```
In [2]: %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np
        import os
        import pandas as pd
```

```
In [3]: df = pd.read_csv('HW1.csv')
        df.head() # To get first n rows from the dataset default value of n is 5
        M=len(df)
        M
```

Out[3]: 100

```
In [4]: df.describe()
```

Out[4]:

|       | X1         | X2         | X3         | Y          |
|-------|------------|------------|------------|------------|
| count | 100.000000 | 100.000000 | 100.000000 | 100.000000 |
| mean  | 2.000000   | 2.000000   | 1.960000   | 1.851276   |
| std   | 1.172181   | 1.172154   | 1.163005   | 2.774643   |
| min   | 0.000000   | 0.070303   | 0.027879   | -5.332455  |
| 25%   | 1.000000   | 0.979394   | 0.952121   | 0.527533   |
| 50%   | 2.000000   | 2.009697   | 1.949091   | 2.879003   |
| 75%   | 3.000000   | 3.040000   | 2.946061   | 3.925389   |
| max   | 4.000000   | 3.949091   | 3.943030   | 5.545892   |

```
In [5]: # Separate features and labels
        X = df.values[:, 2]  # get input values from first column -- X is a list here
        y = df.values[:, 3]  # get output values from forth column -- Y is the list here
        m = len(y)   # Number of training examples
        n = len(X)   # Number of training examples


        # Display first 5 records and the total number of training examples
        print('X = ', X[: 5])
        print('y = ', y[: 5])
        print('m = ', m)
        print('n = ', n)
```
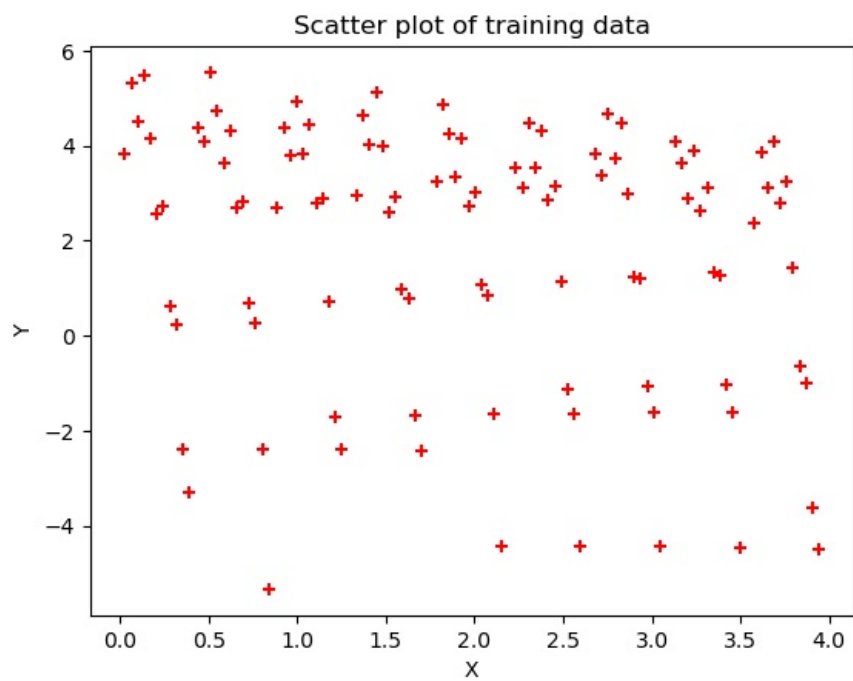
```
X =  [0.44       0.88848485 1.3369697  1.78545454 2.23393939]
y =  [4.38754501 2.6796499  2.96848981 3.25406475 3.53637472]
m =  100
n =  100
```

```
In [6]: # Scatter plot
        plt.scatter(X, y, color='red', marker='+')

        # Grid, labels, and title
        # plt.grid(True)
        plt.rcParams["figure.figsize"] = (10, 10)
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.title('Scatter plot of training data')

        # Show the plot
        plt.show()
```
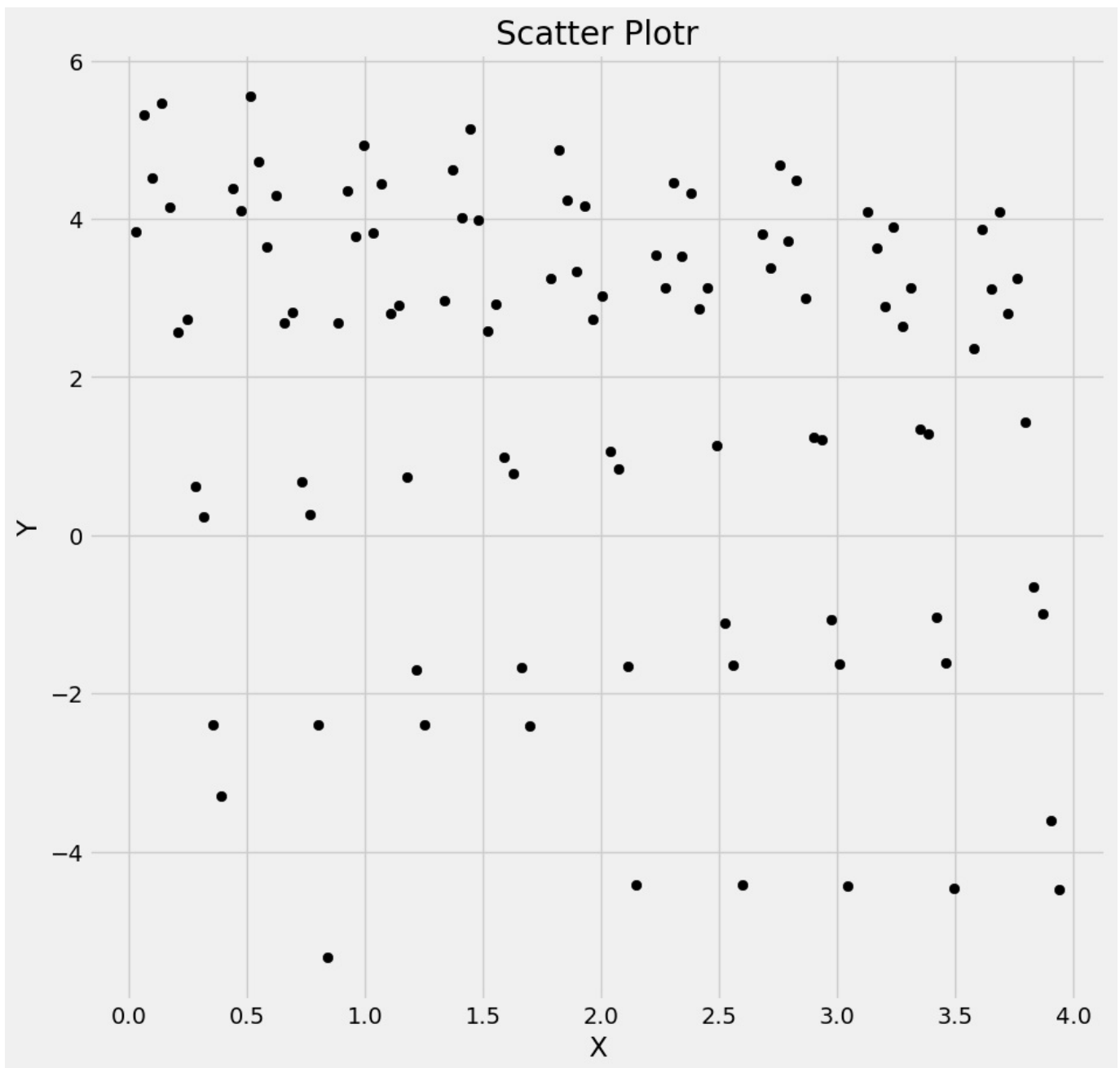
Scatter plot of training data

In [7]:
```python
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')

plt.scatter(X, y, color='black')
plt.xlabel('X')
plt.ylabel('Y')
plt.gca().set_title("Scatter Plotr")
```

Out[7]: Text(0.5, 1.0, 'Scatter Plotr')

**Scatter Plotr**

```
m = len(y)  # Number of training examples
n = len(X)  # Number of training examples
```

```
X_0 = np.ones((m, 1))
X_0[:5]
```

```
array([[1.],
       [1.],
       [1.],
       [1.],
       [1.]])
```

```
X_1 = X.reshape(m, 1)
X_1[:10]
```

```
array([[0.44      ],
       [0.88848485],
       [1.3369697 ],
       [1.78545454],
       [2.23393939],
       [2.68242424],
       [3.13090909],
       [3.57939394],
       [0.02787879],
       [0.47636364]])
```

```
# Lets use hstack() function from numpy to stack X_0 and X_1 horizontally (i.e. column
# This will be our final X matrix (feature matrix)
X = np.hstack((X_0, X_1))
X[:5]
```

```
Out[11]: array([[1.        , 0.44      ],
                 [1.        , 0.88848485],
                 [1.        , 1.3369697 ],
                 [1.        , 1.78545454],
                 [1.        , 2.23393939]])
```

```
In [12]: theta = np.zeros(2)
         theta
```

```
Out[12]: array([0., 0.])
```

```
In [13]: def compute_cost(X, y, theta):
             """
             Compute cost for linear regression.
             Input Parameters
             ----------------
             X : 2D array where each row represent the training example and each column represent
             m= number of training examples
             n= number of features (including X_0 column of ones)
             y : 1D array of labels/target value for each traing example. dimension(1 x m)
             theta : 1D array of fitting parameters or weights. Dimension (1 x n)
             Output Parameters
             -----------------
             J : Scalar value.
             """
             predictions = X.dot(theta)
             errors = np.subtract(predictions, y)
             sqrErrors = np.square(errors)
             J = 1 / (2 * m) * np.sum(sqrErrors)
             return J

         def gradient_descent(X, y, theta, alpha, iterations):
             """
             Compute cost for linear regression.
             Input Parameters
             ----------------
             X : 2D array where each row represent the training example and each column represent
             m= number of training examples
             n= number of features (including X_0 column of ones)
             y : 1D array of labels/target value for each traing example. dimension(m x 1)
             theta : 1D array of fitting parameters or weights. Dimension (1 x n)
             alpha : Learning rate. Scalar value
             iterations: No of iterations. Scalar value.
             Output Parameters
             -----------------
             theta : Final Value. 1D array of fitting parameters or weights. Dimension (1 x n)
             cost_history: Conatins value of cost for each iteration. 1D array. Dimansion(m x 1)
             """
             cost_history = np.zeros(iterations)
             for i in range(iterations):
                 predictions = X.dot(theta)
                 errors = np.subtract(predictions, y)
                 sum_delta = (alpha / m) * X.transpose().dot(errors);
                 theta = theta - sum_delta;
                 cost_history[i] = compute_cost(X, y, theta)
             return theta, cost_history
```

```
In [14]: # Lets compute the cost for theta values
         cost = compute_cost(X, y, theta)
         print('The cost for given values of theta_0 and theta_1 =', cost)
```

```
The cost for given values of theta_0 and theta_1 = 5.524438459196242
```

```
In [15]: theta = [0., 0.]
         iterations = 1500;
         alpha = 0.01;

         theta, cost_history = gradient_descent(X, y, theta, alpha, iterations)
         print('Final value of theta =', theta)
         print('cost_history =', cost_history)
```

```
Final value of theta = [ 2.78048129 -0.48451631]
cost_history = [5.40768785 5.30397076 5.21178297 ... 3.63053597 3.6305311  3.63052625]
```

```
In [16]: # Since X is list of list (feature matrix) lets take values of column of index 1 only
         plt.scatter(X[:,1], y, color='b', marker= '+', label= 'Training Data')
         plt.plot(X[:,1],X.dot(theta), color='r', label='Linear Regression')
         plt.rcParams["figure.figsize"] = (10,6)

         plt.xlabel('Input Variable')
         plt.ylabel('Output Variable')
```
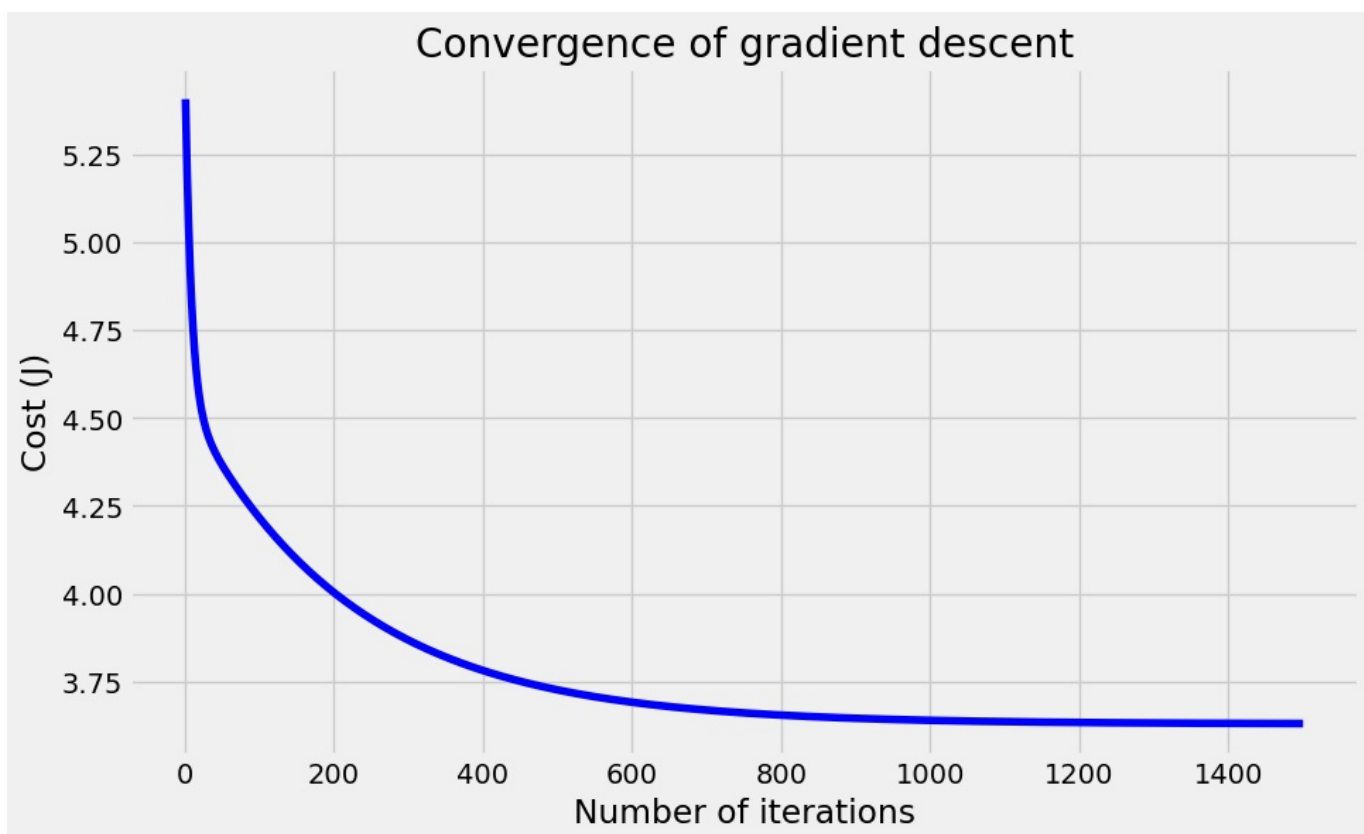
```
plt.title('Linear Regression using 2 variables')
plt.legend()
```

Out[16]: &lt;matplotlib.legend.Legend at 0x1d072fd6090&gt;
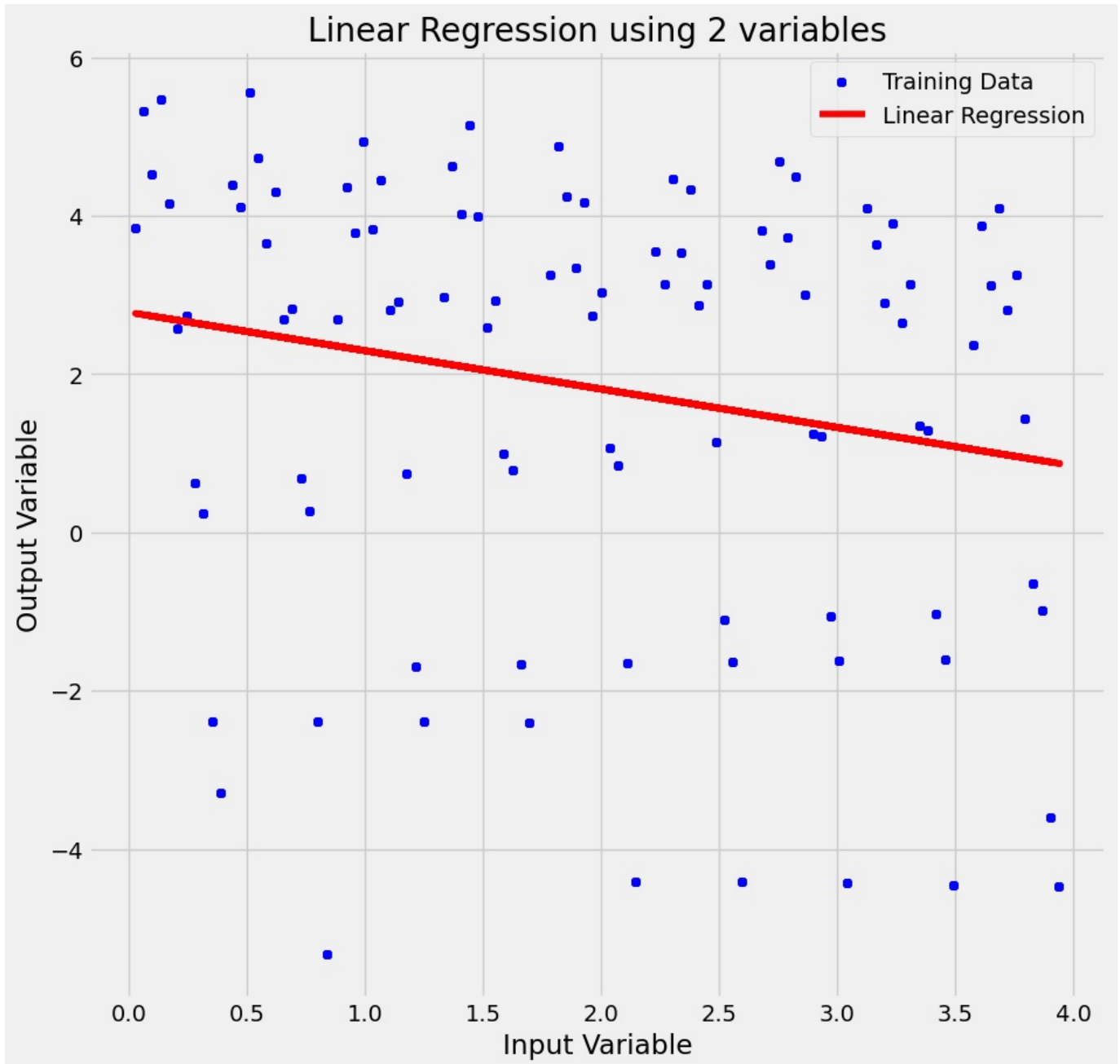


```
In [17]: plt.plot(range(1, iterations + 1),cost_history, color='blue')
         plt.rcParams["figure.figsize"] = (10,6)
         # plt.grid()
         plt.xlabel('Number of iterations')
         plt.ylabel('Cost (J)')
         plt.title('Convergence of gradient descent')
```

Out[17]: Text(0.5, 1.0, 'Convergence of gradient descent')

Convergence of gradient descent

In [ ]:

In [ ]: