CSE514 – Fall 2022 Programming Assignment 1

Sterling Lech – 505660

Introduction:

For this programming assignment, I was given 1030 total data samples with 9 features per sample regarding concrete mixtures. Features within concrete mixture samples such as: cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, fine aggregate, age of the mixture, and the concrete compressive strength of the mixture(the response). The practical application of this assignment is to draw insight on this data by formulating trained univariate and multivariate solutions using the independent features and the response feature of these concrete mixture samples. These models are trained by running a Gradient Descent(not stochastic)2 algorithm that utilizes Mean Squared Error as the loss function. Most of the samples will be used to train our univariate and multivariate models, then the trained models will be applied to a small portion of the samples as testing data.

When running my gradient descent algorithms for my univariate and multivariate analysis', there was one major consideration for choosing my learning rates. The R2 score printed at the end of each run. The better the R2 score, the better I believed my hyper parameters were. I did not use a stopping criterion for either of these models.

Univariate Analysis Pseudocode:

```
Import packages

function calculate_r2:
       Calculate R2 score between predicted y values and real y values


function compute_error_for_line_given_points(b, m, points):
    calculate the total error for all the points
    return total_error/total number of points = Mean Squared Error


function step_gradient(b_current, m_current, points, learning_rate_b,
learning_rate_m):
    # Starting points
    b_gradient = 0
    m_gradient = 0
    n = float(len(points))
    for all points(x and y):
       calculate the new m_gradient and b_gradients with the partial deriva
               -tives of MSE
```

```
    Calculate the new b and new m using the difference between their original
values and the product of the learning rate and gradients calculated with the
partial derivative of MSE above

    return [new_b, new_m]


function gradient_descent_runner(points, starting_b, starting_m,
learning_rate_b, learning_rate_m, num_iterations):
    b = starting_b
    m = starting_m
    # Gradient descent
    for the total number of iterations:
        # continuously update b and m using the partial derivative of MSE
        step_gradient(b, m)
    return [b, m]


function abline(slope, intercept, x_vals):
    Plot a line for the data using the slope and intercept given by our model
after running gradient descent

    return (calculate y vals to be used for the R2 score)

function run(): #RUNS THE PROGRAM
    # Collect our data, randomize it, and separate it into training and
testing ndarrays:
    Extract our data from the spreadsheets
    Randomize the data
    Separate it into x and y data
    Separate it into training and testing data

    Set our hyper parameter that will train our model(iterations, learning
rates, starting values

    # Print error prior to starting gradient descent:
    compute_error_for_line_given_points(initial values of m and b, training
data)
    # Train our model:
    gradient_descent_runner(initial m and b values)
# Print error after gradient descent:
    compute_error_for_line_given_points(final values of m and b, training
data)



    *****Use MatPlot lib to plot the testing data on a scatter plot along
with the linear fit*****

    *****Use MatPlot lib to plot the training data on a scatter plot along
with the linear fit*****

    calculate_r2(y training, y predicted)
    calculate_r2(y testing, y predicted)
```

```
if __name__ == '__main__':
    run()
```

Multivariate Analysis Pseudocode:

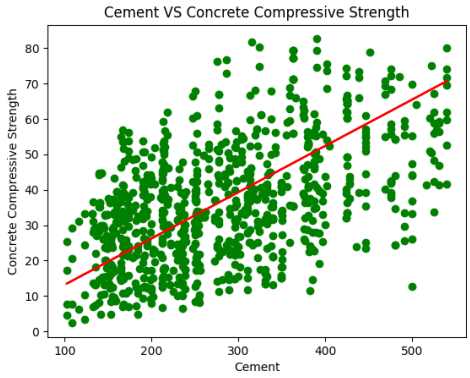
```
function calculate_r2(y, X, theta_vector, column_string_index):

    Create an ndarray from the Xdata by the column parameter passed

    Calculate a predicted y ndarray from the product of the passed theta
value and the Xdata column

    return R2 score

    return metrics.r2_score(y_testing_param, predicted_y_array)

function cost_function(X_param, y_param, theta_param):
    Get the size of the y array
    calculate the mean standard error and the partial derivative of mean
standard error with respect to theta
    return mse, cost, error

function gradient_descent(X_param, y_param, theta_param, alpha, iters):

    cost_array = np.zeros(iters)
    m = y_param.size

    For the total number of iterations used to train the model, run gradient
descent using the cost functions calculation of the error.

    Calculate the new thetas after the cost function is ran with the new
value of error

    return theta_param, cost_array

function run():

    Extract the data from the spreadsheet, split it into X and Y data, split
it into testing and training data

    Normalize the training and testing features

    Add a 1's column to the training features

    Set alpha and the number of iterations to the train the model

    Create a vector of thetas that our model will train for

    # Find the initial cost with the cost_function
    cost_function(X_training_data, y_training_data, theta_vector)

    Print the initial values from the cost function
```

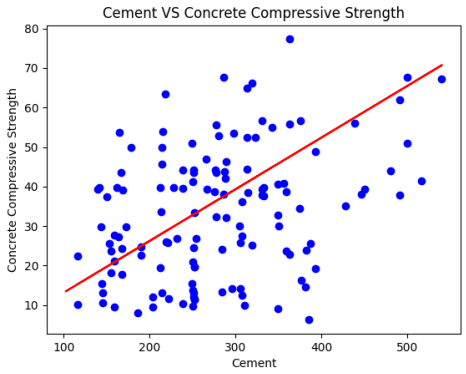
```
    # Run gradient descent to determine the theta values
    gradient_descent(X_training, y_training, theta_vector, alpha, iterations)



    Print the R2 scores for all the points in the testing data with their
corresponding thetas using calculate_r2(…).




if __name__ == "__main__":
    run()
```

Results:

Univariate:

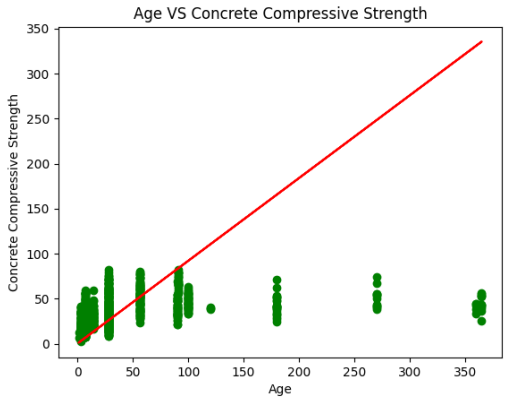
- The green scatterplot is training data used to train the model (900 random samples)
- The blue scatterplot is testing data used to test the integrity of the model (130 random samples)
- Following the plots is the variance explained of both the training data and the testing data. Most of which are negative. Likely because my code is not optimized for something like stopping criteria. Although, even with the use of separate training parameters for the slope and intercept values, I still could not get positive variance explained values.

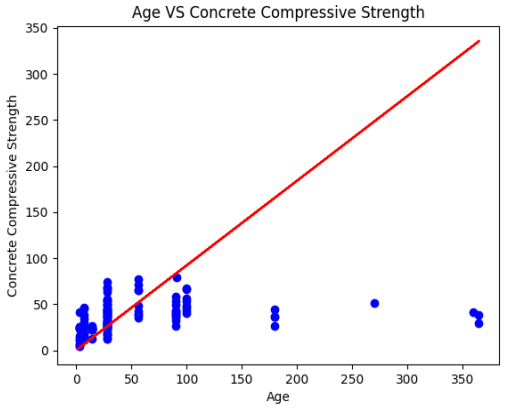




-

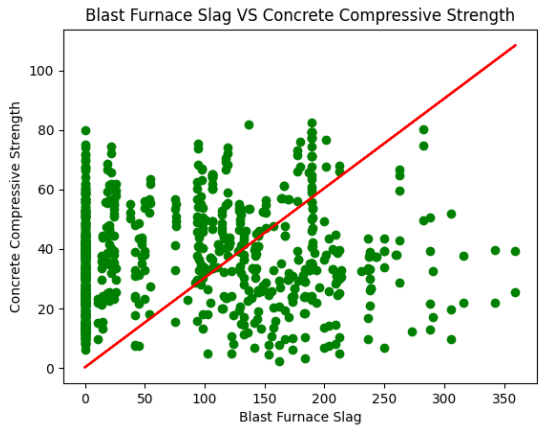
```
The R2 value for the Training Y-Values:  0.15587621276979702
The R2 value for the Testing Y-Values:  0.13521175770108496
```

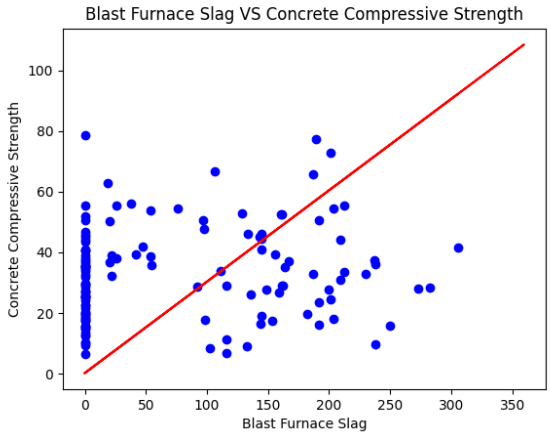




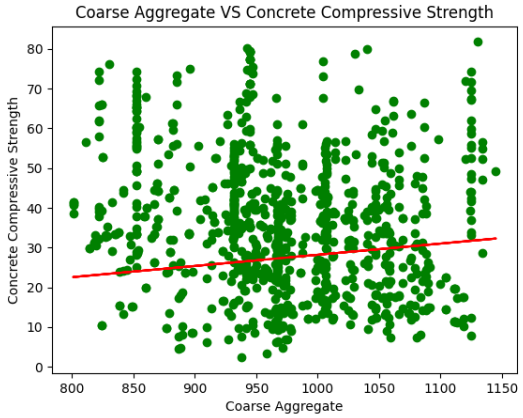

```
The R2 value for the Training Y-Values:  -9.765960692481478
```

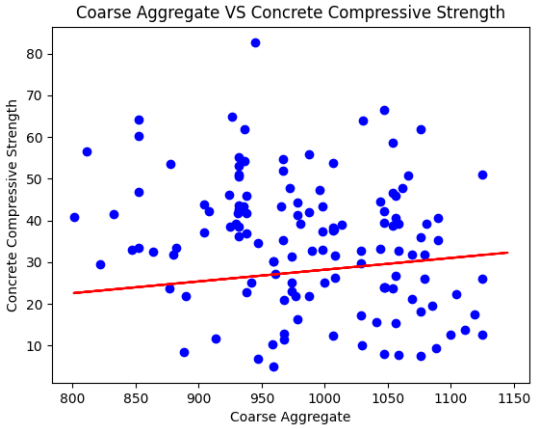
**I was not able to get a variance explained because I was getting NaN values



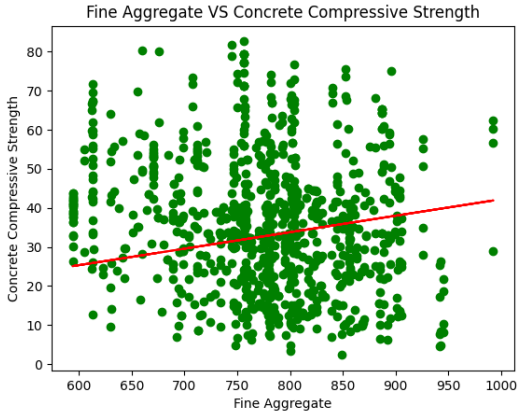


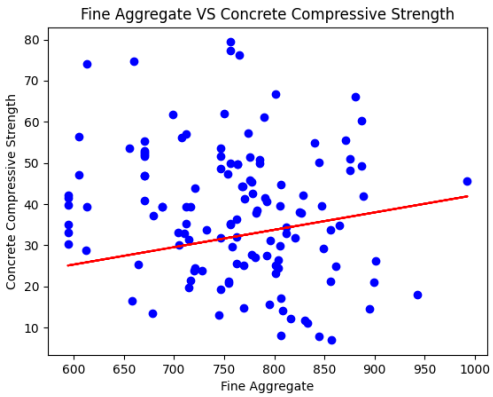

```
The R2 value for the Training Y-Values:  -2.6160338841122535
The R2 value for the Testing Y-Values:  -2.9769038249106936
```



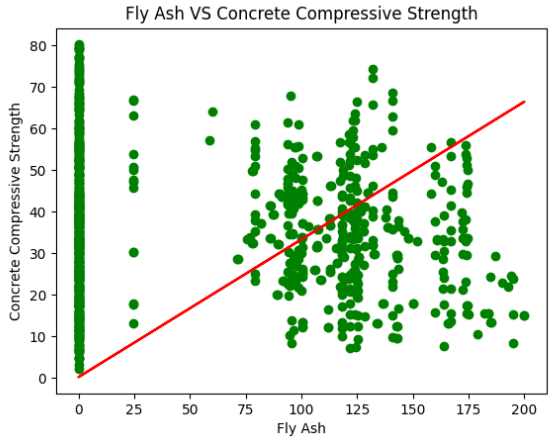


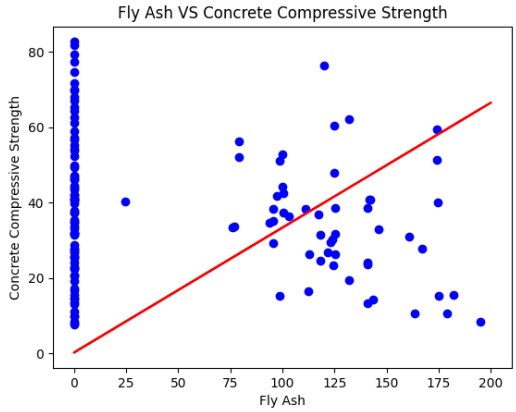

```
The R2 value for the Training Y-Values:  -0.31466824244452263
The R2 value for the Testing Y-Values:  -0.2991353856216852
```



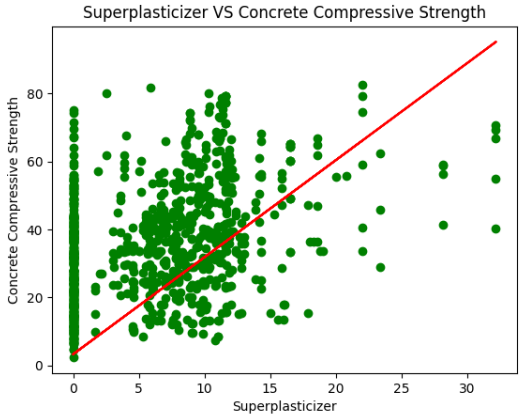


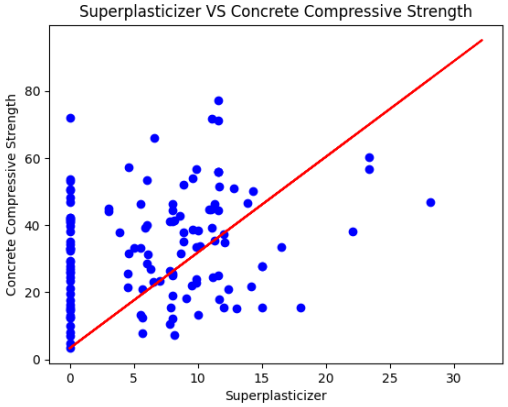

```
The R2 value for the Training Y-Values:  -0.13219565123336374
The R2 value for the Testing Y-Values:  -0.26858971071831395
```



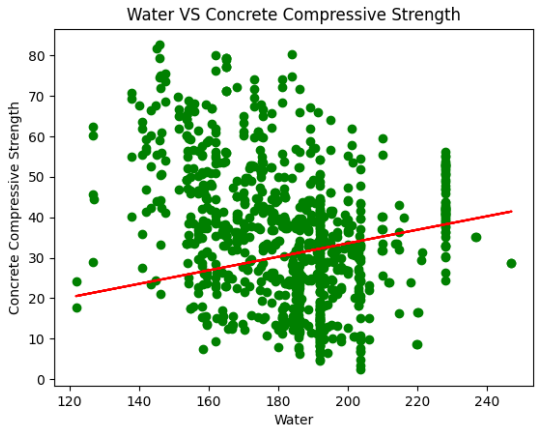


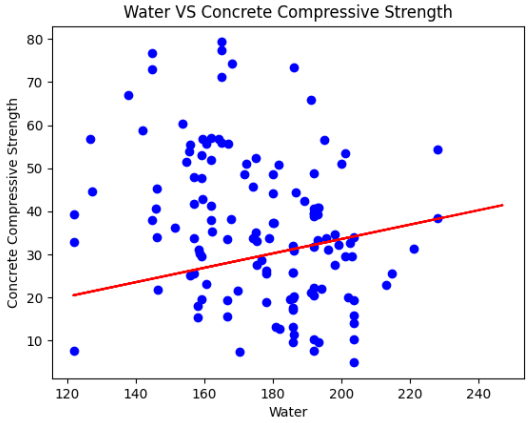

```
The R2 value for the Training Y-Values:  -2.9789934757191654
The R2 value for the Testing Y-Values:  -3.083342163209765
```






```
The R2 value for the Training Y-Values:  -1.072004818292847
The R2 value for the Testing Y-Values:  -1.184392571463324
```






```
The R2 value for the Training Y-Values:  -0.2626694833489587
The R2 value for the Testing Y-Values:  -0.3213624596469016
```

Multivariate:

```
With initial theta values of [0. 0. 0. 0. 0. 0. 0. 0. 0.], cost error is
809.4984664444441, mse is 809.4984664444444
With final theta values of [36.36269079  7.50272638  4.22152766  1.79253626 -
5.90825282  2.96023798
 -1.30218636 -2.68155142  7.45378828], cost error is 58.56982527396875, mse
is 58.56982527396876
The R2 score for the cement component testing data is: -12.015344513105182
The R2 score for the Blast Furnace Slag component testing data is: -
7.024885485635085
The R2 score for the Fly Ash component testing data is: -7.586437483193402
The R2 score for the Water component testing data is: -7.170835295263322
The R2 score for the Superplasticizer (component 5)(kg in a m^3 mixture)
testing data is: -7.210044661071596
The R2 score for the Coarse Aggregate  (component 6)(kg in a m^3 mixture)
testing data is: -7.277159841776934
The R2 score for the Fine Aggregate (component 7)(kg in a m^3 mixture)
testing data is: -7.090320909710533
```

** Note: I did not get an R2 score for Age because I was receiving an NaN output when getting the R2 score

** Note: I was not able to get the R2 scores to print for the training data

Discussion:

Based on my R2 values for the training and testing data, the only model that was somewhat accurate was the Concrete Component VS Concrete Compressive Strength univariate model. The aforementioned model was somewhat accurate at predicting both the training and the testing data based on the results. Otherwise, I received negative R2 scores for just about every univariate model and all the R2 scores for my multivariate model. All the models that did receive negative R2 scores seemed to have relatively close R2 scores though.

Many of the models performance between the training and testing data hardly varied. Looking at the R2 scores for the testing and training data respectively, they are usually within .1 of one another. The Cement VS Concrete Compressive Strength Univariate model was probably the easiest to train out of all of them. I didn't even have to specify separate learning rates for the m_gradient and b_gradient values like I did for all the other Univariate models. Otherwise I used an array of different learning parameters for all the other Univariate models to bring the R2 scores as close to 0 as I possibly could. Almost all of the models are trained with between 6000-10000 iterations which take longer to train as the iterations are increased. Most of the univariate models use learning rates between 10^-4 and 10^-8 while the multivariate model uses 10^-4.

My results didn't pan out that well in order for me to draw meaningful conclusions on the data from my univariate and multivariate models due to their negative variance explained

scores(meaning a flat line to the data is better than mine). But I would say that if you wanted to create the hardest concrete mixture you would want to have anywhere between 360-390 kg in a m^3 mixture and a water component of 145-165 kg in a m^3 mixture to have the highest concrete compressive strength response. The rest of the features don't seem to be the most accurate predictors of concrete compressive strength.