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?

Optimal Value of alpha for Ridge Regression = **0.05**
Optimal Value of alpha for Lasso Regression = **0.0001**

Changes in the model If we double the alpha :

- For both the models (Ridge and Lasso), coefficients shrinks to zero.
- Because we are penalizing the model with double alpha value.
- As lasso is also used for feature selection, by doubling the alpha value i.e. higher alpha value, more coefficients are almost zero than previous.
- Now alpha is high, there is sparse in the model. So fewer coefficients contribute significantly to the model.
- As you can see in below screenshot, after doubling the alpha, most of the coefficients in Lasso regression are zero.

Ridge Regression

Before

```
alpha = 0.05
ridge = Ridge(alpha=alpha)
ridge.fit(X_train_rfe, y_train)
print(ridge.intercept_)
print(ridge.coef_)
```

0.0s

```
[-0.66258862]
[[-0.3970239   0.19778422  0.43395352  0.30762785  0.05526153  0.51241788
  -0.4034596   0.12293394  0.00729917 -0.05164828 -0.03370967 -0.04611102
  -0.04311988 -0.03793651 -0.01768138  0.02211282  0.10073766  0.12656567
  -0.01202445 -0.02313365 -0.00370761  0.00764915  0.00868248  0.01754891
   0.02334123  0.00085335  0.01384238  0.03126088 -0.01875069 -0.01251019
   0.00933024  0.00252769 -0.00586513 -0.03839833 -0.02010811  0.01384238
  -0.55690282  0.00729917  0.67943658  0.72340149  0.65671817  0.63776284
   0.68426499  0.64908776  0.74367758  0.01407886 -0.01138382  0.01407886
  -0.01138382 -0.0192094   -0.02975501 -0.01250656 -0.0179686  -0.01394423
  -0.01869604 -0.03616665 -0.00954152 -0.00977019  0.01837398  0.01837398]]
```

After

```
alpha = 0.05 * 2
ridge = Ridge(alpha=alpha)
ridge.fit(X_train_rfe, y_train)
print(ridge.intercept_)
print(ridge.coef_)
```

0.0s

```
[-0.48815205]
[[-0.37945188  0.18245297  0.27792935  0.27129165  0.04670797  0.49312526
  -0.23791609  0.1208963   0.00815491 -0.05578878 -0.03682151 -0.0484156
  -0.04406753 -0.03805894 -0.01634652  0.02574637  0.10757466  0.12338044
  -0.01001758 -0.02340562 -0.00405647  0.00788056  0.00876166  0.01684912
   0.02196957 -0.00077867  0.0124424   0.02941352 -0.01925991 -0.01015361
   0.00891772  0.00107282 -0.00464443 -0.0419087  -0.03770179  0.0124424
  -0.50050339  0.00815491  0.49919836  0.52549868  0.46404172  0.44362348
   0.50152735  0.46580253  0.56719153  0.01180779 -0.01262946  0.01180779
  -0.01262946 -0.01720259 -0.02599359 -0.00896817 -0.01568516 -0.0091195
  -0.01515674 -0.03328205 -0.00476468 -0.00656296  0.0177873   0.0177873 ]]
```

Lasso Regression

Before

```
alpha = 0.0001
lasso = Lasso(alpha=alpha)
lasso.fit(X_train_rfe, y_train)
print(lasso.intercept_)
print(lasso.coef_)
```

0.0s

```
[-0.10225449]
[-3.31281561e-01  9.42861323e-02  6.41351550e-02  1.65620591e-01
  1.81022912e-02  4.63445719e-01  0.00000000e+00  1.15830729e-01
  4.68505840e-05 -0.00000000e+00 -1.11895840e-03 -1.49421332e-02
 -7.44738663e-03 -0.00000000e+00  2.40494413e-02  7.09729058e-02
  1.57682228e-01  1.45399203e-01 -0.00000000e+00 -2.78523203e-02
 -1.06872283e-02  6.29830195e-04  0.00000000e+00  5.28958544e-03
  8.57070688e-03 -4.21646224e-03  0.00000000e+00  3.44235748e-02
 -4.47663320e-03  0.00000000e+00  0.00000000e+00 -4.57497526e-04
 -0.00000000e+00 -3.87663478e-02 -6.40027575e-02  0.00000000e+00
 -3.65518503e-01  7.45292258e-03  4.97884254e-02  3.08014729e-02
  0.00000000e+00  0.00000000e+00  4.21629682e-02  3.63135336e-04
  1.36749125e-01  0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00  0.00000000e+00 -1.21445780e-02 -0.00000000e+00
  0.00000000e+00  2.29997765e-03 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00  1.19211285e-02  3.26077970e-02  9.41101312e-19]
```

After

```
alpha = 0.0001 * 2
lasso = Lasso(alpha=alpha)
lasso.fit(X_train_rfe, y_train)
print(lasso.intercept_)
print(lasso.coef_)
```

0.0s

```
[-0.06020125]
[-0.30755925  0.02887588  0.07734467  0.12116802  0.00239919  0.4519205
  0.          0.11198351  0.         -0.         -0.         -0.01649383
 -0.          -0.          0.02422415  0.07333213  0.16098783  0.13904453
 -0.          -0.02133217 -0.00777093  0.00191413 -0.          0.00402556
  0.00591017 -0.          0.          0.03145756 -0.00373462  0.
  0.         -0.00196502  0.         -0.04420009 -0.04160697  0.
 -0.21930055  0.          0.01066314  0.          0.          0.
  0.         -0.          0.08856134  0.         -0.          0.
 -0.          0.         -0.01046306 -0.          0.          0.00103654
 -0.         -0.          0.          0.01297648  0.03198951  0.        ]
```

most important predictor variables after the change is implemented :

- For Ridge Regression : GrLivArea, RoofMatl_CompShg, RoofMatl_Membran, RoofMatl_Metal, RoofMatl_Roll

- For Lasso Regression : GrLivArea, LotFrontage, Condition2_PosN

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?

	Metric	Linear Regression	Ridge Regression	Lasso Regression
0	R2 Score (Train)	0.945105	0.938448	0.904765
1	R2 Score (Test)	0.896361	0.901828	0.902355
2	RSS (Train)	1.087869	1.219776	38004.954056
3	RSS (Test)	0.967887	0.916832	7555.192847
4	MSE (Train)	0.001065	0.001195	0.001848
5	MSE (Test)	0.002210	0.002093	0.002082

I choose **Lasso Model**. Because for ridge and lasso both models R2 score and MSE for train and test data is very good. Accuracy for both the models is almost same. But lasso has advantage of feature selection. Most of the feature coefficients in lasso model is almost zero. That's why my lasso model is simpler than redge regression model.

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?

- Previously top 5 features in lasso regression are = 'GrLivArea', 'BsmtFinSF1', 'OverallQual_9', 'Condition2_PosN', 'LotFrontage'
- If these features are not there in the incoming data, means after dropping these top 5 feature, **next top 5 features are** = 'LotArea', 'YearBuilt', 'OverallQual_3', 'OverallQual_4', 'OverallQual_2'

```
top_5_features = ['GrLivArea', 'BsmtFinSF1', 'OverallQual_9', 'Condition2_PosN', 'LotFrontage']

# Dropping these 5 features
X_train_rfe_after_top5drop = X_train_rfe.drop(top_5_features, axis=1)

alpha = 0.0001
lasso = Lasso(alpha=alpha)
lasso.fit(X_train_rfe_after_top5drop, y_train)

feature = X_train_rfe_after_top5drop.columns
lasso_coefficients = lasso.coef_

lasso_metric = pd.DataFrame({'Feature' : feature, 'lasso_coefficients' : lasso_coefficients})
lasso_metric_sorted = lasso_metric.sort_values(by='lasso_coefficients', ascending=False)
print(lasso_metric_sorted)
```

0.0s

	Feature	lasso_coefficients
0	LotArea	0.348603
1	YearBuilt	0.245764
39	RoofMatl_WdShngl	0.117527
13	OverallQual_10	0.069072
53	SaleType_New	0.041709
28	TotRmsAbvGrd_9	0.027767
48	GarageQual_TA	0.018220
20	OverallCond_8	0.010171

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?

Here are **different strategies** which makes sure that model is robust :

- Hyper-parameter tuning - Systematically tune the hyper parameters to find the best combination that generalize well on unseen data. Hyper- parameter tuning can be done using Grid Search or Random Search.

- Regularization - This prevent the over fitting of the model. It penalizes the coefficients and make it shrink to zero for high alpha value.

- Train-Test split - This makes sure that model is fitted on train data set and can perform well on unseen test data also.

- Cross validation - This uses multiple subsets of data and build the model. This ensures that model is consistent on different dataset.

- Data Preprocessing - Carefully select and clean / preprocess the features. Remove redundant features and handle missing values

Implications for the accuracy of the model:

- These strategies give high accuracy on train data and on test data as well.
- It makes model more robust.
- It makes the model more generalisable.
- Hence the model can perform well on real world unseen data.