In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [18]:
```
df = pd.read_csv('https://raw.githubusercontent.com/satishgunjal/datasets/master/univa
#df = pd.DataFrame(pd.read_csv("UProfits.csv"))
df.head(100) # To get first n rows from the dataset default value of n is 5
```

Out[18]:

	population	profit
0	6.1101	17.59200
1	5.5277	9.13020
2	8.5186	13.66200
3	7.0032	11.85400
4	5.8598	6.82330
...	...	...
92	5.8707	7.20290
93	5.3054	1.98690
94	8.2934	0.14454
95	13.3940	9.05510
96	5.4369	0.61705

97 rows × 2 columns

In [19]:
```
len(df)
```

Out[19]:
```
97
```

In [20]:
```
dataset = df.values[:,:]  # get input values from first column
print('dataset = ', dataset[:10,:])
```

```
dataset =  [[ 6.1101 17.592 ]
 [ 5.5277  9.1302]
 [ 8.5186 13.662 ]
 [ 7.0032 11.854 ]
 [ 5.8598  6.8233]
 [ 8.3829 11.886 ]
 [ 7.4764  4.3483]
 [ 8.5781 12.    ]
 [ 6.4862  6.5987]
 [ 5.0546  3.8166]]
```

In [21]:
```
X = df.values[:, 0]  # get input values from first column
y = df.values[:, 1]  # get output values from second column
len(X), len(y)
```

Out[21]:
```
(97, 97)
```

In [22]:
```
print('X = ', X[: 5]) # Show only first 5 records
```

```
print('y = ', y[: 5])
```

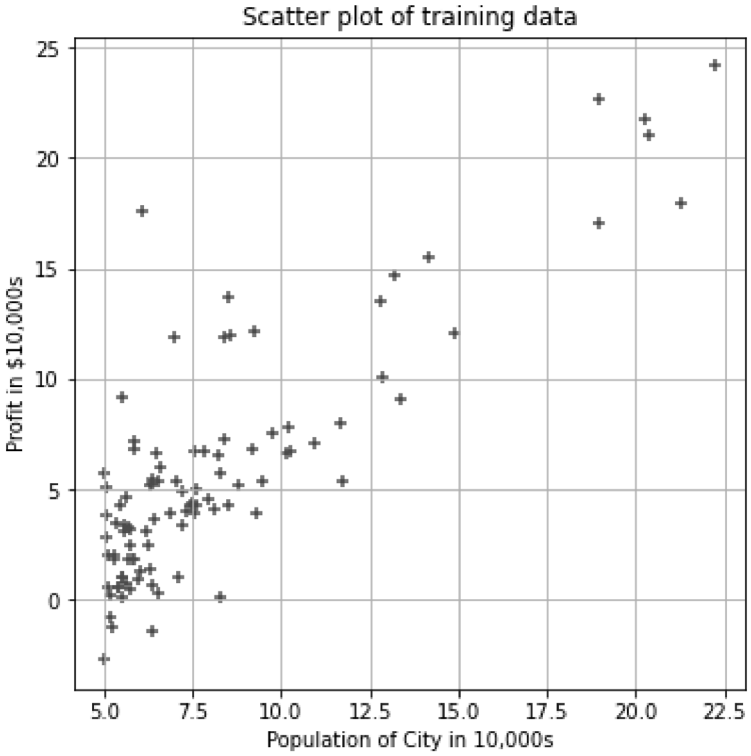
```
X =  [6.1101 5.5277 8.5186 7.0032 5.8598]
y =  [17.592   9.1302 13.662  11.854   6.8233]
```

In [23]:
```
print('X = ', X[:5]) # Show only first 5 records
print('y = ', y[:5])
```

```
X =  [6.1101 5.5277 8.5186 7.0032 5.8598]
y =  [17.592   9.1302 13.662  11.854   6.8233]
```

In [24]:
```
plt.scatter(X,y, color='red',marker= '+')
plt.grid()
plt.rcParams["figure.figsize"] = (6,6)
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
plt.title('Scatter plot of training data')
```

Out[24]:
```
Text(0.5, 1.0, 'Scatter plot of training data')
```



In [25]:
```
# Using reshape function convert X 1D array to 2D array of dimension 97x1
m=len(X)
X_1 = X.reshape(m, 1)
print('X_1 = ', X_1[:5,:]) # Show only first 5 records
```

```
X_1 =  [[6.1101]
 [5.5277]
 [8.5186]
 [7.0032]
 [5.8598]]
```

In [26]:
```
#Lets create a matrix with single column of ones
m=len(X)
#X_0 = np.array((m, 1))
X_0 = np.ones((m, 1))
X_0[:5], len(X_0)
```

Out[26]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 97)
```

In [27]:
```
# Lets use hstack() function from numpy to stack X_0 and X_1 horizontally (i.e. columr
# This will be our final X matrix (feature matrix)
X = np.hstack((X_0, X_1))
X[:5]
```

Out[27]:
```
array([[1.    , 6.1101],
       [1.    , 5.5277],
       [1.    , 8.5186],
       [1.    , 7.0032],
       [1.    , 5.8598]])
```

In [28]:
```
theta = np.zeros((2,1))
theta
```

Out[28]:
```
array([[0.],
       [0.]])
```

In [29]:
```
def compute_loss(X, y, theta):
  """
  Compute loss 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)

  theta : 2D array of fitting parameters or weights. Dimension (n,1)

  Output Parameters
  -----------------
  J : Scalar value : Loss
  """
  predictions = X.dot(theta)  #prediction = h
  errors = np.subtract(predictions, y)
  sqrErrors = np.square(errors)
  J = 1 / (2 * m) * np.sum(sqrErrors)

  return J
```

In [30]:
```
# Lets compute the cost for theta values
cost = compute_loss(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 = 3111.0551861132
```

In [33]:
```
def gradient_descent(X, y, theta, alpha, iterations):
  """
  Compute cost for linear regression.

  Input Parameters
  ----------------
```

```
  X : 2D, Dimension(m x n)
      m= number of training data point
      n= number of features (including X_0 column of ones)
  y : 1D array of labels/target value for each traing data point. dimension(m)
  theta : 2D array of fitting parameters or weights. Dimension (n,1)
  alpha : Learning rate. Scalar value
  iterations: Number of iterations. Scalar value.

  Output Parameters
  -----------------
  theta : Final Value. 2D array of fitting parameters or weights. Dimension (n,1)
  Loss-history: Conatins value of cost for each iteration. 1D array. Dimansion(m)
  """
  loss_history = np.zeros(iterations)

  for i in range(iterations):
    predictions = X.dot(theta) #prediction (M,1)
    errors = np.subtract(predictions, y) #Error (M,1)  = temp
    sum_delta = (alpha / m) * X.transpose().dot(errors); #sum_delta (n,1)
    theta = theta - sum_delta; #theta (n,1)
    loss_history[i] = compute_loss(X, y, theta)

  return theta, loss_history
```

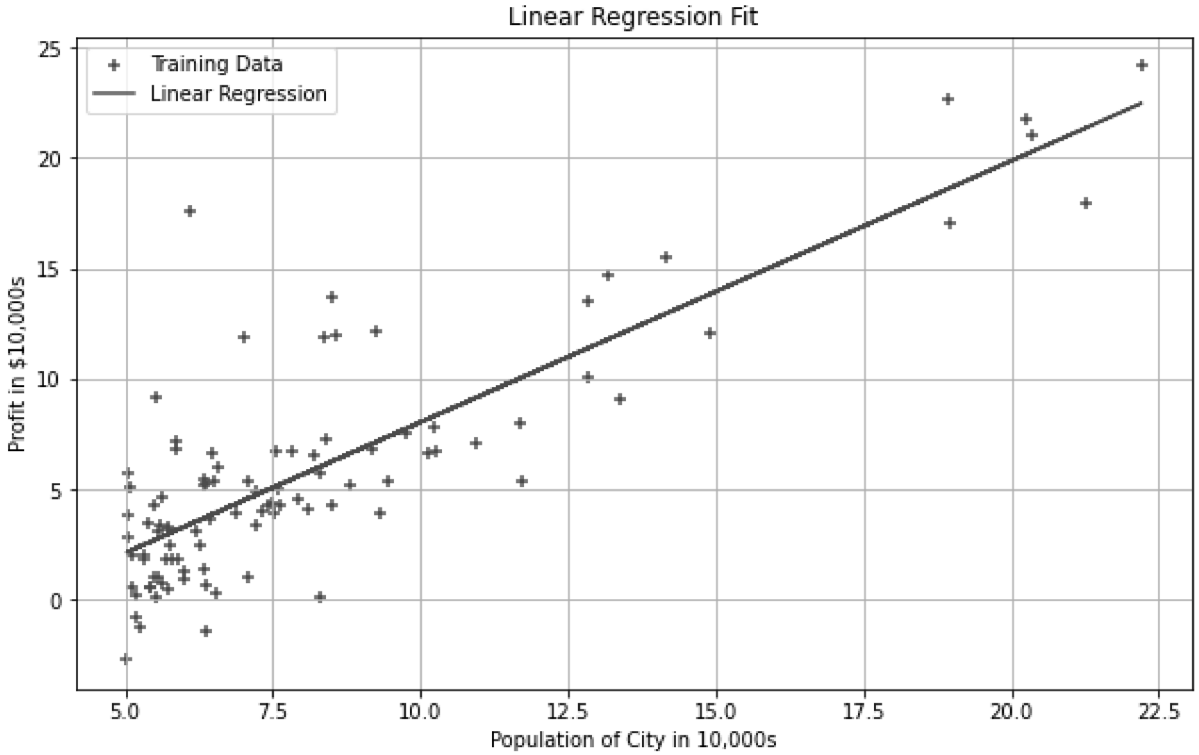
In [51]:
```
theta = [0., 0.]
iterations = 2000;
alpha = 0.01;
```

In [52]:
```
theta, loss_history = gradient_descent(X, y, theta, alpha, iterations)
print('Final value of theta =', theta)
print('cost_history =', loss_history)
```

```
Final value of theta = [-3.78806857  1.18221277]
cost_history = [6.73719046 5.93159357 5.90115471 ... 4.47803526 4.47803143 4.4780276
1]
```

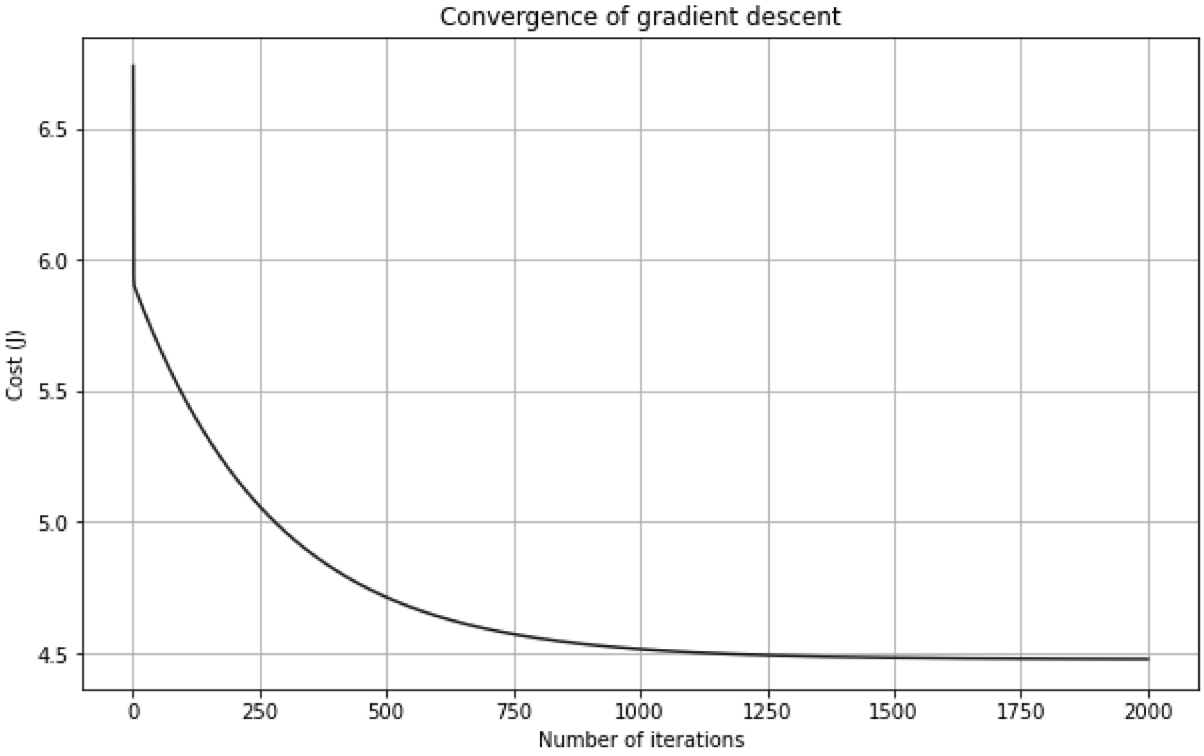
In [53]:
```
# Since X is list of list (feature matrix) lets take values of column of index 1 only
plt.scatter(X[:,1], y, color='red', marker= '+', label= 'Training Data')
plt.plot(X[:,1],X.dot(theta), color='green', label='Linear Regression')

plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
plt.title('Linear Regression Fit')
plt.legend()
```

Out[53]:
```
<matplotlib.legend.Legend at 0x200dd3e2490>
```



In [54]:
```
plt.plot(range(1, iterations + 1),loss_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[54]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



In [ ]: