Programming Exercise 1: Linear Regression

Problem Statement

Suppose you are the CEO of a restaurant franchise and are considering different cities for opening a new outlet.

- You would like to expand your business to cities that may give your restaurant higher profits.
- The chain already has restaurants in various cities and you have data for profits and populations from the cities.
- You also have data on cities that are candidates for a new restaurant. For these cities, you have the city population.

Can you use the data to help you identify which cities may potentially give your business higher profits?

Files included in this exercise

File	Description
data/ex1data1.txt	Dataset for linear regression with one variable
public_tests.py	Functions to test cost and gradient calculation
utils.py	Functions to load data

File	Description
[⋆] `linear_reg.py`	Functions to compute the cost and the gradient of linear regression and to run gradient descent

[⋆] indicates files you will need to complete

Visualize your data

For this dataset, you can use a scatter plot to visualize the data, since it has only two properties to plot (profit and population), as shown in Figure 1.1.

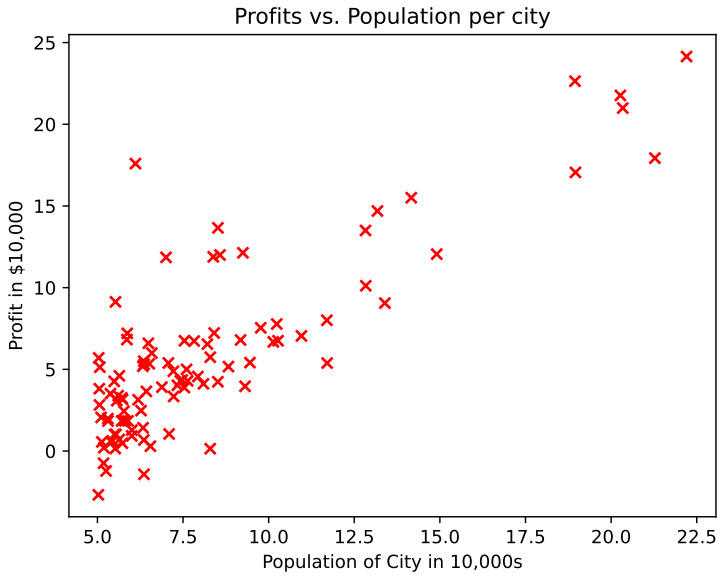

Figure 1.1: Dataset

Compute cost

The cost function for linear regression $J(w, b)$ is defined as

$$J(w,b) = \frac{1}{2m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})^2$$

Complete the `linear_reg.compute_cost` function to implement the computation of the cost:

```
def compute_cost(x, y, w, b):
    """
    Computes the cost function for linear regression.

    Args:
        x (ndarray): Shape (m,) Input to the model (Population of cities)
        y (ndarray): Shape (m,) Label (Actual profits for the cities)
        w, b (scalar): Parameters of the model

    Returns
        total_cost (float): The cost of using w,b as the parameters for
            linear regression to fit the data points in x and y
    """

    return total_cost
```

You can check if your implementation is correct by computing the cost with some initial values for parameters w and b. For example, with $w = 2$ and $b = 1$ the expected output of the cost function is 75.203. In addition you should also run and pass the tests from `public_tests.compute_cost_test`.

Compute gradient

The gradient for linear regression is defined as:

$$\frac{\partial J(w,b)}{\partial b} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})$$

$$\frac{\partial J(w,b)}{\partial w} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)}$$

Complete the `linear_reg.compute_gradient` function to implement the computation of the

gradient:

```
def compute_gradient(x, y, w, b):
    """
    Computes the gradient for linear regression
    Args:
      x (ndarray): Shape (m,) Input to the model (Population of cities)
      y (ndarray): Shape (m,) Label (Actual profits for the cities)
      w, b (scalar): Parameters of the model
    Returns
      dj_dw (scalar): The gradient of the cost w.r.t. the parameters w
      dj_db (scalar): The gradient of the cost w.r.t. the parameter b
     """


    return dj_dw, dj_db
```

You can check if your implementation is correct by computing the gradient with some initial values for parameters w and b. For example, with $w = 0$ and $b = 0$ the expected output of the gradient function is $-65.32884975, -5.83913505154639$, while with $w = 0.2$ and $b = 0.2$ the expected output of the gradient function is $-47.41610118, -4.007175051546391$. In addition you should also run and pass the tests from `public_tests.compute_gradient_test`.

Gradient descent

The gradient descent algorithm is:

$$\text{repeat until convergence:}$$
$$b = b - \alpha \frac{\partial}{\partial b} J(w, b)$$
$$w = w - \alpha \frac{\partial}{\partial w} J(w, b)$$

where, parameters w, b are both updated simultaniously.

Complete the `linear_reg.gradient_descent` function to implement the batch gradient descent algorithm:

```
def gradient_descent(x, y, w_in, b_in, cost_function, gradient_function,
    alpha, num_iters):
```

```
    """
    Performs batch gradient descent to learn theta. Updates theta by taking
    num_iters gradient steps with learning rate alpha

    Args:
      x :    (ndarray): Shape (m,)
      y :    (ndarray): Shape (m,)
      w_in, b_in : (scalar) Initial values of parameters of the model
      cost_function: function to compute cost
      gradient_function: function to compute the gradient
      alpha : (float) Learning rate
      num_iters : (int) number of iterations to run gradient descent
    Returns
      w : (ndarray): Shape (1,) Updated values of parameters of the model
          after running gradient descent
      b : (scalar) Updated value of parameter of the model after
          running gradient descent
      J_history : (ndarray): Shape (num_iters,) J at each iteration,
          primarily for graphing later
    """


    return w, b, J_history
```

You can now check your implementation of gradient descent initializing w and b to 0. and running it 1500 iterations with a learning rate $\alpha = 0.01$ which should learn the parameter values: $w = 1.16636235, b = -3.63029143940436$.

You can now use the final parameters from gradient descent to plot the linear fit shown in Figure 1.2.

Your final values of w, b can also be used to make predictions on profits. Let's predict what the profit would be in areas of 35,000 and 70,000 people.

Submission of the assignment

The assignment must be delivered using the submission mechanism of the virtual campus. A single file will be delivered in pdf format containing the memory of the practice, including the code developed and the comments and graphs that are considered most appropriate to explain the results obtained.

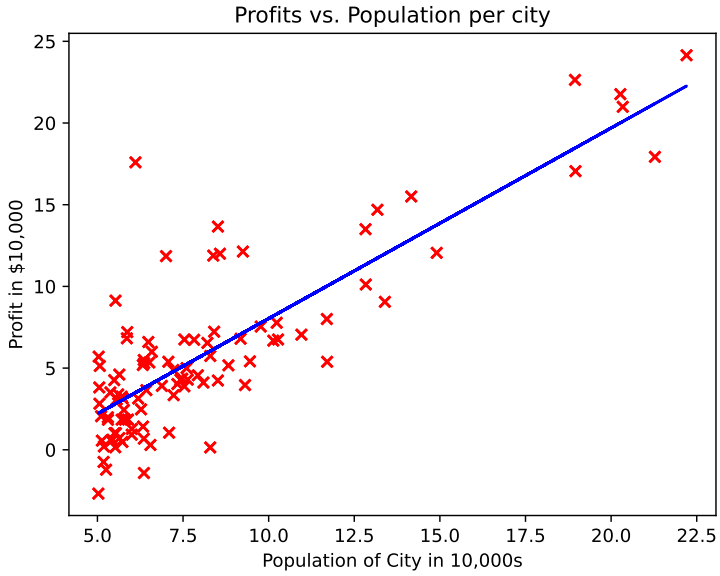

Figure 1.2: Linear fit