Linear Regression(Single Variable) - From Scratch

Cost Function: Mean Squared Error (MSE)
Optimizatin Algorithm: Gradient Descent

In [2]:
```
# Importing dependencies and dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
```

In [3]:
```
#Creating the dataframe
boston = load_boston()
df = pd.DataFrame(data = boston['data'], columns = boston['feature_name
s'])
df['TARGET'] = boston['target']
df.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
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	4.9
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	9.1
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	4.0
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	2.9
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	5.3

In [4]:
```
#Data statistics
df.describe()
```

Out[4]:

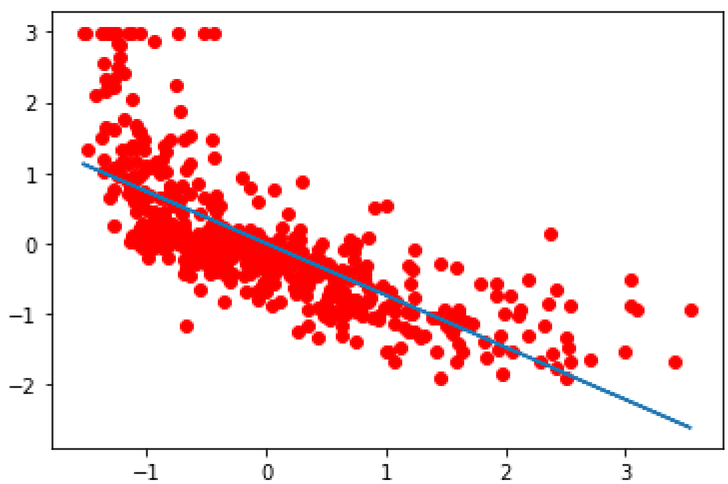
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.(
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.7
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.1
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.1
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.1
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.2
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.1
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.1

In [5]:
```
#Preprocessing the data
x_RM = preprocessing.scale(df['LSTAT'])
y = preprocessing.scale(df['TARGET'])
```

Visualization Of Data Set With Numpy Polyfit Linear Regression Line

In [6]:
```
#Visualization of data
plt.scatter(x_RM, y, c = 'red', marker = 'o')
m,b = np.polyfit(x_RM, y, 1)
plt.plot(x_RM, m*x_RM + b)
print(b,m)
```

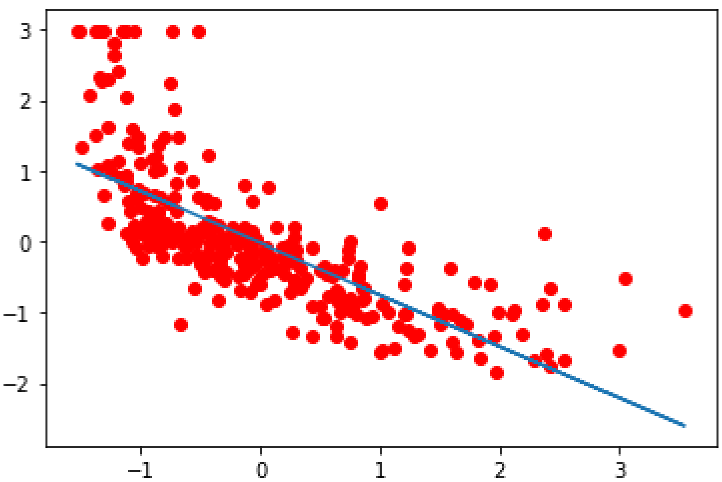
```
-7.769059901369575e-16 -0.7376627261740148
```



In [7]:
```
#Splitting Data Set into Training and Testing Data Sets
X_train, X_test, y_train, y_test = train_test_split(x_RM, y, test_size =
.60, random_state = 1)
```

In [8]:
```
#Visualization of test data set with numpy polyfit linear regression lin
e
#Use to compare with test data set with gradient descent linear regressi
on
plt.scatter(X_test, y_test, c = 'red', marker = 'o')
m,b = np.polyfit(X_test, y_test, 1)
plt.plot(X_test, m*X_test + b)
print(b,m)
```

```
-0.017880996983763033 -0.7285365597959111
```



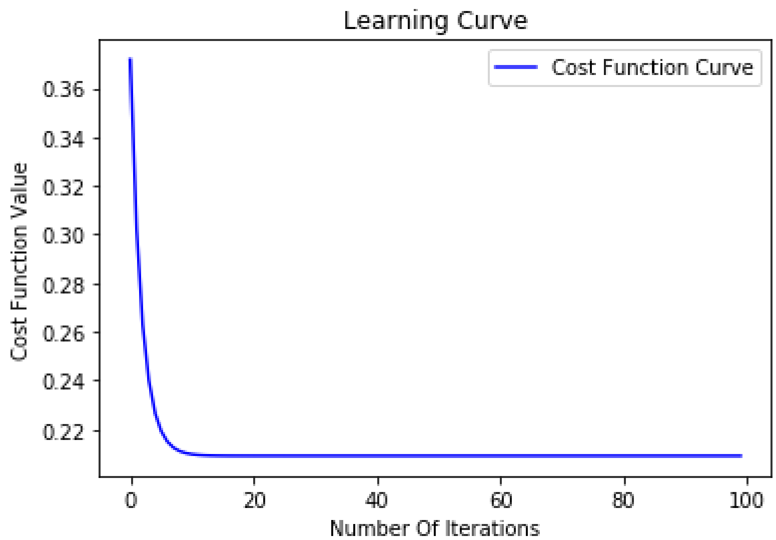
In [9]:
```
#Cost Function(Mean Squared Error)/Calculates MSE for at given point
def cost_function(X_train, y_train, b, m, n):
    errors = []
    for i in range(n):      #y_theoretical = mx + b
        cost = (y_train[i] - (m*X_train[i] + b))**2
        errors.append(cost)
    total_error = sum(errors)
    mse = total_error/(2*n)
    return mse
```

In [10]:
```
#Setting Up Gradient Descent Algorithm
def gradient_descent(X_train, y_train, alpha, b, m, n):

    summation_0 = []
    summation_1 = []

    for i in range(n):
        derivative_0 = (m*X_train[i] + b) - y_train[i]
        derivative_1 = X_train[i]*((m*X_train[i] + b) - y_train[i])

        summation_0.append(derivative_0)
        summation_1.append(derivative_1)

    total_summation_0 = sum(summation_0)/n
    total_summation_1 = sum(summation_1)/n

    new_b = b - (alpha * total_summation_0)
    new_m = m - (alpha * total_summation_1)
    updated_parameters = [new_b, new_m]

    return updated_parameters
```

TRAINING THE DATA SET

In [11]:
```
def training(X_train, y_train, alpha, iters):
    n = len(X_train)

    #initializing optimization parameter
    b = 0
    m = 0

    #creating the optimzation parameter matrix/array
    #param_array = [b, m]

    mse_values = []

    for i in range(iters):
        b,m = gradient_descent(X_train, y_train, alpha, b, m, n)
        mse_values.append(cost_function(X_train, y_train, b, m, n))

    #Plot cost function error per iteration
    x = np.arange(0, len(mse_values), step=1)
    plt.plot(x, mse_values, "-b", label="Cost Function Curve")
    plt.title("Learning Curve")
    plt.xlabel("Number Of Iterations")
    plt.ylabel("Cost Function Value")
    plt.legend()
    plt.show()

    return b,m
```

In [12]:
```
#Graph showing that algorithm is learning and minimizing the cost functi
on
#Start alpha at .001 and increase by 3x until acceptable alpha reached
training(np.array(X_train), np.array(y_train), .243, 100)
```



Out[12]:
```
(0.028172498021124644, -0.7536510428221093)
```

TESTING THE DATA SET

In [13]:
```
def testing(X_test, y_test, b, m):
    n = len(X_test)

    SSTO = []   # total sum of squares
    SSR = []    # regression sum of squares
    SSE = []    # error sum of squares
    y_mean = np.mean(y_train)

    #prediction = []

    for i in range(n):
        predict = m*X_test[i] + b
        #prediction.append(predict)
        SSE.append((predict - y_test[i])**2)
        SSR.append((predict - y_mean)**2)
        SSTO.append((y_test[i] - y_mean)**2)

    print('\naverage error is : ', round(sum(SSE)/len(SSE),4))
    print('\nsum of squares of error (SSE) : ', round(sum(SSE),4))
    print('\nregression sum of squares (SSR) : ', round(sum(SSR),4))
    print('\ntotal sum of squares (SSTO) : ', round(sum(SSTO), 4))
    print('\nThe Coefficient Of Determination R-squared is : ', (round(s
um(SSR)/sum(SSTO),6))*100,'%')


    #return prediction
```

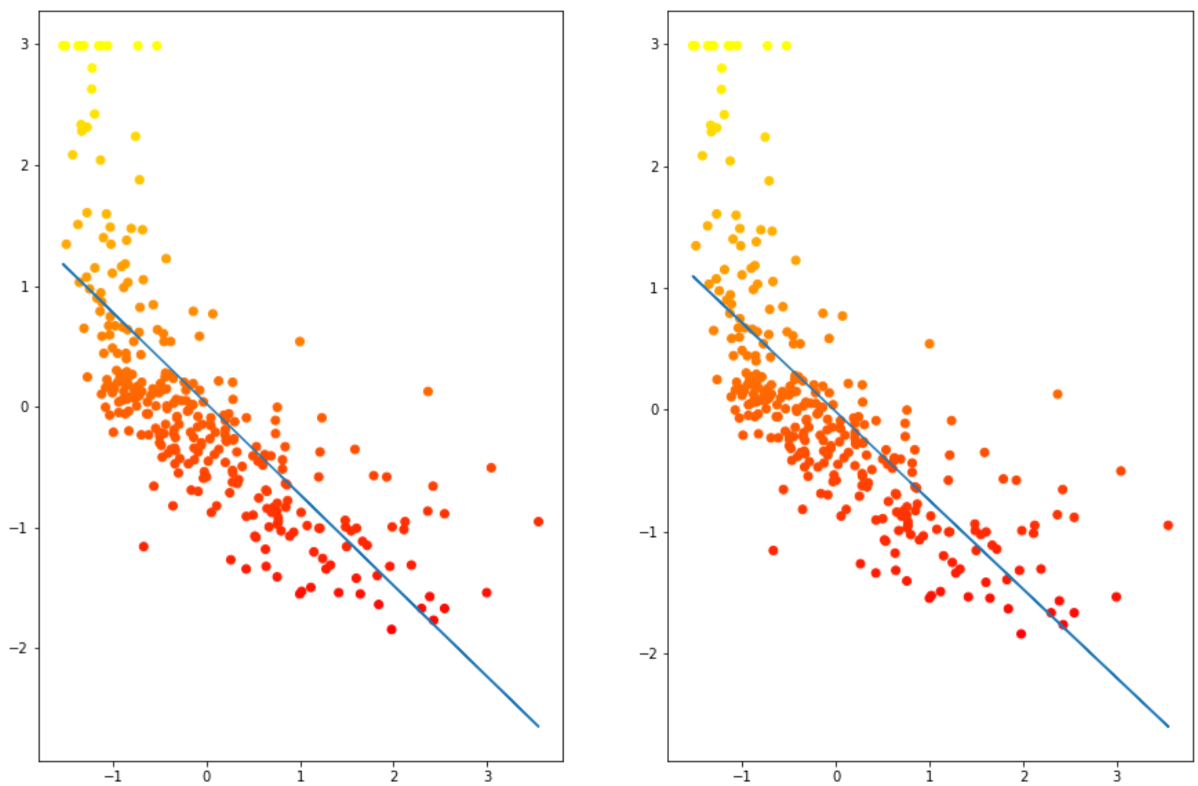
In [14]:
```
testing(np.array(X_test), np.array(y_test),-0.017880996983763033, -0.728
5365597959111)
```

```
average error is :  0.4798

sum of squares of error (SSE) :  145.8596

regression sum of squares (SSR) :  160.991

total sum of squares (SSTO) :  306.8506

The Coefficient Of Determination R-squared is :  52.4656 %
```

Visualization of Regression Line With Gradient Descent vs. Numpy polyfit Method

In [25]:
```
b = 0.028172496146970635
m = -0.7536510404615555
plt.figure(figsize=(15,10))
plt.subplot(1,2,1)
plt.scatter(X_test, y_test, c = y_test, marker = 'o', cmap = plt.cm.autu
mn)
prediction = []
for i in range(len(X_test)):
    predict = m*(np.array(X_test)[i]) + b
    prediction.append(predict)
plt.plot(np.array(X_test), prediction)

plt.subplot(1,2,2)
plt.scatter(X_test, y_test, c = y_test, cmap = plt.cm.autumn, marker =
'o')
m,b = np.polyfit(X_test, y_test, 1)
plt.plot(X_test, m*X_test + b)
```

Out[25]: [<matplotlib.lines.Line2D at 0x1a17df45d0>]



In [ ]: