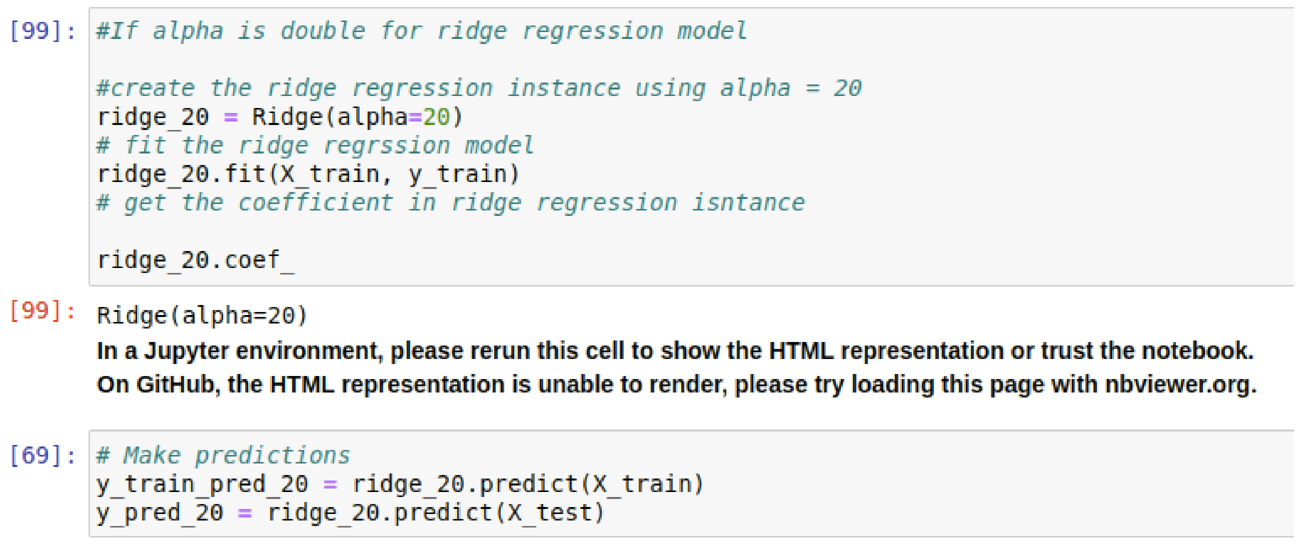
1) What is the optimal value of alpha for ridge and lasso regression? What will be the changes in the model if you choose double the value of alpha for both ridge and lasso? What will be the most important predictor variables after the change is implemented?

- optimal value of lambda for ridge : 10
- optimal value of lambda for lasso : 0.001

When alpha is double for ridge regression:
So here our alpha for ridge regression is 20

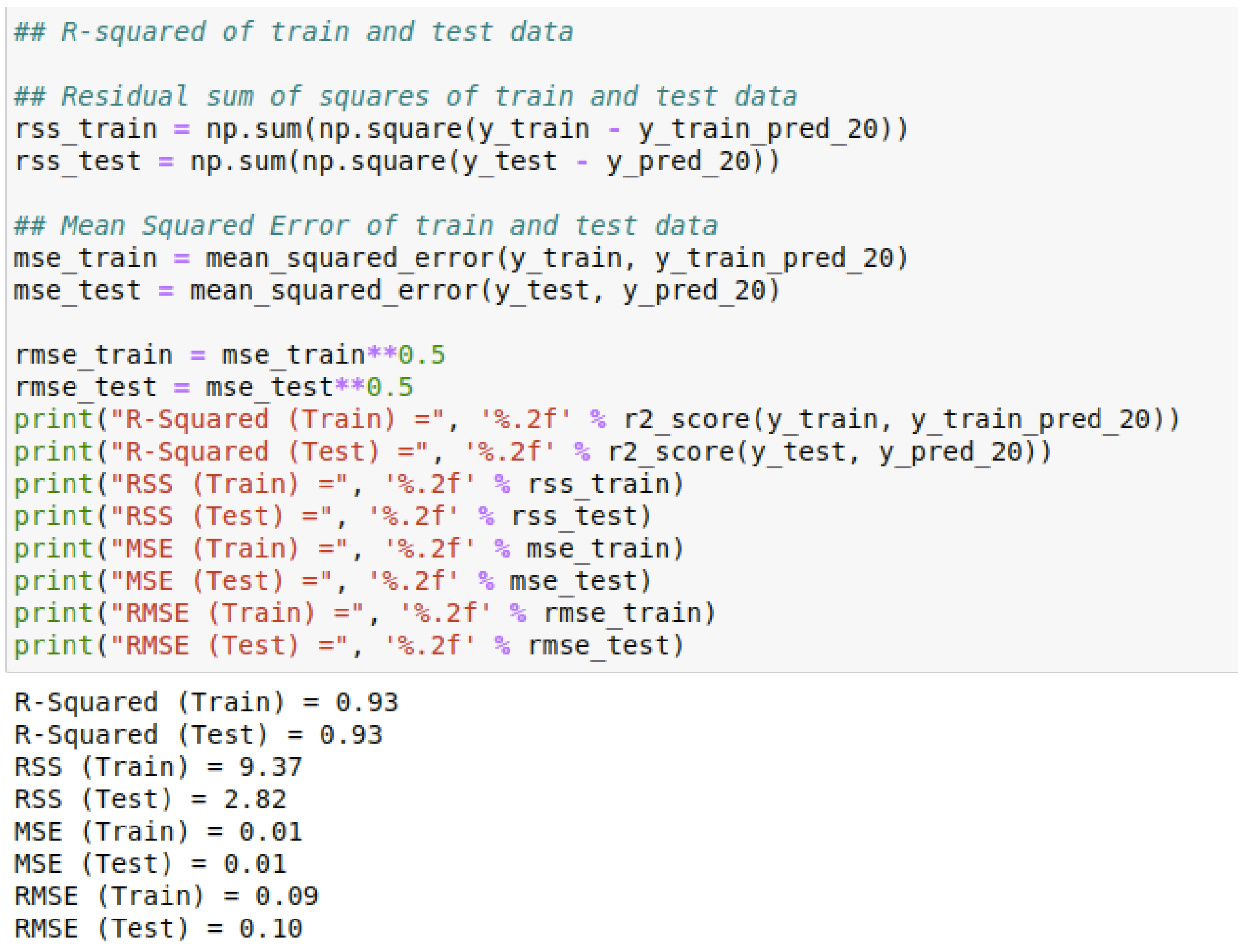


```
[99]: #If alpha is double for ridge regression model

      #create the ridge regression instance using alpha = 20
      ridge_20 = Ridge(alpha=20)
      # fit the ridge regrssion model
      ridge_20.fit(X_train, y_train)
      # get the coefficient in ridge regression isntance

      ridge_20.coef_
```

```
[99]: Ridge(alpha=20)
      In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
      On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

```
[69]: # Make predictions
      y_train_pred_20 = ridge_20.predict(X_train)
      y_pred_20 = ridge_20.predict(X_test)
```

Now we will calculate the R-Squared, RSS, MSE and RMSE for train and test model

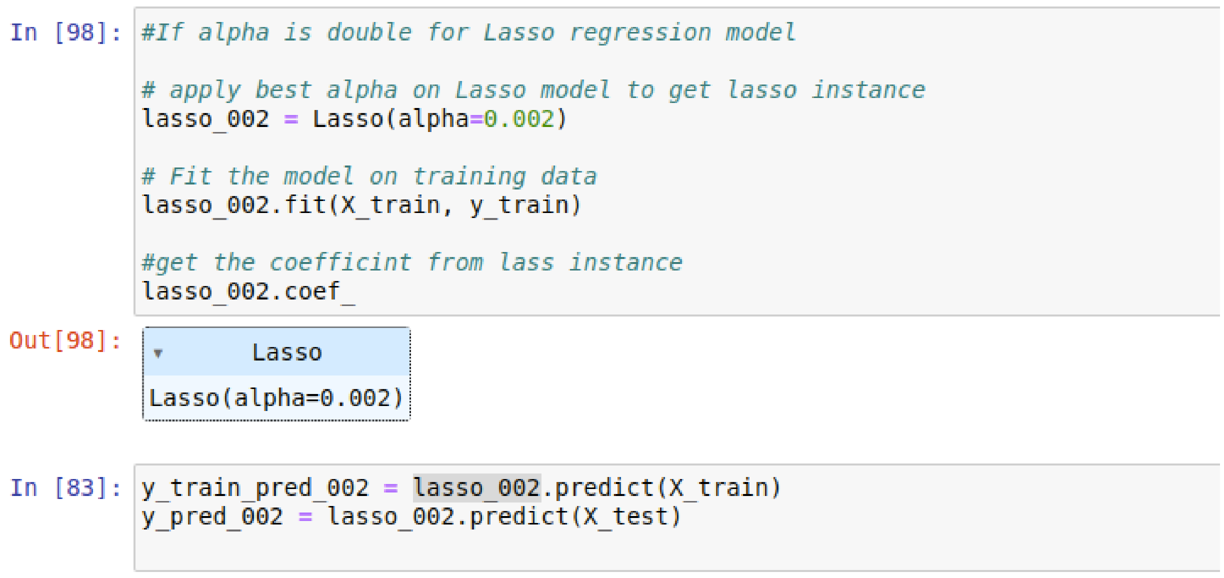


```
## R-squared of train and test data

## Residual sum of squares of train and test data
rss_train = np.sum(np.square(y_train - y_train_pred_20))
rss_test = np.sum(np.square(y_test - y_pred_20))

## Mean Squared Error of train and test data
mse_train = mean_squared_error(y_train, y_train_pred_20)
mse_test = mean_squared_error(y_test, y_pred_20)

rmse_train = mse_train**0.5
rmse_test = mse_test**0.5
print("R-Squared (Train) =", '%.2f' % r2_score(y_train, y_train_pred_20))
print("R-Squared (Test) =", '%.2f' % r2_score(y_test, y_pred_20))
print("RSS (Train) =", '%.2f' % rss_train)
print("RSS (Test) =", '%.2f' % rss_test)
print("MSE (Train) =", '%.2f' % mse_train)
print("MSE (Test) =", '%.2f' % mse_test)
print("RMSE (Train) =", '%.2f' % rmse_train)
print("RMSE (Test) =", '%.2f' % rmse_test)
```

```
R-Squared (Train) = 0.93
R-Squared (Test) = 0.93
RSS (Train) = 9.37
RSS (Test) = 2.82
MSE (Train) = 0.01
MSE (Test) = 0.01
RMSE (Train) = 0.09
RMSE (Test) = 0.10
```

R-Squared value for (Train) = 0.93
R-Squared value (Test) = 0.93
RSS value (Train) = 9.37
RSS value (Test) = 2.82
MSE value (Train) = 0.01
MSE value (Test) = 0.01
RMSE value(Train) = 0.09
RMSE value (Test) = 0.10

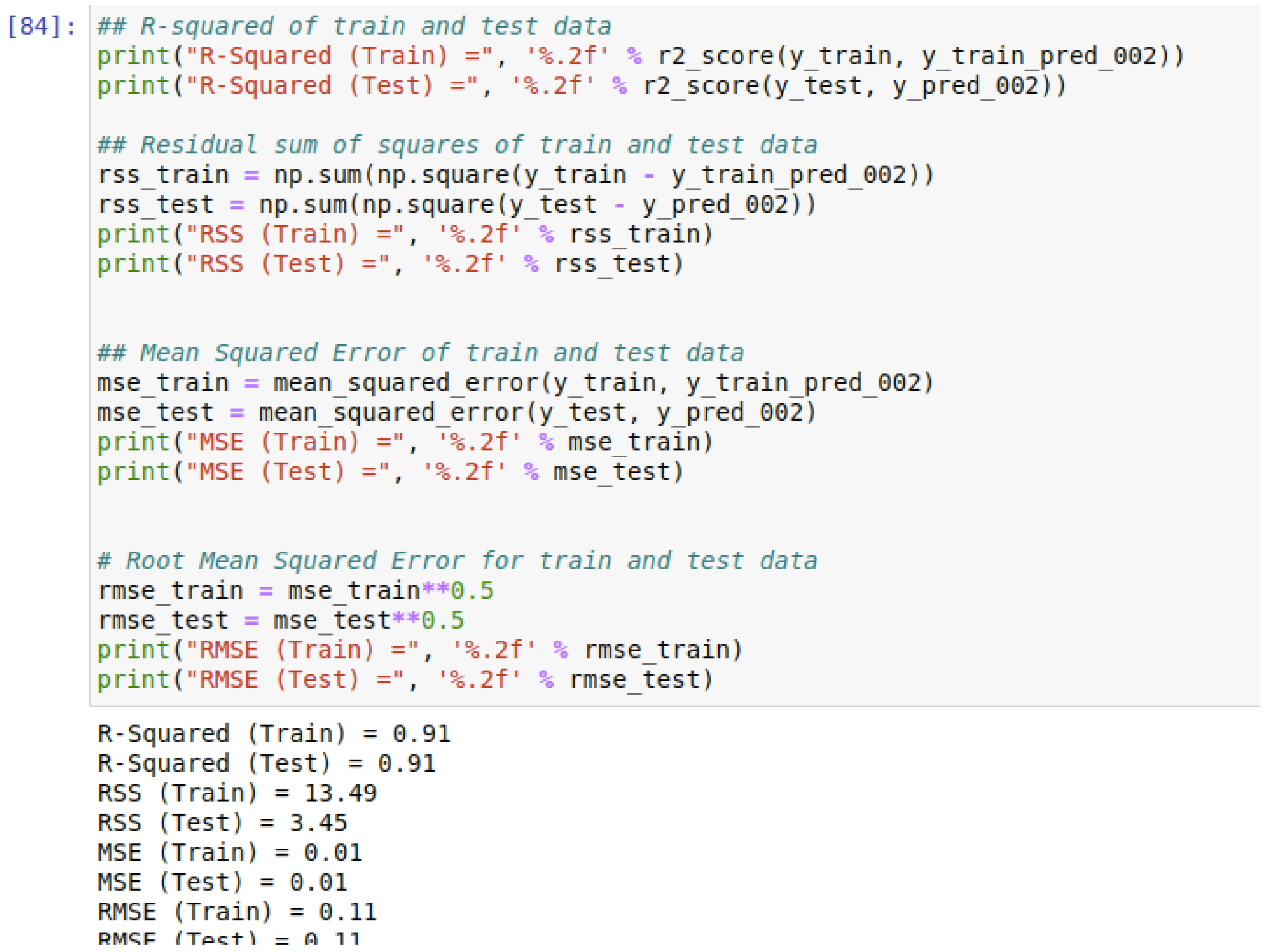
When alpha is double for laso regression:
So here our alpha for lasso regression is 0.002



```
In [98]: #If alpha is double for Lasso regression model

         # apply best alpha on Lasso model to get lasso instance
         lasso_002 = Lasso(alpha=0.002)

         # Fit the model on training data
         lasso_002.fit(X_train, y_train)

         #get the coefficint from lass instance
         lasso_002.coef_
Out[98]:   ▾     Lasso
         Lasso(alpha=0.002)
```

```
In [83]: y_train_pred_002 = lasso_002.predict(X_train)
         y_pred_002 = lasso_002.predict(X_test)
```

Now we will calculate the R-Squared, RSS, MSE and RMSE for train and test model



```
[84]: ## R-squared of train and test data
      print("R-Squared (Train) =", '%.2f' % r2_score(y_train, y_train_pred_002))
      print("R-Squared (Test) =", '%.2f' % r2_score(y_test, y_pred_002))

      ## Residual sum of squares of train and test data
      rss_train = np.sum(np.square(y_train - y_train_pred_002))
      rss_test = np.sum(np.square(y_test - y_pred_002))
      print("RSS (Train) =", '%.2f' % rss_train)
      print("RSS (Test) =", '%.2f' % rss_test)


      ## Mean Squared Error of train and test data
      mse_train = mean_squared_error(y_train, y_train_pred_002)
      mse_test = mean_squared_error(y_test, y_pred_002)
      print("MSE (Train) =", '%.2f' % mse_train)
      print("MSE (Test) =", '%.2f' % mse_test)


      # Root Mean Squared Error for train and test data
      rmse_train = mse_train**0.5
      rmse_test = mse_test**0.5
      print("RMSE (Train) =", '%.2f' % rmse_train)
      print("RMSE (Test) =", '%.2f' % rmse_test)
```

```
R-Squared (Train) = 0.91
R-Squared (Test) = 0.91
RSS (Train) = 13.49
RSS (Test) = 3.45
MSE (Train) = 0.01
MSE (Test) = 0.01
RMSE (Train) = 0.11
RMSE (Test) = 0.11
```

R-Squared value for (Train) = 0.91
R-Squared value for (Test) = 0.91
RSS value for (Train) = 13.49
RSS value for (Test) = 3.45
MSE value for (Train) = 0.01
MSE value for (Test) = 0.01
RMSE value for (Train) = 0.11
RMSE value for (Test) = 0.11

Changes in Ridge regression stats when alpha is doubled:

R-Squared for train model decreased from 0.94 to 0.93
R-Squared for test model did not changed its same which is 0.93

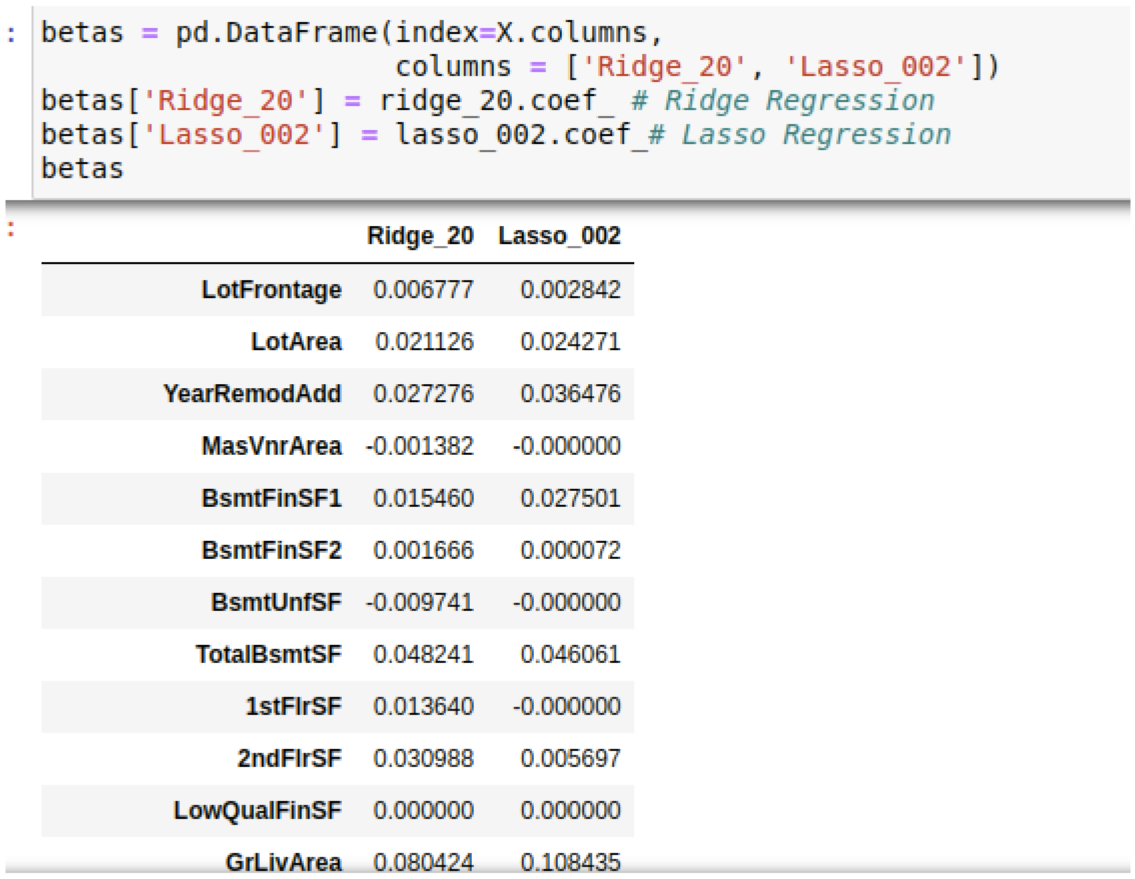
Changes in Lasso regression stats when alpha is doubled:

R-Squared for train model decreased from 0.92 to 0.91
R-Squared for test model decreased from 0.93 to 0.91

Now I have calculated top 10 features when alpha is double for Ridge and Lasso regression

Compared coefficiant for ridge and lasso regression

We have created betas table for comparison of ridge and lasso regressions columns coefficient

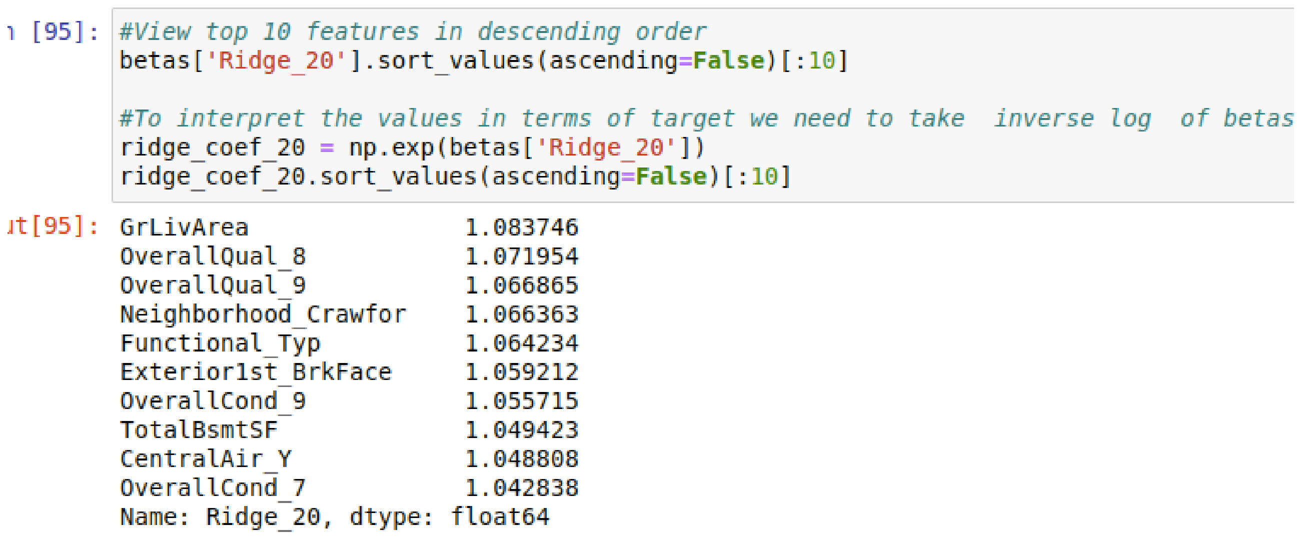


```
betas = pd.DataFrame(index=X.columns,
                     columns = ['Ridge_20', 'Lasso_002'])
betas['Ridge_20'] = ridge_20.coef_ # Ridge Regression
betas['Lasso_002'] = lasso_002.coef_# Lasso Regression
betas
```

	Ridge_20	Lasso_002
LotFrontage	0.006777	0.002842
LotArea	0.021126	0.024271
YearRemodAdd	0.027276	0.036476
MasVnrArea	-0.001382	-0.000000
BsmtFinSF1	0.015460	0.027501
BsmtFinSF2	0.001666	0.000072
BsmtUnfSF	-0.009741	-0.000000
TotalBsmtSF	0.048241	0.046061
1stFlrSF	0.013640	-0.000000
2ndFlrSF	0.030988	0.005697
LowQualFinSF	0.000000	0.000000
GrLivArea	0.080424	0.108435

aa

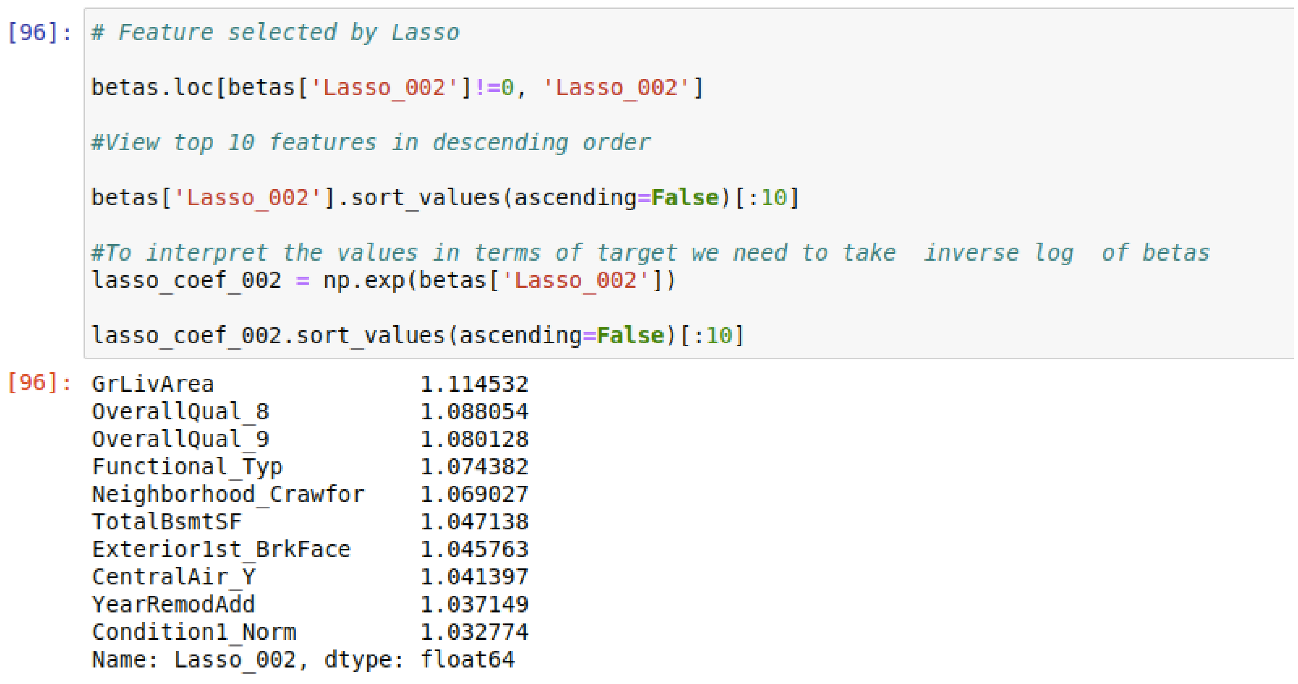
In below screenshot top 10 features of ridge regrssion are in decsening order then to interpret the coefficient value in terms of target we need to take inverse log of betas



```
ı [95]: #View top 10 features in descending order
        betas['Ridge_20'].sort_values(ascending=False)[:10]

        #To interpret the values in terms of target we need to take  inverse log  of betas
        ridge_coef_20 = np.exp(betas['Ridge_20'])
        ridge_coef_20.sort_values(ascending=False)[:10]
```

```
ıt[95]: GrLivArea               1.083746
        OverallQual_8           1.071954
        OverallQual_9           1.066865
        Neighborhood_Crawfor    1.066363
        Functional_Typ          1.064234
        Exterior1st_BrkFace     1.059212
        OverallCond_9           1.055715
        TotalBsmtSF             1.049423
        CentralAir_Y            1.048808
        OverallCond_7           1.042838
        Name: Ridge_20, dtype: float64
```

In below screenshot top 10 features of lasso regrssion are in decsening order then to interpret the coefficient value in terms of target we need to take inverse log of betas



```
[96]: # Feature selected by Lasso

      betas.loc[betas['Lasso_002']!=0, 'Lasso_002']

      #View top 10 features in descending order

      betas['Lasso_002'].sort_values(ascending=False)[:10]

      #To interpret the values in terms of target we need to take  inverse log  of betas
      lasso_coef_002 = np.exp(betas['Lasso_002'])

      lasso_coef_002.sort_values(ascending=False)[:10]
```

```
[96]: GrLivArea               1.114532
      OverallQual_8           1.088054
      OverallQual_9           1.080128
      Functional_Typ          1.074382
      Neighborhood_Crawfor    1.069027
      TotalBsmtSF             1.047138
      Exterior1st_BrkFace     1.045763
      CentralAir_Y            1.041397
      YearRemodAdd            1.037149
      Condition1_Norm         1.032774
      Name: Lasso_002, dtype: float64
```

If we double the alpha value then top features are :

GrLivArea, OverallQual_8, OverallQual_9, Neighborhood_Crawfor, Functional_Typ, Exterior1st_BrkFace
OverallCond_9, TotalBsmtSF, CentralAir_Y,OverallCond_7, YearRemodAdd, Condition1_Norm

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?

Ridge regression:
When you have lots of data and many predictors that could matter, and they might be related, choosing Ridge regression is a smart move. It's like finding a good balance (lambda around 10) to avoid the model from getting too excited and making predictions that don't make sense.

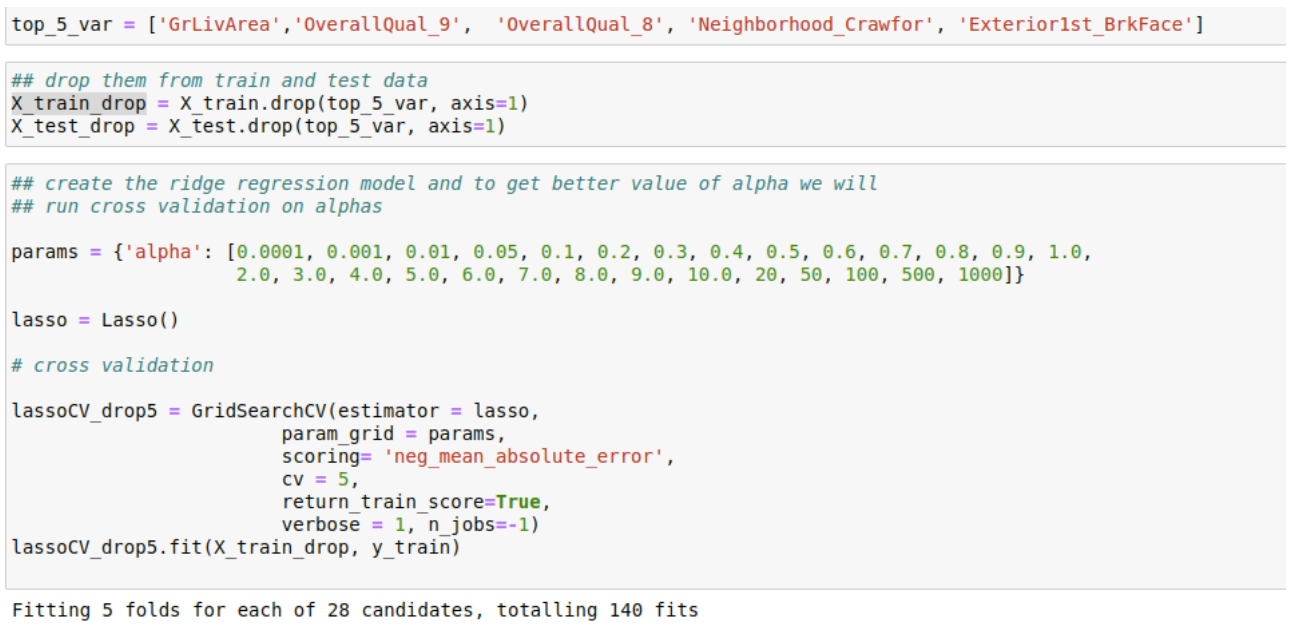
Lasso regression:
On the flip side, if you think only a few predictors really matter, and you want to keep things simple and clear, Lasso regression is the way to go. It helps to set less useful predictors to zero (lambda around 0.001), like tidying up and focusing only on the really important stuff.

To wrap it up, deciding between Ridge and Lasso depends on how many predictors are important and how you want your model to be. Lasso is like a strict cleaner, throwing out what's not needed, great when only a few things matter a lot. Ridge is more inclusive, keeping all the potential factors in mind, useful when many things are important in a similar way. The choice depends on what fits best with your dataset and what you want to achieve with your analysis.

3) After building the model, you realised 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?

Top 5 Lasso predictors are :
GrLivArea', 'OverallQual_9', 'OverallQual_8', 'Neighborhood_Crawfor', 'Exterior1st_BrkFace'

We are dropping above 5 variable and creating another model after dropping above 5 variables:

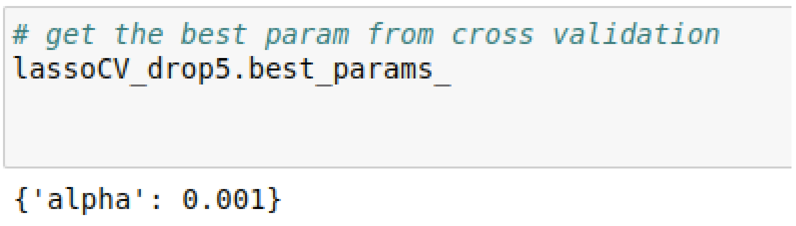


```
top_5_var = ['GrLivArea','OverallQual_9',  'OverallQual_8', 'Neighborhood_Crawfor', 'Exterior1st_BrkFace']
```

```
## drop them from train and test data
X_train_drop = X_train.drop(top_5_var, axis=1)
X_test_drop = X_test.drop(top_5_var, axis=1)
```

```
## create the ridge regression model and to get better value of alpha we will
## run cross validation on alphas

params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0,
                    2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100, 500, 1000]}

lasso = Lasso()

# cross validation

lassoCV_drop5 = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = 5,
                        return_train_score=True,
                        verbose = 1, n_jobs=-1)
lassoCV_drop5.fit(X_train_drop, y_train)
```

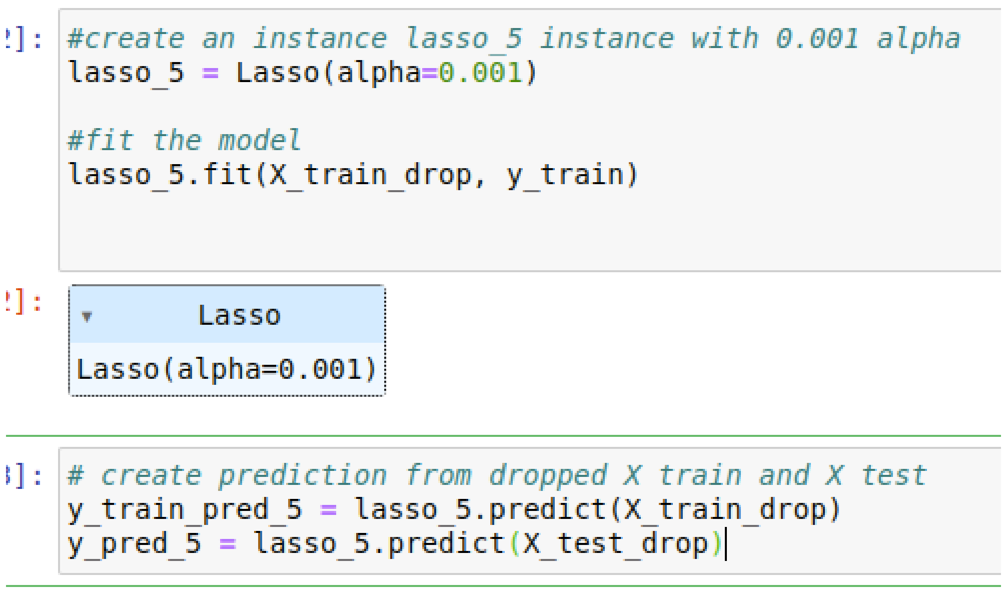
```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```



```
# get the best param from cross validation
lassoCV_drop5.best_params_
```

```
{'alpha': 0.001}
```

Now we get optimum value of lambda for lasso is 0.001

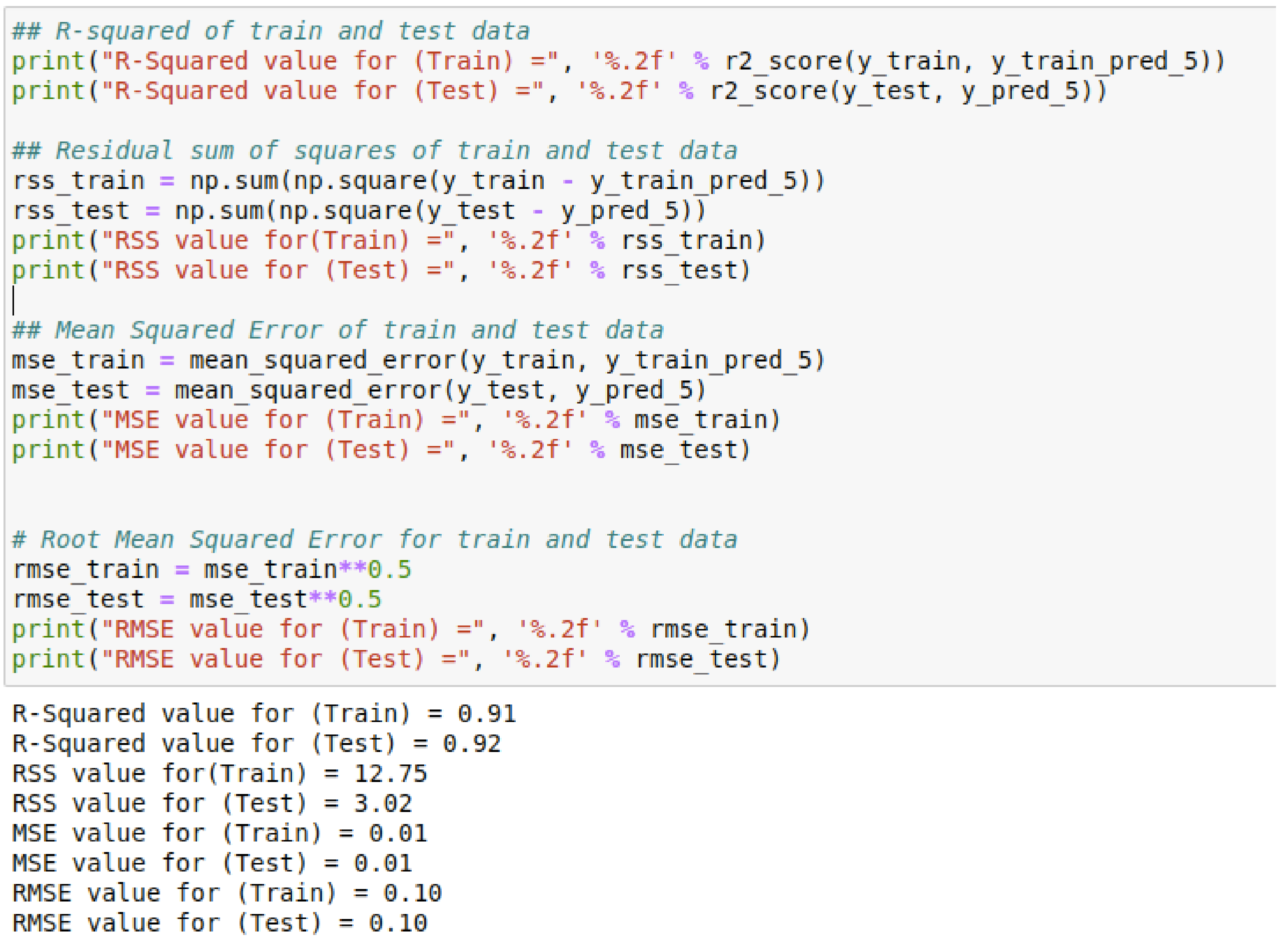


```
#create an instance lasso_5 instance with 0.001 alpha
lasso_5 = Lasso(alpha=0.001)

#fit the model
lasso_5.fit(X_train_drop, y_train)
```

```
Lasso
Lasso(alpha=0.001)
```

```
# create prediction from dropped X train and X test
y_train_pred_5 = lasso_5.predict(X_train_drop)
y_pred_5 = lasso_5.predict(X_test_drop)
```

Now we will calculate the R-Squared, RSS, MSE and RMSE for train and test model

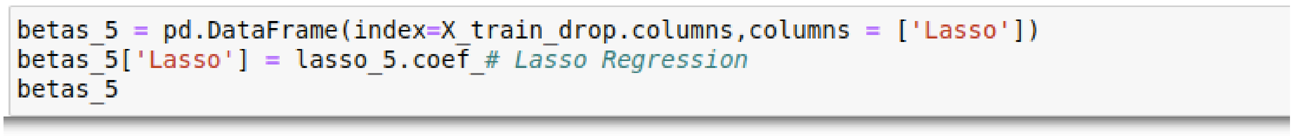


```
## R-squared of train and test data
print("R-Squared value for (Train) =", '%.2f' % r2_score(y_train, y_train_pred_5))
print("R-Squared value for (Test) =", '%.2f' % r2_score(y_test, y_pred_5))

## Residual sum of squares of train and test data
rss_train = np.sum(np.square(y_train - y_train_pred_5))
rss_test = np.sum(np.square(y_test - y_pred_5))
print("RSS value for(Train) =", '%.2f' % rss_train)
print("RSS value for (Test) =", '%.2f' % rss_test)

## Mean Squared Error of train and test data
mse_train = mean_squared_error(y_train, y_train_pred_5)
mse_test = mean_squared_error(y_test, y_pred_5)
print("MSE value for (Train) =", '%.2f' % mse_train)
print("MSE value for (Test) =", '%.2f' % mse_test)


# Root Mean Squared Error for train and test data
rmse_train = mse_train**0.5
rmse_test = mse_test**0.5
print("RMSE value for (Train) =", '%.2f' % rmse_train)
print("RMSE value for (Test) =", '%.2f' % rmse_test)
```

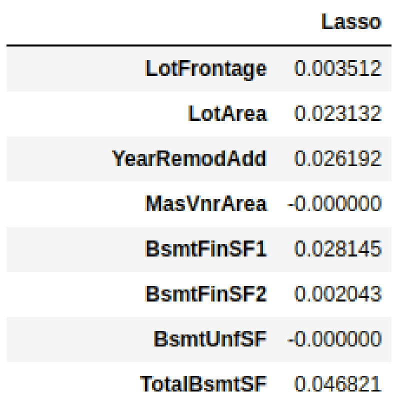
```
R-Squared value for (Train) = 0.91
R-Squared value for (Test) = 0.92
RSS value for(Train) = 12.75
RSS value for (Test) = 3.02
MSE value for (Train) = 0.01
MSE value for (Test) = 0.01
RMSE value for (Train) = 0.10
RMSE value for (Test) = 0.10
```

R-Squared value for (Train) = 0.91
R-Squared value for (Test) = 0.92
RSS value for(Train) = 12.75
RSS value for (Test) = 3.02
MSE value for (Train) = 0.01
MSE value for (Test) = 0.01
RMSE value for (Train) = 0.10
RMSE value for (Test) = 0.10

Now we will see coefficient values after applying regularization



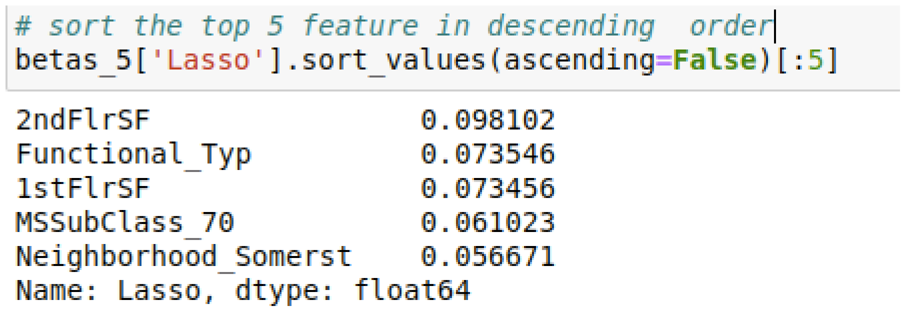
```
betas_5 = pd.DataFrame(index=X_train_drop.columns,columns = ['Lasso'])
betas_5['Lasso'] = lasso_5.coef_# Lasso Regression
betas_5
```



	Lasso
LotFrontage	0.003512
LotArea	0.023132
YearRemodAdd	0.026192
MasVnrArea	-0.000000
BsmtFinSF1	0.028145
BsmtFinSF2	0.002043
BsmtUnfSF	-0.000000
TotalBsmtSF	0.046821

1

After excluding top 5 feature now we are going to get top 5 feture for new Lasso model



```
# sort the top 5 feature in descending  order
betas_5['Lasso'].sort_values(ascending=False)[:5]
```

```
2ndFlrSF               0.098102
Functional_Typ         0.073546
1stFlrSF               0.073456
MSSubClass_70          0.061023
Neighborhood_Somerst   0.056671
Name: Lasso, dtype: float64
```

These are new top 5 predictors after dropping 5 top features

2ndFlrSF, Functional_Typ,   1stFlrSF,   MSSubClass_70, Neighborhood_Somerst

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

To ensure a model is robust and generalizable, we follow a few essential practices:

Use Sufficient Data:
Having enough data helps the model learn patterns effectively. More data allows the model to capture a broader range of scenarios, enhancing its generalization.

Split Data for Training and Testing:
We divide our dataset into two parts: one for training the model and another for testing its performance. This helps assess how well the model generalizes to unseen data.

Cross-Validation:
In addition to a simple train-test split, we use techniques like cross-validation. This involves splitting the data into multiple subsets, training the model on different combinations, and evaluating its performance. It ensures a more robust assessment of the model's generalization.

Feature Selection:
Choose relevant features that genuinely affect the outcome. Too many irrelevant features can hinder generalization. Techniques like Lasso and Ridge regression aid in feature selection by emphasizing important features and reducing the impact of less relevant ones.

Regularization:
Regularization methods like Ridge and Lasso help control the model's complexity. They add penalties to the model parameters, preventing overfitting and promoting a more generalized model.

Avoid Overfitting:
Overfitting occurs when the model learns the training data too well but struggles with new data. By using regularization and suitable model evaluation techniques, we can mitigate overfitting and create a more robust, generalizable model.

Optimize Hyperparameters:
Tuning hyperparameters, such as the alpha in Ridge and Lasso regression, ensures the model is well-adjusted. The right hyperparameters lead to a more accurate and generalizable model.

The implications of these practices on model accuracy are significant. A robust and generalizable model might not achieve the highest accuracy on the training data because it's designed to perform well on unseen data. This trade-off ensures the model doesn't get too specialized in the training data and can handle a broader range of real-world scenarios.

In summary, a balance between training the model well and preventing overfitting is key. A more generalized model might sacrifice a bit of accuracy on the training data, but it's likely to perform better and be more reliable when faced with new, unseen data.