In this work I'm going to use "**Diabetes**" dataset directly from SciKit Learn.

In this dataset, we have **10 features:**

age, sex, body mass index, average blood pressure, and six blood serum measurements (s1, s2, s3, s4, s5 and s6) + Target Column (diabetes_measure)

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	diabetes_measure
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641	135.0

were obtained from each of **442** diabetes patients, along with response of interest, a quantitative measure of disease progression (y) one year after baseline.

```
[442 rows x 11 columns]>
```

This dataset represents a classic regression problem, where the challenge is to model response y based on the ten features. This model can then be used for two purposes:

- one, to identify the important features (out of the ten mentioned above) that contribute to disease progression **(Interpretation)**
- and two, to predict the response for future patients based on the features **(Prediction)**

I will focus on Interpretation of the model.

Each of these ten feature variables has been mean-centered and scaled by the standard deviation times n_samples (i.e. the sum of squares of each column totals 1) **(Features Engineering)**

Data Preprocessing

Since the "diabetes" object belongs to the class Bunch, i.e. it is a collection of various objects bunched together in a dictionary-like format and includes the feature matrix "data" and the target vector "target". I've created a pandas DF containing all the ten features and the response variable (diab_measure) using the following commands:

```
df=pd.DataFrame(diabetes.data)
df.columns= diabetes.feature_names
# Creating a new column containing response variable 'y' (a quantitative measure of disease progression one year after baseline)
df['diabetes_measure']=diabetes.target
df.head()
```

Now that all the features and the target variable are now in a single dataframe object. I'm going to separate the feature matrix(X) and the target variable(y) using the dataframe just created.

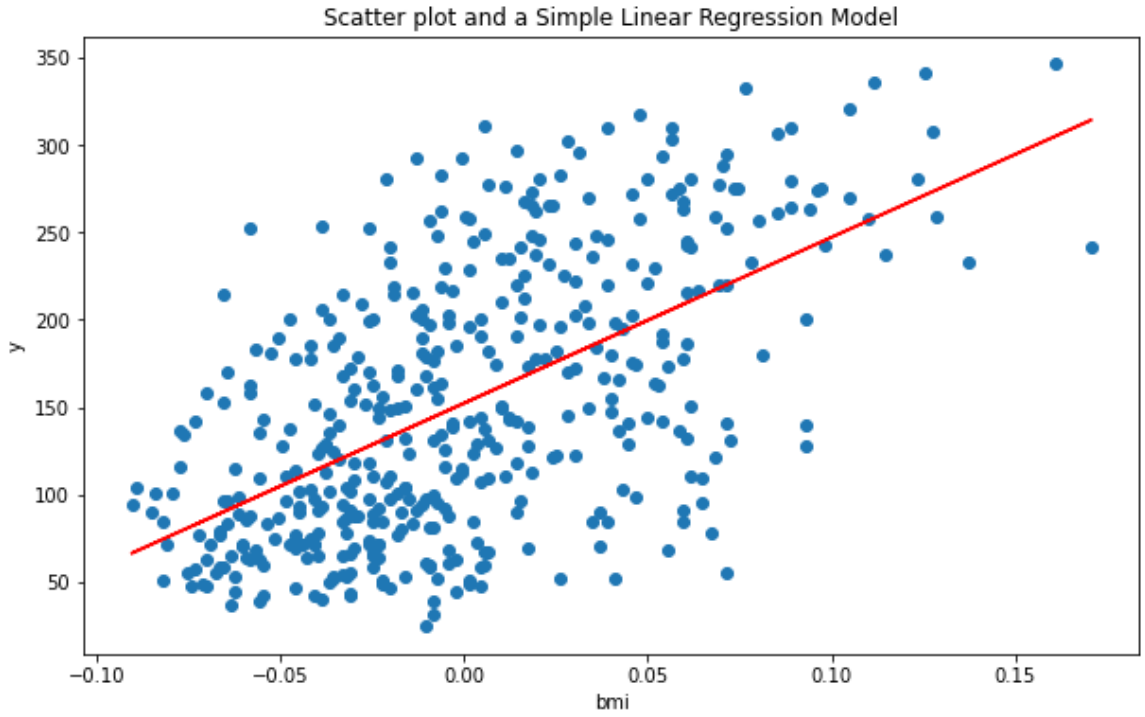
SIMPLE LINEAR REGRESSION MODEL

I can now move on to build and compare different regression models. But in order to do that, I first need to choose a suitable methodology to evaluate and compare these models: **R2_Score**.

I want to start with a **Simple Linear Regression Model** and then adding Features and Regularization Regressions to see if I can improve the score as I add complexity to the model.

Since we often hear that a healthy BMI (Body Mass Index) ratio is conducive to a lower chance of developing a diabetic condition, I will try to quantify this relation using, as I said a SLRM

This the plot **(BMI**) Vs Target Variable (**diabetes_measure**)



In the above plot, the blue dots are the actual data points (x,y). Visually, there seems to be a **positive linear relationship between BMI(Body mass index) and the diabetes measure(y),** that my model (red line) is trying to capture.

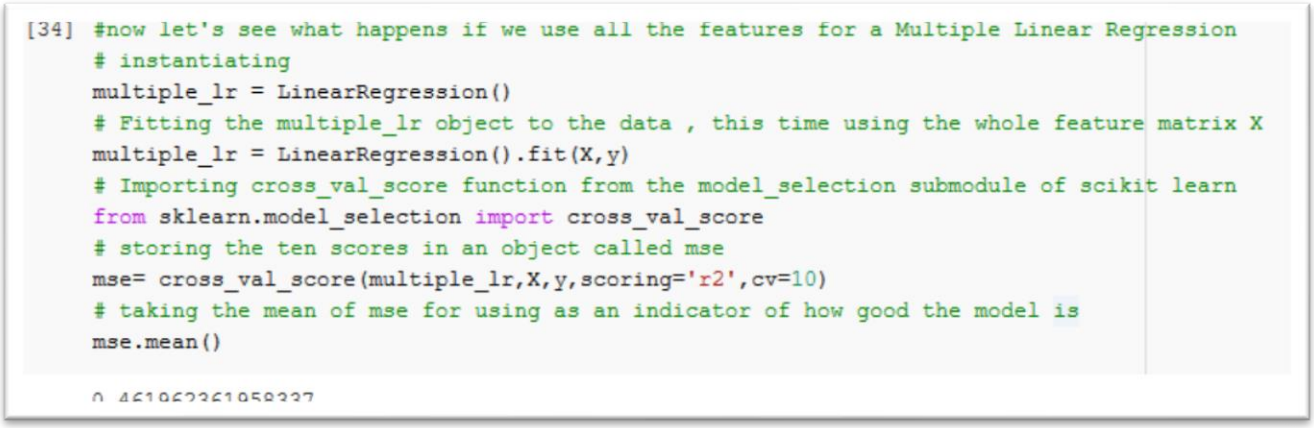
I want to perform a K-fold cross-validation to estimate how good the model is. Setting a value of cv to 10, means 10-fold cross-validation will be performed on the given data using the given estimator and corresponding ten 'R2' scores will be generated.

I will simply use the mean of all these ten scores (mse) as an indicator of how good the model is.

And I got a R2_score of **0.302446485542614. A very Low score**

Let us check if adding **more features** improves model considerably producing an overall better average score. To do this I will be seeking to build a linear regression model with multiple features, i.e. MLR.

Multiple Linear Regression



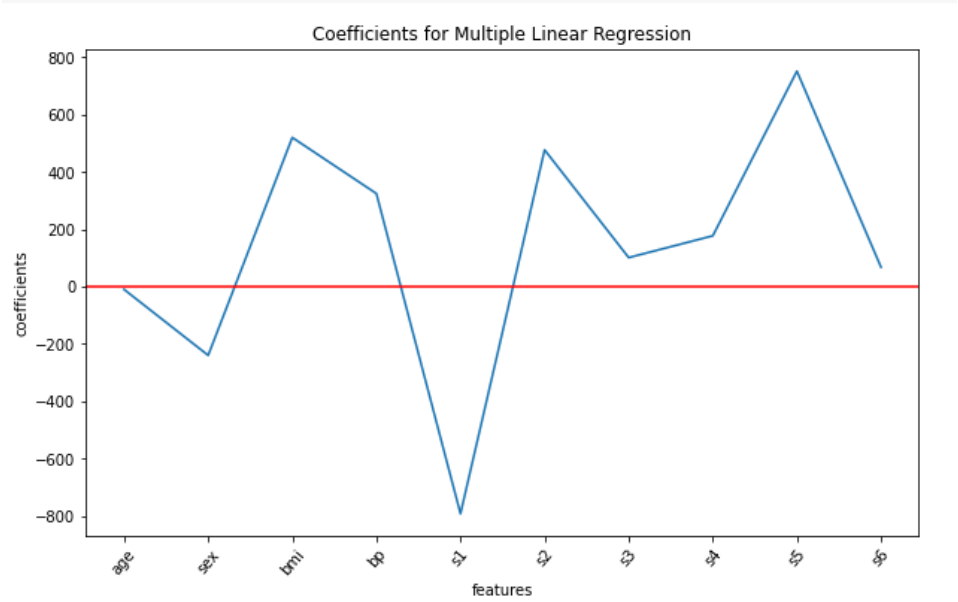
```
[34] #now let's see what happens if we use all the features for a Multiple Linear Regression
     # instantiating
     multiple_lr = LinearRegression()
     # Fitting the multiple_lr object to the data , this time using the whole feature matrix X
     multiple_lr = LinearRegression().fit(X,y)
     # Importing cross_val_score function from the model_selection submodule of scikit learn
     from sklearn.model_selection import cross_val_score
     # storing the ten scores in an object called mse
     mse= cross_val_score(multiple_lr,X,y,scoring='r2',cv=10)
     # taking the mean of mse for using as an indicator of how good the model is
     mse.mean()

     0.461962361958337
```

We see that adding the nine remaining features to the model along with bmi, **increases** the average 'R2_score' from **0.302446485542614** to **0.461962361958337** which is a considerable improvement.

Let us take a look at all the **ten coefficient values** for the 'multiple_lr' model:

```
[-10.012197817470847,
 -239.81908936565472,
 519.8397867901343,
 324.3904276893763,
 -792.1841616283061,
 476.74583782366255,
 101.04457032134488,
 177.0641762322512,
 751.2793210873945,
 67.62538639104386]
```

Note that **none** of the estimated coefficient values are **0**. Plotting the above values along with the corresponding feature names on the x-axis I got this:



We see that in this model, the **features bmi, s1, s2 and s5** are having a considerable impact on the progression of diabetes, as all of them have **high estimated** coefficient values.

It's time to **Regularize**, but before proceeding I want to check for **correlations** among all the feature variables,

```
[39] X.corr().style.background_gradient(cmap='coolwarm')
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
age	1.000000	0.173737	0.185085	0.335427	0.260061	0.219243	-0.075181	0.203841	0.270777	0.301731
sex	0.173737	1.000000	0.088161	0.241013	0.035277	0.142637	-0.379090	0.332115	0.149918	0.208133
bmi	0.185085	0.088161	1.000000	0.395415	0.249777	0.261170	-0.366811	0.413807	0.446159	0.388680
bp	0.335427	0.241013	0.395415	1.000000	0.242470	0.185558	-0.178761	0.257653	0.393478	0.390429
s1	0.260061	0.035277	0.249777	0.242470	1.000000	0.896663	0.051519	0.542207	0.515501	0.325717
s2	0.219243	0.142637	0.261170	0.185558	0.896663	1.000000	-0.196455	0.659817	0.318353	0.290600
s3	-0.075181	-0.379090	-0.366811	-0.178761	0.051519	-0.196455	1.000000	-0.738493	-0.398577	-0.273697
s4	0.203841	0.332115	0.413807	0.257653	0.542207	0.659817	-0.738493	1.000000	0.617857	0.417212
s5	0.270777	0.149918	0.446159	0.393478	0.515501	0.318353	-0.398577	0.617857	1.000000	0.464670
s6	0.301731	0.208133	0.388680	0.390429	0.325717	0.290600	-0.273697	0.417212	0.464670	1.000000

The above matrix shows **correlations** among the features such that darker shades of red implying high positive correlation and darker shades of blue implying high negative correlations.

In the **multiple linear model** I built in the previous section, both features **s1** and **s2** came out as important features. However, we can see that they have a very high positive correlation of about 0.896. This is clearly inducing **multicollinearity** into the model, I need to reduce it.

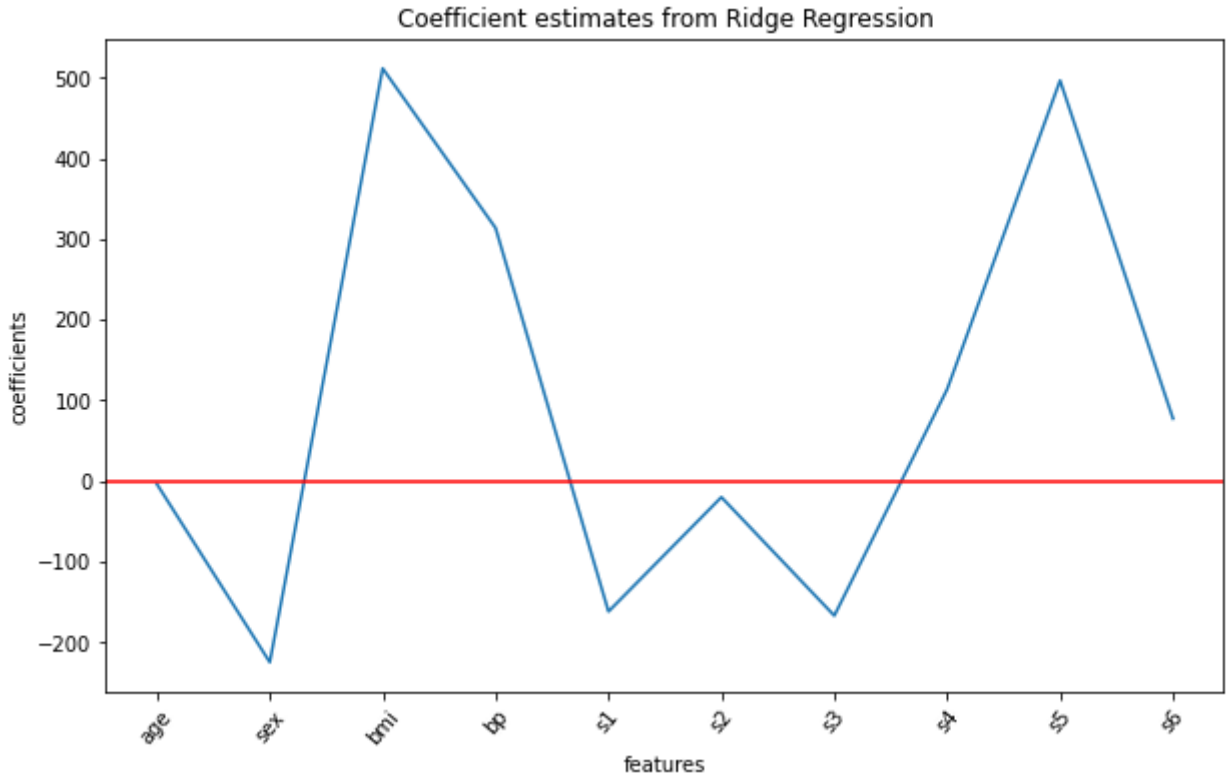
I will try all the 3 Regression we studied in the course and compare the results.

Implementing Ridge Regularization

```
# importing the Ridge class from linear_model submodule of scikit learn
from sklearn.linear_model import Ridge
# importing the GridSearchCV class from model_selection submodule of scikit learn
from sklearn.model_selection import GridSearchCV
# creating a dictionary containing potential values of alpha
alpha_values = {'alpha':[0.001, 0.01,0.02,0.03,0.04, 0.05, 0.06, 0.08, 1, 2, 3, 5, 8,
# Passing in a Ridge estimator, potential alpha values, scoring method and cross valid
ridge= GridSearchCV(Ridge(), alpha_values, scoring='r2', cv=10 )
# Fitting the model to the data and extracting best value of alpha
print('The best value of alpha is:',ridge.fit(X,y).best_params_)
# Printing the average neg_mean_squared_error of a 10-fold cross validation
print('The best score for the best Ridge estimator is:',ridge.fit(X,y).best_score_)
```

I got the best value of alpha is**: {'alpha': 0.04}**
The best score for the best Ridge estimator is: **0.4635352456321318** (better than MLR model!)

Let's see what's happened to our Coeffs after RR with **{'alpha': 0.04}**

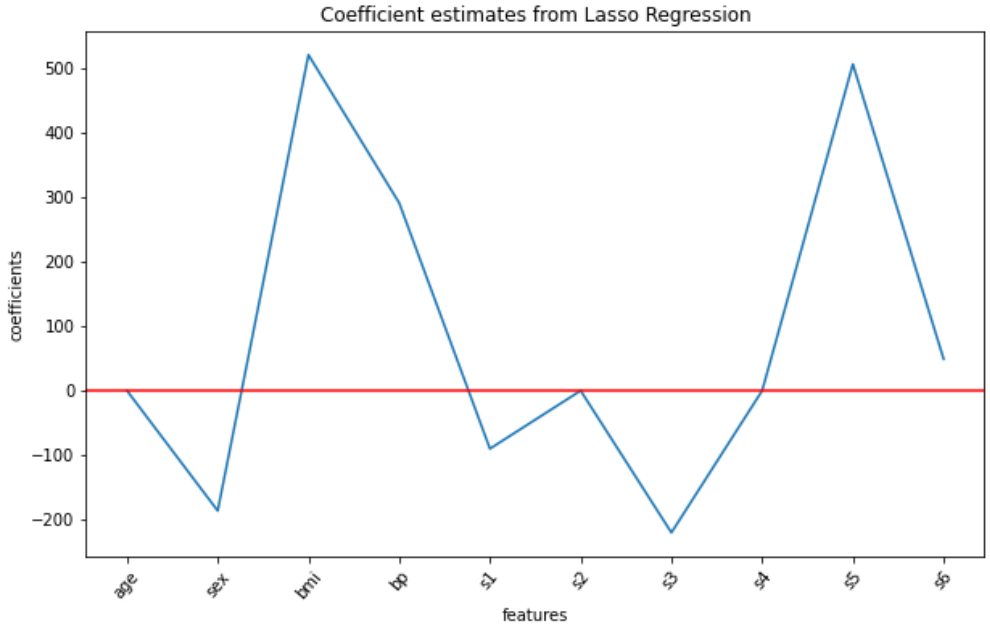


Previously, features **s1** and **s2** came out as an important feature in the multiple linear regression, however, their coefficient values are significantly reduced after ridge regularization, while the features **'bmi'** and **s5** still remain important.

Implementing Lasso Regularization

The best value of alpha is**: {'alpha': 0.06}**
The best score for the best Lasso estimator is**: 0.4652259038051521 (**(better than RR model!)

Let's see what's happened to our Coeffs after LR with **{'alpha': 0.06}**



More details about coeff :

array([-0. , -186.30924508, 520.89411638, 291.19604139,
-90.06855506, -0. , -220.20726443, 0. ,
506.42221212, 49.07461404])

Lasso regularization **has** completely eliminated features **'age', s2 and s4 from the model** (as their estimated coefficients are 0) and gives us a simpler model with less variables with overall best score!!

And what happens now if I use also the **ElasticNet Regularization**? Let's see…..

```
[45] # from sklearn.model_selection import GridSearchCV # If not already imported
     from sklearn.linear_model import ElasticNet
     alpha_values = {'alpha':[0.00005,0.0005,0.001, 0.01, 0.05, 0.06, 0.08, 1, 2, 3, 5, 8, 10, 20, 50, 100],
      'l1_ratio':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,1]}
     elastic= GridSearchCV(ElasticNet(), alpha_values, scoring='r2', cv=10 )
```

I got:

```
[46] elastic.fit(X,y).best_params_

     {'alpha': 0.06, 'l1_ratio': 1}

[47] elastic.fit(X,y).best_score_

     0.4652259038051521
```

In this case, the best **l1_ratio turns out to be 1**, which is the **same as a Lasso** regularization. Consequently, the best score also remains the same as obtained from Lasso regularization earlier.

CONCLUSION

Let us compare the coefficient estimates from:

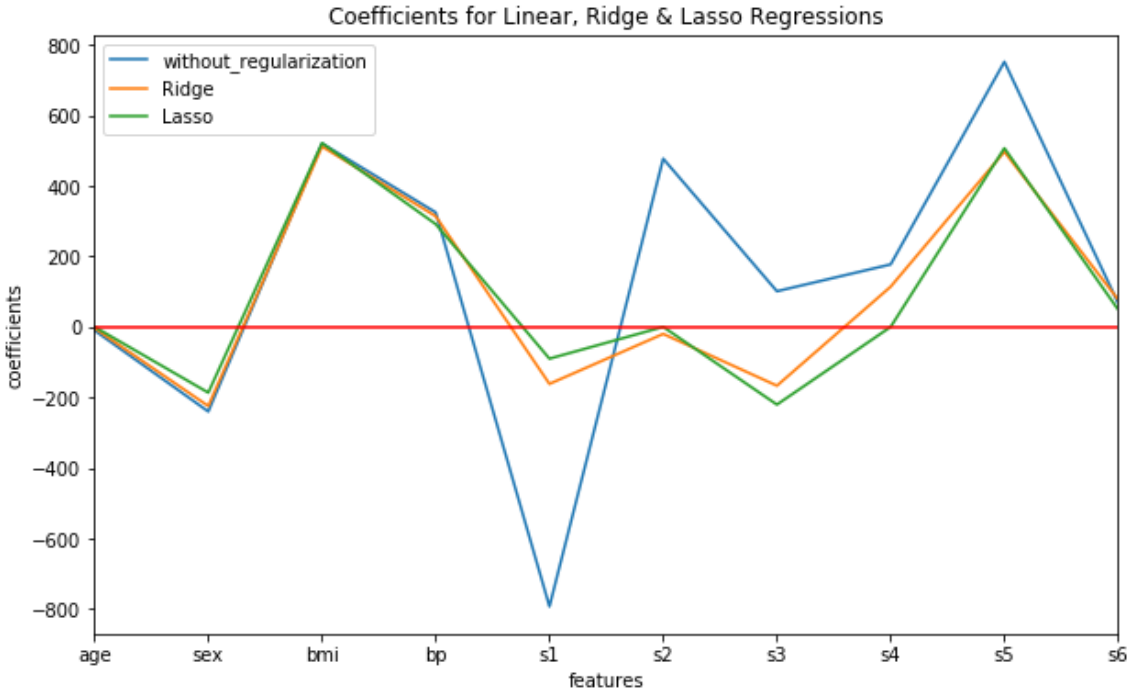
- MLR without any regularization
- Ridge regularization and
- Lasso regularization

Note that in this case, the resulting estimates from Elastic Net regularization were exactly the same as Lasso and hence I will not consider it separately

	without_regularization	Ridge	Lasso
age	-10.012198	-3.609650	-0.000000
sex	-239.819089	-224.329482	-186.309245
bmi	519.839787	511.203719	520.894116
bp	324.390428	313.552715	291.196041
s1	-792.184162	-161.533876	-90.068555
s2	476.745838	-19.892974	-0.000000
s3	101.044570	-166.679798	-220.207264
s4	177.064176	113.950246	0.000000
s5	751.279321	496.222270	506.422212
s6	67.625386	77.443906	49.074614

.

And finally, I plot coefficient values from all the three models on the same plot for visual comparison as follows:



Clearly the winning is Lasso regularization, it produces the best average score: **0.4652259038051521** out of all the three models.
It also gives a smaller/simpler model.

Summary Key Findings and Insights:

- From the SLRM Visually, there seems to be a positive linear relationship between BMI(Body mass index) and the diabetes measure(y), as waited.
- We see from Coeffiecients for MLRM that the features bmi, s1, s2 and s5 are having a considerable impact on the progression of diabetes, as all of them have high estimated coefficient values.
- Both features s1 and s2 came out as important features. However, we can see that they have a very high positive correlation of about 0.896. This is clearly inducing multicollinearity into the model.
- Lasso regularization completely eliminates features 'age', s2 and s4 from the model (as their estimated coefficients are 0) and gives us a simpler model with less variables with overall best score.
- I've tested 3 Linear Regression model and Lasso is the best one

Suggestions for next steps in analyzing this data

I'm sure we could get a better score from the model if we

- Could add more obervations
- Could add more Predictive Features
- Could try to add Polynomial Features