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.preprocessing import StandardScaler
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
import sklearn
from sklearn.model_selection import train_test_split
```

In [2]:

```
from sklearn.model_selection import train_test_split
import sklearn

from sklearn.datasets import load_boston # to load  datasets from sklearn
import matplotlib.pyplot as plt

import numpy as np
import seaborn as sns

from collections import Counter
from sklearn.metrics import accuracy_score

from sklearn.preprocessing import StandardScaler
import pandas as pd
import math
```

In [3]:

```
boston = load_boston()
# Shape of Boston datasets
print(boston.data.shape)
```

```
(506, 13)
```

In [4]:

```
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 25,0
00 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds rive
r; 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 Car
negie 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 diagnost
ics
...', 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 Influenti
al Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning.
In Proceedings on the Tenth International Conference of Machine Learning,
236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [14]:

```
columnNames = boston.feature_names
print(columnNames)
Data = pd.DataFrame(boston.data, columns = columnNames)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [15]:

```
# real price values of bostan house datasets.
print(boston.target[:10])
```

```
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9]
```

In [16]:

```
Data_Labels = boston.target
Data_Labels.shape
Data["PRICE"] = Data_Labels
print(Data.shape)
print(Data.head(2))
```

```
(506, 14)
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0

   PTRATIO      B  LSTAT  PRICE
0     15.3  396.9   4.98   24.0
1     17.8  396.9   9.14   21.6
```

Replace nan values

In [17]:

```
Data.isnull().sum()
Data.isnull().values.any()
Data.apply(lambda x: x.fillna(x.mean()),axis=0)
```

Out[17]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.
10	0.22489	12.5	7.87	0.0	0.524	6.377	94.3	6.3467	5.0	311.0	15.2	392.
11	0.11747	12.5	7.87	0.0	0.524	6.009	82.9	6.2267	5.0	311.0	15.2	396.
12	0.09378	12.5	7.87	0.0	0.524	5.889	39.0	5.4509	5.0	311.0	15.2	390.
13	0.62976	0.0	8.14	0.0	0.538	5.949	61.8	4.7075	4.0	307.0	21.0	396.
14	0.63796	0.0	8.14	0.0	0.538	6.096	84.5	4.4619	4.0	307.0	21.0	380.
15	0.62739	0.0	8.14	0.0	0.538	5.834	56.5	4.4986	4.0	307.0	21.0	395.
16	1.05393	0.0	8.14	0.0	0.538	5.935	29.3	4.4986	4.0	307.0	21.0	386.
17	0.78420	0.0	8.14	0.0	0.538	5.990	81.7	4.2579	4.0	307.0	21.0	386.
18	0.80271	0.0	8.14	0.0	0.538	5.456	36.6	3.7965	4.0	307.0	21.0	288.
19	0.72580	0.0	8.14	0.0	0.538	5.727	69.5	3.7965	4.0	307.0	21.0	390.
20	1.25179	0.0	8.14	0.0	0.538	5.570	98.1	3.7979	4.0	307.0	21.0	376.
21	0.85204	0.0	8.14	0.0	0.538	5.965	89.2	4.0123	4.0	307.0	21.0	392.
22	1.23247	0.0	8.14	0.0	0.538	6.142	91.7	3.9769	4.0	307.0	21.0	396.
23	0.98843	0.0	8.14	0.0	0.538	5.813	100.0	4.0952	4.0	307.0	21.0	394.
24	0.75026	0.0	8.14	0.0	0.538	5.924	94.1	4.3996	4.0	307.0	21.0	394.
25	0.84054	0.0	8.14	0.0	0.538	5.599	85.7	4.4546	4.0	307.0	21.0	303.
26	0.67191	0.0	8.14	0.0	0.538	5.813	90.3	4.6820	4.0	307.0	21.0	376.
27	0.95577	0.0	8.14	0.0	0.538	6.047	88.8	4.4534	4.0	307.0	21.0	306.
28	0.77299	0.0	8.14	0.0	0.538	6.495	94.4	4.4547	4.0	307.0	21.0	387.
29	1.00245	0.0	8.14	0.0	0.538	6.674	87.3	4.2390	4.0	307.0	21.0	380.
...	...	...	...	...	...	...	...	...	...	...	...	
476	4.87141	0.0	18.10	0.0	0.614	6.484	93.6	2.3053	24.0	666.0	20.2	396.
477	15.02340	0.0	18.10	0.0	0.614	5.304	97.3	2.1007	24.0	666.0	20.2	349.
478	10.23300	0.0	18.10	0.0	0.614	6.185	96.7	2.1705	24.0	666.0	20.2	379.
479	14.33370	0.0	18.10	0.0	0.614	6.229	88.0	1.9512	24.0	666.0	20.2	383.
480	5.82401	0.0	18.10	0.0	0.532	6.242	64.7	3.4242	24.0	666.0	20.2	396.
481	5.70818	0.0	18.10	0.0	0.532	6.750	74.9	3.3317	24.0	666.0	20.2	393.

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
482	5.73116	0.0	18.10	0.0	0.532	7.061	77.0	3.4106	24.0	666.0	20.2	395.
483	2.81838	0.0	18.10	0.0	0.532	5.762	40.3	4.0983	24.0	666.0	20.2	392.
484	2.37857	0.0	18.10	0.0	0.583	5.871	41.9	3.7240	24.0	666.0	20.2	370.
485	3.67367	0.0	18.10	0.0	0.583	6.312	51.9	3.9917	24.0	666.0	20.2	388.
486	5.69175	0.0	18.10	0.0	0.583	6.114	79.8	3.5459	24.0	666.0	20.2	392.
487	4.83567	0.0	18.10	0.0	0.583	5.905	53.2	3.1523	24.0	666.0	20.2	388.
488	0.15086	0.0	27.74	0.0	0.609	5.454	92.7	1.8209	4.0	711.0	20.1	395.
489	0.18337	0.0	27.74	0.0	0.609	5.414	98.3	1.7554	4.0	711.0	20.1	344.
490	0.20746	0.0	27.74	0.0	0.609	5.093	98.0	1.8226	4.0	711.0	20.1	318.
491	0.10574	0.0	27.74	0.0	0.609	5.983	98.8	1.8681	4.0	711.0	20.1	390.
492	0.11132	0.0	27.74	0.0	0.609	5.983	83.5	2.1099	4.0	711.0	20.1	396.
493	0.17331	0.0	9.69	0.0	0.585	5.707	54.0	2.3817	6.0	391.0	19.2	396.
494	0.27957	0.0	9.69	0.0	0.585	5.926	42.6	2.3817	6.0	391.0	19.2	396.
495	0.17899	0.0	9.69	0.0	0.585	5.670	28.8	2.7986	6.0	391.0	19.2	393.
496	0.28960	0.0	9.69	0.0	0.585	5.390	72.9	2.7986	6.0	391.0	19.2	396.
497	0.26838	0.0	9.69	0.0	0.585	5.794	70.6	2.8927	6.0	391.0	19.2	396.
498	0.23912	0.0	9.69	0.0	0.585	6.019	65.3	2.4091	6.0	391.0	19.2	396.
499	0.17783	0.0	9.69	0.0	0.585	5.569	73.5	2.3999	6.0	391.0	19.2	395.
500	0.22438	0.0	9.69	0.0	0.585	6.027	79.7	2.4982	6.0	391.0	19.2	396.
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.

506 rows × 14 columns

Train_test_split

In [18]:

```
X_train, X_test, Y_train, Y_test = train_test_split(Data, Data["PRICE"],test_size = 0.2
)
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
print(X_train.head(10))
```

```
         CRIM     ZN  INDUS  CHAS    NOX     RM    AGE     DIS   RAD    TA
X  \
73    0.19539    0.0  10.81   0.0  0.413  6.245    6.2  5.2873   4.0  305.
0
45    0.17142    0.0   6.91   0.0  0.448  5.682   33.8  5.1004   3.0  233.
0
438  13.67810    0.0  18.10   0.0  0.740  5.935   87.9  1.8206  24.0  666.
0
46    0.18836    0.0   6.91   0.0  0.448  5.786   33.3  5.1004   3.0  233.
0
385  16.81180    0.0  18.10   0.0  0.700  5.277   98.1  1.4261  24.0  666.
0
222   0.62356    0.0   6.20   1.0  0.507  6.879   77.7  3.2721   8.0  307.
0
29    1.00245    0.0   8.14   0.0  0.538  6.674   87.3  4.2390   4.0  307.
0
319   0.47547    0.0   9.90   0.0  0.544  6.113   58.8  4.0019   4.0  304.
0
57    0.01432  100.0   1.32   0.0  0.411  6.816   40.5  8.3248   5.0  256.
0
405  67.92080    0.0  18.10   0.0  0.693  5.683  100.0  1.4254  24.0  666.
0

     PTRATIO       B  LSTAT  PRICE
73      19.2  377.17   7.54   23.4
45      17.9  396.90  10.21   19.3
438     20.2   68.95  34.02    8.4
46      17.9  396.90  14.15   20.0
385     20.2  396.90  30.81    7.2
222     17.4  390.39   9.93   27.5
29      21.0  380.23  11.98   21.0
319     18.4  396.23  12.73   21.0
57      15.1  392.90   3.95   31.6
405     20.2  384.97  22.98    5.0
```

In [19]:

```
Y_train.isnull().sum()
```

Out[19]:

```
0
```

Implementation of SGD

In [21]:

```
#Implementation of Stochastic Gradient Descent by taking 10 random samples
#First Standadize the data
from sklearn import preprocessing
scaler=preprocessing.StandardScaler()
std_scale = scaler.fit(X_train[['CRIM', 'ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RA
D','TAX','PTRATIO','B','LSTAT']])
train_standadized= std_scale.transform(X_train[['CRIM', 'ZN','INDUS','CHAS','NOX','RM',
'AGE','DIS','RAD','TAX','PTRATIO','B','LSTAT']])
test_standadized= std_scale.transform(X_test[['CRIM','ZN','INDUS','CHAS','NOX','RM','AG
E','DIS','RAD','TAX','PTRATIO','B','LSTAT']])
```

In [22]:

```
X_train_standadized = pd.DataFrame(train_standadized, columns = columnNames)
X_test_standadized = pd.DataFrame(test_standadized, columns = columnNames)
Y_train=np.array(Y_train)
Y_test=np.array(Y_test)
print(Y_train)
print(Y_train.shape)
print(X_train_standadized.shape)
```

```
[23.4 19.3  8.4 20.   7.2 27.5 21.  21.  31.6  5.  14.1 33.4 19.  21.5
 22.9 29.1 19.1 29.8 18.4 17.8 19.5 23.1 20.3 29.4 24.1 46.7 27.9 29.
 19.4 18.4 24.   7.2 21.4 50.   6.3 50.  10.2 14.1 30.7 34.9 22.6 28.7
 19.8 50.  16.5 10.9 21.6 19.4 20.6 23.8 19.8 19.9 15.  26.5 20.4 23.1
 22.  22.  11.3 23.8 14.6 20.2 32.  24.3 25.1 22.  12.7 16.6 26.4 23.6
 19.2  8.8 24.6 19.1 28.4 37.2 22.2 24.5 18.5 21.5 30.1 13.4 25.  20.8
 12.6 21.2 18.1 13.1 50.  33.  10.2 14.3 20.4 28.1 16.8 20.9 34.6 11.7
 17.4 23.1 18.3 11.9 10.8 48.3 24.8 36.2 19.9 21.4 41.7 21.9 50.  17.2
 18.9 17.8 24.8 16.1 32.5 50.  15.1 38.7 17.2 14.8 14.5 32.7 25.  22.
 22.4 30.5 27.1 24.4  9.5 21.1 24.8 45.4 23.5 36.5 17.8 18.3 23.1 15.6
 17.3  9.7  8.3 14.  20.9 20.  22.9 21.2 35.4 20.1 31.7 34.9  5.  15.
 32.2 16.7 17.2 16.6 13.4 26.2 16.  36.  14.4 18.9 19.1 20.1 20.6 13.8
 18.  20.8 28.6 22.7 19.4 13.6 24.4 13.8 23.8 21.2 26.6 33.1 19.4 14.6
 17.5 21.7 15.  25.3 20.4  8.7 29.6 26.6 42.8 37.  24.6  7.  18.8 10.5
 27.9 19.3 21.1 23.3 19.4 21.8 16.2 24.7 44.8 23.3 22.  15.2 50.  19.5
 21.9 27.5 16.7 48.8 23.7 36.4 20.  29.1 12.8 37.6 24.5 23.  19.6 17.7
 26.4 19.6 21.9 23.2 21.8 23.9 43.8 22.5 37.9 11.5 13.1 13.4 24.3 33.2
 24.2 17.5 23.2 25.  13.3 23.  50.  26.7 15.6 33.2 16.8 11.9 28.2 19.5
  7.  30.1 23.2 50.  11.8 13.3  8.4 50.  22.8 21.4 22.2 21.4 19.3 13.8
 18.6 20.4 18.7 28.4 34.7 13.5 26.6 31.5 15.2  7.2 20.6 13.3 33.3 24.3
 17.8 14.9 20.8 27.1 21.4 50.  20.5 23.3 33.4 13.8 17.1 18.5 19.2 24.7
 23.9 25.  21.7 43.1 20.5 30.1 15.7 17.8 37.3 13.8 23.  22.4  7.4 23.6
 22.6 39.8 12.  50.  44.  24.1 19.  35.1  8.5 10.4 14.5 18.2  7.5 21.7
 17.9 23.1 24.1 20.  23.2 41.3 16.1 31.   8.3 36.1 32.9 21.7 32.  22.6
 16.5 23.9 19.6 33.1 22.9 50.  15.3 20.3 13.4 29.6 48.5 18.6 23.7 19.1
 12.7 20.6 28.5 18.2 17.5 21.2 46.  15.6 18.7 20.1 10.2 24.4 11.  27.5
 50.  23.8 18.5 19.9 12.5 14.9 36.2 29.9 17.6 16.2 18.2 23.3 18.4 23.7
 24.4 13.1 12.3 18.8 20.6 23.  15.6 13.1 19.7 33.8 19.5 17.4 31.2 12.1
  8.8 20.3 29.  22.8 20.1 42.3 15.2 22.3 23.7 19.6 24.7 31.1]
(404,)
(404, 13)
```

In [23]:

```
X_train_standadized['PRICE']=Y_train
X_test_standadized['PRICE']=Y_test
```

In [24]:

```
print(X_train_standadized.shape)
print(X_test_standadized.shape)
```

```
(404, 14)
(102, 14)
```

In [25]:

```
X_train_standadized.isnull().sum()
```

Out[25]:

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

SGD

In [28]:

```
# for references
#https://github.com/gauravtheP/Implementing-Stochastic-GradientDescent/blob/master/LinearRegression_SGD_BostonHomePrices.ipynb
#First step initilize the weights and b
#formulae of slope s=mx+b.
# mx is the weights*x1....weights_d*xd
# b is the intercept term
m = X_train.shape[0]
weight = np.random.randn(13)*np.sqrt(2/m) # defining initial random weight from normal
 distribution
b = np.random.randn(1)*np.sqrt(2/m) # generating initial random y-intercept from normal distribution
# initilize learing rate
learningRate = 0.2
print(m,weight,b,learningRate)
for i in range(2000): # running 2000 iterations
    Data_batch_10 = X_train_standadized.sample(n = 10) # taking 10 stochastic samples
    X_temp = Data_batch_10.drop("PRICE", axis = 1, inplace = False) # DROP the price label, because this is the output label we have to predict.
#X = pd.DataFrame(X_temp, columns = columnNames)
    X=X_temp
    Y = Data_batch_10["PRICE"]
    PartialGradient = np.empty(13)# in this we store the partial derivate with respect
 to w...we have 13 features  13 features
    sum2 = 0
# Update the weights-------------------------------
# formula (w0=w1-lr*derivate)in every iteration
# step 1.
#First calculate the derivative
    for j in range(13): # as there are 13 dimensions in our dataset and dimensions of weight should also be same as dimension of our dataset
        sum1 = 0
        for k in range(10):
            sum1 += -2 * X.iloc[k][j] * (Y.iloc[k] - np.dot(weight, X.iloc[k]) - b) # this is a derivative of linear regression w.r.t 'w
        PartialGradient[j] = sum1
# step 2.
#multiply with learning rate
    PartialGradient *= learningRate
#step 3.
#Update the weights
    for l in range(13):
        weight[l] -= PartialGradient[l] # updating weights
# Update the Intercepts or (b's)-------------------------------
    for m in range(10):
        sum2 += -2 * (Y.iloc[m]- np.dot(weight, X.iloc[m]) - b) # this is the derivative of linear regression w.r.t 'b'
    b = b - learningRate * sum2 #updating y-intercept 'b'
# in every iteration u have to reduce the learing rate bro
    learningRate = 0.01 / pow(i+1, 0.25) #learning rate at every iteration
# just add the regularization term to it
    weight = weight + 0.0001*np.dot(weight, weight) #adding l2 regularization
    b = b + 0.0001*np.dot(weight, weight) #adding L2 regularization
print("Weight = "+str(weight))
print("b = "+str(b))
```

```
404 [ 0.06860788  0.08899781 -0.13299221  0.01283731  0.13891196  0.060981
7
 -0.06443528  0.00046115 -0.10548441  0.03947253  0.05610601 -0.01364426
 -0.1491161 ] [0.10787514] 0.2
Weight = [-0.81797771  1.36606222  0.61476667  0.63295946 -1.82274377  3.5
5709996
  0.04545461 -1.86569186  2.46296413 -2.41683874 -1.94809464  1.25010071
 -2.71808609]
b = [23.0927888]
```

In [29]:

```
# time for testdata.. with our updated weights and coeffcients
import math
test_temp = X_test_standadized.drop("PRICE", axis = 1, inplace = False)
test_data = test_temp
test_labels = Y_test
y_predicted = []
for i in range(102):
    test_i = 0
    test_i = np.dot(weight, test_data.iloc[i]) + b[0] #making prediction by using optim
ize values of weights obtained from SGD
    y_predicted.append(test_i)
```

In [30]:

```
#Make the preditions
d1 = {'True Labels': Y_test, 'Predicted Labels': y_predicted}
df1 = pd.DataFrame(data = d1)
```

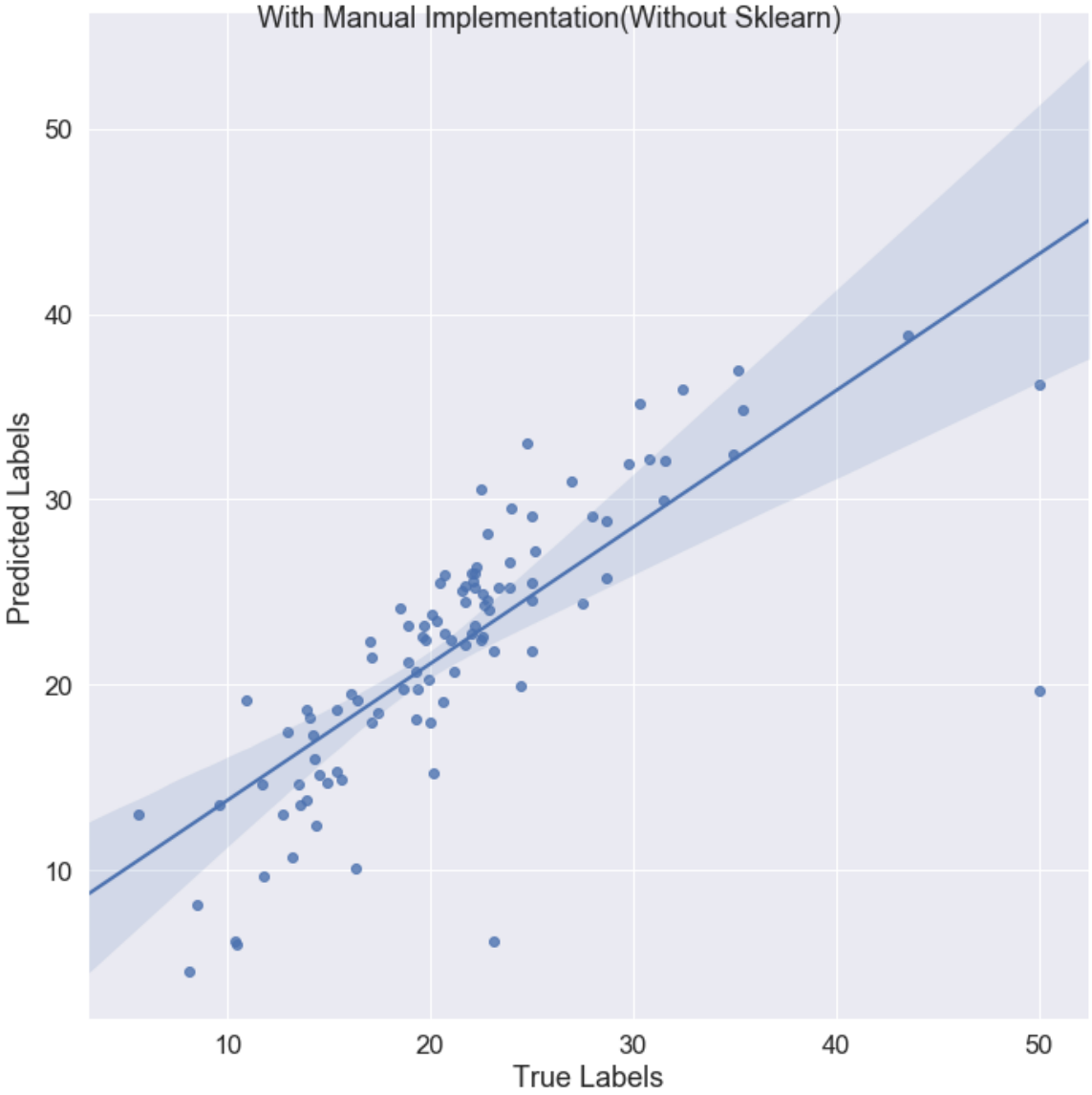
In [31]:

```
Mean_Sq_Error = mean_squared_error(Y_test, y_predicted)
print(Mean_Sq_Error)
```

```
24.398601338356574
```

In [37]:

```
import seaborn as sns
lm1 = sns.lmplot(x="True Labels", y="Predicted Labels", data = df1, height = 10)
fig1 = lm1.fig
fig1.suptitle("With Manual Implementation(Without Sklearn)", fontsize=18)
sns.set(font_scale = 1.5)
```



In [38]:

```
#Sklearn implementation
X_temp = X_train_standadized.drop("PRICE", axis = 1, inplace = False)
X=X_temp
Y = Y_train
X_test_temp = X_test_standadized.drop("PRICE", axis = 1, inplace = False)
X_te=X_test_temp
Y_te = Y_test
clf = SGDRegressor(shuffle = False, learning_rate= 'invscaling', max_iter = 2000)
clf.fit(X, Y)# fir train data
Y_pred = clf.predict(X_te)# predict test error
print("Weight = "+str(clf.coef_))
print("Y Intercept = "+str(clf.intercept_))
```

```
Weight = [-0.90939854  1.02064702  0.25488595  0.80984814 -2.04099089  3.2
2020808
 -0.22713055 -2.9395672   2.20830125 -1.67917236 -2.21350137  0.83330622
 -3.38823594]
Y Intercept = [22.8789032]
```

In [39]:

```
d2 = {'True Labels': Y_te, 'Predicted Labels': Y_pred}
df2 = pd.DataFrame(data = d2)
df2
```

Out[39]:

	True Labels	Predicted Labels
0	14.5	13.597150
1	50.0	21.909100
2	20.1	20.693969
3	21.7	24.377509
4	14.3	17.060124
5	22.1	26.585704
6	5.6	12.043729
7	22.8	28.415784
8	35.2	36.687352
9	21.0	22.785727
10	18.9	19.364951
11	19.7	21.584813
12	20.5	24.010135
13	22.2	24.519807
14	8.5	7.526514
15	22.3	27.027855
16	21.6	25.248728
17	14.1	19.094039
18	22.2	22.054486
19	13.9	18.119375
20	19.8	22.444012
21	25.0	22.189365
22	10.9	18.638962
23	15.6	15.928616
24	21.7	24.867301
25	50.0	36.364691
26	13.0	17.268156
27	28.7	25.138429
28	16.4	19.232039
29	22.0	25.895932
...	...	...
72	22.7	24.857373
73	23.9	24.934855
74	22.5	22.235369
75	20.7	26.013061
76	28.0	28.639048
77	16.1	21.415995

	True Labels	Predicted Labels
78	25.0	24.595126
79	31.6	32.418795
80	25.0	29.066595
81	22.5	29.656035
82	20.3	22.260078
83	20.2	15.853107
84	43.5	39.812799
85	12.7	11.399979
86	22.9	23.260220
87	24.0	30.147889
88	28.7	28.468631
89	30.8	31.361323
90	22.2	25.618825
91	23.1	22.411726
92	19.3	17.528075
93	14.2	17.696257
94	34.9	30.993969
95	20.6	19.424015
96	11.8	8.988089
97	19.3	21.721836
98	22.6	22.836175
99	13.9	13.045792
100	10.5	5.492323
101	24.5	20.701567

102 rows × 2 columns

In [40]:

```
Mean_Sq_Error = mean_squared_error(Y_te, Y_pred)
Mean_Sq_Error
```

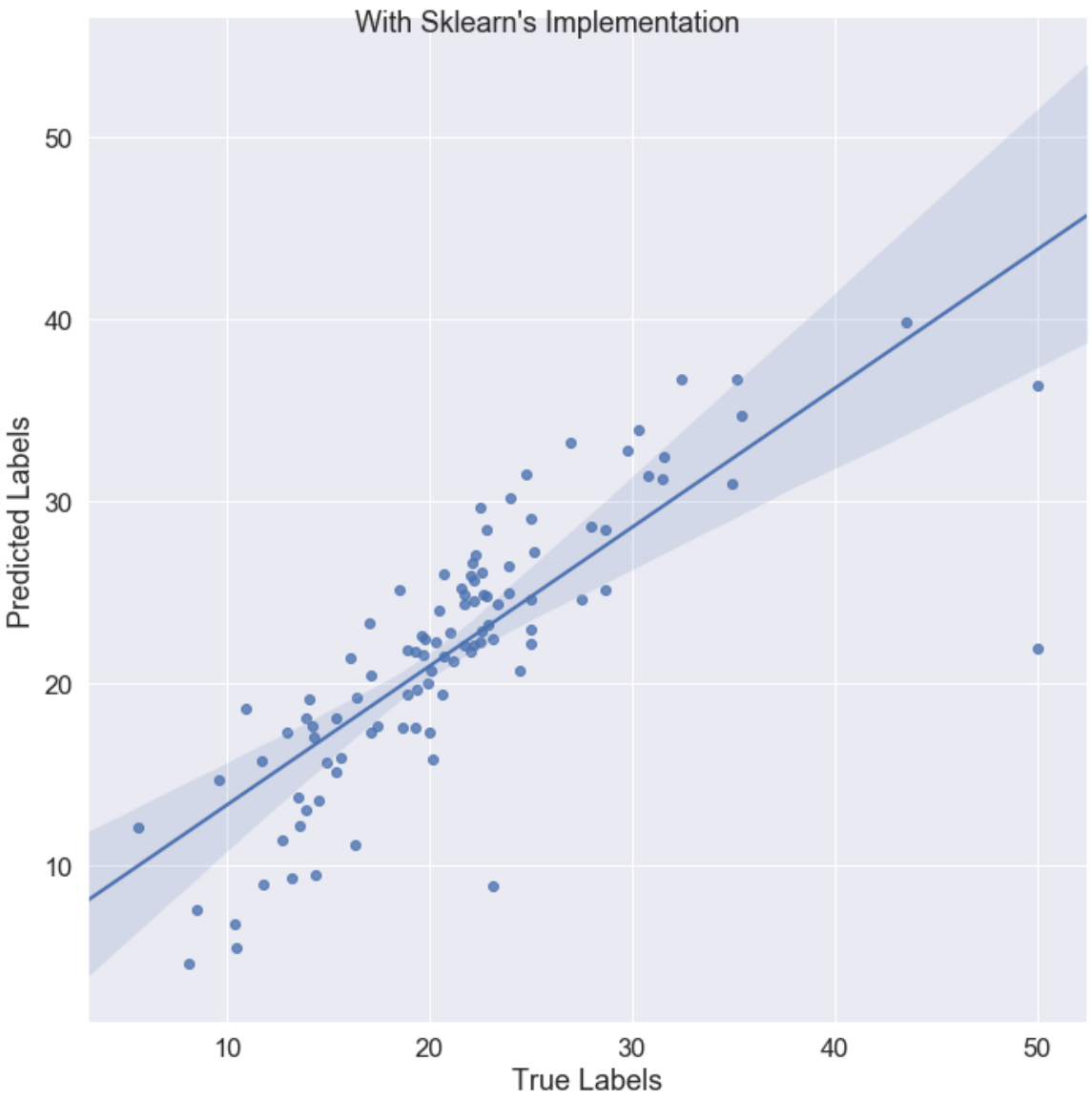
Out[40]:

```
22.22266923710572
```

In [36]:

```
lm2 = sns.lmplot(x="True Labels", y="Predicted Labels", data = df2, size = 10)
fig2 = lm2.fig
# Add a title to the Figure
fig2.suptitle("With Sklearn's Implementation", fontsize=18)
sns.set(font_scale = 1.5)
```

```
C:\anaconda\lib\site-packages\seaborn\regression.py:546: UserWarning: The
`size` paramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



Result:

My SGD impementation error- > 24.39 Sklearn SGD impementation error- > 22.22