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:

(Note: code available in notebook file and answer given based on code)

The optimal value of alpha for ridge and lasso regression

Ridge Alpha 10

lasso Alpha 0.001

```
Ridge(alpha=20)
```

```
# Lets calculate some metrics such as R2 score, RSS and RMSE
y_pred_train = ridge2.predict(X_train)
y_pred_test = ridge2.predict(X_test)

metric2 = []
r2_train_lr = r2_score(y_train, y_pred_train)
print("R2score(train) ",r2_train_lr)
metric2.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print("R2score(test) ", 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)

#Alpha 1
#R2score(train) 0.9557410228835224
#R2score(test)  0.8857204064324237
```

```
R2score(train)  0.9375802536320541
R2score(test)  0.9020292781816347
0.05482007596073344
```

Changes in Ridge Regression metrics:

Change Ridge Alpha 10 to 20

R2 score of train set decreased from 0.95 to 0.93

R2 score of test set increased from 0.88 to 0.90

R2score on training data has decreased but it has increased on testing data

```
Lasso
Lasso(alpha=0.002)
```

```
[182] # Lets calculate some metrics such as R2 score, RSS and RMSE
      y_pred_train = lasso20.predict(X_train)
      y_pred_test = lasso20.predict(X_test)

      metric3 = []
      r2_train_lr = r2_score(y_train, y_pred_train)
      print("R2score(train)  ",r2_train_lr)
      metric3.append(r2_train_lr)

      r2_test_lr = r2_score(y_test, y_pred_test)
      print("R2score(test)  ",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)

      #R2score at alpha-10
      #R2score(train)  0.8945785925985351
      #R2score(Test)  0.8633466965882921

      R2score(train)   0.8756994796385147
      R2score(test)   0.8453702876484173
```

Changes in Lasso metrics:

Change Ridge Alpha 0.001 to 0.002

R2 score of train set decreased from 0.89 to 0.87

R2 score of test set decreased from 0.86 to 0.84

R2score on training data and testing data decreased

And coefficent of these predictor has changed

[183]
```
#important predictor variables
betas = pd.DataFrame(index=X_train.columns)
betas.rows = X_train.columns
betas['Ridge2'] = ridge2.coef_
betas['Ridge'] = ridge.coef_
betas['Lasso'] = lasso.coef_
betas['Lasso20'] = lasso20.coef_
pd.set_option('display.max_rows', None)
betas.head(68)
```

	Ridge2	Ridge	Lasso	Lasso20
MSSubClass	-0.001473	-0.000386	-0.000269	-0.000000
LotArea	0.001990	0.002537	0.002202	0.001848
OverallQual	0.005305	0.004243	0.008172	0.008433
OverallCond	0.003265	0.002891	0.001776	0.000000
YearBuilt	0.003497	0.004391	0.005125	0.003557
YearRemodAdd	0.001899	0.001976	0.002908	0.003243
MasVnrArea	-0.000068	-0.000133	0.000000	0.000000
BsmtFinSF1	0.000540	-0.001765	0.002014	0.001881
BsmtFinSF2	-0.000343	-0.001802	0.000000	0.000000
BsmtUnfSF	-0.001277	-0.003242	-0.000000	-0.000000
TotalBsmtSF	0.005040	0.007458	0.004035	0.003863
1stFlrSF	0.000876	-0.001012	0.000000	0.000000
2ndFlrSF	0.002426	0.001709	0.000000	0.000000
LowQualFinSF	0.000000	0.000000	0.000000	0.000000
GrLivArea	0.006865	0.008771	0.008160	0.007551
BsmtFullBath	0.001042	0.000838	0.000529	0.000000

The most important predictor variables after we double the alpha values are:- OverallQual GrLivArea

YearBuilt
TotalBsmtSF
YearRemodAdd GarageArea
LotArea
Fireplaces
BsmtFinSF1
OverallCond RoofMatl_WdShngl
RoofMatl_Tar&Grv
RoofMatl_CompShg
RoofMatl_Membran
RoofMatl_WdShake
RoofMatl_Roll
Condition2_PosA
RoofMatl_Metal
SaleType_ConLD
BsmtFinType2_No_Basement

```
RoofMatl_Tar&Grv            1.040408
RoofMatl_CompShg            1.040264
RoofMatl_Membran            1.038659
RoofMatl_WdShake            1.038569
RoofMatl_Roll               1.037975
Condition2_PosA             1.032851
RoofMatl_Metal              1.031224
SaleType_ConLD              1.012630
BsmtFinType2_No_Basement    1.012390
Name: Ridge, dtype: float64
```

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

```
OverallQual     0.008172
GrLivArea       0.008160
YearBuilt       0.005125
TotalBsmtSF     0.004035
YearRemodAdd    0.002908
GarageArea      0.002300
LotArea         0.002202
Fireplaces      0.002185
BsmtFinSF1      0.002014
OverallCond     0.001776
Name: Lasso, dtype: float64
```

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

```
OverallQual     1.008205
GrLivArea       1.008194
YearBuilt       1.005139
TotalBsmtSF     1.004043
YearRemodAdd    1.002912
GarageArea      1.002303
LotArea         1.002204
Fireplaces      1.002188
BsmtFinSF1      1.002017
OverallCond     1.001778
Name: Lasso, dtype: float64
```

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:

(Note: code available in notebook file and answer given based on code)

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

Top 5 Lasso predictors were:

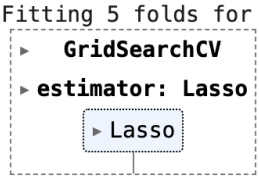
1. OverallQual
2. GrLivArea
3. YearBuilt
4. TotalBsmtSF
5. YearRemodAdd

```
[188] ## Create a list of top 5 lasso predictors that are to be removed
      top5 = ['OverallQual', 'GrLivArea', 'YearBuilt', 'TotalBsmtSF', 'YearRemodAdd']
```

```
[189] ## 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)
```

```
[190] ## 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
```




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

```
{'alpha': 0.0001}
```

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

1. 1stFlrSF
2. Foundation_PConc
3. ndFlrSF
4. BsmtFinSF1
5. GarageArea

Now, we will look at the top 5 features significant in predicting the value of a house according to the new lasso model

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

```
1stFlrSF            0.007313
Foundation_PConc    0.005725
2ndFlrSF            0.003973
BsmtFinSF1          0.003789
GarageArea          0.003728
Name: Lasso, dtype: float64
```

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

1. 1stFlrSF
2. Foundation_PConc
3. ndFlrSF
4. BsmtFinSF1
5. GarageArea

Double-click (or enter) to edit

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:

To ensure that a model is robust and generalizable, several steps can be taken:

- **Cross-Validation**: Use techniques like k-fold cross-validation to evaluate the model's performance on multiple subsets of the data. This helps in assessing how well the model performs on unseen data and reduces the risk of overfitting.

- ✳ **Train-Test Split**: Split the dataset into separate training and testing sets. Train the model on the training data and evaluate its performance on the unseen test data. This helps in simulating how the model would perform on new, unseen data.
- ✳ **Validation Set**: Apart from the training and testing sets, reserve a validation set to fine-tune hyperparameters and assess model performance during training.
- ✳ **Regularization**: Apply regularization techniques like L1 (Lasso) and L2 (Ridge) regularization to prevent overfitting by penalizing large coefficient values.
- ✳ **Feature Selection**: Carefully select relevant features and avoid overfitting by reducing the model's complexity.
- ✳ **Ensemble Methods**: Utilize ensemble methods like bagging, boosting, and stacking to combine multiple models and improve predictive performance.
- ✳ **Robust Evaluation Metrics**: Use robust evaluation metrics such as ROC-AUC, precision, recall, F1-score, and mean squared error (MSE) to assess model performance across different aspects.
- ✳ **External Validation**: Validate the model's performance on completely unseen data from different sources to ensure its generalizability to real-world scenarios.

Implications for Model Accuracy:

- Ensuring robustness and generalizability often involves sacrificing some degree of accuracy on the training data to achieve better performance on unseen data.
- A model that performs exceptionally well on the training data but poorly on unseen data may suffer from overfitting and lack generalizability.
- By employing techniques to enhance robustness and generalizability, the model's accuracy may stabilize or slightly decrease on the training data, but its performance on unseen data is expected to improve, leading to more reliable predictions in real-world applications.