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
```

In [2]:
```
# seed value for random number generators to obtain reproducible results
RANDOM_SEED = 1
```

In [3]:
```
# 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
```

In [4]:
```
# import base packages into the namespace for this program
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [5]:
```
# 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
```

In [7]:
```
# read data for the Boston Housing Study
# creating data frame restdata
boston_input = pd.read_csv('boston.csv')
```

In [8]:
```
# check the pandas DataFrame object boston_input
print('\nboston DataFrame (first and last five rows):')
print(boston_input.head())
print(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 [9]:
```
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 [10]:
```
# 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 [11]:
```
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 [12]:
```
# 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 [13]:
```
# 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 [14]:
```
# 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 [15]:
```
# show standardization constants being employed
print(scaler.mean_)
print(scaler.scale_)
```

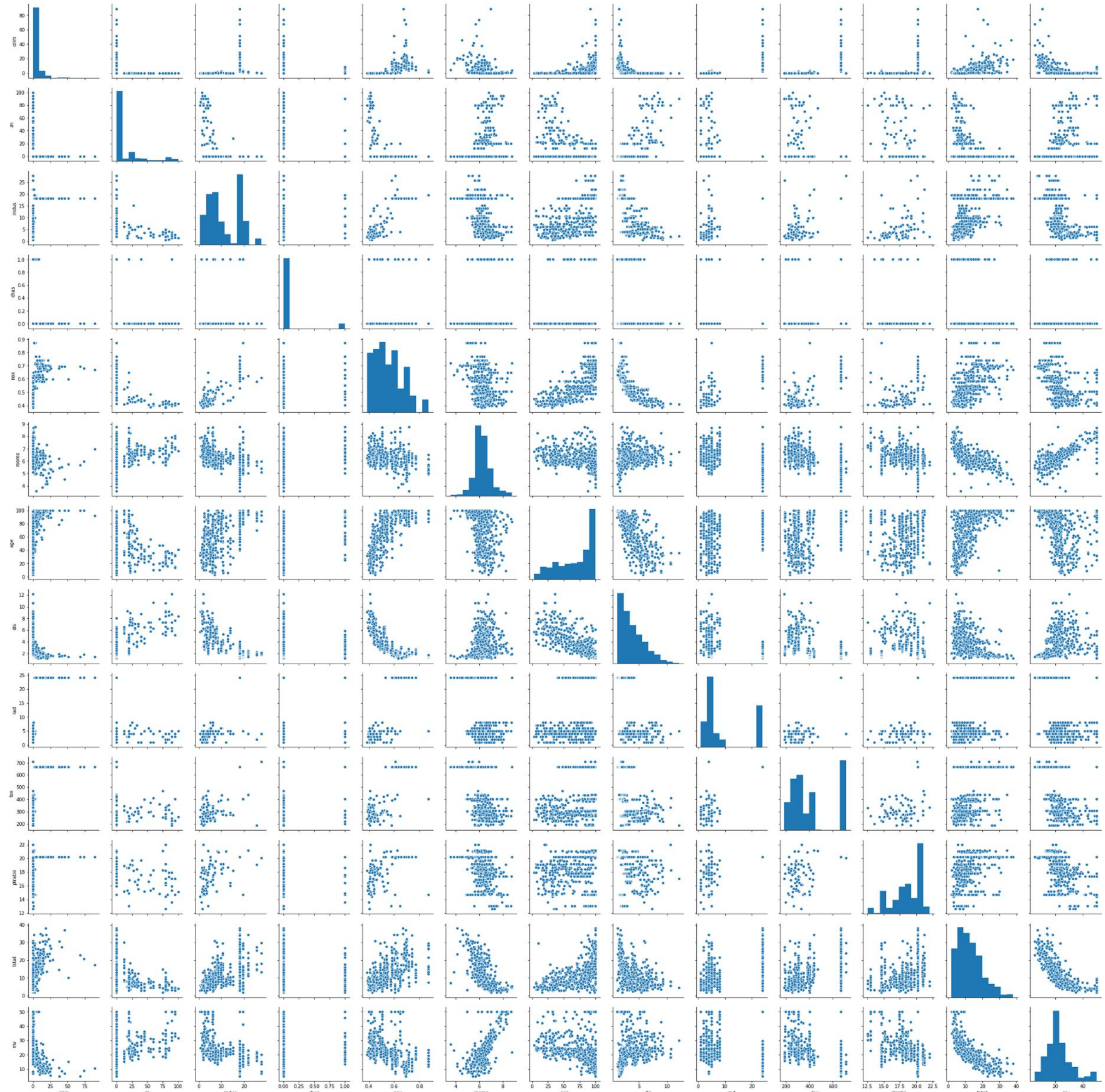
```
[2.25288538e+01 3.61352356e+00 1.13636364e+01 1.11367787e+01
 6.91699605e-02 5.54695059e-01 6.28463439e+00 6.85749012e+01
 3.79504269e+00 9.54940711e+00 4.08237154e+02 1.84555336e+01
 1.26530632e+01]
[9.17309810e+00 8.59304135e+00 2.32993957e+01 6.85357058e+00
 2.53742935e-01 1.15763115e-01 7.01922514e-01 2.81210326e+01
 2.10362836e+00 8.69865112e+00 1.68370495e+02 2.16280519e+00
 7.13400164e+00]
```

In [16]:
```
# the model data will be standardized form of preliminary model data
model_data = scaler.fit_transform(prelim_model_data)
```

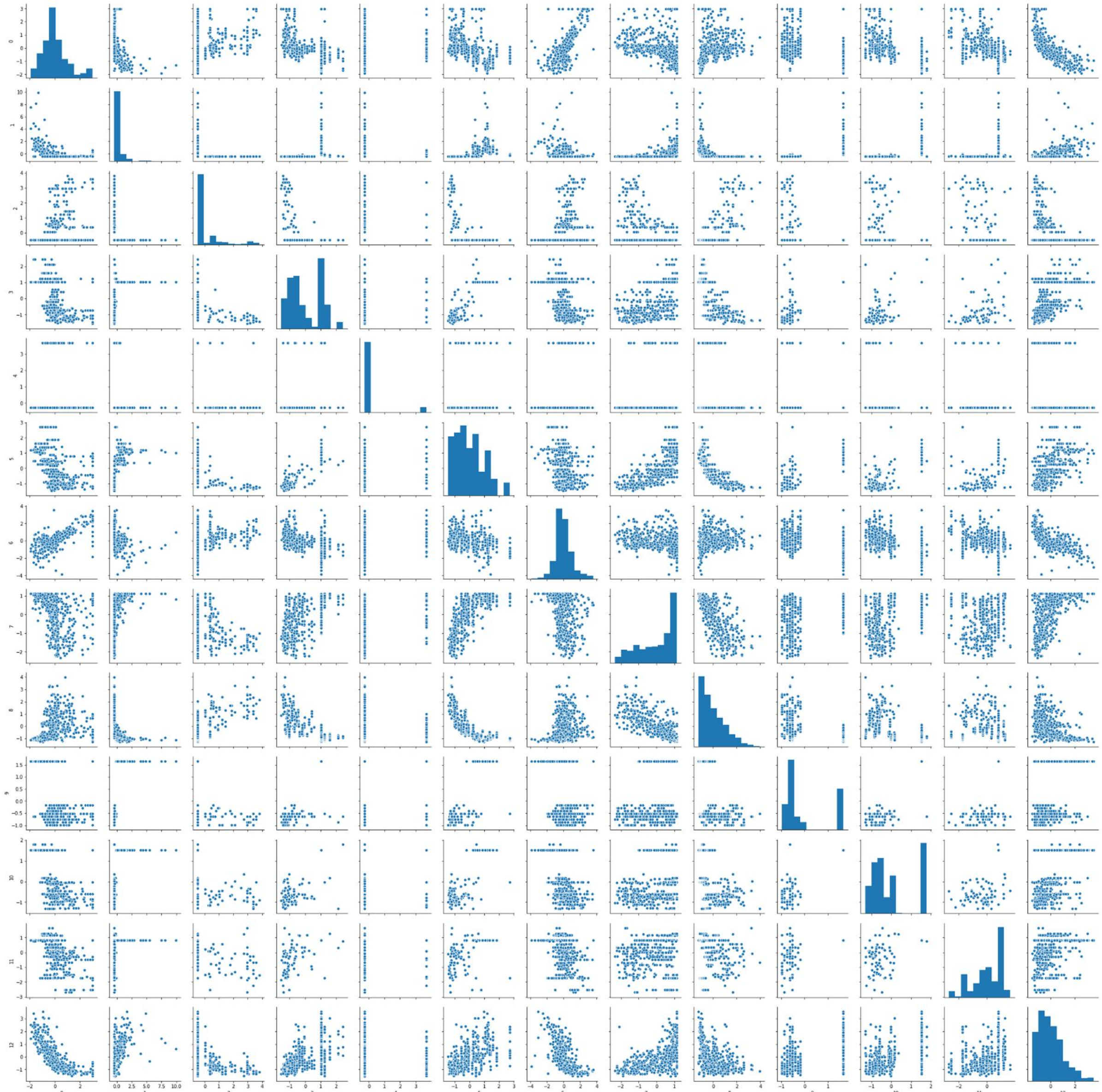
In [17]:
```
# dimensions of the polynomial model X input and y response
# all in standardized units of measure
print('\nDimensions for model_data:', model_data.shape)
```

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

In [18]:
```
#Compare unscaled and normalized data
sns.pairplot(boston)
plt.show()
```

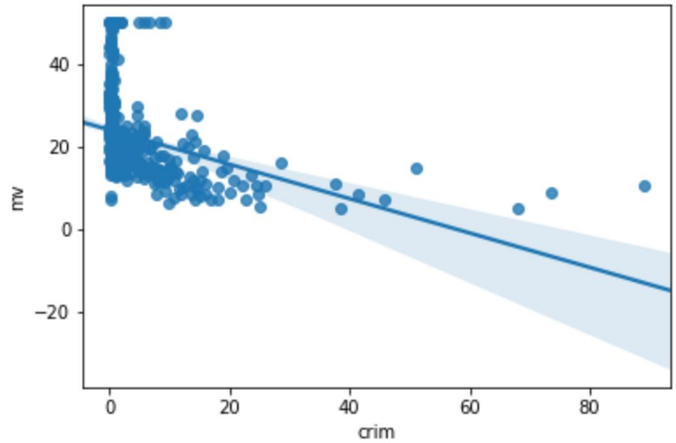


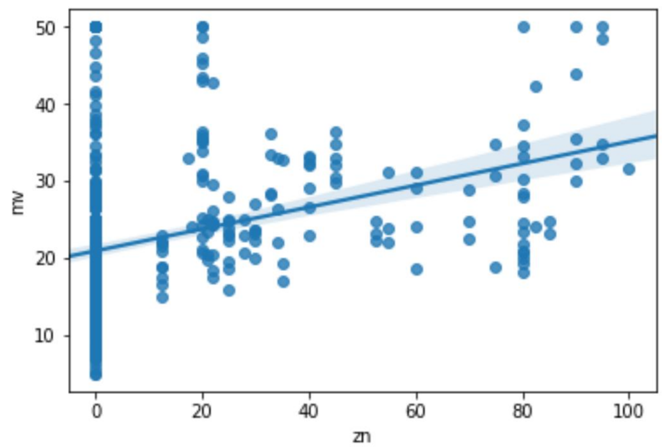
In [19]:
```
boston_scaled = pd.DataFrame(model_data)
sns.pairplot(boston_scaled)
plt.show()
```

