Advanced Linear Regression Assignment Part-2

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 to double the value of alpha for both ridge and lasso? What will be the most important predictor variables after the change is implemented?

Answer:

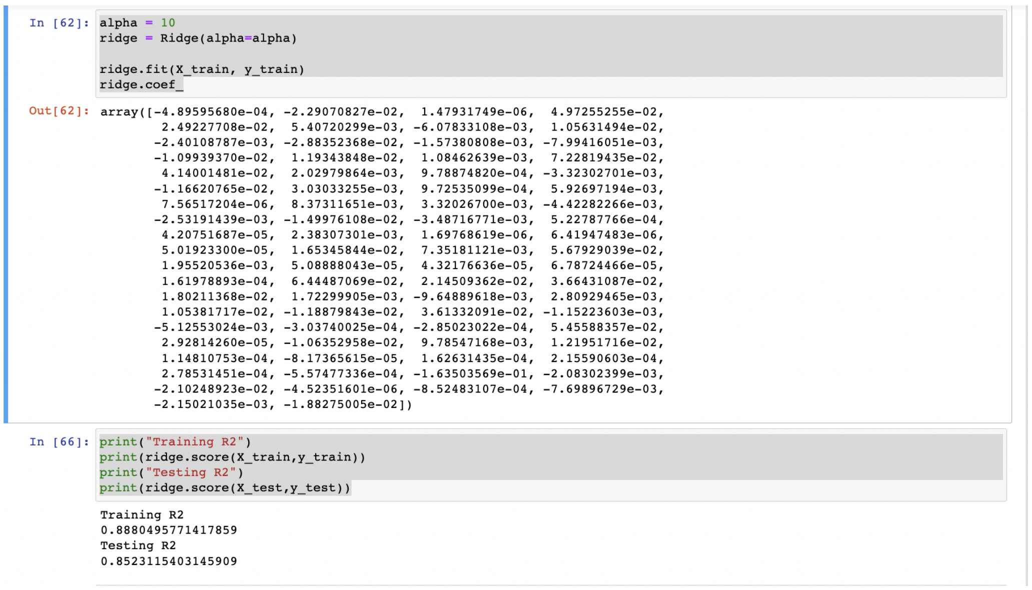
Based on the analysis carried out, it was observed that the optimal value of alpha for Lasso regression was 0.002 and for Ridge regression, the optimal value was 10.

If these values of optimal values are doubled, then the following were the observations for both Ridge and Lasso.

For Ridge Regression:

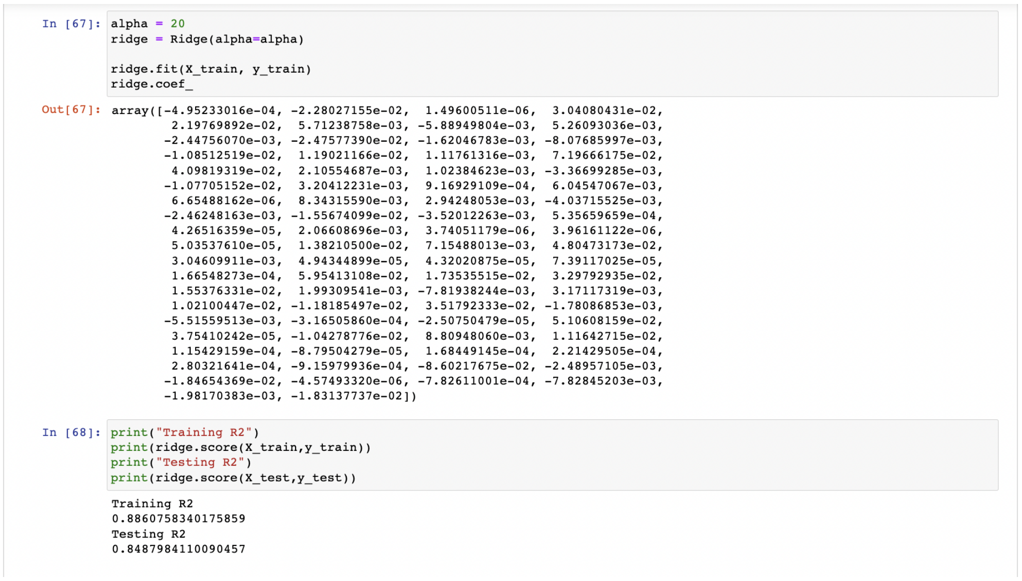
The training and test R2 score value had slightly reduced, but not much though.

