In [1]:

```
from sklearn.datasets import load_boston
boston = load_boston()
```

In [2]:

```
boston.data.shape
```

Out[2]:

```
(506, 13)
```

In [3]:

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

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

In [4]:

```
print(boston.target)
```

```
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  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.  12.7 14.5 13.2 13.1 13.5 18.9 20.  21.  24.7 30.8 34.9 26.6
 25.3 24.7 21.2 19.3 20.  16.6 14.4 19.4 19.7 20.5 25.  23.4 18.9 35.4
 24.7 31.6 23.3 19.6 18.7 16.  22.2 25.  33.  23.5 19.4 22.  17.4 20.9
 24.2 21.7 22.8 23.4 24.1 21.4 20.  20.8 21.2 20.3 28.  23.9 24.8 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22.  22.9 25.  20.6 28.4 21.4 38.7
 43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22.  20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18.  14.3 19.2 19.6 23.  18.4 15.6 18.1 17.4 17.1 13.3 17.8
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4
 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.8
 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.
 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.
 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.1
 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  25.1 31.5
 23.7 23.3 22.  20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8
 29.6 42.8 21.9 20.9 44.  50.  36.  30.1 33.8 43.1 48.8 31.  36.5 22.8
 30.7 50.  43.5 20.7 21.1 25.2 24.4 35.2 32.4 32.  33.2 33.1 29.1 35.1
 45.4 35.4 46.  50.  32.2 22.  20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9
 21.7 28.6 27.1 20.3 22.5 29.  24.8 22.  26.4 33.1 36.1 28.4 33.4 28.2
 22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21.  23.8 23.1
 20.4 18.5 25.  24.6 23.  22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.1
 19.5 18.5 20.6 19.  18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6
 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25.  19.9 20.8 16.8
 21.9 27.5 21.9 23.1 50.  50.  50.  50.  50.  13.8 13.8 15.  13.9 13.3
 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 10.2 11.5 15.1 23.2
  9.7 13.8 12.7 13.1 12.5  8.5  5.   6.3  5.6  7.2 12.1  8.3  8.5  5.
 11.9 27.9 17.2 27.5 15.  17.2 17.9 16.3  7.   7.2  7.5 10.4  8.8  8.4
 16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.   9.5 14.5 14.1 16.1 14.3
 11.7 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
 14.1 13.  13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.  16.4 17.7
 19.5 20.2 21.4 19.9 19.  19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3
 16.7 12.  14.6 21.4 23.  23.7 25.  21.8 20.6 21.2 19.1 20.6 15.2  7.
  8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9
 22.  11.9]
```

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
(attribute 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 25,0
00 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds rive
r; 0 otherwise)
        - 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 town
        - 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/


This dataset was taken from the StatLib library which is maintained at Car
negie 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 diagnost
ics
...', 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 Influenti
al Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning.
In Proceedings on the Tenth International Conference of Machine Learning,
236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [6]:

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

```
        0     1     2    3      4      5     6       7    8      9  
10  \
0  0.00632  18.0  2.31  0.0  0.538  6.575  65.2  4.0900  1.0  296.0  1
5.3
1  0.02731   0.0  7.07  0.0  0.469  6.421  78.9  4.9671  2.0  242.0  1
7.8
2  0.02729   0.0  7.07  0.0  0.469  7.185  61.1  4.9671  2.0  242.0  1
7.8
3  0.03237   0.0  2.18  0.0  0.458  6.998  45.8  6.0622  3.0  222.0  1
8.7
4  0.06905   0.0  2.18  0.0  0.458  7.147  54.2  6.0622  3.0  222.0  1
8.7

       11    12  
0  396.90  4.98  
1  396.90  9.14  
2  392.83  4.03  
3  394.63  2.94  
4  396.90  5.33  
```

In [7]:

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

In [8]:

```
from sklearn.model_selection import  train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size = 0.2, random_state = 0)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(404, 13)
(102, 13)
(404,)
(102,)
```

In [9]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [10]:

```
scaler.fit(X_train)
```

Out[10]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [11]:

```
xtr = scaler.transform(X_train)
xte = scaler.transform(X_test)
```

In [12]:

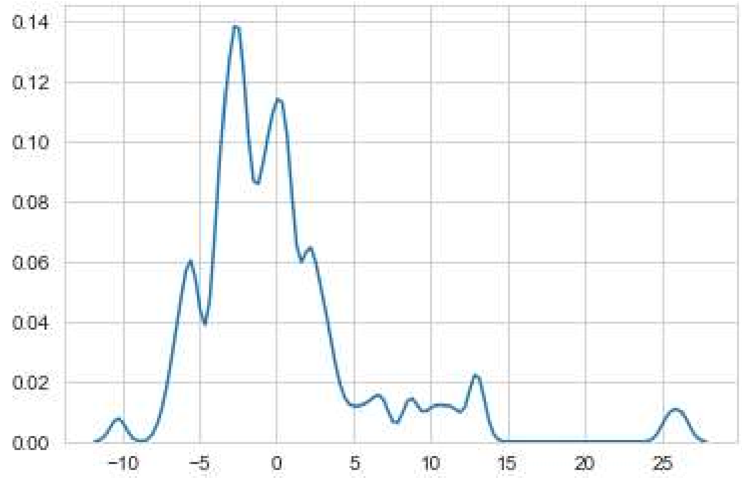
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression(normalize = True,n_jobs = 7)
lr.fit(xtr,y_train)
pred = lr.predict(xte)
```

In [16]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.scatter(y_test,pred)
plt.xlabel('Actual price')
plt.ylabel('Preducted Price')
plt.title('Actual VS Predicted Price')
plt.show()
```

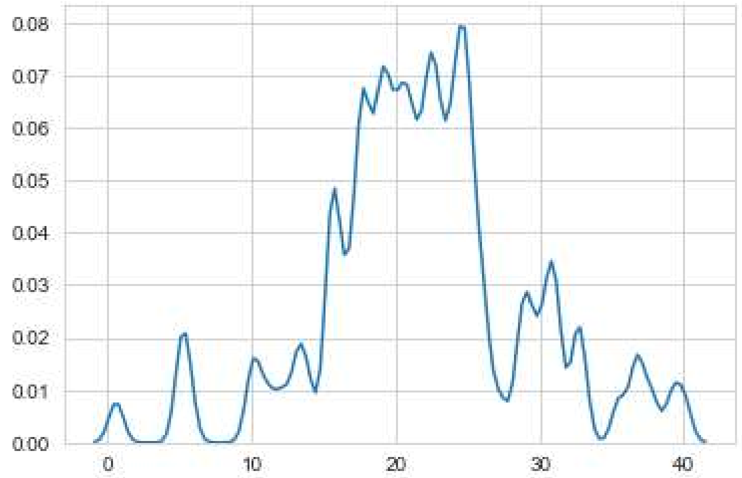
In [17]:

```
delta_y = y_test - 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 [18]:

```
sns.set_style('whitegrid')
sns.kdeplot(np.array(pred),bw = 0.5)
plt.show()
```



Implementation of SGD with Linear Regression

In [65]:

```
def loss_f(m, b, X, y):
    total_Error = 0
    for i in range(0, len(X)):
        x = X
        y = y
        total_Error += (y[:,i] - (np.dot(x[i] , m) + b)) ** 2
    return total_Error / len(x)
```

In [68]:

```
 def linear_reg_sgd(w_present, b_present, X_train, Y_train,X_test, Y_test, learning_rat
e =0.0001,epochs = 100):

        deriv_m = 0
        deriv_b = 0
        training_loss = []
        testing_loss = []
        for j in range(1, epochs + 1):
            y = np.asmatrix(Y_train)
            x = np.asmatrix(X_train)
            for i in range(len(x)):
                deriv_m += np.dot(-2*x[i].T , (y[:,i] - np.dot(x[i] , w_present) + b_pr
esent))
                deriv_b += -2*(y[:,i] - (np.dot(x[i] , w_present) + b_present))
            w1 = w_present - learning_rate * deriv_m
            b1 = b_present - learning_rate * deriv_b
            if (w_present==w1).all():
                break
            else:
                w_present = w1
                b_present = b1
                learning_rate = learning_rate/2
            training_err = loss_f(w_present,b_present, x, y)
            training_loss.append(training_err)
            testing_err = loss_f(w_present, b_present, np.asmatrix(X_test), np.asmatrix
(Y_test))
            testing_loss.append(testing_err)
            print("After {0} epoch training error = {1} and testing error = {2}".format
(j, training_err, testing_err))
        return w_present, b_present, training_loss, testing_loss
```

In [51]:

```
w_present_random = np.random.rand(13)
w_present = np.asmatrix(w0_random).T
b_present= np.random.rand()
```

In [69]:

```
best_slop, best_intercept, training_loss, testing_loss = linear_reg_sgd(w_present, b_pr
esent, X_train, y_train,X_test, y_test)
print("Slop: {} \n y_intercept: {}".format(best_slop, best_intercept))
```

```
After 1 epoch training error = [[6.12396572e+13]] and testing error = [[6.
36916347e+13]]
After 2 epoch training error = [[9.63188184e+21]] and testing error = [[1.
00218684e+22]]
After 3 epoch training error = [[3.78640664e+29]] and testing error = [[3.
93984677e+29]]
After 4 epoch training error = [[3.7194227e+36]] and testing error = [[3.8
7015817e+36]]
After 5 epoch training error = [[9.12531705e+42]] and testing error = [[9.
49513674e+42]]
After 6 epoch training error = [[5.5863473e+48]] and testing error = [[5.8
1274403e+48]]
After 7 epoch training error = [[8.51687058e+53]] and testing error = [[8.
86203203e+53]]
After 8 epoch training error = [[3.2212517e+58]] and testing error = [[3.3
517987e+58]]
After 9 epoch training error = [[2.99893168e+62]] and testing error = [[3.
12046876e+62]]
After 10 epoch training error = [[6.76344609e+65]] and testing error =
[[7.03754686e+65]]
After 11 epoch training error = [[3.57381452e+68]] and testing error =
[[3.71864976e+68]]
After 12 epoch training error = [[4.11233942e+70]] and testing error =
[[4.27899934e+70]]
After 13 epoch training error = [[8.61779724e+71]] and testing error =
[[8.96704892e+71]]
After 14 epoch training error = [[1.81931064e+72]] and testing error =
[[1.89304147e+72]]
After 15 epoch training error = [[1.78269394e+71]] and testing error =
[[1.85494081e+71]]
After 16 epoch training error = [[1.32720884e+71]] and testing error =
[[1.38099635e+71]]
After 17 epoch training error = [[3.82078729e+71]] and testing error =
[[3.97563154e+71]]
After 18 epoch training error = [[3.92793455e+71]] and testing error =
[[4.08712114e+71]]
After 19 epoch training error = [[3.26111583e+71]] and testing error =
[[3.39327839e+71]]
After 20 epoch training error = [[2.66156007e+71]] and testing error =
[[2.76942456e+71]]
After 21 epoch training error = [[2.26558497e+71]] and testing error =
[[2.35740186e+71]]
After 22 epoch training error = [[2.02794708e+71]] and testing error =
[[2.11013327e+71]]
After 23 epoch training error = [[1.89057005e+71]] and testing error =
[[1.96718879e+71]]
After 24 epoch training error = [[1.8125997e+71]] and testing error = [[1.
88605855e+71]]
After 25 epoch training error = [[1.76887151e+71]] and testing error =
[[1.8405582e+71]]
After 26 epoch training error = [[1.74458004e+71]] and testing error =
[[1.81528227e+71]]
After 27 epoch training error = [[1.73119779e+71]] and testing error =
[[1.80135769e+71]]
After 28 epoch training error = [[1.72388018e+71]] and testing error =
[[1.79374351e+71]]
After 29 epoch training error = [[1.71990534e+71]] and testing error =
[[1.78960759e+71]]
After 30 epoch training error = [[1.71775902e+71]] and testing error =
[[1.78737428e+71]]
After 31 epoch training error = [[1.71660612e+71]] and testing error =
```

```
[[1.78617466e+71]]
After 32 epoch training error = [[1.71598972e+71]] and testing error =
[[1.78553328e+71]]
After 33 epoch training error = [[1.71566151e+71]] and testing error =
[[1.78519177e+71]]
After 34 epoch training error = [[1.7154874e+71]] and testing error = [[1.
78501061e+71]]
After 35 epoch training error = [[1.71539534e+71]] and testing error =
[[1.78491481e+71]]
After 36 epoch training error = [[1.71534681e+71]] and testing error =
[[1.78486431e+71]]
After 37 epoch training error = [[1.71532129e+71]] and testing error =
[[1.78483776e+71]]
After 38 epoch training error = [[1.7153079e+71]] and testing error = [[1.
78482383e+71]]
After 39 epoch training error = [[1.7153009e+71]] and testing error = [[1.
78481654e+71]]
After 40 epoch training error = [[1.71529724e+71]] and testing error =
[[1.78481273e+71]]
After 41 epoch training error = [[1.71529533e+71]] and testing error =
[[1.78481075e+71]]
After 42 epoch training error = [[1.71529434e+71]] and testing error =
[[1.78480972e+71]]
After 43 epoch training error = [[1.71529382e+71]] and testing error =
[[1.78480918e+71]]
After 44 epoch training error = [[1.71529355e+71]] and testing error =
[[1.7848089e+71]]
After 45 epoch training error = [[1.71529341e+71]] and testing error =
[[1.78480876e+71]]
After 46 epoch training error = [[1.71529334e+71]] and testing error =
[[1.78480868e+71]]
After 47 epoch training error = [[1.71529331e+71]] and testing error =
[[1.78480864e+71]]
After 48 epoch training error = [[1.71529329e+71]] and testing error =
[[1.78480862e+71]]
After 49 epoch training error = [[1.71529328e+71]] and testing error =
[[1.78480861e+71]]
After 50 epoch training error = [[1.71529327e+71]] and testing error =
[[1.78480861e+71]]
After 51 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 52 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 53 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 54 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 55 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 56 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 57 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 58 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 59 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 60 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 61 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
```

```
After 62 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 63 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 64 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 65 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 66 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 67 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 68 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 69 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 70 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 71 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 72 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 73 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 74 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
After 75 epoch training error = [[1.71529327e+71]] and testing error =
[[1.7848086e+71]]
Slop: [[-5.55370712e+30]
 [-1.33652716e+31]
 [-1.57893484e+31]
 [-8.92043238e+28]
 [-7.44139672e+29]
 [-8.15951340e+30]
 [-9.36645859e+31]
 [-4.62517488e+30]
 [-1.42247064e+31]
 [-5.69212800e+32]
 [-2.43242574e+31]
 [-4.66976586e+32]
 [-1.74495822e+31]]
 y_intercept: [[-1.30302272e+30]]
```

In [70]:

```
plt.figure()
plt.plot(range(len(training_loss)), np.reshape(training_loss,[len(training_loss), 1]),
label = "Train Loss")
plt.plot(range(len(testing_loss)), np.reshape(testing_loss, [len(testing_loss), 1]), la
bel = "Test Loss")
plt.title("loss per epoch")
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```