Housing price prediction without regularization.

Dataset Description:

In this assignment, we had to predict housing price given on the dataset. Dataset contains 546 rows and 12 columns. Out of which 5 columns are categorical features

Categorical features have not been considered for predicting the housing price.

Data Pre Processing:

After removing categorical features our data will look like.

```
In [6]: df.describe()
Out[6]:
```

	price	lotsize	bedrooms	bathrms	stories	garagepl
count	50.000000	50.000000	50.000000	50.000000	50.000000	50.000000
mean	78609.000000	5982.240000	2.900000	1.240000	1.720000	0.780000
std	24180.743076	2180.983148	0.580288	0.431419	0.729551	0.910035
min	31900.000000	2850.000000	2.000000	1.000000	1.000000	0.000000
25%	64625.000000	4117.500000	3.000000	1.000000	1.000000	0.000000
50%	75000.000000	6020.000000	3.000000	1.000000	2.000000	0.000000
75%	89975.000000	6852.750000	3.000000	1.000000	2.000000	2.000000
max	174500.000000	11440.000000	4.000000	2.000000	4.000000	3.000000

The normalization technique, I have used is mean shifting and variance scaling. In which we shift the mean and scale it by variance.

```
X = (X - np.mean(X))/(np.std(X))
```

After normalization, I have added a bias term for the further calculation. Solution from normal equation gives us values of theta row vector.

```
X = np.c_[np.ones(X.shape[0]),X]
```

```
theta = np.matmul(np.linalg.inv(np.matmul(X.T,X)),np.matmul(X.T,y))
print(theta)
```

```
[78609.          7350.84253883   2923.45007788 10651.20328512
  5427.00783509  6414.41197345    160.76852306  4855.06325266
  2776.95324847    884.08217151   4619.18067772]
```

Solution using gradient descent:

It's an iterative approach to minimize the cost function. In this algorithm we initialize the weights by random values and try to find the minima of the function by moving in a direction opposite to the gradient of the function. Below implementation of this algorithm has bene done by me.

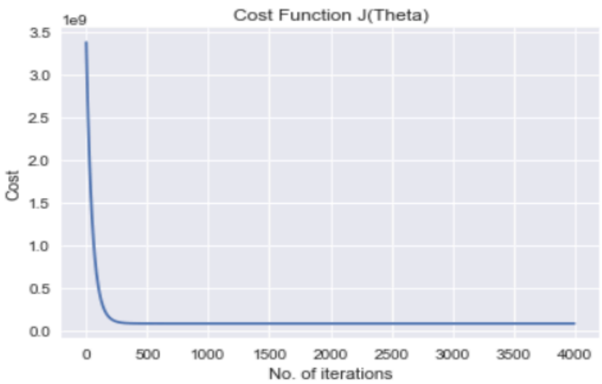
```
def gradient_descent(x,y,theta,num_iterations,alpha):
    past_costs = []
    past_thetas = []
    for i in range(num_iterations):
        prediction = np.dot(x,theta)
        error = prediction - y
        cost = 1 / (2 * m) * np.dot(error.T, error)
        past_costs.append(cost)
        theta = theta - (alpha * (1/m) * np.dot(x.T, error))
        past_thetas.append(theta)
    return past_thetas, past_costs
```

```
alpha = 0.01
iterations = 4000
m = y.size
np.random.seed(123)
theta1 = np.random.rand(11)
```

```
weight,cost = gradient_descent(X,y,theta1,iterations,alpha)
z = weight[-1]
print(z)
```

As we increase the number of iterations cost function decreases first and after some times, it becomes constant.

```
plt.title('Cost Function J(Theta)')
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.plot(cost)
plt.show()
```



After this, we can take the dot product of theta with the feature vector and calculate the housing price.

```
predicted_val = []
for data in X:
    predicted_val.append(np.dot(data,z))
```

After calculating the predicted value, we can compare it with the actual value and calculate the mean squared error.

```
squared_error=np.sqrt(np.sum((new_data['price']-new_data['Predicted_val'])**2)/new_data.shape[0

print(squared_error)
```

```
12376.023550581374
```