Table of Contents

[1] Problem Statement :

- Performing Linear Regression on Boston Dataset using sklearn.
- Appling Stochastic Gradient Descent manually(without sklearn) on Boston dataset to :
  1)Find the optimal weight vector(Slope)
  2)Find the optimal bias(intercept)
  3)Comparing the results obtained with sklearn regressor results.

[2] Overview of Dataset :

In [112]:

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [113]:

```
from sklearn.datasets import load_boston
boston = load_boston()
```

In [114]:

```
print("Shape of boston data: ",boston.data.shape)
print(boston.feature_names)
```

```
Shape of boston data:  (506, 13)
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [115]:

```
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 25,0
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.
http://archive.ics.uci.edu/ml/datasets/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.

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.

- many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)

In [116]:

```
print(boston.target)
```

```
[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]
```

In [117]:

```
bos = pd.DataFrame(boston.data)
bos.head()
```

Out[117]:

	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
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

In [118]:

```
bos['PRICE'] = boston.target
bos.head()
```

Out[118]:

	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
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

In [119]:

```
X = bos.drop('PRICE',axis = 1)
Y = bos['PRICE']
```

[3] Train and Test Split of Data :

In [120]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_stat
e = 42)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(339, 13)
(167, 13)
(339,)
(167,)
```

In [121]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[4] Linerar Regression using sklearn :

In [122]:

```
from sklearn.linear_model import LinearRegression
def linear_regression():
    model = LinearRegression()
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)
    optimal_W = model.coef_
    optimal_b = model.intercept_
    error = Y_test - Y_pred

    MSE = (1/X_test.shape[0]) * np.sum(error**2)
    RMSE = np.sqrt(MSE)

    print("\n\033[1mOptimal W: \033[0m",optimal_W)
    print("\n\033[1mOptimal intercept(bias): \033[0m",np.round(optimal_b,3))
    print("\n\033[1mMSE: \033[0m",np.round(MSE,3))
    print("\n\033[1mRMSE: \033[0m",np.round(RMSE,3))

    return Y_pred,error

if __name__ == "__main__":
    Y_pred_sklearn,error_sklearn = linear_regression()
```

```
Optimal W:  [-0.98213794  0.86729819  0.40781039  0.86221098 -1.90626375
 2.80199743
 -0.35691613 -3.04777133  2.02981068 -1.36223075 -2.08528939  1.05413356
 -3.9329093 ]

Optimal intercept(bias):  22.971

MSE:  20.747

RMSE:  4.555
```

[5] Linear Regression using Stocahastic Gradient Descent Manually(Without sklearn) :

Batch_size = 32
Learning Rate = 0.01
No of Iterations = 1000

In [123]:

```
def batch_train(X, y, batchSize):
    for i in np.arange(0, X.shape[0], batchSize):
        yield (X[i:i + batchSize], y[i:i + batchSize])
```

In [124]:

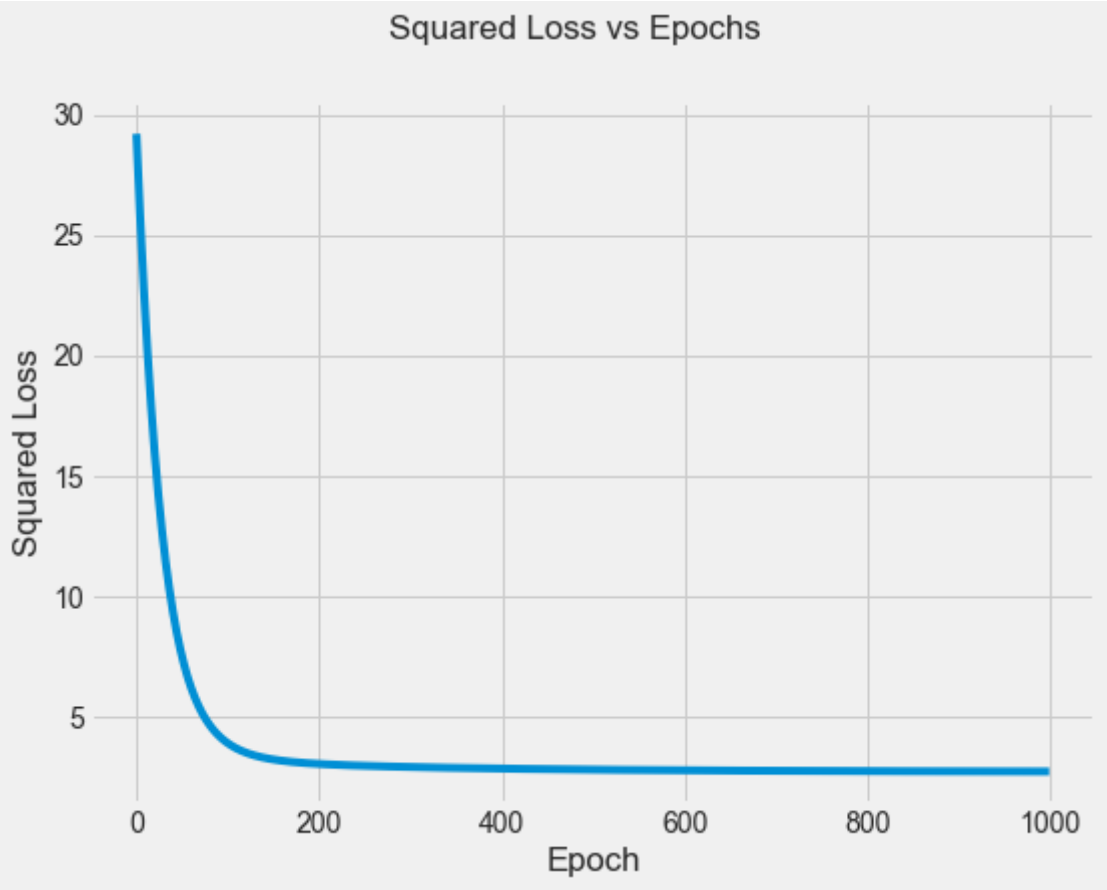
```
def stochastic_grad_descent(x_train, y_train, n_iter = 1000, learning_rate=0.01):

    #Initial weight and bias
    b = 0
    W = np.random.normal(loc = 0, scale = 1, size=(x_train.shape[1],))

    y_train = np.asarray(y_train)
    n = x_train.shape[0]
    loss = []

    for i in range(n_iter):
        print("\n\033[1mEpoch {}---------------->\033[0m".format(i))

        count =0

        #Batch Training
        for (X_batch, y_batch) in batch_train(x_train, y_train, 32):

            yhat = X_batch.dot(W) + b
            error = y_batch-yhat
            sq_loss = (1/n) * np.sum(error ** 2)

            W_grad = -(2/n) * X_batch.T.dot(error)
            b_grad = -(2/n) * np.sum(error)

            #Update Rule
            W = W - (learning_rate * W_grad)
            b = b - (learning_rate * b_grad)
            count+= 1

            #print("Batch: {}, Squared Loss: {}".format(count,sq_loss))

        loss.append(np.average(sq_loss))


    return W, b, loss
```

In [ ]:

```
optimal_W,optimal_b, loss = stochastic_grad_descent(X_train, Y_train)
```

In [126]:

```
plt.style.use('fivethirtyeight')
fig = plt.figure(figsize=(8,6))
plt.plot(np.arange(0,1000), loss)
fig.suptitle("Squared Loss vs Epochs")
plt.xlabel("Epoch")
plt.ylabel("Squared Loss")
plt.show()
```



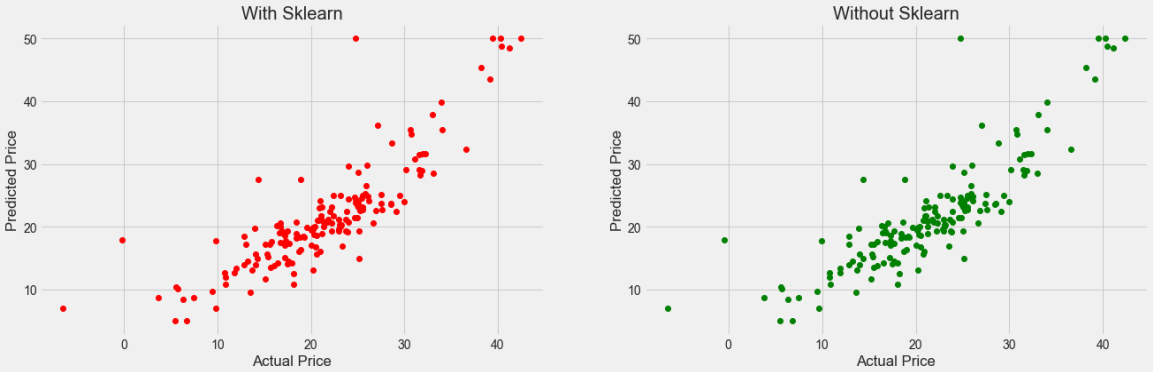
In [127]:

```
def predict(x_test,y_test,W,b):
    print("\n\033[1mOptimal W: \033[0m",W)
    print("\n\033[1mOptimal Intercept(bias): \033[0m",np.round(b,3))
    y_test = np.asarray(y_test)
    n = x_test.shape[0]
    y_pred = x_test.dot(W) + b
    error = y_test-y_pred
    sq_loss = (1/n) * np.sum(error**2)
    rmse = np.sqrt(sq_loss)

    print("\n\033[1mMSE: \033[0m",np.round(sq_loss,3))
    print("\n\033[1mRMSE: \033[0m",np.round(rmse,3))

    return y_pred,error

if __name__ == "__main__":
    Y_pred,error = predict(X_test,Y_test,optimal_W,optimal_b)
```

```
Optimal W:  [-0.95904217  0.84438976  0.40334445  0.86072289 -1.76558169
 2.81036255
 -0.3510435  -2.93099602  2.03254144 -1.41696149 -2.05480504  1.05334962
 -3.96253834]

Optimal Intercept(bias):  22.97

MSE:  20.763

RMSE:  4.557
```

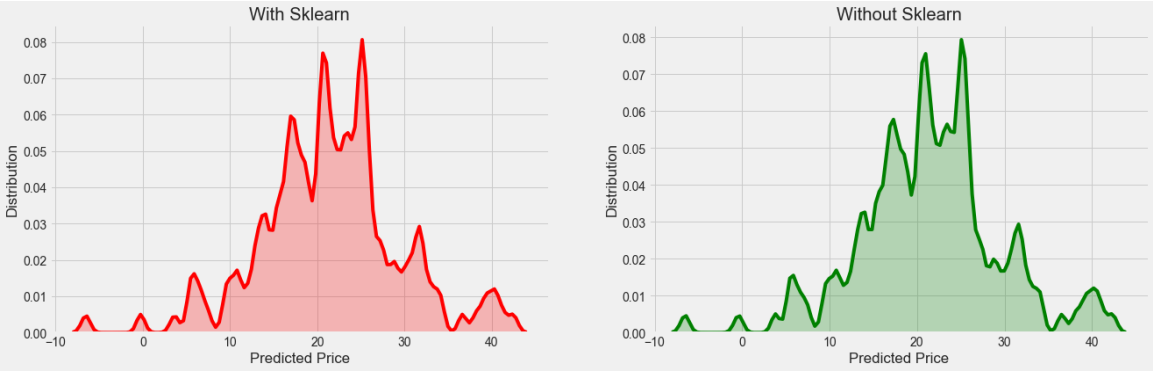
In [128]:

```
plt.figure(figsize = (20,6))
plt.style.use('fivethirtyeight')

plt.subplot(121)
plt.plot(Y_pred_sklearn,Y_test,'ro')
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("With Sklearn")

#plt.figure(figsize = (10,6))
plt.subplot(122)
plt.plot(Y_pred,Y_test,'go')
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Without Sklearn")
plt.show()
```



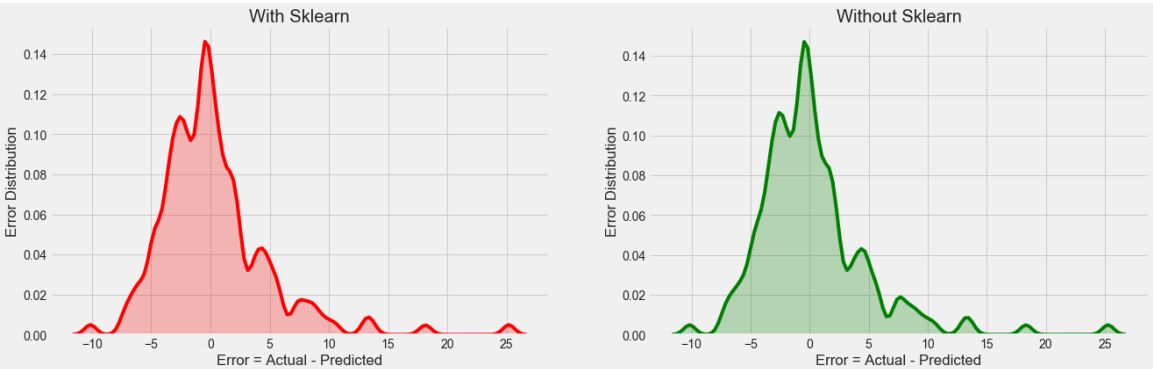
In [129]:

```
plt.figure(figsize = (20,6))
plt.style.use('fivethirtyeight')

plt.subplot(121)
sns.kdeplot(Y_pred_sklearn, bw = 0.5, color = "r", shade = True)
plt.xlabel("Predicted Price")
plt.ylabel("Distribution")
plt.title("With Sklearn")

plt.subplot(122)
sns.kdeplot(Y_pred, bw = 0.5, color = "g", shade = True)
plt.xlabel("Predicted Price")
plt.ylabel("Distribution")
plt.title("Without Sklearn")
plt.show()
```



In [130]:

```
plt.figure(figsize = (20,6))
plt.style.use('fivethirtyeight')

plt.subplot(121)
sns.kdeplot(np.array(error_sklearn), bw = 0.5, color = "r", shade = True)
plt.xlabel("Error = Actual - Predicted")
plt.ylabel("Error Distribution")
plt.title("With Sklearn")

plt.subplot(122)
sns.kdeplot(np.array(error), bw = 0.5, color = "g", shade = True)
plt.xlabel("Error = Actual - Predicted")
plt.ylabel("Error Distribution")
plt.title("Without Sklearn")
plt.show()
```



[6] Conclusion :

	Intercept	MSE	RMSE
Regression(sklearn)	22.97	20.747	4.555
SGD Regression(without sklearn)	22.97	20.763	4.557

1 - It is seen that in Stochastic Gradient Descent,as the number of epochs(iterations) increases Training Loss(squared loss) also decreases.

2 - It can be observed from table, MSE and RMSE obtained by SGD Regression(manually) is almost same as that obtained by sklearn Regression model.

3 - There can be better ways of finding initial weights and bias as well as finding the Learning Rate.