Linear regression via gradient descent (solution)

Please follow these instructions carefully:

- Read all the code in the notebook.
- There are 6 problems. For each, replace the comment YOUR CODE HERE with your code.
- Do not modify the code in other ways. You may want to write additional test code while you're getting your code working. If so, mark your test code clearly and then remove it before submitting.
- Before submitting your code, select 'Restart and run all' from the Runtime menu.

```
Out[2]:
```

Click here to display/hide the code.

Partial derivatives

In multivariable calculus, you are given the definition of a function f(x,y) and told to find the partial derivative df/dx. The partial derivative is a function, and in general is a function of x and y.

For example, if f is defined by f(x,y) = 2x^2 + xy, for which df/dx is 4x + y and df/dy = x.

df/dx and df/dx are functions. If (x,y) = (1,2) then df/dx(x,y) is 6 and df/dy(x,y) is 1.

```
[1.1 2. ]
[1.  2.2]
4.0
4.0602
6.00000
1.00000
```

Gradients

The gradient of f take as input x,y and returns the vector (df/dx(x,y), df/dy(x,y)).

For example, let f be defined by f(x,y) = 2x^2 + xy, for which df/dx is 4x + y and df/dy = x.
The gradient of f, applied at input x=1,y=2 is (df/dx(1,2), df/dy(1,2)), which is (4(1) + 1, 1) = (5,1).

Problem 1

Implement the body of the gradient() function. Use function partial_deriv() in your code. The function that is returned by gradient() should itself return a NumPy array.

Hint: you may want to use a list comprehension.

```
x: [1. 2.], f_grad(x): [6. 1.]
x: [0. 0.], f_grad(x): [0. 0.]
x: [ 2. -1.], f_grad(x): [7. 2.]
```

Gradient descent

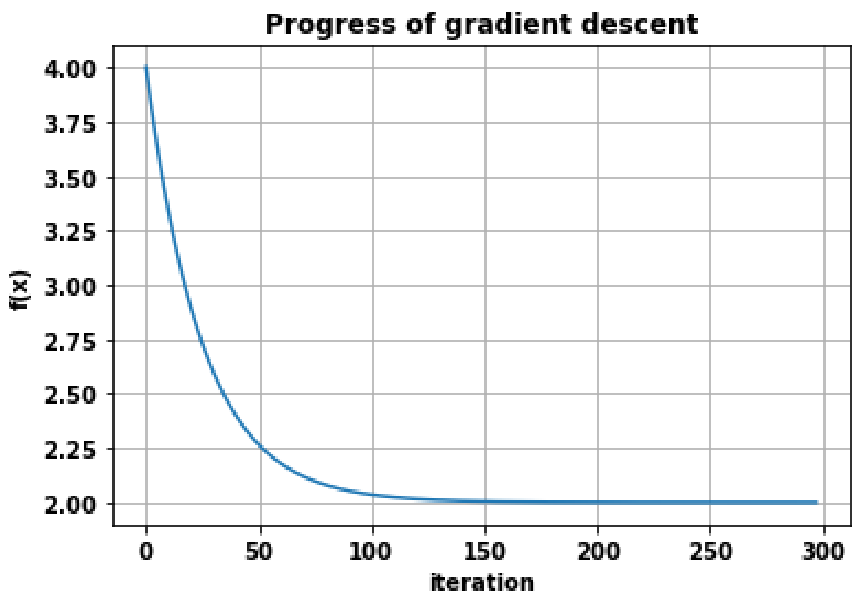
In gradient descent we want to find the value of x that minimizes (or maximizes) a function f. We do this by starting with some x, computing the value of the gradient of f at x, and then using that value to make an adjustment to x.

Problem 2

Implement the body of the grad_descent() function.

```
x = [0.99517], f(x) = 0.000
x = [-1.10297], f(x) = -0.649
x = [ 0.99757 -0.99757], f(x) = 2.000
```

Plot the value of the loss function as gradient descent proceeds.

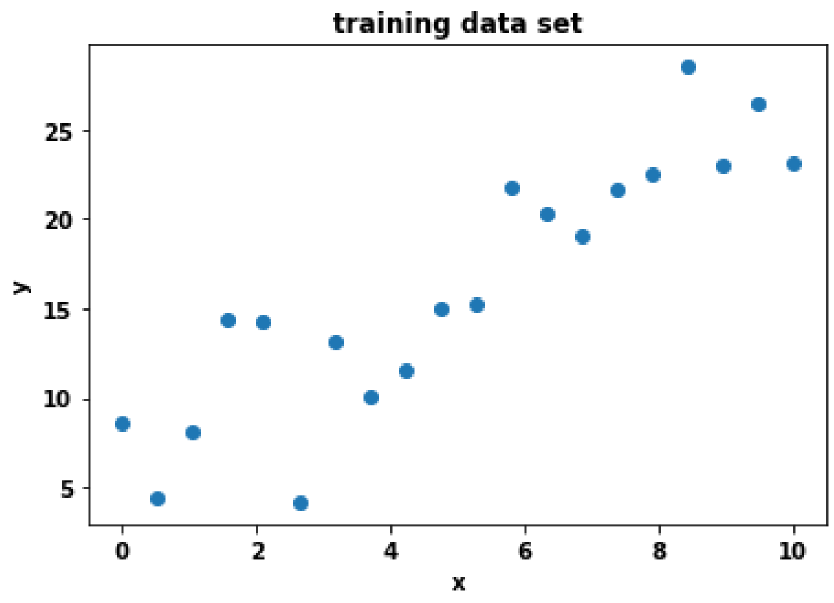


MSE loss

The key idea for training linear regression is that we want to use gradient descent to find the model parameters that minimize the loss function. For linear regression, the loss function is "mean squared error" (MSE).

Create some fake training data. Note that the coefficients are in an array, and that the data has an extra first column of ones to make computing the value of the linear model simple.

Because we generated the data ourselves, we know the true coefficients of the line.



Problem 3

Implement the body of the mse_loss() function. Note that to compute the mean squared error we need both the model parameters and the training data.

```
b: [1.5 2.5], mse_loss(b): 16.751
b: [0.5 2.5], mse_loss(b): 22.306
b: [1.5 3.5], mse_loss(b): 36.568
```

Linear regression

Now we can put together our gradient descent function and MSE loss function to perform linear regression.

The function mse_loss() takes parameters b, X1, and y. However, gradient_descent() expects to be passed a function that takes only a single parameter.

Therefore, we define a function loss() that is like mse_loss() except it only takes parameter b. The data values X1 and y are "hard wired" into loss().

Problem 3

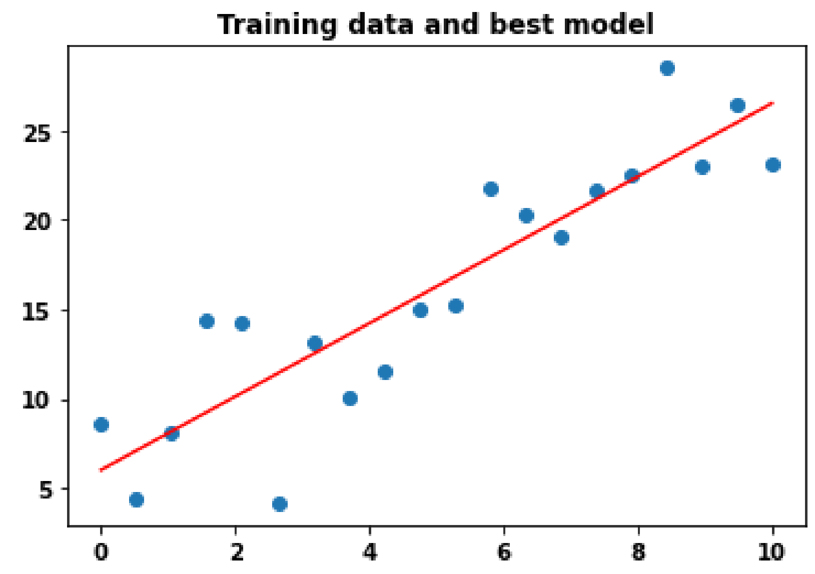
Define function loss(). Use a lambda expression and call mse_loss(). This is a very simple one-liner.

```
b: [1.5 2.5], loss(b): 16.751
b: [0.5 2.5], loss(b): 22.306
b: [1.5 3.5], loss(b): 36.568
```

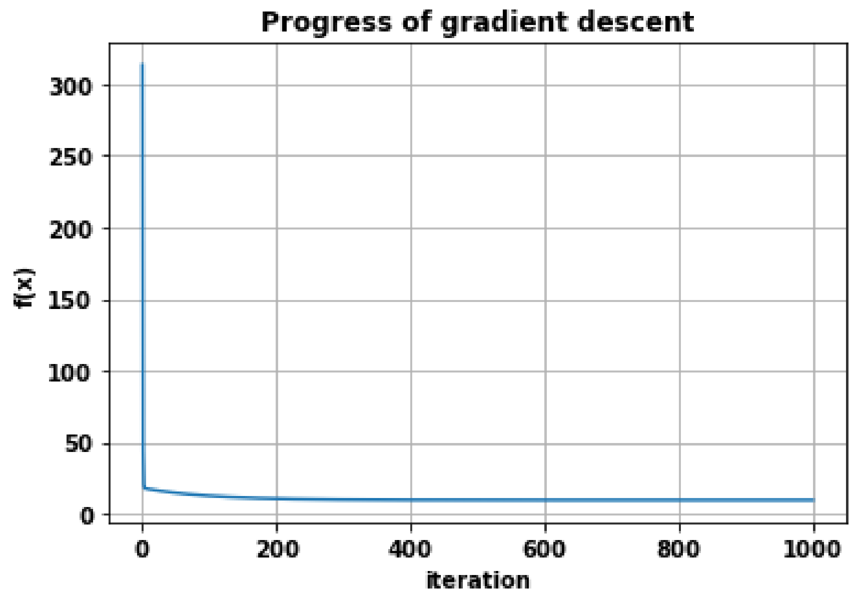
Now we perform gradient descent using the loss function.

```
warning: reached iteration limit
[6.023 2.049]
```

Plot the training data and the best-fit line.



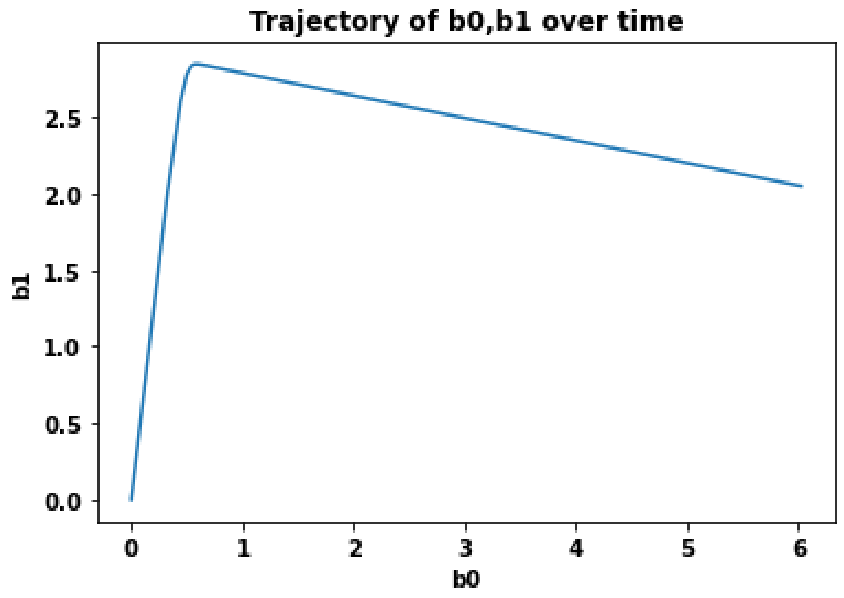
Look at how loss changed during gradient descent.



Plot the change of b0,b1 over time

Another way to visualize the progress of gradient descent is to see how the model parameters change over time. In this plot we show the value of both model parameters. In this plot you can't see the loss value. Remember that the initial value of b0, b1 is 0, 0.

Do you see a "gulley" in the plot?



Scale the x data and try again

We know that data should be scaled prior to running gradient descent. What happens to the gulley once the data is scaled?

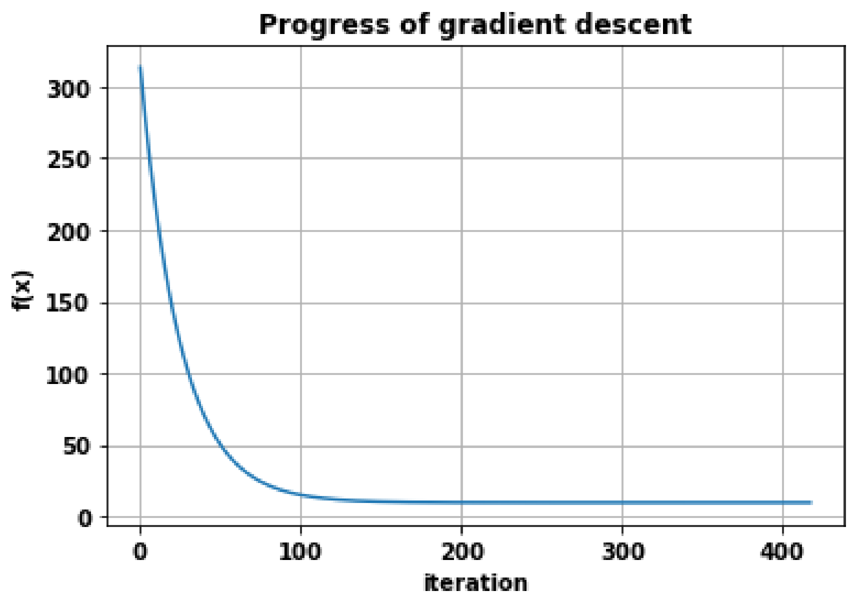
Problem 5

Define a version of the loss() function that uses the scaled data.

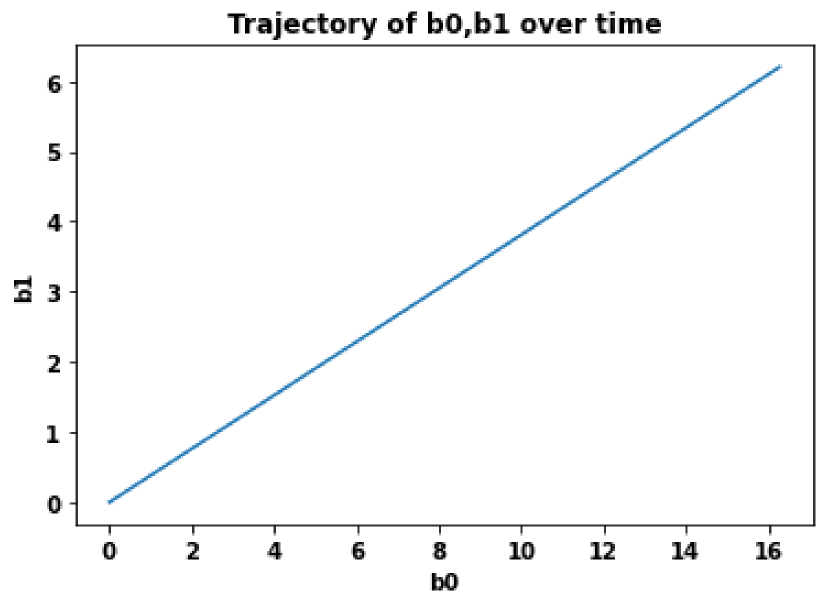
Try gradient descent again. Note the new loss function.

```
[16.27383825  6.20533778]
```

Compare this plot to the plot above with unscaled data. Notice the change in shape and change in number of iterations.



Compare this plot to the trajectory plot above. It is completely different.



Perform linear regression on a real data set

Now let's use our code to perform linear regression on a real data set. This is a small data set with only 38 rows.

We'll predict a car's MPG from data about it's weight, horsepower, etc.

Problem 6

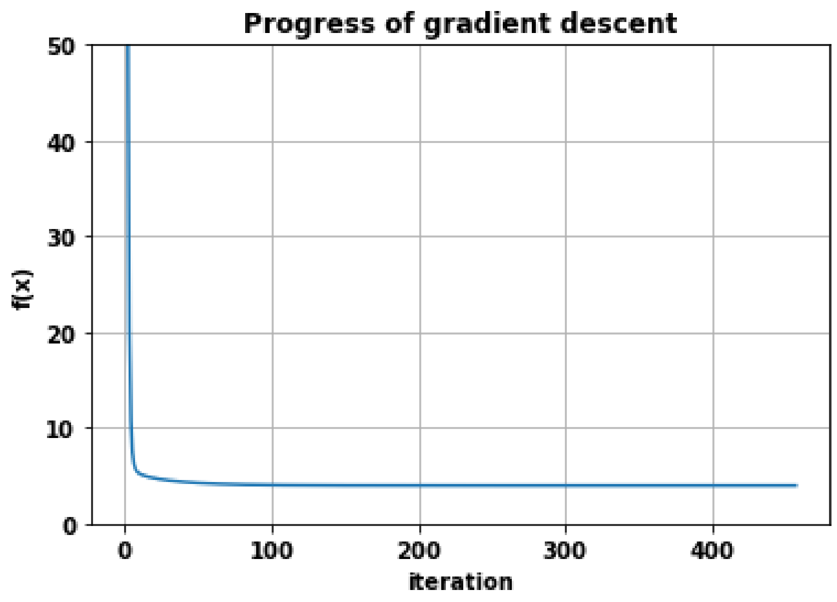
Define a version of the loss() function that uses the cars data.

Problem 7

Train a linear model for the cars data using gradient descent.

You may want to tweak the learning rate and number of iterations here.

```
intercept:24.761
other coefficients:[-7.969 -1.809 -1.267  2.158]
```



Compare the results we get with Scikit Learn's linear regression algorithm, which probably uses the "normal equation".

```
intercept:24.761
other coefficients:[-7.971 -1.809 -1.266  2.16 ]
```