In [116]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#import scikits.statsmodels.api as sm
import seaborn as sns
sns.set_style("whitegrid")
sns.set_context("poster")

from matplotlib import rcParams
from sklearn.datasets import load_boston
boston = load_boston()

%matplotlib inline
```

In [117]:

```
# for using other data set use
# data_set = pd.read_csv("file_name.extention")
# report = pd.read_csv("D:/USA_Housing/.csv")
```

In [118]:

```
print(boston.data.shape)
```

```
(506, 13)
```

In [119]:

```
print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [111]:

```
target = np.array(boston.target)
```

In [115]:

```
for i in range(1,50):
    print(target[i],end=" ")
```

```
21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.0 18.9 21.7 20.4 18.2 19.9 23.1
17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8 18.4 21.0 12.7 14.5 13.2
13.1 13.5 18.9 20.0 21.0 24.7 30.8 34.9 26.6 25.3 24.7 21.2 19.3 20.0 16.6 14.4
19.4
```

In [5]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (att
ribute 14) is usually the target.

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

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (https://arc
hive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Carnegie
Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that
 address regression
problems.

.. topic:: References
```

```
   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Da
ta and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In P
roceedings on the Tenth International Conference of Machine Learning, 236-243,
 University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [6]:

```python
import pandas as pd
bos = pd.DataFrame(boston.data)
bos.head(10)
```

Out[6]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10

In [7]:

```python
bos.columns = boston.feature_names
bos.head(5)
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [8]:
```
bos['PRICE'] = boston.target
bos.head(5)
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [9]:
```
bos.isnull().sum()
```

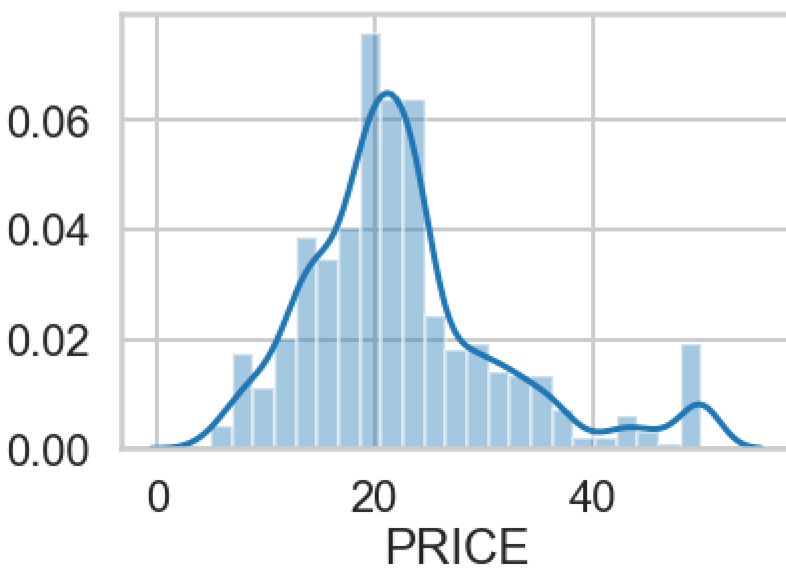
Out[9]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

In [10]:
```
# summary statistics
bos.describe()
```

Out[10]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.79
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.10
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.12
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.10
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.20
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.18
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.12

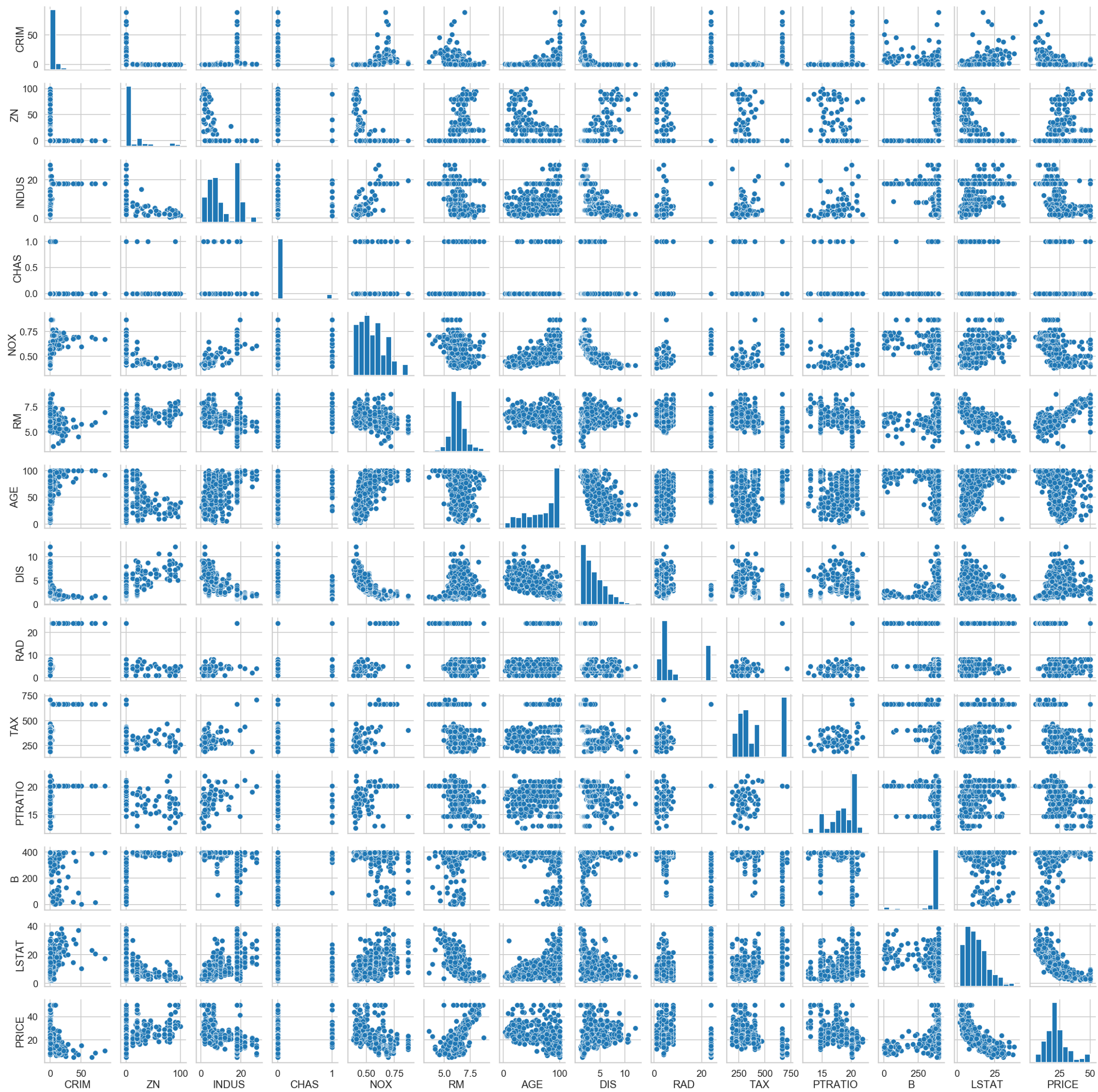
In [11]:
```
sns.distplot(bos['PRICE'])
plt.show()
```



In [12]:
```
sns.distplot(np.log(bos['PRICE']))
plt.show()
```

In [14]:
```
sns.pairplot(bos)
```

Out[14]: <seaborn.axisgrid.PairGrid at 0x1b343a079e8>

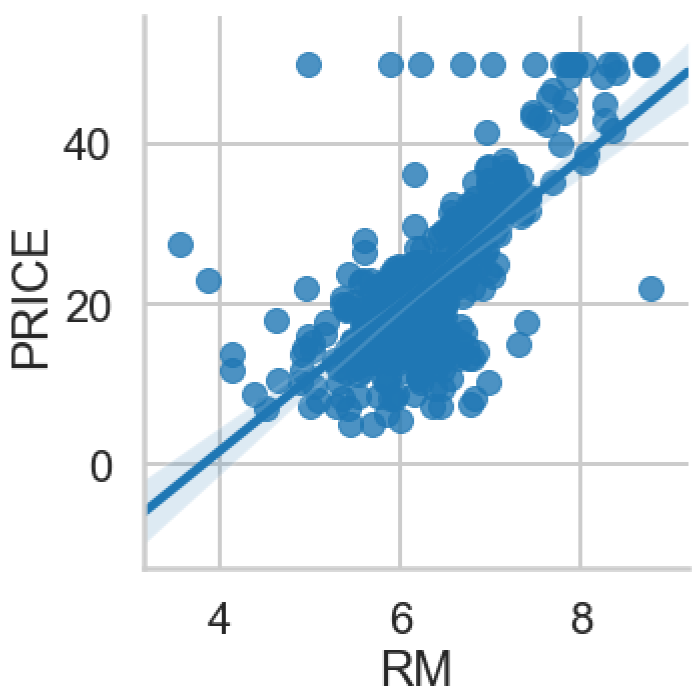


In [15]:
```python
corr_mat = bos.corr().round(2)
sns.heatmap(data=corr_mat, annot=True)
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1b34966ee10>

In [16]:
```python
sns.lmplot(x = 'RM', y = 'PRICE', data = bos)
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x1b34aaec438>



In [17]:

```
X = bos.drop('PRICE', axis = 1)
Y = bos['PRICE']
```

In [18]:
```
X
```

Out[18]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	1
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	1
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	2
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	1
10	0.22489	12.5	7.87	0.0	0.524	6.377	94.3	6.3467	5.0	311.0	15.2	392.52	2
11	0.11747	12.5	7.87	0.0	0.524	6.009	82.9	6.2267	5.0	311.0	15.2	396.90	1
12	0.09378	12.5	7.87	0.0	0.524	5.889	39.0	5.4509	5.0	311.0	15.2	390.50	1
13	0.62976	0.0	8.14	0.0	0.538	5.949	61.8	4.7075	4.0	307.0	21.0	396.90	
14	0.63796	0.0	8.14	0.0	0.538	6.096	84.5	4.4619	4.0	307.0	21.0	380.02	1
15	0.62739	0.0	8.14	0.0	0.538	5.834	56.5	4.4986	4.0	307.0	21.0	395.62	
16	1.05393	0.0	8.14	0.0	0.538	5.935	29.3	4.4986	4.0	307.0	21.0	386.85	
17	0.78420	0.0	8.14	0.0	0.538	5.990	81.7	4.2579	4.0	307.0	21.0	386.75	1
18	0.80271	0.0	8.14	0.0	0.538	5.456	36.6	3.7965	4.0	307.0	21.0	288.99	1
19	0.72580	0.0	8.14	0.0	0.538	5.727	69.5	3.7965	4.0	307.0	21.0	390.95	1
20	1.25179	0.0	8.14	0.0	0.538	5.570	98.1	3.7979	4.0	307.0	21.0	376.57	2
21	0.85204	0.0	8.14	0.0	0.538	5.965	89.2	4.0123	4.0	307.0	21.0	392.53	1
22	1.23247	0.0	8.14	0.0	0.538	6.142	91.7	3.9769	4.0	307.0	21.0	396.90	1
23	0.98843	0.0	8.14	0.0	0.538	5.813	100.0	4.0952	4.0	307.0	21.0	394.54	1
24	0.75026	0.0	8.14	0.0	0.538	5.924	94.1	4.3996	4.0	307.0	21.0	394.33	1
25	0.84054	0.0	8.14	0.0	0.538	5.599	85.7	4.4546	4.0	307.0	21.0	303.42	1
26	0.67191	0.0	8.14	0.0	0.538	5.813	90.3	4.6820	4.0	307.0	21.0	376.88	1
27	0.95577	0.0	8.14	0.0	0.538	6.047	88.8	4.4534	4.0	307.0	21.0	306.38	1
28	0.77299	0.0	8.14	0.0	0.538	6.495	94.4	4.4547	4.0	307.0	21.0	387.94	1
29	1.00245	0.0	8.14	0.0	0.538	6.674	87.3	4.2390	4.0	307.0	21.0	380.23	1
...	...	...	...	...	...	...	...	...	...	...	...	...	
476	4.87141	0.0	18.10	0.0	0.614	6.484	93.6	2.3053	24.0	666.0	20.2	396.21	1
477	15.02340	0.0	18.10	0.0	0.614	5.304	97.3	2.1007	24.0	666.0	20.2	349.48	2
478	10.23300	0.0	18.10	0.0	0.614	6.185	96.7	2.1705	24.0	666.0	20.2	379.70	1

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
479	14.33370	0.0	18.10	0.0	0.614	6.229	88.0	1.9512	24.0	666.0	20.2	383.32	1
480	5.82401	0.0	18.10	0.0	0.532	6.242	64.7	3.4242	24.0	666.0	20.2	396.90	1
481	5.70818	0.0	18.10	0.0	0.532	6.750	74.9	3.3317	24.0	666.0	20.2	393.07	
482	5.73116	0.0	18.10	0.0	0.532	7.061	77.0	3.4106	24.0	666.0	20.2	395.28	
483	2.81838	0.0	18.10	0.0	0.532	5.762	40.3	4.0983	24.0	666.0	20.2	392.92	1
484	2.37857	0.0	18.10	0.0	0.583	5.871	41.9	3.7240	24.0	666.0	20.2	370.73	1
485	3.67367	0.0	18.10	0.0	0.583	6.312	51.9	3.9917	24.0	666.0	20.2	388.62	1
486	5.69175	0.0	18.10	0.0	0.583	6.114	79.8	3.5459	24.0	666.0	20.2	392.68	1
487	4.83567	0.0	18.10	0.0	0.583	5.905	53.2	3.1523	24.0	666.0	20.2	388.22	1
488	0.15086	0.0	27.74	0.0	0.609	5.454	92.7	1.8209	4.0	711.0	20.1	395.09	1
489	0.18337	0.0	27.74	0.0	0.609	5.414	98.3	1.7554	4.0	711.0	20.1	344.05	2
490	0.20746	0.0	27.74	0.0	0.609	5.093	98.0	1.8226	4.0	711.0	20.1	318.43	2
491	0.10574	0.0	27.74	0.0	0.609	5.983	98.8	1.8681	4.0	711.0	20.1	390.11	1
492	0.11132	0.0	27.74	0.0	0.609	5.983	83.5	2.1099	4.0	711.0	20.1	396.90	1
493	0.17331	0.0	9.69	0.0	0.585	5.707	54.0	2.3817	6.0	391.0	19.2	396.90	1
494	0.27957	0.0	9.69	0.0	0.585	5.926	42.6	2.3817	6.0	391.0	19.2	396.90	1
495	0.17899	0.0	9.69	0.0	0.585	5.670	28.8	2.7986	6.0	391.0	19.2	393.29	1
496	0.28960	0.0	9.69	0.0	0.585	5.390	72.9	2.7986	6.0	391.0	19.2	396.90	2
497	0.26838	0.0	9.69	0.0	0.585	5.794	70.6	2.8927	6.0	391.0	19.2	396.90	1
498	0.23912	0.0	9.69	0.0	0.585	6.019	65.3	2.4091	6.0	391.0	19.2	396.90	1
499	0.17783	0.0	9.69	0.0	0.585	5.569	73.5	2.3999	6.0	391.0	19.2	395.77	1
500	0.22438	0.0	9.69	0.0	0.585	6.027	79.7	2.4982	6.0	391.0	19.2	396.90	1
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	

506 rows × 13 columns

In [19]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, rand
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

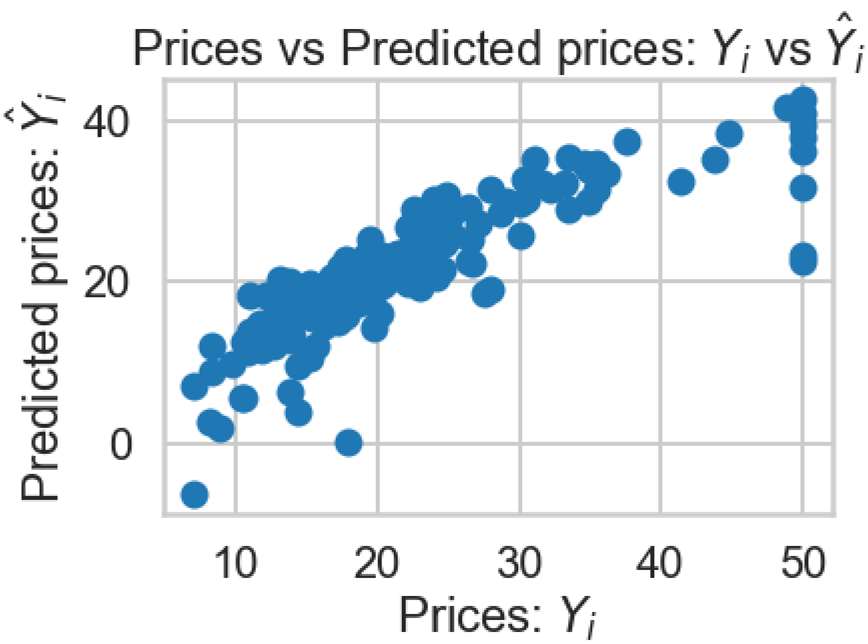
```
(339, 13)
(167, 13)
(339,)
(167,)
```

In [20]:
```
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
lm.fit(X_train, Y_train)

Y_pred = lm.predict(X_test)

plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
plt.show()
```



In [21]:
```
df1 = pd.DataFrame({'Actual': Y_test, 'Predicted':Y_pred})
df2 = df1.head(10)
df2
```

Out[21]:

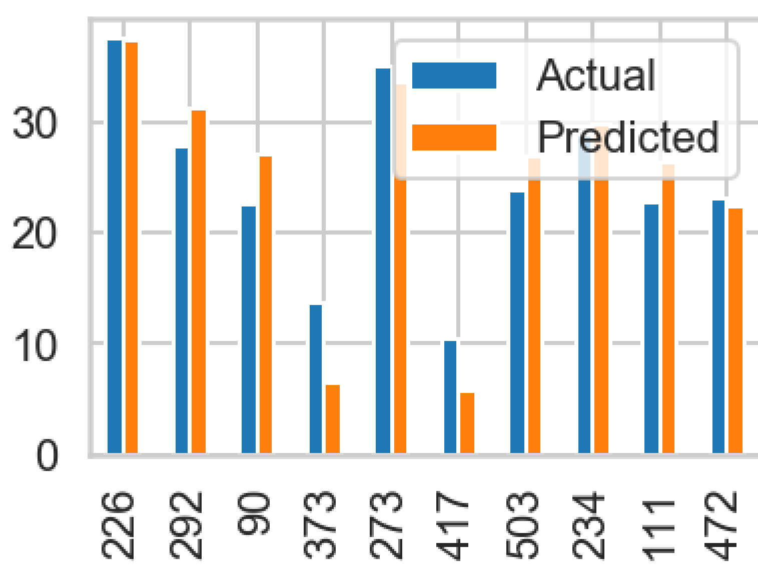
	Actual	Predicted
226	37.6	37.467236
292	27.9	31.391547
90	22.6	27.120196
373	13.8	6.468433
273	35.2	33.629667
417	10.4	5.670680
503	23.9	27.039467
234	29.0	29.927047
111	22.8	26.356613
472	23.2	22.452460

In [22]:
```
import sklearn
mse = sklearn.metrics.mean_squared_error(Y_test,Y_pred)
mse
```

Out[22]: 28.530458765974583

In [23]:
```
df2.plot(kind = 'bar')
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1b34aa3aa58>

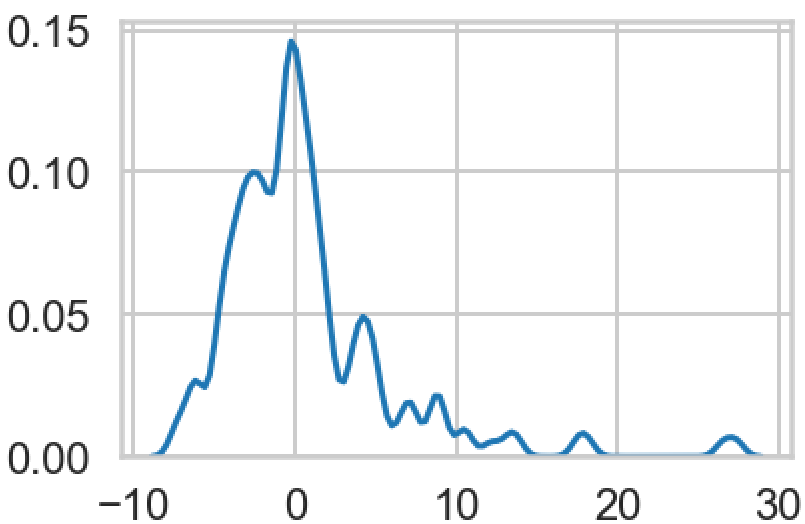


In [105]:
```
from sklearn import metrics
from sklearn.metrics import r2_score
print('MAE', metrics.mean_absolute_error(Y_test,Y_pred))
print('MSE', metrics.mean_squared_error(Y_test,Y_pred))
print('RMSE', np.sqrt(metrics.mean_squared_error(Y_test,Y_pred)))
print('R squared Score', r2_score(Y_test,Y_pred))
```

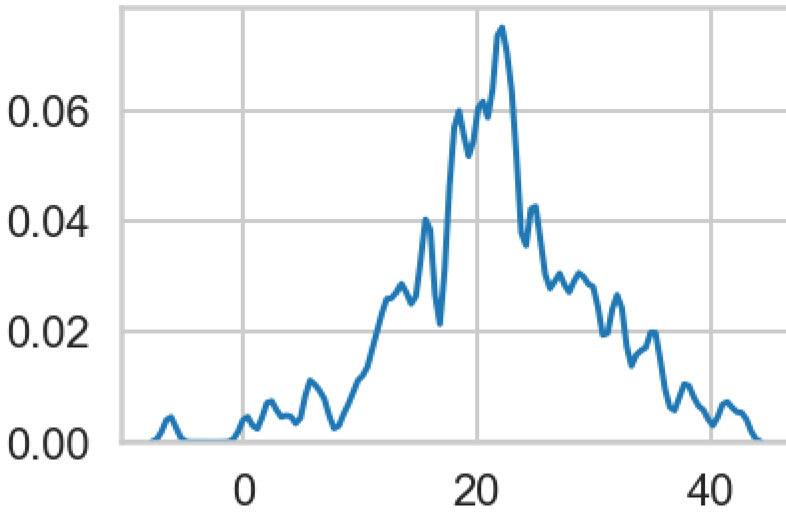
```
MAE 3.4550349322483482
MSE 28.530458765974583
RMSE 5.341391089030514
R squared Score 0.6956551656111607
```

In [25]:
```
delta_y = Y_test - Y_pred;

import seaborn as sns;
import numpy as np;
sns.set_style('whitegrid')
sns.kdeplot(np.array(delta_y), bw=0.5)
plt.show()
```

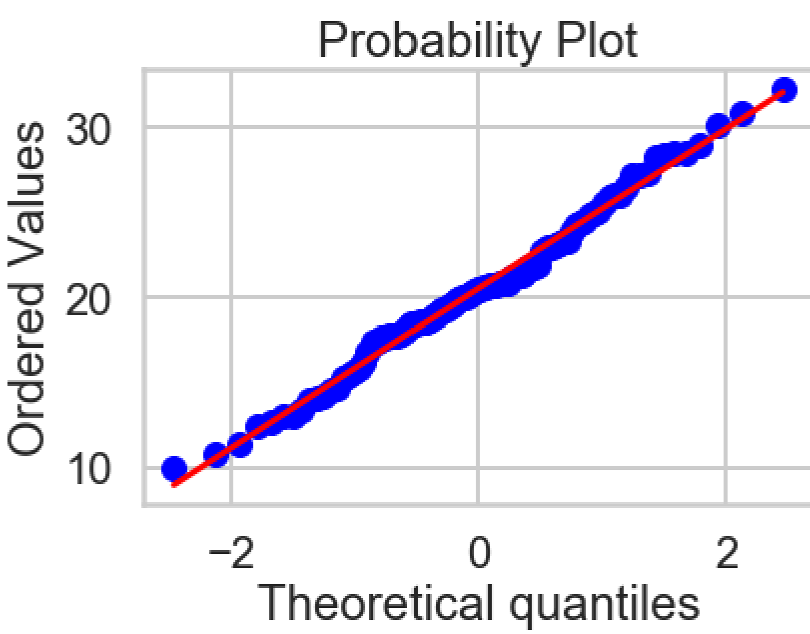


In [26]:
```
sns.set_style('whitegrid')
sns.kdeplot(np.array(Y_pred), bw=0.5)
plt.show()
```

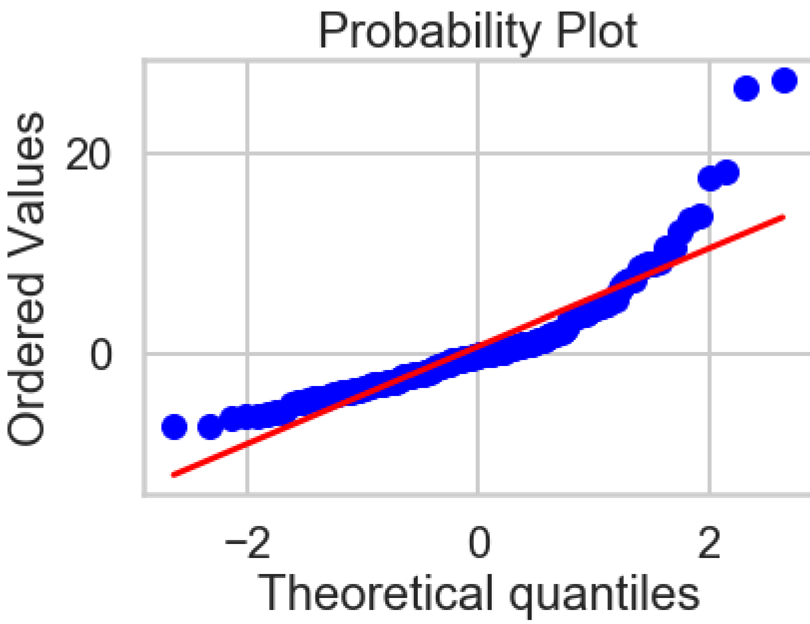


In [27]:
```
import numpy as np
import pylab
import scipy.stats as stats

measurements = np.random.normal(loc = 20, scale = 5, size=100)
stats.probplot(measurements, dist="norm", plot=pylab)
pylab.show()
```



In [28]:
```
stats.probplot(delta_y.values, dist="norm", plot=pylab)
pylab.show()
```



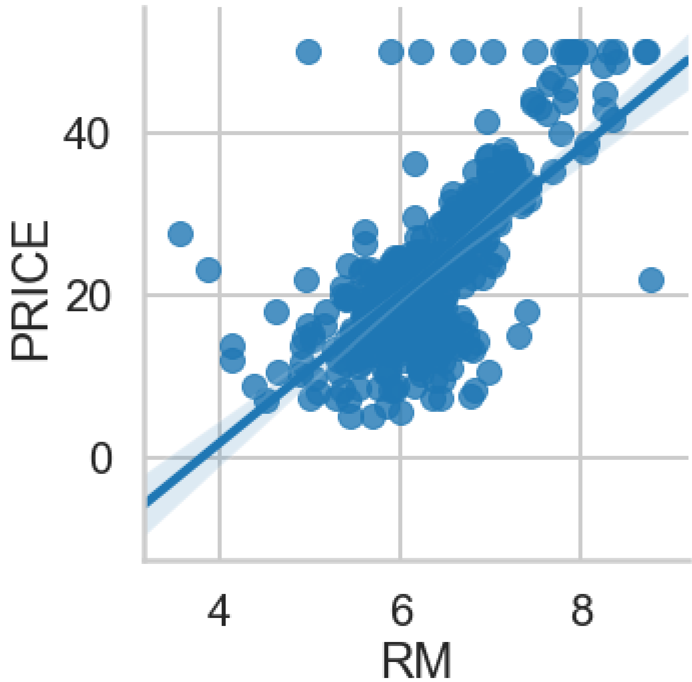
In [29]:
```
# now we are able to see that the error of aur model is very high
# and this is not a right sighn


#Lets do some feature engineering to make our model perfect
```

In [30]:
```
#Lets observe high corealtion RM(Avg no of rooms) & Price is highly
#corelated =>
```

In [32]:
```
sns.lmplot(x = 'RM',y = 'PRICE',data = bos)
```

Out[32]: <seaborn.axisgrid.FacetGrid at 0x1b34d031e48>



In [ ]:
```
#lets try other regretor model in order to improve the accuracy
```

In [ ]:
```
# here we have take the decision tree regrator in order to improve
# the accuracy
```

In [45]:
```
x = X #Features
```

In [46]:
```
x.head()
```

Out[46]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [57]:
```
y = boston.target
```

In [61]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, randor
```

In [62]:
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(354, 13)
(354,)
(152, 13)
(152,)
```

In [63]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [64]:
```
regressor = DecisionTreeRegressor(random_state = 0)
```

In [65]:
```
regressor.fit(x_train,y_train)
```

Out[65]:
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=0, splitter='best')
```

In [66]:
```
predictions = regressor.predict(x_test)
```

In [67]:
```
from sklearn import metrics

print('Mean Abs Error:', metrics.mean_absolute_error(y_test, predictions))
print('Mean Sqrd Error:', metrics.mean_squared_error(y_test, predictions))
print('Root Mean Sqrd Error:', np.sqrt(metrics.mean_squared_error(y_test, predic
```

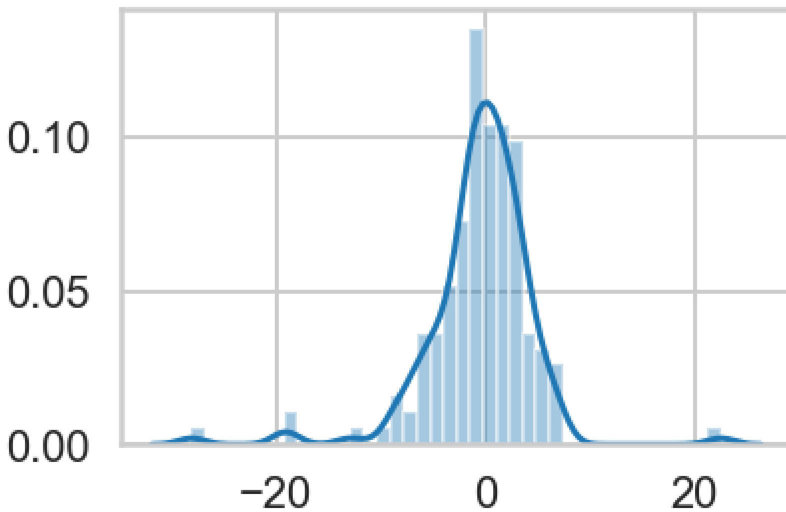
```
Mean Abs Error: 3.3019736842105267
Mean Sqrd Error: 26.002171052631578
Root Mean Sqrd Error: 5.099232398374443
```

In [104]:
```
from sklearn.metrics import r2_score
print('R squared score', r2_score(y_test,predictions))
```

```
R squared score 0.7485681652399021
```

In [70]:
```
sns.distplot((y_test-predictions),bins = 40)
```

Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x1b34d1665c0>

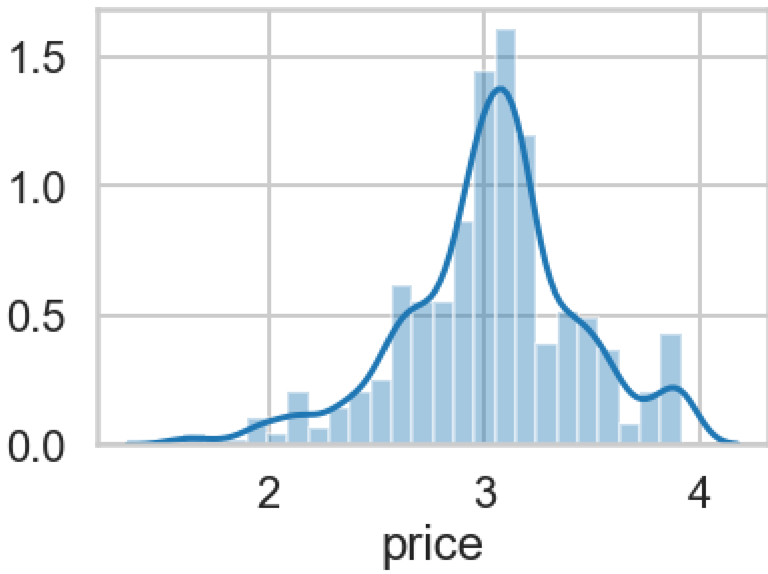


In [71]:
```
# now we will see that our model is getting more worst then privious one
# now it is conformed that the data has some incosistencey and
# have multiple unarranged features
# in order to improve the model we have to perform some feature
#engineering on data set
```

In [72]:
```
# now we have : x as feature
```

In [74]:
```
sns.distplot(bos['price'])
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x1b34c6ff5f8>

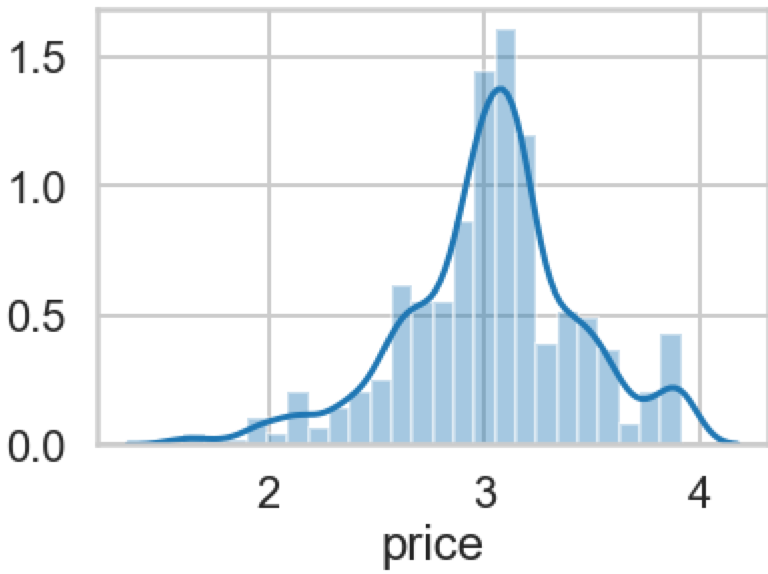


In [75]:
```
# the test data set is not normally distributed this data is left skewed,
# in order to distribute the data
# set normally need to perform some feature engineering
```

In [76]:
```
bos['price'] =np.log(boston.target)
```

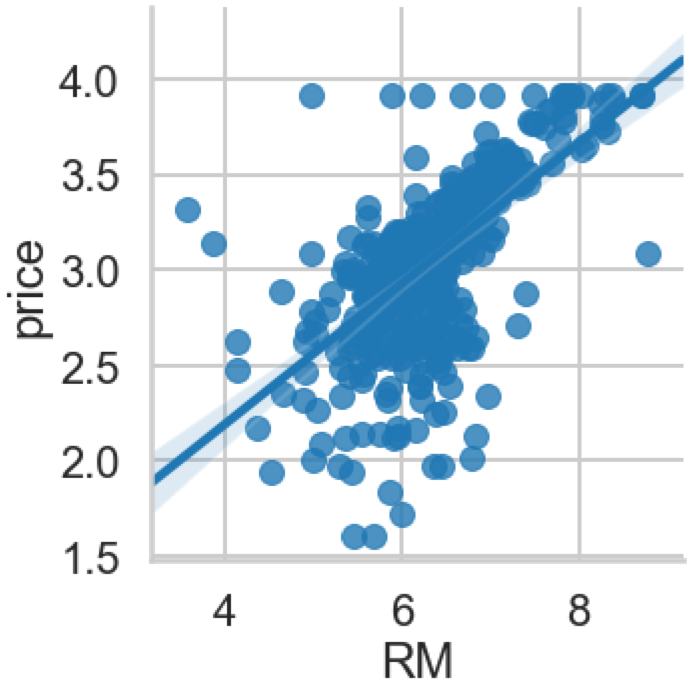
In [81]:
```
sns.distplot(bos['price'])
```

Out[81]: <matplotlib.axes._subplots.AxesSubplot at 0x1b3506cf8d0>



In [79]:
```
# to cheack that which feature is highly corelated :=>
sns.lmplot(x = 'RM',y = 'price',data = bos)
```

Out[79]: <seaborn.axisgrid.FacetGrid at 0x1b35099efd0>



In [80]:
```
# and we get that avg no of room is highly corelated with price
```

In [82]:
```
y = bos['price']
```

In [83]:
```
x.head(2)
```

Out[83]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.9	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.9	9.14

In [84]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, rando
```

In [85]:
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(354, 13)
(354,)
(152, 13)
(152,)
```

In [86]:
```
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
```

In [87]:
```
lm.fit(x_train,y_train)
```

Out[87]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [88]:
```
print('Coefficients: \n',lm.coef_)
```

```
Coefficients: 
 [-1.12345733e-02  1.00658774e-03 -2.12457953e-04  1.15523512e-01
 -5.61601723e-01  9.06111828e-02  3.75059044e-04 -3.80922354e-02
  1.41671662e-02 -5.05322834e-04 -3.61599909e-02  5.95412525e-04
 -3.00651151e-02]
```

In [89]:
```
coeffecients = pd.DataFrame(lm.coef_,x.columns)
coeffecients.columns = ['Coeffec']
coeffecients
```

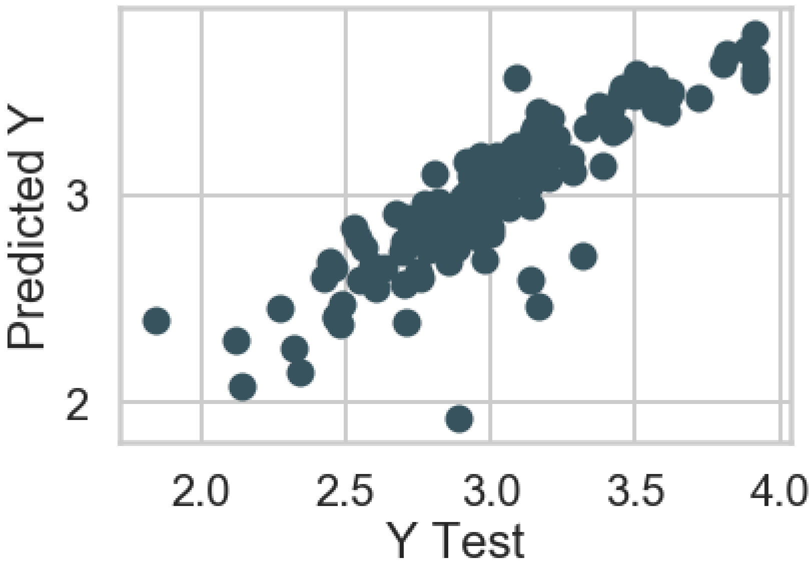
Out[89]:

	Coeffec
CRIM	-0.011235
ZN	0.001007
INDUS	-0.000212
CHAS	0.115524
NOX	-0.561602
RM	0.090611
AGE	0.000375
DIS	-0.038092
RAD	0.014167
TAX	-0.000505
PTRATIO	-0.036160
B	0.000595
LSTAT	-0.030065

In [90]:
```
predictions = lm.predict(x_test)
```

In [91]:
```
sns.set_palette("GnBu_d")
sns.set_style('whitegrid')
plt.scatter(y_test,predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

Out[91]:
```
Text(0, 0.5, 'Predicted Y')
```



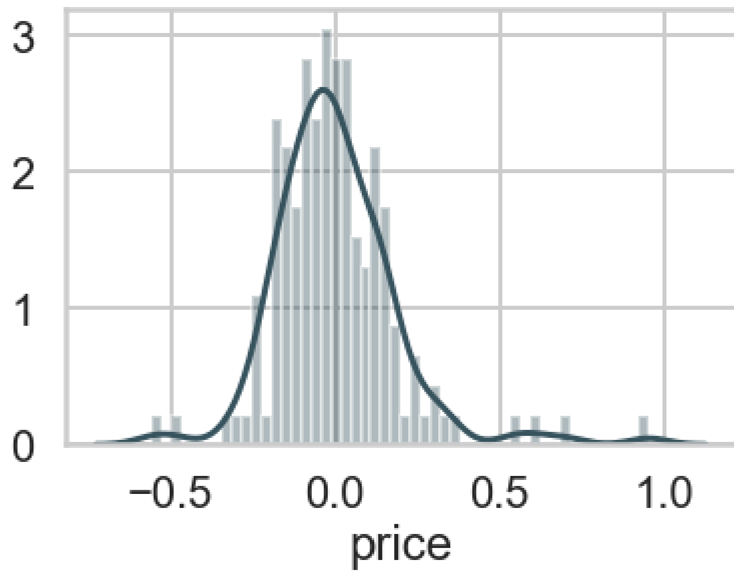
In [92]:
```
from sklearn import metrics

print('Mean Abs Error:', metrics.mean_absolute_error(y_test, predictions))
print('Mean Sqrd Error:', metrics.mean_squared_error(y_test, predictions))
print('Root Mean Sqrd Error:', np.sqrt(metrics.mean_squared_error(y_test, predic
```

```
Mean Abs Error: 0.1312917395101476
Mean Sqrd Error: 0.03515360328532382
Root Mean Sqrd Error: 0.18749294196135444
```

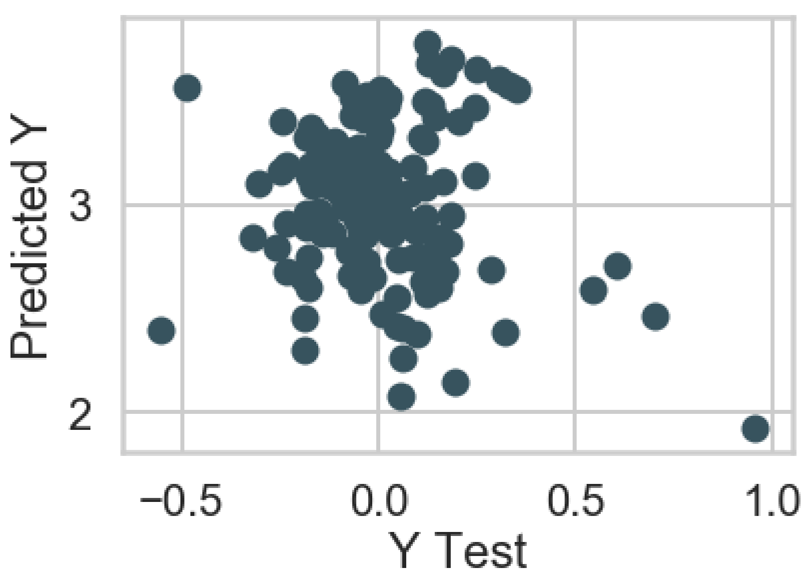
In [93]:
```
sns.distplot((y_test-predictions),bins = 50)
```

Out[93]: <matplotlib.axes._subplots.AxesSubplot at 0x1b350713a20>



In [100]:
```
#performing Heteroskedasticity test generally doesn't show nay pattern
plt.scatter(y_test-predictions,predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

Out[100]: Text(0, 0.5, 'Predicted Y')



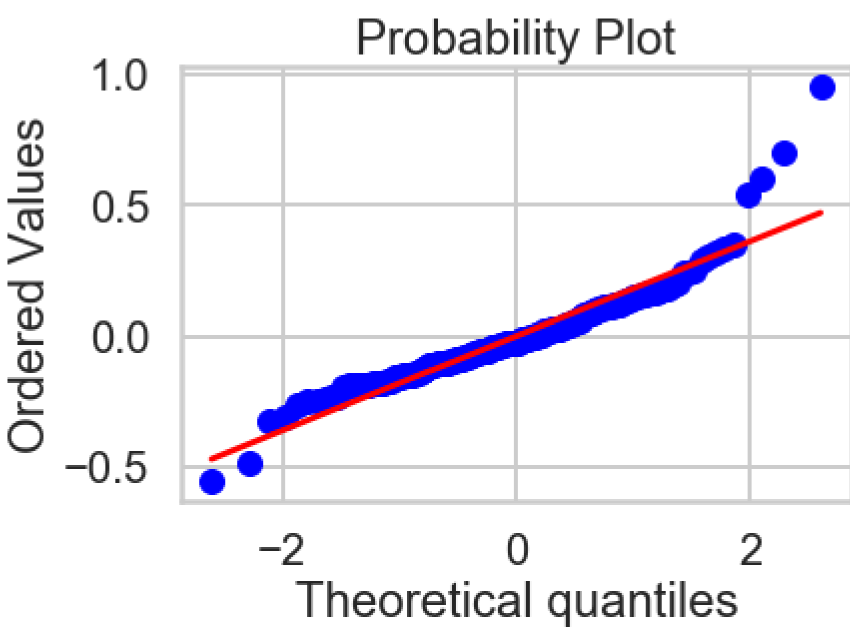
In [101]:
```
import numpy as np
import pylab
import scipy.stats as stats
```

In [94]:
```
df1 = pd.DataFrame({'Actual': y_test, 'Predicted':predictions})
df2 = df1.head(10)
df2
```

Out[94]:

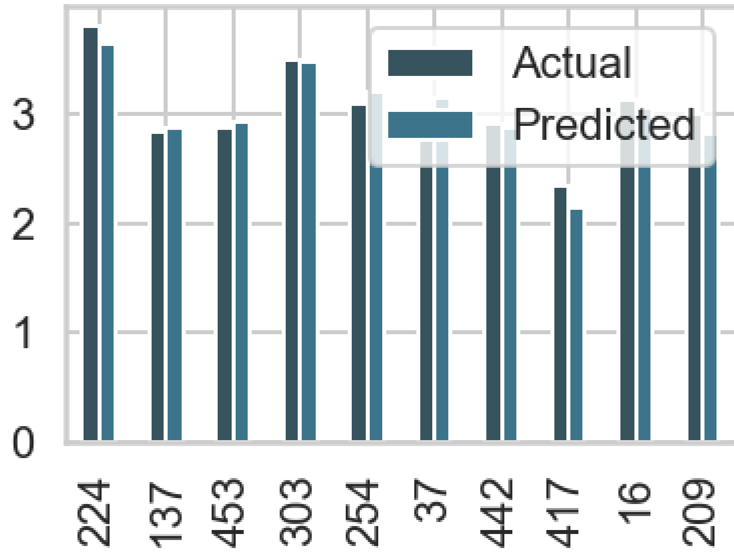
	Actual	Predicted
224	3.802208	3.638967
137	2.839078	2.882396
453	2.879198	2.935271
303	3.499533	3.480778
254	3.086487	3.196624
37	3.044522	3.146555
442	2.912351	2.876369
417	2.341806	2.147546
16	3.139833	3.061331
209	2.995732	2.819070

In [102]:
```
stats.probplot((y_test-predictions).values, dist="norm", plot=pylab)
pylab.show()
```



In [95]:
```
df2.plot(kind = 'bar')
```

Out[95]: <matplotlib.axes._subplots.AxesSubplot at 0x1b34fe274a8>



In [103]:
```
from sklearn.metrics import r2_score
print('R squared Score', r2_score(y_test,predictions))
```

```
R squared Score 0.7485681652399021
```

In [ ]:
```
#now we have reached to 75% accurate model which is a positive
#result for the business and on the perspective of non-medical science
# the model is more accurate then other model(i.e. implemented upper)accurate.

# So this is the end of the coding part.
```