# Introduction

## (+5) Description about project

+ In this project, I will find a linear function on some data sets using Linear regression methodology and gradient descent algorithm for optimization. Linear regression is a statistical technique that models the linear relationship between a dependent variable y and one or more independent variables x. The gradient descent refers to changes in the model moving along the slope or slope of the graph toward the lowest possible error value.

+ The data set is about the strength of concrete when concrete is mixed up with several ingredients. In the data set, 8 feature values affect the strength of concrete.

+ More specifically, I'm going to build a program to figure out the proper linear model by adopting Uni-variate and Multi-variate regression. Uni-variate linear regression is y = mx + b and multi-variate is y = b0 + m1*x1 + m2*x2 + .. + mn*xn. And I will use MSE(Mean Squared Error) as a loss function. The program can use one feature from the data set or all 8 features to find a linear function. It can be done by designating parameters.

## (+5) The details of my algorithm

### Linear Regression

+ In this project, I will implement the Linear Regression model. There're 2 types of Linear Regression. One is uni-variate and the other is multi-variate. Uni-variate model refers to linear regression with one independent variable while the multi-variate model refers to linear regression with two or more features.

+ A function for uni-variate

$$f(x) = b + mx$$

+ A function for multi-variate

$$f(x) = b + m_1x_1 + m_2x_2 + \cdots + m_ix_i$$

+ Final target is to find out coefficient values (b and m)

### Loss function (Objective function)

+ I will use a Mean Squared Error (MSE) as a loss function (objective function) to find a proper co-efficient. MSE is the mean squared difference between the estimated value and the actual value.

$$\frac{1}{n}\sum_{i=1}^{n}(\, y_i - f(x_i)\,)^2$$

### Gradient Descent algorithm

+ But, to find a proper co-efficient value I need to approach a value.

+ I will use Gradient Descent algorithm for optimization. It is like moving a step toward the target value and it is solved by derivation of a loss function like this.

$$\frac{\partial L}{\partial m} = \frac{1}{n}\sum_{i=1}^{n} -2x_i(y_i - (mx_i + b))$$

+ Finally, I will apply a learning rate to update the coefficient values.

$$m_{new} = m_{old} - \frac{\alpha}{n}\sum_{i=1}^{n} -2x_i(y_i - (m_{old}x_i + b_{old}))$$

$$m_{new} = m_{old} - \alpha * loss\_derivative\ of\ m_{old}$$

### When to stop? & Learning rate

+ I will use simple stop numbers as stopping criteria. And I will adjust the learning rate by examining the coefficient value and a line.

## (+5) Pseudo-code of my algorithm

+ Below are my program's core functions

```
Function traning() {
        Call a function to update coefficients until stop condition
}

Function updateCoefficient() {
        Update coefficient value like below.
        Coefficient = Coefficient – Leaning rate * Loss function derivative
}

Function lossFunctionDerivative(Arguments x, y, b0, b1, i) {
        i is pointing which coefficient would be calculated
        If i equals zero return sum(2*(predict(xi, b0, b1)-yi)*1)
        Else equals zero return sum(2*(predict(xi, b0, b1)-yi)*xi)
```

```
}

Function predict(Argument x) {
        This returns predicted value of a linear function
        Return b0 + b1 * x
}

Function mse(Argument expected, prediction) {
        Calculate squared value of a gap between expected and prediction
        Return sum value above divided by length of parameter array
}

Function var(Argument expected) {
        Calculate difference between mean value and input parameter
        Return sum value above divided by length of parameter array
}

Function rsquare(Argument expected, prediction) {
        Return 1 – mse(expected, prediction)/var(expected)
}

{
In the main function
       Read the input excel file and save the value into an array
       Randomly divide the input data into 900 and 130. 900 is for training
and 130 is for testing
       Create a class holding a training data set
       And calls a training function of a created instance
       After calling training, the class has a parameter value of the linear
function
       And then, getting the predicted by using the class above.
       Finally, print out the mse, var, and rsquare values by using the
```

functions above.

}

## (+3 bonus) How my algorithm calculates MAE

+ To calculate MAE, take the difference between the model's predictions and the correct answer. And then, apply an absolute value to the difference and average it over the entire data set.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

## (+3 bonus) How I normalized my data

+ I used two methods to normalize my data.

+ The first method is deleting all zero values in the data set.

+ The second method is substituting zero values with the mean value.

# Result

## (+18) Variance explained on my model on the training data points

```
D:\kbc_dev\src\mining\hw1>python gg.py Concrete_Data.xls train mse nopre noplot

Linear regression

        optimizer: Gradient descent algorithm

        loss function: mse

        total data row: 1030

        randomly divided by: 900 vs 130

        repeat(stop condition): 4000

        alpha(learning rate): 0.000001

uni, train: column[0] time[1.54]: mse, variance, r^2 [231.059] [281.602] [0.179]

uni, train: column[1] time[1.56]: mse, variance, r^2 [276.089] [281.602] [0.020]

uni, train: column[2] time[1.66]: mse, variance, r^2 [290.300] [281.602] [-0.031]

uni, train: column[3] time[1.61]: mse, variance, r^2 [257.028] [281.602] [0.087]

uni, train: column[4] time[1.54]: mse, variance, r^2 [247.809] [281.602] [0.120]
```

```
uni, train: column[5] time[1.54]: mse, variance, r^2 [274.188] [281.602] [0.026]

uni, train: column[6] time[1.53]: mse, variance, r^2 [279.281] [281.602] [0.008]
```
uni, train: column[7] time[1.64]: mse, variance, r^2 [252.651] [281.602] [0.103]

mul, train: column[8] time[21.52]: mse, variance, r^2 [105.321] [281.602] [0.626]

+ Comment: column 0 – 7 is when using one feature value and column 8 is when using all features. There are 3 features r^2 value is above 10% when using a feature. When using all features r^2 value is above 60%.

+ When uni-variate is used, column 0 has good performance because there are no outliers and the data and the distribution is suitable for a linear function. It can be seen that multi-variate performs much better than uni-variate cases.

## (+18) Variance explained on my model on the testing data points

```
D:\kbc_dev\src\mining\hw1>python gg.py Concrete_Data.xls test mse nopre noplot

Linear regression

        optimizer: Gradient descent algorithm

        loss function: mse

        total data row: 1030

        randomly divided by: 900 vs 130

        repeat(stop condition): 4000

        alpha(learning rate): 0.000001
```
uni, test: column[0] time[1.54]: mse, variance, r^2 [259.187] [290.526] [0.108]

```
uni, test: column[1] time[1.52]: mse, variance, r^2 [283.819] [290.526] [0.023]

uni, test: column[2] time[1.54]: mse, variance, r^2 [295.970] [290.526] [-0.019]
```
uni, test: column[3] time[1.53]: mse, variance, r^2 [246.865] [290.526] [0.150]

uni, test: column[4] time[1.53]: mse, variance, r^2 [227.080] [290.526] [0.218]

```
uni, test: column[5] time[1.53]: mse, variance, r^2 [265.515] [290.526] [0.086]

uni, test: column[6] time[1.52]: mse, variance, r^2 [290.365] [290.526] [0.001]
```
uni, test: column[7] time[1.64]: mse, variance, r^2 [253.356] [290.526] [0.128]

mul, test: column[8] time[21.86]: mse, variance, r^2 [91.194] [290.526] [0.686]

+ Comment: column 0 – 7 is when using one feature value and column 8 is when using all features. There are 4 features r^2 value is above 10% when using a feature. When using all features r^2 value is above 60%

+ Even when the trained model is used for the test data, it can be seen that the performance is as good as the training data. This shows the performance of the linear regression model.

## (+16) The plot of Uni-variate models

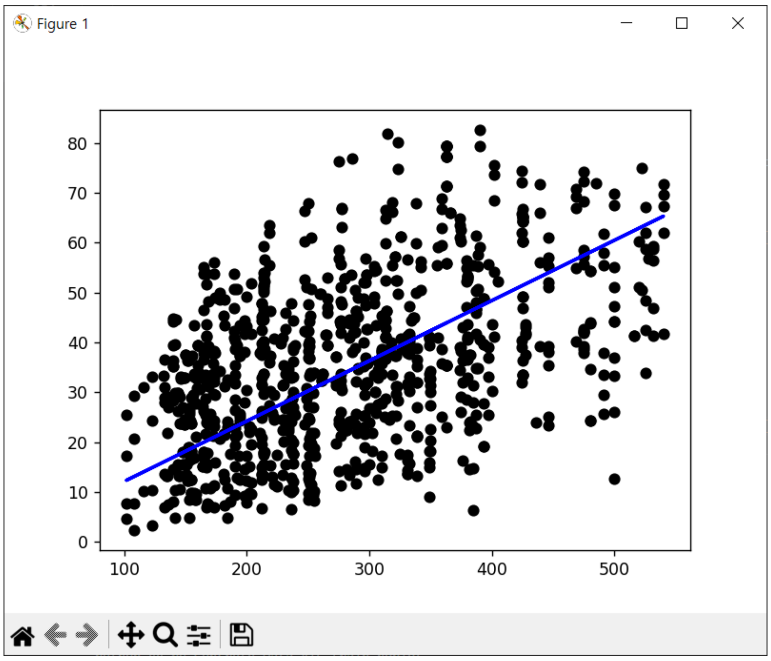

Fig 1 Cement (component 1)(kg in a m^3 mixture)

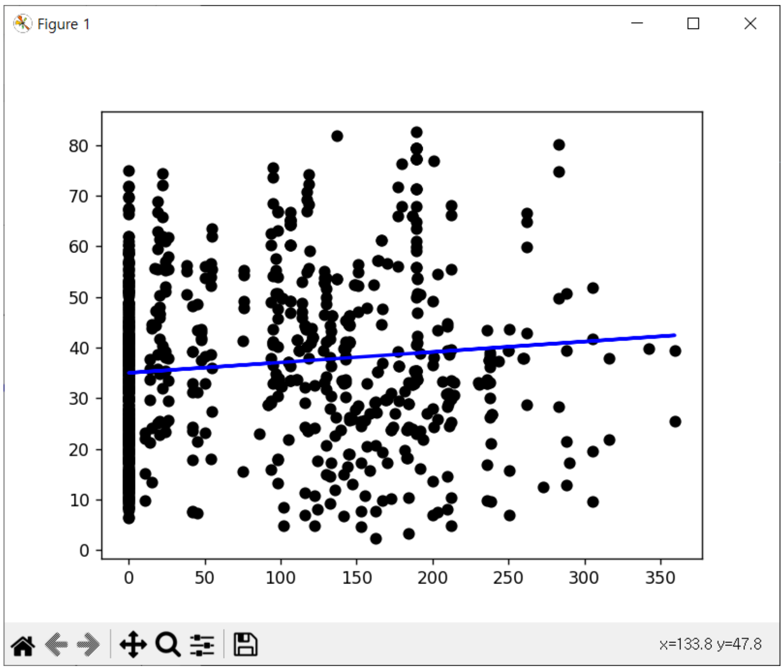

Fig 2 Blast Furnace Slag (component 2)(kg in a m^3 mixture)

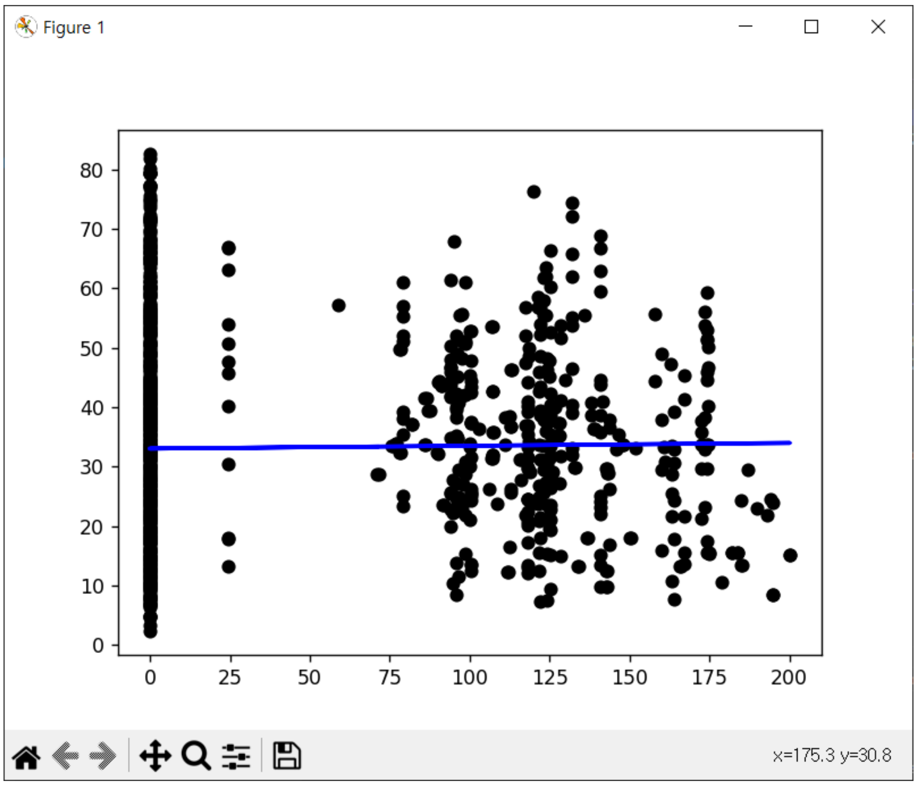

Fig 3 Fly Ash (component 3)(kg in a m^3 mixture)

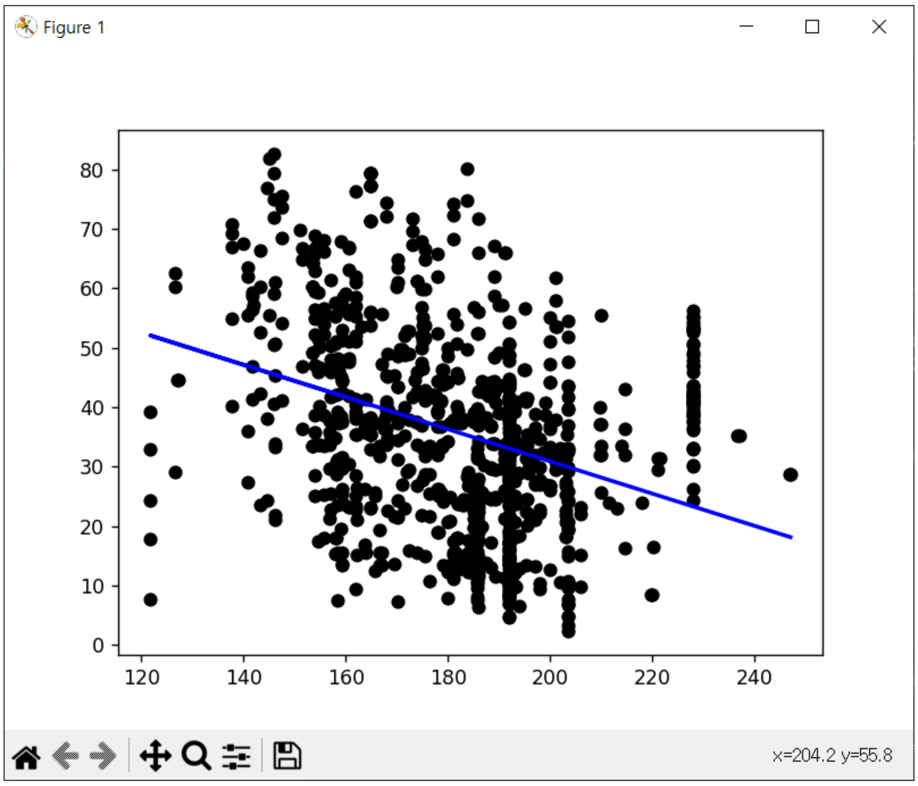

Fig 4 Water (component 4)(kg in a m^3 mixture)

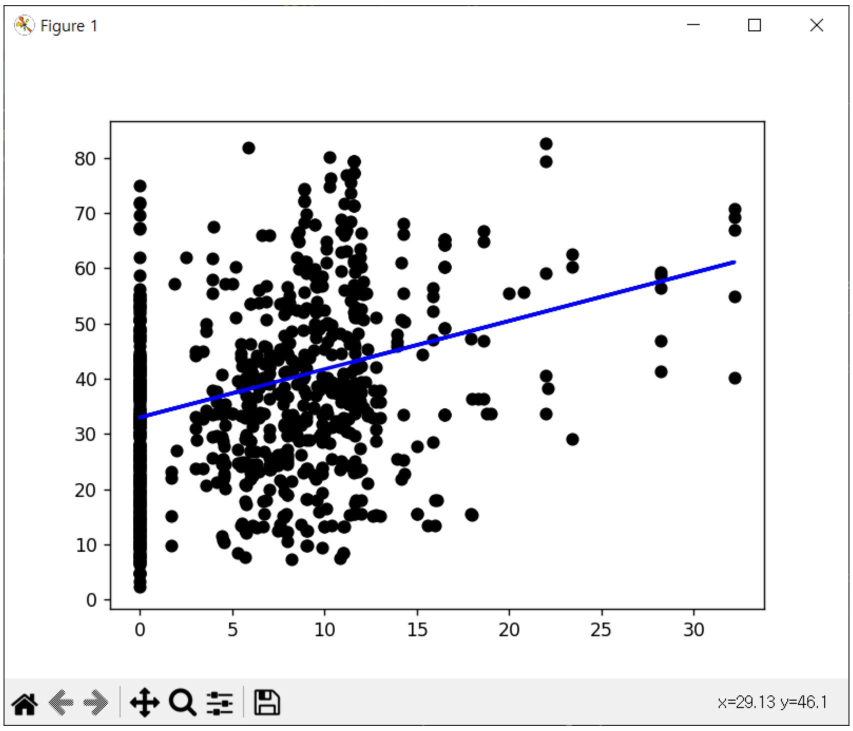

Fig 5 Superplasticizer (component 5)(kg in a m^3 mixture)

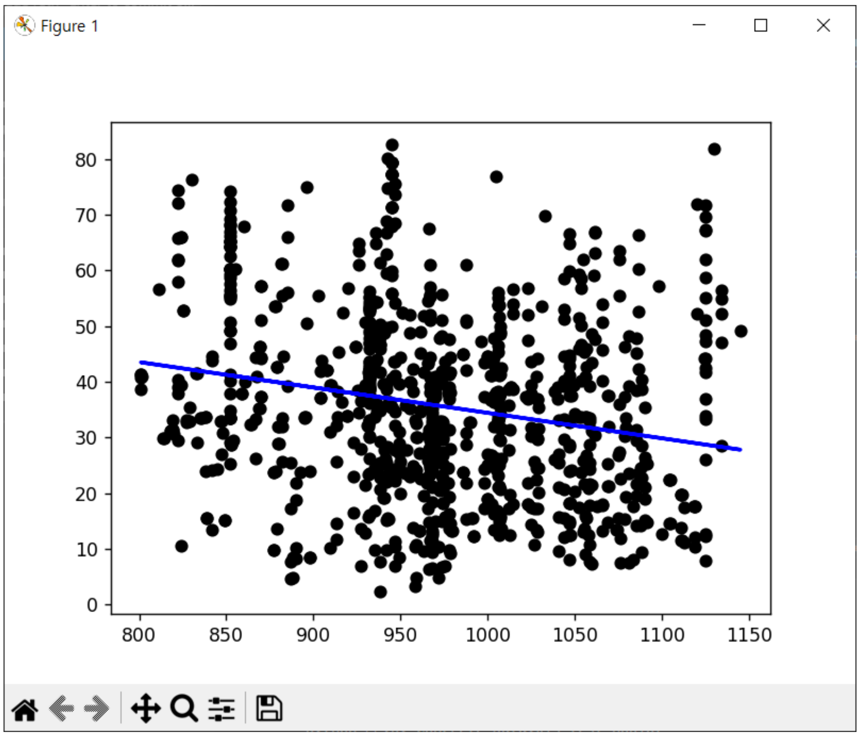

Fig 6 Coarse Aggregate (component 6)(kg in a m^3 mixture)

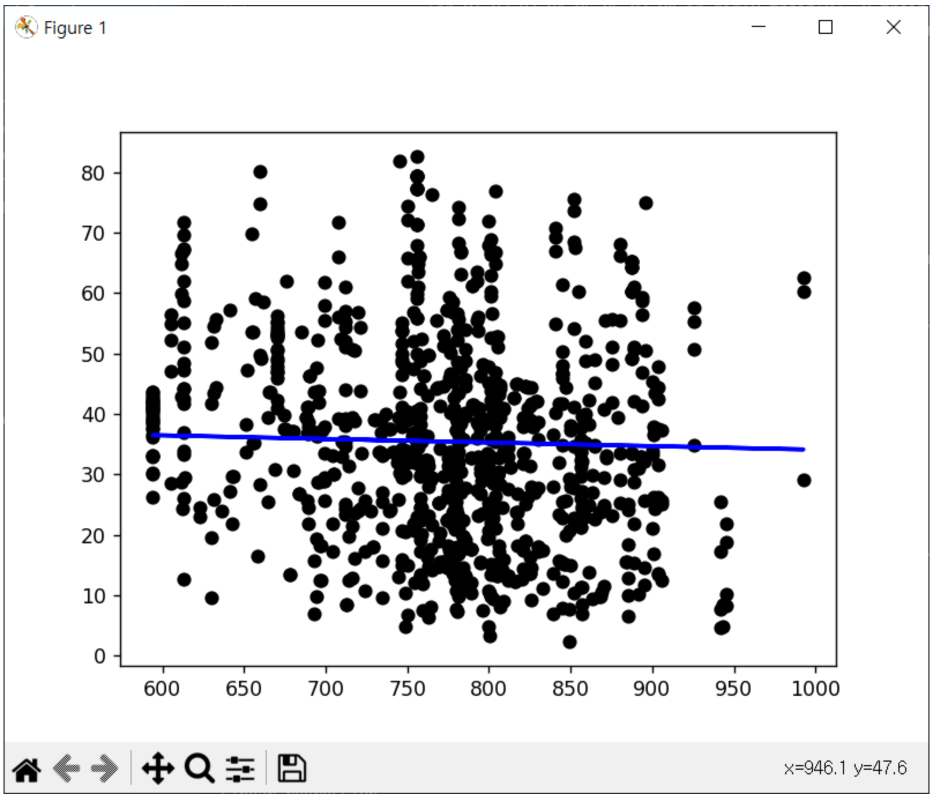

Fig 7 Fine Aggregate (component 7)(kg in a m^3 mixture)

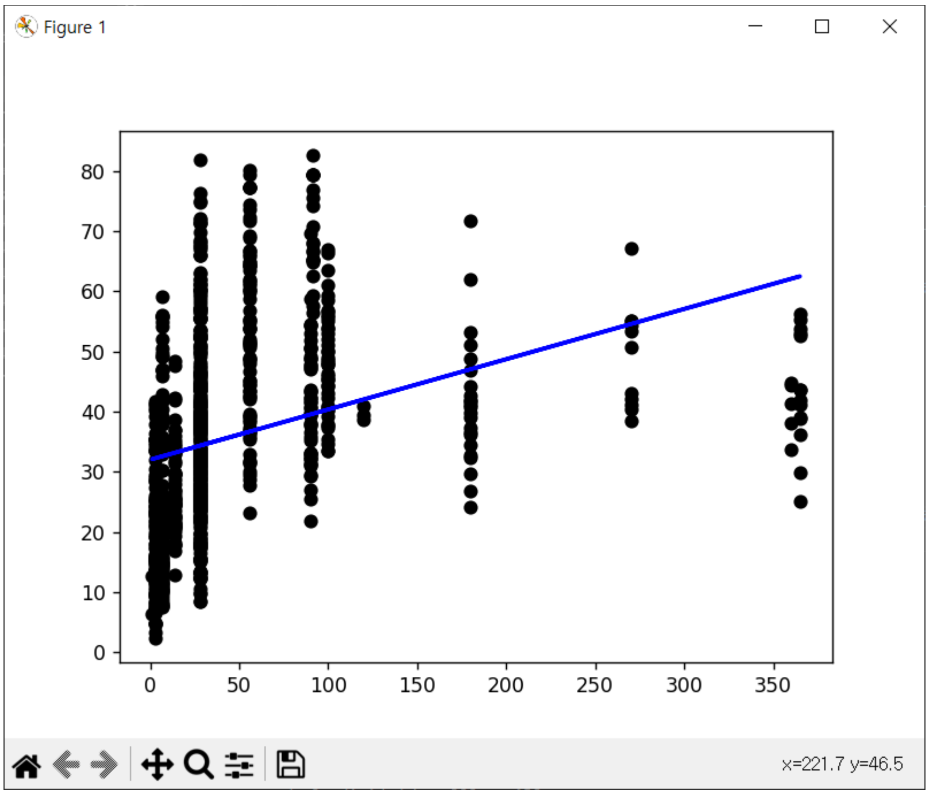

Fig 8 Age (day)

## (+4 bonus) The performance when using MAE for loss function

```
D:\kbc_dev\src\mining\hw1>python gg.py Concrete_Data.xls test mae nopre noplot
Linear regression
        optimizer: Gradient descent algorithm
        loss function: mae
        total data row: 1030
        randomly divided by: 900 vs 130
        repeat(stop condition): 4000
        alpha(learning rate): 0.000010
uni, test: column[0] time[1.24]: mae, variance, r^2 [12.333] [14.508] [0.150]
uni, test: column[1] time[1.28]: mae, variance, r^2 [14.658] [14.508] [-0.010]
uni, test: column[2] time[1.28]: mae, variance, r^2 [14.540] [14.508] [-0.002]
uni, test: column[3] time[1.28]: mae, variance, r^2 [13.977] [14.508] [0.037]
uni, test: column[4] time[1.28]: mae, variance, r^2 [13.561] [14.508] [0.065]
uni, test: column[5] time[1.26]: mae, variance, r^2 [14.773] [14.508] [-0.018]
uni, test: column[6] time[1.29]: mae, variance, r^2 [14.322] [14.508] [0.013]
uni, test: column[7] time[1.38]: mae, variance, r^2 [13.473] [14.508] [0.071]
mul, test: column[8] time[20.53]: mae, variance, r^2 [8.403] [14.508] [0.421]
```

+ Comment: column 0 – 7 is when using one feature value and column 8 is when using all features. There is only one r^2 value above 10% when using a feature. When using all features r^2 value is above 40%.

+ In general, it is known that MAE performs better when there are outliers. But the result is not satisfactory. I think it seems that the reason is that outliers are included in the data too much. Or my implementation of MAE could have a bug as well. I tried to figure it out but I couldn't reach the solution.

## (+4 bonus) The performance when I normalized my data

```
Linear regression
        optimizer: Gradient descent algorithm
        loss function: mse
        total data row: 225
        randomly divided by: 22 vs 203
        repeat(stop condition): 4000
```

```
       alpha(learning rate): 0.000001
uni, train: column[0] time[0.05]: mse, variance, r^2 [223.663] [176.721] [-0.266]
uni, train: column[1] time[0.04]: mse, variance, r^2 [180.325] [176.721] [-0.020]
uni, train: column[2] time[0.05]: mse, variance, r^2 [186.462] [176.721] [-0.055]
uni, train: column[3] time[0.04]: mse, variance, r^2 [141.144] [176.721] [0.201]
uni, train: column[4] time[0.05]: mse, variance, r^2 [165.678] [176.721] [0.062]
uni, train: column[5] time[0.05]: mse, variance, r^2 [221.470] [176.721] [-0.253]
uni, train: column[6] time[0.05]: mse, variance, r^2 [177.866] [176.721] [-0.006]
uni, train: column[7] time[0.05]: mse, variance, r^2 [126.368] [176.721] [0.285]
mul, train: column[8] time[0.57]: mse, variance, r^2 [28.384] [176.721] [0.839]

D:\kbc_dev\src\mining\hw1>python gg.py Concrete_Data.xls train mse premean noplot
Linear regression
       optimizer: Gradient descent algorithm
       loss function: mse
       total data row: 1030
       randomly divided by: 900 vs 130
       repeat(stop condition): 4000
       alpha(learning rate): 0.000001
uni, train: column[0] time[1.53]: mse, variance, r^2 [235.331] [280.995] [0.163]
uni, train: column[1] time[1.52]: mse, variance, r^2 [275.521] [280.995] [0.019]
uni, train: column[2] time[1.53]: mse, variance, r^2 [289.508] [280.995] [-0.030]
uni, train: column[3] time[1.52]: mse, variance, r^2 [259.227] [280.995] [0.077]
uni, train: column[4] time[1.54]: mse, variance, r^2 [251.976] [280.995] [0.103]
uni, train: column[5] time[1.52]: mse, variance, r^2 [276.751] [280.995] [0.015]
uni, train: column[6] time[1.52]: mse, variance, r^2 [278.605] [280.995] [0.009]
uni, train: column[7] time[1.64]: mse, variance, r^2 [251.322] [280.995] [0.106]
mul, train: column[8] time[21.52]: mse, variance, r^2 [110.147] [280.995] [0.608]
```

+ Comment: I tested two types of normalization of the data set. One is removing zero data and the other is substituting zero values with mean value.

+ On the first test, I removed zero values, but did not improve uni-variate performance. This is probably because meaningful data was also deleted.

+ But looking at the last multi-variate, rsquare is over 80 percent. There was a huge performance improvement and multi-variate shows very good performance for data without outliers.

+ The second one is substituting zero value and similar performance is obtained even with data pre-processing. When I checked the result with the graph, it was found that even if the 0 value was specified as the average value, it did not affect the linear function. Because it is a result of only moving the value along the x-axis.

# Discussion

## (+4) Describe how the different models compared in accuracy on the training data. Did the same models that accurately predicted the training data also accurately predict the testing data?

+ First column, Cement, was the best compared to other columns in accuracy. And the third one, Fly Ash, was the worst compared to the other columns. Because Cement data points had a tendency like a linear line, it seemed it had no outlier data. On the other hand, Fly ash seemed to have a lot of outliers (which was zero value). It was hard to find a proper linear line with such data. I think that the more data that is out of tendency, the less accurate it is.

+ Same models can predict the same result on the testing data. I think it shows us the power of Linear regression method.

## (+4) Describe how the different models compared to train/test. Did different models take longer to train? Did you have to use different hyperparameter values for different models?

+ Although there are some differences, the training results were similar in the tests.

+ It is difficult to say that there is a difference in elapsed time between the uni-variate models. Because when training a linear model, they were using the same amount of points.

+ But, there was a difference in elapsed time between uni-variate and multi-variate models. Multi-variate models, of course, took a lot more time than uni-variate models.

+ I used different parameter values for different models because some models did not find an appropriate y-intercept due to a large number of zeros. For example, I used 0 as the starting point for b0 in Cement and 33 in Superplasticizer.

## (+5) Draw some conclusions about what factors predict concrete compressive strength. What would you recommend for making the hardest possible concrete?

+ From the test I've been doing so far, the top 4 factors are useful to predict Concrete compressive strength. Cement, Age, Superplasticizer, Water. They usually had higher VarianceExplain(r square) values than the others. A higher r square value can show that the factor has a significant impact on the linear model.

+ It is judged that the optimum strength is to be shown by putting more than 500 kg of cement, about 150 to 160 kg of water, and 10 to 13 kg of superplasticizer, and then hardening it for 50 to 100 days.

## (+2 bonus) if you include comparisons from using MAE

+ Because MAE takes absolute values, the errors are weighted on the same scale. So, unlike MSE, it doesn't weigh the outliers too much, and the r^2 values are more gradual. The MSE is much better in the good and much worse in the bad, but the MAE is showing a gradual appearance in both the good and the bad.

## (+2 bonus) if you include comparisons from normalized or standardized Data

+ Deleting zero method: Before looking at the results, I expected the result to be good if I removed the 0 value. However, the result is not as good as I thought on uni-variate model. I think the reason is that all rows containing 0 were removed, so other rows with meaningful data also disappeared. But on multi-variate model, it showed rsquare over 80 percent. I found that multi-variate performed very well if there is no outliers.

+ Substituting zero value with mean method: Before seeing the results, I thought the results would be good if I filled the values with the average. But the result was not that good. Because the MSE value is not significantly affected because the x-axis value is simply moved from 0 to the average.

# My program

## (+15) How to run the program

+ 2 files are needed to run the program.

+ One is a python source code and the other is a data excel file.

+ how to execute

Usage:

```
        python gg.py [arg1] [arg2] [arg3] [arg4] [arg5]
        arg1: input excel file
        arg2: train or test
                train - apply learned model to a training data set
                test - apply learned model to a test data set
        arg3: mse or mae
                mse - using mse as a loss function
                mae - using mae as a loss function
        arg4: nopre or prezero or premean
                nopre - no pre-processing on the data
                prezero - pre-processing data by deleting zero values
                premean - pre-processing data by substituing zero with mean
value
        arg5: plot or noplot
                plot - show a plot on a uni-varite model
                noplot - not showing a plot on a uni-varite model
Example:
        python gg.py Concrete_Data.xls train mse nopre noplot
        python gg.py Concrete_Data.xls train mse nopre plot
        python gg.py Concrete_Data.xls train mae nopre noplot
        python gg.py Concrete_Data.xls train mae nopre plot
        python gg.py Concrete_Data.xls test mse nopre noplot
        python gg.py Concrete_Data.xls test mse nopre plot
        python gg.py Concrete_Data.xls test mae nopre noplot
        python gg.py Concrete_Data.xls test mae nopre plot
```

## (+5) The source code

+ I will upload a whole source code file to Gradescope cause it’s a bit long to pit in this document.

## (+5 bonus) The code for MAE loss function

+ The first one is MAE code for uni-variate model

```
    def loss_derivative_mae(self, i):
        x, y = self.x, self.y
        predict = self.predict

        sum = 0
        for xi,yi in zip(x,y):
            if i==0:
                sum += 1 if predict(xi) > yi else -1
            else:
                sum += xi if predict(xi) > yi else -xi
        # print('mae', sum, len(x), sum/len(x))
        return sum/len(x)
```

+ The second one is MAE code for multi-variate model

```
    def loss_derivative_mae(self, i):
        y = self.y
        x1,x2,x3,x4,x5,x6,x7,x8 =
self.x1,self.x2,self.x3,self.x4,self.x5,self.x6,self.x7,self.x8
        predict = self.predict

        sum = 0
        for xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8,yi in
zip(x1,x2,x3,x4,x5,x6,x7,x8,y):
            if i==0:
                sum += 1 if predict(xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8) > yi else
-1
            elif i==1:
                sum += xx1 if predict(xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8) > yi
else -xx1
            elif i==2:
                sum += xx2 if predict(xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8) > yi
else -xx2
            elif i==3:
                sum += xx3 if predict(xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8) > yi
else -xx3
            elif i==4:
                sum += xx4 if predict(xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8) > yi
else -xx4
            elif i==5:
                sum += xx5 if predict(xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8) > yi
else -xx5
            elif i==6:
                sum += xx6 if predict(xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8) > yi
else -xx6
            elif i==7:
                sum += xx7 if predict(xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8) > yi
else -xx7
            elif i==8:
                sum += xx8 if predict(xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8) > yi
else -xx8

        return sum/len(x1)
```

## (+2 bonus) The code for normalizing data

+ This code is for normalizing data and I developed two method. The one is deleting zero value and the other is substituting zero values with mean value.

```
    # pre-processing data
    # prezero is deleting all zeros in the dataset
    # premean is to substitue zero values with mean value
    if pre_type == 'prezero':
        for x0,x1,x2,x3,x4,x5,x6,x7,y in oldPair:
            if x0!=0 and x1!=0 and x2!=0 and x3!=0 and x4!=0 and x5!=0 and
x6!=0 and x7!=0 :
                newPair.append([x0,x1,x2,x3,x4,x5,x6,x7,y])

        pair = newPair
        divide_num = round(len(newPair) * 0.1)
    elif pre_type == 'premean':
        for col in x:
            mm = np.mean(col)
            for i in range(len(col)):
                if col[i] == 0:
                    col[i] = mm
```