Allison Roeser

Boston Housing Study: Random Forests and Gradient Boosting

Data Preparation, Exploration, Visualization:

The Boston housing dataset consists of 506 observations for 13 variables, describing characteristics of houses in Boston neighborhoods (Figure 1). The objective of the study is to predict the Median Home Value in thousands (mv) using the 12 potential predictor variables in the dataset. Figures 2 & 3 show boxplots of the variables before scaling, with the uneven scale of the variables evident. Boxplots with jitter after Standard Scaling show the distribution of data point for each variable (Figures 4 & 5). Tax and rad have uneven, banded distributions. Crim is highly right skewed. The violin plots for each variable illustrate that the distribution for lstat is most similar to the distribution for mv (Figure 6).

Figure 7 contains density and probability plots for mv. Mv has a skewness of 1.11 and a kurtosis of 1.52. Taking the log of mv, creates a more even distribution by reducing the skewness to -0.34 and kurtosis to 0.83 (Figure 8). Figure 9 lists the variables whose correlation with mv is greater than 0.35. Lstat, pratio, and rooms are most highly correlated with mv. Tax and rad are strongly correlated with one another, having a correlation score of 0.91 (Figure 10). A regression model would likely not require both variables. A scatter plot matrix of predictor variables vs. both mv and logmv are displayed in Figures 11 & 12. Strong correlations with lstat, rooms, and pratio are shown.

Implementation and Programming:

Full code is attached in the Appendix to show the specific implementation. Scikit-learn was leveraged for regression model building (Linear, Ridge, Lasso, Elastic Net, Stochastic Gradient Descent, Random Forest, and Extra Trees Regressions). Hyper-parameters were tested using Grid Search. RMSE was used to evaluate potential models, and variable importance scores were used to determine the relative importance of each predictor variable. Pandas and Seaborn were leveraged for exploratory data analysis and visualization.

Review Research Design and Modeling Methods:

Regression models were run to predict logmv. All 12 potential predictor variables were used in modeling, allowing each model to select variables to include or exclude. All variables were scaled using Standard Scaling to ensure proper weighting of variable importance. Grid Search was used to select the best hyperparameters for each model, and root mean squared error (RMSE) was used to score the models.

Ordinary Least Squares Regression was run to create a baseline model. Stochastic Gradient Descent was another regression technique tested. Gradient decent is an iterative optimization technique used to find the local minimum of a function. Stochastic Gradient Decent is a more efficient form of Gradient Decent that uses random samples of the data to reduce the computations required (Srinivasan 2019). Decision tree models were also utilized through Random Forest Regression and Extra Tree Regression.

Two champion models were selected from the various regression models run. Variable importance scores were determined for each of these models. Cumulative variable importance was also investigated to determine how many variables are required to create a robust model that is not overfit to the data. Cutoff thresholds were determined to understand how many variables were required to explain 95%, 90%, and 80% of the variability explained by the model.

<u>Review Results, Evaluate Models</u>:

Random Forest Regression and Extra Trees Regression were most effective at predicting the value of logmv as measured by RMSE scores (Figure 13). The optimal hyperparameters for each model determined through Grid Search methods are shown in Figure 14 & 15. The Random Forest Regression used more splits (10) and less depth (50) than Extra Trees Regression. In contrast, for Extra Trees Regression the optimal number of splits was 7 and the optimal max depth was 75.

Figure 16 displays the variable importances for the Random Forest model. Lstat, crim and rooms were shown to be the most important variables in for this model. Using Extra Trees Regression, the most important variables were lstat, rooms, and nox (Figure 17). Both models placed the most weight on lstat, which represents the proportion of homes of low socio-economic status.

Figures 18 and 19 show the cumulative variable importances for the two models. The Random Forest model requires only 6 variables to reach 90% cumulative variable importance while the Extra Trees model requires 9 variables. Cutoff thresholds of 80% and 95% cumulative importance are also displayed in the graphs.

Exposition, Problem Description and Management Recommendations:

Of all the modeling methods tested to model housing prices in the Boston area, Random Forest Regression and Extras Tree Regression methods were the most accurate at predicting the log of median home value. These models achieved the lowest RMSE. Stochastic Gradient descent was also successful in predicting home prices. Management may be best served by using an ensemble method composed of these various techniques.

All models identified that the population proportion of low socio-economic individuals (lstat) has the largest impact on housing values. If possible, management should find ways to help improve the economic status of Boston residents in the neighborhoods where they conduct business. Community outreach efforts could lead to higher home values. Rooms and crim were two other variables identified as important. When predicting home prices, number or rooms and crime rate should be strongly considered along with socio economic status of residents.

References

Holtz, Y. #39 Hidden date under boxplot. (n.d.). The Python Graph Gallery. Retrieved from: https://python-graph-gallery.com/39-hidden-data-under-boxplot/

Holtz, Y. #125 Small multiples for line charts. (n.d.). The Python Graph Gallery. Retrieved from: https://python-graph-gallery.com/125-small-multiples-for-line-chart/

Koehrsen, W. Improving the Random Forest in Python Part 1 (Jan 6, 2018). Towards Data Science. Retrieved from: https://towardsdatascience.com/improving-random-forest-in-python-part-1-893916666cd

Srinivasan, A. Stochastic Gradient Descent – Clearly Explained!! (Sep 6, 2019). Towards Data Science. Retrieved from: https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31

Xu, W. What's the difference between Linear Regression, Lasso, Ridge, and ElasticNet in sklearn? (Aug 21, 2019). Towards Data Science. Retrieved from: https://towardsdatascience.com/whats-the-difference-between-linear-regression-lasso-ridge-and-elasticnet-8f997c60cf29

Appendix

Figure 1: Variables in the Boston housing dataset

Variable Name	*Description*
neighborhood	Name of the Boston neighborhood (location of the census tract)
mv	Median value of homes in thousands of 1970 dollars
nox	Air pollution (nitrogen oxide concentration)
crim	Crime rate
zn	Percent of land zoned for lots
indus	Percent of business that is industrial or nonretail
chas	On the Charles River (1) or not (0)
rooms	Average number of rooms per home
age	Percentage of homes built before 1940
dis	Weighted distance to employment centers
rad	Accessibility to radial highways
tax	Tax rate
ptratio	Pupil/teacher ratio in public schools
lstat	Percentage of population of lower socio-economic status

Figure 2: Boxplot of variables before scaling

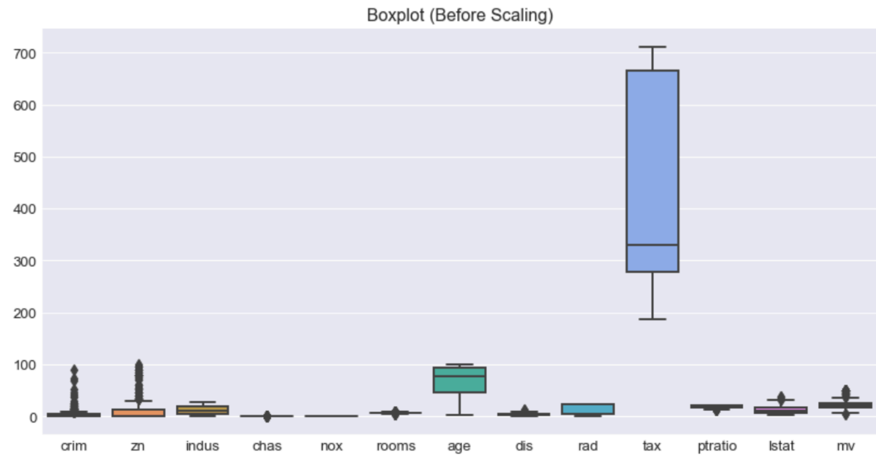


Figure 3: Boxplot of variables before scaling with jitter to show point distribution (Holtz)

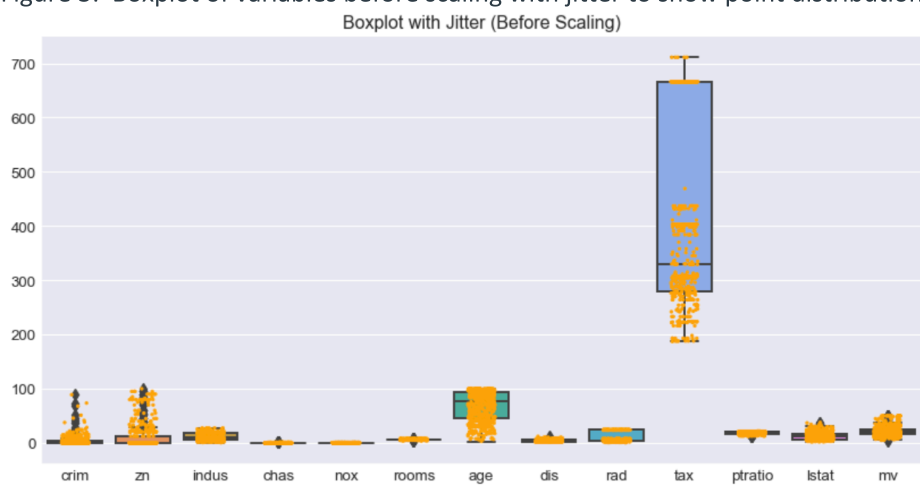


Figure 4: Boxplot after Standard Scaling

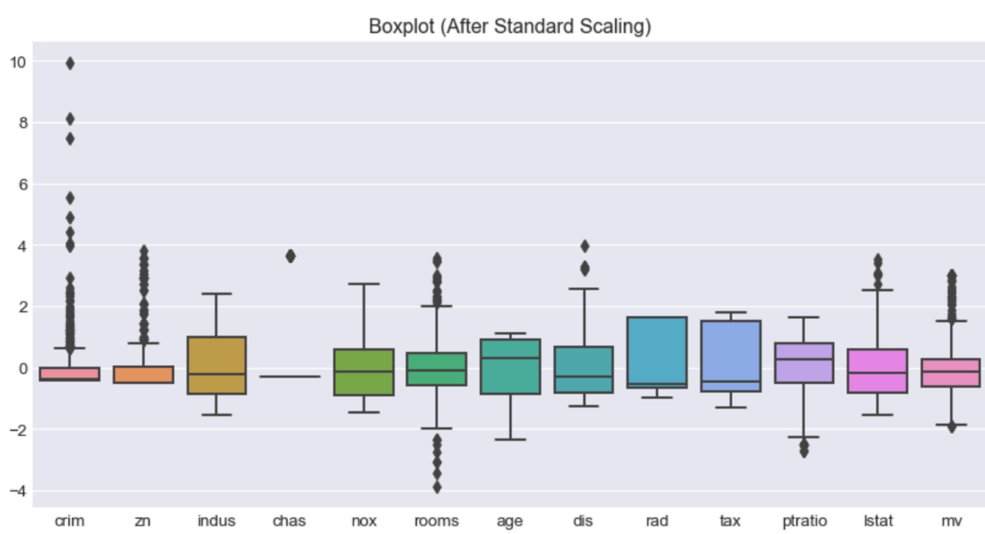


Figure 5: Boxplot with Jitter after Standard Scaling (Holtz)

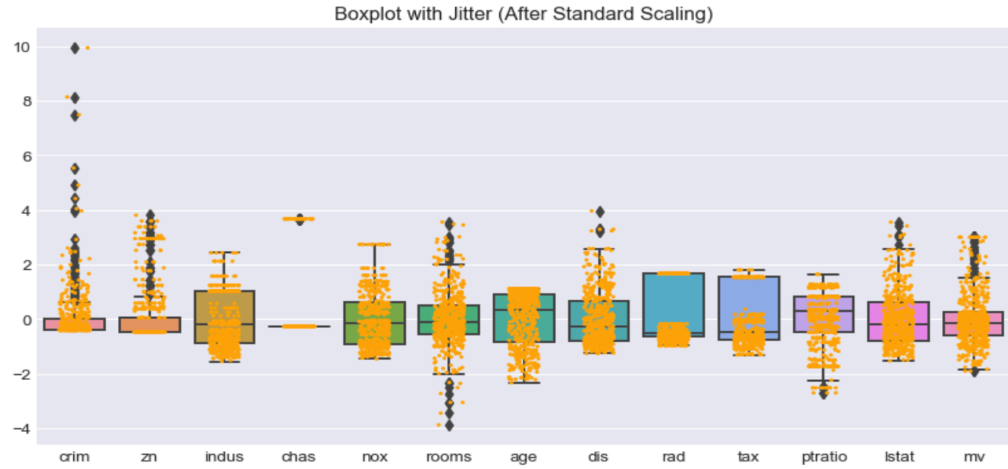


Figure 6: Violin plot after Standard Scaling. Lstat distribution is closest to the distribution of mv (Holtz)

Figure 7: Density and probability plots for mv (Tersakyan 2019)

```
Skewness: 1.110912
Kurtosis: 1.516783
```

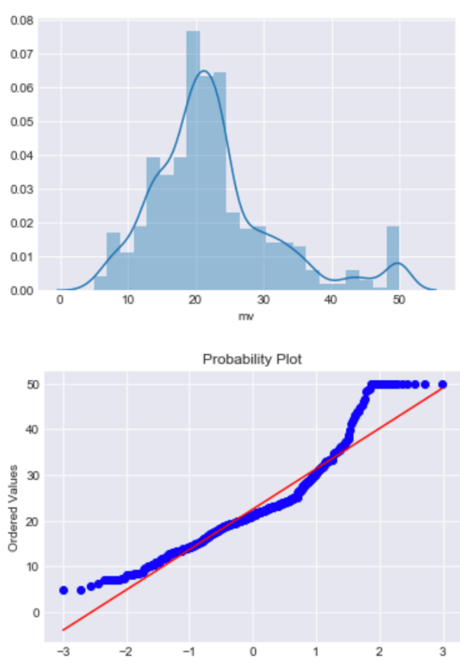


Figure 8: Density and probability plots for logmv (Tersakyan 2019)

```
Skewness: -0.335226
Kurtosis: 0.827799
```

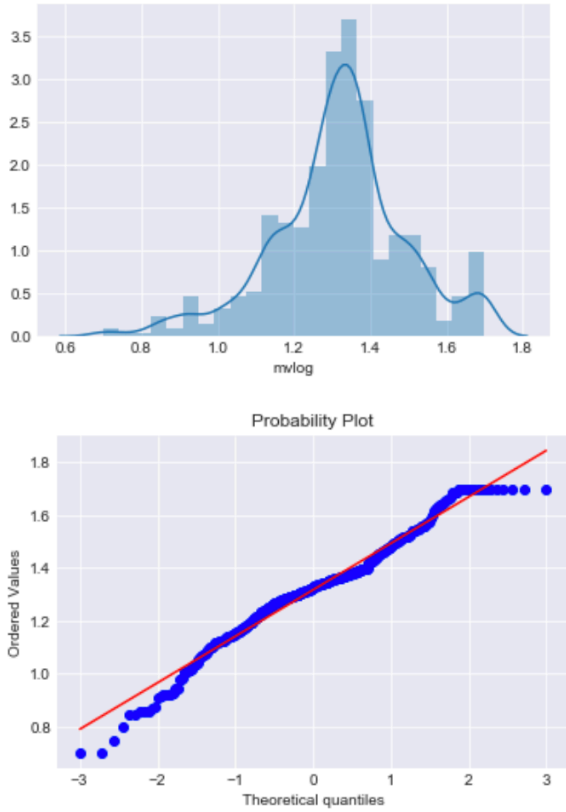


Figure 9: Variables whose correlation with mv is greater than 0.35

crim	0.389582
zn	0.360386
indus	0.484754
nox	0.429300
rooms	0.696304
age	0.377999
rad	0.384766
tax	0.471979
ptratio	0.505655
lstat	0.740836
mv	1.000000

Figure 10: Correlation matrix for tax, rad, and mv showing the multi-collinearity between tax and rad

	rad	tax	mv
rad	1.000000	0.910228	-0.384766
tax	0.910228	1.000000	-0.471979
mv	-0.384766	-0.471979	1.000000

Figure 11: Scatterplot matrix of mv vs. top 9 predictor variables (Holtz)

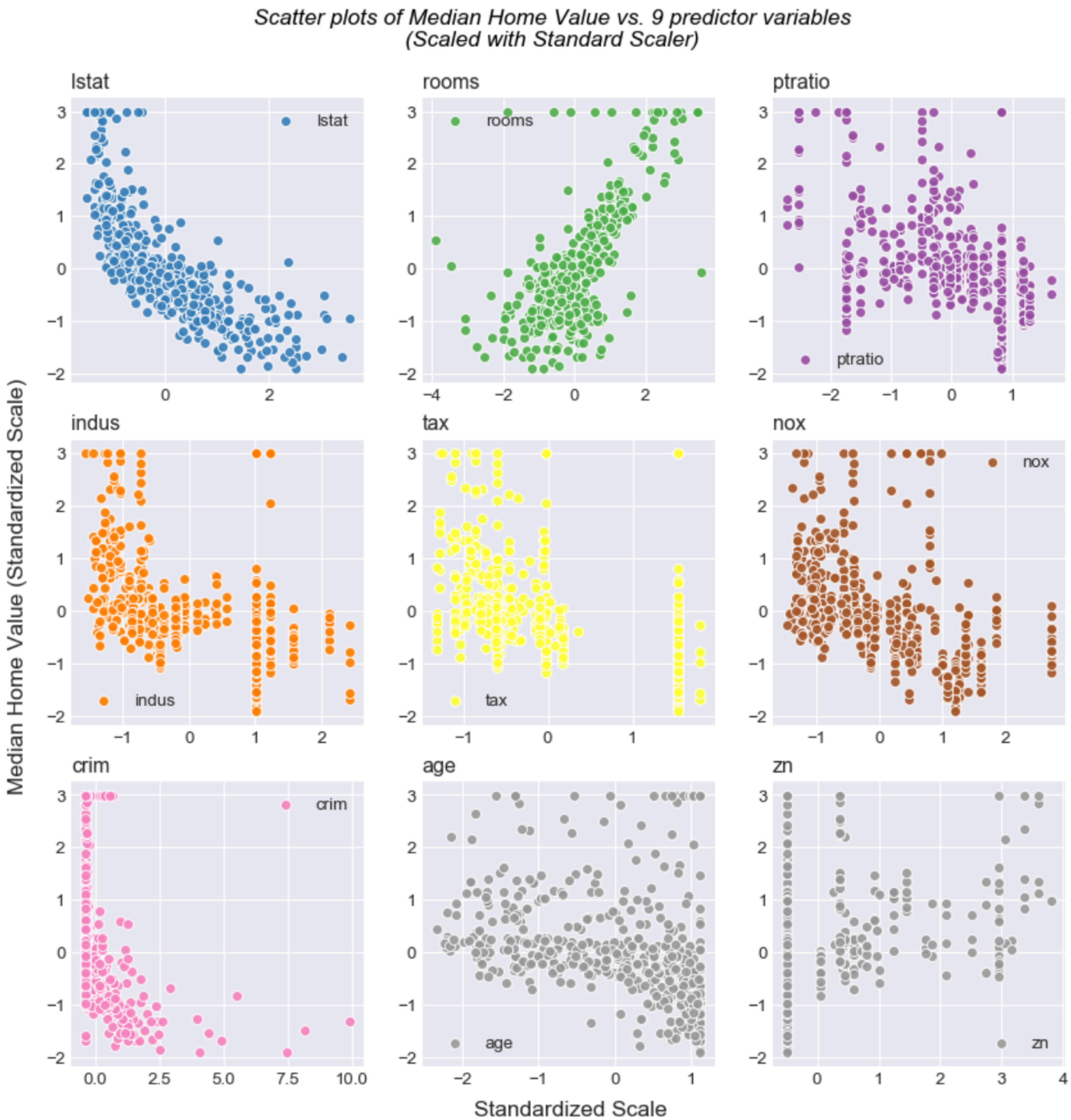


Figure 12: Scatterplot matrix of logmv vs. top 9 predictor variables (Holtz)

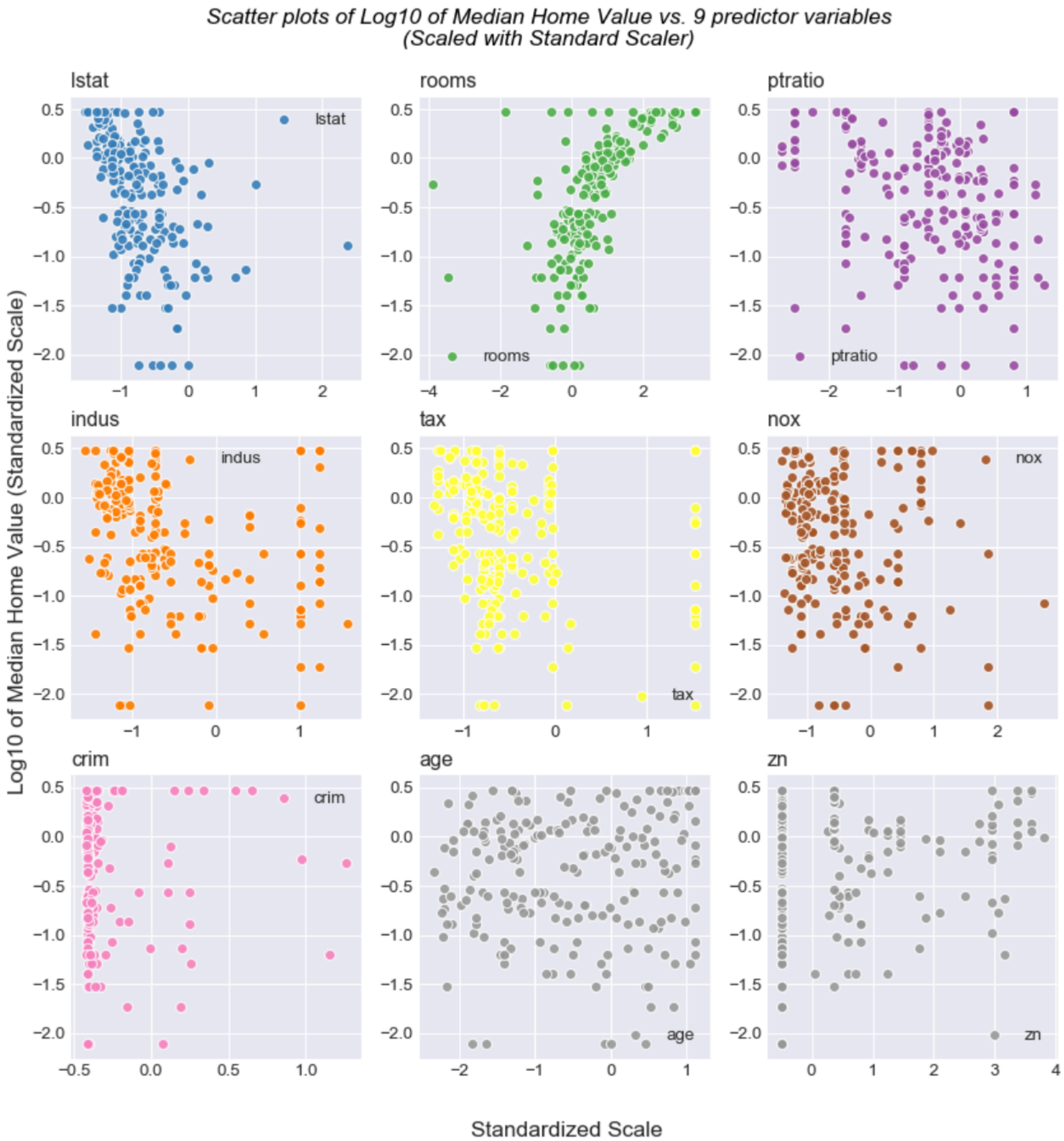


Figure 13: RMSE for each regression model

Method	Root Mean Sq. Error
Linear Regression	0.018462
Ridge Regression	0.018516
Lasso Regression	0.041755
Elastic Net Regression	0.041755
Random Forest Regression	0.018272
Extra Trees Regression	0.016439
SGD Regression	0.018795

Figure 14: Best hyperparameters for Random Forest Regression

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=50,
                      max_features=5, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=3, min_samples_split=10,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

Figure 15: Best hyperparameters for Extra Tree Regression

```
ExtraTreesRegressor(bootstrap=True, criterion='mse', max_depth=75,
                    max_features=5, max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=3, min_samples_split=7,
                    min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
                    oob_score=False, random_state=None, verbose=0,
                    warm_start=False)
```

Figure 16: Variable Importance for Random Forest Regression (Koehrsen 2018)

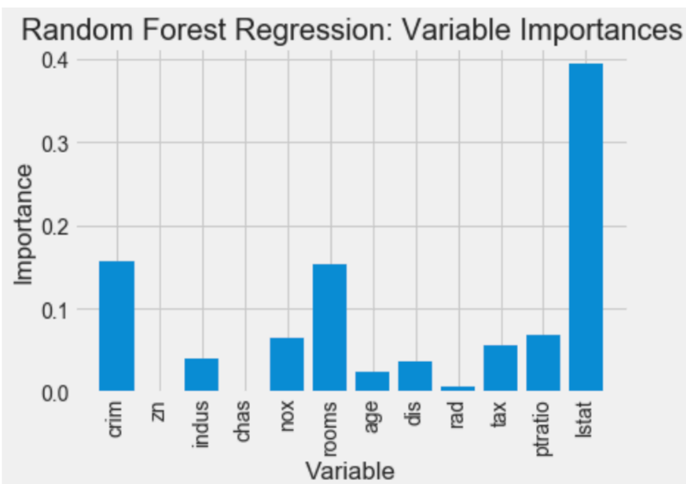


Figure 17: Variable Importance for Extra Trees Regression (Koehrsen 2018)

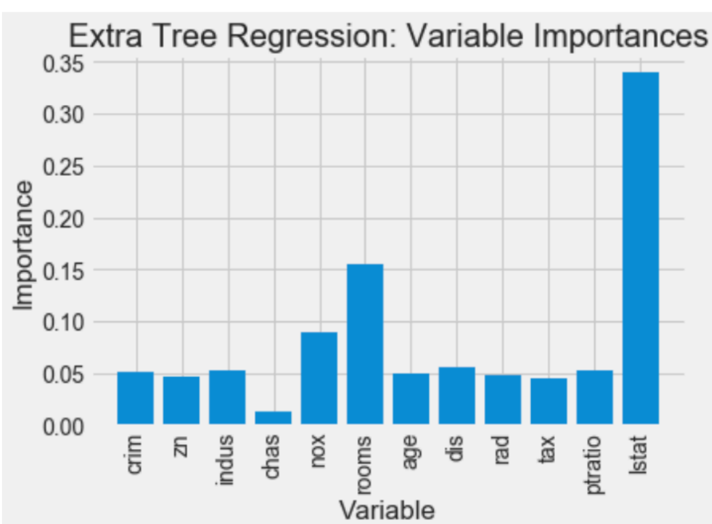


Figure 18: Variable Importance for Random Forest Regression (Koehrsen 2018)

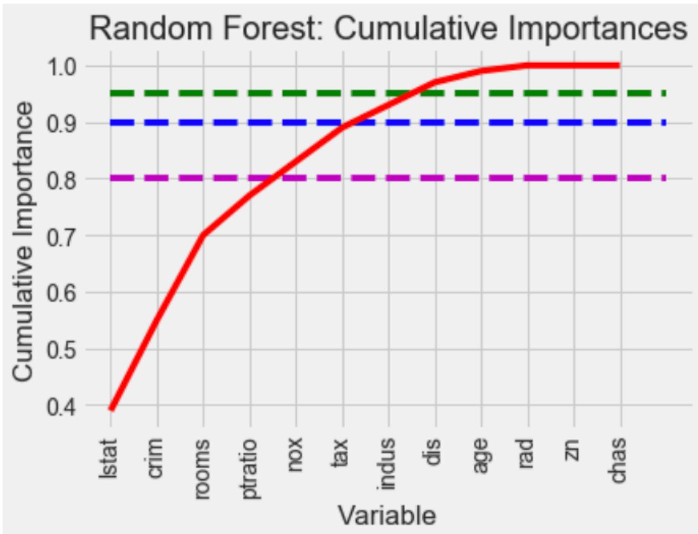
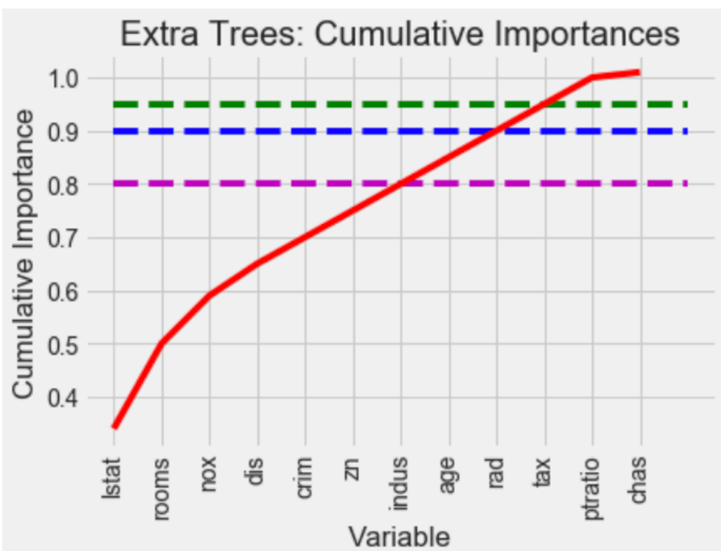


Figure 19: Variable Importance for Extra Trees Regression (Koehrsen 2018)



In [65]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV, Ridge,
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import seaborn as sns
import scipy.stats as stats
from sklearn.model_selection import KFold, GridSearchCV, cross_validate, cr
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
import math
from sklearn.metrics import mean_squared_error
from sklearn.tree import export_graphviz
import pydot
from sklearn.linear_model import SGDRegressor
```

In [66]:
```
# 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
```

In [67]:
```
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

In [68]:
```
boston_in = pd.read_csv('boston.csv')
boston_in = boston_in.drop("neighborhood", axis = 1)
boston_in['log_mv']=np.log(boston_in['mv'])
boston_in.head()
```

Out[68]:

	crim	zn	indus	chas	nox	rooms	age	dis	rad	tax	ptratio	lstat	mv	log_mv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0	3.178054
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6	3.072693
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7	3.546740
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4	3.508556
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2	3.589059

In [69]:
```
boston_train, boston_test = train_test_split(boston_in, test_size = 0.3, ra
```

In [70]:
```
boston_train.head()
```

Out[70]:

	crim	zn	indus	chas	nox	rooms	age	dis	rad	tax	ptratio	lstat	mv	log_r
13	0.62976	0.0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21.0	8.26	20.4	3.0155
61	0.17171	25.0	5.13	0	0.453	5.966	93.4	6.8185	8	284	19.7	14.44	16.0	2.7725
377	9.82349	0.0	18.10	0	0.671	6.794	98.8	1.3580	24	666	20.2	21.24	13.3	2.5877
39	0.02763	75.0	2.95	0	0.428	6.595	21.8	5.4011	3	252	18.3	4.32	30.8	3.4275
365	4.55587	0.0	18.10	0	0.718	3.561	87.9	1.6132	24	666	20.2	7.12	27.5	3.3141

In [71]:
```
boston_in.head()
```

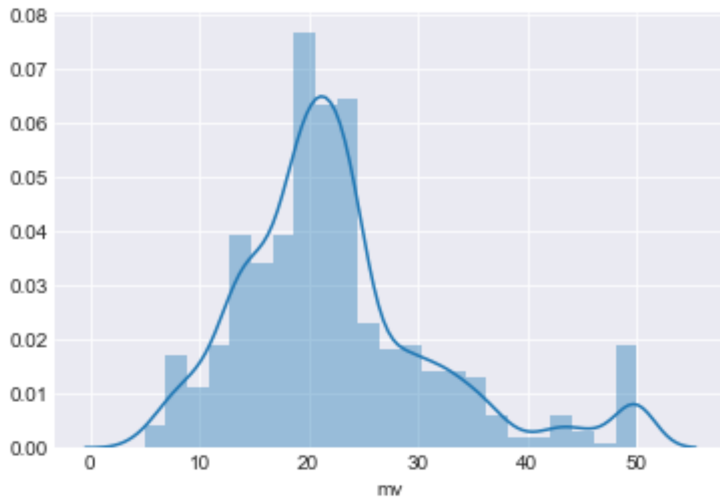
Out[71]:

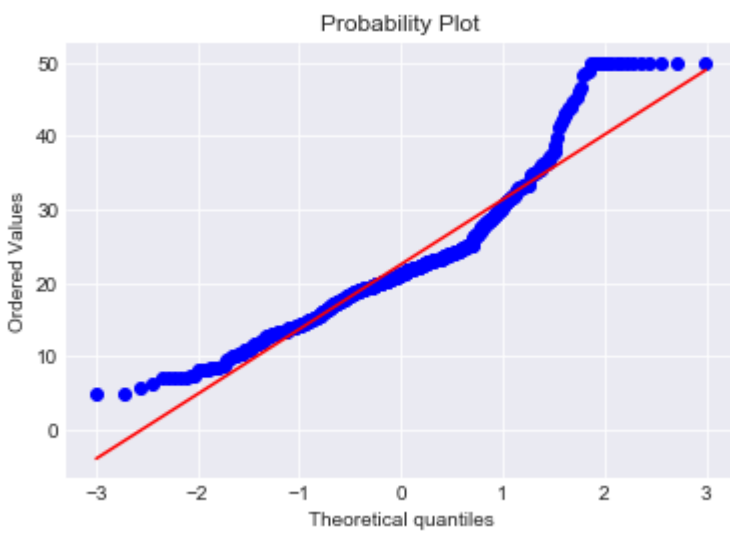
	crim	zn	indus	chas	nox	rooms	age	dis	rad	tax	ptratio	lstat	mv	log_mv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0	3.178054
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6	3.072693
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7	3.546740
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4	3.508556
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2	3.589059

In [72]:
```
features = pd.DataFrame(boston_in)
features = features.drop("mv", axis = 1)
features = features.drop("log_mv", axis = 1)
```

In [73]:
```
sns.distplot(boston_in['mv'], kde=True,);
fig = plt.figure()
res = stats.probplot(boston_in['mv'], plot=plt)
print("Skewness: %f" % boston_in['mv'].skew())
print("Kurtosis: %f" % boston_in['mv'].kurt())
```

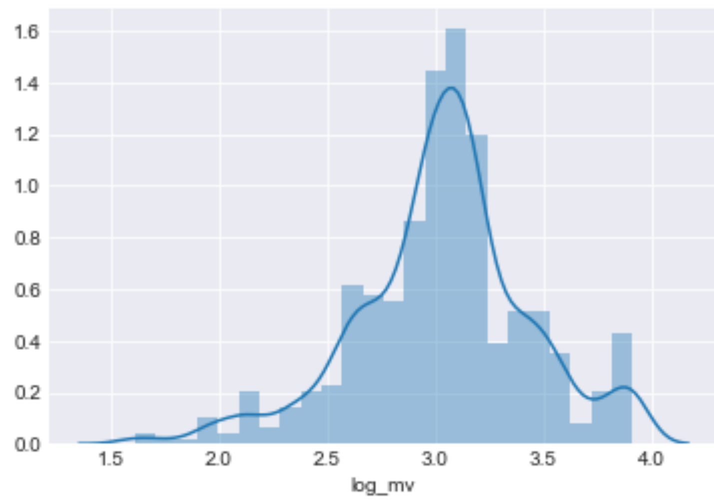
```
Skewness: 1.110912
Kurtosis: 1.516783
```

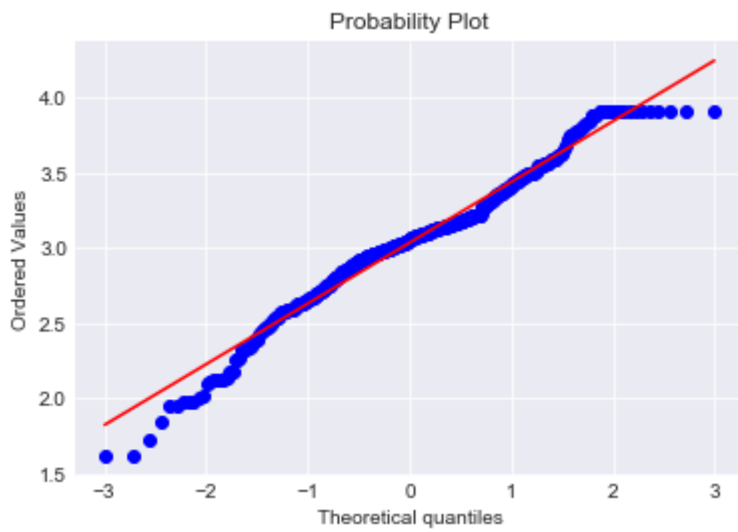




In [74]:
```
sns.distplot(boston_in['log_mv'], kde=True,);
fig = plt.figure()
res = stats.probplot(boston_in['log_mv'], plot=plt)
print("Skewness: %f" % boston_in['log_mv'].skew())
print("Kurtosis: %f" % boston_in['log_mv'].kurt())
```

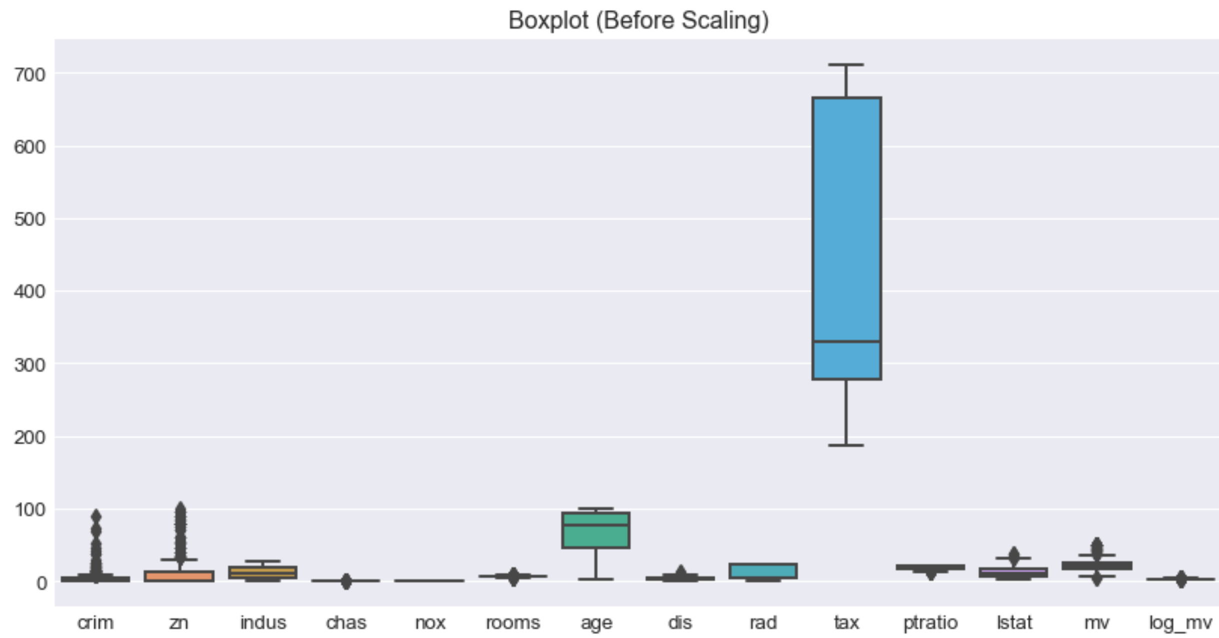
```
Skewness: -0.335226
Kurtosis: 0.827799
```





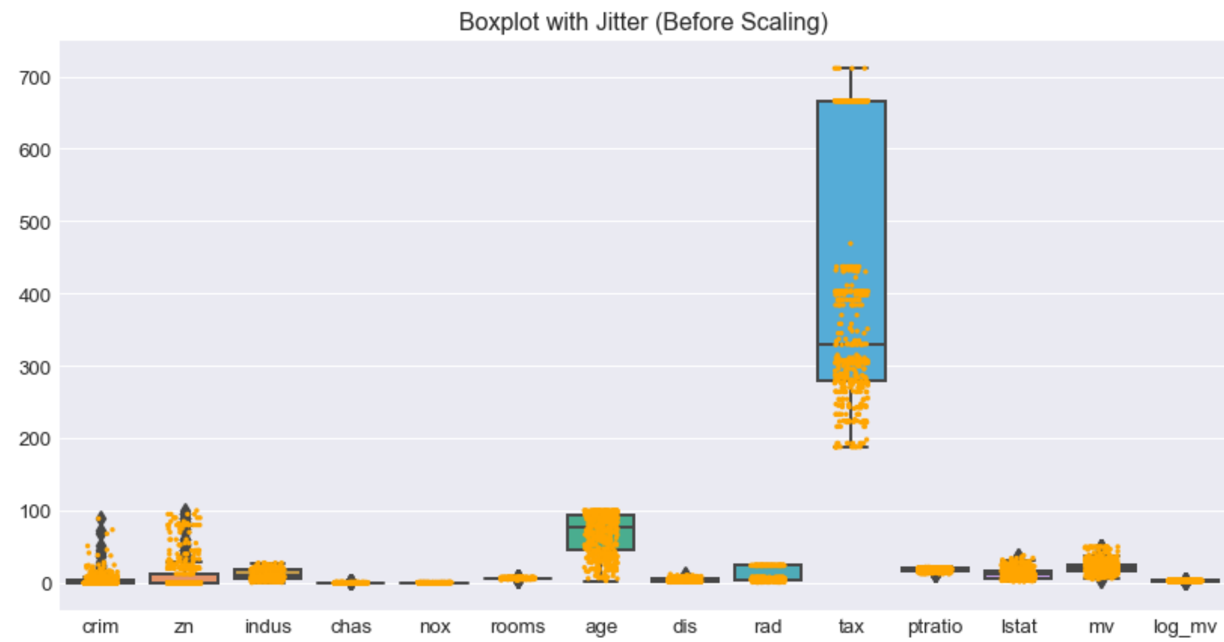
In [75]:
```
plt.style.use('seaborn-darkgrid')
my_dpi=96
plt.figure(figsize=(1000/my_dpi, 500/my_dpi), dpi=my_dpi)

ax = sns.boxplot( data=boston_in)
plt.title("Boxplot (Before Scaling)")
```

Out[75]:
```
Text(0.5, 1.0, 'Boxplot (Before Scaling)')
```



In [76]:
```
plt.style.use('seaborn-darkgrid')
my_dpi=96
plt.figure(figsize=(1000/my_dpi, 500/my_dpi), dpi=my_dpi)

ax = sns.boxplot( data=boston_in)
ax = sns.stripplot(data=boston_in, color="orange", jitter=0.2, size=2.5)
plt.title("Boxplot with Jitter (Before Scaling)")
```

Out[76]:
```
Text(0.5, 1.0, 'Boxplot with Jitter (Before Scaling)')
```



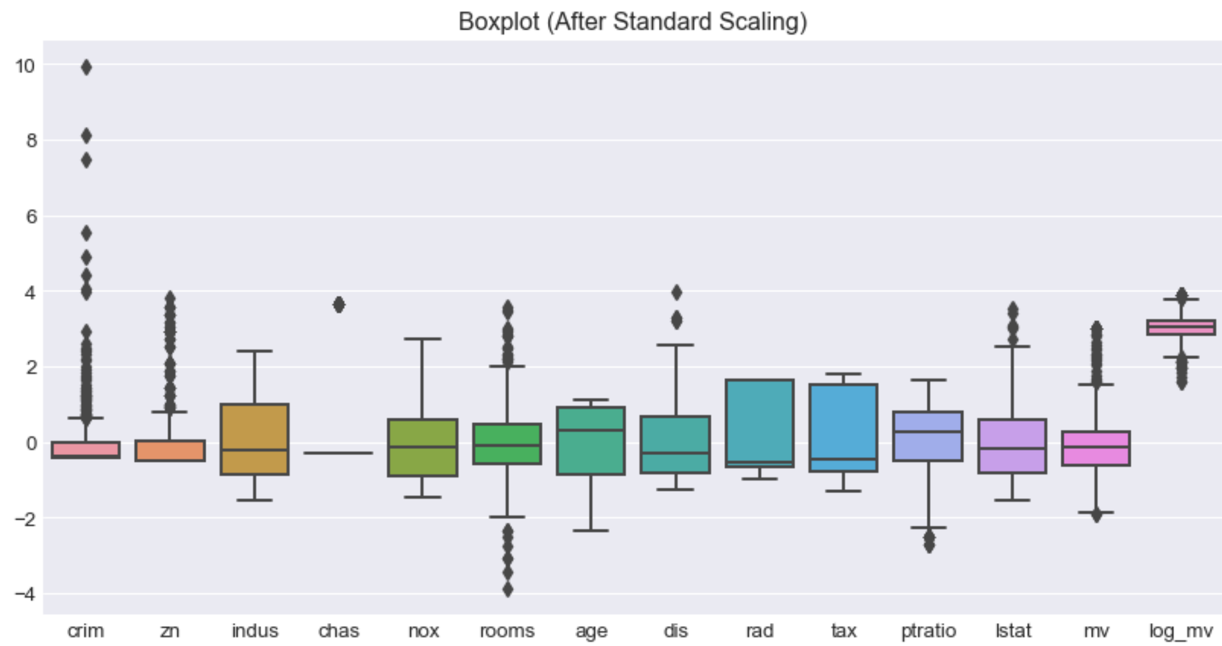
In [77]:
```
# standard scores for the columns... along axis 0
scaler = StandardScaler()
```

In [78]:
```
# the model data will be standardized form of preliminary model data
model_data= pd.DataFrame(boston_in)
model_data.mv = scaler.fit_transform(model_data.mv.values.reshape(-1,1))
model_data.mvlog = scaler.fit_transform(model_data.mv.values.reshape(-1,1))
model_data.crim = scaler.fit_transform(model_data.crim.values.reshape(-1,1)
model_data.zn = scaler.fit_transform(model_data.zn.values.reshape(-1,1))
model_data.indus = scaler.fit_transform(model_data.indus.values.reshape(-1,
model_data.chas = scaler.fit_transform(model_data.chas.values.reshape(-1,1)
model_data.nox = scaler.fit_transform(model_data.nox.values.reshape(-1,1))
model_data.rooms = scaler.fit_transform(model_data.rooms.values.reshape(-1,
model_data.age = scaler.fit_transform(model_data.age.values.reshape(-1,1))
model_data.dis = scaler.fit_transform(model_data.dis.values.reshape(-1,1))
model_data.rad = scaler.fit_transform(model_data.rad.values.reshape(-1,1))
model_data.tax = scaler.fit_transform(model_data.tax.values.reshape(-1,1))
model_data.ptratio = scaler.fit_transform(model_data.ptratio.values.reshape
model_data.lstat = scaler.fit_transform(model_data.lstat.values.reshape(-1,
```

```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/ipykernel_
launcher.py:4: UserWarning: Pandas doesn't allow columns to be created vi
a a new attribute name - see https://pandas.pydata.org/pandas-docs/stabl
e/indexing.html#attribute-access (https://pandas.pydata.org/pandas-docs/s
table/indexing.html#attribute-access)
  after removing the cwd from sys.path.
```

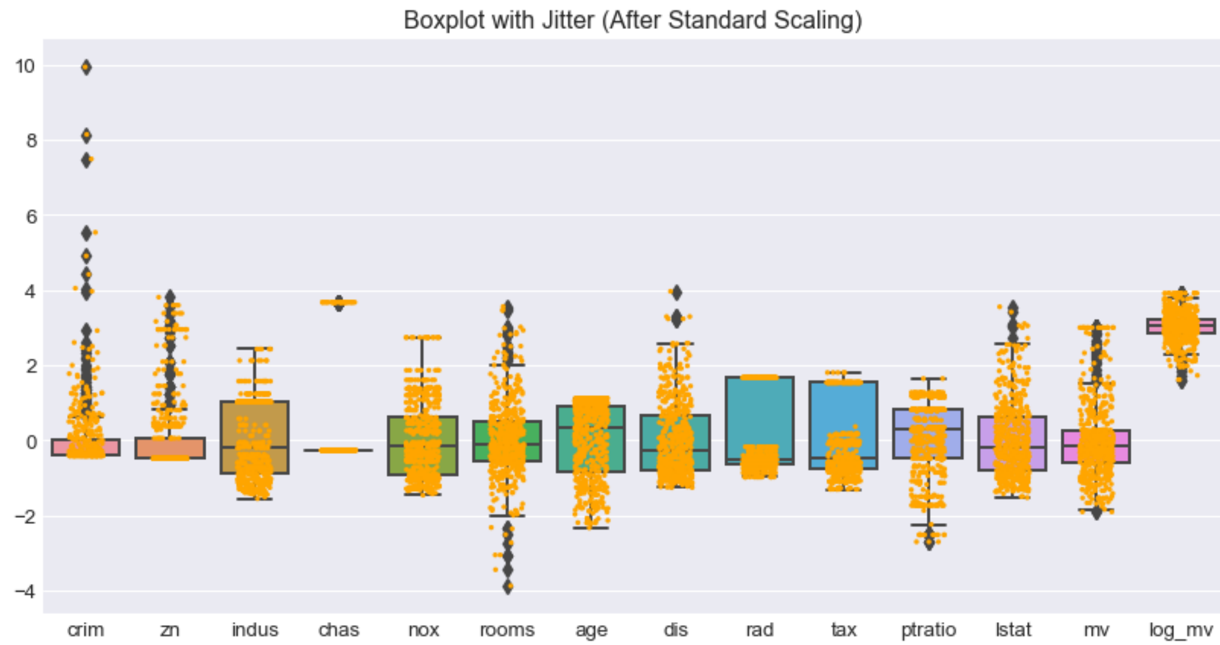
In [79]:
```
plt.style.use('seaborn-darkgrid')
my_dpi=96
plt.figure(figsize=(1000/my_dpi, 500/my_dpi), dpi=my_dpi)

ax = sns.boxplot( data=model_data)
plt.title("Boxplot (After Standard Scaling)")
```

Out[79]: Text(0.5, 1.0, 'Boxplot (After Standard Scaling)')



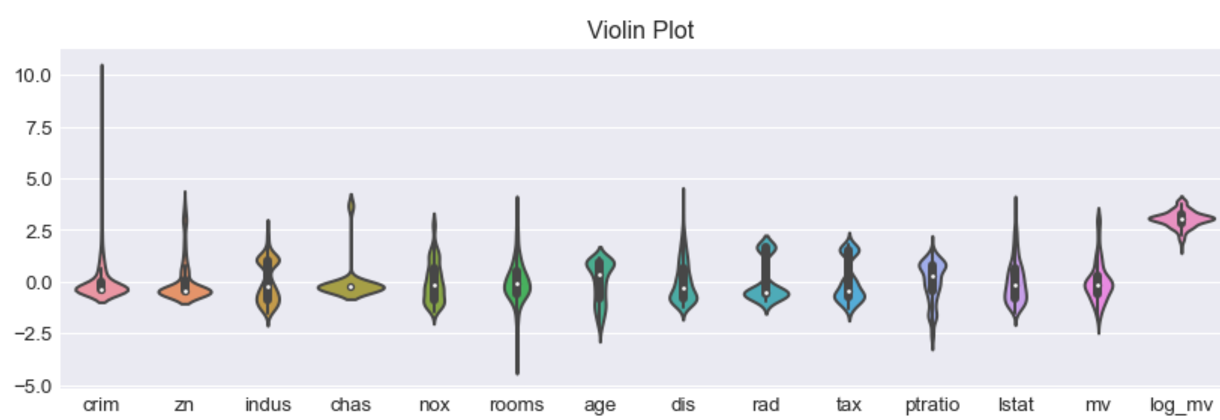
In [80]:
```
plt.style.use('seaborn-darkgrid')
my_dpi=96
plt.figure(figsize=(1000/my_dpi, 500/my_dpi), dpi=my_dpi)

ax = sns.boxplot( data=boston_in)
ax = sns.stripplot(data=boston_in, color="orange", jitter=0.2, size=2.5)
plt.title("Boxplot with Jitter (After Standard Scaling)")
```

Out[80]: `Text(0.5, 1.0, 'Boxplot with Jitter (After Standard Scaling)')`



In [81]:
```
plt.style.use('seaborn-darkgrid')
my_dpi=96
plt.figure(figsize=(1000/my_dpi, 300/my_dpi), dpi=my_dpi)
sns.violinplot(data=model_data)
plt.title("Violin Plot")
```

Out[81]: `Text(0.5, 1.0, 'Violin Plot')`



In [82]:
```
relevant = pd.DataFrame(data = model_data, columns = [
    "lstat", "rooms", "ptratio", "indus",
    "tax", "nox", "crim", "rad", "age", "zn", "log_mv"])
```

In [ ]:

In [83]:
```python
plt.figure(figsize=(15,15))
cor = relevant.corr()
cor
```

Out[83]:

	lstat	rooms	ptratio	indus	tax	nox	crim	rad	
lstat	1.000000	-0.613808	0.374044	0.603800	0.543993	0.590879	0.455621	0.488676	0.60
rooms	-0.613808	1.000000	-0.355501	-0.391676	-0.292048	-0.302188	-0.219247	-0.209847	-0.24
ptratio	0.374044	-0.355501	1.000000	0.383248	0.460853	0.188933	0.289946	0.464741	0.26
indus	0.603800	-0.391676	0.383248	1.000000	0.720760	0.763651	0.406583	0.595129	0.64
tax	0.543993	-0.292048	0.460853	0.720760	1.000000	0.668023	0.582764	0.910228	0.50
nox	0.590879	-0.302188	0.188933	0.763651	0.668023	1.000000	0.420972	0.611441	0.73
crim	0.455621	-0.219247	0.289946	0.406583	0.582764	0.420972	1.000000	0.625505	0.35
rad	0.488676	-0.209847	0.464741	0.595129	0.910228	0.611441	0.625505	1.000000	0.45
age	0.602339	-0.240265	0.261515	0.644779	0.506456	0.731470	0.352734	0.456022	1.00
zn	-0.412995	0.311991	-0.391679	-0.533828	-0.314563	-0.516604	-0.200469	-0.311948	-0.56
log_mv	-0.809234	0.632536	-0.499433	-0.543195	-0.566214	-0.513431	-0.530001	-0.486818	-0.45

```
<Figure size 1080x1080 with 0 Axes>
```

In [ ]:

In [ ]:

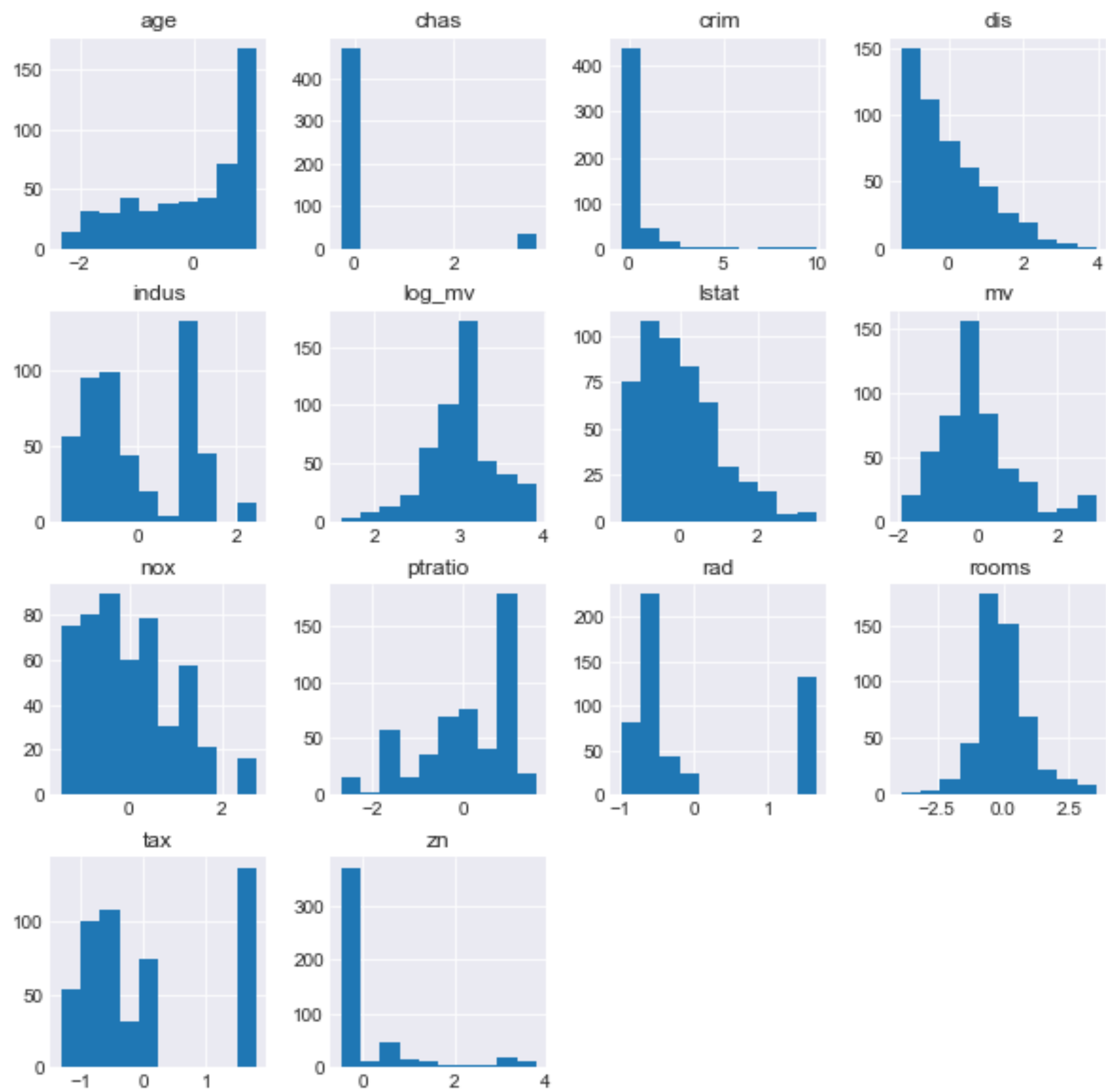
In [84]:
```python
relevant = relevant.drop("rad", axis =1)
cor = relevant.corr() # Whole correlation matrix
cor
```

Out[84]:

	lstat	rooms	ptratio	indus	tax	nox	crim	age	
lstat	1.000000	-0.613808	0.374044	0.603800	0.543993	0.590879	0.455621	0.602339	-0.41
rooms	-0.613808	1.000000	-0.355501	-0.391676	-0.292048	-0.302188	-0.219247	-0.240265	0.31
ptratio	0.374044	-0.355501	1.000000	0.383248	0.460853	0.188933	0.289946	0.261515	-0.39
indus	0.603800	-0.391676	0.383248	1.000000	0.720760	0.763651	0.406583	0.644779	-0.53
tax	0.543993	-0.292048	0.460853	0.720760	1.000000	0.668023	0.582764	0.506456	-0.31
nox	0.590879	-0.302188	0.188933	0.763651	0.668023	1.000000	0.420972	0.731470	-0.51
crim	0.455621	-0.219247	0.289946	0.406583	0.582764	0.420972	1.000000	0.352734	-0.20
age	0.602339	-0.240265	0.261515	0.644779	0.506456	0.731470	0.352734	1.000000	-0.56
zn	-0.412995	0.311991	-0.391679	-0.533828	-0.314563	-0.516604	-0.200469	-0.569537	1.00
log_mv	-0.809234	0.632536	-0.499433	-0.543195	-0.566214	-0.513431	-0.530001	-0.455029	0.36

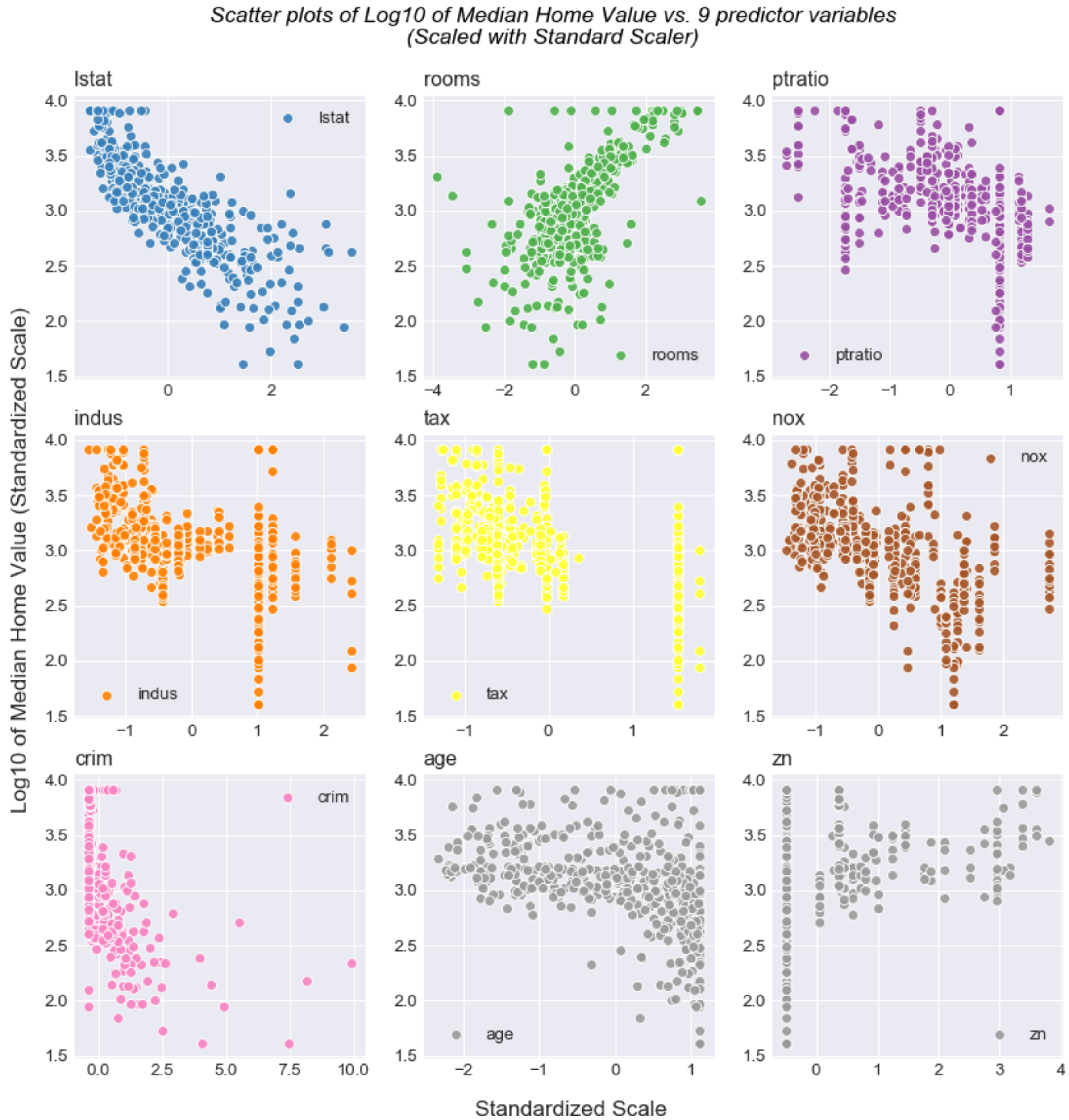
In [85]:
```
model_data.hist(figsize=(10,10))
```

Out[85]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1339dd590>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x133f65c90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x133fa34d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x133fd8cd0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x134019510>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x13404fd10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x13408f550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1340c3d50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1340cc8d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x134110290>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x13417b5d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1341afdd0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1341f1610>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x134224e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x134268650>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x134299e50>]],
      dtype=object)
```



In [86]:
```
# Initialize the figure
plt.style.use('seaborn-darkgrid')
my_dpi=96
plt.figure(figsize=(1000/my_dpi, 1000/my_dpi), dpi=my_dpi)

# create a color palette
palette = plt.get_cmap('Set1')

# multiple line plot
num=0
for column in relevant.drop('log_mv', axis=1):
    num+=1

    # Find the right spot on the plot
    plt.subplot(3,3, num)

    # Plot the lineplot
    #plt.plot( y=relevant["mv", relevant[column]])
    sns.scatterplot(relevant[column], relevant['log_mv'], color=palette(num
    #plt.plot(relevant['mv'], relevant[column], marker='', color=palette(nu


    # Not ticks everywhere
    if num in range(10) :
        #plt.tick_params(labelbottom='off')
        plt.ylabel('')
        plt.xlabel('')
    if num not in [1,4,7] :
        plt.tick_params(labelleft='off')

    # Add title
    plt.title(column, loc='left', fontsize=12, fontweight=0 )

# general title
plt.suptitle("Scatter plots of Log10 of Median Home Value vs. 9 predictor v

# Axis title
plt.text(-4, 1, 'Standardized Scale', ha='center', va='center',fontsize = 1
plt.text(-13, 6, 'Log10 of Median Home Value (Standardized Scale)', ha='cen
```

Out[86]:
```
Text(-13, 6, 'Log10 of Median Home Value (Standardized Scale)')
```


Scatter plots of Log10 of Median Home Value vs. 9 predictor variables
(Scaled with Standard Scaler)
lstat
rooms
ptratio
indus
tax
nox
crim
age
zn
Log10 of Median Home Value (Standardized Scale)
Standardized Scale

```
In [ ]:
```

In [87]:
```
# Used Chris's code as a starting point
# Convert Train and Valid DF to nparrays
def mdl_df(boston_in):
    tmpTest = np.array([boston_in.log_mv,\
    boston_in.crim,\
    boston_in.zn,\
    boston_in.indus,\
    boston_in.chas,\
    boston_in.nox,\
    boston_in.rooms,\
    boston_in.age,\
    boston_in.dis,\
    boston_in.rad,\
    boston_in.tax,\
    boston_in.ptratio,\
    boston_in.lstat]).T
    return tmpTest

trn_data=mdl_df(boston_in = boston_train)
vld_data=mdl_df(boston_in = boston_test)

# scale data
y_col=0
scaler = StandardScaler()
print(scaler.fit(np.delete(trn_data, y_col, axis=1)))
print(scaler.mean_)
print(scaler.scale_)
trn_data_scl = scaler.transform(np.delete(trn_data, y_col, axis=1))
#print('\nDimensions for Training_data:', trn_data_scl.shape)

#Transform array to dataframe and plot post transformation
trn_data_scl_df = pd.DataFrame.from_records(trn_data_scl)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
[3.74292901e+00 1.13658192e+01 1.13053672e+01 8.47457627e-02
 5.55289831e-01 6.25343220e+00 6.88903955e+01 3.82747740e+00
 9.67796610e+00 4.09016949e+02 1.84511299e+01 1.29618362e+01]
[8.51249918e+00 2.34995718e+01 6.85002462e+00 2.78502995e-01
 1.17857260e-01 6.91682826e-01 2.82289533e+01 2.13428315e+00
 8.78964596e+00 1.69973061e+02 2.14406588e+00 7.26807435e+00]
```

In [88]:
```
trn_data_scl_df.head()
```

Out[88]:

	0	1	2	3	4	5	6	7	8
0	-0.365717	-0.483661	-0.462096	-0.30429	-0.146701	-0.440133	-0.251175	0.412327	-0.645983
1	-0.419527	0.580188	-0.901510	-0.30429	-0.867913	-0.415555	0.868243	1.401418	-0.190903
2	0.714310	-0.483661	0.991914	-0.30429	0.981782	0.781526	1.059536	-1.157052	1.629421
3	-0.436452	2.707887	-1.219757	-0.30429	-1.080034	0.493821	-1.668159	0.737307	-0.759754
4	0.095500	-0.483661	0.991914	-0.30429	1.380570	-3.892582	0.673408	-1.037481	1.629421

In [89]:
```
# Used Chris's code as a starting point
cv_train_data, cv_test_data = train_test_split(trn_data, test_size=0.3, ran
random_seed=1
scaler.fit(np.delete(cv_train_data, y_col, axis=1))
#Split Train and Test
y_col=0
y_train_cv=cv_train_data[:,y_col]
X_train_cv=scaler.transform(np.delete(cv_train_data, y_col, axis=1))
y_test_cv=cv_test_data[:,y_col]
X_test_cv=scaler.transform(np.delete(cv_test_data, y_col, axis=1))
y_valid=vld_data[:,y_col]
X_valid=scaler.transform(np.delete(vld_data, y_col, axis=1))
X_lbl = ['CRt' ,'Zon' ,'Ind' ,'Rvr' ,'Air' ,'NRm' ,'Age' ,'Dis' ,'Rad' ,'Ta
y_lbl=['Mhv']
```

In [90]:
```
X_test_cv.shape
```

Out[90]: (107, 12)

In [91]:
```
# Used Chris's code from Week 2 as a starting point
#Compare Classifer performance with and wo regularzation
models = ['Linear Regression', 'Ridge Regression', 'Lasso Regression', 'Ela
clfs = [LinearRegression(), Ridge(), Lasso(), ElasticNet(), RandomForestReg
params ={models[0]: {'fit_intercept':[True,False]},
            models[1]: {'alpha':[ 0.01, 0.1, 1, 10, 25, 75, 100], 'solver':
            models[2]: {'alpha':[ 0.05, 0.1,1,10, 100]},
            models[3]: {'alpha':[ 0.05, 0.1,1,10, 100],'copy_X':[True], 'fi
                     'max_iter':[1000], 'normalize':[False], 'positive':[F
                     'selection':['cyclic'], 'tol':[0.0001], 'warm_start':
            models[4]: {},
            models[5]: {},
            models[6]: {}}
test_scores = []
```

In [92]:
```python
# Used Chris's code from Week 2 as a starting point
for name, estimator in zip(models,clfs):
    print(name)
    clf = GridSearchCV(estimator, params[name], scoring='neg_mean_squared_e
    clf.fit(X_train_cv, y_train_cv)

    #print("Top paramaters: " + str(clf.best_params_))
    #coef = clf.best_estimator_.coef_
    #intrcept=clf.best_estimator_.intercept_
    tmp=clf.predict(X_valid)
    rmse = np.mean((clf.predict(X_train_cv)-y_train_cv)**2)
    rmse_tst = np.mean((clf.predict(X_test_cv)-y_test_cv)**2)
    rmse_vld = np.mean((clf.predict(X_valid)-y_valid)**2)

    print("RMSE: {:}".format(rmse))
    print("RMSE for test set: {:}".format(rmse_tst))
    print("RMSE for validation set: {:}".format(rmse_vld))
    print("")

    test_scores.append((name, rmse, clf.best_score_ , y_valid, clf.predict(
results = pd.DataFrame()
results = results.append(pd.DataFrame(test_scores, columns=['nm','rmse', 'b
                                                            'best','bparm',
```

```
Linear Regression
RMSE: 0.037726984403107115
RMSE for test set: 0.034086023085697
RMSE for validation set: 0.03821129661277439

Ridge Regression
RMSE: 0.03824561654081196
RMSE for test set: 0.03585347110268054
RMSE for validation set: 0.038124060616251176

Lasso Regression
RMSE: 0.05075948961357432
RMSE for test set: 0.0491308594806133
RMSE for validation set: 0.05238191268481157

Elastic Net Regression
RMSE: 0.045914019210583946
RMSE for test set: 0.0429687031234385
RMSE for validation set: 0.045706626917567404

Random Forest Regression



/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/sklearn/
model_selection/_search.py:814: DeprecationWarning: The default of the
`iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set size
s are unequal.
  DeprecationWarning)
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/sklearn/
```

```
model_selection/_search.py:814: DeprecationWarning: The default of the
`iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set size
s are unequal.
  DeprecationWarning)
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/sklearn/
model_selection/_search.py:814: DeprecationWarning: The default of the
`iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set size
s are unequal.
  DeprecationWarning)

RMSE: 0.004555358184212678
RMSE for test set: 0.03208605479363074
RMSE for validation set: 0.029289217285635265

Extra Trees Regression
RMSE: 1.4471765088189107e-31
RMSE for test set: 0.035254671726430105
RMSE for validation set: 0.031016146668477868

SGD Regression
RMSE: 0.039087170355104756
RMSE for test set: 0.03674581420392934
RMSE for validation set: 0.03747404884553792


/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/sklearn/mo
del_selection/_search.py:814: DeprecationWarning: The default of the `iid
` parameter will change from True to False in version 0.22 and will be re
moved in 0.24. This will change numeric results when test-set sizes are u
nequal.
  DeprecationWarning)
```

In [93]:
```
#Random Search from Lecture
RANDOM_SEED =1

names = models
N_FOLDs = 10
cv_results = np.zeros((N_FOLDs, len(names)))
kf = KFold(n_splits = N_FOLDs, shuffle = False, random_state = RANDOM_SEED)
index_for_fold = 0
index_for_method = 0
for name, reg_model in zip(names,clfs):
    reg_model.fit(X_train_cv, y_train_cv)

    y_test_predict = reg_model.predict(X_test_cv)
    fold_method_result = math.sqrt(mean_squared_error(y_test_cv, y_test_pre
    cv_results[index_for_fold,index_for_method] = fold_method_result
    index_for_method += 1

index_for_fold += 0
cv_results_df = pd.DataFrame(cv_results)
cv_results_df.columns = names

print("     Method           Root Mean Sq. Error")
print(cv_results_df.mean())
```

```
     Method           Root Mean Sq. Error
Linear Regression           0.018462
Ridge Regression            0.018516
Lasso Regression            0.041755
Elastic Net Regression      0.041755
Random Forest Regression    0.018272
Extra Trees Regression      0.016439
SGD Regression              0.018795
dtype: float64
```

In [94]:
```
rmse = np.mean((clf.predict(X_train_cv)-y_train_cv)**2)
rmse_tst = np.mean((clf.predict(X_test_cv)-y_test_cv)**2)
rmse_vld = np.mean((clf.predict(X_valid)-y_valid)**2)
```

In [95]:
```
# Grid Search for Extra Trees Regression
```

In [96]:
```
rf = RandomForestRegressor()
etr = ExtraTreesRegressor()

param_grid = {
    'bootstrap': [True],
    'max_depth': [50,75,100],
    'max_features': [2,3,4,5],
    'min_samples_leaf': [3,4,5],
    'min_samples_split': [5,7,10],
    'n_estimators': [10, 25, 50, 100]
}
```

In [97]:
```
grid_search_rf = GridSearchCV(estimator = rf, param_grid = param_grid, cv=3
grid_search_rf.fit(X_train_cv, y_train_cv)
```

```
Fitting 3 folds for each of 432 candidates, totalling 1296 fits
[CV] bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10
[CV]  bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10, total=   0.0s
[CV] bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10
[CV]  bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10, total=   0.0s
[CV] bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10
[CV]  bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10, total=   0.0s
[CV] bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=25
[CV]  bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=25, total=   0.0s
[CV] bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=25
```

In [98]:
```
grid_search_etr = GridSearchCV(estimator = etr, param_grid = param_grid, cv
grid_search_etr.fit(X_train_cv, y_train_cv)
```

```
Fitting 3 folds for each of 432 candidates, totalling 1296 fits
[CV] bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10
[CV]  bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10, total=   0.0s
[CV] bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10
[CV]  bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10, total=   0.0s
[CV] bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10
[CV]  bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=10, total=   0.0s
[CV] bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=25
[CV]  bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=25, total=   0.0s
[CV] bootstrap=True, max_depth=50, max_features=2, min_samples_leaf=3,
min_samples_split=5, n_estimators=25
```

In [99]:
```
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors/test_labels)
    accuracy = 100 - mape
    return accuracy

print(grid_search_rf.best_params_)
best_grid_rf = grid_search_rf.best_estimator_
grid_accuracy_rf = evaluate(best_grid_rf, X_test_cv, y_test_cv)
```

```
{'bootstrap': True, 'max_depth': 75, 'max_features': 4, 'min_samples_lea
f': 3, 'min_samples_split': 7, 'n_estimators': 25}
```

In [100]:
```
print(grid_search_etr.best_params_)
best_grid_etr = grid_search_etr.best_estimator_
grid_accuracy_etr = evaluate(best_grid_etr, X_test_cv, y_test_cv)
```

```
{'bootstrap': True, 'max_depth': 50, 'max_features': 5, 'min_samples_lea
f': 3, 'min_samples_split': 5, 'n_estimators': 25}
```

In [101]:
```
best_grid_rf
```

Out[101]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=75,
                      max_features=4, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=3, min_samples_split=7,
                      min_weight_fraction_leaf=0.0, n_estimators=25,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

In [102]:
```
grid_accuracy_rf
```

Out[102]:
```
95.68596161203726
```

In [103]:
```
best_grid_etr
```

Out[103]:
```
ExtraTreesRegressor(bootstrap=True, criterion='mse', max_depth=50,
                    max_features=5, max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=3, min_samples_split=5,
                    min_weight_fraction_leaf=0.0, n_estimators=25, n_jobs
=None,
                    oob_score=False, random_state=None, verbose=0,
                    warm_start=False)
```

In [104]:
```
grid_accuracy_etr
```

Out[104]:
```
95.5496667737536
```

In [105]:
```
feature_list = list(features.columns)
```

In [106]:
```
importances_rf = list(best_grid_rf.feature_importances_)
feature_importances_rf = [(feature, round(importance, 2)) for feature, impo
feature_importances_rf = sorted(feature_importances_rf, key = lambda x: x[1
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_
```

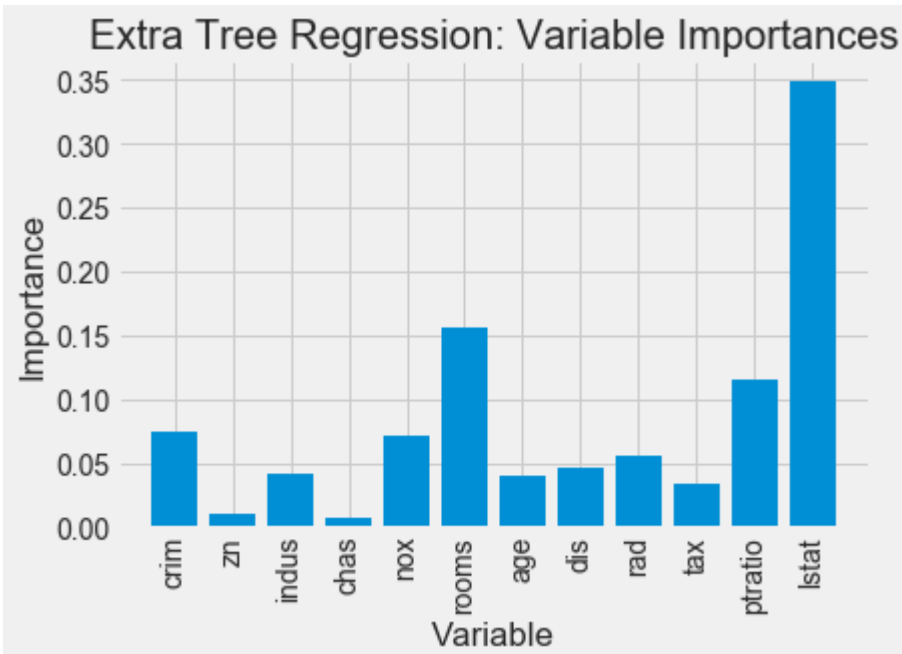
```
Variable: lstat                Importance: 0.37
Variable: rooms                Importance: 0.17
Variable: nox                  Importance: 0.16
Variable: crim                 Importance: 0.1
Variable: age                  Importance: 0.07
Variable: ptratio              Importance: 0.04
Variable: indus                Importance: 0.03
Variable: dis                  Importance: 0.03
Variable: rad                  Importance: 0.02
Variable: tax                  Importance: 0.01
Variable: zn                   Importance: 0.0
Variable: chas                 Importance: 0.0
```

In [107]:
```
importances_etr = list(best_grid_etr.feature_importances_)
feature_importances_etr = [(feature, round(importance, 2)) for feature, imp
feature_importances_etr = sorted(feature_importances_etr, key = lambda x: x
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_
```

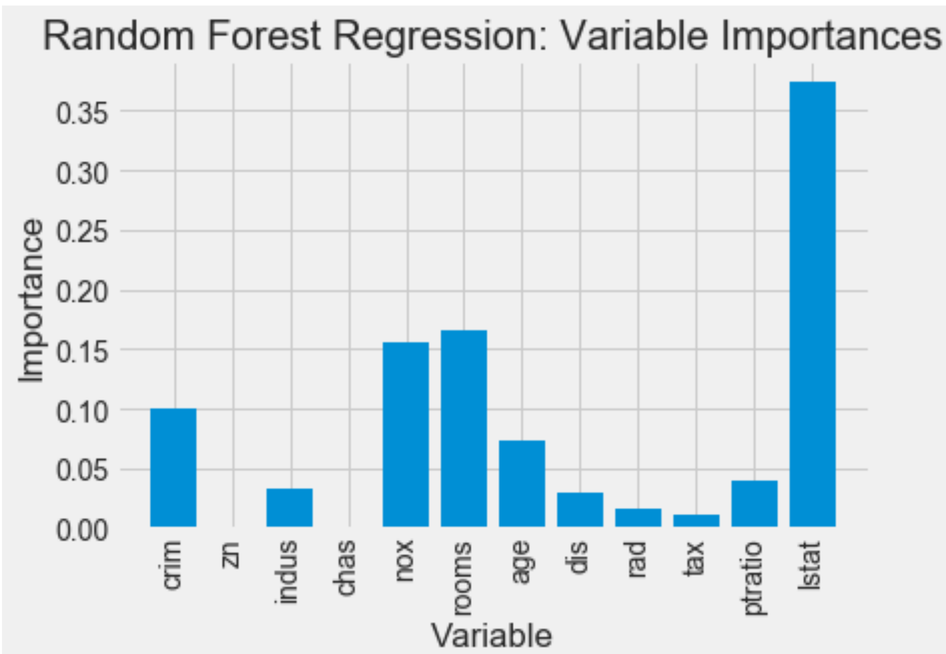
```
Variable: lstat                Importance: 0.35
Variable: rooms                Importance: 0.16
Variable: ptratio              Importance: 0.12
Variable: crim                 Importance: 0.07
Variable: nox                  Importance: 0.07
Variable: rad                  Importance: 0.06
Variable: dis                  Importance: 0.05
Variable: indus                Importance: 0.04
Variable: age                  Importance: 0.04
Variable: tax                  Importance: 0.03
Variable: zn                   Importance: 0.01
Variable: chas                 Importance: 0.01
```

In [108]:

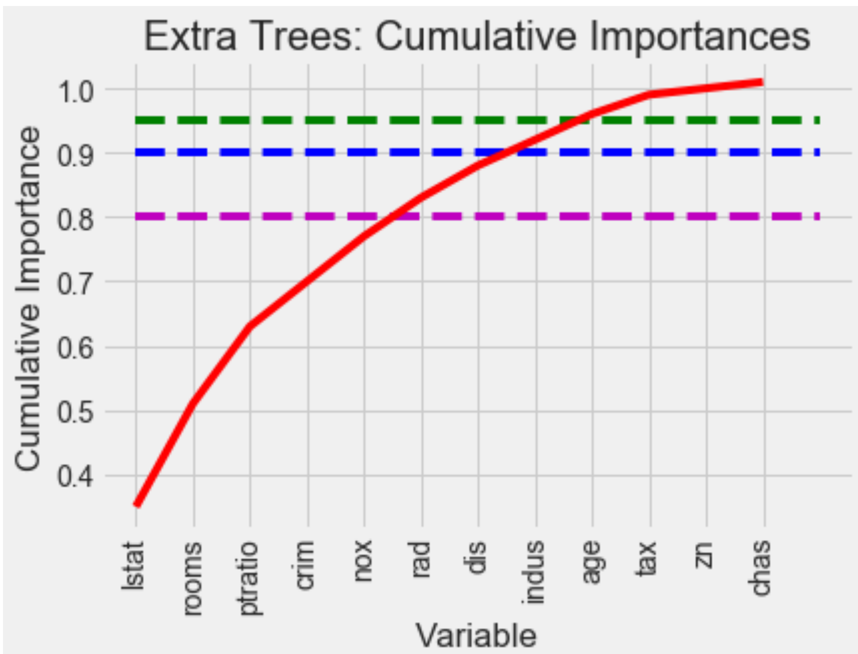
```
# Import matplotlib for plotting and use magic command for Jupyter Notebook
import matplotlib.pyplot as plt
%matplotlib inline
# Set the style
plt.style.use('fivethirtyeight')
# list of x locations for plotting
x_values = list(range(len(importances_etr)))
# Make a bar chart
plt.bar(x_values, importances_etr, orientation = 'vertical')
# Tick labels for x axis
plt.xticks(x_values, feature_list, rotation='vertical')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Extra Tree Reg
```



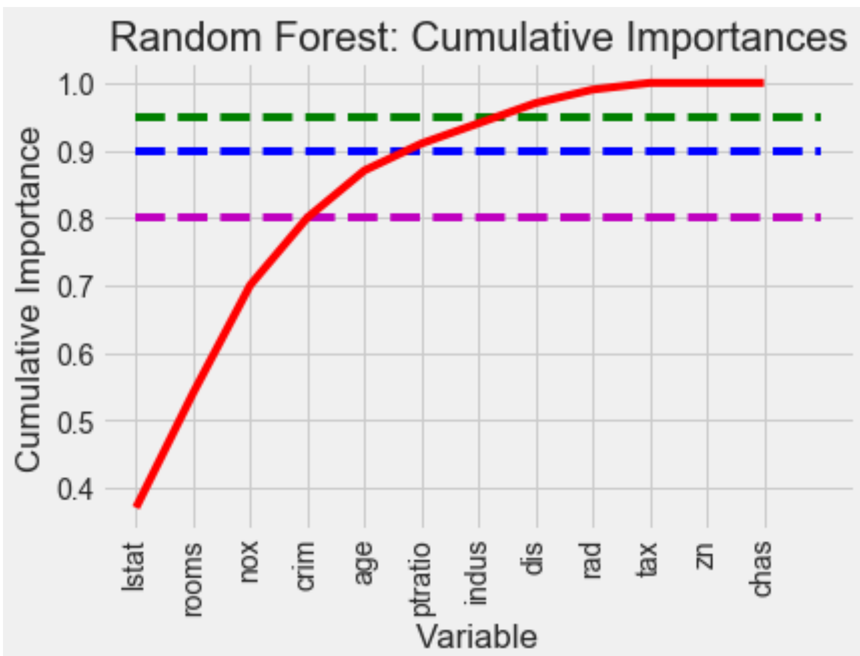
In [109]:
```
# Import matplotlib for plotting and use magic command for Jupyter Notebook
import matplotlib.pyplot as plt
%matplotlib inline
# Set the style
plt.style.use('fivethirtyeight')
# list of x locations for plotting
x_values = list(range(len(importances_rf)))
# Make a bar chart
plt.bar(x_values, importances_rf, orientation = 'vertical')
# Tick labels for x axis
plt.xticks(x_values, feature_list, rotation='vertical')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Random Forest
```



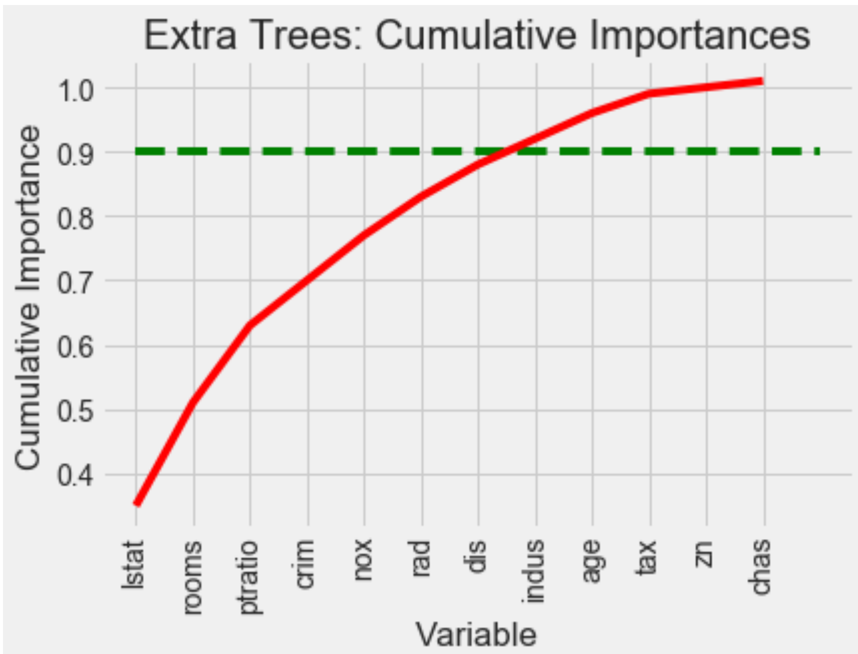
In [110]:
```
# List of features sorted from most to least important
sorted_importances = [importance[1] for importance in feature_importances_e
sorted_features = [importance[0] for importance in feature_importances_etr]
# Cumulative importances
cumulative_importances = np.cumsum(sorted_importances)
# Make a line graph
plt.plot(x_values, cumulative_importances, 'r-')
# Draw line at 95% of importance retained
plt.hlines(y = 0.80, xmin=0, xmax=len(sorted_importances), color = 'm', lin
plt.hlines(y = 0.95, xmin=0, xmax=len(sorted_importances), color = 'g', lin
plt.hlines(y = 0.90, xmin=0, xmax=len(sorted_importances), color = 'b', lin
# Format x ticks and labels
plt.xticks(x_values, sorted_features, rotation = 'vertical')
# Axis labels and title
plt.xlabel('Variable'); plt.ylabel('Cumulative Importance'); plt.title('Ext
```



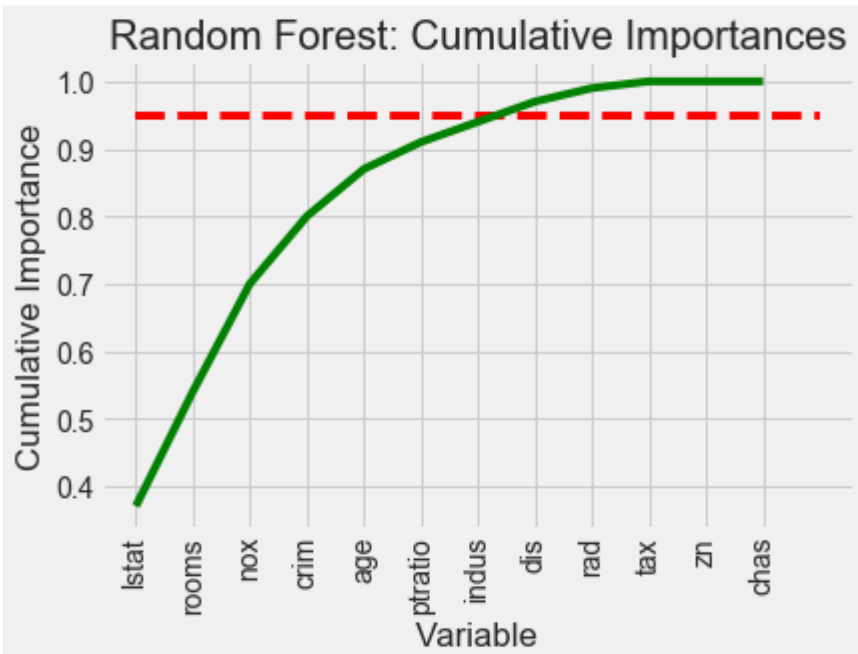
In [111]:
```
# List of features sorted from most to least important
sorted_importances = [importance[1] for importance in feature_importances_r
sorted_features = [importance[0] for importance in feature_importances_rf]
# Cumulative importances
cumulative_importances = np.cumsum(sorted_importances)
# Make a line graph
plt.plot(x_values, cumulative_importances, 'r-')
# Draw line at 95% of importance retained
plt.hlines(y = 0.80, xmin=0, xmax=len(sorted_importances), color = 'm', lin
plt.hlines(y = 0.95, xmin=0, xmax=len(sorted_importances), color = 'g', lin
plt.hlines(y = 0.90, xmin=0, xmax=len(sorted_importances), color = 'b', lin
# Format x ticks and labels
plt.xticks(x_values, sorted_features, rotation = 'vertical')
# Axis labels and title
plt.xlabel('Variable'); plt.ylabel('Cumulative Importance'); plt.title('Ran
```



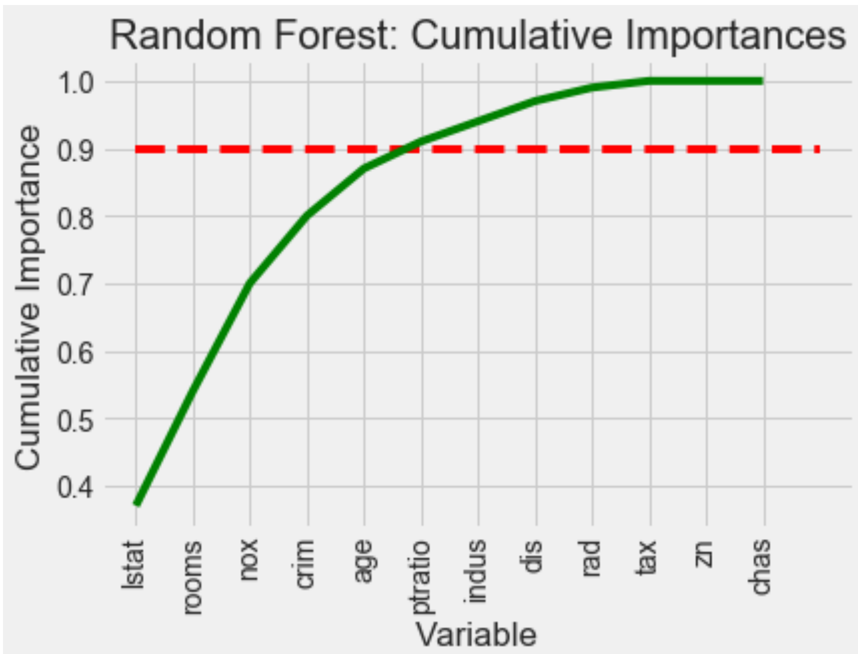
In [112]:
```
# List of features sorted from most to least important
sorted_importances = [importance[1] for importance in feature_importances_e
sorted_features = [importance[0] for importance in feature_importances_etr]
# Cumulative importances
cumulative_importances = np.cumsum(sorted_importances)
# Make a line graph
plt.plot(x_values, cumulative_importances, 'r-')
# Draw line at 95% of importance retained
plt.hlines(y = 0.90, xmin=0, xmax=len(sorted_importances), color = 'g', lin
# Format x ticks and labels
plt.xticks(x_values, sorted_features, rotation = 'vertical')
# Axis labels and title
plt.xlabel('Variable'); plt.ylabel('Cumulative Importance'); plt.title('Ext
```



In [113]:
```
# List of features sorted from most to least important
sorted_importances = [importance[1] for importance in feature_importances_r
sorted_features = [importance[0] for importance in feature_importances_rf]
# Cumulative importances
cumulative_importances = np.cumsum(sorted_importances)
# Make a line graph
plt.plot(x_values, cumulative_importances, 'g-')
# Draw line at 95% of importance retained
plt.hlines(y = 0.95, xmin=0, xmax=len(sorted_importances), color = 'r', lin
# Format x ticks and labels
plt.xticks(x_values, sorted_features, rotation = 'vertical')
# Axis labels and title
plt.xlabel('Variable'); plt.ylabel('Cumulative Importance'); plt.title('Ran
```



In [114]:
```
# List of features sorted from most to least important
sorted_importances = [importance[1] for importance in feature_importances_r
sorted_features = [importance[0] for importance in feature_importances_rf]
# Cumulative importances
cumulative_importances = np.cumsum(sorted_importances)
# Make a line graph
plt.plot(x_values, cumulative_importances, 'g-')
# Draw line at 90% of importance retained
plt.hlines(y = 0.90, xmin=0, xmax=len(sorted_importances), color = 'r', lin
# Format x ticks and labels
plt.xticks(x_values, sorted_features, rotation = 'vertical')
# Axis labels and title
plt.xlabel('Variable'); plt.ylabel('Cumulative Importance'); plt.title('Ran
```



In [ ]: