Gurjus Singh

MSDS 422 – Practical Machine Learning

September 27th, 2020

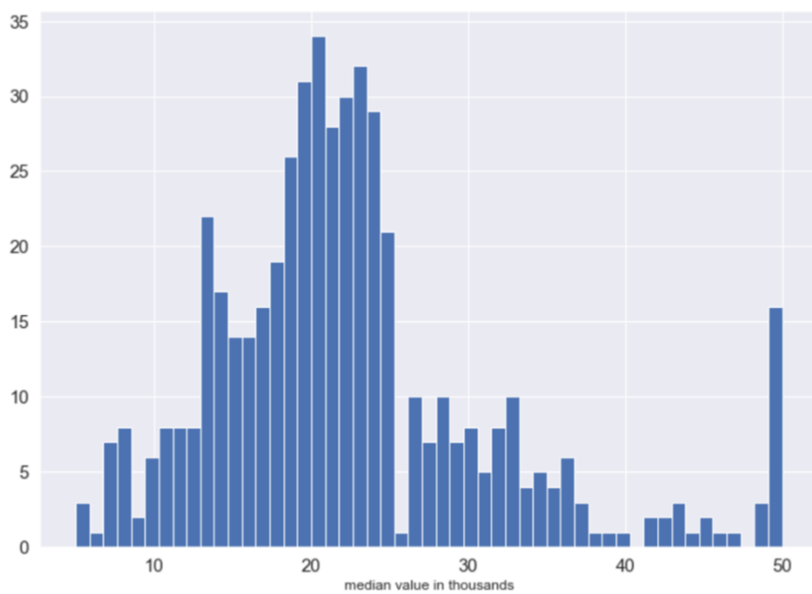
Evaluating Regression Models Assignment # 2

Data preparation, exploration, visualization

I first started out by loading the dataset "Boston.csv". I first wanted to get an initial look and feel for the data set. So, I looked at the size of the data set which was (512, 14). This means there are 512 rows and 14 columns mainly features and one variable we want to predict. After getting a feel for the size of the dataset I wanted to check out the head of data frame and types for every column. I saw non-numeric row, which we do not want to use in linear regression, so I dropped this feature column. Most of the columns were mainly floats, except for two columns which contained integer values. I used .describe() value and .isnull.sum() to find out whether the initial data had any typos, or missing value. From looking at min-max category in describe table I did not see anything obviously wrong, so I decided to continue. I also did not see any columns with NA values which is what isnull.sum() does. This counts up all the values which evaluate to True in a column.

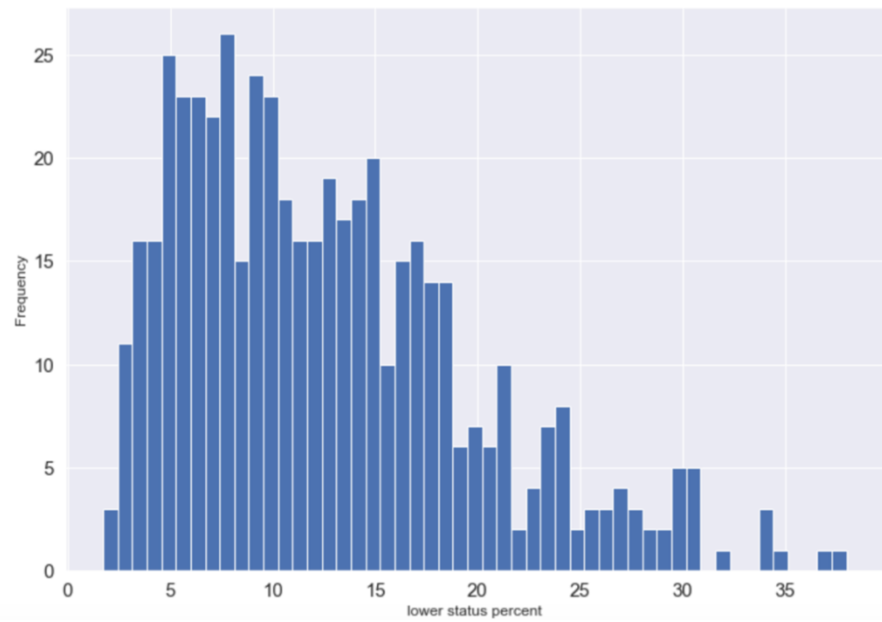
Boxplot of Columns (Plot 1-1)

I then decided to go to exploring through data visualizations of the data set. I first tried to boxplot as depicted in Plot 1-1. In the plot I could see many of our columns had an abundance of outliers such as our target variable MV, lstat, dis, rooms, chas, zn and crim. I wanted to play around more and look closely at some of the variables, so I decided to see distributions through histograms.



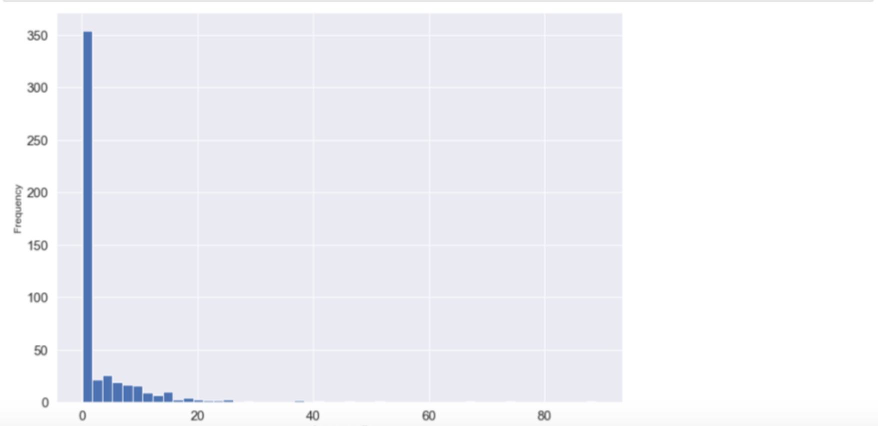
Histogram Plot 1-2

In plot 1-2, it is the median value of the houses, which we will be predicting on this using the independent variables. This looks like a less skewed distribution in comparison to doing the boxplot. I also wanted to look at features I thought were more important to predicting this dataset such as crim, which is the per capita crime rate per town, ptratio which is pupil to teacher ratio by town and lstat which is percent of lower status in an area.

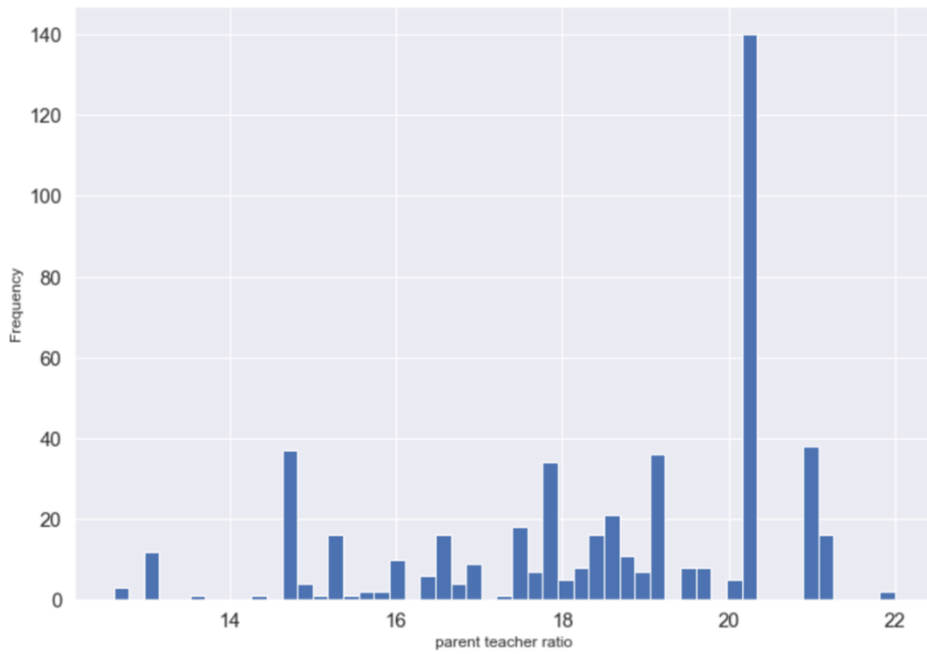


Histogram Plot 1-3

```
In [203]: plt.hist(boston_df['crim'], bins = 51)
          plt.xlabel('crim per capita')
          plt.ylabel('Frequency')
          plt.show()
```

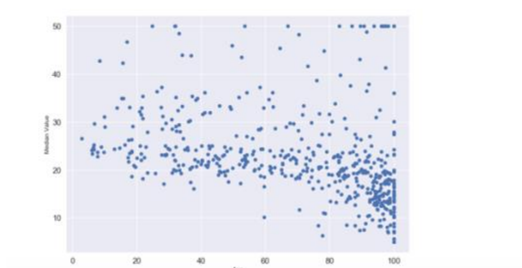


Histogram 1-4



Histogram 1-5

The lstat variable in 1-3 seems to be skewed to the right where 10% in a given neighborhood or region are mostly of lower status. In 1-4 the histogram is right skewed as well with most of the crime happening near 0 in the areas in Boston where the data is collected. In Histogram 1-5, we see for ptratio, is near 20 which mean the difference in teacher to student ratio is high, this data is more skewed to the left. Lastly, I wanted to make some plots to see any initial correlation between MV, and these variables. I also wanted to see if age was in any way correlated.



Dot plot of Age variable compared to MV Plot 1-6

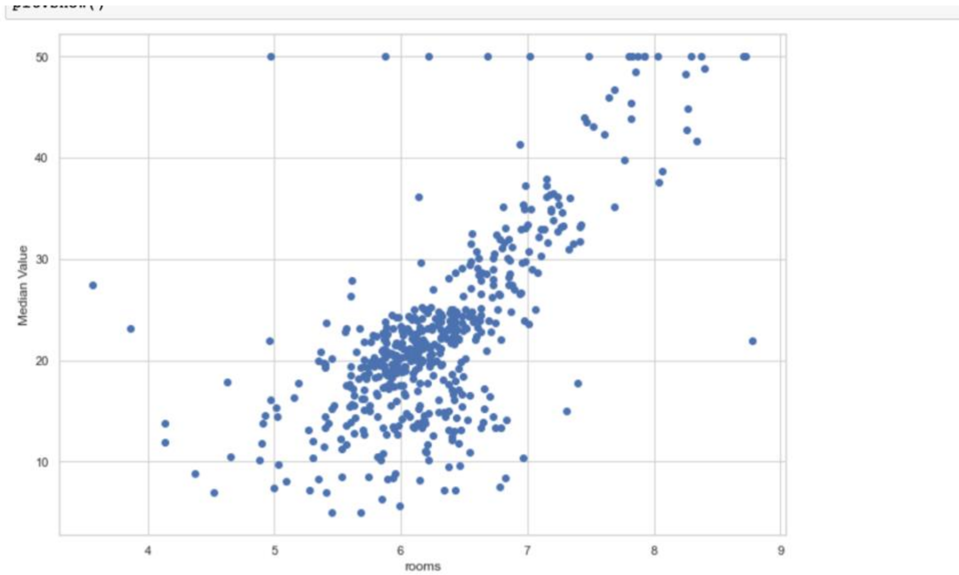


Figure 1-7 of highly linear 'rooms' variable

In plot 1-6, I do not see any linear shape compared to age, which suggests there is no relationship in MV, and the age variable. I also looked at crim variable which did not seem to have a linear shape as well. During my initial findings I also saw that rooms and lstat variables seemed to have a linear shape which means they were highly positive or negatively correlated as seen in figure 1-7.

After examining linear shaped data and nonlinear data, I then wanted to scale the features between 0 and 1 as it helps speed up the learning process. In order to make the data more linear I had to use the lambda function which helps to map the function to all the data in one line of code. I did not want any zeros in my data, so I added .01 to every data point. I then used "boxcox" which is great for making sure my data is more normalized and more linear to use linear models on it.

Before modeling using the linear regression methods, I first wanted to create a correlation heatmap to get an overall picture of which features in the data set might be more useful to predict median value price of a neighborhood of houses. This heatmap showed exactly what might initial findings confirmed in that lstat was negatively correlated, and rooms was highly positively correlated as shown in figure 1-8 below.

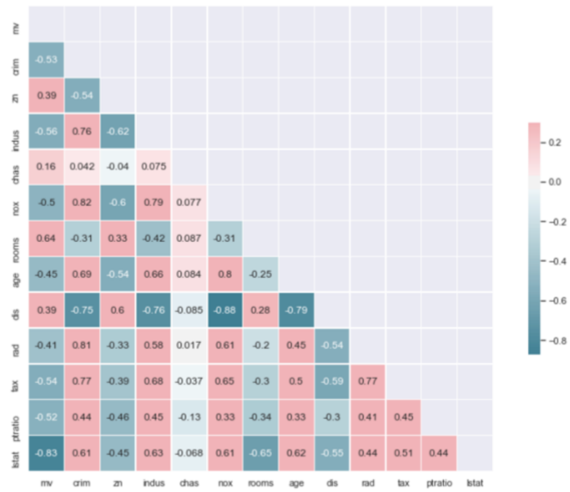


Figure 1-8 Heatmap

Review research design and modeling methods

After we choose, the features I am going to use 4 modeling methods which are main linear regression learning algorithms but have some variations. The four types are Regular Linear Regression, Ridge Regression, Lasso Regression and Elastic Net Regression. The differences between the four is that Linear Regression does not add regularization term which is important in most cases to prevent overfitting [1]. Ridge Regression does incorporate regularization but is different than Lasso Regression in terms of the cost function. ElasticNet Regression is different in a way because it provides a median between both Ridge Regression and Lasso Regression [1]. I think it is important to try a range of algorithms because one might work better than the other in terms of how much it overfits which is what we do not want when using training set. We want the learning algorithm to generalize rather than to remember the data.

Before we start training the algorithm, I had to save the target variable in its own data frame to separate it from the features. In order to use the regression methods, it is important to split up the data into train data, and test data. I did the 80-20 split on the data. Our goal was to make sure that the data trained and predicted well on the test set.

Review Results, and Evaluate Model

I was expecting to find Elastic Net would be the best at prediction as that is what I found out in Geron's book [1]. I first used all the features and did not split the data initially. I used Ordinary Least Squares Regression which is a type of linear regression which involves finding the minimization of the sum of squared differences between predicted and actual value [2]. I found the model was overfitting with and R^2 of around 0.947 so it would not generalize to new data as depicted in Table 1-9.

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                     mv   R-squared (uncentered):                   0.947
Model:                            OLS   Adj. R-squared (uncentered):              0.946
Method:                 Least Squares   F-statistic:                              742.2
Date:                Sat, 26 Sep 2020   Prob (F-statistic):                   5.91e-307
Time:                        15:39:19   Log-Likelihood:                         -1587.5
No. Observations:                 506   AIC:                                      3199.
Df Residuals:                     494   BIC:                                      3250.
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
crim           1.7673      2.636      0.670      0.503      -3.413       6.947
zn             1.6570      0.822      2.017      0.044       0.043       3.271
indus          6.5012      2.024      3.213      0.001       2.525      10.477
chas           2.7501      1.027      2.679      0.008       0.733       4.767
nox            2.2669      2.472      0.917      0.360      -2.591       7.125
rooms         38.5654      2.092     18.439      0.000      34.456      42.675
age            3.6543      1.516      2.410      0.016       0.676       6.633
dis            7.3982      1.869      3.959      0.000       3.726      11.070
rad            4.1958      1.900      2.208      0.028       0.463       7.929
tax           -6.8079      1.651     -4.124      0.000     -10.052      -3.564
ptratio       -4.2149      1.358     -3.103      0.002      -6.884      -1.546
lstat        -17.2472      2.058     -8.381      0.000     -21.290     -13.204
==============================================================================
Omnibus:                      174.584   Durbin-Watson:                   0.921
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1017.532
Skew:                           1.383   Prob(JB):                    1.11e-221
Kurtosis:                       9.372   Cond. No.                         23.4
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Table 1-9

I then split the data as I mentioned in the previous section and did not remove features. I wanted to see how the models performed on the test set. What I found was quite astonishing. I found that Ridge Regression performed slightly better than ElasticNet which was surprising. I thought Lasso Regression would also perform better but that was not the case as Ridge Regression was also better.

```
Linear Regression R_squared =  0.7758492736746525
Linear Regression RMSE =  17.064112097020118
```

```
Ridge Regression R_squared =  0.7956757587559492
Ridge Regression RMSE =  0.08581500092332975
```

```
Lasso Regression R_squared =  0.7868979467222679
Lasso Regression RMSE =  0.0876389329569984
```

```
ElasticNet Regression R_squared =  0.794496153744055
ElasticNet Regression RMSE =  0.08606235807161436
```

Results 1-10

I then tried taking out p-values greater than 0.05 as suggested by professor and shown in Table 1-9. I only found crim variable was above 0.05, and the assignment goal was to predict using nox variable. I saw that the models worsened. This time the best model was Regular Linear Regression. I then wanted to take a peak back at the table in 1-9 to find out if I could find anything other findings. I tried to take out values as shown in Code 1-12. The results got progressively worse. With the best model being yet again Linear Regression. So, in a way, I think keeping all the features was the best way to go about this predicting median values in housing prices.

```
Linear Regression R_squared =  0.7599270264445025
Linear Regression RMSE =  4.9937646239707645
```

```
Ridge Regression R_squared =  0.7518119993517006
Ridge Regression RMSE =  5.077463649091546
```

```
Lasso Regression R_squared =  0.7595814301080018
Lasso Regression RMSE =  4.9973577108524205
```

```
ElasticNet Regression R_squared =  0.7588586540425759
ElasticNet Regression RMSE =  5.00486391124726
```

Results 1-11 taking out p-values greater than 0.05 except for nox

```
columns = ['chas', 'crim', 'indus', 'age', 'ptratio', 'zn', 'rad']
```

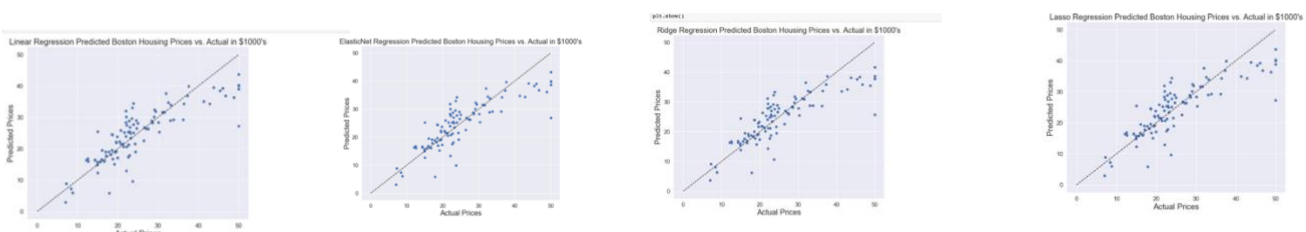
Code 1-12

After seeing different data models and how they ran, I wanted to see cross validation and how it ran. Out of this I think the best models were Lasso Regression and Elastic Net which is what Geron's confirmed would be overall the best models to use. I think Linear Regression was overfitting on the validation set which had the lowest error, while Ridge Regression seemed to be too high.

```
Method                    Root mean-squared error
Linear_Regression             0.097808
Ridge_Regression              0.130851
Lasso_Regression              0.110783
ElasticNet_Regression         0.103750
dtype: float64
```

	Linear_Regression	Ridge_Regression	Lasso_Regression	ElasticNet_Regression
0	0.060632	0.094503	0.067733	0.061043
1	0.064451	0.081897	0.061343	0.063664
2	0.074057	0.059136	0.044540	0.053263
3	0.075747	0.178059	0.141380	0.120119
4	0.087041	0.141569	0.117210	0.107527
5	0.095296	0.156452	0.145455	0.122087
6	0.069371	0.081946	0.080353	0.081682
7	0.207578	0.244361	0.209325	0.202495
8	0.122983	0.172372	0.160443	0.141570
9	0.120922	0.098219	0.080052	0.084045

Results 1-13



Regressions plot 1-14

Implementation and Programming

Different packages I used for this data analysis insight was sklearn package which is important for doing predicting and creating the different linear regression models. In terms of visualizing the data, this is where seaborn and matplotlib came into play. This a great for providing different kinds of barplots on categorical variables, scatter plots on more numerical type variables, and histograms to see how skewed the data is. In order to start implementing the models it is important we transform the data using boxcox which helps normalize or linearlize the data, and make sure we do not have any zeroes in our data [4]. So, in our code we have two lambda functions for doing this [3]. We also want to scale using minmax from the sklearn package. This is important for speeding up the learning algorithms. We then run our models, and it is important the results such R^2 are not high for the training set and are higher for the test set. We can also use cross fold validation to see if our models accurately predict.

Exposition, problem description and management recommendations

From examining the results, on the test set, I do believe it is better to drop features as it performs higher around 0.79 for all the models. From looking at the results in 1-10, the ridge regression results are the best, so I would use this model for the data set. This means the effect of pollution is not a good indicator on the price of houses compared to lstat for example. I do believe if we take out nox, which shows the pollution or nitrogen oxide levels, I think the regression models will perform better, but with nox included it does not perform as best as I wanted. For the future, I would like to see if the models perform higher with nox out of the features. I would also like to automize the task of dropping the features such as using stepwise or a different way to drop the features. I hope to learn other EDA mechanisms as well, as I did see a way, I could use any other EDAs in this assignment to explore the data.

References

[1] Géron, Chapter 4, & Chapter 5
[2] https://www.xlstat.com/en/solutions/features/ordinary-least-squares-regression-ols
[3] https://www.w3schools.com/python/python_lambda.asp
[4] https://www.geeksforgeeks.org/box-cox-transformation-using-python/#:~:text=In%20short%2C%20trying%20to%20move,tests%20than%20we%20could%20have.

Appendix

Import Packages

I imported seaborn, matplotlib, numpy and pandas

In [24]:

```
#imported seaborn; matplotlib
import matplotlib
import numpy as np
import pandas as pd
import os
import itertools
from math import sqrt
from scipy import stats as st
#import cvxopt

import sklearn
from sklearn.preprocessing import StandardScaler # used for
from sklearn.preprocessing import MinMaxScaler as Scaler #
from sklearn.model_selection import train_test_split

import sklearn.linear_model
from sklearn.linear_model import LinearRegression, Ridge, L
from sklearn.ensemble import RandomForestRegressor # Random
from sklearn.ensemble import ExtraTreesRegressor # Extra Tr
from sklearn.ensemble import GradientBoostingRegressor # Gr

from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import make_scorer, accuracy_score

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

import statsmodels.api as sm

from matplotlib import pyplot as plt
from matplotlib import rc
import seaborn as sns
sns.set_style("whitegrid")
sns.set(style="whitegrid", color_codes=True)
plt.rc("font", size=14)
```

In [2]:

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

Read in data from CSV file

In [3]:

```
#import dataset
boston_df=pd.read_csv('/Users/gurjy/Downloads/jump-start-bo
```

Get shape of data

In [4]:

```
#calculate shape
boston_df.shape
```

Out[4]:

```
(506, 14)
```

See the initial data

In [5]:

```
boston_df.head()
```

Out[5]:

	neighborhood	crim	zn	indus	chas	nox	rooms	ag
0	Nahant	0.00632	18.0	2.31	0	0.538	6.575	65
1	Swampscott	0.02731	0.0	7.07	0	0.469	6.421	78
2	Swanpscott	0.02729	0.0	7.07	0	0.469	7.185	61
3	Marblehead	0.03237	0.0	2.18	0	0.458	6.998	45
4	Marblehead	0.06905	0.0	2.18	0	0.458	7.147	54

See the data types of columns

In [6]:

```
boston_df.dtypes
```

Out[6]:

```
neighborhood     object
crim            float64
zn              float64
indus           float64
chas              int64
nox             float64
rooms           float64
age             float64
dis             float64
rad               int64
tax               int64
ptratio         float64
lstat           float64
mv              float64
dtype: object
```

Drop the not quantitative data

In [7]:

```
#Drop non numeric columns
boston_df = boston_df.drop('neighborhood', 1)
```

See the data again with column dropped

In [77]:

```
#see if column has been dropped
boston_df.head()
```

Out[77]:

	crim	zn	indus	chas	nox	rooms	age	dis	rad
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3

Run the traditional statistics to summarize data

In [79]:

```
#check to see if there are typos and look at typical stats
boston_df.describe()
```

Out[79]:

	crim	zn	indus	chas	
count	506.000000	506.000000	506.000000	506.000000	506.000
mean	3.613524	11.363636	11.136779	0.069170	0.554
std	8.601545	23.322453	6.860353	0.253994	0.115
min	0.006320	0.000000	0.460000	0.000000	0.385
25%	0.082045	0.000000	5.190000	0.000000	0.449
50%	0.256510	0.000000	9.690000	0.000000	0.538
75%	3.677082	12.500000	18.100000	0.000000	0.624
max	88.976200	100.000000	27.740000	1.000000	0.871

See if there is any missing data

In [80]:

```
#look at data and check if there is NA values
boston_df.isnull().sum()
```

Out[80]:

```
crim       0
zn         0
indus      0
chas       0
nox        0
rooms      0
age        0
dis        0
rad        0
tax        0
ptratio    0
lstat      0
mv         0
dtype: int64
```

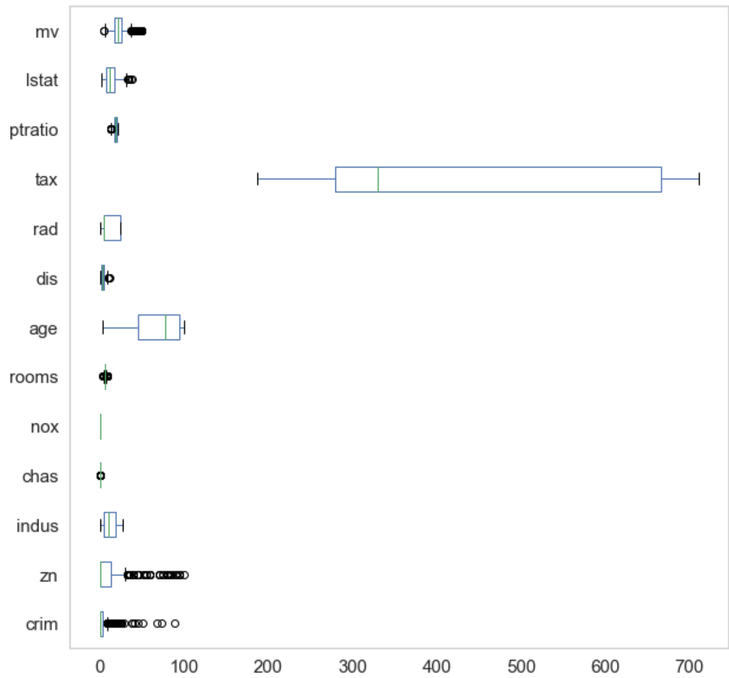
See if there is any outliers in the data

In [81]:

```
boston_df.boxplot(vert=False, figsize=(10,10), grid=False)
```

Out[81]:

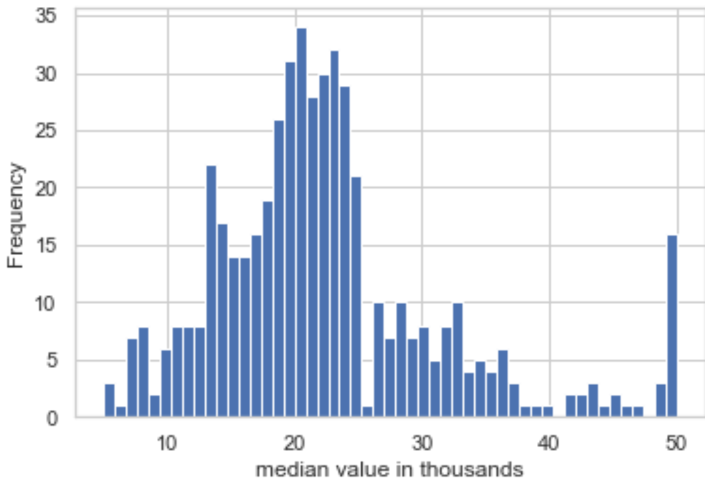
```
<AxesSubplot:>
```



Make a histograms and see median values of house prices in each neigborhood

In [12]:

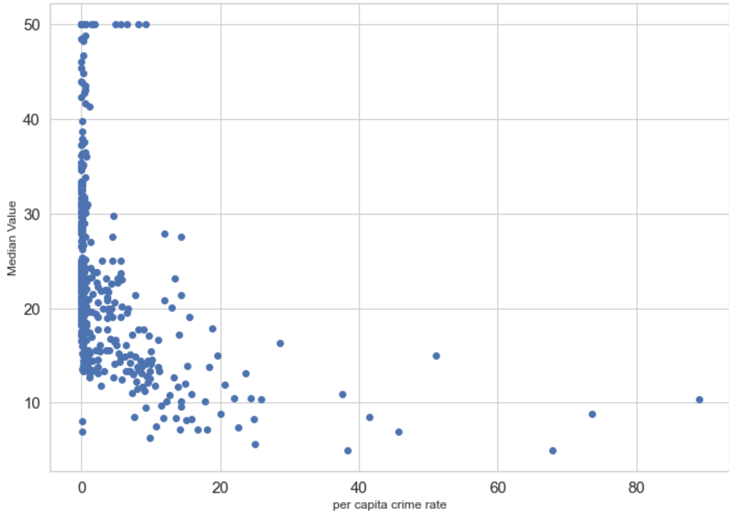
```
plt.hist(boston_df['mv'], bins = 51)
plt.xlabel('median value in thousands')
plt.ylabel('Frequency')
plt.show()
```



Wanted to see which data had linear shape

In [84]:

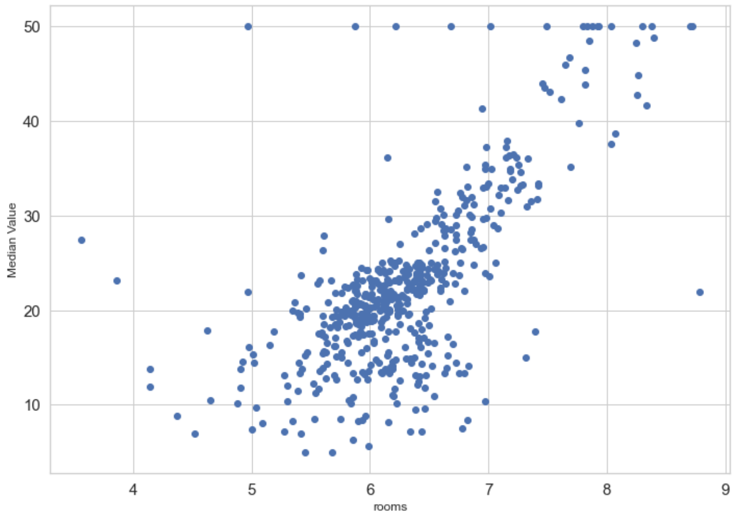
```
plt.plot(boston_df['crim'], boston_df['mv'], 'bo')
plt.xlabel('per capita crime rate')
plt.ylabel('Median Value')
plt.show()
```



Saw rooms column and saw a liner shape which means positive correlation

In [86]:

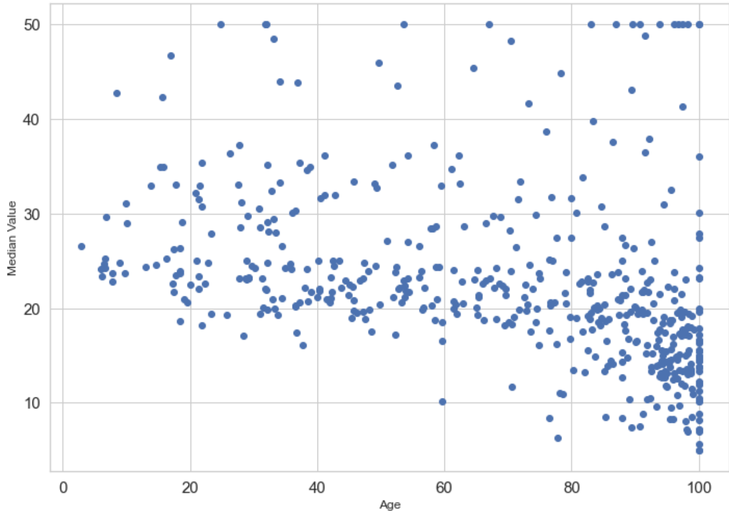
```
plt.plot(boston_df['rooms'], boston_df['mv'], 'bo')
plt.xlabel('rooms')
plt.ylabel('Median Value')
plt.show()
```



Do not see any strong correlation

In [87]:

```
plt.plot(boston_df['age'], boston_df['mv'], 'bo')
plt.xlabel('Age')
plt.ylabel('Median Value')
plt.show()
```



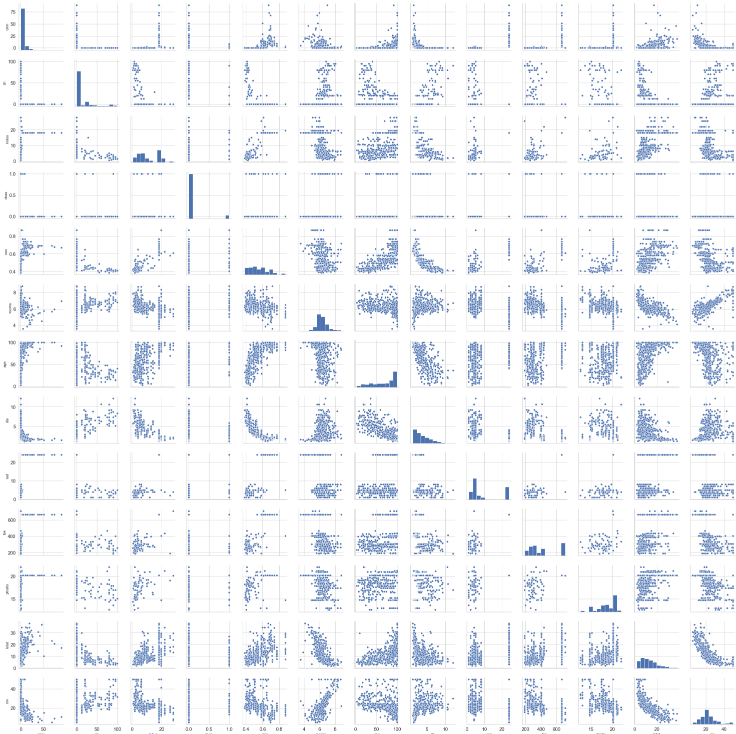
I see initial data in graphs in this pairplot

In [89]:

```
sns.pairplot(boston_df, diag_kind='hist')
```

Out[89]:

<seaborn.axisgrid.PairGrid at 0x1c1f0b6050>



Make copy of data

In [14]:

```
boston_df1=boston_df.copy()
```

Drop all columns except for column

In [15]:

```
#saves the column we want to predict
columns = ['crim','zn','indus','chas','nox','rooms','age','
boston_Target = boston_df1.drop(columns=columns)
```

In [16]:

```
boston_Target
```

Out[16]:

	mv
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2
...	...
501	22.4
502	20.6
503	23.9
504	22.0
505	19.0

506 rows × 1 columns

See shape again should only have 1 column, 506 rows

In [90]:

```
#makes sure shape is right
boston_Target.shape
```

Out[90]:

```
(506, 1)
```

Make lambda function to add .01 to all columns with zeros; then use boxcox to make more normal distribution and make more linear for linear regression

In [91]:

```
#need to transform data to find linear relationship
boston_df2=boston_df.apply(lambda x: x+.01)
boston_df2=boston_df2.transform(lambda x: st.boxcox(x)[0])
```

