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?

Ans: Optimal value of lambda for Ridge Regression = 10

Optimal value of lambda for Lasso = 0.001

```
[162]: ## Let us build the ridge regression model with double value of alpha i.e. 20
       ridge = Ridge(alpha=20)

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

[162]: Ridge(alpha=20)

[163]: ## Make predictions
       y_train_pred = ridge.predict(X_train)
       y_pred = ridge.predict(X_test)

[164]: ## Check metrics
       ridge_metrics = show_metrics(y_train, y_train_pred, y_test, y_pred)

       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

[165]: ## Now we will build the lasso model with double value of alpha i.e. 0.002
       lasso = Lasso(alpha=0.002)

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

[165]: Lasso(alpha=0.002)
```

```
[166]: ## Make predictions
       y_train_pred = lasso.predict(X_train)
       y_pred = lasso.predict(X_test)

[167]: ## Check metrics
       lasso_metrics = show_metrics(y_train, y_train_pred, y_test, y_pred)

       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

[168]: # Again 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)', 'RMSE (Train)', 'RMSE (Test)'],
                   'Ridge Regression' : ridge_metrics,
                   'Lasso Regression' : lasso_metrics
               }

       final_metric = pd.DataFrame(lr_table, columns = ['Metric', 'Ridge Regression', 'Lasso Regression'] )
       final_metric.set_index('Metric')
```

[168]:

Metric	Ridge Regression	Lasso Regression
R2 Score (Train)	0.934148	0.905235
R2 Score (Test)	0.927674	0.911638
RSS (Train)	9.374311	13.490241
RSS (Test)	2.821199	3.446734
MSE (Train)	0.008026	0.011550
MSE (Test)	0.009662	0.011804
RMSE (Train)	0.089588	0.107470
RMSE (Test)	0.098294	0.108646

Changes in Ridge Regression metrics:

R2 score of train set decreased from 0.94 to 0.93 R2 score of test set remained same at 0.93

Changes in Lasso metrics:¶

R2 score of train set decreased from 0.92 to 0.91 R2 score of test set decreased from 0.93 to 0.91

[171]:
```
## changes in coefficients after regularization
betas = pd.DataFrame(index=X.columns)
betas.rows = X.columns
```

[172]:
```
## Now fill in the values of betas, one column for ridge coefficients and one for lasso coefficients
betas['Ridge'] = ridge.coef_
betas['Lasso'] = lasso.coef_
```

[173]:
```
## View the betas/coefficients
betas
```

[173]:

	Ridge	Lasso
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

[174]:
```
## View the top 10 coefficients of Ridge regression in descending order
betas['Ridge'].sort_values(ascending=False)[:10]
```

[174]:
```
GrLivArea               0.080424
OverallQual_8           0.069483
OverallQual_9           0.064724
Neighborhood_Crawfor    0.064254
```

```
[174]: ## View the top 10 coefficients of Ridge regression in descending order
betas['Ridge'].sort_values(ascending=False)[:10]

[174]: GrLivArea              0.080424
OverallQual_8          0.069483
OverallQual_9          0.064724
Neighborhood_Crawfor   0.064254
Functional_Typ         0.062255
Exterior1st_BrkFace    0.057525
OverallCond_9          0.054218
TotalBsmtSF            0.048241
CentralAir_Y           0.047655
OverallCond_7          0.041946
Name: Ridge, dtype: float64

[175]: ## To interpret the ridge coefficients in terms of target, we have to take inverse log (i.e. e to the power) of betas
ridge_coeffs = np.exp(betas['Ridge'])
ridge_coeffs.sort_values(ascending=False)[:10]

[175]: 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, dtype: float64
```

```
[176]: ## View the top 10 coefficients of Lasso in descending order
betas['Lasso'].sort_values(ascending=False)[:10]

[176]: GrLivArea              0.108435
OverallQual_8          0.084391
OverallQual_9          0.077080
Functional_Typ         0.071746
Neighborhood_Crawfor   0.066749
TotalBsmtSF            0.046061
Exterior1st_BrkFace    0.044747
CentralAir_Y           0.040563
YearRemodAdd           0.036476
Condition1_Norm        0.032248
Name: Lasso, dtype: float64

[177]: ## To interpret the lasso coefficients in terms of target, we have to take inverse log (i.e. 10 to the power) of betas
lasso_coeffs = np.exp(betas['Lasso'])
lasso_coeffs.sort_values(ascending=False)[:10]

[177]: 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, dtype: float64
```

So, the most important predictor variables after we double the alpha values are:-
GrLivArea
OverallQual_8
OverallQual_9
Functional_Typ
Neighborhood_Crawfor
Exterior1st_BrkFace
TotalBsmtSF
CentralAir_Y

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 model we will choose to apply will depend on the use case.
If we have too many variables and one of our primary goal is feature selection, then we will use Lasso.
If we don't want to get too large coefficients and reduction of coefficient magnitude is one of our prime goals, then we will use Ridge Regression.

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: We will drop the top 5 features in Lasso model and build the model again.

Top 5 Lasso predictors were:

OverallQual_9,

GrLivArea,

OverallQual_8,

Neighborhood_Crawfor

Exterior1st_BrkFace

```
[178]: ## Create a list of top 5 lasso predictors that are to be removed
       top5 = ['OverallQual_9', 'GrLivArea', 'OverallQual_8', 'Neighborhood_Crawfor', 'Exterior1st_BrkFace']

[179]: ## drop them from train and test data
       X_train_dropped = X_train.drop(top5, axis=1)
       X_test_dropped = X_test.drop(top5, axis=1)

[180]: ## Now to create a Lasso model
       ## we will run a cross validation on a list of alphas to find the optimum value of alpha

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

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

[180]: GridSearchCV(cv=5, estimator=Lasso(), n_jobs=-1,
                    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',
```

[181]:
```
## View the optimal value of alpha
lassoCV.best_params_
```

[181]:
```
{'alpha': 0.001}
```

Thus, we get optimum value of alpha as 0.001. Now we will build a lasso regression model using this value.

[182]:
```
# Create a lasso instance with optimum value alpha=0.001
lasso = Lasso(alpha=0.001)
```

[183]:
```
# Fit the model on training data
lasso.fit(X_train_dropped, y_train)
```

[183]:
```
Lasso(alpha=0.001)
```

[184]:
```
## Make predictions
y_train_pred = lasso.predict(X_train_dropped)
y_pred = lasso.predict(X_test_dropped)
```

[185]:
```
## Check metrics
lasso_metrics = show_metrics(y_train, y_train_pred, y_test, y_pred)
```

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

[186]:
```
## find the top 5 predictors
```

[187]:
```
lr_table = {'Metric': ['R2 Score (Train)','R2 Score (Test)','RSS (Train)','RSS (Test)',
                       'MSE (Train)','MSE (Test)', 'RMSE (Train)', 'RMSE (Test)'],
            'Lasso Regression' : lasso_metrics
        }

final_metric = pd.DataFrame(lr_table, columns = ['Metric', 'Lasso Regression'] )
final_metric.set_index('Metric')
```

[187]:

Metric	Lasso Regression
R2 Score (Train)	0.910457
R2 Score (Test)	0.922473
RSS (Train)	12.746834
RSS (Test)	3.024069
MSE (Train)	0.010913
MSE (Test)	0.010356
RMSE (Train)	0.104467
RMSE (Test)	0.101766

```
[188]: ## changes in coefficients after regularization
```

```
[190]: betas = pd.DataFrame(index=X_train_dropped.columns)
betas.rows = X_train_dropped.columns
betas['Lasso'] = lasso.coef_
```

```
[191]: ## View the betas/coefficients
betas
```

[191]:

	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
1stFlrSF	0.073456

```
[192]: ## View the top 5 coefficients of Lasso in descending order
betas['Lasso'].sort_values(ascending=False)[:5]
```

```
[192]: 2ndFlrSF              0.098102
Functional_Typ        0.073546
1stFlrSF              0.073456
MSSubClass_70         0.061023
Neighborhood_Somerst  0.056671
Name: Lasso, dtype: float64
```

After dropping our top 5 lasso predictors, we get the following new top 5 predictors:-

```
2ndFlrSF
Functional_Typ
1stFlrSF
MSSubClass_70
Neighborhood_Somerst
```

Question: 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:

Ensuring a model is robust and generalizable is crucial for its real-world applicability. Here are steps to enhance robustness and generalizability and their implications on model accuracy:

Cross-validation: Use techniques like k-fold cross-validation to assess the model's performance on various subsets of data. This helps evaluate how well the model generalizes to unseen data. A consistent performance across different folds indicates robustness.

Feature engineering and selection: Carefully engineering features and selecting only relevant ones can improve a model's ability to generalize. Robust models are often less prone to overfitting on irrelevant features.

Regularization: Techniques like Lasso, Ridge, or ElasticNet regularization can prevent overfitting by penalizing large coefficients. These methods promote simpler models, reducing the risk of overfitting to training data and improving generalizability.

Hyperparameter tuning: Optimize model parameters using techniques like grid search or randomized search. Tuning hyperparameters ensures that the model is not overly fit to specific parameters and performs well across different settings.

Handling imbalanced data: If your dataset is imbalanced, apply techniques like oversampling, under sampling, or using specialized algorithms (e.g., SMOTE for synthetic data generation) to handle class imbalances. This ensures that the model learns from all classes adequately.

Implications for model accuracy:

Trade-off between accuracy and robustness: Increasing a model's robustness might slightly reduce its accuracy on the training set. However, this reduction is often beneficial as it prevents overfitting and leads to better performance on new, unseen data.

Better generalization: A robust and generalizable model might not achieve the highest accuracy on the training set, but it's more likely to perform well on new, real-world data. It minimizes the risk of making overly optimistic predictions based on the training data alone.

Consistency across different datasets: A robust model might display consistent accuracy across various datasets, indicating its reliability in different scenarios.

In summary, ensuring robustness and generalizability involves trade-offs with accuracy on the training set. However, it's essential for a model to perform well in real-world applications by accurately predicting unseen data.