Linear supervised regression

0. Import library

Import library

In [75]:

```
# Import libraries

# math library
import numpy as np

# visualization library
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png2x','pdf')
import matplotlib.pyplot as plt

# machine learning library
from sklearn.linear_model import LinearRegression

# 3d visualization
from mpl_toolkits.mplot3d import axes3d

# computational time
import time
```

1. Load dataset

Load a set of data pairs $\{x_i, y_i\}_{i=1}^n$ where x represents label and y represents target.

In [76]:

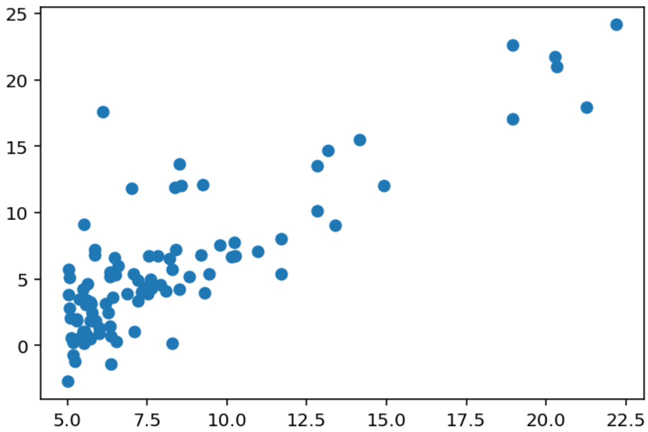
```
# import data with numpy
data = np.loadtxt('/content/drive/My Drive/Colab Notebooks/MachineLearningProject/02/
```

2. Explore the dataset distribution

Plot the training data points.

In [77]:
```python
x_train = data[:,0]
y_train = data[:,1]
plt.scatter(x_train, y_train)
```

```
&lt;matplotlib.collections.PathCollection at 0x7fc2e2604208&gt;
```



3. Define the linear prediction function

$$f_w(x) = w_0 + w_1 x$$

Vectorized implementation:

$$f_w(x) = Xw$$

with

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots \\ 1 & x_n \end{bmatrix} \quad \text{and} \quad w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad \Rightarrow \quad f_w(x) = Xw = \begin{bmatrix} w_0 + w_1 x_1 \\ w_0 + w_1 x_2 \\ \vdots \\ w_0 + w_1 x_n \end{bmatrix}$$

Implement the vectorized version of the linear predictive function.

In [78]:

```python
# construct data matrix
X = np.array([[1,x] for x in x_train])

# parameters vector
w = np.array([[1],[1]])

# predictive function definition
def f_pred(X,w):

    f = np.dot(X,w)

    return f

# Test predicitive function
y_pred = f_pred(X,w)
```

4. Define the linear regression loss

$$L(w)=\frac{1}{n}\sum_{i=1}^{n}\Big(f_w(x_i)-y_i\Big)^2$$

Vectorized implementation:

$$L(w)=\frac{1}{n}(Xw-y)^T(Xw-y)$$

with

$$Xw=\begin{bmatrix} w_0+w_1x_1 \\ w_0+w_1x_2 \\ \vdots \\ w_0+w_1x_n \end{bmatrix} \quad \text{and} \quad y=\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Implement the vectorized version of the linear regression loss function.

In [79]:

```python
# loss function definition
def loss_mse(y_pred,y):

    loss = (np.dot((y_pred - y).T, (y_pred - y))) / len(y)

    return loss


# Test loss function
y = np.array([y_train]).T # label
y_pred = f_pred(X,w) # prediction

loss = loss_mse(y_pred,y)
```

5. Define the gradient of the linear regression loss

Vectorized implementation: Given the loss

$$L(w) = \frac{1}{n}(Xw - y)^T(Xw - y)$$

The gradient is given by

$$\frac{\partial}{\partial w}L(w) = \frac{2}{n}X^T(Xw - y)$$

Implement the vectorized version of the gradient of the linear regression loss function.

In [80]:

```
# gradient function definition
def grad_loss(y_pred,y,X):

    grad = (2 * np.dot(X.T, (y_pred-y))) / len(y)

    return grad


# Test grad function
y_pred = f_pred(X,w)
grad = grad_loss(y_pred,y,X)
```

6. Implement the gradient descent algorithm

- Vectorized implementation:

$$w^{k+1} = w^k - \tau\frac{2}{n}X^T(Xw^k - y)$$

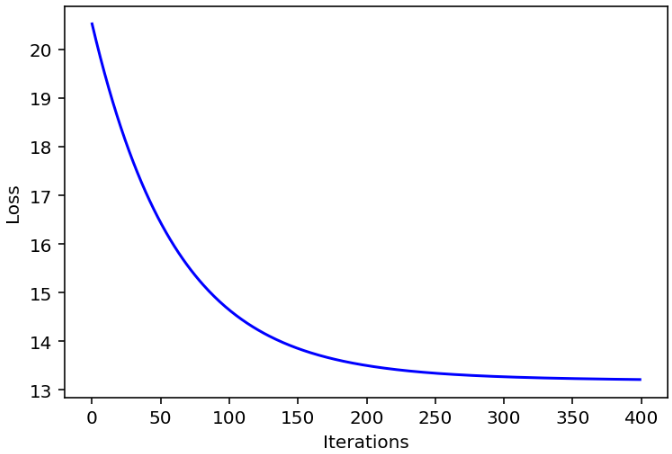
Implement the vectorized version of the gradient descent function.

Plot the loss values $L(w^k)$ with respect to iteration k the number of iterations.

In [87]:

```
# gradient descent function definition
def grad_desc(X, y, w_init, tau, max_iter):

    L_iters = []# record the loss values
    w_iters = []# record the parameter values
    w = w_init # initialization

    for i in range(max_iter): # loop over the iterations

        y_pred = f_pred(X,w) # linear predicition function
        grad_f = grad_loss(y_pred, y, X) # gradient of the loss
        w = w - tau*grad_f # update rule of gradient descent
        L_iters.append(loss_mse(y_pred, y)[0,0]) # save the current loss value
        w_iters.append(w) # save the current w value

    return w, L_iters, w_iters


# run gradient descent algorithm
start = time.time()
w_init = np.array([[1],[1]])
tau = 0.00005
max_iter = 400

w, L_iters, w_iters = grad_desc(X,y,w_init,tau,max_iter)

print('Time=',time.time() - start) # plot the computational cost
print(L_iters[max_iter-1]) # plot the last value of the loss
print(w_iters[max_iter-1]) # plot the last value of the parameter w


# plot
plt.figure(2)
plt.plot([op for op in range(max_iter)], L_iters, c='blue') # plot the loss curve
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```

```
Time= 0.008520364761352539
13.213734995339383
[[0.93641516]
 [0.71857986]]
```

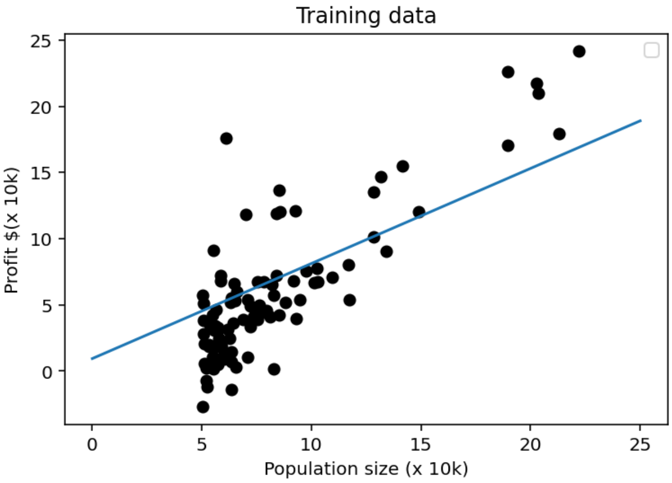


7. Plot the linear prediction function

$$f_w(x) = w_0 + w_1 x$$

In [89]:
```
# linear regression model
x_pred = np.linspace(0,25,100) # define the domain of the prediction function
y_pred = w_iters[max_iter-1][0] + w_iters[max_iter-1][1]*x_pred# compute the predicti

# plot
plt.figure(3)
plt.scatter(x_train, y_train, c='Black')
plt.plot(x_pred, y_pred)
plt.legend(loc='best')
plt.title('Training data')
plt.xlabel('Population size (x 10k)')
plt.ylabel('Profit $(x 10k)')
plt.show()
```

```
No handles with labels found to put in legend.
```



8. Comparison with Scikit-learn linear regression algorithm

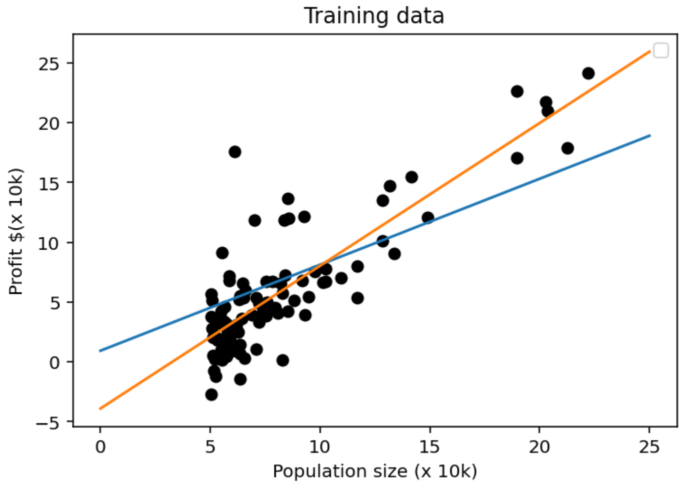
Compare with the Scikit-learn solution

In [104]:

```python
# run linear regression with scikit-learn
start = time.time()
lin_reg_sklearn = LinearRegression()
lin_reg_sklearn.fit(x_train.reshape(-1,1), y_train.reshape(-1,1)) # learn the model p
print('Time=',time.time() - start)


# compute loss value
w_sklearn = np.zeros([2,1])
w_sklearn[0,0] = lin_reg_sklearn.intercept_
w_sklearn[1,0] = lin_reg_sklearn.coef_

print(w_sklearn)

loss_sklearn = loss_mse(lin_reg_sklearn.predict(x_train.reshape(-1,1)), y) # compute

print('loss sklearn=',loss_sklearn)
print('loss gradient descent=',L_iters[-1])


# plot
y_pred_sklearn = lin_reg_sklearn.predict(x_pred.reshape(-1,1)) # prediction obtained

plt.figure(3)

plt.scatter(x_train, y_train, c='Black')
plt.plot(x_pred, y_pred)
plt.plot(x_pred, y_pred_sklearn)
plt.legend(loc='best')
plt.title('Training data')
plt.xlabel('Population size (x 10k)')
plt.ylabel('Profit $(x 10k)')
plt.show()
```

```
No handles with labels found to put in legend.
```

```
Time= 0.0011620521545410156
[[-3.89578088]
 [ 1.19303364]]
loss sklearn= [[8.95394275]]
loss gradient descent= 13.213734995339383
```



9. Plot the loss surface, the contours of the loss and the gradient descent steps

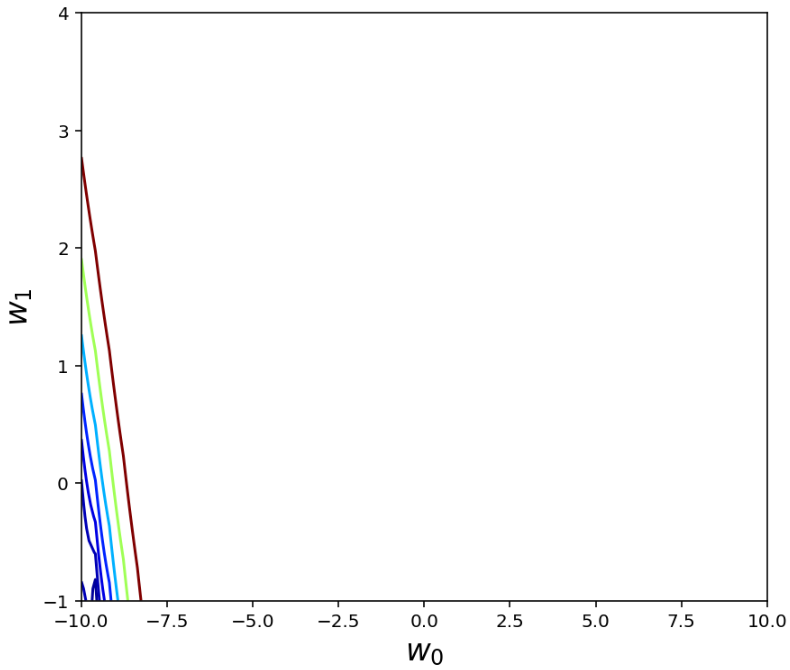
In [128]:

```
# plot gradient descent
def plot_gradient_descent(X,y,w_init,tau,max_iter):

    def f_pred(X,w):

        f = np.dot(X,w)

        return f

    def loss_mse(y_pred,y):

        loss = (np.dot((y_pred - y).T, (y_pred - y))) / len(y)

        return loss

    # gradient descent function definition
    def grad_desc(X, y, w_init, tau, max_iter):

        L_iters = []# record the loss values
        w_iters = []# record the parameter values
        w = w_init # initialization

        for i in range(max_iter): # loop over the iterations

            y_pred = f_pred(X,w) # linear predicition function
            grad_f = grad_loss(y_pred, y, X) # gradient of the loss
            w = w - tau*grad_f # update rule of gradient descent
            L_iters.append(loss_mse(y_pred, y)[0,0]) # save the current loss value
            w_iters.append(w) # save the current w value

        return w, L_iters, w_iters

    # run gradient descent
    w, L_iters, w_iters = grad_desc(X, y, w_init, tau, max_iter)

    # Create grid coordinates for plotting a range of L(w0,w1)-values
    B0 = np.linspace(-10, 10, 50)
    B1 = np.linspace(-1, 4, 50)

    xx, yy = np.meshgrid(B0, B1, indexing='xy')
    Z = np.zeros((B0.size,B1.size))

    # Calculate loss values based on L(w0,w1)-values
    for (i,j),v in np.ndenumerate(Z):
        Z[i,j] = loss_mse(f_pred(X, [[i],[j]]), y)

    # 3D visualization
    fig = plt.figure(figsize=(15,6))
    ax1 = fig.add_subplot(121)
```

```
ax2 = fig.add_subplot(122, projection='3d')

# Left plot
CS = ax1.contour(xx, yy, Z, np.logspace(-2, 3, 20), cmap=plt.cm.jet)
#ax1.scatter( )
#ax1.plot( )

# Right plot
ax2.plot_surface(xx, yy, Z, rstride=1, cstride=1, alpha=0.6, cmap=plt.cm.jet)
ax2.set_zlabel('Loss $L(w_0,w_1)$')
ax2.set_zlim(Z.min(),Z.max())

# plot gradient descent
Z2 = np.zeros([max_iter])

for i in range(max_iter):
    w0 = w_iters[i][0]
    w1 = w_iters[i][1]
    Z2[i] = L_iters[i]

w_iters = np.array(w_iters)
ax2.plot(w_iters[:,0], w_iters[:,1], Z2)
ax2.scatter(w_iters[:,0], w_iters[:,1], Z2)

# settings common to both plots
for ax in fig.axes:
    ax.set_xlabel(r'$w_0$', fontsize=17)
    ax.set_ylabel(r'$w_1$', fontsize=17)
```

In [129]:

```
# run plot_gradient_descent function
w_init = np.array([[1],[1]])
tau = 0.00005
max_iter = 400

plot_gradient_descent(X,y,w_init,tau,max_iter)
```



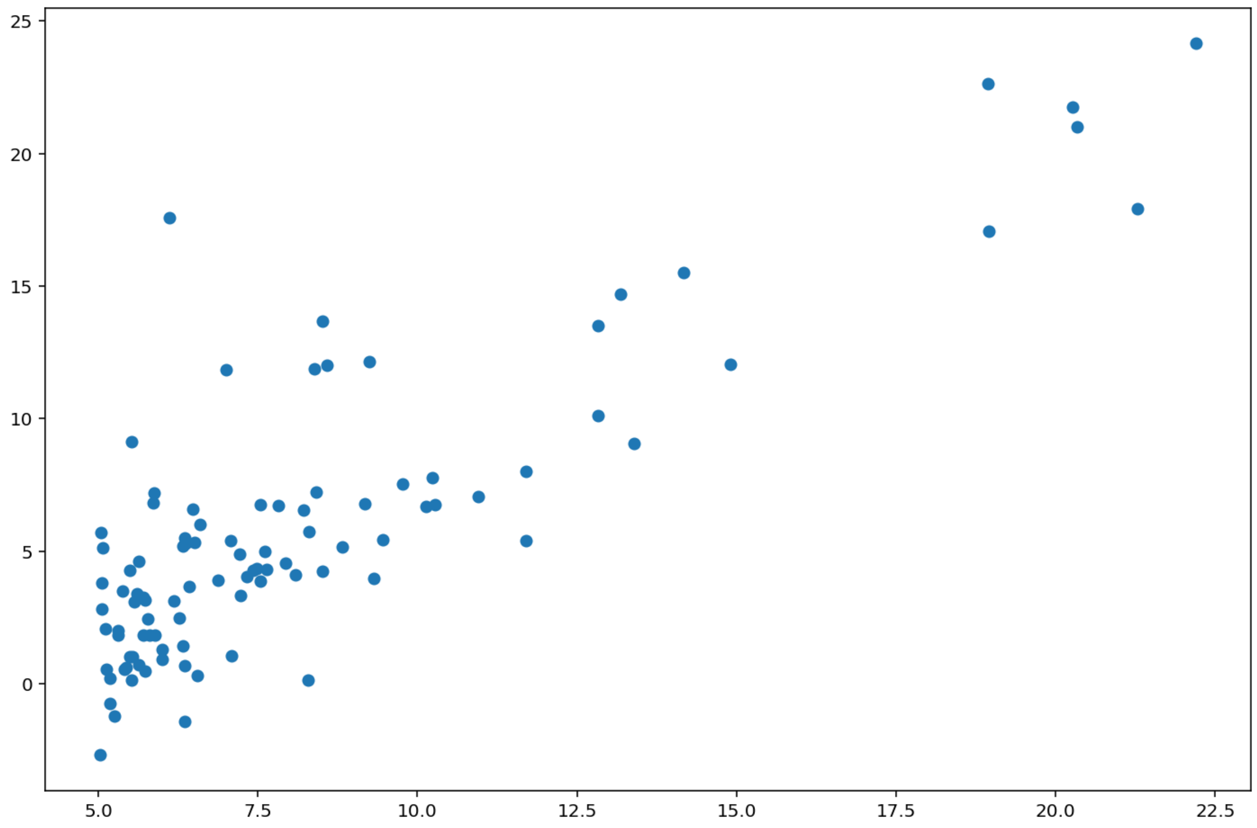
Output results

1. Plot the training data (1pt)

In [14]:

```
plt.figure(figsize=(12,8))
plt.scatter(x_train, y_train)
```

<matplotlib.collections.PathCollection at 0x7fc2e5885c18>

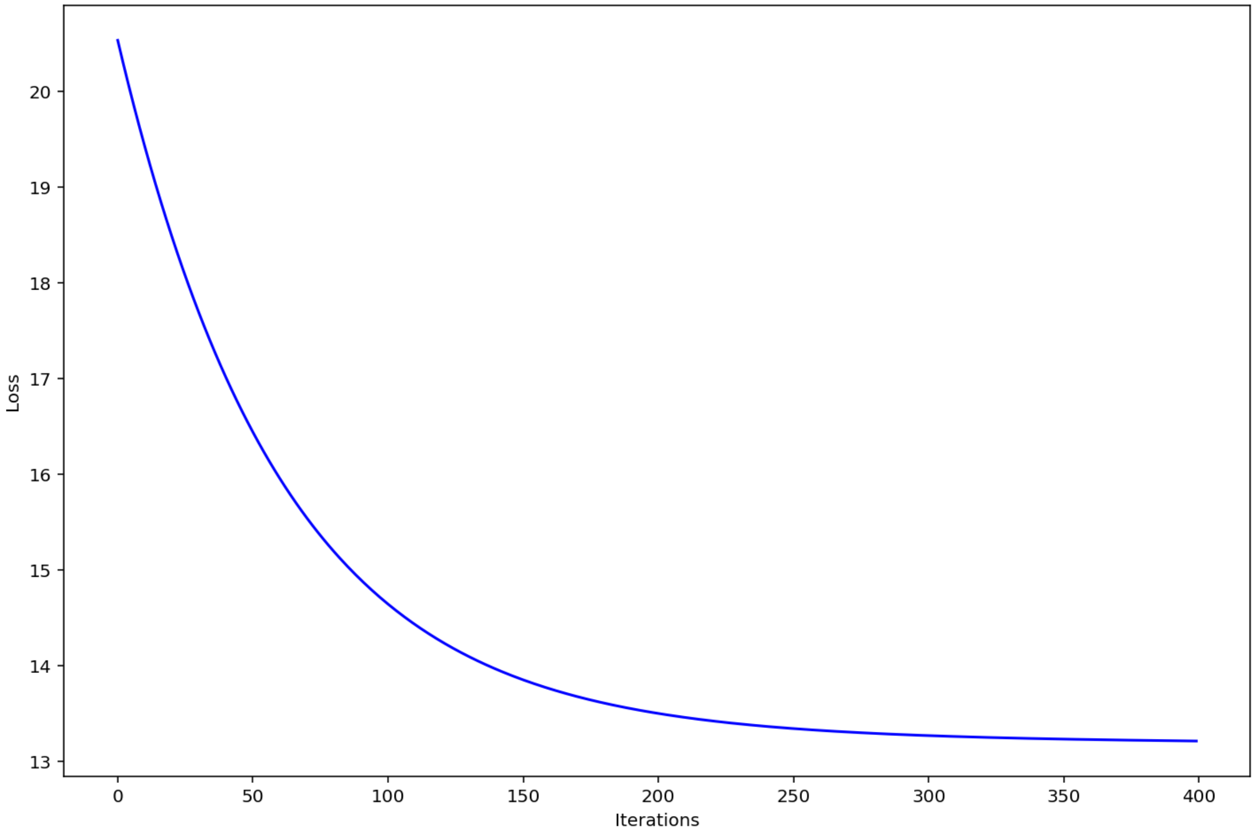


2. Plot the loss curve in the course of gradient descent (2pt)

In [70]:

```
plt.figure(2)
plt.figure(figsize=(12,8))
plt.plot([op for op in range(max_iter)], L_iters, c='blue') # plot the loss curve
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



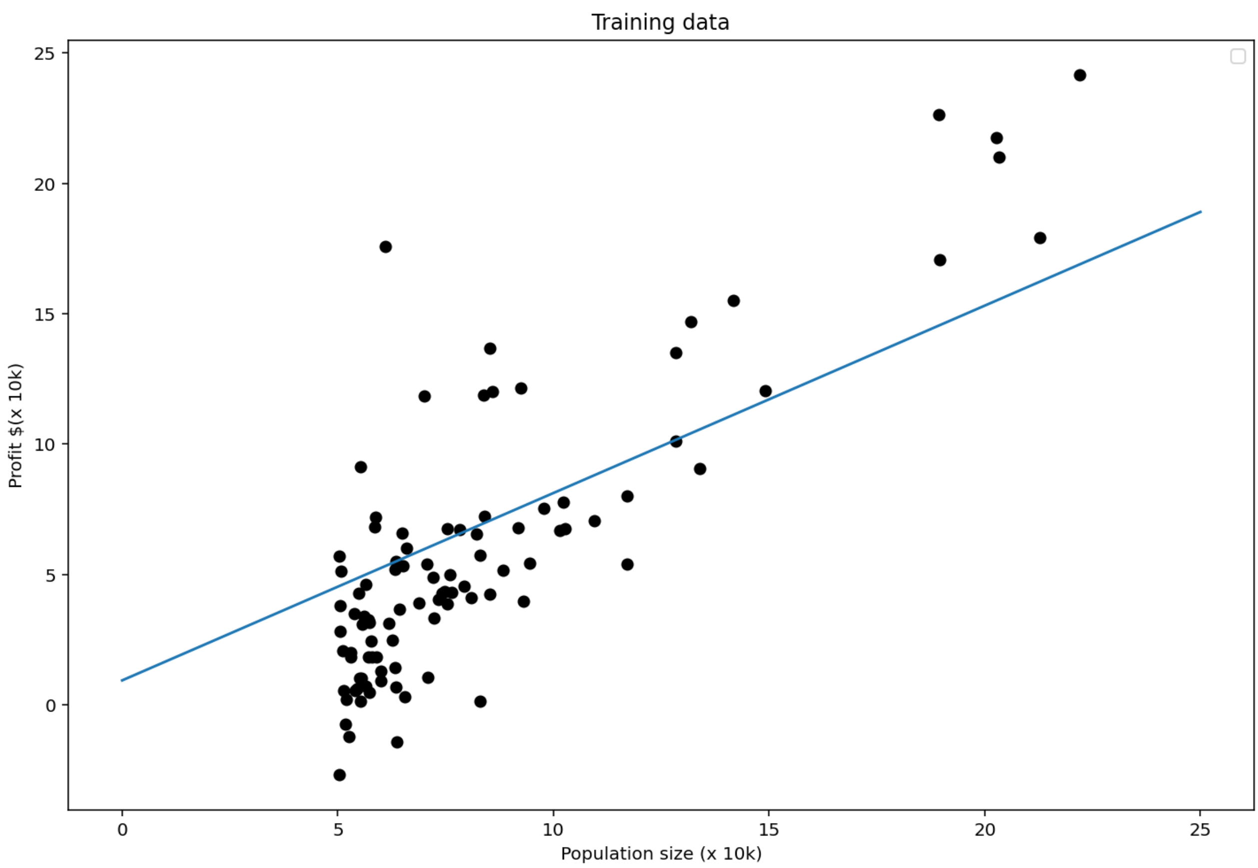
3. Plot the prediction function superimposed on the training data (2pt)

In [91]:

```
# plot
plt.figure(3)
plt.figure(figsize=(12,8))
plt.scatter(x_train, y_train, c='Black')
plt.plot(x_pred, y_pred)
plt.legend(loc='best')
plt.title('Training data')
plt.xlabel('Population size (x 10k)')
plt.ylabel('Profit $(x 10k)')
plt.show()
```

```
No handles with labels found to put in legend.
```

```
<Figure size 432x288 with 0 Axes>
```



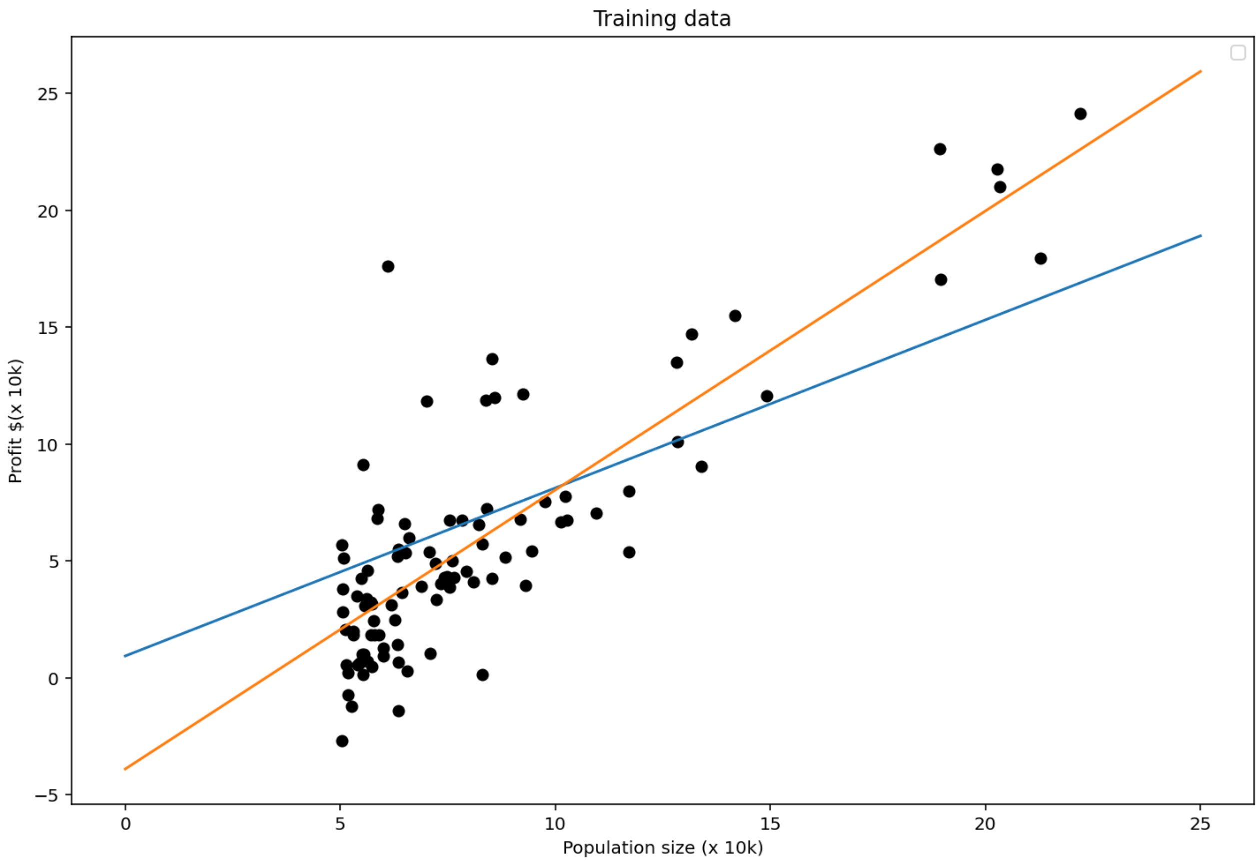
4. Plot the prediction functions obtained by both the Scikit-learn linear regression solution and the gradient descent superimposed on the training data (2pt)

In [105]:

```
plt.figure(3)
plt.figure(figsize=(12,8))
plt.scatter(x_train, y_train, c='Black')
plt.plot(x_pred, y_pred)
plt.plot(x_pred, y_pred_sklearn)
plt.legend(loc='best')
plt.title('Training data')
plt.xlabel('Population size (x 10k)')
plt.ylabel('Profit $(x 10k)')
plt.show()
```

```
No handles with labels found to put in legend.
```

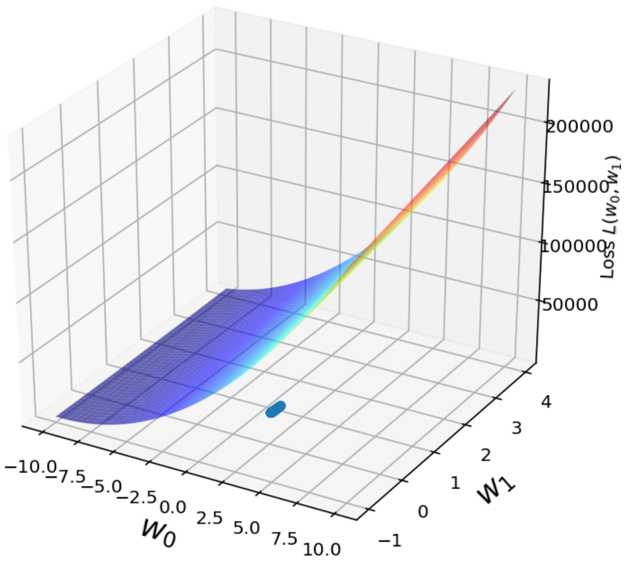
```
<Figure size 432x288 with 0 Axes>
```



5. Plot the loss surface (right) and the path of the gradient descent (2pt)

In [125]:

```
plot_gradient_descent(X,y,w_init,tau,max_iter)
```



6. Plot the contour of the loss surface (left) and the path of the gradient descent (2pt)

In [130]:

```
plot_gradient_descent(X,y,w_init,tau,max_iter)
```