In [1]:
```
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
```

In [2]:
```
X = load_boston().data
Y = load_boston().target
df=pd.DataFrame(X)
#some intuition
df[13]=df[10]//df[12]  #here we set a column 13 such that df[13]=Boston_data['Medv']//Boston_data['B']
X=df.as_matrix()
df.head()
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
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	3.0
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	1.0
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	4.0
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	6.0
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	5.33	3.0

In [3]:
```
#Splitting whole data into train and test
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,
random_state=4)

# applying column standardization on train and test data

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

df_train=pd.DataFrame(X_train)
df_train['price']=y_train
df_train.head()
```

Out[3]:

	0	1	2	3	4	5	6	7	
0	-0.425469	-0.470768	-0.954686	-0.231455	-0.919581	0.215100	-0.747410	0.454022	-0.76446
1	-0.426323	2.992576	-1.330157	-0.231455	-1.227311	-0.883652	-1.691588	3.163428	-0.65156
2	-0.385190	-0.470768	-0.705828	4.320494	-0.423795	-0.125423	0.818985	-0.353904	-0.19996
3	-0.249268	-0.470768	-0.423497	-0.231455	-0.158805	-0.228336	1.021567	-0.021755	-0.65156
4	-0.365945	0.395068	-1.030363	-0.231455	0.157472	3.102729	-0.060078	-0.646202	-0.53866

In [4]:
```
#SGD implementation for linear regression
#function having parameter X_train,y_train,no of iteration,learning rat
e r
#intialising no of iteration=100,learning rate =0.01
#batch size=20



W,B,iteration,lr_rate,k=np.zeros(shape=(1,14)),0,750,0.01,25 #intialise
 W and B to zero

while iteration>=0:
```

```
    w,b,temp_vectors,temp_intercept=W,B,np.zeros(shape=(1,14)),0
    data=df_train.sample(25) #sampling random k=batch size=20 data
    x=np.array(data.drop('price',axis=1))
    y=np.array(data['price'])

    for i in range(k):
        temp_vectors+=(-2)*x[i]*(y[i]-(np.dot(w,x[i])+b))#partial diffe
rentiation wrt w dl/dw=1/k(-2x)*(y-wTx-b)
        temp_intercept+=(-2)*(y[i]-(np.dot(w,x[i])+b))#partial differen
tiation wrt b dl/db=1/k(-2)*(y-wTx-b)

    W=(w-lr_rate*(temp_vectors)/k)
    B=(b-lr_rate*(temp_intercept)/k)

    iteration-=1


print(W)
print(B)
```

```
[[-1.27290222  0.53695967 -0.66141371  0.92096371 -1.4356316   1.636325
89
   0.26658427 -2.81929856  2.14914024 -0.92267911 -2.14815104  0.999576
79
  -1.97622219  2.66317572]]
[22.27101495]
```

In [5]:
```
#prediction on x_test
#https://www.geeksforgeeks.org/numpy-asscalar-in-python/
y_predic_lr=[]
for i in range(len(X_test)):
    val=np.dot(W,X_test[i])+B  #val= wTx+b
    y_predic_lr.append(np.asscalar(val))
```

In [6]:
```
#Scatter plot of actual price vs predicted price

plt.scatter(y_test,y_predic_lr)
plt.xlabel('Actual price')
plt.ylabel('Predictd price')
```

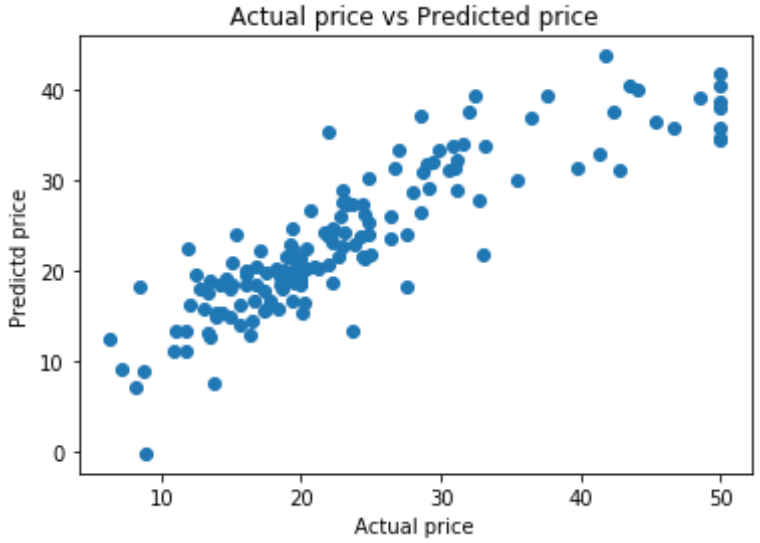
```
plt.title('Actual price vs Predicted price')
plt.show()
```

In [7]:
```
MSE_lr=mean_squared_error(y_test,y_predic_lr)
print('mean squared error =',MSE_lr)
```

```
mean squared error = 24.258404283487806
```

In [10]:
```
#SGD regression sklearn implementation

#intialising no of iteration=100,eta0=1
#taking t=2 and power_t=1 such that for each iteration eta0=eta0/pow(2,
1) ,it means half each times

model=SGDRegressor(learning_rate='constant',eta0=0.01,penalty=None,max_
iter=100)
model.fit(X_train,y_train)
y_pred_sgd=model.predict(X_test)

#Scatter plot of actual price vs predicted price

plt.scatter(y_test,y_pred_sgd)
```

```
plt.xlabel('Actual price')
plt.ylabel('Predictd price')
plt.title('Actual price vs Predicted price')
plt.show()
```



In [11]:
```
MSE_sgd=mean_squared_error(y_test,y_pred_sgd)
print('mean squared error =',MSE_sgd)
```

```
mean squared error = 24.834732966390735
```

In [12]:
```
#Comparison between weights obtained from own implementation and weight
s obtained from sgd implementation
from prettytable import PrettyTable
x = PrettyTable()
x.field_names=['Weight vector manual','Weight vector SGD sklearn']
weight_sgd=model.coef_
for i in range(13):
    x.add_row([W[0][i],weight_sgd[i]])
print(x)
```

```
+----------------------+--------------------------+
| Weight vector manual | Weight vector SGD sklearn |
```

```
+----------------------+---------------------------+
| -1.2729022153064808  |    -1.7383713107960062    |
|  0.5369596698100564  |     0.9455387554214926    |
| -0.6614137122022188  |    -0.41331803536355016   |
|  0.9209637069264854  |     0.9017778913579981    |
| -1.4356315991605868  |    -1.6015520405581498    |
|  1.6363258899143136  |     1.048928088760244     |
|  0.2665842712625248  |    0.08704812372570098    |
|  -2.819298559451317  |    -3.0231576999056533    |
|  2.1491402371476007  |      2.61590860443553     |
| -0.9226791124626965  |    -1.6275359256507602    |
|  -2.148151040199819  |    -2.1822133327621853    |
|  0.9995767929920171  |     0.7908475151392433    |
| -1.9762221899180858  |    -1.8902500986675943    |
+----------------------+---------------------------+
```

In [13]:
```
#comparison between MSE of own implementation and SGD sklearn implement
ation
print('MSE of manual implementation = ',MSE_lr)
print('-'*50)
print('MSE of SGD sklearn implementation = ',MSE_sgd)
```

```
MSE of manual implementation =  24.258404283487806
--------------------------------------------------
MSE of SGD sklearn implementation =  24.834732966390735
```

In [ ]:
```

```