RPYAA Ejercicio 2

Ejercicio 1

Descargar el archivo ‘movie_metadata.csv’

Ejercicio 2

Se usó la biblioteca csv para manejar el archivo. De esta manera en cada línea del archivo con DictReader podemos obtener el atributo con el nombre de la columna, y así obtener solo las columnas que necesitamos.

Código:

```
# Abrir archivo
csvfile = open('movie_metadata.csv')
# Crear nuevo archivo
newfile = open('new_file.csv', 'w')
# Nombre de los campos
fieldnames = ['actor_1_facebook_likes', 'actor_2_facebook_likes',
'actor_3_facebook_likes', 'director_facebook_likes', 'budget',
'movie_facebook_likes']
# Crear un reader para archivos csv
reader = csv.DictReader(csvfile)
# Crea un writer para archivos csv
writer = csv.DictWriter(newfile, fieldnames=fieldnames)
writer.writeheader()
# Valores máximos de cada columna
max_vals = dict.fromkeys(['a1', 'a2', 'a3', 'df', 'b', 'mf'], 0)
# Iterar en el lector para obtener los valores máximos
for row in reader:

    a1, a2, a3 = row[fieldnames[0]], row[fieldnames[1]], row[fieldnames[2]]
    df, b, mf = row[fieldnames[3]], row[fieldnames[4]], row[fieldnames[5]]

    if a1.strip() != '':
        max_vals['a1'] = max(int(a1), max_vals['a1'])

    if a2.strip() != '':
        max_vals['a2'] = max(int(a2), max_vals['a2'])

    if a3.strip() != '':
        max_vals['a3'] = max(int(a3), max_vals['a3'])

    if df.strip() != '':
        max_vals['df'] = max(int(df), max_vals['df'])

    if b.strip() != '':
        max_vals['b'] = max(int(b), max_vals['b'])

    if mf.strip() != '':
        max_vals['mf'] = max(int(mf), max_vals['mf'])

# Regresar al inicio del archivo
csvfile.seek(0)
reader = csv.DictReader(csvfile)
```

Ejercicio 3

El código de arriba, además de leer también fue para obtener el máximo de cada columna, de esta manera podemos escribir en otro archivo los datos normalizados con el siguiente código:

```
# Escribir en un nuevo archivo los valores normalizados
for row in reader:

    a1, a2, a3 = row[fieldnames[0]], row[fieldnames[1]], row[fieldnames[2]]
    df, b, mf = row[fieldnames[3]], row[fieldnames[4]], row[fieldnames[5]]

    if a1.strip() == '' or a2.strip() == '' or a3.strip() == ''
    or df.strip() == '' or b.strip() == '' or mf.strip() == '':
        continue

    a1 = round(float(a1) / max_vals['a1'], 10)
    a2 = round(float(a2) / max_vals['a2'], 10)
    a3 = round(float(a3) / max_vals['a3'], 10)
    df = round(float(df) / max_vals['df'], 10)
    b = round(float(b) / max_vals['b'], 10)
    mf = round(float(mf) / max_vals['mf'], 10)

    writer.writerow({fieldnames[0]: a1, fieldnames[1]: a2, fieldnames[2]:a3,
    fieldnames[3]: df, fieldnames[4]: b, fieldnames[5]: mf})

# Regresar al inicio del archivo
newfile.seek(0)
newfile = open('new_file.csv')
reader = csv.DictReader(newfile)
```

Además, creamos una lista por cada columna del nuevo archivo:

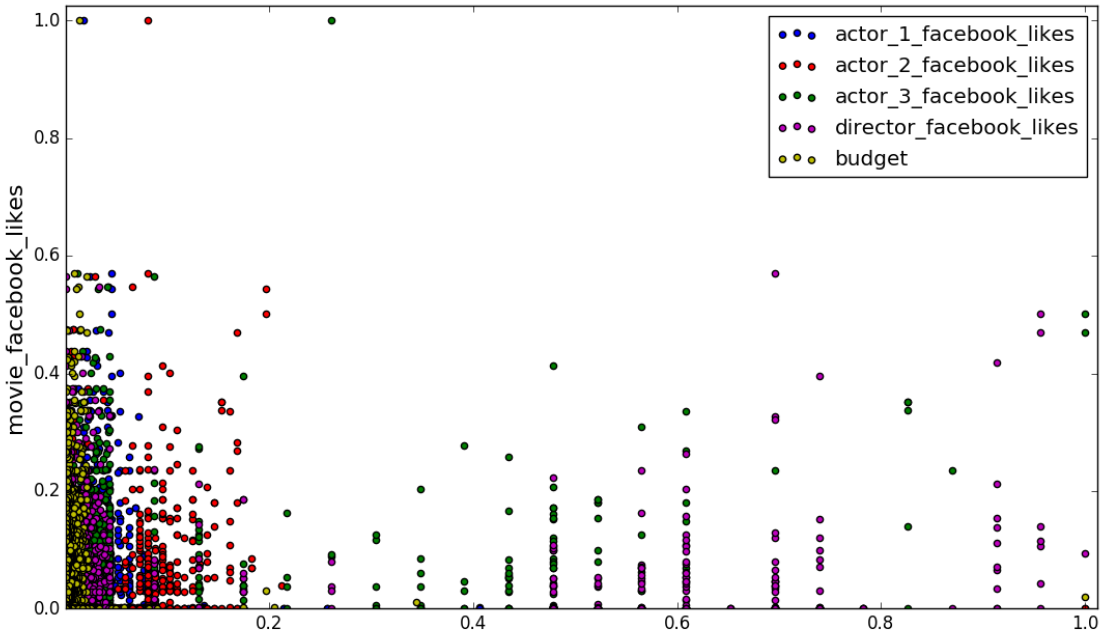
```
l1, l2, l3, l4, l5, l6 = ([] for i in range(6))
# Crear listas por cada columna
for row in reader:

    a1, a2, a3 = row[fieldnames[0]], row[fieldnames[1]], row[fieldnames[2]]
    df, b, mf = row[fieldnames[3]], row[fieldnames[4]], row[fieldnames[5]]

    l1.append(float(a1))
    l2.append(float(a2))
    l3.append(float(a3))
    l4.append(float(df))
    l5.append(float(b))
    l6.append(float(mf))
```

Y así poder generar la gráfica:

```
# Crear la gráfica con todos sus atributos
fig = plt.figure()
ax1 = fig.add_subplot(111)
plt.scatter(l1, l6, s=20, c='b', label=fieldnames[0])
plt.scatter(l2, l6, s=20, c='r', label=fieldnames[1])
plt.scatter(l3, l6, s=20, c='g', label=fieldnames[2])
plt.scatter(l4, l6, s=20, c='m', label=fieldnames[3])
plt.scatter(l5, l6, s=20, c='y', label=fieldnames[4])
plt.xlim(0, 1.1)
plt.ylim(0, 1.1)
plt.ylabel(fieldnames[5], fontsize=16)
plt.legend(loc='upper right');
plt.show()
```



Ejercicio 4

Para hacer el descenso por el gradiente es necesario ajustar los valores para la función, es decir, convertir las listas en una matriz y asignar valores a los parámetros.

```
X = np.column_stack(([1]*len(l1), l1, l2, l3, l4, l5))
y = np.array(l6).transpose().reshape(len(l6), 1)

iterations = 100
alpha = 0.05
theta = np.matrix([[0]]*6)
t = gradientdescent(X, y, theta, alpha, iterations)
```

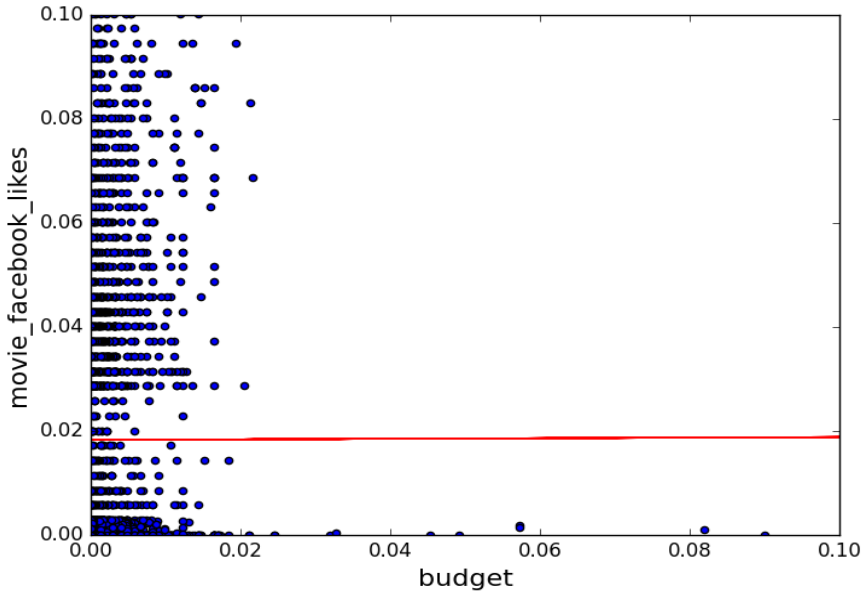
Para así realizar el descenso:

```
def gradientdescent(X, y, theta, alpha, num_iterations):

    m = len(y)
    for i in range(num_iterations):
        hypoth = np.dot(X, theta)
        loss = hypoth - y
        gradient = np.dot(np.transpose(X), loss) / m
        theta = theta - (alpha * gradient)
        cost = np.sum(np.power(loss, 2)) / (2 * m)
        print('Iteracion', i + 1, '| Costo:', cost)

    return theta
```

Graficamos ahora la evolución de los pesos, obteniendo los pesos t con el gradient descent en 2000 iteraciones usando una tasa de aprendizaje de 0.05.



Modificamos el código para que actualice los pesos cada 100 iteraciones:

```
def gradientdescent(X, y, theta, alpha, num_iterations):

    m = len(y)
    c = 0;
    for i in range(num_iterations):
        # El espacio de hipótesis H = OX
        hypoth = np.dot(X, theta)
        loss = hypoth - y
        # Calcular el gradiente con (X^T)(H - y)
        gradient = np.dot(np.transpose(X), loss) / m
        # Ajustar los pesos usando la tasa de aprendizaj
        if c % 100 == 0:
            theta = theta - (alpha * gradient)
        cost = np.sum(np.power(loss, 2)) / (2 * m)
        print('Iteracion', i + 1, '| Costo:', cost)
        c = c+1

    return theta
```

Y graficamos el nuevo modelo: