Wanat_Assignment_3_final

July 14, 2019

In [1]:
```
# Boston Housing Study (Python)
# using data from the Boston Housing Study case
# as described in "Marketing Data Science: Modeling Techniques
# for Predictive Analytics with R and Python" (Miller 2015)

# Here we use data from the Boston Housing Study to evaluate
# regression modeling methods within a cross-validation design.

# program revised by Thomas W. Milller (2017/09/29)

# Scikit Learn documentation for this assignment:
# http://scikit-learn.org/stable/modules/model_evaluation.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.model_selection.KFold.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.linear_model.LinearRegression.html
# http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.linear_model.Ridge.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.linear_model.Lasso.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.linear_model.ElasticNet.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.metrics.r2_score.html

# Textbook reference materials:
# Geron, A. 2017. Hands-On Machine Learning with Scikit-Learn
# and TensorFlow. Sebastopal, Calif.: O'Reilly. Chapter 3 Training Models
# has sections covering linear regression, polynomial regression,
# and regularized linear models. Sample code from the book is
# available on GitHub at https://github.com/ageron/handson-ml

# prepare for Python version 3x features and functions
# comment out for Python 3.x execution
# from __future__ import division, print_function
# from future_builtins import ascii, filter, hex, map, oct, zip
```

```
# seed value for random number generators to obtain reproducible results
RANDOM_SEED = 1

# although we standardize X and y variables on input,
# we will fit the intercept term in the models
# Expect fitted values to be close to zero
SET_FIT_INTERCEPT = True

# import base packages into the namespace for this program
import numpy as np
import pandas as pd

# modeling routines from Scikit Learn packages
import sklearn.linear_model
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt  # for root mean-squared error calculation
import matplotlib
import matplotlib.pyplot as plt  # static plotting
import seaborn as sns  # pretty plotting, including heat map
from sklearn.model_selection import train_test_split
```

In [2]:
```
# correlation heat map setup for seaborn
def corr_chart(df_corr):
    corr=df_corr.corr()
    #screen top half to get a triangle
    top = np.zeros_like(corr, dtype=np.bool)
    top[np.triu_indices_from(top)] = True
    fig=plt.figure()
    fig, ax = plt.subplots(figsize=(12,12))
    sns.heatmap(corr, mask=top, cmap='coolwarm',
        center = 0, square=True,
        linewidths=.5, cbar_kws={'shrink':.5},
        annot = True, annot_kws={'size': 9}, fmt = '.3f')
    plt.xticks(rotation=45) # rotate variable labels on columns (x axis)
    plt.yticks(rotation=0) # use horizontal variable labels on rows (y axis)
    plt.title('Correlation Heat Map')
    plt.savefig('plot-corr-map.pdf',
        bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
        orientation='portrait', papertype=None, format=None,
        transparent=True, pad_inches=0.25, frameon=None)

np.set_printoptions(precision=3)
```

In [3]:
```
# read data for the Boston Housing Study
# creating data frame restdata
boston_input = pd.read_csv('boston.csv')
```

```python
# check the pandas DataFrame object boston_input
print('\nboston DataFrame (first and last five rows):')
display(boston_input.head())
display(boston_input.tail())
```

```
boston DataFrame (first and last five rows):
```

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

	tax	ptratio	lstat	mv
0	296	15.3	4.98	24.0
1	242	17.8	9.14	21.6
2	242	17.8	4.03	34.7
3	222	18.7	2.94	33.4
4	222	18.7	5.33	36.2

	neighborhood	crim	zn	indus	chas	nox	rooms	age	dis	rad	\
501	Winthrop	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	
502	Winthrop	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	
503	Winthrop	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	
504	Winthrop	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	
505	Winthrop	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	

	tax	ptratio	lstat	mv
501	273	21.0	9.67	22.4
502	273	21.0	9.08	20.6
503	273	21.0	5.64	23.9
504	273	21.0	6.48	22.0
505	273	21.0	7.88	19.0

In [4]:
```python
print('\nGeneral description of the boston_input DataFrame:')
print(boston_input.info())
```

```
General description of the boston_input DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
neighborhood    506 non-null object
```

```
crim            506 non-null float64
zn              506 non-null float64
indus           506 non-null float64
chas            506 non-null int64
nox             506 non-null float64
rooms           506 non-null float64
age             506 non-null float64
dis             506 non-null float64
rad             506 non-null int64
tax             506 non-null int64
ptratio         506 non-null float64
lstat           506 non-null float64
mv              506 non-null float64
dtypes: float64(10), int64(3), object(1)
memory usage: 55.4+ KB
None
```

```
In [5]: # drop neighborhood from the data being considered
        boston = boston_input.drop('neighborhood', 1)
        print('\nGeneral description of the boston DataFrame:')
        print(boston.info())
```

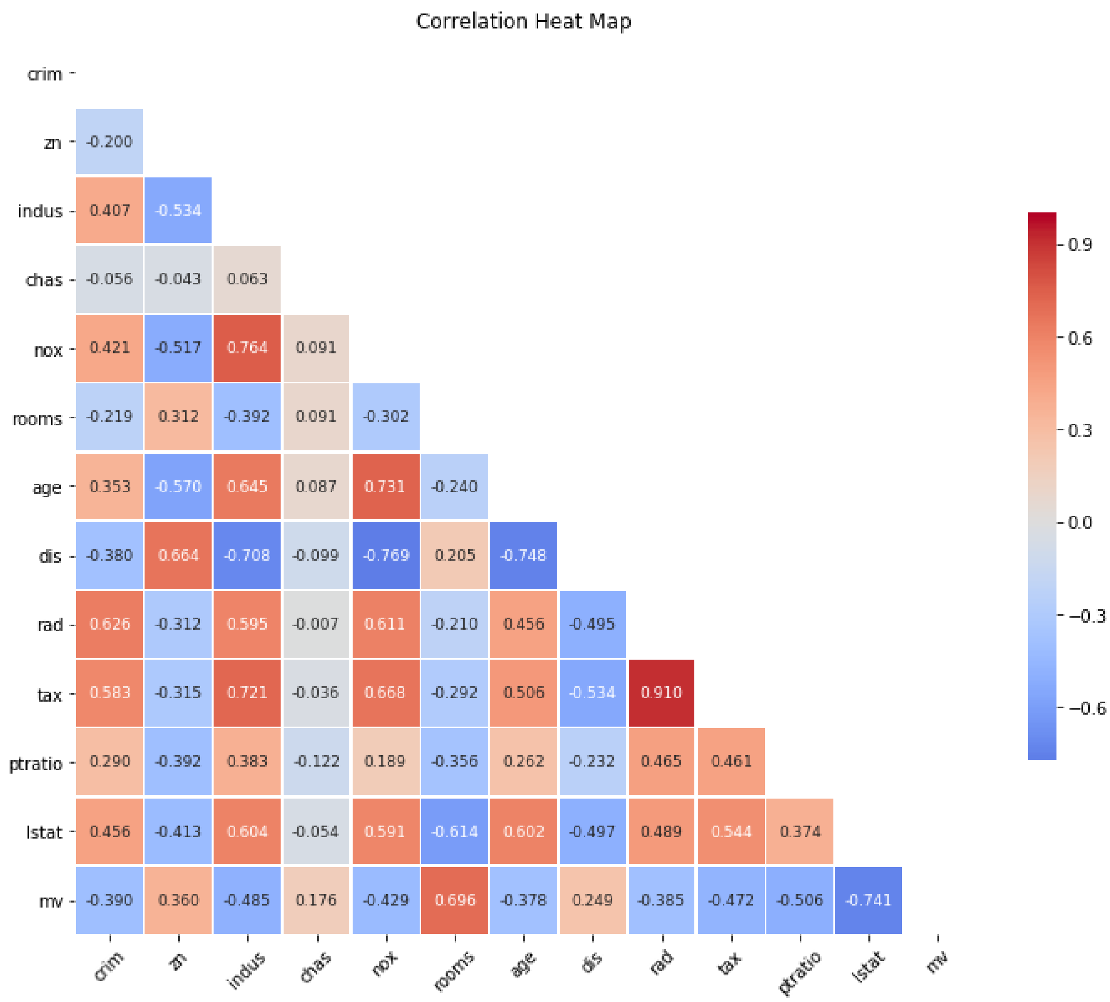
```
General description of the boston DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
crim       506 non-null float64
zn         506 non-null float64
indus      506 non-null float64
chas       506 non-null int64
nox        506 non-null float64
rooms      506 non-null float64
age        506 non-null float64
dis        506 non-null float64
rad        506 non-null int64
tax        506 non-null int64
ptratio    506 non-null float64
lstat      506 non-null float64
mv         506 non-null float64
dtypes: float64(10), int64(3)
memory usage: 51.5 KB
None
```

```
In [6]: print('\nDescriptive statistics of the boston DataFrame:')
        print(boston.describe())
```

```
Descriptive statistics of the boston DataFrame:
             crim          zn       indus        chas         nox       rooms  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677082   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              age         dis         rad         tax     ptratio       lstat  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534   12.653063
std     28.148861    2.105710    8.707259  168.537116    2.164946    7.141062
min      2.900000    1.129600    1.000000  187.000000   12.600000    1.730000
25%     45.025000    2.100175    4.000000  279.000000   17.400000    6.950000
50%     77.500000    3.207450    5.000000  330.000000   19.050000   11.360000
75%     94.075000    5.188425   24.000000  666.000000   20.200000   16.955000
max    100.000000   12.126500   24.000000  711.000000   22.000000   37.970000

               mv
count  506.000000
mean    22.528854
std      9.182176
min      5.000000
25%     17.025000
50%     21.200000
75%     25.000000
max     50.000000
```

```
In [7]: corr_chart(boston)
```

```
<Figure size 432x288 with 0 Axes>
```



In [8]:
```python
# set up preliminary data for data for fitting the models
# the first column is the median housing value response
# the remaining columns are the explanatory variables
prelim_model_data = np.array([boston.mv,\
    boston.crim,\
    boston.zn,\
    boston.indus,\
    boston.chas,\
    boston.nox,\
    boston.rooms,\
    boston.age,\
    boston.dis,\
    boston.rad,\
    boston.tax,\
    boston.ptratio,\
    boston.lstat]).T
```

```
In [9]: # dimensions of the polynomial model X input and y response
        # preliminary data before standardization
        print('\nData dimensions:', prelim_model_data.shape)
```

```

Data dimensions: (506, 13)
```

```
In [10]: # standard scores for the columns... along axis 0
         from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         print(scaler.fit(prelim_model_data))
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
In [11]: # show standardization constants being employed
         print(scaler.mean_)
         print(scaler.scale_)
```

```
[2.253e+01 3.614e+00 1.136e+01 1.114e+01 6.917e-02 5.547e-01 6.285e+00
 6.857e+01 3.795e+00 9.549e+00 4.082e+02 1.846e+01 1.265e+01]
[9.173e+00 8.593e+00 2.330e+01 6.854e+00 2.537e-01 1.158e-01 7.019e-01
 2.812e+01 2.104e+00 8.699e+00 1.684e+02 2.163e+00 7.134e+00]
```

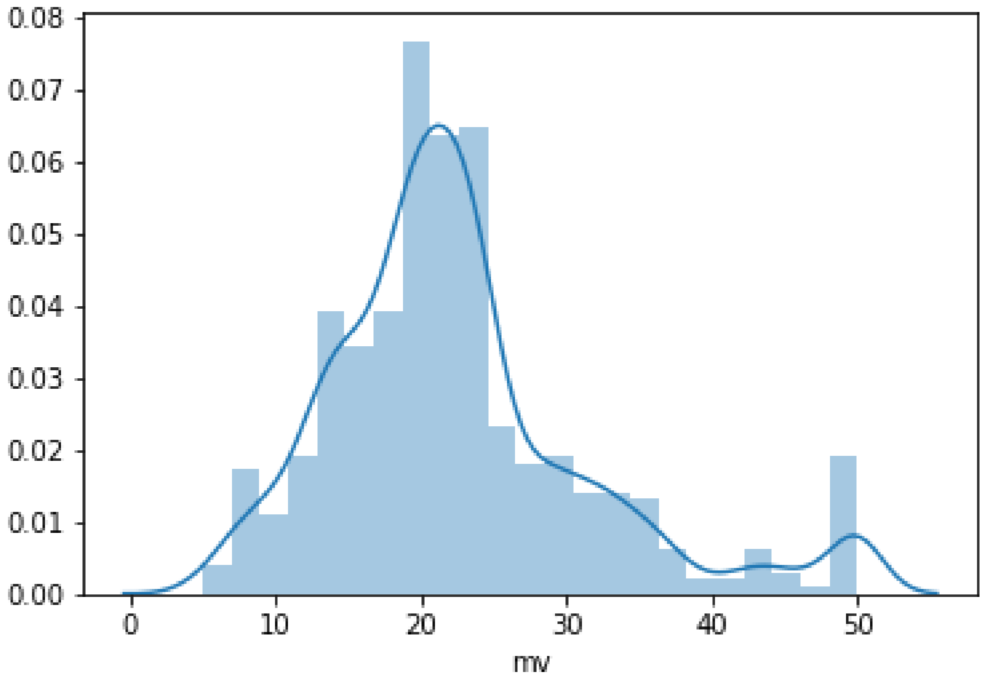
```
In [12]: # the model data will be standardized form of preliminary model data
         model_data = scaler.fit_transform(prelim_model_data)

         # dimensions of the polynomial model X input and y response
         # all in standardized units of measure
         print('\nDimensions for model_data:', model_data.shape)
```

```

Dimensions for model_data: (506, 13)
```

```
In [13]: sns.distplot(boston.mv)
```

```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1164a8cc0>
```



In [14]:
```
#split data and response

boston_X = boston.drop('mv', axis=1)
boston_y = boston.mv.copy()
```

In [15]:
```
boston_X.head()
```

Out[15]:
```
      crim    zn  indus  chas    nox  rooms   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

   lstat
0   4.98
1   9.14
2   4.03
3   2.94
4   5.33
```

In [16]:
```
boston_y.head()
```

Out[16]:
```
0    24.0
1    21.6
```

```
2    34.7
3    33.4
4    36.2
Name: mv, dtype: float64
```

```
In [17]: X_train, X_test, y_train, y_test = train_test_split(boston_X, boston_y, random_state=
```

0.1 Linear Regression

```
In [18]: from sklearn.linear_model import LinearRegression


         lin_reg = LinearRegression()
         lr = lin_reg.fit(X_train, y_train)
         lr_intercept = lr.intercept_
         lr_coef = lr.coef_
         print('\n----------------------------')
         print('Linear Regression\n')
         print('intercept:', lr_intercept)
         print('coefficients:', lr_coef)
```

```
----------------------------
Linear Regression

intercept: 48.18770627941787
coefficients: [-1.195e-01  5.937e-02  3.721e-02  2.529e+00 -2.183e+01  2.780e+00
  7.906e-03 -1.524e+00  2.907e-01 -1.136e-02 -9.388e-01 -5.818e-01]
```

```
In [19]: print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
         print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
```

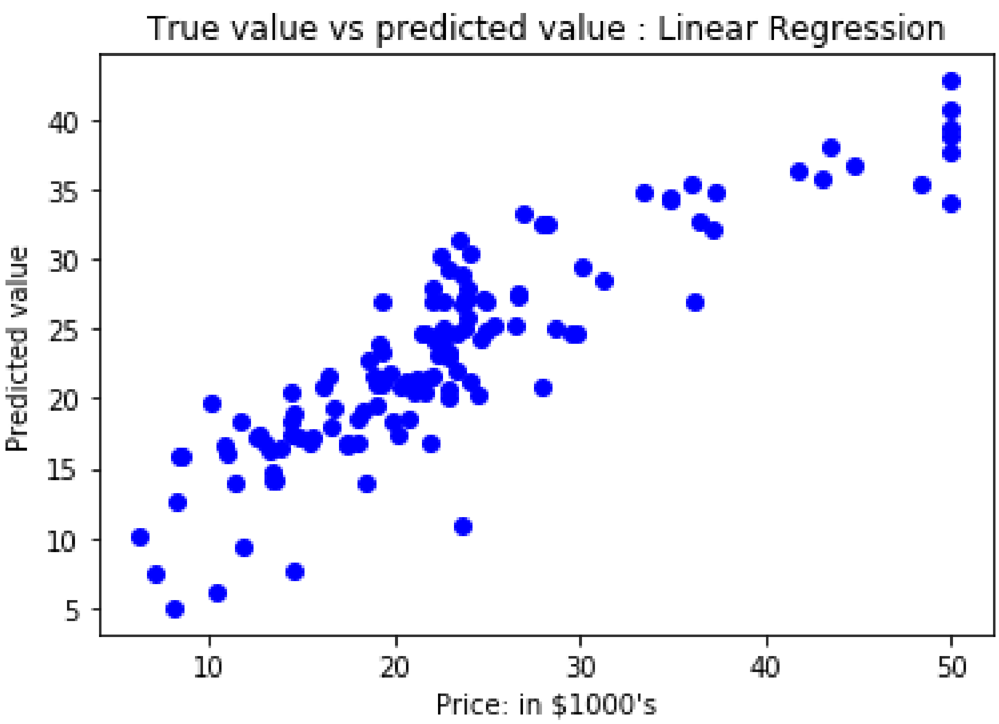
```
Training set score: 0.72
Test set score: 0.77
```

```
In [20]: from sklearn.model_selection import cross_val_score
         scores = cross_val_score(lin_reg, boston_X, boston_y,
                                  scoring="neg_mean_squared_error", cv=10)
         lin_rmse_scores = np.sqrt(-scores)

In [21]: def display_scores(scores):
             print("Scores:", scores)
             print("Mean:", scores.mean())
             print("Standard deviation:", scores.std())

         print('RMSE Linear Regression')
         display_scores(lin_rmse_scores)
```

```
RMSE Linear Regression
Scores: [ 2.826  3.806  4.026  5.983  5.704  4.55   3.154 12.3    6.143  3.056]
Mean: 5.154728385946134
Standard deviation: 2.6514954341431514
```

```
In [22]: # predicting the test set results
         y_pred_lr = lin_reg.predict(X_test)
```

```
In [23]: # Plotting Scatter graph to show the prediction
         # results - 'ytrue' value vs 'y_pred' value
         plt.scatter(y_test, y_pred_lr, c = 'blue')
         plt.xlabel("Price: in $1000's")
         plt.ylabel("Predicted value")
         plt.title("True value vs predicted value : Linear Regression")
         plt.savefig('true_vs_predicted_LR.pdf')
         plt.show()
```



0.2 Ridge Regression

```
In [24]: from sklearn.linear_model import Ridge

         # Linear least squares with l2 regularization.
         # alpha = Regularization strength; must be a positive float.
```

```
# Regularization improves the conditioning of the problem
# and reduces the variance of the estimates.
# Larger values of alpha specify stronger regularization.

ridge = Ridge()
rr = ridge.fit(X_train, y_train)
rr_intercept = ridge.intercept_
rr_coef = ridge.coef_

print('\n----------------------------')
print('Ridge Regression\n')
print('intercept:', rr_intercept)
print('coefficients:', rr_coef)
```

```
----------------------------
Ridge Regression

intercept: 41.23001324745856
coefficients: [-1.149e-01  6.041e-02 -9.447e-03  2.277e+00 -1.182e+01  2.869e+00
 -3.949e-04 -1.386e+00  2.698e-01 -1.250e-02 -8.179e-01 -5.947e-01]
```

```
In [25]: print('Ridge alpha = 1')
         print("Training set score: {:.2f}".format(ridge.score(X_train, y_train)))
         print("Test set score: {:.2f}".format(ridge.score(X_test, y_test)))
```

```
Ridge alpha = 1
Training set score: 0.72
Test set score: 0.77
```

```
In [26]: ridge_scores = cross_val_score(ridge, boston_X, boston_y,
                                   scoring="neg_mean_squared_error", cv=10)
         ridge01_rmse_scores = np.sqrt(-ridge_scores)
         print('RMSE Ridge Regression alpha = 1')
         display_scores(ridge01_rmse_scores)
```

```
RMSE Ridge Regression alpha = 1
Scores: [ 2.846  3.584  3.529  6.117  5.509  4.401  3.081 12.268  6.305  3.207]
Mean: 5.0848231944069475
Standard deviation: 2.683888647423676
```

```
In [27]: ridge10 = Ridge(alpha=10).fit(X_train, y_train)
         print('Ridge alpha = 10')
         print("Training set score: {:.2f}".format(ridge10.score(X_train, y_train)))
         print("Test set score: {:.2f}".format(ridge10.score(X_test, y_test)))
```

```
Ridge alpha = 10
Training set score: 0.70
Test set score: 0.77
```

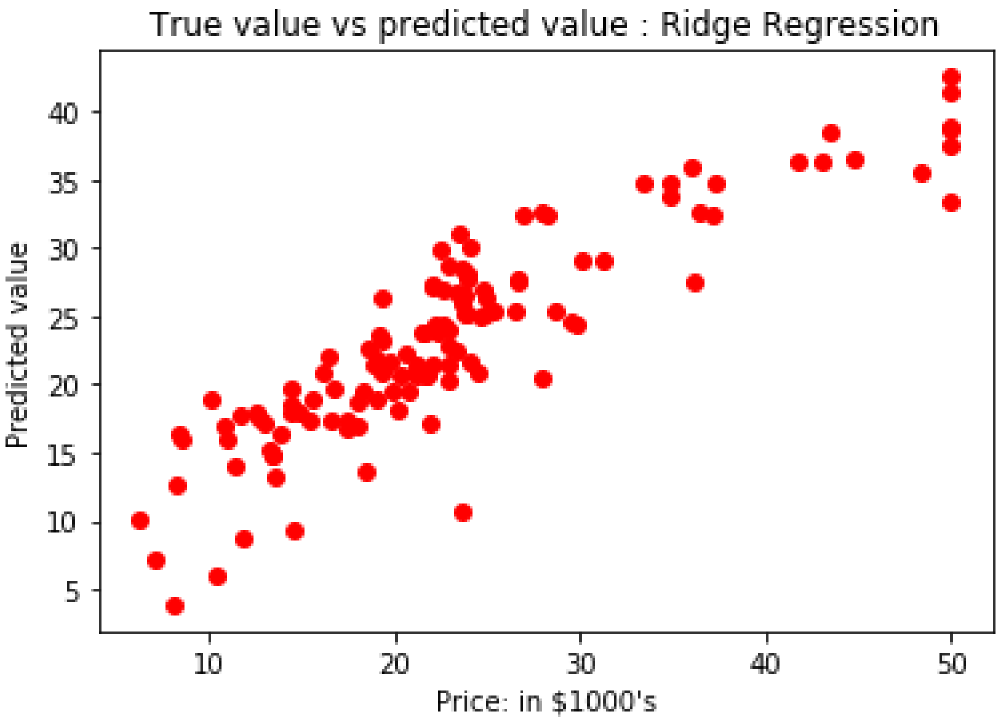
```
In [28]: ridge01 = Ridge(alpha=0.1).fit(X_train, y_train)
         print('Ridge alpha = 0.1')
         print("Training set score: {:.2f}".format(ridge01.score(X_train, y_train)))
         print("Test set score: {:.2f}".format(ridge01.score(X_test, y_test)))
```

```
Ridge alpha = 0.1
Training set score: 0.72
Test set score: 0.77
```

```
In [29]: ridge_scores = cross_val_score(ridge01, boston_X, boston_y,
                                       scoring="neg_mean_squared_error", cv=10)
         ridge01_rmse_scores = np.sqrt(-ridge_scores)
         print('RMSE Ridge Regression alpha = 0.01')
         display_scores(ridge01_rmse_scores)
```

```
RMSE Ridge Regression alpha = 0.01
Scores: [ 2.826  3.768  3.919  6.002  5.67   4.523  3.135 12.297  6.169  3.074]
Mean: 5.138375588249978
Standard deviation: 2.658992425555945
```

```
In [30]: # predicting the test set results
         y_pred_rr = ridge.predict(X_test)

In [31]: # Plotting Scatter graph to show the prediction
         # results - 'ytrue' value vs 'y_pred' value
         plt.scatter(y_test, y_pred_rr, c = 'red')
         plt.xlabel("Price: in $1000's")
         plt.ylabel("Predicted value")
         plt.title("True value vs predicted value : Ridge Regression")
         plt.savefig('true_vs_predicted_RR.pdf')
         plt.show()
```



0.3 Lasso

```
In [32]: from sklearn.linear_model import Lasso

         # Linear Model trained with L1 prior as regularizer
         # alpha = Regularization strength, constant that multiplies the L1 term.

         lasso = Lasso()
         lasso.fit(X_train, y_train)
         lasso.intercept_, lasso.coef_

         lasso_intercept = lasso.intercept_
         lasso_coef = lasso.coef_

         print('\n----------------------------')
         print('Lasso with default alpha = 1.0\n')
         print('intercept:', lasso_intercept)
         print('coefficients:', lasso_coef)
```

```
----------------------------
Lasso with default alpha = 1.0

intercept: 48.83289146303929
```

```
coefficients: [-0.073  0.057 -0.     0.    -0.     0.     0.023 -0.587  0.23  -0.014
 -0.67  -0.812]
```

```
In [33]: print('Lasso alpha = 1, max_iter = 1000')
         print("Training set score: {:.2f}".format(lasso.score(X_train, y_train)))
         print("Test set score: {:.2f}".format(lasso.score(X_test, y_test)))
         print("Number of features used:", np.sum(lasso.coef_ != 0))
```

```
Lasso alpha = 1, max_iter = 1000
Training set score: 0.65
Test set score: 0.66
Number of features used: 8
```

```
In [34]: lasso_scores = cross_val_score(lasso, boston_X, boston_y,
                                  scoring="neg_mean_squared_error", cv=10)
         lasso_rmse_scores = np.sqrt(-lasso_scores)
         print('RMSE Lasso alpha = 1, max_iter = 1000')
         display_scores(lasso_rmse_scores)
```

```
RMSE Lasso alpha = 1, max_iter = 1000
Scores: [ 3.359  4.247  3.411  7.813  6.856  6.478  4.231 10.109  5.276  3.452]
Mean: 5.523163553242023
Standard deviation: 2.1390100357462694
```

```
In [35]: # we increase the default setting of "max_iter",
         # otherwise the model would warn us that we should increase max_iter.
         # regularization parameter, alpha, that controls
         # how strongly coefficients are pushed toward zero.
         # To reduce underfitting, lets try decreasing alpha.

         lasso001 = Lasso(alpha=0.01, max_iter=100000).fit(X_train, y_train)
         print('Lasso alpha = 0.01, max_iter = 100000')
         print("Training set score: {:.2f}".format(lasso001.score(X_train, y_train)))
         print("Test set score: {:.2f}".format(lasso001.score(X_test, y_test)))
         print("Number of features used:", np.sum(lasso001.coef_ != 0))
```

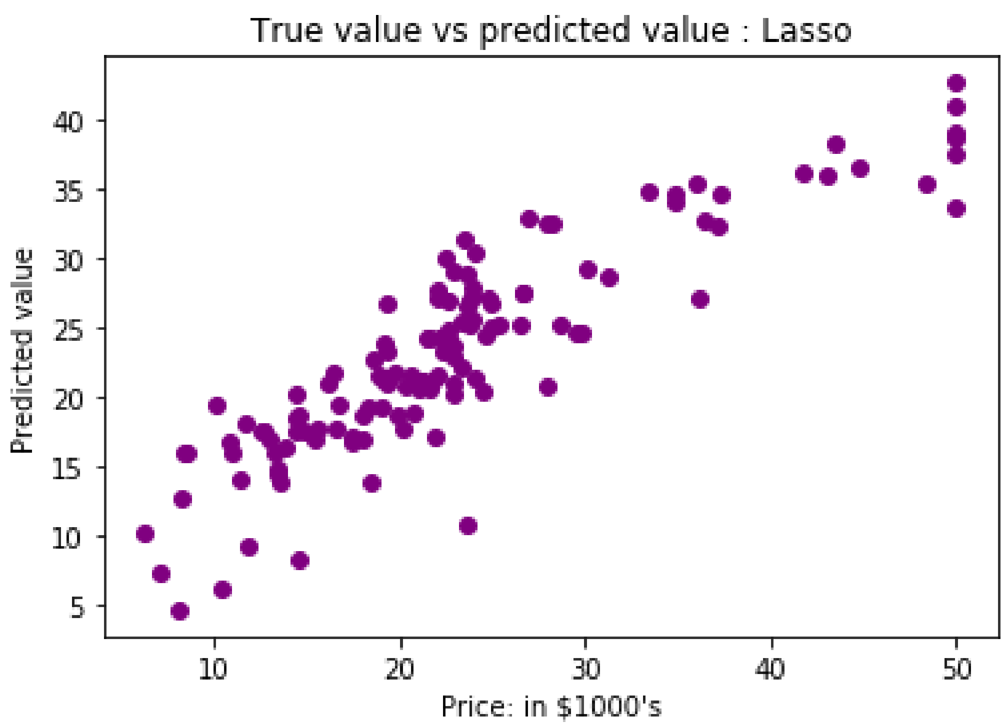
```
Lasso alpha = 0.01, max_iter = 100000
Training set score: 0.72
Test set score: 0.77
Number of features used: 12
```

```
In [36]: lasso001_scores = cross_val_score(lasso001, boston_X, boston_y,
                                  scoring="neg_mean_squared_error", cv=10)
         lasso001_rmse_scores = np.sqrt(-lasso001_scores)
         print('RMSE Lasso alpha = 0.01, max_iter = 100000')
         display_scores(lasso001_rmse_scores)
```

```
RMSE Lasso alpha = 0.01, max_iter = 100000
Scores: [ 2.827  3.729  3.789  6.051  5.606  4.492  3.114 12.306  6.231  3.114]
Mean: 5.1259392630655345
Standard deviation: 2.671885020579878
```

```
In [37]: # predicting the test set results
         y_pred_lasso = lasso001.predict(X_test)
```

```
In [38]: # Plotting Scatter graph to show the prediction
         # results - 'ytrue' value vs 'y_pred' value
         plt.scatter(y_test, y_pred_lasso, c = 'purple')
         plt.xlabel("Price: in $1000's")
         plt.ylabel("Predicted value")
         plt.title("True value vs predicted value : Lasso")
         plt.savefig('true_vs_predicted_lasso.pdf')
         plt.show()
```



0.4 Elastic Net

```
In [39]: from sklearn.linear_model import ElasticNet

         # Linear regression with combined L1 and L2 priors as regularizer
         # alpha = Regularization strength, constant that multiplies the penalty terms.
```

```
elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
elastic_net.fit(X_train, y_train)

elastic_net_intercept = elastic_net.intercept_
elastic_net_coef = elastic_net.coef_

print('\n----------------------------')
print('Elastic Net\n')
print('intercept:', elastic_net_intercept)
print('coefficients:', elastic_net_coef)
```

```
----------------------------
Elastic Net

intercept: 37.13744686978241
coefficients: [-0.11   0.063 -0.055  0.912 -0.307  2.443 -0.004 -1.179  0.264 -0.015
 -0.709 -0.649]
```

In [40]:
```
print('alpha=1.0, l1_ratio=0.5')
print("Training set score: {:.2f}".format(elastic_net.score(X_train, y_train)))
print("Test set score: {:.2f}".format(elastic_net.score(X_test, y_test)))
```

```
alpha=1.0, l1_ratio=0.5
Training set score: 0.70
Test set score: 0.76
```

In [41]:
```
elastic_net_scores = cross_val_score(elastic_net, boston_X, boston_y,
                         scoring="neg_mean_squared_error", cv=10)
elastic_net_rmse_scores = np.sqrt(-elastic_net_scores)
print('RMSE Elastic Net alpha=1.0, l1_ratio=0.5')
display_scores(elastic_net_rmse_scores)
```

```
RMSE Elastic Net alpha=1.0, l1_ratio=0.5
Scores: [ 2.985  3.518  3.336  6.524  5.516  4.506  3.313 11.316  6.237  3.554]
Mean: 5.080429131773322
Standard deviation: 2.4115057536950926
```

In [42]:
```
elastic_net_2 = ElasticNet(alpha=0.5, l1_ratio=0.1)
elastic_net_2.fit(X_train, y_train)
print('alpha=0.5, l1_ratio=0.1')
print("Training set score: {:.2f}".format(elastic_net.score(X_train, y_train)))
print("Test set score: {:.2f}".format(elastic_net.score(X_test, y_test)))
```

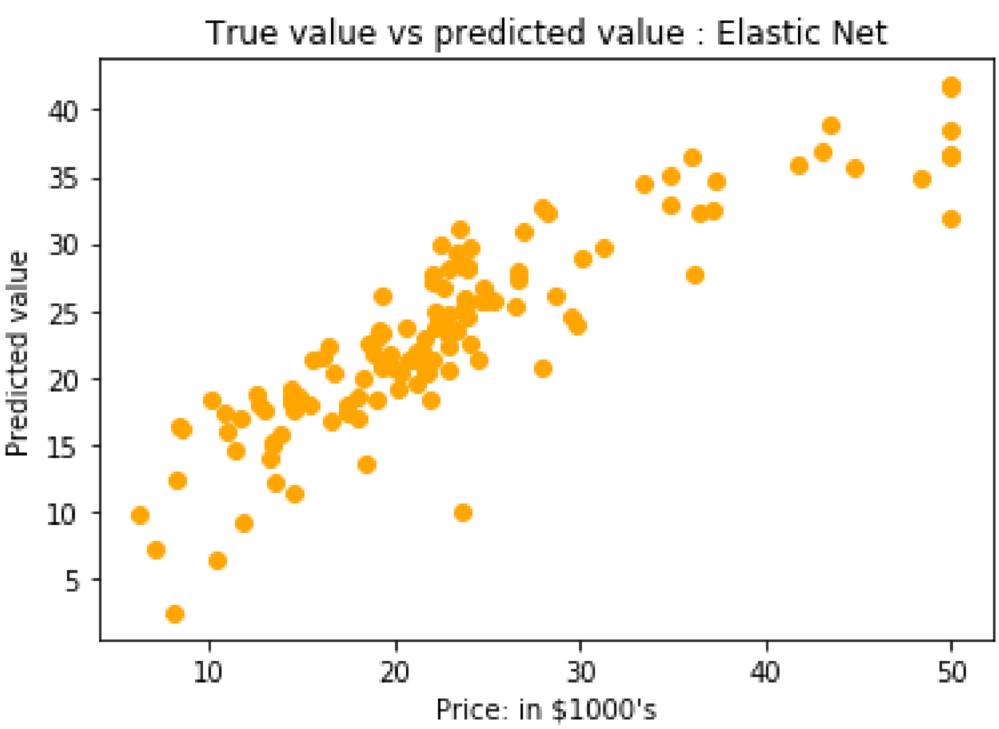
```
alpha=0.5, l1_ratio=0.1
Training set score: 0.70
Test set score: 0.76
```

```
In [43]: elastic_net_2_scores = cross_val_score(elastic_net_2, boston_X, boston_y,
                                  scoring="neg_mean_squared_error", cv=10)
         elastic_net_2_rmse_scores = np.sqrt(-elastic_net_scores)
         print('RMSE Elastic Net alpha=0.5, l1_ratio=0.1')
         display_scores(elastic_net_2_rmse_scores)
```

```
RMSE Elastic Net alpha=0.5, l1_ratio=0.1
Scores: [ 2.985  3.518  3.336  6.524  5.516  4.506  3.313 11.316  6.237  3.554]
Mean: 5.080429131773322
Standard deviation: 2.4115057536950926
```

```
In [44]: # predicting the test set results
         y_pred_en = elastic_net.predict(X_test)
```

```
In [45]: # Plotting Scatter graph to show the prediction
         # results - 'ytrue' value vs 'y_pred' value
         plt.scatter(y_test, y_pred_en, c = 'orange')
         plt.xlabel("Price: in $1000's")
         plt.ylabel("Predicted value")
         plt.title("True value vs predicted value : Elastic Net")
         plt.savefig('true_vs_predicted_EN.pdf')
         plt.show()
```



0.5 Grid Search

In [46]:
```
from sklearn.model_selection import GridSearchCV

# GridSearchCV: specify which hyperparameters you want
# it to experiment with, and what values to try out,
# and it will evaluate all the possible combinations
# of hyperparameter values, using cross-validation


param_grid = [
    {'alpha': [0.1, 0.5, 1, 10, 20], 'l1_ratio': [0.1, 0.25, 0.5, 0.75]},
  ]

elastic_net_gs = ElasticNet()

grid_search = GridSearchCV(elastic_net_gs, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)

grid_search.fit(boston_X, boston_y)
```

Out[46]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'alpha': [0.1, 0.5, 1, 10, 20], 'l1_ratio': [0.1, 0.25, 0.5, 0.75]
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='neg_mean_squared_error', verbose=0)
```

In [47]:
```
print('\n----------------------------')
print('The parameters choosen by Grid Search for Elastic Net:\n ')
grid_search.best_params_
```

```

----------------------------
The parameters choosen by Grid Search for Elastic Net:

```

Out[47]:
```
{'alpha': 0.5, 'l1_ratio': 0.1}
```

In [48]:
```
grid_search.best_estimator_
```

Out[48]:
```
ElasticNet(alpha=0.5, copy_X=True, fit_intercept=True, l1_ratio=0.1,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

```
In [49]: cvres = grid_search.cv_results_

         for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
             print(np.sqrt(-mean_score), params)
```

```
5.6464930839855 {'alpha': 0.1, 'l1_ratio': 0.1}
5.677263610425171 {'alpha': 0.1, 'l1_ratio': 0.25}
5.740279703029267 {'alpha': 0.1, 'l1_ratio': 0.5}
5.818655787231755 {'alpha': 0.1, 'l1_ratio': 0.75}
5.546062243897276 {'alpha': 0.5, 'l1_ratio': 0.1}
5.5740235714285875 {'alpha': 0.5, 'l1_ratio': 0.25}
5.630446396462518 {'alpha': 0.5, 'l1_ratio': 0.5}
5.727868299091872 {'alpha': 0.5, 'l1_ratio': 0.75}
5.613846498566613 {'alpha': 1, 'l1_ratio': 0.1}
5.649877635818467 {'alpha': 1, 'l1_ratio': 0.25}
5.729059741699742 {'alpha': 1, 'l1_ratio': 0.5}
5.832027373152929 {'alpha': 1, 'l1_ratio': 0.75}
6.384233839729841 {'alpha': 10, 'l1_ratio': 0.1}
6.568917497795396 {'alpha': 10, 'l1_ratio': 0.25}
6.684133591183442 {'alpha': 10, 'l1_ratio': 0.5}
6.7331033985792015 {'alpha': 10, 'l1_ratio': 0.75}
6.794337009151011 {'alpha': 20, 'l1_ratio': 0.1}
6.900179452400165 {'alpha': 20, 'l1_ratio': 0.25}
7.0171707864846065 {'alpha': 20, 'l1_ratio': 0.5}
7.190116716832263 {'alpha': 20, 'l1_ratio': 0.75}
```

```
In [50]: elastic_net_try = ElasticNet(alpha=0.5, l1_ratio=0.1, max_iter=1000)
         elastic_net_try.fit(X_train, y_train)

Out[50]: ElasticNet(alpha=0.5, copy_X=True, fit_intercept=True, l1_ratio=0.1,
               max_iter=1000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)

In [51]: elastic_net_try.fit(X_train, y_train)
         print('Elastic Net alpha=0.5, l1_ratio=0.1')
         print("Training set score: {:.2f}".format(elastic_net_try.score(X_train, y_train)))
         print("Test set score: {:.2f}".format(elastic_net_try.score(X_test, y_test)))
```

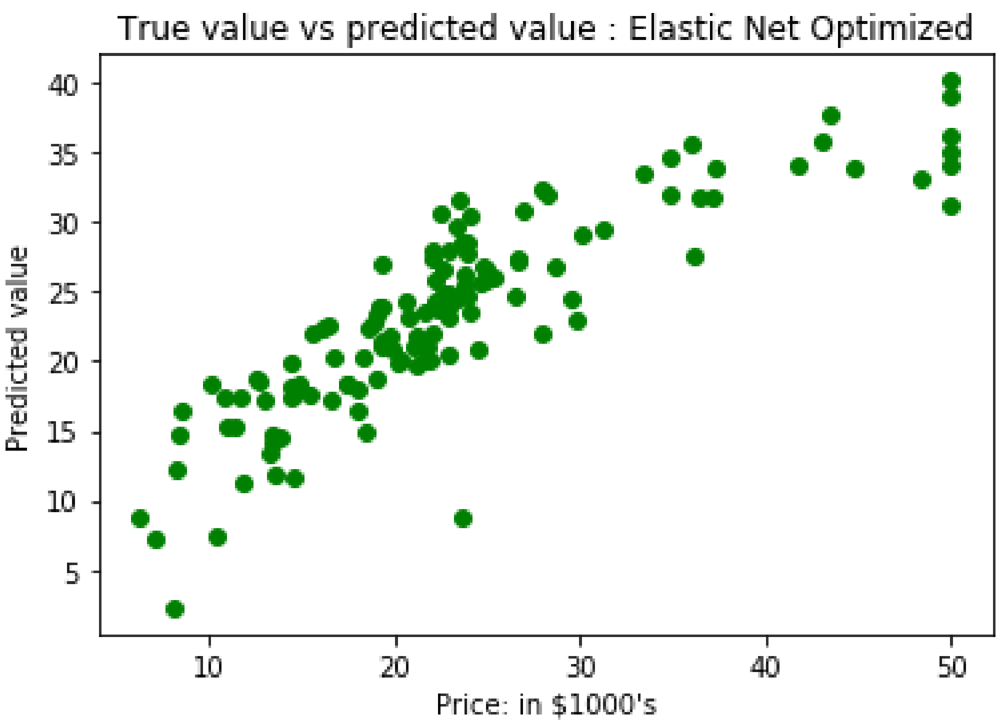
```
Elastic Net alpha=0.5, l1_ratio=0.1
Training set score: 0.68
Test set score: 0.71
```

```
In [52]: elastic_net_try_scores = cross_val_score(elastic_net_try, boston_X, boston_y,
                                  scoring="neg_mean_squared_error", cv=10)
         elastic_net_try_rmse_scores = np.sqrt(-elastic_net_try_scores)
         print('RMSE Elastic Net alpha=0.5, l1_ratio=0.1')
         display_scores(elastic_net_2_rmse_scores)
```

```
RMSE Elastic Net alpha=0.5, l1_ratio=0.1
Scores: [ 2.985  3.518  3.336  6.524  5.516  4.506  3.313 11.316  6.237  3.554]
Mean: 5.080429131773322
Standard deviation: 2.4115057536950926
```

```
In [53]: # predicting the test set results
         y_pred = elastic_net_try.predict(X_test)
```

```
In [54]: # Plotting Scatter graph to show the prediction
         # results - 'ytrue' value vs 'y_pred' value
         plt.scatter(y_test, y_pred, c = 'green')
         plt.xlabel("Price: in $1000's")
         plt.ylabel("Predicted value")
         plt.title("True value vs predicted value : Elastic Net Optimized")
         plt.savefig('true_vs_predicted_EN_optimized.pdf')
         plt.show()
```



0.6 Function to evaluate all algorithms

```
In [55]: # specify the set of classifiers being evaluated
         from sklearn.metrics import mean_squared_error

         names = ["Linear_Regression", "Ridge_Regression",
```

```
         "Lasso", "Elastic_Net"]
classifiers = [LinearRegression(),
              Ridge(),
              Lasso(),
              ElasticNet(alpha=0.1, l1_ratio=0.25)]
```

In [56]:
```
# ------------------------------------------------------------
# specify the k-fold cross-validation design
from sklearn.model_selection import KFold

# ten-fold cross-validation employed here
N_FOLDS = 10

# set up numpy array for storing results
cv_results = np.zeros((N_FOLDS, len(names)))
```

In [57]:
```
# check cv_results array
cv_results
```

Out[57]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [58]:
```
# check names list
names
```

Out[58]:
```
['Linear_Regression', 'Ridge_Regression', 'Lasso', 'Elastic_Net']
```

In [59]:
```
# Instantiate K-Folds cross-validator
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,
          '------------------------------------------')
#   note that 0:model_data.shape[1]-1 slices for explanatory variables
#   and model_data.shape[1]-1 is the index for the response variable
    X_train2 = model_data[train_index, 0:model_data.shape[1]-1]
    X_test2 = model_data[test_index, 0:model_data.shape[1]-1]
    y_train2 = model_data[train_index, model_data.shape[1]-1]
    y_test2 = model_data[test_index, model_data.shape[1]-1]
```

```python
            print('\nShape of input data for this fold:',
                  '\nData Set: (Observations, Variables)')
            print('X_train:', X_train2.shape)
            print('X_test:',X_test2.shape)
            print('y_train:', y_train2.shape)
            print('y_test:',y_test2.shape)

            index_for_method = 0  # initialize
            for name, clf in zip(names, classifiers):
                print('\nClassifier evaluation for:', name)
                print('  Scikit Learn method:', clf)
                clf.fit(X_train2, y_train2)  # fit on the train set for this fold
                # evaluate on the test set for this fold
                y_test_predict = clf.predict(X_test2)
                fold_method_result = mean_squared_error(y_test2, y_test_predict)
                fold_rmse = np.sqrt(fold_method_result)
                print('RMSE:', fold_rmse)
                cv_results[index_for_fold, index_for_method] = fold_rmse
                index_for_method += 1

            index_for_fold += 1
```

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

Shape of input data for this fold:
Data Set: (Observations, Variables)
X_train: (455, 12)
X_test: (51, 12)
y_train: (455,)
y_test: (51,)

Classifier evaluation for: Linear_Regression
  Scikit Learn method: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
RMSE: 0.6153307366878622

Classifier evaluation for: Ridge_Regression
  Scikit Learn method: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
RMSE: 0.6160421022579456

Classifier evaluation for: Lasso
  Scikit Learn method: Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.9313027028555269

Classifier evaluation for: Elastic_Net
```

```
  Scikit Learn method: ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.25,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.6339112262359792

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

Shape of input data for this fold:
Data Set: (Observations, Variables)
X_train: (455, 12)
X_test: (51, 12)
y_train: (455,)
y_test: (51,)

Classifier evaluation for: Linear_Regression
  Scikit Learn method: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
RMSE: 0.3747267699016482

Classifier evaluation for: Ridge_Regression
  Scikit Learn method: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
RMSE: 0.3724873699458746

Classifier evaluation for: Lasso
  Scikit Learn method: Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.7831242969799976

Classifier evaluation for: Elastic_Net
  Scikit Learn method: ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.25,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.3323423970650091

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

Shape of input data for this fold:
Data Set: (Observations, Variables)
X_train: (455, 12)
X_test: (51, 12)
y_train: (455,)
y_test: (51,)

Classifier evaluation for: Linear_Regression
  Scikit Learn method: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
RMSE: 0.7372119896892206
```

```
Classifier evaluation for: Ridge_Regression
  Scikit Learn method: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
RMSE: 0.7361441517989507

Classifier evaluation for: Lasso
  Scikit Learn method: Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 1.1137912300446424

Classifier evaluation for: Elastic_Net
  Scikit Learn method: ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.25,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.718563165173973

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

Shape of input data for this fold:
Data Set: (Observations, Variables)
X_train: (455, 12)
X_test: (51, 12)
y_train: (455,)
y_test: (51,)

Classifier evaluation for: Linear_Regression
  Scikit Learn method: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
RMSE: 0.6535643910371687

Classifier evaluation for: Ridge_Regression
  Scikit Learn method: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
RMSE: 0.6524778781987847

Classifier evaluation for: Lasso
  Scikit Learn method: Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.9651492613503816

Classifier evaluation for: Elastic_Net
  Scikit Learn method: ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.25,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.5656235706545696

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

```
Shape of input data for this fold:
Data Set: (Observations, Variables)
X_train: (455, 12)
X_test: (51, 12)
y_train: (455,)
y_test: (51,)

Classifier evaluation for: Linear_Regression
  Scikit Learn method: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
RMSE: 0.6534559101742444

Classifier evaluation for: Ridge_Regression
  Scikit Learn method: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
RMSE: 0.6528224169003206

Classifier evaluation for: Lasso
  Scikit Learn method: Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.9343170336152345

Classifier evaluation for: Elastic_Net
  Scikit Learn method: ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.25,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.5949392103369477

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

Shape of input data for this fold:
Data Set: (Observations, Variables)
X_train: (455, 12)
X_test: (51, 12)
y_train: (455,)
y_test: (51,)

Classifier evaluation for: Linear_Regression
  Scikit Learn method: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
RMSE: 0.4204278869595973

Classifier evaluation for: Ridge_Regression
  Scikit Learn method: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
RMSE: 0.4194772785083912

Classifier evaluation for: Lasso
```

```
  Scikit Learn method: Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.9521726271573456

Classifier evaluation for: Elastic_Net
  Scikit Learn method: ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.25,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.3973994633664465

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

Shape of input data for this fold:
Data Set: (Observations, Variables)
X_train: (456, 12)
X_test: (50, 12)
y_train: (456,)
y_test: (50,)

Classifier evaluation for: Linear_Regression
  Scikit Learn method: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
RMSE: 0.4044402074544354

Classifier evaluation for: Ridge_Regression
  Scikit Learn method: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
RMSE: 0.4044339063399666

Classifier evaluation for: Lasso
  Scikit Learn method: Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.7600693575647395

Classifier evaluation for: Elastic_Net
  Scikit Learn method: ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.25,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.44973435865315786

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

Shape of input data for this fold:
Data Set: (Observations, Variables)
X_train: (456, 12)
X_test: (50, 12)
y_train: (456,)
```

```
y_test: (50,)

Classifier evaluation for: Linear_Regression
  Scikit Learn method: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
RMSE: 0.8044974066807946

Classifier evaluation for: Ridge_Regression
  Scikit Learn method: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
RMSE: 0.8054744909557435

Classifier evaluation for: Lasso
  Scikit Learn method: Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 1.5513050279342468

Classifier evaluation for: Elastic_Net
  Scikit Learn method: ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.25,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.8365288425953381

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

Shape of input data for this fold:
Data Set: (Observations, Variables)
X_train: (456, 12)
X_test: (50, 12)
y_train: (456,)
y_test: (50,)

Classifier evaluation for: Linear_Regression
  Scikit Learn method: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
RMSE: 0.6890695563753627

Classifier evaluation for: Ridge_Regression
  Scikit Learn method: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
RMSE: 0.6898685677369694

Classifier evaluation for: Lasso
  Scikit Learn method: Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 1.4387124078783402

Classifier evaluation for: Elastic_Net
```

```
  Scikit Learn method: ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.25,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.7277584919693205

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

Shape of input data for this fold:
Data Set: (Observations, Variables)
X_train: (456, 12)
X_test: (50, 12)
y_train: (456,)
y_test: (50,)

Classifier evaluation for: Linear_Regression
  Scikit Learn method: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fa
RMSE: 0.47386940618795154

Classifier evaluation for: Ridge_Regression
  Scikit Learn method: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
RMSE: 0.47172152353606756

Classifier evaluation for: Lasso
  Scikit Learn method: Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.7617856516578413

Classifier evaluation for: Elastic_Net
  Scikit Learn method: ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.25,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
RMSE: 0.4662848152155337
```

In [60]:
```
# convert array into a data frame
# and then assign column names from names list
cv_results_df = pd.DataFrame(cv_results)
cv_results_df.columns = names

print('\n----------------------------------------------')
print('Average results from ', N_FOLDS, '-fold cross-validation\n',
      '\nMethod                 RMSE', sep = '')
print(cv_results_df.mean())
```

```
----------------------------------------------
```

```
Average results from 10-fold cross-validation

Method                  RMSE
Linear_Regression    0.582659
Ridge_Regression     0.582095
Lasso                1.019173
Elastic_Net          0.572309
dtype: float64
```

```
In [61]: print('\n----------------------------------------------')
         print('RMSE for each fold of cross validation:\n')
         print(cv_results_df)
```

```
----------------------------------------------
RMSE for each fold of cross validation:

   Linear_Regression  Ridge_Regression     Lasso  Elastic_Net
0           0.615331          0.616042  0.931303     0.633911
1           0.374727          0.372487  0.783124     0.332342
2           0.737212          0.736144  1.113791     0.718563
3           0.653564          0.652478  0.965149     0.565624
4           0.653456          0.652822  0.934317     0.594939
5           0.420428          0.419477  0.952173     0.397399
6           0.404440          0.404434  0.760069     0.449734
7           0.804497          0.805474  1.551305     0.836529
8           0.689070          0.689869  1.438712     0.727758
9           0.473869          0.471722  0.761786     0.466285
```