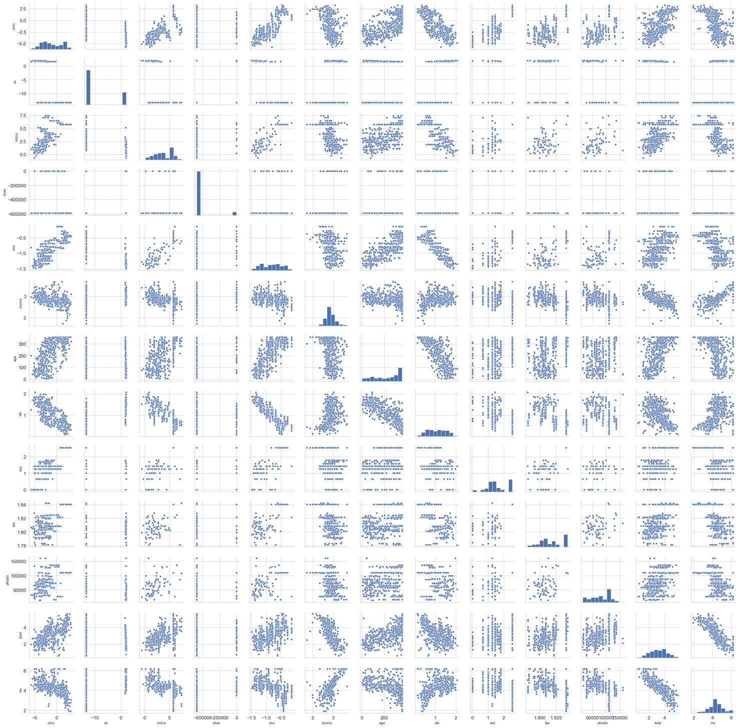
See pairplot again

In [92]:

```
sns.pairplot(boston_df2, diag_kind='hist')
```

Out[92]:

<seaborn.axisgrid.PairGrid at 0x1c262b5ed0>



Use scaler to make everythiing with 0 and 1 using min max scaler

In [93]:

```
#scale data by min max scaler
boston_df3=boston_df2.transform(lambda x: (x - x.min()) / (
```

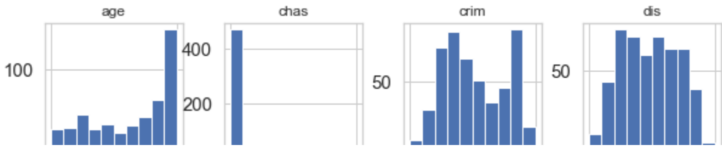
Make histogram matrix and as you can see everything is within 0 and 1.

In [94]:

```
boston_df3.hist(figsize=(10,10))
```

Out[94]:

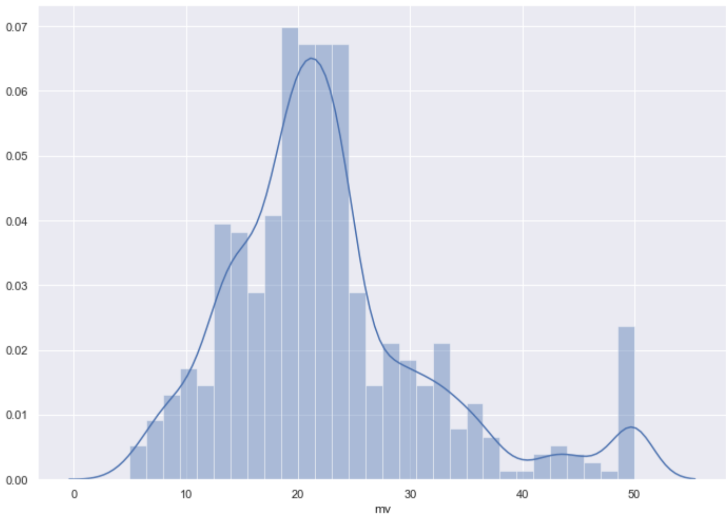
```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'chas'}>,
        <AxesSubplot:title={'center':'crim'}>,
        <AxesSubplot:title={'center':'dis'}>],
       [<AxesSubplot:title={'center':'indus'}
>,
        <AxesSubplot:title={'center':'lstat'}
>,
        <AxesSubplot:title={'center':'mv'}>,
        <AxesSubplot:title={'center':'nox'}>],
       [<AxesSubplot:title={'center':'ptrati
o'}>,
        <AxesSubplot:title={'center':'rad'}>,
        <AxesSubplot:title={'center':'rooms'}
>,
        <AxesSubplot:title={'center':'tax'}>],
       [<AxesSubplot:title={'center':'zn'}>, <
AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>]], dtyp
e=object)
```



Did not have to scale median value of house variable

In [95]:

```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.distplot(boston_Target['mv'], bins=30)
plt.show()
```



See data scaled and transformed and perform tradition summary statistics; move target variable to front

In [100]:

```
cols = boston_df3.columns.tolist()
cols = cols[-1:] + cols[:-1]
boston_df4=boston_df3[cols]
boston_df4.describe(include="all")
```

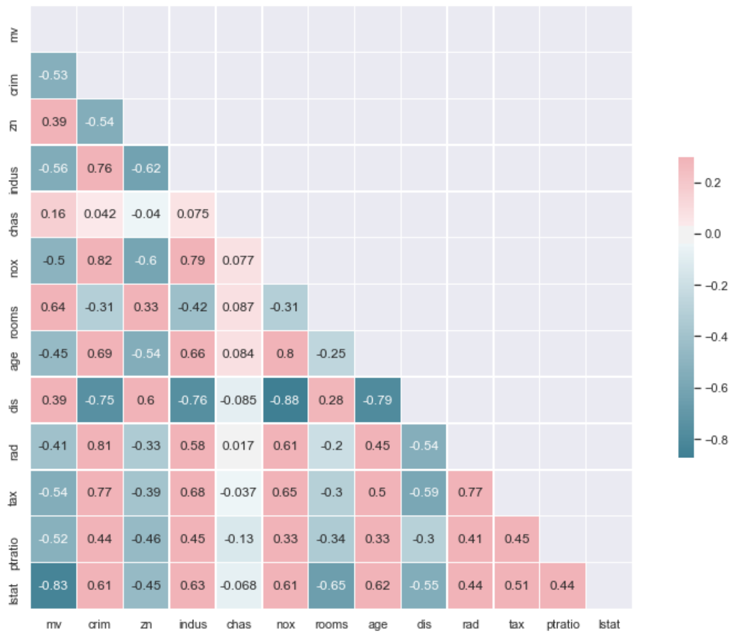
Out[100]:

	mv	crim	zn	indus	cl
count	506.000000	506.000000	506.000000	506.000000	506.000
mean	0.566816	0.518606	0.261061	0.562812	0.069
std	0.184926	0.247164	0.435430	0.232825	0.253
min	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.469326	0.319026	0.000000	0.378568	0.000
50%	0.567460	0.476717	0.000000	0.559586	0.000
75%	0.644398	0.771394	0.967068	0.796857	0.000
max	1.000000	1.000000	1.000000	1.000000	1.000

Make correlation heat map

In [101]:

```
#check correlations
plt.figure(figsize=(15,10))
corr=boston_df4.corr(method='pearson')
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(corr, mask=mask, cmap=sns.diverging_palette(220
            square=True, linewidths=.5, cbar_kws={"shrink":
plt.show()
```



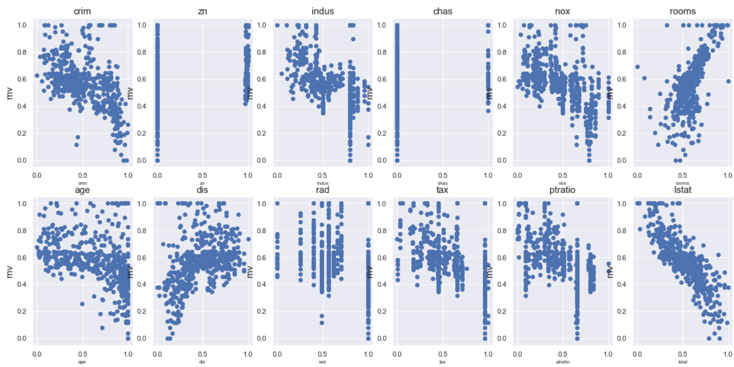
In [98]:

```
boston_df6=boston_df4.copy()
boston_df5 = boston_df4
```

See features scaled and see their dot graph

In [99]:

```
features = boston_df5.drop('mv', 1).columns
target = boston_df5['mv']
plt.figure(figsize=(20,20))
for index, feature_name in enumerate(features):
    plt.subplot(4,len(features)/2, index+1)
    plt.scatter(boston_df5[feature_name], target)
    plt.title(feature_name, fontsize=15)
    plt.xlabel(feature_name, fontsize=8)
    plt.ylabel('mv', fontsize=15)
```



Drop taget variable since we already have it save in it's own df

In [102]:

```
featuresdf = boston_df5.drop('mv', 1)
featuresdf
```

Out[102]:

	crim	zn	indus	chas	nox	rooms
0	0.000000	0.974993	0.206111	0.0	0.500909	0.634505
1	0.163141	0.000000	0.462097	0.0	0.313594	0.606602
2	0.163042	0.000000	0.462097	0.0	0.313594	0.741634
3	0.186434	0.000000	0.195938	0.0	0.278772	0.709345
4	0.294254	0.000000	0.195938	0.0	0.278772	0.735110
...	...	...	...	...	...	...
501	0.280213	0.000000	0.631565	0.0	0.579388	0.637743
502	0.233660	0.000000	0.631565	0.0	0.579388	0.550977
503	0.275852	0.000000	0.631565	0.0	0.579388	0.705515
504	0.360297	0.000000	0.631565	0.0	0.579388	0.673573
505	0.240252	0.000000	0.631565	0.0	0.579388	0.534053

506 rows × 12 columns

In [103]:

```
#remove target and keep all features independent variables
X = featuresdf
```

In [104]:

```
# Save target in Y
Y = boston_Target['mv']
```

Run initial model with Ordinary Least Squares Linear Regression befor splitting.

In [107]:

```
model=sm.OLS(Y, X)
```

In [108]:

```
#save learned algorithm
results=model.fit()
```

Saw R^2 is pretty high means it overfits to existing data

In [109]:

```
print(results.summary())
```

```
                              OLS Regressio
n Results
==============================================
========================================
Dep. Variable:                    mv   R-squa
red (uncentered):                  0.947
Model:                           OLS   Adj. R
-squared (uncentered):             0.946
Method:                Least Squares   F-stat
istic:                             742.2
Date:               Sat, 26 Sep 2020   Prob
(F-statistic):                 5.91e-307
Time:                       15:39:19   Log-Li
kelihood:                        -1587.5
No. Observations:                506   AIC:
3199.
Df Residuals:                    494   BIC:
3250.
Df Model:                         12
Covariance Type:           nonrobust
==============================================
================================
                 coef    std err          t
P>|t|      [0.025      0.975]
----------------------------------------------
--------------------------------
crim           1.7673      2.636      0.670
0.503      -3.413       6.947
zn             1.6570      0.822      2.017
0.044       0.043       3.271
indus          6.5012      2.024      3.213
0.001       2.525      10.477
chas           2.7501      1.027      2.679
0.008       0.733       4.767
nox            2.2669      2.472      0.917
0.360      -2.591       7.125
rooms         38.5654      2.092     18.439
0.000      34.456      42.675
age            3.6543      1.516      2.410
0.016       0.676       6.633
```

```
dis              7.3982      1.869      3.959
0.000       3.726      11.070
rad              4.1958      1.900      2.208
0.028       0.463       7.929
tax             -6.8079      1.651     -4.124
0.000     -10.052      -3.564
ptratio         -4.2149      1.358     -3.103
0.002      -6.884      -1.546
lstat          -17.2472      2.058     -8.381
0.000     -21.290     -13.204
==============================================
================================
Omnibus:                      174.584   Durbin
-Watson:                    0.921
Prob(Omnibus):                  0.000   Jarque
-Bera (JB):              1017.532
Skew:                           1.383   Prob(J
B):                     1.11e-221
Kurtosis:                       9.372   Cond.
No.                          23.4
==============================================
================================

Warnings:
[1] Standard Errors assume that the covariance
matrix of the errors is correctly specified.
```

Train with all features involved

In [311]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, t
```

In [312]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

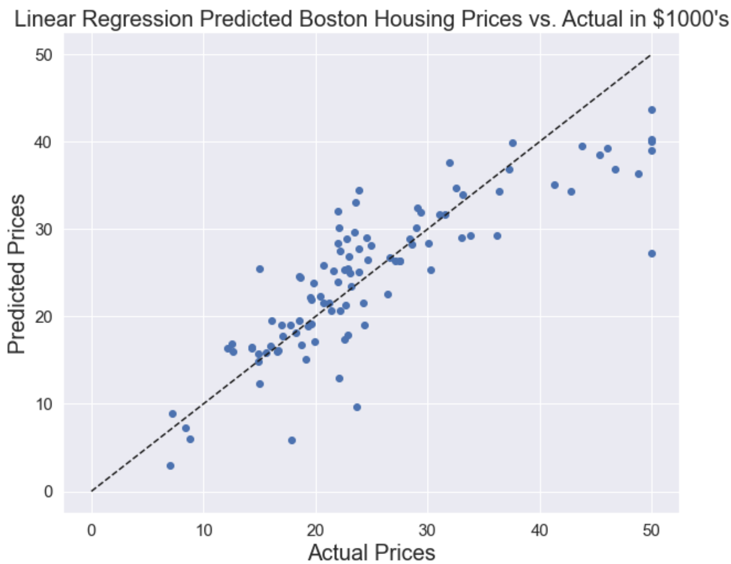
```
(404, 5)
(102, 5)
(404,)
(102,)
```

In [313]:

```
lrm = LinearRegression()

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

# Predict
y_predicted_lrm = lrm.predict(X_test)
```

In [314]:

```
plt.figure(figsize=(10,8))
plt.scatter(y_test, y_predicted_lrm)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.ylabel('Predicted Prices', fontsize=20);
plt.xlabel('Actual Prices', fontsize=20);
plt.title("Linear Regression Predicted Boston Housing Price

plt.rc('xtick', labelsize=15)
plt.rc('ytick', labelsize=15)

plt.show()
```



In [315]:

```
print("Linear Regression R_squared = ",lrm.score(X,Y))
pred= lrm.predict(X_test)
rmse = sqrt(mean_squared_error(pred, y_test))
print('Linear Regression RMSE = ', rmse)
```

```
Linear Regression R_squared =  0.7453805376837
15
Linear Regression RMSE =  5.467651399337476
```

In [316]:

```
print(lrm.coef_)
print(lrm.intercept_)
```

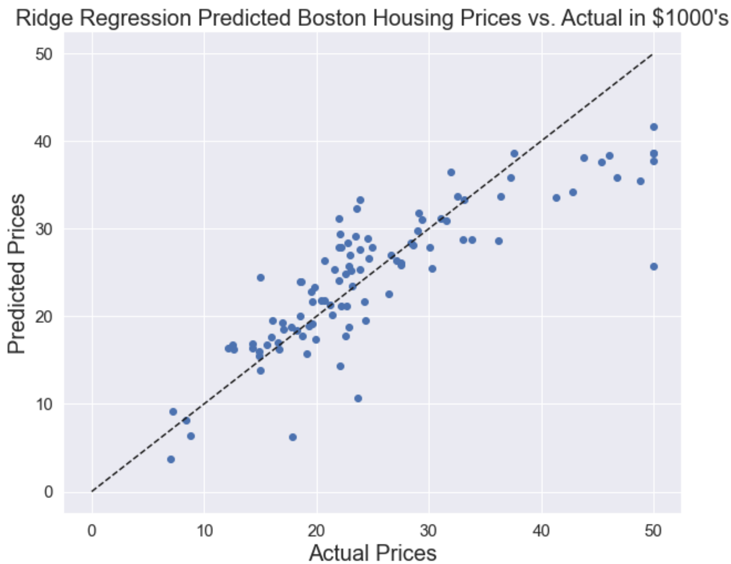
```
[ -6.72941907  14.75574563 -15.55395268  -5.81
938856 -31.24608641]
44.32842703019563
```

In [317]:

```
rrm = Ridge()

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

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

In [318]:

```
plt.figure(figsize=(10,8))
plt.scatter(y_test, y_predicted_rrm)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.ylabel('Predicted Prices', fontsize=20);
plt.xlabel('Actual Prices', fontsize=20);
plt.title("Ridge Regression Predicted Boston Housing Prices

plt.rc('xtick', labelsize=15)
plt.rc('ytick', labelsize=15)

plt.show()
```



In [319]:

```
print("Ridge Regression R_squared = ",rrm.score(X_train,y_t
pred= rrm.predict(X_train)
rmse = sqrt(mean_squared_error(pred, y_train))
print('Ridge Regression RMSE = ', rmse)
```

```
Ridge Regression R_squared =  0.74497668355422
15
Ridge Regression RMSE =  4.429603877876231
```

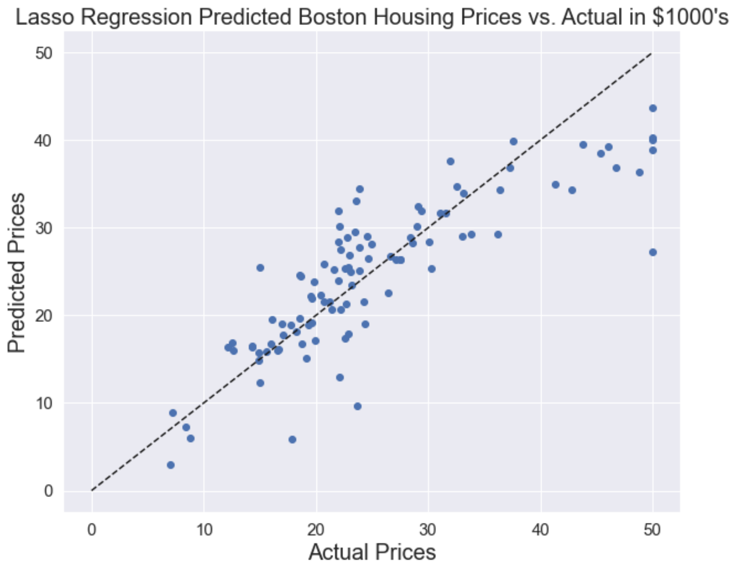
In [320]:

```
print(rrm.coef_)
print(rrm.intercept_)
```

```
[ -5.01097825  14.23731594 -12.01384638  -5.94
549676 -28.53116788]
40.70225192317135
```

In [321]:

```
larm = Lasso(alpha=0.001)

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

# Predict
y_predicted_larm = larm.predict(X_test)

plt.figure(figsize=(10,8))
plt.scatter(y_test,y_predicted_larm)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.ylabel('Predicted Prices', fontsize=20);
plt.xlabel('Actual Prices', fontsize=20);
plt.title("Lasso Regression Predicted Boston Housing Prices

plt.rc('xtick', labelsize=15)
plt.rc('ytick', labelsize=15)

plt.show()
```



In [183]:

```
print("Lasso Regression R_squared = ",larm.score(X_train,y_
pred= larm.predict(X_train)
rmse = sqrt(mean_squared_error(pred, y_train))
print('Lasso Regression RMSE = ', rmse)
```

```
Lasso Regression R_squared =  0.78689794672226
79
Lasso Regression RMSE =  0.0876389329569984
```

In [122]:

```
print(larm.coef_)
print(larm.intercept_)
```

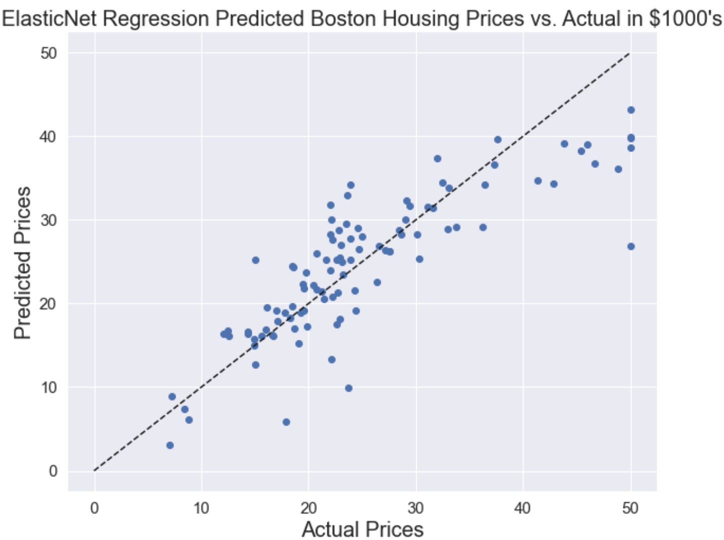
```
[  0.45923068   0.3394121   -3.14057602   2.08
418907  -8.27231794
  11.46389417   2.57625817 -14.8138363    3.68
292213  -5.94886099
  -4.68767432 -30.41608133]
45.83693859549522б
```

In [322]:

```
enrm = ElasticNet(alpha=0.001)

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

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

In [323]:

```
plt.figure(figsize=(10,8))
plt.scatter(y_test, y_predicted_enrm)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.ylabel('Predicted Prices', fontsize=20);
plt.xlabel('Actual Prices', fontsize=20);
plt.title("ElasticNet Regression Predicted Boston Housing P

plt.rc('xtick', labelsize=15)
plt.rc('ytick', labelsize=15)

plt.show()
```



In [192]:

```
print("ElasticNet Regression R_squared = ",enrm.score(X_tra
pred= enrm.predict(X_train)
rmse = sqrt(mean_squared_error(pred, y_train))
print('ElasticNet Regression RMSE = ', rmse)
```

```
ElasticNet Regression R_squared =  0.794496153
744055
ElasticNet Regression RMSE =  0.08606235807161
436
```

In [193]:

```
print(enrm.coef_)
print(enrm.intercept_)
```

```
[-0.          0.00810155 -0.02070781  0.051555
11 -0.09151503  0.15319688
  0.04944581 -0.17060001  0.          -0.137587
8  -0.09957594 -0.6077883 ]
1.0153843628078882
```

In [127]:

```
model_data=boston_df6.values
```

In [128]:

```
# Seed value for random number generators to obtain reprodu
RANDOM_SEED = 1

# The model input data outside of the modeling method calls
names = ['Linear_Regression', 'Ridge_Regression', 'Lasso_Re

# Specify the set of regression models being evaluated (we
regressors = [LinearRegression(fit_intercept = True, normal
              Ridge(alpha = 75, solver = 'cholesky', fit_in
              Lasso(alpha = 0.01, max_iter=10000, tol=0.01,
              ElasticNet(alpha = 0.01, l1_ratio = 0.5, max_
              ]
```

In [129]:

```
mber of cross folds employed for cross-validation


array for storing results
p.zeros((N_FOLDS, len(names)))

itting process
plits = N_FOLDS, shuffle=False, random_state = RANDOM_SEED)

litting process by looking at fold observation counts
 = 0  # Fold count initialized
x, test_index in kf.split(model_data):
old index:', index_for_fold, '------------------------------

e of modeling data for this study has the response variable
ata.shape[1] slices for explanatory variables and 0 is the i
model_data[train_index, 1:model_data.shape[1]]
odel_data[test_index, 1:model_data.shape[1]]
model_data[train_index, 0]
odel_data[test_index, 0]

method = 0  # Method count initialized
reg_model in zip(names, regressors):
del.fit(X_train, y_train)  # Fit on the train set for this f

uate on the test set for this fold
_predict = reg_model.predict(X_test)
ethod_result = sqrt(mean_squared_error(y_test, y_test_predic
ults[index_for_fold, index_for_method] = fold_method_result
for_method += 1

fold += 1

= pd.DataFrame(cv_results)
columns = names

------------------------------------------------------------
 results from ', N_FOLDS, '-fold cross-validation\n',
dardized units (mean 0, standard deviation 1)\n',
d                Root mean-squared error', sep = '')
ts_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 deviat
```

```
ion 1)

Method                      Root mean-squared error
Linear_Regression               0.097808
Ridge_Regression                0.130851
Lasso_Regression                0.110783
ElasticNet_Regression           0.103750
dtype: float64
```

In [130]:

```
cv_results_df.head(10)
```

Out[130]:

	Linear_Regression	Ridge_Regression	Lasso_Regression	Ela
0	0.060632	0.094503	0.067733	
1	0.064451	0.081897	0.061343	
2	0.074057	0.059136	0.044540	
3	0.075747	0.178059	0.141380	
4	0.087041	0.141569	0.117210	
5	0.095296	0.156452	0.145455	
6	0.069371	0.081946	0.080353	
7	0.207578	0.244361	0.209325	
8	0.122983	0.172372	0.160443	
9	0.120922	0.098219	0.080052	

Now we want to same thing with some columns dropped with values higher than 0.05 and variables that involve colinearlity but including nox

In [294]:

```
X = featuresdf
```

In [295]:

```
columns = ['chas', 'crim', 'indus', 'age', 'ptratio', 'zn',
X = X.drop(columns = columns)
```

In [296]:

```
# Split up training and test sets as before
```

In [297]:

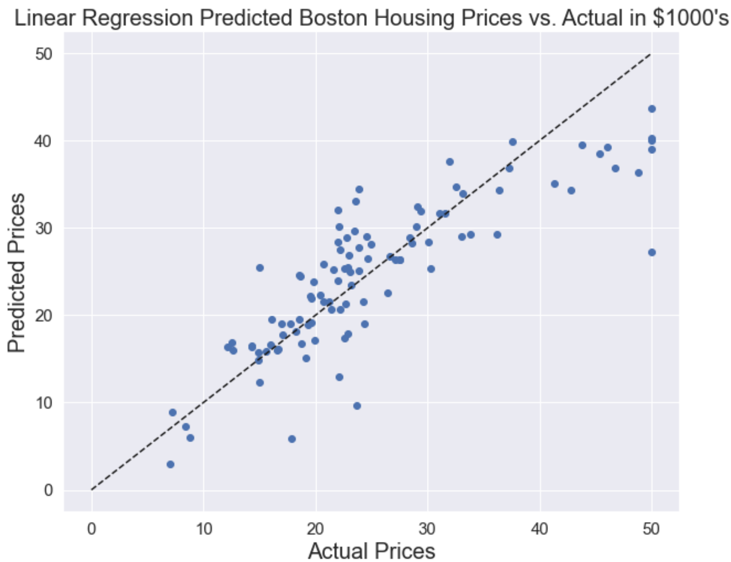
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, t
```

In [298]:

```
lrm = LinearRegression()

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

# Predict
y_predicted_lrm = lrm.predict(X_test)
```

In [299]:

```
plt.figure(figsize=(10,8))
plt.scatter(y_test, y_predicted_lrm)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.ylabel('Predicted Prices', fontsize=20);
plt.xlabel('Actual Prices', fontsize=20);
plt.title("Linear Regression Predicted Boston Housing Price

plt.rc('xtick', labelsize=15)
plt.rc('ytick', labelsize=15)

plt.show()
```



In [300]:

```
print("Linear Regression R_squared = ",lrm.score(X_test,y_t
pred= lrm.predict(X_test)
rmse = sqrt(mean_squared_error(pred, y_test))
print('Linear Regression RMSE = ', rmse)
```

```
Linear Regression R_squared =  0.7122013385206
094
Linear Regression RMSE =  5.467651399337476
```

In [301]:

```
print(lrm.coef_)
print(lrm.intercept_)
```

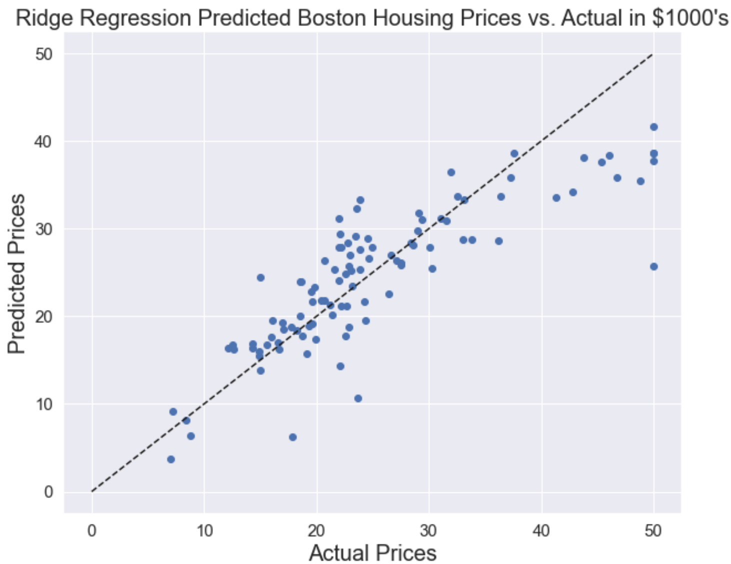
```
[ -6.72941907  14.75574563 -15.55395268  -5.81
938856 -31.24608641]
44.32842703019563
```

In [302]:

```
rrm = Ridge()

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

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

In [303]:

```
plt.figure(figsize=(10,8))
plt.scatter(y_test, y_predicted_rrm)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.ylabel('Predicted Prices', fontsize=20);
plt.xlabel('Actual Prices', fontsize=20);
plt.title("Ridge Regression Predicted Boston Housing Prices

plt.rc('xtick', labelsize=15)
plt.rc('ytick', labelsize=15)

plt.show()
```



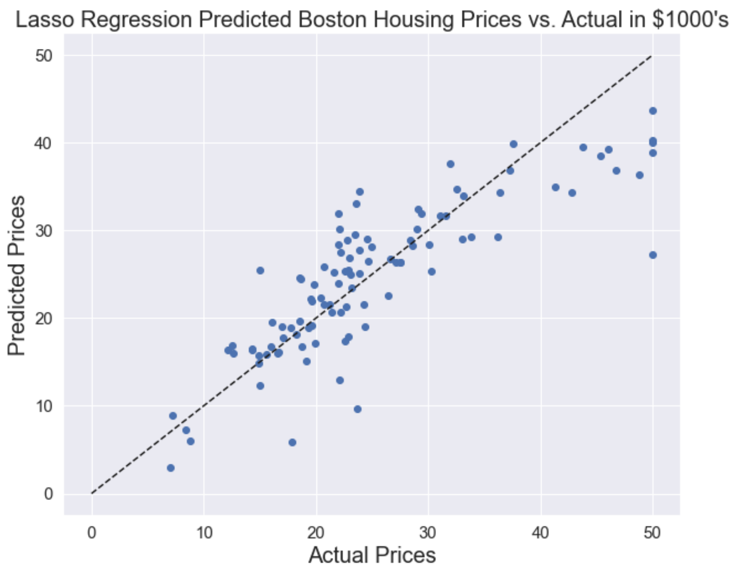
In [304]:

```
print("Ridge Regression R_squared = ",rrm.score(X_test,y_te
pred= rrm.predict(X_test)
rmse = sqrt(mean_squared_error(pred, y_test))
print('Ridge Regression RMSE = ', rmse)
```

```
Ridge Regression R_squared =  0.70288119859201
1
Ridge Regression RMSE =  5.555478866466305
```

In [305]:

```
larm = Lasso(alpha=0.001)

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

# Predict
y_predicted_larm = larm.predict(X_test)

plt.figure(figsize=(10,8))
plt.scatter(y_test,y_predicted_larm)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.ylabel('Predicted Prices', fontsize=20);
plt.xlabel('Actual Prices', fontsize=20);
plt.title("Lasso Regression Predicted Boston Housing Prices

plt.rc('xtick', labelsize=15)
plt.rc('ytick', labelsize=15)

plt.show()
```



In [306]:

```
print("Lasso Regression R_squared = ",larm.score(X_test,y_t
pred= larm.predict(X_test)
rmse = sqrt(mean_squared_error(pred, y_test))
print('Lasso Regression RMSE = ', rmse)
```

```
Lasso Regression R_squared =  0.71196244992596
5
Lasso Regression RMSE =  5.4699201531575685
```

In [307]:

```
print(rrm.coef_)
print(rrm.intercept_)
```

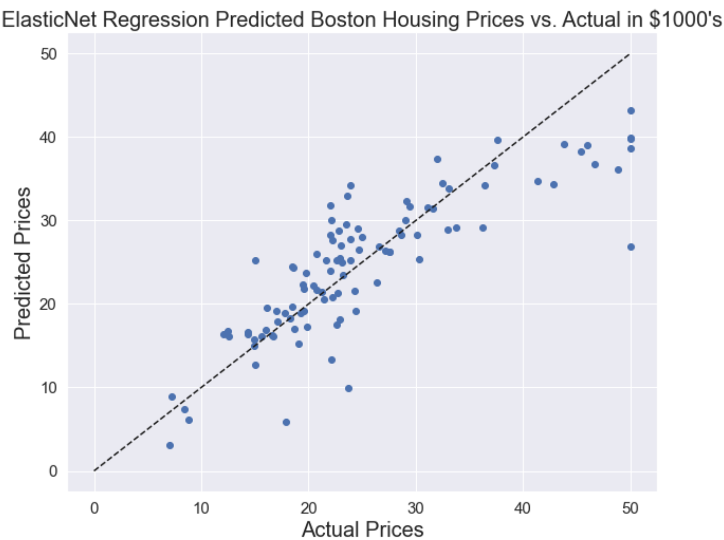
```
[ -5.01097825  14.23731594 -12.01384638  -5.94
549676 -28.53116788]
40.70225192317135
```

In [308]:

```
enrm = ElasticNet(alpha=0.001)

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

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

In [309]:

```
plt.figure(figsize=(10,8))
plt.scatter(y_test, y_predicted_enrm)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.ylabel('Predicted Prices', fontsize=20);
plt.xlabel('Actual Prices', fontsize=20);
plt.title("ElasticNet Regression Predicted Boston Housing P

plt.rc('xtick', labelsize=15)
plt.rc('ytick', labelsize=15)

plt.show()
```



In [310]:

```
print("ElasticNet Regression R_squared = ",enrm.score(X_tes
pred= enrm.predict(X_test)
rmse = sqrt(mean_squared_error(pred, y_test))
print('ElasticNet Regression RMSE = ', rmse)
```

```
ElasticNet Regression R_squared =  0.710867081
683976
ElasticNet Regression RMSE =  5.48031097083872
4
```

In [293]:

```
print(enrm.coef_)
print(enrm.intercept_)
```

```
[ -6.77126923  14.18099508 -14.69867797   2.80
398453  -7.62461321
 -30.70441191]
43.269250791572844
```

In [256]:

```
model_data=boston_df6.values
```

In [257]:

```
# Seed value for random number generators to obtain reprodu
RANDOM_SEED = 1

# The model input data outside of the modeling method calls
names = ['Linear_Regression', 'Ridge_Regression', 'Lasso_Re

# Specify the set of regression models being evaluated (we
regressors = [LinearRegression(fit_intercept = True, normal
              Ridge(alpha = 75, solver = 'cholesky', fit_in
              Lasso(alpha = 0.01, max_iter=10000, tol=0.01,
              ElasticNet(alpha = 0.01, l1_ratio = 0.5, max_
              ]
```

In [258]:

```
Establish number of cross folds employed for cross-validatio
FOLDS = 10

Setup numpy array for storing results
_results = np.zeros((N_FOLDS, len(names)))

Initiate splitting process
 = KFold(n_splits = N_FOLDS, shuffle=False, random_state = RA

Check the splitting process by looking at fold observation co
dex_for_fold = 0  # Fold count initialized
r 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 respons
so 1:model_data.shape[1] slices for explanatory variables and
  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
  for name, reg_model in zip(names, regressors):
      reg_model.fit(X_train, y_train)  # Fit on the train se

      # 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_
      cv_results[index_for_fold, index_for_method] = fold_met
      index_for_method += 1

  index_for_fold += 1

_results_df = pd.DataFrame(cv_results)
_results_df.columns = names

int('\n-----------------------------------------------------
int('Average results from ', N_FOLDS, '-fold cross-validatio
    'in standardized units (mean 0, standard deviation 1)\n',
    '\nMethod               Root mean-squared error', sep =
int(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 deviat
```

```
ion 1)

Method                    Root mean-squared error
Linear_Regression             0.097808
Ridge_Regression              0.130851
Lasso_Regression              0.110783
ElasticNet_Regression         0.103750
dtype: float64
```