log(T[n]) = log(a[0]) + log(2)\*n => linear... you can map logarithmic to linear

https"/github.com/lazyprogrammer/machine\_learning\_examples ... linear\_regression\_class/moore.csv grab this file as a good data set to import and work on

y = mx + b is a line

V = RI + 0 : voltage "V" = resistance "R" \* current "I"

rise over run : delta(y) / delta(x) = R = slope = V/I

All data is the same???

list of 2d data points {(x1,y1), (x2,y2)... (xn,yn)}

yHat is the prediction and it equals our line = ax + b, to minimize error measure the vertical distance between each prediction data point (yHat of x[i]) and the actual data x[i] we want all of those y[i] distances to be as small as possible.

you can't simple sum all of the differences because some of the distance will be below the line and negative, so could either take the abs or the square. \*\*\*\* I'm still curious what would happen if you measured to the nearest point on the prediction line rather than just measuring vertical error, need to find a way to try thid \*\*\*\*

the standard solution to this problem is the sum of squared errors E = from i = 1 to i = n sum up (y[i] - yHat[i])\*\*2 hmm i think i can code that

def sumofSquaredErrors:

E = 0

for i in range(1,len(data)):

E += (y[i] - yHat[i])\*\*2

return E

in english for every data point in your list/array of data subtract your predicted value from the actual value square that result and add it to your total sum.

next we want to find some line that makes E as small as possible, if we had an immaginary situation where E = 0.5t\*\*2 - t then we could take the derivative of E with respect to t and set that to 0 to find the minimum for that function t = 1

but we are not sure what our function is but in our above sumation we know that:

E = sumofSquaredErrors(data, yHat)

y[i] and x [i] are given becuase they are our data points

yhat = (ax + b) of our prediction

def sumofSquaredErrors(data, yHat):

E = 0

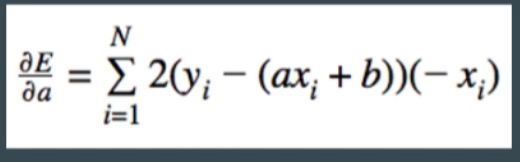
for i in range(1,len(data)):

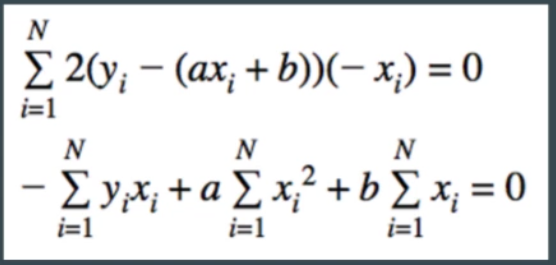
E += (data[i] - yHat[i])\*\*2

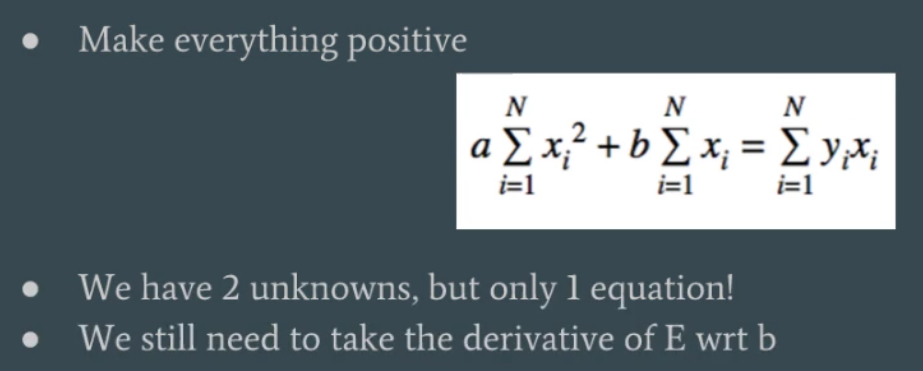
return E

we are trying to minimize E with respect to a and b (slope and intercept of our prediction line) so we need partial derivatives

first step is to take the derivative of E with respect to a (I don't actually know how to do that :P) here are the slides from the instructor:

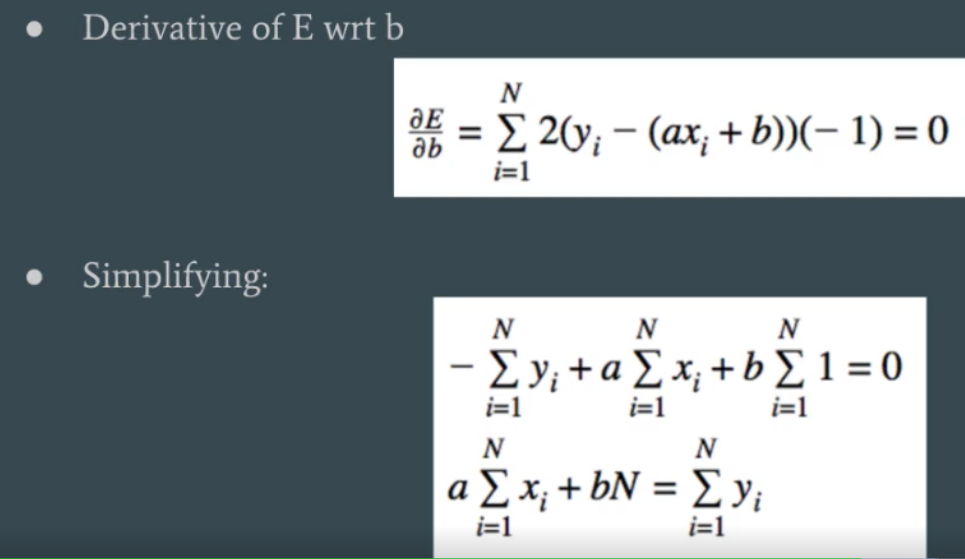


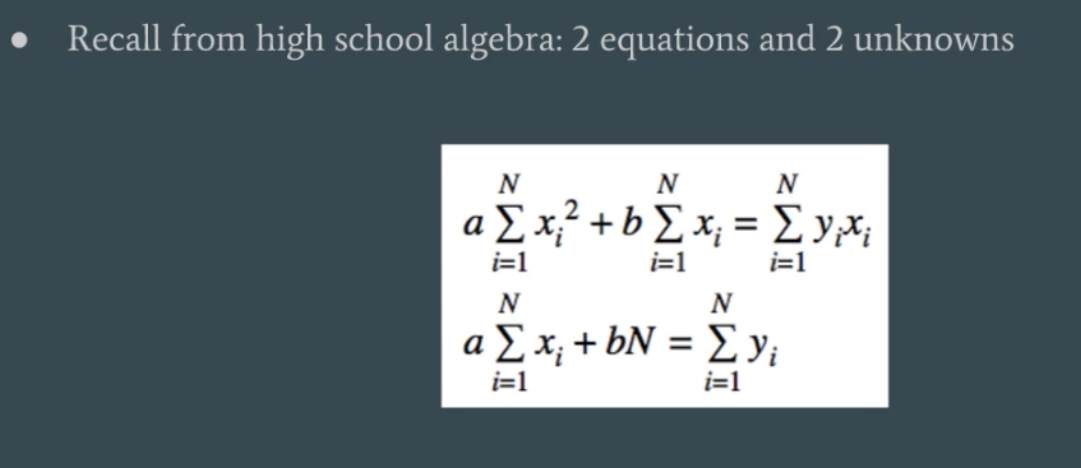


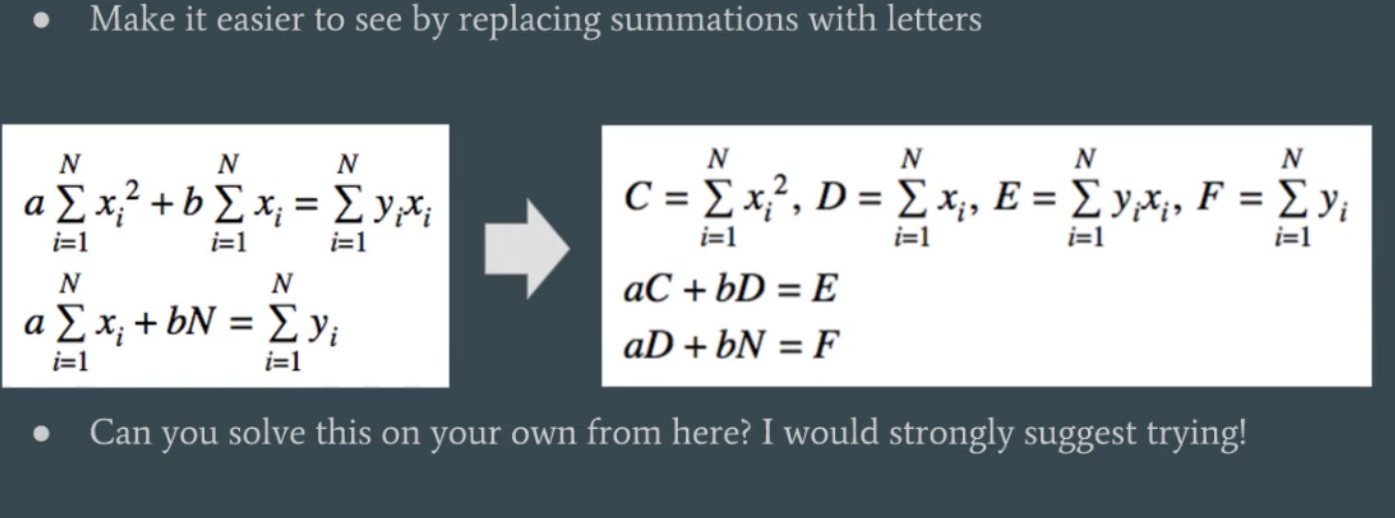


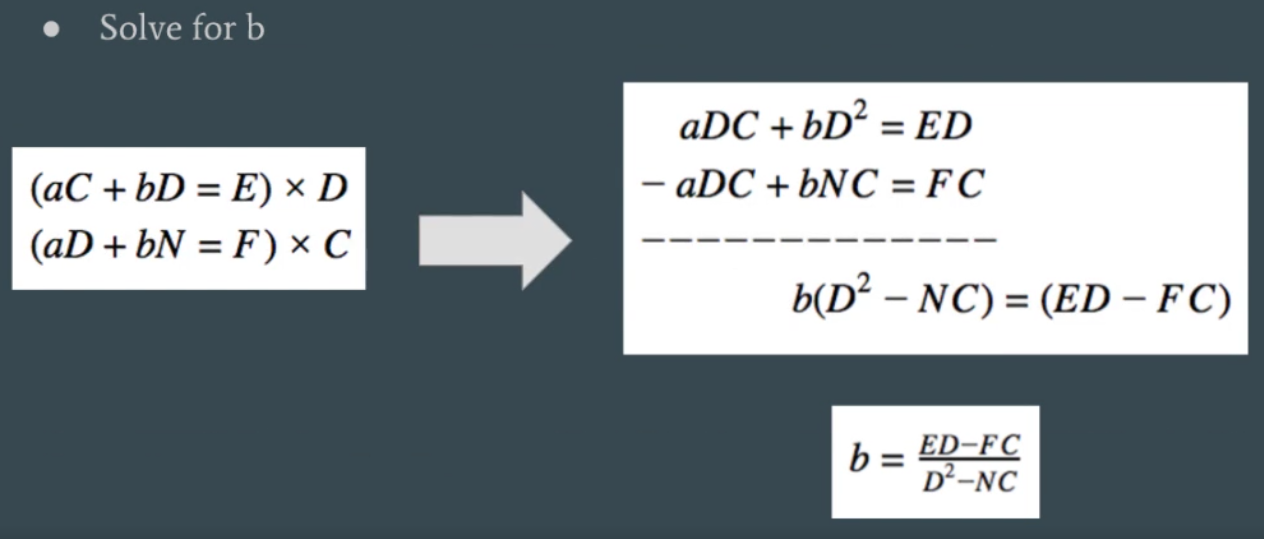
N is the number of data points in our set, x[i]and y[i] are just the pairs of data points so the right hand side is the sum of all of the data points, but we don’t know a or b

So all of that got us the derivative with respect to a now we do the same but with respect to b





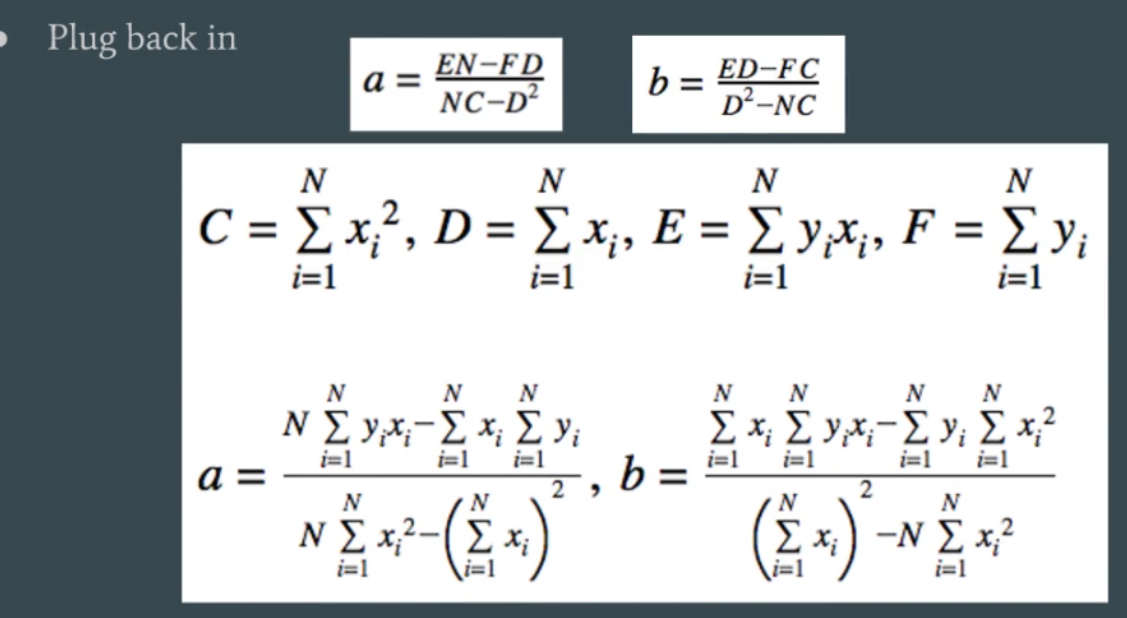




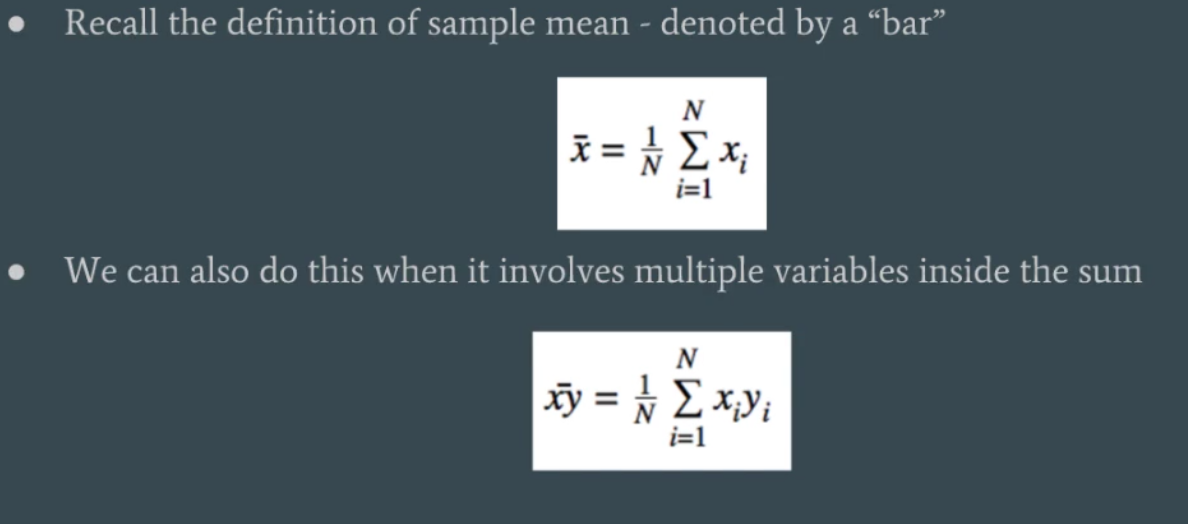
Now solve for a



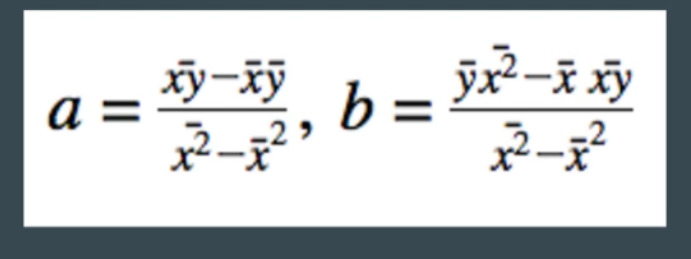
Plug back in a and b



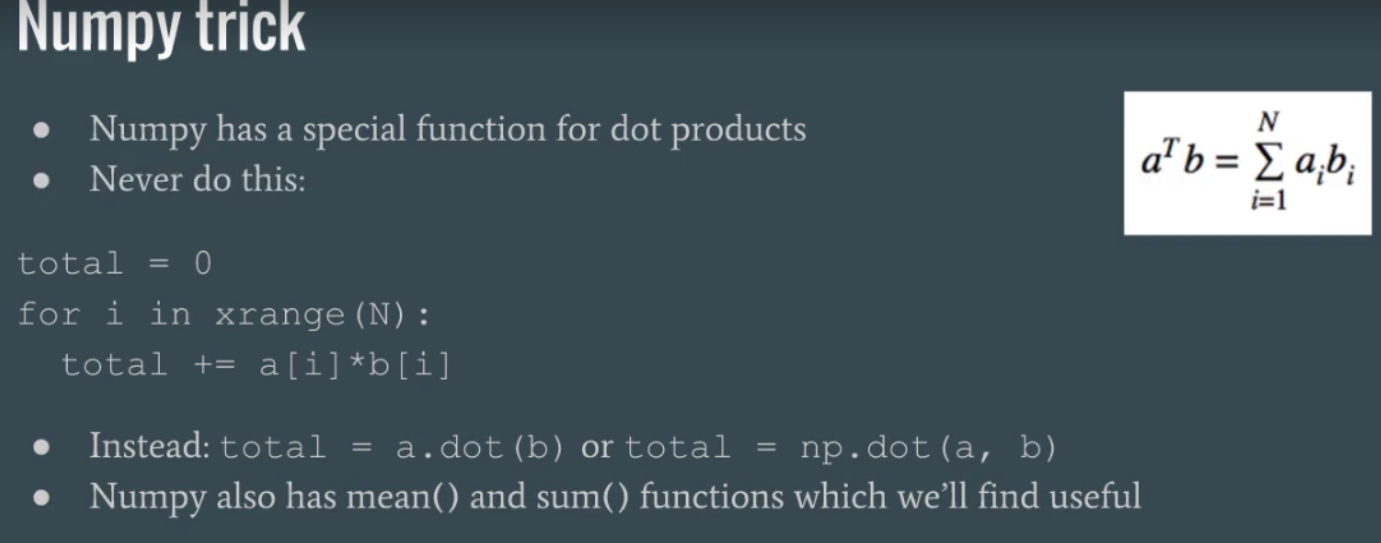
Pretty sure some magic math is about to happen:



So that’s just adding everything up and divding by the number of elements or finding the mean aka average of the entire sample



I’m honestly not sure why he substituted in the sample mean in place of the sums from above…



I don’t really know what a dot product is… but rather than doing a for loop do go through the entire set and do an operation on each element numpy is allowing me to do some quick math on the whole data set by treating it as a matrix

Why cant you take the derivative of |x| abs(x) ???

What is the chain rule when taking partial derivatives???