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:

```
##Optimal Value of alpha for ridge and lasso regression

optimal_alpha_ridge = 8.0      #(Computed Above: For Ridge Regression)
optimal_alpha_lasso = 0.001    #(Computed Above: For Lasso Regression)
```

Changes to the model when we double the value of alpha for both ridge and lasso regression

Ridge Regression

In [710]:

```
##Checking the outcome: coefficient values with double the value of alpha = 8*2 = 16

ridge = Ridge(alpha=16)

ridge.fit(X_train, y_train)
print("Intercept: ", ridge.intercept_)
print("Coefficients:\n",ridge.coef_)
```

```
Intercept:  -0.1296606114567678
Coefficients:
 [ 0.24224166  0.15628609  0.10582416  0.1456706   0.35751861  0.10847221
 -0.27847847  0.02328501  0.15068923  0.00392081 -0.13762511 -0.19444553
  0.08085395  0.20222076  0.16677873 -0.08084691  0.1565555  -0.05164921
 -0.10782731 -0.1159138  -0.10064093 -0.07962394 -0.00295994 -0.07494303
  0.11547805 -0.10354892 -0.06553052  0.12225787  0.01599864 -0.05475906
  0.06061098  0.02551161 -0.0294213  -0.02358916  0.03091463 -0.01717715
  0.05451808  0.04739797  0.04734031  0.05829194 -0.02658924 -0.0303841
  0.04508568 -0.04973488  0.05122138 -0.07733566  0.11897601  0.02918533
 -0.06086114 -0.10235333]
```

In [711]:

```
##Making predictions for train and test sets: Ridge Regression Model
y_pred_train_r = ridge.predict(X_train)
y_pred_test_r = ridge.predict(X_test)

##R2 score for Ridge Regression Model
r2_score_ridge_train = r2_score(y_true= y_train, y_pred= y_pred_train_r)
r2_score_ridge_test = r2_score(y_true= y_test, y_pred= y_pred_test_r)
```

```
##Check the mean squared error (MSE) for Ridge Regression Model
MSE_ridge_train = mean_squared_error(y_train, y_pred_train_r)
MSE_ridge_test = mean_squared_error(y_test, y_pred_test_r)

##Mean Absolute error for train and test sets
MAE_ridge_train = mean_absolute_error(y_train, y_pred_train_r)
MAE_ridge_test = mean_absolute_error(y_test, y_pred_test_r)

##Root Mean Squared Error for Train and Test Sets
RMSE_ridge_train = np.sqrt(MSE_ridge_train)
RMSE_ridge_test = np.sqrt(MSE_ridge_test)


print("For Ridge Regression Model (Doubled alpha model, alpha=8*2=16):\n","*"*40)
print("\nFor Train Set:\nR2 score:",r2_score_ridge_train,"\nMSE score:",MSE_ridge_train,"\nMAE score:",
MAE_ridge_train,\
   "\nRMSE score:",RMSE_ridge_train)
print("\nFor Test Set:\nR2 score:",r2_score_ridge_test,"\nMSE score:",MSE_ridge_test,"\nMAE score:",MA
E_ridge_test,\
    "\nRMSE score:",RMSE_ridge_test,"\n","*"*40)
```

```
For Ridge Regression Model (Doubled alpha model, alpha=8*2=16):
 ****************************************

For Train Set:
R2 score: 0.9118928405717794
MSE score: 0.08810715942822064
MAE score: 0.21267431891866817
RMSE score: 0.2968285017113765

For Test Set:
R2 score: 0.8904731985528808
MSE score: 0.10617901905032019
MAE score: 0.21782052246333078
RMSE score: 0.32585122226304475
 ****************************************
```

In [712]:

```
##Creating a dataframe of features and coefficients

ridge_df = pd.DataFrame({'Features':X_train.columns, 'Coefficient':ridge.coef_.round(4),
                'Abs_Coefficient_Ridge(Desc_Sort)':abs(ridge.coef_.round(4))})

##Sorting coefficient in descending order of absolute values and reset index
ridge_df = ridge_df.sort_values(by='Abs_Coefficient_Ridge(Desc_Sort)', ascending=False)
ridge_df.reset_index(drop=True, inplace=True)

#Dataframe rdige_df
ridge_df.head(10)   #Top10 features display
```

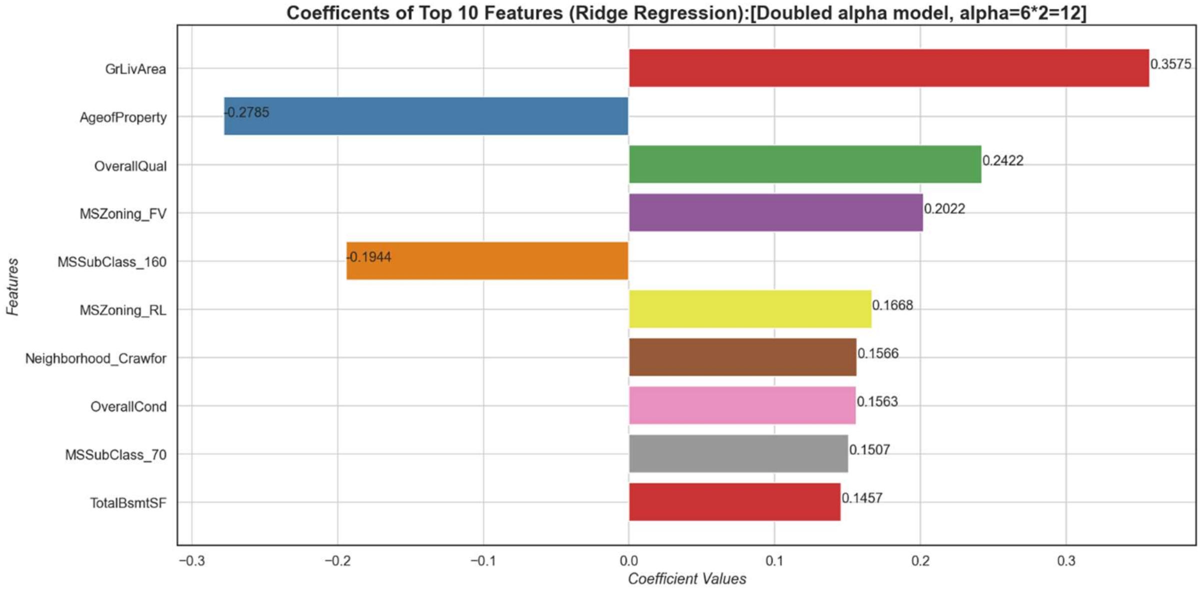
Out[712]:

	Features	Coefficient	Abs_Coefficient_Ridge(Desc_Sort)
0	GrLivArea	0.3575	0.3575
1	AgeofProperty	-0.2785	0.2785
2	OverallQual	0.2422	0.2422
3	MSZoning_FV	0.2022	0.2022
4	MSSubClass_160	-0.1944	0.1944
5	MSZoning_RL	0.1668	0.1668
6	Neighborhood_Crawfor	0.1566	0.1566
7	OverallCond	0.1563	0.1563
8	MSSubClass_70	0.1507	0.1507
9	TotalBsmtSF	0.1457	0.1457

In [713]:

```
##Coefficient value plot (Ridge Regression)

top10_ridge_df= ridge_df.loc[:9] #Ridge_df with top 10 coefficients

sns.set(style='white')
plt.figure(figsize=(16,8), dpi=120)
ax3= sns.barplot(y=top10_ridge_df['Features'], x=top10_ridge_df['Coefficient'], palette='Set1')

plt.xlabel('Coefficient Values', fontsize= 14, fontstyle='italic')
plt.ylabel('Features' , fontsize= 14, fontstyle='italic')
plt.title('Coefficents of Top 10 Features (Ridge Regression):[Doubled alpha model, alpha=6*2=12]', fontsi
ze=18,fontweight='bold')

coef= top10_ridge_df['Coefficient'] #Storing coefficient values
for index, value in enumerate(coef):
    plt.text(value, index, str(value), fontsize=13)

plt.grid(True)
plt.xticks(fontsize=13)
```

```
plt.yticks(fontsize=13)
plt.autoscale()
plt.tight_layout()
plt.show()
```



In [714]:

```
print("For Ridge Regression (Doubled alpha model, alpha=8*2=16): \n","*"*125)
print("The most important top10 predictor variables after the change is implemented are as follows:\n\n",\
    list(top10_ridge_df['Features']),"\n", "*"*125)
```

```
For Ridge Regression (Doubled alpha model, alpha=8*2=16):
 *****************************************************************************************************************************
The most important top10 predictor variables after the change is implemented are as follows:

 ['GrLivArea', 'AgeofProperty', 'OverallQual', 'MSZoning_FV', 'MSSubClass_160', 'MSZoning_RL', 'Neighborhood_Crawfor', 'OverallCond', 'MSSubClass_70', 'TotalBsmtSF']
```

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:

Even though Ridge regression has given good performance, I would choose Lasso model for following reasons.

- It is giving decent performance.
- Efficiently solved high dimensionality problem by shrinking insignificant coefficients to zero.
- Simpler model and easy for maintenance.

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:

```
##From Original Lasso Regression Model, import 'top5_original_lasso_features': Top5 features
print("Top 5 features in original lasso model (dropped):\n", top5_original_lasso_features)

df= df_new1

##Removing these top5 features (as per Original Lasso Model) from 'df'
df= df.drop(top5_original_lasso_features, axis=1)
df.head()
```

```
Top 5 features in original lasso model (dropped):
 ['GrLivArea', 'MSZoning_FV', 'MSSubClass_160', 'Exterior1st_BrkComm', 'Age
ofProperty']
```

```
##Creating a function to find binary value columns from the 'df' dataframe (if any)
def binary_val_cols(df):
  df_1 = df.copy()
  dualsvcol = (df_1.nunique()==2)
  list_dualsvcol = list(dualsvcol[dualsvcol.values==True].index)
  return list_dualsvcol

binary_cols = binary_val_cols(df)
```

Train Test Split

In [722]:

```
##split into train and test

from sklearn.model_selection import train_test_split
np.random.seed(0)
df_train, df_test = train_test_split(df, train_size=0.7, test_size = 0.3, random_state=100)
```

Feature Scaling

In [723]:

```
##Dataframe with binary columns
df_binary_train = df_train.loc[:, binary_cols]
df_binary_test = df_test.loc[:, binary_cols]

##Dropping binary dummy variables and we shall concat them later to preserve the scale
df_train = df_train.drop(binary_cols, axis=1)
df_test = df_test.drop(binary_cols, axis=1)

##StandardScaler

from sklearn.preprocessing import StandardScaler
all_cols =df_train.columns
scaler = StandardScaler()

#scaler fit_transform on train data
df_train[all_cols] = scaler.fit_transform(df_train[all_cols])
#concat dummies:Train set
df_train = pd.concat([df_train, df_binary_train], axis=1)

#scaler fit_transform on test data
df_test[all_cols] = scaler.transform(df_test[all_cols])
#concat dummies: Test set
df_test = pd.concat([df_test, df_binary_test], axis=1)
```

In [724]:

```
##Storing target variable to y_train and y_test respectively

y_train = df_train['SalePrice']
y_test = df_test['SalePrice']
```

In [725]:

```
##Storing all feature variables to X_train and X_test
X_train = df_train.drop('SalePrice',axis=1)
X_test = df_test.drop('SalePrice',axis=1)
```

Recursive Feature Elimination

In [726]:

```
##Running RFE with the output number of the variable equal to 50
lm = LinearRegression()
lm.fit(X_train, y_train)

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

In [727]:

```
##my_zip file zips features, rfe.support_ and rfe.ranking_
my_zip = list(zip(X_train.columns,rfe.support_,rfe.ranking_))
my_zip
```

Out[727]:

```
[('LotFrontage', False, 24),
 ('LotArea', True, 1),
 ('OverallQual', True, 1),
 ('OverallCond', True, 1),
 ('MasVnrArea', False, 45),
 ('BsmtFinSF1', False, 22),
 ('BsmtUnfSF', True, 1),
 ('TotalBsmtSF', True, 1),
 ('BsmtFullBath', False, 35),
 ('FullBath', True, 1),
 ('HalfBath', True, 1),
 ('BedroomAbvGr', False, 39),
 ('Fireplaces', False, 14),
 ('GarageArea', True, 1),
 ('WoodDeckSF', False, 25),
 ('OpenPorchSF', False, 41),
 ('d_LotShape', False, 50),
 ('d_BsmtQual', False, 18),
 ('d_BsmtExposure', False, 44),
 ('d_HeatingQC', False, 21),
 ('d_GarageFinish', False, 48),
 ('WhetherRemodelled', False, 46),
 ('MSSubClass_30', True, 1),
 ('MSSubClass_40', False, 38),
 ('MSSubClass_45', True, 1),
 ('MSSubClass_50', True, 1),
 ('MSSubClass_60', True, 1),
 ('MSSubClass_70', True, 1),
 ('MSSubClass_75', True, 1),
 ('MSSubClass_80', True, 1),
 ('MSSubClass_85', False, 34),
 ('MSSubClass_90', False, 37),
 ('MSSubClass_120', False, 9),
 ('MSSubClass_180', False, 23),
 ('MSSubClass_190', False, 27),
 ('LotConfig_CulDSac', False, 12),
 ('LotConfig_FR2', False, 16),
 ('LotConfig_FR3', True, 1),
 ('LotConfig_Inside', False, 36),
 ('MSZoning_RH', False, 47),
 ('MSZoning_RL', False, 42),
 ('Neighborhood_Blueste', True, 1),
```

```
('Neighborhood_BrDale', True, 1),
('Neighborhood_BrkSide', True, 1),
('Neighborhood_ClearCr', True, 1),
('Neighborhood_CollgCr', True, 1),
('Neighborhood_Crawfor', False, 29),
('Neighborhood_Edwards', True, 1),
('Neighborhood_Gilbert', True, 1),
('Neighborhood_IDOTRR', True, 1),
('Neighborhood_MeadowV', True, 1),
('Neighborhood_Mitchel', True, 1),
('Neighborhood_NAmes', True, 1),
('Neighborhood_NPkVill', True, 1),
('Neighborhood_NWAmes', True, 1),
('Neighborhood_NoRidge', False, 10),
('Neighborhood_NridgHt', False, 11),
('Neighborhood_OldTown', True, 1),
('Neighborhood_SWISU', True, 1),
('Neighborhood_Sawyer', True, 1),
('Neighborhood_SawyerW', True, 1),
('Neighborhood_StoneBr', True, 1),
('Neighborhood_Timber', True, 1),
('Neighborhood_Veenker', True, 1),
('Exterior2nd_AsphShn', False, 17),
('Exterior2nd_Brk Cmn', True, 1),
('Exterior2nd_BrkFace', True, 1),
('Exterior2nd_CBlock', False, 31),
('Exterior2nd_CmentBd', False, 3),
('Exterior2nd_HdBoard', False, 33),
('Exterior2nd_ImStucc', False, 32),
('Exterior2nd_MetalSd', False, 7),
('Exterior2nd_Other', False, 49),
('Exterior2nd_Plywood', False, 20),
('Exterior2nd_Stone', True, 1),
('Exterior2nd_Stucco', False, 19),
('Exterior2nd_VinylSd', False, 5),
('Exterior2nd_Wd Sdng', False, 8),
('Exterior2nd_Wd Shng', False, 13),
('HouseStyle_1Story', False, 28),
('HouseStyle_2.5Fin', True, 1),
('HouseStyle_2.5Unf', True, 1),
('Foundation_CBlock', False, 30),
('Foundation_Slab', True, 1),
('Foundation_Stone', True, 1),
('Foundation_Wood', True, 1),
('MasVnrTyp_BrkFace', False, 51),
('MasVnrTyp_Stone', False, 6),
('RoofStyle_Gable', False, 15),
```

```
 ('RoofStyle_Gambrel', True, 1),
 ('RoofStyle_Mansard', True, 1),
 ('RoofStyle_Shed', False, 40),
 ('Exterior1st_AsphShn', False, 52),
 ('Exterior1st_BrkFace', False, 2),
 ('Exterior1st_ImStucc', False, 53),
 ('Exterior1st_Stone', True, 1),
 ('Exterior1st_WdShing', False, 26),
 ('GarageType_Attchd', True, 1),
 ('GarageType_Basment', False, 43),
 ('GarageType_BuiltIn', True, 1),
 ('GarageType_CarPort', True, 1),
 ('GarageType_None', False, 4)]
```

In [728]:

```
##Checking columns that have RFE support
col_rfe_sup = X_train.columns[rfe.support_]
col_rfe_sup
```

Out[728]:

```
Index(['LotArea', 'OverallQual', 'OverallCond', 'BsmtUnfSF', 'TotalBsmtSF',
       'FullBath', 'HalfBath', 'GarageArea', 'MSSubClass_30', 'MSSubClass_4
5',
       'MSSubClass_50', 'MSSubClass_60', 'MSSubClass_70', 'MSSubClass_75',
       'MSSubClass_80', 'LotConfig_FR3', 'Neighborhood_Blueste',
       'Neighborhood_BrDale', 'Neighborhood_BrkSide', 'Neighborhood_ClearCr
',
       'Neighborhood_CollgCr', 'Neighborhood_Edwards', 'Neighborhood_Gilber
t',
       'Neighborhood_IDOTRR', 'Neighborhood_MeadowV', 'Neighborhood_Mitchel
',
       'Neighborhood_NAmes', 'Neighborhood_NPkVill', 'Neighborhood_NWAmes',
       'Neighborhood_OldTown', 'Neighborhood_SWISU', 'Neighborhood_Sawyer',
       'Neighborhood_SawyerW', 'Neighborhood_StoneBr', 'Neighborhood_Timber
',
       'Neighborhood_Veenker', 'Exterior2nd_Brk Cmn', 'Exterior2nd_BrkFace'
,
       'Exterior2nd_Stone', 'HouseStyle_2.5Fin', 'HouseStyle_2.5Unf',
       'Foundation_Slab', 'Foundation_Stone', 'Foundation_Wood',
       'RoofStyle_Gambrel', 'RoofStyle_Mansard', 'Exterior1st_Stone',
       'GarageType_Attchd', 'GarageType_BuiltIn', 'GarageType_CarPort'],
      dtype='object')
```

In [729]:

```
##Creating a dataframe for RFE supported top 50 indepedent variables.

top50_df = pd.DataFrame(my_zip, columns=['Features', 'rfe_support', 'rfe_ranking']) # assign the 50 featur
es selected using RFE to a dataframe and view them

top50_df = top50_df.loc[top50_df['rfe_support'] == True]
```

```
top50_df.reset_index(drop=True, inplace=True)

top50_df
```

Out[729]:

	Features	rfe_support	rfe_ranking
0	LotArea	True	1
1	OverallQual	True	1
2	OverallCond	True	1
3	BsmtUnfSF	True	1
4	TotalBsmtSF	True	1
5	FullBath	True	1
6	HalfBath	True	1
7	GarageArea	True	1
8	MSSubClass_30	True	1
9	MSSubClass_45	True	1
10	MSSubClass_50	True	1
11	MSSubClass_60	True	1
12	MSSubClass_70	True	1
13	MSSubClass_75	True	1
14	MSSubClass_80	True	1
15	LotConfig_FR3	True	1
16	Neighborhood_Blueste	True	1

	Features	rfe_support	rfe_ranking
17	Neighborhood_BrDale	True	1
18	Neighborhood_BrkSide	True	1
19	Neighborhood_ClearCr	True	1
20	Neighborhood_CollgCr	True	1
21	Neighborhood_Edwards	True	1
22	Neighborhood_Gilbert	True	1
23	Neighborhood_IDOTRR	True	1
24	Neighborhood_MeadowV	True	1
25	Neighborhood_Mitchel	True	1
26	Neighborhood_NAmes	True	1
27	Neighborhood_NPkVill	True	1
28	Neighborhood_NWAmes	True	1
29	Neighborhood_OldTown	True	1
30	Neighborhood_SWISU	True	1
31	Neighborhood_Sawyer	True	1
32	Neighborhood_SawyerW	True	1
33	Neighborhood_StoneBr	True	1
34	Neighborhood_Timber	True	1
35	Neighborhood_Veenker	True	1

	Features	rfe_support	rfe_ranking
36	Exterior2nd_Brk Cmn	True	1
37	Exterior2nd_BrkFace	True	1
38	Exterior2nd_Stone	True	1
39	HouseStyle_2.5Fin	True	1
40	HouseStyle_2.5Unf	True	1
41	Foundation_Slab	True	1
42	Foundation_Stone	True	1
43	Foundation_Wood	True	1
44	RoofStyle_Gambrel	True	1
45	RoofStyle_Mansard	True	1
46	Exterior1st_Stone	True	1
47	GarageType_Attchd	True	1
48	GarageType_BuiltIn	True	1
49	GarageType_CarPort	True	1

In [730]:

```
##Let's Assign top 50 columns to X_train_rfe

X_train_rfe = X_train[col_rfe_sup]
```

In [731]:

```
##Making sure that we have only 50 features (supported by RFE) in X_train and X_test for further analysis

X_train = X_train_rfe[X_train_rfe.columns]
X_test =  X_test[X_train.columns]
```

Model Building: Lasso Regression Model

In [732]:

```
##Lasso Regression Model Building
```

```
lasso = Lasso()

##List of alphas (lambda parameter)

params_1 = {'alpha': [0.00001, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007,0.0008, 0.0009, 0.0
01, 0.005, 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 ]}

##Cross-Validation

folds = 5
lasso_model_cv = GridSearchCV(estimator = lasso,
                      param_grid = params_1,
                      scoring= 'neg_mean_absolute_error',
                      cv = folds,
                      return_train_score=True,
                      verbose = 1)

lasso_model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 38 candidates, totalling 190 fits
```

Out[732]:

GridSearchCV
estimator: Lasso

```
Lasso
```

In [733]:

```
##Display the mean scores

lasso_cv_results = pd.DataFrame(lasso_model_cv.cv_results_)
lasso_cv_results[['param_alpha', 'mean_train_score', 'mean_test_score', 'rank_test_score']].sort_values(by
= ['rank_test_score'])
```

Out[733]:

	param_alpha	mean_train_score	mean_test_score	rank_test_score
4	0.0004	-0.234596	-0.249241	1
5	0.0005	-0.235143	-0.249248	2
6	0.0006	-0.235763	-0.249445	3
3	0.0003	-0.234108	-0.249470	4
2	0.0002	-0.233778	-0.249820	5

	param_alpha	mean_train_score	mean_test_score	rank_test_score
7	0.0007	-0.236438	-0.249826	6
8	0.0008	-0.237125	-0.250209	7
1	0.0001	-0.233522	-0.250210	8
9	0.0009	-0.237848	-0.250543	9
0	0.00001	-0.233426	-0.250817	10
10	0.001	-0.238539	-0.251007	11
11	0.005	-0.258606	-0.264667	12
12	0.01	-0.270600	-0.274730	13
13	0.05	-0.296756	-0.299077	14
14	0.1	-0.323270	-0.324466	15
15	0.2	-0.384300	-0.386380	16
16	0.3	-0.453846	-0.456101	17
17	0.4	-0.518116	-0.520151	18
18	0.5	-0.579658	-0.581251	19
19	0.6	-0.636055	-0.637773	20
20	0.7	-0.697506	-0.699013	21
21	0.8	-0.761580	-0.762717	22
35	100	-0.775470	-0.775675	23
34	50	-0.775470	-0.775675	23

	param_alpha	mean_train_score	mean_test_score	rank_test_score
33	20	-0.775470	-0.775675	23
32	10.0	-0.775470	-0.775675	23
31	9.0	-0.775470	-0.775675	23
30	8.0	-0.775470	-0.775675	23
29	7.0	-0.775470	-0.775675	23
28	6.0	-0.775470	-0.775675	23
27	5.0	-0.775470	-0.775675	23
26	4.0	-0.775470	-0.775675	23
25	3.0	-0.775470	-0.775675	23
24	2.0	-0.775470	-0.775675	23
23	1.0	-0.775470	-0.775675	23
22	0.9	-0.775470	-0.775675	23
36	500	-0.775470	-0.775675	23
37	1000	-0.775470	-0.775675	23

Focusing on smaller alpha values based on above data

In [734]:

```
##Plotting a magnified graph for a lower range of alpha.

lasso = Lasso()

##List of alphas (lambda parameter: consider smaller range on the basis of lasso_cv_results table ranking)

params_2 = {'alpha': [0.00001, 0.00009, 0.00005, 0.00003, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006
, 0.0007, 0.0008, 0.0009, 0.001, 0.005, 0.01, 0.02, 0.05]}

##Cross-Validation
```

```python
folds = 5
lasso_model_cv = GridSearchCV(estimator = lasso,
                param_grid = params_2,
                scoring= 'neg_mean_absolute_error',
                cv = folds,
                return_train_score=True,
                verbose = 1)

lasso_model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

Out[734]:

GridSearchCV
estimator: Lasso

```
Lasso
```

In [735]:

```python
##ReDisplay the mean scores

lasso_cv_results = pd.DataFrame(lasso_model_cv.cv_results_)
lasso_cv_results[['param_alpha', 'mean_train_score', 'mean_test_score', 'rank_test_score']].sort_values(by
= ['rank_test_score'])
```

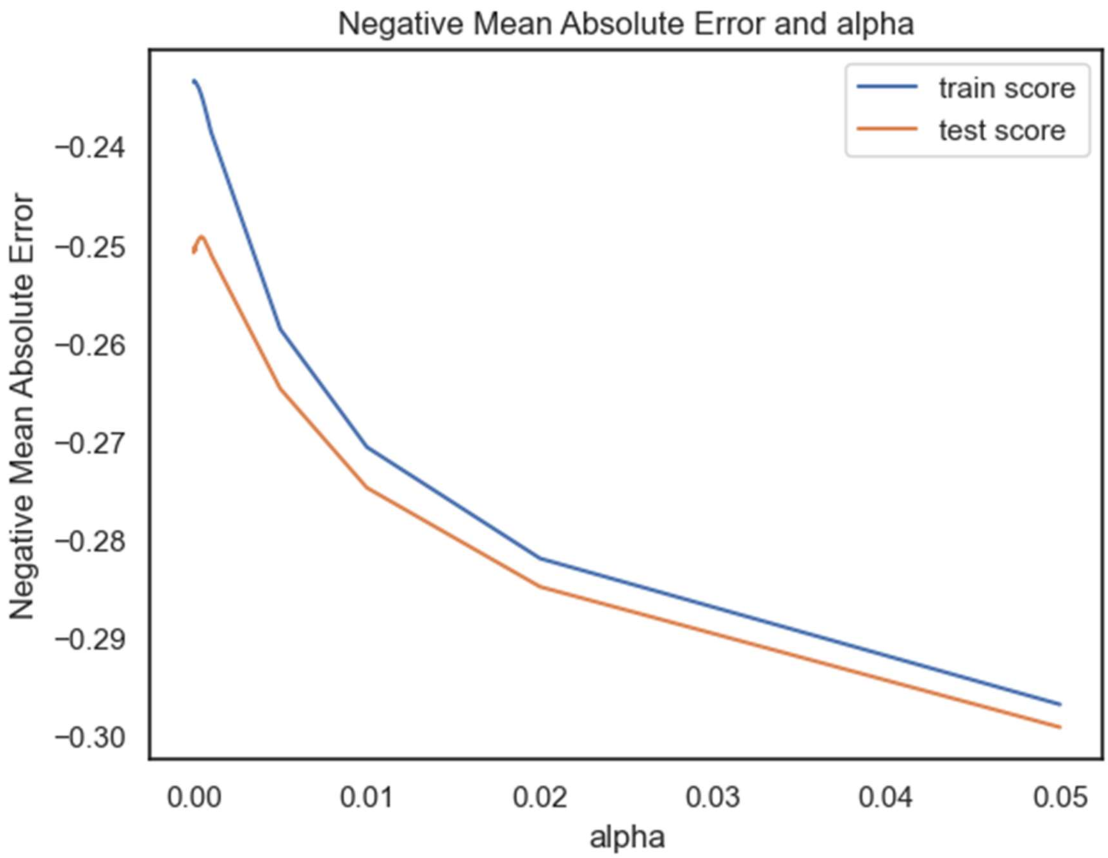
Out[735]:

	param_alpha	mean_train_score	mean_test_score	rank_test_score
7	0.0004	-0.234596	-0.249241	1
8	0.0005	-0.235143	-0.249248	2
9	0.0006	-0.235763	-0.249445	3
6	0.0003	-0.234108	-0.249470	4
5	0.0002	-0.233778	-0.249820	5
10	0.0007	-0.236438	-0.249826	6
11	0.0008	-0.237125	-0.250209	7
4	0.0001	-0.233522	-0.250210	8
1	0.00009	-0.233508	-0.250271	9
2	0.00005	-0.233457	-0.250538	10

	param_alpha	mean_train_score	mean_test_score	rank_test_score
12	0.0009	-0.237848	-0.250543	11
3	0.00003	-0.233437	-0.250671	12
0	0.00001	-0.233426	-0.250817	13
13	0.001	-0.238539	-0.251007	14
14	0.005	-0.258606	-0.264667	15
15	0.01	-0.270600	-0.274730	16
16	0.02	-0.281924	-0.284801	17
17	0.05	-0.296756	-0.299077	18

In [736]:

```
##Plotting mean test and train scoes with alpha
lasso_cv_results['param_alpha'] = lasso_cv_results['param_alpha'].astype('float64')

##plotting
plt.plot(lasso_cv_results['param_alpha'], lasso_cv_results['mean_train_score'])
plt.plot(lasso_cv_results['param_alpha'], lasso_cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('Negative Mean Absolute Error')

plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper right')
plt.show()
```



In [737]:

```
# get the best estimator for lambda

lasso_model_cv.best_estimator_
```

Out[737]:

```
Lasso
 Lasso(alpha=0.0004)
```

In [738]:

```
# check the coefficient values with (lambda) alpha = 0.0004

lasso = Lasso(alpha=0.0004)

lasso.fit(X_train, y_train)
print("Intercepts: ",lasso.intercept_)
print("Coefficients:\n",lasso.coef_)
```

```
Intercepts:  0.07584571170654805
Coefficients:
 [ 0.09257452  0.30778982  0.13188774 -0.11390878  0.37373075  0.12144053
  0.09583899  0.15502864 -0.25033946 -0.16476066  0.07040289  0.26596687
  0.21617641  0.11803384  0.13878431 -0.         -0.         -0.45735369
 -0.17095864 -0.11092654 -0.1447268  -0.24559687 -0.12455549 -0.60633011
 -0.47223459 -0.27919006 -0.35014965 -0.12686095 -0.38044102 -0.47105564
 -0.10230161 -0.35523233 -0.18489726  0.10850556 -0.11484867 -0.05465814
 -0.43401763  0.17806416  0.2235303   0.18216979 -0.          0.36600412
  0.16512971 -0.02406037  0.09251517  0.09754496  0.          0.1060746
```

```
0.33546336 -0.15548198]
```

In [739]:

```python
##Creating a dataframe of features and coefficients

lasso_df = pd.DataFrame({'Features':X_train.columns, 'Coefficient':lasso.coef_.round(4), \
                'Abs_Coefficient_Lasso(Desc_Sort)':abs(lasso.coef_.round(4))})
##Sorting coefficient in descending order of absolute values and reset index
lasso_df = lasso_df.sort_values(by='Abs_Coefficient_Lasso(Desc_Sort)', ascending=False)
lasso_df.reset_index(drop=True, inplace=True)

#lasso df
lasso_df.head(5)  #The Top5 features display
```

Out[739]:

	Features	Coefficient	Abs_Coefficient_Lasso(Desc_Sort)
0	Neighborhood_IDOTRR	-0.6063	0.6063
1	Neighborhood_MeadowV	-0.4722	0.4722
2	Neighborhood_OldTown	-0.4711	0.4711
3	Neighborhood_BrDale	-0.4574	0.4574
4	Exterior2nd_Brk Cmn	-0.4340	0.4340

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:

- A model is **robust** when any variation in the data does not affect its performance much.
- A **generalizable** model is able to adapt properly to new, previously unseen data, drawn from the same distribution as the one used to create the model.
- To make sure a model is robust and generalizable, we have to **take care it doesn't overfit**. This is because an overfitting model has very high variance and a smallest

change in data affects the model prediction heavily. Such a model will identify all the patterns of a training data, but fail to pick up the patterns in unseen test data.

- In other words, the model should not be too complex in order to be robust and generalizable.

- If we look at it from the prespective of **Accuracy**, a too complex model will have a very high accuracy. So, to make our model more robust and generalizable, we will have to decrease variance which will lead to some bias. Addition of bias means that accuracy will decrease.
- In general, we have to find strike some balance between model accuracy and complexity. This can be achieved by Regularization techniques like Ridge Regression and Lasso.