Machine Learning HW

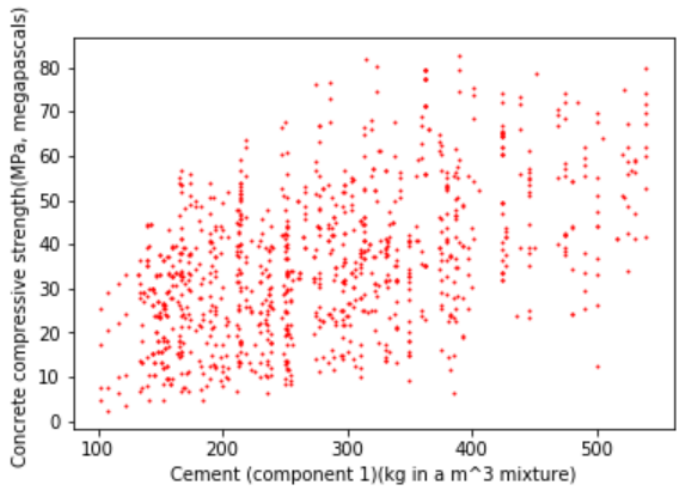
Team Member:
0516017 李柏毅
0516059 劉嘉豪
0516306 尤健羽
0516319 傅信瑀
0516322 朱蝶

1. What environments the members are using:

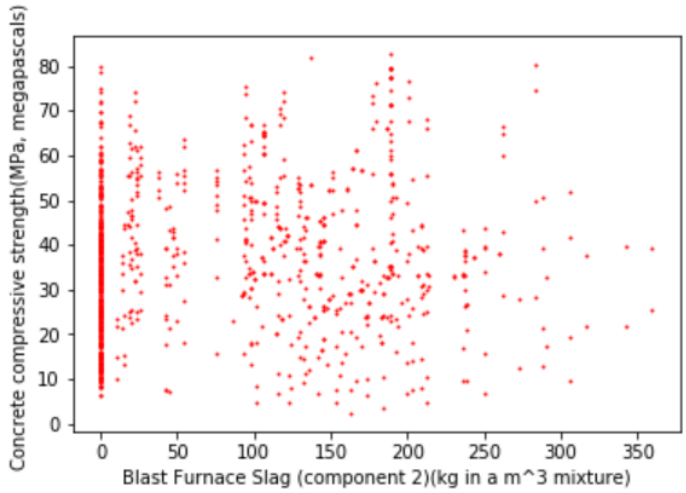
Use Jupyter as the environment

2. Visualization of all the features with the target

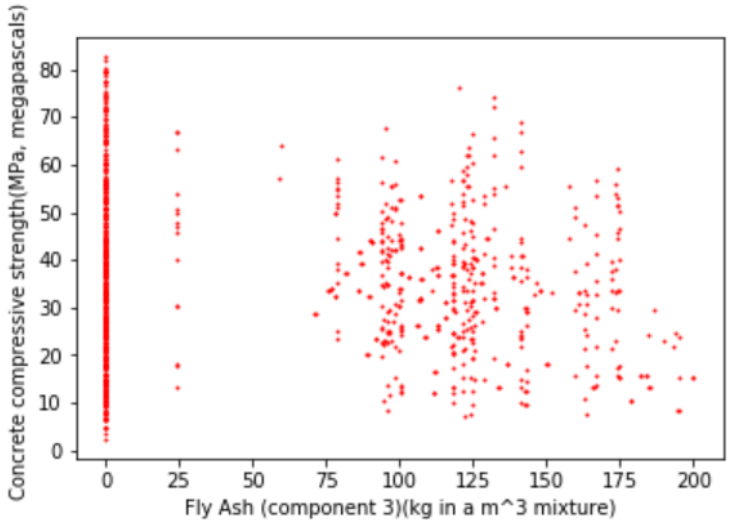
```
feature 0 :target
```



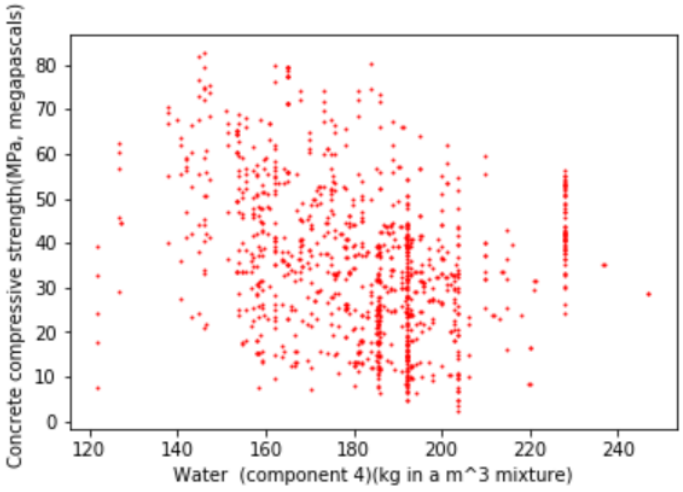

```
feature 1 :target
```



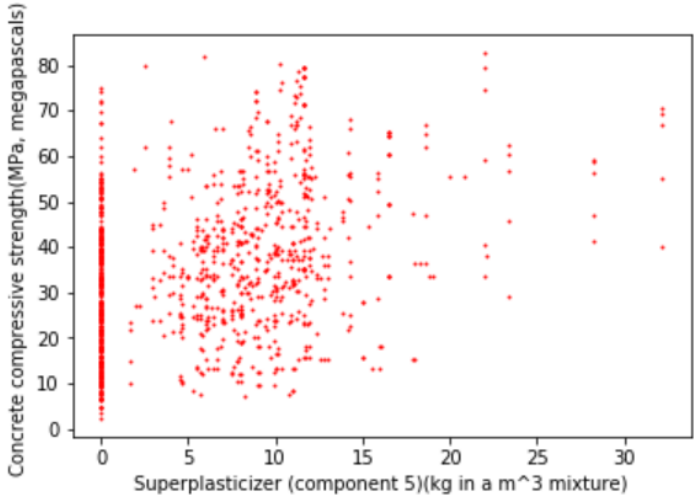
feature 2 :target



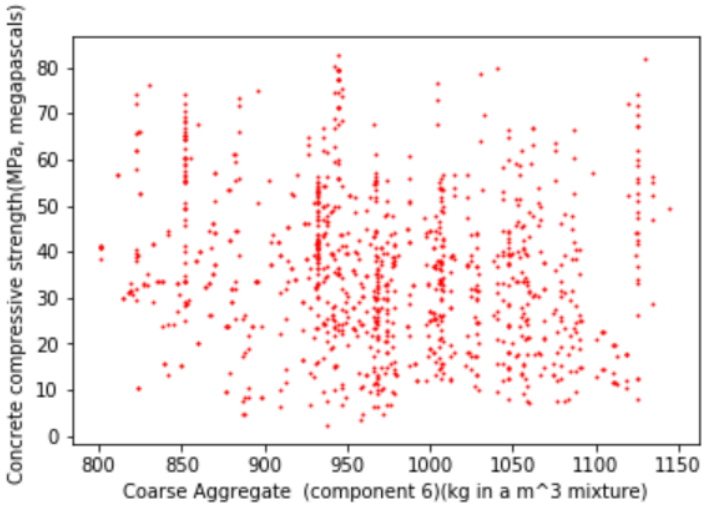
feature 3 :target



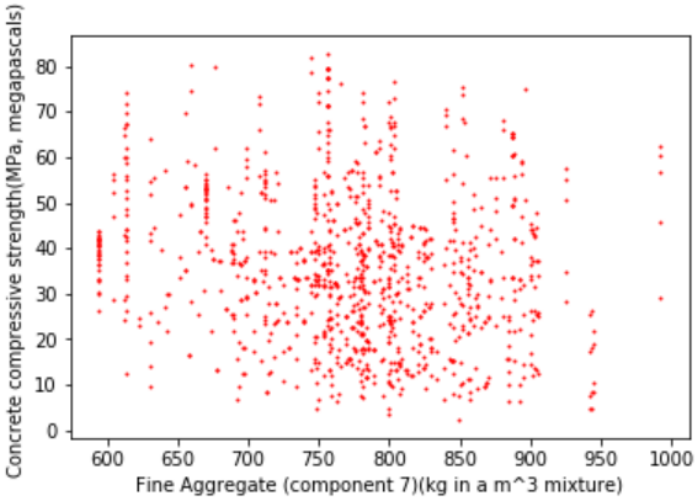
feature 4 :target



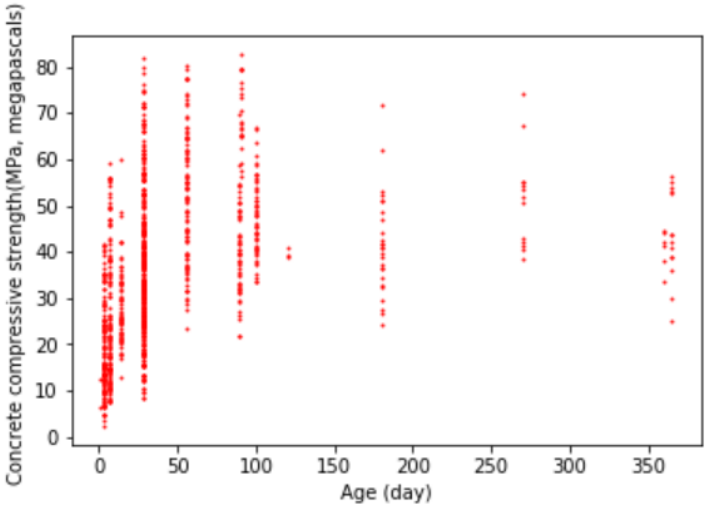
feature 5 :target



feature 6 :target



feature 7 :target



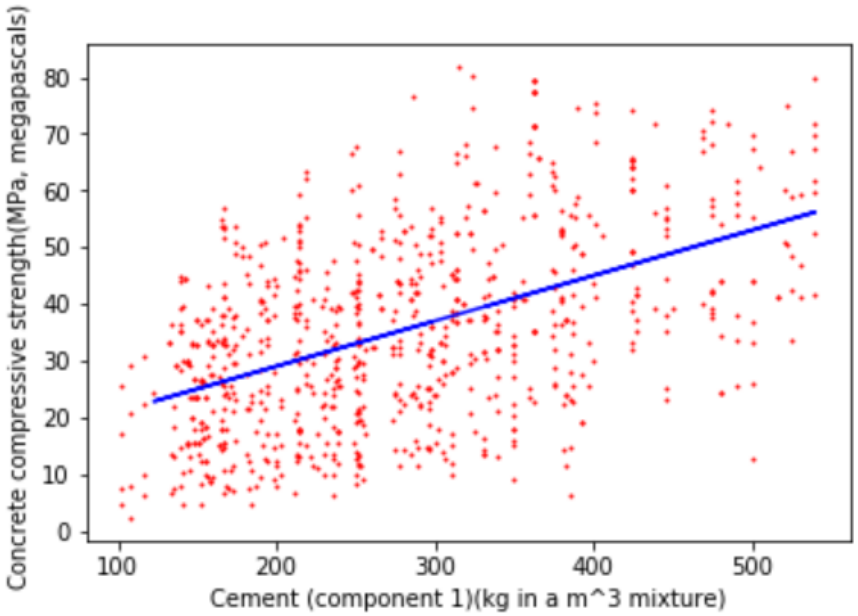
3. Code, graph and attributes for problem 1

Wight 和 bias 初始值設為 0 的是因為由第一題結果的圖看出最適直線是一條斜率為正，並且 bias>0 的線。這樣從 0 開始，每次增加一些，即可較快找到答案。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
#read data
df = pd.read_csv('Concrete_Data.csv')
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

X = df[['Cement (component 1)(kg in a m^3 mixture)']]
y = df[['Concrete compressive strength(MPa, megapascals) ']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)
y_pred = regression_model.predict(X_test)

weight = regression_model.coef_[0][0]
bias = regression_model.intercept_[0]

print("Weight: {}".format(weight))
print("Bias: {}".format(bias))
print("r2_score: {}".format(r2_score(y_test, y_pred)))

plt.scatter(X_train, y_train, c='r', s=1)
plt.plot(X_test,y_pred,c='b')
plt.xlabel(df.columns.values[0])
plt.ylabel(df.columns.values[8])
```

```
Weight: 0.079925383545563
Bias: 13.063163849593124
r2_score: 0.242118137010913
```



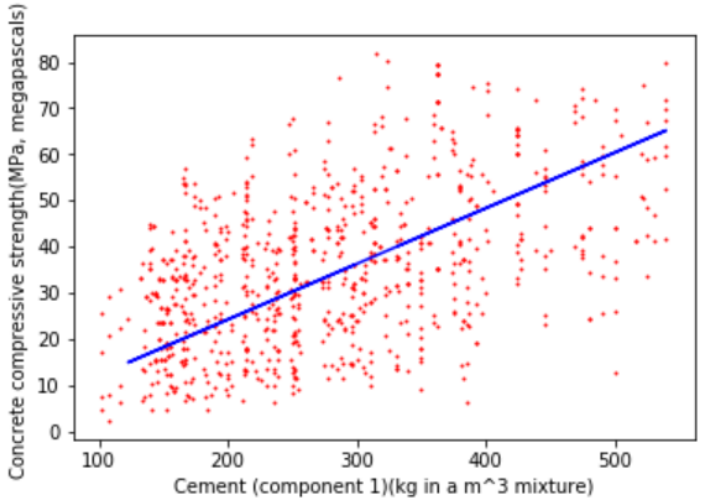
4. Code, graph and attributes for problem 2

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
%matplotlib inline
#read data
df = pd.read_csv('Concrete_Data.csv')
```

```
X = df[['Cement (component 1)(kg in a m^3 mixture)']].values
y = df[['Concrete compressive strength(MPa, megapascals) ']].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
m=0
b=0
h=lambda x : m*x + b
learn=0.00001
N=len(X_train)
```

```
for i in range (1000):
    sum_b, sum_m = 0, 0
    for j in range (N):
        sum_b += h(X_train[j])-y_train[j]
        sum_m += (h(X_train[j])-y_train[j]) * X_train[j]
    sum_b, sum_m = sum_b/N, sum_m/N
    b = b - (0.1 * sum_b)
    m = m - (learn * sum_m)

y_pred = h(X_test)
print("Weight: {}".format(m[0]))
print("Bias: {}".format(b[0]))
print("r2_score: {}".format(r2_score(y_test, y_pred)))

plt.scatter(X_train, y_train, c='r', s=1)
plt.plot(X_test,y_pred,c='b')
plt.xlabel(df.columns.values[0])
plt.ylabel(df.columns.values[8])
```

```
Weight: 0.07992580135699545
Bias: 13.06303241095009
r2_score: 0.24211792543204347
```



5. Compare problem 1 and problem 2

第二題一開始把 weight 和 bias 的 learning rate 設 0.00001，原因是因為 weight 的 sum 太大，會無法收斂，可是 r2_score 只能達到 0.14，低於第一題 0.1。後來發現這樣 bias 的每次的變動會很小，造成 bias 變動速度太慢，不滿足預期。於是把 bias 的 learning rate 改 0.1 後，發現結果則會跟第一題差不多 r2_score 達到 0.24，這個發現讓我了解到針對不同數值大小給予不同的 learning rate 能幫助其增加正確率及收斂。

6. The code,MSE,and the r2_score for problem3

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from random import randint
%matplotlib inline
#read data
df = pd.read_csv('Concrete_Data.csv')
```

```
X = [df['Cement (component 1)(kg in a m^3 mixture)'].values,
     df['Blast Furnace Slag (component 2)(kg in a m^3 mixture)'].values,
     df['Fly Ash (component 3)(kg in a m^3 mixture)'].values,
     df['Water  (component 4)(kg in a m^3 mixture)'].values,
     df['Superplasticizer (component 5)(kg in a m^3 mixture)'].values,
     df['Coarse Aggregate  (component 6)(kg in a m^3 mixture)'].values,
     df['Fine Aggregate (component 7)(kg in a m^3 mixture)'].values,
     df['Age (day)'].values]
y = df['Concrete compressive strength(MPa, megapascals) '].values
X = np.array(X)
X = X.transpose()
y = np.array(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

features = X_train.shape[1]
w = np.empty(features)
w.fill(0)
b=0
h=lambda x : np.matmul(w,x) + b
learn=0.000001
N=len(X_train)
```

Each iteration updates w:

