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 1
As per the model below are the optimal values:

- Ridge -> 2.0
- Lasso -> 0.001

When we double the value of the regularization parameter alpha for both Ridge and Lasso regression, it increases the strength of regularization. Higher alpha values lead to stronger penalties on the coefficients, promoting sparsity and shrinking their values.
Ridge will shrink the values but will not make it to zero, however in case of Lasso more coefficient will become zero if we double the alpha.
To summarize:

- In ridge, all variables will still be persisted however their magnitude will be more heavily penalized.
- In Lasso, some variables may be set to exactly zero excluding them from the model

<u>Effect of doubling the alpha on Ridge model:</u>
Train and Test R2 score is improved. The delta between Train and Test R2Score is also reduced
Most important Predictor variables of Ridge:

5	OverallCond	0.32
13	GrLivArea	0.32
4	OverallQual	0.30
9	TotalBsmtSF	0.29
10	1stFlrSF	0.27

<u>Effect of doubling the alpha on Lasso model:</u>
Train and Test score is reduced.
Feature selection is now 51 instead of 63
Most important Predictor variables:

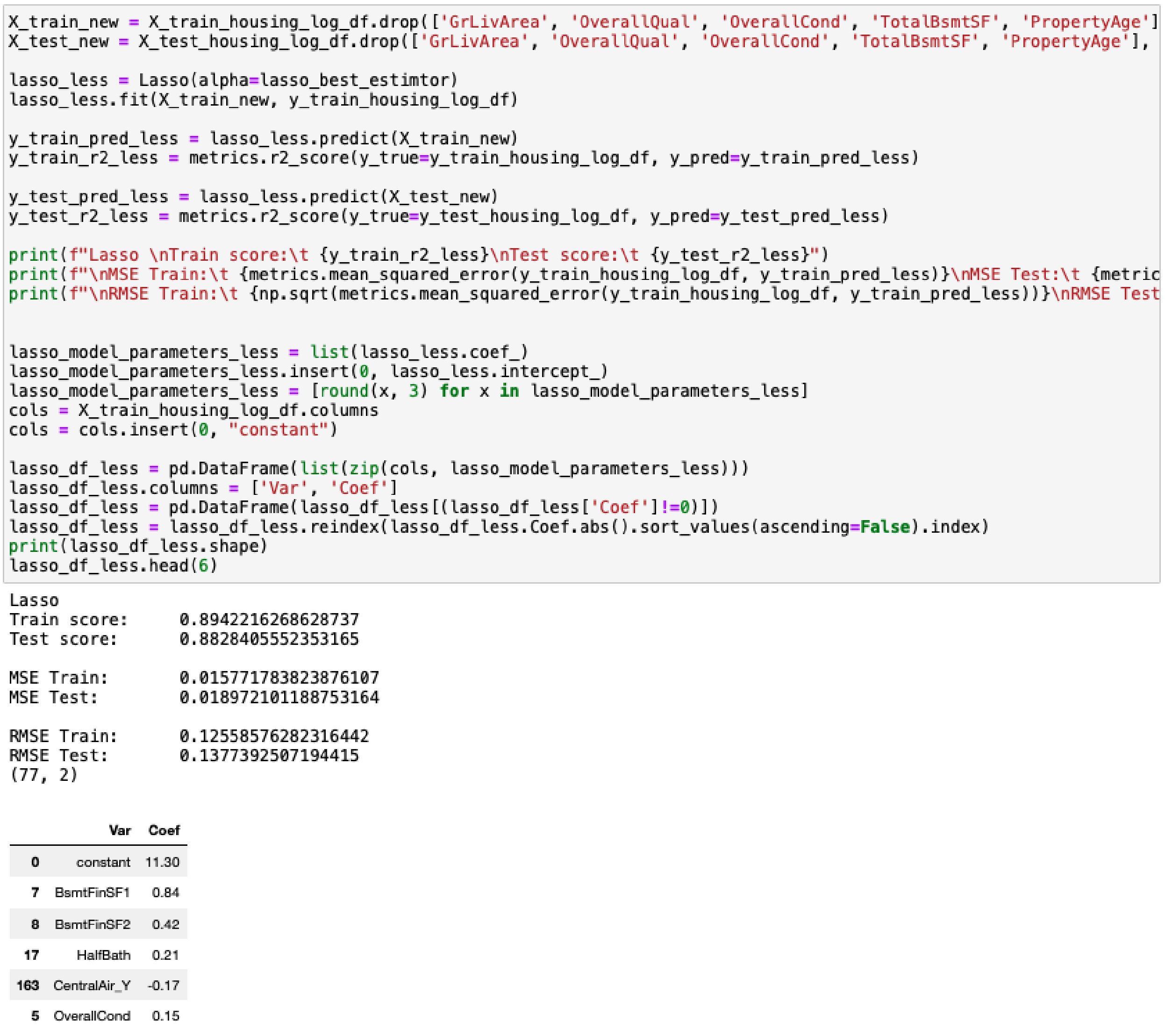
13	GrLivArea	0.78
4	OverallQual	0.52
5	OverallCond	0.29
9	TotalBsmtSF	0.27
21	GarageArea	0.17

Code snippet is added in the ipynb file to review

```
ridge_new_alpha = ride_best_estimator.alpha * 2


ridge_double = Ridge(alpha=ridge_new_alpha)
ridge_double.fit(X_train_housing_log_df, y_train_housing_log_df)

# predict
y_train_pred_double = ridge_double.predict(X_train_housing_log_df)
train_r2_double = metrics.r2_score(y_true=y_train_housing_log_df, y_pred=y_train_pred_double)

y_test_pred_double = ridge_double.predict(X_test_housing_log_df)
test_r2_double = metrics.r2_score(y_true=y_test_housing_log_df, y_pred=y_test_pred_double)

print(f"Ridge \nTrain score:\t {train_r2_double}\nTest score:\t {test_r2_double}")
print(f"\nMSE Train:\t {metrics.mean_squared_error(y_train_housing_log_df, y_train_pred_double)}\nMSE Test:\t {metr
print(f"\nRMSE Train:\t {np.sqrt(metrics.mean_squared_error(y_train_housing_log_df, y_train_pred_double))}\nRMSE Te


ridge_model_parameters_double = list(ridge_double.coef_)
ridge_model_parameters_double.insert(0, ridge_double.intercept_)
ridge_model_parameters_double = [round(x, 3) for x in ridge_model_parameters]
cols = X_train_housing_log_df.columns
cols = cols.insert(0, "constant")

ridge_df_double = pd.DataFrame(list(zip(cols, ridge_model_parameters_double)))
ridge_df_double.columns = ['Var', 'Coef']
ridge_df_double = ridge_df_double.reindex(ridge_df_double.Coef.abs().sort_values(ascending=False).index)
ridge_df_double.head(6)
```

```
Ridge
Train score:     0.9300874058564883
Test score:      0.9153088995985202

MSE Train:       0.010424118737092238
MSE Test:        0.013714371298284529

RMSE Train:      0.10209857362907789
RMSE Test:       0.11710837415951315
```

	Var	Coef
0	constant	10.90
5	OverallCond	0.32
13	GrLivArea	0.32
4	OverallQual	0.30
9	TotalBsmtSF	0.29
10	1stFlrSF	0.27



```python
X_train_new = X_train_housing_log_df.drop(['GrLivArea', 'OverallQual', 'OverallCond', 'TotalBsmtSF', 'PropertyAge']
X_test_new = X_test_housing_log_df.drop(['GrLivArea', 'OverallQual', 'OverallCond', 'TotalBsmtSF', 'PropertyAge'],

lasso_less = Lasso(alpha=lasso_best_estimtor)
lasso_less.fit(X_train_new, y_train_housing_log_df)

y_train_pred_less = lasso_less.predict(X_train_new)
y_train_r2_less = metrics.r2_score(y_true=y_train_housing_log_df, y_pred=y_train_pred_less)

y_test_pred_less = lasso_less.predict(X_test_new)
y_test_r2_less = metrics.r2_score(y_true=y_test_housing_log_df, y_pred=y_test_pred_less)

print(f"Lasso \nTrain score:\t {y_train_r2_less}\nTest score:\t {y_test_r2_less}")
print(f"\nMSE Train:\t {metrics.mean_squared_error(y_train_housing_log_df, y_train_pred_less)}\nMSE Test:\t {metric
print(f"\nRMSE Train:\t {np.sqrt(metrics.mean_squared_error(y_train_housing_log_df, y_train_pred_less))}\nRMSE Test


lasso_model_parameters_less = list(lasso_less.coef_)
lasso_model_parameters_less.insert(0, lasso_less.intercept_)
lasso_model_parameters_less = [round(x, 3) for x in lasso_model_parameters_less]
cols = X_train_housing_log_df.columns
cols = cols.insert(0, "constant")

lasso_df_less = pd.DataFrame(list(zip(cols, lasso_model_parameters_less)))
lasso_df_less.columns = ['Var', 'Coef']
lasso_df_less = pd.DataFrame(lasso_df_less[(lasso_df_less['Coef']!=0)])
lasso_df_less = lasso_df_less.reindex(lasso_df_less.Coef.abs().sort_values(ascending=False).index)
print(lasso_df_less.shape)
lasso_df_less.head(6)
```

```
Lasso
Train score:     0.8942216268628737
Test score:      0.8828405552353165

MSE Train:       0.015771783823876107
MSE Test:        0.018972101188753164

RMSE Train:      0.12558576282316442
RMSE Test:       0.1377392507194415
(77, 2)
```

	Var	Coef
0	constant	11.30
7	BsmtFinSF1	0.84
8	BsmtFinSF2	0.42
17	HalfBath	0.21
163	CentralAir_Y	-0.17
5	OverallCond	0.15

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

While both models seem to perform well, Ridge consistently outperforms Lasso across all metrics. The choice between Ridge and Lasso can also depend on other factors such as interpretability (Lasso's tendency for feature selection) and the specific goals of your modeling.

- For a more simple model in terms of feature selection, choose Lasso.
- For a balance between simplicity and allowing all features to contribute, choose Ridge.

I have selected lasso in the assignment as it will give us the capability of eliminating the features by setting them to zero and will generate a more simpler model.

However, it should be a pure business decision based on the requirements - is it simplicity or contribution of all features and more accurate 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?

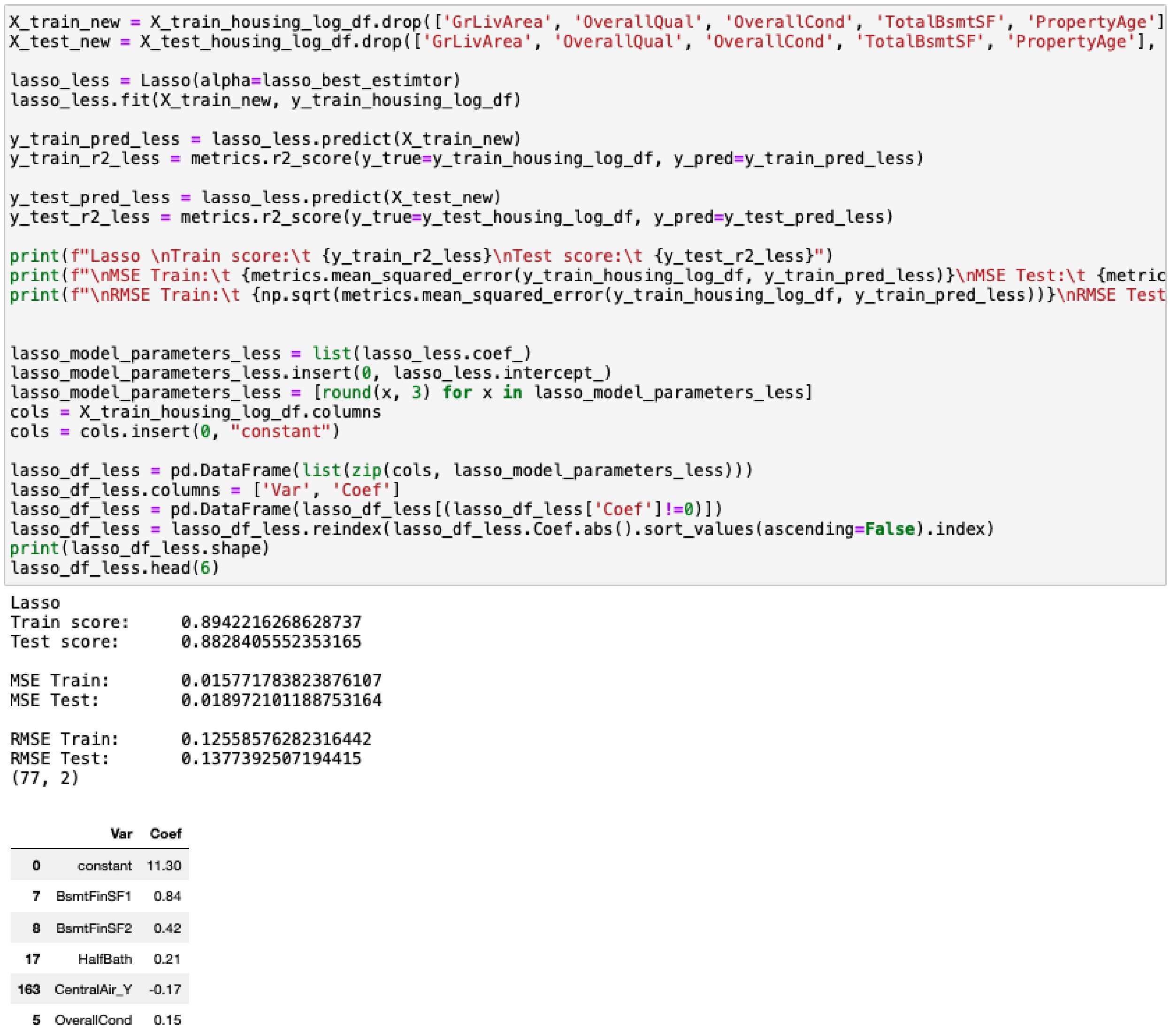
The five most important predictor were:

x1	GrLivArea	0.81
x2	OverallQual	0.48
x3	OverallCond	0.36
x4	TotalBsmtSF	0.33
x5	PropertyAge	-0.19

Since they are now excluded from the model, the new most important params are:

7	BsmtFinSF1	0.84
8	BsmtFinSF2	0.42
17	HalfBath	0.21
163	CentralAir_Y	-0.17
5	OverallCond	0.15

Code snippet is added in the code file for review



```
X_train_new = X_train_housing_log_df.drop(['GrLivArea', 'OverallQual', 'OverallCond', 'TotalBsmtSF', 'PropertyAge']
X_test_new = X_test_housing_log_df.drop(['GrLivArea', 'OverallQual', 'OverallCond', 'TotalBsmtSF', 'PropertyAge'],

lasso_less = Lasso(alpha=lasso_best_estimtor)
lasso_less.fit(X_train_new, y_train_housing_log_df)

y_train_pred_less = lasso_less.predict(X_train_new)
y_train_r2_less = metrics.r2_score(y_true=y_train_housing_log_df, y_pred=y_train_pred_less)

y_test_pred_less = lasso_less.predict(X_test_new)
y_test_r2_less = metrics.r2_score(y_true=y_test_housing_log_df, y_pred=y_test_pred_less)

print(f"Lasso \nTrain score:\t {y_train_r2_less}\nTest score:\t {y_test_r2_less}")
print(f"\nMSE Train:\t {metrics.mean_squared_error(y_train_housing_log_df, y_train_pred_less)}\nMSE Test:\t {metric
print(f"\nRMSE Train:\t {np.sqrt(metrics.mean_squared_error(y_train_housing_log_df, y_train_pred_less))}\nRMSE Test


lasso_model_parameters_less = list(lasso_less.coef_)
lasso_model_parameters_less.insert(0, lasso_less.intercept_)
lasso_model_parameters_less = [round(x, 3) for x in lasso_model_parameters_less]
cols = X_train_housing_log_df.columns
cols = cols.insert(0, "constant")

lasso_df_less = pd.DataFrame(list(zip(cols, lasso_model_parameters_less)))
lasso_df_less.columns = ['Var', 'Coef']
lasso_df_less = pd.DataFrame(lasso_df_less[(lasso_df_less['Coef']!=0)])
lasso_df_less = lasso_df_less.reindex(lasso_df_less.Coef.abs().sort_values(ascending=False).index)
print(lasso_df_less.shape)
lasso_df_less.head(6)
```

```
Lasso
Train score:     0.8942216268628737
Test score:      0.8828405552353165

MSE Train:       0.015771783823876107
MSE Test:        0.018972101188753164

RMSE Train:      0.12558576282316442
RMSE Test:       0.1377392507194415
(77, 2)
```

	Var	Coef
0	constant	11.30
7	BsmtFinSF1	0.84
8	BsmtFinSF2	0.42
17	HalfBath	0.21
163	CentralAir_Y	-0.17
5	OverallCond	0.15

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

Simplicity is a key factor in designing a robust and generalizable model, even at the expense of decreased accuracy. This principle aligns with the Bias-Variance trade-off, where a simpler model tends to have higher bias but lower variance, making it more generalizable. The implication in terms of accuracy is that a robust and generalizable model will exhibit consistent performance on both training and test data, with minimal changes in accuracy between the two datasets.

Bias, representing the error in a model when it struggles to learn from the data, is high when the model is too simplistic and unable to capture intricate details. This leads to poor performance on both training and testing data.

Variance, on the other hand, signifies the error in a model when it over learns from the data. High variance results in excellent performance on the training data but poor performance on testing data, as the model may have memorized the training data specifics.

Maintaining a balance between **Bias** and **Variance** is crucial to prevent overfitting (excessive focus on the training data specifics) and underfitting (inability to capture essential patterns). Striking this balance contributes to the development of a model that generalizes well to new, unseen data while avoiding the pitfalls of being too simplistic or overly complex.

Also, ensuring model robustness and generalizability involves key strategies. Employing k-fold cross-validation provides a reliable estimate of generalization, while splitting the dataset into training and test sets evaluates real-world performance. Careful feature selection and regularization, such as Ridge or Lasso regression, enhance a model's ability to generalize.

Strategies like cross-validation, regularization, and diverse evaluation contribute to creating models that are accurate, robust, and applicable to real-world scenarios. Balancing complexity, feature relevance, and diverse data evaluation ensures the model's effectiveness in handling new and varied datasets