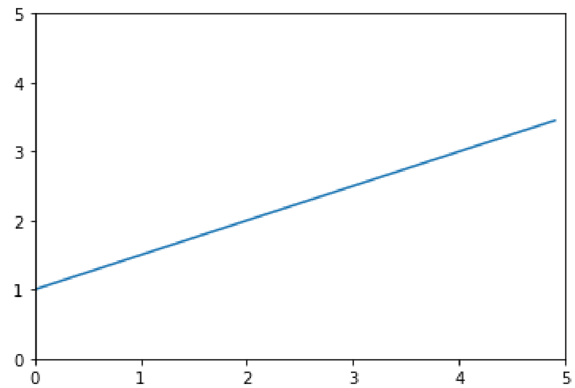
Linear Regression with One Variable

- Model representation:

$$h_\theta(x) = \theta_0 + \theta_1 x$$



- Cost function intuition:

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$= \frac{1}{2m}\sum_{i=1}^{m}\left(\begin{pmatrix} 1, x^{(1)} \\ 1, x^{(2)} \\ \dots \\ 1, x^{(m)} \end{pmatrix}\begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix} - \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{pmatrix}\right)^2$$

Goal:

$$minimize_{\theta_0,\theta_1} \quad J(\theta_0, \theta_1)$$

- Gradient descent:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta\left(x^{(i)}\right) - y^{(i)})^2$$

$$= \frac{1}{2m}\sum_{i=1}^{m}((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{2m}(\left((\theta_0 + \theta_1 x^{(1)}) - y^{(1)}\right)^2 + \left((\theta_0 + \theta_1 x^{(2)}) - y^{(2)}\right)^2 + \cdots$$
$$+ ((\theta_0 + \theta_1 x^{(m)}) - y^{(m)})^2)$$

$$\rightarrow \frac{\partial J}{\partial \theta_0} = \frac{1}{2m}(2\left((\theta_0 + \theta_1 x^{(1)}) - y^{(1)}\right) + 2\left((\theta_0 + \theta_1 x^{(2)}) - y^{(2)}\right) + \cdots$$
$$+ 2((\theta_0 + \theta_1 x^{(m)}) - y^{(m)}))$$

$$= \frac{1}{m}(\left((\theta_0 + \theta_1 x^{(1)}) - y^{(1)}\right) + \left((\theta_0 + \theta_1 x^{(2)}) - y^{(2)}\right) + \cdots + ((\theta_0 + \theta_1 x^{(m)})$$
$$- y^{(m)}))$$

$$= \frac{1}{m}((h_\theta(x^{(1)}) - y^{(1)}) + (h_\theta(x^{(2)}) - y^{(2)}) + \cdots + (h_\theta(x^{(m)}) - y^{(m)}))$$

$$= \frac{1}{m}\sum_{i=1}^{m}(h_\theta\left(x^{(i)}\right) - y^{(i)})$$

$$= \frac{1}{m}\sum_{i=1}^{m}(\begin{pmatrix} 1, x_1 \\ 1, x_2 \\ \cdots \\ 1, x_m \end{pmatrix}_{m*2} \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix}_{2*1} - \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{pmatrix})$$

$$\rightarrow \frac{\partial J}{\partial \theta_1} = \frac{1}{2m}(2\left(\left(\theta_0 + \theta_1 x^{(1)}\right) - y^{(1)}\right)x^{(1)} + 2\left(\left(\theta_0 + \theta_1 x^{(2)}\right) - y^{(2)}\right)x^{(2)} + \cdots$$
$$+ 2(\left(\theta_0 + \theta_1 x^{(m)}\right) - y^{(m)})x^{(m)})$$

$$= \frac{1}{m}((h_\theta\left(x^{(1)}\right) - y^{(1)})x^{(1)} + \left(h_\theta\left(x^{(2)}\right) - y^{(2)}\right)x^{(2)} + \cdots + (h_\theta\left(x^{(m)}\right)$$
$$- y^{(m)})x^{(m)})$$

$$= \frac{1}{m}\sum_{i=1}^{m}(h_\theta\left(x^{(m)}\right) - y^{(m)})x^{(i)}$$

$$= \frac{1}{m}\sum_{i=1}^{m}(\begin{pmatrix} 1, x_1 \\ 1, x_2 \\ \cdots \\ 1, x_m \end{pmatrix}_{m*2} \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix}_{2*1} - \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{pmatrix}) \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_m \end{pmatrix}$$

Repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1), \;\; for\ j = 0\ and\ j = 1$$

Implementation in Python:

