Question 1

What is the optimal value of alpha for ridge and lasso regression? What will be the changes in the model if you choose double the value of alpha for both ridge and lasso? What will be the most important predictor variables after the change is implemented?

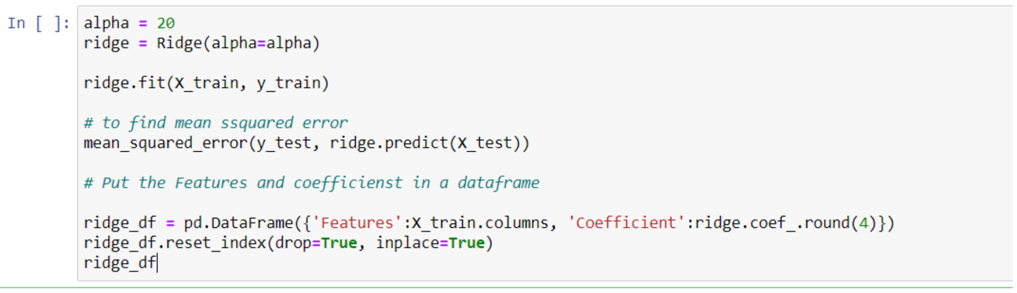
Answer

- The optimal lambda value in case of Ridge and Lasso is as below:

  ♣ Ridge - 10

  ♣ Lasso - 0.0004

- Changes in the model if you choose double the value of alpha for both ridge and lasso:
  - ➢ RSS increases as the optimal value increases because of model complexity.
  - ➢ Coefficients are getting shrinked as optimal value increases.
- If we use optimal value as 20 For Ridge, most important predictor variables are shown in Fig 2



```
In [ ]: alpha = 20
        ridge = Ridge(alpha=alpha)

        ridge.fit(X_train, y_train)

        # to find mean ssquared error
        mean_squared_error(y_test, ridge.predict(X_test))

        # Put the Features and coefficienst in a dataframe

        ridge_df = pd.DataFrame({'Features':X_train.columns, 'Coefficient':ridge.coef_.round(4)})
        ridge_df.reset_index(drop=True, inplace=True)
        ridge_df
```

Fig 1

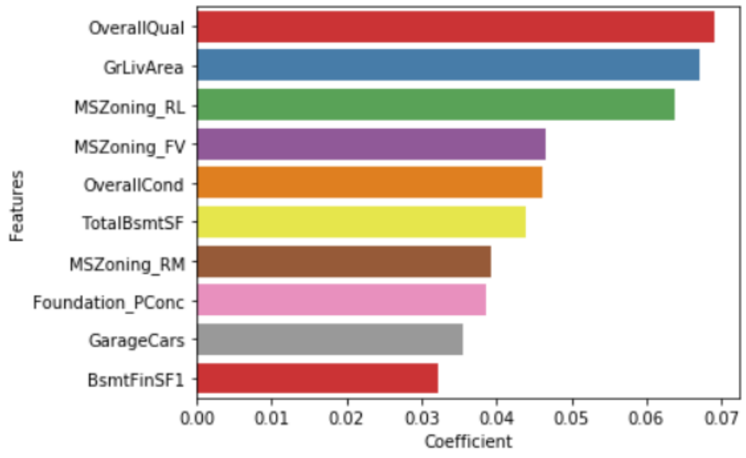


Fig 2

- If we use optimal value as 0.0008 For Lasso, most important predictor variables are shown in Fig4

```
alpha = 0.0008

lasso = Lasso(alpha=alpha)

lasso.fit(X_train, y_train)
lasso.coef_

# to find mean ssquared error
mean_squared_error(y_test, lasso.predict(X_test))

# Put the Features and coefficienst in a dataframe
lasso_df = pd.DataFrame({'Features':X_train.columns, 'Coefficient':lasso.coef_.round(4)})
lasso_df = lasso_df[lasso_df['Coefficient'] != 0.00]
lasso_df.reset_index(drop=True, inplace=True)
lasso_df
```

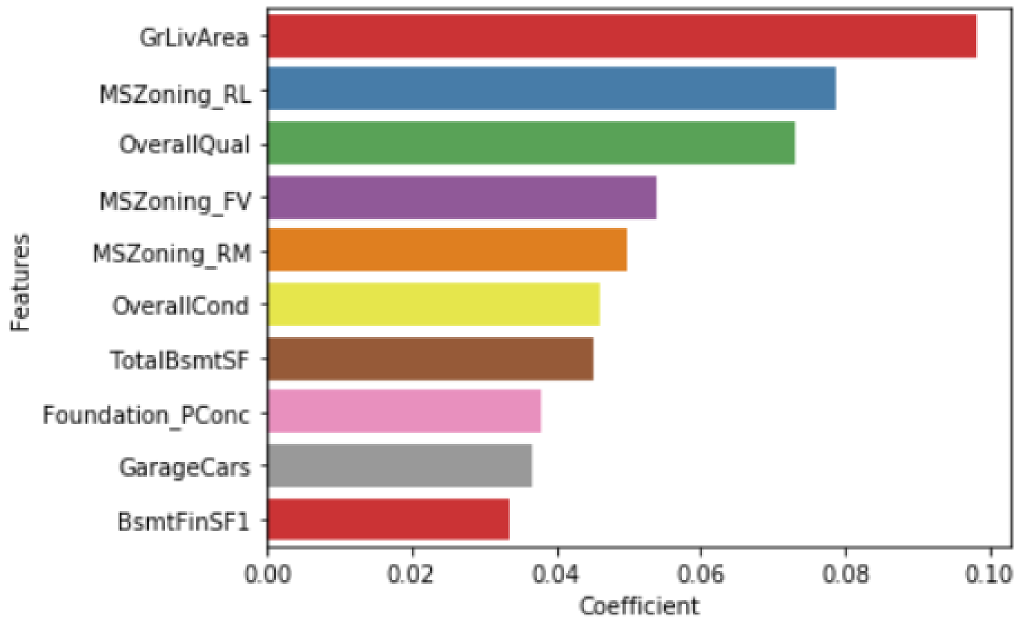


Fig 4

Question 2

You have determined the optimal value of lambda for ridge and lasso regression during the assignment. Now, which one will you choose to apply and why?

Answer

- The optimal lambda value in case of Ridge and Lasso is as below:
  - ♣ Ridge - 10

♣ Lasso - 0.0004

• The Mean Squared error in case of Ridge and Lasso are:

♣ Ridge - 0.013743

♣ Lasso - 0.013556

• The Mean Squared Error of Lasso is slightly lower than that of Ridge.

• Also, since Lasso helps in feature reduction (as the coefficient value of one of the feature became 0), Lasso has a better edge over Ridge. Therefore, the variables predicted by Lasso can be applied to choose significant variables for predicting the price of a house.

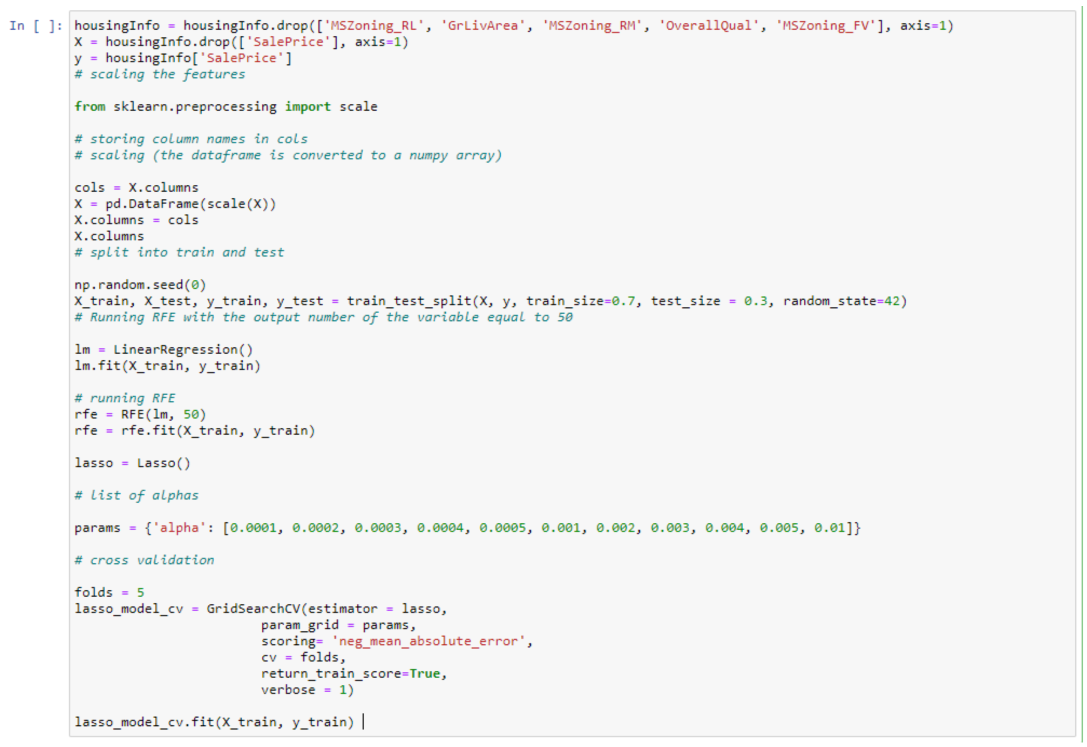
Question 3

After building the model, you realized that the five most important predictor variables in the lasso model are not available in the incoming data. You will now have to create another model excluding the five most important predictor variables. Which are the five most important predictor variables now?

Answer:

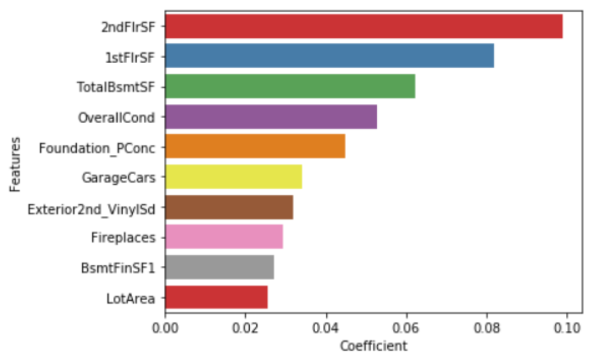
The five most important predictor variables in the lasso model are MSZoning_RL, MSZoning_RM, MSZoning_FV, GrLivArea, OverallQual. If we remove those variables and then build model,

Pseudo code:



```
In [ ]: housingInfo = housingInfo.drop(['MSZoning_RL', 'GrLivArea', 'MSZoning_RM', 'OverallQual', 'MSZoning_FV'], axis=1)
        X = housingInfo.drop(['SalePrice'], axis=1)
        y = housingInfo['SalePrice']
        # scaling the features

        from sklearn.preprocessing import scale

        # storing column names in cols
        # scaling (the dataframe is converted to a numpy array)

        cols = X.columns
        X = pd.DataFrame(scale(X))
        X.columns = cols
        X.columns
        # split into train and test

        np.random.seed(0)
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size = 0.3, random_state=42)
        # Running RFE with the output number of the variable equal to 50

        lm = LinearRegression()
        lm.fit(X_train, y_train)

        # running RFE
        rfe = RFE(lm, 50)
        rfe = rfe.fit(X_train, y_train)

        lasso = Lasso()

        # list of alphas

        params = {'alpha': [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.001, 0.002, 0.003, 0.004, 0.005, 0.01]}

        # cross validation

        folds = 5
        lasso_model_cv = GridSearchCV(estimator = lasso,
                                param_grid = params,
                                scoring= 'neg_mean_absolute_error',
                                cv = folds,
                                return_train_score=True,
                                verbose = 1)

        lasso_model_cv.fit(X_train, y_train)
```

After exclusion, most important predictor variables are:



The above graph displays the top 10 variables based on the Lasso Regression model that are significant in predicting the price of a house.

Question 4

How can you make sure that a model is robust and generalizable? What are the implications of the same for the accuracy of the model and why?

Answer:

Per, Occam's Razor— given two models that show similar 'performance' in the finite training or test data, we should pick the one that makes fewer on the test data due to following reasons:-

- Simpler models are usually more 'generic' and are more widely applicable
- Simpler models require fewer training samples for effective training than the more complex ones and hence are easier to train.
- Simpler models are more robust.
  - ➢ Complex models tend to change wildly with changes in the training data set
  - ➢ Simple models have low variance, high bias and complex models have low bias, high variance
- Simpler models make more errors in the training set. Complex models lead to overfitting — they work very well for the training samples, fail miserably when applied to other test samples.

Therefore to make the model more robust and generalizable, make the model simple but not simpler which will not be of any use.

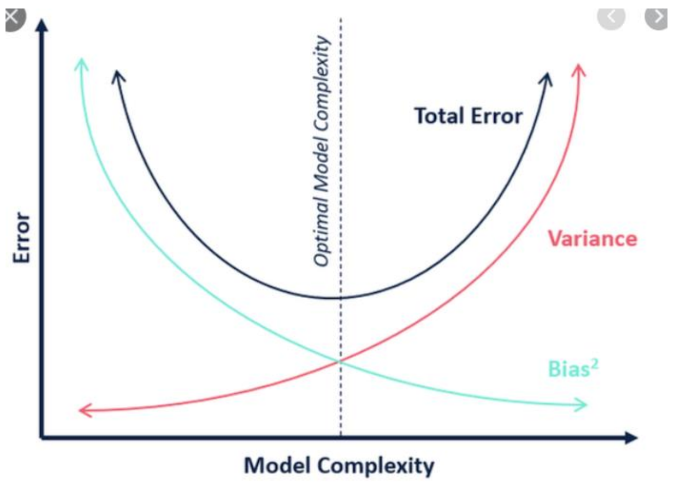
Regularization can be used to make the model simpler. Regularization helps to strike the delicate balance between keeping the model simple and not making it too naive to be of any use. For regression, regularization involves adding a regularization term to the cost that adds up the absolute values or the squares of the parameters of the model.

Also, making a model simple leads to Bias-Variance Trade-off:

- A complex model will need to change for every little change in the dataset and hence is very unstable and extremely sensitive to any changes in the training data.
- A simpler model that abstracts out some pattern followed by the data points given is unlikely to change wildly even if more points are added or removed.

Bias quantifies how accurate is the model likely to be on test data. A complex model can do an accurate job prediction provided there is enough training data. Models that are too naïve, for e.g., one that gives same answer to all test inputs and makes no discrimination whatsoever has a very large bias as its expected error across all test inputs are very high.

Variance refers to the degree of changes in the model itself with respect to changes in the training data. Thus accuracy of the model can be maintained by keeping the balance between Bias and Variance as it minimizes the total error as shown in the below graph.


Optimal Model Complexity
Total Error
Variance
Bias²
Error
Model Complexity