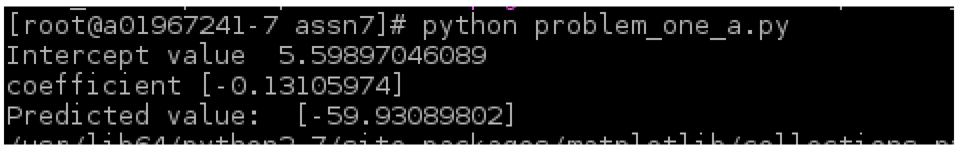
Assn 7: The end is near

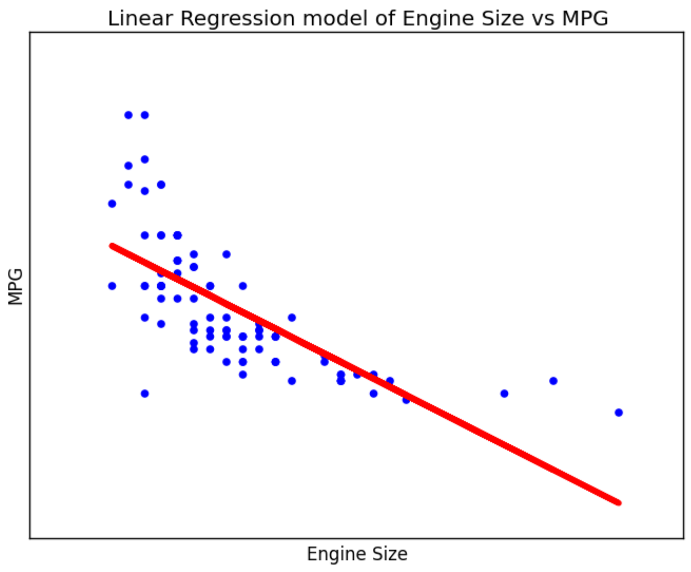
Austin Derbique A01967241

1. a) Using simple linear regression to show "interesting" relationship between two variables presented in the data.

For this problem, I looked at City MPG and Engine Size. Although they are not obvious, it makes sense that the larger the engine size, the worse the gas mileage. The code for is is in the script called problem_one_a.py. This can be seen here:



```
[root@a01967241-7 assn7]# python problem_one_a.py
Intercept value  5.59897046089
coefficient [-0.13105974]
Predicted value:  [-59.93089802]
```



1. b) 8 or more independent variables using multiple linear regression to predict the value of the dependent variable from the independent variables.

For this part, I decided to use the following:

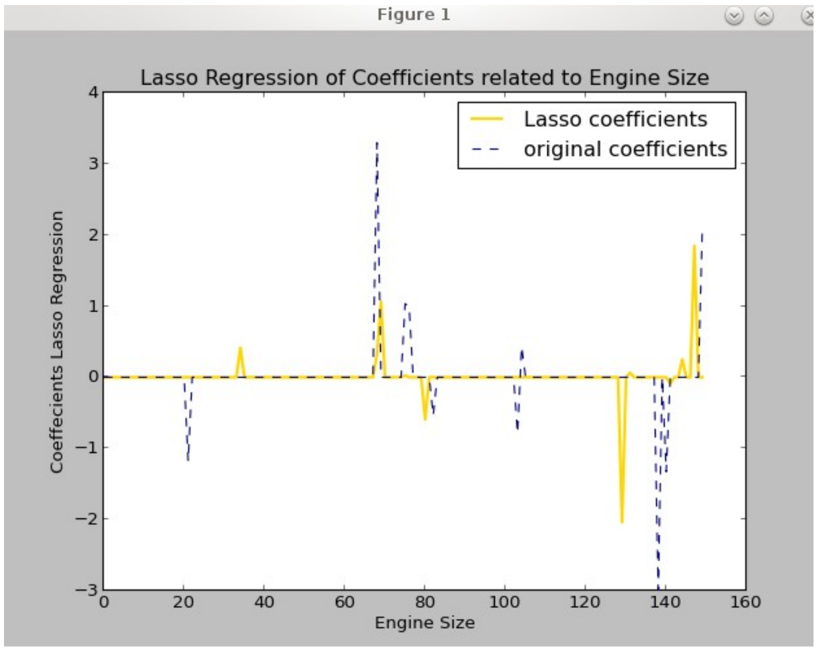
Independent Variable: Engine Size

Dependent 1-8: MinimumPrice, CityMPG, NumberOfCylinders, Horsepower, FuelTankCapacity, LuggageCapacity, EngineRevPerMile, and Weight.

My experience with this part only cluttered up the graph. I could not tell that it helped make the result more accurate. My theory for this is somethings like Weight being a factor and CityMPG also being a factor causes inverses. Meaning they contradict each other and confuse the regression. It appears that for this to work properly, all the data must be going the same direction. An example of this would be weight of the vehicle and Luggage capacity. Both of them are proportional to one another. That is why this model did not appear to perform better. Code is available in multipleregression.py.

1. c) Lasso Regression

In this code, computes the sum of squares and r-squared values. The r-squared value is the y axis. This result appears to be similar to B as the Lasso Regression performs both variable selection and regularization in order to enhance accuracy. Or in this case, recognize that the data is not producing a very accurate graph. As previously mentioned in 1.b, I believe this is due to some fields being inverses of each other, but being trained together as if they are the same. This could be because of the way I wrote my code. Fixing that would likely fix the issue and make for a better r-squared value. See problem_one_c.py for code.

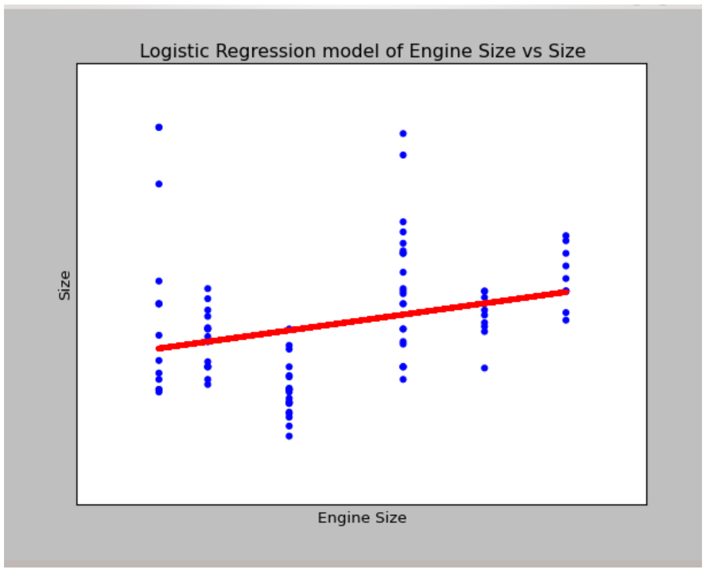


1. d) Logistic Regression and Horsepower

In order to predict if a car is compact or not, I used certain features such as weight, the inverse of MPG, and horse power. Knowing that cars that get good gas mileage, do not weigh much, and have small horse power, it is likely to assume that they are compact. Then, using a threshold, determine whether the car is compact or not. This is what I tried to do for my algorithm in 1.d but it was not properly working. I'd image that if properly tuned, the accuracy would be close to 100% and precision and recall would be high as well. Potential problems with this logistic regression is determining the difference between a compact car and a sports car as many sports cars share the same characteristics as a compact car. This might be able to be circumnavigated by choosing engine size. problem_one_d.py

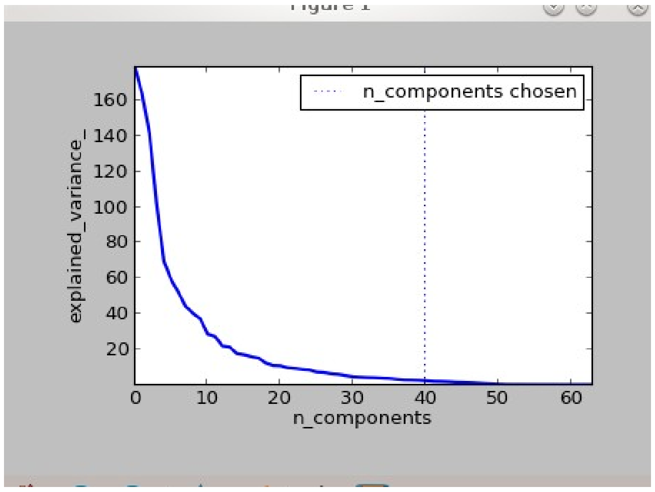
1. e) Enhanced Logistic Regression

For this part, I converted the “Type” into a value to represent what kind of car it is. The larger cars were assigned larger values. The smaller type cars were assigned smaller values. For this case, Large vehicles were 30, midsize were 20, small were 13, and compact were 8. Vans were 20. Using this information, I computed the regression information in accordance with engine size. This resulted in a graph showing a correlation between the size of the vehicle and the size of the engine. Code for this can be found at problem_one_e.py and 93cars_e.csv. Because model 1.d was not working entirely, the two models could not be compared. Therefore, TP,TN,FP,FN,accuracy, precision, recall, and F1 measures are not compared to those in part D.



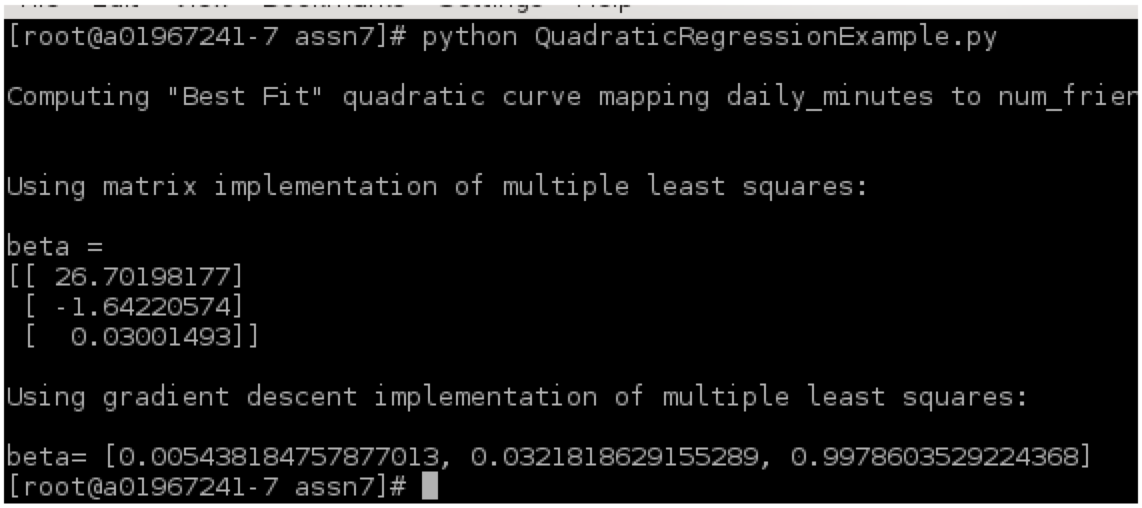
1. f) Principal Components Analysis

The 8 numeric variables I used in this dataset were the same that I used in part B. These were: MinimumPrice, CityMPG, NumberOfCylinders, Horsepower, FuelTankCapacity, LuggageCapacity, EngineRevPerMile, and Weight. Using a principle components analysis, it was determined that as the number of components went up, the model became more refined. As for the linear combinations, the vehicles that would receive high positive values for MPG would be those that are small or compact. Those with high negative values are large size vehicles, vans, and sport vehicles. The ones close to zero are midsize, economy vehicles. What Is interesting is that the sport vehicles.. although they may be small, do not provide good gas mileage as they have large engines. The principal component that distinguishes large vehicles from compact/economy vehicles was fuel tank capacity. Larger vehicles have a larger fuel tank. These results were found from both inference, and running the script live in idle. A screenshot below shows the Principal component analysis and variance of that set.



2. Quadratic Regression Example

As seen below, the results are significantly different. This is likely due because of how fundamentally different gradient descent is from the matrices calculations. Gradient descent is first order iterative optimization for finding the minimum of the function. This differs from the matrices calculations which iterate through the matrix and perform calculations as they go. If I were to guess, it could be because of an error in the gradient descent calculation that gets magnified when iterated through. Something interesting is that the values are consistently the same when run over and over again. This means that both functions are stable, just one is likely to have that possible error that gets carried through all the way to the result.



```
[root@a01967241-7 assn7]# python QuadraticRegressionExample.py

Computing "Best Fit" quadratic curve mapping daily_minutes to num_frien


Using matrix implementation of multiple least squares:

beta =
[[ 26.70198177]
 [ -1.64220574]
 [  0.03001493]]

Using gradient descent implementation of multiple least squares:

beta= [0.005438184757877013, 0.0321818629155289, 0.9978603529224368]
[root@a01967241-7 assn7]# 
```