1. Importing Packages

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 import preprocessing
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
from sklearn.metrics import mean_absolute_error
```

2. Loading Dataset

In [2]:

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

In [3]:

```
print(boston.data.shape)
```

```
(506, 13)
```

In [4]:

```
print(boston.feature_names)
```

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

In [5]:

```
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
```

```
 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 [6]:

```python
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 25,000 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.
https://archive.ics.uci.edu/ml/machine-learning-databases/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.

.. topic:: 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.

In [7]:

```
import pandas as pd
bos = pd.DataFrame(boston.data)
bos.head()
```

Out[7]:

	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

3. As the column names are in index value form so we rename them accoring to the description data

In [8]:

```
bos.columns = boston['feature_names']
```

In [9]:

```
bos.head()
```

Out[9]:

	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

In [10]:

```
# bos =
bos.rename({0:'CRIM',1:'ZN',2:'INDUS',3:'CHAS',4:'NOX',5:'RM',6:'AGE',7:'DIS',8:'RAD',9:'TAX',10:'P
IO',11:'B',
#                    12:'LSTAT'},axis=1)
```

In [11]:

```
bos['PRICE'] = boston.target

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

In [12]:

```
Y.head()
```

Out[12]:

```
0     24.0
1     21.6
```

```
2    34.7
3    33.4
4    36.2
Name: PRICE, dtype: float64
```

4. Splitting Data into Train and Test set

In [13]:

```
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_state = 5)

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

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

In [14]:

```
from sklearn.preprocessing import MinMaxScaler
scalar = MinMaxScaler()

X_train = scalar.fit_transform(X_train)
X_test = scalar.transform(X_test)
print(X_train.shape)
print(X_test.shape)
```

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

In [15]:

```
Xtrain = pd.DataFrame.from_records(X_train)
Xtrain.columns = boston["feature_names"]
Xtest = pd.DataFrame.from_records(X_test)
Xtest.columns = boston["feature_names"]
```

In [16]:

```
Xtrain.head()
```

Out[16]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.125324	0.0	0.636089	0.0	0.730453	0.587852	0.944387	0.104533	1.000000	0.913958	0.808511	0.272149	0.594371
1	0.000513	0.0	0.081540	0.0	0.213992	0.660280	0.858908	0.241800	0.043478	0.156788	0.553191	1.000000	0.104029
2	0.051086	0.0	0.636089	0.0	0.685185	0.000000	0.875386	0.050398	1.000000	0.913958	0.808511	0.892997	0.148731
3	0.001033	0.3	0.138920	0.0	0.088477	0.535926	0.514933	0.624266	0.217391	0.214149	0.425532	0.938765	0.261865
4	0.041389	0.0	0.636089	0.0	0.674897	0.539375	0.880536	0.151354	1.000000	0.913958	0.808511	0.986130	0.356512

In [17]:

```
manual_Xtrain = Xtrain.T
manual_Xtest = Xtest.T
manual_ytrain = np.array([Y_train])
manual_ytest = np.array([Y_test])
```

5. Applying Manual Linear Regression on our Train and Test

5. Applying Manual Linear Regression on our Train and Test sets

In [20]:

```
def initialize_parameters(lenw):
    w = np.random.randn(1,lenw)
    b = 0
    return w,b

def forward_prop(X,w,b):
    z = np.dot(w,X) + b
    return z

def back_prop(X,y,z):
    m = y.shape[1]
    dz = (1/m) * (z-y)
    dw = np.dot(dz,X.T)
    db = np.sum(dz)
    return dw,db

def gradient_descent_update(w,b,dw,db,lr):
    w = w - lr*dw
    b = b - lr*db
    return w,b

def linear_regression_model(xtrain,ytrain,lr,epochs):
    lenw = xtrain.shape[0]
    w,b = initialize_parameters(lenw)
    for i in range(1,epochs+1):
        z = forward_prop(xtrain,w,b)
        dw,db = back_prop(xtrain,ytrain,z)
        w,b = gradient_descent_update(w,b,dw,db,lr)
    return w,b

def pred(X_test,w, b):
    y_pred=[]
    for i in range(len(X_test)):
        y=np.asscalar(np.dot(w,X_test[i])+b)
        y_pred.append(y)
    return np.array(y_pred)

def plot_(X_test,y_pred):
    #scatter plot
    plt.scatter(Y_test,y_pred)
    plt.grid(b=True, linewidth=0.3)
    plt.title('scatter plot between actual y and predicted y')
    plt.xlabel('actual y')
    plt.ylabel('predicted y')
    plt.show()
```

In [22]:

```
w,b = linear_regression_model(manual_Xtrain,manual_ytrain,0.4,500)
```

In [23]:

```
w,b
```

Out[23]:

```
(array([[ -8.39794673,   2.81152147,  -0.67684544,   0.94131682,
          -3.86387878,  24.14918406,  -0.76032164,  -9.23962616,
           5.434565  ,  -5.26516346,  -8.50229375,   5.48755893,
         -16.73926421]]), 19.99478422423874)
```

In [24]:

```
y_pred_manual = pred(np.array(Xtest), w=w, b=b)
```

In [25]:

```
y_pred_manual
```

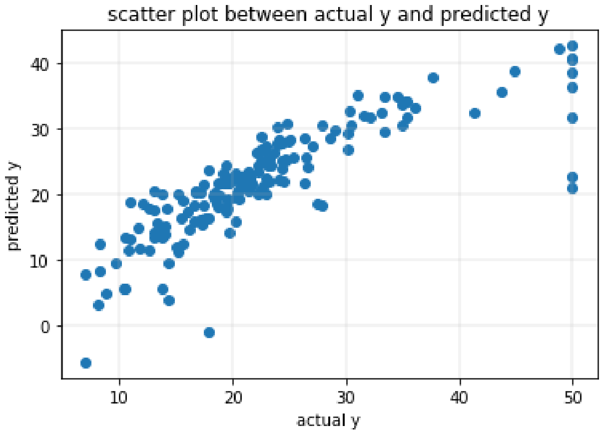
Out[25]:

```
array([37.74777303, 30.37003301, 26.77665346,  5.66822103, 34.12904734,
        5.6495394 , 27.55477754, 29.65824848, 26.54307865, 22.05356863,
       32.42899723, 21.64226279, 23.10120576, 32.95773268, 27.77488574,
       16.35775712, -1.08934277, 19.10973331, 14.14305076, 12.06246644,
        3.33776958, 20.03036901, 38.6792785 , 24.45280669, 31.71994315,
       11.53934045, 24.98869779, 23.16571715, 23.39690331, 22.49778306,
       15.42970796,  7.88266519, 17.7514154 , 23.87890383, 28.56363465,
       19.44576721, 28.35323257,  8.40872985, 42.15369838, 33.49600281,
       20.16499578,  3.92103109, 29.17938958, 11.4915768 , 27.41988222,
       30.88256916, -5.37942365, 19.37700289, 22.46760626, 13.88486912,
       20.10358069, 20.1197512 , 23.54641144, 13.6172954 , 18.17258553,
       25.47472936, 35.66086267, 15.14573431, 28.73915485, 21.67532832,
       20.48276008, 25.39642676, 14.74642904, 32.43013127, 21.35885306,
       12.66959923, 20.30845802, 25.12847291, 21.45300294, 20.64149728,
       20.35434348, 26.24493353, 17.49926586, 18.53489966, 18.35555518,
       26.96449288, 21.44033803, 16.35922155, 34.97550083, 17.83628371,
       21.82265035, 40.65341989, 21.75139977, 15.0153848 , 24.40915316,
       17.51636799, 18.14119804,  9.61054117, 19.64646213, 18.65630031,
       36.37577772, 17.91003585, 21.0647574 , 19.25322985, 25.04724107,
       28.1993273 , 12.55547673, 24.23142233, 20.57059992, 13.47366402,
       22.60582024, 22.1908283 , 14.2773397 , 42.51185108,  5.17882754,
       22.00107657, 18.47900291, 20.94689917, 28.32875109, 17.26725197,
       27.84246029, 23.84314436, 20.4908912 , 32.72517478, 20.08933545,
       13.50037562, 21.56710089, 18.42544684, 19.97835841, 16.55261117,
       21.66826185, 34.72574114, 22.86873111, 20.22332417, 24.36861364,
       25.59789831, 20.58460164, 22.55706247, 23.17121463, 40.57581941,
       38.72008889, 27.44017179, 13.43744303, 16.42084643, 18.77147296,
       21.72741558, 14.90784834,  5.69561009, 24.42586726, 30.54681075,
       22.69326398, 18.97322197, 15.73013297, 21.7141724 , 34.93184568,
       22.76298838, 30.28557492, 18.13607681, 22.25840705, 28.8538589 ,
       13.69736979, 31.71591134, 11.6284262 , 13.25948995, 26.31034766,
       31.72664929, 11.4906407 , 25.11814497, 29.64997105, 32.0337451 ,
       15.67329418, 30.36167208,  9.76050384, 34.20497882, 25.53362151,
       19.76099658, 15.8897928 ])
```

5.1. Visualizing Data using Scatter Plots

In [23]:

```
plotting = plot_(np.array(Xtest),y_pred_manual)
```



5.2. Printing Data Frame of Actual and Predicted values using Linear Regression Model

In [27]:

```
df_manual = pd.DataFrame({'Actual Values': Y_test,'Predicted Values': y_pred_manual})
```

In [28]:

```
df_manual.head()
```

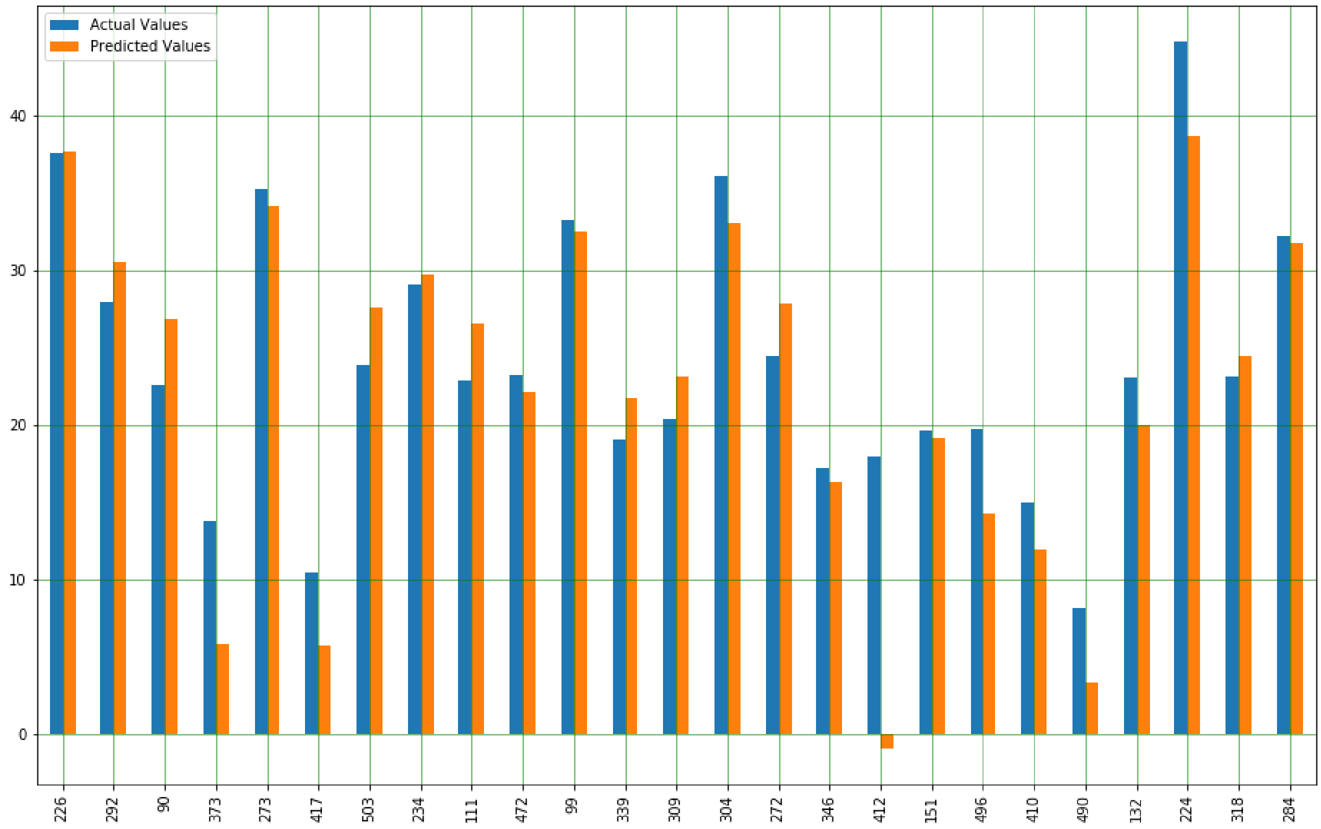
Out[28]:

	Actual Values	Predicted Values
226	37.6	37.691440
292	27.9	30.502252
90	22.6	26.847459
373	13.8	5.769157
273	35.2	34.092390

5.3. Visualizing Data Using Bar Plots

In [29]:

```
#https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-python-with-scikit-learn
-83a8f7ae2b4f
df1 = df_manual.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



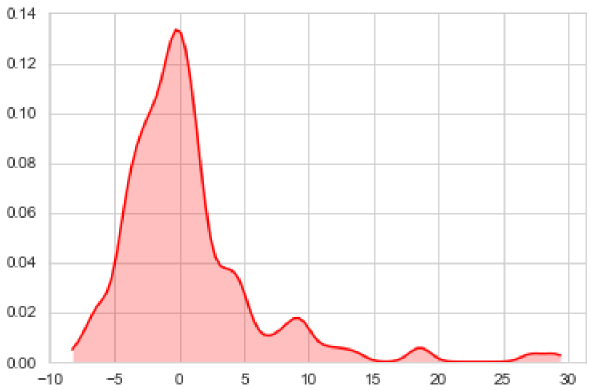
5.4. Visualizing Data usind Kernel Density(KDE) plot

In [30]:

```
delta_y = Y_test - y_pred_manual;

import seaborn as sns;
import numpy as np;
sns.set_style('whitegrid')
sns.kdeplot(np.array(delta_y), bw=.15,shade=True, color="r")
```

```
plt.show()
```



5.5. Finding Mean Absolute Error , Mean Squared Error , Root Mean Squared Error

In [31]:

```
print('Mean Absolute Error:', mean_absolute_error(Y_test, y_pred_manual))
print('Mean Squared Error:', mean_squared_error(Y_test, y_pred_manual))
print('Root Mean Squared Error:', sqrt(mean_squared_error(Y_test, y_pred_manual)))
```

```
Mean Absolute Error: 3.369378635648655
Mean Squared Error: 28.669500644115576
Root Mean Squared Error: 5.3543907817897995
```

6. Applying Linear Regression on our Train and Test sets

In [32]:

```
# code source:https://medium.com/@haydar_ai/learning-data-science-day-9-linear-regression-on-boston-housing-dataset-cd62a80775ef
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

lm = LinearRegression()
lm.fit(Xtrain, Y_train)

Y_pred_linear = lm.predict(Xtest)
```

In [33]:

```
#Retrieve intercept
print(lm.intercept_)
```

```
25.059473071710524
```

In [34]:

```
#Retrieve slope
print(lm.coef_)
```

```
[-13.91249965   3.85490972  -0.66391866   0.78643968  -6.29219928
  20.89003164  -1.12658717 -12.9285439    7.86040903  -7.06828342
  -9.29534072   4.75575802 -17.12862872]
```

6.1. Printing Data Frame of Actual and Predicted values using Linear Regression Model

In [35]:

```
df_linear = pd.DataFrame({'Actual Values': Y_test,'Predicted Values': Y_pred_linear})
```

In [36]:

```
df_linear.head()
```

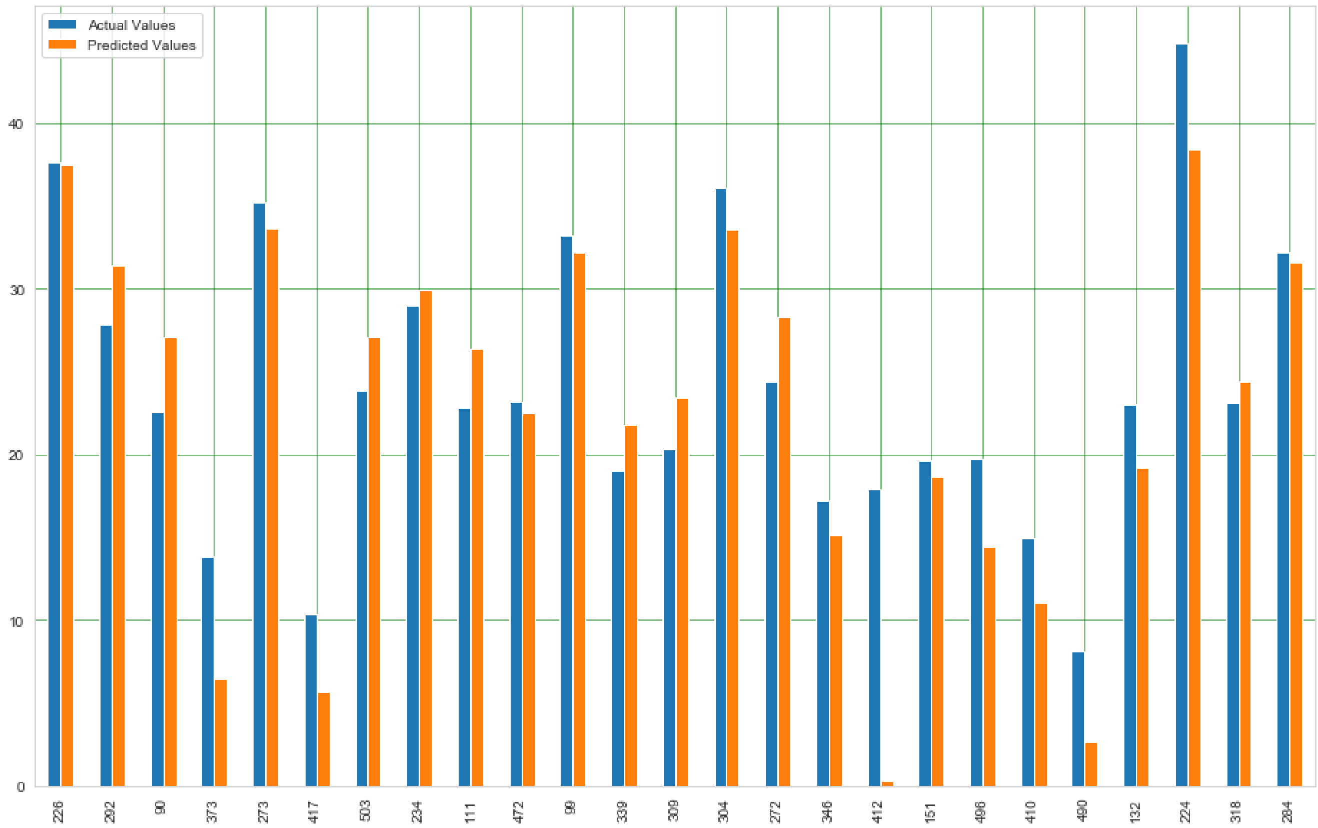
Out[36]:

	Actual Values	Predicted Values
226	37.6	37.467236
292	27.9	31.391547
90	22.6	27.120196
373	13.8	6.468433
273	35.2	33.629667

6.2. Visualizing Data Using Bar Plots

In [37]:

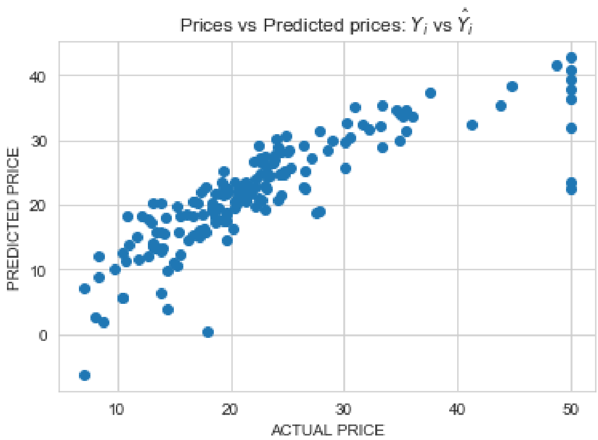
```
#https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-python-with-scikit-learn
-83a8f7ae2b4f
df2 = df_linear.head(25)
df2.plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



6.3. Visualizing Data using Scatter Plots

In [38]:

```
plt.scatter(Y_test, Y_pred_linear)
plt.xlabel("ACTUAL PRICE")
plt.ylabel("PREDICTED PRICE")
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
plt.show()
```

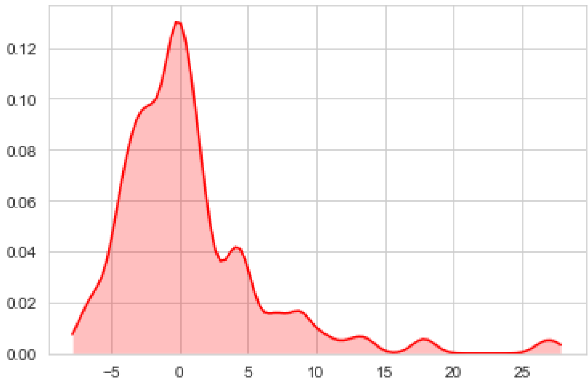


6.4. Visualizing Data usind Kernel Density(KDE) plot

In [39]:

```
delta_y = Y_test - Y_pred_linear;

import seaborn as sns;
import numpy as np;
sns.set_style('whitegrid')
sns.kdeplot(np.array(delta_y), bw=.15,shade=True, color="r")
plt.show()
```



The above KDE plot shows that our linear regression model is working moderate as the curve is not falling down smoothly on the left and is peakered.

6.5. Finding Mean Absolute Error , Mean Squared Error , Root Mean Squared Error

In [40]:

```
print('Mean Absolute Error:', mean_absolute_error(Y_test, Y_pred_linear))
print('Mean Squared Error:', mean_squared_error(Y_test, Y_pred_linear))
print('Root Mean Squared Error:', sqrt(mean_squared_error(Y_test, Y_pred_linear)))
```

```
Mean Absolute Error: 3.4550349322483522
Mean Squared Error: 28.53045876597462
Root Mean Squared Error: 5.341391089030518
```

7. Applying SGDRegressor using 'l2' Regularization on our Train and Test sets

In [41]:

```
# code source:https://medium.com/@haydar_ai/learning-data-science-day-9-linear-regression-on-bosto
n-housing-dataset-cd62a80775ef
from sklearn.linear_model import SGDRegressor
import matplotlib.pyplot as plt

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

lm1 = SGDRegressor(learning_rate = 'constant', penalty ='l2' , max_iter = 1000)
lm1.fit(Xtrain, Y_train)

Y_pred_sgd_l2 = lm1.predict(Xtest)
```

7.1. Printing Data Frame of Actual and Predicted values using SGDRegressor with 'l2' Regularization Model

In [42]:

```
df_sgd_l2 = pd.DataFrame({'Actual Values': Y_test,'Predicted Values': Y_pred_sgd_l2})
```

In [43]:

```
df_sgd_l2.head()
```

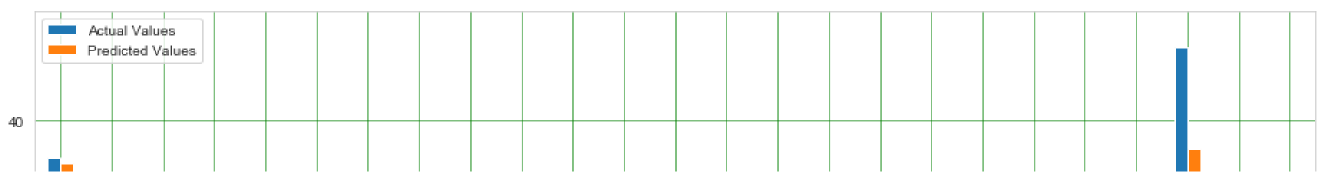
Out[43]:

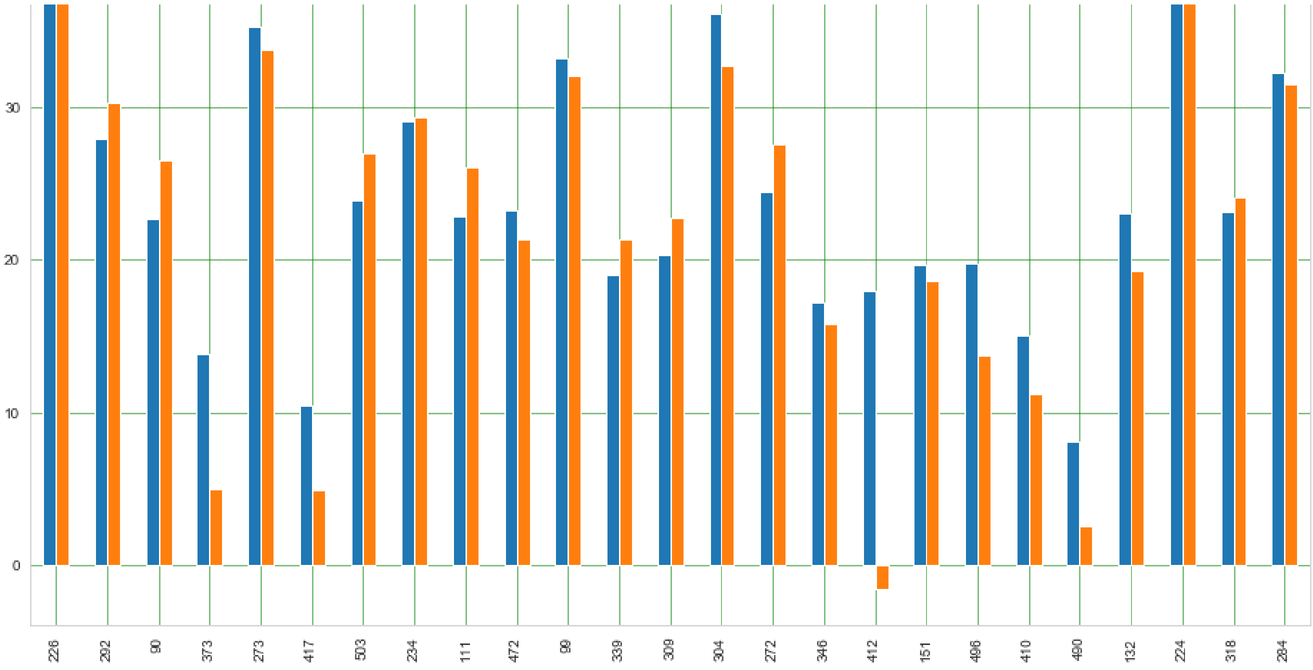
	Actual Values	Predicted Values
226	37.6	37.205189
292	27.9	30.222423
90	22.6	26.466172
373	13.8	4.970637
273	35.2	33.720718

7.2. Visualizing Data using Bar Plots

In [44]:

```
df3 = df_sgd_l2.head(25)
df3.plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```





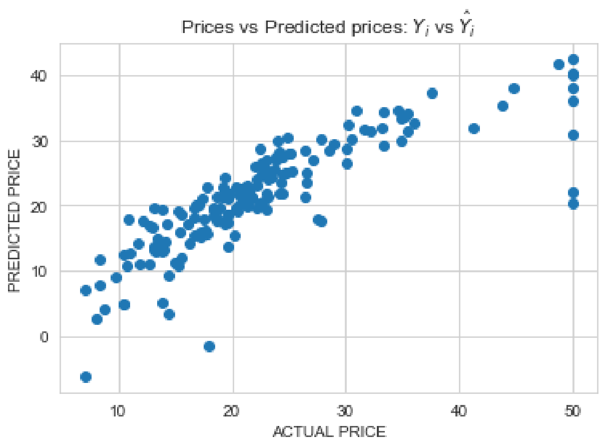
7.3. Visualizing Data using Scatter Plots

In [45]:

```
plt.scatter(Y_test, Y_pred_sgd_l2)
plt.xlabel("ACTUAL PRICE")
plt.ylabel("PREDICTED PRICE")
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
plt.show
```

Out[45]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```

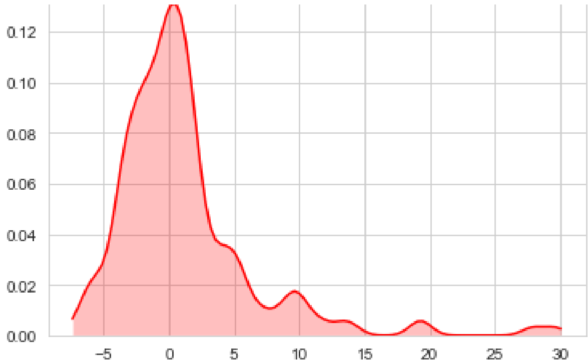


7.4. Visualizing Data usind Kernel Density(KDE) plot

In [46]:

```
delta_y = Y_test - Y_pred_sgd_l2;

import seaborn as sns;
import numpy as np;
sns.set_style('whitegrid')
sns.kdeplot(np.array(delta_y), bw=.15,shade=True, color="r")
plt.show()
```



7.5. Finding Mean Absolute Error , Mean Squared Error , Root Mean Squared Error

The above KDE plot shows that our SGDRegressor with 'l2' regularization model is working much better than the Linear Regression model as the curve is falling down smoothly till value 0.065 on the left and is less peakered.

In [47]:

```
print('Mean Absolute Error using SGD with l2 regularization:', mean_absolute_error(Y_test,
Y_pred_sgd_l2))
print('Mean Squared Error using SGD with l2 regularization:', mean_squared_error(Y_test,
Y_pred_sgd_l2))
print('Root Mean Squared Error using SGD with l2 regularization:', sqrt(mean_squared_error(Y_test,
Y_pred_sgd_l2)))
```

```
Mean Absolute Error using SGD with l2 regularization: 3.3693414342742187
Mean Squared Error using SGD with l2 regularization: 29.75040721068125
Root Mean Squared Error using SGD with l2 regularization: 5.454393386132068
```

8. Applying SGDRegressor using 'l1' Regularization on our Train and Test sets

In [48]:

```
# code source:https://medium.com/@haydar_ai/learning-data-science-day-9-linear-regression-on-bosto
n-housing-dataset-cd62a80775ef
from sklearn.linear_model import SGDRegressor
import matplotlib.pyplot as plt

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

lm2 = SGDRegressor(learning_rate = 'constant', penalty ='l1' , max_iter = 1000)
lm2.fit(Xtrain, Y_train)

Y_pred_sgd_l1 = lm2.predict(Xtest)
```

8.1. Printing Data Frame of Actual and Predicted values using SGDRegressor with 'l1' Regularization Model

In [49]:

```
df_sgd_l1 = pd.DataFrame({'Actual Values': Y_test,'Predicted Values': Y_pred_sgd_l1})
```

In [50]:

```
df_sgd_l1.head()
```

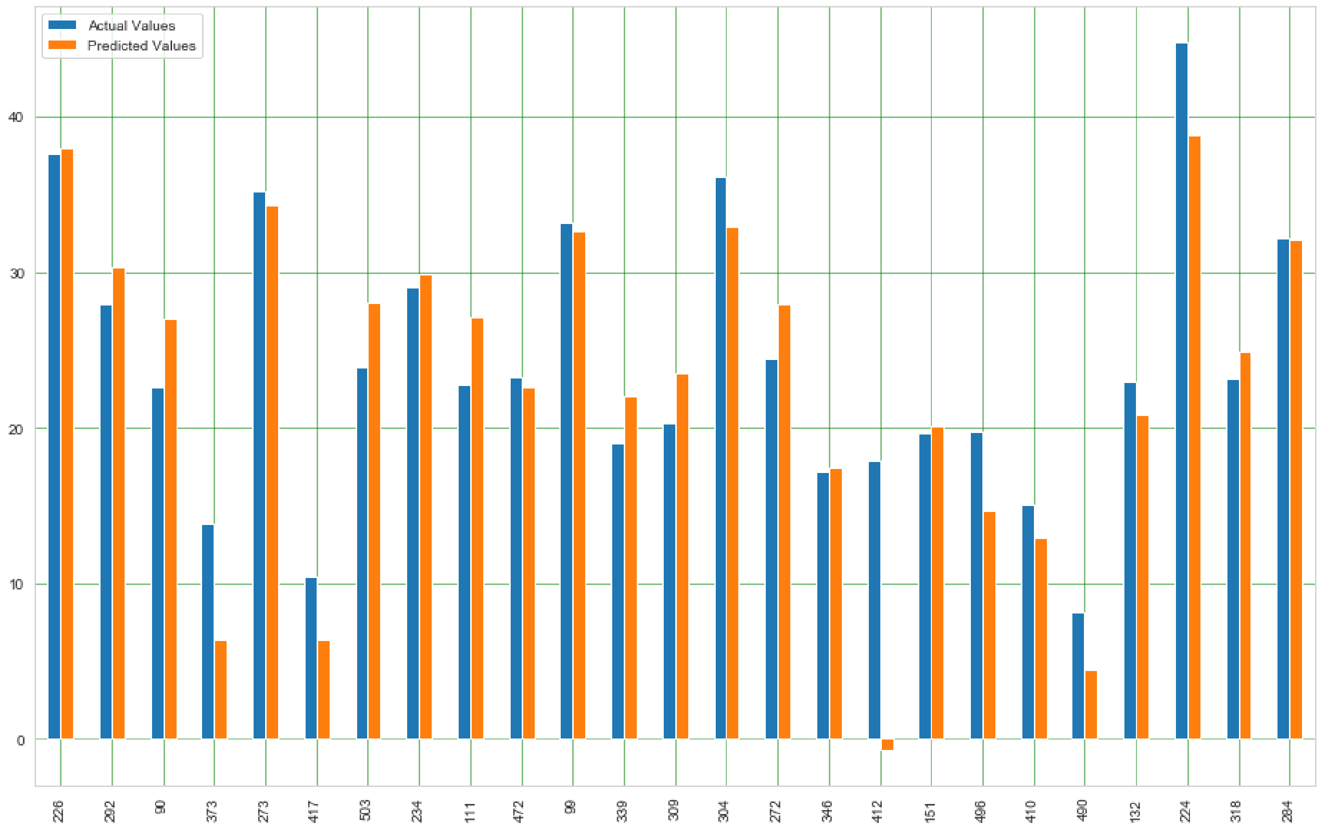
Out[50]:

	Actual Values	Predicted Values
226	37.6	38.023230
292	27.9	30.367456
90	22.6	27.022105
373	13.8	6.387844
273	35.2	34.325044

8.2. Visualizing Data using Bar Plots

In [51]:

```
df4 = df_sgd_l1.head(25)
df4.plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



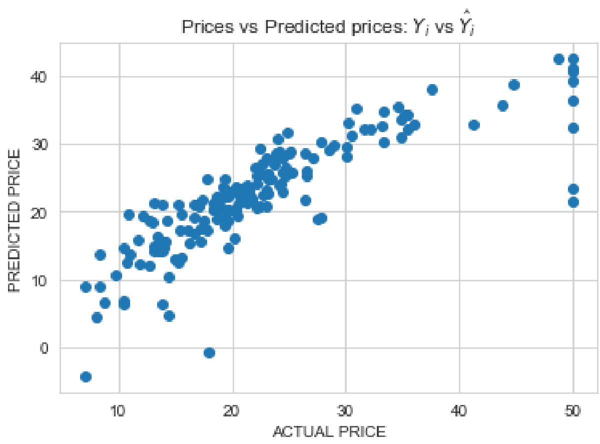
8.3. Visualizing Data using Scatter Plots

In [52]:

```
plt.scatter(Y_test, Y_pred_sgd_l1)
plt.xlabel("ACTUAL PRICE")
plt.ylabel("PREDICTED PRICE")
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
plt.show
```

Out[52]:

<function matplotlib.pyplot.show(*args, **kw)>

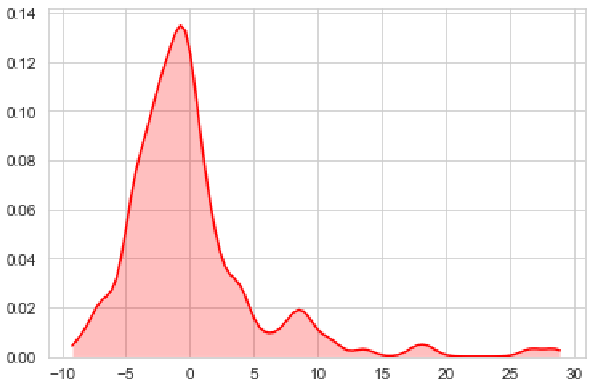


8.4. Visualizing Data usind Kernel Density(KDE) plot

In [53]:

```
delta_y = Y_test - Y_pred_sgd_l1;

import seaborn as sns;
import numpy as np;
sns.set_style('whitegrid')
sns.kdeplot(np.array(delta_y), bw=.15,shade=True, color="r")
plt.show()
```



The above KDE plot shows that our SGDRegressor with 'l1' regularization model is not working good as compared to the above two models as the width is more.

8.5. Finding Mean Absolute Error , Mean Squared Error , Root Mean Squared Error

In [54]:

```
print('Mean Absolute Error using SGD with l1 regularization:', mean_absolute_error(Y_test, 
Y_pred_sgd_l1))
print('Mean Squared Error using SGD with l1 regularization:', mean_squared_error(Y_test, 
Y_pred_sgd_l1))
print('Root Mean Squared Error using SGD with l1 regularization:', sqrt(mean_squared_error(Y_test, 
Y_pred_sgd_l1)))
```

```
Mean Absolute Error using SGD with l1 regularization: 3.4608270591915025
Mean Squared Error using SGD with l1 regularization: 28.581949753429775
Root Mean Squared Error using SGD with l1 regularization: 5.3462089141212745
```

9. Conclusion

Conclusion

In [56]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

x = PrettyTable()

x.field_names = [ "Model","Mean Absolute Error","Mean Squared Error", "Root Mean Squared Error"]


x.add_row(["Manual LinearRegression",3.3693, 28.5304,5.3543])
x.add_row(["LinearRegression",3.4550, 28.5304,5.3413])
x.add_row(["SGDRegressor using l2 regularization", 3.3693,29.7504, 5.4543])
x.add_row(["ASGDRegressor using l1 regularization", 3.4608 , 28.5819, 5.3462])
```

In [57]:

```
print(x)
```

```
+---------------------------------------+---------------------+--------------------+---------------
------+
|                 Model                 | Mean Absolute Error | Mean Squared Error | Root Mean Squa
ed Error |
+---------------------------------------+---------------------+--------------------+---------------
------+
|        Manual LinearRegression        |        3.3693       |      28.5304       |          5.354
|
|            LinearRegression           |        3.455        |      28.5304       |          5.341
|
|  SGDRegressor using l2 regularization |        3.3693       |      29.7504       |          5.454
|
| ASGDRegressor using l1 regularization |        3.4608       |      28.5819       |          5.346
|
+---------------------------------------+---------------------+--------------------+---------------
------+
```