In [38]:

```
#Multiple_regression_analysis
import numpy as np
from sympy import Symbol,solve
import sympy as sym
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
```

In [45]:

```
#説明変数をX、目的変数をyとして作成する
X = np.array([[1,1],[2,2]])
y = np.array([5, 10])
y = np.reshape(y, (1,len(y)))
y
```

Out[45]:

```
array([[ 5, 10]])
```

説明変数 $X = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}$, 目的変数 $y = \begin{pmatrix} 5 & 10 \end{pmatrix}$

In [46]:

```
#切片の計算がきるようにに次元を1つ上げる
pra1_X = []
for i in range(len(X)):
        pra1_X.append(np.append(X[i],1))
pra1_X = np.array(pra1_X)
pra1_X
```

Out[46]:

```
array([[1, 1, 1],
       [2, 2, 1]])
```

$X = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}$ -----------------------> $X = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}$ (各行の3つの要素1が切片と対応)

In [80]:

```
#次元を上げた説明変数に対応する重みベクトル作成
w = []
for i in range(len(X[0])):
        obj = sym.Symbol('w' + str(i))
        w = np.append(w,obj)
w = np.append(w, sym.Symbol('b'))
w = np.reshape(w,(1,len(w)))
w
```

Out[80]:

```
array([[w0, w1, b]], dtype=object)
```

$w = \begin{pmatrix} w_0 & w_1 & b \end{pmatrix}$

In [58]:

```
fa = np.dot(pra1_X, w.T)
fa
```

Out[58]:

```
array([[b + w0 + w1],
       [b + 2*w0 + 2*w1]], dtype=object)
```

$$fa = Xw^T$$

$$fa = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ b \end{pmatrix}$$

$$fa = \begin{pmatrix} w_0 + w_1 + b \\ 2w_0 + 2w_1 + b \end{pmatrix}$$

In [69]:

```
Sqer = (y.T - fa)
Sqer
```

Out[69]:

```
array([[-b - w0 - w1 + 5],
       [-b - 2*w0 - 2*w1 + 10]], dtype=object)
```

2乗の計算をするために目的変数と説明変数の誤差を算出

$$Sqer = \begin{pmatrix} 5 \\ 10 \end{pmatrix} - \begin{pmatrix} w_0 & a_1 & b \\ 2w_0 & 2w_1 & b \end{pmatrix}$$

$$Sqer = \begin{pmatrix} 5 - w_0 - w_1 - b \\ 10 - 2w_0 - 2w_1 - b \end{pmatrix}$$

In [71]:

```
E = np.dot(Sqer.T, Sqer)
E
```

Out[71]:

```
array([[(-b - 2*w0 - 2*w1 + 10)**2 + (-b - w0 - w1 + 5)**2]], dtype=object)
```

$$E = Sqer^T Sqer$$

$$E = \begin{pmatrix} 5 - w_0 - w_1 - b & 10 - 2w_0 - 2w_1 - b \end{pmatrix} \begin{pmatrix} 5 - w_0 - w_1 - b \\ 10 - 2w_0 - 2w_1 - b \end{pmatrix}$$

$$E = \begin{pmatrix} 5 - w_0 - w_1 - b \end{pmatrix}^2 + (10 - 2w_0 - 2w_1 - b)^2$$

In [95]:

```
#2乗和誤差を各重みで偏微分
dE = np.array(sym.diff(E, w))
dE
```

Out[95]:

```
array([6*b + 10*w0 + 10*w1 - 50, 6*b + 10*w0 + 10*w1 - 50,
       4*b + 6*w0 + 6*w1 - 30], dtype=object)
```

$$E = \left(5 - w_0 - w_1 - b \right)^2 + (10 - 2w_0 - 2w_1 - b)^2$$

$$\frac{dE}{dw} = \begin{pmatrix} 2\left(5 - w_0 - w_1 - b \right)(-1) + 2(10 - 2w_0 - 2w_1 - b)(-2) \\ 2\left(5 - w_0 - w_1 - b \right)(-1) + 2(10 - 2w_0 - 2w_1 - b)(-2) \\ 2\left(5 - w_0 - w_1 - b \right)(-1) + 2(10 - 2w_0 - 2w_1 - b)(-1) \end{pmatrix}^T$$

$$\frac{dE}{dw} = \begin{pmatrix} -10 + 2w_0 + 2w_1 + 2b - 40 + 8w_0 + 8w_1 + 4b \\ -10 + 2w_0 + 2w_1 + 2b - 40 + 8w_0 + 8w_1 + 4b \\ -10 + 2w_0 + 2w_1 + 2b - 20 + 4w_0 + 4w_1 + 2b \end{pmatrix}^T$$

$$\frac{dE}{dw} = \begin{pmatrix} -50 + 10w_0 + 10w_1 + 6b & -50 + 10w_0 + 10w_1 + 6b & -30 + 6w_0 + 6w_1 + 4b \end{pmatrix}$$

In [122]:

```
#偏微分した値の係数を取得
poly_list = []
for i in range(len(pra1_X[0])):
    poly = sym.poly(dE[i])
    poly_list.append(poly.coeffs())
poly_list = np.array(poly_list)
poly_list
```

Out[122]:

```
array([[10, 10, 6, -50],
       [10, 10, 6, -50],
       [6, 6, 4, -30]], dtype=object)
```

$$\frac{dE}{dw} = \begin{pmatrix} 10w_0 + 10w_1 + 6b - 50 & 10w_0 + 10w_1 + 6b - 50 & 6w_0 + 6w_1 + 4b - 30 \end{pmatrix}$$

$$poly\ list = \begin{pmatrix} 10 & 10 & 6 & -50 \\ 10 & 10 & 6 & -50 \\ 6 & 6 & 4 & -30 \end{pmatrix}$$

In [123]:

```
#取得した係数を文字があるものとないものに分ける
keisuu = []
ans = []
for i in range(len(pra1_X[0])):
    keisuu.append(poly_list[i][:-1])
    ans = np.append(ans, poly_list[i][-1:])

ans = -ans
print(keisuu)
print(ans)
```

```
[array([10, 10, 6], dtype=object), array([10, 10, 6], dtype=object), array([6, 6, 4], dtyp
e=object)]
[50 50 30]
```

$$keisuu = \begin{pmatrix} 10 & 10 & 6 \\ 10 & 10 & 6 \\ 6 & 6 & 4 \end{pmatrix}, ans = \begin{pmatrix} 50 \\ 50 \\ 30 \end{pmatrix} \quad >>>>>>> \begin{pmatrix} 10 & 10 & 6 \\ 10 & 10 & 6 \\ 6 & 6 & 4 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ b \end{pmatrix} = \begin{pmatrix} 50 \\ 50 \\ 30 \end{pmatrix}$$

In [124]:

```
#keisuuの逆行列を作成
keisuu = np.array(keisuu).astype(np.float64)
Ainv = np.linalg.pinv(keisuu)
print(Ainv)
```

```
[[ 0.25  0.25 -0.75]
 [ 0.25  0.25 -0.75]
 [-0.75 -0.75  2.5 ]]
```

$$\begin{pmatrix} 10 & 10 & 6 \\ 10 & 10 & 6 \\ 6 & 6 & 4 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ b \end{pmatrix} = \begin{pmatrix} 50 \\ 50 \\ 30 \end{pmatrix}$$

$$\begin{pmatrix} w_0 \\ w_1 \\ b \end{pmatrix} = \underbrace{\begin{pmatrix} 0.25 & 0.25 & -0.75 \\ 0.25 & 0.25 & -0.75 \\ -0.75 & -0.75 & 2.5 \end{pmatrix}}_{\substack{\uparrow \\ (keisuu)}} \underbrace{\begin{pmatrix} 50 \\ 50 \\ 30 \end{pmatrix}}_{\substack{\uparrow \\ (ans)}}$$

In [127]:

```
np.dot(Ainv, ans)
```

Out[127]:

```
array([2.50000000000000, 2.50000000000000, 0], dtype=object)
```

keisuuとansの内積をとればそれぞれ w_0、w_1、b に対応する値を取得できる

In [28]:

```
#微分された後の式を使っての重回帰分析
def make_model(X_train, y):
    X = np.array(X_train)
    y = np.reshape(y, (1,len(y)))
    y = np.dot(np.dot(np.linalg.inv(np.dot(X.T,X)), X.T),y.T)
    return np.reshape(y,(len(y),1))

def use_model(X_test, w):
    return np.dot(w.T, X_test)
```

In [29]:

```
def Multiple_regression(y, X):
    if type(X) == list:
        X = np.array(X)
    y = np.reshape(y, (1,len(y)))

    roop = len(X[0]) + 1
    pra1_X = []#<---------------------------------------------------------1ベクトルにより次元を上げたデータセットを格納
    poly_list = []#<----------------------------------------------------
    keisuu = []
    w = np.array([])
    ans = np.array([])

    for i in range(len(X)):#<--------------------------------------------切片の計算するように次元を1つ上げる
        pra1_X.append(np.append(X[i],1))
    pra1_X = np.array(pra1_X)

    for i in range(roop):#<--------------------------------説明変数に対応する重みベクトル作成
        obj = sym.Symbol('a' + str(i))#<--------------------------------各重みを文字にする
        w = np.append(w,obj)#<--------------------------------------------文字にした重みを重みベクトルに追加
    w = np.reshape(w,(1,len(w)))

    fa = np.dot(pra1_X, w.T)#<---------------------------------説明変数と重みの内積

    Sq_er = (y.T - fa)#<-----------------------------------------------誤差

    E = np.dot(Sq_er.T, Sq_er)#<--------------------------------二乗和誤差の計算

    dE = np.array(sym.diff(E, w))#<-----------------------------------------2乗和誤差を各定数で偏微分

    for i in range(roop):
        poly = sym.poly(dE[i])
        poly_list.append(poly.coeffs())#<---------------------------------偏微分した行列の係数をpoly_listに保存

    for i in range(roop):
        keisuu.append(poly_list[i][:-1])#<--------------------------------文字付きの係数をkeisuuに保存
        ans = np.append(ans, poly_list[i][-1:])#<-------------------------偏微分したそれぞれの切片をansの保存

    ans = -ans#<-----------------------------------------------------------------切片を右辺に移項

    keisuu = np.array(keisuu).astype(np.float64)#<---------------------
    Ainv = np.linalg.pinv(keisuu)#<------------------------------------------------------keisuuの逆行列を作成

    return np.dot(Ainv, ans)#<-----------------------------------------------A の逆行列と b の内積

def moke_b(x):
    return np.append(x, 1)

def use_model(X, w):
    return np.dot(X,w)
```

In [ ]:

```

```