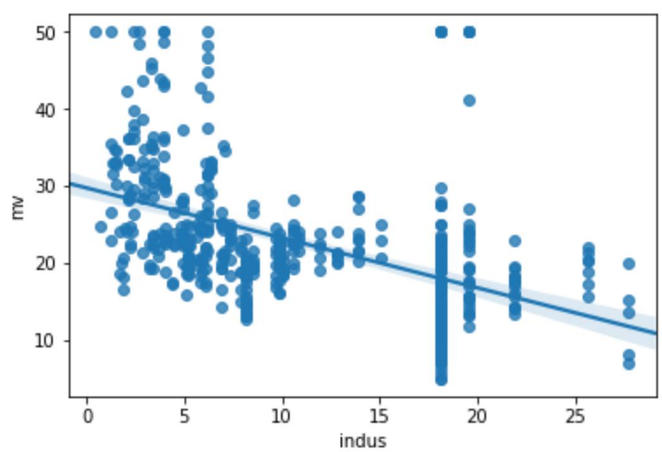


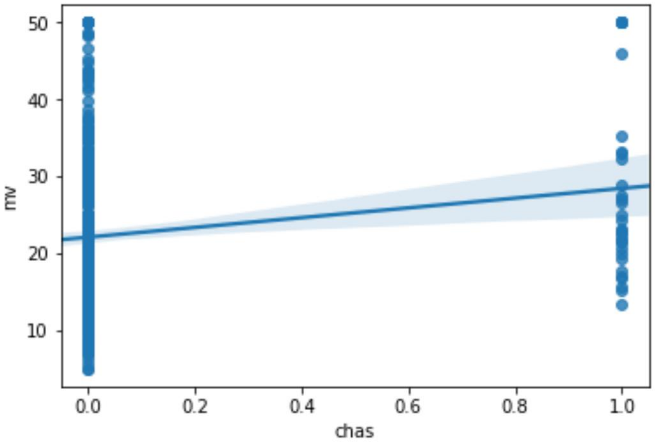
In [20]:
```
# plot linear relationships with regression line where mv is the dependent variable
sns.regplot(x="crim", y="mv", data=boston)
plt.show()
sns.regplot(x="zn", y="mv", data=boston)
plt.show()
sns.regplot(x="indus", y="mv",data=boston)
plt.show()
sns.regplot(x="chas", y="mv", data=boston)
plt.show()
sns.regplot(x="nox", y="mv", data=boston)
plt.show()
sns.regplot(x="rooms", y="mv", data=boston)
plt.show()
sns.regplot(x="age", y="mv",data=boston)
plt.show()
sns.regplot(x="dis", y="mv", data=boston)
plt.show()
sns.regplot(x="rad", y="mv", data=boston)
plt.show()
sns.regplot(x="tax", y="mv", data=boston)
plt.show()
sns.regplot(x="ptratio", y="mv",data=boston)
plt.show()
sns.regplot(x="lstat", y="mv", data=boston)
plt.show()
```

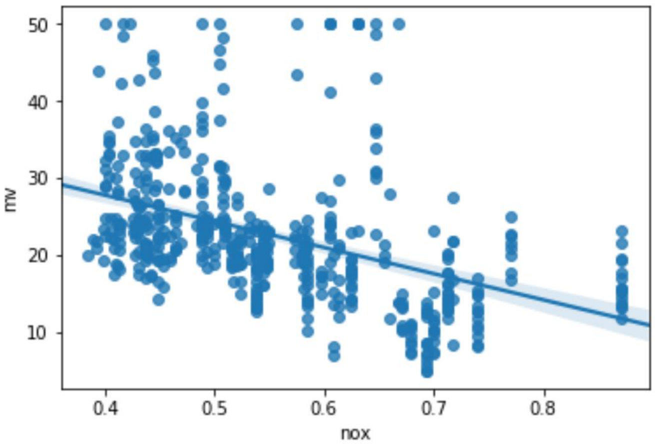
```
C:\Users\Jimmy\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarn
ing: Using a non-tuple sequence for multidimensional indexing is deprecated; use
`arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted
as an array index, `arr[np.array(seq)]`, which will result either in an error or
a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

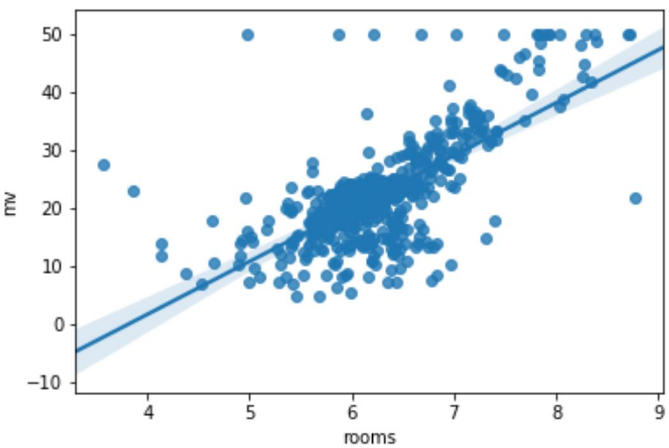


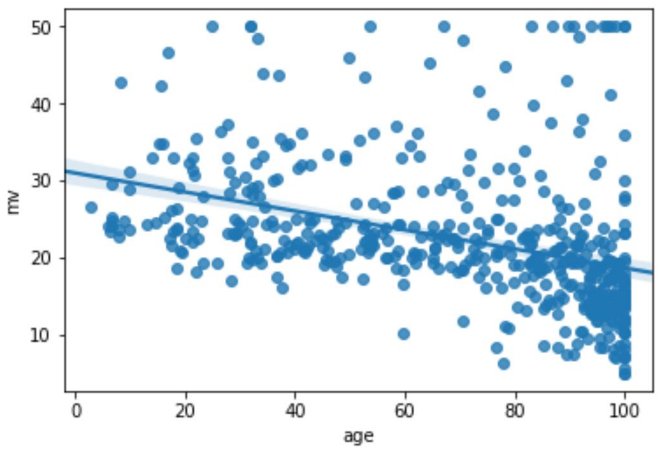


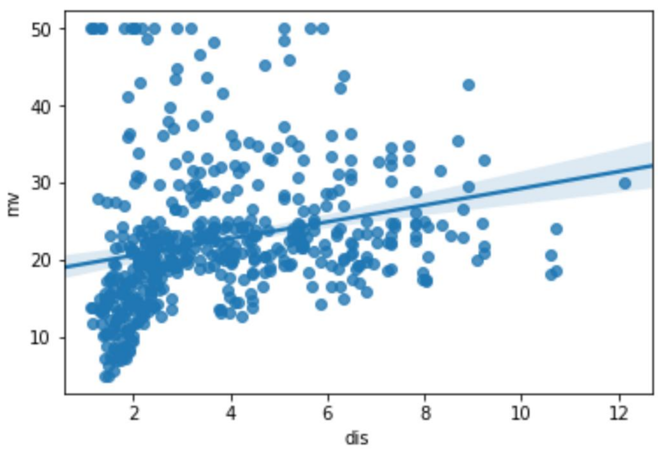


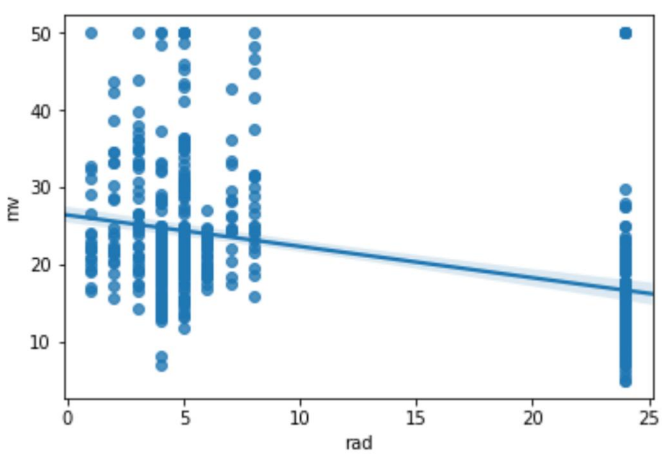

mv
chas
0.0
0.2
0.4
0.6
0.8
1.0
10
20
30
40
50

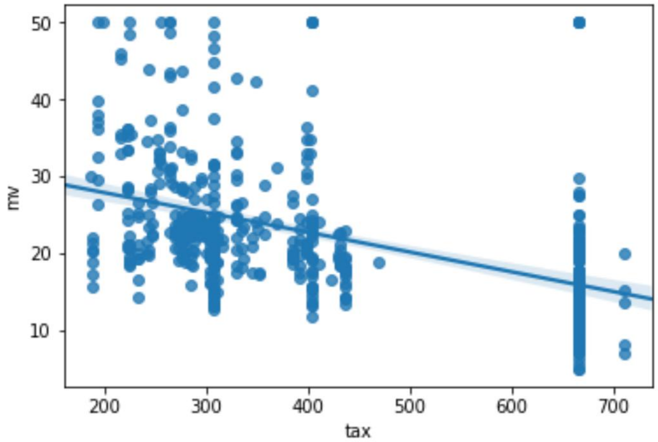

mv
nox
0.4
0.5
0.6
0.7
0.8
10
20
30
40
50

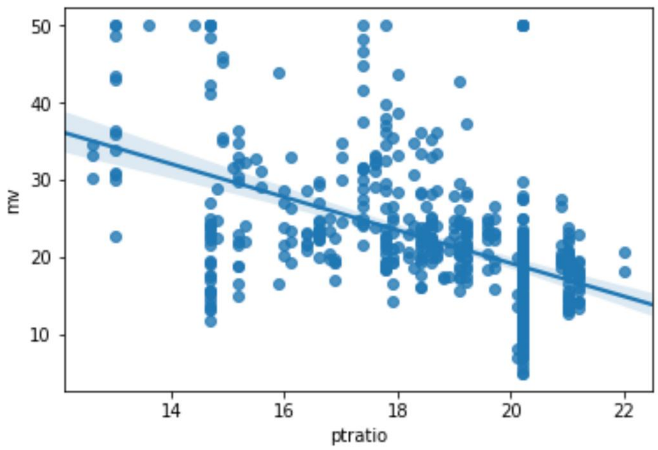

mv
rooms
4
5
6
7
8
9
−10
0
10
20
30
40
50

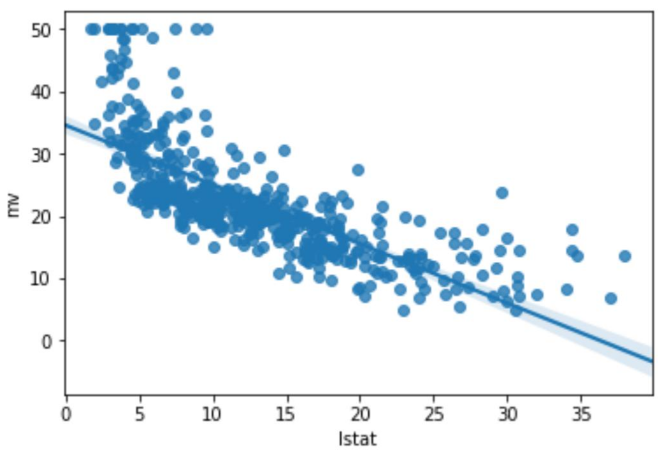

mv
age
0
20
40
60
80
100
10
20
30
40
50


mv
dis
2
4
6
8
10
12
10
20
30
40
50


mv
rad
0
5
10
15
20
25
10
20
30
40
50

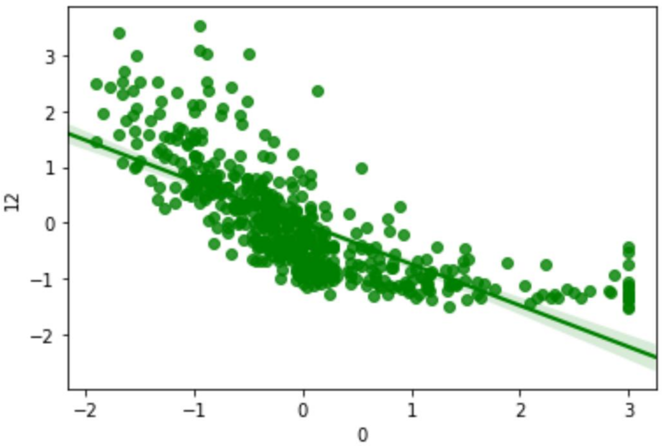

mv
tax
10
20
30
40
50
200
300
400
500
600
700

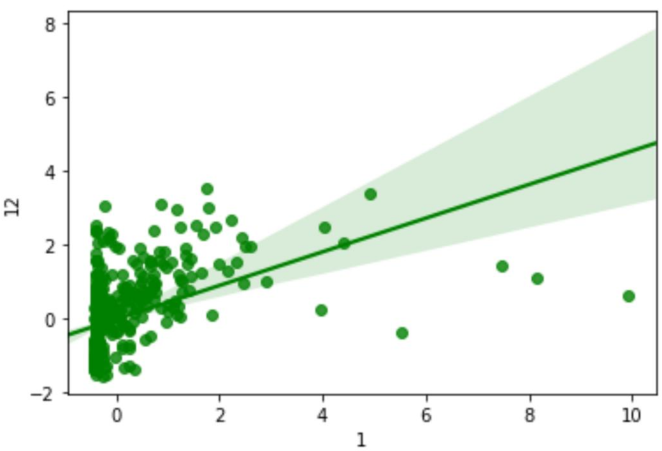

mv
ptratio
10
20
30
40
50
14
16
18
20
22

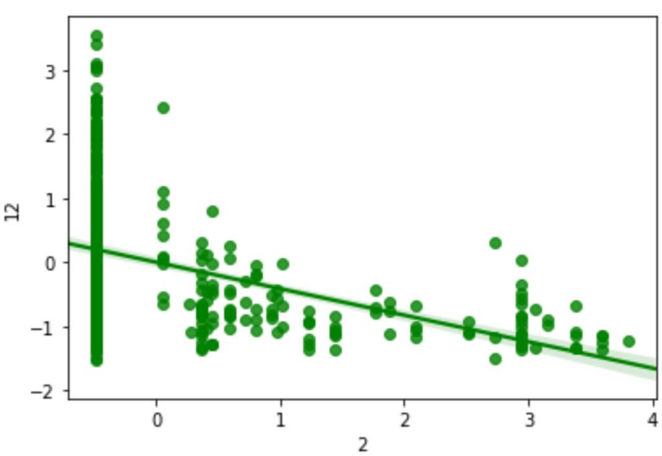

mv
lstat
0
10
20
30
40
50
0
5
10
15
20
25
30
35

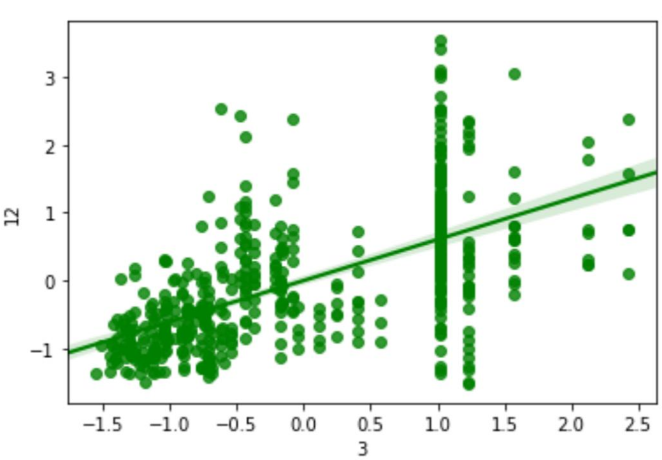
In [42]:

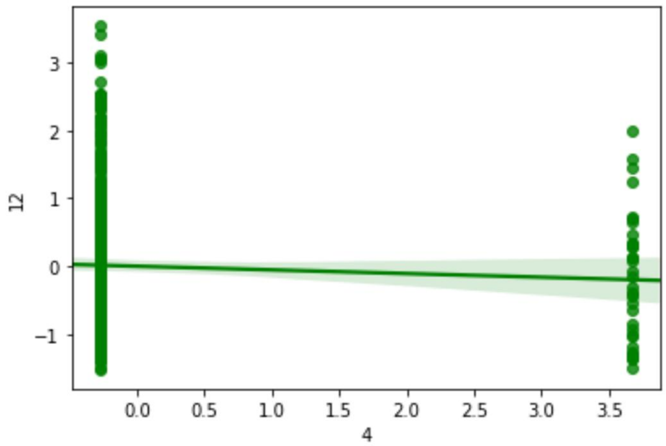
```
sns.regplot(x=boston_scaled[0], y=boston_scaled[12], data=boston_scaled,  color="green")
plt.show()
sns.regplot(x=boston_scaled[1], y=boston_scaled[12], data=boston_scaled, color="green")
plt.show()
sns.regplot(x=boston_scaled[2], y=boston_scaled[12],data=boston_scaled, color="green")
plt.show()
sns.regplot(x=boston_scaled[3], y=boston_scaled[12], data=boston_scaled, color="green")
plt.show()
sns.regplot(x=boston_scaled[4], y=boston_scaled[12], data=boston_scaled, color="green")
plt.show()
sns.regplot(x=boston_scaled[5], y=boston_scaled[12], data=boston_scaled, color="green")
plt.show()
sns.regplot(x=boston_scaled[6], y=boston_scaled[12],data=boston_scaled, color="green")
plt.show()
sns.regplot(x=boston_scaled[7], y=boston_scaled[12], data=boston_scaled, color="green")
plt.show()
sns.regplot(x=boston_scaled[8], y=boston_scaled[12], data=boston_scaled, color="green")
plt.show()
sns.regplot(x=boston_scaled[9], y=boston_scaled[12], data=boston_scaled, color="green")
plt.show()
sns.regplot(x=boston_scaled[10], y=boston_scaled[12],data=boston_scaled, color="green")
plt.show()
sns.regplot(x=boston_scaled[11], y=boston_scaled[12], data=boston_scaled, color="green")
plt.show()
```

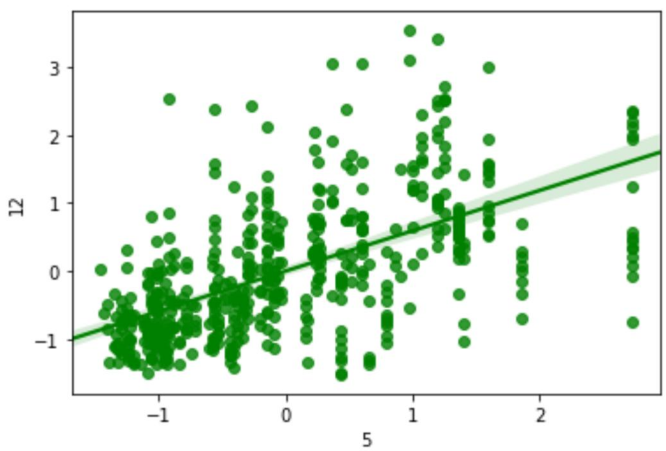

12
0

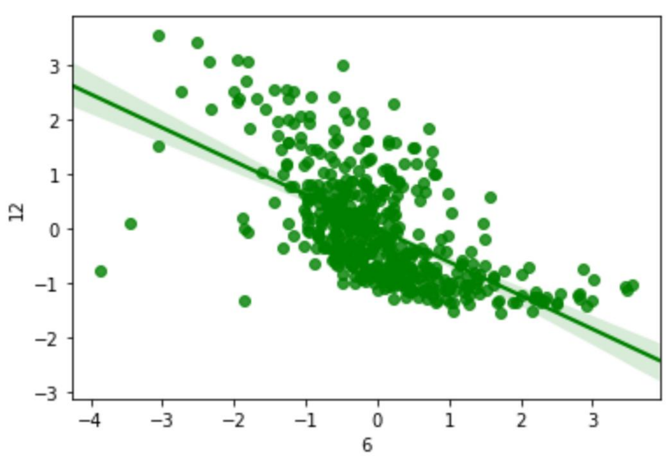

12
1

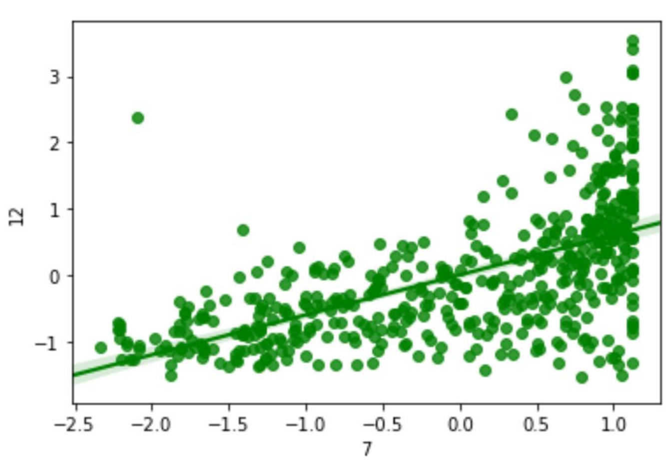

12
2

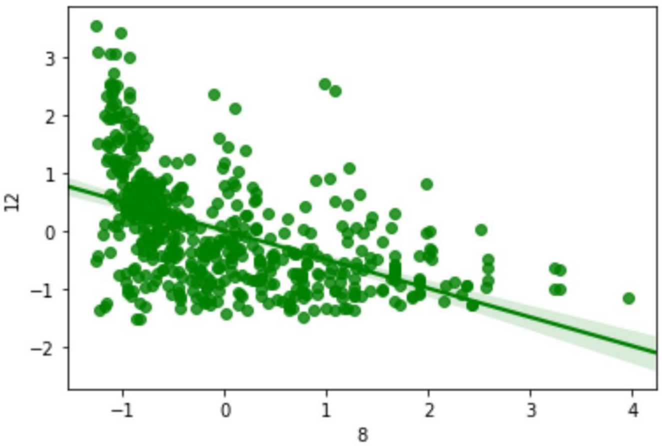

12
3

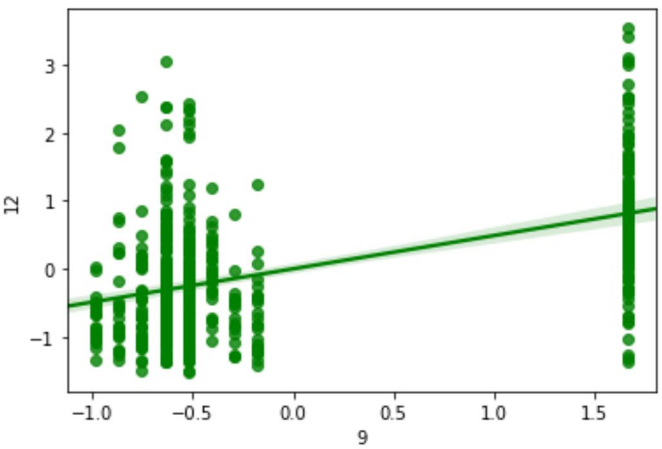

12
4

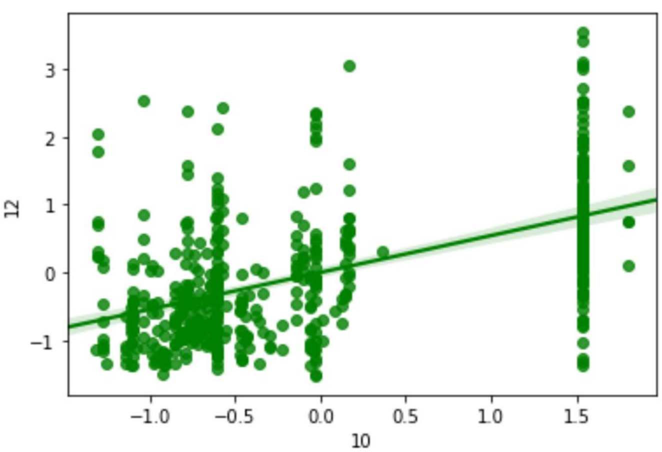

12
5

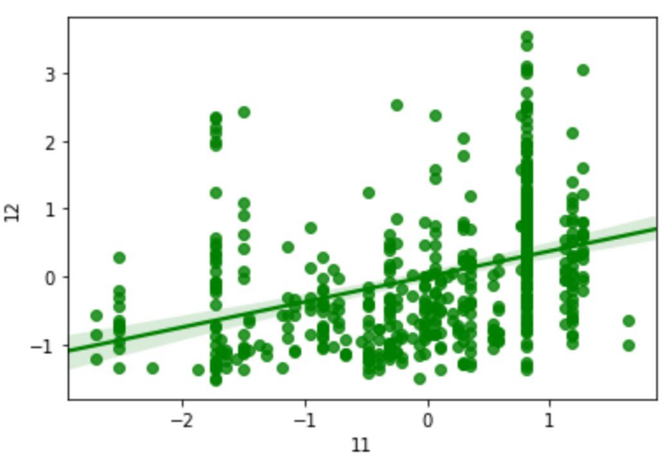

12
6


12
7


12
8


12
9

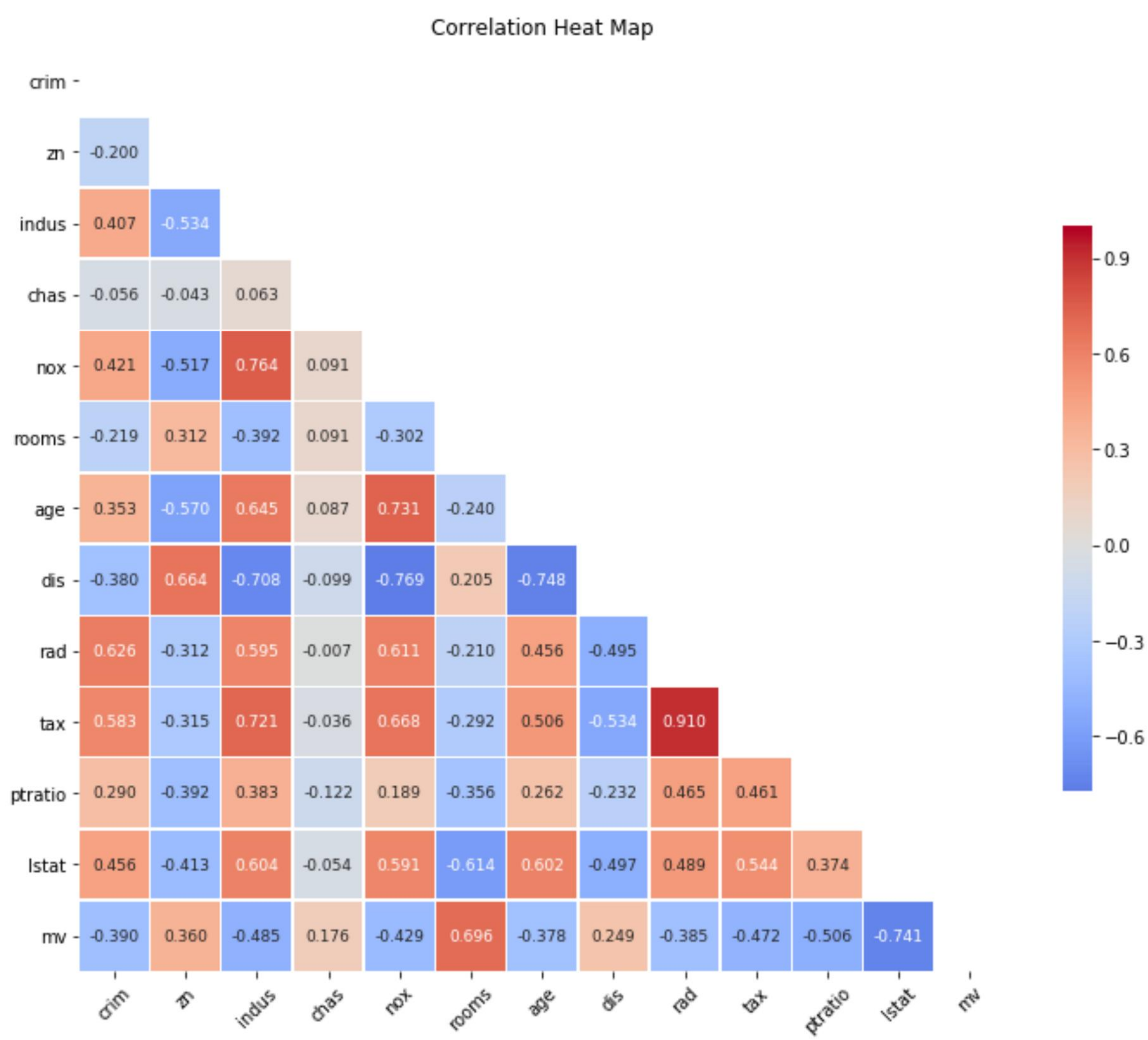

12
10


12
11

In [29]:
```
# 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)
```

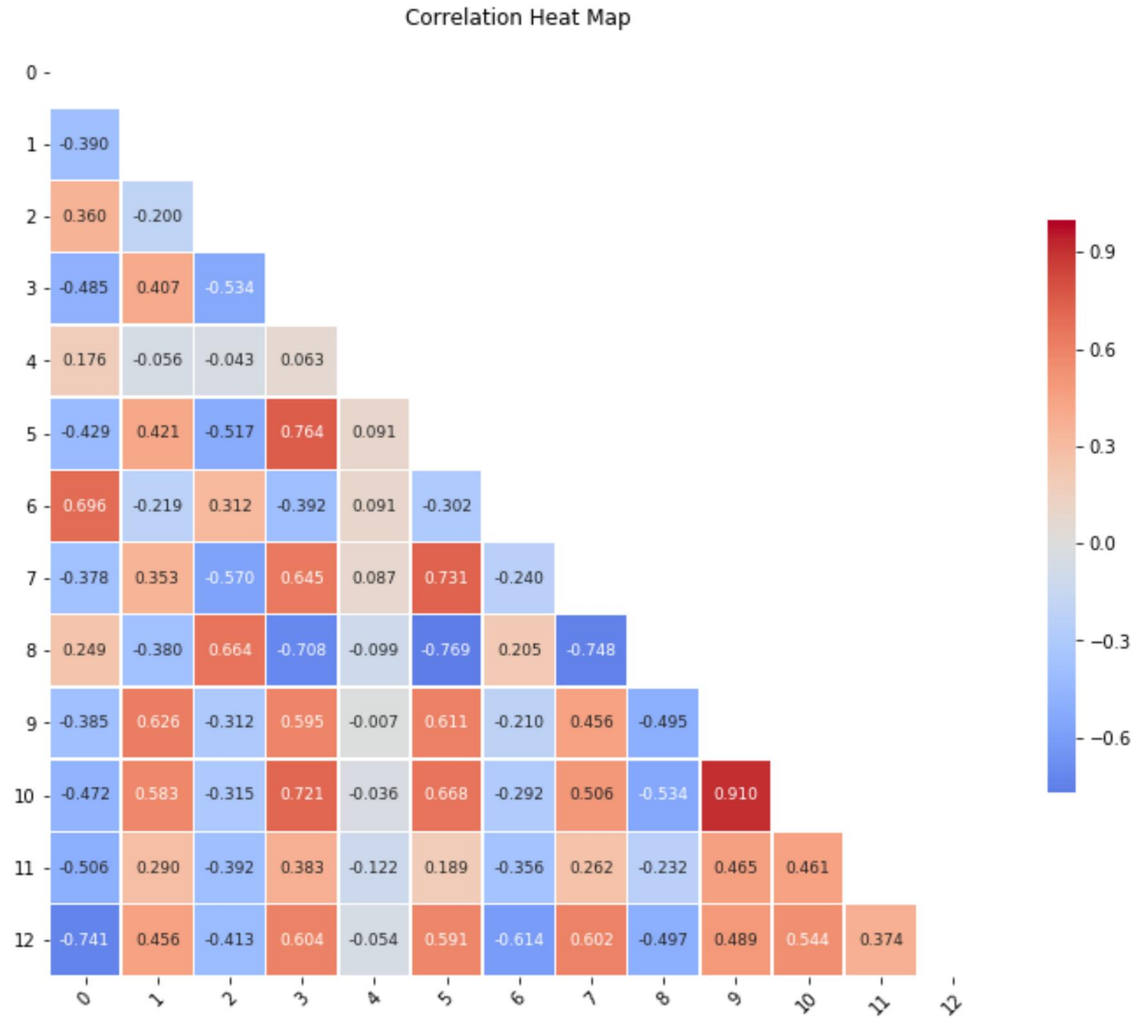
In [30]:
```
corr_chart(boston)
```

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



In [121]:
```
corr_chart(boston_scaled)
```

<Figure size 432x288 with 0 Axes>



In [101]:
```
#Regression analysis - use scaled data
# Let's put together x and y training and test sets

boston_scaled.dropna()
x = boston_scaled.drop([12]).values
y = (boston_scaled[12]).values[0:505]
```

In [107]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split

#take random shuffle of data for training and testing datasets - 20% of data used for test
x_train, x_test , y_train, y_test = train_test_split(x,y,test_size=0.2)
```

In [108]:
```
#use to training data and compare to test data set
lin= LinearRegression()
linfit=lin.fit(x_train,y_train)
```

In [109]:
```
#Find RMSE
y_pred=LinearRegression.predict(linfit, x_test)
print("RMSE for Linear Regression:")
print( sqrt(mean_squared_error(y_test,y_pred)))
```

```
RMSE for Linear Regression:
0.5619360369051359
```

In [110]:
```
#Lasso Regression
las=Lasso()
lasfit=las.fit(x_train,y_train)
```

In [111]:
```
y_pred_las=Lasso.predict(lasfit, x_test)
print("RMSE for Lasso:")
print( sqrt(mean_squared_error(y_test,y_pred_las)))
```

```
RMSE for Lasso:
0.9620739310707962
```

In [116]:
```
#Ridge Regression
rid=Ridge()
ridfit=rid.fit(x_train,y_train)
```

In [117]:
```
y_pred_rid=Ridge.predict(ridfit, x_test)
print("RMSE for Ridge:")
print( sqrt(mean_squared_error(y_test,y_pred_rid)))
```

```
RMSE for Ridge:
0.561424623010174
```

In [118]:
```
#ElasticsNet Regression
elas=ElasticNet()
elasfit=elas.fit(x_train,y_train)
```

In [119]:
```
y_pred_elas=ElasticNet.predict(elasfit, x_test)
print("RMSE for ElasticNet:")
print( sqrt(mean_squared_error(y_test,y_pred_elas)))
```

```
RMSE for ElasticNet:
0.8412368223631603
```

In [ ]:

In [ ]: