Author : Saurabh Singhai

Table of Contents

Import Libraries

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
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
from sklearn.model_selection import train_test_split
```

Load Dataset

In [2]:

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

```
(506, 13)
```

In [3]:

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

```
Boston House Prices dataset
===========================

Notes
```

```
------
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.
http://archive.ics.uci.edu/ml/datasets/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.

**References**

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of C
ollinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the T
enth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst.
Morgan Kaufmann.
   - many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)
```

In [4]:

```
boston.values()
```

Out[4]:

```
dict_values([array([[  6.32000000e-03,   1.80000000e+01,   2.31000000e+00, ...,
          1.53000000e+01,   3.96900000e+02,   4.98000000e+00],
       [  2.73100000e-02,   0.00000000e+00,   7.07000000e+00, ...,
          1.78000000e+01,   3.96900000e+02,   9.14000000e+00],
       [  2.72900000e-02,   0.00000000e+00,   7.07000000e+00, ...,
          1.78000000e+01,   3.92830000e+02,   4.03000000e+00],
       ...,
       [  6.07600000e-02,   0.00000000e+00,   1.19300000e+01, ...,
          2.10000000e+01,   3.96900000e+02,   5.64000000e+00],
       [  1.09590000e-01,   0.00000000e+00,   1.19300000e+01, ...,
          2.10000000e+01,   3.93450000e+02,   6.48000000e+00],
       [  4.74100000e-02,   0.00000000e+00,   1.19300000e+01, ...,
          2.10000000e+01,   3.96900000e+02,   7.88000000e+00]]), array([ 24. ,  21.6,  34.7,  33.4,
36.2,  28.7,  22.9,  27.1,  16.5,
        18.9,  15. ,  18.9,  21.7,  20.4,  18.2,  19.9,  23.1,  17.5,
        20.2,  18.2,  13.6,  19.6,  15.2,  14.5,  15.6,  13.9,  16.6,
```

```
        14.8,  18.4,  21. ,  12.7,  14.5,  13.2,  13.1,  13.5,  18.9,
        20. ,  21. ,  24.7,  30.8,  34.9,  26.6,  25.3,  24.7,  21.2,
        19.3,  20. ,  16.6,  14.4,  19.4,  19.7,  20.5,  25. ,  23.4,
        18.9,  35.4,  24.7,  31.6,  23.3,  19.6,  18.7,  16. ,  22.2,
        25. ,  33. ,  23.5,  19.4,  22. ,  17.4,  20.9,  24.2,  21.7,
        22.8,  23.4,  24.1,  21.4,  20. ,  20.8,  21.2,  20.3,  28. ,
        23.9,  24.8,  22.9,  23.9,  26.6,  22.5,  22.2,  23.6,  28.7,
        22.6,  22. ,  22.9,  25. ,  20.6,  28.4,  21.4,  38.7,  43.8,
        33.2,  27.5,  26.5,  18.6,  19.3,  20.1,  19.5,  19.5,  20.4,
        19.8,  19.4,  21.7,  22.8,  18.8,  18.7,  18.5,  18.3,  21.2,
        19.2,  20.4,  19.3,  22. ,  20.3,  20.5,  17.3,  18.8,  21.4,
        15.7,  16.2,  18. ,  14.3,  19.2,  19.6,  23. ,  18.4,  15.6,
        18.1,  17.4,  17.1,  13.3,  17.8,  14. ,  14.4,  13.4,  15.6,
        11.8,  13.8,  15.6,  14.6,  17.8,  15.4,  21.5,  19.6,  15.3,
        19.4,  17. ,  15.6,  13.1,  41.3,  24.3,  23.3,  27. ,  50. ,
        50. ,  50. ,  22.7,  25. ,  50. ,  23.8,  23.8,  22.3,  17.4,
        19.1,  23.1,  23.6,  22.6,  29.4,  23.2,  24.6,  29.9,  37.2,
        39.8,  36.2,  37.9,  32.5,  26.4,  29.6,  50. ,  32. ,  29.8,
        34.9,  37. ,  30.5,  36.4,  31.1,  29.1,  50. ,  33.3,  30.3,
        34.6,  34.9,  32.9,  24.1,  42.3,  48.5,  50. ,  22.6,  24.4,
        22.5,  24.4,  20. ,  21.7,  19.3,  22.4,  28.1,  23.7,  25. ,
        23.3,  28.7,  21.5,  23. ,  26.7,  21.7,  27.5,  30.1,  44.8,
        50. ,  37.6,  31.6,  46.7,  31.5,  24.3,  31.7,  41.7,  48.3,
        29. ,  24. ,  25.1,  31.5,  23.7,  23.3,  22. ,  20.1,  22.2,
        23.7,  17.6,  18.5,  24.3,  20.5,  24.5,  26.2,  24.4,  24.8,
        29.6,  42.8,  21.9,  20.9,  44. ,  50. ,  36. ,  30.1,  33.8,
        43.1,  48.8,  31. ,  36.5,  22.8,  30.7,  50. ,  43.5,  20.7,
        21.1,  25.2,  24.4,  35.2,  32.4,  32. ,  33.2,  33.1,  29.1,
        35.1,  45.4,  35.4,  46. ,  50. ,  32.2,  22. ,  20.1,  23.2,
        22.3,  24.8,  28.5,  37.3,  27.9,  23.9,  21.7,  28.6,  27.1,
        20.3,  22.5,  29. ,  24.8,  22. ,  26.4,  33.1,  36.1,  28.4,
        33.4,  28.2,  22.8,  20.3,  16.1,  22.1,  19.4,  21.6,  23.8,
        16.2,  17.8,  19.8,  23.1,  21. ,  23.8,  23.1,  20.4,  18.5,
        25. ,  24.6,  23. ,  22.2,  19.3,  22.6,  19.8,  17.1,  19.4,
        22.2,  20.7,  21.1,  19.5,  18.5,  20.6,  19. ,  18.7,  32.7,
        16.5,  23.9,  31.2,  17.5,  17.2,  23.1,  24.5,  26.6,  22.9,
        24.1,  18.6,  30.1,  18.2,  20.6,  17.8,  21.7,  22.7,  22.6,
        25. ,  19.9,  20.8,  16.8,  21.9,  27.5,  21.9,  23.1,  50. ,
        50. ,  50. ,  50. ,  50. ,  13.8,  13.8,  15. ,  13.9,  13.3,
        13.1,  10.2,  10.4,  10.9,  11.3,  12.3,   8.8,   7.2,  10.5,
         7.4,  10.2,  11.5,  15.1,  23.2,   9.7,  13.8,  12.7,  13.1,
        12.5,   8.5,   5. ,   6.3,   5.6,   7.2,  12.1,   8.3,   8.5,
         5. ,  11.9,  27.9,  17.2,  27.5,  15. ,  17.2,  17.9,  16.3,
         7. ,   7.2,   7.5,  10.4,   8.8,   8.4,  16.7,  14.2,  20.8,
        13.4,  11.7,   8.3,  10.2,  10.9,  11. ,   9.5,  14.5,  14.1,
        16.1,  14.3,  11.7,  13.4,   9.6,   8.7,   8.4,  12.8,  10.5,
        17.1,  18.4,  15.4,  10.8,  11.8,  14.9,  12.6,  14.1,  13. ,
        13.4,  15.2,  16.1,  17.8,  14.9,  14.1,  12.7,  13.5,  14.9,
        20. ,  16.4,  17.7,  19.5,  20.2,  21.4,  19.9,  19. ,  19.1,
        19.1,  20.1,  19.9,  19.6,  23.2,  29.8,  13.8,  13.3,  16.7,
        12. ,  14.6,  21.4,  23. ,  23.7,  25. ,  21.8,  20.6,  21.2,
        19.1,  20.6,  15.2,   7. ,   8.1,  13.6,  20.1,  21.8,  24.5,
        23.1,  19.7,  18.3,  21.2,  17.5,  16.8,  22.4,  20.6,  23.9,
        22. ,  11.9]), array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'],
      dtype='<U7'), "Boston House Prices dataset\n===========================\n\nNotes\n------\nDat
a Set Characteristics:  \n\n    :Number of Instances: 506 \n\n    :Number of Attributes: 13 numeric
/categorical predictive\n    \n    :Median Value (attribute 14) is usually the target\n\n    :Attri
bute Information (in order):\n        - CRIM     per capita crime rate by town\n        - ZN
proportion of residential land zoned for lots over 25,000 sq.ft.\n        - INDUS    proportion of
non-retail business acres per town\n        - CHAS     Charles River dummy variable (= 1 if tract b
ounds river; 0 otherwise)\n        - NOX      nitric oxides concentration (parts per 10 million)\n
- RM       average number of rooms per dwelling\n        - AGE      proportion of owner-occupied un
its built prior to 1940\n        - DIS      weighted distances to five Boston employment centres\n
- RAD      index of accessibility to radial highways\n        - TAX      full-value property-tax ra
te per $10,000\n        - PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^
2 where Bk is the proportion of blacks by town\n        - LSTAT    % lower status of the populatior
\n        - MEDV     Median value of owner-occupied homes in $1000's\n\n    :Missing Attribute Valu
es: None\n\n    :Creator: Harrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing
dataset.\nhttp://archive.ics.uci.edu/ml/datasets/Housing\n\n\nThis dataset was taken from the Stat
Lib library which is maintained at Carnegie Mellon University.\n\nThe Boston house-price data of H
arrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the demand for clean air', J. Environ. Econom
ics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression
diagnostics\n...', Wiley, 1980.   N.B. Various transformations are used in the table on\npages
244-261 of the latter.\n\nThe Boston house-price data has been used in many machine learning paper
s that address regression\nproblems.   \n     \n**References**\n\n   - Belsley, Kuh & Welsch, 'Regr
ession diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-26
```

```
1.\n   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on t
he Tenth International Conference of Machine Learning, 236-243, University of Massachusetts,
Amherst. Morgan Kaufmann.\n   - many more! (see
http://archive.ics.uci.edu/ml/datasets/Housing)\n"])
```

In [5]:

```
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 [6]:

```
Data.head(5)
```

Out[6]:

	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
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
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

Read Target Coloumn

In [7]:

```
print(boston.target[:10])
```

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

In [8]:

```
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
```

Find Null Values

In [9]:

```
Data.isnull().sum()
```

Out[9]:

```
CRIM       0
ZN         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
```

Train_test_split

In [10]:

```
X_train, X_test, Y_train, Y_test = train_test_split(Data, Data["PRICE"],test_size = 0.2)

print(X_train.head(10))
```

```
         CRIM    ZN  INDUS  CHAS    NOX     RM    AGE     DIS  RAD    TAX  \
221   0.40771   0.0   6.20   1.0  0.507  6.164   91.3  3.0480  8.0  307.0
116   0.13158   0.0  10.01   0.0  0.547  6.176   72.5  2.7301  6.0  432.0
74    0.07896   0.0  12.83   0.0  0.437  6.273    6.0  4.2515  5.0  398.0
337   0.03041   0.0   5.19   0.0  0.515  5.895   59.6  5.6150  5.0  224.0
361   3.83684   0.0  18.10   0.0  0.770  6.251   91.1  2.2955 24.0  666.0
504   0.10959   0.0  11.93   0.0  0.573  6.794   89.3  2.3889  1.0  273.0
402   9.59571   0.0  18.10   0.0  0.693  6.404  100.0  1.6390 24.0  666.0
298   0.06466  70.0   2.24   0.0  0.400  6.345   20.1  7.8278  5.0  358.0
438  13.67810   0.0  18.10   0.0  0.740  5.935   87.9  1.8206 24.0  666.0
211   0.37578   0.0  10.59   1.0  0.489  5.404   88.6  3.6650  4.0  277.0

     PTRATIO       B  LSTAT  PRICE
221     17.4  395.24  21.46   21.7
116     17.8  393.30  12.04   21.2
74      18.7  394.92   6.78   24.1
337     20.2  394.81  10.56   18.5
361     20.2  350.65  14.19   19.9
504     21.0  393.45   6.48   22.0
402     20.2  376.11  20.31   12.1
298     14.8  368.24   4.97   22.5
438     20.2   68.95  34.02    8.4
211     18.6  395.24  23.98   19.3
```

In [11]:

```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[11]:

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

Standardize the data

In [12]:

```
#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','RAD','TAX','PTR
ATIO','B','LSTAT']])
train_standardized= std_scale.transform(X_train[['CRIM',
'ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PTRATIO','B','LSTAT']])
test_standardized= std_scale.transform(X_test[['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','
RAD','TAX','PTRATIO','B','LSTAT']])
```

In [13]:

```
X_train_standardized = pd.DataFrame(train_standardized, columns = columnNames)
X_test_standardized = pd.DataFrame(test_standardized, columns = columnNames)
Y_train=np.array(Y_train)
Y_test=np.array(Y_test)

print("="*50)
print(Y_train)
print("="*50)
print(Y_train.shape)
print("="*50)
print(X_train_standardized.shape)
```

```
==================================================
[ 21.7  21.2  24.1  18.5  19.9  22.   12.1  22.5   8.4  19.3  20.4  24.5
  18.3  50.   50.   23.8  19.9  20.8  28.4  21.5  17.5  21.9  12.7  23.2
  22.9  15.6  23.    7.4  29.   37.6  19.6  43.1  10.2  19.4  29.8  21.4
  11.9  22.2  21.8  11.7  27.9  17.8   5.   20.6  33.3  20.5  50.   10.2
  18.3  31.5  19.7  17.   23.3  16.6  28.7  15.3  19.3  20.6  22.1  43.5
  23.8  25.   20.4  17.8  10.5  19.4  17.6  24.8  42.3  17.5  13.3  18.7
  29.    9.7  18.2  22.   35.2  20.7  18.4  14.4  30.1  22.   19.3  29.4
  21.   34.7  13.3  31.6  13.1  30.1  17.2  46.   35.4  16.8  32.2  22.
  25.   50.   20.1  19.4  18.7  22.2  19.6  21.1  21.   28.1  29.9  19.5
  18.4   7.   36.1  21.7  17.4  30.1   7.2  48.5  30.3  41.3  32.5  22.2
  21.7  14.6  24.7  22.6  19.5  19.5  19.8   6.3  20.   25.    5.   23.1
  23.6  11.5  17.2  31.1  13.4  10.5  20.8  20.4  19.6  11.3  15.   34.9
  31.7  26.7  17.8  20.3  11.9  13.4  21.8  20.2  16.1  15.4  18.   19.1
  21.1  23.9   8.4   8.8  19.3  31.5  19.1  14.3  25.   16.1  14.9  14.1
  20.9  15.1  10.8  29.1  23.2  22.9  15.6  22.   32.   28.   15.2  18.5
  21.2  21.6  27.5  21.9  41.7  23.4  16.8  32.7  28.2  12.8  22.5  23.1
  23.9  44.8  14.5  12.3  13.8  23.1  21.7  24.   13.2   7.5  18.9  23.7
  10.9  16.2  13.9  19.6  18.9  15.6  16.4  34.9  25.   36.4  12.7  24.7
  20.6  24.3  19.4  21.2  18.9  17.2  20.   13.8  21.7  22.9  26.6  20.5
   9.5  23.7  18.9  16.   24.3  13.4  33.1  33.4  16.7  22.8  22.2  23.9
  50.   17.1  21.7  25.2  19.6  15.2  13.8  37.9   8.7  33.   17.5  20.6
  17.8  33.2  13.5  10.2  21.5  15.2  26.5  11.8  38.7  14.3  36.2  24.8
  22.7  13.8  26.6  27.    8.5  16.7  13.   18.2  19.5  27.9  33.2  48.8
  20.   24.   26.2  20.3  50.   17.1  28.6  23.3  27.5  19.1  29.1  20.1
  12.5  23.8  23.1  15.6  19.8  20.    7.   25.    8.1  15.6  23.2  18.7
  13.5  19.2  22.2  26.6  50.   20.8  22.8  19.   14.9  27.5  24.1  23.3
  20.1  23.1  21.7  15.4   5.6  25.1  22.9  24.4  10.9  34.6  18.6  23.6
  14.5  11.8  35.1  28.4  10.4   8.3  16.1  13.3  32.4  24.8  22.6  22.8
  21.6  24.8  21.9  23.2  43.8  22.6  29.8  23.   23.1  30.8  18.8  16.5
  34.9  13.9  14.4  20.3  13.4  11.   15.7  21.2  22.4  19.7  50.    7.2
  23.5  20.6  21.2  18.8  24.4  14.1  50.   24.1  22.4  24.4  22.3  23.9
  22.   22.7  17.4  24.6  21.4  36.2  50.   35.4  44.   24.5  20.7  23.
  50.   18.5  21.4  33.4  24.7  24.4  36.   32.   14.   16.2  23.9  12.6
  19.1  29.6  17.7  14.8  28.7  17.3  45.4  14.6]
==================================================
(404,)
==================================================
(404, 13)
```

In [14]:

```
X_train_standardized['PRICE']=Y_train
X_test_standardized['PRICE']=Y_test
```

In [15]:

```
print(X_train_standardized.shape)
print(X_test_standardized.shape)
```

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

Implement SGD

In [16]:

```
# 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 distributio
n

# initilize learing rate
learningRate = 0.2
print(m,weight,b,learningRate)


for i in range(2000): # running 2000 iterations
    Data_batch_10 = X_train_standardized.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=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 r
egression w.r.t 'b'
    b = b - learningRate * sum2 #updating y-intercept 'b'
# in every iteration u have to reduce the learing rate
    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 [ 0.02227997  0.12424859  0.07334289  0.13013477  0.0360016   0.04797156
  0.04805543 -0.00959031  0.10509333  0.00672592 -0.09315377 -0.12427005
  0.11633426] [-0.05379845] 0.2
Weight = [-0.40759152  1.08887901  0.43421475  0.89736121 -1.83728164  2.97616813
  0.6330621  -2.16034948  2.24545928 -2.26065628 -1.81281784  1.29698828
 -3.60069238]
b = [ 22.09991046]
```

In [17]:

```
# time for testdata.. with our updated weights and coeffcients
import math
test_temp = X_test_standardized.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 optimize values o
f weights obtained from SGD
    y_predicted.append(test_i)
```

In [18]:

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

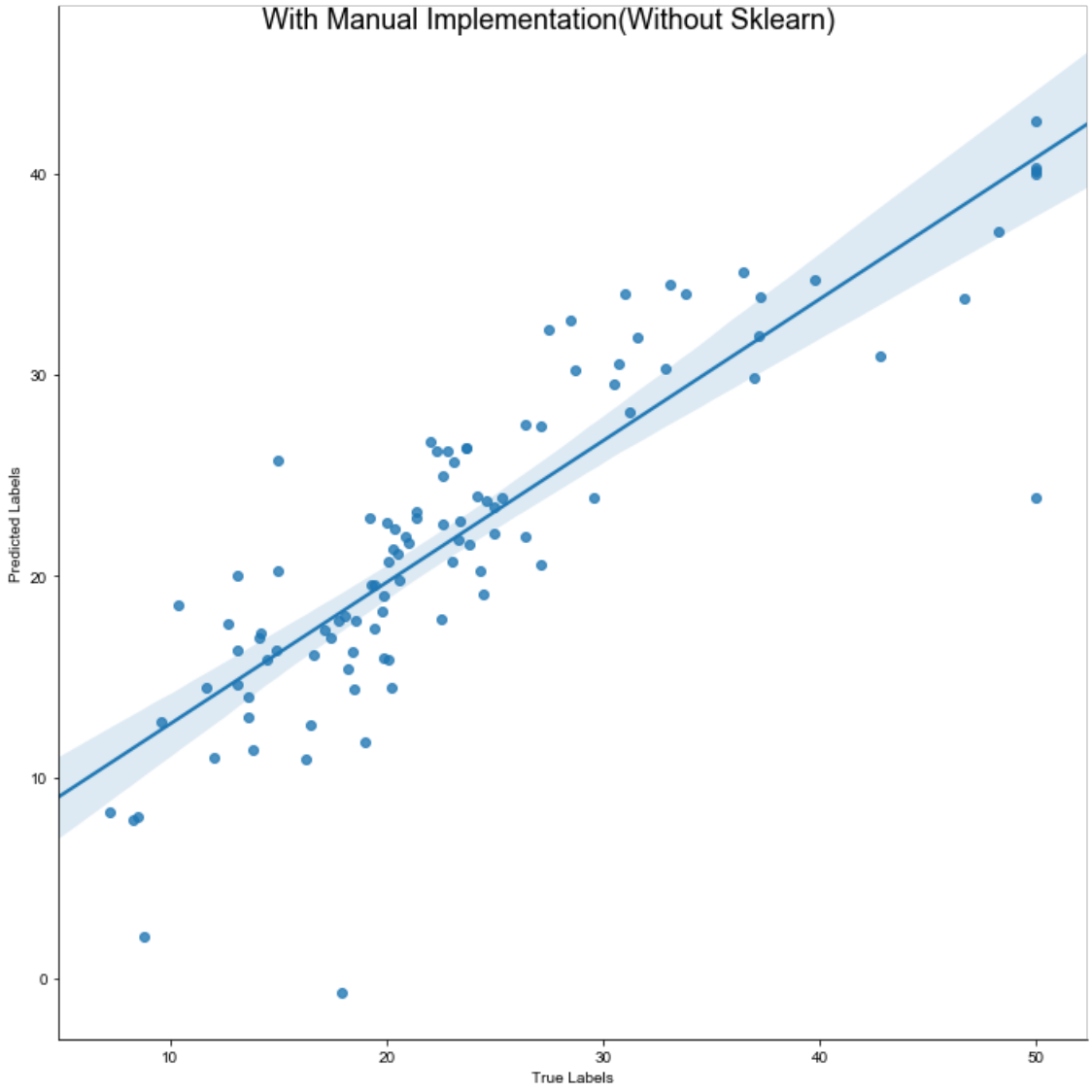
In [19]:

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

```
28.2303459746
```

In [20]:

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



Use SKLearn SGD

In [21]:

```
#Sklearn implementation
X_temp = X_train_standardized.drop("PRICE", axis = 1, inplace = False)
X=X_temp
Y = Y_train
X_test_temp = X_test_standardized.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 = [-0.61173094  1.00475583  0.13724696  0.88966024 -2.01484363  2.30659191
  0.05914557 -3.12641841  2.39890687 -1.77518791 -2.0349143   0.90127311
 -4.13311225]
Y Intercept = [ 22.3107813]
```

In [22]:

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

Out[22]:

	Predicted Labels	True Labels
0	23.815617	19.2
1	27.231606	22.0
2	26.614621	22.3
3	11.623463	12.0
4	33.630880	37.3
5	21.399262	20.6
6	32.442752	37.2
7	30.454901	30.7
8	22.459225	20.3
9	17.094959	17.1
10	35.311102	36.5
11	26.978603	23.7
12	20.059651	20.5
13	20.148670	23.0
14	17.092589	18.1
15	28.763733	42.8
16	13.990801	13.1
17	14.195747	9.6
18	12.824939	18.5
19	11.410011	13.8
20	26.981820	23.7
21	21.340915	20.9
22	30.018774	32.9
23	20.139406	13.1

	Predicted Labels	True Labels
24	24.870854	24.2
25	3.687029	8.8
26	26.982311	27.1
27	12.184880	13.6
28	22.759950	20.4
29	18.874445	27.1
...	...	...
72	33.726596	31.0
73	34.675520	46.7
74	28.313408	26.4
75	8.809106	7.2
76	34.891977	33.1
77	36.194895	48.3
78	34.203244	33.8
79	18.224409	12.7
80	38.630935	50.0
81	19.614798	24.3
82	16.800947	19.4
83	14.046803	18.2
84	33.791987	39.8
85	8.068913	8.5
86	17.233905	17.4
87	15.267594	16.6
88	16.982627	14.9
89	39.780334	50.0
90	16.549138	20.1
91	1.493049	17.9
92	19.486149	19.4
93	39.735011	50.0
94	23.118783	23.8
95	18.143916	14.2
96	21.737523	25.0
97	14.167421	13.6
98	13.908332	19.0
99	21.356009	23.3
100	16.373994	10.4
101	15.919510	11.7

102 rows × 2 columns

In [23]:

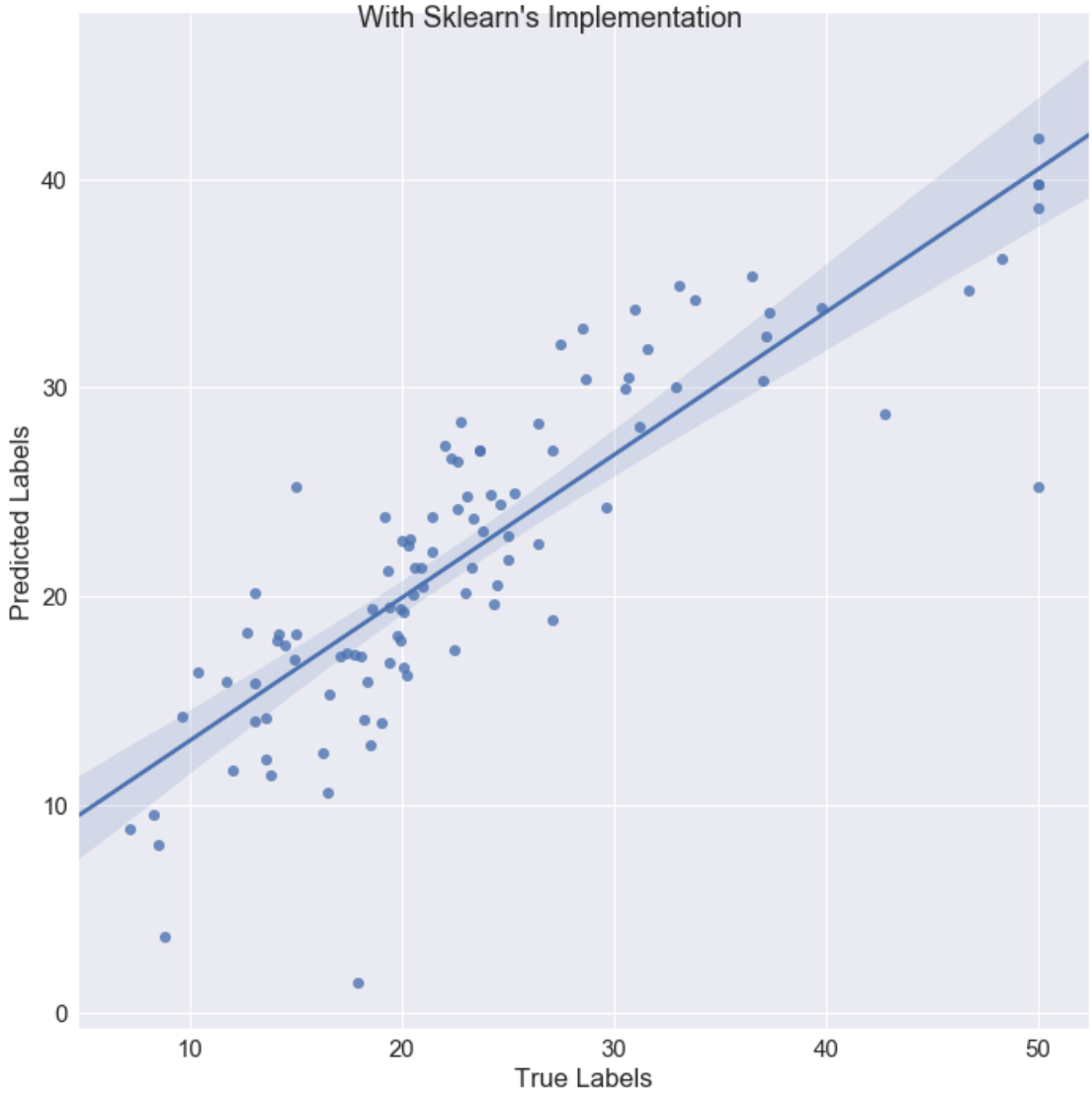
```
Mean_Sq_Error = mean_squared_error(Y_te, Y_pred)
Mean_Sq_Error
```

Out[23]:

```
28.240640661030529
```

In [24]:

```
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)
```



Conclusion

In [25]:

```
print("Sklearn SGD Impementation error=", Mean_Sq_Error)
print("*"*50)
print("My SGD Impementation error=", My_Mean_Sq_Error)
```

```
Sklearn SGD Impementation error= 28.240640661
**************************************************
My SGD Impementation error= 28.2303459746
```