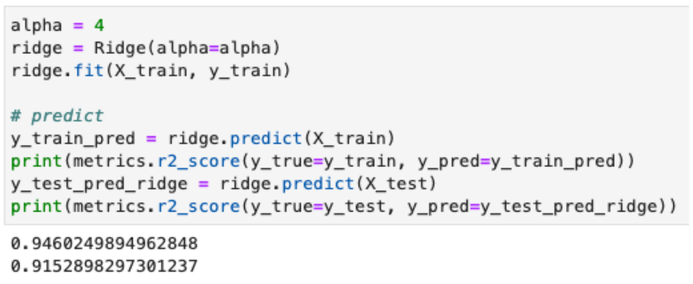
Question 1
What is the optimal value of alpha for ridge and lasso regression? What will be the changes in the model if you choose double the value of alpha for both ridge and lasso? What will be the most important predictor variables after the change is implemented?

Answer:
The optimal lambda value in case of Ridge and Lasso is as below:

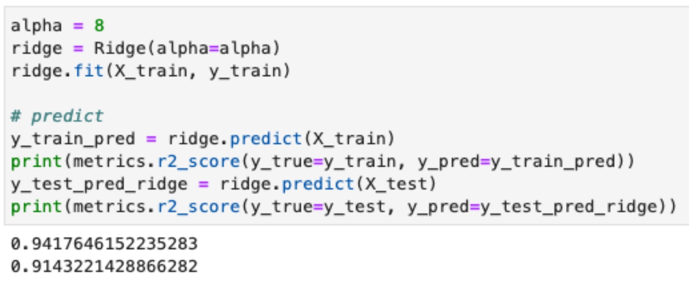
- Ridge - 4
- Lasso – 50

Ridge



```
alpha = 4
ridge = Ridge(alpha=alpha)
ridge.fit(X_train, y_train)

# predict
y_train_pred = ridge.predict(X_train)
print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
y_test_pred_ridge = ridge.predict(X_test)
print(metrics.r2_score(y_true=y_test, y_pred=y_test_pred_ridge))
```

```
0.9460249894962848
0.9152898297301237
```

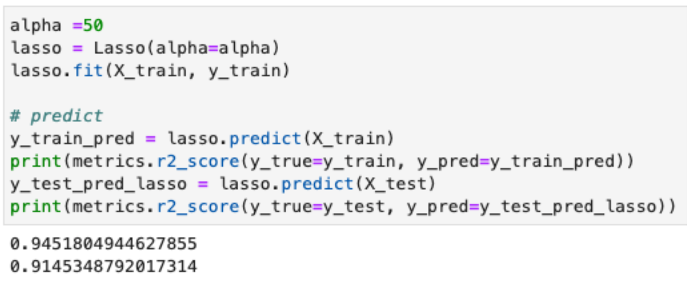


```
alpha = 8
ridge = Ridge(alpha=alpha)
ridge.fit(X_train, y_train)

# predict
y_train_pred = ridge.predict(X_train)
print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
y_test_pred_ridge = ridge.predict(X_test)
print(metrics.r2_score(y_true=y_test, y_pred=y_test_pred_ridge))
```

```
0.9417646152235283
0.9143221428866282
```

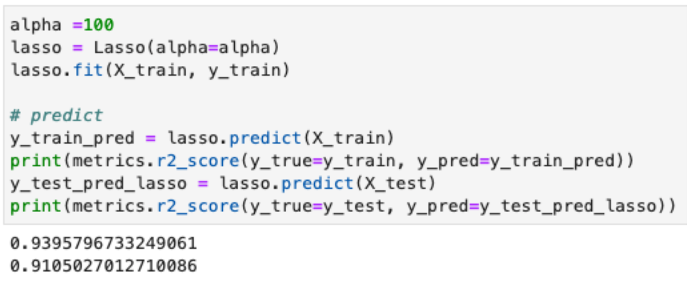
The predict values changes.

Lasso:



```
alpha =50
lasso = Lasso(alpha=alpha)
lasso.fit(X_train, y_train)

# predict
y_train_pred = lasso.predict(X_train)
print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
y_test_pred_lasso = lasso.predict(X_test)
print(metrics.r2_score(y_true=y_test, y_pred=y_test_pred_lasso))
```

```
0.9451804944627855
0.9145348792017314
```



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

# predict
y_train_pred = lasso.predict(X_train)
print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
y_test_pred_lasso = lasso.predict(X_test)
print(metrics.r2_score(y_true=y_test, y_pred=y_test_pred_lasso))
```

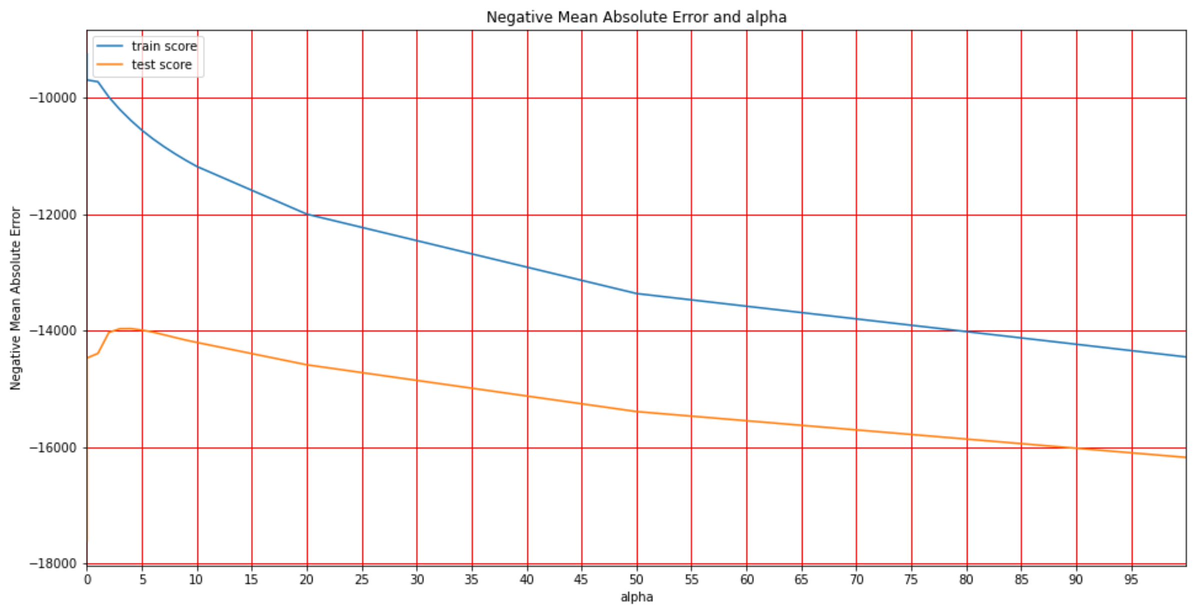
```
0.9395796733249061
0.9105027012710086
```

The predict values changes.

