Assignment - 6 (Implement Stochastic Gradient Descent on Linear Regression)

September 6, 2018

1 OBJECTIVE :- Implement SGD on Linear Regression

In [1]:
```
# Ignoring warnings
import warnings
warnings.filterwarnings('ignore')

# Importing libraries
import numpy as np
import pandas as pd

# Loading Boston dataset
from sklearn.datasets import load_boston
boston = load_boston()

# Shape of dataset
print(boston.data.shape)
```

```
(506, 13)
```

In [2]:
```
# Features of dataset
print(boston.feature_names)
```

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

In [3]:
```
# Shape of target values
print(boston.target.shape)
```

```
(506,)
```

2 Implementing SGD on LINEAR REGRESSION

In [4]:
```
# Feature matrix
data = boston.data
```

```
# Standardizing the data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
standardised_data = sc.fit_transform(data)

# Adding a new feature to the data which will contain only ones for ease in computatio
additional_feature = np.ones(boston.data.shape[0])

# Matrix having new additional feature X0 which will be multiplied with W0 for the eas
feature_data = np.vstack((additional_feature,standardised_data.T)).T

# Actual prices of houses
target_price = boston.target


# Stochastic Gradient Descent Algorithm :
# Let 'K' be the number of random rows selected out of the dataset
# Initialize the weight vector
#Let r = learning_rate and m = number of training_examples
# Let r =1
# repeat until convergence {
#    weight[j] = weight[j] - (r/m)*((from i=1 to K)of(((weight.T * feature_data[i]) -
#    r /= 2
#}

# Final hypothesis for linear regression
# predicted_prices = (final_weights.T)*(test_data_matrix)


# Train and Test split of data
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(feature_data, target_price, test_s
```

IMPLEMENTING STOCHASTIC GRADIENT DESCENT ALGORITHM

In [5]:
```
# Initialising weight vector
# Generating 14 normally distributed values
weights = np.random.normal(0,1,feature_data.shape[1])

# Initialised Weights
weights
```

Out[5]:
```
array([-0.65763982,  1.22217528,  1.22978936, -0.87919219,  0.34683624,
       -0.39228968,  0.40783691, -1.18427173,  1.39299815,  0.36218083,
        1.52073051, -0.965439  ,  0.55735123,  0.7466486 ])
```

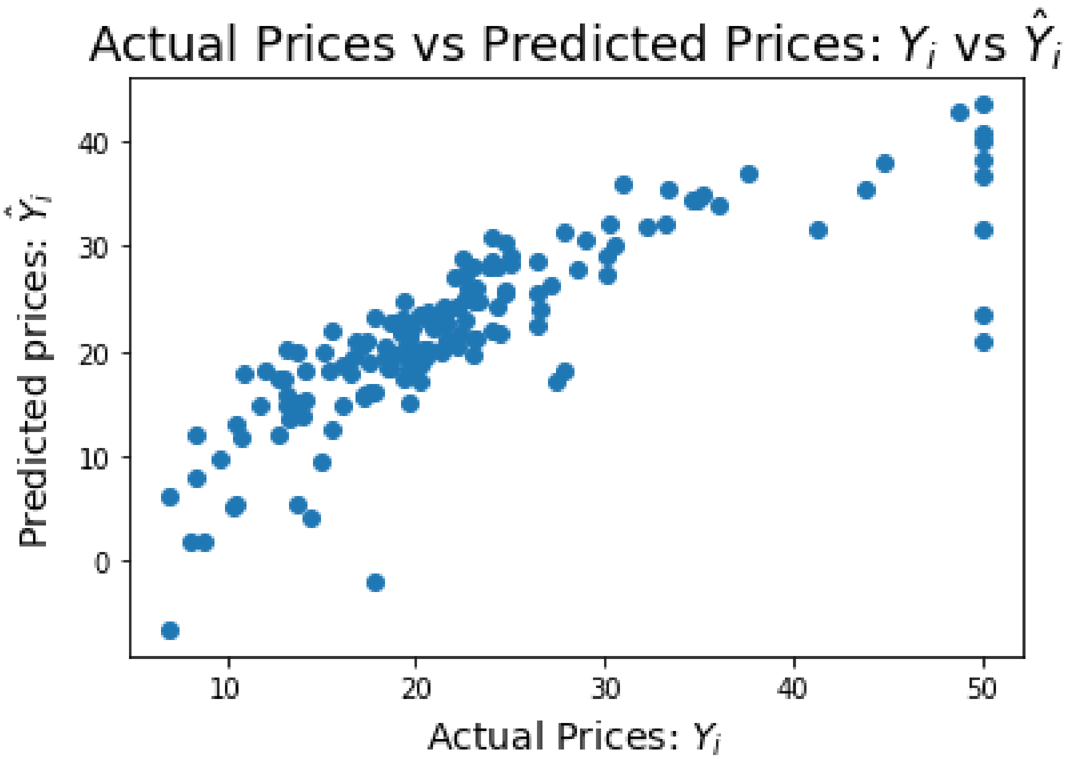
In [6]:
```
# Temporary vector to store intermediate computed weight values
temp_w = np.zeros(feature_data.shape[1])
```

```
# Initialising learning rate
r = 0.001

# Number of training examples
m = X_train.shape[0]

# Code to get batches for Stochastic Gradient Descent
# batch size
batch_size = 20
from numpy import random
random_ids = random.choice(m,m,replace=False)
X_shuffled = X_train[random_ids,:]
y_shuffled = Y_train[random_ids]
mini_batches = [(X_shuffled[i:i+batch_size,:], y_shuffled[i:i+batch_size]) for i in ran


# Number of iterations for training the data
iterations = 1000

# SGD
while(iterations >=0):
    for batch in mini_batches:
        X_batch = batch[0]
        Y_batch = batch[1]
        for j in range(0,feature_data.shape[1]):
            temp_sum = 0
            for i in range(0,X_batch.shape[0]):
                temp_sum += (( (np.sum( sc.inverse_transform(weights[1:14] * X_batch[i
            temp_w[j] = weights[j] - ((r/X_batch.shape[0])*temp_sum)
        weights = temp_w
    iterations -= 1

# Weights of manual sgd
manual_sgd_weights = weights
```

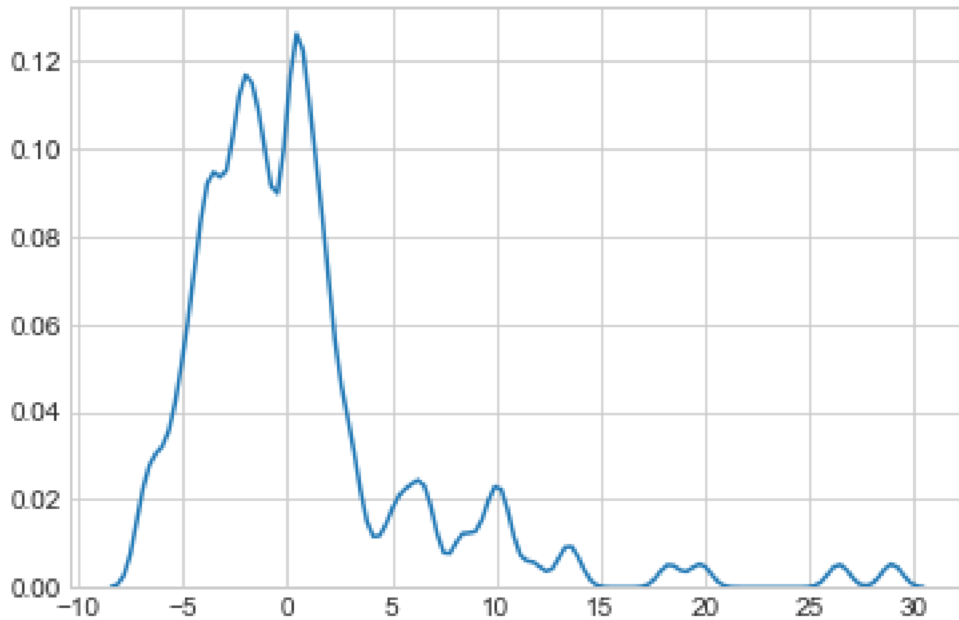
In [7]:
```
# Now predicting the house prices on X_test data
manual_sgd_predictions = np.zeros(X_test.shape[0])
for itr in range(0,X_test.shape[0]):
    manual_sgd_predictions[itr] = np.sum(sc.inverse_transform(weights[1:14]*X_test[itr
```

In [8]:
```
# Plotting the Scatter plot of Actual Price VS Predicted Price
import matplotlib.pyplot as plt
%matplotlib inline

plt.scatter(Y_test, manual_sgd_predictions)
plt.xlabel("Actual Prices: $Y_i$",size=14)
plt.ylabel("Predicted prices: $\hat{Y}_i$",size=14)
```

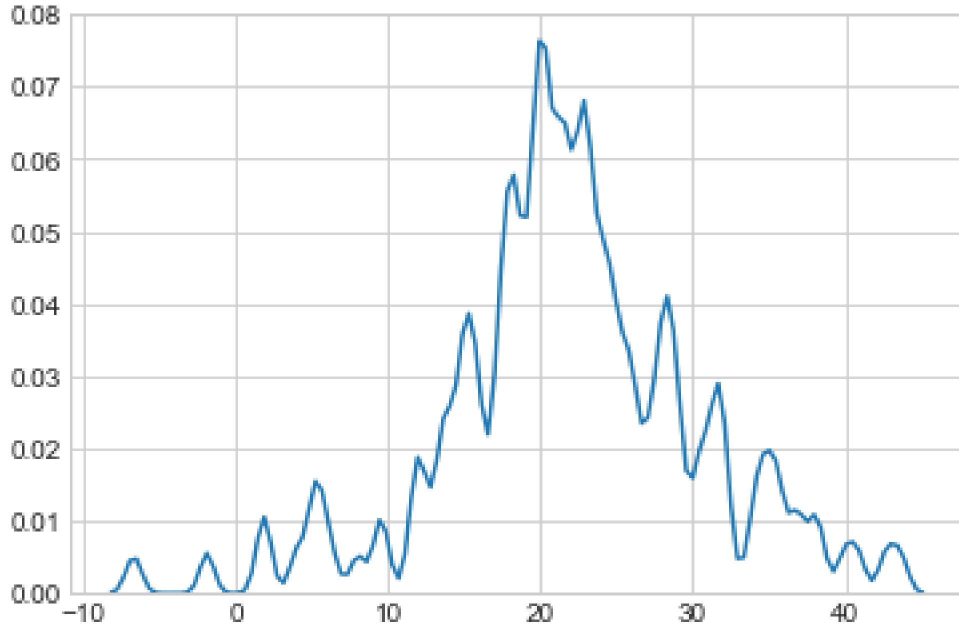
```
plt.title("Actual Prices vs Predicted Prices: $Y_i$ vs $\hat{Y}_i$",size=18)
plt.show()
```




```
In [9]: delta_y = Y_test - manual_sgd_predictions;

        import seaborn as sns;
        import numpy as np;
        sns.set_style('whitegrid')
        sns.kdeplot(np.array(delta_y), bw=0.5)
        plt.show()
```




```
In [10]: sns.set_style('whitegrid')
         sns.kdeplot(np.array(manual_sgd_predictions), bw=0.5)
         plt.show()
```

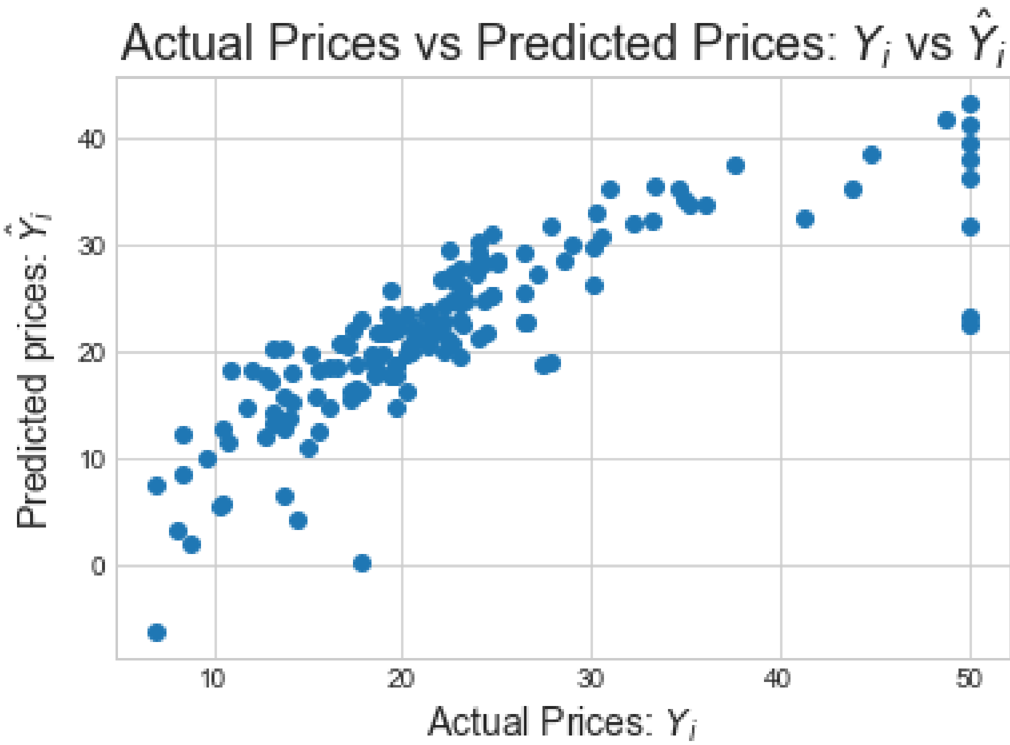


```
In [11]: # Calculating accuracy for Implementation of SGD from Scratch
         from sklearn.metrics import mean_absolute_error,mean_squared_error

         # calculate Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
         print("Mean Absolute Error for Implementation of SGD from Scratch is : ",mean_absolut
         print("Mean Squared Error for Implementation of SGD from Scratch is : ",mean_squared_
         print("Root Mean Squared Error for Implementation of SGD from Scratch is : ",np.sqrt(
```

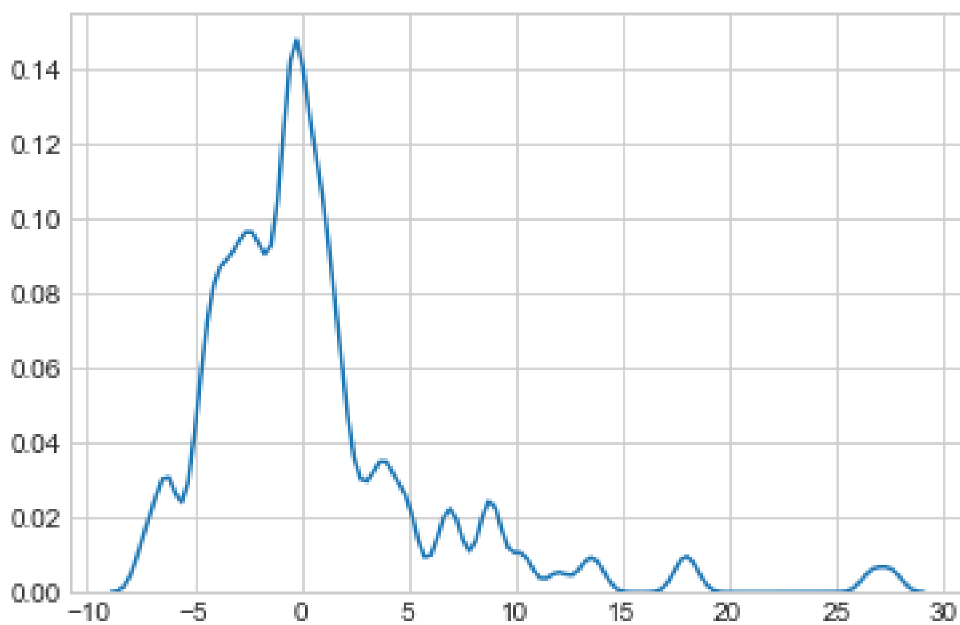
```
Mean Absolute Error for Implementation of SGD from Scratch is :  3.701859781693028
Mean Squared Error for Implementation of SGD from Scratch is :  32.28686713645806
Root Mean Squared Error for Implementation of SGD from Scratch is :  5.68215338902938
```

3 Implementing SKLEARN's SGD Regression

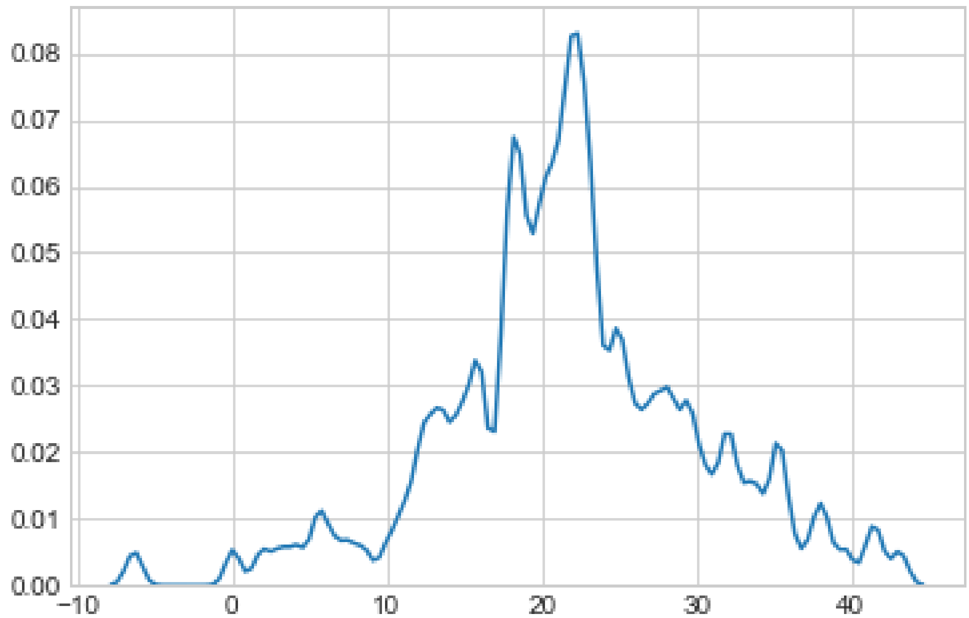
```
In [13]: # Implement Sklearn SGD with following parameters as used in manual SGD :
         #  (1) No regularization (2) Learning_rate = 0.001 and (3) Number of iterations = 100

         from sklearn.linear_model import SGDRegressor
         sgd = SGDRegressor(penalty='none', max_iter=1000, learning_rate='constant' , eta0=0.0
         sgd.fit(X_train, Y_train)

         sklearn_sgd_predictions = sgd.predict(X_test)

         # Weights of Sklearn's SGD
         sklearn_sgd_weights = sgd.coef_

         plt.scatter(Y_test, sklearn_sgd_predictions)
         plt.xlabel("Actual Prices: $Y_i$",size=14)
         plt.ylabel("Predicted prices: $\hat{Y}_i$",size=14)
         plt.title("Actual Prices vs Predicted Prices: $Y_i$ vs $\hat{Y}_i$",size=18)
         plt.show()
```




```
In [14]: delta_y = Y_test - sklearn_sgd_predictions;
         sns.set_style('whitegrid')
         sns.kdeplot(np.array(delta_y), bw=0.5)
         plt.show()
```



```
In [15]: sns.set_style('whitegrid')
         sns.kdeplot(np.array(sklearn_sgd_predictions), bw=0.5)
         plt.show()
```




```
In [16]: # Calculating accuracy for Implementation of SGD using SKLEARN
         from sklearn.metrics import mean_absolute_error,mean_squared_error

         # calculate Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
         print("Mean Absolute Error for Implementation of SGD using SKLEARN is : ",mean_absolu
         print("Mean Squared Error for Implementation of SGD using SKLEARN is : ",mean_squared
         print("Root Mean Squared Error for Implementation of SGD using SKLEARN is : ",np.sqrt

Mean Absolute Error for Implementation of SGD using SKLEARN is :  3.556915630433543
Mean Squared Error for Implementation of SGD using SKLEARN is :  30.61978530193464
Root Mean Squared Error for Implementation of SGD using SKLEARN is :  5.533514733145168
```

4 Comparing the weights produced by both Manual SGD and Sklearn's SGD

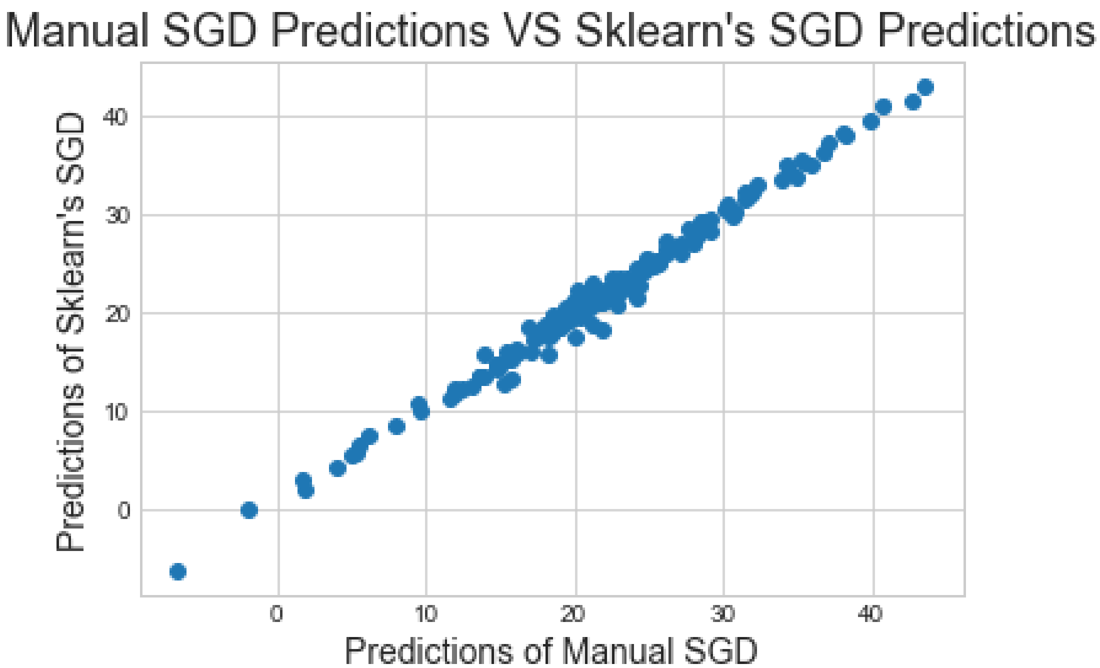
```
In [17]: # Creating the table using PrettyTable library
         from prettytable import PrettyTable
```

```
numbering = [1,2,3,4,5,6,7,8,9,10,11,12,13,14]
# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("Weights of Manual SGD",manual_sgd_weights)
ptable.add_column("Weights of Sklearn's SGD",sklearn_sgd_weights)

# Printing the Table
print(ptable)
```

```
+--------+-----------------------+-------------------------+
| S.NO.  | Weights of Manual SGD | Weights of Sklearn's SGD |
+--------+-----------------------+-------------------------+
|   1    |   -888.5749950375097  |    11.177567732785874   |
|   2    |  -0.1434070504295682  |   -1.3146239333999497   |
|   3    |  0.03994450385231205  |    0.9752378812002891   |
|   4    |  -0.09315535824536102 |   -0.16804847365693787  |
|   5    |   1.7990682598396177  |   0.21825705676155369   |
|   6    |   0.7535038797911041  |   -1.5082818365282562   |
|   7    |   4.174741181671668   |    2.838533260608808    |
|   8    |  -0.02063723669602402 |   -0.28202446561066363  |
|   9    |  -1.1220212029714827  |   -2.8163645475795347   |
|   10   |   0.2625651578959432  |    2.7760051102407384   |
|   11   | -0.014246060301718584 |    -2.124483330021912   |
|   12   |  -0.8160364688568095  |   -2.1262242677517595   |
|   13   |  0.013141609075295867 |     1.17475928537059    |
|   14   |  -0.4715393808317694  |   -3.3234118197020743   |
+--------+-----------------------+-------------------------+
```

5 Scatter Plot of the predictions of both manual and sklearn SGD Regression

In [18]:
```
# Scatter Plot of the predictions of both manual SGD Regression and Sklearn's SGD Reg
plt.scatter(manual_sgd_predictions, sklearn_sgd_predictions)
plt.xlabel("Predictions of Manual SGD",size=14)
plt.ylabel("Predictions of Sklearn's SGD",size=14)
plt.title("Manual SGD Predictions VS Sklearn's SGD Predictions",size=18)
plt.show()
```



6 OBSERVATION :

By observing the graphs , mean absolute error , mean squared error and root mean squared error for both (Manual sgd Regression and Sklearn's sgd Regression) implementation of SGD we can say that Manual SGD model and Sklearn's SGD model is giving approximately similar results

7 CONCLUSION :

8 (a). Procedure Followed :

STEP 1 :- Load the boston data
STEP 2 :- Column standardized the data and split it into train_data and test_data
STEP 3 :- Implement Manual SGD Regression
STEP 4:- Draw scatter plot of Actual Prices vs Predicted Prices for Manual sgd implementation
STEP 5:- Calculate Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) for Manual sgd implementation
STEP 6:- Implement Sklearn's SGD Regression
STEP 7:- Draw scatter plot of Actual Prices vs Predicted Prices for Sklearn's sgd implementation
STEP 8:- Calculate Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) for Sklearn's sgd implementation
STEP 9:- Compare the weights produced by both Manual SGD and Sklearn's SGD
STEP 10:- Draw Scatter Plot of the predictions of both manual SGD Regression and sklearn's SGD Regression