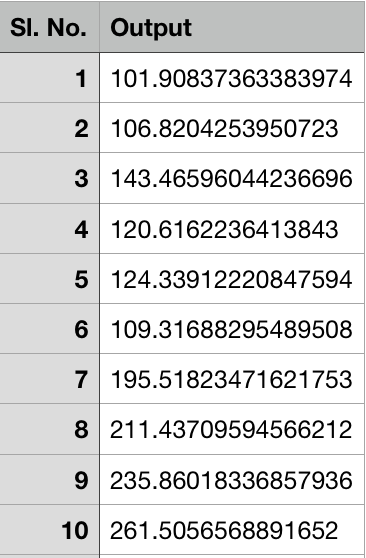
**Project 1**

****

**By: Keith Simmons**

**Kaggle Name: Keith Simmons**

**Description:**

This project, I am provided with medical data where the input variables are corresponding to physical, physiological, and blood related measurements.  The target variable corresponds to the level of diabetic condition in the patient.  The training data contains 64 attributes and 242 observations, and the testing data contains 64 attributes and 200 observations.  The goal of this project is to create a regression with the lowest mean squared error on the testing data.

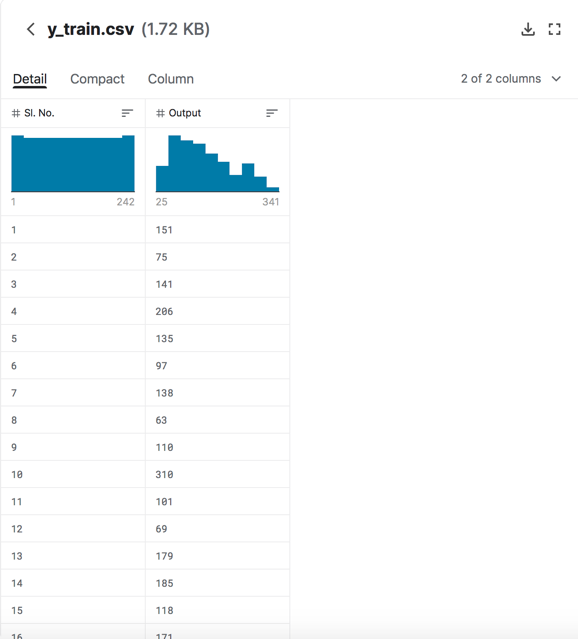


Figure 1:y\_train dataset

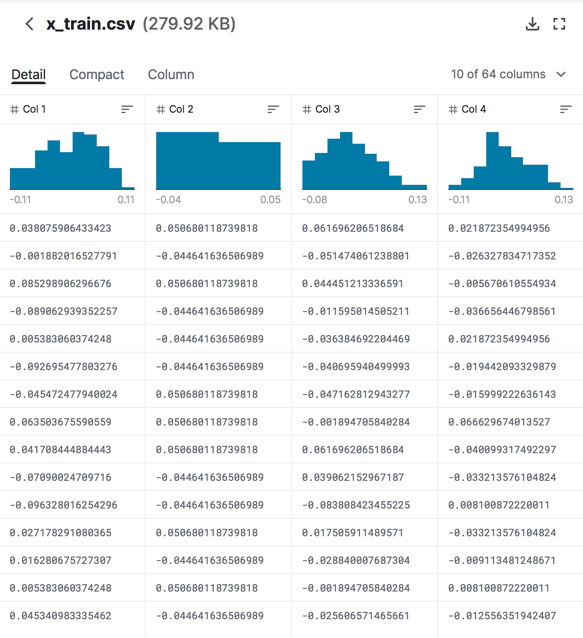


Figure 2:x\_train dataset

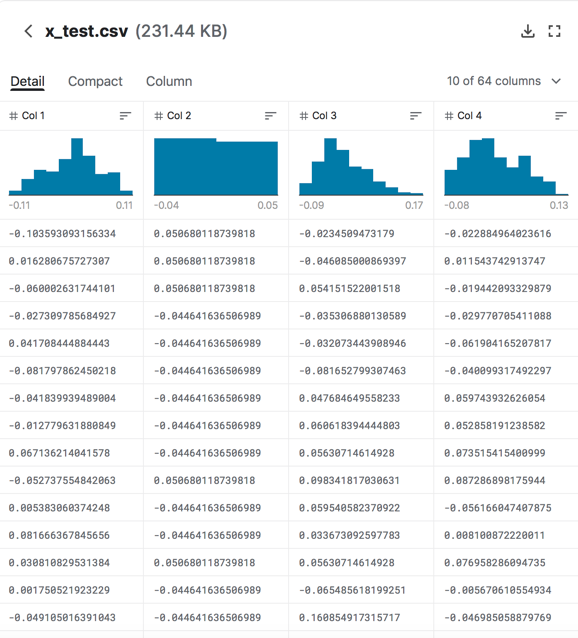


Figure 3:x\_test dataset

**Pre-Processing:**

To begin, I inspected the information of the data using the info method to determine that there are no missing values.  Then, I used Sklearn’s StandardScaler function which standardizes the values of the attributes by removing the mean and scaling the data to unit variance.  This ensures that all the predictors are measured on the same scale. Finally, I removed the highly correlated predictors, which removes multicollinearity from the model and starts to remove some of the predictors from the model.  When removing predictors for high correlation with other predictors, I did so with a threshold of 0.6 correlation, as during the hyperparameter tuning phase of testing, I discovered that the correlation threshold of 0.6 gave models the best MSE.  Further, I indexed through the columns in reverse order and removed the predictor with a higher column number.  I found that doing this gave the best result, as opposed to ascending through the column numbers and removing the lower column numbers.

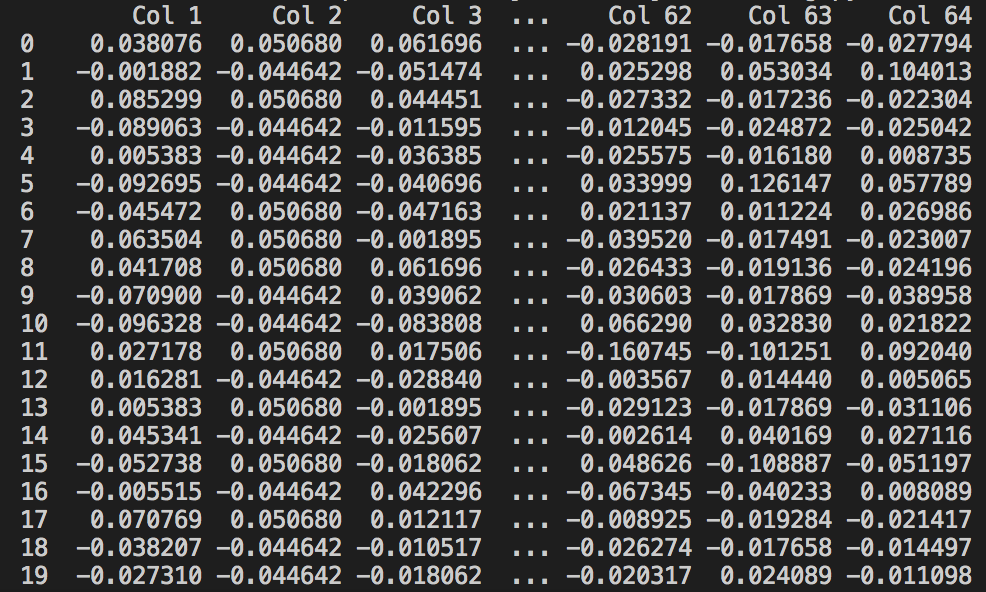


Figure 4:x\_train before standardization

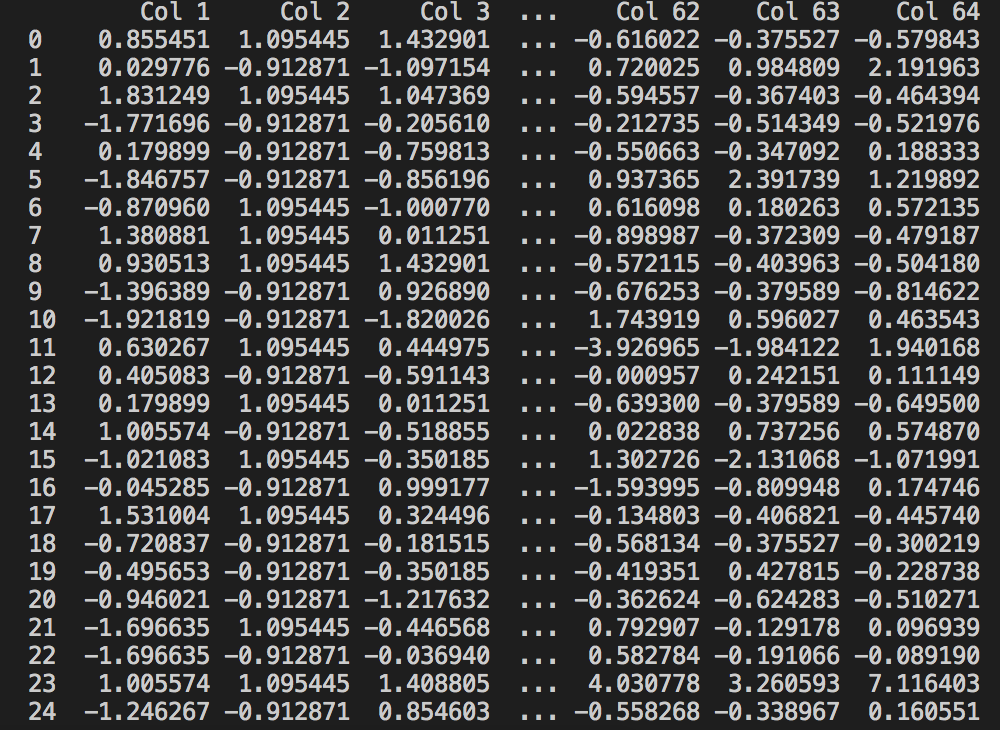


Figure 5:x\_train after standarization

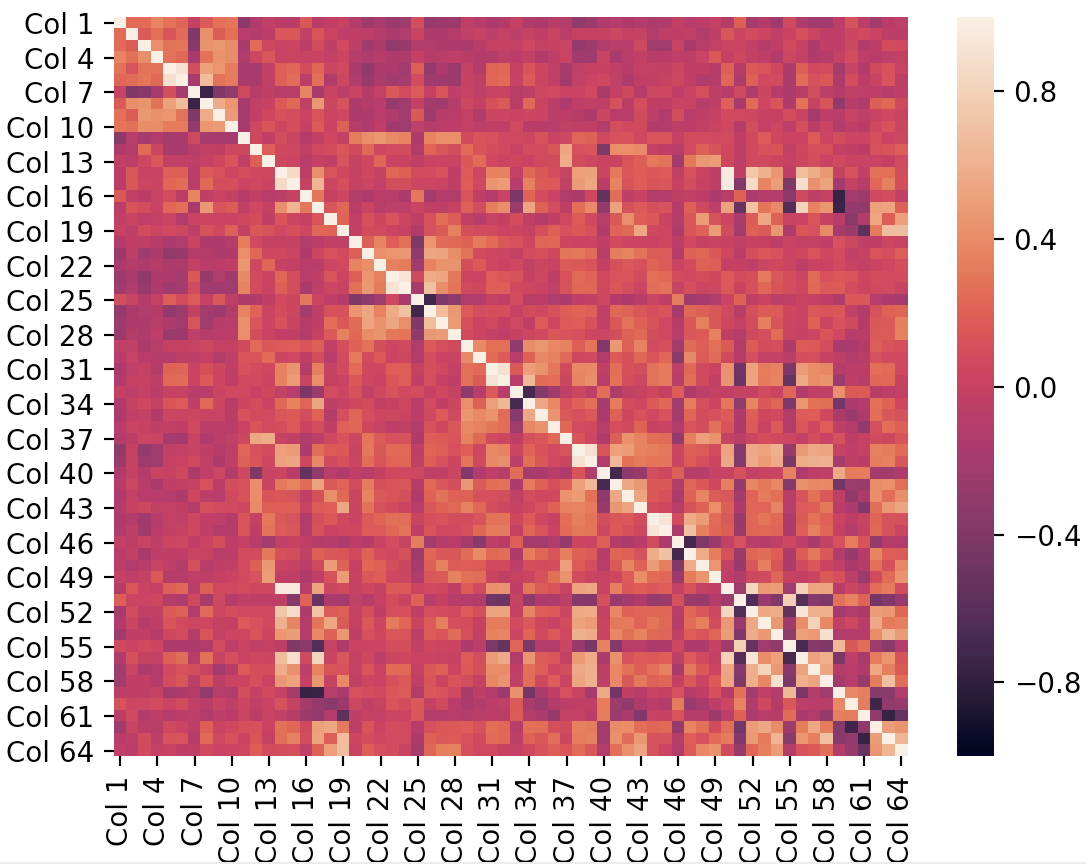


Figure 6:correlation matrix of x\_train before dropping highly correlated predictors

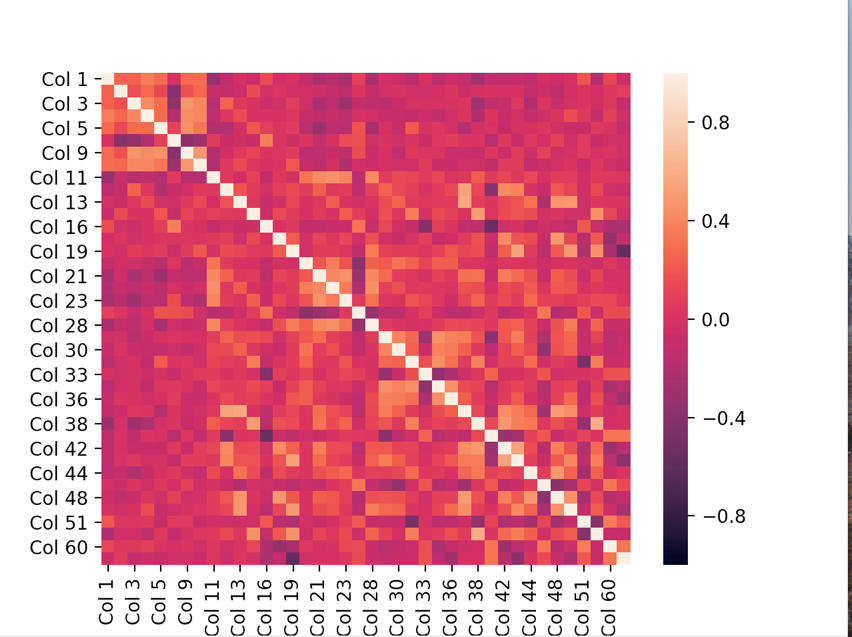


Figure 7:correlation matrix of x\_train after dropping highly correlated predictors

**Feature Selection:**

To choose the best predictors, I utilized Sklearn’s SelectKBest function to select the best k predictors for a model with k predictors.  Then, for each k, I constructed both a Linear Regression model and Lasso model with the best k predictors chosen by the SelectKBest function.  With these models, I found the Adjusted R^2, AIC, and BIC to score each of the models based on how well it can predict the training data.  Each of the criteria gave a different amount of best k to use, but the BIC very harshly penalizes amount of predictors and was quite often resulting with the best model being a simple linear regression, so I chose not to use BIC further, as I felt like it penalized the amount of predictors too harshly in this scenario.

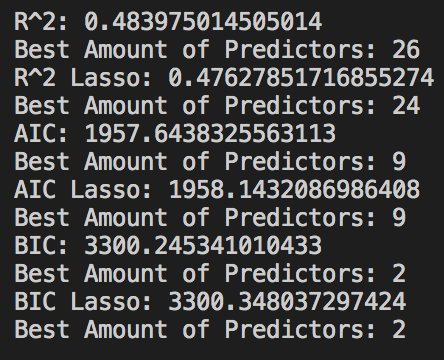


Figure 8:Scoring Criteria for best k predictors

**Model Building and Comparison:**

At this point, I am left with many different models to choose from: Linear Regression with Adjusted R^2, Linear Regression with AIC, Lasso with Adjusted R^2, and Lasso with AIC.  In addition to this, I also tested ten alpha levels for the Lasso models to further discover what a good alpha could be for the Lasso model.  To select the best one, I split the training data further into new training and testing data, as the original testing data has labels to create a scoring system for the models.  I trained all of the models on the new training data and tested it on the new testing data to obtain a mean squared error for each model.  Then, I repeated this 1000 times to receive the mean MSE for each model on unseen data.  The best model found was a Lasso model with an alpha of 0.5 using the 9 best predictors as found using the AIC scoring criterion.

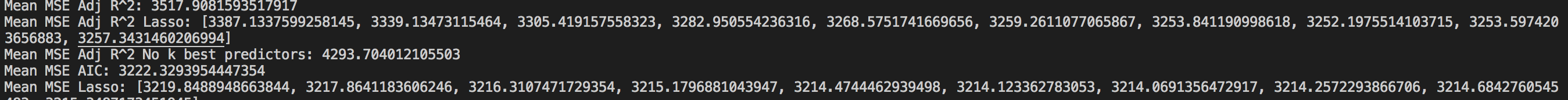


Figure 9:Mean MSE's for models

**Hyperparameter Setting and Tuning:**

One hyperparameter, alpha off the Lasso model, was found previously by testing 10 alphas for the Lasso model and then, by looking at the differences in MSE produced by the Lasso models with varying alphas, it is easy to determine which is the best by looking for the lowest MSE.  Further, I had to tune the correlation threshold, as aforementioned in Pre-Processing.  I did this by changing the threshold value and then running the program again.  To begin, I started with a threshold of 0.5, and realized that as I lowered the number, the resulting MSE’s went up and as I increased the threshold, the resulting MSE’s went up.  This reached a limit at 0.6 when changing the value in both directions would increase the MSE’s, so I figured that this was the best correlation threshold.  Further, I noticed that defaulting to dropping the low column-numbered predictor when dropping the highly correlated predictors resulted in higher MSE’s than when defaulting to dropping the high column-numbered predictors.

**Performance Evaluation:**

To evaluate the performance of the best model, I first had to standardize the testing data with Sklearn’s StandardScaler, as I did with the training data.  Next, I had to drop all the columns of the testing data that the model did not use.  Then, I used the Lasso, alpha =0.5, AIC model to predict the target variable, which I then put into a Pandas DataFrame and exported the output to a .csv file.  By submitting the file to Kaggle, I can find that the MSE of my resulting model is 3307.61 when it is to predict the true y\_test.

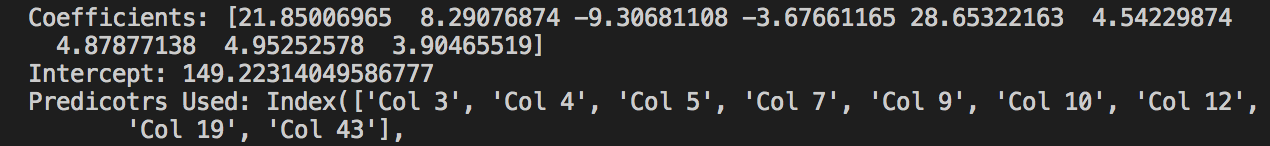


Figure 10:Coefficients, Intercept, and predictors used in final model



Figure 11:Final submission on Kaggle

**Novel Methods:**

One thing that I thought was interesting during my completion of this project was that by testing and training the models on many different subsets of the data and averaging the MSE gave a very close MSE to the one given from Kaggle.  This allowed me to generally know before submitting which model would be the best and which one would improve my previous submission.  By using this method, you can approximate the MSE the model will obtain on unlabeled data.

**Lessons Learned:**

One lesson that I learned is how easy many of the Python libraries make doing linear regression and other techniques in Python.  Having taken Applied Linear Regression, which was taught in R without any libraries, it is very easy to see how useful and effective these libraries are when constructing a regression.  Further, I am seeing how hyperparameter tuning can have a large effect on the results of the model, and thus, are very important when creating the best model possible.