*) for Boston Dataset, the important feature might be age.
So we prepare the dummy feature to be age * age.

↓

Load the boston dataset from scikit learn into a dataframe

↓

```
from sklearn import model_selection

xTrain, xTest, yTrain, yTest =
        model_selection.train_test_split(X, Y,
                                         random_state=0)

x2Train, x2Test, y2Train, y2Test =
        model_selection.train_test_split(X2, Y,
                                         random_state=0)
```

↓

Comparing these w/o the random state is not the right way to go about it as the same rows won't go on both sides and fitting would be difficult as per different data.

↓

We want to keep all other constraints same, so that the dummy introduction can be accurately compared or evaluated.

↓

compare scores for both datasets.

→ coding gradient descent

```
import numpy as np
data = np.loadtxt("data.csv", delimiter=",")

x = data[:,0]
y = data[:,1]

def step_gradient(x_train, y_train, lr, m, c):

    m_slope = 0
    c_slope = 0
    N = x_train.shape[0]
    for i in range(N):
        m_slope += (2/N) * (y_train[i]
                   - m * x_train[i] - c)
                   * (-x_train[i])

        c_slope += (2/N) * (y_train[i]
                   - m * x_train[i] - c)
                   * (-1)

    m = m - lr * slope_m
    c = c - lr * slope_c

    return m, c

def fit(x_train, y_train, lr=0.0001, epoche=100,
        decay_point=10000, decay_factor=5,
        verbose=false):

    m = 0
    c = 0
```

```
error_array = []
for i in range(epochs):
    m, c = step_gradient(X_train, Y_train,
                         lr, m, c)
    error_array.append(cost(X_train, m, c,
                            Y_train))
    if verbose and (epochs % 10 == 0):
        print(cost(X_train, m, c, Y_train))
    if i % decay_point == 0:
        lr /= 10
        decay_point *= decay_factor
error = cost(X_train, m, c, Y_train)
return m, c, error, error_array
```

→ Generic gradient descent

In the fit function, for vector based computation the multiplication between m * X_train[i], we need to take sum, and then multiply the resultant scalar with outside vector.

```
m_slope += (2/N) * (Y_train[i] -
                    (X_train[i] * m).sum())
                    * (-X_train[i])
```

```
cost_ += (1/N) * ((y_train[i] -
                   ((X_train[i] * m).sum()))
                   ** 2)
```