

# Linear Regression in Python

MACHINE LEARNING

***CODING***  
**<LANE>**

BY JAY PATEL

# Linear Regression in Python

Let's first start by importing our dependencies . We're going to import numpy and matplotlib libraries.

Numpy is basically useful when we are dealing with the matrices and matplotlib is useful when we are dealing with the graphs.

If you don't know anything about Numpy then click on this video tutorial link from freeCodeCamp : [Numpy freeCodeCamp](#)

Numpy is super useful while we are doing machine learning in python so make sure you have the knowledge about that before you begin.

```
import numpy as np
import matplotlib.pyplot as plt
```

so once we have our dependencies we are going to get our data set. I already have made a data set which is not real but i have made it myself. You can get this dataset from : [Here : Dataset](#)

```
data = np.loadtxt("dataset.txt", delimiter = ",")
```

okay so our data set has x on one side and y on the other side. so let's say this is the square foot area of the house and this is the price of the house.

---

so we are making a very simple model which will predict the price of the house based on the square foot area.

Now what we want is that we want to have a column of ones before the column of the square foot area. We also want our y to be in a proper shape

```
X = data[:, 0]
Y = data[:, 1].reshape(X.size, 1)
X = np.vstack((np.ones((X.size, )), X)).T
```

let's see how our shape of the x looks like and how our shape of the y looks like.

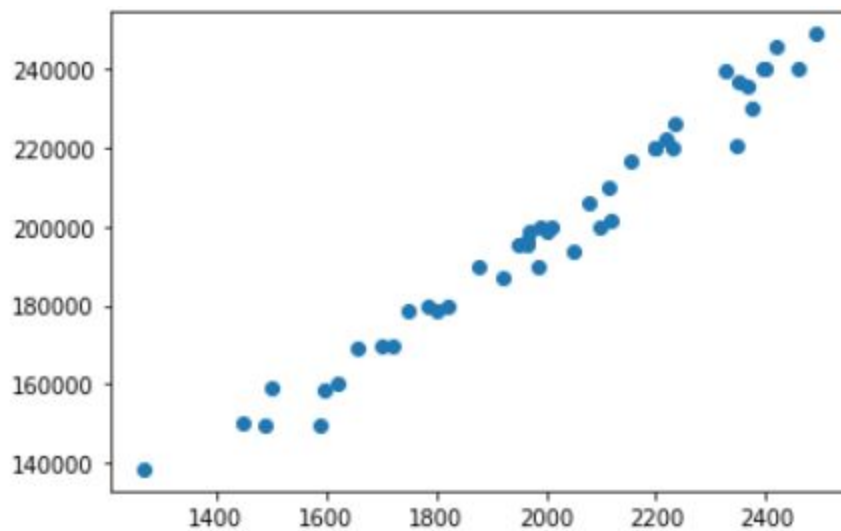
```
print(X.shape)
print(Y.shape)
```

```
(45, 2)
(45, 1)
```

Here 45 is the size of the data set.

okay that's great now let's plot this into a graph and see how it looks like.

```
plt.scatter(X[:, 1], Y)
plt.show()
```



okay now let's have a quick overview of linear regression. We know that in the linear regression we make predictions by plotting a straight line that approximately fits our data set. It also makes the use of the cost function which will determine the error between the predicting value and the actual value. We want this cost function which is the representation of the error to be minimum. For the minimum value of the cost function we need to use a gradient descent algorithm. And we're going to run this in a loop which will in every iteration decrease the cost values and reach to our local minimum

## Overview of Linear Regression

### Straight Line

$$y = \theta_1 x_1 + \theta_2$$

### Cost Function

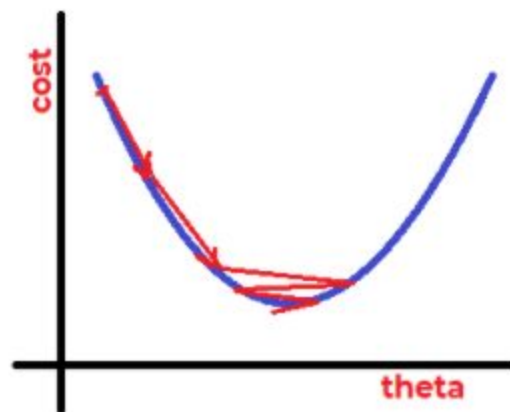
$$y\_pred = matrix\_mul(X, theta)$$

$$cost = \frac{1}{2m} \sum [(y\_pred - Y)^2]$$

### Gradient Descent

$$d\_theta = \frac{1}{m} [matrix\_mul(X^T, y\_pred - Y)]$$

$$theta = theta - \alpha * d\_theta$$




if you want to know more about the cost function click [Here : Cost Function](#) which will take you to the video which has a detailed explanation on the cost function

And if you want to know about the gradient descent then click [Here : Gradient Descent](#) which will take you to the gradient descent algorithm.

Now let's move ahead.

We are going to make the complete model of the linear linear regression and i'm going to take four parameters for this and they are going to be our x, y, learning rate, and the iterations.



Learning rate determines how fast we want to train. I will get to this learning rate again. And iteration determines how long how many times you want to run the loop.

theta is going to be a vector of zeros of shape (2, 1)

And at the end return the 'theta' parameter which will be trained.

```
def model(X, Y, learning_rate, iteration):
    m = Y.size
    theta = np.zeros((2, 1))
    cost_list = []

    for i in range(iteration):

        y_pred = np.dot(X, theta)
        cost = (1/(2*m))*np.sum(np.square(y_pred - Y))

        d_theta = (1/m)*np.dot(X.T, y_pred - Y)
        theta = theta - learning_rate*d_theta

        cost_list.append(cost)

    return theta, cost_list
```

I have also added cost\_list which will store the cost value at every iteration. This will be used to plot the graph between cost function and iteration.

Now lets train our model and see how it works ! Say i'm running a loop for 100 and let's say i have a learning rate of 0.00000005. I'm going to have a very small learning rate and the learning rate should be actually always small.

We are going to get more about the learning rate in a while. Let's first call our model.

```
iteration = 100
learning_rate = 0.00000005
theta, cost_list = model(X, Y, learning_rate = learning_rate,
iteration = iteration)
```

Our model has been run so let's try to predict any value.

```
new_houses = np.array([[1, 1547], [1, 1896], [1, 1934], [1,
2800], [1, 3400], [1, 5000]])
for house in new_houses :
    print("Our model predicts the price of house with",
house[1], "sq. ft. area as : $", round(np.dot(house, theta)[0],
2))
```

Our model predicts the price of house with 1547 sq. ft. area as :  
\$ 153759.0

Our model predicts the price of house with 1896 sq. ft. area as :  
\$ 188446.7

Our model predicts the price of house with 1934 sq. ft. area as :  
\$ 192223.59

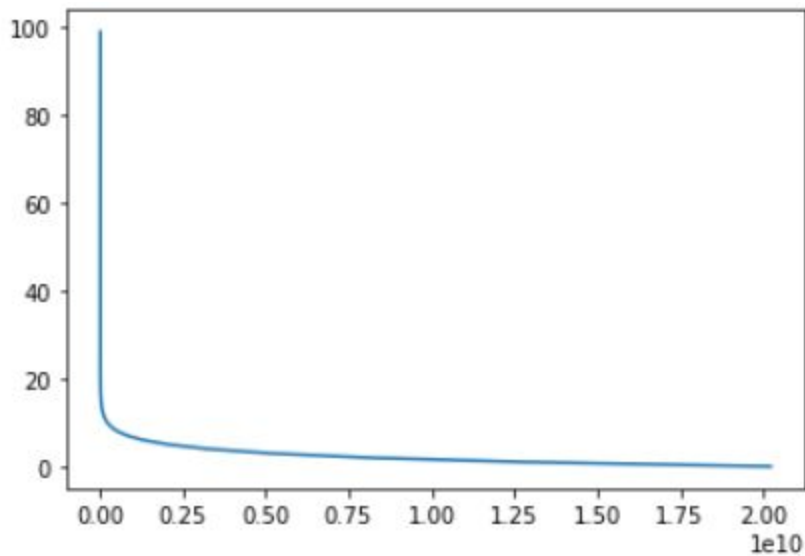
Our model predicts the price of house with 2800 sq. ft. area as :  
\$ 278296.8

Our model predicts the price of house with 3400 sq. ft. area as :  
\$ 337931.81

Our model predicts the price of house with 5000 sq. ft. area as :  
\$ 496958.52

Lets plot our cost vs iteration graph.

```
rng = np.arange(0, iteration)
plt.plot(cost_list, rng)
plt.show()
```



That was our linear regression model !!

What will happen if we change the value of learning\_rate and iterations ? This can be a home work for you, try changing the learning rate to be 0.00005 and see what happens.

**Subscribe to Coding Lane for more such amazing contents :**

[Click here : Coding Lane](#)

■ ■ ■



***CODING***  
***<LANE>***