**Machine Learning HW 3 Report**

組員：0516025張智閔、0516032邱繼聖、0516049吳柏劭、0516215林亮穎、0516220李元毓

1. **What environments the members are using?**

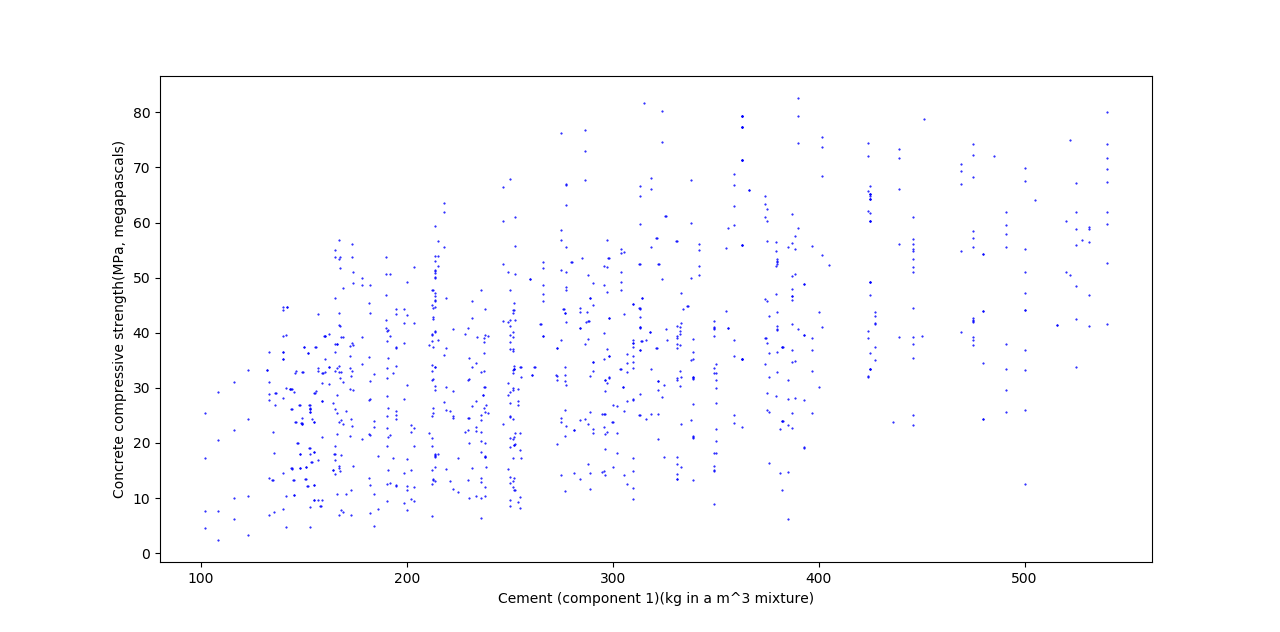
OS: Window 10

Language: Python

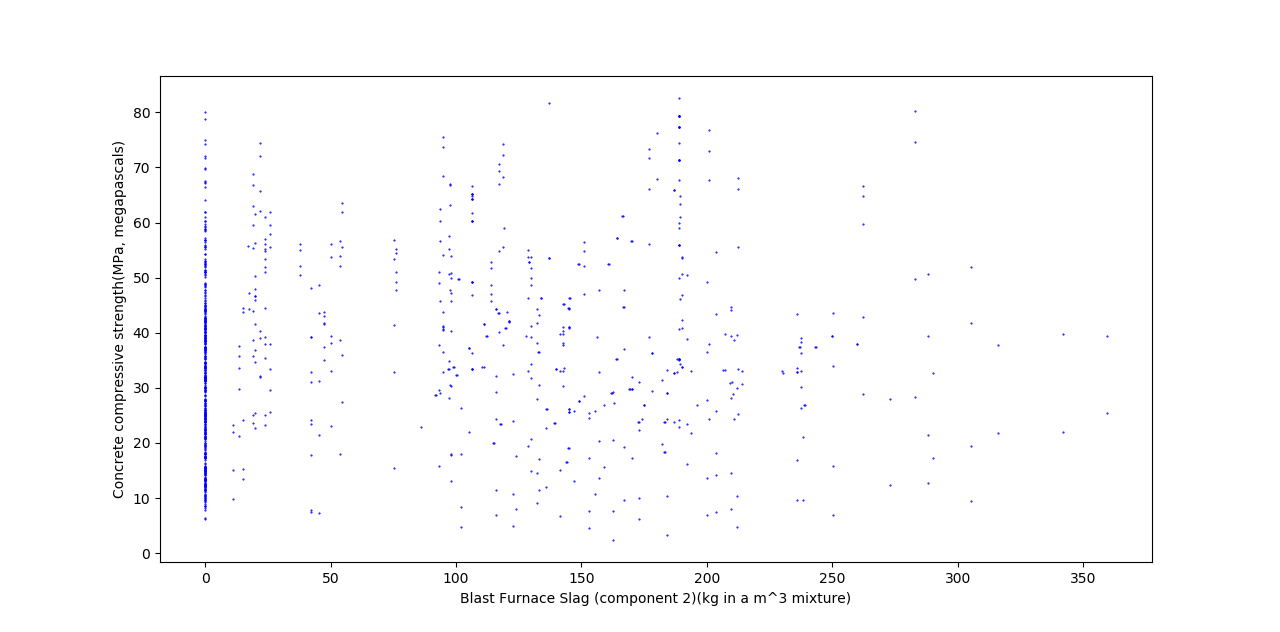
Packages:

1. numpy 1.15.2
2. pandas 0.23.4
3. scikit-learn 0.20.0
4. matplotlib 3.0.0
5. **Visualization of all the features with the target**

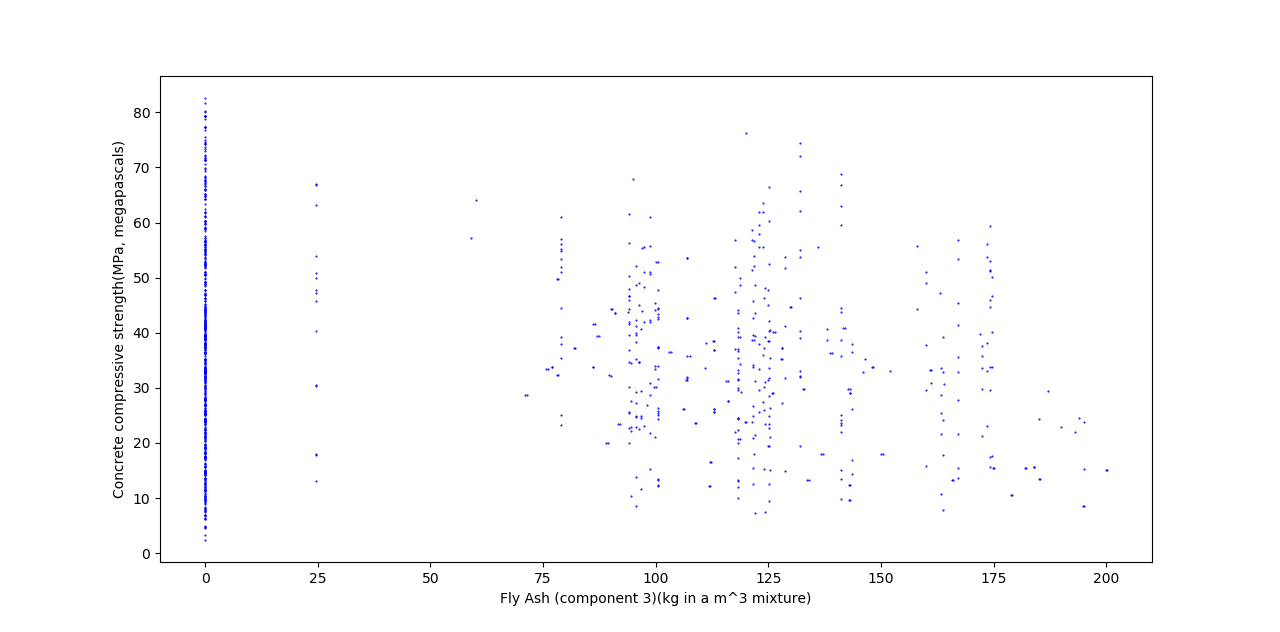
x--Cement y--Concrete compressive strength



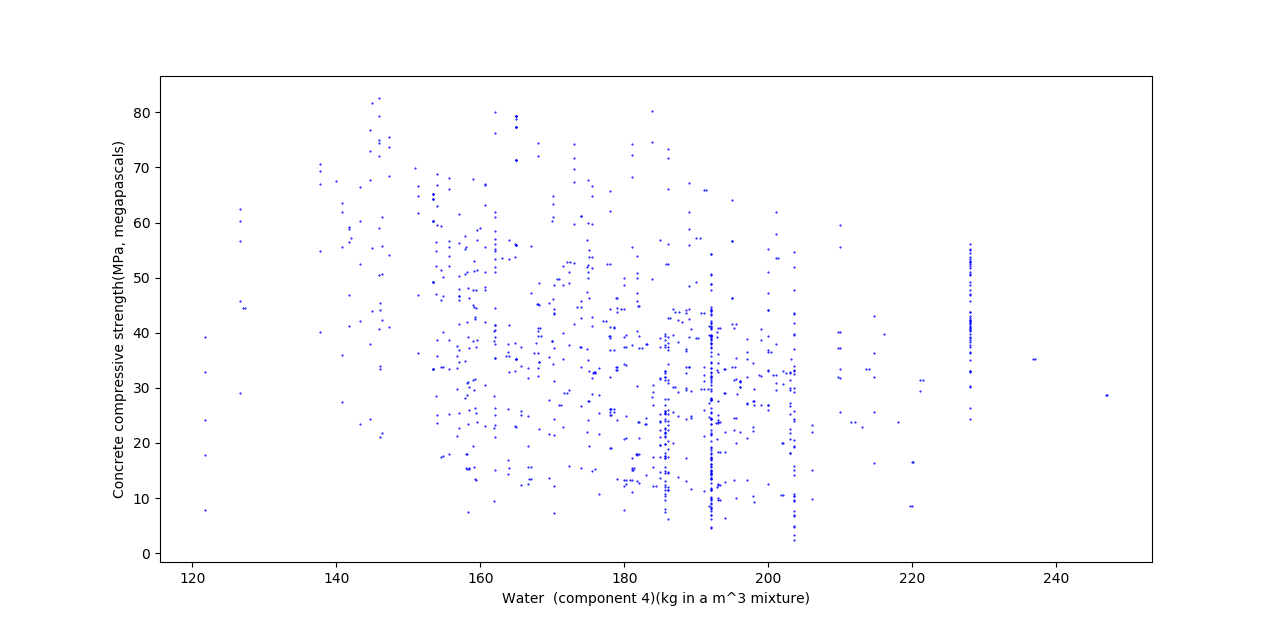
x-- Blast Furnace Slag y--Concrete compressive strength



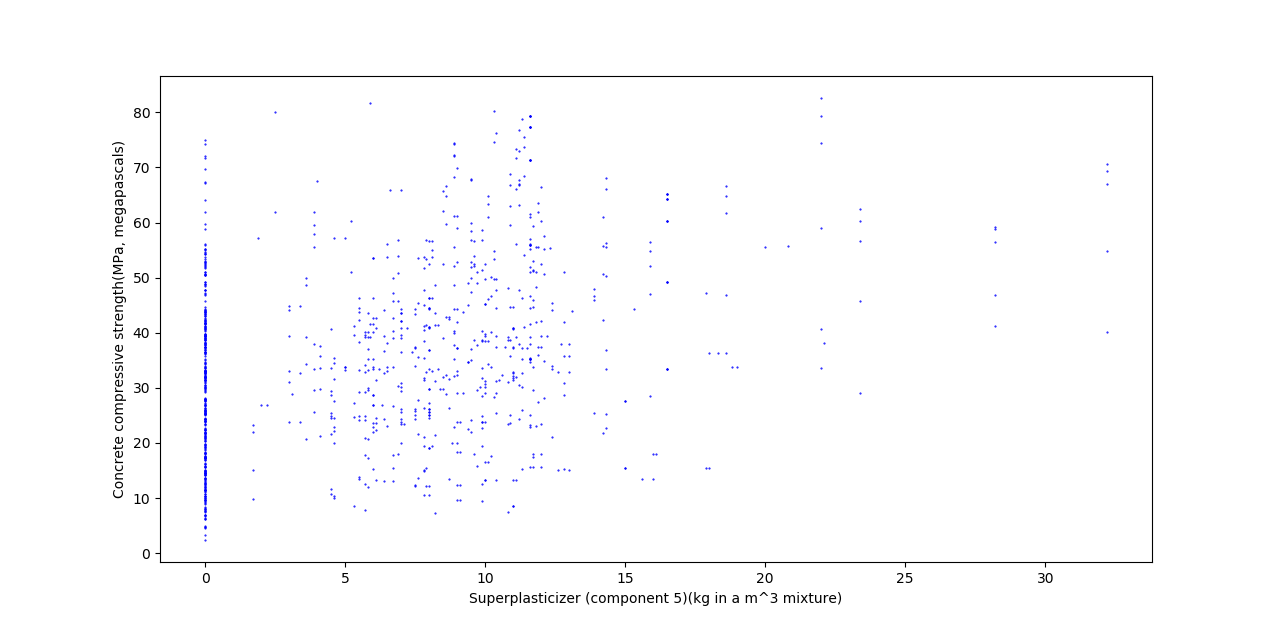
x-- Fly Ash y--Concrete compressive strength



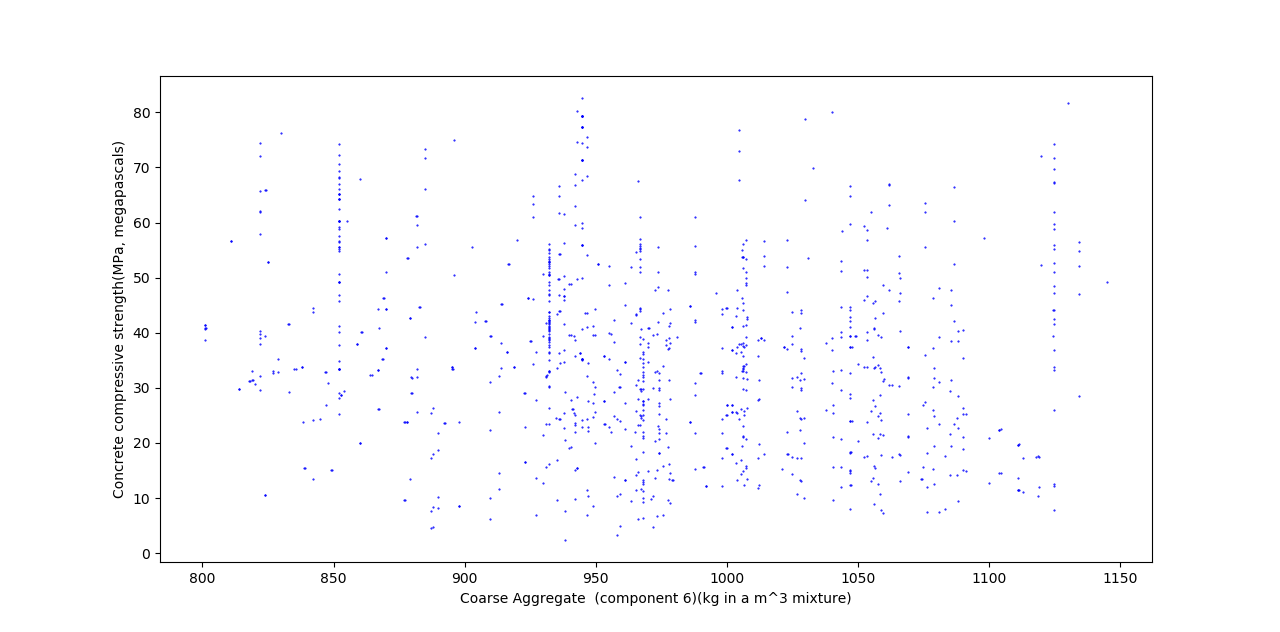
x-- Water y--Concrete compressive strength



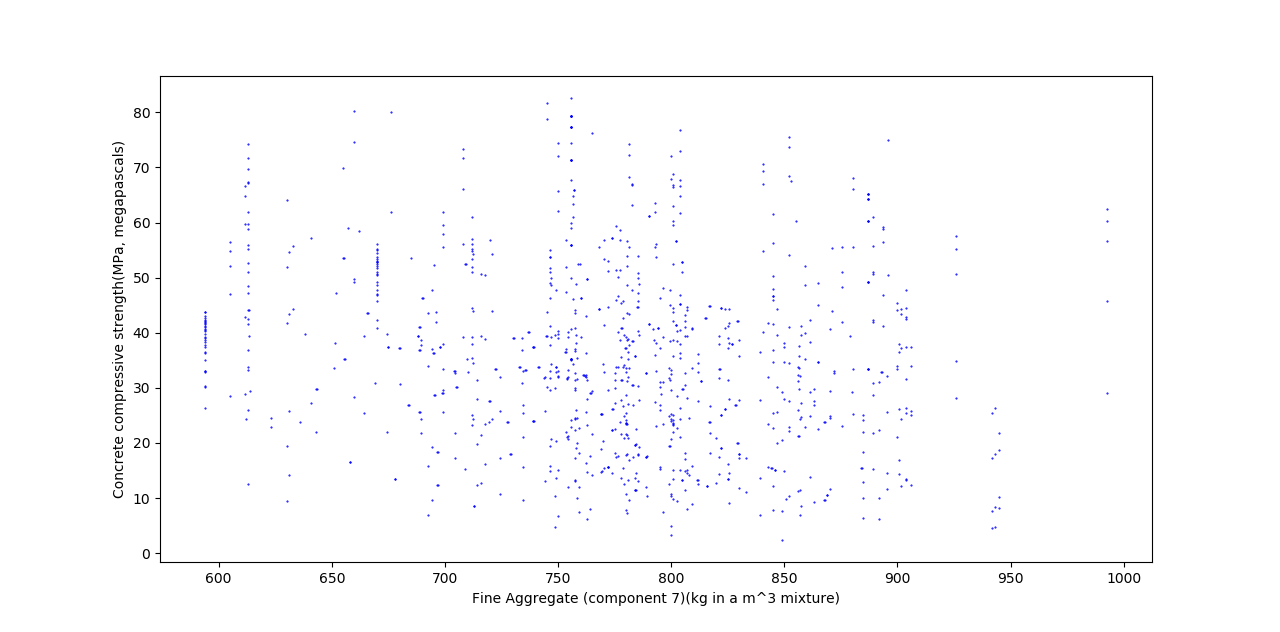
x-- Superplasticizer y--Concrete compressive strength



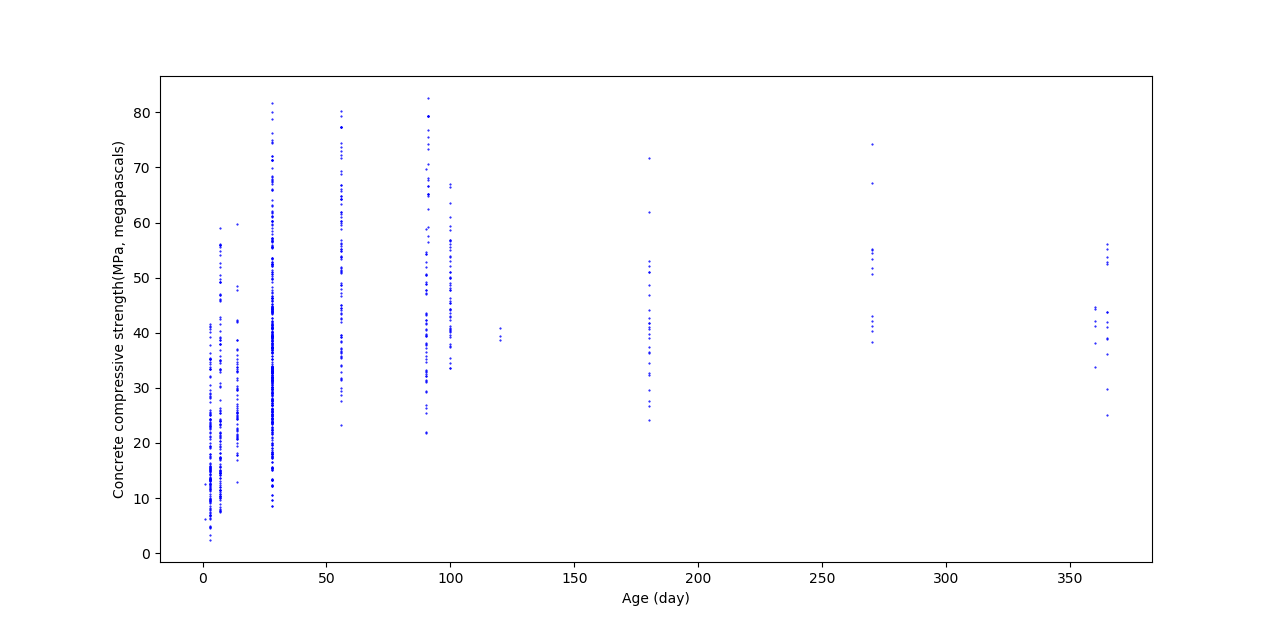
x-- Coarse Aggregate y--Concrete compressive strength



x-- Fine Aggregate y--Concrete compressive strength



x-- Age y--Concrete compressive strength



1. **The code, graph, r2\_score, weight and bias for problem 1**

dataset = pd.read\_csv('Concrete\_Data.csv')

dataset\_df = pd.DataFrame(dataset)

n\_data = 1030

attrs = dataset.columns

attrs = attrs[:len(attrs)-1]

dataset\_df = dataset\_df.sample(n = n\_data).reset\_index(drop = True)

for column in attrs:

print(column)

dataset\_X = dataset\_df[column].values

dataset\_X = np.reshape(dataset\_X, (len(dataset\_X), 1))

dataset\_X = dataset\_X.astype(float)

dataset\_X = preprocessing.scale(dataset\_X)

dataset\_Y = dataset\_df['Concrete compressive strength(MPa, megapascals) '].values

dataset\_Y = np.reshape(dataset\_Y, (len(dataset\_Y), 1))

dataset\_X\_test = dataset\_X[int(-n\_data\*0.2) : ]

dataset\_X\_train = dataset\_X[ : int(-n\_data\*0.2)]

dataset\_Y\_test = dataset\_Y[int(-n\_data\*0.2) : ]

dataset\_Y\_train = dataset\_Y[ : int(-n\_data\*0.2)]

regr = linear\_model.LinearRegression()

regr.fit(dataset\_X\_train, dataset\_Y\_train)

dataset\_Y\_pred = regr.predict(dataset\_X\_test)

print('Weight: ', regr.coef\_)

print('Bias: ', regr.intercept\_)

print("Mean squared error: %.2f" % mean\_squared\_error(dataset\_Y\_test, dataset\_Y\_pred))

print('R2 score: %.2f\n' % r2\_score(dataset\_Y\_test, dataset\_Y\_pred))

將資料讀入後利用sklearn內建的LinearRegression function進行分析

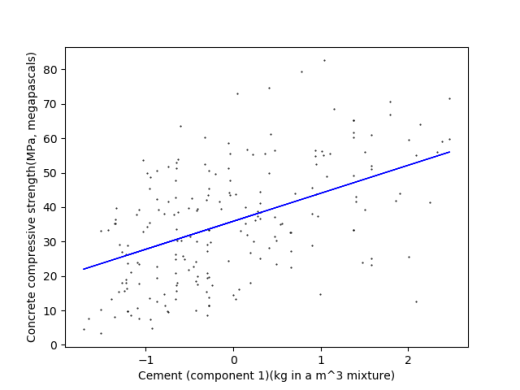
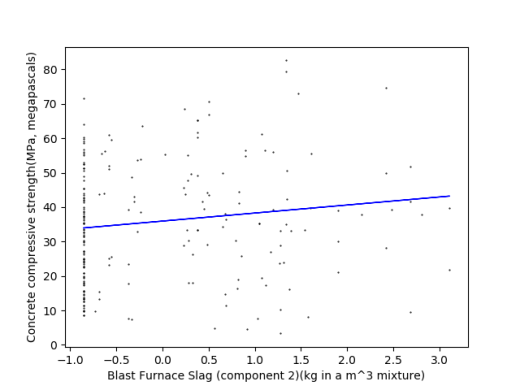
在這邊我們將所有的features都作了normalization，使得每個feature的平均值為0標準差為1，這樣做是為了之後實作gradient descent的時候，learning rate跟 iteration count可以不用取的那麼大。(Problem 2 , 3, 4同樣也對features都作了normalization)

**Cement** Blast Furnace Slag

**Weight: 8.12266927** Weight: 2.33483699

**Bias: 35.89714205** Bias: 35.9546199

**R2\_score: 0.28 (most relative)** R2\_score: 0.01

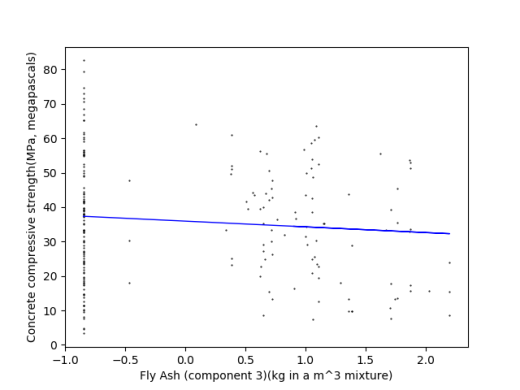
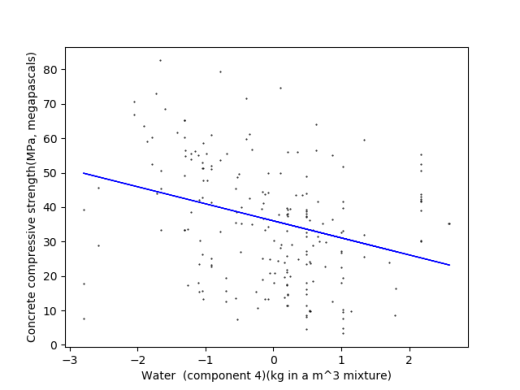
 

Fly Ash Water

Weight: -1.65915058 Weight: -4.94666936

Bias: 35.942211 Bias: 36.02561127

R2\_score: 0.01 R2\_score: 0. 08

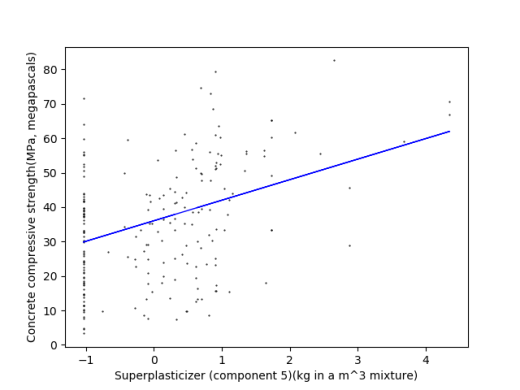
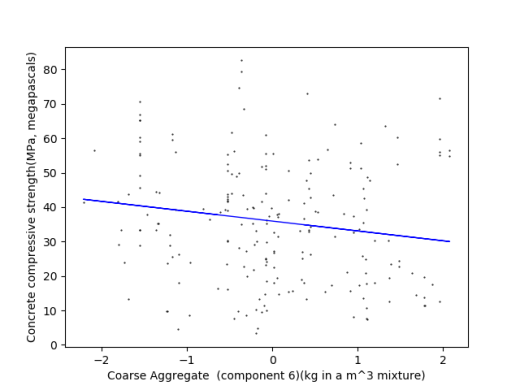
 

Superplasticizer Coarse Aggregate

Weight: 5.96125137 Weight: -2.86218863

Bias: 36.0575923 Bias: 35.95651227

R2\_score: 0.17 R2\_score: 0.02

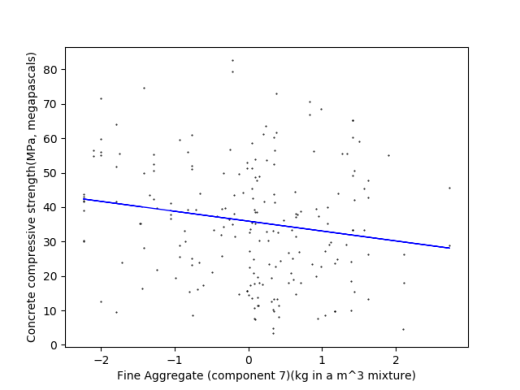
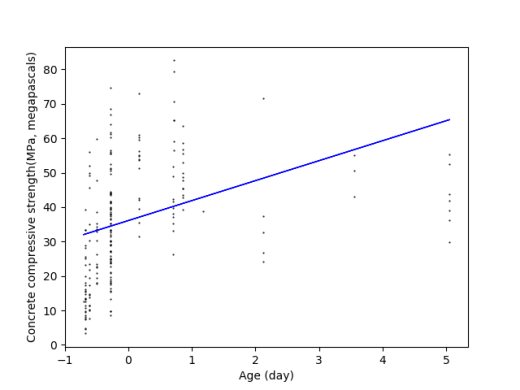
 

Fine Aggregate Age

Weight: -2.86776344 Weight: 5.79482615

Bias: 35.93586509 Bias: 36.10255938

R2\_score: 0.02 R2\_score: 0.10

1. **The code, graph, r2\_score, weight and bias for problem 2**

def CostFunc(X, Y, w):

m = len(Y)

return (1.0/2.0) \* (1.0/m) \* sum((np.dot(X, w) - Y) \*\* 2)

def GradientDescent(X, Y, w, learning\_rate, itr\_limit):

m = len(Y)

n = len(w)

tmp = w[:]

for itr in range(itr\_limit):

for i in range(n):

tmp[i] = tmp[i] - learning\_rate \* (1/m) \* sum( np.dot(np.transpose((np.dot(X, w) - Y)), X[:, i]) )

w = tmp[:]

return w

if \_\_name\_\_ == '\_\_main\_\_':

dataset = pd.read\_csv('Concrete\_Data.csv')

dataset\_df = pd.DataFrame(dataset)

attrs = dataset.columns

attrs = attrs[:len(attrs)-1]

n\_data = 1030

dataset\_df = dataset\_df.sample(n = n\_data).reset\_index(drop = True)

for column in attrs:

dataset\_X = dataset\_df[column].values

dataset\_X = np.reshape(dataset\_X, (len(dataset\_X), 1))

dataset\_X = dataset\_X.astype(float)

dataset\_X = preprocessing.scale(dataset\_X)

dataset\_Y = dataset\_df['Concrete compressive strength(MPa, megapascals) '].values

dataset\_Y = np.reshape(dataset\_Y, (len(dataset\_Y), 1))

X\_test = dataset\_X[int(-n\_data\*0.2) : ]

X\_train = dataset\_X[ : int(-n\_data\*0.2)]

Y\_test = dataset\_Y[int(-n\_data\*0.2) : ]

Y\_train = dataset\_Y[ : int(-n\_data\*0.2)]

one\_train = np.ones((len(X\_train), 1))

X\_train\_padded = np.concatenate((one\_train, X\_train), axis=1)

w = np.array([[0.2], [0.2]])

iterations = 100000

alpha = 0.0001

print(column)

w = GradientDescent(X\_train\_padded, Y\_train, w, alpha, iterations)

print("Bias: %f" %w[0])

print("Weight: %f" %w[1])

one\_test = np.ones((len(X\_test), 1))

X\_test\_padded = np.concatenate((one\_test, X\_test), axis=1)

print("Mean squared error: %f" %CostFunc(X\_test\_padded, Y\_test, w))

Y\_predict = np.dot(X\_test\_padded, w)

print('R2 score: %.2f\n' % r2\_score(Y\_test, Y\_predict))

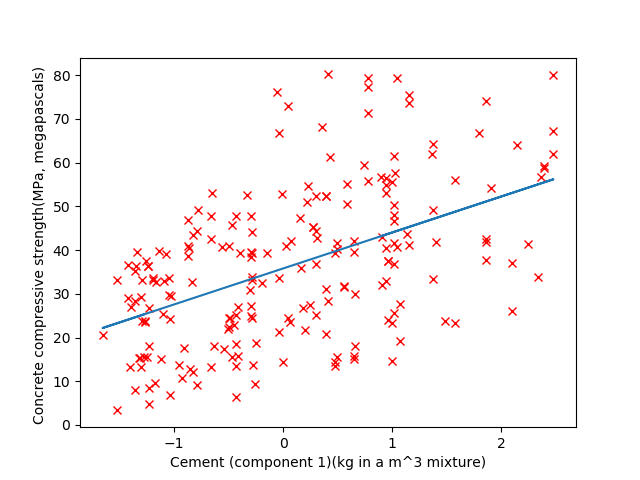
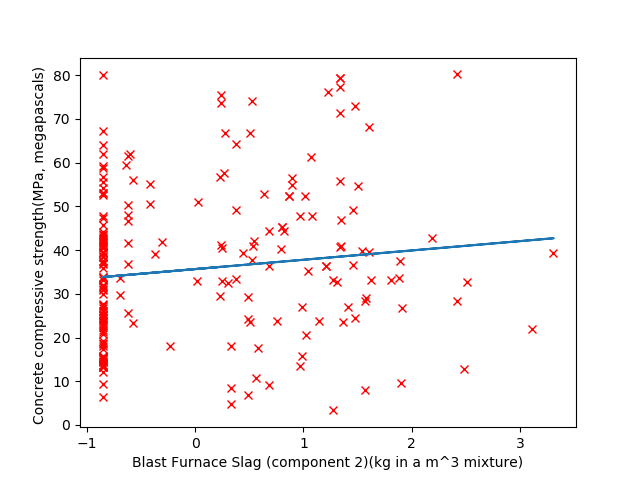
自己實作Gradient Descent，每次iteration都對weight及bias做調整，進行100000次後終止迴圈，取這時的weight和bias作為最終結果

**Cement** Blast Furnace Slag

**Weight: 8.236693** Weight: 2.130658

**Bias: 35.795857**  Bias: 35.656680

**R2\_score: 0.26 (most relative)** R2\_score: 0.02

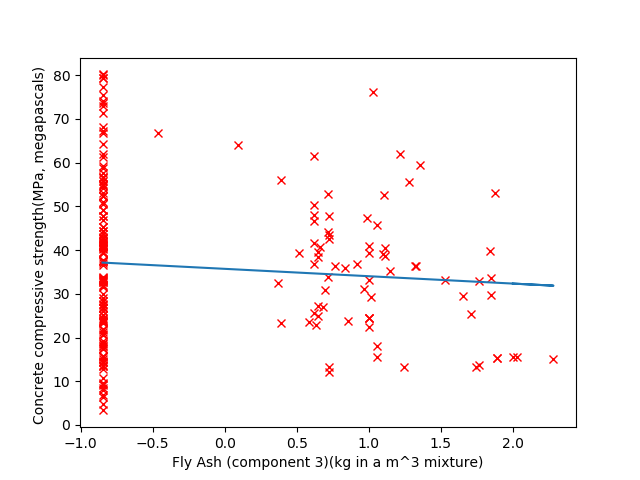
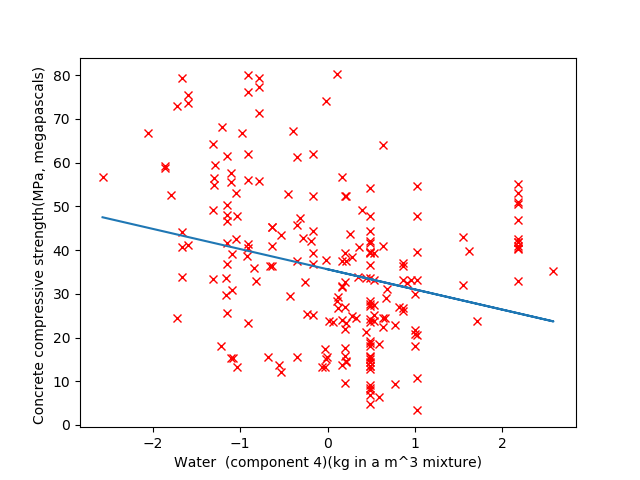
 

Fly Ash Water

Weight: -1.687074 Weight: -4.620336

Bias: 35.711575 Bias: 35.610997

R2\_score: 0.01 R2\_score: 0.10

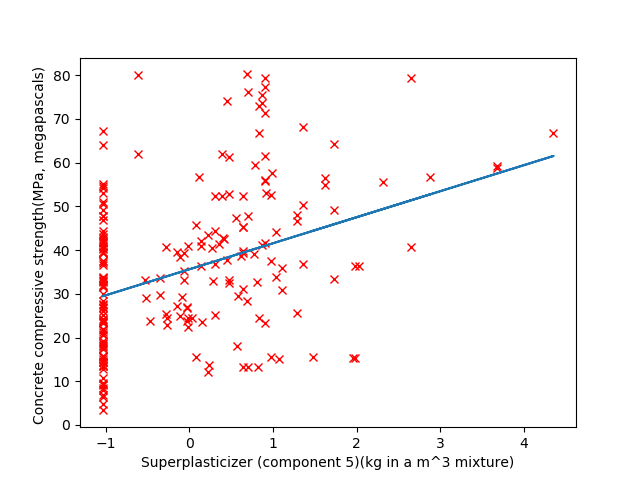
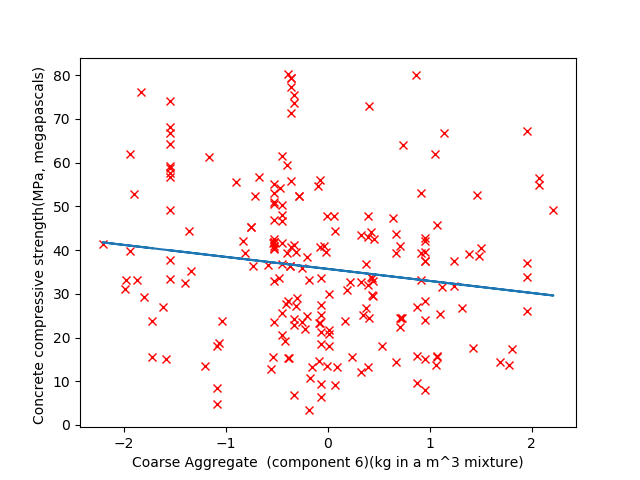
 

Superplasticizer Coarse Aggregate

Weight: 5.953609 Weight: -2.745940

Bias: 35.616071 Bias: 35.688620

R2\_score: 0.17 R2\_score: 0.02

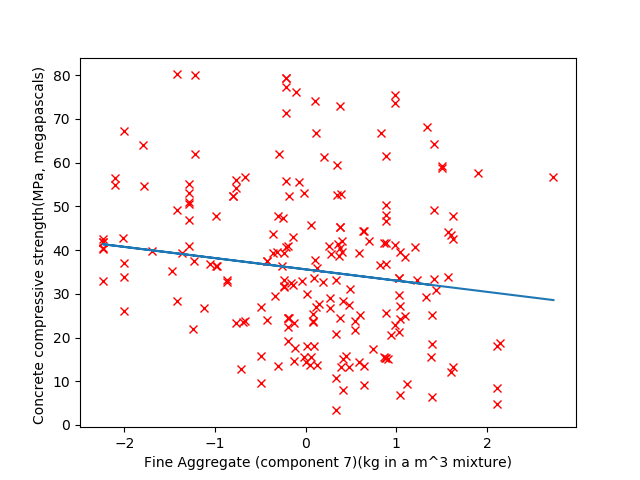
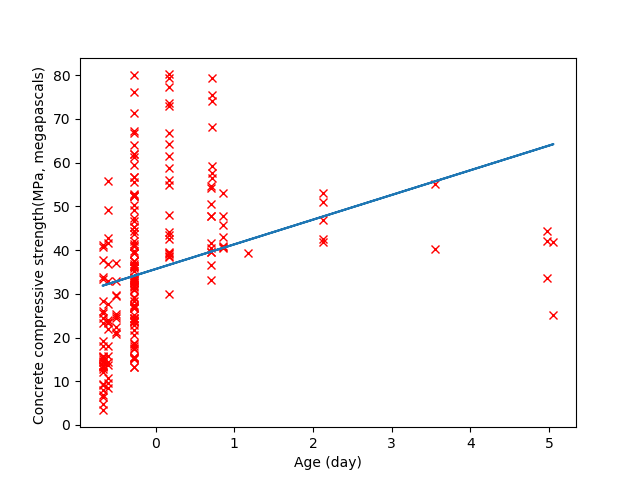
 

Fine Aggregate Age

Weight: -2.572224 Weight: 5.648707

Bias: 35.590758 Bias: 35.669246

R2\_score: 0.04 R2\_score: 0.08

  ****

1. **Compare Problem1 and Problem2**

Problem 1 和Problem 2 所得出的結果十分相近，經過多次測試Cement都是相關度最高的feature, 次之則是Water, Superplasticizer及Age

Problem 1 和Problem 2 最大差異在於執行速度，sklearn 中內建的function有經過相當程度的優化，以提升效率，但我們自己做的如果要加快速度，可能得犧牲準確度，減少Gradient Descent 執行的次數

在Minimum Square Error的部分，Problem2的會比Problem 1的小將近一半，這是因為Problem2在設計cost function時，我們是參照講義上的公式有在前面乘上1/2，所以在計算MSE的時候，會和內建函式算出的MSE差上1/2。(Problem 2 3 4的MSE皆是如此)

1. **The code, MSE, and the r2\_score for problem 3**

def ComputeCost(X, y, theta):

m = y.shape[0]

C = X.dot(theta) - y

J = (C.T.dot(C)) / (2 \* m)

return J

def GradientDescent(X, y, theta, alpha, max\_itrs):

m = y.shape[0]

for itr in range(max\_itrs):

theta = theta - (alpha / m) \* (X.T.dot(X.dot(theta) - y))

return theta

def GradientDescent\_wj(X, y, theta, alpha, max\_itrs):

m = y.shape[0]

n = theta.shape[0]

for itr in range(max\_itrs):

for i in range(n):

Xj = X[:, i]

theta[i] = theta[i] - (alpha / m) \* (Xj.T.dot(X.dot(theta) - y))

return theta

dataset = pd.read\_csv('Concrete\_Data.csv')

dataset\_df = pd.DataFrame(dataset)

n\_data = 1030

attrs = dataset.columns

attrs = attrs[:len(attrs)-1]

dataset\_X = dataset\_df.drop(['Concrete compressive strength(MPa, megapascals) '], axis = 1).values

dataset\_X = preprocessing.scale(dataset\_X)

dataset\_Y = dataset\_df['Concrete compressive strength(MPa, megapascals) '].values

dataset\_Y = np.reshape(dataset\_Y, (-1, 1))

dataset\_X\_train, dataset\_X\_test, dataset\_Y\_train, dataset\_Y\_test = train\_test\_split(dataset\_X, dataset\_Y, test\_size = 0.2)

dataset\_X\_train = np.hstack([dataset\_X\_train, np.ones((dataset\_X\_train.shape[0], 1))])

dataset\_X\_test = np.hstack([dataset\_X\_test, np.ones((dataset\_X\_test.shape[0], 1))])

iterations = 1000

alpha = 0.1

w = np.zeros((dataset\_X\_train.shape[1], 1))

J = ComputeCost(dataset\_X\_train, dataset\_Y\_train, w)

print(J)

print('update w')

print('w1~w8 + Bias:')

w = GradientDescent(dataset\_X\_train, dataset\_Y\_train, w, alpha, iterations)

print(w)

print('MSE:')

J = ComputeCost(dataset\_X\_test, dataset\_Y\_test, w)

print(J[0][0])

dataset\_Y\_pred = dataset\_X\_test.dot(w)

print('R2 score:')

print(r2\_score(dataset\_Y\_test, dataset\_Y\_pred))

print('-----------------------')

w = np.zeros((dataset\_X\_train.shape[1], 1))

print('update wj')

print('w1~w8 + Bias:')

w = GradientDescent\_wj(dataset\_X\_train, dataset\_Y\_train, w, alpha, iterations)

print(w)

print('MSE:')

J = ComputeCost(dataset\_X\_test, dataset\_Y\_test, w)

print(J[0][0])

dataset\_Y\_pred = dataset\_X\_test.dot(w)

print('R2 score:')

print(r2\_score(dataset\_Y\_test, dataset\_Y\_pred))

**(1)each iteration only update wj**

以GradientDescent\_wj()實作每個weight個別更新的方法

每項feature都跑過一次wj = wj + αwj’，且過程中直接將wj做更新，所有feature和bias都做過後便完成一次iteration，結果:

MSE: 56.90626217916698

R2\_score: 0.5554463411798324

**(2)each iteration updates w**

以GradientDescent()實作將weight同步更新的方法

利用上個iteration的weight及bias計算出error後，把所有weight和bais一起更新

w = w + αw’，結果:

MSE: 56.889181662408056

R2\_score: 0.5555797747586503

1. **Compare the performance between two different update method.**

each iteration only update wj

計算完wj’後馬上對wj做更新，每次計算wj’都是用新的w

each iteration updates w

算完w’後再把w更新，每次計算wj’都是用上次iteration後的w

雖然計算完wj’後馬上對wj做更新，每次計算wj’都是用新的w是錯誤的作法，但實際上在測試中，兩種方法的差距很小，不過相較於每次只更新wj，整個w一起更新不但是正確的且方便得多，因為可以直接利用矩陣運算對w做更新，能少做一層for迴圈

1. **The code, MSE, and the r2\_score for problem 4**

def CostFunc(X, Y, w):

m = len(Y)

return (1.0/2.0) \* (1.0/m) \* sum((np.dot(X, w) - Y) \*\* 2)

def GradientDescent(X, Y, w, learning\_rate, itr\_limit, w\_degree):

m = len(Y)

n = len(w)

tmp = w[:]

for itr in range(itr\_limit):

for i in range(n):

Xi\_power = X[:, i] \*\* i

Xi\_power = Xi\_power.reshape(m, 1)

tmp[i] = tmp[i] - learning\_rate \* (1/m) \* sum( np.dot(np.transpose((np.dot(X, w) - Y)), Xi\_power) )

w = tmp[:]

return w

if \_\_name\_\_ == '\_\_main\_\_':

dataset = pd.read\_csv('Concrete\_Data.csv')

dataset\_df = pd.DataFrame(dataset)

attrs = dataset.columns

attrs = attrs[:len(attrs)-1]

n\_data = 1030

for column in attrs:

pass

dataset\_X = dataset\_df['Cement (component 1)(kg in a m^3 mixture)'].values

dataset\_X = np.reshape(dataset\_X, (len(dataset\_X), 1))

dataset\_X = dataset\_X.astype(float)

dataset\_X = preprocessing.scale(dataset\_X)

dataset\_Y = dataset\_df['Concrete compressive strength(MPa, megapascals) '].values

dataset\_Y = np.reshape(dataset\_Y, (len(dataset\_Y), 1))

poly\_degree = 2

poly = preprocessing.PolynomialFeatures(degree=poly\_degree)

poly\_x = poly.fit\_transform(dataset\_X)

X\_test\_original = dataset\_X[int(-n\_data\*0.2) : ]

X\_test = poly\_x[int(-n\_data\*0.2) : ]

X\_train = poly\_x[ : int(-n\_data\*0.2)]

Y\_test = dataset\_Y[int(-n\_data\*0.2) : ]

Y\_train = dataset\_Y[ : int(-n\_data\*0.2)]

w = np.array([0.5]\*(poly\_degree+1))

w = w.reshape(poly\_degree+1, 1)

iterations = 100000

alpha = 0.0001

w = GradientDescent(X\_train, Y\_train, w, alpha, iterations, poly\_degree)

print("Bias: %f" %w[0])

print("Weight: ", end='')

print(w[1:])

print("Mean squared error: %f" %CostFunc(X\_test, Y\_test, w))

Y\_predict = np.dot(X\_test, w)

print('R2 score: %.2f\n' % r2\_score(Y\_test, Y\_predict))

在這一題，我們對單一個feature作了polynomial regression

因為Cement與output的相關度最高

所以我們嘗試令x = {1, x1, x12}做非線性的迴歸分析，結果:

MSE: 45.068461 R2\_score: 0.41

1. **Answer the question**
2. **What is overfitting?**

過於追求model的預測完美對應training set，使得實際導入testing set後的預測效果不佳，以下是示意圖

Overfitting Better one

以本次作業來說的話，在Problem4 polynomial regression時，若是硬要用過高次方的函數來train的話，就有可能產生over fitting，反而使得r2 score降低。

1. **Stochastic gradient descent is also a kind of gradient descent, what is the benefit of using SGD?**

Stochastic gradient descent 在每次iteration中只使用單一個training example來計算error而非整個training set。雖然這會造成它沒辦法收斂到最佳解，但結果也十分接近。當資料量很大時，Stochastic gradient descent 比較方便計算，不須進行龐大的矩陣運算，節省的時間和計算量讓這誤差可以被接受。

1. **Why the different initial value to GD model may cause different result?**

在作多變數的regression時，gradient descent出來的可能會有多個local minimum，

不同的initial value可能使得Gradient Descent找到不同的local minimum，因此會產生其他結果。

1. **What is the bad learning rate? What problem will happen if we use it?**

過大或過小的learning rate，讓model無法逼近最佳解或者花費太多次iteration才得到最佳解。太大的learning rate會使每次的error修正過多，讓得到的結果在最佳解附近震盪，或者是直接發散，無法收斂到local minimum；太小的learning rate使error的修正量過小，得經過更多次的iteration以達到最佳解。

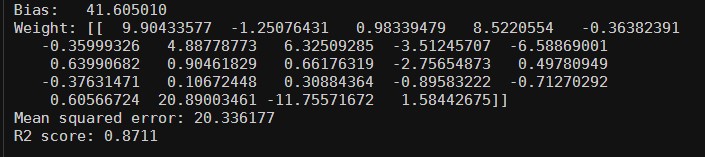
1. **After finishing this homework, what have you learned, what problems you encountered, and how the problems were solved?**

我們學會如何讓程式從資料中自行找出最佳的迴歸函式。

這次的作業中，比較苦惱的問題是在如何找到適當的learning rate值。當設定過大，造成model無法趨近於最佳解。而設定過小則造成需要足夠多次的iteration才能收斂到local minimum。我們當初在實作時社的learning rate太小而且iteration跑的次數不夠多，使得在每一次iteration時，數值的改變量非常小，讓我誤以為已經收斂了，但是收斂到的值卻跟built-in function的結果相差很多，我還花了很多時間檢查model有沒有寫錯，最後才發現是iteration跑的次數不夠多，只要調一下參數就好。

另外，當初因為learning rate設的小，iteration次數必須足夠多才能獲得比較接近的答案，使得每次training都要花上很多時間。後來我們查了資料後，將所有features都作了normalization，這樣feature對regression係數比較敏感，使得 learning rate可以設的大一點，iteration也不需要跑那麼多次，就可收斂到local minimum，可有效改善程式的效能跟執行時間。

1. **Bonus**



在Bonus的部分，我將Problem 4的model改寫為多變數版本的，也就是作多變數的polynomial regression。

參數的部分，我將所有的features都丟入gredient descent，回歸方程式的最高次方設為3，learning rate = 0.007，iterations = 100000，就跑出上圖R2 score = 0.8711 > 0.87的結果。