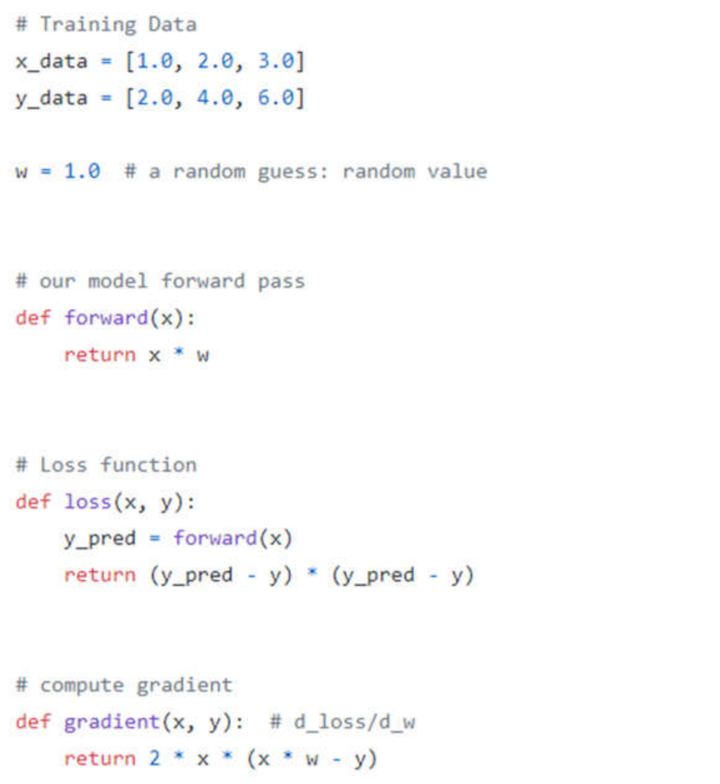
Task 1: Exploring “Linear Model” without python libraries.

Solution:



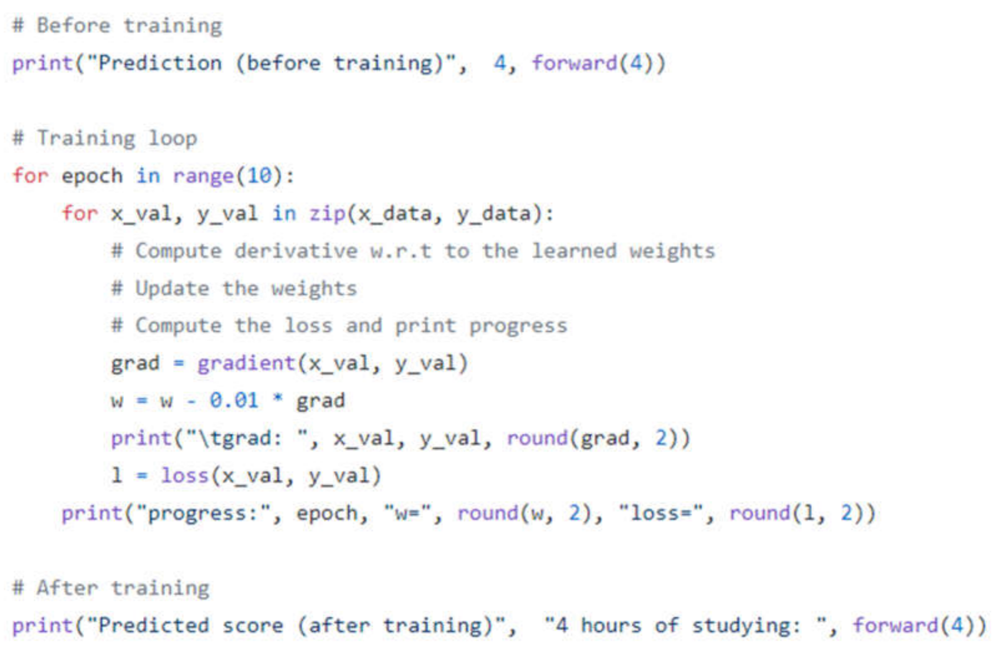
```python
# Training Data
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0  # a random guess: random value


# our model forward pass
def forward(x):
    return x * w


# Loss function
def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)


# compute gradient
def gradient(x, y):  # d_loss/d_w
    return 2 * x * (x * w - y)
```



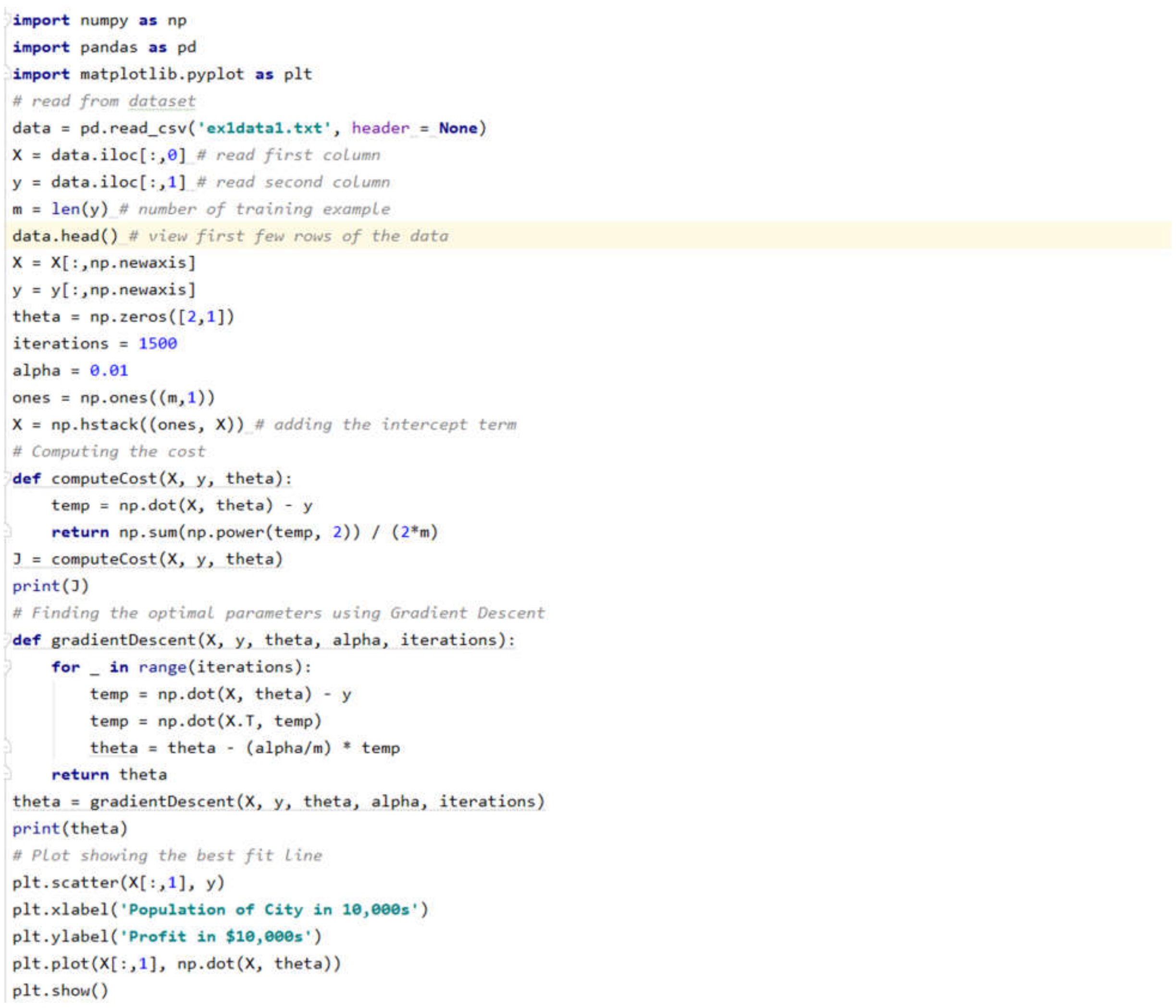
```python
# Before training
print("Prediction (before training)",  4, forward(4))

# Training loop
for epoch in range(10):
    for x_val, y_val in zip(x_data, y_data):
        # Compute derivative w.r.t to the learned weights
        # Update the weights
        # Compute the loss and print progress
        grad = gradient(x_val, y_val)
        w = w - 0.01 * grad
        print("\tgrad: ", x_val, y_val, round(grad, 2))
        l = loss(x_val, y_val)
    print("progress:", epoch, "w=", round(w, 2), "loss=", round(l, 2))

# After training
print("Predicted score (after training)",  "4 hours of studying: ", forward(4))
```

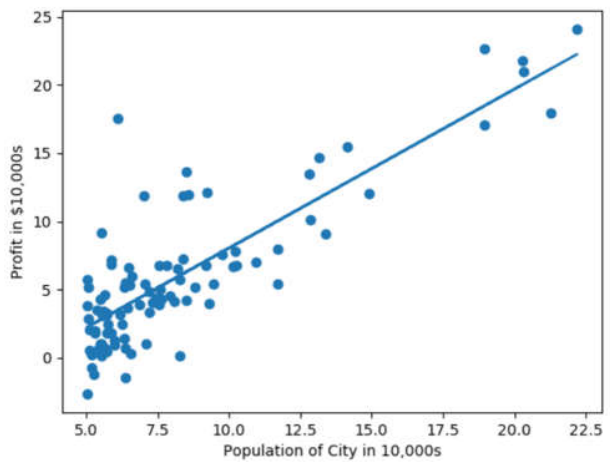
Source code credit: S. Kim

Task 2: Univariate Linear Regression with Gradient Descent



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# read from dataset
data = pd.read_csv('ex1data1.txt', header = None)
X = data.iloc[:,0] # read first column
y = data.iloc[:,1] # read second column
m = len(y) # number of training example
data.head() # view first few rows of the data
X = X[:,np.newaxis]
y = y[:,np.newaxis]
theta = np.zeros([2,1])
iterations = 1500
alpha = 0.01
ones = np.ones((m,1))
X = np.hstack((ones, X)) # adding the intercept term
# Computing the cost
def computeCost(X, y, theta):
    temp = np.dot(X, theta) - y
    return np.sum(np.power(temp, 2)) / (2*m)
J = computeCost(X, y, theta)
print(J)
# Finding the optimal parameters using Gradient Descent
def gradientDescent(X, y, theta, alpha, iterations):
    for _ in range(iterations):
        temp = np.dot(X, theta) - y
        temp = np.dot(X.T, temp)
        theta = theta - (alpha/m) * temp
    return theta
theta = gradientDescent(X, y, theta, alpha, iterations)
print(theta)
# Plot showing the best fit line
plt.scatter(X[:,1], y)
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
plt.plot(X[:,1], np.dot(X, theta))
plt.show()
```

Output obtained after Task 2:



Source code credit: Srikar