機器學習 HW2

班級:電機三甲

姓名:林士恩

學號:B043011031

一、完整程式碼:

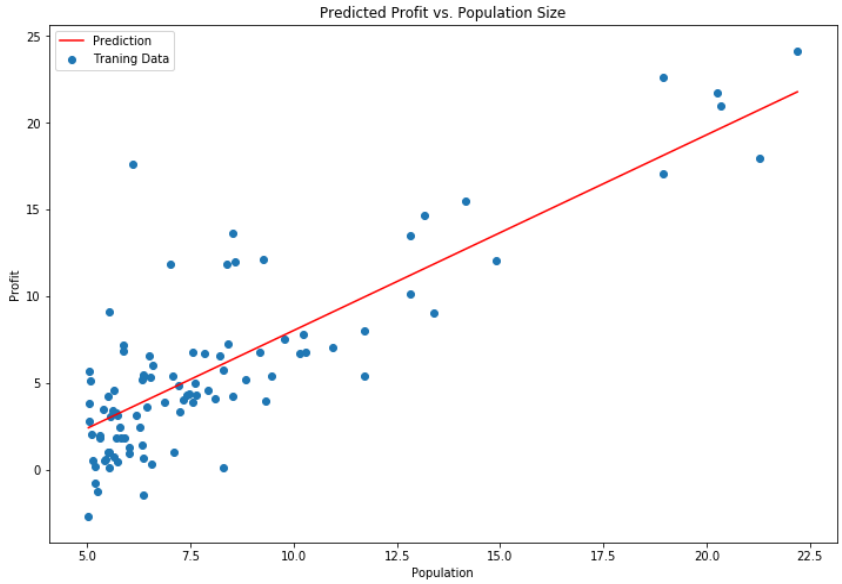
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#func to compute the total cost error
def computeCost(X, y, theta) :    #X represents training sets
    inner = np.power(((X * theta.T) - y), 2)   #inner is also a matrix object (brocasting)
    return np.sum(inner) / (2 * X.shape[0])

def gradientDescent(theta, X, y, alpha, iters) :
    temp = np.matrix(np.zeros(theta.shape))  #numpy.zeros會回傳一個限定維度的array object
    parameters = int(theta.ravel().shape[1])  #matrix.ravel()會回傳個整個攤平的matrix(即row vector)
    cost = np.zeros(iters)
    for i in range(iters):
        error = (X * theta.T) - y       #error為97 x 1
        for j in range(parameters):      #parameters即為3的數量, 進行30, 31的運算
            term = np.multiply(error, X[:,j])    #矩陣乘法, X[:, j]為 97 x 2矩陣的column 0, 1 (bitwise product),
            temp[0,j] = theta[0,j] - ((alpha / len(X)) * np.sum(term))   #temp為 1 x 2 matrix

        theta = temp
        cost[i] = computeCost(X, y, theta)
    return theta, cost

path = 'ex1data1.txt'
data = pd.read_csv(path, header = None, names = ['Population', 'Profit'])
data.head()
data.describe()

#data.plot(kind='scatter', x='Population', y='Profit', figsize=(10, 5))

data.insert(0, 'Ones', 1)

cols = data.shape[1]      #shape is a tuple attribute 代表data維度, shape[1]查看第二維的維度

X = data.iloc[:,0:cols-1]
y = data.iloc[:,cols-1:cols]
X = np.matrix(X.values)    #X, y變成matrix Objects
y = np.matrix(y.values)
theta = np.matrix(np.array([0,0])) #theta 也為1X2維矩陣，值都為零
```

```

alpha = 0.01
iters = 1000
g, cost = gradientDescent(theta, X, y, alpha, iters)
x = np.linspace(data.Population.min(), data.Population.max(), 100)
f = g[0, 0] + (g[0, 1] * x)
fig, ax = plt.subplots(figsize=(12,8))
ax.plot(x, f, 'r', label='Prediction')
ax.scatter(data.Population, data.Profit, label='Traning Data')
ax.legend(loc=2)
ax.setxlabel('Population')
ax.setylabel('Profit')
ax.settitle('Predicted Profit vs. Population Size')

```

其中運算細節也可以寫成這樣

```
        term1 = np.multiply(error, X[:,0])
        temp[0,0] = theta[0,0] - ((alpha / len(X)) * np.sum(term1))
        term2 = np.multiply(error, X[:,1])
        temp[0,1] = theta[0,1] - ((alpha / len(X)) * np.sum(term2))
```

Console:



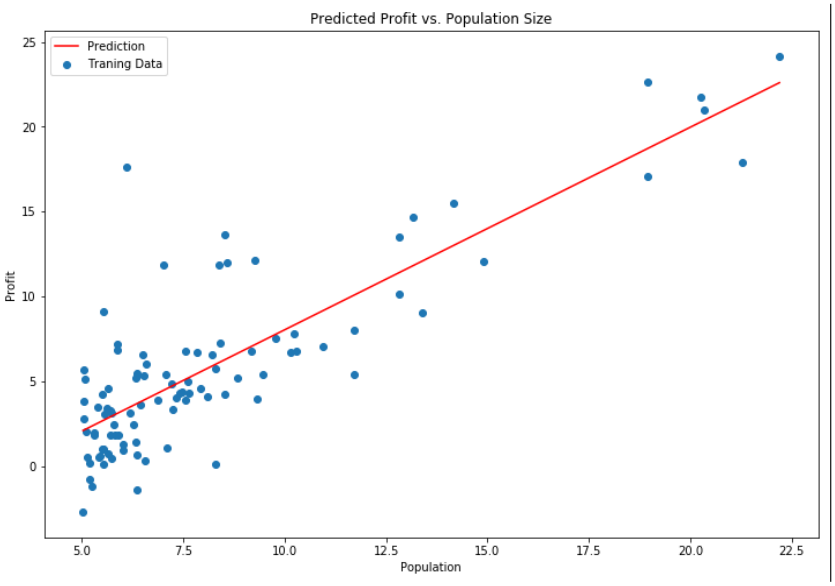
二、問題討論:

1. 加入 Early Stop 機制的影響:

```
def gradientDescent(theta, X, y, alpha, iters) :
    temp = np.matrix(np.zeros(theta.shape))  #numpy.zeros會回傳一個限定維度的array object
    parameters = int(theta.ravel().shape[1])  #matrix.ravel()會回傳個整個攤平的matrix(即row vector)
    cost = np.zeros(iters)
    current = 0
    for i in range(iters):
        error = (X * theta.T) - y       #error為97 x 1
        for j in range(parameters):      #parameters即為θ的數量, 進行θ0, θ1的運算
            term = np.multiply(error, X[:,j])    #矩陣乘法, X[:, j]為 97 x 2矩陣的column 0, 1 (bitwise product)
            temp[0,j] = theta[0,j] - ((alpha / len(X)) * np.sum(term))   #temp為 1 x 2 matrix

        theta = temp
        cost[i] = computeCost(X, y, theta)
        current += 1
        if cost[i - 1] - cost[i] < 10**-10 and (i > 0):
            print("Early Stop at %d iters" % current)
            break
    return theta, cost
```

利用 cost function 值間的差距來判斷是否需要直接停止計算，隨著計算的斜率愈來越小，目前的($\theta 0$, $\theta 1$)與下組($\theta 0$, $\theta 1$)間的距離也越來越小，進而導致 cost function 值的下降程度也趨於緩和，所以到達某個程度時我們就判斷為已經收斂。

```
Early Stop at 4924 iters
θ0: -3.895230 θ1: 1.192978
```

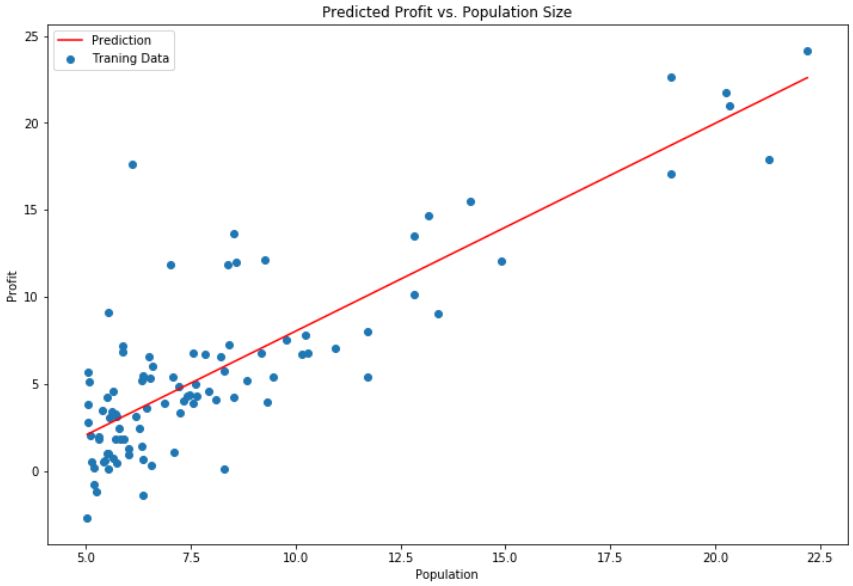



```
alpha = 0.01
iters = 100000
```

由兩圖可以觀察到，如果沒有 Early Stop 的機制之下，程式一定會跑完 100000 次的 iterations；若加入 Early Stop 的機制，在 4924 次的 iterations 即可完成收斂近似

2. 加入 error point 的影響:

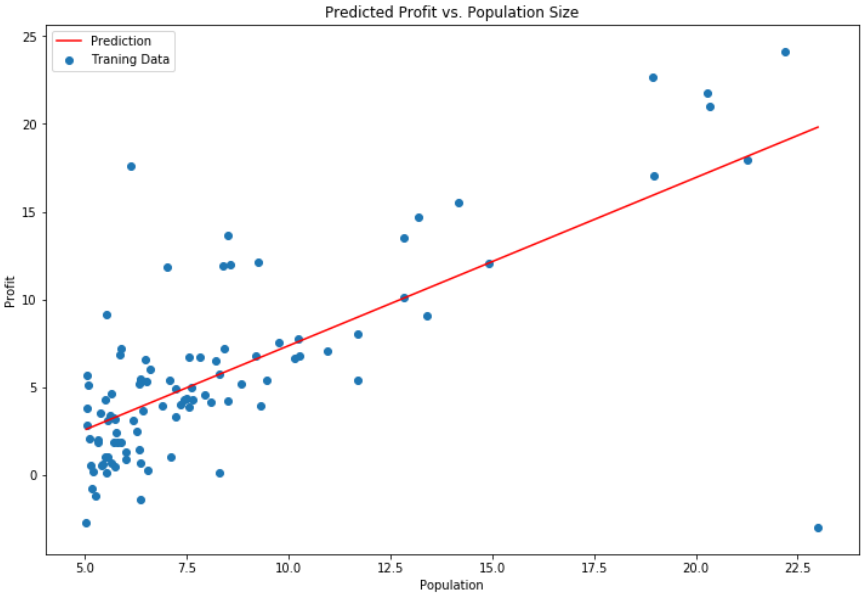
```
Early Stop at 4924 iters
θ0: -3.895230 θ1: 1.192978
```



(上圖沒有加入 error point)

(下圖有加入 error point)

```
Early Stop at 4313 iters
θ0: -2.208904 θ1: 0.957493
```

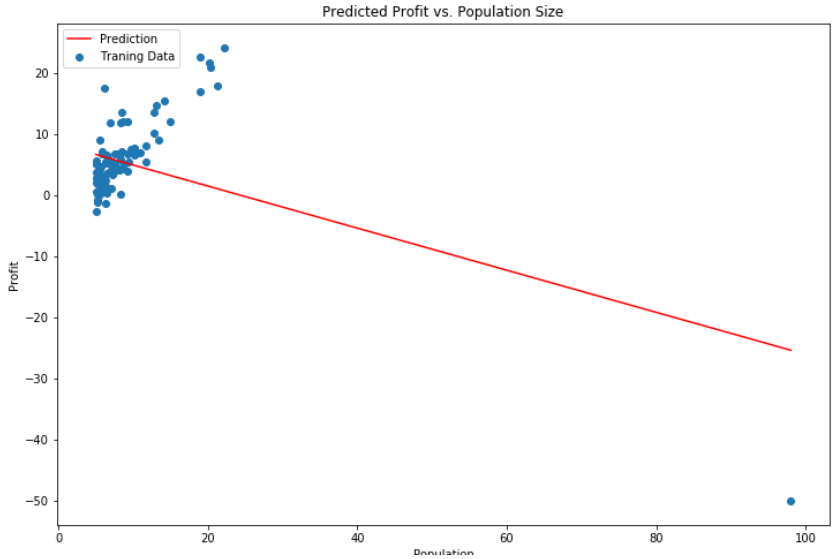


由兩圖可以發現加入 error point 後最後產生的 hypothesis 發生改變(截距與斜率改變)，下圖的 hypothesis 較上圖的 hypothesis 往 error point 的方向傾斜，若把 error point 放在更極端的位置，如下

```
33 data = data.append({'Population': 98, 'Profit': -50}, ignore_index = True)
```

產生的結果如下:

```
Early Stop at 1991 iters
θ0: 8.398656 θ1: -0.344780
```



可以從中發現，因(98, -50)的緣故，hypothesis 變得更不一樣，是因為其他的資料較(98, -50)靠左上方，但(98, -50)對於 cost function 的權重貢獻最大，使得 hypothesis 變成這樣。

3. 為什麼要標準化:

```
67 model = linear_model.LinearRegression(normalize = True)
68 model.fit(X, y)
```

標準化的意義在於，把所有 training sets 壓縮成平均值只有 0，標準差只有 1(standard deviation normalization)，原因在於假設今天有一群資料(x, y, z)，而 hypothesis 為 ax + by = z，而 x 介於-100 到 100，y 介於 0~1 之間，可以發現只要 a 變動一點點，會大大地影響之後預測的結果(因 x 的權重較大)，而 y 的權重較小，對於最後結果的影響相較於 x 不大，進而導致在訓練中效率不高，但只要利用標準化，讓資料都分布在一特定範圍內，能使收斂速度加快，增加預測的精確度。