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 :
The optimal value of alpha for **Ridge and Lasso Regression** got from the model as follows:

Regression	Alpha value

Ridge best hyperparameter:	`# Printing the best hyperparameter alpha for Ridge`
	
	`ridge_cv_alpha = ridge_cv.best_params_['alpha']`
	`print("Ridge best alpha value is:", ridge_cv.best_params_)`
	
	`Ridge best alpha value is: {'alpha': 10.0}`

Lasso best hyperparameter:	`# Printing the best hyperparameter alpha for Lasso`
	
	`lasso_alpha = lasso_cv.best_params_['alpha']`
	`print(lasso_cv.best_params_)`
	
	`{'alpha': 0.001}`

As per the question, what are the changes in the model once we double the value of alpha for both ridge and lasso, which is describe as below:

Ridge Regression:-

```
# Fitting Ridge model for alpha = 20 and printing coefficients
# which have been penalised

alpha = 20
ridge = Ridge(alpha=alpha)
ridge.fit(X_train, y_train)
```

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

```
# Lets calculate some metrics such as R2 score and RMSE

print("="*20+" Ridge"+"="*20)
y_pred_train = ridge.predict(X_train)
y_pred_test = ridge.predict(X_test)

r2_train_lr = r2_score(y_train, y_pred_train)
print("R2 score for tain:\t", r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print("R2 score for test:\t", r2_test_lr)

print("*"*50)
mse_train_lr = mean_squared_error(y_train, y_pred_train)
print("RMSE for tain:\t\t", mse_train_lr)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print("RMSE for test:\t\t", mse_test_lr)
print("="*50)
```

```
==================== Ridge====================
R2 score for tain:       0.9274704870547599
R2 score for test:       0.9176960285904247
**************************************************
RMSE for tain:           0.01059354049478184
RMSE for test:           0.01241712632033141
```

Lasso Regression:-

```
# Fitting Ridge model for alpha = 0.002 and printing coefficients
# which have been penalised

alpha = 0.002

lasso = Lasso(alpha=alpha)
lasso.fit(X_train, y_train)
```

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

```
# Lets calculate some metrics such as R2 score, RSS and RMSE

print("="*20+" Lasso= "+"="*20)
y_pred_train = lasso.predict(X_train)
y_pred_test = lasso.predict(X_test)

r2_train_lr = r2_score(y_train, y_pred_train)
print("R2 score for train: \t", r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print("R2 score for test: \t", r2_test_lr)

print("="*50)
mse_train_lr = mean_squared_error(y_train, y_pred_train)
print("RMSE score for train: \t", mse_train_lr)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print("RMSE score for test: \t", mse_test_lr)
print("="*50)
```

```
==================== Lasso= ====================
R2 score for train:      0.9118168386236352
R2 score for test:       0.9120823052879093
==================================================
RMSE score for train:    0.012879886449860819
RMSE score for test:     0.013264063718137356
```

Below are the comparison after changing the 'alpha' value:

Regression	Metrics	Before	After
Ridge	**R2 score**	tain: 0.931156 test: 0.917088	tain: 0.927470 test: 0.917696
	RMSE	tain: 0.010055 test: 0.012508	tain: 0.010593 tain: 0.012417
Lasso	**R2 score**	train: 0.921096 test: 0.916931	train: 0.911816 test: 0.912082
	RMSE	train: 0.011524 test: 0.012532	train: 0.012879 test: 0.013264

As we can see the difference between the metrics obtained after changing in the alpha value, the R2 score slightly decrease.

Below are the top 5 predictor variables after the change:

$$'GrLivArea, OverallQual, SaleType(New), Neighborhood(Crawfor), OverallCond'$$

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 :
Ridge and Lasso are the two regularisation techniques and it's help to managed the model complexity by essentially shrinking the model coefficient estimates towards 0. When we use regularisation, we add a penalty term to the model's cost function.

Here, the cost function would be **Cost = RSS + Penalty.**
Adding this penalty term in the cost function helps suppress or shrink the magnitude of the model coefficients towards 0. This discourages the creation of a more complex model, thereby preventing the risk of overfitting.

Ridge Regression

- Cost function for Ridge::
- $Cost = \sum_{i=1}^{n}(y_i + \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$
  - where λ = 0, 0.01, 0.001,...,10,100
  - But it should not be equal to 0 else it will behave simple linear regression.

Lasso Regression

- Cost function for Lasso::
- $Cost = \sum_{i=1}^{n}(y_i + \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$
  - where λ = 0, 0.01, 0.001,...,10,100

- Like Ridge regression, the Lasso shrinks the coefficients towards zero.
- Penalty in Lasso forces some of the coefficient estimates to be exactly equal to zero - the lasso performs variable selection.
- Model generated from the Lasso are generally easier to interpret than those produced by Ridge regression.
- Here, selecting a good value of λ for Lasso is crucial.
- In lasso also, as λ increase, the variance decreased and the bias increase.

Generally, Lasso should perform better in situations where only a few among all the predictors that are used to build our model have a significant influence on the response variable. So, feature selection, which removes the unrelated

variables, should help. But Ridge should do better when all the variables have almost the same influence on the response variable.
It is not the case that one of the techniques always performs better than the other – the choice would depend upon the data that is used for modelling.

So, Lasso, some of these coefficients become 0, thus resulting in model selection and, hence, easier interpretation, particularly when the number of coefficients is very large.

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 :

Below are the five most predator variables given by the model are:

Feature	Description
GrLivArea	Above grade (ground) living area square feet
Neighborhood(Crawfor)	Physical locations within Ames city limits (Crawford)
SaleType(New)	Type of sale (Home just constructed and sold)
OverallQual	Rates the overall material and finish of the house
SaleCondition(Normal)	Condition of sale (Normal Sale)

So, as per the question statement if create another model excluding these five most important predictor variables are below:

And the code snippet as follows:

'MSZoning_FV, MSZoning_RL, MSZoning_RM, MSZoning_RH, Neighborhood_StoneBr'

```
# Top 5 predictor given by previous model are:
top5 = ['GrLivArea','Neighborhood_Crawfor','SaleType_New','OverallQual','SaleCondition_Normal']

# Removing top 5 feature
X_train_new = X_train.drop(top5, axis= 1)
X_test_new = X_test.drop(top5, axis= 1)

```

```
# Creating lasso regression object
lasso = Lasso()

# lambda params
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 ]}

# folds
folds = 5

# cross validation
lasso_cv_new = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

lasso_cv_new.fit(X_train_new, y_train)
```

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

```
# best parameter
lasso_cv_new.best_params_
```

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

```
# Fitting Ridge model for alpha = 0.0001 and printing coefficients
# which have been penalised

alpha = 0.0001

lasso = Lasso(alpha=alpha)
lasso.fit(X_train_new, y_train)

y_pred_train = lasso.predict(X_train_new)
y_pred_test = lasso.predict(X_test_new)
```

```
# Lets calculate some metrics such as R2 score, RSS and RMSE

print("="*20+" Lasso= "+"="*20)

r2_train_lr = r2_score(y_train, y_pred_train)
print("R2 score for train: \t", r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print("R2 score for test: \t", r2_test_lr)

print("="*50)
mse_train_lr = mean_squared_error(y_train, y_pred_train)
print("RMSE score for train: \t", mse_train_lr)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print("RMSE score for test: \t", mse_test_lr)
print("="*50)
```

```
==================== Lasso= ====================
R2 score for train:      0.9278049926395151
R2 score for test:       0.9154934157688804
==================================================
RMSE score for train:    0.01054468316327721
RMSE score for test:     0.012749432540450386
==================================================
```

```
# Sort the coefficients in ascending order with top 5 feature

coeff_top5 = pd.DataFrame(index= X_train_new.columns)
coeff_top5 ['lasso_top5'] = lasso.coef_
coeff_top5.sort_values(ascending=False, by=['lasso_top5']).head()
```

	lasso_top5
MSZoning_FV	0.340913
MSZoning_RL	0.316646
MSZoning_RM	0.283028
MSZoning_RH	0.277828
Neighborhood_StoneBr	0.113168

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 :
The model should be as simple as possible and not be unnecessarily complex, though it may impact the accuracy of model but it will be more robust and generalisable. By using **Bias-Variance** trade-off we can understand it, Simpler the model more bias but less variance result in more generic. Its implication in terms of accuracy is that model will perform uniformly on both training and test data i.e., accuracy does not change much.

Simpler model having some advantage over Complex model, due to which we prefer this.
- Simpler models are usually more generic.
- Simpler models require fewer training samples for train the model as compare to complex one.
- Simpler models are more robust.
- Simpler models make more errors in the training set.

Bias:-
Bias is basically the correctness of the model; it occurs when our model fails to learn pattern from the data. Model with high bias pays little attention on training data. It always leads to high error on training and test data.
- High bias - when model fails to fit on the training data itself.

Variance:-
Variance is the inconsistency of model prediction for given data points. Model with high variance pays a lot of attention to training data and does not generalise on the unseen data. So as a result, such models perform very well on training data but has high error rates on test data. So, it is very important to keep balance between Bias and Variance to avoid model complexity and overfitting. We should try to model have low bias and low variance or we can say find an optimal point where total error is low.
- High variance - when model fails to fit on the testing data.

- Simple model - high bias & low variance
- We can manage overfitting is through Regularisation.
- Regularisation helps with managing model complexity by essentially shrinking the model coefficient estimates towards 0. This discourages the model from becoming too complex, thus avoiding the risk of overfitting.