Source data (rows = 97, columns = 2):

X	Y
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

Load data

In [4]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
data1 = pd.read_csv("E:\\Machine Learning\\1\\ex1data1.csv")
```

In [5]:
```
data1.head(10)
```

Out[5]:

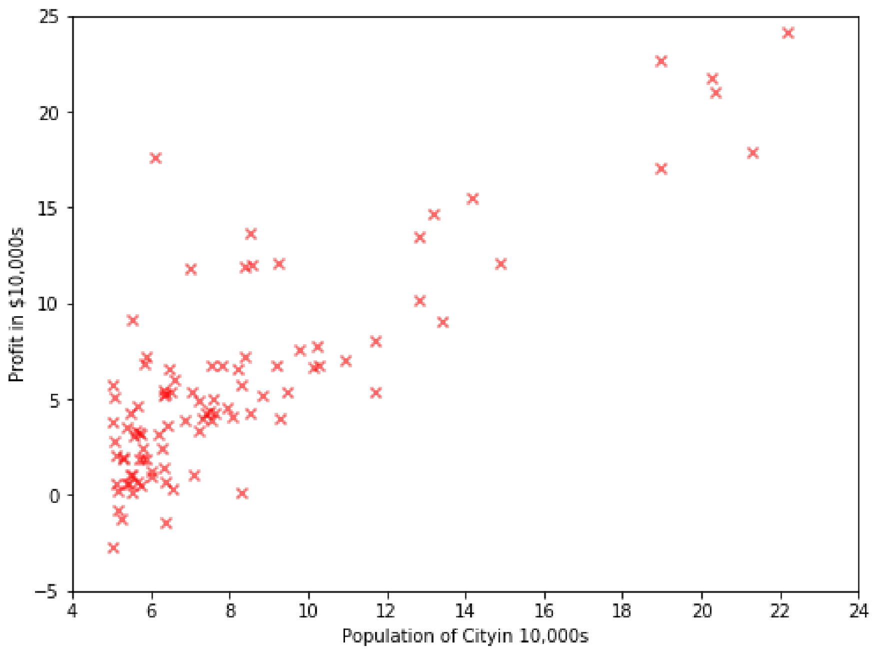
	X	Y
0	6.1101	17.5920
1	5.5277	9.1302
2	8.5186	13.6620
3	7.0032	11.8540
4	5.8598	6.8233
5	8.3829	11.8860
6	7.4764	4.3483
7	8.5781	12.0000
8	6.4862	6.5987
9	5.0546	3.8166

In [6]:
```
type(data1), np.shape(data1)
```

Out[6]:
```
(pandas.core.frame.DataFrame, (97, 2))
```

Visualize data

```
X = np.array(data1["X"])
Y = np.array(data1["Y"])

plt.figure(num = 1, figsize = (8, 6))
plt.scatter(X, Y, marker = "x", color = "red", alpha = 0.6)
plt.xlim((4, 24))
plt.ylim((4, 25))
plt.xticks(range(4, 26, 2))
plt.yticks(range(-5, 30, 5))
plt.xlabel("Population of Cityin 10,000s")
plt.ylabel("Profit in $10,000s")
```



Cost and Gradient Descent

```
X_mat = np.transpose(np.mat(X))
Y_mat = np.transpose(np.mat(Y))

theta_mat = np.transpose(np.mat(np.zeros(2)))
```

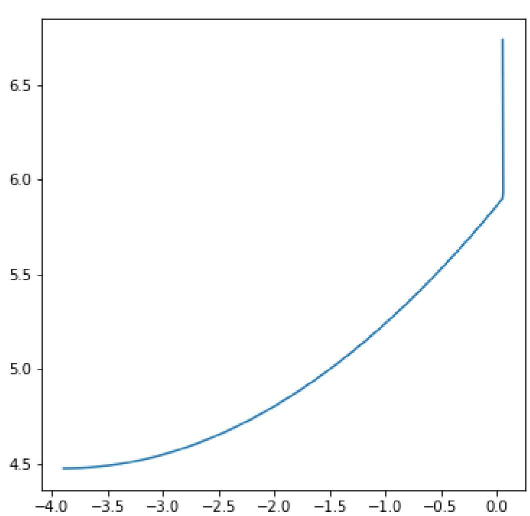
```
def computeCost(X, Y, theta):
    m = np.shape(X)[0]
    ones = np.transpose(np.mat(np.ones(m)))
    X = np.hstack((ones, X))

    J = 1/(2*m) * sum(np.power(X * theta - Y, 2))


    return J[0, 0]
```

```python
def gradientDesc(X, Y, theta, alpha):
    m = np.shape(X)[0]
    ones = np.transpose(np.mat(np.ones(m)))
    X_ones = np.hstack((ones, X))

    G_theta0 = 1/m * sum(X_ones * theta - Y)
    G_theta1 = 1/m * sum(np.multiply(X_ones * theta - Y, X))

    theta0 = theta[0, :][0, 0]
    theta1 = theta[1, :][0, 0]

    theta0_list = []
    theta1_list = []
    J_list = []


    while len(J_list) <= 2 or J_list[len(J_list) - 1] < J_list[len(J_list) - 2]:
        theta0 = theta0 - alpha * G_theta0[0, 0]
        theta1 = theta1 - alpha * G_theta1[0, 0]

        theta0_list.append(theta0)
        theta1_list.append(theta1)
        theta = np.transpose(np.mat([theta0, theta1]))

        G_theta0 = 1/m * sum(X_ones * theta - Y)
        G_theta1 = 1/m * sum(np.multiply(X_ones * theta - Y, X))

        J = computeCost(X, Y, theta)
        J_list.append(J)
```

```python
    return theta0_list, theta1_list, J_list
```

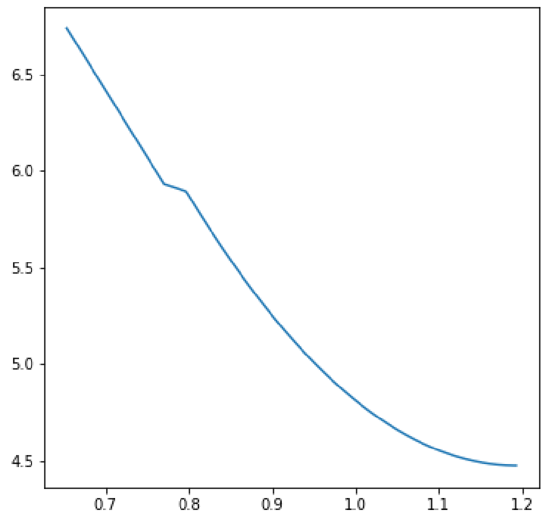
```python
theta0 = gradientDesc(X_mat, Y_mat, np.transpose(np.mat([0, 0])), 0.01)[0]
theta1 = gradientDesc(X_mat, Y_mat, np.transpose(np.mat([0, 0])), 0.01)[1]
J = gradientDesc(X_mat, Y_mat, np.transpose(np.mat([0, 0])), 0.01)[2]
```

```python
plt.figure(num = 1, figsize = (6, 6))

plt.plot(np.array(theta0), np.array(J))
```



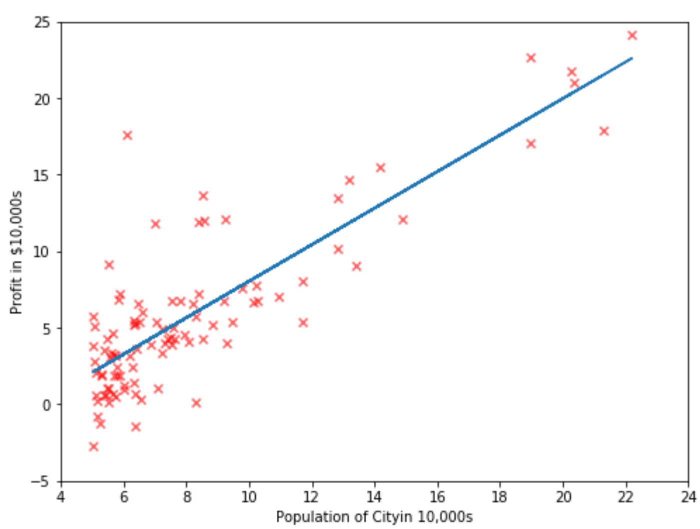
```
plt.figure(num = 1, figsize = (6, 6))

plt.plot(np.array(theta1), np.array(J))
```

```
[<matplotlib.lines.Line2D at 0xe4f9690>]
```



Prediction

```
plt.figure(num = 1, figsize = (8, 6))

plt.scatter(X, Y, marker = "x", color = "red", alpha = 0.6)
plt.plot(X, Y_predict)
plt.xlim((4, 24))
plt.ylim((4, 25))
plt.xticks(range(4, 26, 2))
plt.yticks(range(-5, 30, 5))
plt.xlabel("Population of Cityin 10,000s")
plt.ylabel("Profit in $10,000s")
```



Case Study --- Predict House Prices (Coursera)