manual implementation of Sgd

In [80]:
```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_boston
boston=load_boston()
```

In [81]:
```
data=boston.data
print(data.shape)
target=boston.target
print(target.shape)
```

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

In [82]:
```
data=pd.DataFrame(data,columns=boston.feature_names)
data.head()
```

Out[82]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
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

In [83]:
```
data.describe()
```

Out[83]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	A(
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0000
mean	3.593761	11.363636	11.136779	0.069170	0.554695	6.284634	68.57490
std	8.596783	23.322453	6.860353	0.253994	0.115878	0.702617	28.14886
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.02500
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.50000
75%	3.647423	12.500000	18.100000	0.000000	0.624000	6.623500	94.07500
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.0000

In [84]:
```
data = (data - data.mean())/data.std()
data.head()
```

Out[84]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
0	-0.417300	0.284548	-1.286636	-0.272329	-0.144075	0.413263	-0.119895	0.140075	-0
1	-0.414859	-0.487240	-0.592794	-0.272329	-0.739530	0.194082	0.366803	0.556609	-0
2	-0.414861	-0.487240	-0.592794	-0.272329	-0.739530	1.281446	-0.265549	0.556609	-0
3	-0.414270	-0.487240	-1.305586	-0.272329	-0.834458	1.015298	-0.809088	1.076671	-0
4	-0.410003	-0.487240	-1.305586	-0.272329	-0.834458	1.227362	-0.510674	1.076671	-0

In [85]:
```
data["VALUE"] =target
data.head()
```

Out[85]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
0	-0.417300	0.284548	-1.286636	-0.272329	-0.144075	0.413263	-0.119895	0.140075	-0
1	-0.414859	-0.487240	-0.592794	-0.272329	-0.739530	0.194082	0.366803	0.556609	-0
2	-0.414861	-0.487240	-0.592794	-0.272329	-0.739530	1.281446	-0.265549	0.556609	-0
3	-0.414270	-0.487240	-1.305586	-0.272329	-0.834458	1.015298	-0.809088	1.076671	-0
4	-0.410003	-0.487240	-1.305586	-0.272329	-0.834458	1.227362	-0.510674	1.076671	-0

In [86]:

```
#we are taking that whole dataframe with x and y becoz of sampling the
 correct data
Y = data["VALUE"]
X = data.drop("VALUE", axis = 1)
```

In [87]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size =
0.3)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

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

In [88]:

```
x_train["VALUE"] = y_train
```

In [89]:

```
#https://www.kaggle.com/premvardhan/stocasticgradientdescent-implementa
tion-lr-python
def error_function(b, m, features, target):
    sumofError = 0
    for i in range(0, len(features)):
        x = features
        y = target
        sumofError += (y[:,i] - (np.dot(x[i] , m) + b)) ** 2
    return sumofError / len(x)
```

In [90]:

```
#https://www.kaggle.com/premvardhan/stocasticgradientdescent-implementa
```

```
tion-lr-python
def manual_gradient_decent(w0, b0, train_data, x_test, y_test, learning
_rate):
    n_iter = 700
    partial_deriv_m = 0
    partial_deriv_b = 0
    cost_train = []
    cost_test = []
    for j in range(1, n_iter):

        # Train sample we are taking the sample dataframe to get x and
 y as same sorted values
        x_train_sample = train_data.sample(160)
        y = np.asmatrix(x_train_sample["VALUE"])
        x = np.asmatrix(x_train_sample.drop("VALUE", axis = 1))

        for i in range(len(x)):
            partial_deriv_m += np.dot(-2*x[i].T , (y[:,i] - np.dot(x[i]
 , w0) + b0))
            partial_deriv_b += -2*(y[:,i] - (np.dot(x[i] , w0) + b0))


        w1 = w0 - learning_rate * partial_deriv_m
        b1 = b0 - learning_rate * partial_deriv_b

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


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


        error_train = error_function(b0, w0, x, y)
        cost_train.append(error_train)
        error_test = error_function(b0, w0, np.asmatrix(x_test), np.asm
atrix(y_test))
```

```
        cost_test.append(error_test)


    return w0, b0, cost_train, cost_test
```

In [91]:

```
#as per in
learning_rate = 0.001
w0_random = np.random.normal(0,1,13)
w0 = np.asmatrix(w0_random).T
b0 = np.random.rand()

optimal_w, optimal_b, cost_train, cost_test = manual_gradient_decent(w0
, b0, x_train, x_test, y_test, learning_rate)
print("Coefficient: {} \n y_intercept: {}".format(optimal_w, optimal_b
))
```

```
Coefficient: [[-1.02032106]
 [ 0.49019189]
 [-0.70222783]
 [ 1.85169007]
 [-1.35385686]
 [ 3.83131196]
 [ 0.86425597]
 [-3.13822588]
 [ 1.46911404]
 [ 0.48620907]
 [-2.40092123]
 [ 0.2023733 ]
 [-4.87224064]]
 y_intercept: [[21.73169692]]
```

SKLEARN SGD

In [92]:

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
```

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

In [93]:
```
data=boston.data
target=boston.target
```

In [94]:
```
#defining the variable
X=data
Y=target
```

In [95]:
```
#spliting the data into train and test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
0.33,random_state=10)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

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

In [96]:
```
#standardising the data
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

In [97]:
```
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error, r2_score
sgd = SGDRegressor()
sgd.fit(X_train, Y_train)
Y_pred = sgd.predict(X_test)
#co_efficient
```

```
w=sgd.coef_
w
```

Out[97]:
```
array([-0.929211  ,  0.79455267, -0.25964558,  0.43805506, -0.55796146,
        2.94442338,  0.04536379, -1.60041586,  0.83639864, -0.66800707,
       -1.60734169,  1.23982162, -3.23711953])
```

In [98]:
```
#intercepts
y_inter=sgd.intercept_
y_inter
```

Out[98]:
```
array([21.42210671])
```

In [99]:
```
print("Mean squared error: %.2f" % mean_squared_error(Y_test, Y_pred))
```

```
Mean squared error: 28.30
```

In [100]:
```
error = error_function(optimal_b, optimal_w, np.asmatrix(x_test), np.as
matrix(y_test))
print("Mean squared error: %.2f" % (error))
```

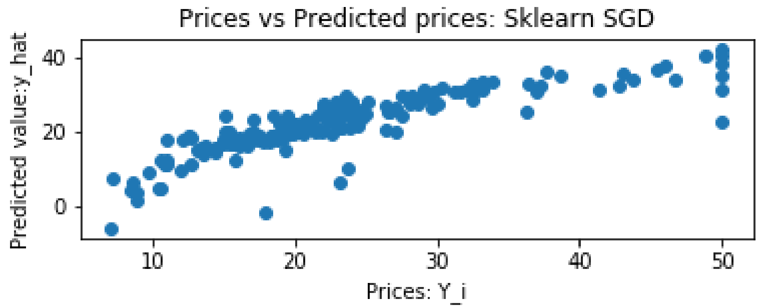
```
Mean squared error: 30.81
```

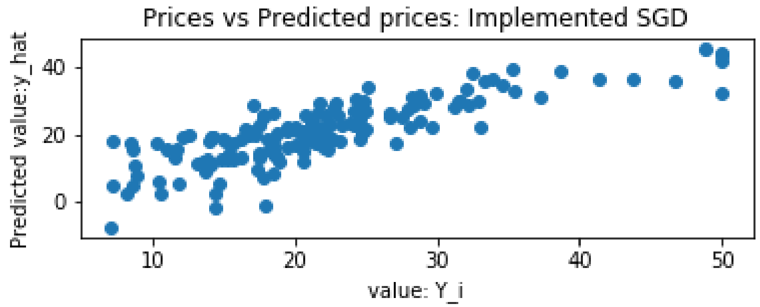
MANUAL V/S SKLEARN

In [101]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.figure(1)
plt.subplot(211)
plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices: Y_i")
plt.ylabel("Predicted value:y_hat")
plt.title("Prices vs Predicted prices: Sklearn SGD")
plt.show()

# Implemented SGD
plt.subplot(212)
```

```
plt.scatter([y_test], [(np.dot(np.asmatrix(x_test), optimal_w) + optima
l_b)])
plt.xlabel("value: Y_i")
plt.ylabel("Predicted value:y_hat")
plt.title("Prices vs Predicted prices: Implemented SGD")
plt.show()
```





EXPLANATION

1. manual

1. first we loaded the boston data and boston target.
2. manually standardised the data.
3. we created the data frame like data and target is in one frame only.

4. we defined variable x and y from dataframe only and we splitted it into train and test data.
5. why we created dataframe with x and y variable because we wont use full data to get optimized so we randomly uses that x and y . so if we didn't use dataframe with x and y .if we choose randomly x and y means dependent and independent variable changes orderly.
6. so we taken the random choice as n=160, we converge w* until it will get iterated until wj=wk if we get wj=wk means it will get function is break . if it didn"t converge means it get iterated with learning rate =learning rate/2 for each iteration.
7. we taken randomly because to reduce cost of time .
8. we initially we randomly take weights and b and with learning rate =0.001 and no.iteration =700.

2. sklearn

1. we load the boston data and boston target.
2. we splitting the data into train and test randomly.
3. we standardised the data into fit_transform for train and transform for test.
4. we fitted x_train into sgd algorithm.
5. we compared both mse values for manual and sklearn