In [1]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso, Elast
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Download the housing dataset referenced in the textbook on page 279

Attribute Information

1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centres
9. RAD: index of accessibility to radial highways
10. TAX: full-value property-tax rate per $10,000
11. PTRATIO: pupil-teacher ratio by town
12. B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT: % lower status of the population
14. MEDV: Median value of owner-occupied homes in $1000's

In [2]:
```
# Load data into a dataframe named "df_raw"
df_raw = pd.read_csv('housing.data', header=None, delim_whitespace=Tru

headers = ["CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "
df_raw.columns = headers
df_raw
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	

506 rows × 14 columns

In [3]:
```
# Use StandardScaler to scale the data. Save the result as "df"
scaler = preprocessing.StandardScaler().fit(df_raw)
scaled_data = scaler.transform(df_raw)
df = pd.DataFrame(data=scaled_data, columns=df_raw.columns)
df.sample(n=10)
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
504	-0.407764	-0.487722	0.115738	-0.272599	0.158124	0.725672	0.736996	-0.668437	-0.9
453	0.539339	-0.487722	1.015999	-0.272599	1.367490	1.579043	1.092602	-0.638108	1.(
458	0.481635	-0.487722	1.015999	-0.272599	1.367490	0.023315	0.537857	-0.481046	1.(
159	-0.254683	-0.487722	1.231945	-0.272599	2.732346	0.321069	1.117494	-0.964592	-0.5
324	-0.380824	-0.487722	-0.548149	-0.272599	-0.532942	0.185727	-1.012584	0.440219	-0.5
180	-0.412851	-0.487722	-1.266023	-0.272599	-0.576134	2.109016	0.523633	-0.501059	-0.1
251	-0.395603	0.456508	-0.769931	-0.272599	-1.068519	0.218494	-2.122074	1.712117	-0.2
62	-0.407685	0.585267	-0.876445	-0.272599	-0.878475	0.244138	-0.027556	1.630734	-0.1
176	-0.412346	-0.487722	-1.034027	-0.272599	-0.386091	-0.377014	-0.760104	-0.114156	-0.5
28	-0.330562	-0.487722	-0.437258	-0.272599	-0.144217	0.299699	0.918355	0.313581	-0.(

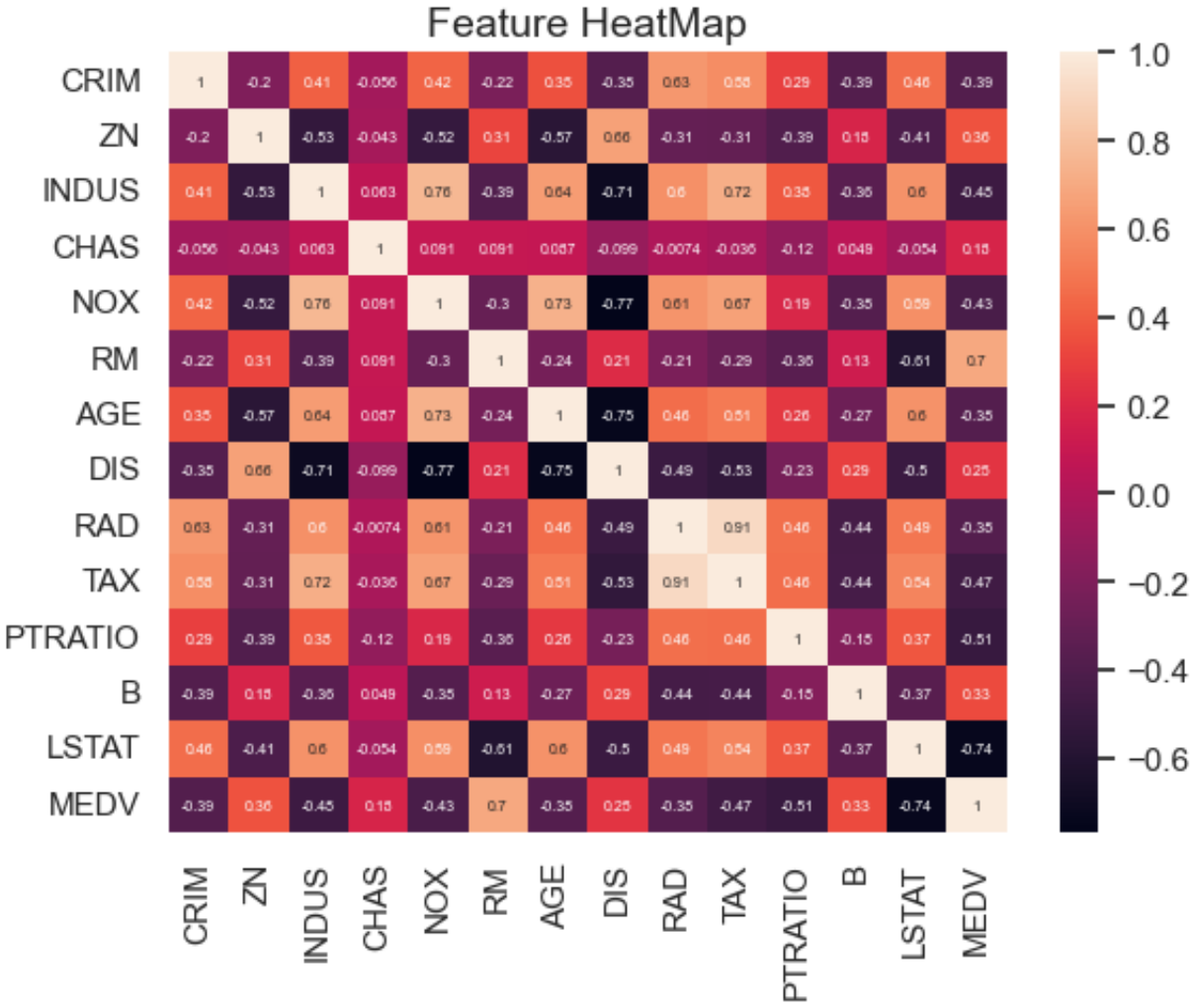
2. Pick a number of features that you think may be correlated and plot pairs of them to confirm that they are correlated.

In [4]:
```
# Print the correlation matrix for the dataset
df_corr = df.corr()
df_corr
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996
MEDV	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929

In [5]:
```
# Use Seaborn to print the heatmap of the correlation matrix
sns.set()
corr = df.corr()
ax = sns.heatmap(corr, annot=True, annot_kws={'size':5})
ax.set_title('Feature HeatMap', fontsize=14)
plt.show()
```



In [6]:
```
# Generate flattened dataframe, so we can find the column pairs with t
# abs=absolute value of correlation, raw=raw correlation value
corr_flat = pd.DataFrame(columns=['abs', 'raw', 'col1', 'col2'])
features = df.columns
for col1 in range(0, len(features)):
    for col2 in range(col1+1, len(features)):
        corrVal = df_corr.iloc[col1, col2]
        #print(col1, features[col1], col2, features[col2], corrVal)
        corr_flat.loc[len(corr_flat.index)] = [abs(corrVal), corrVal,
```

In [7]:
```
# Sort by absolute value of correlation
corr_flat = corr_flat.sort_values(by=['abs'], ascending=False)
```

In [8]:
```
# Print the 5 pairs with the highest absolute value correlation
corr_flat.iloc[0:5]
```

Out[8]:

	abs	raw	col1	col2
76	0.910228	0.910228	RAD	TAX
48	0.769230	-0.769230	NOX	DIS
26	0.763651	0.763651	INDUS	NOX
63	0.747881	-0.747881	AGE	DIS
90	0.737663	-0.737663	LSTAT	MEDV

In [9]:
```
# Pull out individual feature columns so we can plot them
CRIM = df.loc[:, "CRIM"]
ZN = df.loc[:, "ZN"]
INDUS = df.loc[:, "INDUS"]
CHAS = df.loc[:, "CHAS"]
NOX = df.loc[:, "NOX"]
RM = df.loc[:, "RM"]
AGE = df.loc[:, "AGE"]
DIS = df.loc[:, "DIS"]
RAD = df.loc[:, "RAD"]
TAX = df.loc[:, "TAX"]
PTRATIO = df.loc[:, "PTRATIO"]
B = df.loc[:, "B"]
LSTAT = df.loc[:, "LSTAT"]
MEDV = df.loc[:, "MEDV"]
```

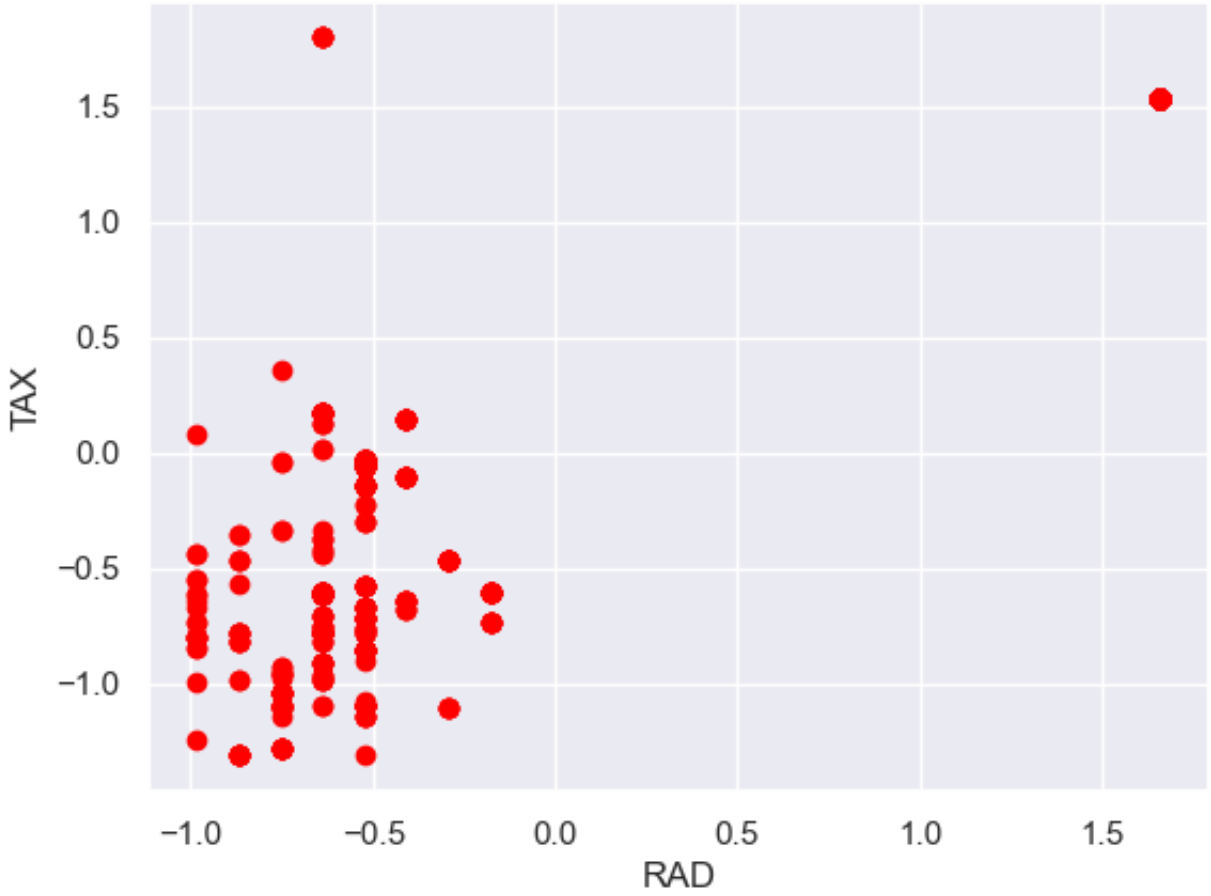
RAD and TAX have a correlation of 0.91. Let's plot those variables together

In [10]:
```
%matplotlib inline
import matplotlib.pyplot as plt

# plot data
plt.scatter(RAD, TAX,
            color='red', marker='o')

plt.xlabel('RAD')
plt.ylabel('TAX')

plt.show()
```



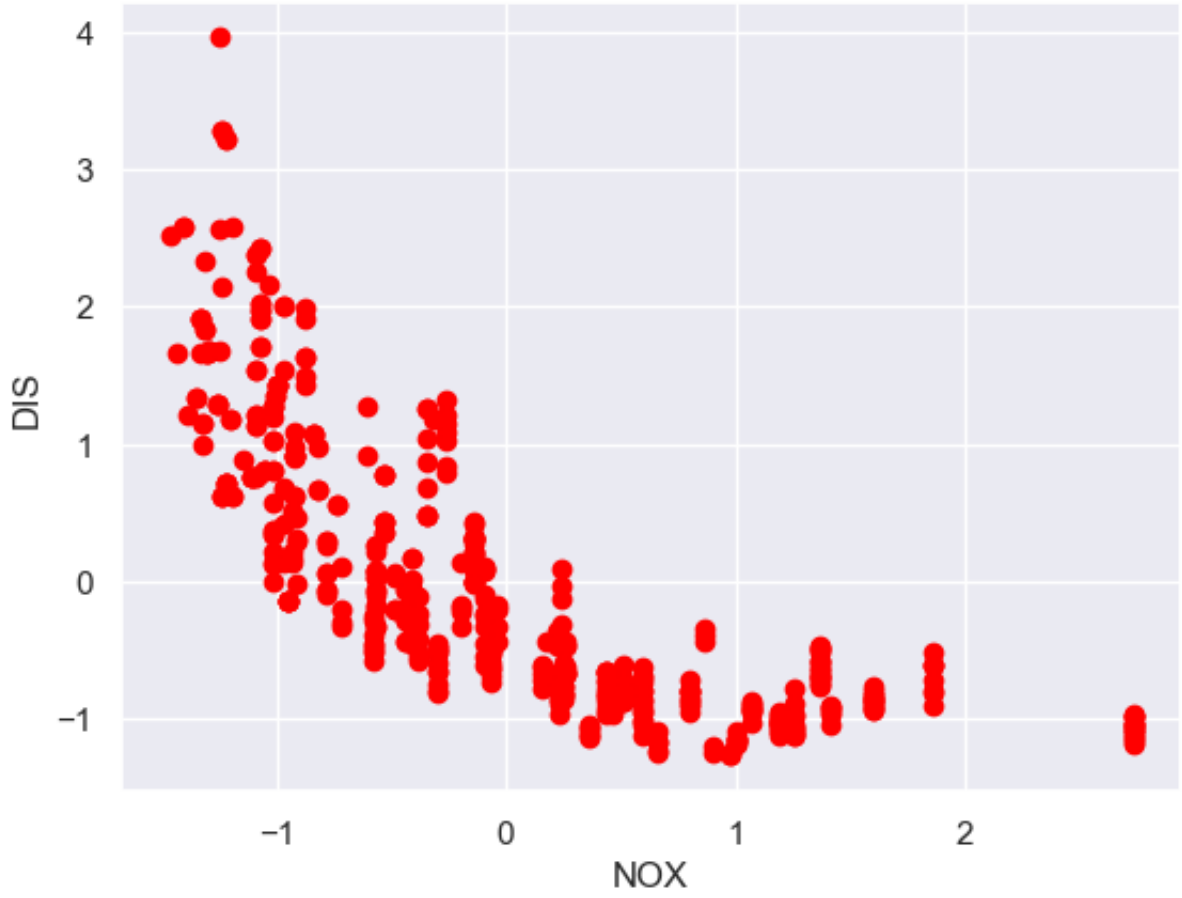
NOX and DIS have a correlation of -0.77. Let's plot those variables together

In [11]:
```
%matplotlib inline
import matplotlib.pyplot as plt

# plot data
plt.scatter(NOX, DIS,
            color='red', marker='o')

plt.xlabel('NOX')
plt.ylabel('DIS')

plt.show()
```



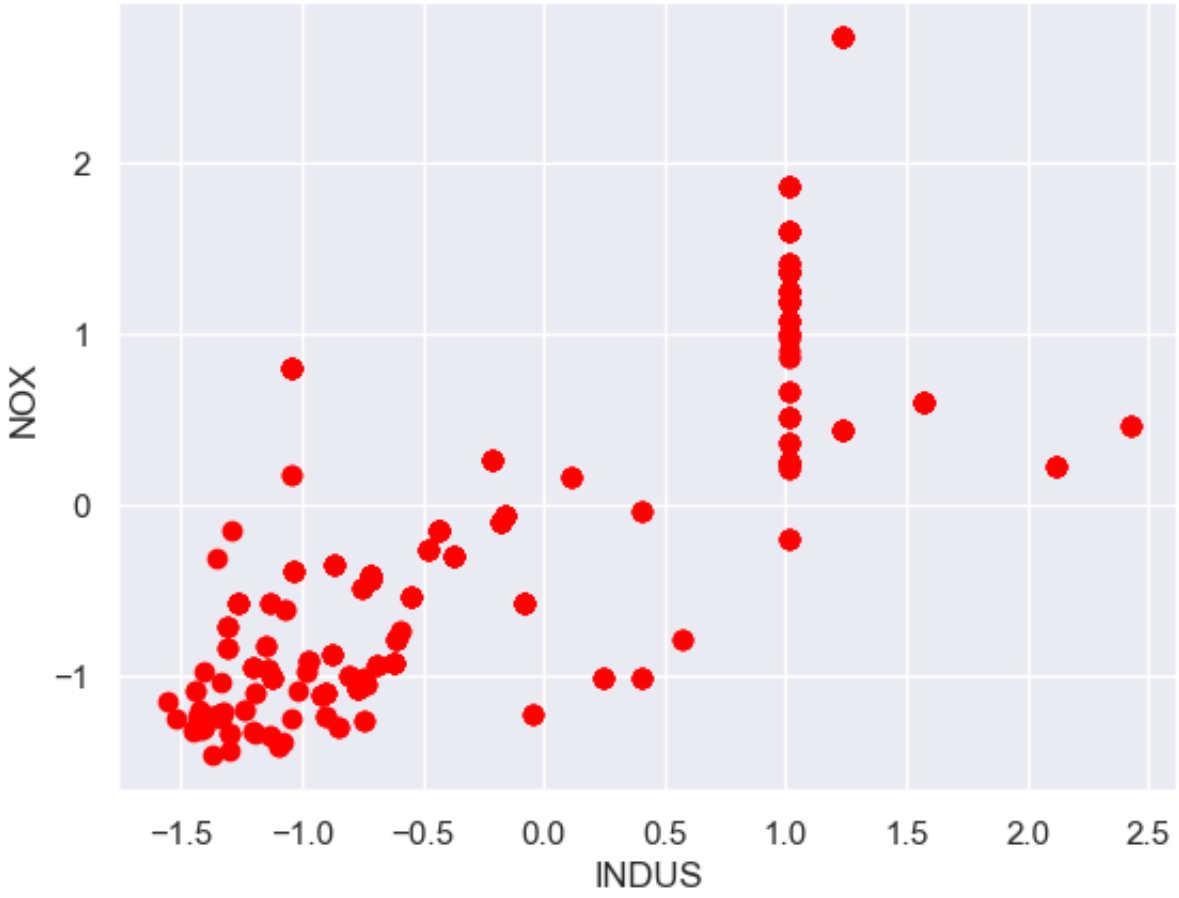
INDUS and NOX have a correlation of 0.76. Let's plot those variables together

In [12]:
```
%matplotlib inline
import matplotlib.pyplot as plt

# plot data
plt.scatter(INDUS, NOX,
            color='red', marker='o')

plt.xlabel('INDUS')
plt.ylabel('NOX')

plt.show()
```



3. Pick one feature that you think can be predicted by the other features in the dataset. The feature to be predicted needs to have numerical values.

First we will pick "MEDV" and see if it can be predicted by the other features

In [13]:
```
X1 = df.drop(['MEDV'], axis=1)
y1 = df['MEDV']
```

In [14]:
```
# Split the data into a training, validation, and testing sets
train_ratio = 0.75
validation_ratio = 0.15
test_ratio = 0.10

x1_train, x1_test, y1_train, y1_test = train_test_split(X1, y1, test_s
x1_val, x1_test, y1_val, y1_test = train_test_split(x1_test, y1_test,

print(x1_train.shape, x1_val.shape, x1_test.shape)
```

```
(379, 13) (76, 13) (51, 13)
```

In [15]:
```
reg1 = LinearRegression().fit(x1_train, y1_train)
reg1.score(x1_train, y1_train)
```

Out[15]:
```
0.748087259862344
```

In [16]:
```
reg1.score(x1_val, y1_val)
```

Out[16]:
```
0.7095686775934622
```

In [17]:
```
reg1.coef_
```

Out[17]:
```
array([-0.1200131 ,  0.07493879,  0.03644524,  0.07659512, -0.2045989
4,
        0.3337531 , -0.02830488, -0.32073347,  0.24403268, -0.1824612
7,
       -0.21729784,  0.1308848 , -0.40191952])
```

In [18]:
```
ridge1 = Ridge(alpha=1.0).fit(x1_train, y1_train)
ridge1.score(x1_train, y1_train)
```

Out[18]:
```
0.7480712069570565
```

In [19]:
```
ridge1.score(x1_val, y1_val)
```

Out[19]:
```
0.7101224105709807
```

In [20]:
```
ridge1.coef_
```

Out[20]:
```
array([-0.11868947,  0.07296915,  0.03362026,  0.07703152, -0.2012854
,
        0.33438851, -0.02837602, -0.31671567,  0.23518052, -0.1745458
5,
       -0.21619104,  0.13050042, -0.40054349])
```

In [21]:
```
lasso1 = Lasso(alpha=0.3).fit(x1_train, y1_train)
lasso1.score(x1_train, y1_train)
```

Out[21]:
```
0.5563525290217233
```

In [22]:
```
lasso1.score(x1_val, y1_val)
```

Out[22]:
```
0.5639840466423658
```

In [23]:
```
lasso1.coef_
```

Out[23]:
```
array([-0.        ,  0.        , -0.        ,  0.        , -0.
,
        0.22999153, -0.        ,  0.        , -0.        , -0.
,
       -0.0219538 ,  0.        , -0.30654977])
```

In [24]:
```
enet1 = ElasticNet(alpha=0.3).fit(x1_train, y1_train)
enet1.score(x1_train, y1_train)
```

Out[24]:
```
0.6403816753738955
```

In [25]:
```
enet1.score(x1_val, y1_val)
```

Out[25]:
```
0.6566129651074187
```

In [26]:
```
enet1.coef_
```

Out[26]:
```
array([-0.        ,  0.        , -0.        ,  0.        , -0.
,
        0.2826577 , -0.        , -0.        , -0.        , -0.
,
       -0.11458164,  0.01684473, -0.33293526])
```

In [27]:
```
tree1 = DecisionTreeRegressor(criterion="squared_error", random_state=
tree1.fit(x1_train, y1_train)
tree1.score(x1_train, y1_train)
```

Out[27]:
```
1.0
```

In [28]:
```
tree1.score(x1_val, y1_val)
```

Out[28]:
```
0.7493403525717777
```

In [29]:
```
classifiers1 = (("Linear", reg1),
                ("Ridge", ridge1),
                ("Lasso", lasso1),
                ("Elastic Net", enet1),
                ("Decision Tree", tree1))
print (f'{"Classifier":15} {"R2":4}  {"MSE":4}  {"MAE":4}')
for cf_name, cf in classifiers1:
    pred = cf.predict(x1_test)
    r2 = r2_score(y1_test, pred)
    mse = mean_squared_error(y1_test, pred)
    mae = mean_absolute_error(y1_test, pred)
    print (f"{cf_name:15} {r2:4.2f}  {mse:4.2f}  {mae:4.2f}")
```

```
Classifier      R2    MSE   MAE
Linear          0.67  0.41  0.44
Ridge           0.67  0.41  0.44
Lasso           0.49  0.63  0.57
Elastic Net     0.57  0.54  0.52
Decision Tree   0.87  0.17  0.31
```

Analysis of Classifiers for predicting "MEDV"

Decision Tree did the best job in terms of all 3 scores. Decision tree can be expected to outperform the linear regressors in most cases, especially when the solution is not linear. In addition, we saw high correlation between some features, which can be a problem for the linear regressors. We can see that Lasso and Elastic Net eliminated many of the features; this may lead to its lower scores for R2, MSE, and MAE. We had to reduce the alpha of Lasso from 1.0 to 0.3, because at 1.0, it set the value of all the coefficients to zero.

Now we will pick "RAD" and see if it can be predicted by the other features

In [30]:
```
X2 = df.drop(['RAD'], axis=1)
y2 = df['RAD']
```

In [31]:
```
# Split the data into a training, validation, and testing sets
train_ratio = 0.75
validation_ratio = 0.15
test_ratio = 0.10

x2_train, x2_test, y2_train, y2_test = train_test_split(X2, y2, test_s
x2_val, x2_test, y2_val, y2_test = train_test_split(x2_test, y2_test,

print(x2_train.shape, x2_val.shape, x2_test.shape)
```

```
(379, 13) (76, 13) (51, 13)
```

In [32]:
```
reg2 = LinearRegression().fit(x2_train, y2_train)
reg2.score(x2_train, y2_train)
```

Out[32]:
```
0.8684389595979438
```

In [33]:
```
reg2.score(x2_val, y2_val)
```

Out[33]:
```
0.8586916159996225
```

In [34]:
```
reg2.coef_
```

Out[34]:
```
array([ 0.15399545, -0.07389212, -0.22320312,  0.03790113,  0.1790195
6,
        0.0405148 , -0.04747883,  0.05140544,  0.84034329,  0.1252219
4,
       -0.06405499,  0.04345803,  0.12752719])
```

In [35]:
```
ridge2 = Ridge(alpha=1.0).fit(x2_train, y2_train)
ridge2.score(x2_train, y2_train)
```

Out[35]:
```
0.8684191027701267
```

In [36]:
```
ridge2.score(x2_val, y2_val)
```

Out[36]:
```
0.8592446521468772
```

In [37]:
```
ridge2.coef_
```

Out[37]:
```
array([ 0.15522561, -0.07182092, -0.2181567 ,  0.03774428,  0.1785080
1,
        0.04130536, -0.04687457,  0.05001968,  0.83314651,  0.1258522
7,
       -0.06469757,  0.04185555,  0.12509771])
```

In [38]:
```
lasso2 = Lasso(alpha=0.3).fit(x2_train, y2_train)
lasso2.score(x2_train, y2_train)
```

Out[38]:
```
0.7264536419776678
```

In [39]:
```
lasso2.score(x2_val, y2_val)
```

Out[39]:
```
0.7158517417443959
```

In [40]:
```
lasso2.coef_
```

Out[40]:
```
array([ 0.        , -0.        ,  0.        ,  0.        ,  0.
,
       -0.        ,  0.        , -0.        ,  0.59402364,  0.
,
       -0.        ,  0.        , -0.        ])
```

In [41]:
```
enet2 = ElasticNet(alpha=0.3).fit(x2_train, y2_train)
enet2.score(x2_train, y2_train)
```

Out[41]:
```
0.7726266894145735
```

In [42]:
```
enet2.score(x2_val, y2_val)
```

Out[42]:
```
0.7727257479944225
```

In [43]:
```
enet2.coef_
```

Out[43]:
```
array([ 0.09872461, -0.        ,  0.        ,  0.        ,  0.0241429
3,
        0.        ,  0.        , -0.        ,  0.58470806,  0.
,
       -0.        ,  0.        , -0.        ])
```

In [44]:
```
tree2 = DecisionTreeRegressor(criterion="squared_error", random_state=
tree2.fit(x2_train, y2_train)
tree2.score(x2_train, y2_train)
```

Out[44]:
```
1.0
```

In [45]:
```
tree2.score(x2_val, y2_val)
```

Out[45]:
```
0.9867849690772036
```

In [46]:
```
classifiers2 = (("Linear", reg2),
                ("Ridge", ridge2),
                ("Lasso", lasso2),
                ("Elastic Net", enet2),
                ("Decision Tree", tree2))
print (f'{"Classifier":15} {"R2":4}  {"MSE":4}  {"MAE":4}')
for cf_name, cf in classifiers2:
    pred = cf.predict(x2_test)
    r2 = r2_score(y2_test, pred)
    mse = mean_squared_error(y2_test, pred)
    mae = mean_absolute_error(y2_test, pred)
    print (f"{cf_name:15} {r2:4.2f}  {mse:4.2f}  {mae:4.2f}")
```

```
Classifier      R2    MSE   MAE
Linear          0.89  0.11  0.27
Ridge           0.89  0.12  0.28
Lasso           0.75  0.26  0.44
Elastic Net     0.81  0.21  0.39
Decision Tree   0.90  0.10  0.05
```

Analysis of Classifiers for predicting "RAD"

Again, as expected, Decision Tree did the best job in terms of all 3 scores. This test was interesting in that RAD had a 0.91 correlation with TAX as shown in Part 2. For Lasso, we again had to lower alpha to 0.3, but even then, all coefficients were set to 0 except for TAX. The R2, MSE and MAE scores were still not bad for Lasso, although not as good as for the other classifiers. As expected, since Elastic Net is a balance between Ridge and Lasso, its scores were about in the middle between those two. For this set, Linear and Ridge scores were very close to Decision Tree; this indicates that the solution is closer to linear than it was for predicting "MEDV".