LogisticRegression With SkitLearn Library

In [2]:

```
from sklearn.datasets import load_boston

import sklearn
#from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report,mean_squared_error

import numpy as np
import pandas as pd
import matplotlib.pyplot as pyplt
import seaborn as sns
```

In [3]:

```
boston = load_boston()
print(boston.data.shape)
print(boston.feature_names)
boston.data[1]
```

```
(506, 13)
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

Out[3]:

```
array([2.7310e-02, 0.0000e+00, 7.0700e+00, 0.0000e+00, 4.6900e-01,
       6.4210e+00, 7.8900e+01, 4.9671e+00, 2.0000e+00, 2.4200e+02,
       1.7800e+01, 3.9690e+02, 9.1400e+00])
```

In [4]:

```
boston_data = pd.DataFrame(boston.data)
#print(bos.head())
boston_data['PRICE'] = boston.target
boston_data.head(4)
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	PRICE
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	24.0
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	21.6
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	34.7
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	33.4

In [5]:

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

In [18]:

```
(((y_test1 - y_pred)**2).sum(axis=0))/y_pred.shape[0]
```

Out[18]:

```
28.530458765974583
```

In [17]:

```
#y_pred[:20]
y_test1=np.array(y_test)
print(y_test1[:20])
print(y_pred[:20])
np.sqrt((((y_test1 - y_pred)**2).sum(axis=0))/y_pred.shape[0])
```

```
[37.6 27.9 22.6 13.8 35.2 10.4 23.9 29.  22.8 23.2 33.2 19.  20.3 36.1
 24.4 17.2 17.9 19.6 19.7 15. ]
[37.46723562 31.39154701 27.1201962   6.46843347 33.62966737  5.67067989
 27.03946671 29.92704748 26.35661334 22.45246021 32.20504441 21.78641653
 23.41138441 33.60894362 28.28619511 15.13859055  0.30087325 18.71850376
 14.4706712  11.10823598]
```

Out[17]:

```
5.341391089030514
```

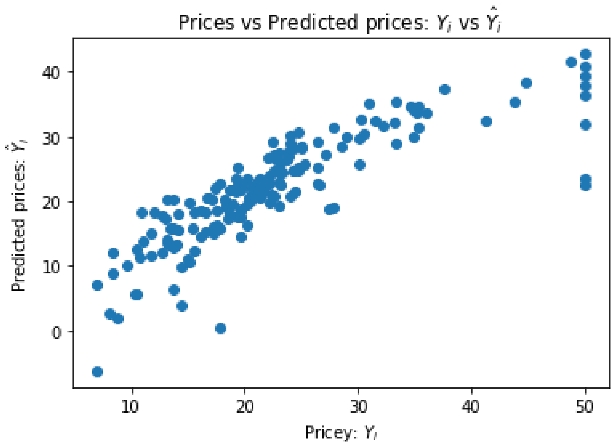
In [19]:

```
X_train, X_test, y_train, y_test = sklearn.model_selection.train_test_split(X, Y, test_size = 0.33,
random_state = 5)

clf = LinearRegression()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(y_pred.shape)
print(y_test.shape)
print("Mean Squared Error ",mean_squared_error(y_test, y_pred))
print("Root Maen Mean Squared Error ",np.sqrt((((y_test1 - y_pred)**2).sum(axis=0))/y_pred.shape[0]
))
print(clf.score(X_test, y_test))


pyplt.scatter(y_test, y_pred)
pyplt.xlabel("Pricey: $Y_i$")
pyplt.ylabel("Predicted prices: $\hat{Y}_i$")
pyplt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
pyplt.show()
```

```
(167,)
(167,)
Mean Squared Error  28.530458765974583
Root Maen Mean Squared Error  5.341391089030514
0.6956551656111607
```

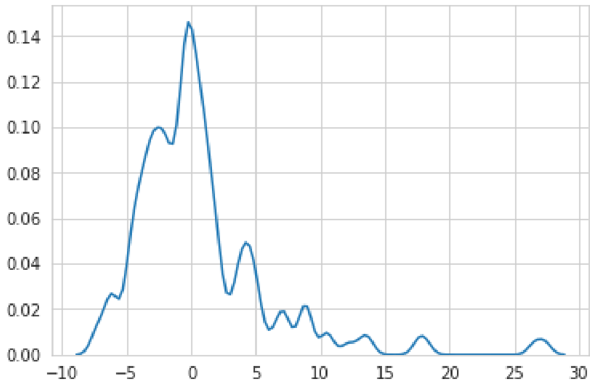


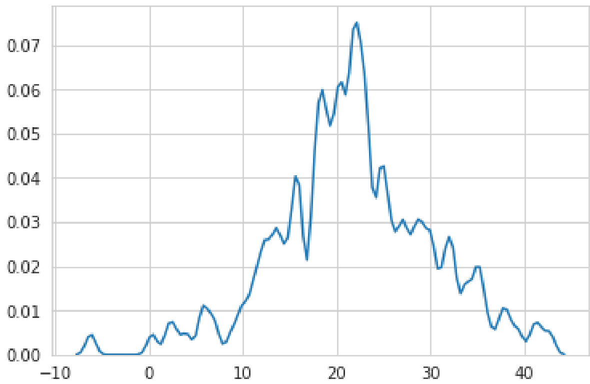
In [24]:

```
delta_y = y_test - y_pred;


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

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





Logistic Regression coding without SKITLearn Library

In [25]:

```
from sklearn.preprocessing import StandardScaler
import numpy as np
boston_data['PRICE'] = boston.target
X = boston_data.drop('PRICE', axis = 1)
Y = boston_data['PRICE']
df_sample =X
price =Y
xi_1=[]
price_1=[]
N = len(df_sample)
scaler = StandardScaler()
scaler.fit(df_sample)
xi = scaler.transform(df_sample)
#scaler.fit(price)
yi = Y
#yi=price
print(xi.shape)
print(yi.shape)
```

```
(506, 13)
(506,)
```

In [26]:

```
xi[0:3]
yi[0:3]
```

Out[26]:

```
0    24.0
1    21.6
2    34.7
Name: PRICE, dtype: float64
```

In [ ]:

```python
r= 1
m_deriv = 0
b_deriv = 0
learning_rate = 0.0001
it = 1
w0=0
b0=0
w0_grad=0
b0_grad=0
w0_random = np.random.rand(13)
w_old = np.asmatrix(w0_random).T
b = np.random.rand()
b_old = np.random.rand()

for j in range(100):
    for i in range(1):
        #print("====Entering to calculate m_deriv1 ====== ")
        w1=w_old
        b1=b_old
        m_deriv1 =np.dot((yi - (np.dot(xi , w1)+b1).T ), (-2*np.asmatrix(xi)))
        #print("=Entering to calculate b_deriv1 =")

        for i in range(506):

            a=(np.dot( w1.T ,xi[i].T) + b1)
            a1=yi[0,i]

            b_deriv1=-2*(a1-a)

    w_grad = m_deriv1 * learning_rate
    b_grad = b_deriv1 * learning_rate

    #print(w_grad)
    #print(b_grad)


    w_old=w_old.T

    #print(w_old)
    #print(w_grad)

    w_new = w_old - w_grad
    b_new = b_old - b_grad

    #print(w_old)
    #print(w_new)

    it += 1
    if (w_new==w_old).all():
        break
    else:
        w_old = w_new.T
        b_old = b_new[0,0]
        learning_rate = learning_rate
    #print(j)
#print(w_new)
#print(b_new)
```

In [463]:

```python
w_old = np.asmatrix(w0_random)
b_old = np.random.rand()
print(w_old)
print(b_old)
print(w_new)
print(b_new)
```

```
[[0.65393577 0.63335701 0.13769644 0.7763089  0.17720172 0.16220672
  0.20564501 0.16828612 0.49144455 0.02715377 0.22350059 0.7955394
  0.03948138]]
0.1725768839024161
```

```
[[-0.78383075  0.8363682  -0.22830063  0.73574746 -1.56745879  2.87960394
  -0.09404347 -2.73791613  1.56653738 -1.02091713 -1.94546819  0.87289608
  -3.65543201]]
[[0.87535196]]
```

In [466]:

```
np.dot(X_test ,( w_new.T )).shape
```

Out[466]:

```
(167, 1)
```

In [469]:

```
yi_test = np.asmatrix(y_test).T
print(X_test.shape)
print(yi_test.shape)

np.sqrt(np.mean(np.square(yi_test - np.dot(X_test ,( w_new.T )) + b_new)) )
```

```
(167, 13)
(167, 1)
```

Out[469]:

```
312.8356397387943
```