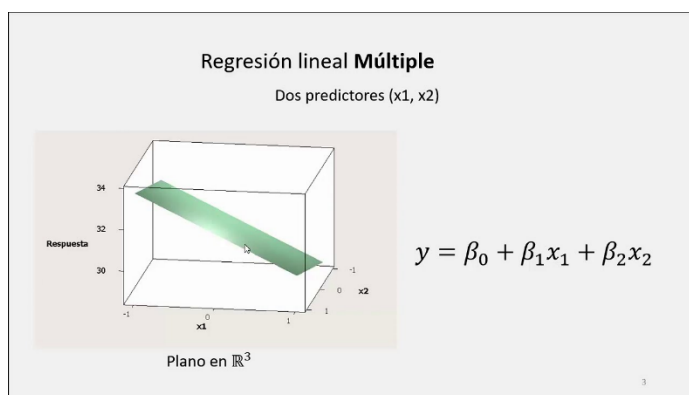




**Centro Universitario de Ciencias Exactas e  
Ingenierías**  
*Universidad de Guadalajara*



**Práctica 2: Regresión Lineal Múltiple**  
*Aprendizaje Máquina*



**Alumno:** Samuel David Pérez Brambila

**Código:** 222966286

**Profesora:** Karla Ávila Cárdenas

**Sección:** D01

**Fecha de Entrega:** 29 de Septiembre de 2024

## Introducción

La regresión lineal es “un modelo matemático que describe la relación entre varias variables. Los modelos de regresión lineal son un procedimiento estadístico que ayuda a predecir el futuro. Se utiliza en los campos científicos y en los negocios, y en las últimas décadas se ha utilizado en el aprendizaje automático. La tarea de la regresión en el aprendizaje automático consiste en predecir un parámetro (Y) a partir de un parámetro conocido X.” (EBAC, 2023)

Pero ¿por qué son importantes los modelos de regresión lineal? De acuerdo con EBAC (2023), son importantes debido a que, debido a su capacidad para transformar datos, pueden utilizarse para simular una amplia gama de relaciones, y debido a su forma, que es más simple que la de las redes neuronales, sus parámetros estadísticos se analizan y comparan con facilidad, lo que permite que se les extraiga información valiosa.

La regresión lineal no sólo se utiliza con fines de predicción: también se ha demostrado su eficacia para describir sistemas. Si se quieren modelar los valores de una variable numérica, se tendrá una lista relativamente corta de variables independientes y, como se espera que el modelo sea comprensible, es probable que se elija la regresión lineal como herramienta de modelización.

Además, encontramos diversos tipos de regresión lineal, los cuales se enlistan a continuación:

- Simple: En una regresión lineal, se trata de establecer una relación entre una variable independiente y su correspondiente variable dependiente. Esta relación se expresa como una línea recta. No es posible trazar una línea recta que pase por todos los puntos de un gráfico si estos se encuentran ordenados de manera caótica. Por lo tanto, sólo se determina la ubicación óptima de esta línea mediante una regresión lineal. Algunos puntos seguirán distanciados de la recta, pero esta distancia debe ser mínima. El cálculo de la distancia mínima de la recta a cada punto se denomina función de pérdida.
- Múltiple: La regresión lineal múltiple encuentra la relación entre dos o más variables independientes y su correspondiente variable dependiente.

En la presente práctica, se trabajará con una regresión lineal múltiple utilizando el lenguaje de programación Python y algunas de sus bibliotecas más populares, como Matplotlib, Pandas, SciKit Learn y Numpy. Estas herramientas son fundamentales para el análisis de datos y la visualización de resultados, permitiendo gestionar y manipular grandes conjuntos de datos de manera eficiente. Pero además, esa misma regresión se realizará con el método del gradiente, que a diferencia de utilizar solamente las librerías de Python, es un proceso un poco más “manual”, ya que se tienen realizar cálculos que las librerías simplifican en gran medida.

## Contenido de la Actividad

El código es un análisis de regresión lineal múltiple que utiliza los datos de bateo de jugadores de béisbol de grandes ligas (MLB) para predecir el promedio de bateo (AVG) en función de los turnos al bate (At-bat) y los hits.

### Regresión lineal múltiple con librerías de Python y con validación cruzada


Empezamos con la importación de las librerías, donde encontramos a numpy (la cual permite un manejo eficiente de arreglos y para realizar operaciones numéricas, esto se verá más adelante), pandas (para la lectura y manipulación de los datos incluidos en el archivo csv), matplotlib (librería necesaria para graficar), mpl\_toolkits.mplot3d.Axes3D (herramienta para generar gráficos 3D), sklearn.linear\_model.LinearRegression (implementa el modelo de regresión lineal múltiple), sklearn.model\_selection.train\_test\_split (separa los datos en conjuntos de entrenamiento y prueba) y sklearn.metrics.mean\_squared\_error (permite calcular el error cuadrático medio (MSE), una métrica para evaluar la precisión del modelo).



```
Prácticas - practica2_librerias.py

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from mpl_toolkits.mplot3d import Axes3D
5 from sklearn.linear_model import LinearRegression
6 from sklearn.model_selection import train_test_split
7 from sklearn.metrics import mean_squared_error
```

La variable **data** se encarga de almacenar el contenido del archivo CSV con apoyo de pandas (pd.read\_csv). Se especifica la codificación latin-1 para garantizar que se lean correctamente los caracteres especiales.



```
Prácticas - practica2_librerias.py

9 # Regresión lineal múltiple usando librerías de Python
10
11 # Cargar el archivo CSV
12 data = pd.read_csv('baseball_hitting.csv', encoding='latin-1')
```

**X** extrae los datos de las columnas 'At-bat' (turnos al bate) y 'Hits' (hits conseguidos) y los convierte en una matriz numpy (np.array). Estas son las variables independientes.

**y** extrae la columna 'AVG' (promedio de bateo) y se define como la variable dependiente que se quiere predecir.

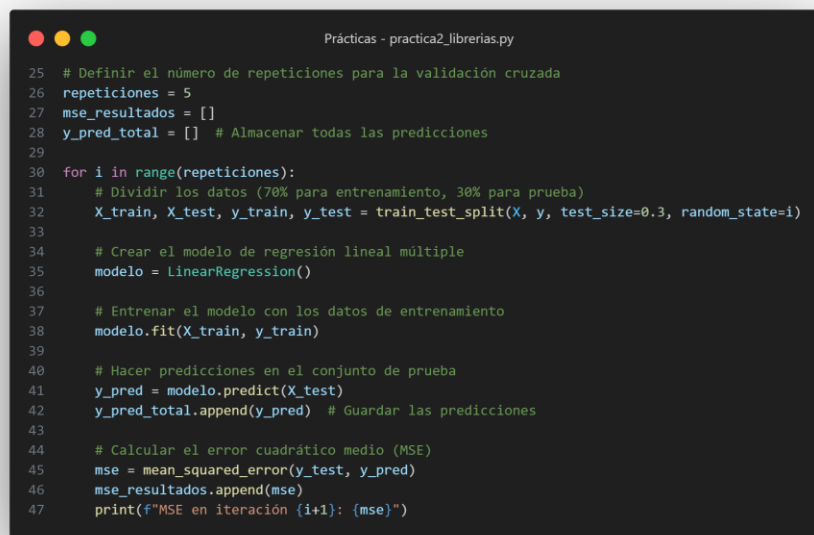
```
Prácticas - practica2_librerias.py
14 # Se forma el arreglo a partir de los datos At-bat (turnos de bateo conseguidos) y Hits (Hits conseguidos), Variables independientes
15 X = np.array(data[['At-bat', 'Hits']])
16
17 # Para la variable dependiente, se usará AVG (promedio de bateo o average)
18 y = np.array(data['AVG'])
```

Previo a seguir avanzando con el código, se realiza el proceso para obtener la correlación que tienen las variables independientes (At-bat y Hits) con la variable dependiente (AVG) esto con apoyo del método **corr()**.

```
Prácticas - practica2_librerias.py
20 # Obtener correlación entre las variables independientes y la dependiente
21 correlaciones = data[['At-bat', 'Hits', 'AVG']].corr()
22 print("Correlación entre variables independientes y dependiente:")
23 print(correlaciones)
```

Para realizar la validación cruzada, se define un número de **repeticiones** o iteraciones que serán las veces en que se dividirá el conjunto de datos en 70% (para entrenamiento) y 30% (para prueba), elegidos al azar. Se medirá la eficiencia del modelo con el cálculo del error cuadrático medio en cada iteración y sacando el promedio (pero esto se ve más adelante). Luego, inicializamos 2 arreglos vacíos llamados **mse\_resultados** y **y\_pred\_total**, el primero se utiliza para ir guardando cada resultado del cálculo de MSE y el segundo sirve para ir guardando las predicciones que se obtienen con los datos de prueba de cada iteración.

Luego, con un bucle for se itera cinco veces (definido por repeticiones), donde cada vez separa los datos de manera diferente y evalúa el modelo, para esto nos apoyamos de las variables **X\_train**, **X\_test**, **y\_train** y **y\_test** en las cuales se almacenarán los respectivos datos que le corresponden y obtenidos con apoyo de **train\_test\_split**, que divide los datos en dos conjuntos: entrenamiento (70%) y prueba (30%). El parámetro **random\_state=i** garantiza que la separación sea diferente en cada iteración.



```

25 # Definir el número de repeticiones para la validación cruzada
26 repeticiones = 5
27 mse_resultados = []
28 y_pred_total = [] # Almacenar todas las predicciones
29
30 for i in range(repeticiones):
31     # Dividir los datos (70% para entrenamiento, 30% para prueba)
32     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=i)
33
34     # Crear el modelo de regresión lineal múltiple
35     modelo = LinearRegression()
36
37     # Entrenar el modelo con los datos de entrenamiento
38     modelo.fit(X_train, y_train)
39
40     # Hacer predicciones en el conjunto de prueba
41     y_pred = modelo.predict(X_test)
42     y_pred_total.append(y_pred) # Guardar las predicciones
43
44     # Calcular el error cuadrático medio (MSE)
45     mse = mean_squared_error(y_test, y_pred)
46     mse_resultados.append(mse)
47     print(f"MSE en iteración {i+1}: {mse}")

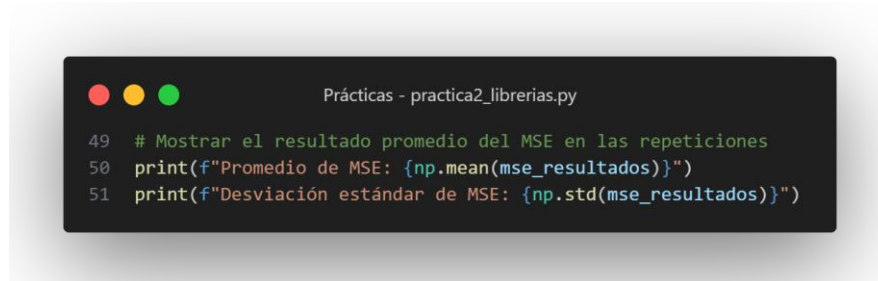
```

Luego, con **modelo = LinearRegression()** se crea el modelo de regresión lineal múltiple y se entrena el modelo con los respectivos datos de entrenamiento previamente otorgados por **train\_test\_split**, esto se visualiza en la línea **modelo.fit(X\_train, y\_train)**.

Con el conjunto de prueba (**X\_test**), a través de **modelo.predict(X\_test)** se obtienen las predicciones y se asigna a la variable **y\_pred**. Cada una de estas predicciones se guarda en el arreglo **y\_pred\_total**, esto será útil más adelante para poder graficar las predicciones en el gráfico 3D.

Para calcular el error cuadrático medio de la respectiva iteración, la cual es una métrica que mide la diferencia promedio entre los valores reales (**y\_test**) y las predicciones (**y\_pred**) y es útil para verificar la eficiencia del modelo de regresión, se utiliza **mean\_squared\_error(y\_test, y\_pred)** que es propio de la librería **scikit learn** y se asigna a la variable **mse**. Cada uno de estos resultados, se guarda en el arreglo previamente definido de **mse\_resultados** y como información se imprime el MSE que se obtiene en cada una de las iteraciones, que posteriormente con el apoyo del arreglo se calculará su promedio para así realizar correctamente la validación cruzada.

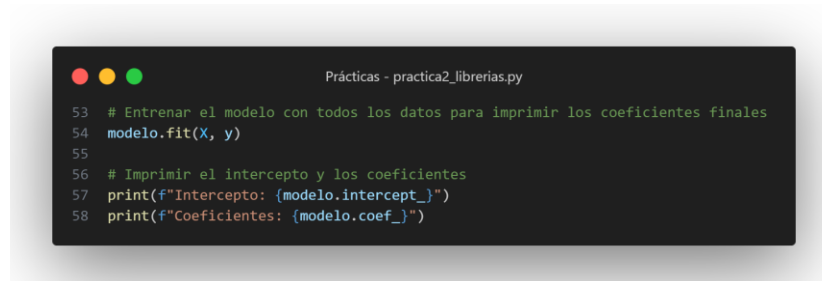
Con apoyo de **np.mean(mse\_resultados)**, se obtiene el promedio de los valores de mse que se fueron obteniendo en cada iteración. Así también, se muestra la desviación estándar (variabilidad de los resultados), con apoyo de **np.std(mse\_resultados)**.



```
Prácticas - practica2_librerias.py

49 # Mostrar el resultado promedio del MSE en las repeticiones
50 print(f"Promedio de MSE: {np.mean(mse_resultados)}")
51 print(f"Desviación estándar de MSE: {np.std(mse_resultados)}")
```

Para obtener el intercepto, como los coeficientes (o betas), se hace un entrenamiento del modelo con todos los datos y se imprimen los resultados, esto con apoyo de **modelo.intercept\_** y **modelo.coef\_**.



```
Prácticas - practica2_librerias.py

53 # Entrenar el modelo con todos los datos para imprimir los coeficientes finales
54 modelo.fit(X, y)
55
56 # Imprimir el intercepto y los coeficientes
57 print(f"Intercepto: {modelo.intercept_}")
58 print(f"Coeficientes: {modelo.coef_}")
```

**plt.figure()** y **fig.add\_subplot(projection='3d')** permiten crear una figura y un gráfico en 3D para visualizar los datos.

Luego se imprime cada dato original, con puntos de color azul; esto se hace con **scatter**, como el gráfico 3D se denominó como **ax**, es por eso que la sentencia es **ax.scatter**.

Después, con **np.meshgrid** se genera una cuadrícula para graficar el hiperplano del modelo. Respetando la ecuación de la regresión múltiple:

$$y = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n$$

Se realiza el cálculo y luego se procede a imprimir la superficie compuesta de las 3 variables (**plot\_surface**).

Luego, procedemos a realizar el proceso de graficar las predicciones previamente obtenidas, **X\_test\_flat** se encarga de "aplanar" o "concatenar" (**np.concatenate**) todos los conjuntos de prueba (**X\_test**) generados en cada repetición del bucle for de validación cruzada. Después, **y\_pred\_flat** concatena todas las predicciones realizadas en las 5 iteraciones, es entonces, que se grafican con **scatter** los puntos de las predicciones de color verde y con forma de "x".

Por último, simplemente definimos los nombres de los ejes del gráfico 3D, se definen los ángulos de visualización iniciales (se puede rotar el modelo cuando este es otorgado), se agrega un cuadro de simbología (con **ax.legend()**) y se muestra la figura o gráfico.

```

Prácticas - practica2_librerias.py

60 # Graficar los datos y el hiperplano resultante en 3D (usando los datos completos)
61 fig = plt.figure()
62 ax = fig.add_subplot(projection='3d')
63
64 # Scatter plot de los datos originales (en azul)
65 ax.scatter(X[:,0], X[:,1], y, c='b', marker='o', label='Datos')
66
67 # Superficie del hiperplano
68 xx, yy = np.meshgrid(X[:,0], X[:,1])
69
70 # Ecuación del hiperplano
71 zz = modelo.intercept_ + modelo.coef_[0] * xx + modelo.coef_[1] * yy
72 ax.plot_surface(xx, yy, zz, color='r', alpha=0.2)
73
74 # Graficar las predicciones en otro color (en verde)
75 X_test_flat = np.concatenate([X_test for _ in range(repeticiones)]) # Concatenar los conjuntos de prueba
76 y_pred_flat = np.concatenate(y_pred_total) # Concatenar todas las predicciones
77 ax.scatter(X_test_flat[:, 0], X_test_flat[:, 1], y_pred_flat, c='g', marker='x', label='Predicciones')
78
79 # Etiquetas de los ejes
80 ax.set_xlabel('At-bat')
81 ax.set_ylabel('Hits')
82 ax.set_zlabel('AVG')
83
84 # Ajustar ángulos de visualización
85 ax.azim = 30
86 ax.elev = 20
87
88 # Añadir leyenda
89 ax.legend()
90
91 plt.show()

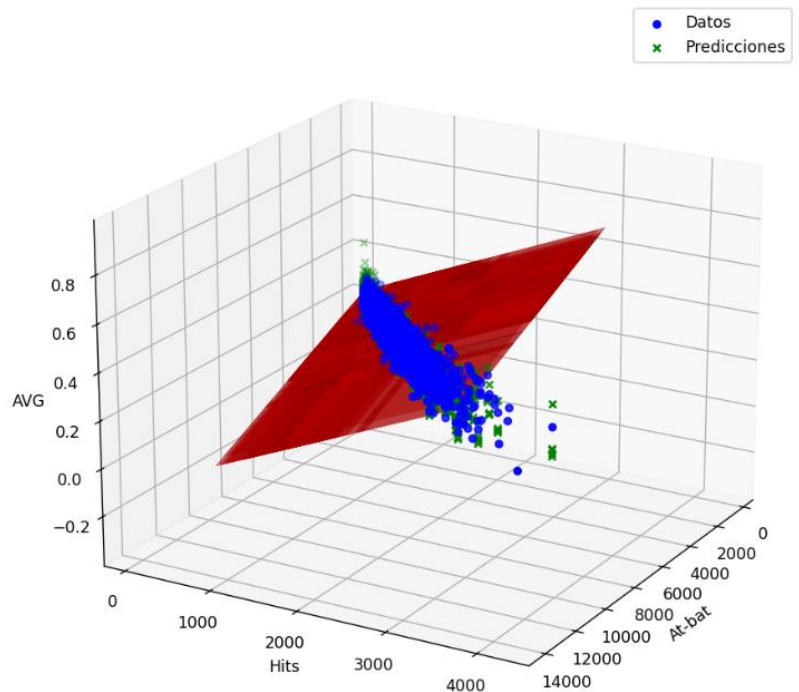
```

Es entonces, que obtenemos:

```

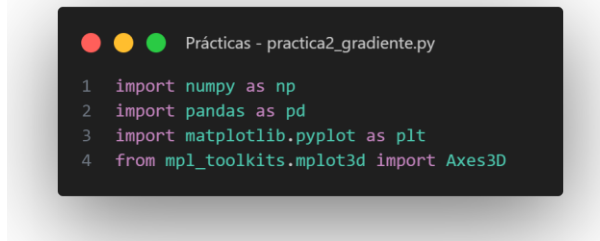
Correlación entre variables independientes y dependiente:
      At-bat      Hits      AVG
At-bat  1.000000  0.990322  0.574635
Hits    0.990322  1.000000  0.654739
AVG     0.574635  0.654739  1.000000
MSE en iteración 1: 0.00020051547010371075
MSE en iteración 2: 0.00017720208193011027
MSE en iteración 3: 0.0001556380499905294
MSE en iteración 4: 0.0001521571135637064
MSE en iteración 5: 0.0001663041939837747
Promedio de MSE: 0.0001703633819143663
Desviación estándar de MSE: 1.7439592049486778e-05
Intercepto: 0.25380308294652715
Coeficientes: [-4.14290909e-05  1.61668193e-04]

```



## Regresión lineal múltiple con método del gradiente y con validación cruzada K-Folk

Empezamos con la importación de las librerías, donde encontramos a numpy (la cual permite un manejo eficiente de arreglos y para realizar operaciones numéricas, esto se verá más adelante), pandas (para la lectura y manipulación de los datos incluidos en el archivo csv), matplotlib (librería necesaria para graficar) y mpl\_toolkits.mplot3d.Axes3D (herramienta para generar gráficos 3D).



```
Prácticas - practica2_gradiente.py
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from mpl_toolkits.mplot3d import Axes3D
```

Definimos la clase **RegresionLinealMultiple** con diferentes métodos que permiten realizar todos los cálculos necesarios, dichos métodos se explican a continuación:

- **\_\_init\_\_**: Método constructor que inicializa el modelo con una tasa de aprendizaje y un número de iteraciones. También inicializa los coeficientes y el intercepto.
- **fit**: Método para entrenar el modelo. Incluye la variable **X\_con\_intercepto** que se le asigna el valor de **np.column\_stack**, el cual agrega una columna de unos a X para incluir el término de sesgo (intercepto). **self.coef\_** que inicializa los coeficientes de manera aleatoria con apoyo de **np.random.randn**. Luego, se incluye un bucle de gradiente descendente (for) el cual se ejecuta n\_iter veces, dicho bucle calcula el gradiente, que es la derivada de la función de pérdida con respecto a los coeficientes, además actualiza los coeficientes usando la tasa de aprendizaje. **self.intercept\_** toma el primer coeficiente como intercepto y el resto en **self.coef\_** como los coeficientes de las variables independientes.
- **predict**: Método para hacer predicciones en nuevos datos. Al igual que el anterior, se agrega una columna de unos y calcula las predicciones.
- **cross\_validate**: Método para realizar validación cruzada con k-folds, para ello, se divide los datos en k pliegues para evaluar el modelo. Inicializa una lista para almacenar los errores cuadráticos medios (MSE), luego con un bucle for se itera sobre cada pliegue, definiendo los índices de inicio y fin para el conjunto de prueba, luego con **X\_train**, **y\_train**, **X\_test** y **y\_test** se crean conjuntos de entrenamiento y prueba, los datos de prueba son los que están en el rango de start a end y el resto se utiliza para el entrenamiento. Entrena el modelo con el conjunto de entrenamiento y predice el conjunto de prueba y calcula el MSE para las predicciones y lo agrega a la lista de



puntuaciones. Para finalizar devuelve el MSE promedio de todas las iteraciones de validación cruzada con K-Folk.

```
Prácticas - practica2_gradiente.py

6 # Regresión lineal múltiple con método del gradiente
7
8 class RegresionLinealMultiple:
9     def __init__(self, learning_rate=0.01, n_iter=1000):
10         self.learning_rate = learning_rate
11         self.n_iter = n_iter
12         self.coef_ = None
13         self.intercept_ = None
14
15     def fit(self, X, y):
16         # Agregar una columna de unos para el término de sesgo (intercepto)
17         X_con_intercepto = np.column_stack((np.ones(len(X)), X))
18
19         # Inicializar los coeficientes aleatoriamente
20         self.coef_ = np.random.randn(X_con_intercepto.shape[1])
21
22         # Gradiente descendente
23         for _ in range(self.n_iter):
24             gradient = -2 * X_con_intercepto.T @ (y - X_con_intercepto @ self.coef_)
25             self.coef_ -= self.learning_rate * gradient
26
27         # El primer valor de coef_ corresponde al intercepto
28         self.intercept_ = self.coef_[0]
29         self.coef_ = self.coef_[1:]
30
31     def predict(self, X):
32         # Agregar una columna de unos para el término de sesgo (intercepto)
33         X_con_intercepto = np.column_stack((np.ones(len(X)), X))
34         return X_con_intercepto @ np.concatenate([self.intercept_, self.coef_])
35
36     def cross_validate(self, X, y, k=5):
37         fold_size = len(X) // k
38         mse_scores = []
39
40         for i in range(k):
41             # Crear conjuntos de entrenamiento y prueba
42             start = i * fold_size
43             end = (i + 1) * fold_size if i < k - 1 else len(X)
44
45             X_train = np.concatenate((X[:start], X[end:]))
46             y_train = np.concatenate((y[:start], y[end:]))
47             X_test = X[start:end]
48             y_test = y[start:end]
49
50             self.fit(X_train, y_train)
51             predictions = self.predict(X_test)
52             mse = np.mean((y_test - predictions) ** 2)
53             mse_scores.append(mse)
54
55         return np.mean(mse_scores)
```

Luego de terminar de definir la clase con los respectivos métodos, es entonces que definimos los valores para las variables independientes (que en este caso y repitiendo la misma definición del método de las librerías, son At-bat y hits) y la variable dependiente (AVG), pero para ello se deben de obtener de un dataset que está en un archivo csv, es por ello que se define **data** que con el método de pandas **pd.read\_csv** podemos obtener la información contenida en este, para no tomar toda la información del archivo (variables que no necesitamos), es entonces que se crean 2 arreglos con apoyo de **np.array**, lo que facilita el resto del proceso.

```
Prácticas - practica2_gradiente.py

57 # Cargar el archivo CSV
58 data = pd.read_csv('baseball_hitting.csv', encoding='latin-1')
59
60 # Variables independientes (At-bat y Hits) y dependiente (AVG)
61 X = np.array(data[['At-bat', 'Hits']])
62 y = np.array(data['AVG'])
```

Procedemos a crear el modelo de regresión lineal múltiple, donde se crea una instancia de la clase previamente definida y otorgándole los argumentos solicitados, cabe aclarar que los valores que se le asignaron en este caso fueron un poco a prueba y error, ya que por ejemplo si el valor de `learning_rate` se incrementaba, los gradientes tendían a infinito y daban errores en los demás cálculos).

```
Prácticas - practica2_gradiente.py

64 # Crear el modelo de regresión lineal múltiple
65 modelo = RegresionLinealMultiple(learning_rate=0.000000000019, n_iter=10000)
```

Luego, se llama al método de la validación cruzada, se debe dejar en claro que dicho método trabaja con los demás métodos para realizar las predicciones, así como todos los cálculos necesarios.

```
Prácticas - practica2_gradiente.py

67 # Validación cruzada
68 mse = modelo.cross_validate(X, y)
69 print("MSE promedio de la validación cruzada:", mse)
```

Después de haber realizado el proceso de validación, procedemos a entrenar el modelo con todos los datos, esto con el fin de obtener el intercepto y coeficientes finales. **fig** y **ax** a través de `plt.figure()` y `fig.add_subplot(projection='3d')`, permite definir el espacio de trabajo para el gráfico, además de definir que es una proyección en 3D.

```
Prácticas - practica2_gradiente.py

71 # Entrenar el modelo completo
72 modelo.fit(X, y)
73
74 # Graficar los datos y el hiperplano resultante en 3D
75 fig = plt.figure()
76 ax = fig.add_subplot(projection='3d')
```

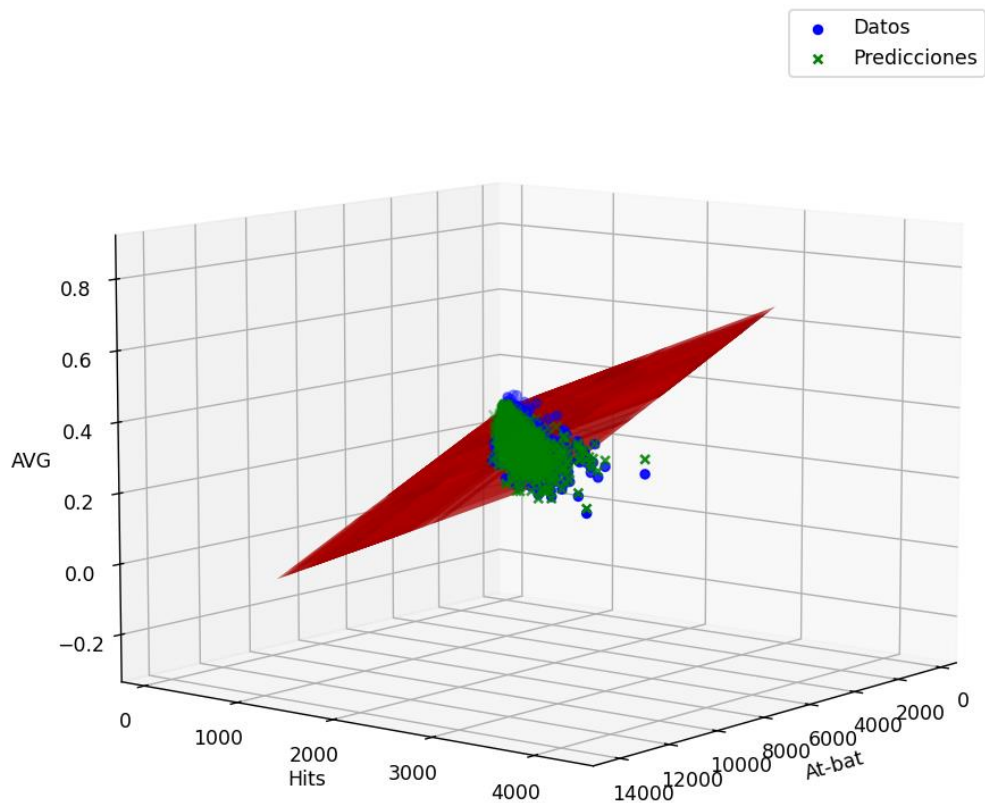
El resto del código se mantiene igual al método anterior, donde se hace la impresión (scatter) de los puntos originales, luego se calcula con la ecuación del hiperplano, sus respectivas dimensiones y posicionamiento en el gráfico, para diferenciarlo de los otros datos se coloca en color rojo ('r'), al igual que se grafican los datos originales, se grafican las predicciones pero en color verde y con marcas en forma de "x" para diferenciarlos, así también definimos una "simbología" para que el usuario pueda distinguir cuáles son los originales y cuáles son las predicciones. Por último, se ajustan los ángulos iniciales de visualización del gráfico (pero igual al hacer el plt.show()) el gráfico puede irse rotando a gusto del usuario, por lo cual no es muy relevante. Finalmente, se imprimen los valores del intercepto, de los coeficientes y se muestra el gráfico junto a la simbología.

```
Prácticas - practica2_gradiente.py

78 # Scatter plot de los datos originales (en azul)
79 ax.scatter(X[:,0], X[:,1], y, c='b', marker='o', label='Datos')
80
81 # Superficie del hiperplano
82 xx, yy = np.meshgrid(X[:,0], X[:,1])
83
84 # Ecuación del hiperplano
85 zz = modelo.intercept_ + modelo.coef_[0] * xx + modelo.coef_[1] * yy
86 ax.plot_surface(xx, yy, zz, color='r', alpha=0.2)
87
88 # Graficar las predicciones
89 predicciones = modelo.predict(X)
90 ax.scatter(X[:,0], X[:,1], predicciones, c='g', marker='x', label='Predicciones')
91
92 # Etiquetas de los ejes
93 ax.set_xlabel('At-bat')
94 ax.set_ylabel('Hits')
95 ax.set_zlabel('AVG')
96
97 # Ajustar ángulos de visualización
98 ax.azim = 60
99 ax.elev = 25
100
101 # Coeficientes de la regresión (intercepto y coeficientes)
102 print("Intercepto: ", modelo.intercept_)
103 print("Coeficientes: ", modelo.coef_)
104
105 # Añadir leyenda
106 ax.legend()
107
108 plt.show()
```

Obteniendo:

```
MSE promedio de la validación cruzada: 0.2133292306394683  
Intercepto: 0.2742725408020357  
Coeficientes: [-3.81986096e-05 1.35752864e-04]
```



## Conclusiones

En conclusión, la implementación de la regresión lineal múltiple en Python con el uso de bibliotecas, así como con el método del gradiente, permitió analizar de manera eficiente la relación entre la variable independiente y las variables dependientes del conjunto de datos que está contenido en el archivo .csv, donde con la verificación de la correlación, así como una relación “lógica” en base a las reglas del deporte se decidieron que “At-bat” y “Hits” serían las variables dependientes y “AVG” la respectiva variable independiente.

Los resultados obtenidos evidencian que, para llevar a cabo una regresión lineal múltiple, es más sencillo y preciso utilizar las bibliotecas de Python, especialmente SciKit Learn, que proporciona herramientas robustas y optimizadas. En contraste, al aplicar el método del gradiente, se constató que este puede ser más susceptible a errores, dependiendo de la calidad de los datos y de los parámetros de aprendizaje e iteraciones.

La regresión lineal múltiple es especialmente útil en situaciones donde hay múltiples factores que influyen en una variable dependiente, ya que permite modelar relaciones más complejas al considerar múltiples variables independientes simultáneamente, como lo fue este caso donde a través de 2 variables independientes (turnos de bateo y hits) se pudo diseñar un modelo de regresión para predecir el valor del promedio de bateo o AVG, que es un cálculo que si se realiza manualmente, llega a ser muy tardado y susceptible a errores debido a la gran cantidad de datos que se van actualizando día con día, oportunidad de bateo tras oportunidad de bateo del jugador.

Además, la regresión lineal múltiple resulta ventajosa en contextos donde una sola variable no puede capturar adecuadamente el comportamiento del fenómeno en estudio. Sin embargo, es importante reconocer que la regresión lineal simple también es muy útil, siendo ideal para situaciones más sencillas o cuando se desea una interpretación más directa. Ambas técnicas son útiles y pueden ser elegidas en función de la complejidad del problema y la cantidad de datos disponibles.

## Referencias

- Bobadilla, J. (2020). *Machine Learning y Deep Learning Usando Python, Scikit y Keras* (1ª ed.). Ra-Ma Editorial; Ediciones de la U. <https://elibro-net.wdg.biblio.udg.mx:8443/es/lc/udg/titulos/222698>
- EBAC (2023, 3 de mayo). *Regresión Lineal: teoría y ejemplos*. Escuela Británica de Artes Creativas y Tecnología. Recuperado el 12 de Septiembre de 2024 de: <https://ebac.mx/blog/regreson-lineal>