In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import numpy as np;
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error as MSE
#getting boston house price data from sklearn
boston = load_boston()
```

In [2]:

```
#finding no of datapoints in boston data
print(type(boston['data']))
X = pd.DataFrame(data=boston['data'], columns=boston['feature_names']);
print("Shape of feature matrix: ", X.shape)
print(X.head(5))

print('*'*100)

y = pd.DataFrame(np.array(boston['target']))
print("Shape of output matrix", y.shape)
```

```
<class 'numpy.ndarray'>
Shape of feature matrix:  (506, 13)
      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
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
****************************************************************************************************

Shape of output matrix (506, 1)
```

In [3]:

```
scaler = preprocessing.StandardScaler().fit(X)
X = scaler.transform(X)
X = pd.DataFrame(data=X, columns=boston['feature_names'])
X.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.982843	-0.666608	-1.459000	0.441052	-1.075562
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.867883	-0.987329	-0.303094	0.441052	-0.492439
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.867883	-0.987329	-0.303094	0.396427	-1.208727
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.752922	-1.106115	0.113032	0.416163	-1.361517
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.752922	-1.106115	0.113032	0.441052	-1.026501

In [53]:

```
k = X.values;
k[1][2]
```

Out[53]:

```
-0.5933810131002436
```

In [12]:

```
class LinearRegression():
    def __init__(self, X, y, plot_error_per_iteration = False):
        self.X = X
        self.y = y #dimension: n*1
        self.data = pd.concat([X, y], axis=1);
        self.lr = 0.01 #defining learning rate
        self.b = np.random.normal(0,1,1);
        self.w = np.ones((self.X.shape[1], 1)) #generating intial weight vector of dimension 1*d, w
here d is no of columns/features
        self.MSE_list = [];
        self.plot_error_per_iteration = plot_error_per_iteration
        self.n_iterations=500


    def matrixMultiplication1(self, x, y):
#         y = pd.DataFrame(y)
        rows = x.shape[0];
        columns = y.shape[1];
        rows2 = y.shape[0]
        result = [];
        for i in range(rows):
            med_res = [];
            for j in range(columns):
                temp = 0;
                for k in range(rows2):
                    temp = temp + (x[i, k] * y[k, j]);
                med_res.append(temp);
                temp = 0;
            result.append(med_res);
        return np.array(result);

    def matrixMultiplication(self, x, y):
        a = np.tile(y[0] , (x.shape[0],1))
        res = x * a;
        res = np.sum(res, axis=1)
        res = res.reshape(-1, 1)
        return res;

    def computeDerivative(self, X, y):
        """
        function to obtain derivative value which is -2*x * (y - W_T * X)
        """
        X = X.values;
        y = y.values;
        wT_x = np.dot(X, self.w)#calculating x * w => multiplying X(n*d - dim) with weight(d*1), ou
tput will be of n*1 - dim
        y_wT_x = y - (wT_x + self.b) #calculating y - (w_t * x) => output will be of dimension n*1
        x_prod = -2 * X;
        der = (x_prod*y_wT_x).sum(axis=0).reshape(-1,1) #der will be of dimension 1 * d
#         print('der ', type(der))
        #print('der',der.shape)
        return self.w - self.lr * (der / y.shape[0]);


    def computeIntercept(self, X, y):
        """
        function to compute new intercept
        """
        X = X.values;
        y = y.values;
        wT_x = np.dot(X, self.w)#calculating x * w => multiplying X(n*d - dim) with weight(d*1), ou
tput will be of n*1 - dim
        y_wT_x = y - (wT_x + self.b)#calculating y - (w_t * x) => output will be of dimension n*1
        y_prod = -2 * y_wT_x
        #print(y_prod.shape)
```

```
        intercept = y_prod.sum(axis=0).reshape(-1,1)
        #print('intercept',intercept.shape)
        #intercept = intercept[0]
        diff_in_intercept = self.b - self.lr * (intercept / y_prod.shape[0])
#         print(type(diff_in_intercept))
        return diff_in_intercept

    def predict(self, X):
        """
        function to predict output given query points
        """
        wT_x = np.dot(self.w.T, X.T)
        y = wT_x + self.b;
        return y.T

    def addMSE(self, ):
        """
        function to add mean squared error for each W, obtain in each iteration
        """
        y_pred = self.predict(self.X);
        self.MSE_list.append(MSE(y, y_pred));

    def plotError(self, ):
        """
        function to plot error
        """
        plt.plot(np.arange(0, len(self.MSE_list)), self.MSE_list);
        plt.ylabel('MSE');
        plt.xlabel('iterations')
        plt.grid();
        plt.show();

    def fit(self, ):
        while self.n_iterations>0:
            sample_data = self.data.sample(n=30)
            X = sample_data.loc[:, sample_data.columns != 0];
            y = sample_data.loc[:, [0]]
            new_w = self.computeDerivative(X, y);
            new_b = self.computeIntercept(X, y);
            if (np.all(new_w - self.w) < 0.00001 and np.all(new_b - self.b) < 0.00001):
                self.w = new_w
                self.b = new_b
                if self.plot_error_per_iteration:
                    self.addMSE();
                    self.plotError();
                break
            else:
                #self.lr /= 2
                self.w = new_w
                self.b = new_b
                self.n_iterations-=1
                if self.plot_error_per_iteration:
                    self.addMSE()
```

In [13]:

```
LR = LinearRegression(X, y, plot_error_per_iteration=True);
LR.fit()
y_pred = LR.predict(X);
self_model_error = MSE(y, y_pred)
print(self_model_error)
```

22.368216627128106

In [14]:

```
from sklearn import linear_model
model = linear_model.SGDRegressor(penalty='none');
model.fit(X, y);
y_pred = model.predict(X);
sgd_model_error = MSE(y, y_pred)
print(sgd_model_error)
```

22.87212701556153

22.87212701556153

In [15]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Model", "Mean squared error"]
x.add_row(["SGDRegressor", sgd_model_error])
x.add_row(["our created model",self_model_error ])
print(x)
```

```
+-------------------+--------------------+
|       Model       | Mean squared error |
+-------------------+--------------------+
|    SGDRegressor   | 22.87212701556153  |
| our created model | 22.368216627128106 |
+-------------------+--------------------+
```

In [ ]:

```

```