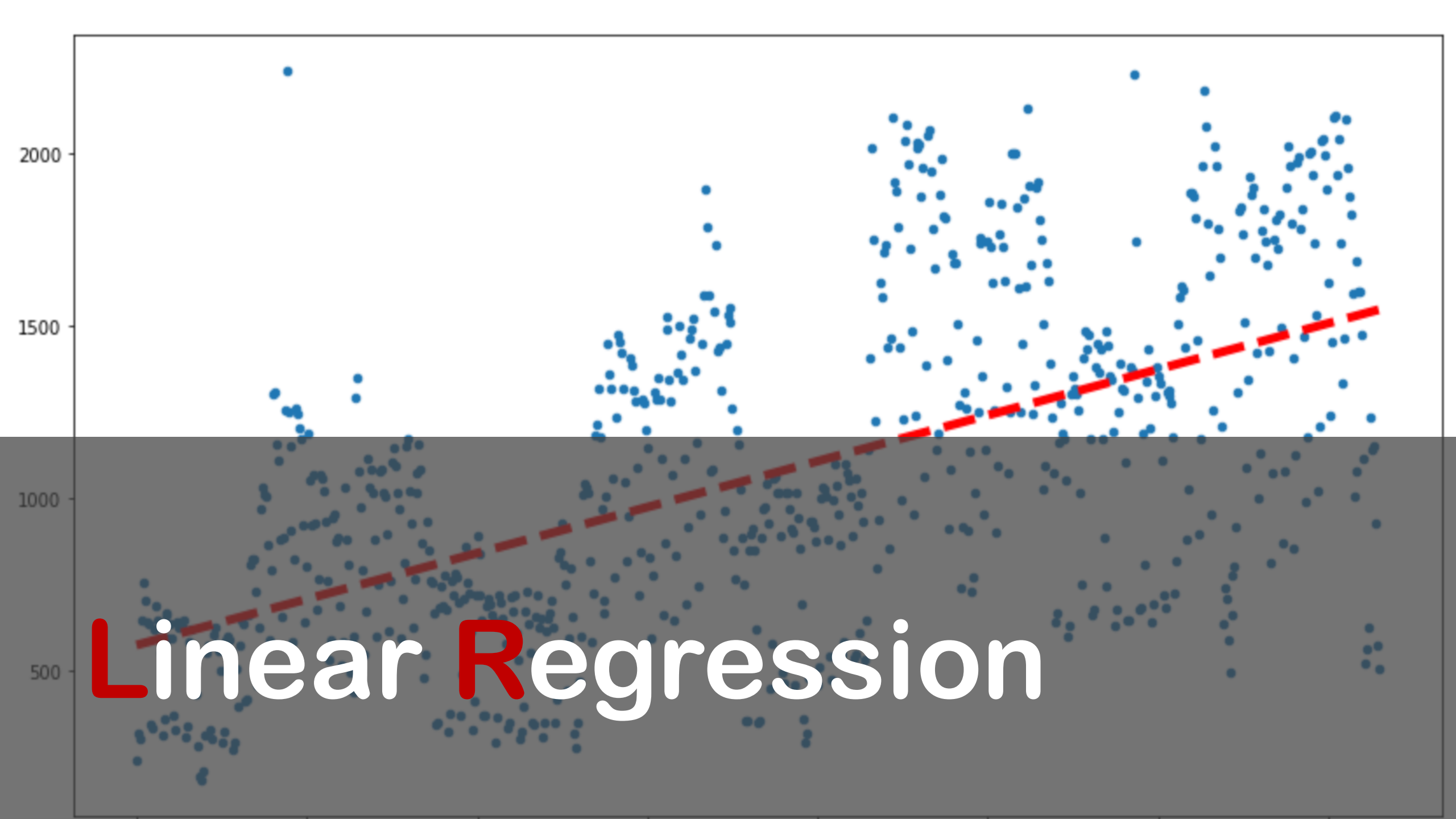
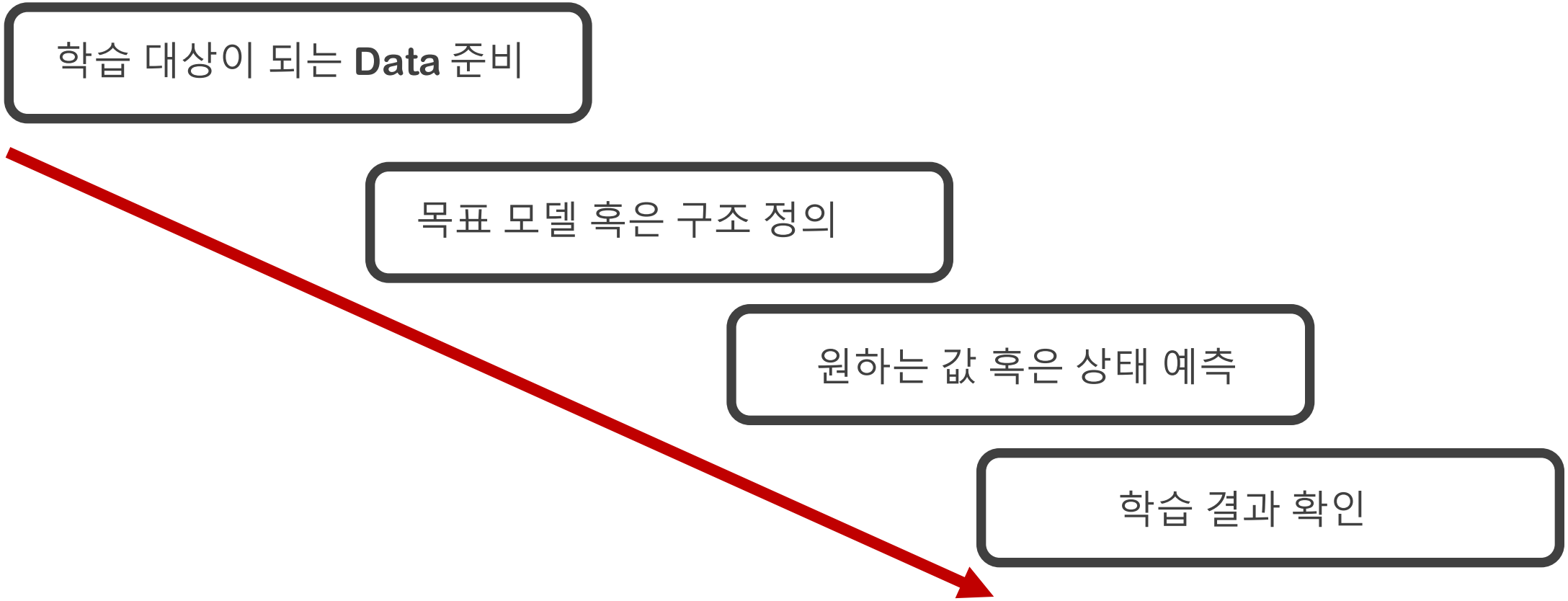
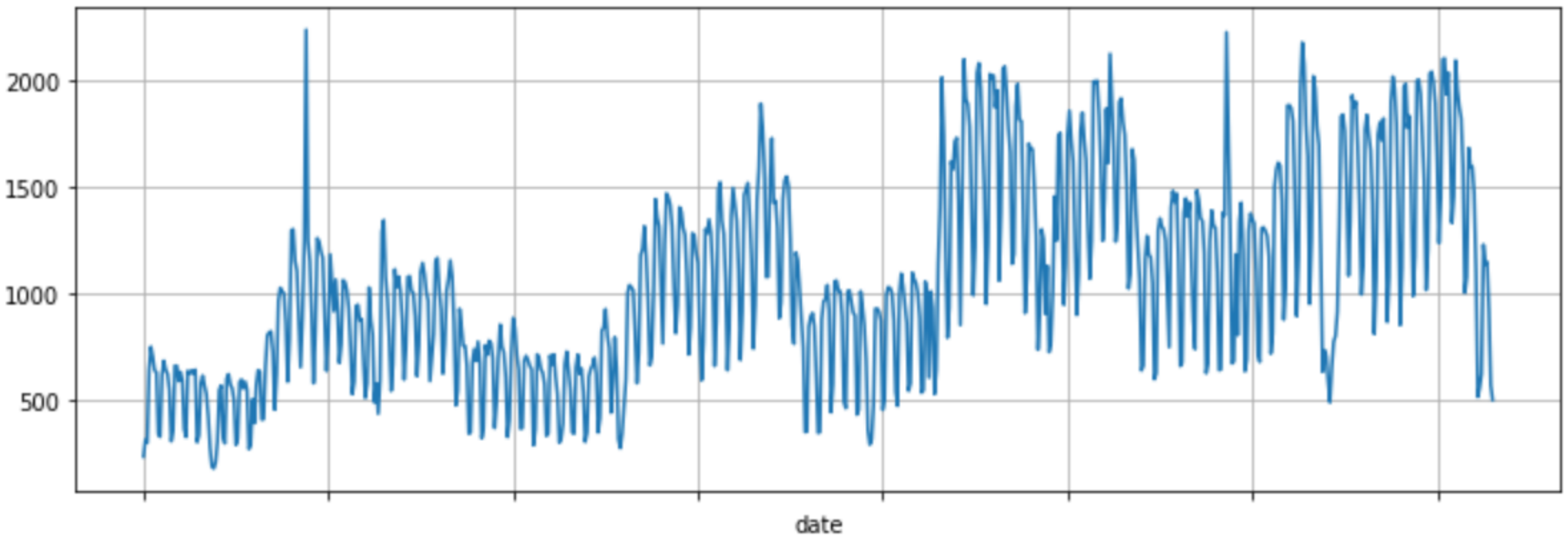

2000
1500
1000
500
Linear Regression

Machine Learning ???

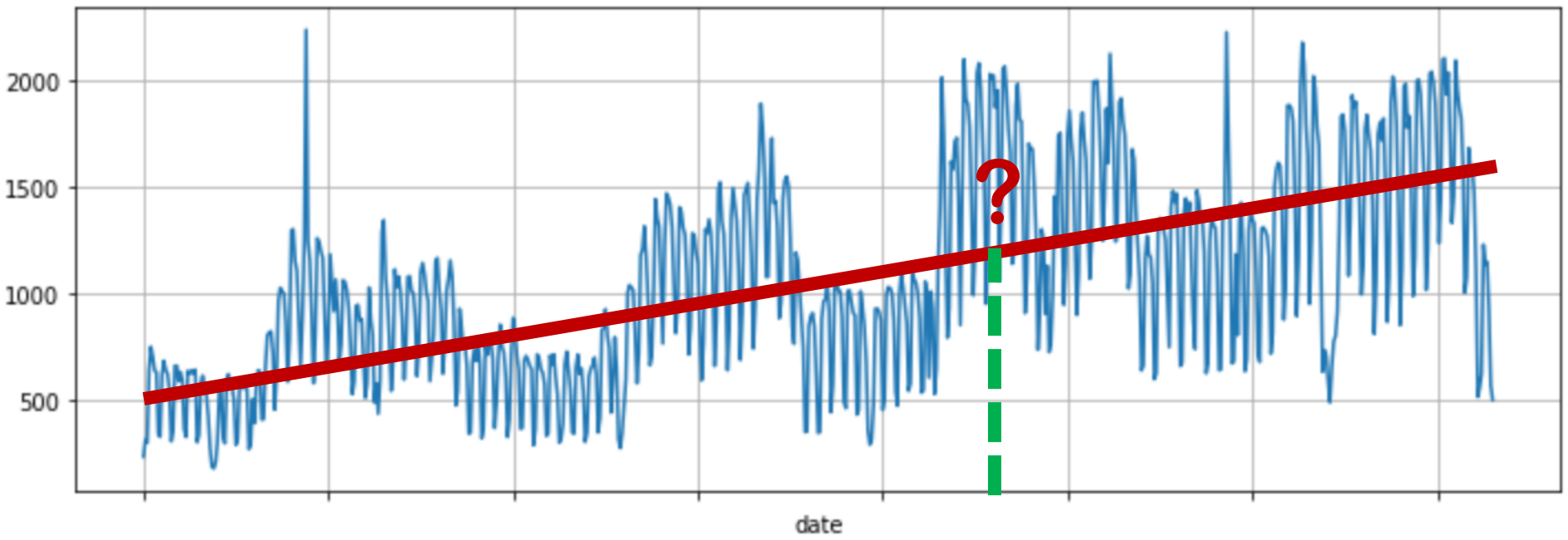
- 머신러닝 이란
  - 어떠한 작업에 대해
  - 주어진 데이터를 학습하여
  - 정의된 성능을 향상 시키는 것


학습 대상이 되는 **Data** 준비
목표 모델 혹은 구조 정의
원하는 값 혹은 상태 예측
학습 결과 확인

Linear Regression using NUMPY



- 이 데이터를 가지고 직선으로 표현한 후...
- 원하는 시점의 데이터를 예측하는 것



- 만약 누가 어떤 시점의 대표값을 뭐라고 할 수 있을까?

In [1]:

```
import pandas as pd
import numpy as np
from matplotlib import font_manager, rc
import matplotlib.pyplot as plt
%matplotlib inline

plt.rcParams['axes.unicode_minus'] = False

# f_path = "c:/Windows/Fonts/malgun.ttf"
f_path = "/Users/pinkwink/Library/Fonts/D2Coding-Ver1.3-20171129.ttf"
# f_path = "/Library/Fonts/AppleGothic.ttf"
font_name = font_manager.FontProperties(fname=f_path).get_name()
rc('font', family=font_name)
```

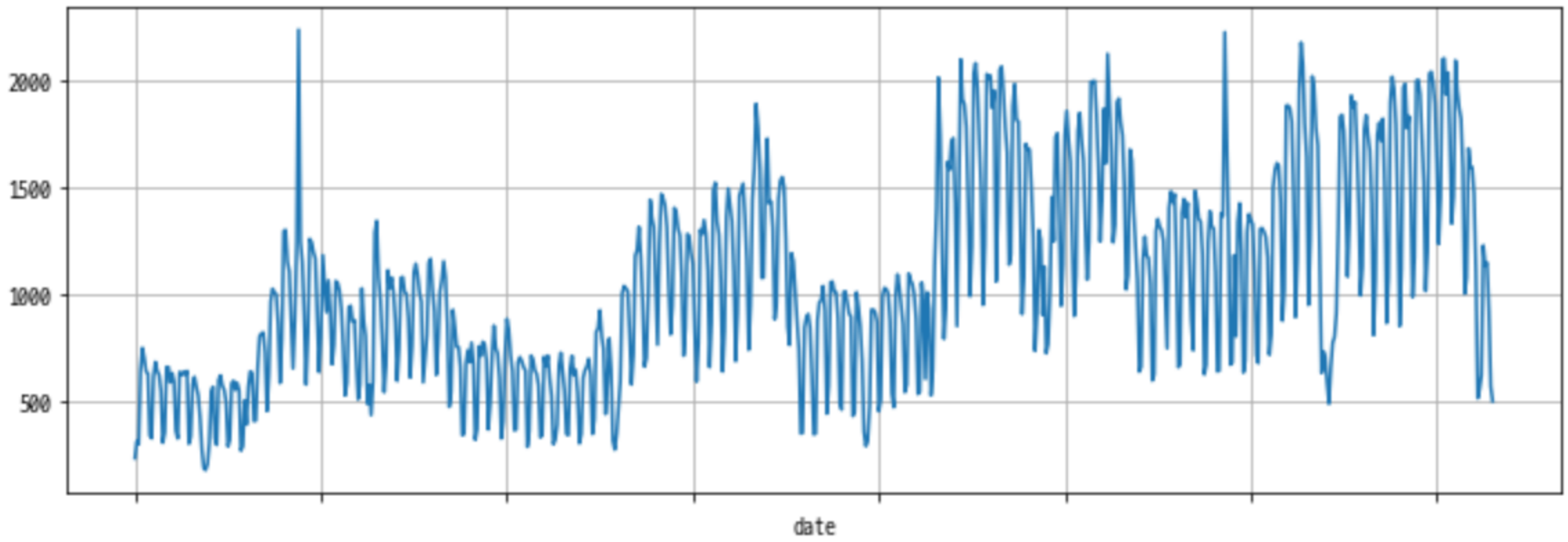
In [2]:

```
pinkwink_web = pd.read_csv('07_PW_Web.csv',
                           encoding='utf-8', thousands=',',
                           names = ['date','hit'], index_col=0)
pinkwink_web = pinkwink_web[pinkwink_web['hit'].notnull()]
pinkwink_web.head()
```

date	hit
16. 1. 1.	238
16. 1. 2.	316
16. 1. 3.	303
16. 1. 4.	645
16. 1. 5.	753

In [3]:

```
pinkwink_web['hit'].plot(figsize=(12,4), grid=True);
```

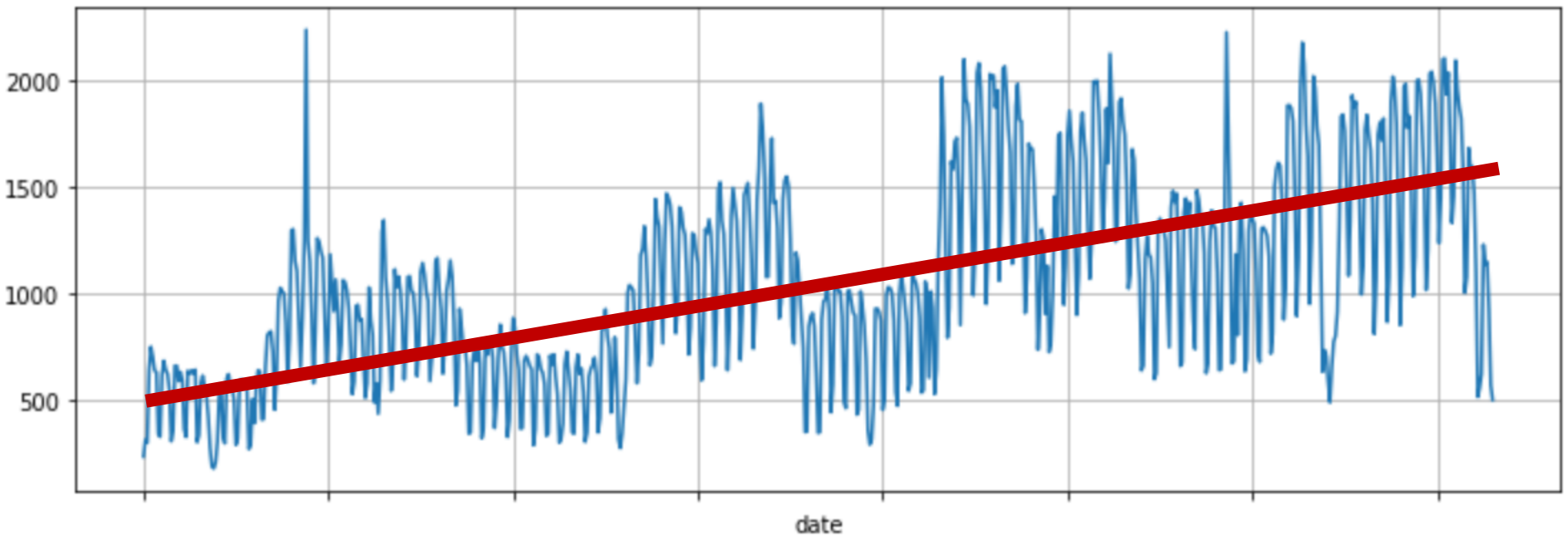


In [4]:

```
pinkwink_web['cnt'] = np.arange(0,len(pinkwink_web),1)
pinkwink_web.head()
```

date	hit	cnt
16. 1. 1.	238	0
16. 1. 2.	316	1
16. 1. 3.	303	2
16. 1. 4.	645	3
16. 1. 5.	753	4

- 학습을 위한 **x** 축을 만들어야 해서 **cnt** 라는 컬럼을 하나 만든다



- 우리의 목적은 위 데이터를 직선으로 표현하는 것
- 많은 방법 중에 첫 시작은 **numpy**를 이용해서 직선을 찾는 것

In [5]:

```
fp1 = np.polyfit(pinkwink_web['cnt'], pinkwink_web['hit'], 1)
fp1
```

```
array([   1.33240162, 573.15630957])
```

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

- 직선을 얻기 위해서 알아야하는 계수는 기울기와 절편...
- 그걸 찾아주는 **np.polyfit**

In [6]:
```
f1 = np.poly1d(fp1)
f1
```

```
poly1d([  1.33240162, 573.15630957])
```

- 알아낸 계수로

$$f(300) = ?$$

이런 값을 추정할려면 함수를 만들어야 한다..

- **np.poly1d ➔** 근데 반환값이 **poly1d???**

```
np.poly1d([1, 1]) + np.poly1d([1, -1])
```

```
np.poly1d([1, 1]) * np.poly1d([1, -1])
```

```
np.poly1d([1, -2, 1]) / np.poly1d([1, -1])
```

- 위 예제를 테스트 해보자...
- **polynomial class**의 위력 ~~

In [6]:
```
f1 = np.poly1d(fp1)
f1
```

```
poly1d([   1.33240162, 573.15630957])
```

In [8]:
```
f1(400)
```

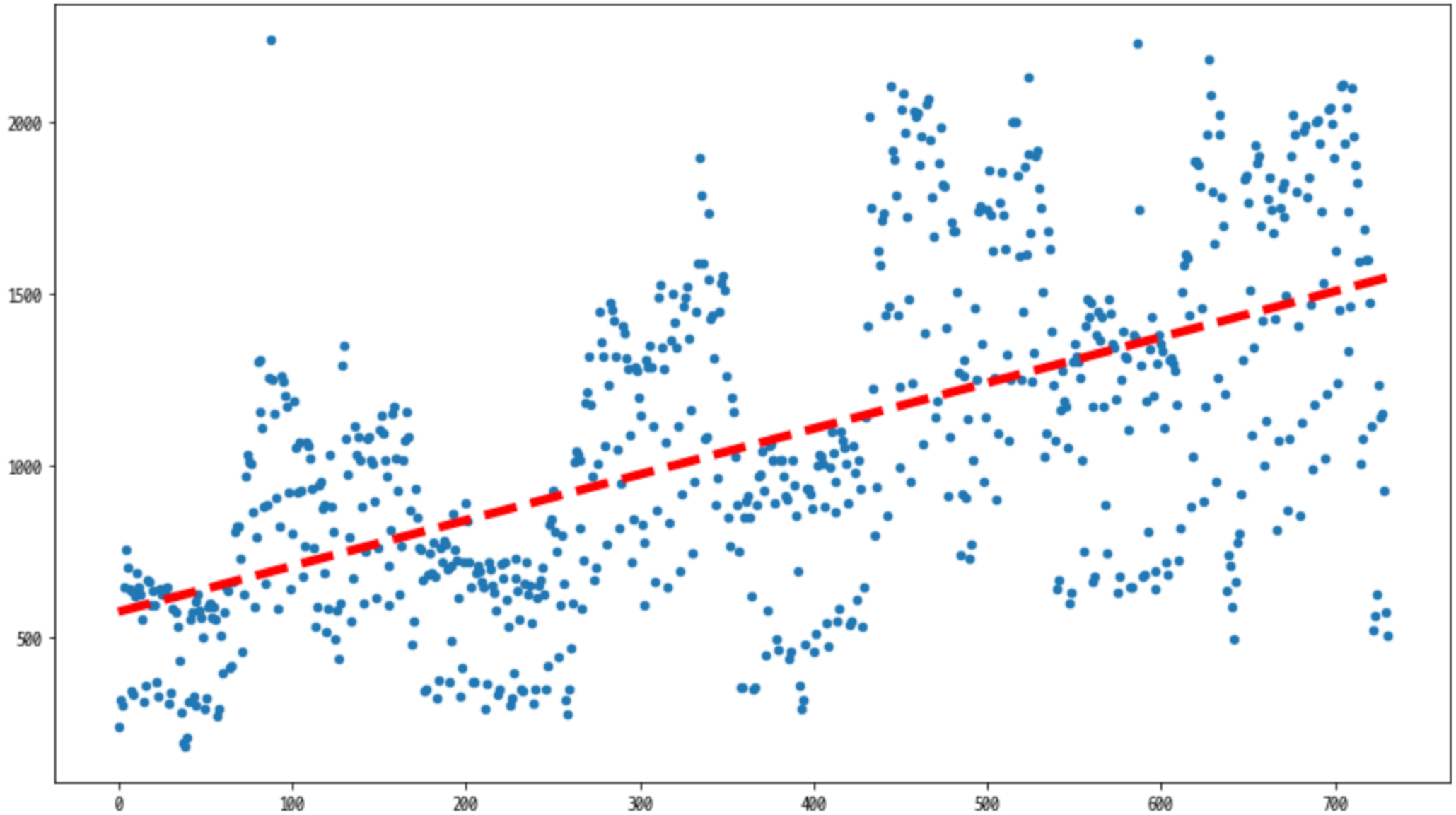
```
1106.1169567665693
```

- **f1**을 함수처럼 만든 것이다.
- **polyfit → poly1d**

In [7]:

```
plt.figure(figsize=(14,8))
plt.scatter(pinkwink_web['cnt'], pinkwink_web['hit'], s=20)
plt.plot(pinkwink_web['cnt'], f1(pinkwink_web['cnt']), ls='dashed', lw=5, color='r')
plt.show()
```

- 혹시 수업 마치고... 집에서 복습들을 하시나요?
- 이제 이 코드 정도는 이해가????


2000
1500
1000
500
0
100
200
300
400
500
600
700

Linear Regression using Neural Network

In [9]:

```
raw_data = np.genfromtxt('x09.txt', skip_header=36)
raw_data
```

```
array([[  1.,   1.,  84.,  46., 354.],
       [  2.,   1.,  73.,  20., 190.],
       [  3.,   1.,  65.,  52., 405.],
       [  4.,   1.,  70.,  30., 263.],
       [  5.,   1.,  76.,  57., 451.],
       [  6.,   1.,  69.,  25., 302.],
       [  7.,   1.,  63.,  28., 288.],
       [  8.,   1.,  72.,  36., 385.],
       [  9.,   1.,  79.,  57., 402.],
       [ 10.,   1.,  75.,  44., 365.],
       [ 11.,   1.,  27.,  24., 209.],
       [ 12.,   1.,  89.,  31., 290.],
```

- **Numpy**도 **txt** 파일을 읽는 능력이 있다.

x09.txt

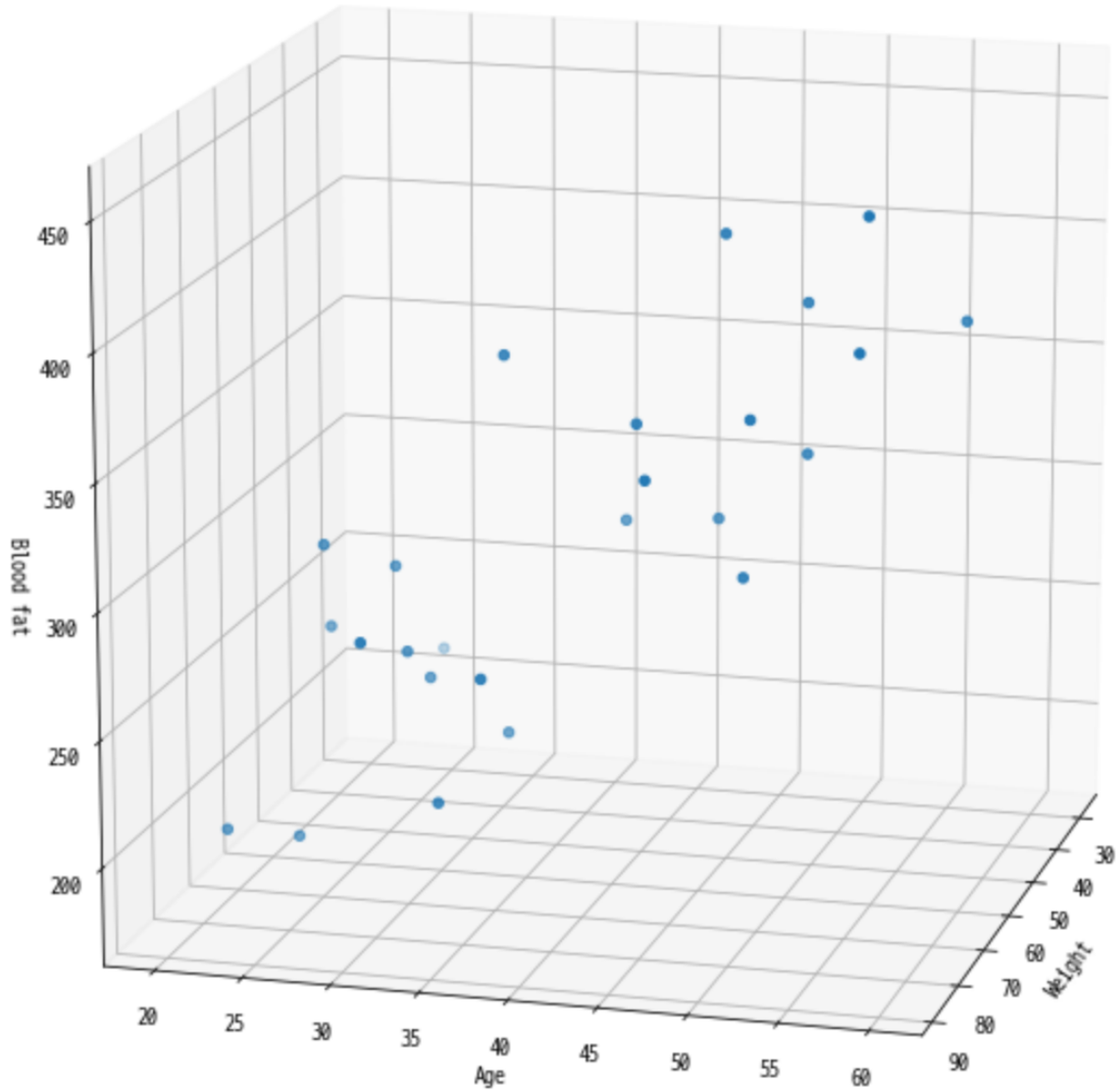
```
#  x09.txt
#
#  Reference:
#
#    D G Kleinbaum and L L Kupper,
#    Applied Regression Analysis and Other Multivariable Methods,
#    Duxbury Press, 1978, page 149.
#
#    Helmut Spaeth,
#    Mathematical Algorithms for Linear Regression,
#    Academic Press, 1991,
#    ISBN 0-12-656460-4.
#
#  Discussion:
#
#    Age and weight are to be related to blood fat content.
#
#    There are 25 rows of data.  The data columns include:
#
#      I,  the index;
#      A0, 1;
#      A1, the weight;
#      A2, the age;
#      B,  the blood fat content.
#
#    We seek a model of the form:
#
#      B = A0 * X0 + A1 * X1 + A2 * X2.
#
5 columns
```

In [13]:

```
from mpl_toolkits.mplot3d import Axes3D
%matplotlib notebook

xs = np.array(raw_data[:,2], dtype=np.float32)
ys = np.array(raw_data[:,3], dtype=np.float32)
zs = np.array(raw_data[:,4], dtype=np.float32)

# fig = plt.figure(figsize=(12,12))
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xs, ys, zs)
ax.set_xlabel('Weight')
ax.set_ylabel('Age')
ax.set_zlabel('Blood fat')
ax.view_init(15,15)
plt.show()
```

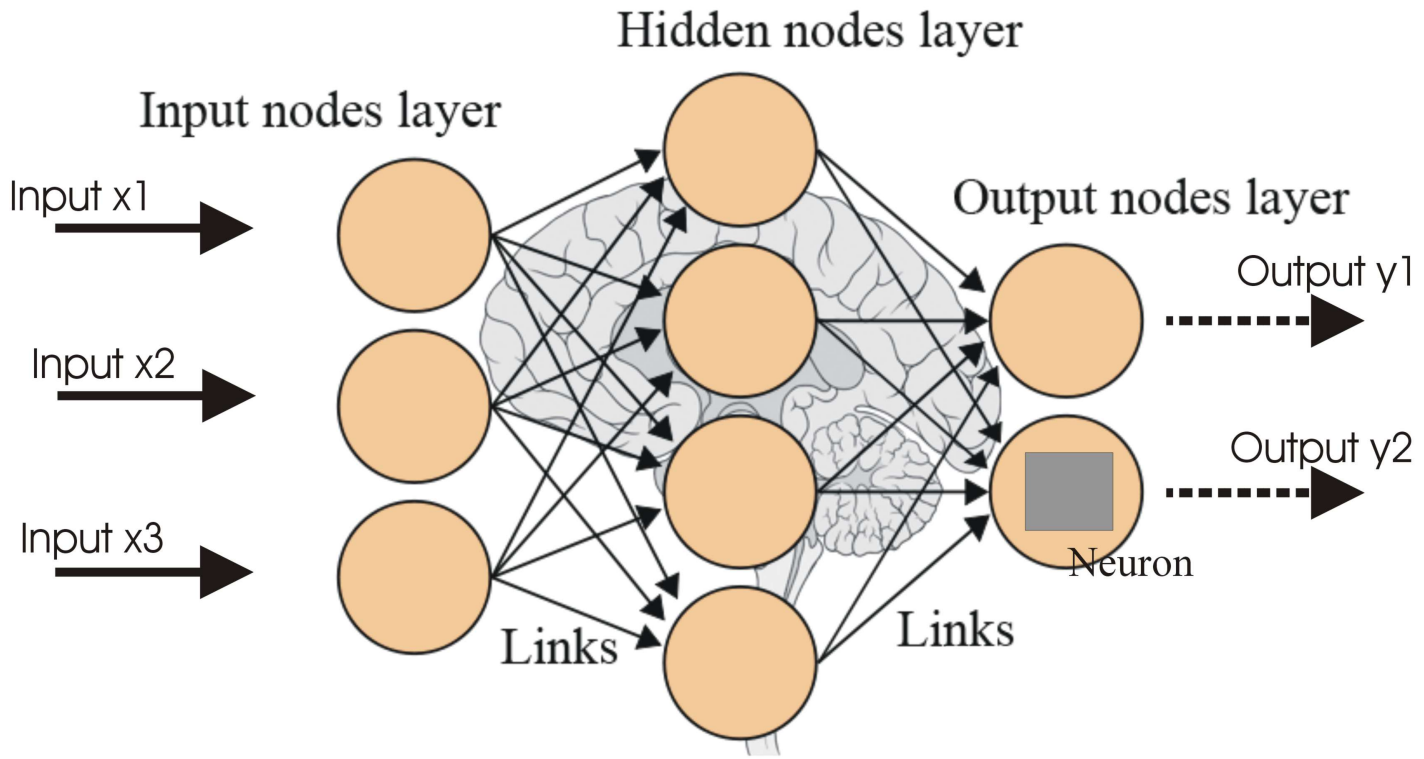

Blood fat
450
400
350
300
250
200
Age
20
25
30
35
40
45
50
55
60
Weight
30
40
50
60
70
80
90

뉴런

신경

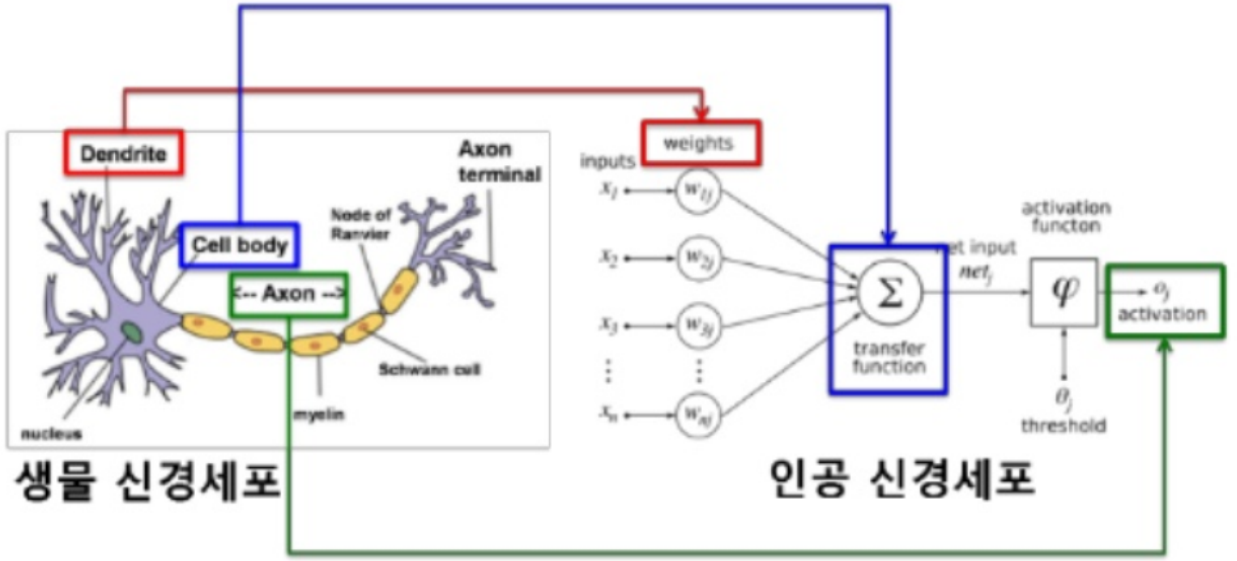
기관

- 인간의 뇌에는 약 천 억 개 정도의 뉴런이 있다고 한다...

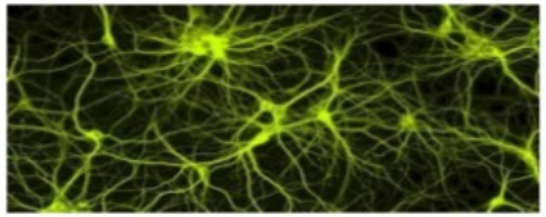


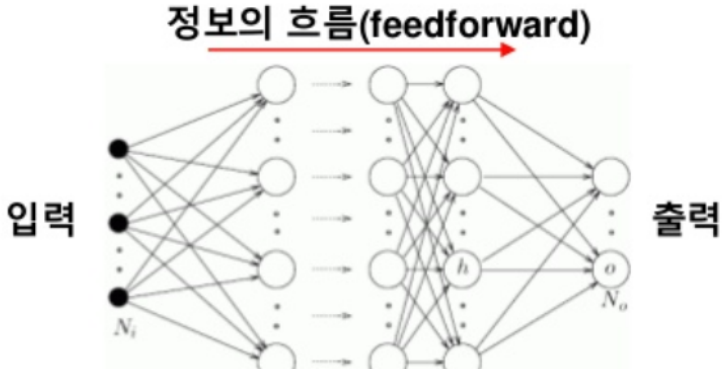
- 그걸 흉내내서 구현한 것이 인공 신경망 알고리즘 **Neural Network**

뉴런

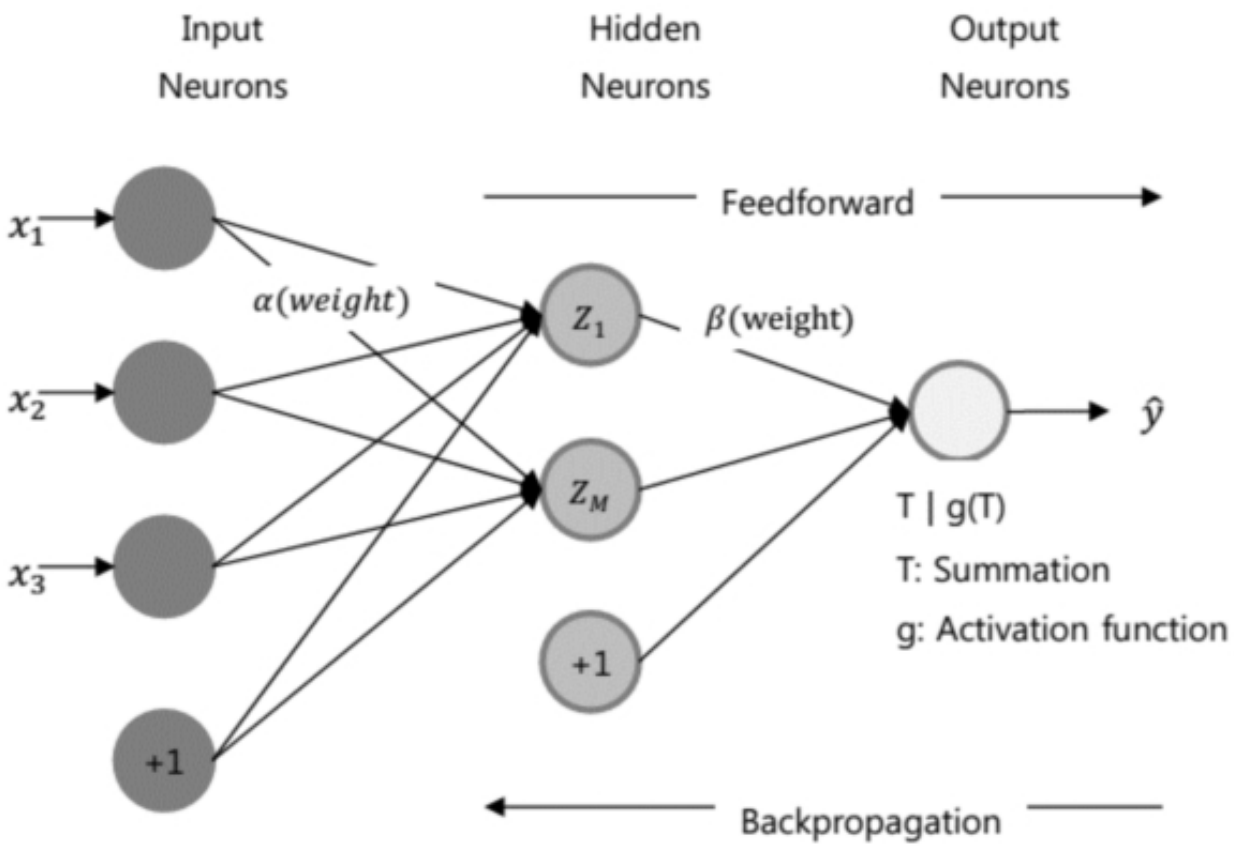


신경망

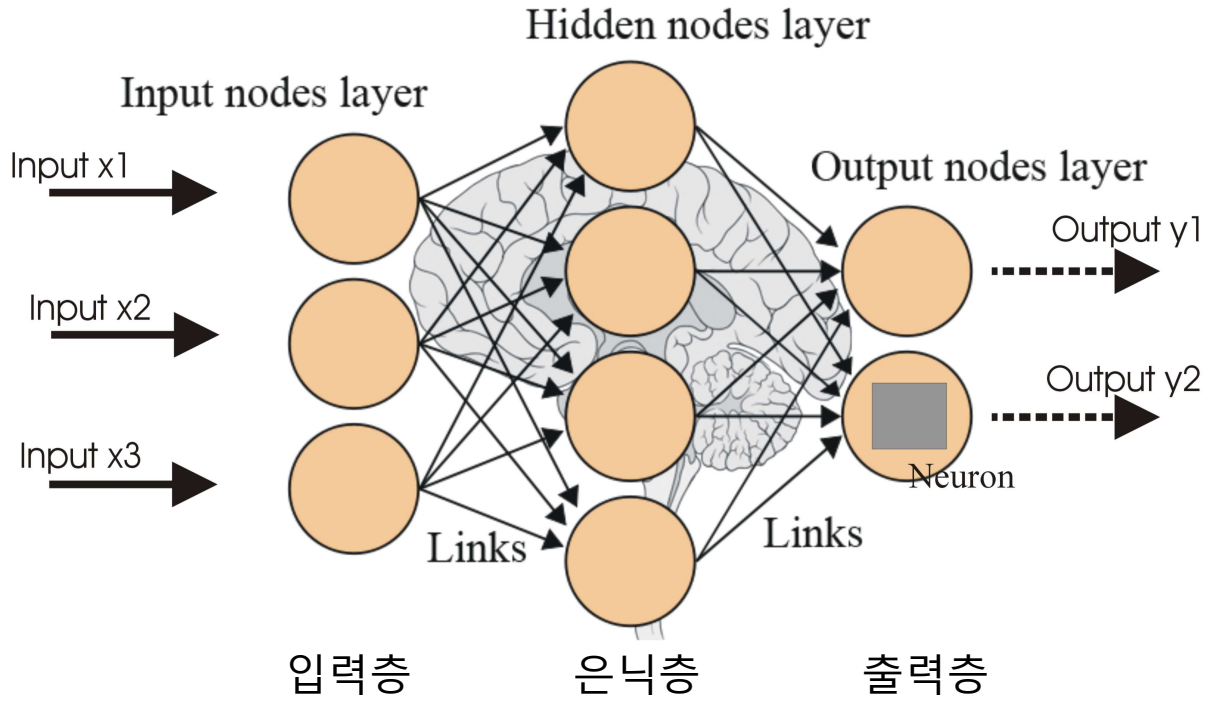




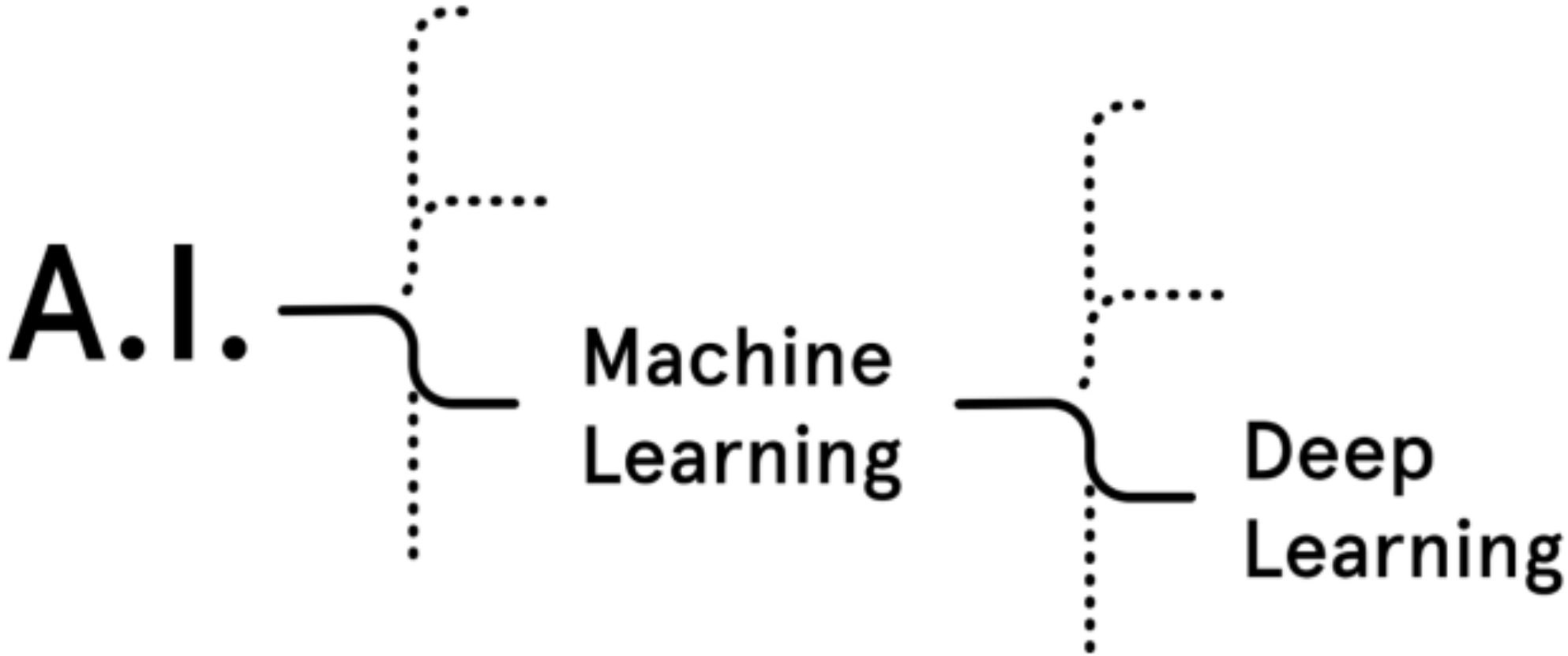
- 사람의 뇌는 뉴런의 연결 관계를 가지고 학습하고...
- 인공 신경망은 노드 사이의 연결 가중치를 변경해서 학습한다.

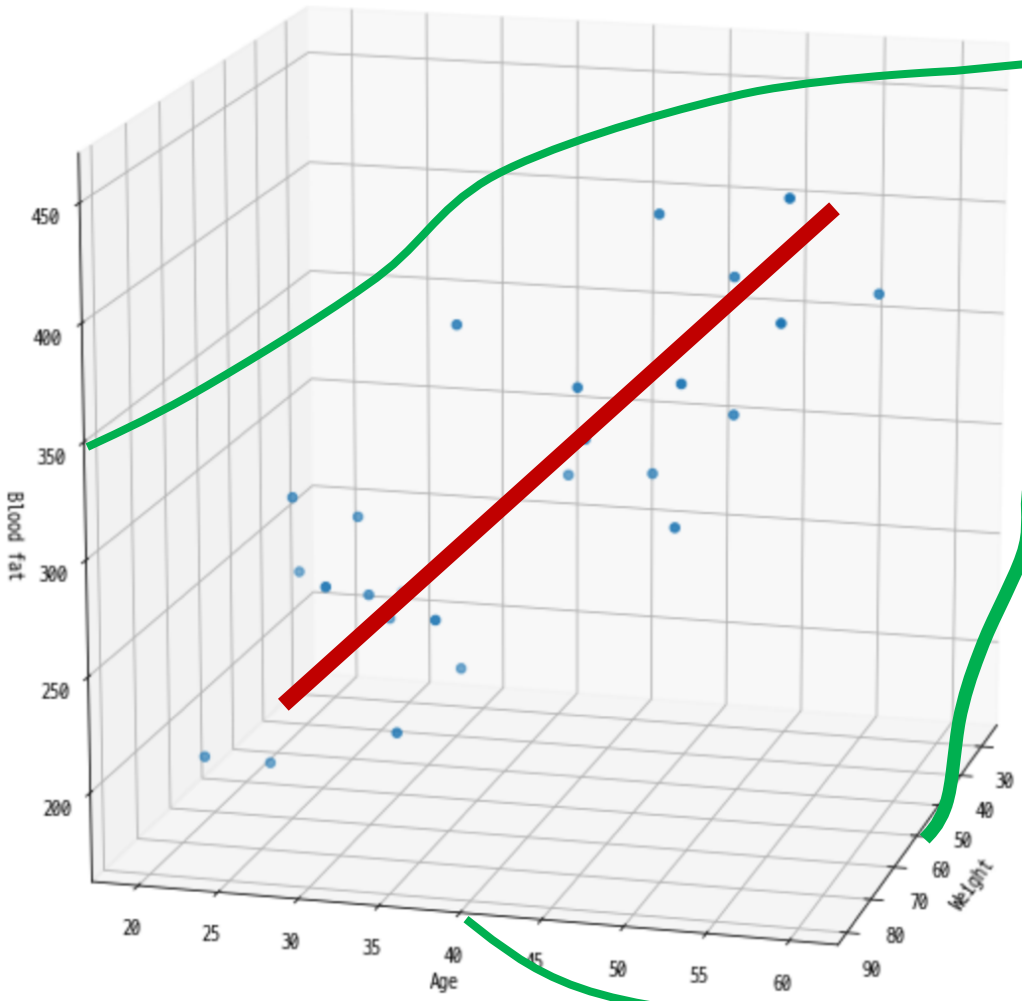


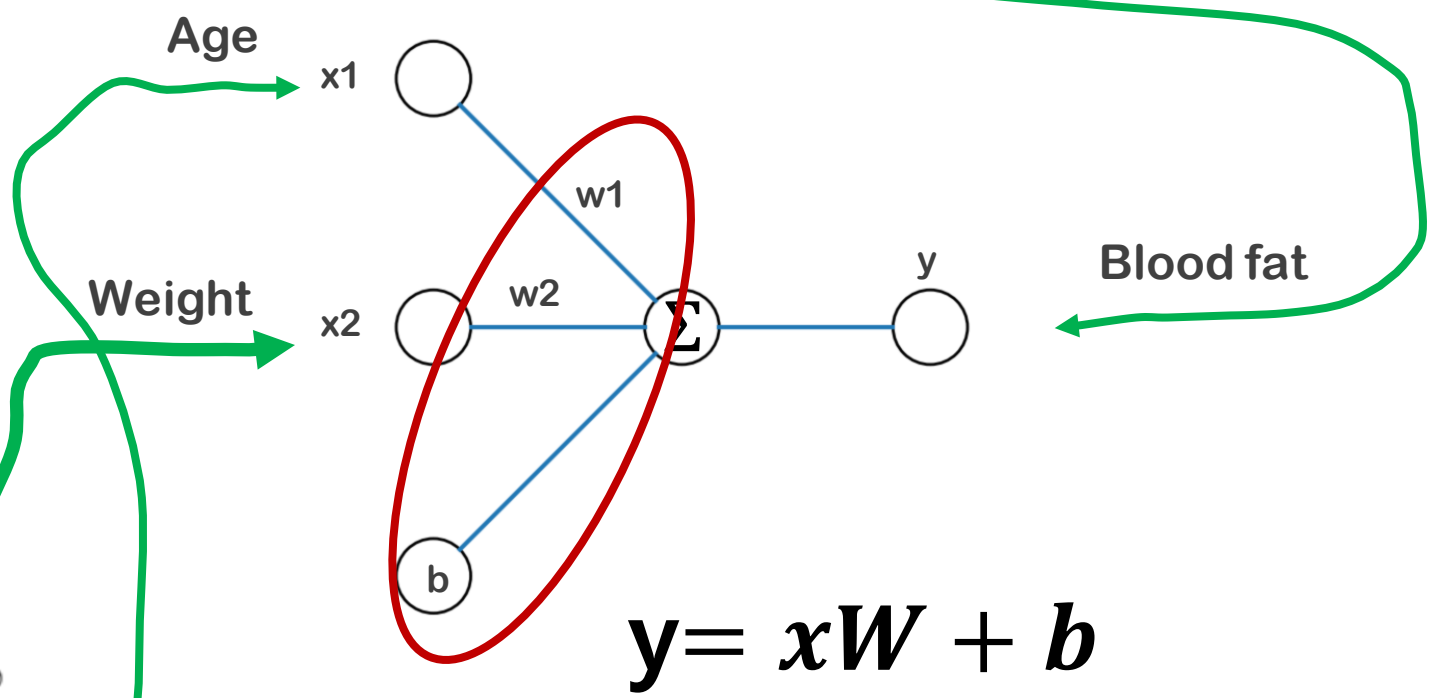
- 결국 인공 신경망은 노드 사이의 가중치를 변경해서 원하는 결과를 얻는다.



단층 신경망		입력층-출력층
다층 신경망	얕은 신경망	입력 – 은닉 – 출력
	심층 신경망	입력 – 은닉 … 은닉 - 출력

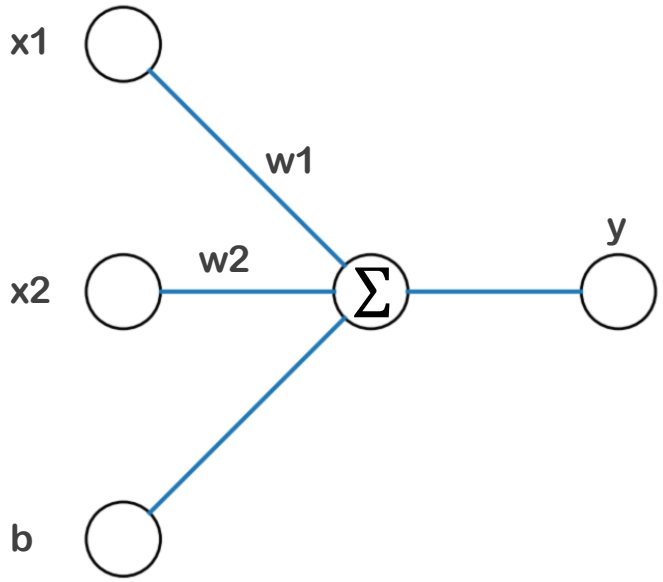

A.I.
Machine Learning
Deep Learning





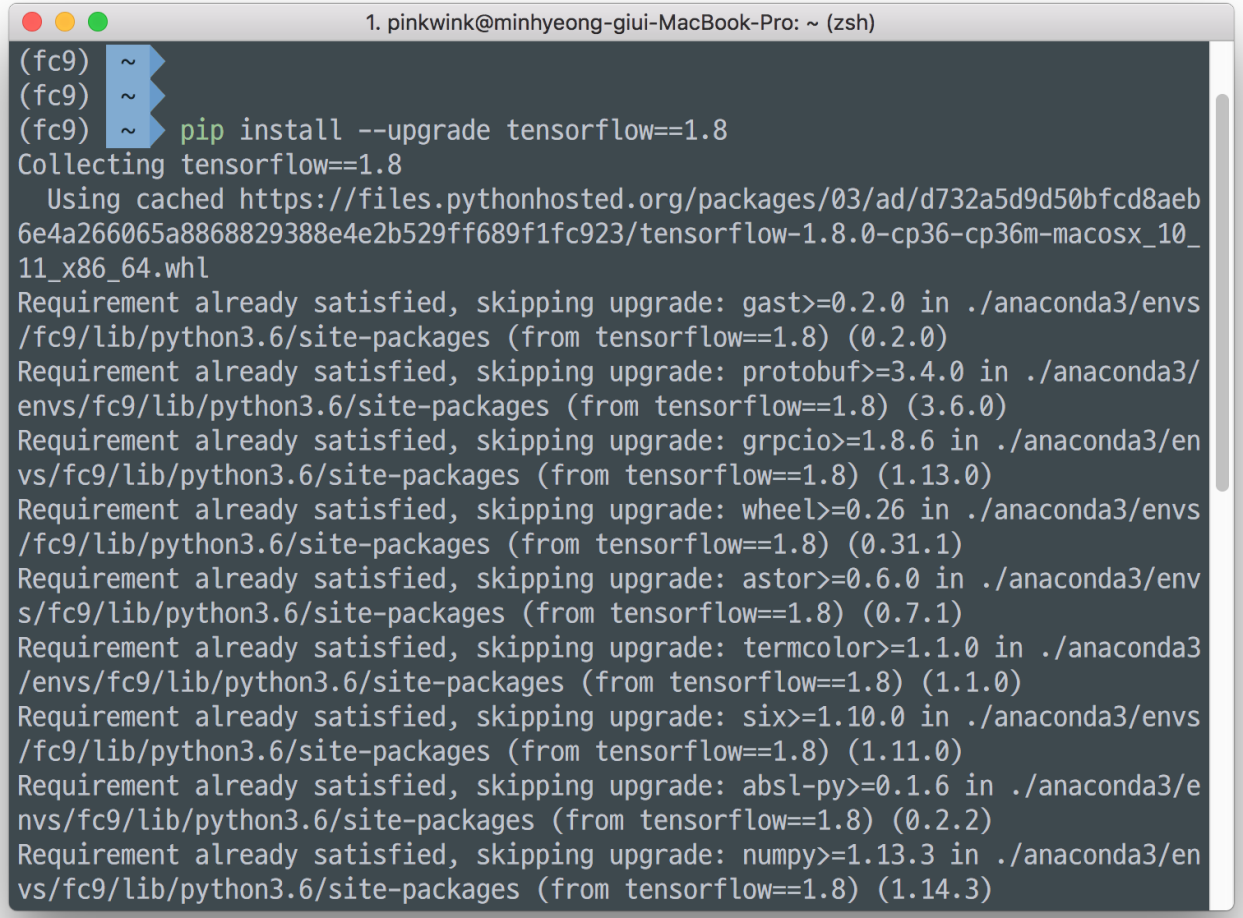
$$\mathbf{y} = \boldsymbol{xW} + \boldsymbol{b}$$

$$y = XW + b$$

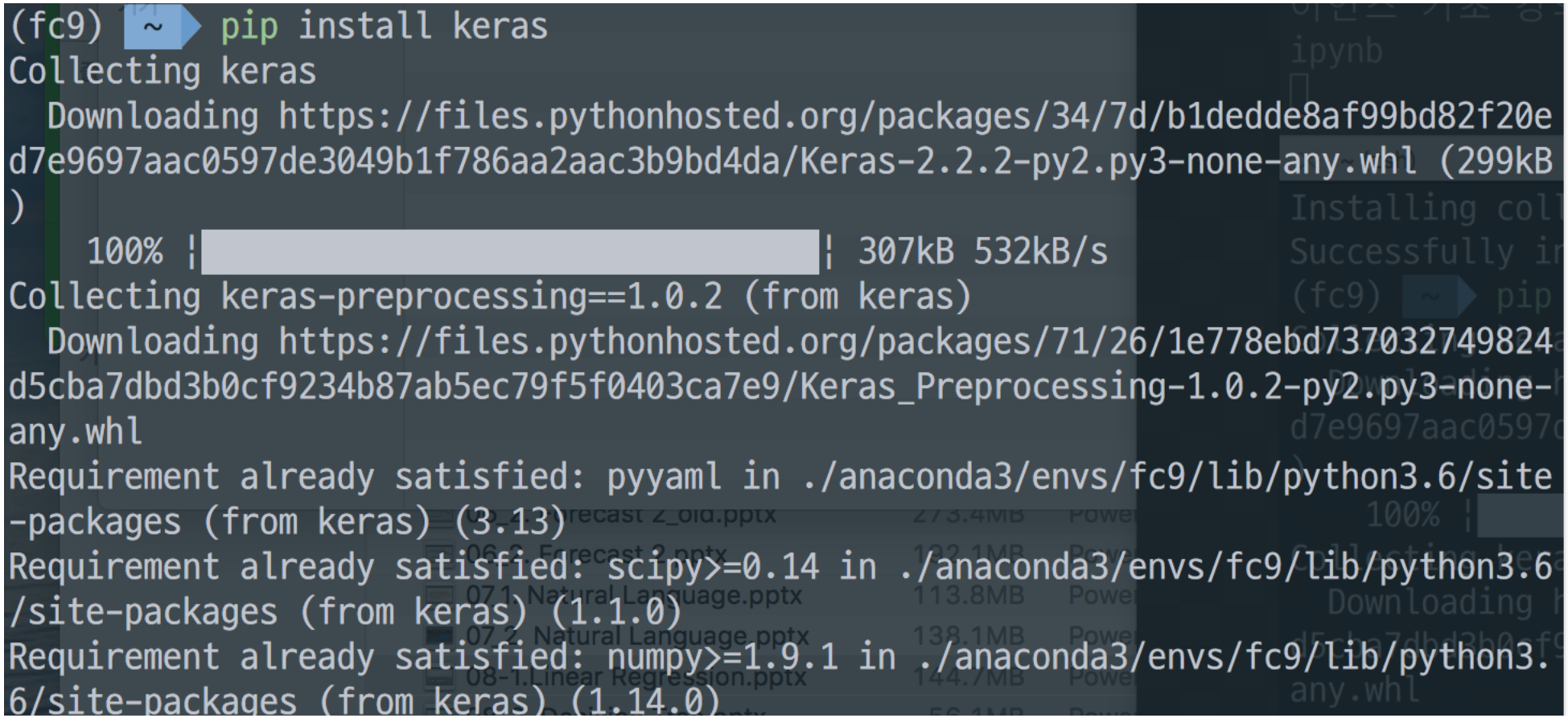


- 목적: **x1, x2**를 입력해서 **y**가 나오게 하는 **Weight**와 **bias**를 구하는 것

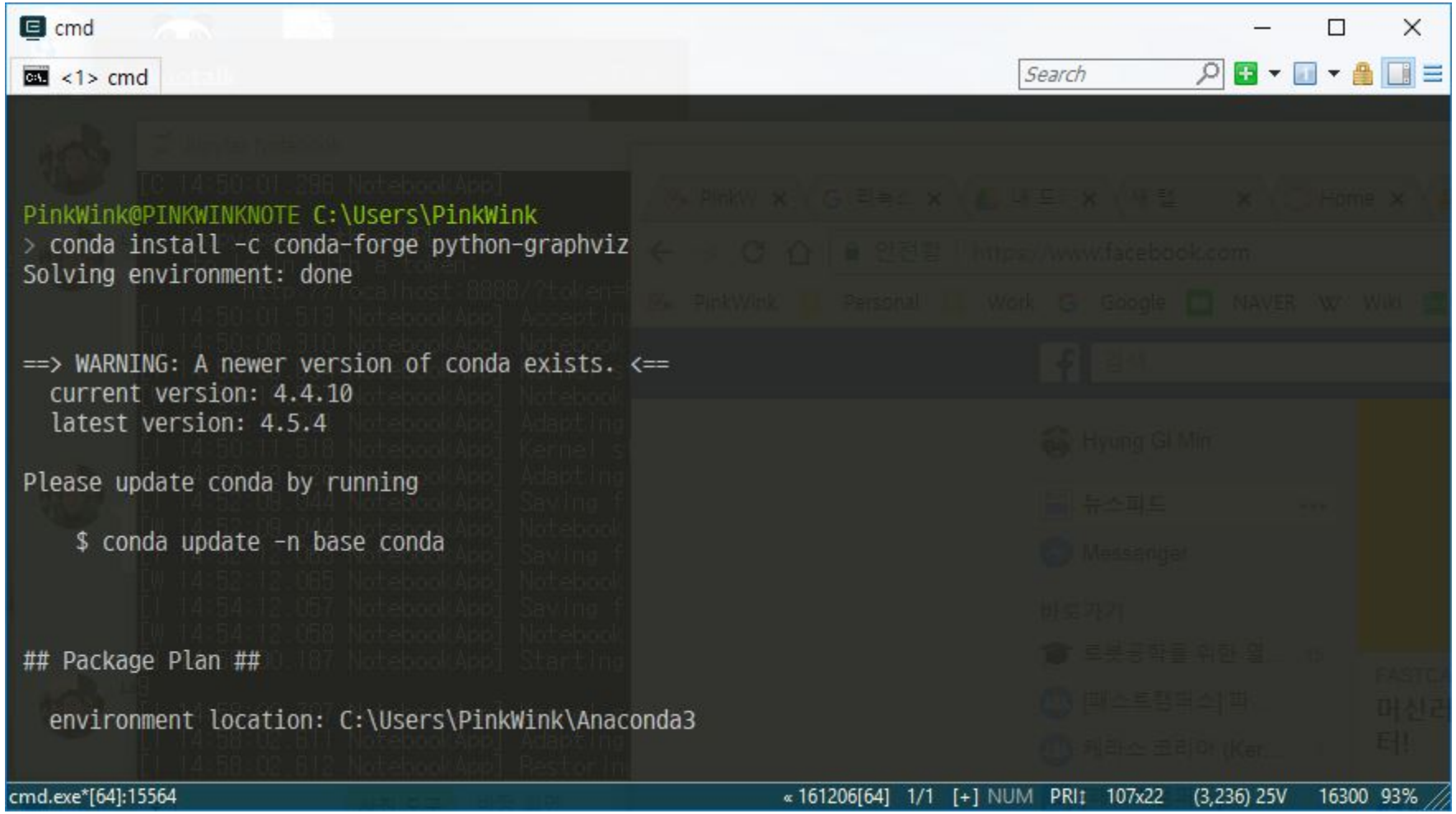
- **Weight**와 **Bias**를 얻기 위해 코드를 짜긴 해야 하는데ㅠㅠ.
- 어떻게**???**
- 이럴때 사용하는 것이 **Tensorflow, Keras, PyTorch** 등이다…
- 오늘 우리는 **Keras** 도전 **~**



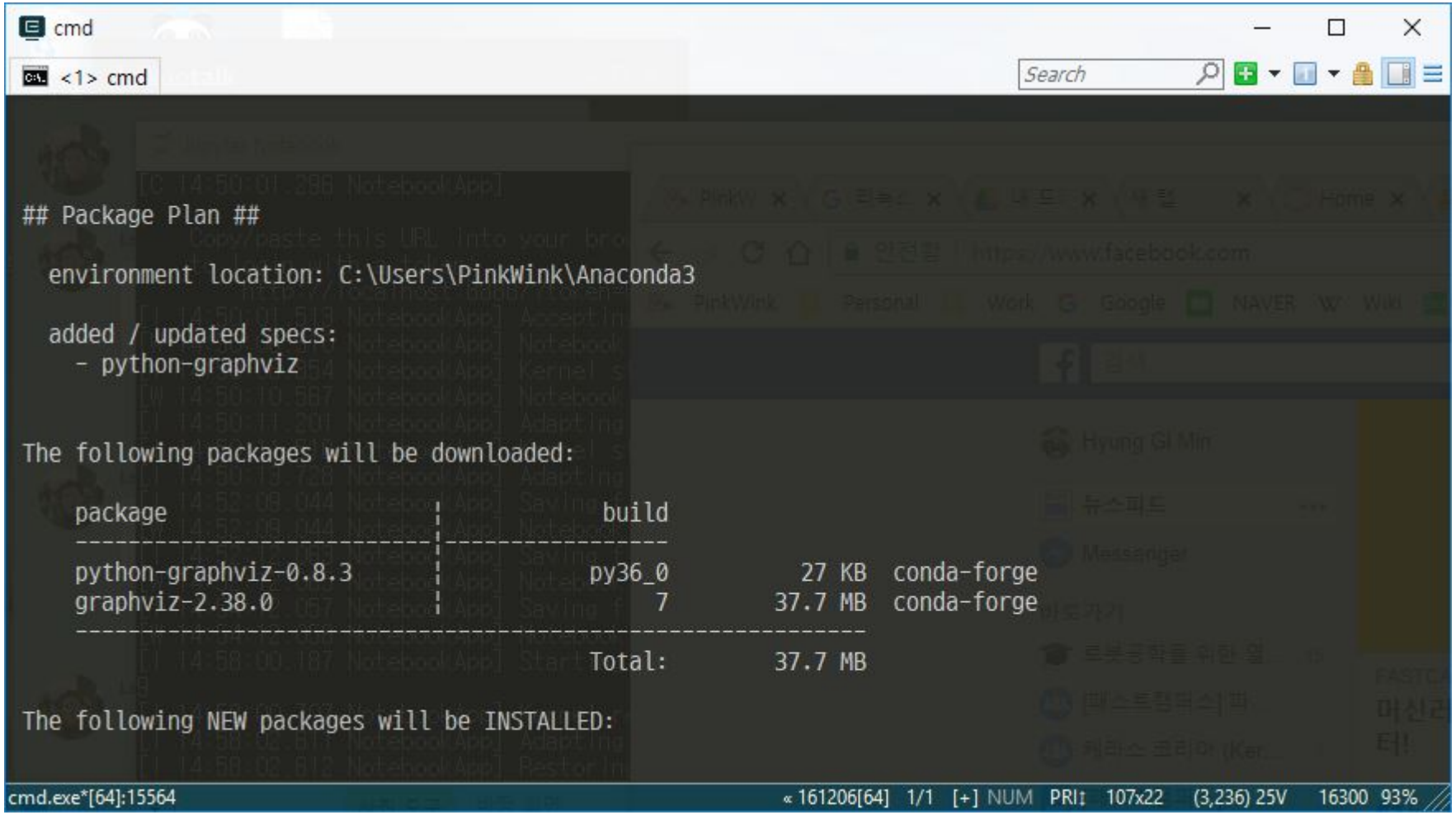
- **pip install –upgrade tensorflow==1.8**



- pip install keras==2.2.2



- conda install –c conda-forge python-graphviz


cmd
<1> cmd
Search
Package Plan
environment location: C:\Users\PinkWink\Anaconda3
added / updated specs:
- python-graphviz
The following packages will be downloaded:
package | build
python-graphviz-0.8.3 | py36_0 27 KB conda-forge
graphviz-2.38.0 | 7 37.7 MB conda-forge
Total: 37.7 MB
The following NEW packages will be INSTALLED:
cmd.exe*[64]:15564
« 161206[64] 1/1 [+] NUM PRI↕ 107x22 (3,236) 25V 16300 93%

In [13]:

```
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers.core import Dense
from keras.optimizers import RMSprop

np.random.seed(13)
```

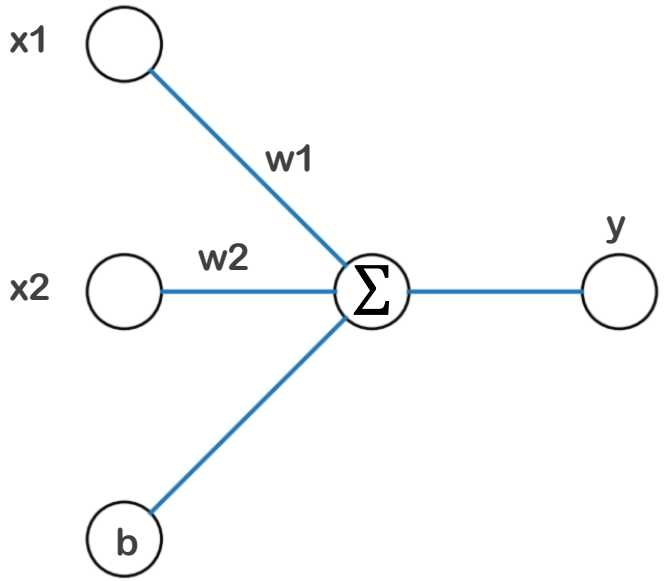
- **Keras**는 **Tensorflow**를 백엔드로 하는 상위 **API** 개념이다

In [14]:

```
x_data = np.array(raw_data[:,2:4], dtype=np.float32)
y_data = np.array(raw_data[:,4], dtype=np.float32)
y_data = y_data.reshape((25,1))

rmsprop = RMSprop(lr=0.01)
```

- 학습해야 할 데이터만 추리자



In [15]:

```
model = Sequential()
model.add(Dense(1, input_shape=(2,)))
model.compile(loss='mse', optimizer=rmsprop)
model.summary()
```

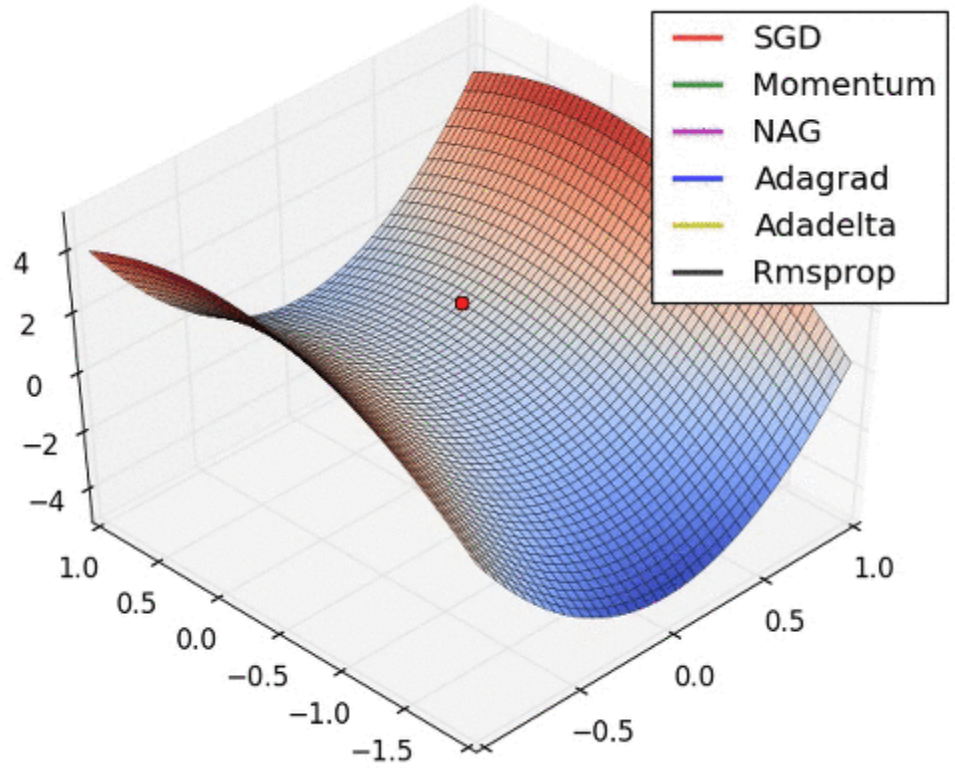
In [15]:

```
model = Sequential()
model.add(Dense(1, input_shape=(2,)))
model.compile(loss='mse', optimizer=rmsprop)
model.summary()
```

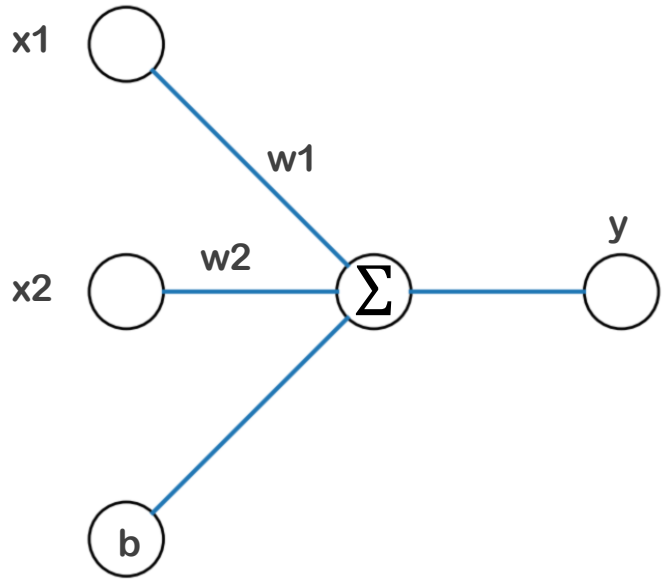
- 학습을 위해서는 **loss (cost)** 함수를 정해주어야 한다
- 이번에는 **mse:mean square error** 오차 제곱의 평균을 사용

```
In [15]:
model = Sequential()
model.add(Dense(1, input_shape=(2,)))
model.compile(loss='mse', optimizer=rmsprop)
model.summary()
```

- 그리고, **optimizer**도 정해주어야 한다.
- **optimizer**는 **loss** 함수를 최소화하는 가중치를 찾아가는 과정에 대한 알고리즘이다.
- 여기서는 **rmsprop**를 사용



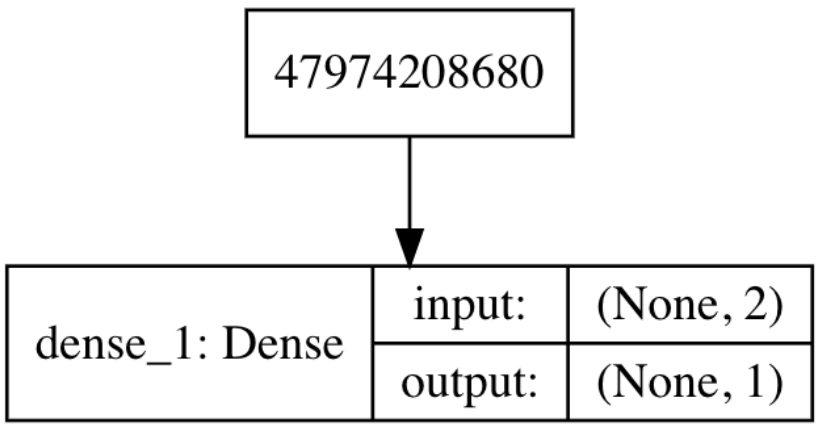
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```



In [16]:

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
%matplotlib inline

SVG(model_to_dot(model, show_shapes=True).create(prog='dot', format='svg'))
```



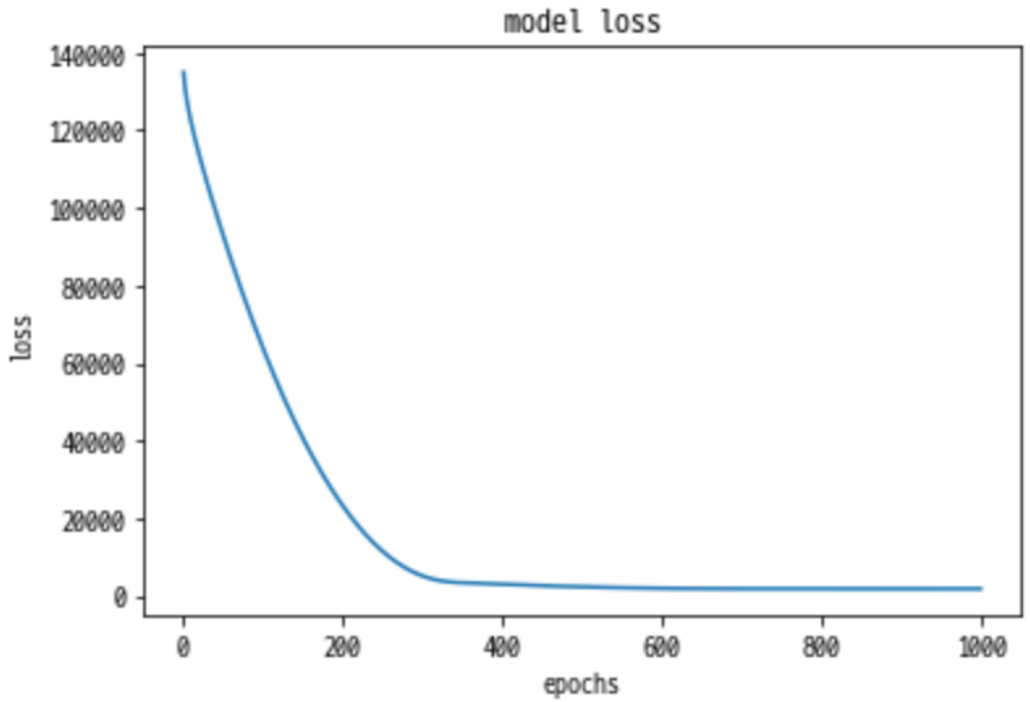
In [17]:
```
hist = model.fit(x_data, y_data, epochs=1000)
```

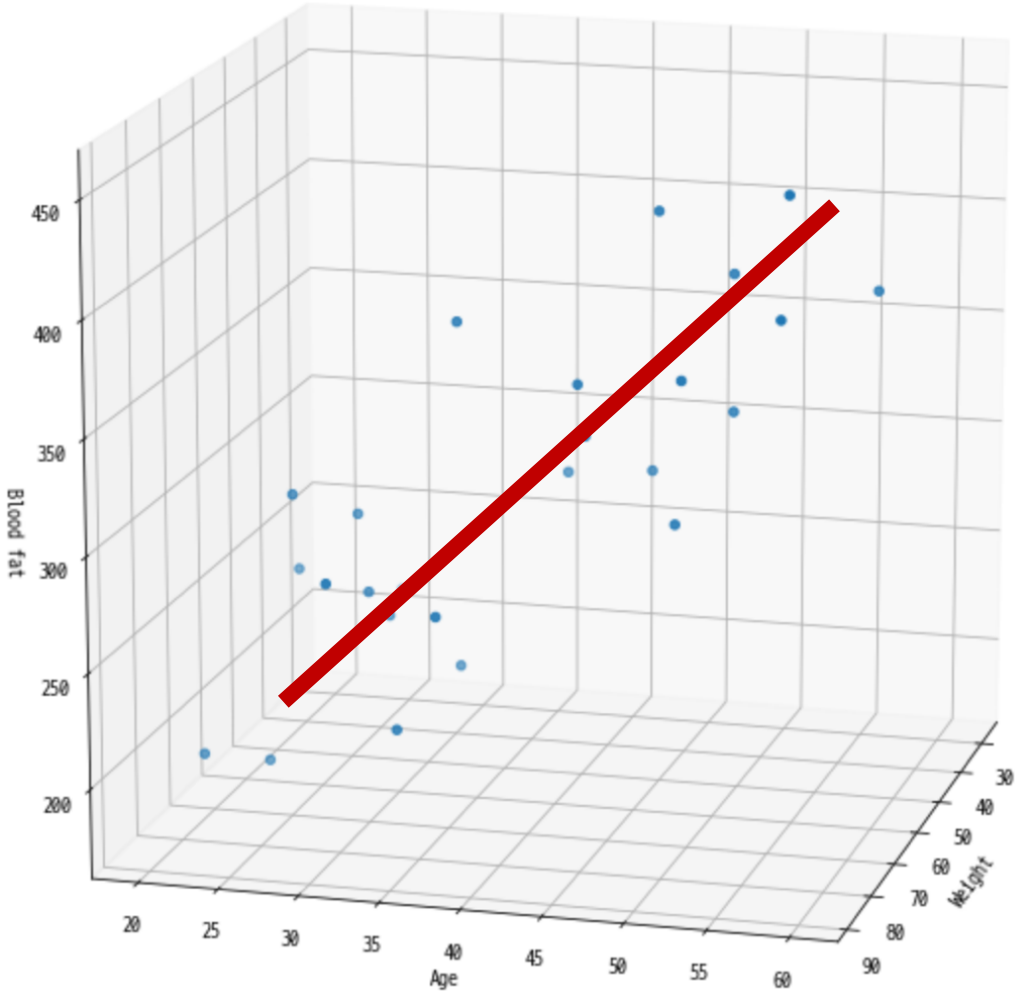
```
Epoch 1/1000
25/25 [==============================] - 0s 4ms/step - loss: 134896.7344
Epoch 2/1000
25/25 [==============================] - 0s 39us/step - loss: 132375.2031
Epoch 3/1000
25/25 [==============================] - 0s 46us/step - loss: 130569.5000
Epoch 4/1000
25/25 [==============================] - 0s 47us/step - loss: 129071.4922
Epoch 5/1000
25/25 [==============================] - 0s 37us/step - loss: 127751.9062
```

- 방금 전까지 모델**(model)**을 만들었다
- 이제 학습 시작 **fit ~~~**
- **epochs**는 몇 번 학습할지 정했다고 생각하자

In [19]:

```
plt.plot(hist.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.show()
```





- 데이터를 이용해서
- 모델을 만들고
- 학습했으니..
- 예측해보자
- 몸무게 **100**에 **44**살 된 사람의

  이 데이터에서의

  Blood Fat은**???**

```
In [20]: model.predict(np.array([100,44]).reshape(1,2))

array([[372.68344]], dtype=float32)

In [21]: model.predict(np.array([60,25]).reshape(1,2))

array([[219.60904]], dtype=float32)
```

- **Keras**에서는 **predict** 함수를 사용하면 된다

In [22]:
```
W_, b_ = model.get_weights()
print('Weight is : ', W_)
print('bias is : ', b_)
```

```
Weight is :  [[1.1937457]
 [5.543399 ]]
bias is :  [9.399319]
```

- 모델의 가중치를 얻어올 수 있다

In [25]:

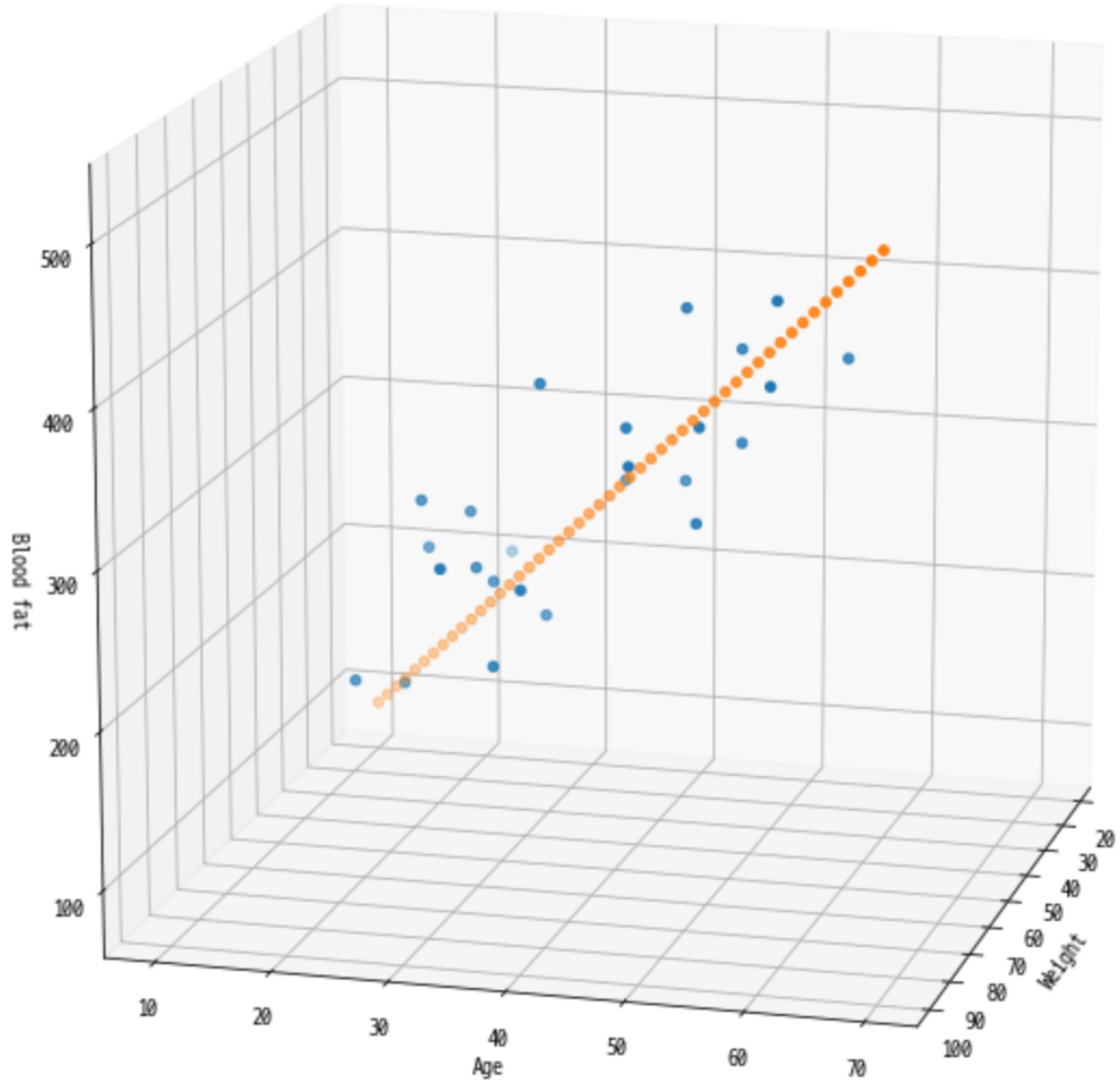
```
x = np.linspace(20,100,50).reshape(50,1)
y = np.linspace(10,70,50).reshape(50,1)

X = np.concatenate((x,y), axis=1)
Z = np.matmul(X, W_) + b_
```

- 그래프를 그리기 위한 데이터를 만들고
- **matmul**을 이용해서 직선의 결과를 만듦

In [24]:

```
%matplotlib notebook
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xs, ys, zs)
ax.scatter(x, y, Z)
ax.set_xlabel('Weight')
ax.set_ylabel('Age')
ax.set_zlabel('Blood fat')
ax.view_init(15,15)
plt.show()
```


Blood fat
500
400
300
200
100
Age
10
20
30
40
50
60
70
Weight
20
30
40
50
60
70
80
90
100