Regresión con varias variables

Imports

In [49]:

```
import numpy as np
import matplotlib.pyplot as plt
from pandas.io.parsers import read_csv
import scipy as sp
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

In [50]:

```
def carga_csv(file_name):  #Metodo proporcionado por el guión
    valores = read_csv(file_name, header=None).to_numpy()
    # suponemos que siempre trabajaremos con float
    return valores.astype(float)
```

In [51]:

```
values = carga_csv('D:/GDV/5/AA/AprendizajeAutomatico-main/P1/ex1data2.csv')  #Carga del archivo


  ##   Generacion de matrices y variables ##
_X = values[:, :-1]
np.shape(_X)

_Y = values[:, -1]
np.shape(_Y)

m = np.shape(_X)[0]
n = np.shape(_X)[1]
alpha = 0.6
```

Normalización

In [52]:

```
def normalizacion(x):
    nCols = len(x[1])  #Conociendo el nº de columnas
    mediaX = np.zeros(nCols) # asignamos espacio para los arrays necesarios
    desvX = np.zeros(nCols)

    for i in range(nCols):
        x_nor = x[:,i] #Recorremos los campos disponibles extrayendo las metricas
        mediaX[i] = np.mean(x_nor)
        desvX[i] = np.std(x_nor)

    x_Normalized = (x - mediaX) / desvX # Asignamos a la matriz final los valores correspondientes
    return x_Normalized
```

Descenso de gradiente

In [53]:

```
def descenso_grad(x, Y, alpha):
    Thetas = np.zeros(x[1].size) #Creamos el array de thetas
    costes = []

    for k in range(3): #Iteramos sobre 3 valores de alpha para ver las diferencias
        alpha = (alpha/3)  # dividiendo su valor entre 3 en cada vuelta
        for i in range(1500):  # 1500 iteraciones por alpha
            Thetas = Thetas - alpha*(1/m)*(x.T.dot(np.dot(x,Thetas)-Y)) # Y calculamos el nuevo valor de las thetas para cada vuelta
            coste(x,Y,Thetas)
            costes = np.append(costes, coste(x, Y, Thetas))  #Apuntamos los costes en el array

    return Thetas
```

Función de coste

In [54]:

```
def coste(X,Y,thetas): #  ESTA FUNCION ES LLAMADA DENTRO DE DESCENSO DE GRADIENTE
    hip = np.dot(X, thetas) #                    Θ * X

    aux = (hip - Y)**2 #                        ((Θ * X ) - Y) * ((Θ * X ) - Y)
    return aux.sum()/(2*(len(X)))#           1/2m * ((Θ * X ) - Y) * ((Θ * X ) - Y)
```

In [55]:

```
x_norm = normalizacion(_X) # Normalizamos la X
x_norm = np.hstack([np.ones([m,1]), x_norm])  #Y añadimos la columna de 1's para el descenso de gradiente
thetaVanilla = descenso_grad(x_norm, _Y,alpha)
print(thetaVanilla)
```

```
[340412.65957447 109447.79646964  -6578.35485416]
```

Ecuación Normal

In [56]:

```
def ec_Normal(matX, matY):
    Thetas = np.zeros(matX[1].size)

    X_trans = np.transpose(matX)  # X transpuesta --> Xt
    aux = np.dot(X_trans, matX)   # Xt * X
    auxInv = np.linalg.pinv(aux)  # (Xt * X)^-1   --> H
    aux2 = np.dot(auxInv,X_trans) # H* Xt         --> H2
    Thetas = np.dot(aux2, matY)   # H2 * Y = Θ
    return Thetas
```

In [57]:

```
_X = np.hstack([np.ones([m,1]), _X])
thetaNormal = ec_Normal(_X,_Y)
print(thetaNormal)
```

```
[89597.90954361   139.21067402 -8738.01911255]
```