Práctica 1 (Una variable)

¿Cómo he planteado la práctica?

Como es lógico, primero necesitamos los datos, en este caso leemos el CSV 'ex1data1', y separamos los datos en X (primera columna) e Y (segunda columna).

Además, por comodidad, he añadido una columna de 1's al principio del eje X para que, al multiplicar las Thetas, se pueda hacer de forma vectorizada.

Posteriormente, he implementado una función que calcula la Regresión Lineal, en este caso con una variable. Sus parámetros de entrada son:

- X: Valores
- Y: Resultado
- a: La tasa de aprendizaje
- iteraciones: El número de vueltas que da el bucle en el aprendizaje

En el siguiente paso he comprobado si el entrenamiento ha sido correcto, en este caso el enunciado se comenta que, con 1.500 iteraciones y una tasa de aprendizaje del 0.01, para una población de 70.000 habitantes (X = 7), hay unos beneficios de aproximadamente $45.300.

Además, para confirmar los resultados, he dibujado la recta y los datos.

Finalmente he generado la gráfica de contorno y la gráfica 3D.

Código comentado y gráficas

1. Importar las librerías

```
import numpy as np
from pandas.io.parsers import read_csv

from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

2. Obtener los datos del CSV:

```
data = read_csv("ex1data1.csv", header=None).to_numpy()
X = np.array([data[:, 0]])
Y = np.array([data[:, 1]]).T
m = X.shape[1]

# Columna de 1s al principio
X = np.hstack([np.ones([m, 1]), X.T])

```

Como he comentado anteriormente, he añadido con hstack una columna de unos por comodidad.

3. Implementar la Regresión Lineal, en este caso he puesto como valores predeterminados $\alpha = 0.01$ e $iteraciones = 1500$ por el enunciado.

```
1  def reg_lin_una_var (X, Y, a = 0.01, iteraciones = 1500):
2
3      Theta = np.array([[0, 0]])
4      J_Theta = 0
5
6      # Hipotesis
7      for i in range(iteraciones):
8          h_theta = np.array([np.sum(X * Theta, axis = 1)])
9          difference = h_theta.T - Y
10
11         # Funcion de coste
12         J_theta = (1/(2 * m) * np.sum(np.square( difference )))
13
14         # Actualizamos Thetas
15         Theta = Theta - a *(1/m)* np.sum( difference * np.array([X]), axis=1 )
16
17     return (J_Theta, Theta)
```

Para implementar la regresión lineal, he creado un array de Thetas e inicializado el coste a 0.

En el bucle he guardado $h_\theta - y$, dado que este cálculo se repite en reiteradas ocasiones y al guardarlo ahorrado coste computacional, esto lo podemos observar en las líneas #12 y #15.

Posteriormente he calculado la función del coste y he actualizado las θs.

Al terminar las iteraciones, se devuelve el coste y las θs entrenadas.

4. Ahora comprobamos que el resultado con:
   - Iteraciones: 1.500
   - Tasa de aprendizaje: 0.01
   - Prueba de 70.000 habitantes (X = 7)

   Es de aproximadamente $45.300.

   Para ello, tenemos que calcular la θs y a partir de ahí, multiplicarlas por los datos de los que se desea conocer el resultado.

   Como la fórmula es:

$$\theta_0 + \theta_1 * x_1$$

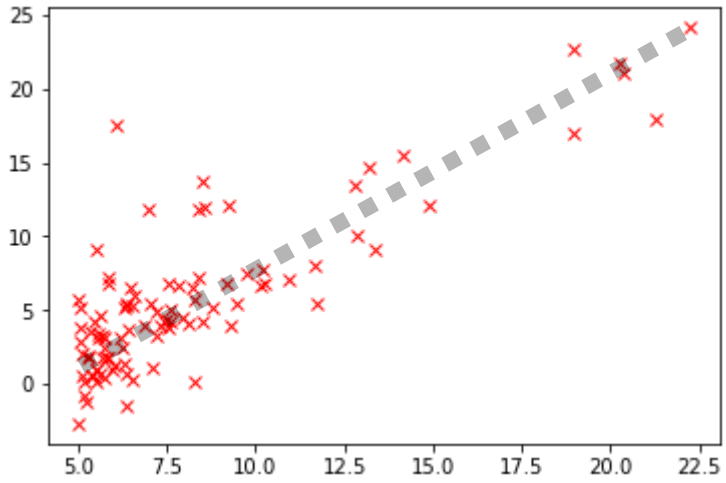
   es más sencillo si al primer elemento de X_prueba, le añadimos un '1', de forma que la fórmula haga la siguiente operación:

$$\theta_0 * 1 + \theta_1 * 7$$

```
1  J_Theta, Theta = reg_lin_una_var(X, Y, 0.01, 1500)
2
3  X_prueba = [1,   # Theta_0
4              7]   # + Theta_1 * X_1
5
6  print("$", np.sum(Theta * X_prueba) * 10000)
```

```
$ 45342.45012944714
```

5. Tras ello, vamos a ver si la recta de regresión se ha dibujado correctamente, esto quiere decir que se 'amolda' a los datos, por lo que hipotéticamente debería ser algo parecido a esto:

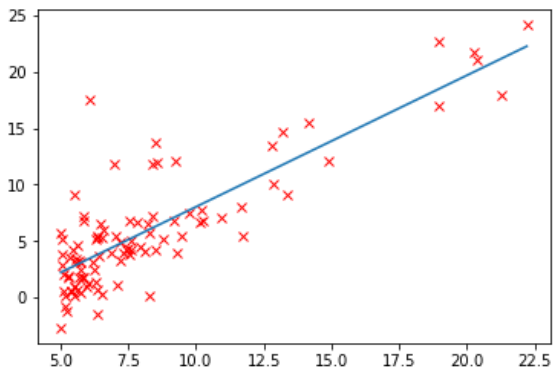


Para dibujarla, tenemos que buscar el mínimo y máximo valor de la X, y multiplicarlo por las Thetas para así, obtener la Y.

NOTA: Otra vez, por comodidad, añado un 1 como 'característica' para que, al hacer la multiplicación, no tener que hacer: $\theta_0 + \theta_1 * x$

```
# Visualizar datos
plt.plot([X[:, 1]], [Y[:, 0]], marker='x', c='red')

# Visualizar recta
X_min = np.min(X[:, 1])
X_max = np.max(X[:, 1])

X_min = [1, X_min]
X_max = [1, X_max]

Y_min = np.sum(X_min * Theta)
Y_max = np.sum(X_max * Theta)

print("El valor mínimo de X es:", X_min[1], "y su valor Y es de:", Y_min)
print("El valor máximo de X es:", X_max[1], "y su valor Y es de:", Y_max)

resultado_X = [X_min[1], X_max[1]]
resultado_Y = [Y_min, Y_max]

plt.plot(resultado_X, resultado_Y)
plt.show()
```

```
El valor mínimo de X es: 5.0269 y su valor Y es de: 2.2328954594975774
El valor máximo de X es: 22.203 y su valor Y es de: 22.266451825096567
```



6. Ahora implementamos la función de coste y además creamos los datos que les pasaremos a las gráficas

```
1  def coste(X, Y, Theta):
2      h_theta = np.array([np.sum(X * Theta, axis = 1)])
3      difference = h_theta.T - Y
4      return (1/(2 * len(X)) * np.sum(np.square( difference )))
```

```
1  def make_data(t0_range, t1_range, X, Y):
2      """  GENERA LAS MATRICES THETA0, THETA1, COSTE PARA GENERAR UN PLOT EN 3
3      DEL COSTE PARA VALORES DE THETA_0 EN EL INTERVALO T0_RANGE Y VALORES DE
4      step = 0.1
5      theta_0 = np.arange(t0_range[0], t0_range[1], step)
6      theta_1 = np.arange(t1_range[0], t1_range[1], step)
7
8      theta_0, theta_1 = np.meshgrid(theta_0, theta_1)
9      Coste = np.empty_like(theta_0)
10
11     for ix, iy in np.ndindex(theta_0.shape):
12         Coste[ix, iy] = coste(X, Y, [[theta_0[ix, iy], theta_1[ix, iy]]])
13
14     return [theta_0, theta_1, Coste]
```

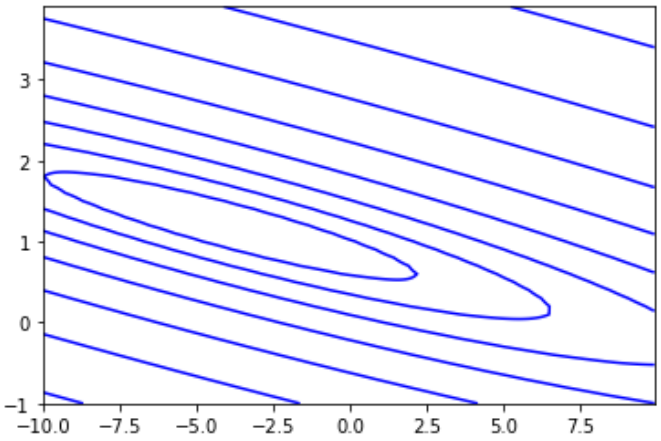
Básicamente lo que hacemos es una 'rejilla' de valores entre θ_0 y θ_1.

Para ello, generamos en cada θ un rango de valores (líneas #5 y #6) y posteriormente generamos una rejilla con la función np.meshgrid para cada θ.

Además, inicializamos una rejilla 'Coste' que estará vacía. A esta 'rejilla' le asignamos el valor del coste X e Y con sus correspondientes θs.

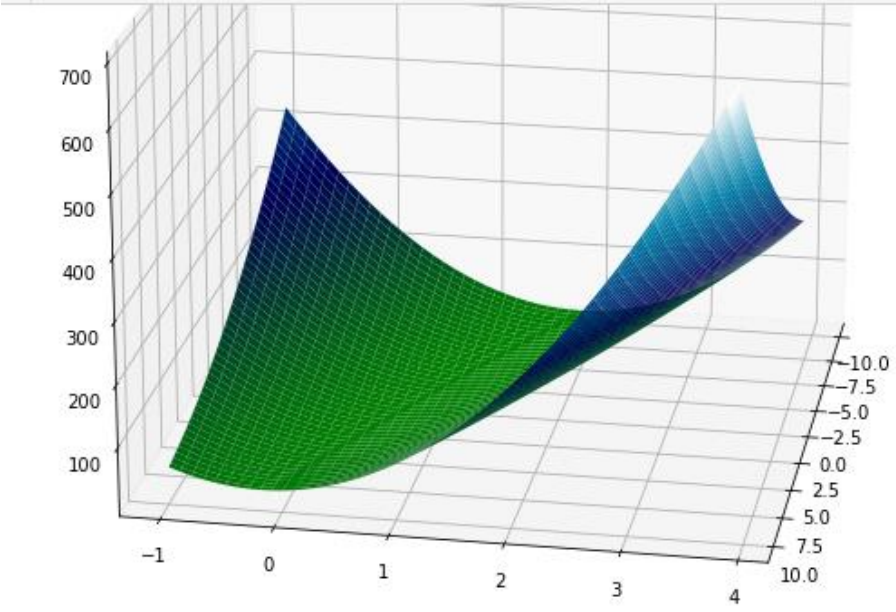
7. Ahora, llamamos a la función anterior y la dibujamos de distintas formas:
    a. Para la gráfica de contorno

```
Xx, Yy, Zz = make_data([-10, 10], [-1, 4], X, Y)

plt.contour(Xx, Yy, Zz, np.logspace(-2, 3, 20), colors='blue')
plt.show()
```



    b. Para la gráfica 3D

```
fig = plt.figure(figsize=(10,10))
ax = fig.gca(projection='3d')
ax.view_init(20, 10)
ax.plot_surface(Xx, Yy, Zz, cmap = cm.ocean)

plt.show()
```



Práctica 1 (Múltiples variables)

¿Cómo he planteado la práctica?

En este caso leemos el CSV 'ex1data2', y separamos los datos en X (primera columna) e Y (segunda columna).

En lugar de añadir la columna de 1's ahora, primero hay que normalizar la X, ya que los valores que tiene son muy distintos, y esto afectaría de forma negativa al aprendizaje, dado que tardaría más iteraciones en aprender.

Tras la normalización, por comodidad, he añadido una columna de 1's al principio del eje X para que, al multiplicar las Thetas, se pueda hacer de forma vectorizada.

Posteriormente, he implementado una función que calcula la Regresión Lineal, en este caso con múltiples variables. Sus parámetros de entrada son:

- X: Valores
- Y: Resultado
- a: La tasa de aprendizaje
- iteraciones: El número de vueltas que da el bucle en el aprendizaje

Tras ello, he mostrado las gráficas de múltiples valores para α, de forma que todos convergen a excepción de uno

Posteriormente he creado dos funciones más, una que nos permite calcular la ecuación normal, y otra que nos permite normalizar una 'fila' (conjunto de características), con los valores μ y σ del entrenamiento

Finalmente he llamado a las funciones de Regresión Lineal y Ecuación Normal, para comparar sus resultados.

1. Al igual que en la primera parte, he obtenido los datos (Sin añadir la columna de 1's). En este caso sabemos que la Y es la última columna y que solo tiene un valor, así que las X tienen que ser las anteriores columnas, por lo que usamos la notación ':-1', que devuelve todos los valores excepto el último, y este último se lo asignamos a la Y.

```
# Formatear datos
data = read_csv("ex1data2.csv", header=None).to_numpy().astype(float)
X = np.array(data[:, :-1])
Y = np.array(data[:, -1])
```

2. Ahora definimos la función 'normalizar'. Hay distintos tipos de normalización, como la min-max, pero en este caso he usado la normalización estándar, por lo que he buscado aplicar la fórmula:

$$\frac{X - \mu}{\sigma}$$

```
def normalizar(X):
    mu = np.mean(X, axis = 0)
    sigma = np.std(X, axis = 0)
    X_norm = np.divide(np.subtract(X, mu), sigma)
    return [X_norm, mu, sigma]
```

Además, como tendremos que normalizar valores posteriormente, debemos guardarnos μ y σ

3. Ahora llamamos la función y posteriormente, añadimos la columna de 1's.

```
m = X.shape[0]

X_norm, mu, sigma = normalizar(X)
#Y_norm, _, _ = normalizar(Y)
X = np.hstack([np.ones([m, 1]), X])
X_norm = np.hstack([np.ones([m, 1]), X_norm])

```

4. Ahora definimos la función que nos permitirá usar la Regresión Lineal con múltiples variables. Como observamos, creamos una θ por cada columna de X que tengamos, y las inicializamos a 0.
   Además, iré concatenando a las listas *J_theta(#6)* y *gradient (#7)*.

   Al igual que en la primera parte, dentro de un bucle con las iteraciones del parámetro, entrenamos la Regresión Lineal, es decir, calculamos h_θ e $h_\theta - Y$ y aplicamos las fórmulas pertinentes.

   Es interesante guardar $h_\theta - Y$ dado que lo usamos múltiples veces y nos ahorramos recalcularlo todo el rato.

   Ahora concatenamos los valores de las fórmulas en *J_theta* y *gradient*, y finalmente, actualizamos las θ.

```
def reg_lin_mul_var (X, Y, a, iteraciones = 1500):

    m = X.shape[0]
    n = X.shape[1]
    Theta = np.zeros(n)
    J_theta = []
    gradient = []


    for iter in range(iteraciones):
        H = np.dot(X, Theta)
        diff = (H - Y)
        J_theta.append((1 / (2 * m)) * np.sum(np.square(diff)))
        gradient.append((1 / m) * np.sum(diff * X.T))

        for i in range(n):
            aux = diff * X[:, i]
            Theta[i] -= (a / m) * np.sum(aux)

    return (Theta, gradient, J_theta)
```

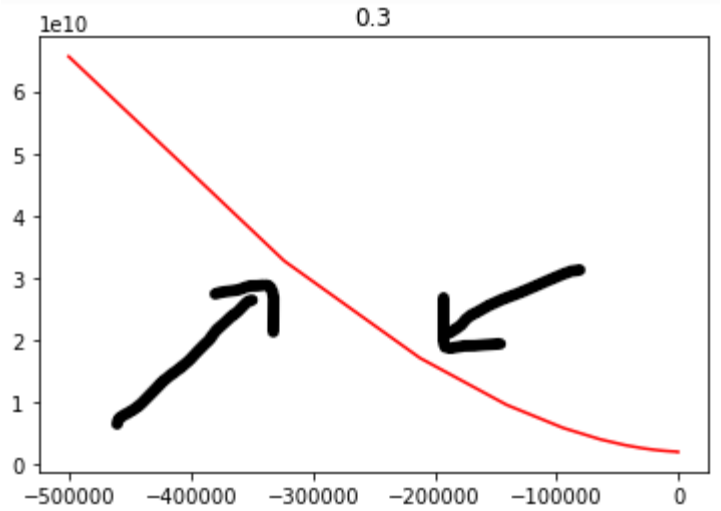
Como resultado devolvemos las θs y las listas de *J_theta* y *gradient*.

5. Ahora entrenamos la Regresión Lineal con distintos parámetros, por lo que he insertado en una lista dichos α y he ido iterando sobre ellos. Además, al llamar a la función de Regresión Lineal, he hecho solo 100 iteraciones para que las gráficas se vean correctamente.
   Finalmente he dibujado las gráficas con las listas *gradient y J_theta*.

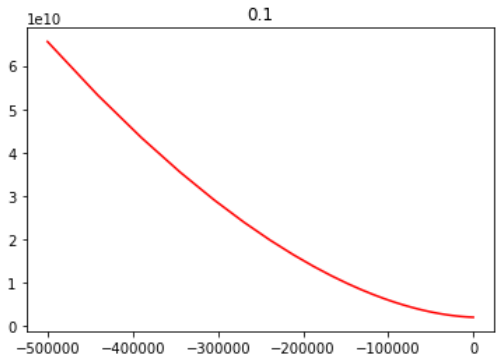
```
alpha = np.array([.3, .1, .03, .01, 3])

for a in alpha:
    plt.figure()
    plt.title(a)
    Theta, gradient, J_theta = reg_lin_mul_var(X_norm, Y, a, 100)
    plt.plot(gradient, J_theta, c='red')


    plt.show()
```

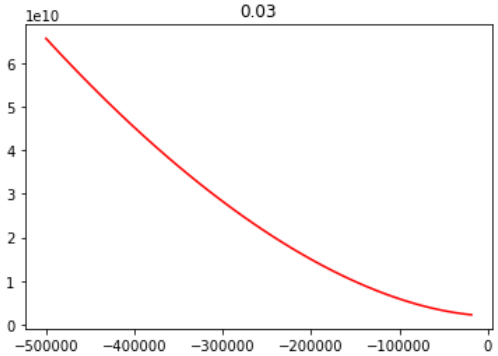
A excepción del último, todas las tasas de aprendizaje son convergentes, y esto lo deducimos por las gráficas.

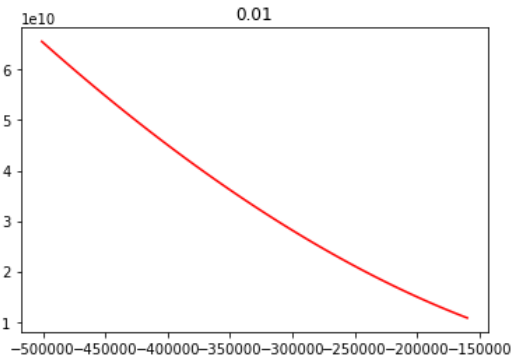
Para $\alpha = 0.3$ lo único destacable es que el aprendizaje es algo 'errático', esto es normal dado que aprende muy rápido en los primeros pasos.



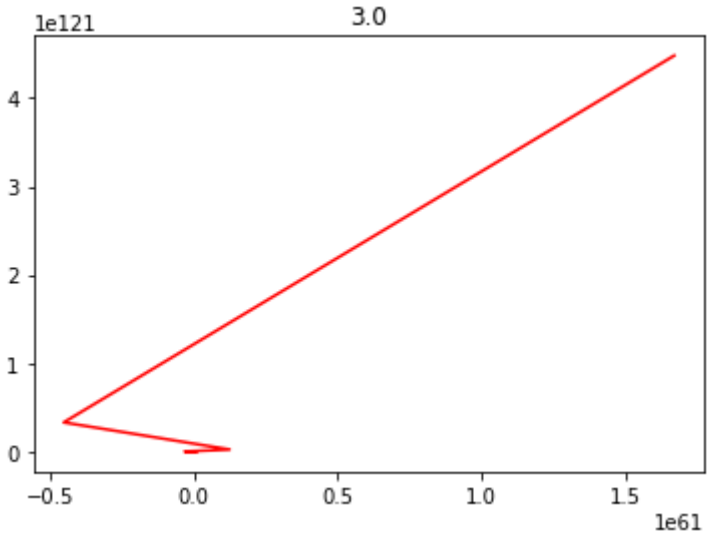
En el resto de gráficas podemos observar que el aprendizaje tiende a ser más lineal y lento.







Finalmente, para $\alpha = 3$ diverge.



6. Creamos la función de la ecuación normal.

$$\theta = (X^T * X)^{-1} * X^T * Y$$

```
def ecuacion_normal(X, Y):
    inversa = np.linalg.pinv( np.dot(X.T, X) )
    multiplicacion = np.dot(X.T, Y)
    Theta = np.dot(inversa, multiplicacion)

    return Theta
```

7. Creamos una función que nos permita normalizar valores con los μ y σ anteriores.

```
def normalizar(X, mu, sigma):
    X_norm = np.divide(np.subtract(X, mu), sigma)
    return X_norm
```

8. Y finalmente comprobamos los resultados.

   Primero creamos los resultados de la gradiente, para ello llamamos a la gradiente con la X normalizada (#2).
   Tras ello, creamos los valores que deseemos testear, en este caso el enunciado pide 1.650 pies y 3 habitaciones (#5)
   Normalizamos los datos (#8) y añadimos un 1 (#9) para evitar multiplicar valor por valor.
   Finalmente, multiplicamos las θs y X_prueba y los sumamos.
   Conceptualmente es:

$$\theta_0 * 1 + \theta_1 * 1650 + \theta_2 * 3$$

```
1  # Predicciones / entrenamientos
2  prediccion_gradiente, _, _ = reg_lin_mul_var (X_norm, Y, .1, 10000)
3
4  # Valor prueba
5  X_prueba = [1650, 3]
6
7  # Normalizar valores
8  X_prueba = normalizar(X_prueba, mu, sigma)
9  X_prueba = np.append(1, X_prueba)
10
11 print("Theta. Gradiente:", prediccion_gradiente)
12 print("Pred. Gradiente:", np.dot(prediccion_gradiente, X_prueba))
```

```
Theta. Gradiente: [340412.65957447 109447.79646964  -6578.35485416]
Pred. Gradiente: 293081.46433489595
```

Posteriormente hacemos lo mismo, pero con la ecuación normal (Aquí no hace falta normalizar), y al igual que antes, multiplicamos y sumamos.

```
1  # Predicciones / entrenamientos
2  prediccion_normalizada = ecuacion_normal(X, Y)
3
4  # Valor prueba
5  X_prueba = [1, 1650, 3]
6
7  print("Theta. Normalizada:", prediccion_normalizada)
8  print("Pred. Normalizada:", np.dot(prediccion_normalizada, X_prueba))
```

```
Theta. Normalizada: [89597.90954355   139.21067402 -8738.01911255]
Pred. Normalizada: 293081.4643349727
```