Implement SGD for linear regression

Objective:

Manually implement linear regression using SGD algorithm for boston house price dataset and compare the results with sklearn SGDRegressor implementation.

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 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
```

In [3]:
```
scaler = preprocessing.StandardScaler().fit(X)
X = scaler.transform(X)
```

In [4]:
```
# Define a function to plot the results
def scatter_plot(y_true, y_pred):
    plt.grid(b=True)
    plt.scatter(y_true, y_pred)
    plt.title("Actual prices Vs. Predicted prices")
    plt.xlabel("Actual prices")
    plt.ylabel("Predicted prices")
    plt.show()
```

In [5]:
```
clf = SGDRegressor()
clf.fit(X, Y)
print(mean_squared_error(Y, clf.predict(X)))
```

```
21.99415046022759
```

Implementing our own SGD regressor algorithm

To implement SGD algorithm we will define a function that we can repeatedly use.

The implementation details for SGD are given below:

1. First initialize the weight and intercept term with initial random values.
2. Decide for the number of iterations for which the SGD algorithm will run to find the minima
3. Repeatedly divide the dataset into a fixed number of points to create a batch for every iteration
4. Compute the partial derivatives for each batch
5. Update the weight and intercept term at the end of each iteration
6. Update the learning rate at the very end of the iteration

So first I will initialize the weight (w) and intercept term (b).

In [6]:
```
# Create a dataframe for the boston dataset for ease of use
boston = pd.DataFrame(data=X)
boston['Price'] = Y
data = boston.drop('Price', axis=1)
target = boston['Price']
```

In [7]:
```
def sgd_implementation(X, learning_rate=0.01, n_iterations=100):
    '''
        In this SGD implementation, X is a dataframe
        - learning_rate has a default value of 0.01 and will decrease with
          each iteration accoring to the below formula:
                learning_rate = learning_rate/2

        - n_iterations is the number of iterations with a default value of 100.
          We can change the number of iterations to see
          how the performance of the model changes with higher iterations.

          This function will return the optimal w and b for the dataset X
    '''
    w = np.zeros(shape=(1,13)) # Initialize the weights as a (1,13)D vector
    b = 0.0 # initialize the intercept term as 0
    decay = learning_rate/10000 # Decay rate

    for i in range(n_iterations):
        # print("Current w: {} and b: {}".format(w,b))
        # First randomly take batch size of 10 from the data
        batch = X.sample(10)

        # Separate independent and dependent variables and convert into numpy ar
        # for ease of computation.
        data = np.array(batch.drop('Price', axis=1)) # Independent variables fro
        target = np.array(batch['Price']) # Dependent variable

        # Initialize the partial derivative terms
        # l_der_w is derivative of L (Loss function) with respect to w
        # l_der_b is derivative of L (Loss function) with respect to b

        l_der_w = np.zeros(shape=(1,13)) # derivative of L w.r.t w will also be
        l_der_b = 0.0

        # l_der_w = -2/n * Sum(x_i * (y - y_hat))
        # l_der_b = -2/n * Sum(y - y_hat)
        # As we are performing SGD, we will first take the summation of all the
        # the batch, and then perform the -2/n part for ease of computation
        for j in range(10):
            y = np.dot(w, data[j]) + b
            l_der_w += data[j] * (target[j] - y)
            l_der_b += target[j] - y

        l_der_w *= - 2/data.shape[0] # data.shape[0] is number of points, 10 in
        l_der_b *= - 2/data.shape[0]

        # Update the weight and intercept term
        w1 = w - learning_rate * l_der_w
        b1 = b - learning_rate * l_der_b

        # Replace w and b with updated values
        w = w1
        b = b1

        # Update the learning rate
        learning_rate *= (1./ (1. + decay * i))
```

```
    return w, b
```

In [8]:
```
# Define function to predict test data for ease of use
def predict (data, w, b):
    predicted = []
    data = np.array(data)
    for i in range(data.shape[0]):
        yhat = np.dot(w,data[i]) + b # By following the equation y = (w.T * x) +
        predicted.append(yhat)
    return predicted
```

To make the results of my implementation similar to that of sklearn's SGDRegressor, I will be using default learning rate as 0.01 as it is done in the sklearn implementation.

Now keeping the initial learning rate same, there are two things we can vary to make our model more similar to the sklearn implementation. The first one is how we are going to decrease the learning rate. For that I am using the following formula:

learning_rate *= 1./ (1. + decay+iterations)

The other thing that we can modify is the number of iterations for which our model will run to find the minima point. SO to test the manual implementation for how effectively it can find the optimal parameters compared to the sklearn version of the algorithm, I will run both my implementation and sklearn for 100, 1000, 10000 iterations to see how the error rate changes over the number of iterations.

In [9]:
```
# Divide the dataset into train and test
from sklearn.model_selection import train_test_split
train_data, test_data, train_y, test_y = train_test_split(data, target, test_siz
```

Compare sklearn and sgd_implementation using n_iterations = 100

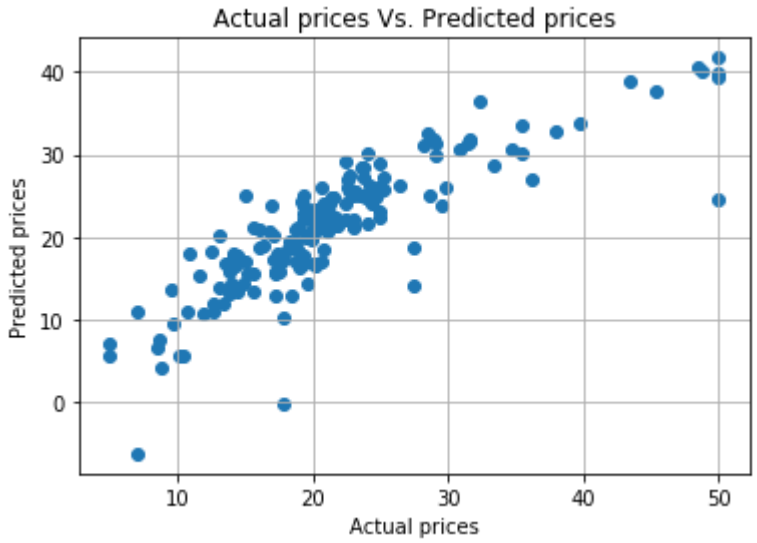
For sklearn implementation

In [10]:
```
clf = SGDRegressor(max_iter=100)
clf.fit(train_data, train_y)
pred_sklearn_100 = clf.predict(test_data)
```

In [11]:
```
print("Error for sklearn sgd implementation after 100 iterations: ", mean_square
print("Weight is :\n {}".format(clf.coef_))
print("Intercept term is: ", clf.intercept_)
```

```
Error for sklearn sgd implementation after 100 iterations:  21.164667136894007
Weight is :
 [-1.0337966   0.7279046   0.19097468  0.86211595 -1.72122605  2.77398882
 -0.4033986  -2.86411795  1.40707398 -0.72521862 -1.96255155  1.06930931
 -3.86768486]
Intercept term is:  [22.44606377]
```

In [12]:
```
scatter_plot(test_y, pred_sklearn_100)
```



For manual SGD implementation

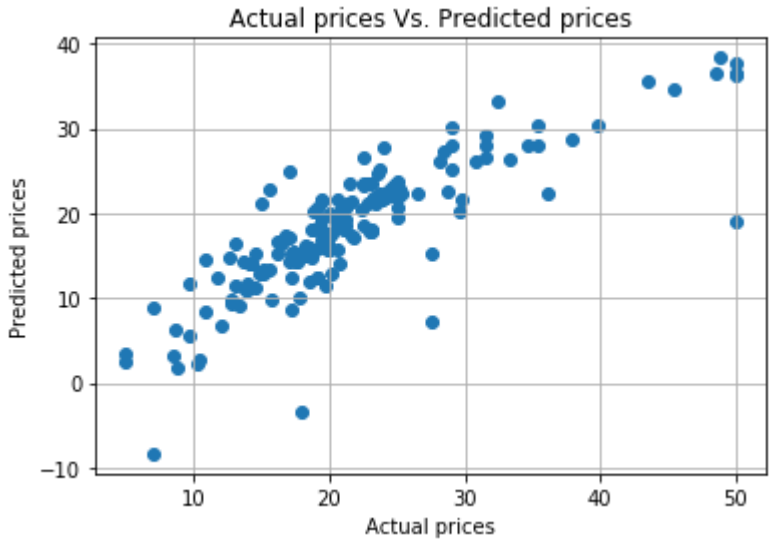
In [13]:
```
train_df = pd.DataFrame(data=train_data, copy=True)
train_df['Price'] = train_y
```

In [14]:
```
w_100, b_100 = sgd_implementation(train_df)
pred_sgd_100 = predict(test_data, w_100, b_100)
```

In [15]:
```
print("Error for manual sgd implementation after 100 iterations: ", mean_squared_
print("Weight is :\n {}".format(w_100))
print("Intercept term is: ", b_100)
```

```
Error for manual sgd implementation after 100 iterations:  33.62158870033977
Weight is :
 [[-0.98728877  0.38823902 -0.03485412  1.31281231 -0.70766024  3.26386419
  -0.17123255 -1.29340844  0.28699944 -0.12057444 -2.15030377  0.8004351
  -3.13020927]]
Intercept term is:  [19.50868604]
```

In [16]:
```
scatter_plot(test_y, pred_sgd_100)
```



Compare sklearn and sgd_implementation using n_iterations = 1000

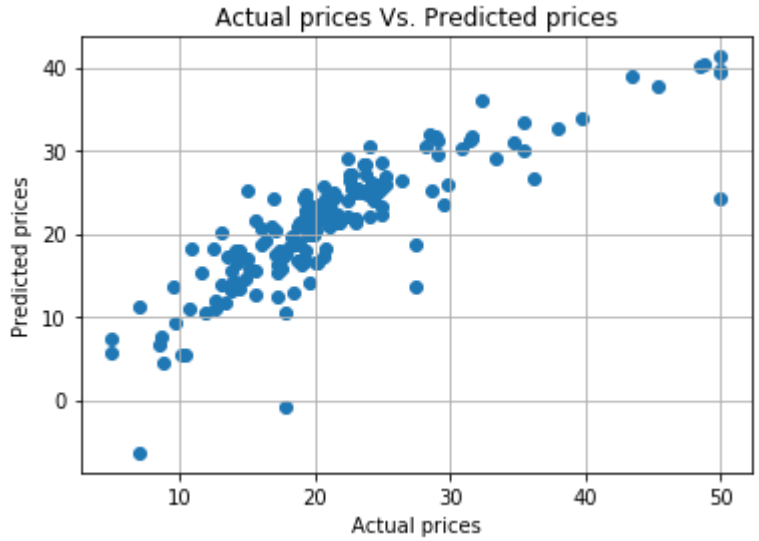
For sklearn implementation

In [17]:
```
clf = SGDRegressor(max_iter=1000)
clf.fit(train_data, train_y)
pred_sklearn_1000 = clf.predict(test_data)
```

In [18]:
```
print("Error for sklearn sgd implementation after 1000 iterations: ", mean_squar
print("Weight is :\n {}".format(clf.coef_))
print("Intercept term is: ", clf.intercept_)
```

```
Error for sklearn sgd implementation after 1000 iterations:  21.544894516529396
Weight is :
 [-0.97708972  0.58132342  0.11348625  0.86641227 -1.5205134   2.87914702
 -0.40892507 -2.63521885  1.06924268 -0.41513319 -1.94897477  1.08184087
 -3.8520676 ]
Intercept term is:  [22.45389246]
```

In [19]:
```
scatter_plot(test_y, pred_sklearn_1000)
```



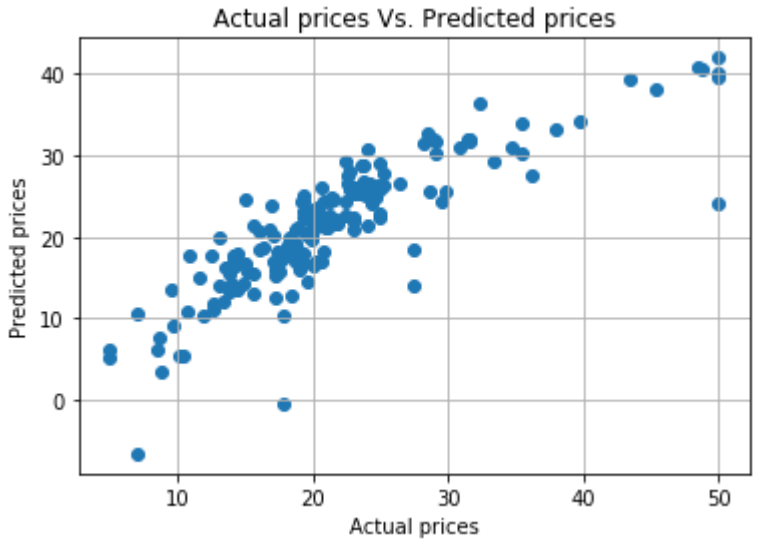
For manual SGD implementation

In [20]:
```
w_1000, b_1000 = sgd_implementation(train_df, n_iterations=1000)
pred_sgd_1000 = predict(test_data, w_1000, b_1000)
```

In [21]:
```
print("Error for manual sgd implementation after 1000 iterations: ", mean_square
print("Weight is :\n {}".format(w_1000))
print("Intercept term is: ", b_1000)
```

```
Error for manual sgd implementation after 1000 iterations:  21.250599211173597
Weight is :
 [[-1.11089179  0.6806159  -0.04175759  0.80722595 -1.62666979  2.75318474
  -0.55170528 -3.00754978  1.32846858 -0.79947169 -1.96597687  1.00042843
  -3.85121107]]
Intercept term is:  [22.5328715]
```

In [22]:
```
scatter_plot(test_y, pred_sgd_1000)
```



Compare sklearn and sgd_implementation using n_iterations = 10000

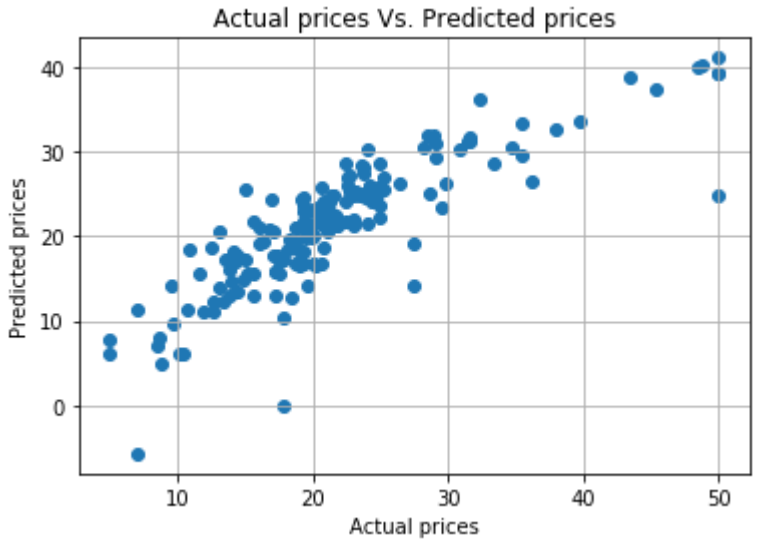
For sklearn implementation

In [23]:
```
clf = SGDRegressor(max_iter=10000)
clf.fit(train_data, train_y)
pred_sklearn_10000 = clf.predict(test_data)
```

In [24]:
```
print("Error for sklearn sgd implementation after 1000 iterations: ", mean_squar
print("Weight is :\n {}".format(clf.coef_))
print("Intercept term is: ", clf.intercept_)
```

```
Error for sklearn sgd implementation after 1000 iterations:  21.30170583497408
Weight is :
 [-0.95824851  0.63470745  0.18242194  0.90420189 -1.5604892   2.81415682
 -0.36757937 -2.76559842  1.24477825 -0.49397855 -1.91811447  1.05932604
 -3.86036931]
Intercept term is:  [22.46901712]
```

In [25]:
```
scatter_plot(test_y, pred_sklearn_10000)
```



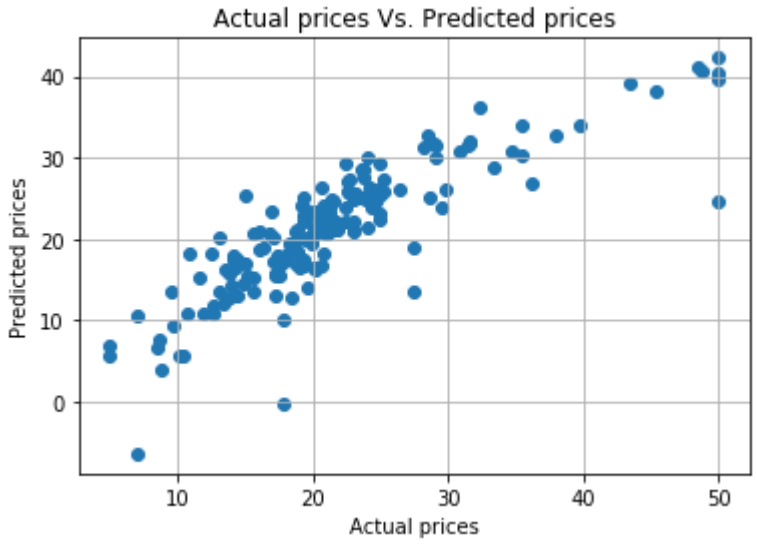
For manual SGD implementation

In [26]:
```
w_10000, b_10000 = sgd_implementation(train_df, n_iterations=10000)
pred_sgd_10000 = predict(test_data, w_10000, b_10000)
```

In [27]:
```
print("Error for manual sgd implementation after 1000 iterations: ", mean_square
print("Weight is :\n {}".format(w_10000))
print("Intercept term is: ", b_10000)
```

```
Error for manual sgd implementation after 1000 iterations:  20.935815828034034
Weight is :
 [[-1.06486567  0.7535466   0.28182885  0.79603073 -1.86797309  2.87260287
  -0.46522883 -2.9401687   1.66632885 -0.90142283 -2.05704551  1.0795211
  -3.78160089]]
Intercept term is:  [22.48935225]
```

In [28]:
```
scatter_plot(test_y, pred_sgd_10000)
```



Conclusion:

Model	Learning rate	iterations	Error	Intercept
Sklearn SGD	0.01	100	20.80	22.46
Manual SGD	0.01	100	33.78	19.57
Sklearn SGD	0.01	1000	20.72	22.47
Manual SGD	0.01	1000	20.81	22.39
Sklearn SGD	0.01	10000	20.75	22.46
Manual SGD	0.01	10000	20.77	22.45

- From the above table it can be clearly stated that with the increase in the number of iterations, the model's probability to find a point that is a local or global minima increases.
- For 100 iterations, the sklearn and manual SGD error rate was varying quite a lot (20.8 and 33.78 respectively) but as the number of iteration increased, at 10000 iterations the error rate and also the intercept term are almost equal (20.75 and 20.77) which means our model is almost similar in performance with the sklearn's SGDRegressor implementation.