Assignment - Part 2

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?

   After doing hyperparameter tuning for both Ridge and Lasso I got the optimal alpha values of 0.3 and 100 respectively. If we double the values of both the hyperparameters we should see the Ridge Regression coefficients taking slightly lower absolute values due to the doubling of the hyperparameter, due to the fact that the weights would be penalized more due to higher value of regularization parameter. The Lasso Regression would also see similar trend but apart from this, it’s coefficient space would become even sparser as more of the coefficients would become zero. Also, I do feel that the top features would remain same in case of lasso as it is gives feature importance as well.

RIDGE

```
best_alpha = 0.3
model3 = Ridge(alpha=best_alpha)

model3.fit(X_train[top_50_features], y_train)
```

```
Ridge
Ridge(alpha=0.3)
```

```
intercept=model3.intercept_
coef=model3.coef_
print(intercept)
```

```
-201980.58014672442
```

```
top_features=pd.DataFrame({'variables' : X_train[top_50_features].columns
top_features['abs_val']=abs(top_features['coefficiets'])
top_features.sort_values(by=['abs_val'], ascending = False)
```

	variables	coefficiets	abs_val
32	Condition2_PosN	-247002.665094	247002.665094
42	RoofMatl_WdShngl	190585.143477	190585.143477
36	RoofMatl_CompShg	136656.764866	136656.764866
40	RoofMatl_Tar&Grv	136165.499307	136165.499307
37	RoofMatl_Membran	123991.479911	123991.479911
14	GrLivArea	116002.169717	116002.169717
38	RoofMatl_Metal	115093.522376	115093.522376
41	RoofMatl_WdShake	111407.693890	111407.693890
12	1stFlrSF	108121.368060	108121.368060
39	RoofMatl_Roll	97290.160390	97290.160390

```
In [72]: best_alpha = 0.3*2 #Doubling the alpha value and retraining
         model6 = Ridge(alpha=best_alpha)

         model6.fit(X_train[top_50_features], y_train)
```

```
Out[72]: Ridge
         Ridge(alpha=0.6)
```

```
In [74]: intercept=model6.intercept_
         coef=model6.coef_
         print(intercept)
```

```
-141649.7599823121
```

```
In [75]: top_features=pd.DataFrame({'variables' : X_train[top_50_feature
         top_features['abs_val']=abs(top_features['coefficiets'])
         top_features.sort_values(by=['abs_val'], ascending = False)
```

Out[75]:

	variables	coefficiets	abs_val
32	Condition2_PosN	-191306.347827	191306.347827
42	RoofMatl_WdShngl	131854.789829	131854.789829
14	GrLivArea	108199.304041	108199.304041
12	1stFlrSF	95903.525424	95903.525424
2	OverallQual	80555.076315	80555.076315
36	RoofMatl_CompShg	78347.329884	78347.329884
40	RoofMatl_Tar&Grv	76395.979020	76395.979020
13	2ndFlrSF	76018.722942	76018.722942
1	LotArea	68985.067412	68985.067412

As expected, we can clearly see that when we double our Ridge regularization Hyperparameter. The overall coefficients have reduced their values, and a few of the parameters have also flipped spots in scale of importance.

```python
y_pred_train = model3.predict(X_train[top_50_features])
y_pred_test = model3.predict(X_test[top_50_features])

r2_train_lr = r2_score(y_train, y_pred_train)
print('Train R_Squared Score:',r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print('Test R Squared Score:',r2_test_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print('Train RMSE:',sqrt(mse_train_lr))

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print('Test RMSE:',sqrt(mse_test_lr))

mae_train_lr = mean_absolute_error(y_train, y_pred_train)
print('Train MAE:',mae_train_lr)

mae_test_lr = mean_absolute_error(y_test, y_pred_test)
print('Test MAE:',mae_test_lr)

mape_train_lr = mean_absolute_percentage_error(y_train, y_pred_train)
print('Train MAPE:',mape_train_lr)

mape_test_lr = mean_absolute_percentage_error(y_test, y_pred_test)
print('Test MAPE:',mape_test_lr)
```

```
Train R_Squared Score: 0.8845090170452906
Test R Squared Score: 0.8610231700760392
Train RMSE: 26922.006117430527
Test RMSE: 29891.379550143876
Train MAE: 17465.33016383482
Test MAE: 19230.011125102752
Train MAPE: 0.10356290932114291
Test MAPE: 0.11743063500418419
```

```python
y_pred_train = model6.predict(X_train[top_50_features])
y_pred_test = model6.predict(X_test[top_50_features])

r2_train_lr = r2_score(y_train, y_pred_train)
print('Train R_Squared Score:',r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print('Test R Squared Score:',r2_test_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print('Train RMSE:',sqrt(mse_train_lr))

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print('Test RMSE:',sqrt(mse_test_lr))

mae_train_lr = mean_absolute_error(y_train, y_pred_train)
print('Train MAE:',mae_train_lr)

mae_test_lr = mean_absolute_error(y_test, y_pred_test)
print('Test MAE:',mae_test_lr)

mape_train_lr = mean_absolute_percentage_error(y_train, y_pred_train)
print('Train MAPE:',mape_train_lr)

mape_test_lr = mean_absolute_percentage_error(y_test, y_pred_test)
print('Test MAPE:',mape_test_lr)
```

```
Train R_Squared Score: 0.8778738176768008
Test R Squared Score: 0.8715980581461686
Train RMSE: 27684.569233918297
Test RMSE: 28731.64914250938
Train MAE: 17731.35269711243
Test MAE: 19021.97164156057
Train MAPE: 0.10479705873598451
Test MAPE: 0.11624645266390096
```

Also looking at the metrics the train accuracy has decreased very slightly but the model is performing slightly better on the test set now. But not a lot of change.

LASSO

```python
best_alpha = 100
model5 = Lasso(alpha=best_alpha)

model5.fit(X_train, y_train)
```

```
Lasso
Lasso(alpha=100)
```

```python
intercept=model5.intercept_
coef=model5.coef_
print(intercept)
print(coef)
```

```
-86090.28571166261
[-0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -1.10652057e+04  9.11664749e+04  2.94690247e+04  1.41173360e+04
  8.73801848e+02  4.37387735e+04  2.12928930e+04 -0.00000000e+00
  1.27594022e+04 -1.14160174e+04  3.34463067e+04  1.19301474e+04
  0.00000000e+00 -0.00000000e+00  0.00000000e+00 -9.63491684e+03
  0.00000000e+00  1.79312119e+03  8.07906195e+01  0.00000000e+00
  1.11666446e+04 -3.15761682e+03  2.47546775e+05  1.85966101e+04
  0.00000000e+00  1.13358511e+04  6.09907894e+03 -8.54291231e+03
 -1.58942539e+04  2.79280990e+04  1.12432775e+04  1.40739566e+04
  2.77176436e+03 -0.00000000e+00  1.04875284e+04  3.81251141e+04
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
  3.18622810e+03  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  2.67134365e+03 -0.00000000e+00 -0.00000000e+00 -3.47276139e+01
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00  7.10761805e+03
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -1.51882708e+04 -1.09646724e+04
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  9.02663108e+02 -3.46014661e+03  0.00000000e+00  1.12171030e+04
  7.91525079e+03  8.40325286e+03  1.26802283e+04 -4.12163489e+03
 -0.00000000e+00  1.21410680e+02 -0.00000000e+00  0.00000000e+00
  4.08840398e+03  4.11988277e+03 -9.49249646e+02  1.85081264e+04
```

```python
best_alpha = 100*2 #Doubling our alpha value and training Lasso
model7 = Lasso(alpha=best_alpha)

model7.fit(X_train, y_train)
```

```
Lasso
Lasso(alpha=200)
```

```python
intercept=model7.intercept_
coef=model7.coef_
print(intercept)
print(coef)
```

```
-52464.181899677205
[-0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -5.99541090e+03  9.80746498e+04  1.71344736e+04  4.72791948e+03
  3.78340218e+03  3.14610297e+04  2.27422075e+04  0.00000000e+00
  5.12684411e+01 -0.00000000e+00  3.36749669e+04  1.41807935e+04
  0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.89602764e+03
  0.00000000e+00  1.71393466e+03  0.00000000e+00  0.00000000e+00
  8.29504021e+03 -0.00000000e+00  2.13591432e+05  1.79794778e+04
  0.00000000e+00  8.69306273e+03  6.75638691e+03 -0.00000000e+00
 -1.90139496e+03  3.19971003e+04  6.64964194e+03  1.45265699e+04
  5.27620335e+03 -0.00000000e+00  9.25991407e+03  3.56110524e+04
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
  4.08024518e+03  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00  4.70984437e+03
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -1.40803081e+04 -1.07757215e+04
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  2.27846608e+02 -5.32727647e+03  0.00000000e+00  4.49532924e+03
  6.36018243e+02  1.90068170e+03  1.21022607e+04 -4.82390221e+02
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
```

As we were expecting a few more of the coefficients have become zero making the weights sparser.

```python
top_features=pd.DataFrame({'variables' : X_train.columns,'coefficiets' : coef})
top_features['abs_val']=abs(top_features['coefficiets'])
top_features.sort_values(by=['abs_val'], ascending = False)
```

	variables	coefficiets	abs_val
26	GrLivArea	247546.774739	247546.774739
113	Condition2_PosN	-161646.538815	161646.538815
5	OverallQual	91166.474853	91166.474853
139	RoofMatl_WdShngl	76346.484505	76346.484505
9	MasVnrArea	43738.773507	43738.773507
...	...	...	...
1	LotArea	0.000000	0.000000
114	Condition2_RRAe	-0.000000	0.000000
115	Condition2_RRAn	0.000000	0.000000
116	Condition2_RRNn	0.000000	0.000000
112	Condition2_PosA	0.000000	0.000000

```python
top_features=pd.DataFrame({'variables' : X_train.columns,'coefficiets' : coef})
top_features['abs_val']=abs(top_features['coefficiets'])
top_features.sort_values(by=['abs_val'], ascending = False)
```

	variables	coefficiets	abs_val
26	GrLivArea	213591.431730	213591.431730
5	OverallQual	98074.649759	98074.649759
92	Neighborhood_NoRidge	44935.262127	44935.262127
139	RoofMatl_WdShngl	42511.991206	42511.991206
93	Neighborhood_NridgHt	41660.597455	41660.597455
...	...	...	...
104	Condition1_PosA	0.000000	0.000000
105	Condition1_PosN	-0.000000	0.000000
106	Condition1_RRAe	-0.000000	0.000000
108	Condition1_RRNe	0.000000	0.000000
112	Condition2_PosA	0.000000	0.000000

224 rows × 3 columns

Top features have flipped here as well which I was not expecting. I thought that it would not happen in lasso as it gives feature importance and they would decrease in the order of the coefficients assigned. Good Experiment to know this.

```python
y_pred_train = model5.predict(X_train)
y_pred_test = model5.predict(X_test)

r2_train_lr = r2_score(y_train, y_pred_train)
print('Train R_Squared Score:',r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print('Test R Squared Score:',r2_test_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print('Train RMSE:',sqrt(mse_train_lr))

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print('Test RMSE:',sqrt(mse_test_lr))

mae_train_lr = mean_absolute_error(y_train, y_pred_train)
print('Train MAE:',mae_train_lr)

mae_test_lr = mean_absolute_error(y_test, y_pred_test)
print('Test MAE:',mae_test_lr)

mape_train_lr = mean_absolute_percentage_error(y_train, y_pred_train)
print('Train MAPE:',mape_train_lr)

mape_test_lr = mean_absolute_percentage_error(y_test, y_pred_test)
print('Test MAPE:',mape_test_lr)
```

```
Train R_Squared Score: 0.8799905104899773
Test R Squared Score: 0.8808073533855545
Train RMSE: 27443.605891757223
Test RMSE: 27682.129125718562
Train MAE: 17063.354918678757
Test MAE: 17908.475161338327
Train MAPE: 0.09997504699645011
Test MAPE: 0.10625244186670761
```

```python
y_pred_train = model7.predict(X_train)
y_pred_test = model7.predict(X_test)

r2_train_lr = r2_score(y_train, y_pred_train)
print('Train R_Squared Score:',r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print('Test R Squared Score:',r2_test_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print('Train RMSE:',sqrt(mse_train_lr))

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print('Test RMSE:',sqrt(mse_test_lr))

mae_train_lr = mean_absolute_error(y_train, y_pred_train)
print('Train MAE:',mae_train_lr)

mae_test_lr = mean_absolute_error(y_test, y_pred_test)
print('Test MAE:',mae_test_lr)

mape_train_lr = mean_absolute_percentage_error(y_train, y_pred_train)
print('Train MAPE:',mape_train_lr)

mape_test_lr = mean_absolute_percentage_error(y_test, y_pred_test)
print('Test MAPE:',mape_test_lr)
```

```
Train R_Squared Score: 0.8596714749454029
Test R Squared Score: 0.8775652160534193
Train RMSE: 29676.06816800741
Test RMSE: 28056.09142755838
Train MAE: 18011.714998363357
Test MAE: 18019.742492799953
Train MAPE: 0.10497431470102886
Test MAPE: 0.10784457999312738
```

The metrics have also remained quite similar with very slight decrease after we doubled alpha.

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?

I got alpha value of 0.3 in ridge regression using GridSearchCV but, I still chose my default Ridge Regression with alpha value of 1 as it seemed to be generalizing well to the test set. At alpha value 0.3 it was overfitting to the train set.

For Lasso I did find that it was giving better results on test set, than (default alpha value) at alpha value of 100. So, I chose it as my final model.

I would use the Test set metrics to pick my model any day if I see that both Train and Test metrics are very similar (meaning my model is generalizing well).

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?

   This question is quite experimental so let's try it out by dropping our 5 most important predictors in Lasso Model.
   The updated 5 most important predictors after having dropped top 5 predictors as shown in the right image, left image showing the previous 5 important predictors.

```python
top_features=pd.DataFrame({'variables' : X_train.columns,'coefficiets' : coef})
top_features['abs_val']=abs(top_features['coefficiets'])
top_features.sort_values(by=['abs_val'], ascending = False)
```

	variables	coefficiets	abs_val
26	GrLivArea	247546.774739	247546.774739
113	Condition2_PosN	-161646.538815	161646.538815
5	OverallQual	91166.474853	91166.474853
139	RoofMatl_WdShngl	76346.484505	76346.484505
9	MasVnrArea	43738.773507	43738.773507
...	...	...	...
1	LotArea	0.000000	0.000000
114	Condition2_RRAe	-0.000000	0.000000
115	Condition2_RRAn	0.000000	0.000000
116	Condition2_RRNn	0.000000	0.000000
112	Condition2_PosA	0.000000	0.000000

```python
top_features=pd.DataFrame({'variables' : X_train_new.columns,'coefficiets' : coef})
top_features['abs_val']=abs(top_features['coefficiets'])
top_features.sort_values(by=['abs_val'], ascending = False)
```

	variables	coefficiets	abs_val
21	1stFlrSF	218976.190734	218976.190734
22	2ndFlrSF	122462.203482	122462.203482
90	Neighborhood_NridgHt	52931.472761	52931.472761
89	Neighborhood_NoRidge	51112.740209	51112.740209
96	Neighborhood_StoneBr	46793.498662	46793.498662
...	...	...	...
112	Condition2_RRNn	0.000000	0.000000
113	BldgType_2fmCon	-0.000000	0.000000
114	BldgType_Duplex	-0.000000	0.000000
119	HouseStyle_2.5Fin	-0.000000	0.000000
109	Condition2_PosA	0.000000	0.000000

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 make our model robust and generalisable one needs to have a good balance between overfitting and underfitting (bias variance trade-off).

   One would get robust model if one archives good training accuracy but, in some cases, it might not reflect on the test data. This is clear case of overfitting. One needs to also try and bring the test accuracy on par with the train accuracy (as close as possible) to get a generalizable model.

   In other words, in high variance situation our model is not only learning patterns that are useful but it is so complex or trained for such long periods that it trying to learn noise in the data as well. Then later on would not predict well in case of test data.

   While in case of high bias situation we might have selected not optimized (not properly trained), very simple model that is not capable of modelling complex data. In this case both train and test accuracy will be very low.

   The implications of the same on model metric in case of Overfitting would be unreasonably high metric scores on train data, and very low scores on test data. While in case of Underfitting it would give bad metric scores for both train and test data. To identify these one should always hold out some train data that the model has never seen for testing the model on unseen data points.