Description

A medical data set is provided under the Data tab along with the target assignment.

The input variables correspond to measurements (physical, physiological, and blood related) for a given patient and the target variable corresponds to the level of diabetic condition in the patient. It contains:

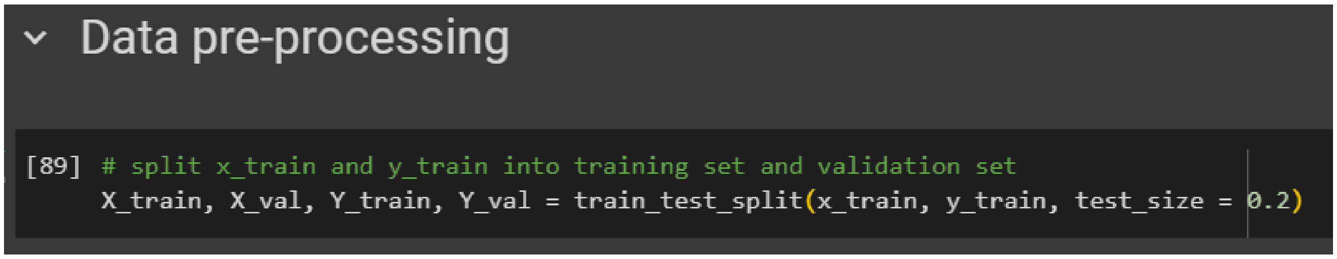
- x train (242 × 64) and y train (242 × 1) for training.
- x test (200 × 64) for testing.

The goal is to use regression methods as outlined in the Project 1 documentation to come up with y test (200x2) i.e. it MUST CONTAIN 2 columns:

Col. 1: The Sl. No. column which corresponds to each row and

Col. 2: Output (which is the predicted output).

Data Pre-Processing


Data pre-processing

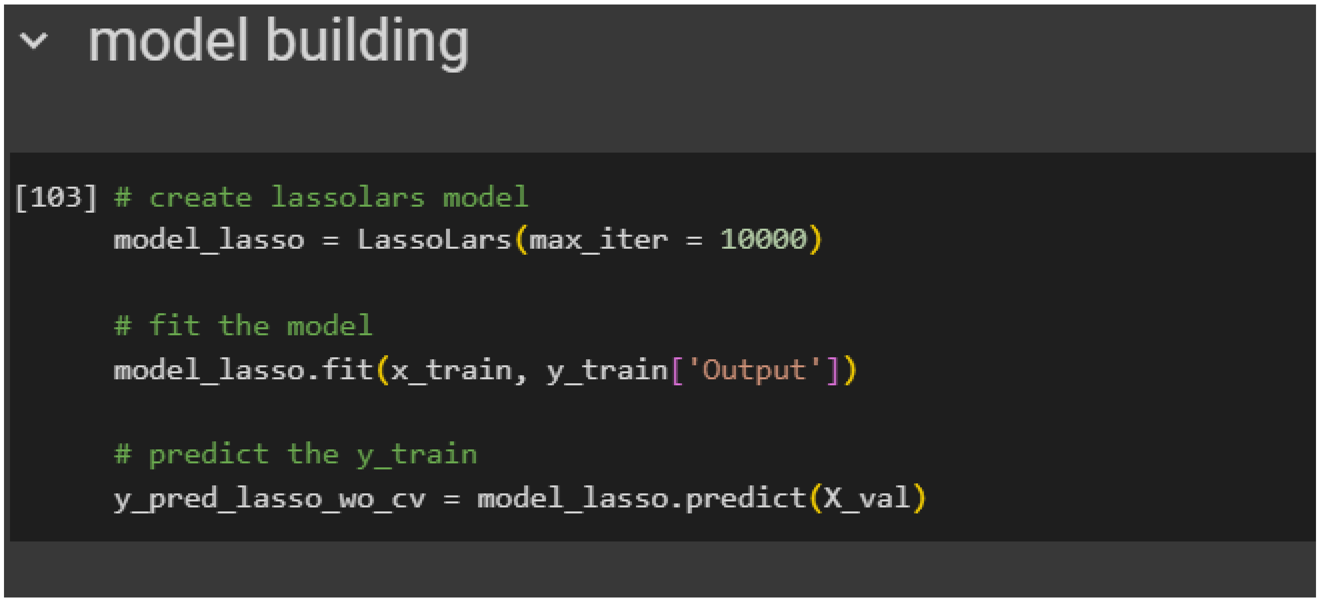
```
[89] # split x_train and y_train into training set and validation set
     X_train, X_val, Y_train, Y_val = train_test_split(x_train, y_train, test_size = 0.2)
```

Even though the initial data is already divided into train and test sets, I would still have to create validation sets for both x and y due. I have to test the model's evaluation using these validation sets because I do not have a true y_test value to compare.

Feature Engineering

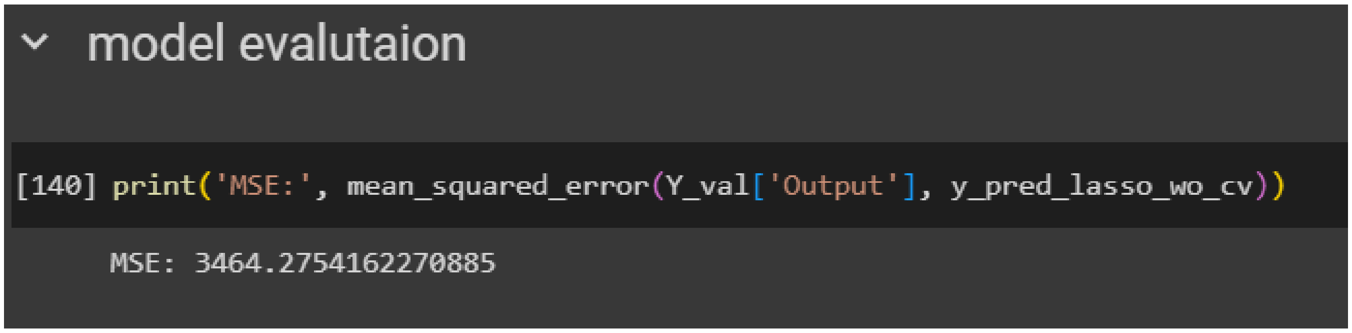
I do need to perform feature engineering since the features of the data are already engineered. Moreover, since going to use a LassoLars algorithm for modeling, the algorithm itself will automatically perform feature selection inside of the algorithm process.

Model Building and Comparison & Performance Evaluation



```
[103] # create lassolars model
      model_lasso = LassoLars(max_iter = 10000)

      # fit the model
      model_lasso.fit(x_train, y_train['Output'])

      # predict the y_train
      y_pred_lasso_wo_cv = model_lasso.predict(X_val)
```

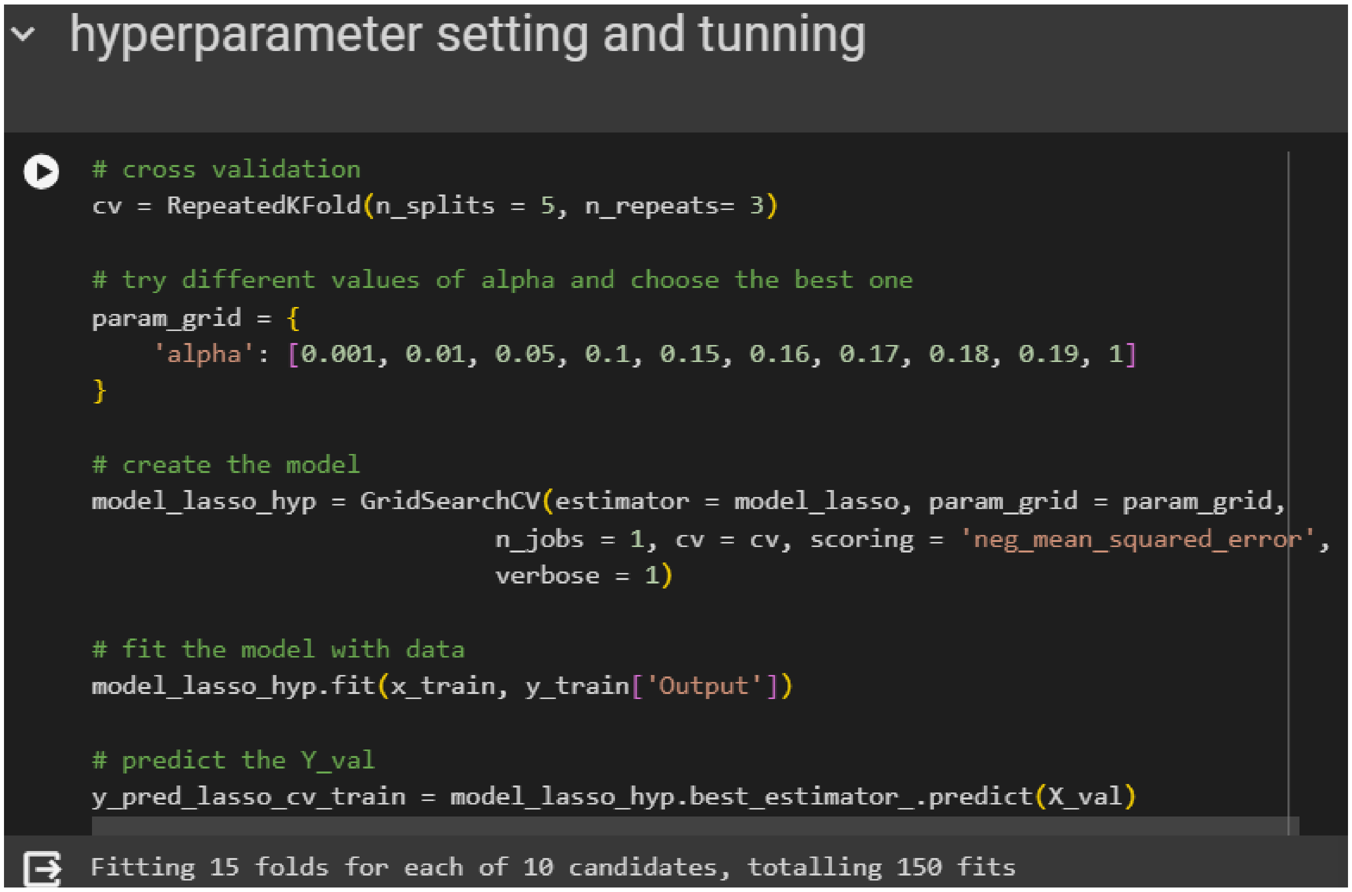
I selected the LassoLars model for machine learning. LassoLars effectively handels many independent features by applying both Lasso regularization and the Least Angle Regression (LARS) algorithm. First, the Lasso algorithm helps in feature selection and shrinking coefficients to zero, which is especially useful when data includes a large number of float independent variables similar to the data, as it aids in reducing complexity and avoiding overfitting. Furthermore, the LARS algorithm enhances computational efficiency, making it more suitable for high-dimensional data.



model evalutaion

```
[140] print('MSE:', mean_squared_error(Y_val['Output'], y_pred_lasso_wo_cv))

      MSE: 3464.2754162270885
```

At first, the model was run without any hyperparameter tuning. Comparing the model's predicted value to Y_val (Y_validation set), the MSE came out to be 2858.2.

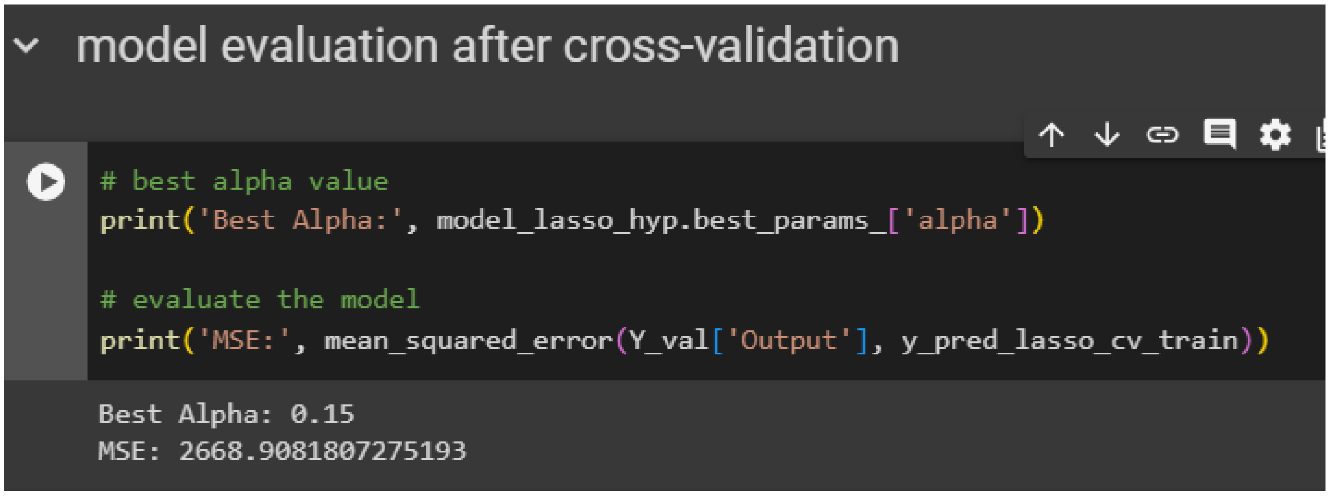


hyperparameter setting and tunning

```
# cross validation
cv = RepeatedKFold(n_splits = 5, n_repeats= 3)

# try different values of alpha and choose the best one
param_grid = {
    'alpha': [0.001, 0.01, 0.05, 0.1, 0.15, 0.16, 0.17, 0.18, 0.19, 1]
}

# create the model
model_lasso_hyp = GridSearchCV(estimator = model_lasso, param_grid = param_grid,
                               n_jobs = 1, cv = cv, scoring = 'neg_mean_squared_error',
                               verbose = 1)

# fit the model with data
model_lasso_hyp.fit(x_train, y_train['Output'])

# predict the Y_val
y_pred_lasso_cv_train = model_lasso_hyp.best_estimator_.predict(X_val)
```

```
Fitting 15 folds for each of 10 candidates, totalling 150 fits
```

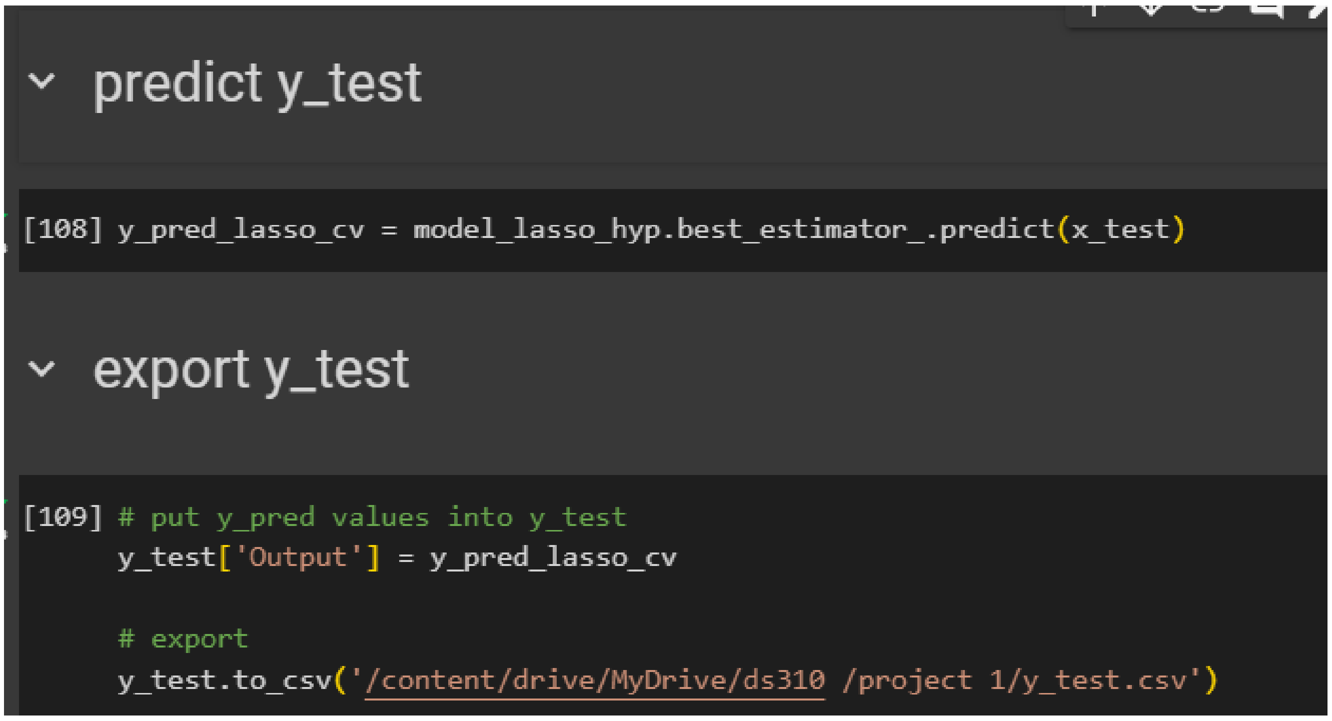
For cross-validation, I decided to use RepeatedKFold. As the RepeatedKFold method helps ensure that every observation from the original dataset has the chance of appearing in both the training and test set, the RepeatedKFold provides a more robust estimate of model performance.



```
# best alpha value
print('Best Alpha:', model_lasso_hyp.best_params_['alpha'])

# evaluate the model
print('MSE:', mean_squared_error(Y_val['Output'], y_pred_lasso_cv_train))
```

```
Best Alpha: 0.15
MSE: 2668.9081807275193
```

After performing hyperparameter tuning in the model, I can observe that the model performs better than the model without hyperparameter tuning.



```
[108] y_pred_lasso_cv = model_lasso_hyp.best_estimator_.predict(x_test)
```

```
[109] # put y_pred values into y_test
      y_test['Output'] = y_pred_lasso_cv

      # export
      y_test.to_csv('/content/drive/MyDrive/ds310 /project 1/y_test.csv')
```

After the model is completely trained with hyperparameter tuning, I can finally predict y_test using x_test data.

Lesson Learnt

Throughout this project, I learned the important role of the validation sets for model evaluation when I don’t have actual test values, highlighting their importance in hyperparameter tuned model for better prediction. The use of LassoLars model, combining Lasso regularization with Least Angle Regression (LARS), was a great choice for managing the high-dimensional medical data. It was efficient when handling feature selection and reducing overfitting risks. The initial model with MSE of 2858, showed the necessity of hyperparameter tuning. Through RepeatedKFold cross validation and adjustments of alpha parameters, I significantly enhanced the model performance with the reduced MSE of 2668. This process shows how critical the hyperparameter selection is for the accuracy and efficiency of the model.

This project was a great learning experience, taught the significance of the model development process and the strategic choice of algorithms for specific data types. Through hyperparameter tuning and using advanced cross validation techniques, I enhanced the model performance, showing the importance of these methods on the results of the prediction model. This experience not only deepened my understanding of regression analysis in high-dimensional data, but also highlighted the importance of adaptability and precision in the machine learning field.

Code Appendix

https://drive.google.com/file/d/1r08BszNwGAH5j-x9nES4c1AUT6doMgcJ/view?usp=sharing