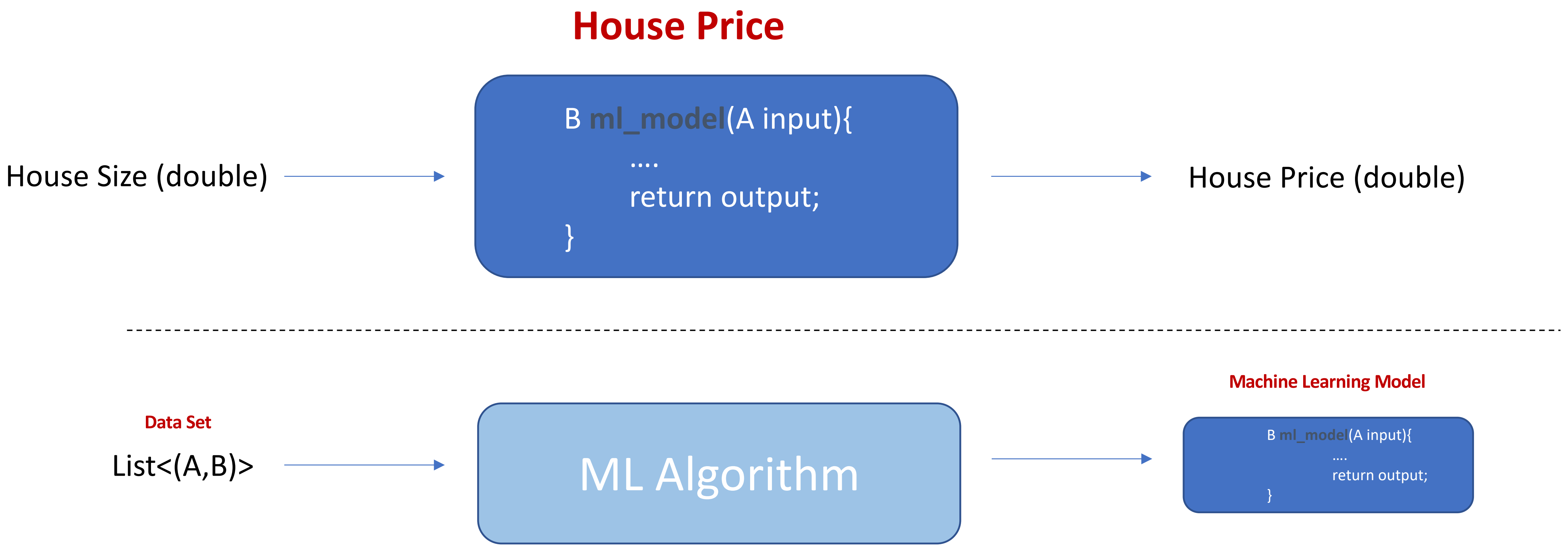
# Linear Regression I

# Andrew Ng’s Machine Learning Course


# Predict House Price given House Size



# Predict House Price given House Size

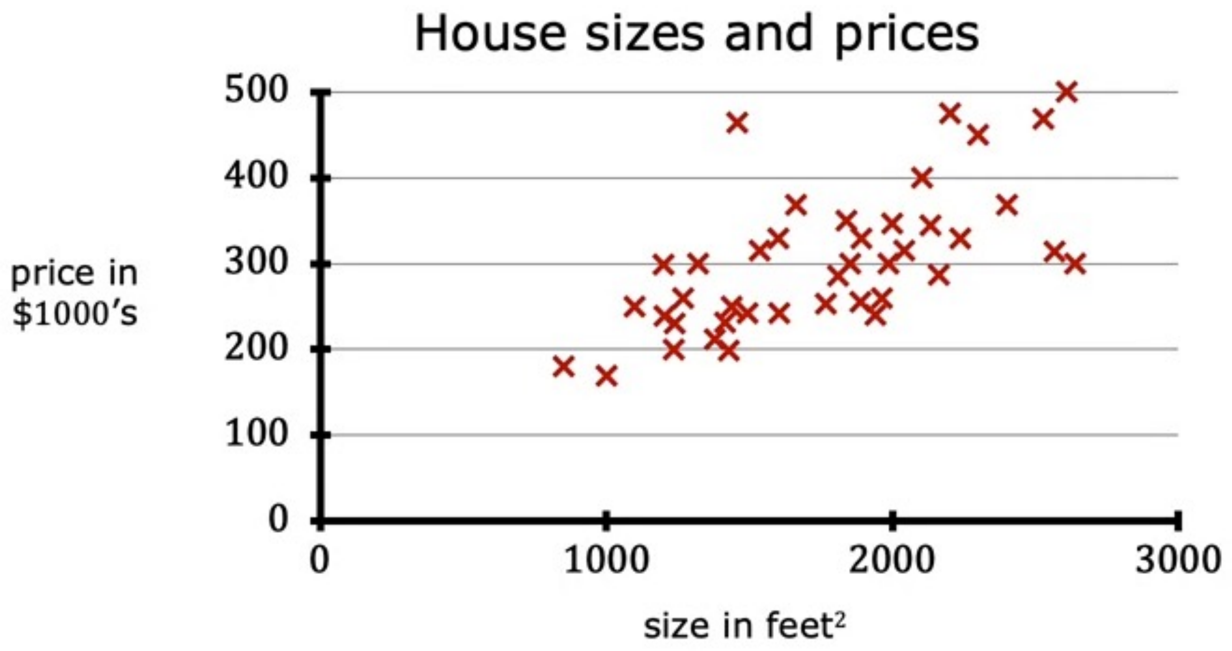
Data table

| size in feet$^2$ | price in \$1000's |
|---|---|
| 2104 | 400 |
| 1416 | 232 |
| 1534 | 315 |
| 852 | 178 |
| ... | ... |
| 3210 | 870 |

# Predict House Price given House Size

Data table

| size in feet² | price in $1000's |
|---|---|
| 2104 | 400 |
| 1416 | 232 |
| 1534 | 315 |
| 852 | 178 |
| ... | ... |
| 3210 | 870 |



# Terminology and Notation

## Terminology

Training set: Data used to train the model

| size in feet$^2$ | price in \$1000's |
|---|---|
| 2104 | 400 |
| 1416 | 232 |
| 1534 | 315 |
| 852 | 178 |
| ... | ... |
| 3210 | 870 |

## Notation:

$x$ = "input" variable
feature

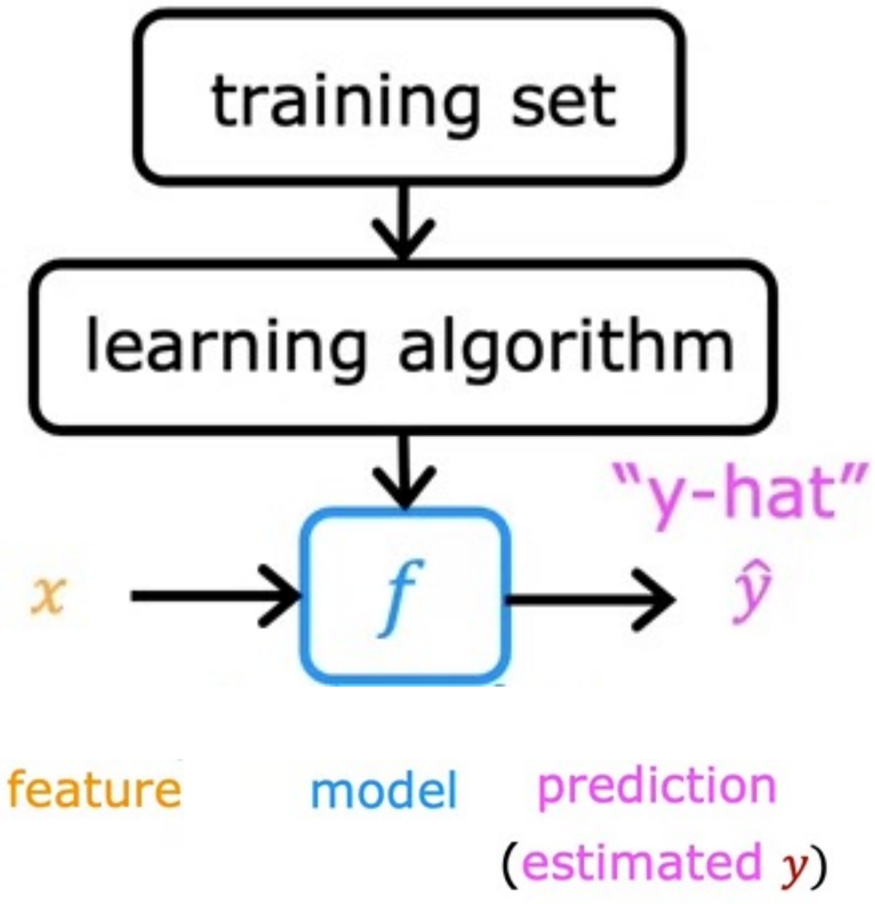
$y$ = "output" variable
"target" variable

$m$ = number of training examples

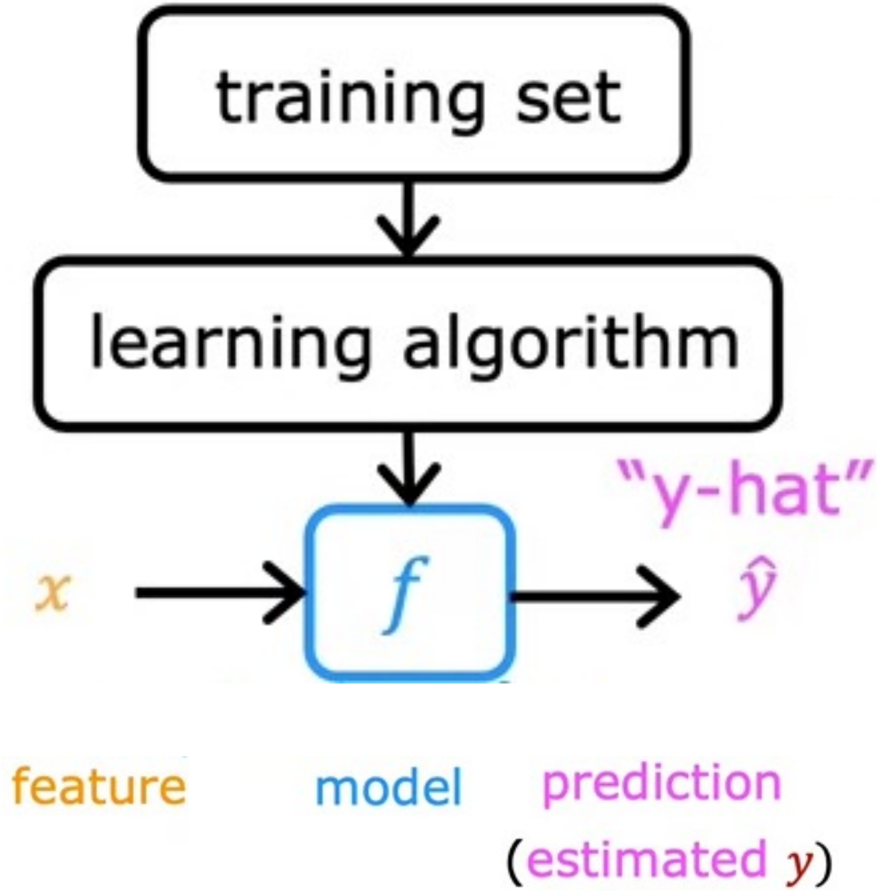
= single training example

$(x^{(i)}, y^{(i)})$ = $\text{i}^{\text{th}}$ training example

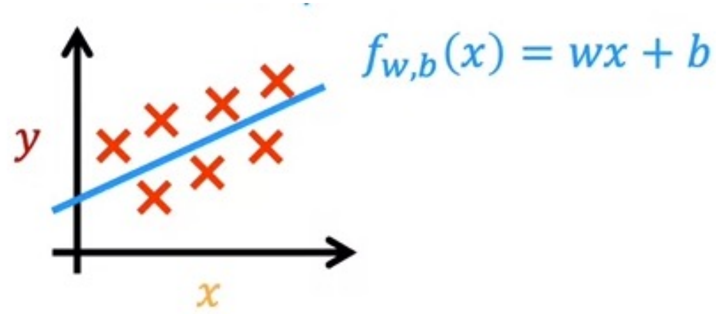
# Machine Learning Model



# Machine Learning Model




How to represent $f$?



# Linear Regression

## Training set

| size in feet$^2$ $(x)$ | price \$1000's $(y)$ |
|---|---|
| 2104 | 460 |
| 1416 | 232 |
| 1534 | 315 |
| 852 | 178 |
| ... | ... |

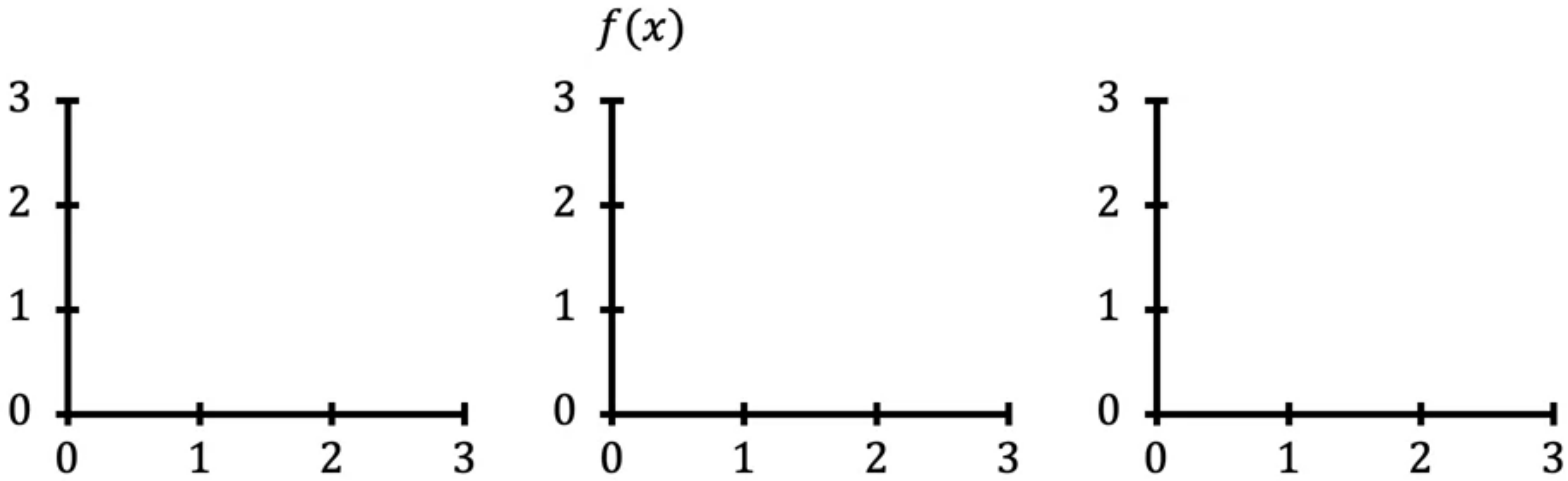
Model: $f_{w,b}(x) = wx + b$

$w, b$: parameters

What do $w, b$ do?

# Linear Regression

$$f_{w,b}(x) = wx + b$$



# Exercise 1: Model Function

## Exercise 1

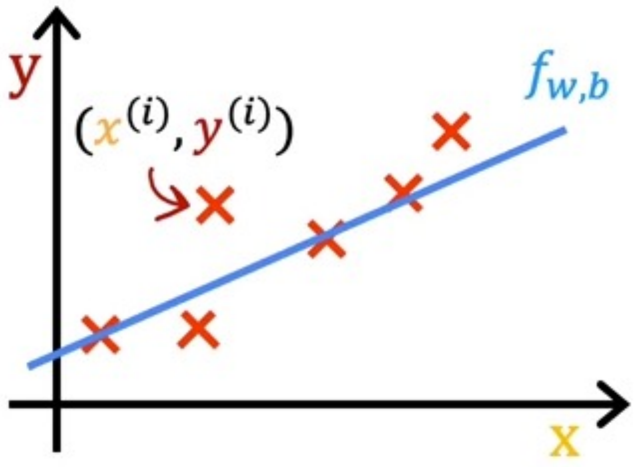
Complete the function `compute_model_output` to compute the output of a linear regression model given a vector of $x$ values and a two parameters $w$ and $b$.

> **Note**: The argument description `(ndarray (m,))` describes a Numpy n-dimensional array of shape (m,). `(scalar)` describes an argument without dimensions, just a magnitude.
> **Note**: `np.zero(n)` will return a one-dimensional numpy array with $n$ entries

```
[ ] def compute_model_output(x, w, b):
        """
        Computes the prediction of a linear model
        Args:
          x (ndarray (m,)): Data, m examples
          w,b (scalar)    : model parameters
        Returns
          y (ndarray (m,)): target values
        """
        m = x.shape[0]
        f_wb = np.zeros(m)
        for i in range(m):
            f_wb[i] = 0# Introduce code here

        return f_wb
```

# Cost Function




$$\hat{y}^{(i)} = f_{w,b}(x^{(i)})$$

$$f_{w,b}(x^{(i)}) = wx^{(i)} + b$$

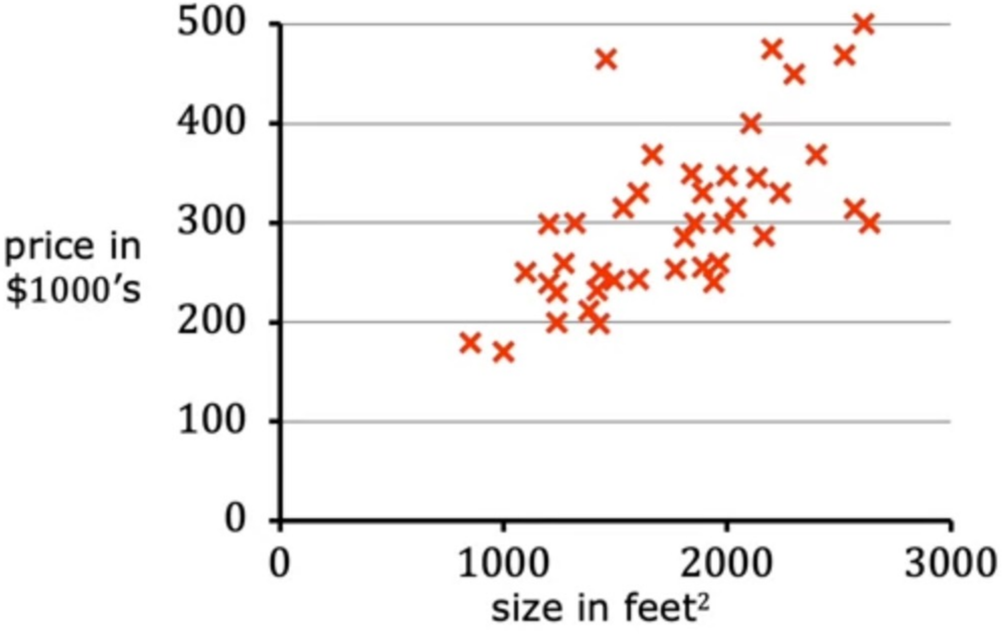
$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2$$

Find $w, b$:
$\hat{y}^{(i)}$ is close to $y^{(i)}$ for all $(x^{(i)}, y^{(i)})$.

# Cost Function

$f_{w,b}$

(function of $x$)



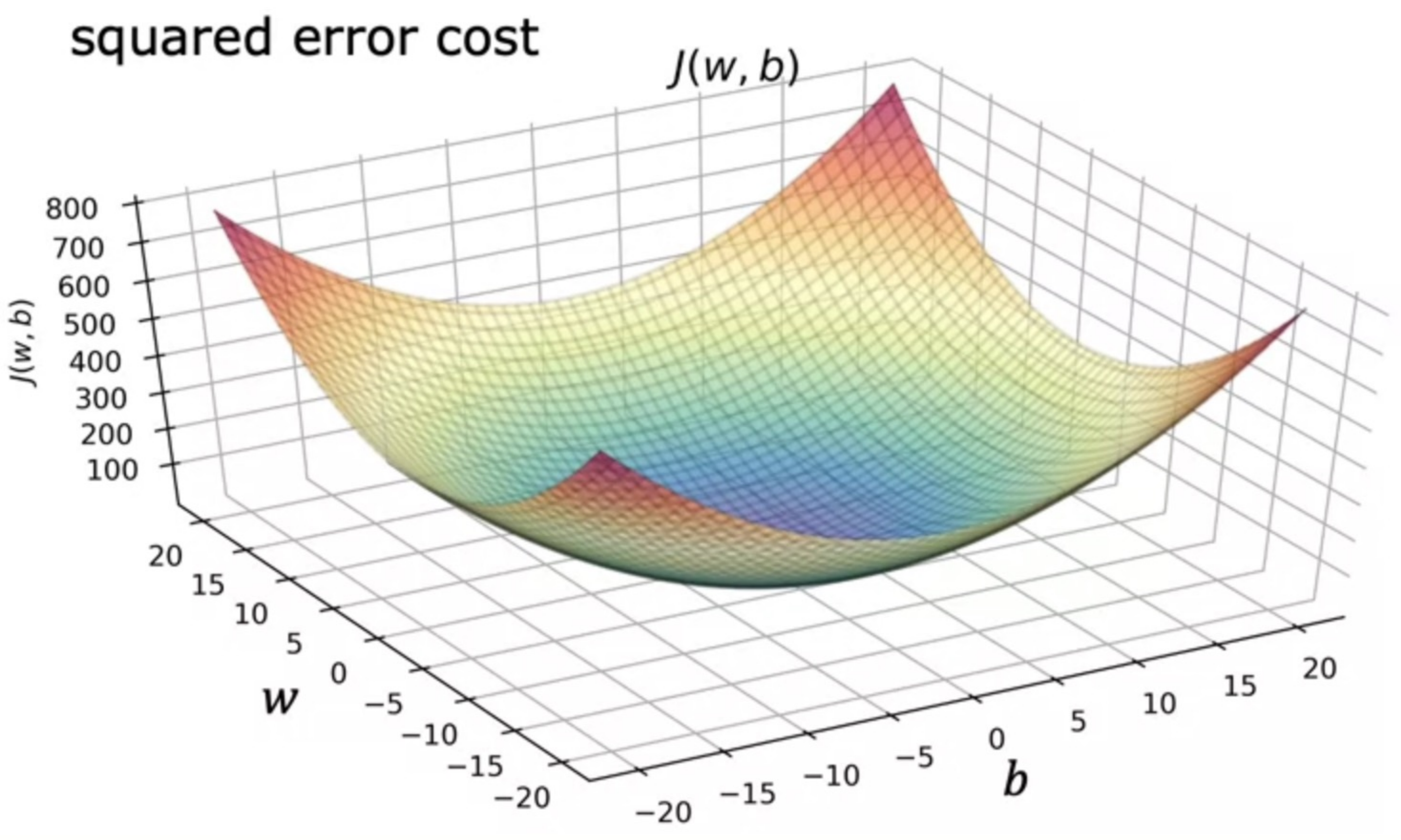

$J$

(function of $w, b$)

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

Objective $\qquad \underset{w,b}{\text{minimize}}\, J(w,b)$

# Cost Function for Linear Regression



# Exercise 2: Cost Function

## Exercise 2: Cost Function

Complete the code of the `compute_cost` methdod below to:

- Iterate over the training examples, and for each example, compute:
  - The prediction of the model for that example
$$f_{wb}(x^{(i)}) = wx^{(i)} + b$$
  - The cost for that example
$$cost^{(i)} = (f_{wb}(x^{(i)}) - y^{(i)})^2$$
- Return the total cost over all examples
$$J(\mathbf{w}, b) = \frac{1}{2m} \sum_{i=0}^{m-1} cost^{(i)}$$
  - Here, $m$ is the number of training examples and $\sum$ is the summation operator

If you get stuck, you can check out the hints presented after the cell below to help you with the implementation.

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

    Args:
        x_train (ndarray): Shape (m,) Input to the model (Population of cities)
        y_train (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
    """
    # number of training examples
    m = y_train.shape[0]

    total_cost = 0
    for i in range(m):
      f_i = 0
      cost_i = 0 # Hint: Use y_train[i]
      total_cost += cost_i

    total_cost/=2*m

    return total_cost
```

▸ **Click for hints**

# Minimizing the Cost Function

Have some function $J(w, b)$

Want $\min_{w,b} J(w, b)$

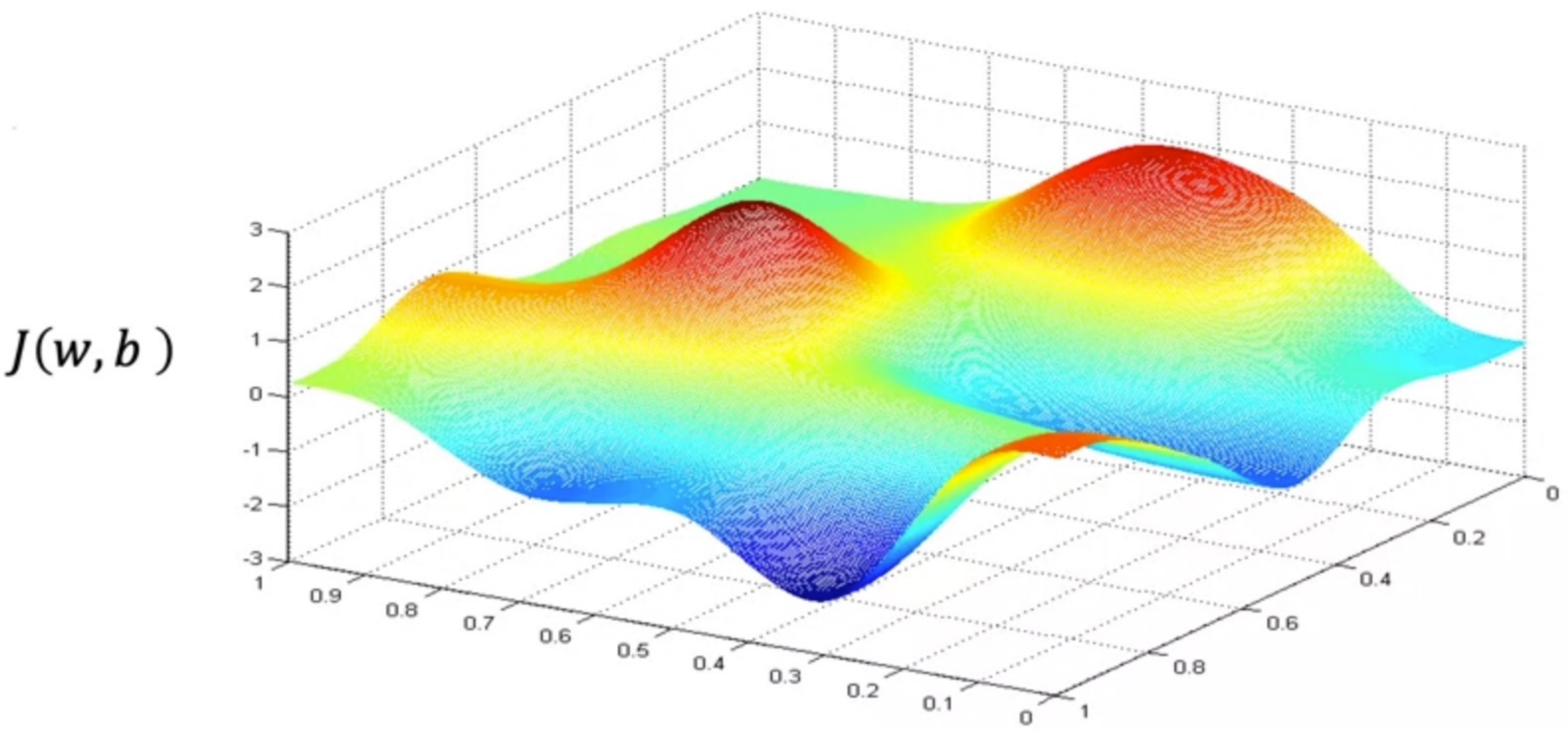
Outline:

Start with some $w, b$

Keep changing $w, b$ to reduce $J(w, b)$

Until we settle at or near a minimum

# Minimizing the Cost Function



# Gradient Descent Algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w, b)$$

# Gradient Descent Algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b)$$

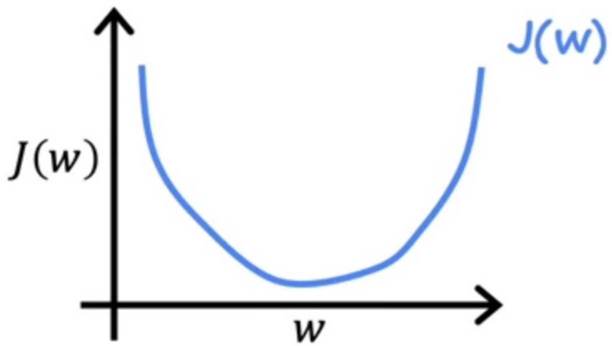
$$b = b - \alpha \frac{\partial}{\partial b} J(w, b)$$

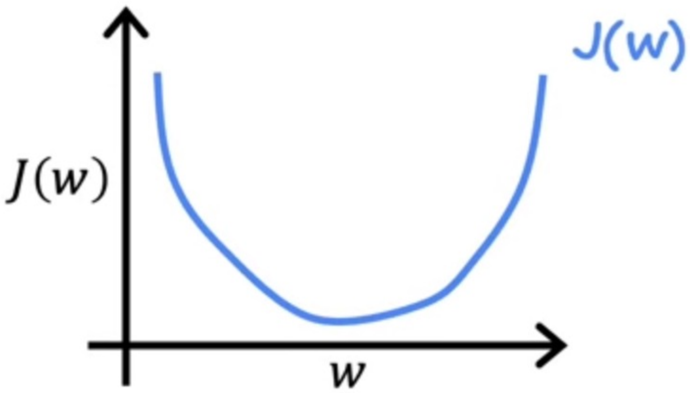
$$J(w)$$

$$w = w - \alpha \frac{\partial}{\partial w} J(w)$$

$$\min_{w} J(w)$$

# GD: Moving along the gradient



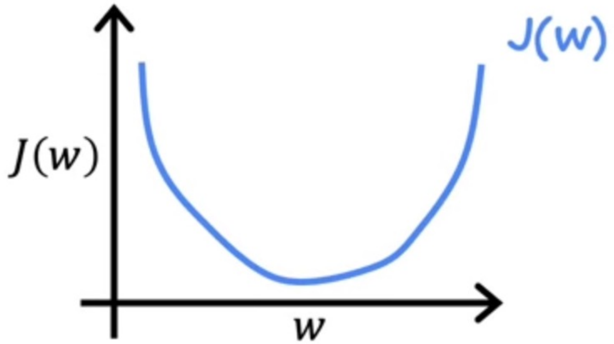



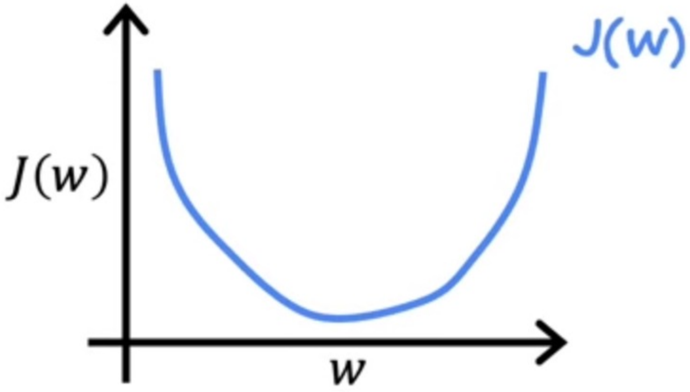

$$J(w)$$

$$w = w - \alpha \frac{\partial}{\partial w} J(w)$$

$$\min_{w} J(w)$$

# GD: Learning Rate




$$J(w)$$

$$w = w - \alpha \frac{\partial}{\partial w} J(w)$$

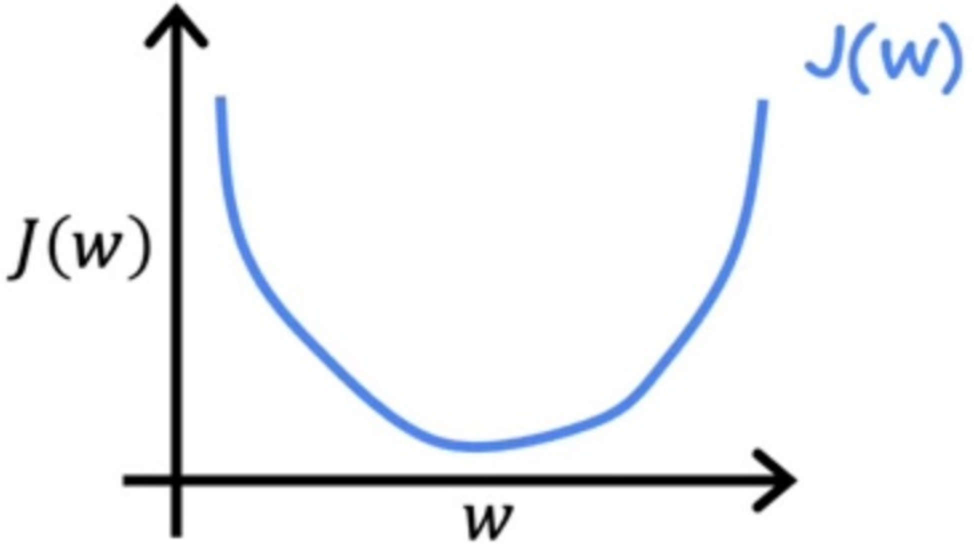
$$\min_{w} J(w)$$

# GD: Convergence

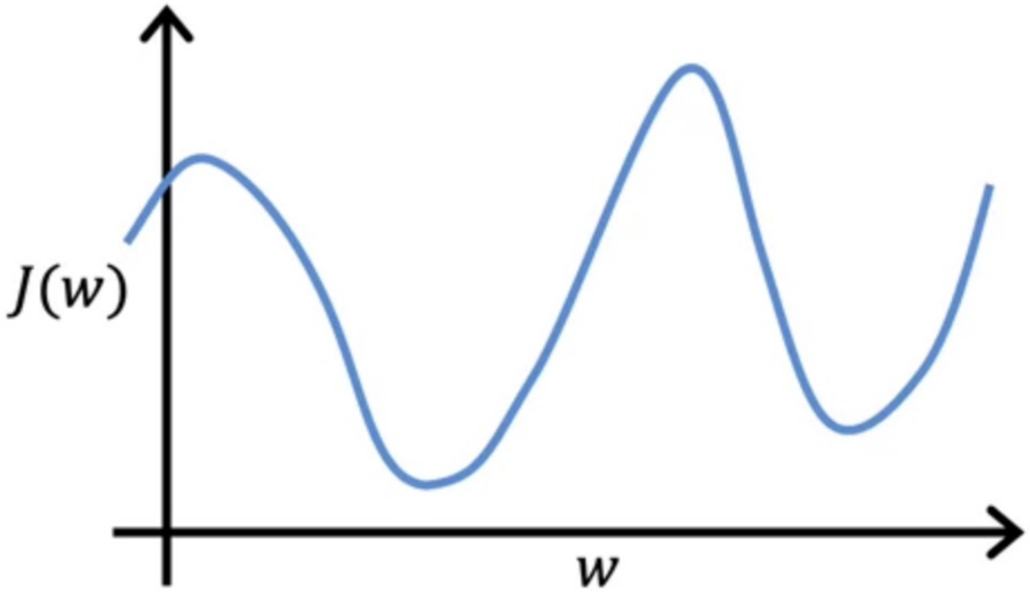
$J(w)$

$w = w - \alpha \frac{\partial}{\partial w} J(w)$

$\min_{w} J(w)$



# GD: Local Minima




$$w = w - \alpha \frac{d}{dw} J(w)$$

# GD for Linear Regression

Linear regression model

$$f_{w,b}(x) = wx + b$$

Cost function

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

Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

}

# GD for Linear Regression

Linear regression model

$$f_{w,b}(x) = wx + b$$

Cost function

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

Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha\frac{\partial}{\partial w}J(w,b)$$

$$b = b - \alpha\frac{\partial}{\partial b}J(w,b)$$

}

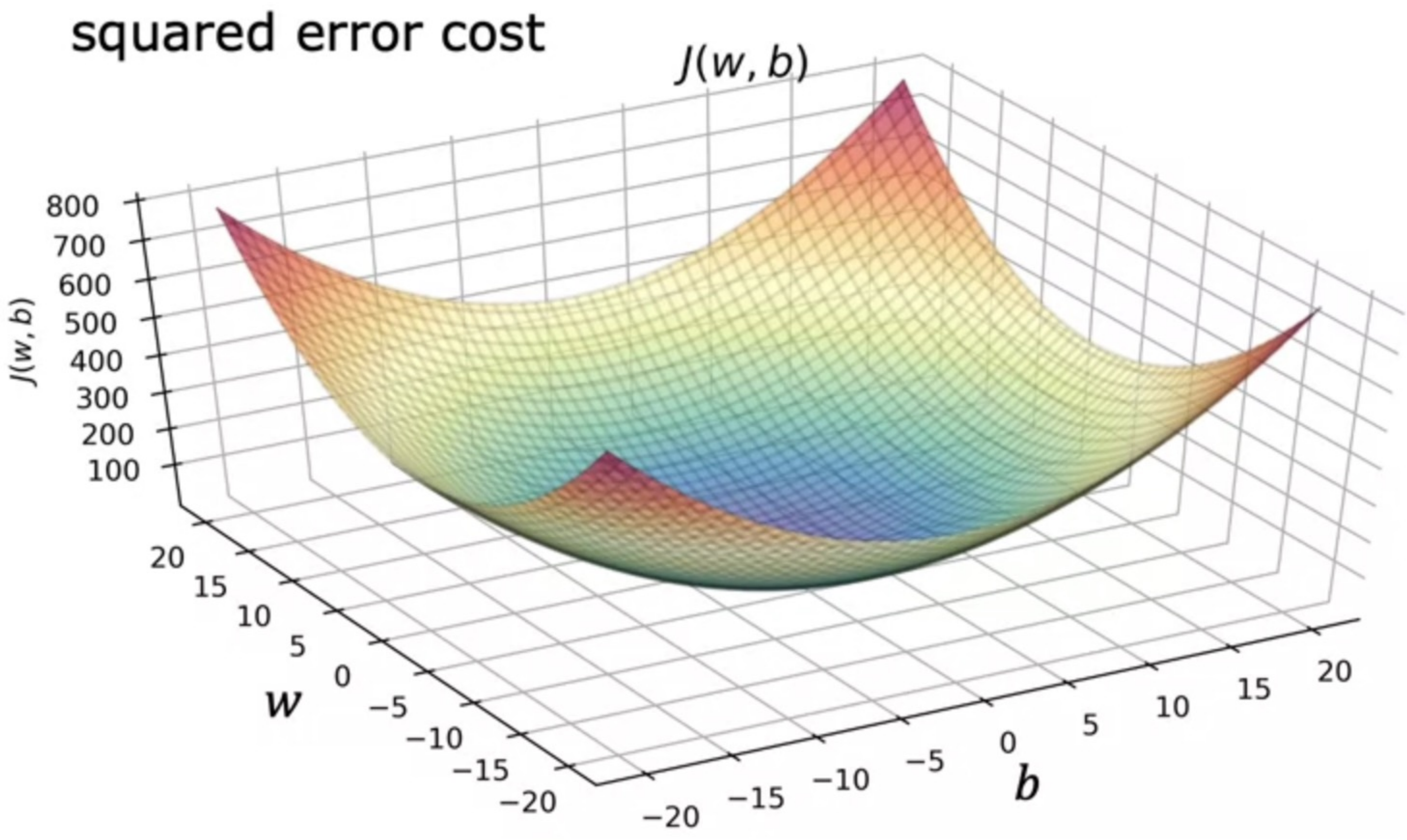
repeat until convergence {

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

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

}

# GD for Linear Regression



# Exercises 3 and 4: Gradient Descent

## Exercise 3

Please complete the `compute_gradient` function to:

- Iterate over the training examples, and for each example, compute:
  - The prediction of the model for that example
$$f_{wb}(x^{(i)}) = wx^{(i)} + b$$
  - The gradient for the parameters $w, b$ from that example
$$\frac{\partial J(w,b)}{\partial b}^{(i)} = (f_{w,b}(x^{(i)}) - y^{(i)})$$
$$\frac{\partial J(w,b)}{\partial w}^{(i)} = (f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)}$$
- Return the total gradient update from all the examples
$$\frac{\partial J(w,b)}{\partial b} = \frac{1}{m}\sum_{i=0}^{m-1}\frac{\partial J(w,b)}{\partial b}^{(i)}$$
$$\frac{\partial J(w,b)}{\partial w} = \frac{1}{m}\sum_{i=0}^{m-1}\frac{\partial J(w,b)}{\partial w}^{(i)}$$
  - Here, $m$ is the number of training examples and $\sum$ is the summation operator

If you get stuck, you can check out the hints presented after the cell below to help you with the implementation.

```python
# UNQ_C2
# GRADED FUNCTION: compute_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
     """

    # Number of training examples
    m = x.shape[0]

    # You need to return the following variables correctly
    dj_dw = 0
    dj_db = 0

    ### START CODE HERE ###

    ### END CODE HERE ###

    return dj_dw, dj_db
```

▸ **Click for hints**

## Exercise 4

You will now find the optimal parameters of a linear regression model by using batch gradient descent. Recall batch refers to running all the examples in one iteration.

- **TASK**: Implement the updating equation of gradient descent
$$b := b - \alpha\frac{\partial J(w,b)}{\partial b}$$
$$w := w - \alpha\frac{\partial J(w,b)}{\partial w} \qquad (1)$$
- A good way to verify that gradient descent is working correctly is to look at the value of $J(w,b)$ and check that it is decreasing with each step.
- Assuming you have implemented the gradient and computed the cost correctly and you have an appropriate value for the learning rate alpha, $J(w,b)$ should never increase and should converge to a steady value by the end of the algorithm.

```python
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
    """
```

# Exercise 5: Vectorization using Numpy

```
def compute_model_output(x, w, b):
    """
    Computes the prediction of a linear model
    Args:
      x (ndarray (m,)): Data, m examples
      w,b (scalar)    : model parameters
    Returns
      y (ndarray (m,)): target values
    """

    """
    m = x.shape[0]
    f_wb = np.zeros(m)
    for i in range(m):
        f_wb[i] = x[i]*w + b
    """

    f_wb = x*w + b

    return f_wb
```