Housing Price using regularized linear regression.

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. I am using regularization for housing price prediction. I have used L2 regularization which is also known as Ridge regression.

```
df = pd.read_csv('housing_price.csv')
```

```
df.head()
```

	price	lotsize	bedrooms	bathrms	stories	driveway	recroom	fullbase	gashw	airco	garagepl	prefarea
0	42000.0	5850	3	1	2	yes	no	yes	no	no	1	no
1	38500.0	4000	2	1	1	yes	no	no	no	no	0	no
2	49500.0	3060	3	1	1	yes	no	no	no	no	0	no
3	60500.0	6650	3	1	2	yes	yes	no	no	no	0	no
4	61000.0	6360	2	1	1	yes	no	no	no	no	0	no

Pre Processing:
I have used min-max normalization strategy for normalizing our data. Min-Max strategy shifts the data by min and then scale it by a difference of max and min, And also a bias term has been added to the dataset.
The value of theta can be calculated same as using the normal equation with the addition of one regularizer term.

```
X = (X - np.min(X)) / (np.max(X) - np.min(X))
```

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

```
lemda = 10
a = np.identity(X.shape[1])
a[0][0] = 0
theta = np.matmul(np.linalg.inv(np.matmul(X.T,X) + lemda * a),np.matmul(X.T,y))
print(theta)
```

```
[40258.60166749 38571.11193726 12725.66543547 32027.71391458
 22746.16796064 19101.72772245]
```

Predicting values using normal equation:
Dot product of theta and feature will give the predicted housing price for a given feature. And Root mean squared error can be calculated for the same.

```
predicted_val = np.matmul(X,theta.T)
```

```
df['prediction'] = predicted_val
```

```
df['Squared_error'] = (df['price'] - df['prediction']) ** 2
```

```
mean_error = np.sum(df['Squared_error']) * (1 / (df.shape[0]))
```

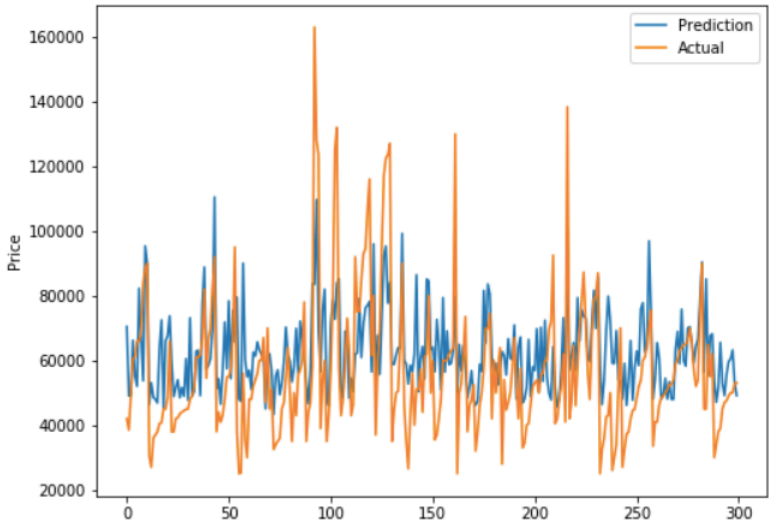
```
print(np.sqrt(mean_error))
```

```
18503.829230050946
```

Plotting:

I have also plotted actual value and predicted value to visualize results in a better way.

```
figure = plt.figure(figsize=(8,6))
plt.plot(predicted_val[0:300],label = 'Prediction')
plt.plot(y[0:300],label='Actual')
plt.xlabel('Data points')
plt.ylabel('Price')
plt.legend()
plt.show()
```



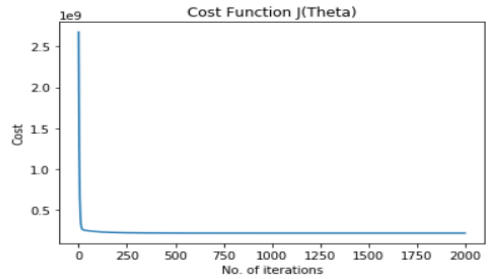
Predicting Prices using gradient Descent:

Gradient descent algorithm can be modified a bit to implement the regularized gradient descent algorithm. Below is my implementation of gradient descent.

```
def gradient_descent(x,y,theta,num_iterations,alpha,lemda):
    past_costs = []
    past_thetas = []
    lembda_mat = lemda * np.identity(X.shape[1])
    lembda_mat[0][0] = 0
    for i in range(num_iterations):
        prediction = np.dot(x,theta)
        error = prediction - y
        cost = 1 / (2 * m) * ( np.dot(error.T, error) + lemda * np.dot(theta.T, theta))
        past_costs.append(cost)
        theta = theta - (alpha * (1/m) * ( np.dot(x.T, error) + np.matmul(lembda_mat,theta)))
        past_thetas.append(theta)
    return past_thetas, past_costs
```

I am keeping track of theta and cost in every iteration for plotting the cost vs. iteration curve and last value of theta will be taken for the prediction.

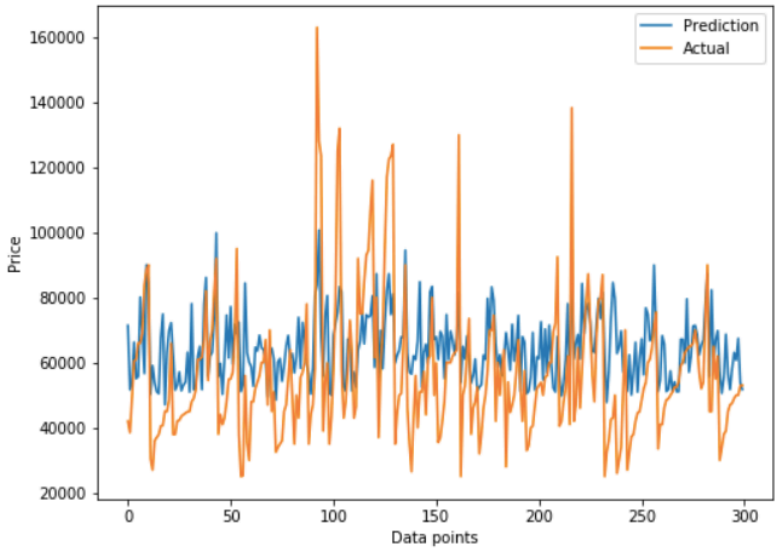
```
plt.title('Cost Function J(Theta)')
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.plot(cost)
plt.show()
```



Cost vs. Number of iterations

Similarly prediction can be done for by taking dot product and we can plot the actual value vs predicted value.

```
figure = plt.figure(figsize=(8,6))
plt.plot(predicted_val[0:300],label = 'Prediction')
plt.plot(y[0:300],label='Actual')
plt.xlabel('Data points')
plt.ylabel('Price')
plt.legend()
plt.show()
```



Observation:
While using the polynomial features our model is prone to overfitting, i.e will perform better on training set while perform worst on test set. To overcome this, we add a regularization term to introduce some nor-linearity in the model and it helps in generalizing the model.

Locally weighted Regression:

It is also known as non-parameterized regression. In this algorithm we fit different curve for different regions and hence it is known as locally weighted regression.
Cost function and gradient Descent for Locally weighted regression can be implemented as

In [66]:
```
def cost(xl, x, y, w,t = 1):
    m = len(y)
    e = (np.dot(x, w) - y) ** 2
    tau = np.array(np.exp(-1 * np.sum(((x - xl) **2), axis = 1)/(2 * t))).reshape(-1,1)
    return np.sum(np.multiply(tau,e))
```

In [68]:
```
def gradient_descend(x, y, alpha, xl, iterations, t = 1):
    cost_list = []
    w = np.zeros((6,1))

    for i in range(iterations):
        tau = np.array(np.exp(-1 * np.sum(((x - xl) **2), axis = 1)/(2 * t)  )).reshape(-1,1)
        grade = (np.dot(x.T, np.multiply(np.dot(x,w) - y, tau)) )
        w = w - alpha * grade
        cost_list.append(cost(xl, x, y, w, t))
    return cost_list, w
```

I have calculated the weights for different values of G and below observation has been found.

Observation:

When value of tau is very small, it will overfit the model because width of the curve will be small and very few points will be responsible for the training. And hence, value of actual cost and predicted costs will be same.