```
for i in range (1000):
    sum_b = 0
    sum_w = np.empty(features)
    sum_w.fill(0)
    for j in range (N):
        sum_b += h(X_train[j])-y_train[j]
        for k in range (features):
            sum_w[k] += (h(X_train[j])-y_train[j]) * X_train[j][k]
    sum_b, sum_w = sum_b/N, sum_w/N
    b = b - (0.1 * sum_b)
    w = w - (learn * sum_w)

y_pred = np.empty(len(X_test))
for i in range (len(X_test)):
    y_pred[i] = h(X_test[i])

print("Weight: {}".format(w))
print("Bias: {}".format(b))
print("r2_score: {}".format(r2_score(y_test, y_pred)))
print("mean_squared_error: {}".format(mean_squared_error(y_test, y_pred)))
```

```
Weight: [ 0.11438799  0.0886608   0.08375223 -0.07982181  0.02329674 -0.00692622
  0.01107201  0.09982904]
Bias: 0.618698751047819
r2_score: 0.4943192616369696
mean_squared_error: 131.8282275874818
```

Each iteration only updates w*j*:

```
#only update one random weight per iteration
for i in range (1000):
    sum_b = 0
    sum_w = np.empty(features)
    sum_w.fill(0)
    for j in range (N):
        sum_b += h(X_train[j])-y_train[j]
        for k in range (features):
            sum_w[k] += (h(X_train[j])-y_train[j]) * X_train[j][k]
    sum_b, sum_w = sum_b/N, sum_w/N
    rd = randint(0,features-1)
    b = b - (0.1 * sum_b)
    w[rd] = w[rd] - (learn * sum_w[rd])

y_pred = np.empty(len(X_test))
for i in range (len(X_test)):
    y_pred[i] = h(X_test[i])

print("Weight: {}".format(w))
print("Bias: {}".format(b))
print("r2_score: {}".format(r2_score(y_test, y_pred)))
print("mean_squared_error: {}".format(mean_squared_error(y_test, y_pred)))
```

```
Weight: [ 0.1169027   0.09207942  0.08761112 -0.08887036  0.02469588 -0.00440782
  0.01311798  0.10241045]
Bias: -3.082208234252107
r2_score: 0.5016566284328341
mean_squared_error: 129.91541583398435
```

7. Compare the performance between two different update method

嘗試多次之後，發現兩者的 performance 其實差不多。一開始認為隨機更新參數的 performance 可以跟上全部參數更新的 performance 是因為 iteration 夠多，以至於已經達到需更新的量，因此我們試著調低 iteration 的次數，卻發現 performance 仍然接近，有時甚至更高，讓人不禁佩服想出這個作法的人，在 weight 很多的情況下，隨機更新參數可以降低運算量，同時又可以有不錯的 performance，是個很不錯的參數更新方法。

8. The code, MSE, and the r2_score for problem 4

```
import pandas as pd
import numpy as np
from decimal import *
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from random import randint
%matplotlib inline
#read data
df = pd.read_csv('Concrete_Data.csv')
print(getcontext())
getcontext().prec = 50
```

因為 y_train, y_test 都是(N,)所以都樣 reshape 變成(N, 1)

```
X = [df['Cement (component 1)(kg in a m^3 mixture)'].values,
     df['Blast Furnace Slag (component 2)(kg in a m^3 mixture)'].values,
     df['Fly Ash (component 3)(kg in a m^3 mixture)'].values,
     df['Water  (component 4)(kg in a m^3 mixture)'].values,
     df['Superplasticizer (component 5)(kg in a m^3 mixture)'].values,
     df['Coarse Aggregate  (component 6)(kg in a m^3 mixture)'].values,
     df['Fine Aggregate (component 7)(kg in a m^3 mixture)'].values,
     df['Age (day)'].values]
y = df['Concrete compressive strength(MPa, megapascals) '].values
X = np.array(X) #many arrays to one high dimention array
X = X.transpose()
y = np.array(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
y_train_shaped = y_train.reshape((824, 1))
y_test_shaped = y_test.reshape((len(y_test), 1))

features = X_train.shape[1]
w = np.empty(165) # 8->164
w.fill(0)
h=lambda x : sum_w*x
learn = (10e-19)
N=len(X_train) #824 row numbers
```

把 train, test 從原本的 8 features 變成 165 features(三次方)

```
# train: make 8 to 164 features
X_train_new = np.zeros((N, 165))
for j in range(N):

    tmp = np.empty(8)
    tmp = X_train[j]

    c = 0
    X_train_new[j][c] = 1
    c = c+1
    for m in range(8): # 0 to 7
        for n in range(m, 8): # m to 7
            for s in range(n, 8):
                X_train_new[j][c] = tmp[m]*tmp[n]*tmp[s]
                c = c+1
    for m in range(8): # 0 to 7
        for n in range(m, 8): # m to 7
            X_train_new[j][c] = tmp[m]*tmp[n]
            c = c+1
    for m in range(8):
        X_train_new[j][c] = tmp[m]
        c=c+1
print(X_train[0])
print(X_train_new)
```

```
# test: make 8 to 45 features
X_test_new = np.zeros((len(X_test), 165))
for i in range(len(X_test)):

    tmp3 = np.empty(8)
    tmp3 = X_test[i]
    c = 0
    X_test_new[i][c] = 1
    c = c+1
    for m in range(8): # 0 to 7
        for n in range(m, 8): # m to 7
            for s in range(n, 8):
                X_test_new[i][c] = tmp3[m]*tmp3[n]*tmp3[s]
                c = c+1
    for m in range(8): # 0 to 7
        for n in range(m, 8): # m to 7
            X_test_new[i][c] = tmp3[m]*tmp3[n]
            c = c+1
    for m in range(8):
        X_test_new[i][c] = tmp3[m]
        c=c+1
#xprint(X_test_new)
sum_w = np.empty([165, 1])
sum_w.fill(0)
```

把 sum_w*X 得到 prediction, 再用此 prediction 算出 gradient, 最後 update sum_w, 因為是 165 個 feature, 所以總共做那個多次，然後跑十萬次 iteration，後來用了 adagrad optimizer 加強，發現結果有比較好一點。

```python
#training
for i in range (100000):
    pred = 0
    grad = 0
    st=0
    for j in range (165):
        pred = np.dot(X_train_new, sum_w)
        grad = np.dot((y_train_shaped - pred).T, X_train_new[:, j])/X_train_new.shape[0]
        np.nan_to_num(grad)
        sum_w[j, 0] += (learn*grad)
        #print(grad, end=" ")
        st+=grad**2
        sum_w[j, 0] -= learn/np.sqrt(st+1e-6)*grad
        #print(grad/np.sqrt(st))

#testing
#for j in range (165):
    y_pred = np.dot(X_test_new, sum_w)
 #    grad = np.dot((y_test_shaped - y_pred).T, X_test_new[:, j])/X_test_new.shape[0]
  #   sum_w[j, 0] += (10e-22*grad)
    if i%100000 ==0:
        print("Weight: {}".format(sum_w))
        print("r2_score: {:.2f}".format(r2_score(y_test, y_pred)))


print("Weight: {}".format(sum_w))
print("r2_score: {:.2f}".format(r2_score(y_test, y_pred)))
#print("Bias: {}".format(b))
n =len(X_test)
r2 = r2_score(y_test, y_pred)
r2_adj =1- (1-r2)*(n-1)/(n-(165+1))
#print(r2_adj)

print("mean_squared_error: {}".format(mean_squared_error(y_test, y_pred)))
```

最後的 MSE 跟 r2_score

```
r2_score: 0.76
mean_squared_error: 62.465193540856085
```

9. Answer the question

1. What is overfitting?
   The hypothesis learned by machine learning is extremely close to the training data. That may let error gets bigger.
2. Stochastic gradient descent is also a kind of gradient descent, what is the benefit of using SGD?
   1. When the sample is big, it can compute the data more easy.
   2. It can be close to the optimal solution in a relatively short time.

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

GD model is that find local minima based on the initial value. So different initial value may have different local minima, the cause different result.

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

The learning rate is not in the proper range of speed.

If the learning rate is too big, the result may exceed the best solution.

If the learning rate is too small, the descending rate may be small, and cost tons of time.

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

We have learned that adjusting the learning rate is really important. The lower the value is, the slower we will travel along the slope, although it might be a good idea since we don’t want to miss the local minima, it could really take a lot of time to update and converge. So the solution is to make learning rates bigger, but sometimes we may encounter overflow problems, so we need to be really careful to adjust learning rates.

10. Bonus

Hope we can finish this!!!