Introduction

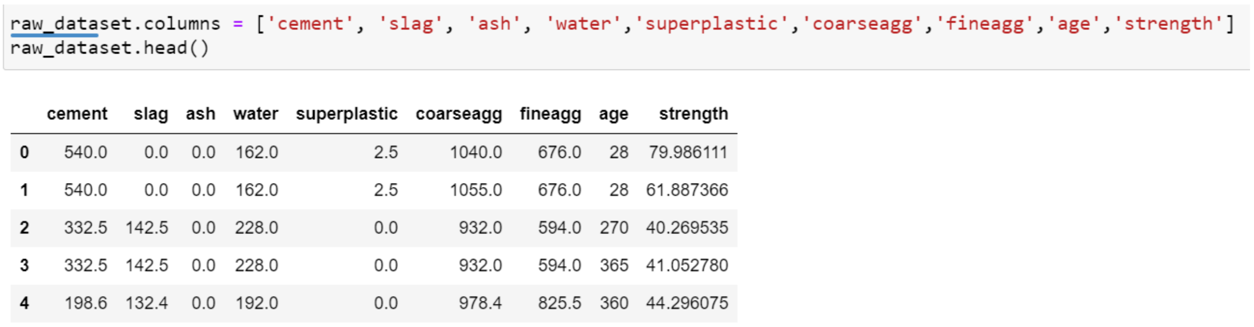
1. Formulation of the problem

This assignment is about applying data mining technique on the dataset from UCI Machine Learning Repository: Concrete Compressive Strength. The data showed 8 features with 1 response value (Concrete strength). The problem is trying to find the relation between these features contributing to the response value. We calculate 8 univariate models (i.e. each model with one column as input feature) and 1 multivariate model (i.e. one model with 8 columns as input features).

By using gradient descend as the algorithm with MSE as loss function, we can calculate the R squared value according to the training set and testing set. By doing so, we can understand how a single feature contributes to the result outcome of concrete strength. If the R squared value is high for one univariate model, then it indicates that single feature is able to explain the variance of response value. However, high R squared on training set does not mean it would perform better on new data. Therefore, we need to also calculate R squared value on the testing set. So that if the model performs well on the training set but not testing set, we know that the model is not generalizing well.

2. Pre-process the dataset.

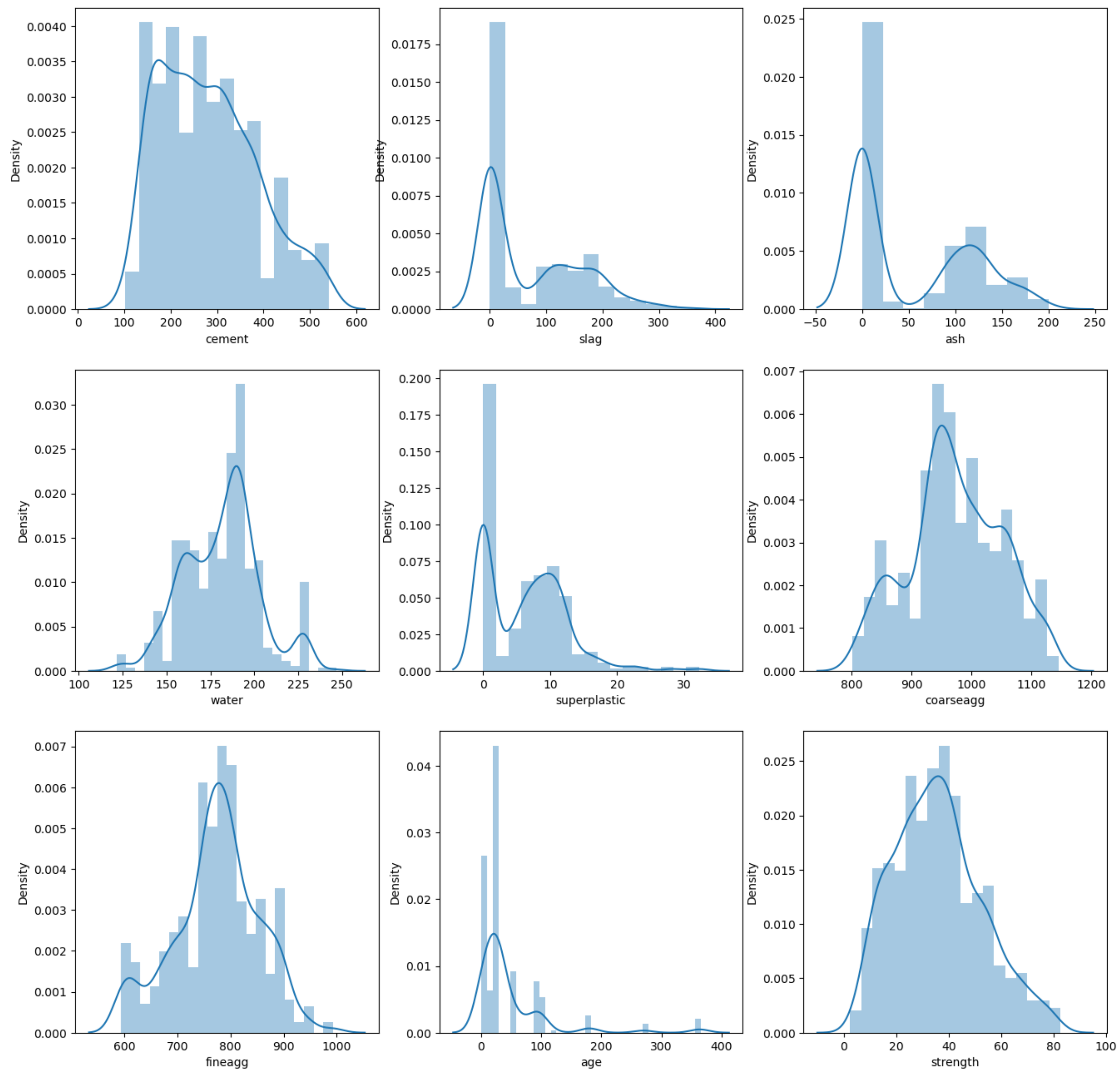
Before we look into the dataset, let’s first change the name of the columns, given that the words are just too long. It is not user-friendly to read all the column names. The columns names are shortened into 'cement', 'slag', 'ash', water', 'superplastic', 'coarseagg', 'fineagg', 'age', 'strength'.



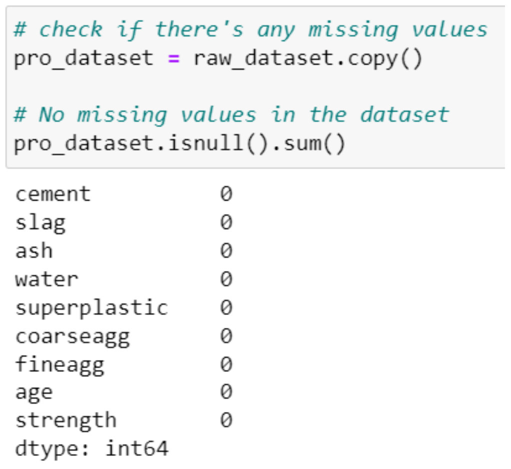
```
raw_dataset.columns = ['cement', 'slag', 'ash', 'water','superplastic','coarseagg','fineagg','age','strength']
raw_dataset.head()
```

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.986111
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.887366
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.269535
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.052780
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.296075

Then, we can first look at the distribution of the raw dataset. It shows that cement, coarseagg, water, fineagg, and the response variable strength are all distributed similarly to normal distribution. In addition, slag, ash, superplastic and age all got small values distributed at the very front of normal distribution.



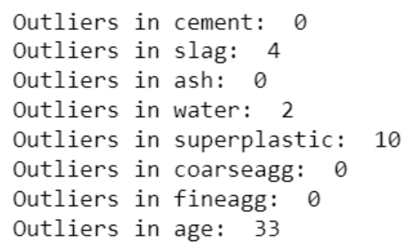
Check if there is any null values in the dataset.



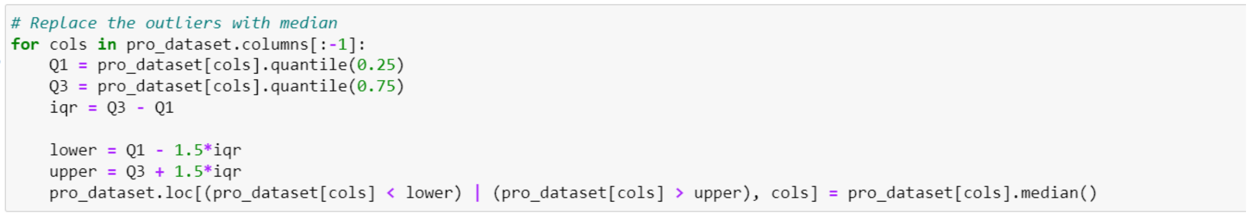
```
# check if there's any missing values
pro_dataset = raw_dataset.copy()

# No missing values in the dataset
pro_dataset.isnull().sum()
```

```
cement          0
slag            0
ash             0
water           0
superplastic    0
coarseagg       0
fineagg         0
age             0
strength        0
dtype: int64
```

Check if there are any outliers, which are over three standard deviations (std). It seems like age has the most outliers. Given that we need to split the data into training 900 and testing 130, we cannot drop any data. Therefore, I choose to replace the outliers with median.

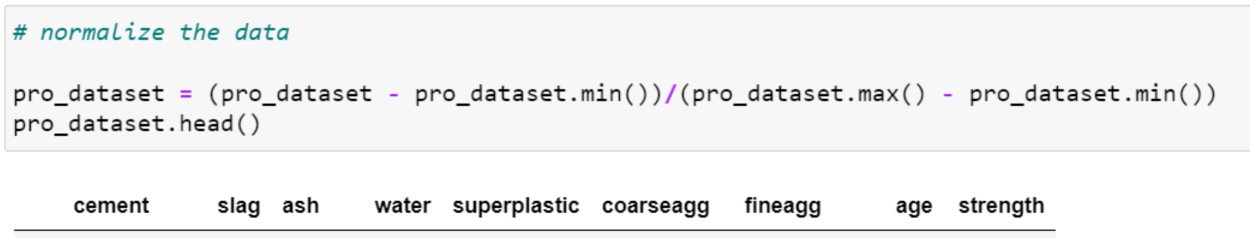


```
Outliers in cement:  0
Outliers in slag:  4
Outliers in ash:  0
Outliers in water:  2
Outliers in superplastic:  10
Outliers in coarseagg:  0
Outliers in fineagg:  0
Outliers in age:  33
```

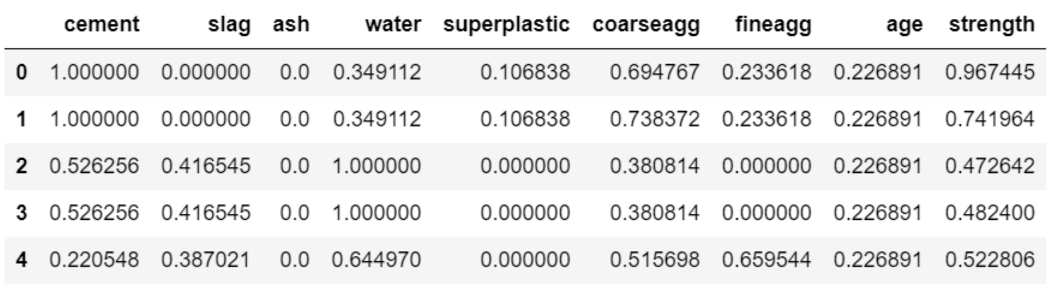


```python
# Replace the outliers with median
for cols in pro_dataset.columns[:-1]:
    Q1 = pro_dataset[cols].quantile(0.25)
    Q3 = pro_dataset[cols].quantile(0.75)
    iqr = Q3 - Q1

    lower = Q1 - 1.5*iqr
    upper = Q3 + 1.5*iqr
    pro_dataset.loc[(pro_dataset[cols] < lower) | (pro_dataset[cols] > upper), cols] = pro_dataset[cols].median()
```

Then, given that the features in raw data have different numbers with huge differences, I choose to do normalization. It would make my data in each feature to normalize between 0 to 1.

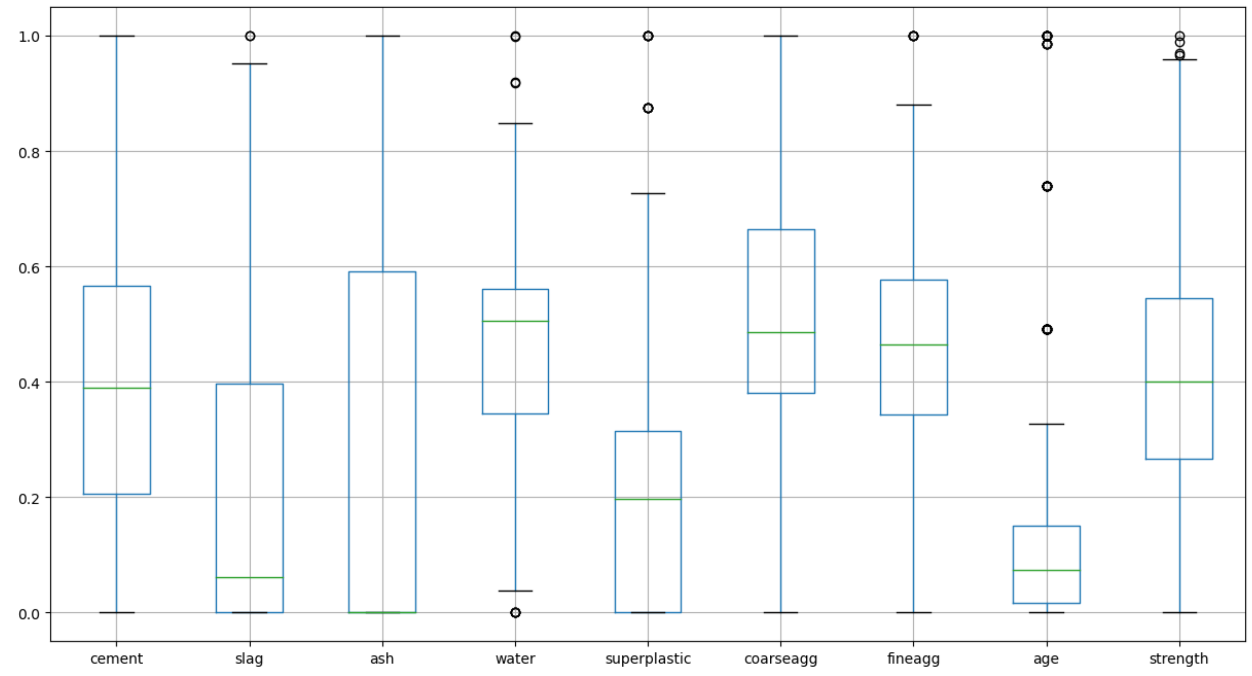


```python
# normalize the data

pro_dataset = (pro_dataset - pro_dataset.min())/(pro_dataset.max() - pro_dataset.min())
pro_dataset.head()
```

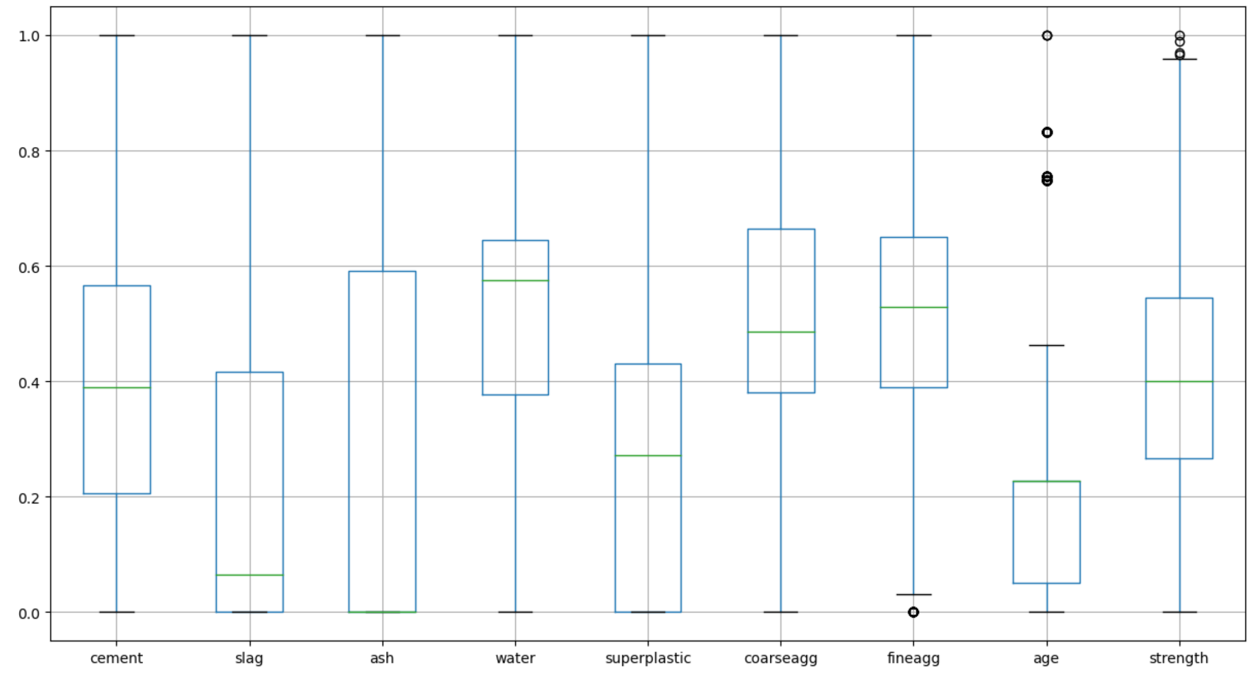


	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength
0	1.000000	0.000000	0.0	0.349112	0.106838	0.694767	0.233618	0.226891	0.967445
1	1.000000	0.000000	0.0	0.349112	0.106838	0.738372	0.233618	0.226891	0.741964
2	0.526256	0.416545	0.0	1.000000	0.000000	0.380814	0.000000	0.226891	0.472642
3	0.526256	0.416545	0.0	1.000000	0.000000	0.380814	0.000000	0.226891	0.482400
4	0.220548	0.387021	0.0	0.644970	0.000000	0.515698	0.659544	0.226891	0.522806

If I only normalize the dataset but did not replace outliers with median:

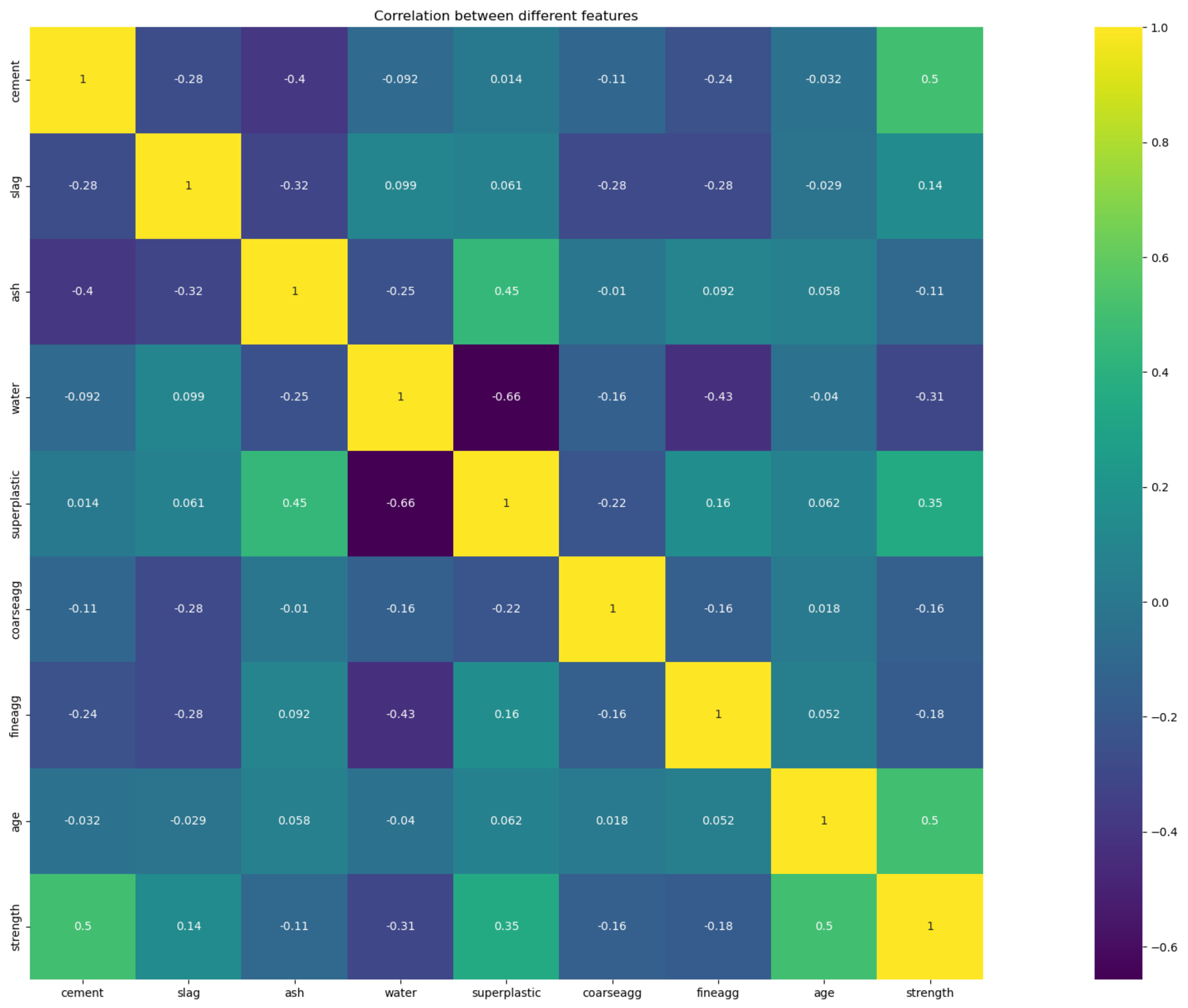


If I normalize the dataset, but replace the outliers with median:



We can see that the outliers reduce significantly on the second graph.

Besides normalizing the dataset, we can take a sneak peak at how the features relate to each other using a heatmap:



	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength
cement	1	-0.28	-0.4	-0.092	0.014	-0.11	-0.24	-0.032	0.5
slag	-0.28	1	-0.32	0.099	0.061	-0.28	-0.28	-0.029	0.14
ash	-0.4	-0.32	1	-0.25	0.45	-0.01	0.092	0.058	-0.11
water	-0.092	0.099	-0.25	1	-0.66	-0.16	-0.43	-0.04	-0.31
superplastic	0.014	0.061	0.45	-0.66	1	-0.22	0.16	0.062	0.35
coarseagg	-0.11	-0.28	-0.01	-0.16	-0.22	1	-0.16	0.018	-0.16
fineagg	-0.24	-0.28	0.092	-0.43	0.16	-0.16	1	0.052	-0.18
age	-0.032	-0.029	0.058	-0.04	0.062	0.018	0.052	1	0.5
strength	0.5	0.14	-0.11	-0.31	0.35	-0.16	-0.18	0.5	1

It shows that cement and age have the most significant impact on the response value, followed by superplastic. Additionally, ash has correlation with superplastic too.

3. The design of my gradient descent algo

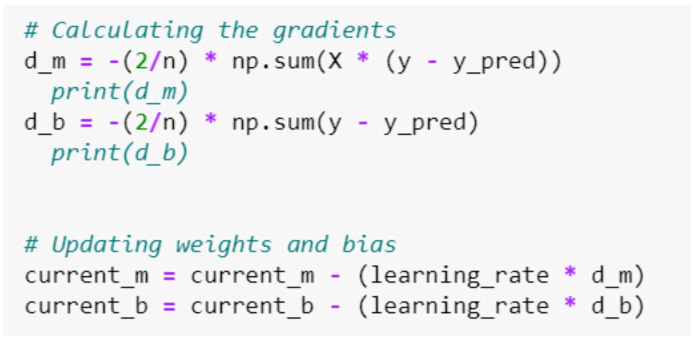
I separated the models into two kinds: univariate and multivariate. Each has another two kinds: preprocessed data and raw data. Given that first, the input size of univariate model are different from the multivariate model, and second, raw data requires different learning rate compared to the preprocessed data. For instance, raw data need a learning rate of 0.00000001 to not having the derivative of loss function overflow, which on the other hand, preprocessed data only need around 0.01 or 0.001. I did not use a stochastic gradient descend algo, because the normal algo I wrote is already running pretty fast. Every model has a stopping threshold of 1e-6. During the iterations, I have a current_cost equals to mean_squared_error of y and y prediction, and if the change in cost is less than or equal to the stopping threshold, we stop the gradient descent. This way, we make sure we are not wasting time if we got the ideal weight.

4. Pseudo code of the algo

```
def Gd_(X, y, num_iterations, learning_rate,  stopping_threshold):
    #Initializing weight, bias, learning rate, n, and iterations
   # weight
   current_m = 0
   # bias
   current_b = 0
   n = float(len(X))
  for i in range(num_iterations):
      y prediction = np.dot( X , current_m) + current_b
       # Calculating the current cost
       current_cost = mean_squared_error(y, y prediction)
       # If the change in cost is less than or equal to stopping_threshold we stop
       if previous_cost and abs(previous_cost - current_cost) <= stopping_threshold:
           break
       # Calculating the gradients
       d_m = -(2/n) * np.sum(X * (y – y prediction))
       d_b = -(2/n) * np.sum(y – y prediction)
       # Updating weights and bias
       current_m = current_m - (learning_rate * d_m)
       current_b = current_b - (learning_rate * d_b)
 #        Printing the parameters for each 1000th iteration
#        print(f"Iteration {i+1}: Cost {current_cost}, Weight  {current_m}, Bias {current_b}")
    return current_m , current_b
```

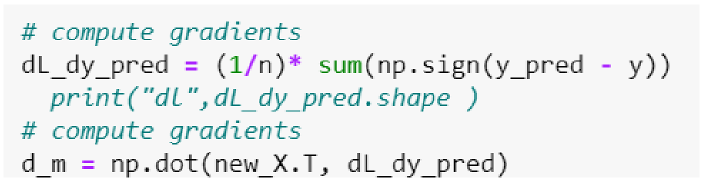
5. How I implement MAE

In this case, I replace the loss function from MSE into MAE. There difference is the derivative of loss function. For MSE, it is written as:



```
# Calculating the gradients
d_m = -(2/n) * np.sum(X * (y - y_pred))
  print(d_m)
d_b = -(2/n) * np.sum(y - y_pred)
  print(d_b)


# Updating weights and bias
current_m = current_m - (learning_rate * d_m)
current_b = current_b - (learning_rate * d_b)
```

For MAE:

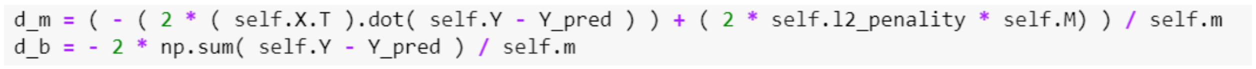


```
# compute gradients
dL_dy_pred = (1/n)* sum(np.sign(y_pred - y))
  print("dL",dL_dy_pred.shape )
# compute gradients
d_m = np.dot(new_X.T, dL_dy_pred)
```

6. How I implement Ridge

It is similar to how I implement MAE which is changing the derivative of loss function.

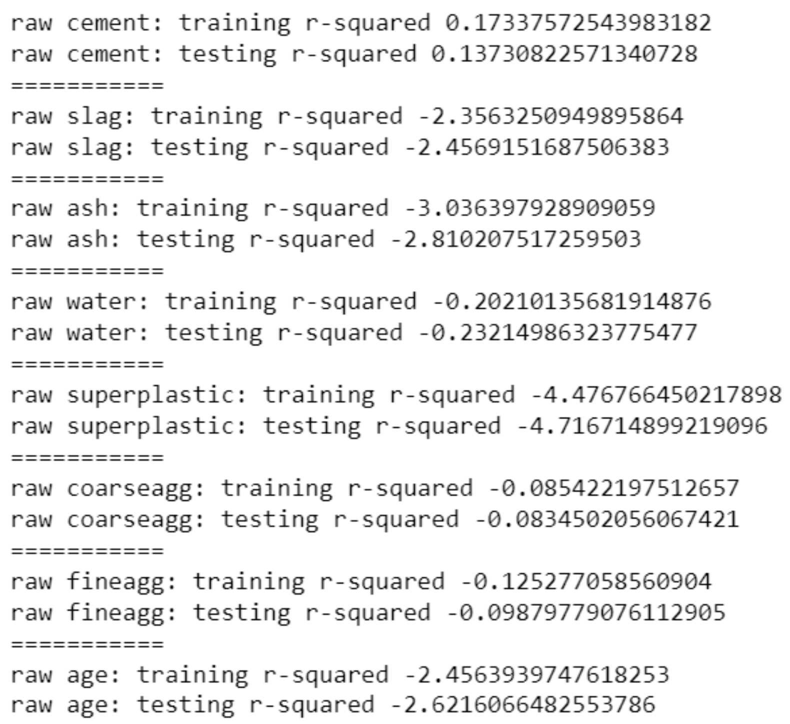
For ridge:



```
d_m = ( - ( 2 * ( self.X.T ).dot( self.Y - Y_pred ) ) + ( 2 * self.l2_penality * self.M) ) / self.m
d_b = - 2 * np.sum( self.Y - Y_pred ) / self.m
```

Results

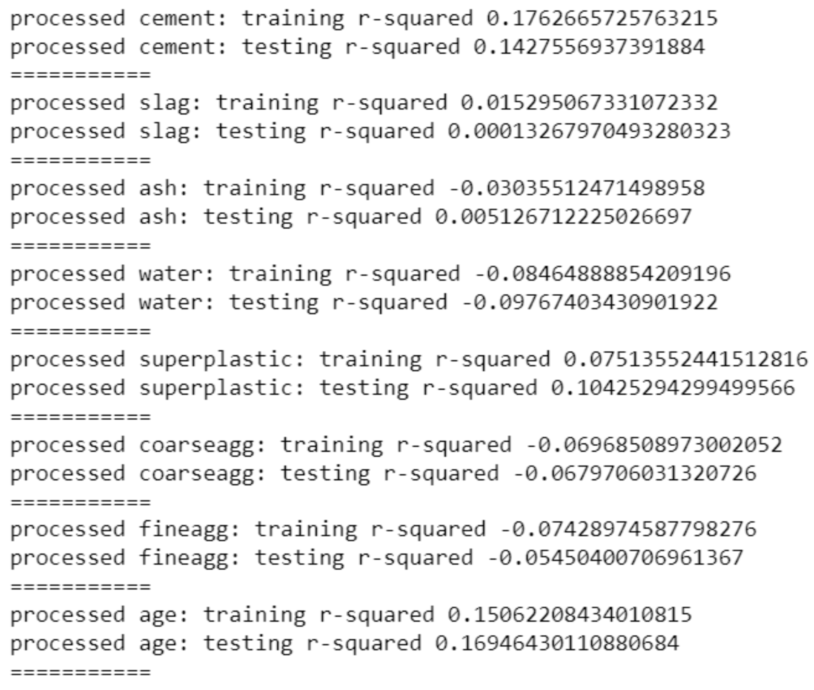
1. Variance explained on training set vs testing set with MSE.
   a. 8 Univariate models with raw data



```
raw cement: training r-squared 0.17337572543983182
raw cement: testing r-squared 0.13730822571340728
===========
raw slag: training r-squared -2.3563250949895864
raw slag: testing r-squared -2.4569151687506383
===========
raw ash: training r-squared -3.036397928909059
raw ash: testing r-squared -2.810207517259503
===========
raw water: training r-squared -0.20210135681914876
raw water: testing r-squared -0.23214986323775477
===========
raw superplastic: training r-squared -4.476766450217898
raw superplastic: testing r-squared -4.716714899219096
===========
raw coarseagg: training r-squared -0.085422197512657
raw coarseagg: testing r-squared -0.0834502056067421
===========
raw fineagg: training r-squared -0.125277058560904
raw fineagg: testing r-squared -0.09879779076112905
===========
raw age: training r-squared -2.4563939747618253
raw age: testing r-squared -2.6216066482553786
```

The variance explained on raw dataset in general performs poorly. Each model performs identically poor on training set and testing set.

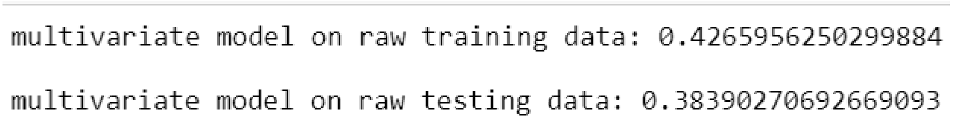
b. 8 Univariate on processed data



```
processed cement: training r-squared 0.1762665725763215
processed cement: testing r-squared 0.1427556937391884
===========
processed slag: training r-squared 0.015295067331072332
processed slag: testing r-squared 0.00013267970493280323
===========
processed ash: training r-squared -0.03035512471498958
processed ash: testing r-squared 0.005126712225026697
===========
processed water: training r-squared -0.08464888854209196
processed water: testing r-squared -0.09767403430901922
===========
processed superplastic: training r-squared 0.07513552441512816
processed superplastic: testing r-squared 0.10425294299499566
===========
processed coarseagg: training r-squared -0.06968508973002052
processed coarseagg: testing r-squared -0.0679706031320726
===========
processed fineagg: training r-squared -0.07428974587798276
processed fineagg: testing r-squared -0.05450400706961367
===========
processed age: training r-squared 0.15062208434010815
processed age: testing r-squared 0.16946430110880684
===========
```

These results show that cement and age models have better variance explained score compared to other univariate models. They both perform similarly on training set and testing set. For ash and superplastic, they have better performance on testing than training set. This shows that the model may be able to capture the underlying signal in the new dataset and generalize better than it did on the training set, resulting in a positive variance explained score.

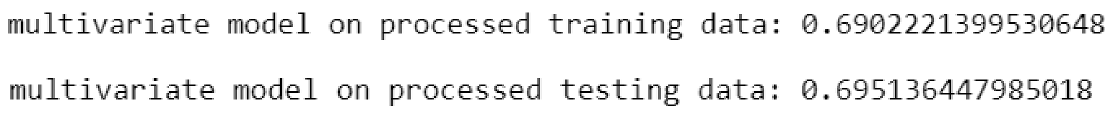
c. 1 Multivariate model on raw data



```
multivariate model on raw training data: 0.4265956250299884

multivariate model on raw testing data: 0.38390270692669093
```

The multivariate model performs extremely better than any univariate models. It has a slightly worse performance on testing, but overall, still able to explain almost 40 percent of data.

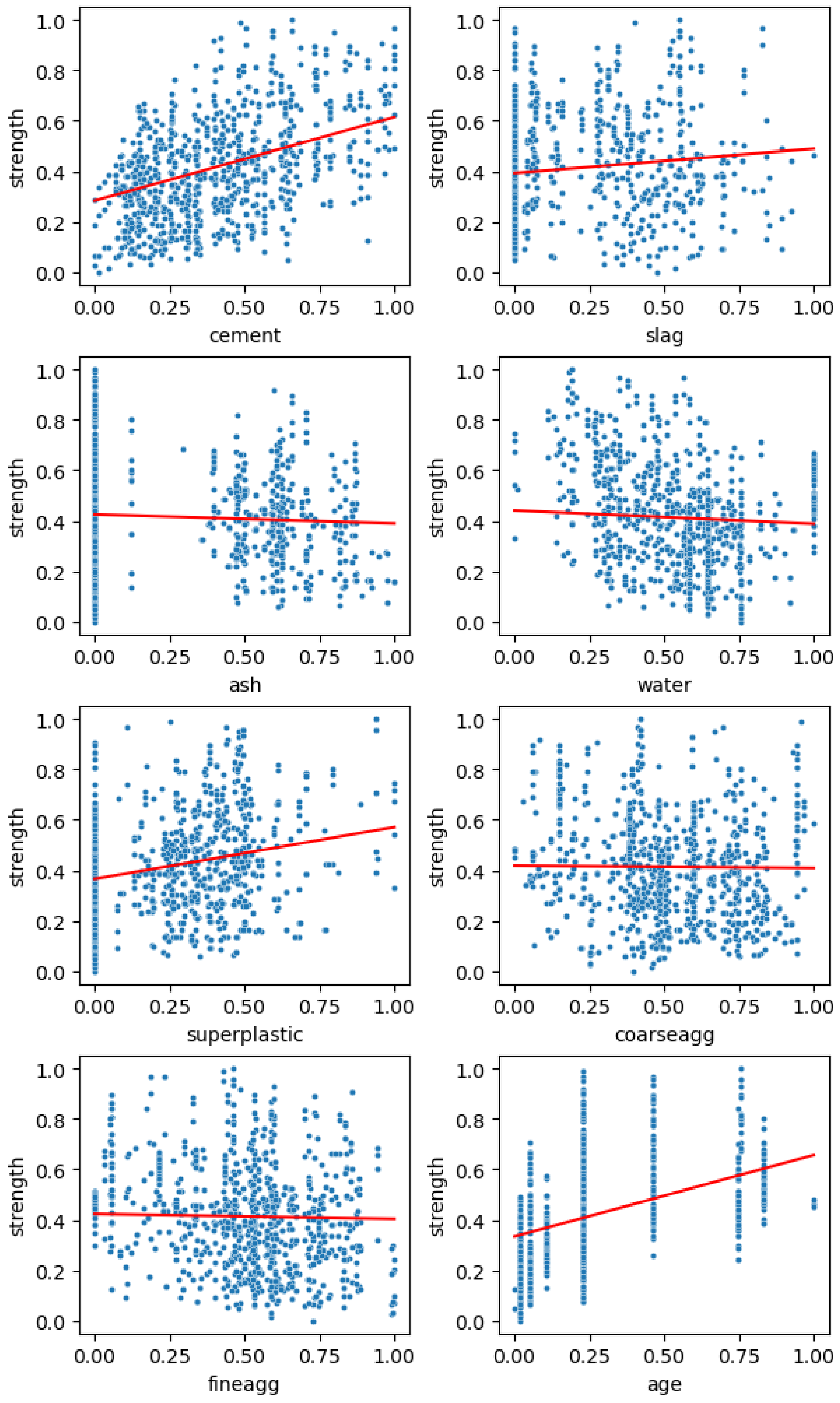
d. 1 Multivariate model on processed data



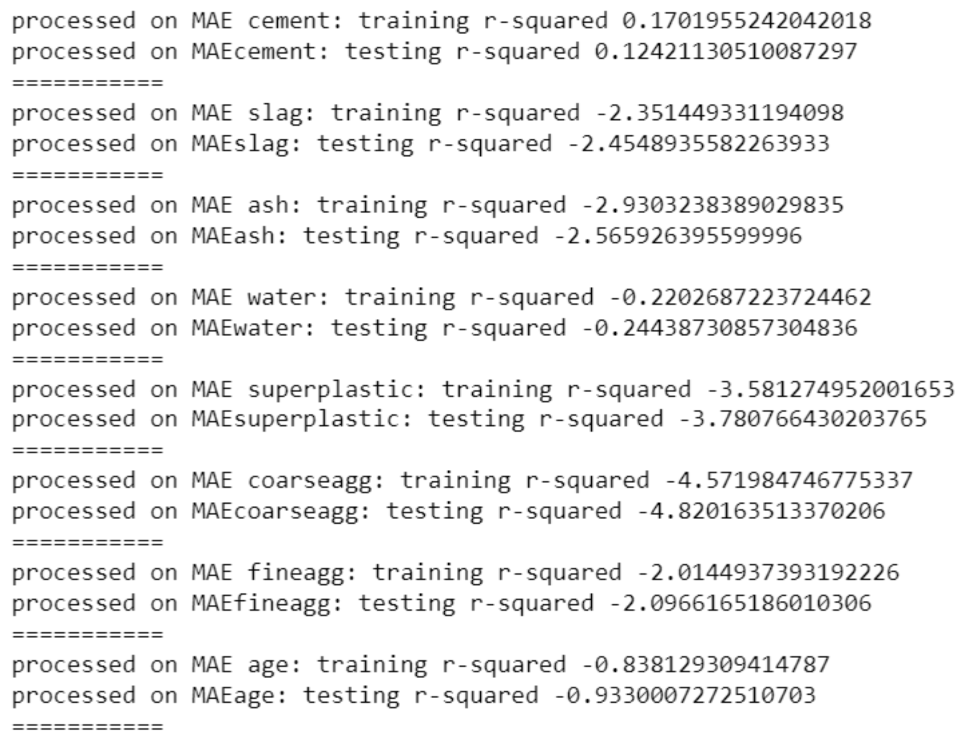
```
multivariate model on processed training data: 0.6902221399530648

multivariate model on processed testing data: 0.695136447985018
```

The multivariate model on processed data is also extremely better than any univariate models. Even better than the raw data one.

2. Plot univariate models
   In the graph, it shows that the univariate linear model performs poorly is pretty normal, given that the data for every feature is scatter around. It is difficult to use a single linear model to explain them.


strength
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00
cement
slag
ash
water
superplastic
coarseagg
fineagg
age

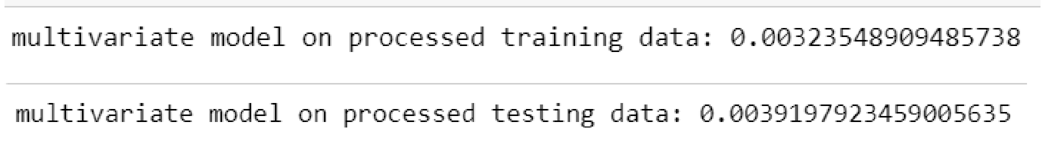
3. Results of MAE on processed data on training v.s. testing set
    a. 8 univariate models



```
processed on MAE cement: training r-squared 0.1701955242042018
processed on MAEcement: testing r-squared 0.12421130510087297
===========
processed on MAE slag: training r-squared -2.351449331194098
processed on MAEslag: testing r-squared -2.4548935582263933
===========
processed on MAE ash: training r-squared -2.9303238389029835
processed on MAEash: testing r-squared -2.565926395599996
===========
processed on MAE water: training r-squared -0.2202687223724462
processed on MAEwater: testing r-squared -0.24438730857304836
===========
processed on MAE superplastic: training r-squared -3.581274952001653
processed on MAEsuperplastic: testing r-squared -3.780766430203765
===========
processed on MAE coarseagg: training r-squared -4.571984746775337
processed on MAEcoarseagg: testing r-squared -4.820163513370206
===========
processed on MAE fineagg: training r-squared -2.0144937393192226
processed on MAEfineagg: testing r-squared -2.0966165186010306
===========
processed on MAE age: training r-squared -0.838129309414787
processed on MAEage: testing r-squared -0.9330007272510703
===========
```

This performs equally bad just like using MSE as loss function.

    b. 1 multivariate model

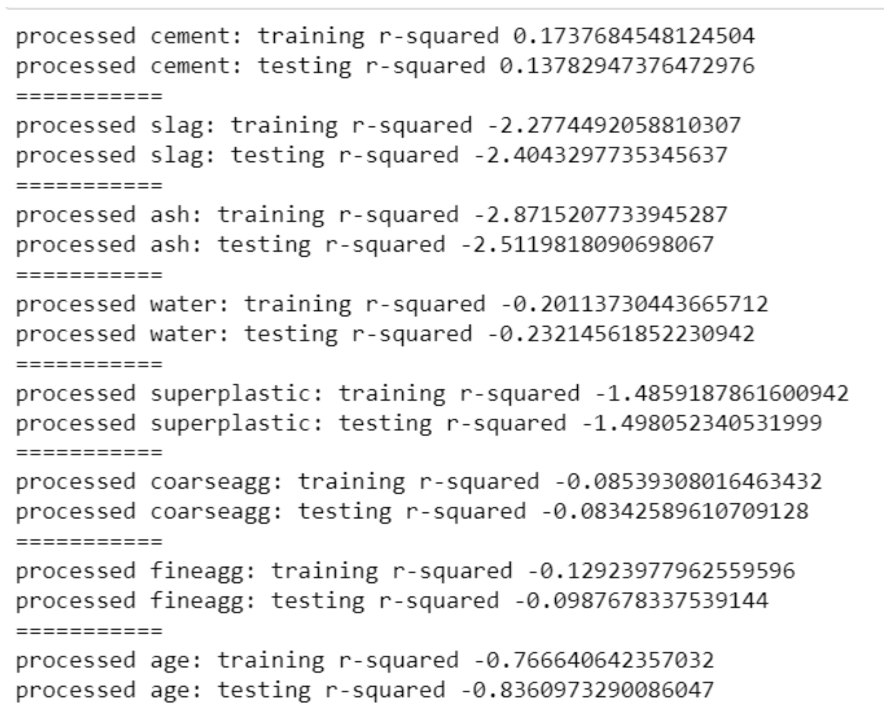


```
multivariate model on processed training data: 0.00323548909485738

multivariate model on processed testing data: 0.0039197923459005635
```

The multivariate model is slightly better than any 8 univariate models, however, the results are just really poor.

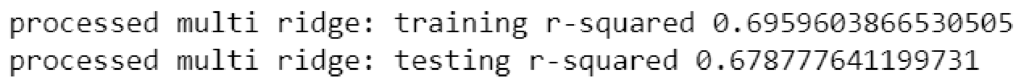
4. Results of Ridge regression on processed data on training v.s. testing set
    a. 8 univariate models

The 8 univariate models all perform poorly using ridge regardless of training set or testing set.



```
processed cement: training r-squared 0.1737684548124504
processed cement: testing r-squared 0.13782947376472976
===========
processed slag: training r-squared -2.2774492058810307
processed slag: testing r-squared -2.4043297735345637
===========
processed ash: training r-squared -2.8715207733945287
processed ash: testing r-squared -2.5119818090698067
===========
processed water: training r-squared -0.20113730443665712
processed water: testing r-squared -0.23214561852230942
===========
processed superplastic: training r-squared -1.4859187861600942
processed superplastic: testing r-squared -1.498052340531999
===========
processed coarseagg: training r-squared -0.08539308016463432
processed coarseagg: testing r-squared -0.08342589610709128
===========
processed fineagg: training r-squared -0.12923977962559596
processed fineagg: testing r-squared -0.0987678337539144
===========
processed age: training r-squared -0.766640642357032
processed age: testing r-squared -0.8360973290086047
```

b. 1 Multivariate model



```
processed multi ridge: training r-squared 0.6959603866530505
processed multi ridge: testing r-squared 0.678777641199731
```

However, for multivariate model, the model performs extremely well compare to univariate models.

Discussion

1. Compare and contrast
   a. Did the same model perform equally to training set and testing set?
      In my case, yes. For example, for univariate models, it occasionally performs better using different loss functions. But overall, it performs pretty poor on every type of loss function and regardless on training set or testing set. On the other hand, multivariate model generally perform really well on MSE, MAE, and ridge regression. Especially with processed data, and overall the performance on training set is similar to testing set.
   b. Did different model take longer to train or require different hyperparameters?
      Yes, model using raw data requires smaller learning rate so that the weight would not overflow. Especially, I have encountered multiple times that the model with raw data tends to have the weight bouncing back and forth between two values, thus, I need to tune the learning rate into a smaller values to make it converge.
   c. How did preprocessed data change the results?

Preprocessed data overall perform better than raw data, given that it handles the outliers and normalize the data, so that the model can read the value equally and easier.

d. What factor contributes the most?
   In my opinion, the only features that make the most influence would be the cement. It constantly have the best performance on models. So, I would say the higher the value cement has, it is more likely to have stronger concrete strength.
e. MAE and Ridge
   MAE overall did not change the result that much. I think it is because it has similar loss calculation to MSE.
   Ridge make multivariate model perform better. This may be because there may be high correlations between predictor variables, which can cause instability in the estimates of the regression coefficients. Ridge regression can help to reduce the impact of multicollinearity by adding a penalty term to the regression equation that shrinks the coefficients towards zero.