Allison Roeser

Boston Housing Study: Evaluating Regression Models

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 other 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 9). 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 10 & 11. The 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 and testing of hyper-parameters through Grid Search. Pandas and Seaborn were leveraged for exploratory data analysis and visualization.

Review Research Design and Modeling Methods:

Linear, Ridge, Lasso, and Elastic Net Regression models were run to build a model to accurately predict logmv. Ordinary Least Squares Regression (Linear Regression) does not use the process of regularization. Regularization is used to prevent overfitting by penalizing weighting particular variables too strongly. Lasso, Ridge and Elastic Net use hyperparameter called alpha to

penalize weights (Xu 2019). Using both regularized and non-regularized modeling provides a clear picture of how each variable impact the value for logmv. All variables were scaled using Standard Scaling to ensure proper weighting of variable importance. All 12 potential predictor variables were used in modeling, allowing each model to select variables to include or exclude. Train, test, and validation sets were created to cross-validate the models. Grid Search was used to select the best hyperparameters (such as the best value for alpha) for each model. Root mean squared error (RMSE) was used to score the models. RMSE is a measure of standard deviation of the residuals. Residuals measure the prediction error for each observation.

Review Results, Evaluate Models:

Linear Regression performed best with a RMSE of 0.377 (Figure 12) and retained all of the predictor variables (Figure 13). Lstat, nox, and dis were weighted the highest. These variables were deemed most important to the model. The linear regression equation is: **logmv = 3.03 -0.09crim + 0.03zn + 0.03indus + 0.03chas -0.14nox + 0.05rooms +0.01age – 0.12dis +0.1rad -0.06tax -0.09pratio -0.2lstat**. Lasso retained five predictor variables and Elastic Net kept six. While these models had a slightly higher RMSEs of 0.0508 and 0.0459 respectively, they had benefits of being less complex. The Lasso regression equation is: **logmv = 3.03 – 0.04crim - 0.01nox + 0.04rooms -0.05pratio - 0.2lstat**. All four models placed the most weight on lstat, which represents the proportion of homes of low socio-economic status. Additionally, there was little difference between the train, test, and validation RMSE for each model. This means that the models were not overfit and generalized well on new data (Figure 14).

Exposition, Problem Description and Management Recommendations:

When modeling housing prices in the Boston area, a regression model predicting log of median home value will provide the most accurate prediction. Linear regression provides slightly improved accuracy, but Lasso and Elastic Net Regression are easier to understand because they have fewer variables retained in the model. If management's objective is simply to predict the most accurate hosing price, Linear regression is best. If management's objective is to make strategy

changes based on which variables are the most impactful, Lasso and Elastic Net give fewer variables to consider. All models show that the population proportion of low socio-economic individuals 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.

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/

Tersakyan, A. Airbnb Price Prediction Using Linear Regression. (Oct 16, 2019). Towards Data Science. Retrieved from: https://towardsdatascience.com/airbnb-price-prediction-using-linear-regression-scikit-learn-and-statsmodels-6e1fc2bd51a6

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

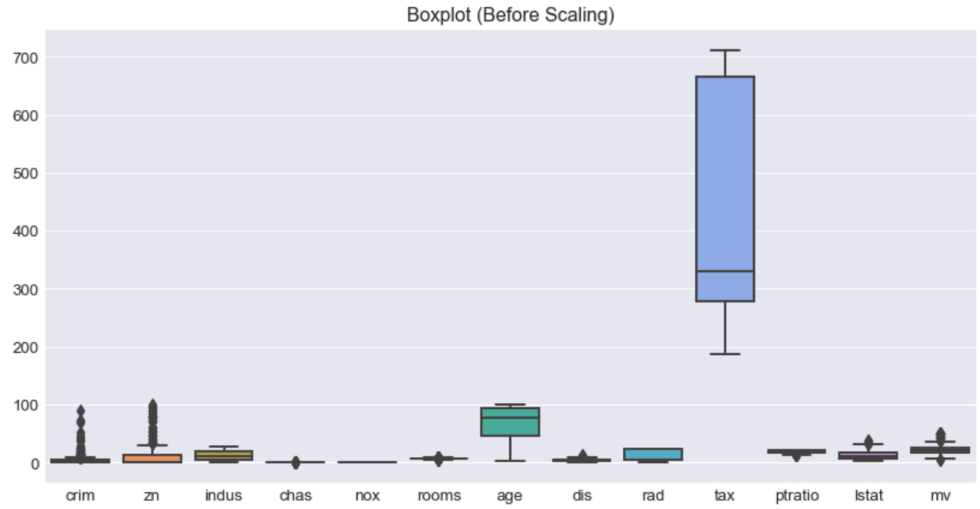


Figure 3: Boxplot of variables before scaling with jitter to show point distribution (Holtz)

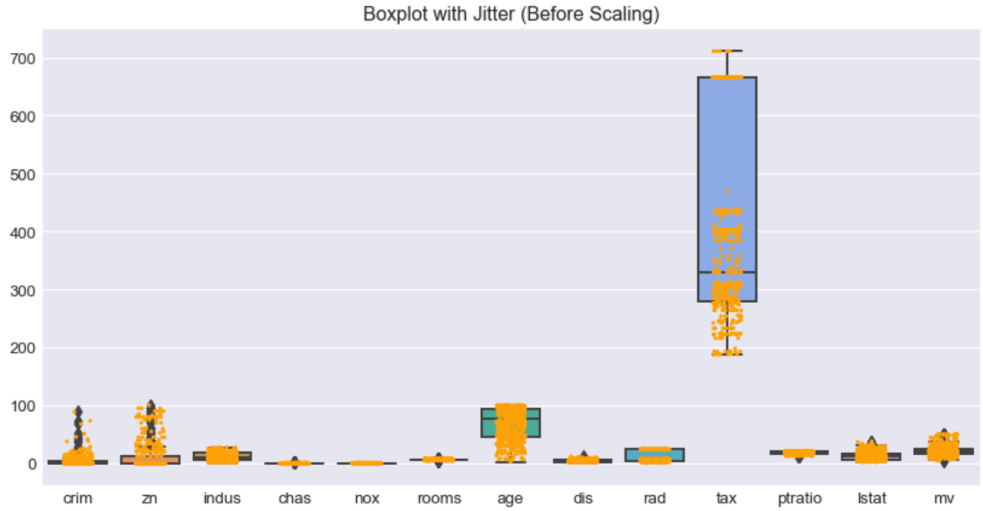


Figure 4: Boxplot after Standard Scaling

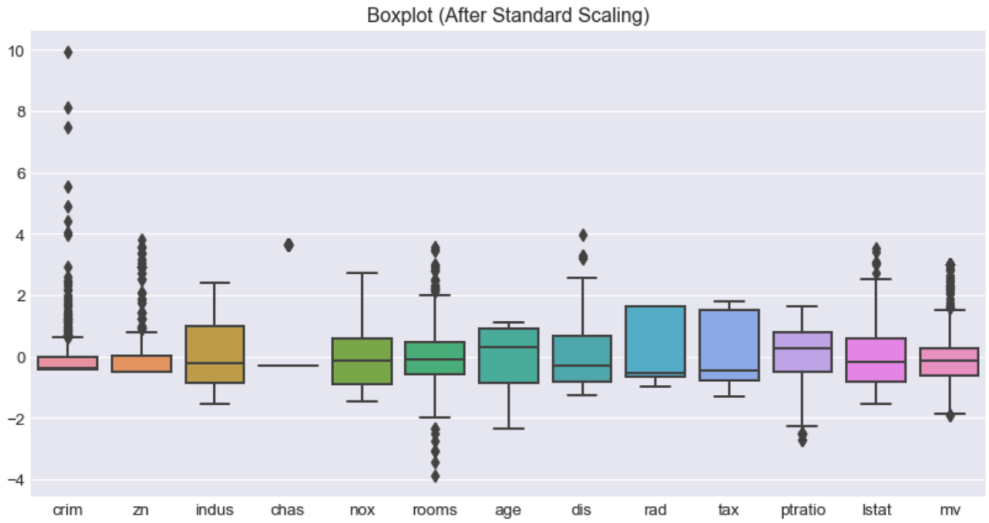


Figure 5: Boxplot with Jitter after Standard Scaling (Holtz)

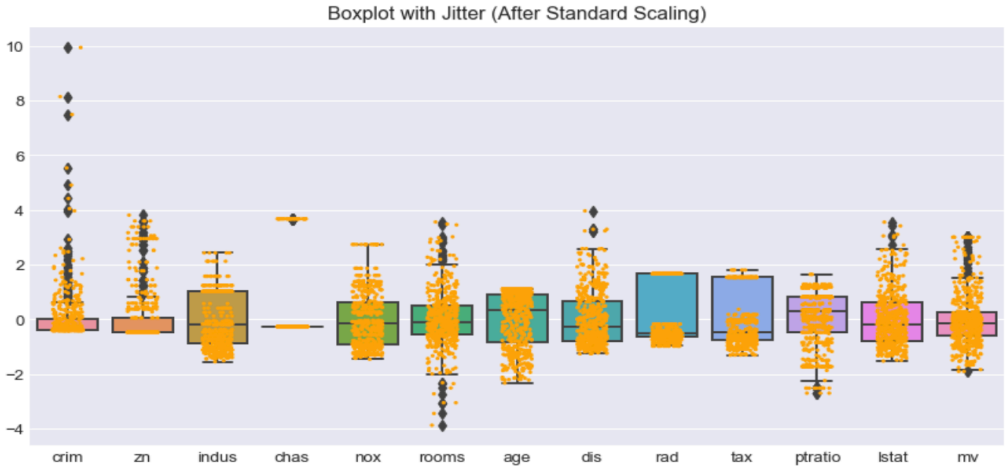


Figure 6: Violin plot after Standard Scaling. Lstat distribution is closest to the distribution of mv (Holtz)

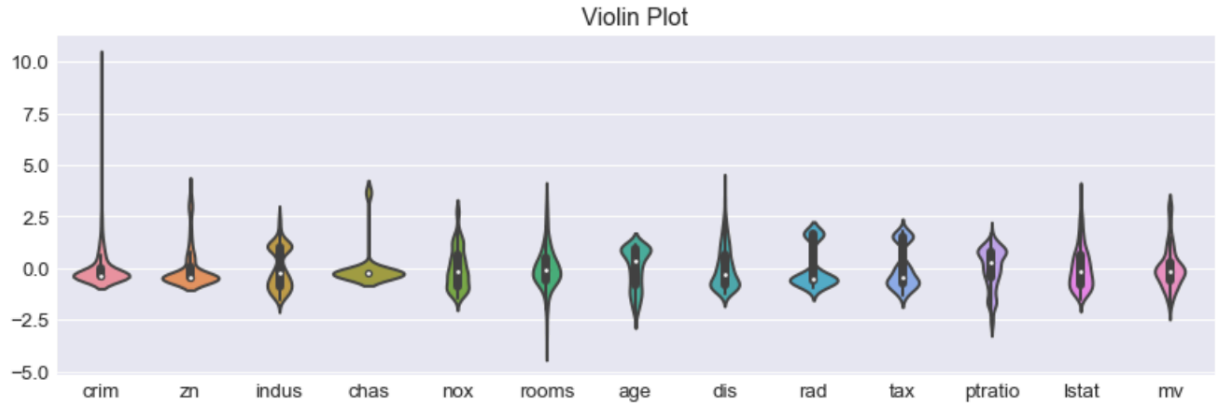
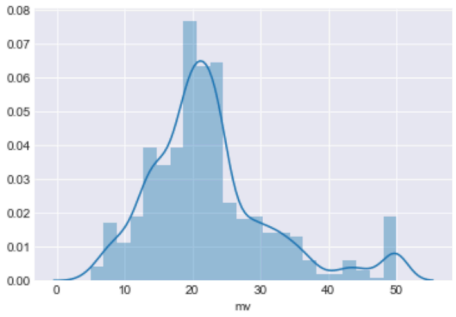


Figure 7: Density and probability plots for mv (Tersakyan 2019)

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



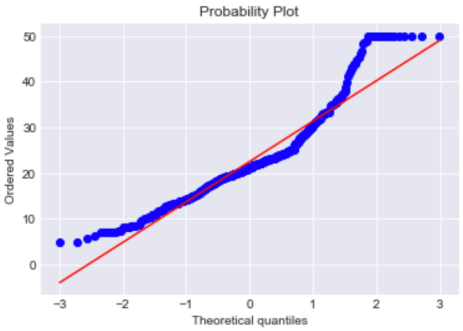


Figure 8: Density and probability plots for logmv (Tersakyan 2019)

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

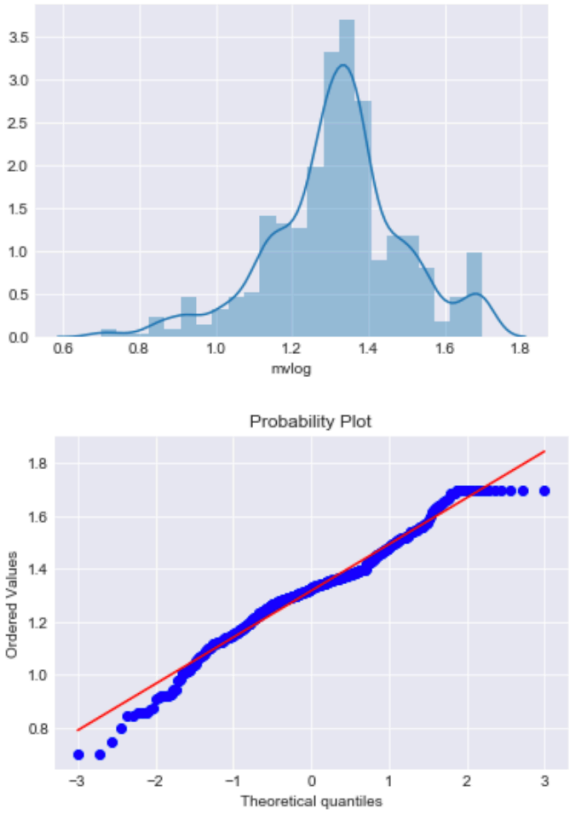


Figure 8: Variables whose correlation with mv is greater than 0.35

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 9: 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 10: Scatterplot matrix of mv vs. top 9 predictor variables (Holtz)

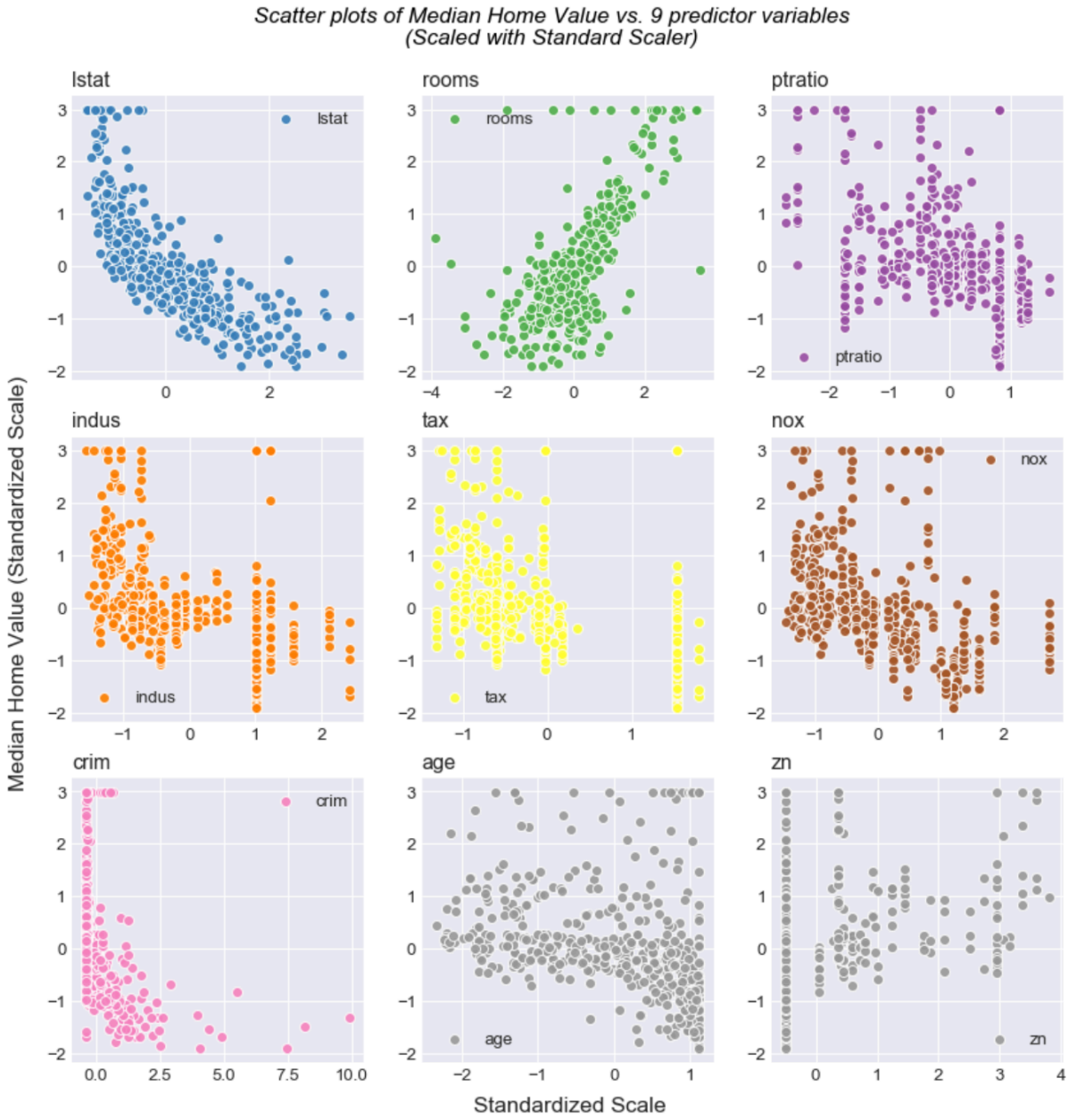


Figure 11: Scatterplot matrix of logmv vs. top 9 predictor variables (Holtz)

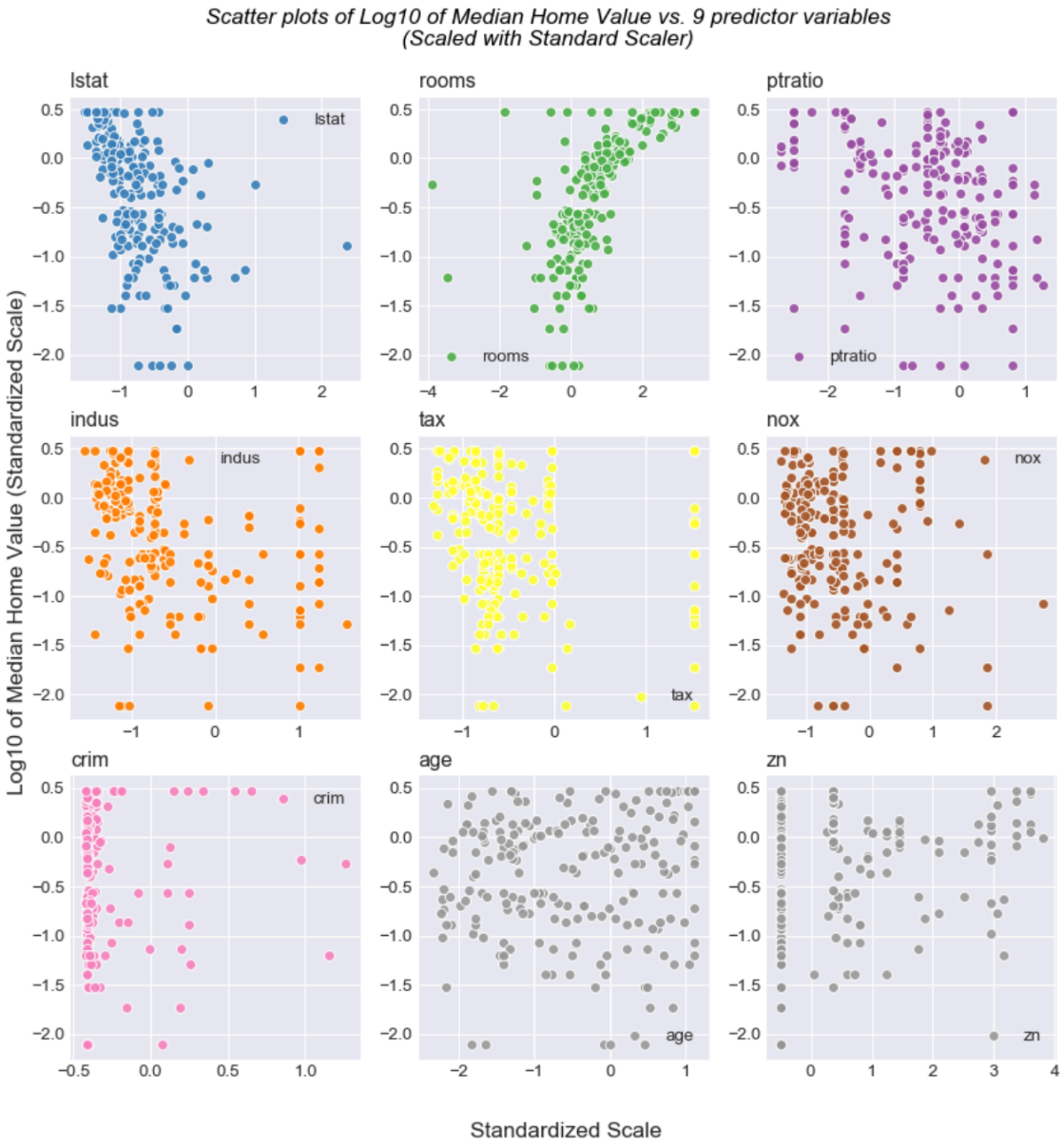


Figure 12: RMSE for the four regression models used to predict logMV

Model	RMSE
Linear Regression	0.037727
Ridge Regression	0.038246
Lasso Regression	0.050759
Elastic Net Regression	0.045914

Figure 13: Coefficients for each predictor variable in the logmv regression models

	Predictors	Linear Regression	Ridge Regression	Lasso Regression	Elastic Net Regression
0	Crim	-0.090	-0.082	-0.035	-0.048
1	Zn	0.034	0.022	0.000	0.000
2	Indus	0.031	0.018	-0.000	-0.000
3	Chas	0.031	0.033	0.000	0.018
4	Nox	-0.144	-0.118	-0.011	-0.030
5	Rooms	0.048	0.060	0.042	0.058
6	Age	0.005	-0.006	-0.000	-0.000
7	Dis	-0.119	-0.098	-0.000	-0.000
8	Rad	0.100	0.058	-0.000	-0.000
9	Tax	-0.061	-0.029	-0.000	-0.000
10	Pratio	-0.093	-0.085	-0.046	-0.059
11	Lstat	-0.201	-0.183	-0.195	-0.181

Figure 14: Results from regression models predicting logmv.

```
Linear Regression
Top paramaters: {'fit_intercept': True}
RMSE: 0.037726984403107115
RMSE for test set: 0.034086023085697
RMSE for validation set: 0.03821129661277439

Ridge Regression
Top paramaters: {'alpha': 10, 'solver': 'cholesky'}
RMSE: 0.03824561654081196
RMSE for test set: 0.03585347110268054
RMSE for validation set: 0.038124060616251176

Lasso Regression
Top paramaters: {'alpha': 0.05}
RMSE: 0.05075948961357432
RMSE for test set: 0.0491308594806133
RMSE for validation set: 0.05238191268481157

Elastic Net Regression
Top paramaters: {'alpha': 0.05, 'copy_X': True, 'fit_intercept': True, 'l1_ratio': 0.5, 'max_
iter': 1000, 'normalize': False, 'positive': False, 'precompute': False, 'random_state': 0,
'selection': 'cyclic', 'tol': 0.0001, 'warm_start': False}
RMSE: 0.045914019210583946
RMSE for test set: 0.0429687031234385
RMSE for validation set: 0.045706626917567404
```

In [755]:
```
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, Rid
ge, Lasso, ElasticNet
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import seaborn as sns
import scipy.stats as stats
from sklearn.model_selection import KFold, GridSearchCV, cross_validate,
cross_val_score, cross_val_predict
from sklearn.model_selection import train_test_split
```

In [756]:
```
# 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 [757]:
```
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

In [758]:
```
# read data for the Boston Housing Study
# creating data frame restdata
boston = pd.read_csv('boston.csv')
boston_in = pd.read_csv('boston.csv')
# drop neighborhood from the data being considered
boston = boston.drop('neighborhood', 1)
```

In [759]:
```
boston.head()
```

Out[759]:

	crim	zn	indus	chas	nox	rooms	age	dis	rad	tax	ptratio	lstat	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
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6
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	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2

In [760]:
```
boston.tail()
```

Out[760]:

	crim	zn	indus	chas	nox	rooms	age	dis	rad	tax	ptratio	lstat	mv
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	7.88	19.0

In [761]:
```
boston.info()
```

```
<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
```

In [762]:
```
boston.shape
```

Out[762]: (506, 13)

In [763]:
```
boston.describe()
```

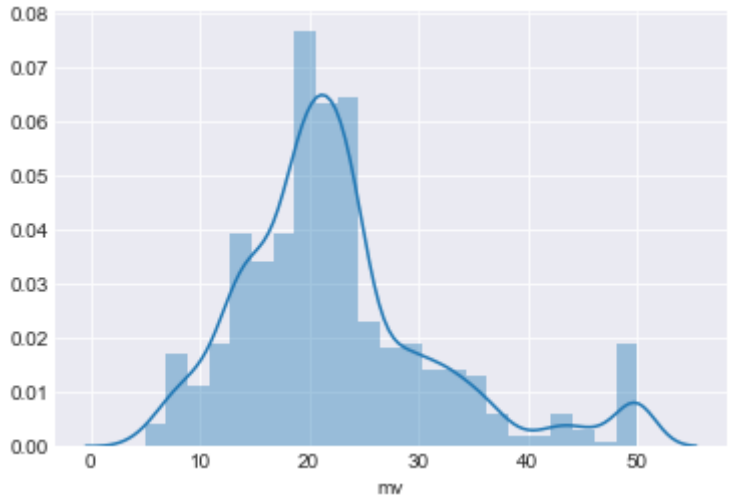
Out[763]:

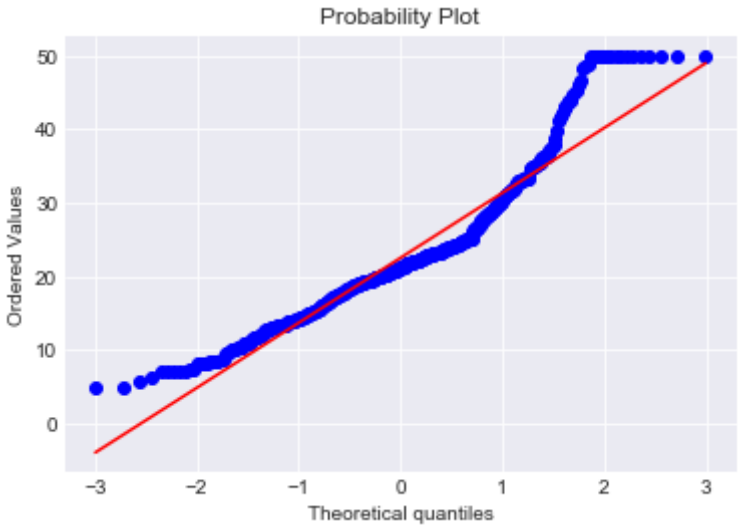
	crim	zn	indus	chas	nox	rooms	age	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.(
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.7
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.1
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.1
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.1
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.2
75%	3.677082	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.1
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.1

In [764]:
```
boston["mvlog"] = np.log10(boston["mv"].replace(0, np.nan))
#boston["mvlog"] = np.log10(boston["mv"])
```

In [765]:
```
sns.distplot(boston['mv'], kde=True,);
fig = plt.figure()
res = stats.probplot(boston['mv'], plot=plt)
print("Skewness: %f" % boston['mv'].skew())
print("Kurtosis: %f" % boston['mv'].kurt())
```

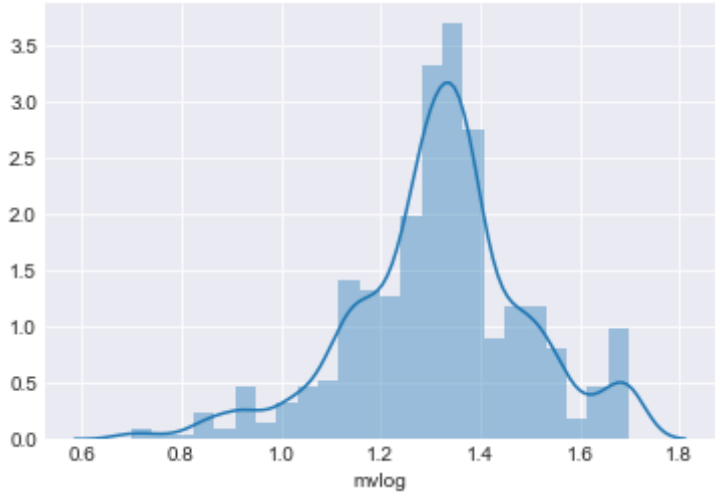
```
Skewness: 1.110912
Kurtosis: 1.516783
```

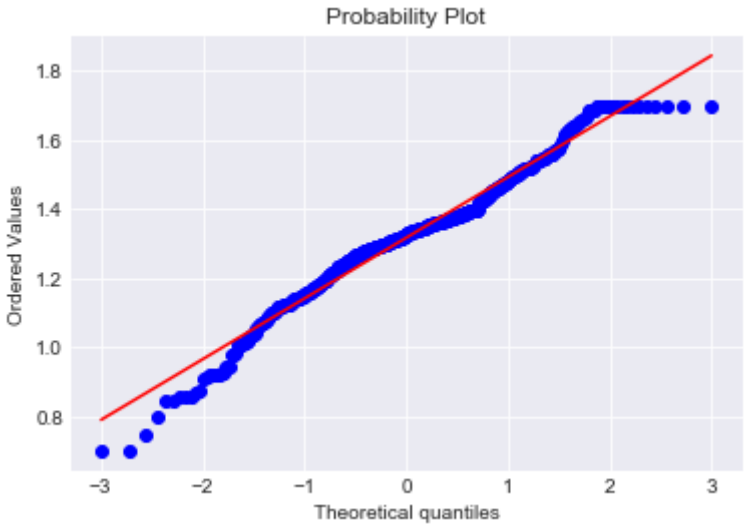




In [766]:
```
sns.distplot(boston['mvlog'], kde=True,);
fig = plt.figure()
res = stats.probplot(boston['mvlog'], plot=plt)
print("Skewness: %f" % boston['mvlog'].skew())
print("Kurtosis: %f" % boston['mvlog'].kurt())
```

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

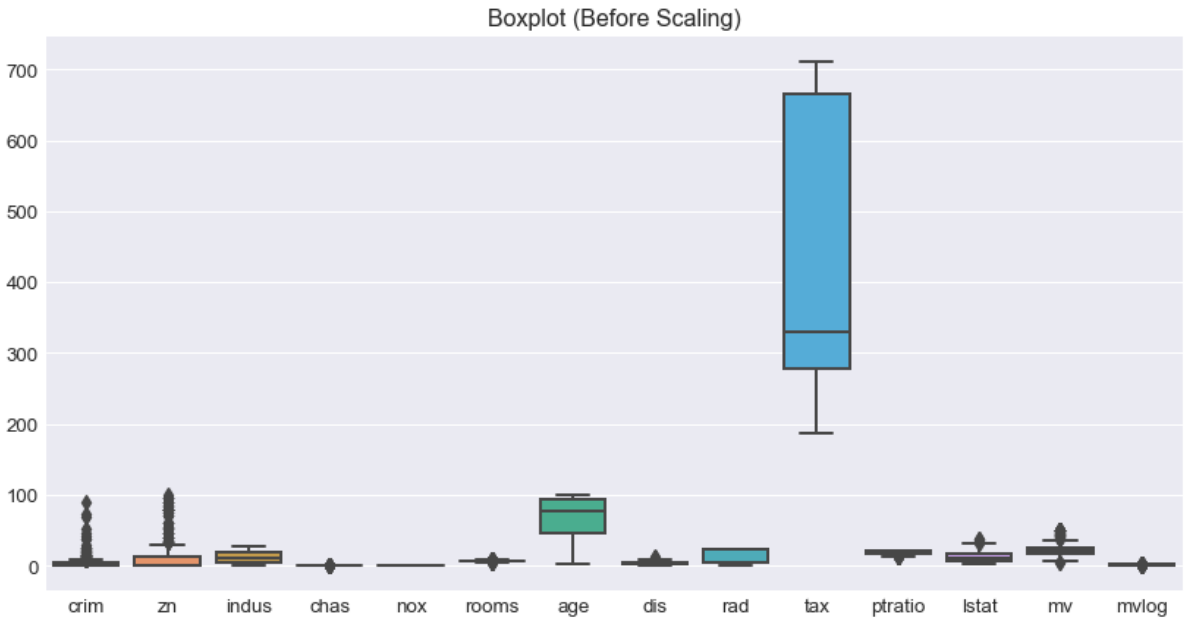




In [ ]:

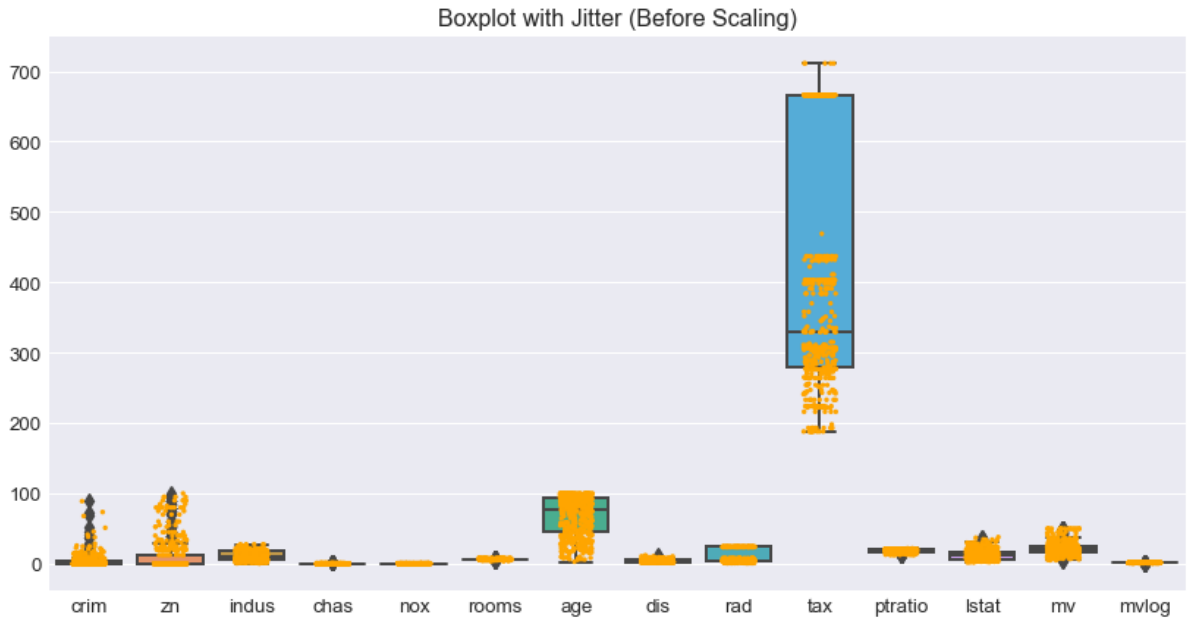
In [767]:
```
plt.style.use('seaborn-darkgrid')
my_dpi=96
plt.figure(figsize=(1000/my_dpi, 500/my_dpi), dpi=my_dpi)

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

Out[767]: Text(0.5, 1.0, 'Boxplot (Before Scaling)')



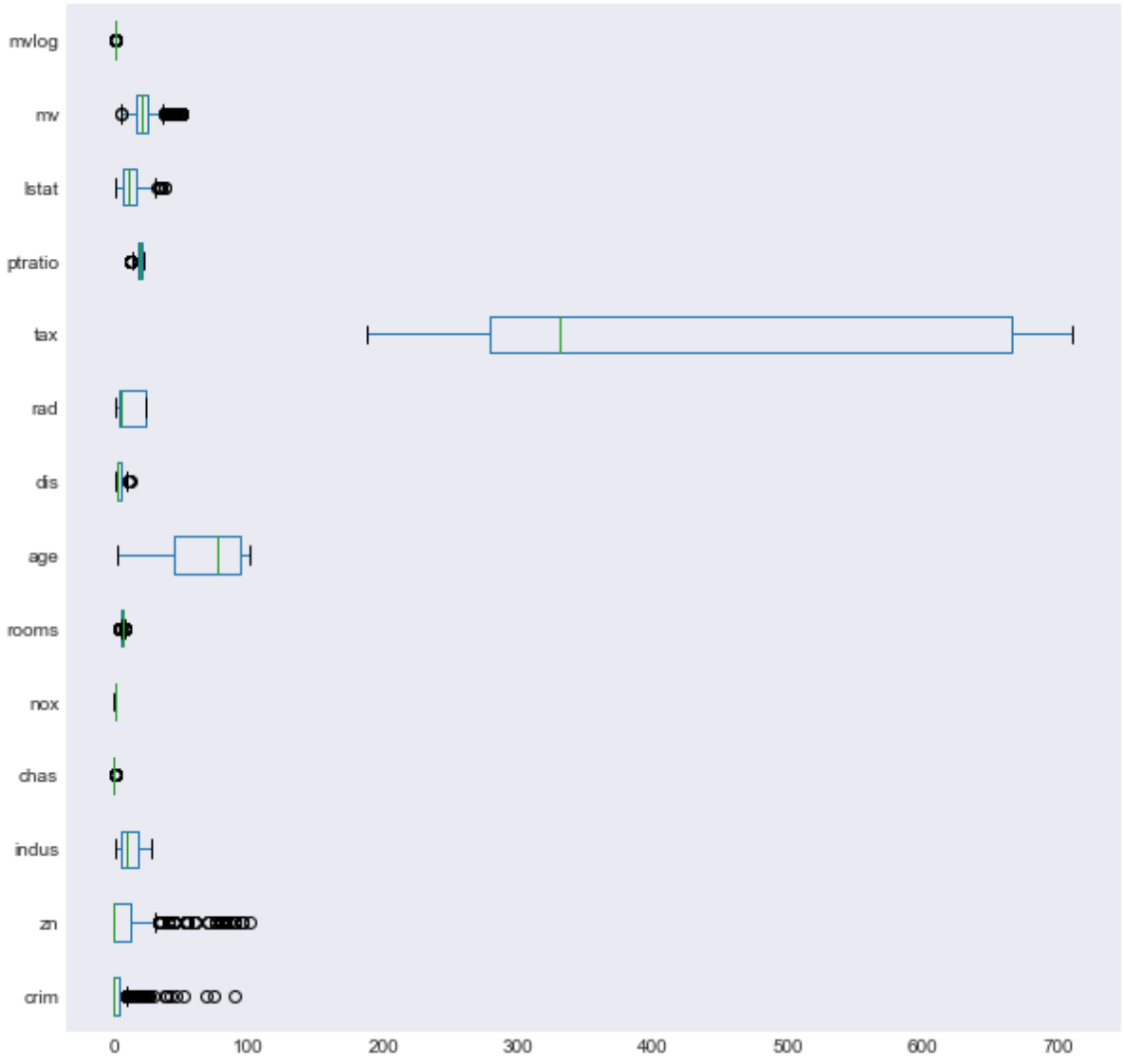
In [768]:
```
plt.style.use('seaborn-darkgrid')
my_dpi=96
plt.figure(figsize=(1000/my_dpi, 500/my_dpi), dpi=my_dpi)

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

Out[768]:
```
Text(0.5, 1.0, 'Boxplot with Jitter (Before Scaling)')
```



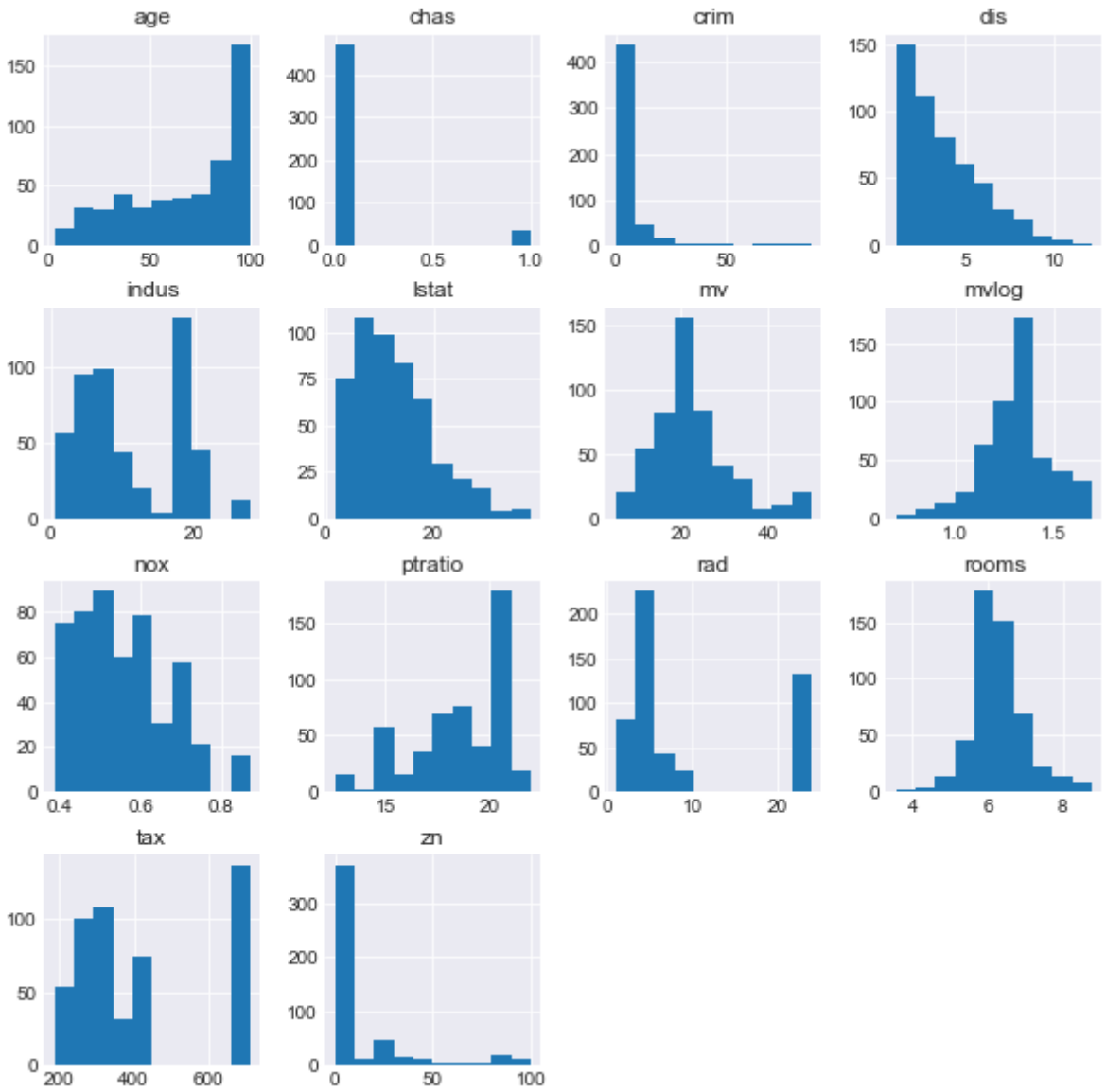
In [769]:
```
boston.boxplot(vert = False, figsize = (10,10), grid = False)
```

Out[769]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14b167850>
```



In [770]:
```
boston.hist(figsize=(10,10))
```

Out[770]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x150462810>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x151f0d750>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x151f3ff50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x151f80790>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x151fb4f90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x151ff67d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x152028fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x15206c810>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x152074390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1520a9d10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x15211fbd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x152157890>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x152192c10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1521cb8d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x15220ac50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x152240910
>]],
      dtype=object)
```



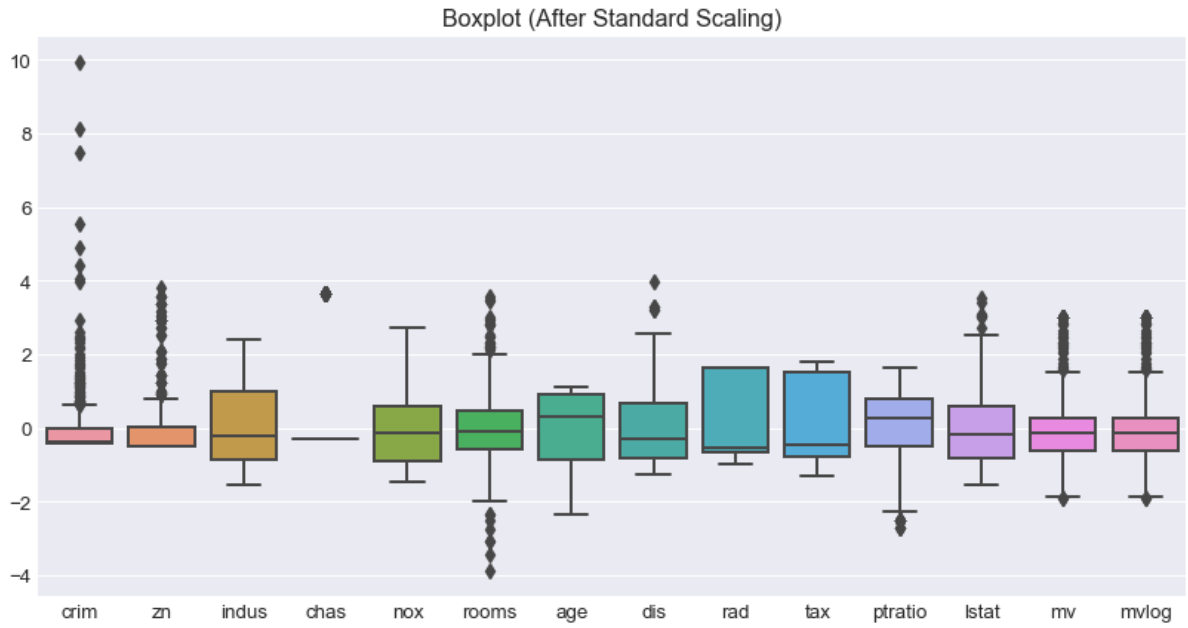
In [771]:
```
# standard scores for the columns... along axis 0
scaler = StandardScaler()
```

In [772]:
```
# the model data will be standardized form of preliminary model data
model_data= pd.DataFrame(boston)
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(
-1,1))
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(
-1,1))
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.resh
ape(-1,1))
model_data.lstat = scaler.fit_transform(model_data.lstat.values.reshape(
-1,1))
```

ax = sns.boxplot(x='group', y='value', data=df) ax = sns.stripplot(x='group', y='value', data=df, color="orange", jitter=0.2, size=2.5) plt.title("Boxplot with jitter", loc="left") PREVIOUS POST

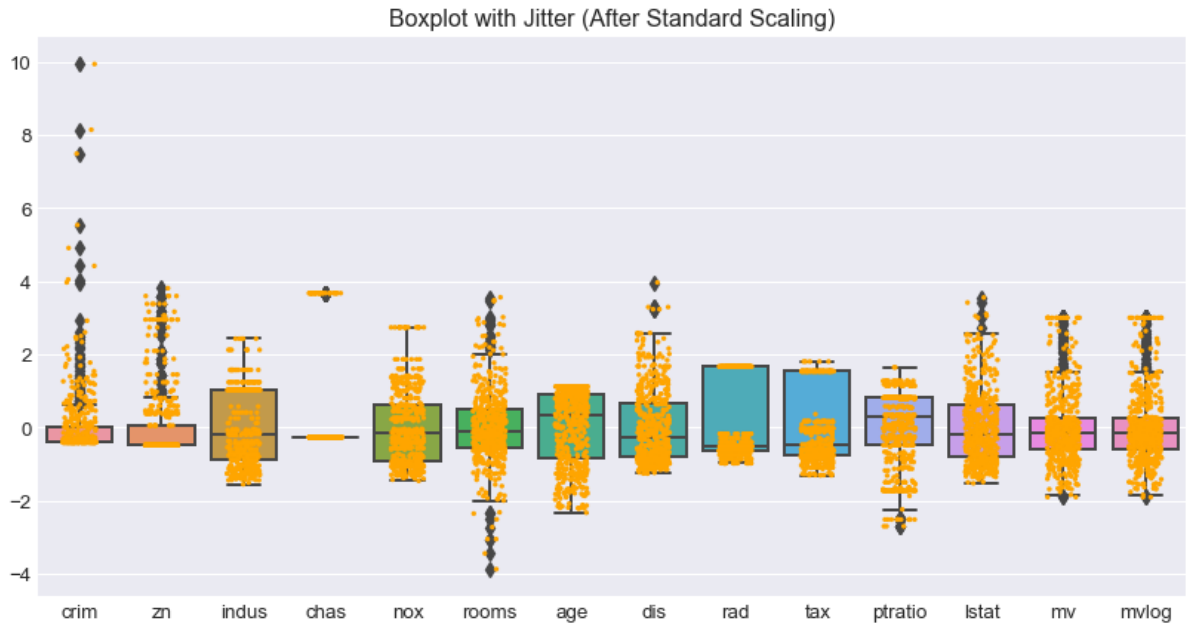
In [773]:
```
plt.style.use('seaborn-darkgrid')
my_dpi=96
plt.figure(figsize=(1000/my_dpi, 500/my_dpi), dpi=my_dpi)

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

Out[773]:
```
Text(0.5, 1.0, 'Boxplot (After Standard Scaling)')
```



In [774]:
```
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)
ax = sns.stripplot(data=boston, color="orange", jitter=0.2, size=2.5)
plt.title("Boxplot with Jitter (After Standard Scaling)")
```

Out[774]: Text(0.5, 1.0, 'Boxplot with Jitter (After Standard Scaling)')

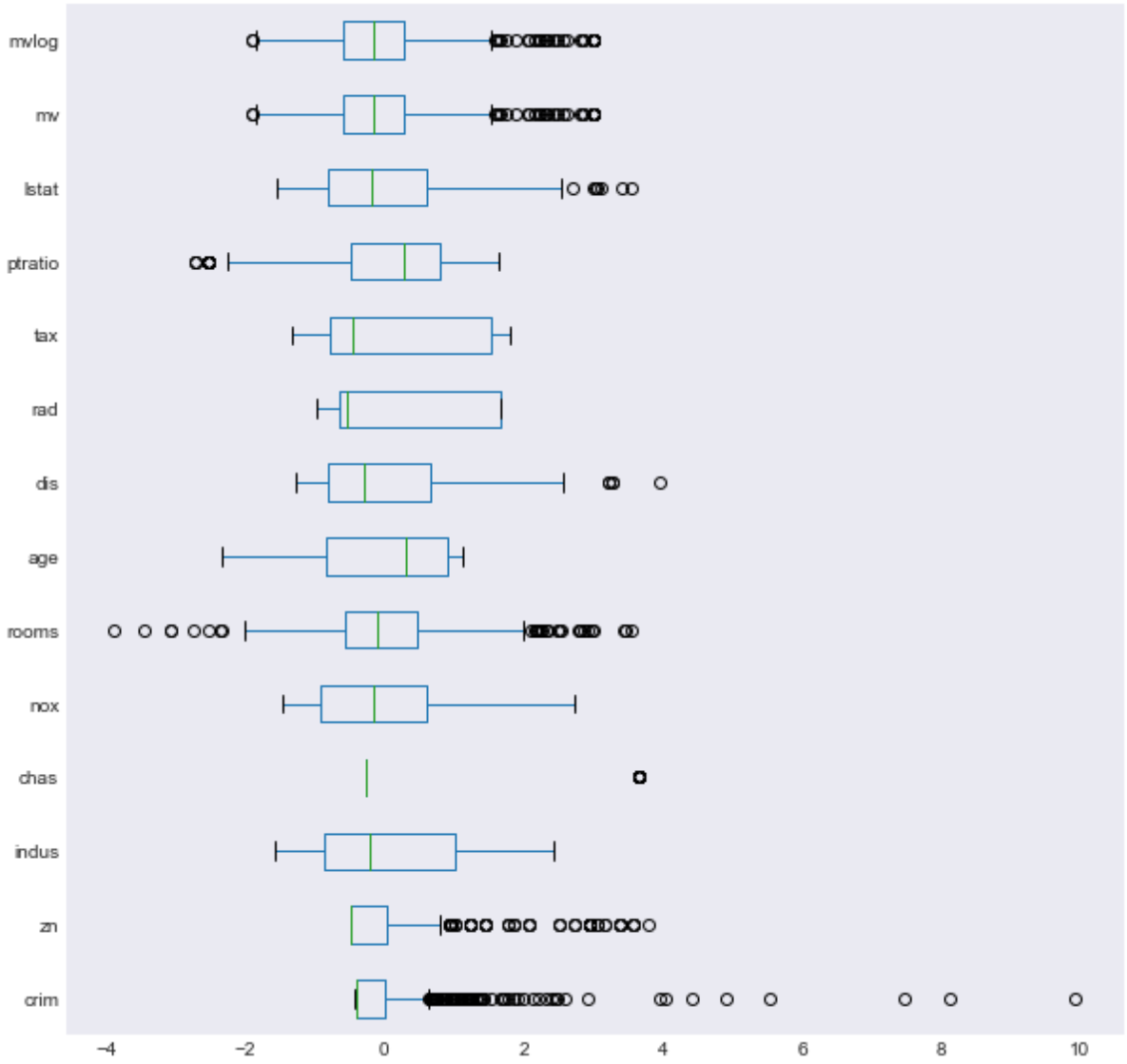


In [775]:
```
plt.style.use('seaborn-darkgrid')
my_dpi=96
plt.figure(figsize=(1000/my_dpi, 300/my_dpi), dpi=my_dpi)
sns.violinplot(data=model_data)
plt.title("Violin Plot")
```

Out[775]: Text(0.5, 1.0, 'Violin Plot')

In [776]:
```
model_data.boxplot(vert = False, figsize = (10,10), grid = False)
```

Out[776]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x152dee810>
```



In [777]:
```
# show standardization constants being employed
print(scaler.mean_)
```

```
[12.65306324]
```

In [778]:
```
print(scaler.scale_)
```

```
[7.13400164]
```

Scaled / Unscladed distribution plots

In [779]:
```
model_data.shape
```

Out[779]:
```
(506, 14)
```

In [780]:
```
model_data.describe()
```

Out[780]:

	crim	zn	indus	chas	nox	rooms	
count	5.060000e+02	5.060000e+02	5.060000e+02	5.060000e+02	5.060000e+02	5.060000e+02	
mean	-8.688702e-17	3.306534e-16	2.804081e-16	-3.100287e-16	-8.071058e-16	-5.978968e-17	-
std	1.000990e+00	1.000990e+00	1.000990e+00	1.000990e+00	1.000990e+00	1.000990e+00	
min	-4.197819e-01	-4.877224e-01	-1.557842e+00	-2.725986e-01	-1.465882e+00	-3.880249e+00	-3
25%	-4.109696e-01	-4.877224e-01	-8.676906e-01	-2.725986e-01	-9.130288e-01	-5.686303e-01	-
50%	-3.906665e-01	-4.877224e-01	-2.110985e-01	-2.725986e-01	-1.442174e-01	-1.084655e-01	
75%	7.396560e-03	4.877224e-02	1.015999e+00	-2.725986e-01	5.986790e-01	4.827678e-01	
max	9.933931e+00	3.804234e+00	2.422565e+00	3.668398e+00	2.732346e+00	3.555044e+00	

In [781]:
```
# 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, 14)
```

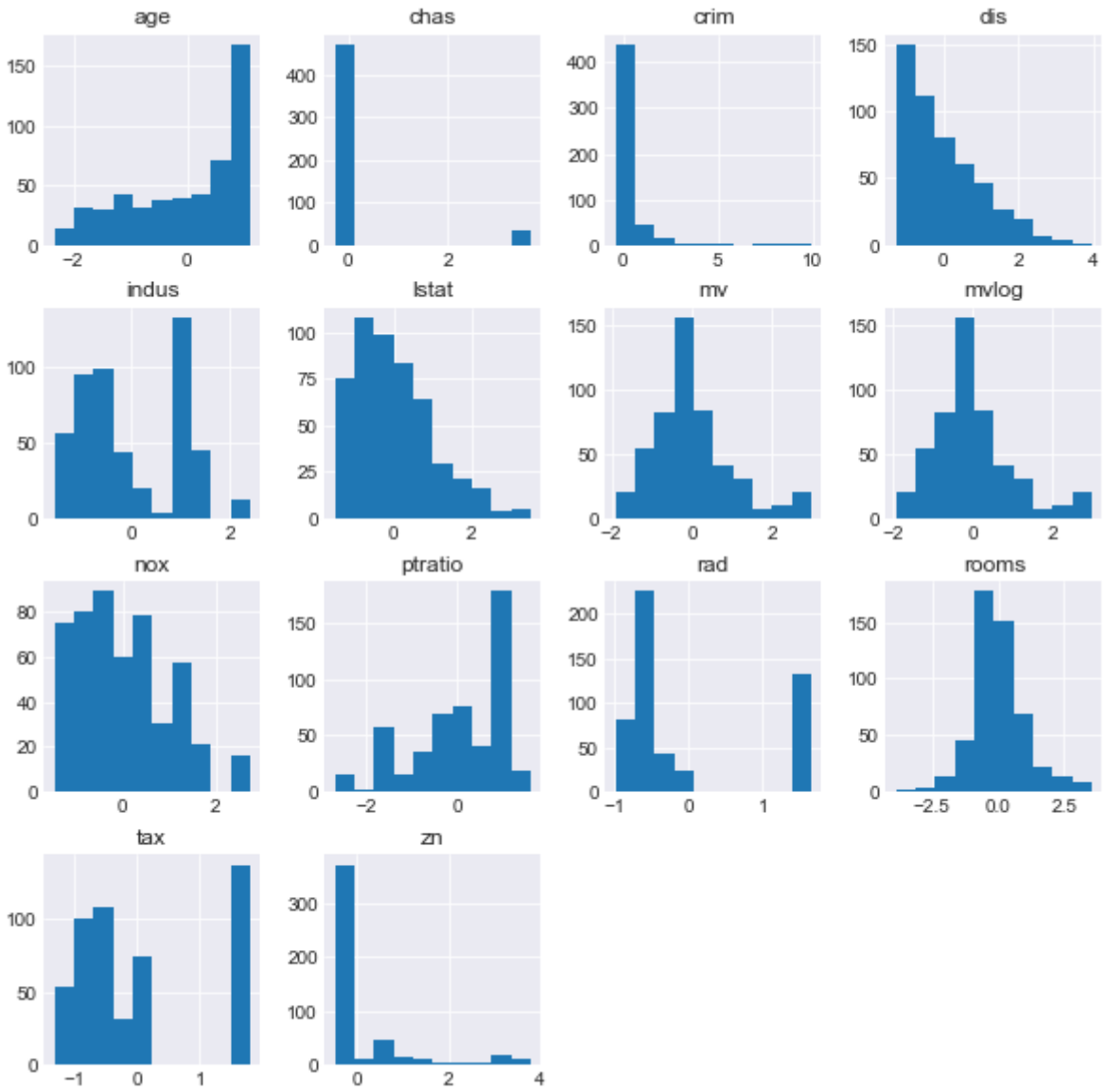
In [782]:
```
cor = model_data.corr() # Whole correlation matrix
cor
```

Out[782]:

	crim	zn	indus	chas	nox	rooms	age	dis	
crim	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.6
zn	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.3
indus	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.5
chas	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.0
nox	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.6
rooms	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.2
age	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.4
dis	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.4
rad	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.0
tax	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.9
ptratio	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.4
lstat	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.4
mv	-0.389582	0.360386	-0.484754	0.175663	-0.429300	0.696304	-0.377999	0.249315	-0.3
mvlog	-0.389582	0.360386	-0.484754	0.175663	-0.429300	0.696304	-0.377999	0.249315	-0.3

In [783]:
```
model_data.hist(figsize=(10,10))
```

Out[783]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1531df310>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x153206e90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x153397650>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1533cce50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x15340b690>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x153441e90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1534816d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1534b6ed0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1534c0a50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x153501410>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x15356d750>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1535a0f50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1535e3790>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x153616f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x153658750>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x15368bf50
>]],
      dtype=object)
```



In [784]:
```
model_data["mvlog"] = np.log10(model_data["mv"])
```

```
/Users/allisonroeser/opt/anaconda3/lib/python3.7/site-packages/pandas/c
ore/series.py:853: RuntimeWarning: invalid value encountered in log10
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [785]:
```
model_data.head()
```

Out[785]:

	crim	zn	indus	chas	nox	rooms	age	dis	rad
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.982843
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.867883
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.867883
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.752922
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.752922

In [786]:
```
#Correlation with output variable
cor_target = abs(cor["mv"])
#Selecting highly correlated features
relevant = cor_target[cor_target>0.35]
relevant
```

Out[786]:
```
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
mvlog      1.000000
Name: mv, dtype: float64
```

In [787]:
```
#Correlation with output variable
cor_target = abs(cor["mvlog"])
#Selecting highly correlated features
relevant = cor_target[cor_target>0.10]
relevant
```

Out[787]:
```
crim       0.389582
zn         0.360386
indus      0.484754
chas       0.175663
nox        0.429300
rooms      0.696304
age        0.377999
dis        0.249315
rad        0.384766
tax        0.471979
ptratio    0.505655
lstat      0.740836
mv         1.000000
mvlog      1.000000
Name: mvlog, dtype: float64
```

In [788]:
```
relevant = pd.DataFrame(data = model_data, columns = [
    "lstat", "rooms", "ptratio", "indus",
    "tax", "nox", "crim", "rad", "age", "zn", "mvlog"])
```

In [789]:
```
plt.figure(figsize=(15,15))
cor = relevant.corr()
cor
```

Out[789]:

	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	0.6
rooms	-0.613808	1.000000	-0.355501	-0.391676	-0.292048	-0.302188	-0.219247	-0.209847	-0.2
ptratio	0.374044	-0.355501	1.000000	0.383248	0.460853	0.188933	0.289946	0.464741	0.2
indus	0.603800	-0.391676	0.383248	1.000000	0.720760	0.763651	0.406583	0.595129	0.6
tax	0.543993	-0.292048	0.460853	0.720760	1.000000	0.668023	0.582764	0.910228	0.5
nox	0.590879	-0.302188	0.188933	0.763651	0.668023	1.000000	0.420972	0.611441	0.7
crim	0.455621	-0.219247	0.289946	0.406583	0.582764	0.420972	1.000000	0.625505	0.3
rad	0.488676	-0.209847	0.464741	0.595129	0.910228	0.611441	0.625505	1.000000	0.4
age	0.602339	-0.240265	0.261515	0.644779	0.506456	0.731470	0.352734	0.456022	1.0
zn	-0.412995	0.311991	-0.391679	-0.533828	-0.314563	-0.516604	-0.200469	-0.311948	-0.5
mvlog	-0.494807	0.646773	-0.304099	-0.231947	-0.162026	-0.101810	-0.025131	-0.055886	0.0

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

In [790]:
```
print(relevant[["rad","tax"]].corr())
```

```
          rad       tax
rad  1.000000  0.910228
tax  0.910228  1.000000
```

In [791]:
```
print(boston[["nox","indus"]].corr())
```

```
            nox     indus
nox    1.000000  0.763651
indus  0.763651  1.000000
```

In [792]:
```
print(boston[["nox","age"]].corr())
```

```
         nox      age
nox  1.00000  0.73147
age  0.73147  1.00000
```

In [793]:
```
print(relevant[["tax","indus"]].corr())
```

```
           tax    indus
tax    1.00000  0.72076
indus  0.72076  1.00000
```

In [794]:
```
print(relevant[["rad","indus"]].corr())
```

```
            rad     indus
rad    1.000000  0.595129
indus  0.595129  1.000000
```

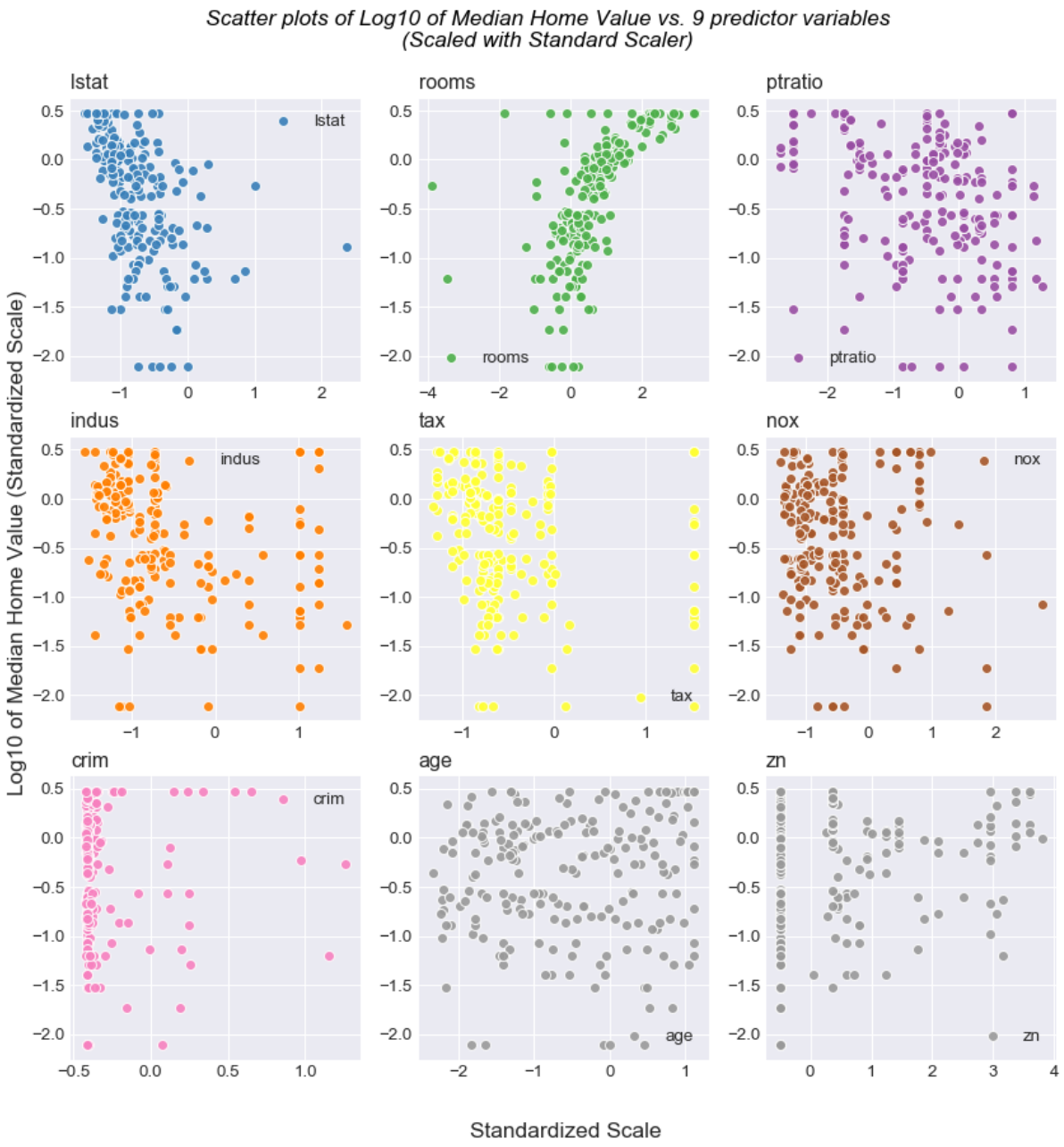
In [795]:
```
relevant = relevant.drop("rad", axis =1)
cor = relevant.corr() # Whole correlation matrix
cor
```

Out[795]:

	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	-0.4
rooms	-0.613808	1.000000	-0.355501	-0.391676	-0.292048	-0.302188	-0.219247	-0.240265	0.3
ptratio	0.374044	-0.355501	1.000000	0.383248	0.460853	0.188933	0.289946	0.261515	-0.3
indus	0.603800	-0.391676	0.383248	1.000000	0.720760	0.763651	0.406583	0.644779	-0.5
tax	0.543993	-0.292048	0.460853	0.720760	1.000000	0.668023	0.582764	0.506456	-0.3
nox	0.590879	-0.302188	0.188933	0.763651	0.668023	1.000000	0.420972	0.731470	-0.5
crim	0.455621	-0.219247	0.289946	0.406583	0.582764	0.420972	1.000000	0.352734	-0.2
age	0.602339	-0.240265	0.261515	0.644779	0.506456	0.731470	0.352734	1.000000	-0.5
zn	-0.412995	0.311991	-0.391679	-0.533828	-0.314563	-0.516604	-0.200469	-0.569537	1.0
mvlog	-0.494807	0.646773	-0.304099	-0.231947	-0.162026	-0.101810	-0.025131	0.058756	0.2

In [796]:
```
# 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('mvlog', 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['mvlog'], color=palette(num),  alpha=0.9, label=column)
    #plt.plot(relevant['mv'], relevant[column], marker='', color=palette(num), linewidth=1.9, alpha=0.9, label=column)


    # 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 variables\n(Scaled with Standard Scaler)", fontsize=13, fontweight=0, color='black', style='italic', y=.95)

# Axis title
plt.text(-4, -3, 'Standardized Scale', ha='center', va='center',fontsize = 13)
plt.text(-13, 3, 'Log10 of Median Home Value (Standardized Scale)', ha='center', va='center', rotation='vertical', fontsize = 13)
```

Out[796]: Text(-13, 3, 'Log10 of Median Home Value (Standardized Scale)')



In [797]:
```
model_data.shape
```

Out[797]: (506, 14)

In [798]:
```
relevant.shape
```

Out[798]: (506, 10)

In [799]:
```
predictors = relevant.drop('mvlog', axis=1)
predictors.head()
```

Out[799]:

	lstat	rooms	ptratio	indus	tax	nox	crim	age	zı
0	-1.075562	0.413672	-1.459000	-1.287909	-0.666608	-0.144217	-0.419782	-0.120013	0.28483(
1	-0.492439	0.194274	-0.303094	-0.593381	-0.987329	-0.740262	-0.417339	0.367166	-0.48772
2	-1.208727	1.282714	-0.303094	-0.593381	-0.987329	-0.740262	-0.417342	-0.265812	-0.48772
3	-1.361517	1.016303	0.113032	-1.306878	-1.106115	-0.835284	-0.416750	-0.809889	-0.48772
4	-1.026501	1.228577	0.113032	-1.306878	-1.106115	-0.835284	-0.412482	-0.511180	-0.48772

In [800]:
```
mv = model_data["mv"]
```

In [801]:
```
model_data.head()
```

Out[801]:

	crim	zn	indus	chas	nox	rooms	age	dis	rad
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.982843
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.867883
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.867883
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.752922
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.752922

In [802]:
```
boston_in = boston_in.drop("neighborhood", axis = 1)
boston_in['log_mv']=np.log(boston_in['mv'])
boston_in.head()
```

Out[802]:

	crim	zn	indus	chas	nox	rooms	age	dis	rad	tax	ptratio	lstat	mv	log_m
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0	3.17805
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6	3.07269
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7	3.54674
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4	3.50855
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2	3.58905

In [803]:
```
boston_train, boston_test = train_test_split(boston_in, test_size = 0.3,
random_state =1)
```

In [804]:
```
boston_train.head()
```

Out[804]:

	crim	zn	indus	chas	nox	rooms	age	dis	rad	tax	ptratio	lstat	mv	lo
13	0.62976	0.0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21.0	8.26	20.4	3.01
61	0.17171	25.0	5.13	0	0.453	5.966	93.4	6.8185	8	284	19.7	14.44	16.0	2.77
377	9.82349	0.0	18.10	0	0.671	6.794	98.8	1.3580	24	666	20.2	21.24	13.3	2.58
39	0.02763	75.0	2.95	0	0.428	6.595	21.8	5.4011	3	252	18.3	4.32	30.8	3.42
365	4.55587	0.0	18.10	0	0.718	3.561	87.9	1.6132	24	666	20.2	7.12	27.5	3.31

Predicting log of median value

In [805]:
```
boston_in.head()
```

Out[805]:

	crim	zn	indus	chas	nox	rooms	age	dis	rad	tax	ptratio	lstat	mv	log_m
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0	3.17805
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6	3.07269
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7	3.54674
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4	3.50855
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2	3.58905

In [806]:
```
# 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 [807]:
```
trn_data_scl_df.head()
```

Out[807]:

	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 [808]:
```
# Used Chris's code as a starting point
cv_train_data, cv_test_data = train_test_split(trn_data, test_size=0.3,
random_state=1)
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' ,
'Tax' ,'Ptr' ,'Lst' ]
y_lbl=['Mhv']
```

In [809]:
```
X_test_cv.shape
```

Out[809]:
```
(107, 12)
```

In [810]:
```
# Used Chris's code as a starting point
#Compare Classifer performance with and wo regularzation
models = ['Linear Regression', 'Ridge Regression', 'Lasso Regression',
'Elastic Net Regression']
clfs = [LinearRegression(), Ridge(), Lasso(), ElasticNet()]
params ={models[0]: {'fit_intercept':[True,False]},
            models[1]: {'alpha':[ 0.01, 0.1, 1, 10, 25, 75, 100], 'solve
r':['cholesky']},
            models[2]: {'alpha':[ 0.05, 0.1,1,10, 100]},
            models[3]: {'alpha':[ 0.05, 0.1,1,10, 100],'copy_X':[True],
'fit_intercept':[True], 'l1_ratio':[0.5],
                        'max_iter':[1000], 'normalize':[False], 'positive'
:[False], 'precompute':[False],'random_state':[0],
                        'selection':['cyclic'], 'tol':[0.0001], 'warm_star
t':[False]}}

test_scores = []
```

In [811]:
```
# Used Chris's code as a starting point
for name, estimator in zip(models,clfs):
    print(name)
    clf = GridSearchCV(estimator, params[name], scoring='neg_mean_square
d_error', return_train_score=True, cv=7)
    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.predi
ct(X_valid),  clf.best_estimator_.coef_ ,
                        clf.best_estimator_.intercept_, clf.best_estimator
_, clf.best_params_, rmse_tst, rmse_vld))
results = pd.DataFrame()
results = results.append(pd.DataFrame(test_scores, columns=['nm','rmse',
'bscr','y_vld', 'x_prd','coef',
                                                            'intr','best'
,'bparm', 'rmse_tst', 'rmse_vld']),ignore_index=True)
```

```
Linear Regression
Top paramaters: {'fit_intercept': True}
RMSE: 0.037726984403107115
RMSE for test set: 0.034086023085697
RMSE for validation set: 0.03821129661277439

Ridge Regression
Top paramaters: {'alpha': 10, 'solver': 'cholesky'}
RMSE: 0.03824561654081196
RMSE for test set: 0.03585347110268054
RMSE for validation set: 0.038124060616251176

Lasso Regression
Top paramaters: {'alpha': 0.05}
RMSE: 0.050759489613574 32
RMSE for test set: 0.0491308594806133
RMSE for validation set: 0.05238191268481157

Elastic Net Regression
Top paramaters: {'alpha': 0.05, 'copy_X': True, 'fit_intercept': True,
'l1_ratio': 0.5, 'max_iter': 1000, 'normalize': False, 'positive': Fals
e, 'precompute': False, 'random_state': 0, 'selection': 'cyclic', 'to
l': 0.0001, 'warm_start': False}
RMSE: 0.045914019210583946
RMSE for test set: 0.0429687031234385
RMSE for validation set: 0.045706626917567404


/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)
```

In [812]:

```
mdl_pred=results.iloc[0::1,4]
mdl_coef=results.iloc[0::1,5]
mdl_intercept=results.iloc[0::1,6]
feat = ['Crim' ,'Zn' ,'Indus' ,'Chas' ,'Nox' ,'Rooms' ,'Age' ,'Dis' ,'Ra
d' ,'Tax' ,'Pratio' ,'Lstat' ]
```

In [813]:
```
mdl_coef
```

Out[813]:
```
0    [-0.08953158083877281, 0.03403613966706227, 0....
1    [-0.08175060651143756, 0.022147752832909623, 0...
2    [-0.035145489647394834, 0.0, -0.0, 0.0, -0.011...
3    [-0.0476723344021444, 0.0, -0.0, 0.01826152474...
Name: coef, dtype: object
```

In [814]:
```
mdl_intercept
```

Out[814]:
```
0    3.031382
1    3.031382
2    3.031382
3    3.031382
Name: intr, dtype: float64
```

In [815]:
```
s1 = pd.Series(mdl_coef[0], name='coeficent')
s2 = pd.Series(feat, name='feature')
s3 = pd.concat([s1, s2], axis=1)
```

In [816]:
```
s1
```

Out[816]:
```
0    -0.089532
1     0.034036
2     0.031300
3     0.031206
4    -0.143659
5     0.047806
6     0.004667
7    -0.119461
8     0.100192
9    -0.061171
10   -0.092695
11   -0.201396
Name: coeficent, dtype: float64
```

In [817]:
```
feat
```

Out[817]:
```
['Crim',
 'Zn',
 'Indus',
 'Chas',
 'Nox',
 'Rooms',
 'Age',
 'Dis',
 'Rad',
 'Tax',
 'Pratio',
 'Lstat']
```

In [ ]:

In [818]:
```
Coef = pd.DataFrame()
Coef["Predictors"] = feat
Coef["Linear Regression"] = round(pd.Series(mdl_coef[0]),3)
Coef["Ridge Regression"] = round(pd.Series(mdl_coef[1]),3)
Coef["Lasso Regression"] = round(pd.Series(mdl_coef[2]),3)
Coef["Elastic Net Regression"] = round(pd.Series(mdl_coef[3]),3)
Coef
```

Out[818]:

	Predictors	Linear Regression	Ridge Regression	Lasso Regression	Elastic Net Regression
0	Crim	-0.090	-0.082	-0.035	-0.048
1	Zn	0.034	0.022	0.000	0.000
2	Indus	0.031	0.018	-0.000	-0.000
3	Chas	0.031	0.033	0.000	0.018
4	Nox	-0.144	-0.118	-0.011	-0.030
5	Rooms	0.048	0.060	0.042	0.058
6	Age	0.005	-0.006	-0.000	-0.000
7	Dis	-0.119	-0.098	-0.000	-0.000
8	Rad	0.100	0.058	-0.000	-0.000
9	Tax	-0.061	-0.029	-0.000	-0.000
10	Pratio	-0.093	-0.085	-0.046	-0.059
11	Lstat	-0.201	-0.183	-0.195	-0.181

In [819]:
```
ResultslogMV = pd.DataFrame()
ResultslogMV["Model"] = models
ResultslogMV["RMSE"] = results.iloc[0::1,1]
ResultslogMV
```

Out[819]:

	Model	RMSE
0	Linear Regression	0.037727
1	Ridge Regression	0.038246
2	Lasso Regression	0.050759
3	Elastic Net Regression	0.045914

In [820]:
```
ResultslogMV = pd.DataFrame()
ResultslogMV["Model"] = models
ResultslogMV["RMSE logMV"] = results.iloc[0::1,1]
ResultslogMV["RMSE logMV transformed"] = np.exp(ResultslogMV["RMSE logM
V"])
ResultslogMV
```

Out[820]:

	Model	RMSE logMV	RMSE logMV transformed
0	Linear Regression	0.037727	1.038448
1	Ridge Regression	0.038246	1.038986
2	Lasso Regression	0.050759	1.052070
3	Elastic Net Regression	0.045914	1.046984

In [825]:
```
mdl_pred[0].max()
```

Out[825]:
```
3.8096277904754707
```

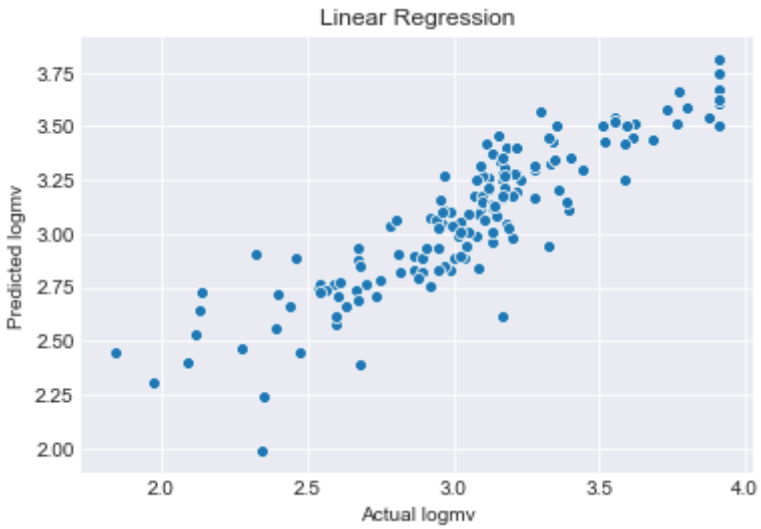
In [826]:
```
y_valid.max()
```

Out[826]:
```
3.912023005428146
```

Predicted vs. Actual graph

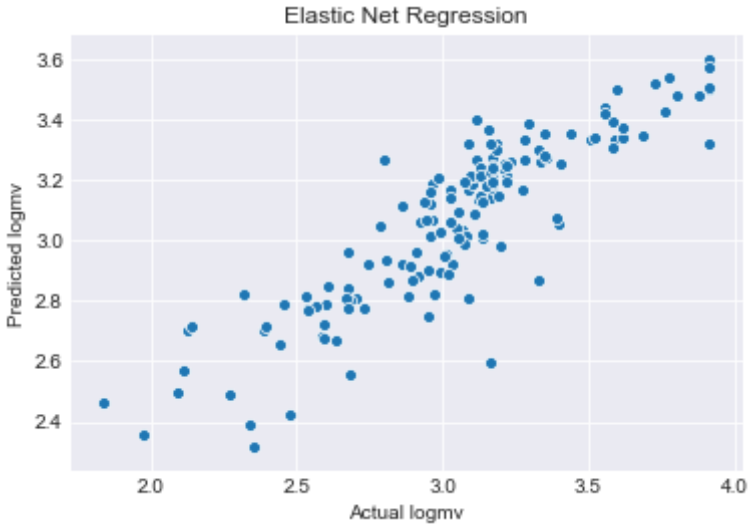
In [834]:
```
ax = sns.scatterplot(x=y_valid, y=mdl_pred[0])
ax.set_xlabel("Actual logmv")
ax.set_ylabel("Predicted logmv")
ax.set_title("Linear Regression")
```

Out[834]:
```
Text(0.5, 1.0, 'Linear Regression')
```



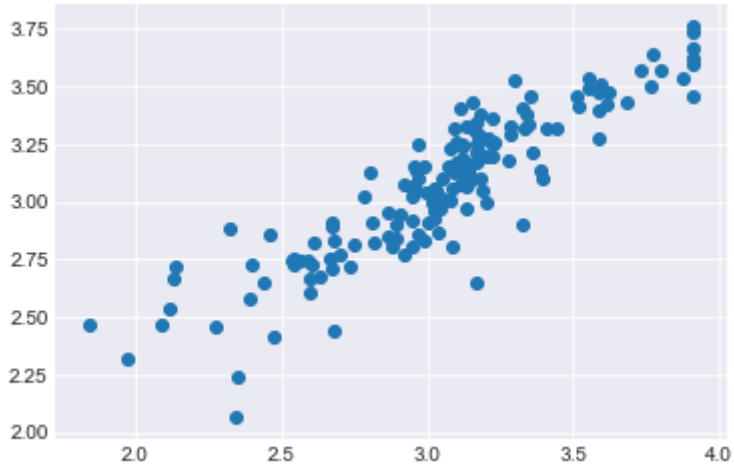
In [836]:
```
ax = sns.scatterplot(x=y_valid, y=mdl_pred[3])
ax.set_xlabel("Actual logmv")
ax.set_ylabel("Predicted logmv")
ax.set_title("Elastic Net Regression")
```

Out[836]: Text(0.5, 1.0, 'Elastic Net Regression')



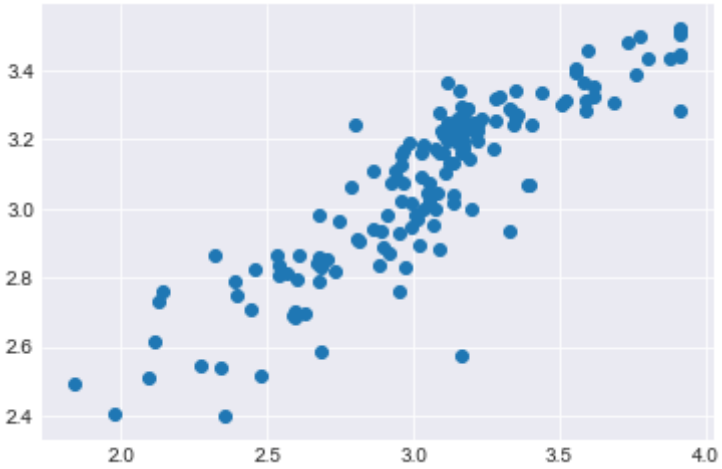
In [822]:
```
plt.scatter(y_valid, mdl_pred[1])
```

Out[822]: <matplotlib.collections.PathCollection at 0x1543278d0>



In [823]:
```
plt.scatter(y_valid, mdl_pred[2])
```

Out[823]: <matplotlib.collections.PathCollection at 0x15484efd0>



In [824]:
```
plt.scatter(y_valid, mdl_pred[3])
```

Out[824]: <matplotlib.collections.PathCollection at 0x15484e3d0>