In [0]:
```python
from sklearn.datasets import load_boston
boston = load_boston()
```

In [0]:
```python
print(boston.data.shape)
```

```
(506, 13)
```

In [0]:
```python
print(boston.feature_names)
```

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

In [0]:
```python
print(boston.DESCR)
```

```
.. _boston_dataset:

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

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Va
lue (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 2
5,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds ri
ver; 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 19
40
        - 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 blac
ks 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
Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedoni
c
prices and the demand for clean air', J. Environ. Economics & Managemen
t,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagn
ostics
...', 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 pape
rs that address regression
problems.

.. topic:: References
```

```
   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influe
ntial Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learni
ng. In Proceedings on the Tenth International Conference of Machine Lea
rning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [0]:

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

```
         0     1     2    3      4  ...    8      9    10      11    12
0  0.00632  18.0  2.31  0.0  0.538  ...  1.0  296.0  15.3  396.90  4.98
1  0.02731   0.0  7.07  0.0  0.469  ...  2.0  242.0  17.8  396.90  9.14
2  0.02729   0.0  7.07  0.0  0.469  ...  2.0  242.0  17.8  392.83  4.03
3  0.03237   0.0  2.18  0.0  0.458  ...  3.0  222.0  18.7  394.63  2.94
4  0.06905   0.0  2.18  0.0  0.458  ...  3.0  222.0  18.7  396.90  5.33

[5 rows x 13 columns]
```

In [0]:

```
bos['PRICE'] = boston.target
print('printing dataset with target value')
print(bos.head())
X = bos.drop('PRICE', axis = 1)
Y = bos['PRICE']
```

```
printing dataset with target value
          0     1     2    3      4  ...      9    10      11    12  PRI
CE
0  0.00632  18.0  2.31  0.0  0.538  ...  296.0  15.3  396.90  4.98   2
4.0
1  0.02731   0.0  7.07  0.0  0.469  ...  242.0  17.8  396.90  9.14   2
1.6
2  0.02729   0.0  7.07  0.0  0.469  ...  242.0  17.8  392.83  4.03   3
4.7
3  0.03237   0.0  2.18  0.0  0.458  ...  222.0  18.7  394.63  2.94   3
3.4
4  0.06905   0.0  2.18  0.0  0.458  ...  222.0  18.7  396.90  5.33   3
6.2
```

```
[5 rows x 14 columns]
```

In [0]:
```
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, random_state = 5)
print(X_train.shape)
print(X_test.shape)
Y_train=Y_train.values.reshape(Y_train.shape[0],1)
Y_test=Y_test.values.reshape(Y_test.shape[0],1)
print(Y_train.shape)
print(Y_test.shape)
```

```
(339, 13)
(167, 13)
(339, 1)
(167, 1)
```

In [0]:
```
#standardization of data
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

implement own version of sgd regressor

In [0]:
```
#initiating weight value
#initiate bias vaue
import numpy as np
w=np.random.normal(size=(13,1))
b=np.random.normal()
```

In [0]:
```
#my own implementaion of sgd regressor
#weight update term(finding optimal weight)
r=0.01
for i in range(0,200):
```

```
  dw=-2*np.mean(X_train*(Y_train-(np.dot(X_train,w)+b0)),axis=0).reshap
e(13,1)
  db=-2*np.mean(Y_train-(np.dot(X_train,w)+b0))
  w=w-r*dw
  b0 = b0- r*db
```

In [0]:

```
w_optimal=w.T
b_optimal=b0
print(w_optimal)
print(b_optimal)
```

```
[[-0.8912016   0.33759072 -0.45046978  0.35727199 -0.99984152  3.001510
59
  -0.12470173 -1.7586858   0.64112216 -0.16400223 -2.11102017  0.842437
06
  -3.36982702]]
[[22.53716811]]
```

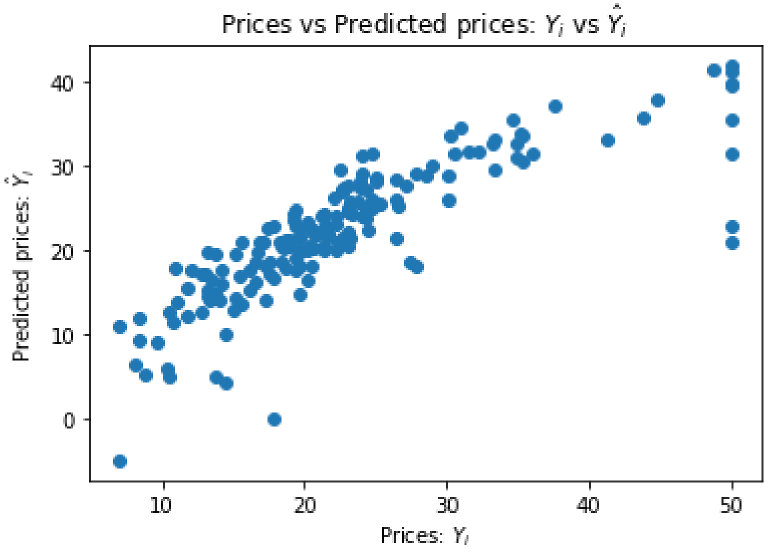
In [0]:

```
print(w_optimal.shape)
```

```
(1, 13)
```

In [0]:

```
#Plotting a chart of predicted values Vs actual values of my own SGD Im
plementation:
import matplotlib.pyplot as plt
Y_pred=np.dot(w_optimal,X_test.T)+b0
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 [0]:
```
#MSE of my own sgd implementation
#code for getting mean square error value
from sklearn.metrics import mean_squared_error
mse_own=mean_squared_error(Y_test, Y_pred.T)
print(mse_own)
```

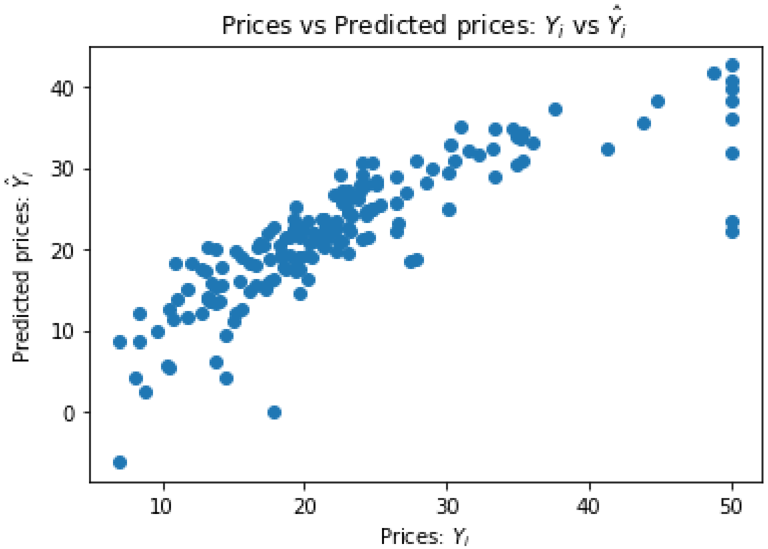
```
28.96528689410586
```

Sklearn version of sgd

In [0]:
```
#sklearn version of sgd regressor
import numpy as np
from sklearn import linear_model
clf = linear_model.SGDRegressor(max_iter=1000, tol=1e-3)
clf.fit(X_train, Y_train)
w_sgd_regressor=clf.coef_
Y_pred = clf.predict(X_test)
```

```
#Plotting a chart of predicted values Vs actual values of sklearn versi
on of sgd
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()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:724:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y = column_or_1d(y, warn=True)
```



In [0]:

```
#sklears sgd regressor
#printing weight values of sklearn implemented:
print(w_sgd_regressor)
```

```
[-1.24689462  0.76104495 -0.40300212  0.23879233 -1.3497314   2.8714581
7
 -0.37140269 -2.72914389  2.11845631 -1.37606423 -2.10834319  1.0422068
```

```
4
 -3.33245567]
```

In [0]:
```
#MSE of my own sgd implementation
#code for getting mean square error value
from sklearn.metrics import mean_squared_error
mse_sklearn=mean_squared_error(Y_test, Y_pred)
print(mse_sklearn)
```

```
28.476258862964894
```

In [0]:
```
w_sgd_regressor=w_sgd_regressor.reshape(1,13)
```

In [0]:
```
#comparing weight values of my own implementation of sgd and skleanrs s
gd:
print('\033[1m' + "w of own impl\t\tw of sklearns impl" + '\033[0m')
for i in range(0,12):
  print("%.4f\t\t\t%-15s" % (w_optimal[0][i], w_sgd_regressor[0][i]))
```

```
w of own impl           w of sklearns impl
-0.8912                 -1.246894623889562
0.3376                  0.761044952409226
-0.4505                 -0.4030021196148545
0.3573                  0.238792334307354
-0.9998                 -1.3497314043494244
3.0015                  2.871458167599385
-0.1247                 -0.3714026875031446
-1.7587                 -2.729143893787323
0.6411                  2.1184563076670777
-0.1640                 -1.3760642321421557
-2.1110                 -2.108343193059245
0.8424                  1.042206844734608
```

In [0]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["model","MSE"]
x.add_row(["own imple",29.67])
```

```
x.add_row(["sklearn impl",28.68])
print(x)
```

```
+--------------+-------+
|    model     |  MSE  |
+--------------+-------+
|  own imple   | 29.67 |
| sklearn impl | 28.68 |
+--------------+-------+
```