In [607]:
```
import warnings
warnings.filterwarnings("ignore")
from sklearn.datasets import load_boston
from random import seed
from random import randrange
from csv import reader
from math import sqrt
from sklearn import preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
from sklearn.linear_model import SGDRegressor
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
```

1. Boston Dataset Exploration

In [608]:
```
df =load_boston()
print(df.feature_names)
print(df.DESCR)
```

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

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

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attr
ibute 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 sq.
ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 ot
herwise)
        - 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 tow
n
        - 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. '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 a
ddress regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Dat
a and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Pr
```

oceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.

In [609]:

```
#data = pd.DataFrame(bostan.data, columns = bostan.feature_names)

X = load_boston().data
Y = load_boston().target

X= pd.DataFrame(X, columns = df.feature_names)
Y = pd.DataFrame(Y, columns = ['PRICE'])
```

In [610]:
```
print(X.shape)
print(X.head(10))
Y.head(10)
```

```
(506, 13)
      CRIM    ZN  INDUS  CHAS    NOX     RM    AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575   65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421   78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185   61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998   45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147   54.2  6.0622  3.0  222.0
5  0.02985   0.0   2.18   0.0  0.458  6.430   58.7  6.0622  3.0  222.0
6  0.08829  12.5   7.87   0.0  0.524  6.012   66.6  5.5605  5.0  311.0
7  0.14455  12.5   7.87   0.0  0.524  6.172   96.1  5.9505  5.0  311.0
8  0.21124  12.5   7.87   0.0  0.524  5.631  100.0  6.0821  5.0  311.0
9  0.17004  12.5   7.87   0.0  0.524  6.004   85.9  6.5921  5.0  311.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
5     18.7  394.12   5.21
6     15.2  395.60  12.43
7     15.2  396.90  19.15
8     15.2  386.63  29.93
9     15.2  386.71  17.10
```

Out[610]:

	PRICE
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2
5	28.7
6	22.9
7	27.1
8	16.5
9	18.9

1.1 Split Training and test data

In [611]:
```
# Split data into training and test dataset
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, rando
m_state = 10)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

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

In [629]:
```
print(Y_train['PRICE'].sample(10))
print(np.asmatrix(Y_train['PRICE']))
print(type(X_train))
print(type(Y_train))
```

```
389    11.5
405     5.0
415     7.2
407    27.9
368    50.0
501    22.4
471    19.6
393    13.8
109    19.4
238    23.7
Name: PRICE, dtype: float64
[[14.6 19.8 10.2 23.7 18.7 20.3 50.  25.  14.8  9.5 33.  19.4 15.6 20.2
  21.7 24.7 19.5  7.2 31.7 22.4 14.1 21.4 21.7 23.  20.3 30.7 35.1 20.9
  27.  10.2 22.4 23.3 33.1 28.7 31.6 22.2 17.4 16.1 23.  35.4 22.9 34.9
  19.5 21.  24.8 30.5 36.5 21.5 17.5 19.4 34.7 19.6 25.  13.9 13.3 18.5
  12.3 20.  22.  13.6 14.9 32.  22.5 23.9 50.  13.8 20.1 20.8 18.8 50.
  22.2 26.2 20.1 13.8 19.8 17.8 24.1 19.4 22.3 20.  13.3 21.  29.1 22.7
  18.2 21.7 19.6 24.5 14.1 19.9 22.  20.1 15.  23.  42.3  6.3 34.9 34.6
  20.5 28.7 17.8 10.4 13.9 11.7 50.   7.4 19.6 22.6 16.2 19.4 20.6 27.5
   5.6  7.  24.8 23.2  7.5  5.  19.2 21.1 20.3 24.  16.2 35.4 23.1 13.5
  15.1 22.9 36.1 22.6 10.2 12.8 50.  22.5 20.  19.  50.  22.5 19.7  8.3
  21.4 17.2 36.  17.1 13.8 16.4 33.3 39.8 18.9 30.1 13.2 17.2 31.  11.9
  19.3 30.8 18.4 25.  11.7 22.3 19.1 21.1 24.4 19.4 17.5 12.7 20.9 19.3
  28.1 17.4 13.4 13.4 23.8 20.1 31.5 44.8 19.1 12.7 50.  18.3  8.5 14.4
  13.4 17.2  8.3 24.8 28.4 22.  20.4 21.4 15.2 21.9 17.6 23.2 21.7 23.3
  11.5 11.8 20.  17.7  5.  22.6 26.6 48.3 18.5 13.3 19.  21.9 23.4 29.9
  35.2 19.1  7.2 20.5 18.2 13.  20.6 18.7 24.4 22.  33.4 14.5 18.  25.
  24.1 19.3 15.2 30.1 24.  14.1 31.5 25.3 41.7  8.4 20.3 29.6 14.6 26.7
  43.5 18.8 18.9 21.5 29.  26.6 24.3 15.6 15.6 14.2 25.2 25.  28.5 37.9
  21.9 33.4 20.8  9.6  8.7 36.2 18.4 21.2 18.6 24.4 24.8 32.2 48.5 18.1
  14.5 13.8 11.8 23.7 21.7 20.4 16.1 20.6 13.8 23.3 24.1 21.2 21.8 24.5
  20.6 27.9 11.9 19.9 25.  20.2 50.  27.9 21.6 14.5 23.1 14.  28.7 24.3
  20.1 16.5 24.5 14.4 11.3 27.5 23.7 20.4 17.8 20.8 25.  37.2 24.6 19.5
  21.2 32.9 13.1 22.2 20.5 16.8 31.2 18.9 13.1 23.3 17.8 23.1 50.  16.7
  34.9  8.1 44.  23.4 16.5 16.8 16.3 21.7 29.8 13.1 17.3 50.  23.8 19.9
  20.6 21.4 22.8]]
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

1.2 Preprocess Numerical Value

In [613]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train = pd.DataFrame(X_train)
```

In [614]:
```
print(X_train.shape)

print(X_train.shape)
print(X_test.shape)
```

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

2. SGD in Python

In [674]:
```
def SGD_own (w0, b0, x, y, learning_rate,n_iter):
    grad_m = 0
    grad_b = 0
    for j in range(1, n_iter):
        #x= x.sample(160)
        #y= y.sample(160)
        y = np.asmatrix(y)
        x = np.asmatrix(x)
        for i in range(len(x)):
            #grad_m += -2*x[i].T , (y[:,i] - np.dot(x[i] , w0) - b0)
            grad_m += -2*x[i].T * (y[:,i] - np.dot(x[i] , w0) - b0)
            grad_b += -2*(y[:,i] - (np.dot(x[i] , w0) + b0))

        w1 = w0 - learning_rate * grad_m
        b1 = b0 - learning_rate * grad_b

        if (w0==w1).all():

            break
        else:
            w0 = w1
            b0 = b1
            learning_rate = learning_rate/2

    return w0, b0
```

In [675]:
```
print(type(X_train))
print(type(Y_train))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

In [723]:
```
w0_random = np.random.rand(13)
w0 = np.asmatrix(w0_random).T
b0 = np.random.rand()


optimal_w , optimal_b= SGD_own(w0, b0, X_train, Y_train['PRICE'],learning_rate =
0.000485, n_iter = 10000)
print(optimal_w,optimal_b)
```

```
[[-0.4054812 ]
 [ 0.30596823]
 [-0.14248085]
 [ 1.06061206]
 [ 0.27256523]
 [ 3.44591036]
 [ 0.51097156]
 [-0.77915757]
 [ 0.37569295]
 [-0.14418825]
 [-1.59839591]
 [ 0.97176776]
 [-2.54343297]] [[21.42210946]]
```

In [724]:
```
y_pred_own = np.dot(X_test,optimal_w)+optimal_b
print(y_pred_own.shape)
print(Y_test.shape)
print(mean_squared_error(Y_test, y_pred_own))
```

```
(167, 1)
(167, 1)
29.75498247935324
```

In [ ]:

In [ ]:

3. Compare with Python SGD

3.1 Sklearn results

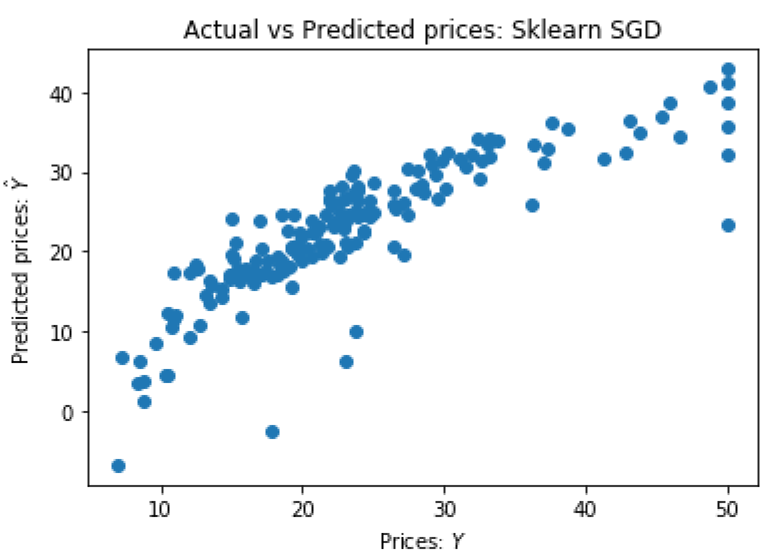
In [725]:
```
clf = SGDRegressor()
clf.fit(X_train, Y_train)

Y_predict = clf.predict(X_test)

print(clf)
print("Coefficients: \n", clf.coef_)
print("Y_intercept", clf.intercept_)
```

```
SGDRegressor(alpha=0.0001, average=False, early_stopping=False, epsilon=0.1,
       eta0=0.01, fit_intercept=True, l1_ratio=0.15,
       learning_rate='invscaling', loss='squared_loss', max_iter=None,
       n_iter=None, n_iter_no_change=5, penalty='l2', power_t=0.25,
       random_state=None, shuffle=True, tol=None, validation_fraction=0.1,
       verbose=0, warm_start=False)
Coefficients:
 [-0.8328814   0.86104532 -0.4366005   0.5926068  -0.48103866  2.88662273
 -0.13853722 -1.84424267  0.61214727 -0.56195917 -1.62445177  1.27991093
 -3.46516903]
Y_intercept [21.46480823]
```

In [726]:
```
print(mean_squared_error(Y_test, Y_predict))
```

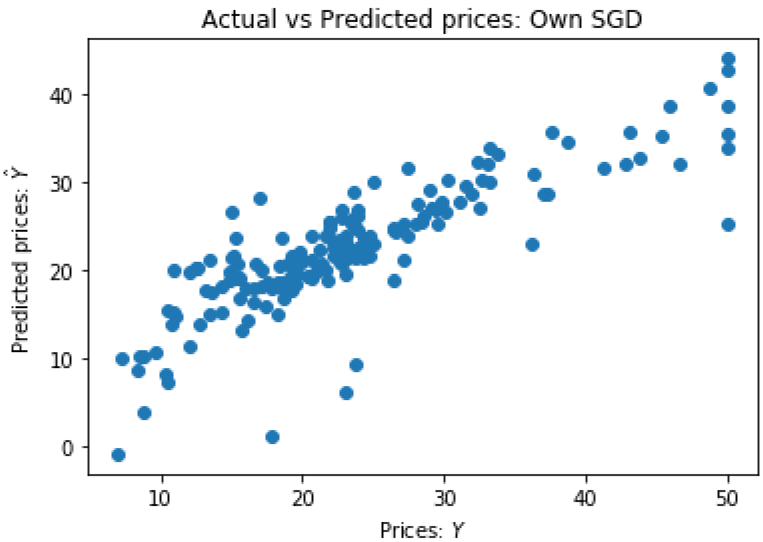
```
27.993673103581344
```

In [727]:
```
#sklearn SGD actual vs predict
plt.figure(1)
plt.scatter(Y_test, Y_predict)
plt.xlabel("Prices: $Y $")
plt.ylabel("Predicted prices: $\hat{Y} $")
plt.title("Actual vs Predicted prices: Sklearn SGD")
plt.show()
```



3.2 Own SGD function results

In [728]:
```
# Own SGD actual vs predict
plt.scatter(Y_test,pd.DataFrame(y_pred_own))
plt.xlabel("Prices: $Y $")
plt.ylabel("Predicted prices: $\hat{Y} $")
plt.title("Actual vs Predicted prices: Own SGD")
plt.show()
```



In [729]:
```
result = pd.concat([pd.DataFrame(clf.coef_), pd.DataFrame(optimal_w)], axis=1, ig
nore_index=True)

result = result.rename(index=str, columns={0:"Sklearn_weights",1:"Own_weights"})

result
```

Out[729]:

	Sklearn_weights	Own_weights
0	-0.832881	-0.405481
1	0.861045	0.305968
2	-0.436600	-0.142481
3	0.592607	1.060612
4	-0.481039	0.272565
5	2.886623	3.445910
6	-0.138537	0.510972
7	-1.844243	-0.779158
8	0.612147	0.375693
9	-0.561959	-0.144188
10	-1.624452	-1.598396
11	1.279911	0.971768
12	-3.465169	-2.543433

In [730]:

```
print("-"*100)

print("The MSE Sklearn SGD is\n", mean_squared_error(Y_test, Y_predict))
print("The MSE own SGD is\n", mean_squared_error(Y_test, pd.DataFrame(y_pred_own
)))
```

```
--------------------------------------------------------------------------------
--------------------
The MSE Sklearn SGD is
 27.993673103581344
The MSE own SGD is
 29.75498247935324
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: