SGD implementation of Linear regression

In [19]:

```python
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,mean_absolute_error
from numpy import random
from sklearn.model_selection import train_test_split
```

Data Preprocessing:

In [20]:

```python
boston_data=pd.DataFrame(load_boston().data,columns=load_boston().feature_names)
Y=load_boston().target
X=load_boston().data
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.3)
```

In [21]:

```python
# data overview
boston_data.head(3)
```

Out[21]:

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

File failed to load: /extensions/MathZoom.js

In [22]:
```
# standardizing data
scaler = preprocessing.StandardScaler().fit(x_train)
x_train = scaler.transform(x_train)
x_test=scaler.transform(x_test)
```

In [23]:
```
train_data=pd.DataFrame(x_train)
train_data['price']=y_train
train_data.head(3)
```

Out[23]:

	0	1	2	3	4	5	6	7	
0	-0.226183	-0.474301	1.192799	3.412163	0.404192	2.184216	1.035946	-0.799665	-0.52884
1	-0.418239	0.398600	-0.637802	3.412163	-0.797951	2.026146	-0.609753	0.326104	-0.75604
2	-0.438001	-0.474301	-1.290552	-0.293069	-0.593331	2.225513	-0.545911	-0.238910	-0.75604

In [24]:
```
x_test=np.array(x_test)
y_test=np.array(y_test)
```

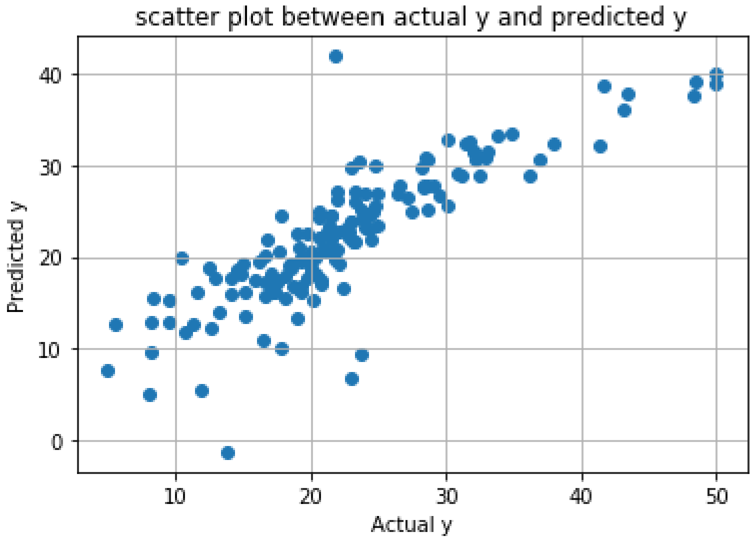
In [25]:
```
# shape of test and train data matxis
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

SGD on Linear Regression : SKLearn Implementation

File failed to load: /extensions/MathZoom.js

In [26]:
```
# SkLearn SGD classifier
clf_ = SGDRegressor()
clf_.fit(x_train, y_train)
plt.scatter(y_test,clf_.predict(x_test))
plt.grid()
plt.xlabel('Actual y')
plt.ylabel('Predicted y')
plt.title('scatter plot between actual y and predicted y')
plt.show()
print('Mean Squared Error :',mean_squared_error(y_test, clf_.predict(x_test)))
print('Mean Absolute Error :',mean_absolute_error(y_test, clf_.predict(x_test
)))
```



```
Mean Squared Error : 20.298006670135905
Mean Absolute Error : 3.0954808822581468
```

In [27]:
```
# SkLearn SGD classifier predicted weight matrix
sklearn_w=clf_.coef_
sklearn_w
```

Out[27]:
```
array([-0.35740571,  0.46525096, -0.17520602,  1.20142239, -0.83070563,
        3.54279961, -0.05180555, -1.71807425,  0.87434284, -0.45418361,
       -1.55177869,  0.97245208, -3.56209867])
```

Custom Implementation

File failed to load: /extensions/MathZoom.js

In [28]:

```
# implemented SGD Classifier
def CustomGradientDescentRegressor(train_data,learning_rate=0.001,n_itr=1000,k
=10):
    w_cur=np.zeros(shape=(1,train_data.shape[1]-1))
    b_cur=0
    cur_itr=1
    while(cur_itr<=n_itr):
        w_old=w_cur
        b_old=b_cur
        w_temp=np.zeros(shape=(1,train_data.shape[1]-1))
        b_temp=0
        temp=train_data.sample(k)
        #print(temp.head(3))
        y=np.array(temp['price'])
        x=np.array(temp.drop('price',axis=1))
        for i in range(k):
            w_temp+=x[i]*(y[i]-(np.dot(w_old,x[i])+b_old))*(-2/k)
            b_temp+=(y[i]-(np.dot(w_old,x[i])+b_old))*(-2/k)
        w_cur=w_old-learning_rate*w_temp
        b_cur=b_old-learning_rate*b_temp
        if(w_old==w_cur).all():
            break
        cur_itr+=1
    return w_cur,b_cur
def predict(x,w,b):
    y_pred=[]
    for i in range(len(x)):
        y=np.asscalar(np.dot(w,x[i])+b)
        y_pred.append(y)
    return np.array(y_pred)


def plot_(test_data,y_pred):
    #scatter plot
    plt.scatter(test_data,y_pred)
    plt.grid()
    plt.title('scatter plot between actual y and predicted y')
    plt.xlabel('actual y')
    plt.ylabel('predicted y')
    plt.show()
```

Hyper Parameter tuning for optimal Learning rate

File failed to load: /extensions/MathZoom.js

In [29]:

```
# Funtion to get optimal learning rate on the implemented SGD Classifier
from math import log
x1_train,x1_test,y1_train,y1_test=train_test_split(X,Y,test_size=0.3)
x1_train,x1_cv,y1_train_,y1_cv_=train_test_split(x1_train,y1_train,test_size=
0.3)

x1_train = scaler.transform(x1_train)
x1_cv=scaler.transform(x1_cv)

x1_train_=np.array(x1_train)
x1_train_data=pd.DataFrame(x1_train)
x1_train_data['price']=y1_train_

x1_cv_data=pd.DataFrame(x1_cv)
x1_cv_data['price']=y1_cv_

y1_train_=np.array(y1_train_)
y1_cv_=np.array(y1_cv_)
#print(y1_cv_.shape)

def tuneParams_learning_rate():
    train_error=[]
    cv_error=[]
    r=[0.00001,0.0001,0.001,0.01,0.1]
    for itr in r:
        w,b=CustomGradientDescentRegressor(x1_train_data,learning_rate=itr,n_i
tr=1000)
       # print(w.shape,b.shape,x1_train_.shape)
        y1_pred_train=predict(x1_train_,w,b)
        train_error.append(mean_squared_error(y1_train_,y1_pred_train))
        w,b=CustomGradientDescentRegressor(x1_cv_data,learning_rate=itr,n_itr=
1000)
        y1_pred_cv=predict(x1_cv,w,b)
        cv_error.append(mean_squared_error(y1_cv_,y1_pred_cv))
    return train_error,cv_error
```

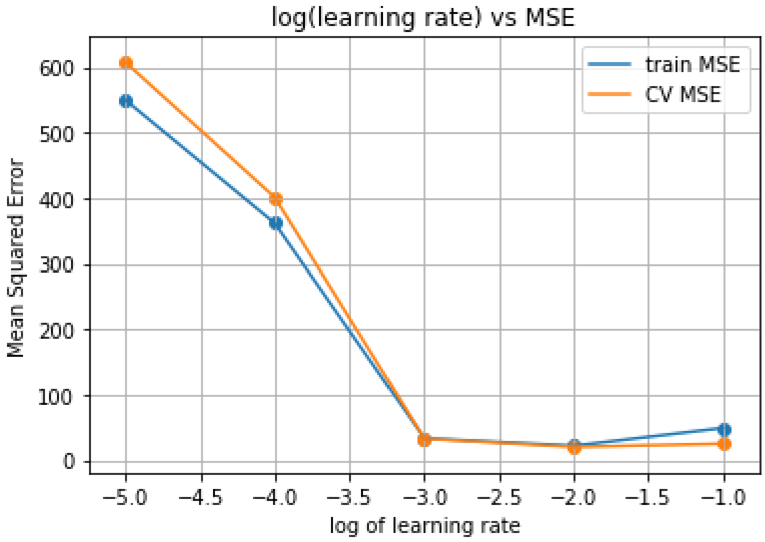
In [30]:

```
train_error,cv_error=tuneParams_learning_rate()
```

File failed to load: /extensions/MathZoom.js

In [31]:

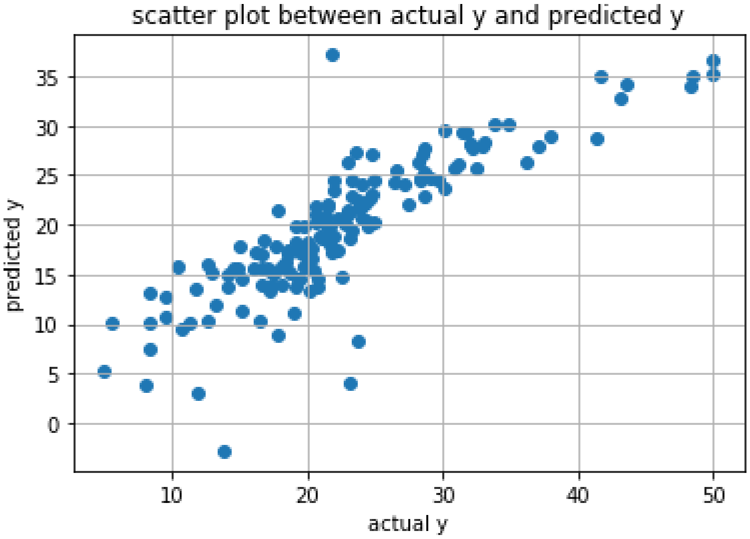
```
 # plotting obtained values
import math
r=[0.00001,0.0001,0.001,0.01,0.1]
x1=[math.log10(i) for i in r]
plt.plot(x1,train_error,label='train MSE')
plt.plot(x1,cv_error,label='CV MSE')
plt.scatter(x1,train_error)
plt.scatter(x1,cv_error)
plt.legend()
plt.xlabel('log of learning rate')
plt.ylabel('Mean Squared Error')
plt.title('log(learning rate) vs MSE')
plt.grid()
plt.show()
```



SGD with optimal learning rate

File failed to load: /extensions/MathZoom.js

In [32]:
```
# running implemented SGD Classifier with obtained optimal learning rate
w,b=CustomGradientDescentRegressor(train_data,learning_rate=0.001,n_itr=1000)
y_pred=predict(x_test,w,b)
plot_(y_test,y_pred)
```



In [33]:
```
# Errors in implemeted model
print(mean_squared_error(y_test,y_pred))
print(mean_absolute_error(y_test,y_pred))
```

```
27.787082574419173
3.8619401806038254
```

In [34]:
```
#weight vector obtained from impemented SGD Classifier
custom_w=w
custom_w
```

Out[34]:
```
array([[-0.45547852,  0.22361356, -0.33317742,  1.01659259, -0.54123265,
         3.41119829, -0.1125143 , -1.21257381,  0.50713945, -0.32815929,
        -1.52460644,  0.87383731, -3.08782792]])
```

Comparing Models

File failed to load: /extensions/MathZoom.js

In [35]:
```
from prettytable import PrettyTable
# MSE = mean squared error
# MAE =mean absolute error
x=PrettyTable()
x.field_names=['Model','Weight Vector','MSE','MAE']
x.add_row(['sklearn',sklearn_w,mean_squared_error(y_test, clf_.predict(x_test
)),mean_absolute_error(y_test, clf_.predict(x_test))])
x.add_row(['custom',custom_w,mean_squared_error(y_test,y_pred),(mean_absolute_
error(y_test,y_pred))])
print(x)
```

```
+---------+------------------------------------------------------------------
---------+--------------------+--------------------+
|  Model  |                               Weight Vector
|        MSE         |        MAE         |
+---------+------------------------------------------------------------------
---------+--------------------+--------------------+
| sklearn |  [-0.35740571  0.46525096 -0.17520602  1.20142239 -0.83070563  3.
54279961 | 20.298006670135905 | 3.0954808822581468 |
|         |   -0.05180555 -1.71807425  0.87434284 -0.45418361 -1.55177869  0.
97245208 |                    |                    |
|         |                                -3.56209867]
|                    |                    |
|  custom | [[-0.45547852  0.22361356 -0.33317742  1.01659259 -0.54123265  3.
41119829 | 27.787082574419173 | 3.8619401806038254 |
|         |   -0.1125143  -1.21257381  0.50713945 -0.32815929 -1.52460644  0.
87383731 |                    |                    |
|         |                                -3.08782792]]
|                    |                    |
+---------+------------------------------------------------------------------
---------+--------------------+--------------------+
```

Comparison Between top 15 predicted value of both models:

File failed to load: /extensions/MathZoom.js

In [36]:

```
sklearn_pred=clf_.predict(x_test)
implemented_pred=y_pred
x=PrettyTable()
x.field_names=['SKLearn SGD predicted value','Implemented SGD predicted value'
]
for itr in range(15):
    x.add_row([sklearn_pred[itr],implemented_pred[itr]])
print(x)
```

```
+-----------------------------+---------------------------------+
| SKLearn SGD predicted value | Implemented SGD predicted value |
+-----------------------------+---------------------------------+
|      22.50229058328867      |        19.78138695110681        |
|      19.529619242375333     |        17.111326053534448       |
|      16.96672572560702      |        13.817998263886187       |
|      23.08531223974879      |        20.31801543591488        |
|      14.021831515275348     |        11.85554142407669        |
|      24.989351114660796     |        21.843575403703817       |
|      20.975846961288887     |        18.23277996120414        |
|      33.30864273925246      |        30.24805634058287        |
|      29.674052553655343     |        26.226328467463976       |
|      32.26493259766083      |        28.97091653861654        |
|      40.071349346458156     |        36.603904969860295       |
|      28.84885129684657      |        26.086885304731098       |
|      19.542588731439505     |        16.780942611170165       |
|      22.83468188050603      |        20.271434559150318       |
|      26.99591963670726      |        24.127288747863204       |
+-----------------------------+---------------------------------+
```

1.The predicted values between two implementations are almost similar. 2.The SGD classifier is implemented with batch size of 20 and a learning rate of 0.001 without any regularization term.

File failed to load: /extensions/MathZoom.js