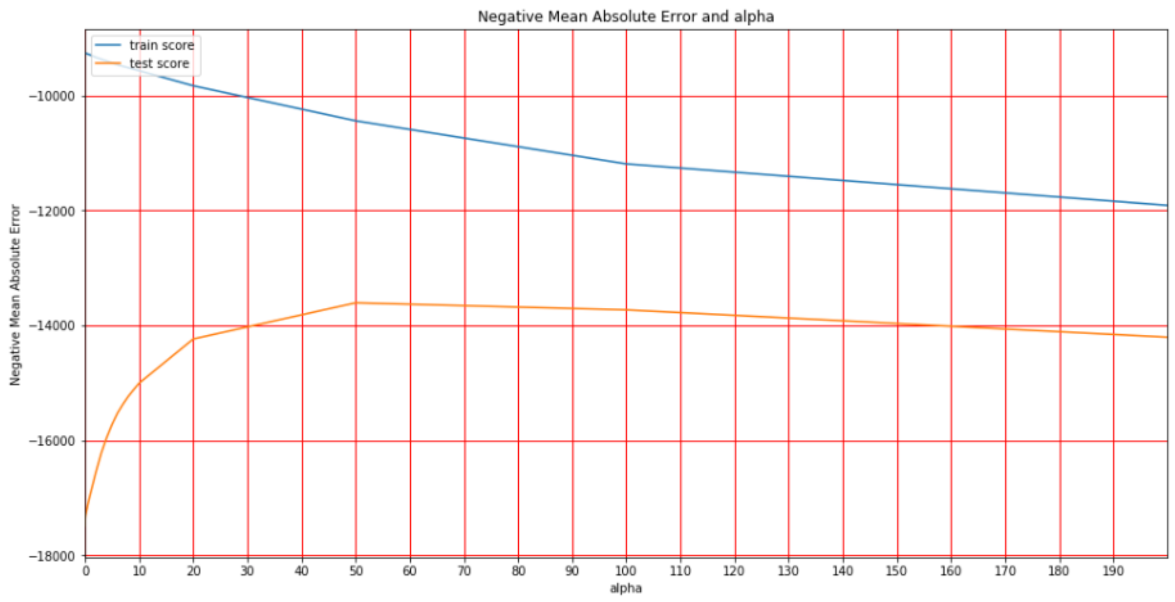
There is no change in the order of predictor variable but coefficient of predictor get change.

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:
We would decide that on the basis of plots and chose a value of alpha where we have good training as well as the test score.
Ridge regression plot:



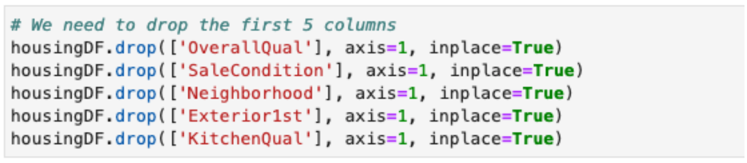
Based on the plot, we choose 4 as the value for lambda for Ridge Regression, since it has the best train as well as the test score.
Lasso Regression Plot:



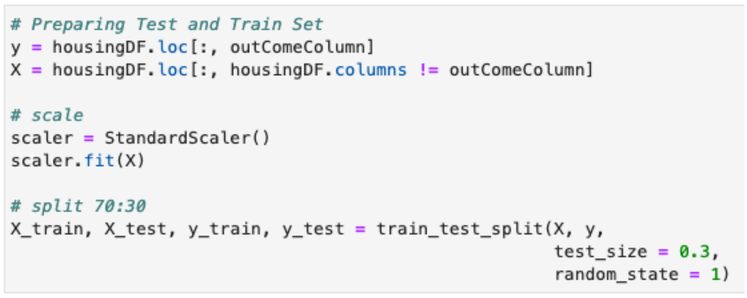
Based on the plot, we choose 50 as the value for lambda for Lasso Regression, since it has the best train as well as the test score.

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?

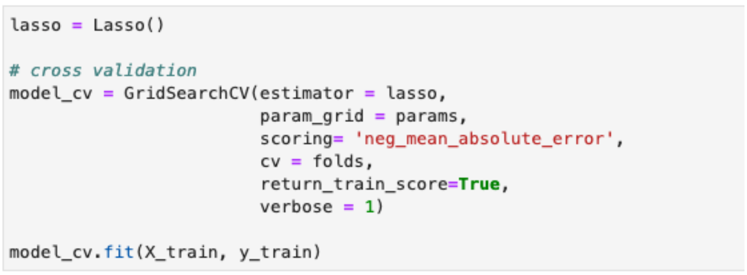
Answer:
We need to update the Test Data by deleting the columns and then do a Lasso.



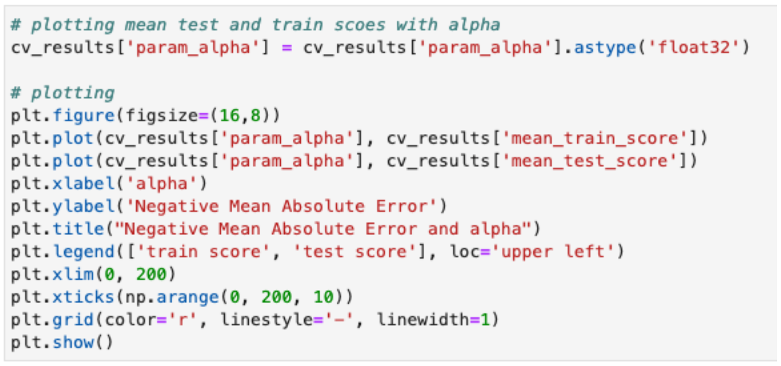
```
# We need to drop the first 5 columns
housingDF.drop(['OverallQual'], axis=1, inplace=True)
housingDF.drop(['SaleCondition'], axis=1, inplace=True)
housingDF.drop(['Neighborhood'], axis=1, inplace=True)
housingDF.drop(['Exterior1st'], axis=1, inplace=True)
housingDF.drop(['KitchenQual'], axis=1, inplace=True)
```



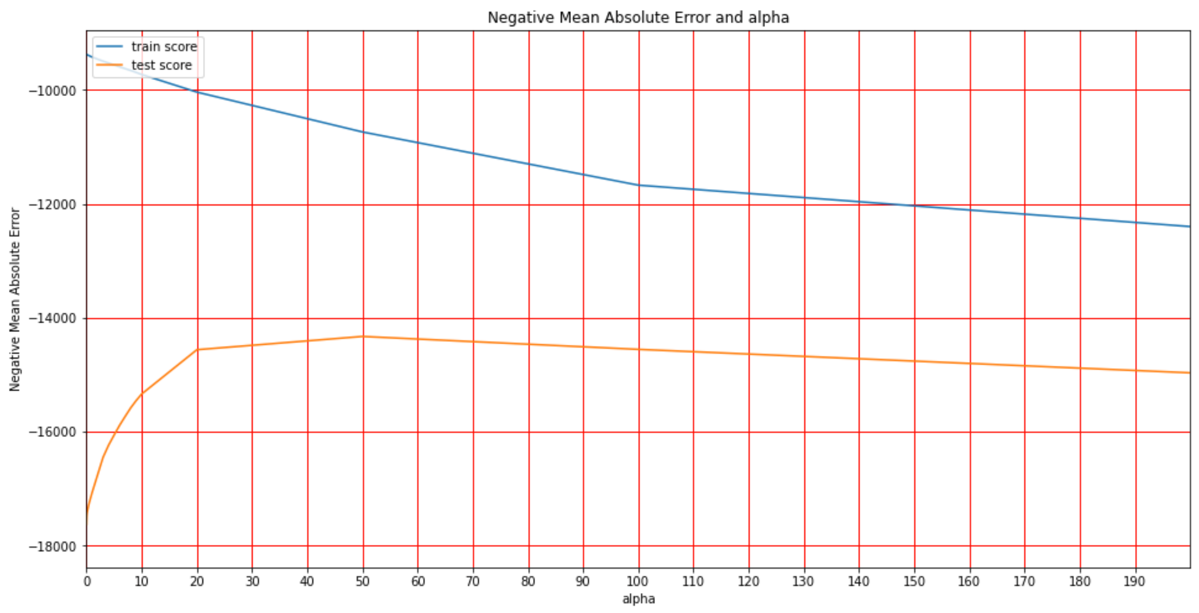
```
# Preparing Test and Train Set
y = housingDF.loc[:, outComeColumn]
X = housingDF.loc[:, housingDF.columns != outComeColumn]

# scale
scaler = StandardScaler()
scaler.fit(X)

# split 70:30
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.3,
                                                    random_state = 1)
```



```
lasso = Lasso()

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

model_cv.fit(X_train, y_train)
```



```
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('float32')

# plotting
plt.figure(figsize=(16,8))
plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('Negative Mean Absolute Error')
plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper left')
plt.xlim(0, 200)
plt.xticks(np.arange(0, 200, 10))
plt.grid(color='r', linestyle='-', linewidth=1)
plt.show()
```



```
alpha =50
lasso = Lasso(alpha=alpha)
lasso.fit(X_train, y_train)

# predict
y_train_pred = lasso.predict(X_train)
print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
y_test_pred_lasso = lasso.predict(X_test)
print(metrics.r2_score(y_true=y_test, y_pred=y_test_pred_lasso))
```

```
0.9413038218325367
0.9077423800355201
```

```
# lasso model parameters
model_parameters = list(lasso.coef_)
model_parameters.insert(0, lasso.intercept_)
model_parameters = [round(x, 3) for x in model_parameters]
cols = X.columns
cols = cols.insert(0, "constant")
lasso_selected_features = []
for k,v in sorted(list(zip(model_parameters,cols)), key=lambda x:abs(x[0]), reverse=True)[1:31]:
    lasso_selected_features.append(v)
sorted(list(zip(model_parameters,cols)), key=lambda x:abs(x[0]), reverse=True)
```

```
...
```

```
# Running RFE with the output number of the variable equal to 30
rfe = RFE(lm, 30)             # running RFE
rfe = rfe.fit(X_train, y_train)
```

```
# Listing out the top 30 columns that are selected in RFE (Automated selection of Predictors)
auto_selected_predictors = X_train.columns[rfe.support_]
auto_selected_predictors
```

```
# final model parameters
lasso_selected_features
```

```
['Exterior1st_BrkFace',
 'MSSubClass_160',
 'Neighborhood_OldTown',
 'BsmtExposure_Gd',
 'OverallCond_3',
 'MSSubClass_70',
 'BsmtQual_TA',
 'MasVnrType_Stone',
 'BsmtQual_Gd',
 'BsmtFinType1_GLQ']
```

Most important features that are affecting Sales pricing are:

- Exterior1st
- MSSubClass
- Neighborhood

- BsmtExposure
- OverallCond
- MSSubClass
- BsmtQual
- MasVnrType
- BsmtQual
- BsmtFinType1

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:
A model is considered to be robust if the model is stable, i.e. does not change drastically upon changing the training set. The model is considered generalisable if it does not overfits the training data, and works well with new data.

Per, Occam's Razor — suggested two models that show similar 'performance' in the finite training or test data, we should pick the one that makes limited on the test data due to following reasons-

- Simpler models are generally more 'generic' and are more extensively applicable
- Simpler models need smaller training samples for effective training than the more complex ones and hence are easier to train.
- Simpler models are more robust.
    - Complex models tend to change hectically with changes in the training data set
    - Simple models have low variance, complex models high variance
- Simpler models make more error in the training set. Complex models lead to overfitting — they work veritably well for the training data, fail miserably when applied to other test data

Thus to make the model more robust and generalizable, make the model simple but not simpler which won't be of any use.

Regularization can be used to make the model simpler. Regularization helps to strike the delicate balance between keeping the model simple and not making it too naive to be of any use. For retrogression, regularization involves adding a regularization term to the cost that adds up the absolute values or the places of the parameters of the model.

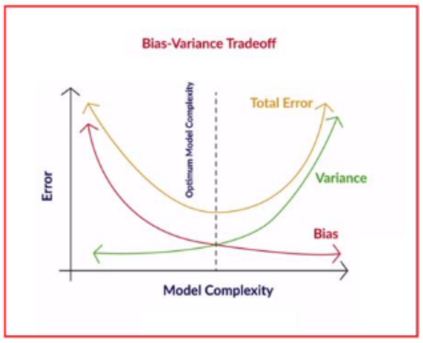
Also, Making a model simple leads to Bias-Variance Trade-off

- A complex model will need to change for every little change in the dataset and hence is very unstable and extremely sensitive to any changes in the training data.
- A simpler model that formed out some pattern followed by the data points given is doubtful to change widely even if further points are added or removed.

Bias quantifies how accurate is the model likely to be on test data. A complex model can do an accurate job prediction handed there's enough training data. Models that are too naïve, for e.g., one that gives same answer to all test inputs and makes no demarcation whatsoever has a very large bias as its anticipated error across all test inputs are very high.

Variance refers to the degree of changes in the model itself with respect to changes in the training data.

Therefore acuracy of the model can be maintained by keeping the balance between Bias and Variance as it minimizes the total error as shown in the graph below


Bias-Variance Tradeoff
Optimum Model Complexity
Total Error
Variance
Bias
Error
Model Complexity