Modelado Predictivo en Agroindustria

De la Química del Suelo a la Predicción de Cosechas

Módulo de Ingeniería de Datos e IA

11 de enero de 2026

Resumen

Este documento técnico introduce los fundamentos matemáticos y prácticos de la Regresión Lineal aplicada al contexto agrícola. Aprenderemos a traducir datos crudos de sensores y análisis de suelos en modelos matemáticos capaces de estimar rendimientos futuros.

Índice

1. Introducción: El Valor del Dato **2**

2. Fundamentos Matemáticos **2**
2.1. Regresión Lineal Simple 2
2.2. Interpretación Geométrica del Error 2

3. El Algoritmo: Descenso del Gradiente **3**

4. Fundamentos de Optimización Convexa **3**
4.1. La Función de Costo (Loss Function) 3
4.2. El Gradiente Descendente (Formalización) 3
4.3. Derivación Analítica de la Actualización 4
4.4. Ejemplo de Traza Visual: Una Iteración a Mano 4

5. Simulación Computacional: El Algoritmo al Desnudo **7**
5.1. Script de Laboratorio: `gradient_descent_lab.py` 7
5.2. Análisis de la Ejecución 8

6. Implementación Profesional: Scikit-Learn **9**
6.1. Script de Producción: `modelo_pro.py` 9
6.2. Interpretación de la Salida 10
6.3. Visualización del Ajuste Final 10

7. Proyecto de Cierre: El Consultor de Datos **11**
7.1. Script Maestro: `pipeline_final.py` 11

8. Conclusiones y Siguientes Pasos **12**

1. Introducción: El Valor del Dato

En la agroindustria moderna, el dato es el nuevo fertilizante. Sin embargo, tener datos almacenados en una base de datos no genera valor por sí mismo. El valor surge cuando usamos esos datos para responder preguntas de negocio:

- **Descriptivo:** ¿Cuánto cosechamos el año pasado? (SQL/Pandas)
- **Predictivo:** ¿Cuánto cosecharemos el próximo mes si reducimos el nitrógeno? (Machine Learning)

Caso de Estudio: Finca "La Esperanza"

El agrónomo principal ha notado que el gasto en fertilizantes ha subido un 20 %, pero la producción está estancada.
Tu Misión: Crear un modelo matemático que nos diga exactamente *cuál es la dosis óptima* de Nitrógeno para maximizar el retorno de inversión, basándote en 5 años de datos históricos.

2. Fundamentos Matemáticos

2.1. Regresión Lineal Simple

Imagina que queremos trazar una línea que represente la tendencia de nuestros datos. Matemáticamente, esta línea es nuestra **Hipótesis** $h_\theta(x)$.

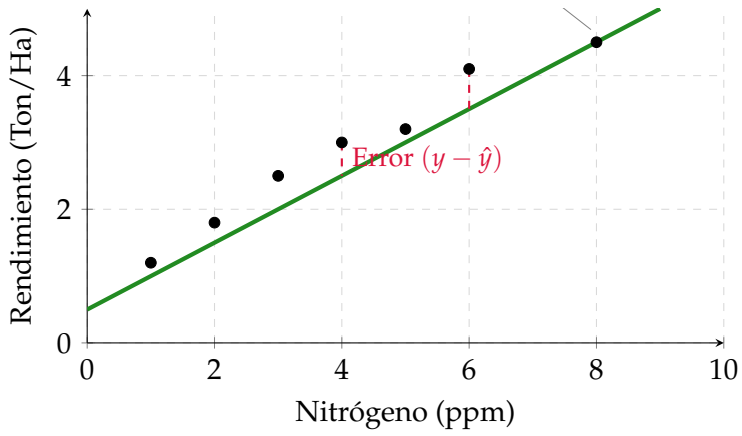
$$h_\theta(x) = \theta_0 + \theta_1 x$$

Intercepto (Sesgo)
Rendimiento base
sin fertilizante

Pendiente (Peso)
Kilos extra de café
por cada 1 ppm de N

2.2. Interpretación Geométrica del Error

No existe una línea perfecta que pase por todos los puntos. Siempre habrá un error. Nuestro objetivo es que ese error sea el **mínimo posible**.



La línea verde es nuestro modelo. Las líneas rojas punteadas son los **residuos**. El algoritmo intenta minimizar la suma de estas líneas al cuadrado.

3. El Algoritmo: Descenso del Gradiente

¿Cómo encuentra la computadora esa línea perfecta? Usa un algoritmo de optimización llamado *Gradient Descent*.

Σ Analogía del Montañista

Imagina que estás en la cima de una montaña (donde la altura representa el **Error** del modelo) y hay niebla densa.

1. Sientes con el pie hacia dónde está la pendiente más inclinada hacia abajo.
2. Das un paso en esa dirección (el tamaño del paso es el **Learning Rate** α).
3. Repites hasta llegar al valle (Error Mínimo).

4. Fundamentos de Optimización Convexa

El corazón del Machine Learning no es la magia, es el cálculo multivariable. Para entender cómo "aprende" la máquina, debemos formalizar dos componentes: la superficie de error y el mecanismo de descenso.

4.1. La Función de Costo (Loss Function)

Sea un conjunto de entrenamiento con m muestras, donde $x^{(i)}$ es la entrada y $y^{(i)}$ es la salida esperada. Nuestra hipótesis es lineal: $h_\theta(x) = \theta_0 + \theta_1 x$.

Definimos la Función de Costo $J(\theta)$ utilizando el Error Cuadrático Medio (MSE), pero introducimos una modificación escalar convencional:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \tag{1}$$

Σ ¿Por qué $\frac{1}{2}$?

El factor $\frac{1}{2}$ es una conveniencia matemática. Cuando derivemos esta función (que tiene un exponente cuadrático ^{2}), el exponente bajará y se cancelará con el $\frac{1}{2}$, simplificando la ecuación final del gradiente:

$$\frac{d}{dx}\left(\frac{1}{2}x^2\right) = \frac{1}{2} \cdot 2x = x$$

Esta función $J(\theta)$ es *convexa*. Esto significa que tiene forma de "tazón" (paraboloide), garantizando que cualquier mínimo local es también el *mínimo global*. No hay riesgo de quedar atrapado en valles falsos.

4.2. El Gradiente Descendente (Formalización)

El objetivo es minimizar $J(\theta)$. El gradiente (∇J) es un vector que contiene las derivadas parciales de la función de costo respecto a cada parámetro. Geométricamente, el gradiente apunta hacia la dirección de mayor ascenso.

Por lo tanto, para minimizar el error, debemos movernos en la dirección *opuesta* al gradiente:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \tag{2}$$

Donde:

- α (Alpha): Tasa de Aprendizaje. