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 value for alpha for ridge is 0.3 and the optimal value for lasso regression is 50</i>

Let's double the alpha value for ridge

In [79]:
```python
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 ]}

ridge = Ridge()

# cross validation
folds = 5
model_cv = GridSearchCV(estimator = ridge,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)
model_cv.fit(X_train, y_train)
print(model_cv.best_params_)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
{'alpha': 0.3}
```

In [80]:
```python
alpha = 0.6   #doubling alpha
ridge = Ridge(alpha=alpha)

ridge.fit(X_train, y_train)
print(ridge.coef_)
```

```
[  90294.28482529  142913.47874463   39494.19052757   61629.60281109
   25888.68415516   21225.25569959   71849.45117603   55136.34371791
   54284.98601548  -16272.41973434   53721.08590405  -10103.16701939
  -65444.30276826   57972.59151671  -29510.21876633   26920.77647077
   30360.03920854   20314.33533742  -24783.12727807       0.
 -197318.35151522  -16510.52724713   26723.29145461  -17370.78868175
   21257.01561289   17711.43807094   -1897.91220283    3447.9749362
  -12281.68167077  -10866.04606468  -34011.94762456       0.
       0.               0.           32164.30143714   -3959.50954667
     545.89947043    3413.61007624  -17831.9435029   -65253.98762327
   -2194.39655531   15751.0758664    -5094.46294719    5144.24097985
    -533.02687968    1302.25347797    8073.87915407    4763.35159138
  -19973.25564723   20323.94515146]
```

In [81]:
```python
# Lets calculate some metrics such as R2 score, RSS and RMSE
y_pred_train = ridge.predict(X_train)
y_pred_test = ridge.predict(X_test)

metric2 = []
r2_train_lr = r2_score(y_train, y_pred_train)
print(r2_train_lr)
metric2.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print(r2_test_lr)
metric2.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print(rss1_lr)
metric2.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print(rss2_lr)
metric2.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print(mse_train_lr)
metric2.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print(mse_test_lr)
metric2.append(mse_test_lr**0.5)
```

```
0.8704481316669731
-596627472.58226
396067297523.43884
7.917691503156823e+20
441546597.01609683
2.056543247573201e+18
```

Let's double the alpha value for Lasso

In [82]:
```python
lasso = Lasso()

# cross validation
model_cv = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

model_cv.fit(X_train, y_train)
```

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

Out[82]:
```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'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]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [83]:
```python
# Printing the best hyperparameter alpha
print(model_cv.best_params_)
```

```
{'alpha': 50}
```

In [84]:
```python
alpha =100 #doubling the alpha

lasso = Lasso(alpha=alpha)

lasso.fit(X_train, y_train)
```

Out[84]:
```
Lasso(alpha=100)
```

In [85]:
```python
lasso.coef_
```

Out[85]:
```
array([  61637.06626021,  154436.95132414,   34709.68994909,
         37935.87975256,    3571.0526077 ,      -0.        ,
        101824.29760127,   83958.01968029,   68380.89526796,
            -0.        ,       0.        ,      -0.        ,
        -58001.64145975,   57621.15676299,      -0.        ,
         24178.28298834,   26680.24285438,   17288.75086436,
         -5767.83260734,       0.        , -216619.21342456,
             0.        ,       0.        ,   -4749.96058276,
             0.        ,       0.        ,       0.        ,
             0.        ,       0.        ,       0.        ,
            -0.        ,       0.        ,       0.        ,
             0.        ,       0.        ,   -1996.9082266 ,
             0.        ,    1754.20201055,      -0.        ,
            -0.        ,   -4949.14561511,    3217.62126578,
            -0.        ,       0.        ,      -0.        ,
             0.        ,      -0.        ,    1793.90981351,
            -0.        ,       0.        ])
```

In [86]:
```python
# Lets calculate some metrics such as R2 score, RSS and RMSE

y_pred_train = lasso.predict(X_train)
y_pred_test = lasso.predict(X_test)

metric3 = []
r2_train_lr = r2_score(y_train, y_pred_train)
print(r2_train_lr)
metric3.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print(r2_test_lr)
metric3.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print(rss1_lr)
metric3.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print(rss2_lr)
metric3.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print(mse_train_lr)
metric3.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print(mse_test_lr)
metric3.append(mse_test_lr**0.5)
```

```
0.8600601168644859
-313470738.64233243
427825642674.2644
4.159990482590774e+20
476951664.07387334
1.0805170084651361e+18
```

In [87]:
```python
# Creating a table which contain all the metrics

lr_table = {'Metric': ['R2 Score (Train)','R2 Score (Test)','RSS (Train)','RSS (Test)',
                       'MSE (Train)','MSE (Test)'],
        'Linear Regression': metric
        }

lr_metric = pd.DataFrame(lr_table ,columns = ['Metric', 'Linear Regression'] )

rg_metric = pd.Series(metric2, name = 'Ridge Regression')
ls_metric = pd.Series(metric3, name = 'Lasso Regression')

final_metric = pd.concat([lr_metric, rg_metric, ls_metric], axis = 1)

final_metric
```

Out[87]:

	Metric	Linear Regression	Ridge Regression	Lasso Regression
0	R2 Score (Train)	8.796891e-01	8.704481e-01	8.600601e-01
1	R2 Score (Test)	-1.264357e+09	-5.966275e+08	-3.134707e+08
2	RSS (Train)	3.678157e+11	3.960673e+11	4.278256e+11
3	RSS (Test)	1.677896e+21	7.917692e+20	4.159990e+20
4	MSE (Train)	2.024972e+04	2.101301e+04	2.183922e+04
5	MSE (Test)	2.087624e+09	1.434065e+09	1.039479e+09

In [88]:
```python
betas = pd.DataFrame(index=X[col].columns)
```

In [89]:
```python
betas.rows = X[col].columns
betas['Linear'] = lm.coef_
betas['Ridge'] = ridge.coef_
betas['Lasso'] = lasso.coef_
```

pd.set_option('display.max_rows', None) betas.head(50)

From the above table we can see that the doubling the alpha, value of the coefficient also changed. The most important feature is LotArea after doubling the alpha

In [ ]:

In [ ]:

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

Lasso eliminate feature completely so there are less number of features in X .Lasso regression would be a better option it would help in feature elimination and the model will be more robust.

Question 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?

Answer

The five most significant feature by lasso regression are

- LotArea
- OverallQual
- OverallCond
- BsmtFinSF1
- BsmtFinSF2

So lets remove these and train again

In [90]:
```python
X_train_new = X_train.drop(['LotArea', 'OverallQual', 'OverallCond','BsmtFinSF1','BsmtFinSF2'], axis=1)
X_test_new = X_test.drop(['LotArea', 'OverallQual', 'OverallCond','BsmtFinSF1','BsmtFinSF2'], axis=1)
```

In [91]:
```python
X_train_new.shape
```

Out[91]:
```
(897, 45)
```

In [92]:
```python
X_test_new.shape
```

Out[92]:
```
(385, 45)
```

In [93]:
```python
lasso_modified = Lasso()
param = {'alpha': [0.0001, 0.001, 0.01]}
folds = 5
# cross validation
lasso_cv_model_modified = GridSearchCV(estimator = lasso,
                        param_grid = param,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

lasso_cv_model_modified.fit(X_train_new, y_train)
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
```

Out[93]:
```
GridSearchCV(cv=5, estimator=Lasso(alpha=100),
             param_grid={'alpha': [0.0001, 0.001, 0.01]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)
```

In [94]:
```python
lasso_cv_new_results = pd.DataFrame(lasso_cv_model_modified.cv_results_)

lasso_cv_new_results.head()
```

Out[94]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	...	mean_test_score	std_test_score	rank_test_score	split0_train_score	split1_train_score	split2_train_score	split3_train_score
0	0.024535	0.001824	0.002232	0.000203	0.0001	{'alpha': 0.0001}	-19700.611294	-23065.675171	-22606.237204	-21154.911698	...	-21485.341501	1215.877029	1	-19858.738204	-19595.147030	-19290.291962	-19617.24087
1	0.020817	0.004050	0.001837	0.000486	0.001	{'alpha': 0.001}	-19700.616556	-23065.671486	-22606.854591	-21154.905155	...	-21485.462305	1215.989544	3	-19858.749123	-19595.154091	-19290.298041	-19617.24817
2	0.015662	0.000411	0.001337	0.000069	0.01	{'alpha': 0.01}	-19700.668893	-23065.634515	-22606.808048	-21154.839610	...	-21485.427108	1215.967193	2	-19858.858313	-19595.224692	-19290.359092	-19617.30928

3 rows × 21 columns

In [95]:
```python
# Printing the best hyperparameter alpha
model_cv.best_params_
```

Out[95]:
```
{'alpha': 50}
```

In [96]:
```python
lasso = Lasso(alpha=50)
lasso.fit(X_train_new,y_train)

y_train_pred = lasso.predict(X_train_new)
y_test_pred = lasso.predict(X_test_new)

print("Lasso Regression train r2:",r2_score(y_true=y_train,y_pred=y_train_pred))
print("Lasso Regression test r2:",r2_score(y_true=y_test,y_pred=y_test_pred))
```

```
Lasso Regression train r2: 0.7914417306990572
Lasso Regression test r2: -61447644.607664935
```

In [97]:
```python
model_param = list(lasso.coef_)
model_param.insert(0,lasso.intercept_)
cols = X_train_new.columns
cols.insert(0,'const')
lasso_coef = pd.DataFrame(list(zip(cols,model_param,(abs(ele) for ele in model_param))))
lasso_coef.columns = ['Feature','Coef','mod']
lasso_coef.sort_values(by='mod',ascending=False).head(5)
```

Out[97]:

	Feature	Coef	mod
16	Condition2_RRAe	-297024.912591	297024.912591
2	1stFlrSF	187545.233695	187545.233695
3	2ndFlrSF	155568.917420	155568.917420
4	LowQualFinSF	120102.580751	120102.580751
8	GarageArea	-102299.322824	102299.322824

So after the elimination of 5 most significant feature, here is new list

- Condition2_RRAe
- 1stFlrSF
- 2ndFlrSF
- LowQualFinSF
- GarageArea

In [ ]:

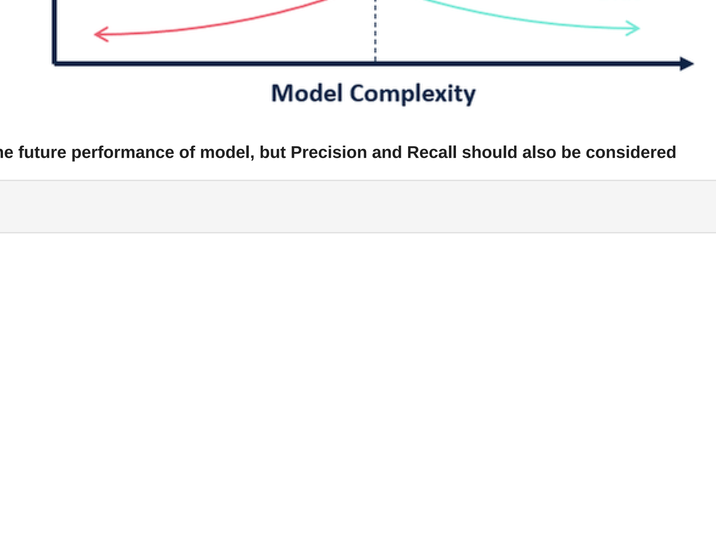
Question 4

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?

Answer:

In order to make the model more robust and generalisable

- Perform Exploratory Data Analaysis, such as handling null values, outliers, classify numerical value to category if necessary etc.,



- Maintain bias and variance trade-off as shown below

Although the accuracy is important factor to realize the future performance of model, but Precision and Recall should also be considered

In [ ]: