Trabajo práctico 1 - Regresión lineal

Integrantes:

- Pablo Terrone. Legajo: 17554/1
- Matías Terrone. Legajo: 25278/6
- Camila Florencia Tarrío. Legajo: 21016/6

Google Colab: https://colab.research.google.com/drive/1qZUCl6gJ8UCpFH4LkyzB9RTl1LWeME0j?usp=sharing

Lo primero que hicimos fue tomar el csv *players_21.csv,* y borrar todas las columnas que contengan datos no cuantitativos, utilizando el siguiente script en Python con la libreria Pandas.

```python
def remove_non_numeric_columns(csv_file, output_file):
    # Leer el CSV
    df = pd.read_csv(csv_file)

    # Verifico que columnas tienen numeros
    numeric_columns = df.apply(lambda col: pd.to_numeric(col, errors='coerce')).notna().any(

    # Filtro las columnas que contienen números
    df_filtered = df.loc[:, numeric_columns]

    # Guardo el resultado en un nuevo CSV
    df_filtered.to_csv(output_file, index=False)
    print(f"Archivo guardado sin columnas no numéricas: {output_file}")

input_csv = './data/players_21.csv'
output_csv = './data/players_21_limpio.csv'
remove_non_numeric_columns(input_csv, output_csv)
```

Luego, realizamos un script con Pandas y Numpy que calcula el coeficiente de correlacion de una variable pasada por parametro.

```python
# Fijamos la columna y ('value_eur') para evitar repetir los calculos en cada columna
y = df['value_eur']

# Calculamos la media y desviación estándar de 'y'
mean_y = np.mean(y)
std_y = np.std(y)

def calcular_correlacion(columna_x):
    # Obtenemos la columna 'x' del dataframe por nombre
    x = df[columna_x]

    # Calculamos la media y desviación estándar de 'x'
    mean_x = np.mean(x)
    std_x = np.std(x)

    # Calculamos la covarianza entre 'x' y 'y'
    cov = np.mean((x - mean_x) * (y - mean_y))

    # Calculamos el coeficiente de correlación
    corr = cov / (std_x * std_y)

    return corr
```

Utilizamos esta función para iterar sobre las columnas calculando sus coeficientes de correlación:

```
Correlación entre international_reputation y 'value_eur': 0.5641924047531065
Correlación entre movement_reactions y 'value_eur': 0.5580710917045648
Correlación entre gk_handling y 'value_eur': 0.5229123820277584
Correlación entre gk_reflexes y 'value_eur': 0.5197747027793406
Correlación entre gk_diving y 'value_eur': 0.5177861311368185
Correlación entre gk_positioning y 'value_eur': 0.4880858516257015
Correlación entre passing y 'value_eur': 0.4654025626770451
Correlación entre dribbling y 'value_eur': 0.44942886526408077
Correlación entre mentality_composure y 'value_eur': 0.44707560566064863
Correlación entre gk_kicking y 'value_eur': 0.4440211050448146
Correlación entre mentality_vision y 'value_eur': 0.36641008317431617
Correlación entre power_shot_power y 'value_eur': 0.35421248441167724
Correlación entre shooting y 'value_eur': 0.3423525040947196
Correlación entre attacking_short_passing y 'value_eur': 0.32967227687745154
Correlación entre skill_long_passing y 'value_eur': 0.3118199552110246
Correlación entre skill_ball_control y 'value_eur': 0.30046610624005893
Correlación entre skill_moves y 'value_eur': 0.2937222892441878
Correlación entre skill_curve y 'value_eur': 0.2906953535626929
Correlación entre attacking_volleys y 'value_eur': 0.2702101230406138
Correlación entre skill_dribbling y 'value_eur': 0.2692412237686186
Correlación entre power_long_shots y 'value_eur': 0.26884890401714123
Correlación entre attacking_crossing y 'value_eur': 0.2607274194062276
```

Esto fue realizado con la intención de encontrar las variables independientes que más influyan en la variable dependiente

Parte 1

Predicción del valor de mercado

a) **Recta de regresión para predecir el valor de mercado de un jugador** a partir de la característica más relevante (a la que se destinará mayor proporción del presupuesto), respaldada por:

i) Prueba de significancia de regresión, coeficiente de determinación (R2) y correlación lineal (r).

Para este punto realizamos la prueba de significancia para evaluar si los coeficientes de la regresión son estadísticamente significativos. Se calculó la pendiente (β1\beta_1) y la intersección (β0\beta_0) de la recta de regresión lineal que modela la relación entre el salario en euros (wage_eur) y el valor de mercado en euros (value_eur). Además veremos, mediante el Coeficiente de Determinación (R^2), la proporción de la varianza en el valor de mercado que es explicada por el salario, mientras que el coeficiente de correlación nos muestra la fuerza y la dirección de la relación lineal entre ambas variables. (Ambos indicadores dan resultados favorables)

```
# Seleccionar las columnas relevantes
X = df['wage_eur'].values
Y = df['value_eur'].values

# Filtrar valores nulos
mask = ~np.isnan(X) & ~np.isnan(Y)
X = X[mask]
Y = Y[mask]

# Calcular los coeficientes de la recta de regresión
m = len(X)  # Número de ejemplos
X_mean = np.mean(X)
Y_mean = np.mean(Y)

# Pendiente (beta_1) y la intersección (beta_0)
beta_1 = np.sum((X - X_mean) * (Y - Y_mean)) / np.sum((X - X_mean
beta_0 = Y_mean - beta_1 * X_mean
```

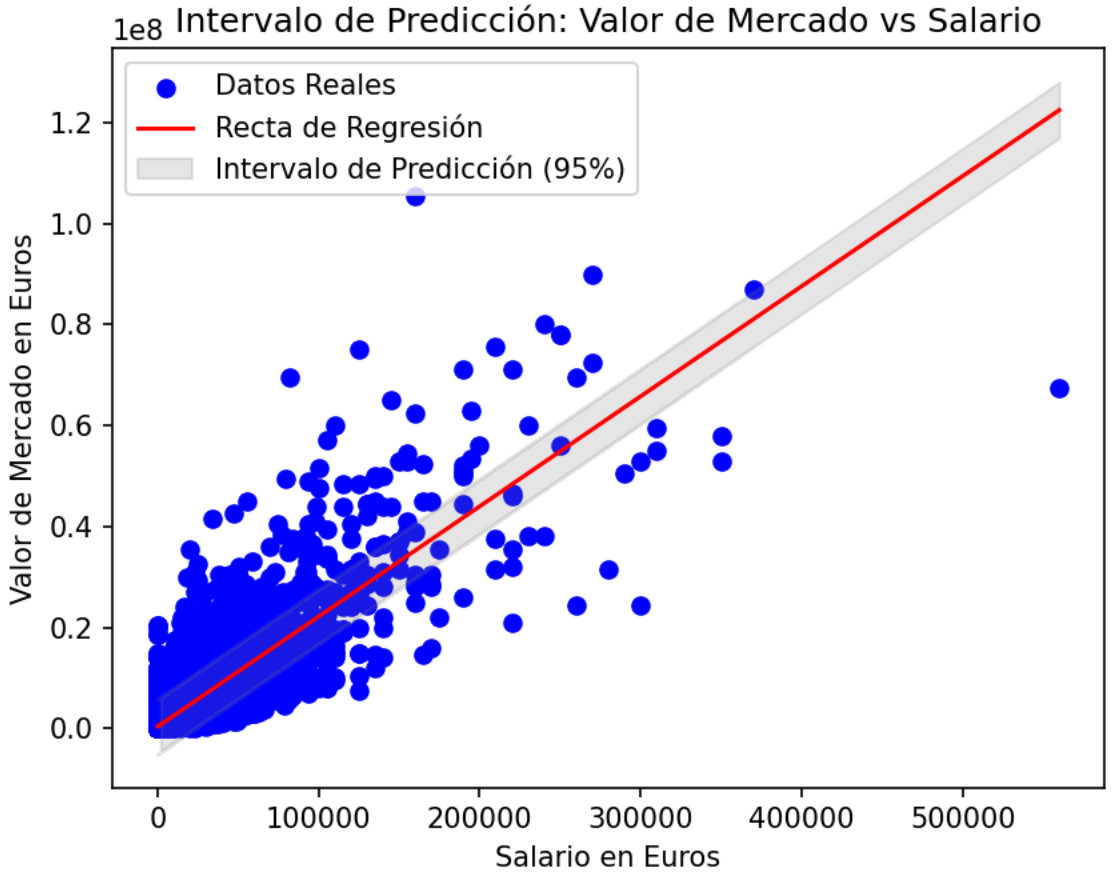
```
# Predicciones
Y_pred = beta_0 + beta_1 * X

# Coeficiente de determinación (R²)
ss_total = np.sum((Y - Y_mean) ** 2)
ss_residual = np.sum((Y - Y_pred) ** 2)
r_squared = 1 - (ss_residual / ss_total)

# Correlación lineal (r)
correlation = np.corrcoef(X, Y)[0, 1]

# Calcular el error estándar de los coeficientes
residuals = Y - Y_pred
se = np.sqrt(np.sum(residuals**2) / (m - 2))  # Error estándar de

# Error estándar de beta_1 y beta_0
se_beta_1 = se / np.sqrt(np.sum((X - X_mean) ** 2))
se_beta_0 = se * np.sqrt(1/m + (X_mean**2 / np.sum((X - X_mean) *

# Prueba de significancia (valor t y valor p)
t_beta_1 = beta_1 / se_beta_1
t_beta_0 = beta_0 / se_beta_0

# Grados de libertad
df = m - 2

# Valores p
p_value_beta_1 = 2 * (1 - stats.t.cdf(np.abs(t_beta_1), df))
p_value_beta_0 = 2 * (1 - stats.t.cdf(np.abs(t_beta_0), df))
```

Los resultados que obtenemos son:

```
Pendiente (beta_1): 218.1012721707944, Error estandar: 1.023089126394174, t: 213.17915179050075, p-value: 0.0000000000,
Intervalo de confianza: (216.09592619169874, 220.10661814989007)
Interseccion (beta_0): 332598.82162815845, Error estandar: 21979.99785425026, t: 15.131885991692396, p-value: 0.0000000000,
Intervalo de confianza: (289516.06453455007, 375681.5787217668)
Coeficiente de determinacion (R^2): 0.7058117817091276
Correlacion lineal (r): 0.8401260510834825
```

En este caso, la cantidad de valores y los outliers que tiene el dataset, producen que nuestro estadístico de valores altos, lo cual genera un p-valor = 0.

Si recortamos nuestro dataset, los valores se estabilizan, pero el modelo pierde calidad.



ii) Inferencias sobre los parámetros de la recta, estimando las fluctuaciones con una confianza del 95%.

Para evaluar la confiabilidad de las estimaciones de los coeficientes, calculamos los intervalos de confianza al 95% para β1 y β0. Esto permite estimar el rango en el cual se espera que se encuentren los coeficientes poblacionales con una confianza del 95%. (ver el gráfico del punto anterior)

```
# Intervalos de confianza del 95%
t_value = stats.t.ppf(0.975, m - 2)  # t crítico para 95% de conf

# Intervalos de confianza para beta_1 y beta_0
beta_1_ci = (beta_1 - t_value * se_beta_1, beta_1 + t_value * se_
beta_0_ci = (beta_0 - t_value * se_beta_0, beta_0 + t_value * se_
```

Los resultados de los intervalos, anticipados en el inciso anterior, son:

```
Pendiente (beta_1): 218.1012721707944, Error estandar: 1.023089126394174, t: 213.17915179050075, p-value: 0.0000000000,
Intervalo de confianza: (216.09592619169874, 220.10661814989007)
Interseccion (beta_0): 332598.82162815845, Error estandar: 21979.99785425026, t: 15.131885991692396, p-value: 0.0000000000,
Intervalo de confianza: (289516.06453455007, 375681.5787217668)
```

iii) La proporción de veces que el valor de mercado supera la incertidumbre de predicción comparada con la respuesta media del valor de mercado para una característica fija, ambas con la misma confianza y ancho mínimo.

Calculamos los intervalos de predicción para estimar la incertidumbre en las predicciones del valor de mercado en función del salario. Definimos límites superiores e inferiores para el intervalo de predicción del 95%.

```
# Calcular el error estándar de la predicción para el inciso iii)
se_prediction = se * np.sqrt(1 + 1/m + (X - X_mean)**2 / np.sum((

# Intervalo de predicción del 95%
t_value = stats.t.ppf(0.975, m - 2)  # Valor t crítico para 95% c
prediction_interval_upper = Y_pred + t_value * se_prediction
prediction_interval_lower = Y_pred - t_value * se_prediction

# Comparación del valor de mercado real con el intervalo de predi
# Verificar cuántas veces el valor de mercado real está por encim
above_upper_bound = Y > prediction_interval_upper
proportion_above = np.sum(above_upper_bound) / m

# Verificar cuántas veces el valor de mercado real está por debaj
below_lower_bound = Y < prediction_interval_lower
proportion_below = np.sum(below_lower_bound) / m

# Proporción total fuera del intervalo de predicción
proportion_outside_interval = proportion_above + proportion_below
```

Los resultados fueron:

```
Proporcion de veces que el valor de mercado esta fuera del intervalo de prediccion: 0.0449
Proporcion por encima del intervalo de prediccion: 0.0312
Proporcion por debajo del intervalo de prediccion: 0.0137
```

Respondiendo la pregunta del punto iii), la proporción de veces que el valor de mercado supera la incertidumbre es 0.0312

Ver código completo en:
https://colab.research.google.com/drive/1lUY2pS09BBcCUOR3SKKLGi3_rqWU38vj?usp=sharing

Parte 2

Ecuación de predicción del valor de mercado

b) *Ecuación para predecir el valor de mercado del jugador a partir de varias características.*

i) Usando el método de mínimos cuadrados. Explica los indicadores obtenidos (como el coeficiente de determinación y la correlación) y proporciona una breve interpretación de los resultados.

Para realizar este punto, debemos seleccionar las características que vemos que influyen en el valor de los jugadores en mayor medida. En este caso estas características son lo que conforman nuestras variables independientes (X), mientras que nosotros buscamos predecir la variable dependiente (Y).

En particular, nuestras variables independientes van a ser: *'wage_eur*', '*overall*' y '*potential '*

Y nuestra variable dependiente: *'value_eur'*.

Para calcular los coeficientes, usaremos la fórmula que definimos como:

$$\beta=(XTX)\text{-}1XTY$$

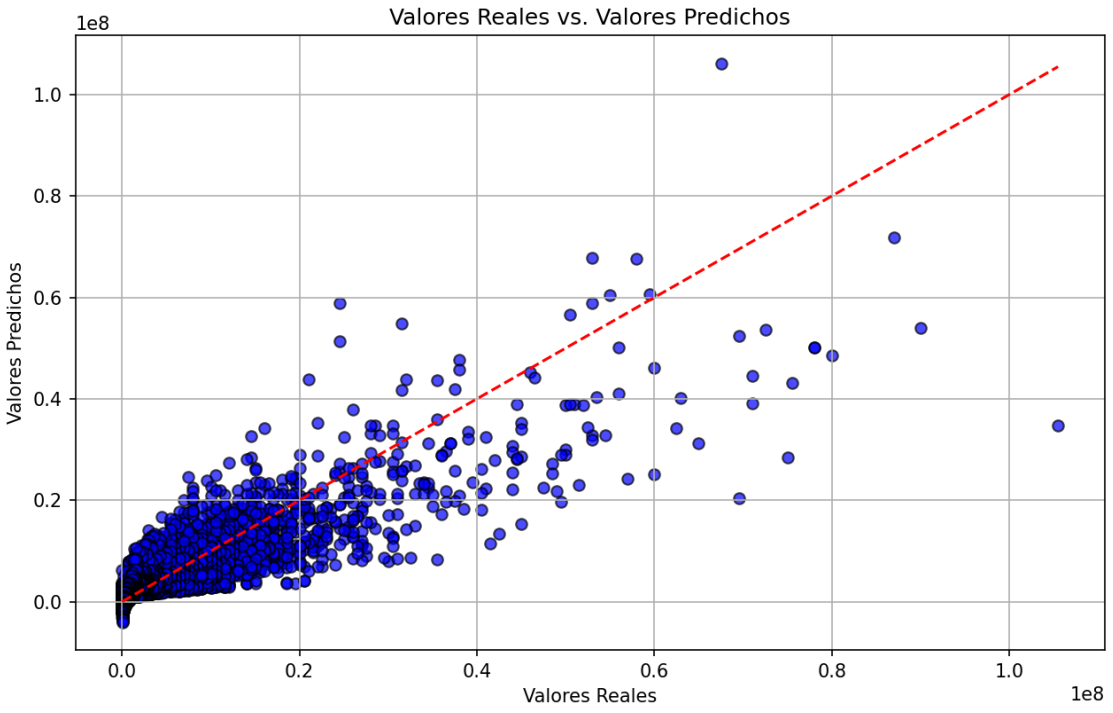
La función de la estimación tiene la forma:

$$y = \beta 0 + \beta 1\, x1 + \beta 2\, x2 + \cdots + \beta k\, xk$$

(Solo que en este caso, nuestros coeficientes van a ser 3, igual que las variables independientes, más el intercepto)

```
# Seleccionamos algunas características relevantes
features = ['wage_eur','overall','potential']

# Verifica si esas columnas contienen valores nulos
print(df[features].isnull().sum())

# Elimina filas con NaN en las columnas seleccionadas
df = df.dropna(subset=features)

# Definir las variables independientes y dependientes
```

```
X = df[features]
Y = df['value_eur']

# Añadir una columna de unos para el término independiente (inter
X = np.c_[np.ones(X.shape[0]), X]

# Convertir X e Y a matrices NumPy
X = np.array(X)
Y = np.array(Y)

# Calcular los coeficientes usando la fórmula de mínimos cuadrado
beta = np.linalg.inv(X.T @ X) @ X.T @ Y
```



Luego podemos analizar los indicadores que calculamos (la estimación de la varianza, el coeficiente de determinación y el de correlación)

```
# Calcular el coeficiente de determinación (R^2)
Y_pred = X @ beta
SS_total = np.sum((Y - np.mean(Y)) ** 2)
SS_residual = np.sum((Y - Y_pred) ** 2)
R2 = 1 - (SS_residual / SS_total)

n = len(Y)  # Número de observaciones
```

```
k = X.shape[1] - 1  # Número de variables independientes (restamo

# Cálculo de la estimación de la varianza residual (σ^2)
varianza_residual = SS_residual / (n - k - 1)
print(f'Varianza residual estimada (sigma^2): {varianza_residual}

# Calcular la correlación
correlation = math.sqrt(R2)
```

Los valores que obtenemos son:

```
Varianza residual estimada (sigma^2): 6423458330827.832
Coeficientes del modelo: ['-15245125.04', '178.44', '96825.93', '134518.57']
Coeficiente de determinacion (R^2): 0.7533185548063746
Correlacion entre valores reales y predicciones: 0.8679392575557201
```

- La estimación de la varianza se basa en la suma de los residuos al cuadrado y determina la variabilidad en el modelo, un valor grande indica que los puntos observados están bastante dispersos en torno a la recta de regresión.
- El coeficiente de determinación varía entre los valores 0 y 1 (siendo 1 un modelo perfecto). Significa que el 75% de la variabilidad en el valor de mercado de los jugadores puede ser explicado por las características que fueron seleccionadas para el modelo, como también significa que el 25% de la variabilidad no está siendo explicado por las características.
- El coeficiente de correlación lineal varía entre -1 y 1 (siendo 1 una correlación perfecta) e indica cuán fuerte es la relación lineal. Tanto el coeficiente de determinación como el de correlación indican valores alentadores.

ii) Usando el método de descenso por gradiente. ¿Son los valores obtenidos iguales a los conseguidos mediante la resolución del sistema de ecuaciones normales? Muestra los resultados obtenidos junto con las últimas iteraciones del algoritmo. Indica los valores de los parámetros utilizados (como tasa de aprendizaje y número de iteraciones).

Para este punto trabajamos como en el ejercicio anterior. Solo que ahora usaremos el método del descenso del gradiente.

```
# Parámetros iniciales
theta = np.zeros(X.shape[1])  # Inicializar los coeficientes en 0
alpha = 0.001  # Tasa de aprendizaje
```

```
tol = 0.01  # Tolerancia para el error
max_iterations = 10000  # Máximo número de iteraciones

# Función de costo (Error Cuadrático Medio)
def compute_cost(X, Y, theta):
    m = len(Y)
    predictions = X @ theta
    cost = (1 / (2 * m)) * np.sum((predictions - Y) ** 2)
    return cost

# Actualización de coeficientes usando descenso por gradiente
def gradient_descent(X, Y, theta, alpha, tol, max_iterations):
    m = len(Y)
    data = {'iteration': [], 'prev': [], 'new': [], 'fnew': [], '
    cost_history = []

    iteration = 0
    error = float('inf')  # Inicializamos el error con un valor a

    while iteration < max_iterations and error > tol:
        iteration += 1
        predictions = X @ theta
        prev_theta = theta.copy()
        theta -= (alpha / m) * (X.T @ (predictions - Y))
        cost = compute_cost(X, Y, theta)
        cost_history.append(cost)

        # Calcular el error absoluto entre los coeficientes previ
        error = np.linalg.norm(theta - prev_theta)

        # Guardar información para análisis
        add_row(data, iteration, prev_theta, theta, cost, error)

        # Mostrar las últimas 5 iteraciones
        if iteration >= max_iterations - 5:
            print(f"Iteracion {iteration}: Coeficientes: {theta},

    return theta, cost_history, data

# Función auxiliar para agregar datos en cada iteración
def add_row(data: dict, iteration, prev, new, fnew, error):
```

```
    data['iteration'].append(iteration)
    data['prev'].append(prev)
    data['new'].append(new)
    data['fnew'].append(fnew)
    data['error'].append(error)
```

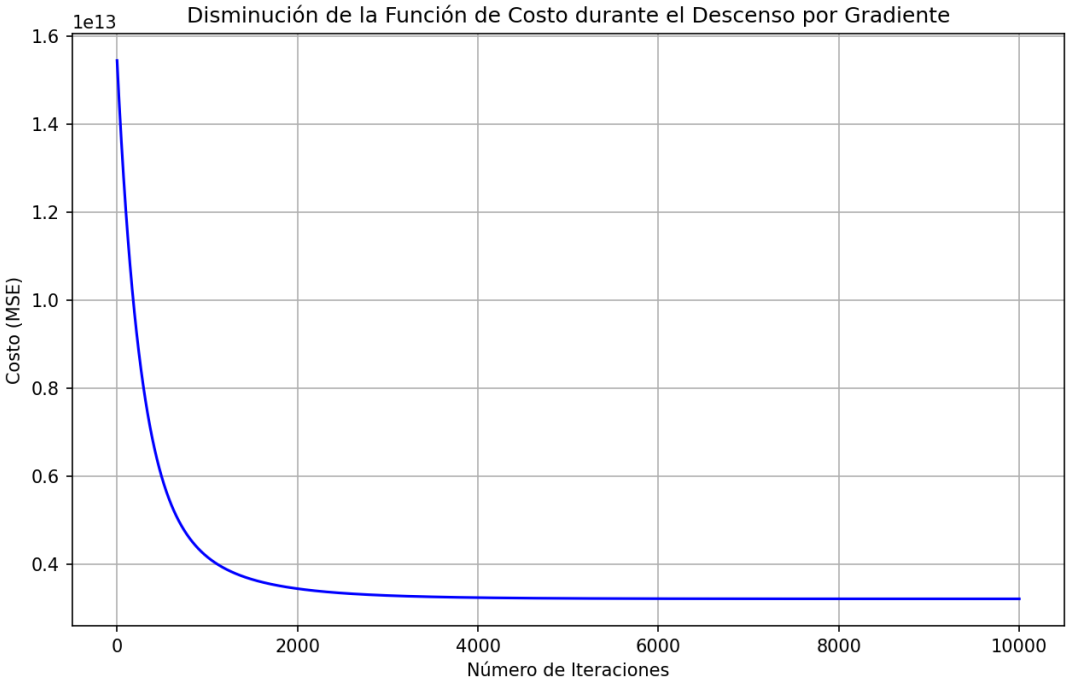
A la hora de realizar el cálculo, el algoritmo tiene problemas con los valores, los cuales producen divergencia y no logran converger a una solución. Al achicar la tasa de aprendizaje a valores extremadamente bajos logramos generar una solución que no es similar a la que conseguimos mediante el método de mínimos cuadrados. Aunque si aplicamos una normalización en los valores, nuestro modelo funciona bien, produce resultados que, al desnormalizarlos, se transforman en casi los mismos que conseguimos con el método del ejercicio anterior:

```
Tiempo de ejecucion: 0.7969 segundos
Coeficiente de determinacion (R^2): 0.7533043286506678
Coeficiente de correlacion (r): 0.8679310621533647
Ecuacion del modelo de regresion lineal:
y = -15354128.83 + (177.58 * X1) + (100408.44 * X2) + (132845.66 * X3)
```

Estas son las últimas iteraciones con los valores aún normalizados:

```
Iteracion 9995: Coeficientes: [2224712.28552451 3490202.53218347  703113.65555796  811652.1103302 ], Costo: 3211236886790.709,
Error: 11.777918654492792
Iteracion 9996: Coeficientes: [2224712.38653079 3490209.91105258  703104.77187377  811654.39824845], Costo: 3211236748205.1274,
Error: 11.773373615988106
Iteracion 9997: Coeficientes: [2224712.48743607 3490217.28655802  703095.89138014  811656.68602933], Costo: 3211236609726.4785,
Error: 11.768830481576291
Iteracion 9998: Coeficientes: [2224712.58824044 3490224.6587014   703087.0140759   811658.97367256], Costo: 3211236471354.678,
Error: 11.76428250142738
Iteracion 9999: Coeficientes: [2224712.688944   3490232.02748432  703078.13995989  811661.26117783], Costo: 3211236333089.639,
Error: 11.75974992071405
Iteracion 10000: Coeficientes: [2224712.78954687 3490239.39290836  703069.26903093  811663.54854484], Costo: 3211236194931.2773,
Error: 11.755212492715243
```

Y podemos ver como se minimiza el costo mediante cada iteración:



iii) *Da una interpretación del criterio de corte utilizado en el algoritmo del gradiente. Explica si presenta alguna falla. Si no es una buena condición de corte, ¿puedes sugerir un criterio alternativo más eficaz?*

El algoritmo se detiene cuando el error es menor que un umbral definido por la tolerancia (tol) o cuando llega al máximo de iteraciones.

En cada iteración del algoritmo de descenso del gradiente, se calculan nuevos coeficientes. Luego, se compara el nuevo vector de coeficientes con el vector de coeficientes de la iteración anterior.

Si la distancia entre estos dos vectores es menor que el umbral de tolerancia (tol), el algoritmo se detiene, ya que esto indica que los coeficientes no están cambiando significativamente y se ha alcanzado una solución estable.

Posibles fallas:

- Si las características están en escalas muy diferentes, el cálculo de la distancia puede verse afectado.
- Si el algoritmo se encuentra en un mínimo local y no en el mínimo global, puede que los cambios en los coeficientes sean muy pequeños pero aún no óptimos.
- En algunos casos, el algoritmo podría no converger, y el criterio de corte puede llevar a un resultado prematuro (que alcance el máximos de iteraciones, pero la distancia no sea menor a la tolerancia).

Posible solución a algunos problemas:

Criterio de Tolerancia Basado en el Costo: En lugar de comparar los cambios en los coeficientes, se puede evaluar la variación en la función de costo (MSE) entre iteraciones:

- Se detiene el algoritmo si la disminución en el costo es menor que un umbral específico en un número determinado de iteraciones consecutivas. Esto evita la detención prematura basada en cambios insignificantes en los coeficientes.

Ver código completo en:

https://colab.research.google.com/drive/1eqqEQ3f1Ef1fFHWSuGFQxYsxpqXX_FrM?usp=sharing

Parte 3

Comportamiento del método de descenso por gradiente

c) Convergencia del método de descenso por gradiente. Explicar si el método siempre converge al mínimo de la función. En caso contrario, proporciona un contraejemplo para ilustrar este comportamiento.

El método de descenso por gradiente no siempre converge al mínimo de la función, esto se puede deber a varios factores como:

- **Mala elección de la tasa de aprendizaje:**

  Si la tasa de aprendizaje seleccionada es **muy grande**, el algoritmo puede "saltarse" el mínimo y oscilar o divergir.

- **Función no convenxa:**
  - Si la función tiene múltiples mínimos locales, el descenso por gradiente puede frenarse en un mínimo local en lugar de llegar al mínimo global. También puede quedar atrapado en un punto de silla (Como vimos en clase).

Como contraejemplo, planteamos la siguiente ecuación: f(x) = x ^ 3

Hicimos el siguiente script en Python para poder graficar la función y su descenso de gradiente:

```
import numpy as np
import matplotlib.pyplot as plt

# Definimos la función y su derivada
def f(x):
    return x**3

def df(x):
    return 3 * x**2

# Implementación el descenso por gradiente
def gradient_descent(starting_x, learning_rate, num_iterations):
    x = starting_x
    history = [x]  # Almacenar los valores de x para el gráfico
    for i in range(num_iterations):
        grad = df(x)  # Derivada en el punto actual
        x = x - learning_rate * grad  # Actualizar x usando el gr
        history.append(x)  # Guardar la posición actual para graf
    return np.array(history)

# Parámetros para el descenso por gradiente
starting_x = 2.0  # Punto de partida inicial
learning_rate = 0.01  # Tasa de aprendizaje
num_iterations = 50  # Número de iteraciones

# Ejecutar el descenso por gradiente
history = gradient_descent(starting_x, learning_rate, num_iterati

# Crear una gráfica para visualizar la evolución del descenso
x_vals = np.linspace(-3, 3, 500)
y_vals = f(x_vals)

plt.figure(figsize=(8, 6))

# Gráfica de la función
plt.plot(x_vals, y_vals, label='f(x) = x^3', color='blue')

# Graficar los puntos visitados por el descenso por gradiente
plt.scatter(history, f(history), color='red', label='Descenso por
```

```
# Etiquetas para el gráfico
plt.title("Descenso por Gradiente en $f(x) = x^3$")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()

plt.grid(True)
plt.show()
```

Gráfico obtenido: