Importemos nuestro dataset

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import cm
import pandas as pd
# Importemos nuestro google drive
from google.colab import drive

drive.mount("/content/gdrive")
!pwd # Print working directory
```

```
    Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", forc
    /content/gdrive/My Drive/IA avanzada para la ciencia de datos/Machine learning
```

```
# Buscamos el el path donde esta nuestro csv
%cd "/content/gdrive/MyDrive/IA avanzada para la ciencia de datos/Machine learning"
!ls # List files located in defined folder
```

```
    /content/gdrive/MyDrive/IA avanzada para la ciencia de datos/Machine learning
     Act.1_loading_dataset.ipynb           Iris.csv
     Act.2_hypothesis_function.ipynb      'log_reg_multiclase_alumnos (1).ipynb'
    'Act.3_linear_reg_gd1_alumno .ipynb'  'perceptron_and_or_xnor-1 (2).ipynb'
     Act.4_log_reg_gd_V06_alumno-1.ipynb   Presentaciones
     Act.5_over_under_fitting.ipynb        score_updated.csv
    'Canada income per capita.csv'
```

Analisis del dataset

Para esta actividad vamos a ver un el PIB pero capita en Canada del año 1970 al 2020.

```
columnas=["Year","Income"]
df = pd.read_csv('Canada income per capita.csv',names=columnas)
df.head()
```

	Year	Income
0	1970	3429.756623
1	1971	3774.182973
2	1972	4278.766586
3	1973	4861.052386
4	1974	5705.506544

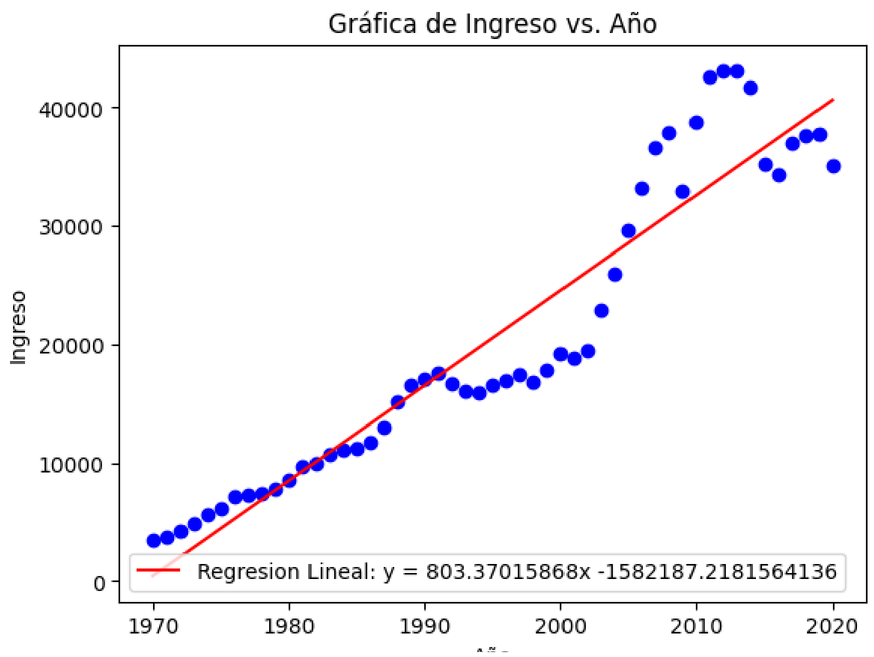
```
#Vemos mas informacion de los datos
df.info()
```

```
    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 51 entries, 0 to 50
    Data columns (total 2 columns):
     #   Column  Non-Null Count  Dtype  
    ---  ------  --------------  -----  
     0   Year    51 non-null     int64  
     1   Income  51 non-null     float64
    dtypes: float64(1), int64(1)
    memory usage: 944.0 bytes
```

No hay valores faltantes, tenemos 51 filas, 2 columnas. Es un dataframe muy simple pero debido a esto es facil de analizar

Grafica de los datos

```
# Define X(features) & y(labels) from the dataset
df_X = df[["Year"]]
df_y = df[["Income"]]

plt.scatter(df_X, df_y, marker='o', c='b')
plt.xlabel("Año")
plt.ylabel("Ingreso")
plt.title("Gráfica de Ingreso vs. Año")


##La linea recta la hice despues de ver los valores de la regresion lineal


# Supongamos que tienes los valores de la pendiente (m) y la ordenada al origen (b)
m =803.37015868
b = -1582187.2181564136

# Calcula los valores de Y para la línea recta
x_values = np.array([min(df_X.values), max(df_X.values)])  # Tomamos los valores mínimos y máximos de X
y_values = m * x_values + b  # Ecuación de la línea recta

# Dibuja la línea recta
plt.plot(x_values, y_values, c='r', label=f'Regresion Lineal: y = {m}x {b}')
plt.legend()  #a

plt.show()
```



```
x_list = np.array(range(1970,2022)) # Set x range
print(x_list)
y_list = (x_list * w) + b # Set function for the model based on w & b
print(y_list)
```

```
[1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983
 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997
 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011
 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021]
[-574408.47800195 -573896.91518969 -573385.35237743 -572873.78956517
 -572362.22675291 -571850.66394065 -571339.10112839 -570827.53831613
 -570315.97550387 -569804.4126916  -569292.84987934 -568781.28706708
 -568269.72425482 -567758.16144256 -567246.5986303  -566735.03581804
 -566223.47300578 -565711.91019352 -565200.34738125 -564688.78456899
 -564177.22175673 -563665.65894447 -563154.09613221 -562642.53331995
 -562130.97050769 -561619.40769543 -561107.84488317 -560596.2820709
 -560084.71925864 -559573.15644638 -559061.59363412 -558550.03082186
 -558038.4680096  -557526.90519734 -557015.34238508 -556503.77957281
 -555992.21676055 -555480.65394829 -554969.09113603 -554457.52832377
 -553945.96551151 -553434.40269925 -552922.83988699 -552411.27707473
 -551899.71426246 -551388.1514502  -550876.58863794 -550365.02582568
 -549853.46301342 -549341.90020116 -548830.3373889  -548318.77457664]
```

Creamos nuestro modelo de regresion lineal

```
def update_w_and_b(X, y, w, b, alpha):
  '''Update parameters w and b during 1 epoch'''
  dl_dw = 0.0
  dl_db = 0.0
  N = len(X)
  for i in range(N):
    dl_dw += -2*X[i]*(y[i] - (w*X[i] + b))
    dl_db += -2*(y[i] - (w*X[i] + b))
  # update w and b
  w = w - (1/float(N))*dl_dw*alpha
  b = b - (1/float(N))*dl_db*alpha
  return w, b

def train(X, y, w, b, alpha, epochs):
  '''Loops over multiple epochs and prints progress'''
  print('Training progress:')
  for e in range(epochs):
    w, b = update_w_and_b(X, y, w, b, alpha)
  # log the progress
    if e % 400 == 0:
      avg_loss_ = avg_loss(X, y, w, b)
      # print("epoch: {} | loss: {}".format(e, avg_loss_))
      print("Epoch {} | Loss: {} | w:{}, b:{}".format(e, avg_loss_, np.round(w, 4), np.round(b, 4)))
  return w, b

def train_and_plot(X, y, w, b, alpha, epochs):
  '''Loops over multiple epochs and plot graphs showing progress'''
  for e in range(epochs):
    w, b = update_w_and_b(X, y, w, b, alpha)
  # plot visuals for last epoch
    if e == epochs-1:
      avg_loss_ = avg_loss(X, y, w, b)
      x_list = np.array(range(1970,2023)) # Set x range
      y_list = (x_list * w) + b # Set function for the model based on w & b
      plt.scatter(x=X, y=y)
      plt.plot(x_list,y_list, c='r')
      plt.title("Epoch {} | Loss: {} | w:{}, b:{}".format(e, np.round(avg_loss_,2), np.round(w, 4), np.round(b, 4)))
      plt.show()
  return w, b
```

```
def avg_loss(X, y, w, b):
  '''Calculates the MSE'''
  N = len(X)
  total_error = 0.0
  for i in range(N):
    total_error += (y[i] - (w*X[i] + b))**2
  return total_error / float(N)

def predict(x, w, b):
  return w*x + b
```

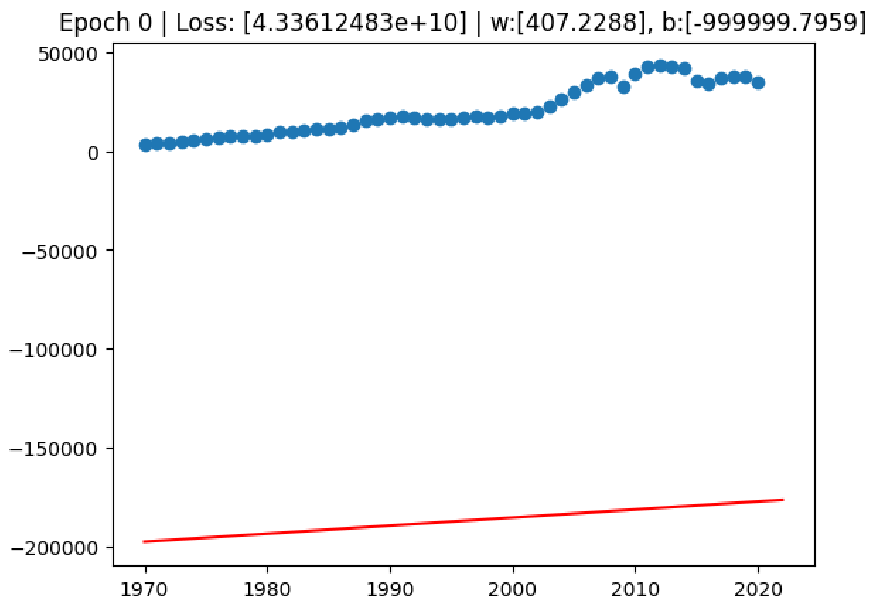
Entrenamos el modelo

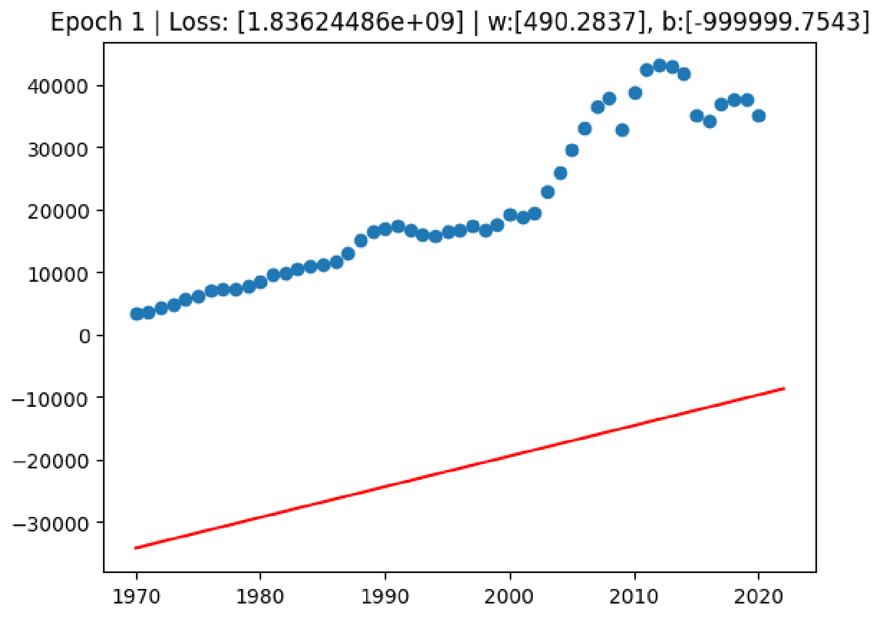
```
# Definimos el peso 0 y el bias 0, con un tamano de paso alpha de 0.0001, y 20000 epochs

w, b = train(X=df_X.values, y=df_y.values, w=0, b=-1000000, alpha=0.0000001, epochs=20000)
```

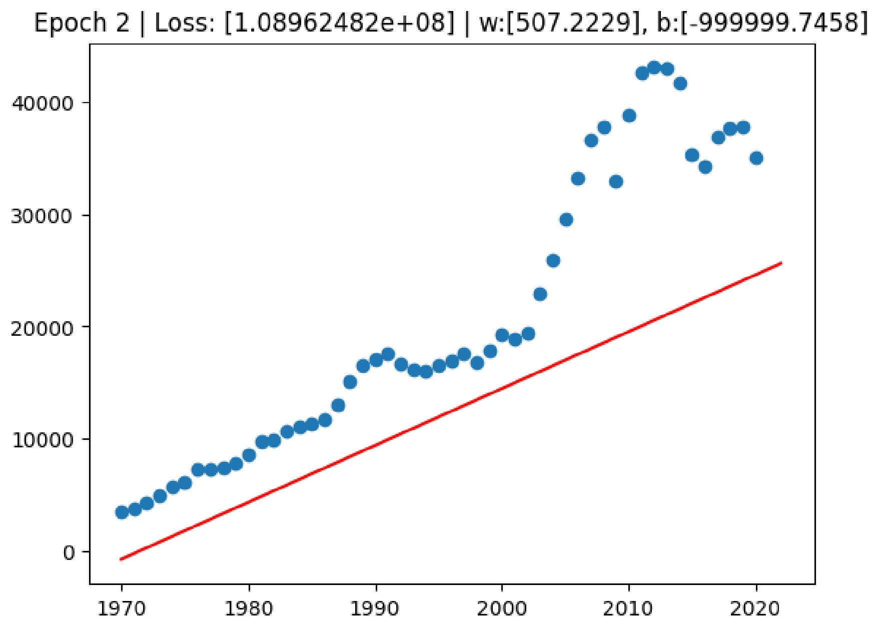
```
Training progress:
Epoch 0 | Loss: [4.33612483e+10] | w:[407.2288], b:[-999999.7959]
Epoch 400 | Loss: [33995766.45232853] | w:[511.5627], b:[-999999.7461]
Epoch 800 | Loss: [33995766.29163009] | w:[511.5628], b:[-999999.7487]
Epoch 1200 | Loss: [33995766.13093166] | w:[511.5628], b:[-999999.7512]
Epoch 1600 | Loss: [33995765.97023325] | w:[511.5628], b:[-999999.7537]
Epoch 2000 | Loss: [33995765.80953478] | w:[511.5628], b:[-999999.7563]
Epoch 2400 | Loss: [33995765.6488364] | w:[511.5628], b:[-999999.7588]
Epoch 2800 | Loss: [33995765.48813792] | w:[511.5628], b:[-999999.7613]
Epoch 3200 | Loss: [33995765.32743952] | w:[511.5628], b:[-999999.7639]
Epoch 3600 | Loss: [33995765.16674119] | w:[511.5628], b:[-999999.7664]
Epoch 4000 | Loss: [33995765.00604275] | w:[511.5628], b:[-999999.769]
Epoch 4400 | Loss: [33995764.84534426] | w:[511.5628], b:[-999999.7715]
Epoch 4800 | Loss: [33995764.68464586] | w:[511.5628], b:[-999999.774]
Epoch 5200 | Loss: [33995764.52394743] | w:[511.5628], b:[-999999.7766]
Epoch 5600 | Loss: [33995764.36324898] | w:[511.5628], b:[-999999.7791]
Epoch 6000 | Loss: [33995764.20255053] | w:[511.5628], b:[-999999.7816]
Epoch 6400 | Loss: [33995764.04185208] | w:[511.5628], b:[-999999.7842]
Epoch 6800 | Loss: [33995763.88115379] | w:[511.5628], b:[-999999.7867]
Epoch 7200 | Loss: [33995763.72045533] | w:[511.5628], b:[-999999.7892]
Epoch 7600 | Loss: [33995763.5597572] | w:[511.5628], b:[-999999.7918]
Epoch 8000 | Loss: [33995763.39905855] | w:[511.5628], b:[-999999.7943]
Epoch 8400 | Loss: [33995763.23836007] | w:[511.5628], b:[-999999.7968]
Epoch 8800 | Loss: [33995763.0776617] | w:[511.5628], b:[-999999.7994]
Epoch 9200 | Loss: [33995762.91696328] | w:[511.5628], b:[-999999.8019]
Epoch 9600 | Loss: [33995762.75626491] | w:[511.5628], b:[-999999.8044]
Epoch 10000 | Loss: [33995762.59556647] | w:[511.5628], b:[-999999.807]
Epoch 10400 | Loss: [33995762.43486805] | w:[511.5628], b:[-999999.8095]
Epoch 10800 | Loss: [33995762.27416968] | w:[511.5628], b:[-999999.8121]
Epoch 11200 | Loss: [33995762.11347122] | w:[511.5628], b:[-999999.8146]
Epoch 11600 | Loss: [33995761.95277313] | w:[511.5628], b:[-999999.8171]
Epoch 12000 | Loss: [33995761.79207443] | w:[511.5628], b:[-999999.8197]
Epoch 12400 | Loss: [33995761.63137603] | w:[511.5628], b:[-999999.8222]
Epoch 12800 | Loss: [33995761.47067757] | w:[511.5628], b:[-999999.8247]
Epoch 13200 | Loss: [33995761.30997921] | w:[511.5628], b:[-999999.8273]
Epoch 13600 | Loss: [33995761.14928083] | w:[511.5628], b:[-999999.8298]
Epoch 14000 | Loss: [33995760.98858242] | w:[511.5628], b:[-999999.8323]
Epoch 14400 | Loss: [33995760.827884] | w:[511.5628], b:[-999999.8349]
Epoch 14800 | Loss: [33995760.66718566] | w:[511.5628], b:[-999999.8374]
Epoch 15200 | Loss: [33995760.50648716] | w:[511.5628], b:[-999999.8399]
Epoch 15600 | Loss: [33995760.34578876] | w:[511.5628], b:[-999999.8425]
Epoch 16000 | Loss: [33995760.18509042] | w:[511.5628], b:[-999999.845]
Epoch 16400 | Loss: [33995760.02439203] | w:[511.5628], b:[-999999.8475]
Epoch 16800 | Loss: [33995759.86369362] | w:[511.5628], b:[-999999.8501]
Epoch 17200 | Loss: [33995759.70299522] | w:[511.5628], b:[-999999.8526]
Epoch 17600 | Loss: [33995759.54229684] | w:[511.5628], b:[-999999.8552]
Epoch 18000 | Loss: [33995759.3815985] | w:[511.5628], b:[-999999.8577]
Epoch 18400 | Loss: [33995759.22090002] | w:[511.5628], b:[-999999.8602]
Epoch 18800 | Loss: [33995759.06020167] | w:[511.5628], b:[-999999.8628]
```

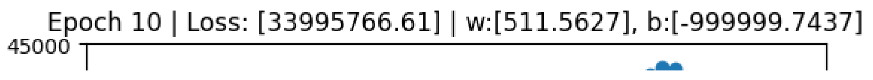
```
    Epoch 19200 | Loss: [33995758.89950318] | w:[511.5628], b:[-999999.8653]
    Epoch 19600 | Loss: [33995758.73880479] | w:[511.5628], b:[-999999.8678]
```

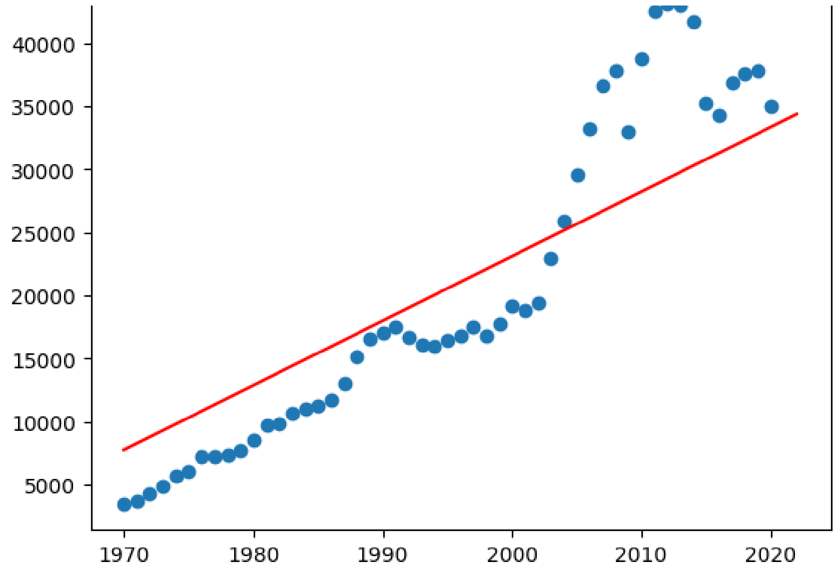
```
epoch_plots = [1, 2, 3, 11, 51, 101, epochs+1]
for epoch_plt in epoch_plots:
  w, b = train_and_plot(X=df_X.values, y=df_y.values, w=0, b=-1000000, alpha=0.0000001, epochs=epoch_plt)
```


Epoch 0 | Loss: [4.33612483e+10] | w:[407.2288], b:[-999999.7959]
50000
0
−50000
−100000
−150000
−200000
1970
1980
1990
2000
2010
2020

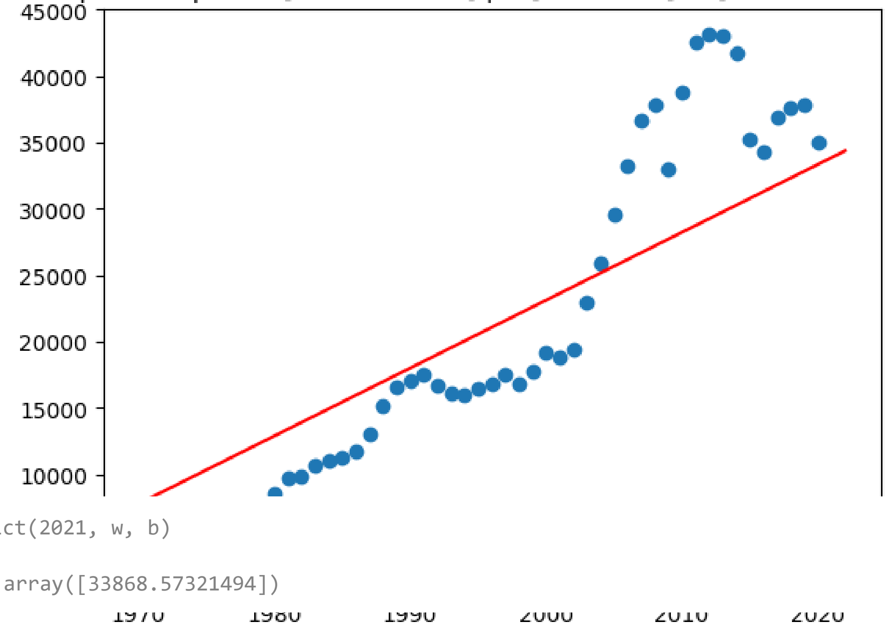

Epoch 1 | Loss: [1.83624486e+09] | w:[490.2837], b:[-999999.7543]
40000
30000
20000
10000
0
−10000
−20000
−30000
1970
1980
1990
2000
2010
2020


Epoch 2 | Loss: [1.08962482e+08] | w:[507.2229], b:[-999999.7458]
40000
30000
20000
10000
0
1970
1980
1990
2000
2010
2020


Epoch 10 | Loss: [33995766.61] | w:[511.5627], b:[-999999.7437]
45000



Epoch 50 | Loss: [33995766.59] | w:[511.5627], b:[-999999.743



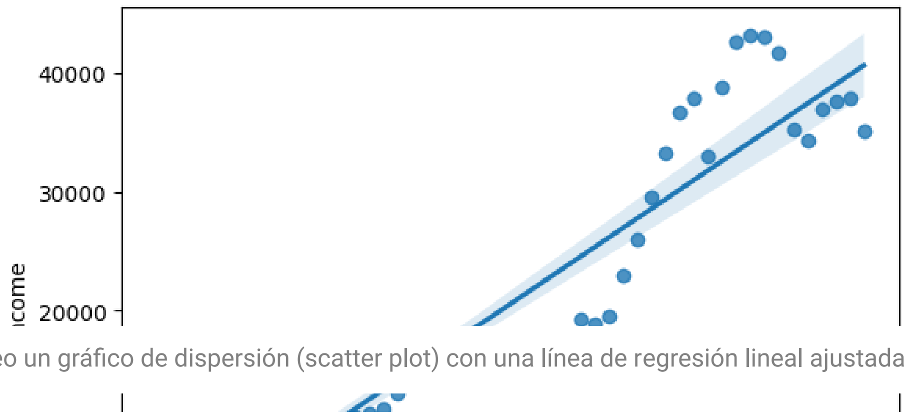
```
predict(2021, w, b)
```

```
array([33868.57321494])
```

Probemos con una libreria

```
import seaborn as sns

from sklearn.linear_model import LinearRegression

sns.regplot(x='Year',y='Income',data=df)
```

```
<Axes: xlabel='Year', ylabel='Income'>
```



Se creo un gráfico de dispersión (scatter plot) con una línea de regresión lineal ajustada a los datos.

```
lr=LinearRegression()
##Primero entrenemos al modelo
lr.fit(df[['Year']].values,df.Income)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Obtiene los coeficientes (pesos)
coeficientes = lr.coef_

# Obtiene el término de intercepción (bias)
intercepcion = lr.intercept_

print("Coeficientes:", coeficientes)
print("Intercepción:", intercepcion)
```

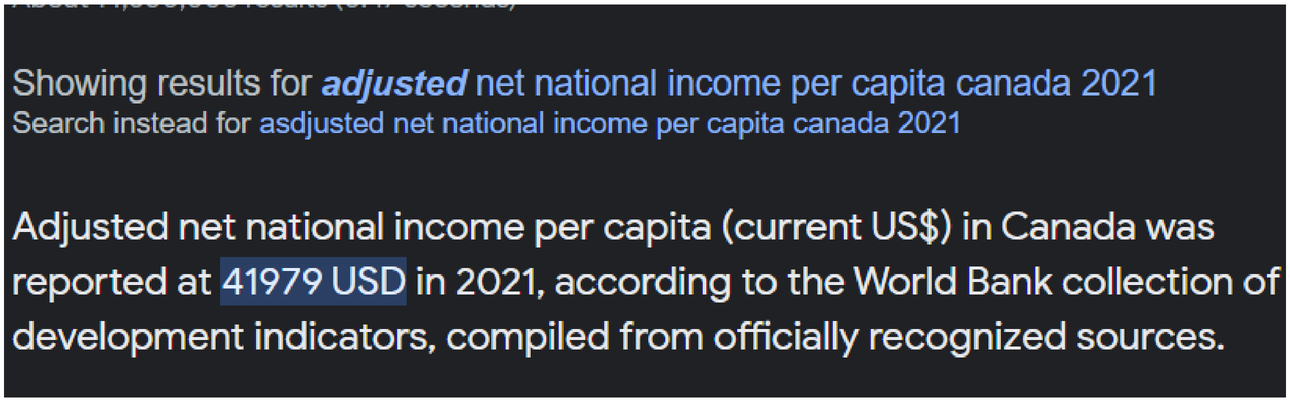
```
Coeficientes: [803.37015868]
Intercepción: -1582187.2181564136
```

```
from sklearn.metrics import mean_squared_error

# Predice los valores utilizando el modelo entrenado
y_pred = lr.predict(df[['Year']].values)

# Calcula el MSE
mse = mean_squared_error(df.Income, y_pred)

# Imprime el MSE
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 15545256.606611153
```

```
##Predecimos el valor del ano 2021

lr.predict([[2021]])
```

```
array([41423.87253806])
```



Showing results for ***adjusted*** net national income per capita canada 2021
Search instead for asdjusted net national income per capita canada 2021

Adjusted net national income per capita (current US$) in Canada was reported at 41979 USD in 2021, according to the World Bank collection of development indicators, compiled from officially recognized sources.

Nuestra prediccion para el 2021 fue de 41,423 USD y el valor real fue de 41,979 USD. Por lo que vemos que usando el modelo de scikitlearn, tenemos una buena aproximacion.

Canada - Adjusted Net National Income Per Capita

Reporte

Para este proyecto, utilicé un conjunto de datos proporcionado por el Banco Mundial que contenía información sobre el ingreso per cápita ajustado de Canadá. Al aplicar el método de regresión lineal simple que estudiamos en clase, observé que fue necesario reducir significativamente el tamaño de paso, representado por "alfa". Esta modificación se debió a que, si el tamaño de paso se establecía en un valor más grande, el error cuadrático medio (MSE) aumentaba considerablemente, lo que provocaba que el código generara resultados incoherentes. Aún no he podido identificar la razón exacta detrás de este comportamiento y agradecería mucho su orientación para comprenderlo mejor.

Comenzamos con valores iniciales de m=0 y b=0, lo que resultó en un error significativamente alto. Con el tamaño de paso que utilizamos, nunca alcanzamos una aproximación óptima. Supongo que, si aumentáramos la cantidad de iteraciones, podríamos acercarnos a una mejor solución, pero esto requeriría un tiempo considerable. Sin embargo, al iniciar con otros valores iniciales para m y b, acercándonos a los valores que se encontraron con el modelo de scikit-learn, observamos que logramos una aproximación mucho mejor a los parámetros que minimizan el error con nuestra funcion de costo.

✓ 0s completed at 3:11 AM