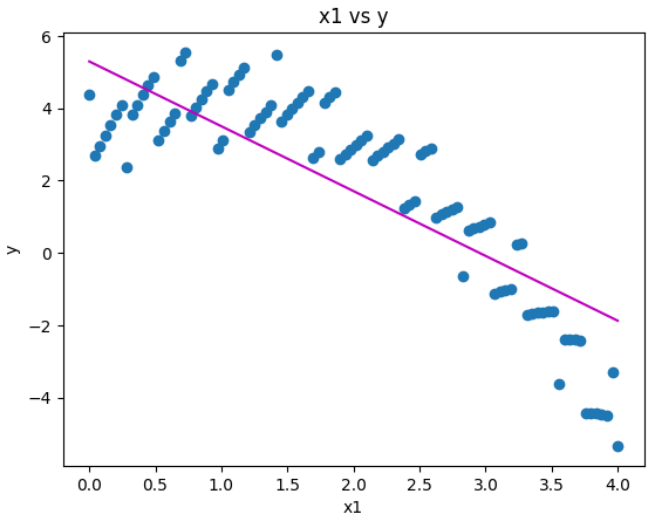
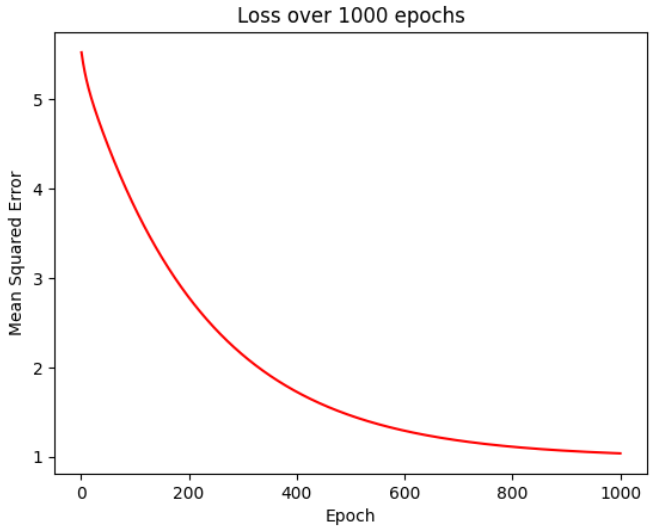
Question 1:

The learning rates overall did not affect the final loss and training iterations all that much. I started out with a learning rate of 0.005, which I later realized was causing the program to run for far too many iterations. But even this slow learning rate provided a pretty good starting point for optimizing the algorithm.

y = -1.791190790033247x + 5.2929472827573





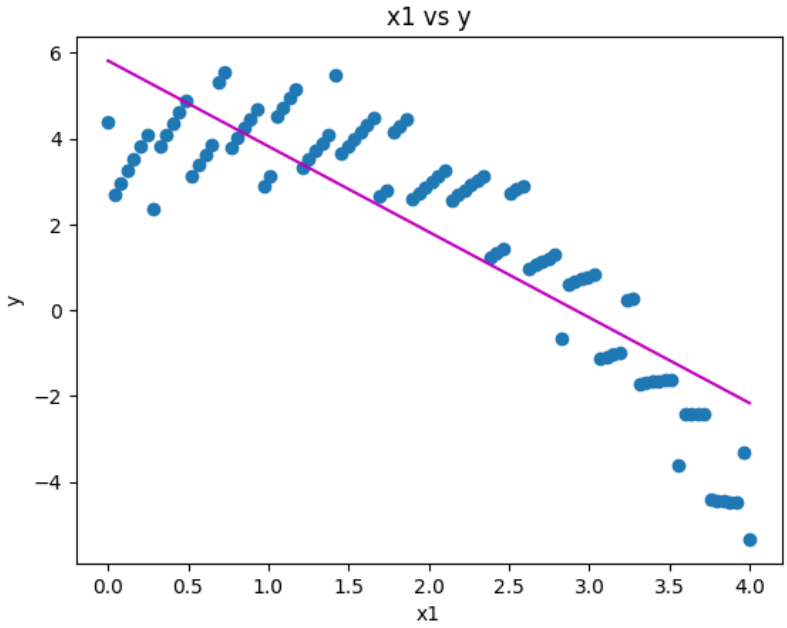
Final loss: 1.0366648249472639

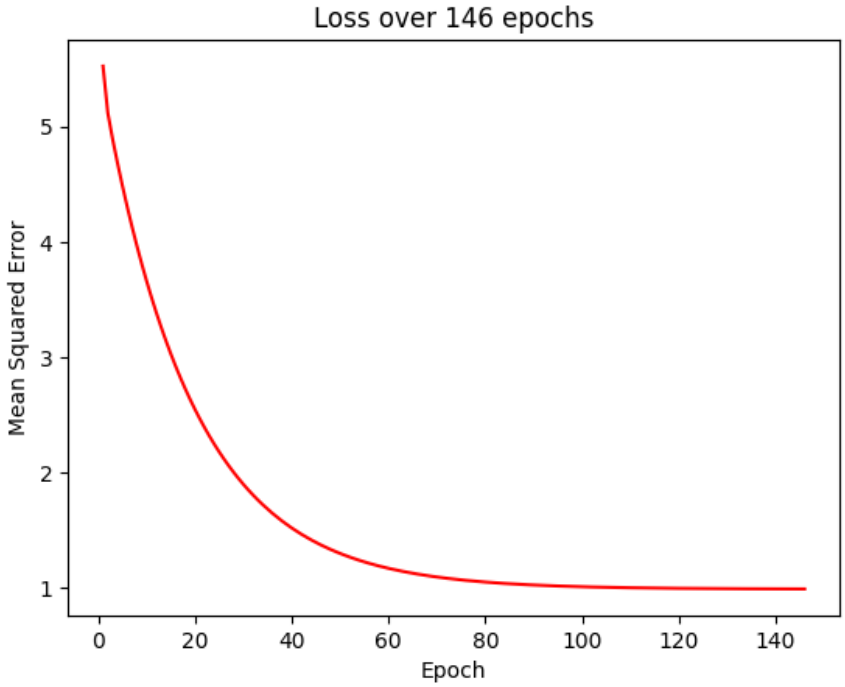
The only issue with this slower learning rate is that the algorithm runs for the full 1000 epochs, and the loss convergence check never gets evaluated. It's clear that we must increase the learning rate so that the algorithm reaches the local minimum of the loss function in fewer

iterations. With a tweaked learning rate of 0.06 and loss threshold of 1e-3, the algorithm is more accurate, and takes fewer iterations to find the most optimal parameters.

A short note about loss thresholds: When the difference between the previous total and the next total loss (between epochs) falls below a certain threshold (1e-3 in this case), it doesn't make sense to keep running iterations for such little gain, so we cut off the algorithm.

y = -1.9939236630603008x + 5.813836908161361

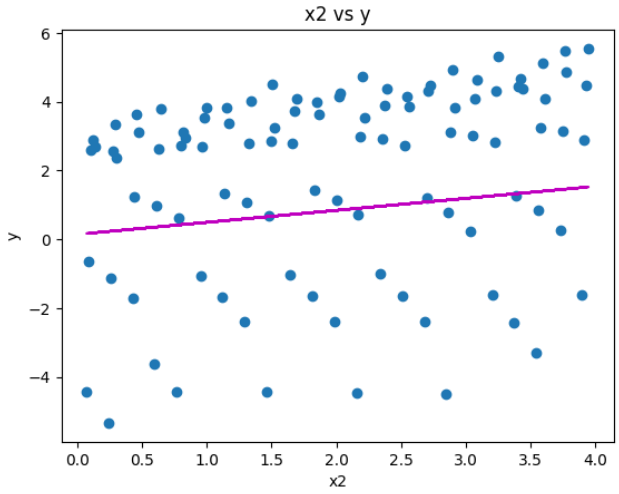


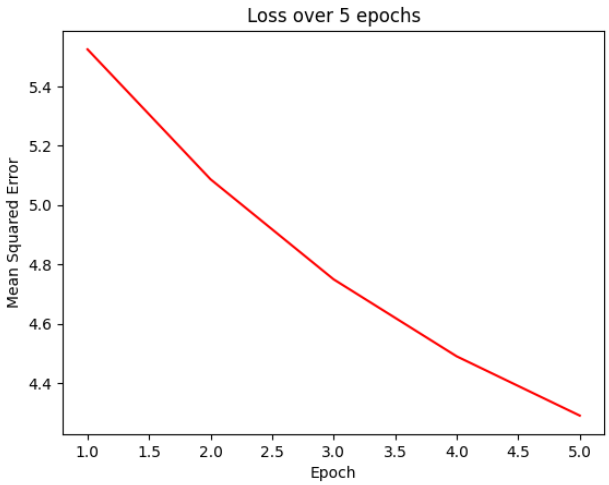


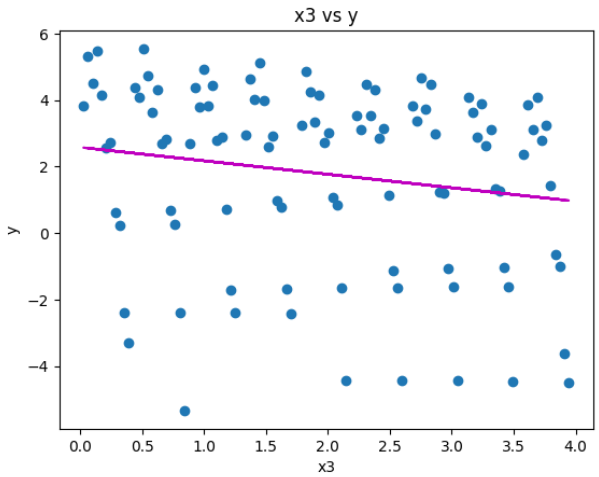
final loss: 0.9867463508530391

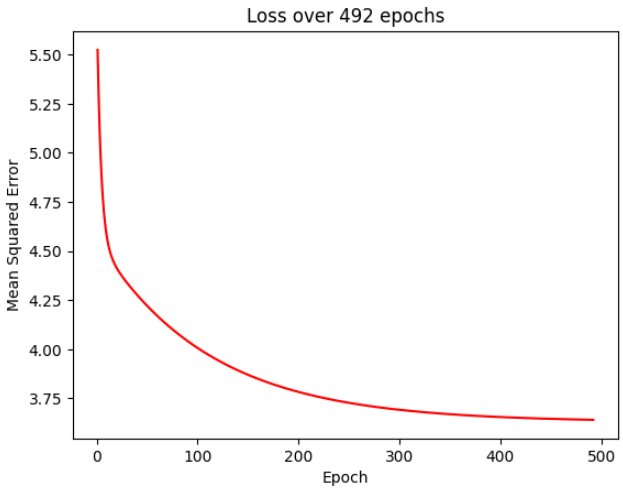
	Something else	Bias	Final loss
X1	-1.9939	5.8138	0.9867
X2	0.3484	0.1470	4.2904
X3	-0.4059	2.5816	3.6405

The regressions for x2 and x3 are shown below. The data doesn't fit the linear model, and the model struggles. X1 has the lowest cost of prediction out of these three individual variables.









Question 2:

Now instead of predicting one variable at a time, we can stack all three independent variables on top of each other and use 4 theta variables instead of 2 (x1, x2, x3, and the bias value). Here is the result.

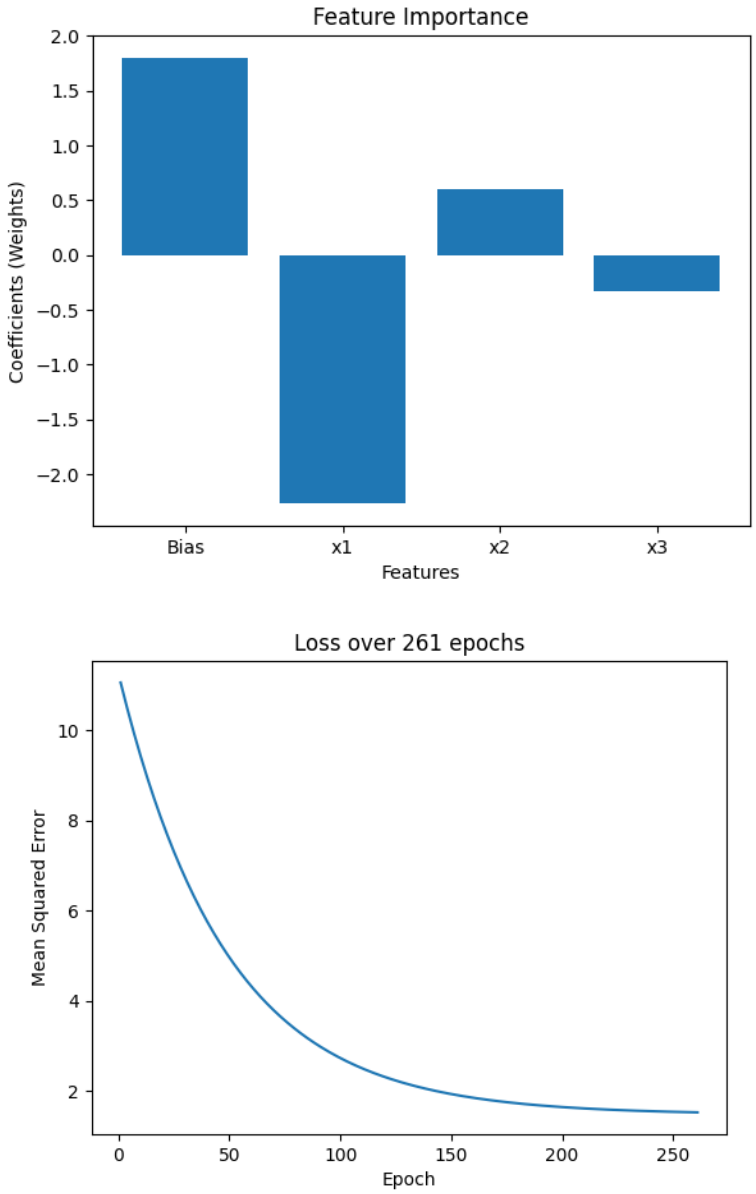


Figure 1: learning rate of 0.01

Final loss: 1.5256

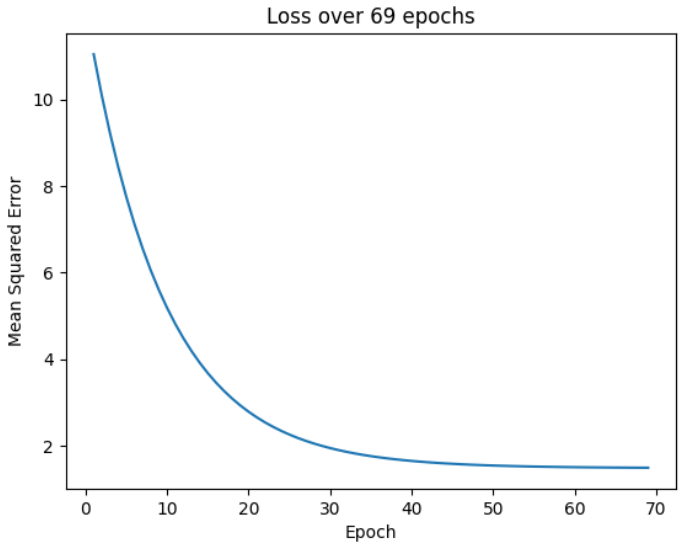


Figure 2: learning rate of 0.05

Final loss: 1.4851

It's important to note that the learning rate is very important for the efficiency of the algorithm. In this case, we reduced the amount of epochs by almost 200 and got a lower final cost. If you make the learning rate too high, the model will overshoot the local minimum, and if you set it too low, the model will run for too many iterations before reaching the desired value.

Predicting values for the inputs given is a trivial task. An equation can be created using the theta values:

y = -2.16650110608838x1 + 0.5788299745411857x2 + -0.340980849126567x2 + 1.7169197085196473

The final predicted values are:

(1,1,1): 0.805

(2,0,4): -3.054

(3,2,1): -3.124

Final thoughts:

In all these problems, we initialized the theta values to 0. In most cases, this is not the most optimal strategy. We can either make visual predictions of what the first theta value might be, or we can just choose random values for theta. In either case, the model would take less iterations and might end with a lower final cost.

All code is at https://github.com/Anu78/intro-to-ml-hw/tree/main/hw1