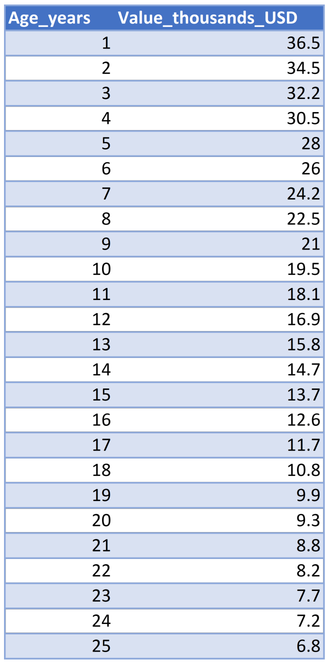
MLDS 400 Lab 200

October 2025

1 Objective

In today's lab you will read in data from "car_value_vs_age.csv" and use a basic optimization algorithm to train a linear regression model based on the car value over time data. The point of this assignment is not to learn about linear regression or machine learning. Rather, this simple example will serve as a toy problem to help you learn how to set up a virtual environment and workflow in Github.

Your first objective is to download the "linear_regression.py" file from Canvas and complete the code to perform linear regression on the attached .csv file. To create a linear model predicting car value as a function of age, you will start with the data from the .csv file:



Age_years	Value_thousands_USD
1	36.5
2	34.5
3	32.2
4	30.5
5	28
6	26
7	24.2
8	22.5
9	21
10	19.5
11	18.1
12	16.9
13	15.8
14	14.7
15	13.7
16	12.6
17	11.7
18	10.8
19	9.9
20	9.3
21	8.8
22	8.2
23	7.7
24	7.2
25	6.8

Each row of the table above will serve as one of the ordered pairs in your training set, with car age serving as the predictor and car value as the response.

2 Conceptual Background on Gradient Descent

The goal here is to construct a linear regression model of the form

$$\hat{y} = \alpha x + \beta \tag{1}$$

where x represents an input predictor and α and β are model parameters to be determined by our best linear fit. The "best" linear fit is defined as the pair of α and β which minimizes the sum of the squared differences between the response values in our training data above and those responses $\hat{y}$ predicted by our linear model. In other words, we wish to minimize the loss function

$$\mathcal{L}(\alpha, \beta) = \frac{1}{2}\sum_{i=1}^{25}(\hat{y}_i - y_i)^2 \tag{2}$$

Plugging in Eq. (1), this becomes

$$\mathcal{L}(\alpha, \beta) = \frac{1}{2}\sum_{i=1}^{25}(\alpha x_i + \beta - y_i)^2 \tag{3}$$

Since we have all our ordered pairs (x_i, y_i) in the table of training data above, our task is to find the best values of α and β which minimize $\mathcal{L}(\alpha, \beta)$.

While analytic solutions to this basic problem exist, we will use gradient descent to solve this problem in Python. To begin, start with an initial guess vector $v_0 \in \mathbb{R}^2$ for our parameters α and β:

$$v_0 = \begin{pmatrix} \alpha_0 \\ \beta_0 \end{pmatrix} \tag{4}$$

Plugging these initial guess for α and β into $\mathcal{L}(\alpha, \beta)$ will give a (most likely crappy) initial value for our loss function. That's okay! We can brush off our multivariable calculus and calculate the *gradient* of $\mathcal{L}(\alpha, \beta)$:

$$\nabla\mathcal{L}(\alpha, \beta) = \begin{pmatrix} \dfrac{\partial \mathcal{L}}{\partial \alpha} \\ \\ \dfrac{\partial \mathcal{L}}{\partial \beta} \end{pmatrix} \tag{5}$$

Calculating our partial derivatives, we have

$$\frac{\partial \mathcal{L}}{\partial \alpha} = \sum_{i=1}^{25} x_i(\alpha x_i + \beta - y_i) \tag{6}$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = \sum_{i=1}^{25} (\alpha x_i + \beta - y_i) \tag{7}$$

For appropriately chosen scalar η (also known as the learning rate), an iteration of gradient descent gives us a new, better fit for α and β, which we call v_1:

$$v_1 = v_0 - \eta \nabla \mathcal{L}(v_0) \tag{8}$$

Repeating this process will give us an even better guess for α and β, and so on. So we need to rinse and repeat

$$v_{n+1} = v_n - \eta \nabla \mathcal{L}(v_n) \tag{9}$$

until our iterates converge to give values for α and β which minimize our loss function. We will know our iterates have converged once we're not changing much between iterations, or mathematically when the magnitude of $||\nabla \mathcal{L}(\alpha, \beta)||$ drops below a certain tolerance ϵ. This will give our optimal values

$$v^* = \begin{pmatrix} \alpha^* \\ \beta^* \end{pmatrix} \tag{10}$$

which, when plugged into our model, give the equation for our line of best fit:

$$\hat{y} = \alpha^* x + \beta^* \tag{11}$$

3 Today's goal

With the provided data and conceptual background above, your goal for today is to find the optimal parameters α^* and β^* for fitting the given data with a line of best fit. To ensure you've accomplished the goal with your own code, you will also complete the code in the "lin_reg_test.py" to fit the same data with a linear regression using the np.polyfit function. If your parameters match numpy's to a specified precision, then the test should return a success. Otherwise, the test should flag that you've made a mistake somewhere.

While this is a trivial example, in the real world Github proves a great asset for organizing, sharing, and continually testing our code in response to both our own code revisions as well as package version updates. You should create a Github repo complete with a workflow file, README, and requirements file to which you can upload your completed "linear_regression.py" and "lin_reg_test.py" and have all of your code tested successfully by your Github workflow.

Hints: ask for help and play around with different values of η if youra algorithm isnt' converging.