For Alpha=10 (the optimal value)



```
In [62]: alpha = 10
         ridge = Ridge(alpha=alpha)

         ridge.fit(X_train, y_train)
         ridge.coef_

Out[62]: array([-4.89595680e-04, -2.29070827e-02,  1.47931749e-06,  4.97255255e-02,
                 2.49227708e-02,  5.40720299e-03, -6.07833108e-03,  1.05631494e-02,
                -2.40108787e-03, -2.88352368e-02, -1.57380808e-03, -7.99416051e-03,
                -1.09939370e-02,  1.19343848e-02,  1.08462639e-03,  7.22819435e-02,
                 4.14001481e-02,  2.02979864e-03,  9.78874820e-04, -3.32302701e-03,
                -1.16620765e-02,  3.03033255e-03,  9.72535099e-04,  5.92697194e-03,
                 7.56517204e-06,  8.37311651e-03,  3.32026700e-03, -4.42282266e-03,
                -2.53191439e-03, -1.49976108e-02, -3.48716771e-03,  5.22787766e-04,
                 4.20751687e-05,  2.38307301e-03,  1.69768619e-06,  6.41947483e-06,
                 5.01923300e-05,  1.65345844e-02,  7.35181121e-03,  5.67929039e-02,
                 1.95520536e-03,  5.08888043e-05,  4.32176636e-05,  6.78724466e-05,
                 1.61978893e-04,  6.44487069e-02,  2.14509362e-02,  3.66431087e-02,
                 1.80211368e-02,  1.72299905e-03, -9.64889618e-03,  2.80929465e-03,
                 1.05381717e-02, -1.18879843e-02,  3.61332091e-02, -1.15223603e-03,
                -5.12553024e-03, -3.03740025e-04, -2.85023022e-04,  5.45588357e-02,
                 2.92814260e-05, -1.06352958e-02,  9.78547168e-03,  1.21951716e-02,
                 1.14810753e-04, -8.17365615e-05,  1.62631435e-04,  2.15590603e-04,
                 2.78531451e-04, -5.57477336e-04, -1.63503569e-01, -2.08302399e-03,
                -2.10248923e-02, -4.52351601e-06, -8.52483107e-04, -7.69896729e-03,
                -2.15021035e-03, -1.88275005e-02])

In [66]: print("Training R2")
         print(ridge.score(X_train,y_train))
         print("Testing R2")
         print(ridge.score(X_test,y_test))

         Training R2
         0.8880495771417859
         Testing R2
         0.8523115403145909
```

For Alpha=20



```
In [67]: alpha = 20
         ridge = Ridge(alpha=alpha)

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

```
Out[67]: array([-4.95233016e-04, -2.28027155e-02,  1.49600511e-06,  3.04080431e-02,
                 2.19769892e-02,  5.71238758e-03, -5.88949804e-03,  5.26093036e-03,
                -2.44756070e-03, -2.47577390e-02, -1.62046783e-03, -8.07685997e-03,
                -1.08512519e-02,  1.19021166e-02,  1.11761316e-03,  7.19666175e-02,
                 4.09819319e-02,  2.10554687e-03,  1.02384623e-03, -3.36699285e-03,
                -1.07705152e-02,  3.20412231e-03,  9.16929109e-04,  6.04547067e-03,
                 6.65488162e-06,  8.34315590e-03,  2.94248053e-03, -4.03715525e-03,
                -2.46248163e-03, -1.55674099e-02, -3.52012263e-03,  5.35659659e-04,
                 4.26516359e-05,  2.06608696e-03,  3.74051179e-06,  3.96161122e-06,
                 5.03537610e-05,  1.38210500e-02,  7.15488013e-03,  4.80473173e-02,
                 3.04609911e-03,  4.94344899e-05,  4.32020875e-05,  7.39117025e-05,
                 1.66548273e-04,  5.95413108e-02,  1.73535515e-02,  3.29792935e-02,
                 1.55376331e-02,  1.99309541e-03, -7.81938244e-03,  3.17117319e-03,
                 1.02100447e-02, -1.18185497e-02,  3.51792333e-02, -1.78086853e-03,
                -5.51559513e-03, -3.16505860e-04, -2.50750479e-05,  5.10608159e-02,
                 3.75410242e-05, -1.04278776e-02,  8.80948060e-03,  1.11642715e-02,
                 1.15429159e-04, -8.79504279e-05,  1.68449145e-04,  2.21429505e-04,
                 2.80321641e-04, -9.15979936e-04, -8.60217675e-02, -2.48957105e-03,
                -1.84654369e-02, -4.57493320e-06, -7.82611001e-04, -7.82845203e-03,
                -1.98170383e-03, -1.83137737e-02])
```

