ml-regresion-multiple-clase-1

February 4, 2024

0.1 MULTIPLE LINEAR REGRESSION USING GRADIENT DESCENDET FROM SCRATCH

[1]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

[42]:

```python
data = pd.read_csv('oregon_houses.csv')
data.head()
#---------------------------------------------------------
```

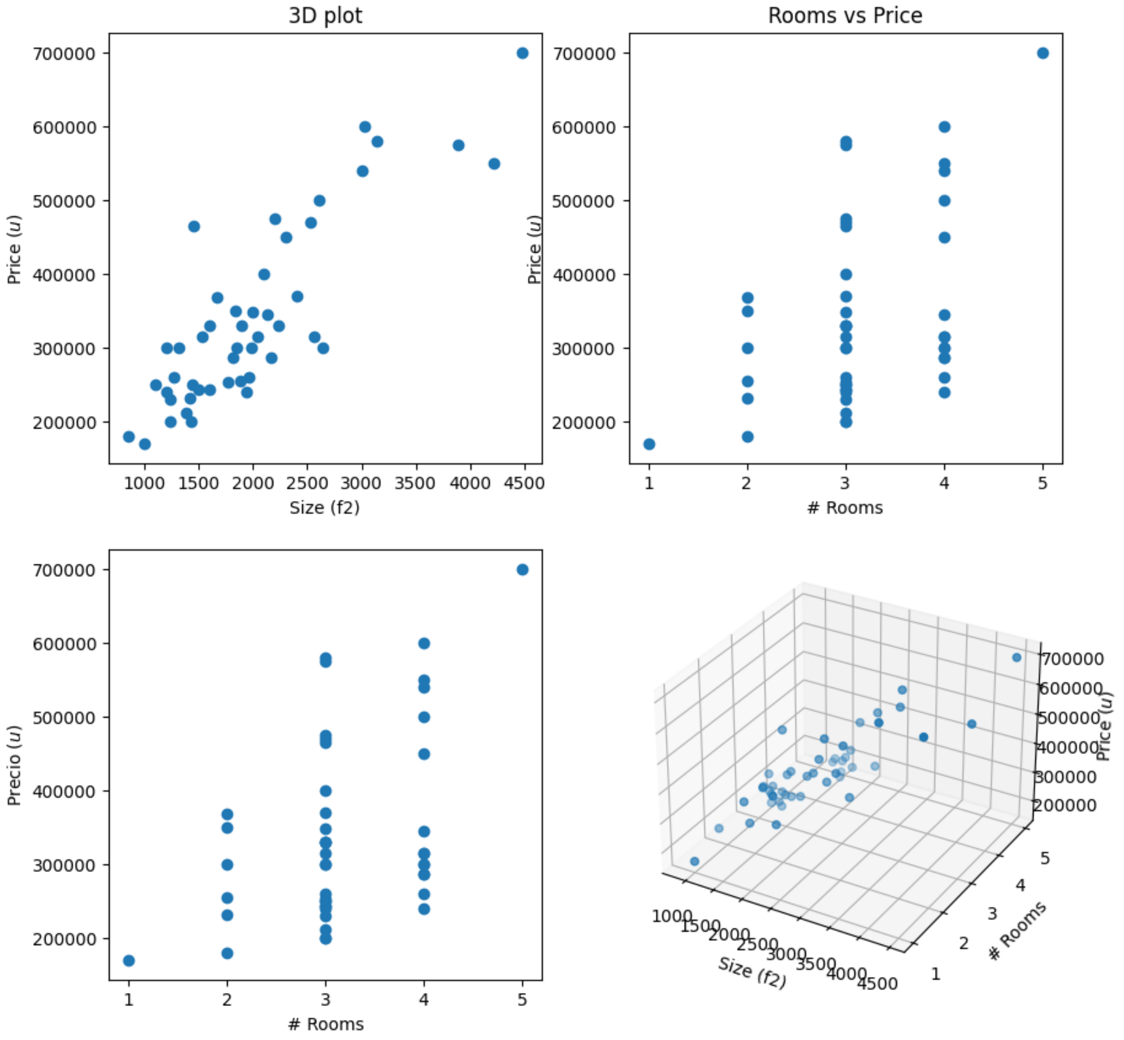
[42]:

```
   size(in square feet)  number of bedrooms   price
0                  2104                   3  399900
1                  1600                   3  329900
2                  2400                   3  369000
3                  1416                   2  232000
4                  3000                   4  539900
```

[43]:

```python
X = data.values[:, 0:2]
Y = data.values[:, 2]

# Plot
fig = plt.figure(figsize=(10, 10))

# 1
ax1 = fig.add_subplot(221)
ax1.set_title("Size vs Price")
ax1.scatter(X[:,0], Y)
ax1.set_xlabel("Size (f2)")
ax1.set_ylabel("Price ($u$)")

# 2
ax2 = fig.add_subplot(222)
ax2.set_title("Rooms vs Price")
ax2.scatter(X[:,1], Y)
ax2.set_xlabel("# Rooms")
```

```
ax2.set_ylabel("Price ($u$)")

# 3
ax3 = fig.add_subplot(223)
ax1.set_title("Rooms vs Price")
ax3.scatter(X[:,1], Y)
ax3.set_xlabel("# Rooms")
ax3.set_ylabel("Precio ($u$)")

# 4
ax4 = fig.add_subplot(224, projection='3d')
ax1.set_title("3D plot")
ax4.scatter(X[:,0],X[:,1],Y)
ax4.set_xlabel("Size (f2)")
ax4.set_ylabel("# Rooms")
ax4.set_zlabel("Price ($u$)")

plt.show()

#---------------------------------------------------------
```



0.1.1 Normalize Data

[44]:

```
# --- Mean
mu = np.mean(X, axis = 0)
print("La media de X es igual a: "+str(mu))

# --- standard deviations
sigma = np.std(X, axis= 0, ddof = 1)
print("La desviación estandar de X es igual a: "+str(sigma))

# --- Matrix X normalized
X_norm = (X - mu)/sigma
print("La matriz X normalizada es igual a: ")
print(np.around(X_norm,2))
#----------------------------------------------------------
```

```
La media de X es igual a: [2000.68085106    3.17021277]
La desviación estandar de X es igual a: [7.94702354e+02 7.60981887e-01]
La matriz X normalizada es igual a:
[[ 0.13 -0.22]
 [-0.5  -0.22]
 [ 0.5  -0.22]
 [-0.74 -1.54]
 [ 1.26  1.09]
 [-0.02  1.09]
 [-0.59 -0.22]
 [-0.72 -0.22]
 [-0.78 -0.22]
 [-0.64 -0.22]
 [-0.08  1.09]
 [-0.   -0.22]
 [-0.14 -0.22]
 [ 3.12  2.4 ]
 [-0.92 -0.22]
 [ 0.38  1.09]
 [-0.86 -1.54]
 [-0.96 -0.22]
 [ 0.77  1.09]
 [ 1.3   1.09]
 [-0.29 -0.22]
 [-0.14 -1.54]
 [-0.5  -0.22]
 [-0.05  1.09]
 [ 2.38 -0.22]
 [-1.13 -0.22]
 [-0.68 -0.22]
 [ 0.66 -0.22]
 [ 0.25 -0.22]
 [ 0.8  -0.22]
 [-0.2  -1.54]
 [-1.26 -2.85]
 [ 0.05  1.09]
 [ 1.43 -0.22]
 [-0.24  1.09]
 [-0.71 -0.22]
 [-0.96 -0.22]
 [ 0.17  1.09]
 [ 2.79  1.09]
 [ 0.2   1.09]
 [-0.42 -1.54]
 [ 0.3  -0.22]
 [ 0.71  1.09]
 [-1.01 -0.22]
 [-1.45 -1.54]
 [-0.19  1.09]
 [-1.   -0.22]]
```

[45]:

```
X_or = X
X = np.hstack((np.ones((len(Y),1)), X_norm))
```

0.1.2 Train

[95]:

```
# --- Regression coeficient
w = np.zeros(X.shape[1])

# --- Train parameters
n = 0.000001 #  (stepsize)
epochs = 100000  # Iterations
N = float(len(Y)) # Number of elements

# --- error
Error = [0] * epochs
T = np.arange(1, epochs + 1)
#--------------------------------------------------------
```

0.1.3 Training process

[96]:

```
# --- Iterations
for i in range(epochs):
    Y_pred = X.dot(w)  # predict Y -- Hw
    error = np.subtract(Y, Y_pred)  # Error -- Y-Hw
    RSS = error.transpose().dot(error)  # Performance -- (Y-Hw)^t (Y-Hw)
    Error[i] = RSS  # Almacenar el error
    D_w = -2 * X.transpose().dot(error)  # Gradient -- 2HT(Y-Hw)
    w = w - n * D_w  # Gradient

    #--------------------------------------------------------
```

[97]:

```
print("Training Result: ")
print(w)
#--------------------------------------------------------
```

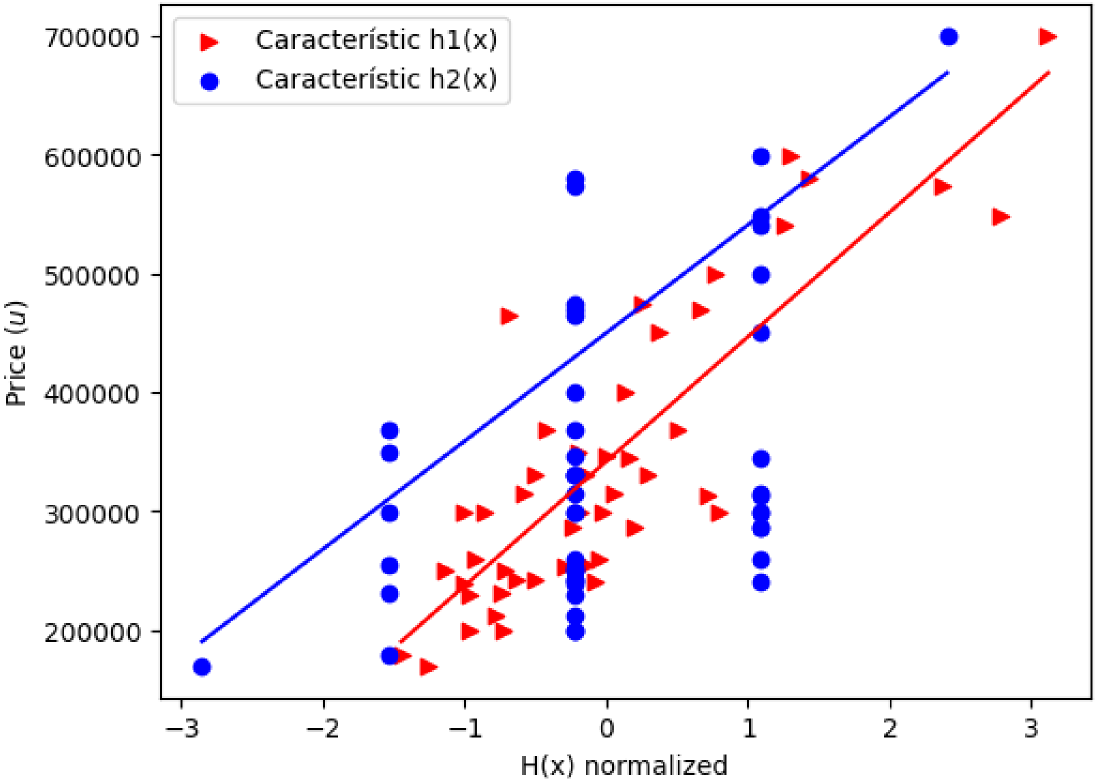
```
Training Result:
[340384.51169456 109607.71461149  -5626.19936737]
```

0.1.4 Model

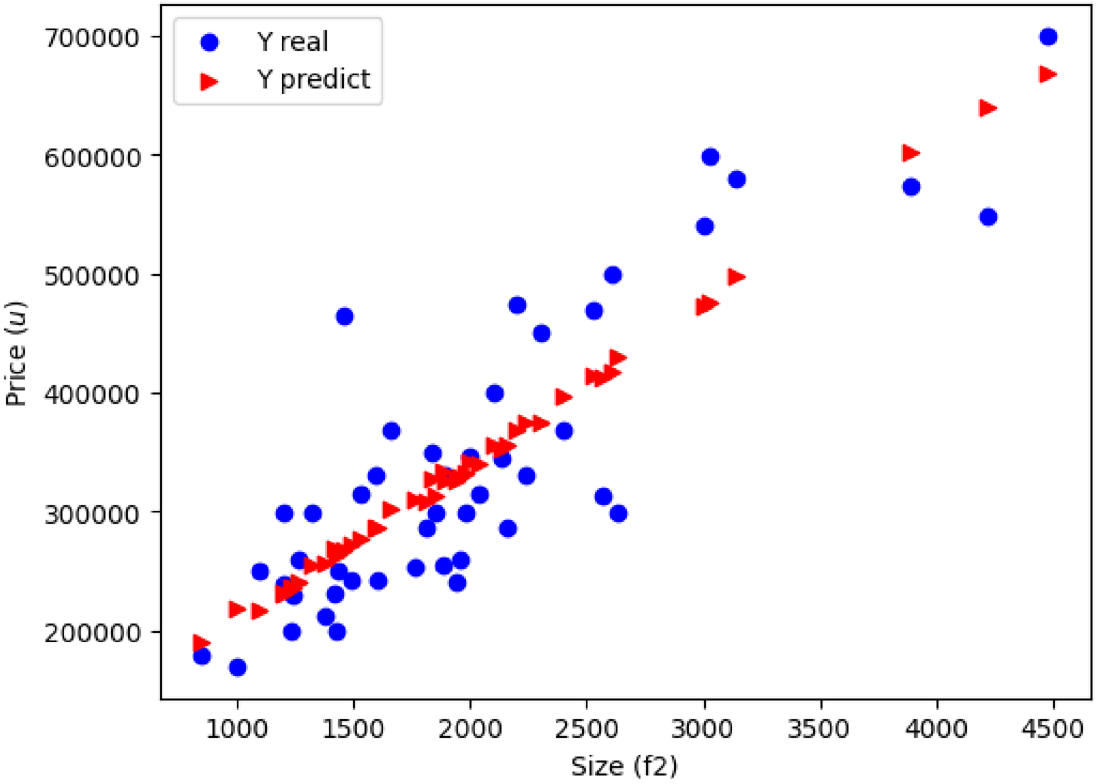
[98]:

```
# --- Graph 2D

plt.scatter(X[:,1], Y, marker='>', color = 'red', label = 'Característic␣
 ↪h1(x)') # Datos h1(x)
```

```
plt.plot([min(X[:,1]), max(X[:,1])], [min(Y_pred), max(Y_pred)], color='red') #␣
↪Linea de regresion h1(x)
plt.scatter(X[:,2], Y, marker='o', color = 'blue',label = 'Característic␣
↪h2(x)') # Datos h2(x)
plt.plot([min(X[:,2]), max(X[:,2])], [min(Y_pred), max(Y_pred)], color='blue')␣
↪# Linea de regresion h2(x)
plt.xlabel("H(x) normalized")
plt.ylabel("Price ($u$)")
plt.legend()
plt.show()
#--------------------------------------------------------
```

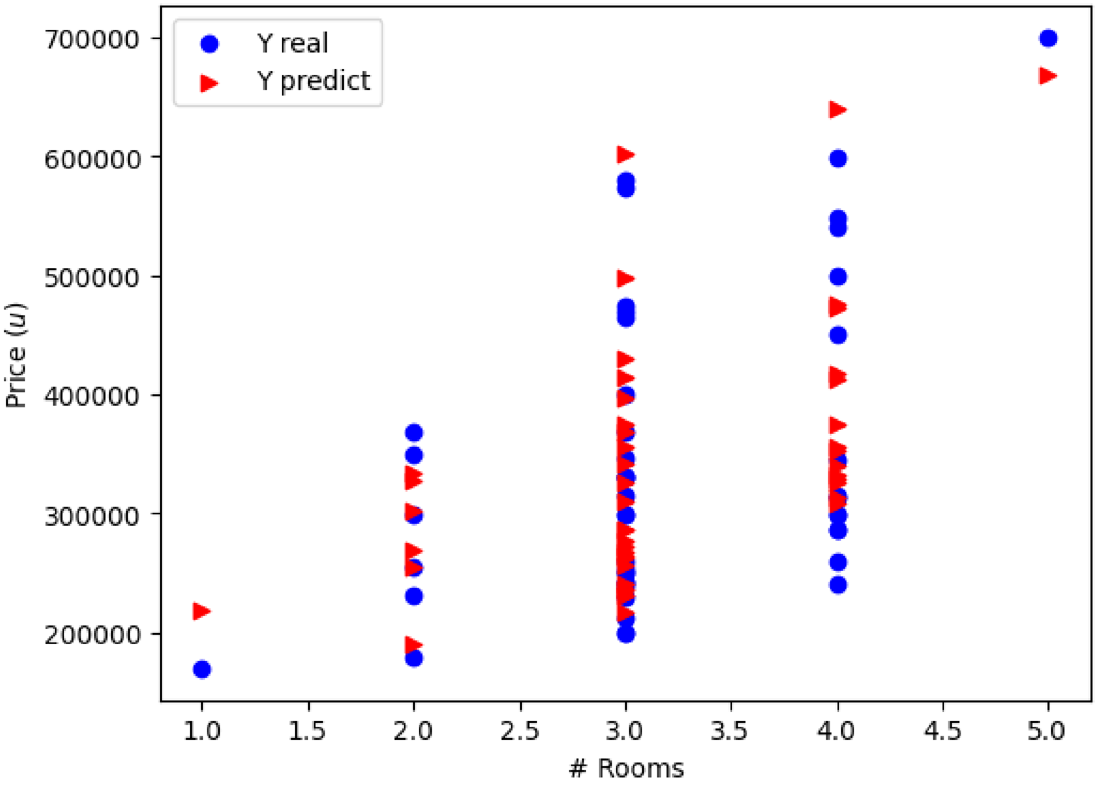


[99]:
```
plt.scatter(X_or[:,0], Y, marker='o',
            color = 'blue', label = 'Y real') #Real data
plt.scatter(X_or[:,0], Y_pred, marker='>',
            color = 'red', label = 'Y predict') # Estimated data
plt.xlabel("Size (f2)")
plt.ylabel("Price ($u$)")
plt.legend()
plt.show()
#--------------------------------------------------------
```



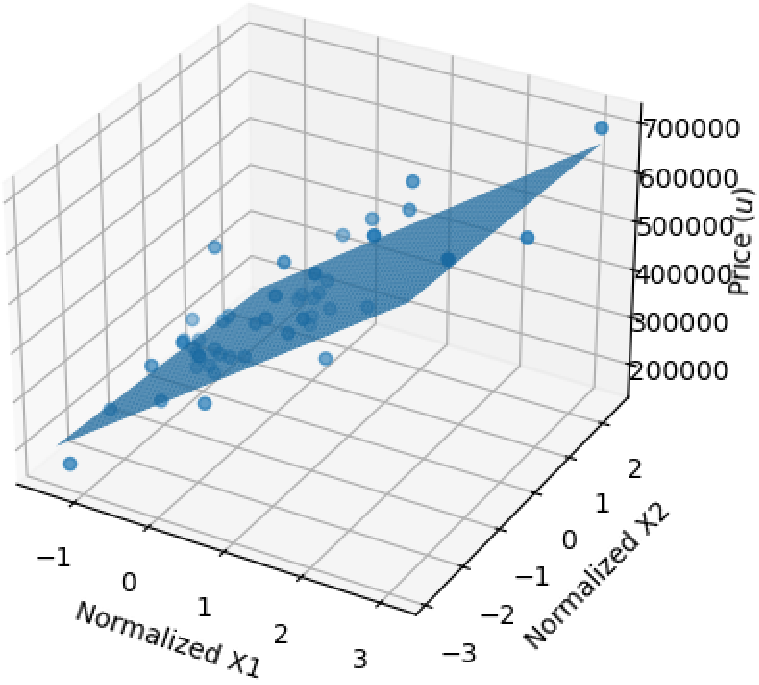
[100]:

```
plt.scatter(X_or[:,1], Y, marker='o',
            color = 'blue', label = 'Y real') # Real data
plt.scatter(X_or[:,1], Y_pred, marker='>',
            color = 'red', label = 'Y predict') # Estimated data
plt.xlabel("# Rooms")
plt.ylabel("Price ($u$)")
plt.legend()
plt.show()
#--------------------------------------------------------
```



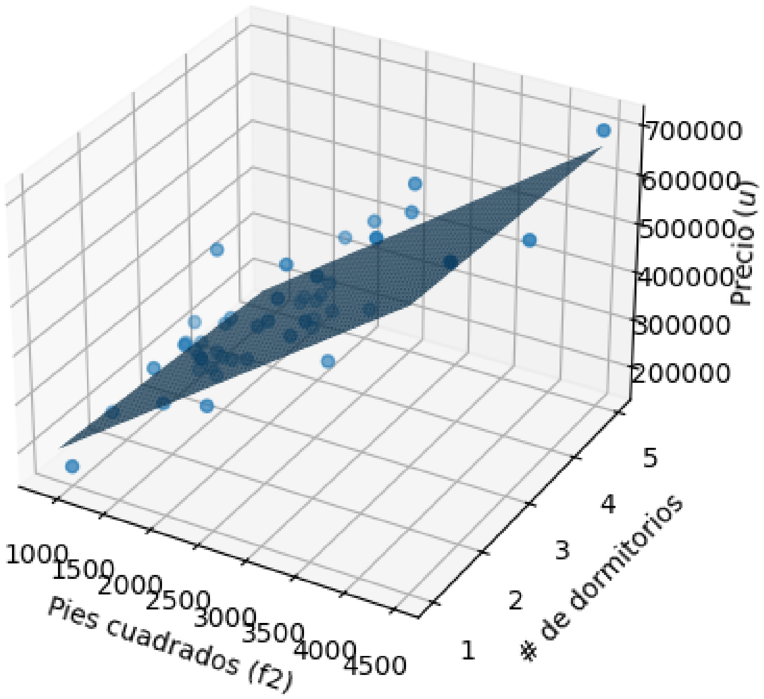
[101]:

```
# --- 3D plot


x = np.linspace(min(X[:,1]), max(X[:,1]))
y = np.linspace(min(X[:,2]), max(X[:,2]))

# --- Ecuation
x_eq, y_eq = np.meshgrid(x, y)
eq = w[0] + w[1] * x_eq + w[2] * y_eq

# --- 3d plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:,1],X[:,2],Y) # Dibujar puntos
ax.plot_surface(x_eq, y_eq, eq) # Dibujar plano
ax.set_xlabel("Normalized X1")
ax.set_ylabel("Normalized X2")
ax.set_zlabel("Price ($u$)")
plt.show()
#--------------------------------------------------------
```



[102]:

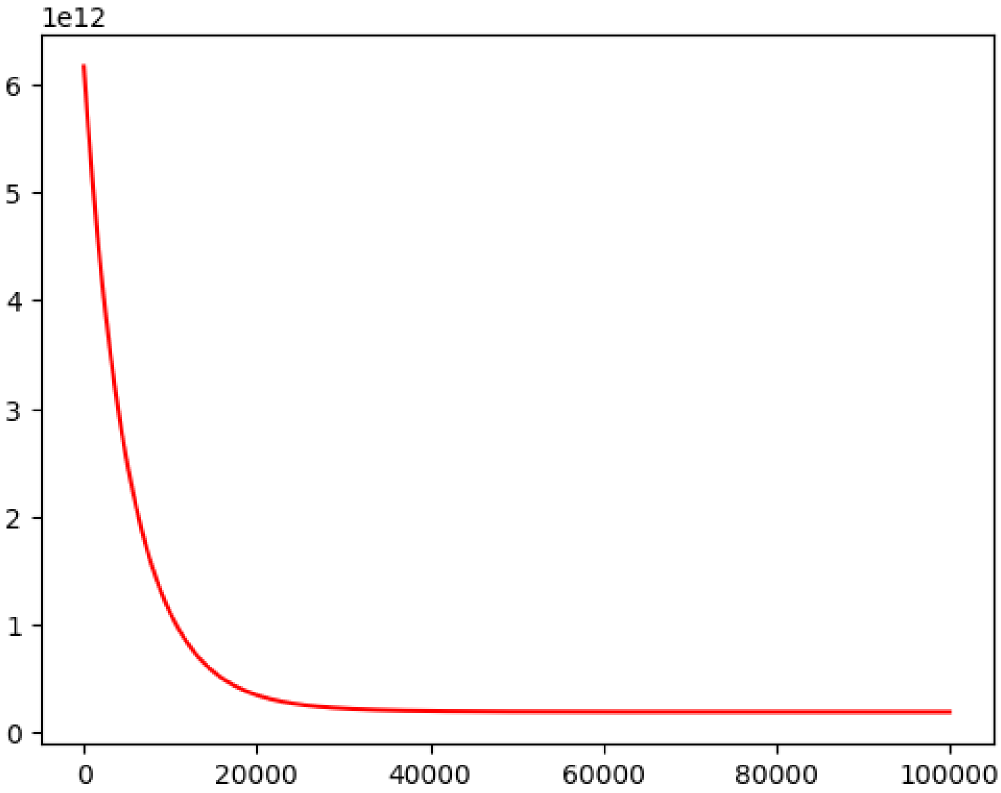
```
# --- Real Data


x_or = np.linspace(min(X_or[:,0]), max(X_or[:,0]))
y_or = np.linspace(min(X_or[:,1]), max(X_or[:,1]))


x_1_or, y_1_or = np.meshgrid(x_or, y_or)


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_or[:,0],X_or[:,1],Y) # Dibujar puntos
ax.plot_surface(x_1_or, y_1_or, eq) # Dibujar plano
ax.set_xlabel("Size (f2)")
ax.set_ylabel("# of rooms")
ax.set_zlabel("Price ($u$)")
plt.show()
#--------------------------------------------------------
```



0.1.5 Error behavior

[105]:

```
# --- error

plt.plot(T, Error, '-r') # Data plot
plt.show()
Min_E = min(Error)
print("Minimun error value = "+str(Min_E))
#-------------------------------------------------------
```



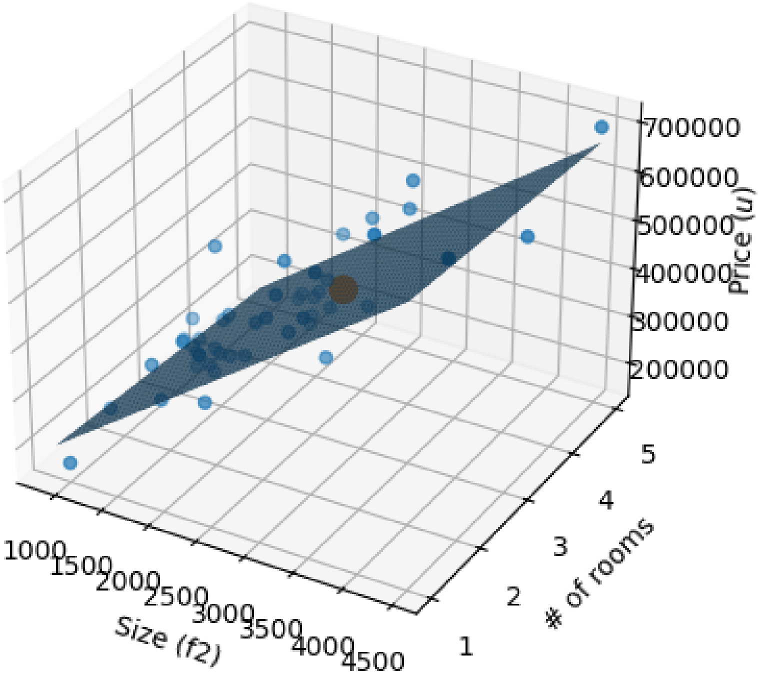
```
Minimun value of error = 192110757376.10178
```

0.1.6 Predicted Values

[106]:
```
# --- Predict values

X1 = 2800 #  h1(X)
X2 = 3 #  h2(X)

# --- Normalize
X1_norm = (X1 - mu[0])/sigma[0]
X2_norm = (X2 - mu[1])/sigma[1]

# --- Show values
Y_point = w[0] + w[1]*X1_norm + w[2]*X2_norm #
print ("Caracteristic h1(x) = "+str(X1)+
       ", Caracteristic h2(x) = "+str(X2)+
       ", Predicted Y = "+str(Y_point))

# --- Plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

```
ax.scatter(X_or[:,0],X_or[:,1],Y)
ax.plot_surface(x_1_or, y_1_or, eq)
ax.scatter(X1, X2, Y_point, s=100) #
ax.set_xlabel("Size (f2)")
ax.set_ylabel("# of rooms")
ax.set_zlabel("Price ($u$)")
plt.show()
#---------------------------------------------------------
```

```
Caracteristic h1(x) = 2800, Caracteristic h2(x) = 3, Predicted Y =
451887.4296697386
```


Price (u)
Size (f2)
# of rooms