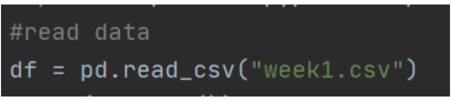
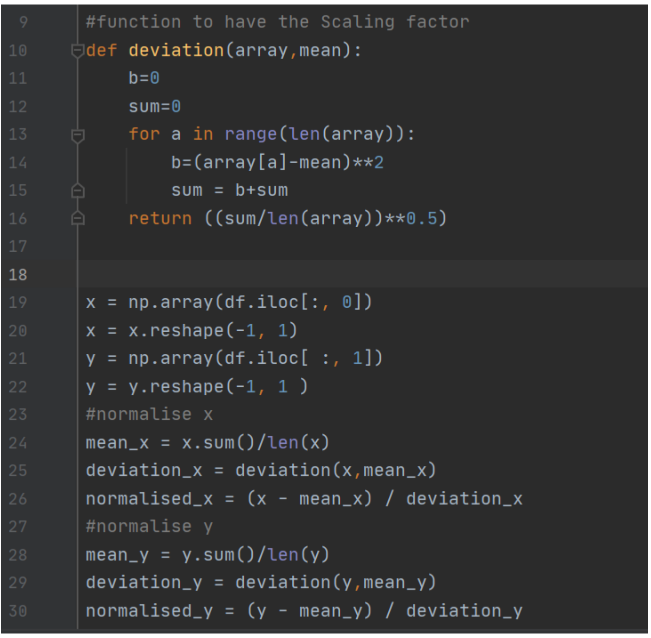
(a)

i. Read data



```
#read data
df = pd.read_csv("week1.csv")
```



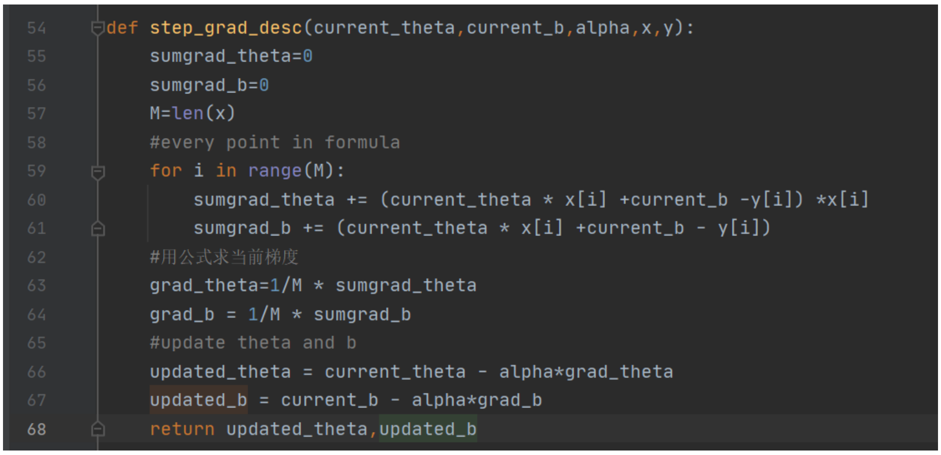
```
#function to have the Scaling factor
def deviation(array,mean):
    b=0
    sum=0
    for a in range(len(array)):
        b=(array[a]-mean)**2
        sum = b+sum
    return ((sum/len(array))**0.5)


x = np.array(df.iloc[:, 0])
x = x.reshape(-1, 1)
y = np.array(df.iloc[ :, 1])
y = y.reshape(-1, 1 )
#normalise x
mean_x = x.sum()/len(x)
deviation_x = deviation(x,mean_x)
normalised_x = (x - mean_x) / deviation_x
#normalise y
mean_y = y.sum()/len(y)
deviation_y = deviation(y,mean_y)
normalised_y = (y - mean_y) / deviation_y
```

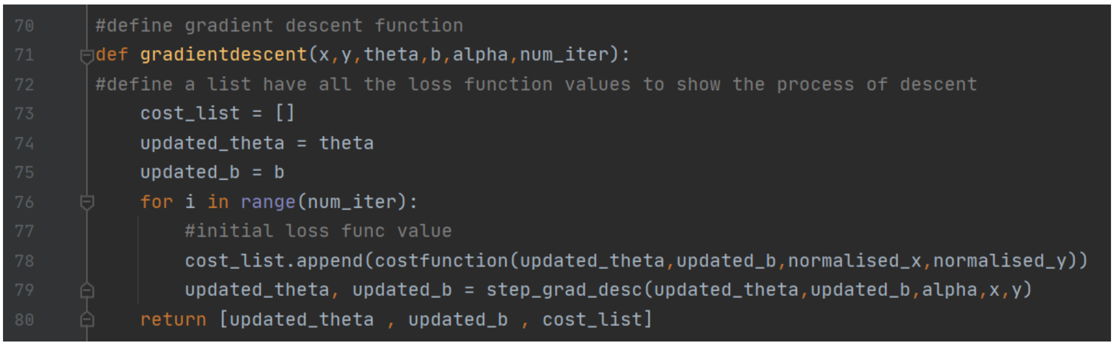
ii. Normalize the data

First I define a function to calculate the deviation, and then I use (x-mean_x)/deviation_x to replace x input. (the same to y)

iii. Gradient descent(2 functions).I define step_grad_desc(…) to update parameters for each step, then in function gradiantdescent(…) I apply step_grad_desc(…) for each iterations.



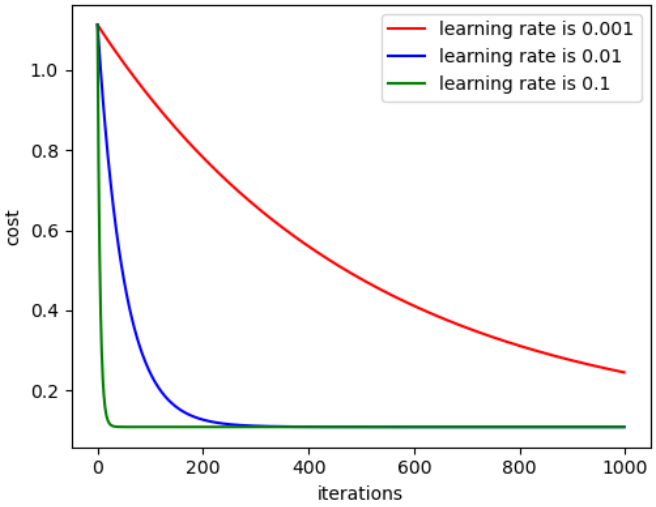
```
def step_grad_desc(current_theta,current_b,alpha,x,y):
    sumgrad_theta=0
    sumgrad_b=0
    M=len(x)
    #every point in formula
    for i in range(M):
        sumgrad_theta += (current_theta * x[i] +current_b -y[i]) *x[i]
        sumgrad_b += (current_theta * x[i] +current_b - y[i])
    #用公式求当前梯度
    grad_theta=1/M * sumgrad_theta
    grad_b = 1/M * sumgrad_b
    #update theta and b
    updated_theta = current_theta - alpha*grad_theta
    updated_b = current_b - alpha*grad_b
    return updated_theta,updated_b
```



```
#define gradient descent function
def gradientdescent(x,y,theta,b,alpha,num_iter):
#define a list have all the loss function values to show the process of descent
    cost_list = []
    updated_theta = theta
    updated_b = b
    for i in range(num_iter):
        #initial loss func value
        cost_list.append(costfunction(updated_theta,updated_b,normalised_x,normalised_y))
        updated_theta, updated_b = step_grad_desc(updated_theta,updated_b,alpha,x,y)
    return [updated_theta , updated_b , cost_list]
```

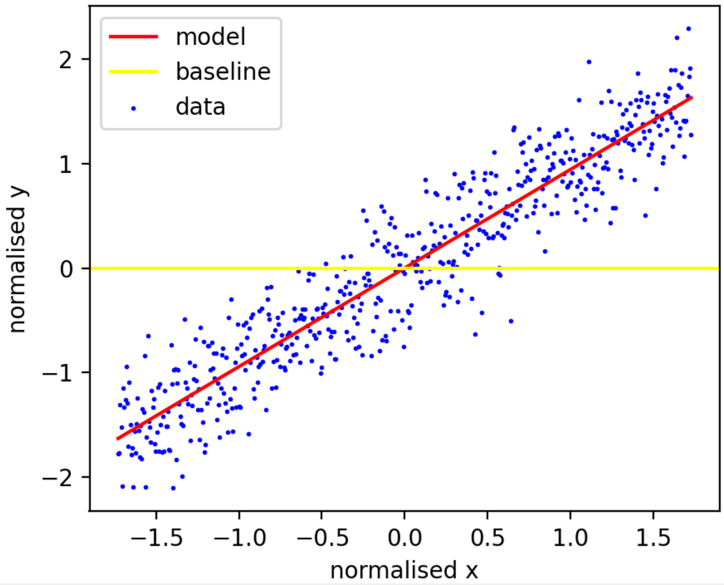
(b)

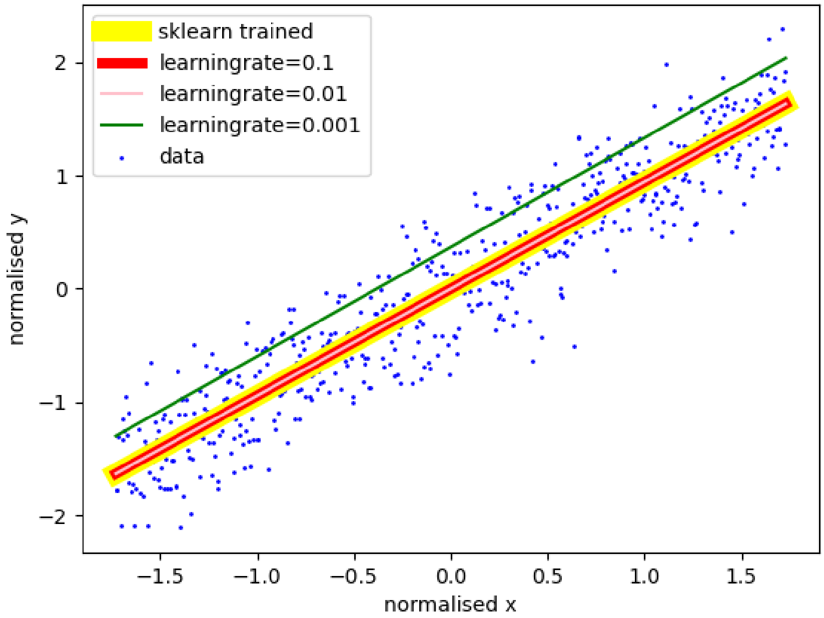
i. Learning rate varies, and the 0.001 of learning rate is too small which seems hard to converge.



ii. The liner regression model is y=theta*x + b , and the final result is :
theta = 0.9644183
b = 0.36806349

iii. The cost is: 1.1126116296011173. The baseline is the average of the normalized y, I find the data is evenly distributed.





iv.

As is shown in the figure, The learning rate of 0.001 is too small and the green model is not good , the sk-learn trained is the best and together with the 0.1 and 0.01 of the learning rate. (I changed the width of the line so as to see clearly).