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

Optimal value for Ridge model is 10

Optimal value for Lasso model is 100

If value of Alpha increased by double

Ridge : R2 value fall , when new_Alpha= 2 X old_Alpha=20

Accuracy fall marginally

```
[460] ridge = Ridge(alpha=10)

      ridge.fit(x_train, y_train)
      ridge.coef_
```

```
y_train_pred = ridge.predict(x_train)
y_test_pred =ridge.predict(x_test)

print("y_train  r2",metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
print("y_test  r2",metrics.r2_score(y_true=y_test, y_pred=y_test_pred))
print('RMSE :', np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
y_train  r2 0.9170975652094083
y_test  r2 0.8413862200430957
RMSE : 30281.38745546789
```

Test for double value of Alpha

```
ridge_2 = Ridge(alpha=20)

ridge_2.fit(x_train, y_train)
y_train_pred = ridge_2.predict(x_train)
y_test_pred =ridge_2.predict(x_test)

print("y_train  r2",metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
print("y_test  r2",metrics.r2_score(y_true=y_test, y_pred=y_test_pred))
print('RMSE :', np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
y_train  r2 0.9062775189583938
y_test  r2 0.8386032127012943
RMSE : 30545.888025834523
```

Top feature from ridge with Alpha*2

Variable
constant
OverallQual
FullBath
GarageCars
PoolQC
Fireplaces
Neighborhood_NoRidge
Neighborhood
KitchenQual

Lasso : R2 value fall , when new_Alpha= 2 X old_Alpha

Surprisingly : R2 for new alpha (2*alpha) is better

Alpha=100

```
lasso = Lasso(alpha=100)
lasso.fit(x_train, y_train)

# prediction on the test set(Using R2)
y_train_pred = lasso.predict(x_train)
print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
y_test_pred = lasso.predict(x_test)
print(metrics.r2_score(y_true=y_test, y_pred=y_test_pred))
print('RMSE :', np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
0.9377593633295717
0.6074370934865668
RMSE : 47638.70452225018
```

Test with double Alpha = 200

```
[614] lasso_2 = Lasso(alpha=200)
      lasso_2.fit(x_train, y_train)

      # prediction on the test set(Using R2)
      y_train_pred = lasso_2.predict(x_train)
      print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
      y_test_pred = lasso_2.predict(x_test)
      print(metrics.r2_score(y_true=y_test, y_pred=y_test_pred))
      print('RMSE :', np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
0.9221761295168154
0.721978025288129
RMSE : 40090.814022303246
```

Top feature from lasso with Alpha*2

Coeff
PoolQC
OverallQual
constant
FullBath
KitchenAbvGr
GarageCars
Neighborhood_NoRidge
OverallCond
Neighborhood_StoneBr

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?

Although Lasso reduce feature count , Lasso only perform better when feature count is less. But here feature count are higher than 100

Ridge model has higher accuracy in prediction with error estimation (R2), I will prefer Ridge model, due to high accuracy and high feature count

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?

Here are feature , after excluding top 5

Coeff
constant
Neighborhood_StoneBr
FullBath_2
Fireplaces_2
FullBath_1
Neighborhood_NoRidge
KitchenQual_Gd
RoofMatl_WdShngl
KitchenQual_TA
GarageCars_3

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?

Prioritizing robustness and generalizability can have implications for the accuracy of a machine learning model. While achieving high accuracy on the training data may initially appear desirable, it often leads to overfitting, whereby the model becomes too specialized to the training data and performs poorly on unseen data. By focusing on robustness and generalization, the model's accuracy on new, unseen data can be improved, even if it means sacrificing some accuracy on the training data.

There are many method to achive robustness , here are top 3 ways

Regularization: Apply regularization methods to mitigate overfitting and enhance generalization. These techniques introduce penalty terms in the model's training process, discouraging overly complex representations that might fit the training data excessively.

Model complexity: Choose a model that balances complexity and simplicity. While complex models may achieve high accuracy on the training data, they often suffer from overfitting and struggle to generalize to unseen data. Simpler models, on the other hand, tend to exhibit greater robustness and are less prone to overfitting.

Data preprocessing: Properly preprocess the data by addressing missing values, normalizing features, and handling outliers. Effective data preprocessing reduces biases, inconsistencies, or noise that could hinder the model's performance and impact its generalizability.

Maintaining balance between Bias and Variance will help to make simple ,generalize but robust model