# SGD FOR LINEAR REGRESSION ON BOSTON HOME PRICE DATASET

## BOSTON HOME PRICE DATASET

Number of Instances 506

Number of Attributes 13 numeric/categorical predictive. Median Value (attribute 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 otherwise)

**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 town

**LSTAT** - % lower status of the population

**MEDV** - Median value of owner-occupied homes in $1000's (**prices of the house**)

```
1 from sklearn.datasets import load_boston
2
3 boston=load_boston()
```

```
1 boston.data.shape
```

```
(506, 13)
```

```
1 boston.feature_names
```

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

```
1 print(boston.DESCR)
```

```
.. _boston_dataset:

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

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 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 otherwise)
        - 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 town
        - 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 address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity',
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Internati
```

```
1 import numpy as np
2 from sklearn.preprocessing import StandardScaler
3 import pandas as pd
4 import warnings
5 warnings.filterwarnings('ignore')
```

```
1 boston_dataset=pd.DataFrame(data=boston.data)
```

```
1 boston_dataset.head(3)
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 |
| **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 |
| **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 |

```
1 boston_dataset.columns=boston.feature_names
```

```
1 boston_dataset.head(3)
```

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 |
| **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 |
| **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 |

```
1 boston_dataset['MEDV']=boston.target
```

```
1 boston_dataset
```

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | MEDV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 24.0 |
| **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 | 21.6 |
| **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 | 34.7 |
| **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 | 33.4 |
| **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 | 36.2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **501** | 0.06263 | 0.0 | 11.93 | 0.0 | 0.573 | 6.593 | 69.1 | 2.4786 | 1.0 | 273.0 | 21.0 | 391.99 | 9.67 | 22.4 |
| **502** | 0.04527 | 0.0 | 11.93 | 0.0 | 0.573 | 6.120 | 76.7 | 2.2875 | 1.0 | 273.0 | 21.0 | 396.90 | 9.08 | 20.6 |
| **503** | 0.06076 | 0.0 | 11.93 | 0.0 | 0.573 | 6.976 | 91.0 | 2.1675 | 1.0 | 273.0 | 21.0 | 396.90 | 5.64 | 23.9 |
| **504** | 0.10959 | 0.0 | 11.93 | 0.0 | 0.573 | 6.794 | 89.3 | 2.3889 | 1.0 | 273.0 | 21.0 | 393.45 | 6.48 | 22.0 |
| **505** | 0.04741 | 0.0 | 11.93 | 0.0 | 0.573 | 6.030 | 80.8 | 2.5050 | 1.0 | 273.0 | 21.0 | 396.90 | 7.88 | 11.9 |

506 rows × 14 columns

```
1 house_price=boston_dataset['MEDV']
2 boston_dataset = boston_dataset.drop("MEDV", axis = 1)
```

```
1 from sklearn.model_selection import train_test_split
2
3 X_Train,X_Test,Y_Train,Y_Test = train_test_split(boston_dataset,house_price,test_size=0.2,random_state=0)
```

## Standarize the data,

```
1 X_Train.shape
```

```
(404, 13)
```

```
1 from sklearn.preprocessing import StandardScaler
2
```

```
scalar = StandardScaler()
X_Train =scalar.fit_transform(np.array(X_Train))
X_Test=scalar.transform(np.array(X_Test))
```

```
manual_training=pd.DataFrame(data=X_Train)
```

```
manual_training
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.372574 | -0.499608 | -0.704925 | 3.664502 | -0.424879 | 0.935678 | 0.693669 | -0.437218 | -0.162242 | -0.561656 | -0.48 |
| **1** | -0.397099 | -0.499608 | -0.044878 | -0.272888 | -1.241859 | -0.491181 | -1.835528 | 0.730055 | -0.624648 | -0.573376 | 0.33 |
| **2** | -0.402693 | 0.771168 | -0.886760 | -0.272888 | -1.111490 | 0.857849 | -0.524621 | 1.234769 | -0.393445 | -0.602677 | -0.84 |
| **3** | -0.405769 | 0.029882 | -0.465819 | -0.272888 | -0.277127 | -0.417676 | -0.086464 | 0.861527 | -0.509046 | -0.538216 | -1.48 |
| **4** | 2.774932 | -0.499608 | 0.998884 | -0.272888 | 1.070021 | -1.438097 | 0.715042 | -1.021528 | 1.687378 | 1.542121 | 0.79 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **399** | -0.381700 | -0.499608 | -0.535975 | -0.272888 | -0.546557 | -0.855823 | 0.187829 | 0.457582 | -0.509046 | -0.678858 | 0.51 |
| **400** | -0.405972 | 1.406555 | -1.100094 | -0.272888 | -1.033268 | 1.262847 | -1.522051 | 1.303922 | -0.509046 | -0.028387 | -1.48 |
| **401** | -0.398056 | -0.499608 | -0.159419 | -0.272888 | -0.077228 | -0.404705 | 0.483496 | -0.492223 | -0.393445 | 0.170857 | -0.30 |
| **402** | -0.388424 | -0.499608 | -0.603269 | -0.272888 | -0.937664 | -0.391733 | 0.586801 | 0.923558 | -0.740249 | -0.995304 | -0.25 |
| **403** | -0.399513 | -0.499608 | -1.012756 | -0.272888 | -0.398805 | -1.051836 | 0.693669 | -0.565033 | -0.509046 | -0.626117 | -0.84 |

404 rows × 13 columns

```
type(manual_training)
```

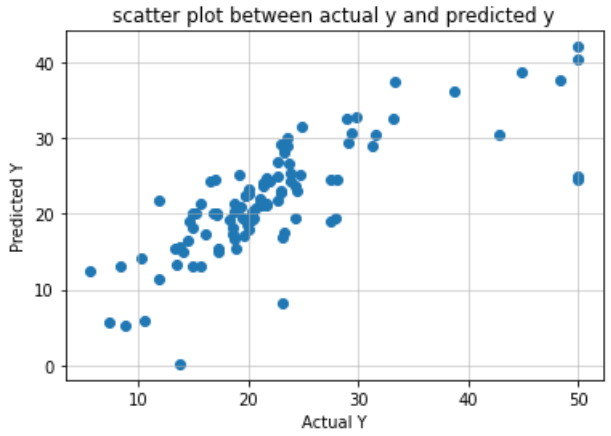
```
pandas.core.frame.DataFrame
```

```
manual_training.columns=boston.feature_names
```

```
manual_training['MEDV']=Y_Train
```

```
from sklearn.linear_model import SGDRegressor
import seaborn as sns
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

def sklearn_SGDRegressor(X_Train,Y_Train,X_Test,Y_Test,alpha,learning_rate,n_iterations):

  linear_regression=SGDRegressor(alpha=alpha,learning_rate=learning_rate,n_iter_no_change=n_iterations)
  linear_regression.fit(X_Train,Y_Train)

  Y_Predicted=linear_regression.predict(X_Test)

  plt.scatter(Y_Test,Y_Predicted)
  plt.title('scatter plot between actual y and predicted y')
  plt.xlabel('Actual Y')
  plt.ylabel('Predicted Y')
  plt.grid(b=True, linewidth=0.5)
  plt.show()

  return Y_Predicted,n_iterations,linear_regression.coef_, linear_regression.intercept_
```

```
y_predicted,n_iter,weight_sgd,intercept_sgd=sklearn_SGDRegressor(X_Train,Y_Train,X_Test,Y_Test,0.0001,'constant',1
```




```
sgd_error=mean_squared_error(Y_Test,y_predicted)
print('Mean Square Error(MSE)=', sgd_error)
print('Number of iteration=', n_iter)
```

```
Mean Square Error(MSE)= 32.80756905086225
Number of iteration= 1
```

```
print('W[j+1] :', weight_sgd )
print('b[j+1] :', intercept_sgd )
```

```
W[j+1] : [-0.81096515  1.0039064   0.02367056  0.54872808 -1.25903411  3.04495628
  0.08831497 -2.72093591  1.4928305  -0.85047471 -2.50149652  1.23202212
 -3.70111891]
b[j+1] : [22.56270378]
```

```
manual_training = manual_training.drop("MEDV", axis = 1)
```

## Using Manual Approach to find the Linear Regression

```
def find_differential_L_wrt_W(X_Data,Y_Data,W_tranpose,K,b,d):

  # Calcuate Differential of L w.r.t W
  sum_diff = np.zeros(shape=(1,d))
  for i in range(K):
    second_term = (Y_Data.iloc[i] - b - np.dot(W_tranpose,X_Data[i]))
    final_term = X_Data[i] * second_term
    sum_diff += final_term
  sum_diff *= (-2/K)
  return sum_diff
```

```
def find_differential_L_wrt_b(X_Data,Y_Data,W_Transpose,K,b,d):
  #Calculate differential of L w.r.t b
  sum_diff = 0
  for i in range(K):
    final_term = Y_Data.iloc[i] - b - np.dot(W_Transpose,X_Data[i])
    sum_diff += final_term
  sum_diff *= (-2/K)
  return sum_diff
```

```
def manual_predict(X_Test,W,b):
  Y_Predicted=[]

  for i in range(len(X_Test)) :
    #predict = (np.dot(W,X_Test[i]) + b)
```

```
    y=np.asscalar(np.dot(W,X_Test[i]) + b)
    Y_Predicted.append(y)
  return np.array(Y_Predicted)
```

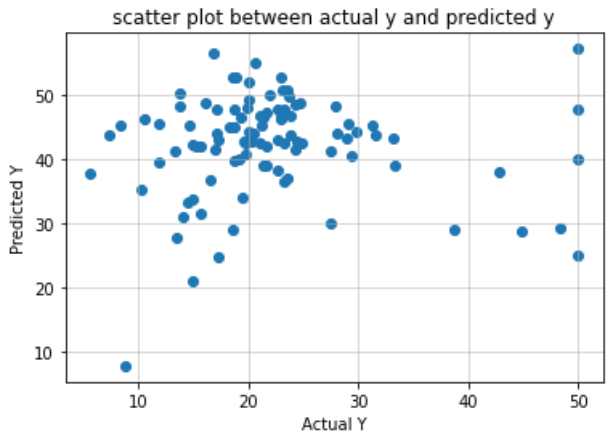
```
def draw_scatterPlot(Y_Test,Y_Predicted) :
  plt.scatter(Y_Test,Y_Predicted)
  plt.title('scatter plot between actual y and predicted y')
  plt.xlabel('Actual Y')
  plt.ylabel('Predicted Y')
  plt.grid(b=True, linewidth=0.5)
  plt.show()
```

```
def find_and_transform(X,Y,K,n_iterations,threshold,power_t,lr_rate) :
  n,d=X.shape
  b=0
  weight = np.random.randn(1,d)
  r,iter_ = 1,1
  n=1
  while(n<=n_iterations):
    w_prev = weight
    b_prev = b
    X_data=X.sample(K)
    X_new=X_data[X_data.columns[0:]].values
    Y_new = Y

    Diff_W = find_differential_L_wrt_W(X_new,Y_new,w_prev,K,b_prev,d)
    Diff_b = find_differential_L_wrt_b(X_new,Y_new,w_prev,K,b_prev,d)

        #Calculate Wj+1 & bj+1

    Wj_plus_1 = w_prev - (r*Diff_W)
    bj_plus_1= b_prev - (r*Diff_b)

    if(iter_ >= n_iterations):
      break

    #r = r/2
    iter_ +=1
    n=n+1


  print('************************************')

  print('W[j+1] :' , Wj_plus_1)
  print('b[j+1] :' , bj_plus_1)
  #print('r :' , r)
  print('************************************')
  return Wj_plus_1,bj_plus_1
```

```
W_star,b_star=find_and_transform(manual_training,Y_Train,50,1,0.1,0.25,0.01)
```

```
************************************
W[j+1] : [[-2.81993685  1.82824696  2.49608589  1.78137557 -0.80908617 -4.67501745
  -1.6640677   0.30820965  2.08038961  1.41089452  1.94006647  3.70436801
  -2.89953773]]
b[j+1] : [42.01570198]
r : 1
************************************
```

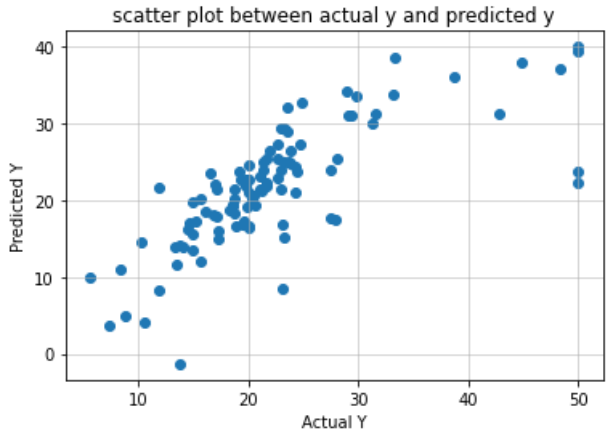
```
Y_Predicted=manual_predict(X_Test, W_star,b_star)
draw_scatterPlot(Y_Test,Y_Predicted)
manual_error=mean_squared_error(Y_Test,Y_Predicted)
print('Mean Square Error(MSE) Using Manual Approach =', manual_error)
```




```
Mean Square Error(MSE) Using Manual Approach = 543.8083421548782
```

## Using SGD Regressor for No Of Iterations = 100

```
y_predicted,n_iter,weight_sgd,intercept_sgd=sklearn_SGDRegressor(X_Train,Y_Train,X_Test,Y_Test,0.0001,'constant',1
```




```
sgd_error=mean_squared_error(Y_Test,y_predicted)
print('Mean Square Error(MSE)=', sgd_error)
print('Number of iteration=', n_iter)
```

```
Mean Square Error(MSE)= 36.589875144670955
Number of iteration= 100
```

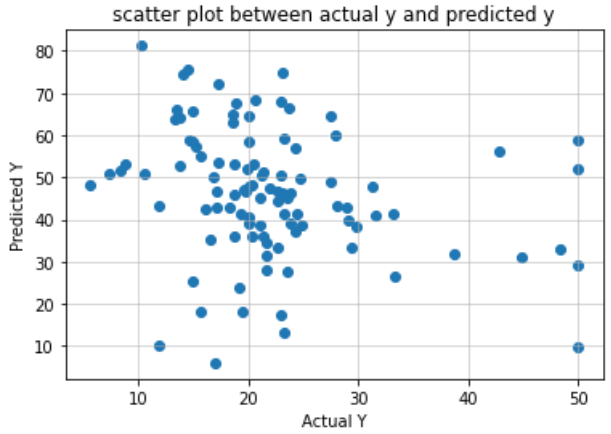
```
print('W[j+1] :', weight_sgd )
print('b[j+1] :', intercept_sgd )
```

```
W[j+1] : [-0.76946944  1.25748731 -0.21317185  0.73145965 -1.95077354  2.68340876
 -0.20734028 -2.65472201  2.01102926 -2.01512418 -2.3411727   0.7714277
 -3.70722231]
b[j+1] : [22.36872328]
```

## Using Manual Linear Regression for No Of Iterations = 100

```
W_star,b_star=find_and_transform(manual_training,Y_Train,50,100,0.1,0.25,0.01)
Y_Predicted=manual_predict(X_Test, W_star,b_star)
draw_scatterPlot(Y_Test,Y_Predicted)
manual_error=mean_squared_error(Y_Test,Y_Predicted)
print('Mean Square Error(MSE) Using Manual Approach =', manual_error)
```

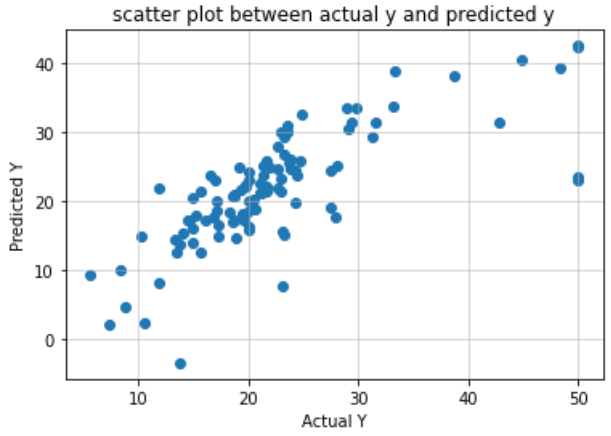
```
************************************
W[j+1] : [[-2.68493813  0.56919869  0.9558448  -2.27750919 -3.72175452 -1.67978081
  -2.52142237  3.16574685  3.80873151  6.9708999   8.99502337 -4.45016381
  -1.40969299]]
b[j+1] : [43.98986384]
r : 1
************************************
```




```
Mean Square Error(MSE) Using Manual Approach = 953.3641591193748
```

## Using SGD Regressor for No Of Iterations = 1000

```
y_predicted,n_iter,weight_sgd,intercept_sgd=sklearn_SGDRegressor(X_Train,Y_Train,X_Test,Y_Test,0.0001,'constant',1
```




```
sgd_error=mean_squared_error(Y_Test,y_predicted)
print('Mean Square Error(MSE)=', sgd_error)
print('Number of iteration=', n_iter)
```

```
Mean Square Error(MSE)= 33.49280515235464
Number of iteration= 1000
```

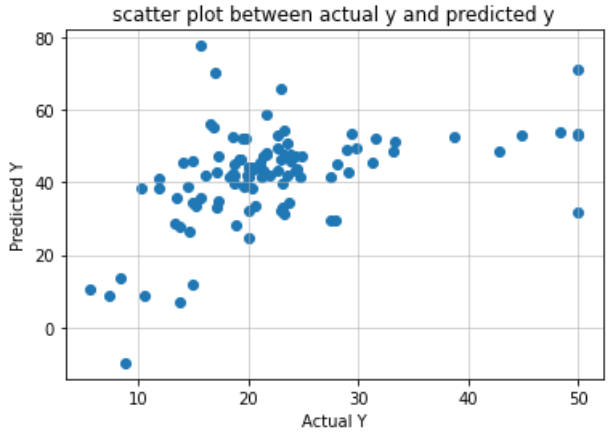
```
print('W[j+1] :', weight_sgd )
print('b[j+1] :', intercept_sgd )
```

```
W[j+1] : [-0.95180992  1.05603303 -0.06516896  0.59334444 -2.03775517  3.24860141
  0.29953031 -2.88869107  2.02163147 -2.09274246 -2.64764731  0.49126429
 -3.89843943]
b[j+1] : [22.3750692]
```

## Using Manual Regression for No Of Iterations = 1000

```
W_star,b_star=find_and_transform(manual_training,Y_Train,50,1000,0.1,0.25,0.01)
Y_Predicted=manual_predict(X_Test, W_star,b_star)
draw_scatterPlot(Y_Test,Y_Predicted)
manual_error=mean_squared_error(Y_Test,Y_Predicted)
print('Mean Square Error(MSE) Using Manual Approach =', manual_error)
```

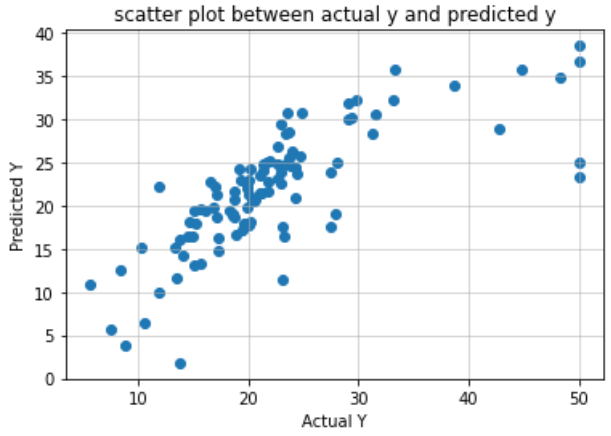
```
************************************
W[j+1] : [[-6.33871176 -3.66884595 -3.22007523  4.79155866  2.11014711  0.70846533
  -0.02835366  1.16817197 -0.24489098 -0.2492538  -3.90611166 -3.38394822
  -3.8233088 ]]
b[j+1] : [41.81694397]
************************************
```




```
Mean Square Error(MSE) Using Manual Approach = 522.542501849131
```

## Using SGD Regressor for No Of Iterations = 10000

```
y_predicted,n_iter,weight_sgd,intercept_sgd=sklearn_SGDRegressor(X_Train,Y_Train,X_Test,Y_Test,0.0001,'constant',1
```




```
sgd_error=mean_squared_error(Y_Test,y_predicted)
print('Mean Square Error(MSE)=', sgd_error)
print('Number of iteration=', n_iter)
```

```
Mean Square Error(MSE)= 34.5350056094487
Number of iteration= 10000
```

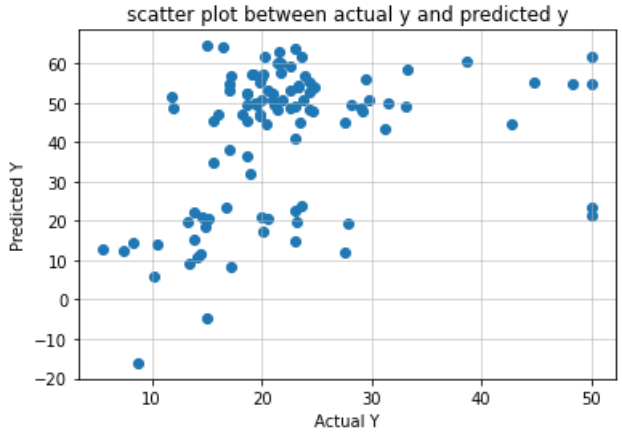
```
print('W[j+1] :', weight_sgd )
print('b[j+1] :', intercept_sgd )
```

```
W[j+1] : [-1.00132737e+00  9.95600160e-01  8.65082611e-04  7.34974995e-01
 -1.99042815e+00  1.95265621e+00 -5.56339997e-02 -2.61382905e+00
  2.13846077e+00 -1.78132803e+00 -2.12006519e+00  9.32669169e-01
 -3.67285574e+00]
b[j+1] : [22.27394321]
```

## Using Manual Regression for No Of Iterations = 10000

```
W_star,b_star=find_and_transform(manual_training,Y_Train,50,10000,0.1,0.25,0.01)
Y_Predicted=manual_predict(X_Test, W_star,b_star)
draw_scatterPlot(Y_Test,Y_Predicted)
manual_error=mean_squared_error(Y_Test,Y_Predicted)
print('Mean Square Error(MSE) Using Manual Approach =', manual_error)
```

```
**********************************
W[j+1] : [[-3.23708485 -4.76309083 -2.38928386  0.86176101 -1.53611454 -0.70770322
   3.66204276  2.23203837 -4.93908421 -4.1641031  -6.15103513  2.59545455
  -1.32903266]]
b[j+1] : [42.57705016]
**********************************
```




```
Mean Square Error(MSE) Using Manual Approach = 663.7141338124138
```

## COMPARISON TABLE

```
from prettytable import PrettyTable
x= PrettyTable()
x.field_names = ["Algorithm" , "Alpha", "No of Iterations","MSE", "B_Star"]
x.add_row(["SGD Regressor",0.0001,1,32.80,22.56])
x.add_row(["Manual SGD",0.0001,1,543.80,42.01])
x.add_row(["SGD Regressor",0.0001,100,36.58,22.36])
x.add_row(["Manual SGD",0.0001,100,953.80,44.98])
x.add_row(["SGD Regressor",0.0001,1000,33.49,22.37])
x.add_row(["Manual SGD",0.0001,1000,522.54,41.81])
x.add_row(["SGD Regressor",0.0001,10000,34.53,22.27])
x.add_row(["Manual SGD",0.0001,10000,663.71,42.57])
print(x)
```

```
+---------------+--------+------------------+--------+--------+
|   Algorithm   | Alpha  | No of Iterations |  MSE   | B_Star |
+---------------+--------+------------------+--------+--------+
| SGD Regressor | 0.0001 |        1         |  32.8  | 22.56  |
|   Manual SGD  | 0.0001 |        1         | 543.8  | 42.01  |
| SGD Regressor | 0.0001 |       100        | 36.58  | 22.36  |
|   Manual SGD  | 0.0001 |       100        | 953.8  | 44.98  |
| SGD Regressor | 0.0001 |       1000       | 33.49  | 22.37  |
|   Manual SGD  | 0.0001 |       1000       | 522.54 | 41.81  |
| SGD Regressor | 0.0001 |      10000       | 34.53  | 22.27  |
|   Manual SGD  | 0.0001 |      10000       | 663.71 | 42.57  |
+---------------+--------+------------------+--------+--------+
```

Looks like the Mean Square Error(MSE) getting decrease when No of Iterations are increasing. Also i h
50

Note : I have not reduced the r to r/2 for each iterations. I tried this approach but then the MSE and B_S
getting increased drastically.

```
1
```