```
In [68]: print("Training R2")
         print(ridge.score(X_train,y_train))
         print("Testing R2")
         print(ridge.score(X_test,y_test))
```

```
Training R2
0.8860758340175859
Testing R2
0.8487984110090457
```

However, it was observed that the important predictor variables had not changed even after doubling the value of alpha for ridge regression.

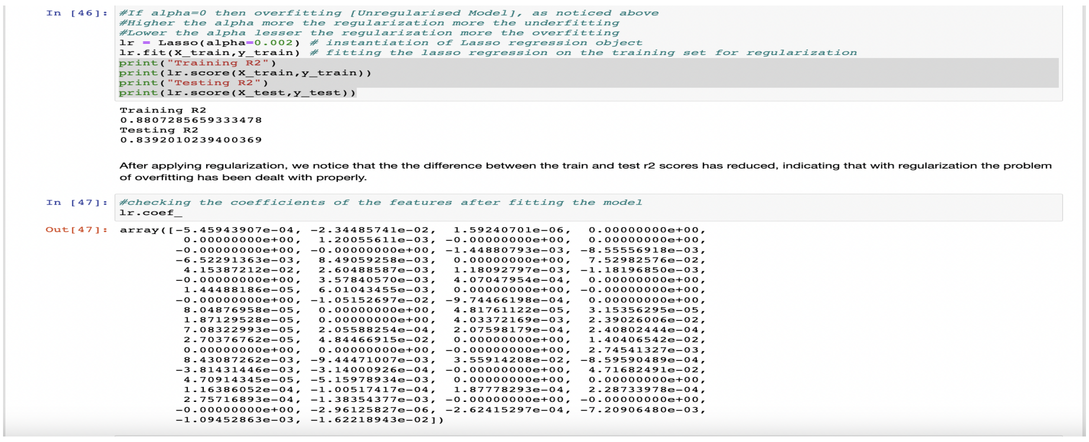
The important predictor variables were:

- Overall Quality of the material and finish of the house
- Basement Full Bathrooms
- Size of Garage in Car capacity
- Overall Condition of the house
- Central Air conditioning
- the type of road access to the street

For Lasso Regression:

The training and test R2 score value had largely remained the same and not much deviation was observed.

For Alpha=0.002 (the optimal value)



```
In [46]: #If alpha=0 then overfitting [Unregularised Model], as noticed above
         #Higher the alpha more the regularization more the underfitting
         #Lower the alpha lesser the regularization more the overfitting
         lr = Lasso(alpha=0.002) # instantiation of Lasso regression object
         lr.fit(X_train,y_train) # fitting the lasso regression on the training set for regularization
         print("Training R2")
         print(lr.score(X_train,y_train))
         print("Testing R2")
         print(lr.score(X_test,y_test))
```

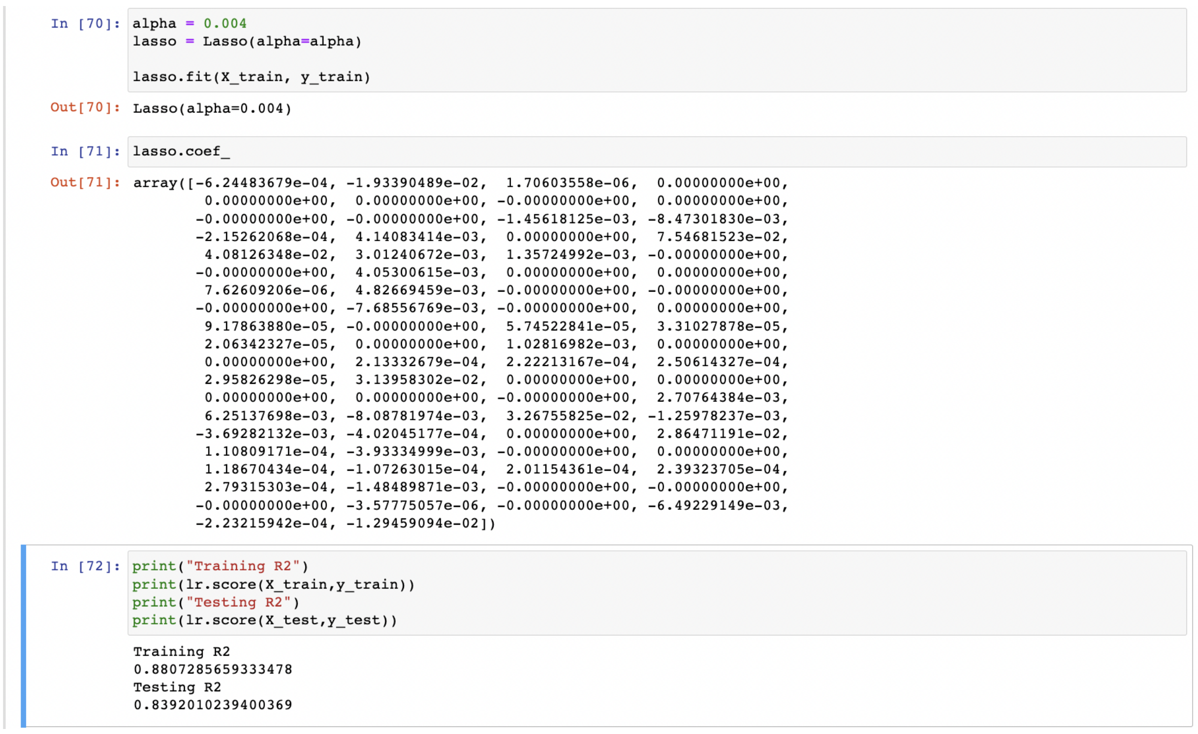
```
Training R2
0.8807285659333478
Testing R2
0.8392010239400369
```

After applying regularization, we notice that the the difference between the train and test r2 scores has reduced, indicating that with regularization the problem of overfitting has been dealt with properly.

```
In [47]: #checking the coefficients of the features after fitting the model
         lr.coef_
```

```
Out[47]: array([-5.45943907e-04, -2.34485741e-02,  1.59240701e-06,  0.00000000e+00,
                 0.00000000e+00,  1.20055611e-03, -0.00000000e+00,  0.00000000e+00,
                -0.00000000e+00, -0.00000000e+00, -1.44880793e-03, -8.55556918e-03,
                -6.52291363e-03,  8.49059258e-03,  0.00000000e+00,  7.52982576e-02,
                 4.15387212e-02,  2.60488587e-03,  1.18092797e-03, -1.18196850e-03,
                -0.00000000e+00,  3.57840570e-03,  4.07047954e-04,  0.00000000e+00,
                 1.44488186e-05,  6.01043455e-03,  0.00000000e+00, -0.00000000e+00,
                -0.00000000e+00, -1.05152697e-02, -9.74466198e-04,  0.00000000e+00,
                 8.04876958e-05,  0.00000000e+00,  4.81761122e-05,  3.15356295e-05,
                 1.87129528e-05,  0.00000000e+00,  4.03372169e-03,  2.39026006e-02,
                 7.08322993e-05,  2.05588254e-04,  2.07598179e-04,  2.40802444e-04,
                 2.70376762e-05,  4.84466915e-02,  0.00000000e+00,  1.40406542e-02,
                 0.00000000e+00,  0.00000000e+00, -0.00000000e+00,  2.74541327e-03,
                 8.43087262e-03, -9.44471007e-03,  3.55914208e-02, -8.59590489e-04,
                -3.81431446e-03, -3.14000926e-04, -0.00000000e+00,  4.71682491e-02,
                 4.70914345e-05, -5.15978934e-03,  0.00000000e+00,  0.00000000e+00,
                 1.16386052e-04, -1.00517417e-04,  1.87778293e-04,  2.28733978e-04,
                 2.75716893e-04, -1.38354377e-03, -0.00000000e+00, -0.00000000e+00,
                -0.00000000e+00, -2.96125827e-06, -2.62415297e-04, -7.20906480e-03,
                -1.09452863e-03, -1.62218943e-02])
```

For Alpha=0.004



```
In [70]: alpha = 0.004
         lasso = Lasso(alpha=alpha)

         lasso.fit(X_train, y_train)

Out[70]: Lasso(alpha=0.004)

In [71]: lasso.coef_

Out[71]: array([-6.24483679e-04, -1.93390489e-02,  1.70603558e-06,  0.00000000e+00,
                 0.00000000e+00,  0.00000000e+00, -0.00000000e+00,  0.00000000e+00,
                -0.00000000e+00, -0.00000000e+00, -1.45618125e-03, -8.47301830e-03,
                -2.15262068e-04,  4.14083414e-03,  0.00000000e+00,  7.54681523e-02,
                 4.08126348e-02,  3.01240672e-03,  1.35724992e-03, -0.00000000e+00,
                -0.00000000e+00,  4.05300615e-03,  0.00000000e+00,  0.00000000e+00,
                 7.62609206e-06,  4.82669459e-03, -0.00000000e+00, -0.00000000e+00,
                -0.00000000e+00, -7.68556769e-03, -0.00000000e+00,  0.00000000e+00,
                 9.17863880e-05, -0.00000000e+00,  5.74522841e-05,  3.31027878e-05,
                 2.06342327e-05,  0.00000000e+00,  1.02816982e-03,  0.00000000e+00,
                 0.00000000e+00,  2.13332679e-04,  2.22213167e-04,  2.50614327e-04,
                 2.95826298e-05,  3.13958302e-02,  0.00000000e+00,  0.00000000e+00,
                 0.00000000e+00,  0.00000000e+00, -0.00000000e+00,  2.70764384e-03,
                 6.25137698e-03, -8.08781974e-03,  3.26755825e-02, -1.25978237e-03,
                -3.69282132e-03, -4.02045177e-04,  0.00000000e+00,  2.86471191e-02,
                 1.10809171e-04, -3.93334999e-03, -0.00000000e+00,  0.00000000e+00,
                 1.18670434e-04, -1.07263015e-04,  2.01154361e-04,  2.39323705e-04,
                 2.79315303e-04, -1.48489871e-03, -0.00000000e+00, -0.00000000e+00,
                -0.00000000e+00, -3.57775057e-06, -0.00000000e+00, -6.49229149e-03,
                -2.23215942e-04, -1.29459094e-02])

In [72]: print("Training R2")
         print(lr.score(X_train,y_train))
         print("Testing R2")
         print(lr.score(X_test,y_test))

         Training R2
         0.8807285659333478
         Testing R2
         0.8392010239400369
```

It was observed that the important predictor variables had not changed even after doubling the value of alpha for Lasso regression.

The important predictor variables were:

- Overall Quality of the material and finish of the house
- Basement Full Bathrooms
- Size of Garage in Car capacity
- Overall Condition of the house
- Number of Fireplaces

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?

Considering both models, I would opt for Lasso regression as Lasso helps with the feature elimination for the non-significant features and that will eventually help our model to be more robust.

Question 3

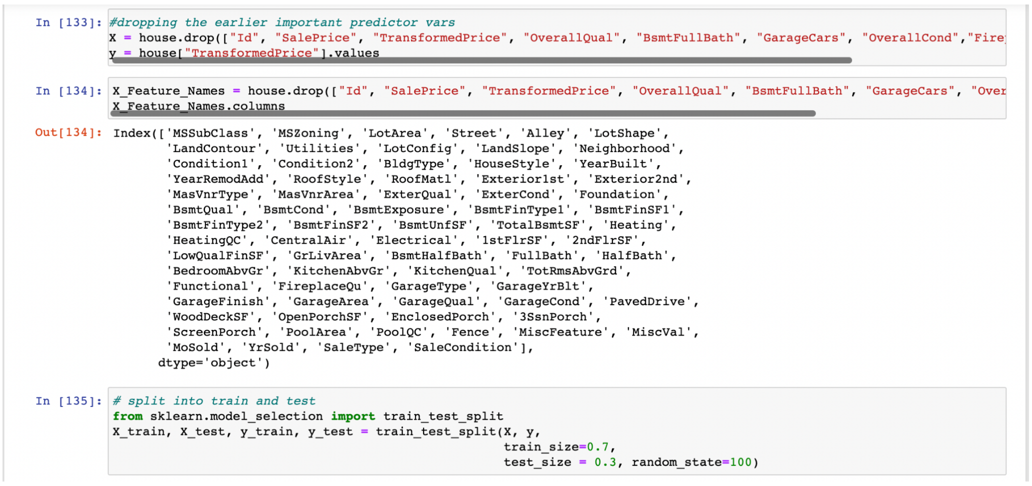
After building the model, you realised that the five most important predictor variables in the lasso model are not available in the incoming data. You will now have to create another model excluding the five most important predictor variables. Which are the five most important predictor variables now?

After dropping the following five most important predictor variables from the Lasso Model obtained earlier
- Overall Quality of the material and finish of the house
- Basement Full Bathrooms
- Size of Garage in Car capacity
- Overall Condition of the house
- Number of Fireplaces

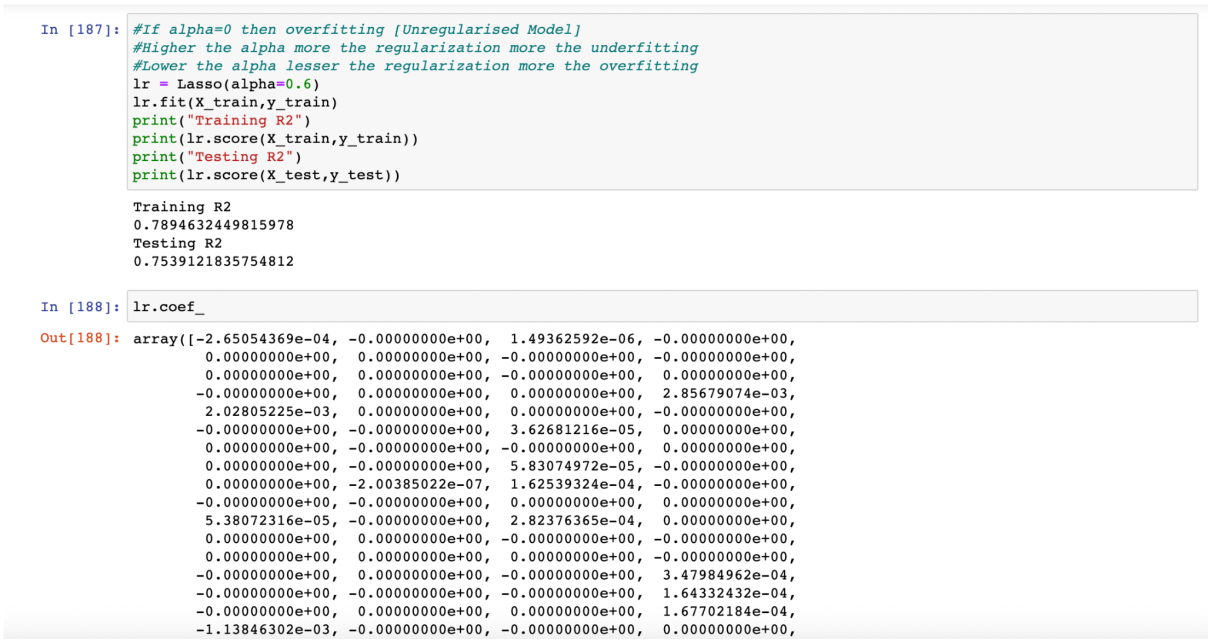
The observations were as follows:

1. The optimal value of alpha changed from 0.002 to 0.6
2. The five most important predictor variables got changed to the following:

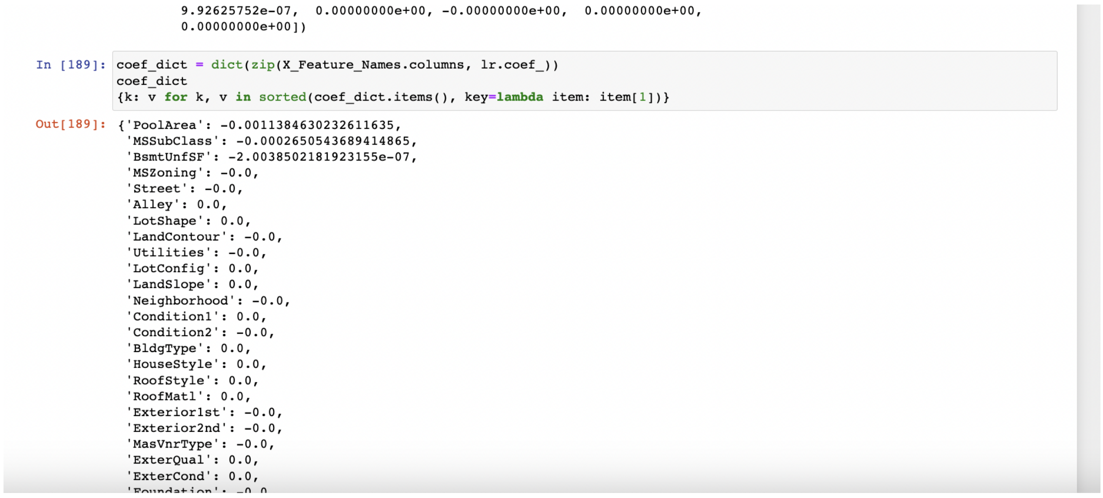
- Year Built: The original date of construction
- YearRemodAdd: Remodel Date
- GarageArea: Area of Garage
- GrLivArea: Above Grade Living Area Square ft
- ScreenPorch: Screen porch area in sq ft



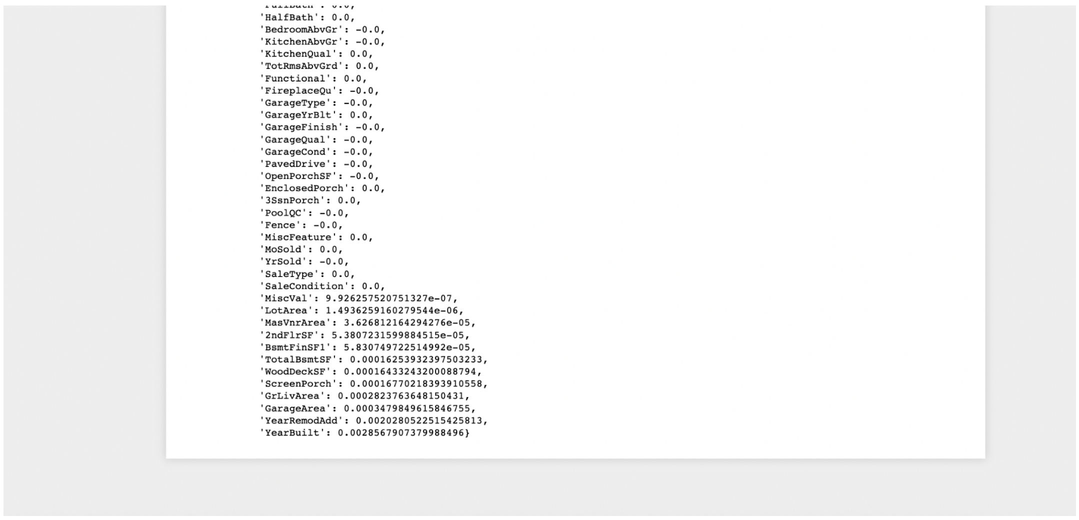
```
In [133]: #dropping the earlier important predictor vars
          X = house.drop(["Id", "SalePrice", "TransformedPrice", "OverallQual", "BsmtFullBath", "GarageCars", "OverallCond","Fire
          y = house["TransformedPrice"].values

In [134]: X_Feature_Names = house.drop(["Id", "SalePrice", "TransformedPrice", "OverallQual", "BsmtFullBath", "GarageCars", "Over
          X_Feature_Names.columns

Out[134]: Index(['MSSubClass', 'MSZoning', 'LotArea', 'Street', 'Alley', 'LotShape',
                 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood',
                 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'YearBuilt',
                 'YearRemodAdd', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd',
                 'MasVnrType', 'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation',
                 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
                 'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
                 'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
                 'LowQualFinSF', 'GrLivArea', 'BsmtHalfBath', 'FullBath', 'HalfBath',
                 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd',
                 'Functional', 'FireplaceQu', 'GarageType', 'GarageYrBlt',
                 'GarageFinish', 'GarageArea', 'GarageQual', 'GarageCond', 'PavedDrive',
                 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
                 'ScreenPorch', 'PoolArea', 'PoolQC', 'Fence', 'MiscFeature', 'MiscVal',
                 'MoSold', 'YrSold', 'SaleType', 'SaleCondition'],
                dtype='object')

In [135]: # split into train and test
          from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                              train_size=0.7,
                                                              test_size = 0.3, random_state=100)
```



```
In [187]: #If alpha=0 then overfitting [Unregularised Model]
          #Higher the alpha more the regularization more the underfitting
          #Lower the alpha lesser the regularization more the overfitting
          lr = Lasso(alpha=0.6)
          lr.fit(X_train,y_train)
          print("Training R2")
          print(lr.score(X_train,y_train))
          print("Testing R2")
          print(lr.score(X_test,y_test))

          Training R2
          0.7894632449815978
          Testing R2
          0.7539121835754812

In [188]: lr.coef_

Out[188]: array([-2.65054369e-04, -0.00000000e+00,  1.49362592e-06, -0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00, -0.00000000e+00, -0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00, -0.00000000e+00,  0.00000000e+00,
                 -0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  2.85679074e-03,
                  2.02805225e-03,  0.00000000e+00,  0.00000000e+00, -0.00000000e+00,
                 -0.00000000e+00, -0.00000000e+00,  3.62681216e-05,  0.00000000e+00,
                  0.00000000e+00, -0.00000000e+00, -0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00, -0.00000000e+00,  5.83074972e-05, -0.00000000e+00,
                  0.00000000e+00, -2.00385022e-07,  1.62539324e-04, -0.00000000e+00,
                 -0.00000000e+00, -0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  5.38072316e-05, -0.00000000e+00,  2.82376365e-04,  0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00, -0.00000000e+00, -0.00000000e+00,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00, -0.00000000e+00,
                 -0.00000000e+00,  0.00000000e+00, -0.00000000e+00,  3.47984962e-04,
                 -0.00000000e+00, -0.00000000e+00, -0.00000000e+00,  1.64332432e-04,
                 -0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  1.67702184e-04,
                 -1.13846302e-03, -0.00000000e+00, -0.00000000e+00,  0.00000000e+00,
```



```
                  9.92625752e-07,  0.00000000e+00, -0.00000000e+00,  0.00000000e+00,
                  0.00000000e+00])

In [189]: coef_dict = dict(zip(X_Feature_Names.columns, lr.coef_))
          coef_dict
          {k: v for k, v in sorted(coef_dict.items(), key=lambda item: item[1])}

Out[189]: {'PoolArea': -0.0011384630232611635,
           'MSSubClass': -0.0002650543689414865,
           'BsmtUnfSF': -2.0038502181923155e-07,
           'MSZoning': -0.0,
           'Street': -0.0,
           'Alley': 0.0,
           'LotShape': 0.0,
           'LandContour': -0.0,
           'Utilities': -0.0,
           'LotConfig': 0.0,
           'LandSlope': 0.0,
           'Neighborhood': -0.0,
           'Condition1': 0.0,
           'Condition2': -0.0,
           'BldgType': 0.0,
           'HouseStyle': 0.0,
           'RoofStyle': 0.0,
           'RoofMatl': 0.0,
           'Exterior1st': -0.0,
           'Exterior2nd': -0.0,
           'MasVnrType': -0.0,
           'ExterQual': 0.0,
           'ExterCond': 0.0,
           'Foundation': -0.0,
```



```
           'HalfBath': 0.0,
           'BedroomAbvGr': -0.0,
           'KitchenAbvGr': -0.0,
           'KitchenQual': 0.0,
           'TotRmsAbvGrd': 0.0,
           'Functional': 0.0,
           'FireplaceQu': -0.0,
           'GarageType': -0.0,
           'GarageYrBlt': 0.0,
           'GarageFinish': -0.0,
           'GarageQual': -0.0,
           'GarageCond': -0.0,
           'PavedDrive': -0.0,
           'OpenPorchSF': -0.0,
           'EnclosedPorch': 0.0,
           '3SsnPorch': 0.0,
           'PoolQC': -0.0,
           'Fence': -0.0,
           'MiscFeature': 0.0,
           'MoSold': 0.0,
           'YrSold': -0.0,
           'SaleType': 0.0,
           'SaleCondition': 0.0,
           'MiscVal': 9.926257520751327e-07,
           'LotArea': 1.4936259160279544e-06,
           'MasVnrArea': 3.626812164294276e-05,
           '2ndFlrSF': 5.3807231599884515e-05,
           'BsmtFinSF1': 5.830749722514992e-05,
           'TotalBsmtSF': 0.00016253932397503233,
           'WoodDeckSF': 0.00016433243200088794,
           'ScreenPorch': 0.00016770218393910558,
           'GrLivArea': 0.0002823763648150431,
           'GarageArea': 0.0003479849615846755,
           'YearRemodAdd': 0.0020280522515425813,
           'YearBuilt': 0.002856790737998496}
```

Question 4

How can you make sure that a model is robust and generalisable? What are the implications of the same for the model's accuracy and why?

Ensuring that a model is robust and generalisable involves multiple steps:

1. Use a diverse and representative dataset: A diverse and representative dataset can help to ensure that the model is exposed to a broad range of examples and can learn to generalise to new cases. This can help to prevent overfitting, where the model memoizes the training data and performs poorly on new data.
2. Use cross-validation: Cross-validation is a technique used to estimate how well the model will generalise to new data. By splitting the data into training and validation sets and evaluating the model on the validation set, we can get a sense of how well the model will perform on new data.
3. Regularisation: Regularization is a technique used to prevent overfitting by adding a penalty term to the loss function. This penalty term discourages the model from learning complex relationships that may only be present in the training data.
4. Data augmentation: Data augmentation involves artificially increasing the size of the dataset by applying transformations such as rotation, flipping, or scaling. This can help to expose the model to more examples and make it more robust to variations in the input data.

The implications of achieving robustness and generalisability for the accuracy of the model are significant. A robust and generalisable model is more likely to perform well on new and unseen data, which is the ultimate goal of any machine learning model. However, achieving robustness and generalisability can sometimes come at the cost of accuracy in the training data. For example, regularisation may cause the model to generalise better but result in slightly lower accuracy on the training data. It's important to strike a balance between achieving robustness and generalisability while still maintaining good performance on the training data.