MSDS 422 Assignment 2

Husein Adenwala

Introduction

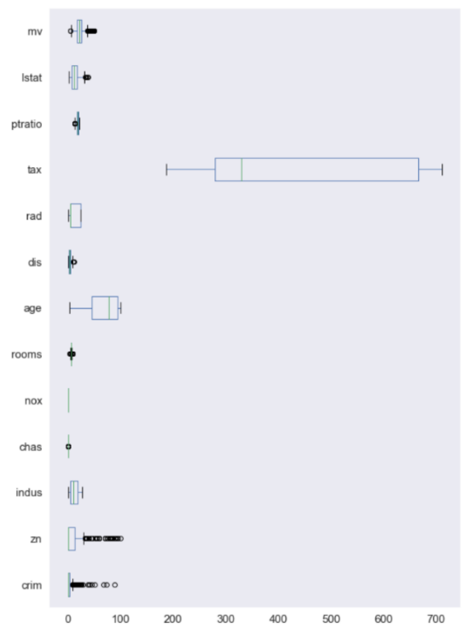
The purpose of this study is to evaluate machine learning regression models for a real estate firm to understand the impact of environmental & other variables and predict the median housing prices in Boston.

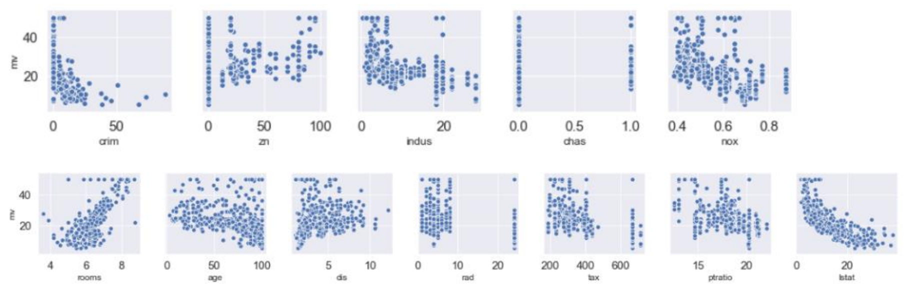
Data preparation, exploration, visualization

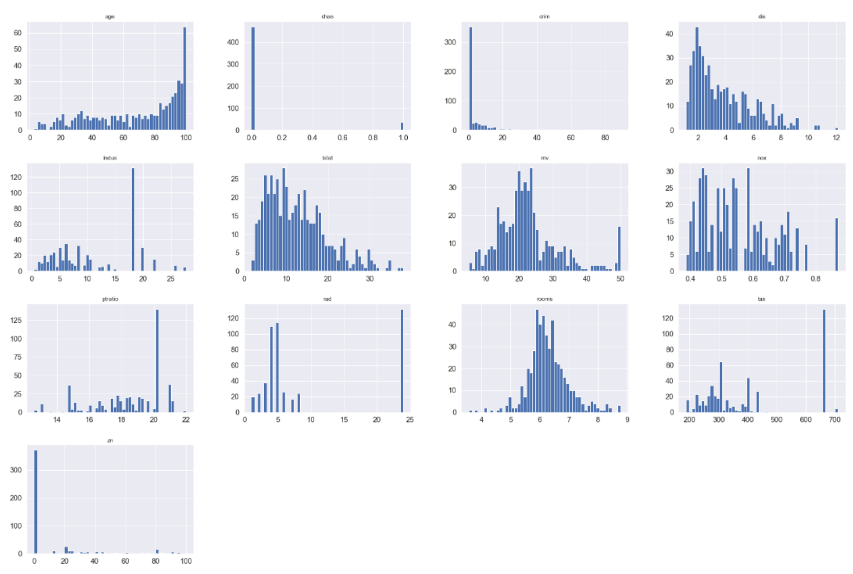
The Boston Housing Study Data contains 506 observations and 14 features. The data did not have any missing values. For the purpose of this study, neighborhood, a categorical attribute, was removed and all numeric attributes were included.

I created boxplots, scatter plots and histograms to evaluate all the features from the raw data prior to transforming it.

The boxplot shows some of the variables such as tax are on a different scale and would require scaling. Additionally, from the scatterplot, we can see that some of the distribution, such as lstat, crim, age and indus don't have a linear relationship to the response variable mv. We can also see from the histogram that some of the distributions are skewed, such as age, dis, and lstat.







In order to perform regression using gradient descent, we want the features to be on a similar scale and have more or less normal distribution.

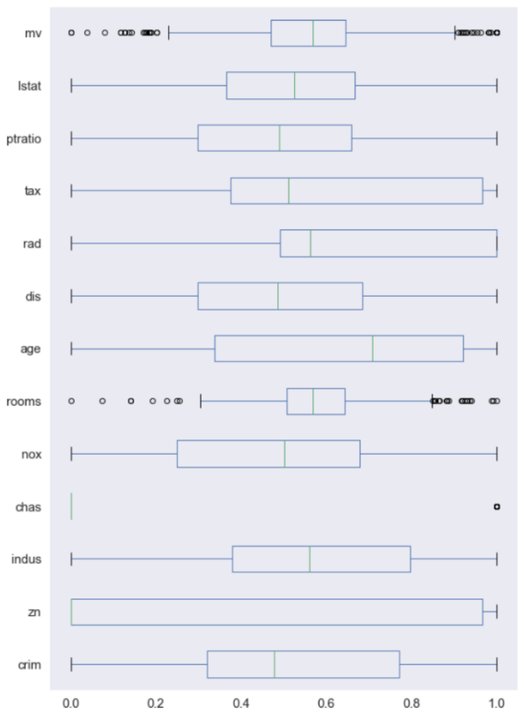
To start, I performed a box-cox variable transformation and then did min-max scaling to the features to achieve normality and similar scales.

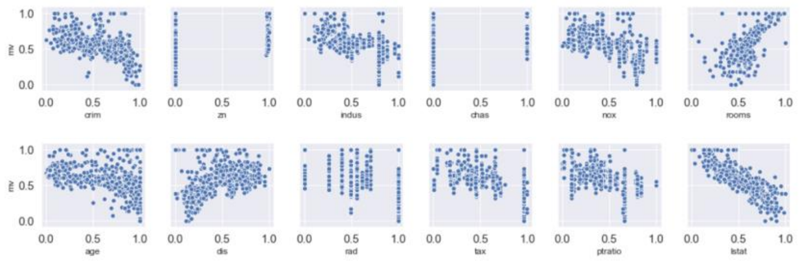
The Box-Cox transformation is a family of power transform functions that are used to stabilize variance and make a dataset look more like a normal distribution.

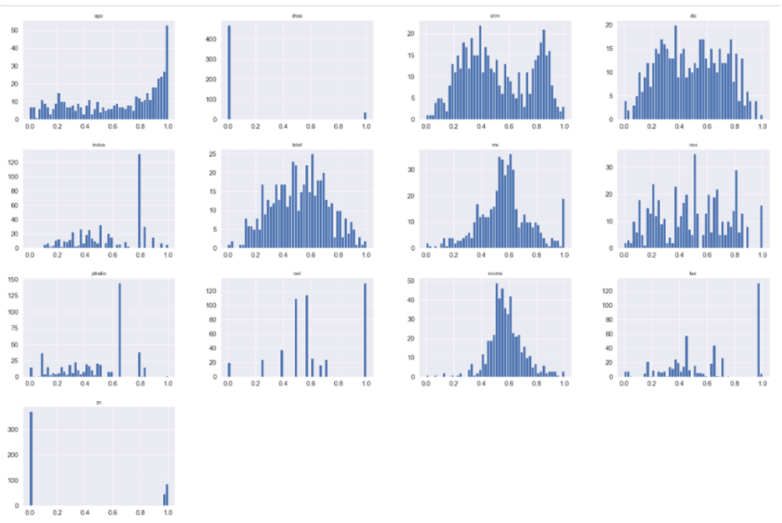
Min-max feature scaling standardizes the independent features in the data in a fixed range (0-1). If feature scaling is not done, then a machine learning algorithm will weigh greater values higher and consider smaller values as the lower values, regardless of the unit of the values and this would compromise the model.

In order to verify that the transformation and scaling were effective, I created a new set of scatter plots. These showed that the relationship with the response variable mv are was closer to linear than before, for example nox, indus, age, dis and lstat, which is required to effectively perform regression using gradient descent.

The boxplot shows that all the features are on the same scale and the histogram shows that the features are more normally distributed than before.







Feature Selection

To select the most relevant features I used backward elimination (using linear regression) which at first included all 12 explanatory variables and then, with each iteration, I discarded the least statistically significant variables that had a p-value >0.05. This resulted in the following 8 features that I used for this study.

chas nox rooms dis rad tax ptratio lstat

It is important to note that some of the features have high collinearity, for example chas and nox.

Review research design and modeling methods

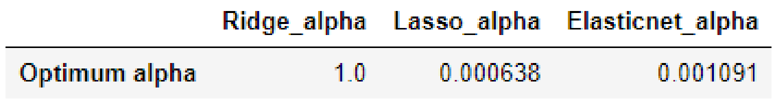
Standard linear Regression, Ridge, Lasso and Elastic-net Regression

For this analysis, I used a regression model because of the continuous nature of our response variable MV and its approximate linear relationships with all predictor variables. Specifically, I used standard Linear, Ridge, Lasso and Elastic Net regression. Each of these models have various characteristics that make them suitable for specific datasets.

Standard linear regression is simple to implement and will be the benchmark model. However, because the standard linear regression model can over-fit the data, I used regularization techniques (Ridge,Lasso and Elastic-net) to avoid over-fitting.

Ridge regression decreases the complexity of a model but does not reduce the number of variables since it never leads to a coefficient being zero; rather, it only minimizes it. Hence, this model is not good for feature reduction, but on the other hand, a benefit of Lasso regression is that it tends to make coefficients to absolute zero as compared to Ridge but if there are two or more highly collinear variables, then Lasso regression selects one of them randomly which is not good for the interpretation of data. Elastic Net combines the regularization of both Lasso and Ridge. The advantage of that is that it does not easily eliminate the coefficient with high collinearity.

For the Ridge, Lasso and Elastic-net regression models, I used the cross validation function that automatically selects the alpha value for which the cross-validation error is minimal (see code in the appendix).



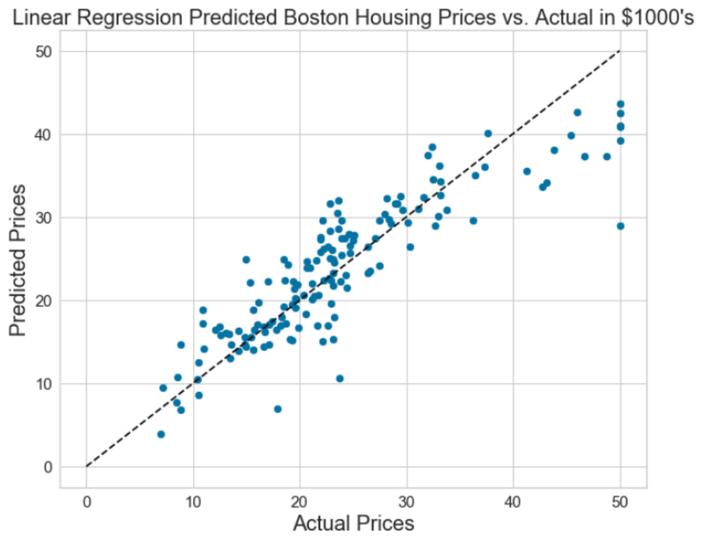
	Ridge_alpha	Lasso_alpha	Elasticnet_alpha
Optimum alpha	1.0	0.000638	0.001091

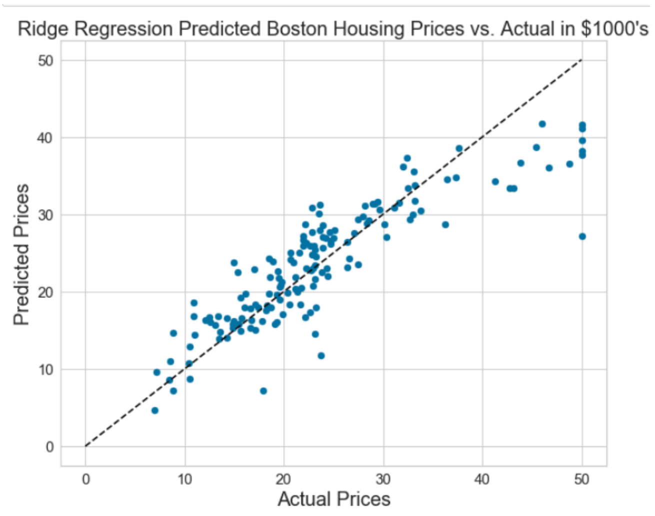
To evaluate these models, I used only the selected 8 features and did a 70/30 split on the dataset for Training and Testing respectively. I used the same test and training data on all of the models to do a fair comparison.

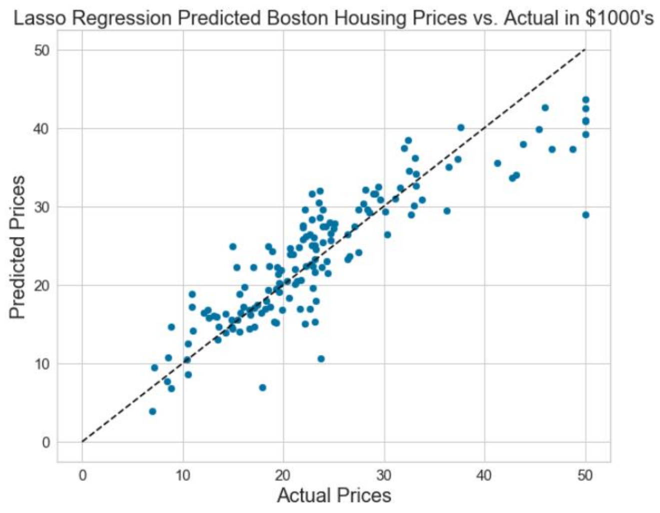
I also used the k-fold cross validation (k=10) technique to calculate and compare the root mean square error for each model. Additionally, I ran a comparison of R squared and root mean squared error between train and test data as an evaluation metric.

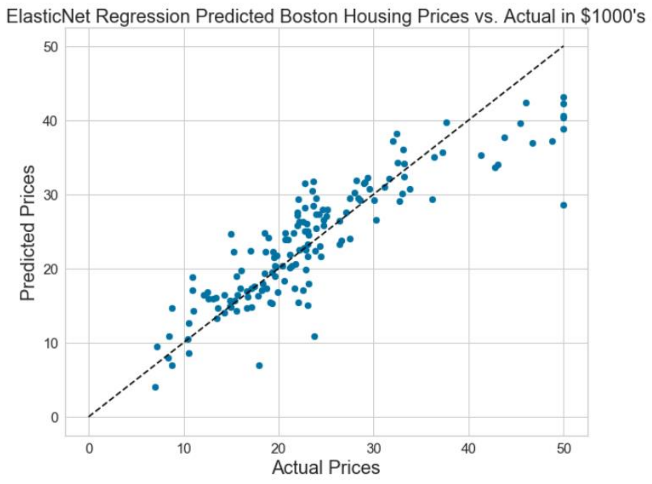
Review results, evaluate models

The scatter plot for actual vs predicted value on the test data shows that there is no big difference in the prediction performance of all four models.







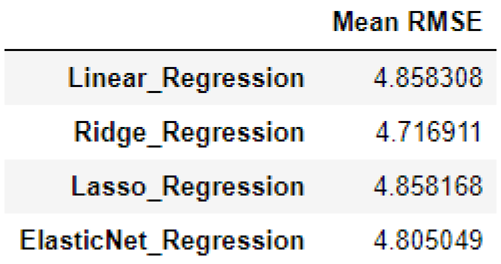


The R squared and residual mean square error for test and train data is similar for all models. The difference in RMSE for test and train data for all the models suggests there is some over-fitting. Linear regression is slightly better in performance than all the other models, as it had the lowest RMSE value.

In general, any of these models can explain between 76-77 % of the variation in the mean home price and can predict the sale price with a ± standard error of $4200 - $4600

	Linear R2	Linear RMSE	Ridge R2	Ridge RMSE	Lasso R2	Lasso RMSE	Elastic Net R2	Elastic Net RMSE
Train	0.768381	4.241345	0.763039	4.289973	0.768378	4.241368	0.768073	4.244159
Test	0.778093	4.639329	0.768340	4.740181	0.777910	4.641241	0.776771	4.653123

Mean RMSE using K-fold cross validation on the data set shows that Ridge regression has the lowest standard error followed by Elastic-net, Lasso and Linear, although the range of the standard error is approximately between $4710-$4860.



	Mean RMSE
Linear_Regression	4.858308
Ridge_Regression	4.716911
Lasso_Regression	4.858168
ElasticNet_Regression	4.805049

The intercept and coefficient (see Appendix) are very similar. Lasso or Ridge did not drop any of the coefficients, although Ridge shrunk the coefficients and to some extent, Elastic-net did as well. Lasso and Linear have similar coefficients.

The coefficients of Chas has the lowest weight, Lstat has the highest and nox is in the middle with the fourth highest weight for all models.

The weights of the coefficients show the impact each feature has on the median housing price. Nox is the only pollutant variable that impacts the mean housing prices.

Linear: Y= 44.99 + 2.17*chas -8.09*nox +13.45*rooms -15.4*dis +4*rad -6.86*tax -5.1*ptratio -29.17*lstat

Ridge: Y= 40.98 + 2.27*chas -5.84*nox +13.02*rooms -11.31*dis +3.19*rad -6.36*tax -5.53*ptratio -29.16*lstat

Lasso: Y= 44.93 + 2.16*chas -8.0*nox +13.43*rooms -15.3*dis +3.95*rad -6.83*tax -5.09*ptratio -28.1*lstat

Elastic-net: Y= 44.99 + 2.2*chas -7.14*nox +13.36*rooms -13.79*dis +3.69*rad -6.67*tax -5.1*ptratio -29.1*lstat

Implementation and programming

Using Python Pandas package, I loaded the Boston housing market data csv file to the Data Frame object, then obtained a new subset Data Frame by dropping the Neighborhood features.

Exploratory data analysis was done utilizing the function in the Pandas Numpy, seaborn and matplotlib packages.

I then created a numpy array (y) for response variable mv and removed it from the Boston Data Frame.

I then used the box-cox function from the scipy package to do the variable transformation on the Data Frame and then used the Minmaxsclar() from the scikit-learn package to scale all the values in the Data Frame.

I then did feature selection by doing backward selection utilizing the statsmodel package (in scipy), then created a sub dataframe with the 8 selected features which were then put into a numpy array (X).

At this point, I created a 70/30 train and test split of the Numpy array (X: response variable) and Numpy array (Y: features), then used the linear regression function in scikit learn package to create a linear regression model and then evaluated RMES and R squared for train and test data.

I used the Linear Regression function from scikit learn to create the linear regression model.

Then I used the RidgeCV, LassoCV and ElasticNetCV function from scikit learn to create the Ridge, Lasso and Elastic-net models and also evaluated RMES and R squared for train and test data. The advantage of using the RidgeCV, LassoCV and ElasticNetCV function by default does cross-validation to optimize the alpha value.

Finally, I did K-Fold cross-validation (k=10) using the K-Fold package in scikit-learn. To do this, I created a numpy array of all the data for the 8 full transformed and scaled and the original data response variable and calculated the mean RMSE of the 10 folds for comparison.

Exposition, problem description and management recommendations

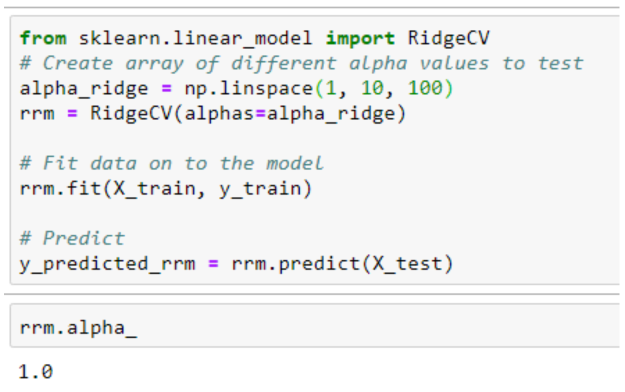
After evaluating all four models, my recommendation to the management is to utilize the Ridge regression model. All four models are very similar in performance, but based on the following analysis, the Ridge regression is a better fit for this dataset.

The R squared and RMSE for test vs train data is similar for all the models and RMSE shows that there is some overfitting in all the models. However, K-fold validation (k=10) is a better technique to evaluate the model fit. Based on the root mean square error calculated by cross validation, I found that the RMSE was lowest for Ridge, followed by Elastic-net and then Lasso and standard linear regression, the last two of which were very similar. The Ridge estimator is good at improving the least-squares estimate when multicollinearity is present and based on my EDA, we know there is multicollinearity between some features (see Appendix for corrplot).

All the reasons mentioned above make a strong case for using Ridge regression for predicting and understanding the impact of pollution, among other variables, on median housing cost.

Appendix

<u>validation using RIdgeCV ,LassoSV and ElasticCV</u>



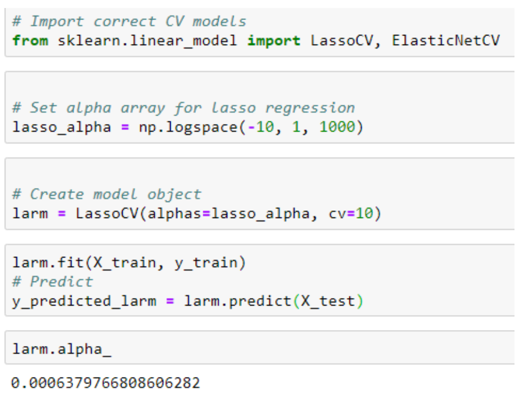
```
from sklearn.linear_model import RidgeCV
# Create array of different alpha values to test
alpha_ridge = np.linspace(1, 10, 100)
rrm = RidgeCV(alphas=alpha_ridge)

# Fit data on to the model
rrm.fit(X_train, y_train)

# Predict
y_predicted_rrm = rrm.predict(X_test)
```

```
rrm.alpha_
```

```
1.0
```



```
# Import correct CV models
from sklearn.linear_model import LassoCV, ElasticNetCV
```

```
# Set alpha array for lasso regression
lasso_alpha = np.logspace(-10, 1, 1000)
```

```
# Create model object
larm = LassoCV(alphas=lasso_alpha, cv=10)
```

```
larm.fit(X_train, y_train)
# Predict
y_predicted_larm = larm.predict(X_test)
```

```
larm.alpha_
```

```
0.0006379766808606282
```

```
# Create model object
enrm = ElasticNetCV(l1_ratio=[.1, .5, .7, .9, .95, .99, 1],alphas=pd.np.linspace(0.0001, .99, 1000), cv=10)
```

```
# Fit data on to the model
enrm.fit(X_train, y_train)

# Predict
y_predicted_enrm = enrm.predict(X_test)
```

```
enrm.alpha_
```

```
0.001090890890890891
```

Intercept and Coef of all the models

```
# linear intercept and coef
print(lrm.intercept_,lrm.coef_)
```

```
44.99364861001558 [  2.17031317  -8.09124374  13.45382475 -15.398248     3.99960024
  -6.85716379  -5.0976918  -29.16742617]
```

```
#ridge intercept and ceof
print(rrm.intercept_,rrm.coef_)
```

```
40.97532471684381 [  2.2704903   -5.83660895  13.01686422 -11.30537419   3.18987709
  -6.35862512  -5.53323678 -26.16201721]
```

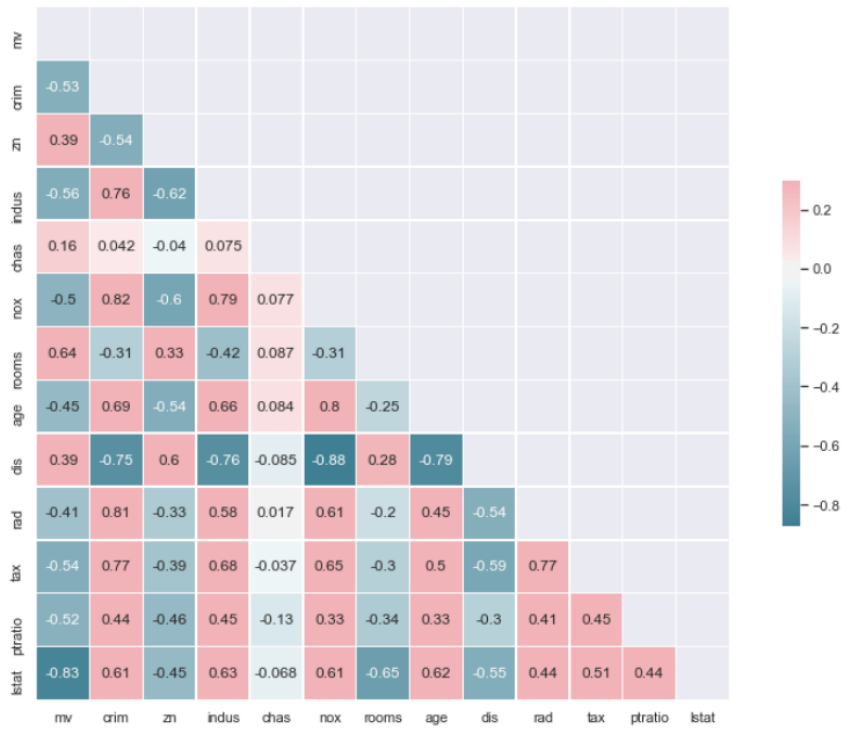
```
#lasso intercept and ceof
print(larm.intercept_, larm.coef_)
```

```
44.92774417277128 [  2.1614633   -8.002155    13.42510142 -15.30008703   3.95227773
  -6.8253145   -5.09107995 -29.16902137]
```

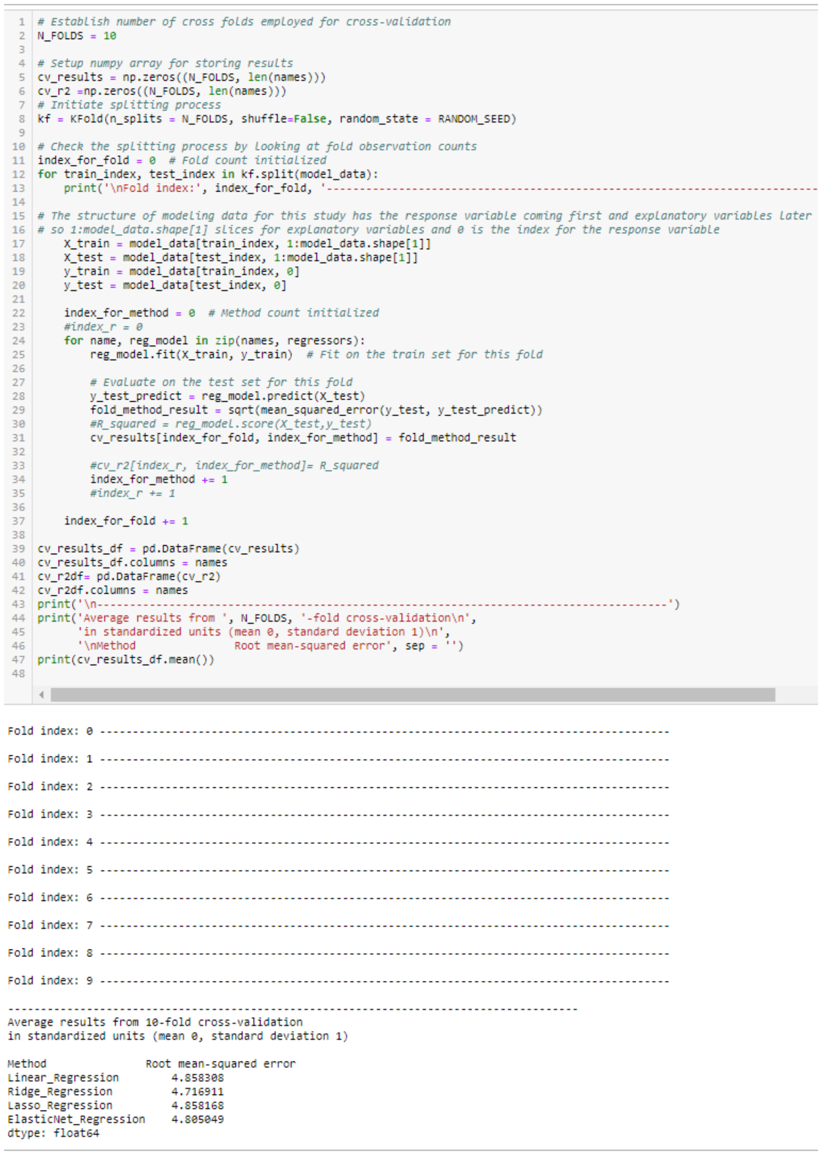
```
# Elastic net intercept and ceof
print(enrm.intercept_, enrm.coef_)
```

```
43.391778766646794 [  2.20244743  -7.13723806  13.36151989 -13.79013107   3.69104128
  -6.67009627  -5.26028005 -28.10135159]
```

Correlation plot



<u>K-FOLD Cross-Validation.</u>



```python
# Establish number of cross folds employed for cross-validation
N_FOLDS = 10

# Setup numpy array for storing results
cv_results = np.zeros((N_FOLDS, len(names)))
cv_r2 =np.zeros((N_FOLDS, len(names)))
# Initiate splitting process
kf = KFold(n_splits = N_FOLDS, shuffle=False, random_state = RANDOM_SEED)

# Check the splitting process by looking at fold observation counts
index_for_fold = 0  # Fold count initialized
for train_index, test_index in kf.split(model_data):
    print('\nFold index:', index_for_fold, '-----------------------------------------------------------------------------

# The structure of modeling data for this study has the response variable coming first and explanatory variables later
# so 1:model_data.shape[1] slices for explanatory variables and 0 is the index for the response variable
    X_train = model_data[train_index, 1:model_data.shape[1]]
    X_test = model_data[test_index, 1:model_data.shape[1]]
    y_train = model_data[train_index, 0]
    y_test = model_data[test_index, 0]

    index_for_method = 0  # Method count initialized
    #index_r = 0
    for name, reg_model in zip(names, regressors):
        reg_model.fit(X_train, y_train)  # Fit on the train set for this fold

        # Evaluate on the test set for this fold
        y_test_predict = reg_model.predict(X_test)
        fold_method_result = sqrt(mean_squared_error(y_test, y_test_predict))
        #R_squared = reg_model.score(X_test,y_test)
        cv_results[index_for_fold, index_for_method] = fold_method_result

        #cv_r2[index_r, index_for_method]= R_squared
        index_for_method += 1
        #index_r += 1

    index_for_fold += 1

cv_results_df = pd.DataFrame(cv_results)
cv_results_df.columns = names
cv_r2df= pd.DataFrame(cv_r2)
cv_r2df.columns = names
print('\n------------------------------------------------------------------------------------')
print('Average results from ', N_FOLDS, '-fold cross-validation\n',
      'in standardized units (mean 0, standard deviation 1)\n',
      '\nMethod                 Root mean-squared error', sep = '')
print(cv_results_df.mean())
```

```
Fold index: 0 -------------------------------------------------------------------------------------

Fold index: 1 -------------------------------------------------------------------------------------

Fold index: 2 -------------------------------------------------------------------------------------

Fold index: 3 -------------------------------------------------------------------------------------

Fold index: 4 -------------------------------------------------------------------------------------

Fold index: 5 -------------------------------------------------------------------------------------

Fold index: 6 -------------------------------------------------------------------------------------

Fold index: 7 -------------------------------------------------------------------------------------

Fold index: 8 -------------------------------------------------------------------------------------

Fold index: 9 -------------------------------------------------------------------------------------

------------------------------------------------------------------------------------
Average results from 10-fold cross-validation
in standardized units (mean 0, standard deviation 1)

Method                 Root mean-squared error
Linear_Regression       4.858308
Ridge_Regression        4.716911
Lasso_Regression        4.858168
ElasticNet_Regression   4.805049
dtype: float64
```