Machine Learning | Week 1 Assignment
17324263 | Stephen Byrne

a (i)
To read in the data I used the provided code given *(Appendix a.provided)*. It separates the given CSV data into separate NumPy arrays

a (ii)
To normalise the data between 0 and 1 I wrote a function (*Appendix a.0*) that uses the formula:

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)}$$

It takes in a NumPy array and returns the normalized array. It gets the max and min value of the input array and to normalize each element it subtracts the min value from the element and divides it by the max value taken away from the min value. This function is run on both the X and y input data to give two new normalized arrays.

a (iii)
Linear regression with gradient descent based on the formula below, each individual function I built is explained below and linked in the appendix.

$$\begin{aligned}&\text{repeat until convergence } \{\\ &\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)\\ &\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}\\ &\}\end{aligned}$$

The hypothesis estimates Y (outputs) based on X (inputs), as this is a linear model it resembles the equation of a line (y=mx+c) in the form $= {}_0 + {}_1$ (h0 = theta0 + theta1x - formula wouldn't display on pdf sometimes so listing here also). This hypothesis function (*Appendix a.1)* takes in the theta parameters and x value to map and returns the estimated y value, we use this throughout the algorithm. My linear regression function *(Appendix a.2)* initializes the theta (parameter) values at random before starting the algorithm. The function takes the learning rate (a) and the number of iterations (epochs) along with X and y arrays as parameters. Each iteration the minimize function *(Appendix a.3)* runs and works to minimize the MSE by calculating the partial derivatives and subtracting them multiplied by our learning rate (a) to update the parameter value (ie - this is the gradient descent).

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)$$

Gradient Descent

(taken from medium article - *Appendix c.1*)

The derivatives function *(Appendix a.4)* is where the cost function minimization occurs, as shown below.

$$\theta_0\ j=0: \underline{\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)})-y^{(i)}\right)$$

$$\theta_1\ j=1: \underline{\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1)} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)})-y^{(i)}\right)\cdot x^{(i)}$$

Image from Andrew Ng's machine learning course on Coursera.com

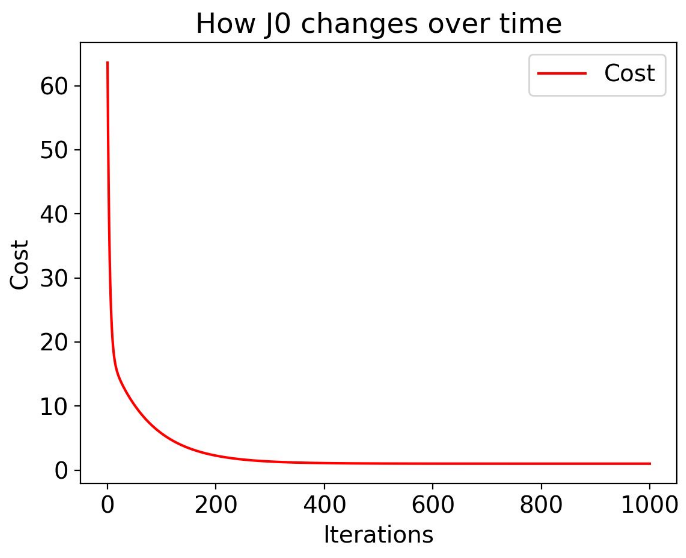
(taken from medium article - *Appendix c.1*)

Upon completion of the training, I then returned the parameter values from the model to evaluate for the assignment.

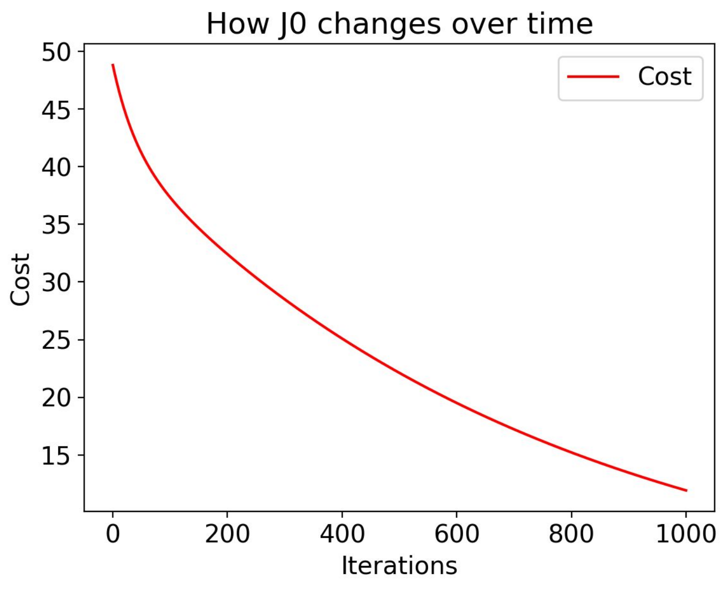
b (i)

Explanation of 3 graphs below: As shown below in all three graphs, the cost (MSE) is represented on the y-axis and the iteration on the x-axis. The graphs, therefore, depict how the cost function changes over time. *(Appendix b.2)* The graph with learning rate 0.1 converges on the minimum quite quickly while 0.01 takes nearly all the iterations and 0.001 does not converge at all. No extremely large learning rates are used so there is no overshooting situation.

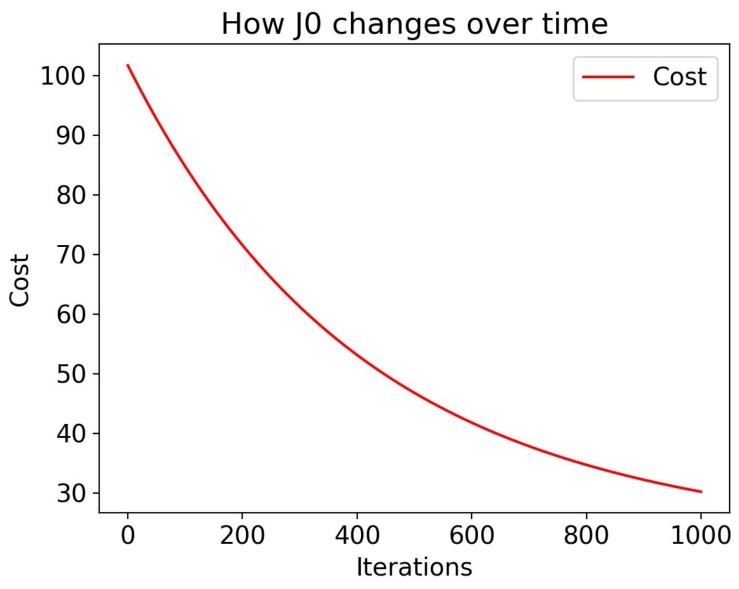
Learning rate = 0.1



Learning rate = 0.01



Learning rate = 0.001



To get the cost array to plot I modified my linear regression function slightly *(Appendix b.1)* to return the cost for each iteration in an array instead of parameter values at the end.

b (ii) when a=0.1 & 1000 epochs

$_o = \ 0.8579$

$_1 = \ -0.7091$

Note - theta values would not display on my pdf sometimes, listing them below in case they don't.

Theta0 = 0.8579

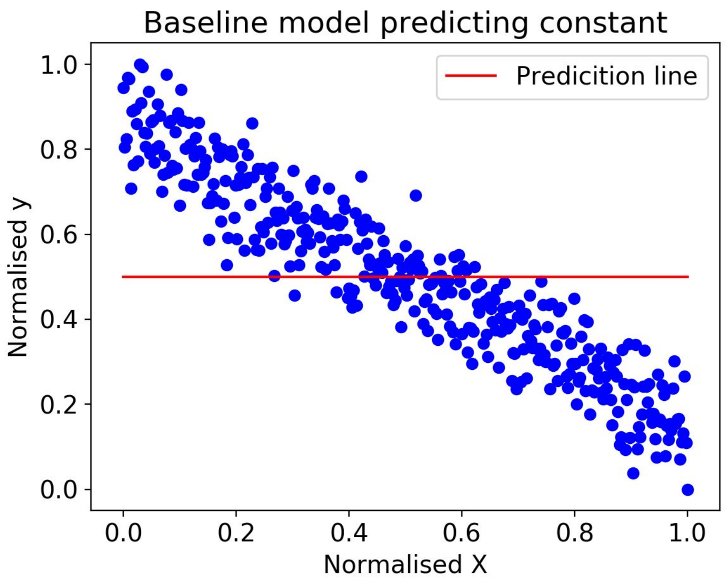
Theta1 = -0.7091

b (iii) when a=0.1 & 1000 epochs

Value of cost function at completion = 1.0131

<u>Comparison to baseline predicting constant</u>

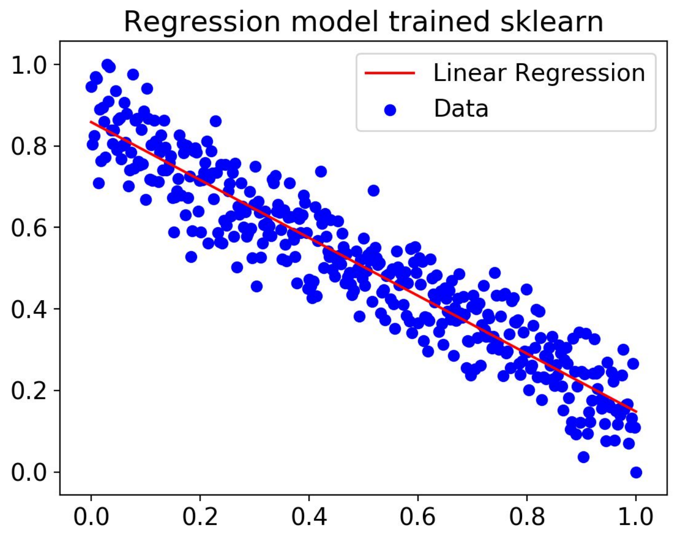
When picking a baseline model to compare to that always selects a constant, I drew a line roughly in the middle of the graph on the normalized data at 0.5. For every X input, the predicted value is 0.5. This can be seen below in the graph labeled baseline model predicting a constant with the normalized X values on the x-axis and normalized y values on the y-axis

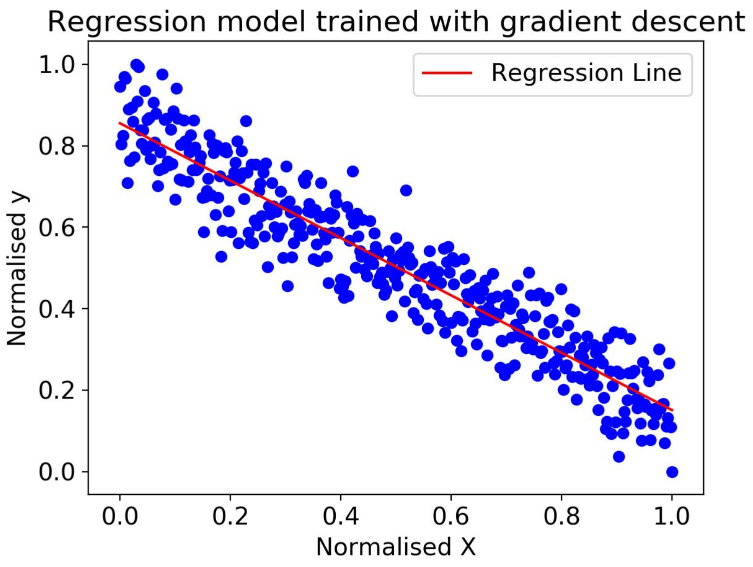


The cost value of the baseline model: 36.1559.

As we can see 35.1559 is substantially more costly than our linear regression model which had a cost value of 1.0131. The linear regression model therefore significantly outperforms the baseline model,

B (iv)

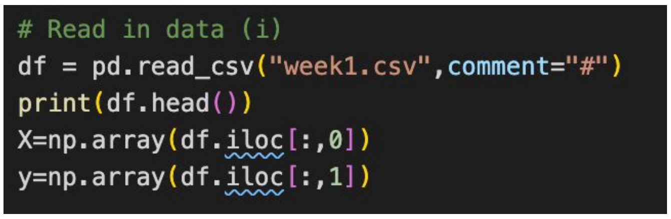




As seen in the above graphs, the model trained using vanilla python and gradient descent performs extremely similarly to Sklearn, assuming that the learning rate and epochs are chosen optimally (a=0.1 and epochs=1000). Code in *(Appendix b.4)*

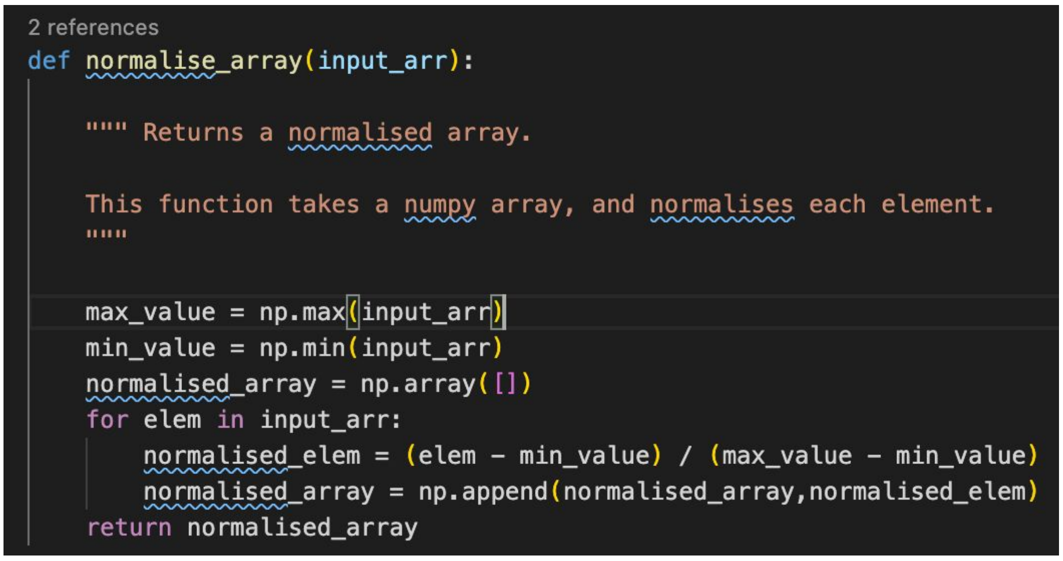
Appendix

a.provided



```python
# Read in data (i)
df = pd.read_csv("week1.csv",comment="#")
print(df.head())
X=np.array(df.iloc[:,0])
y=np.array(df.iloc[:,1])
```

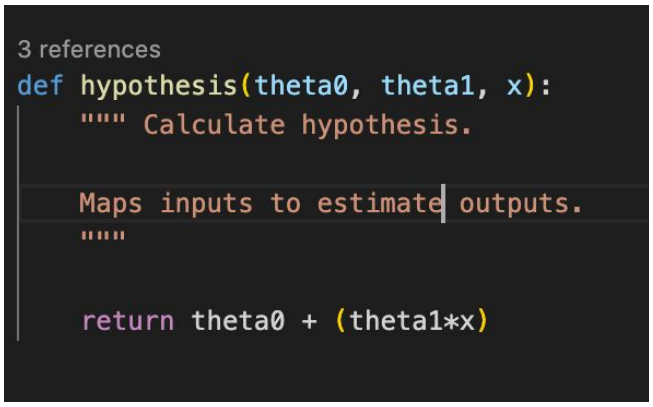
a.0



2 references

```python
def normalise_array(input_arr):

    """ Returns a normalised array.

    This function takes a numpy array, and normalises each element.
    """

    max_value = np.max(input_arr)
    min_value = np.min(input_arr)
    normalised_array = np.array([])
    for elem in input_arr:
        normalised_elem = (elem - min_value) / (max_value - min_value)
        normalised_array = np.append(normalised_array,normalised_elem)
    return normalised_array
```

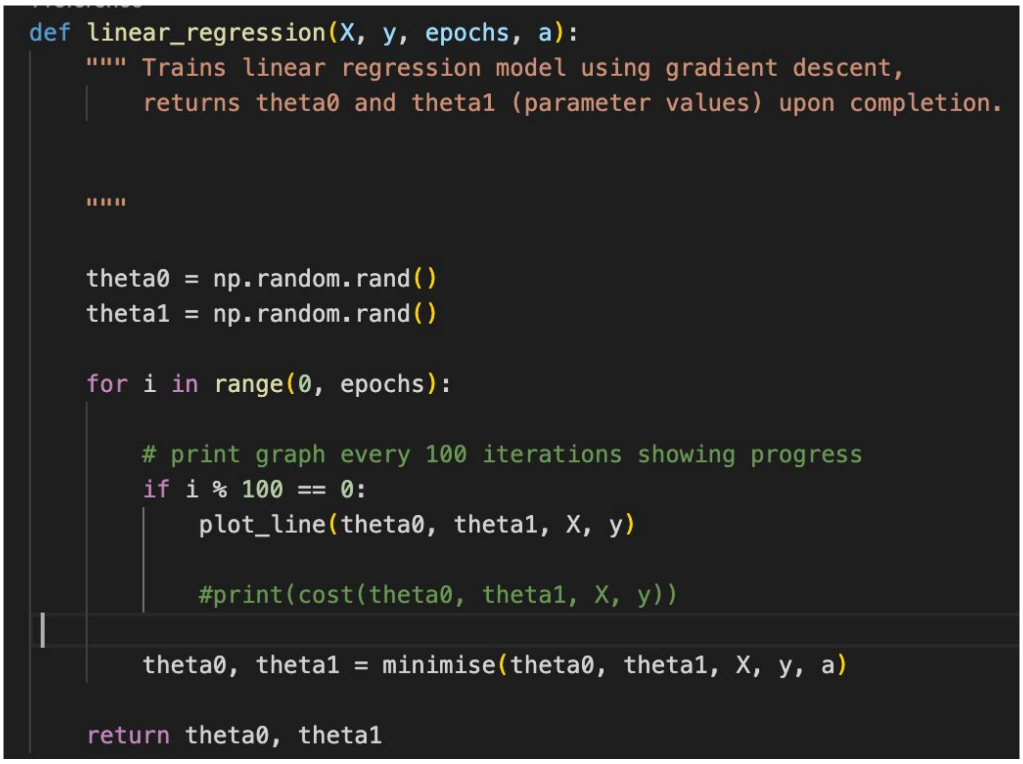
a.1



3 references

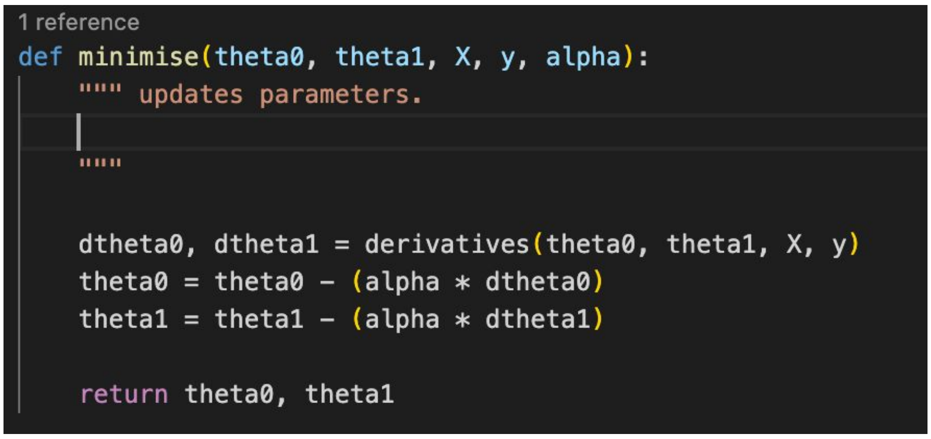
```python
def hypothesis(theta0, theta1, x):
    """ Calculate hypothesis.

    Maps inputs to estimate outputs.
    """

    return theta0 + (theta1*x)
```

a.2



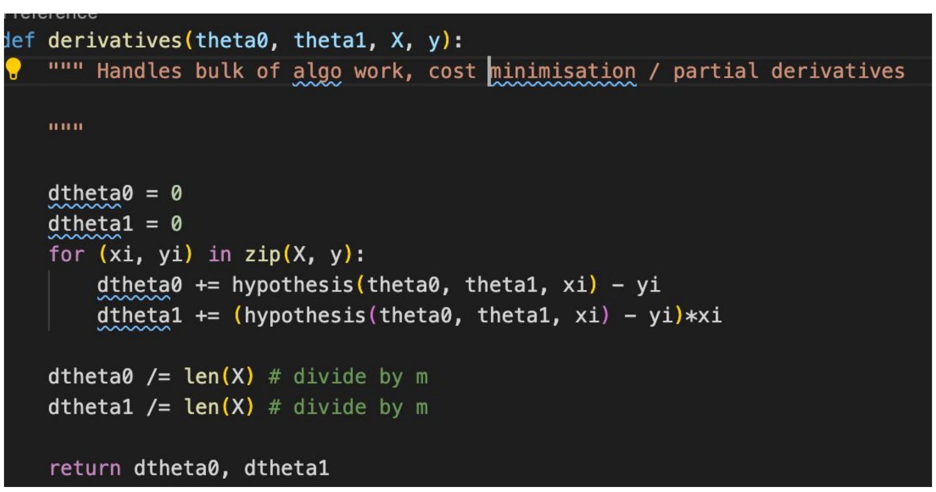
```
def linear_regression(X, y, epochs, a):
    """ Trains linear regression model using gradient descent,
        returns theta0 and theta1 (parameter values) upon completion.


    """

    theta0 = np.random.rand()
    theta1 = np.random.rand()

    for i in range(0, epochs):

        # print graph every 100 iterations showing progress
        if i % 100 == 0:
            plot_line(theta0, theta1, X, y)

            #print(cost(theta0, theta1, X, y))

        theta0, theta1 = minimise(theta0, theta1, X, y, a)

    return theta0, theta1
```

a.3



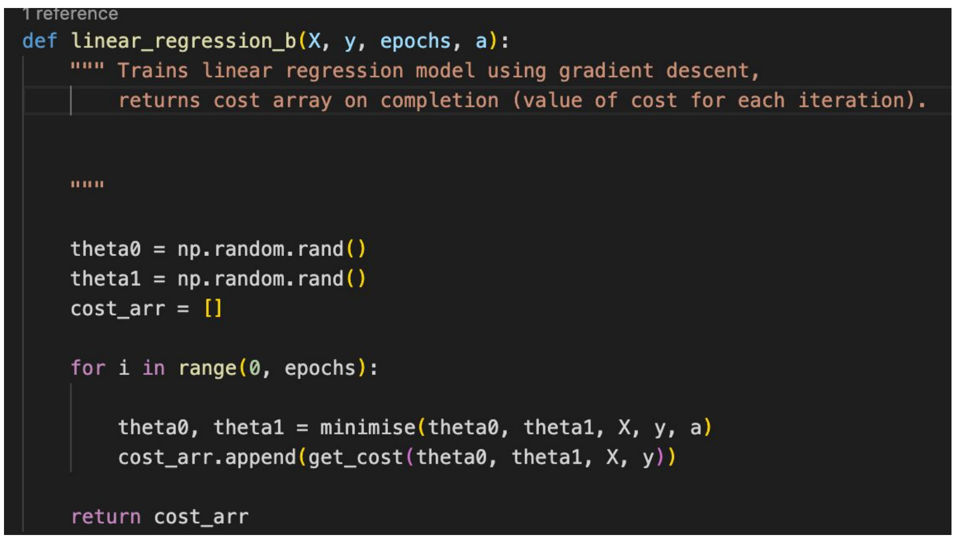
```
1 reference
def minimise(theta0, theta1, X, y, alpha):
    """ updates parameters.

    """

    dtheta0, dtheta1 = derivatives(theta0, theta1, X, y)
    theta0 = theta0 - (alpha * dtheta0)
    theta1 = theta1 - (alpha * dtheta1)

    return theta0, theta1
```

a.4



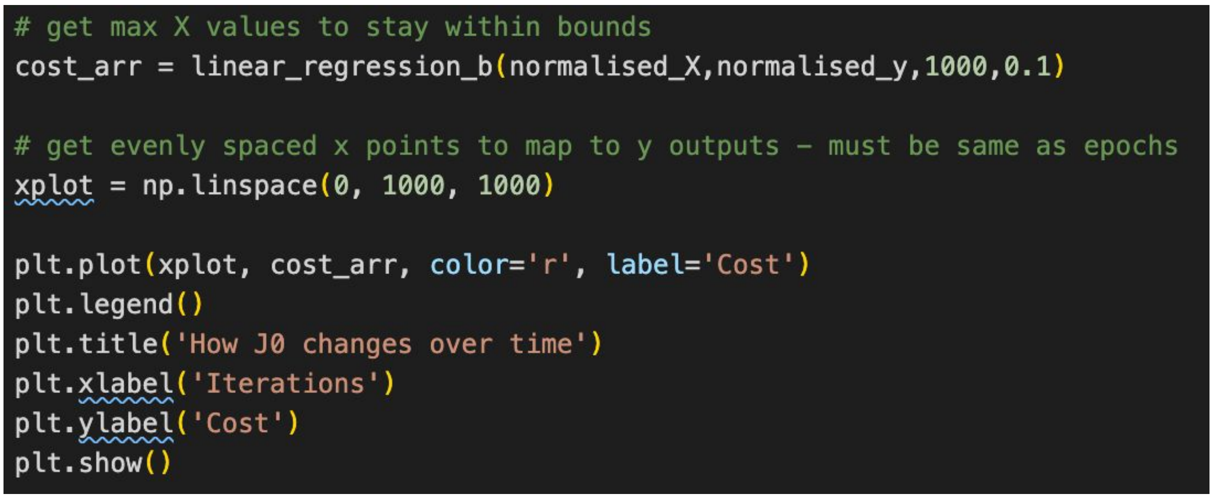
```
def derivatives(theta0, theta1, X, y):
    """ Handles bulk of algo work, cost minimisation / partial derivatives

    """

    dtheta0 = 0
    dtheta1 = 0
    for (xi, yi) in zip(X, y):
        dtheta0 += hypothesis(theta0, theta1, xi) - yi
        dtheta1 += (hypothesis(theta0, theta1, xi) - yi)*xi

    dtheta0 /= len(X) # divide by m
    dtheta1 /= len(X) # divide by m

    return dtheta0, dtheta1
```

b.1



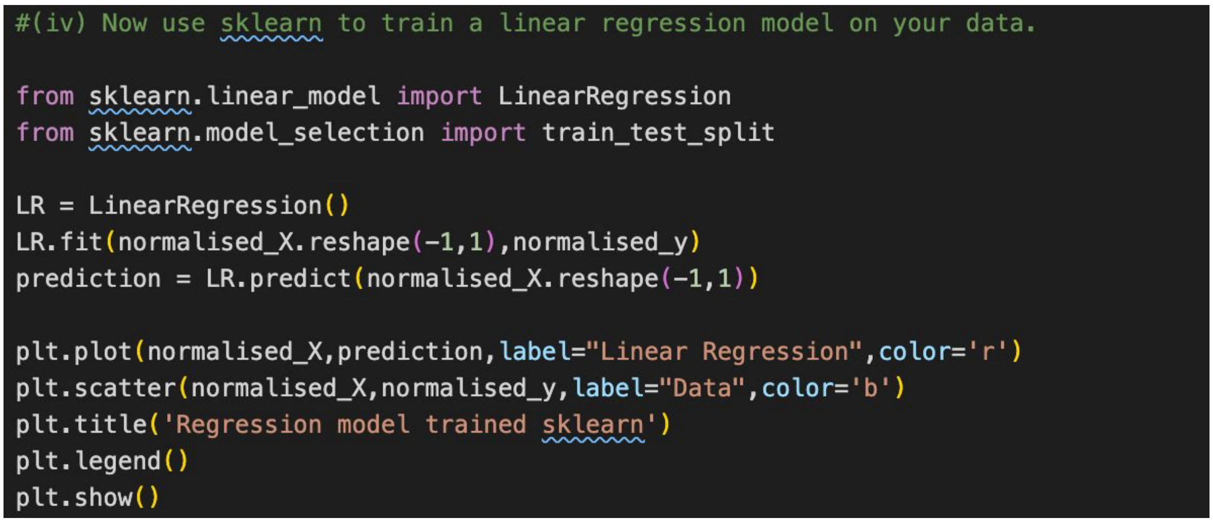
```
1 reference
def linear_regression_b(X, y, epochs, a):
    """ Trains linear regression model using gradient descent,
        returns cost array on completion (value of cost for each iteration).


    """

    theta0 = np.random.rand()
    theta1 = np.random.rand()
    cost_arr = []

    for i in range(0, epochs):

        theta0, theta1 = minimise(theta0, theta1, X, y, a)
        cost_arr.append(get_cost(theta0, theta1, X, y))

    return cost_arr
```

b.2



```
# get max X values to stay within bounds
cost_arr = linear_regression_b(normalised_X,normalised_y,1000,0.1)

# get evenly spaced x points to map to y outputs - must be same as epochs
xplot = np.linspace(0, 1000, 1000)

plt.plot(xplot, cost_arr, color='r', label='Cost')
plt.legend()
plt.title('How J0 changes over time')
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()
```

b.4



```
#(iv) Now use sklearn to train a linear regression model on your data.

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

LR = LinearRegression()
LR.fit(normalised_X.reshape(-1,1),normalised_y)
prediction = LR.predict(normalised_X.reshape(-1,1))

plt.plot(normalised_X,prediction,label="Linear Regression",color='r')
plt.scatter(normalised_X,normalised_y,label="Data",color='b')
plt.title('Regression model trained sklearn')
plt.legend()
plt.show()
```

c.1

- https://medium.com/@lachlanmiller_52885/machine-learning-week-1-cost-function-gradient-descent-and-univariate-linear-regression-8f5fe69815fd
-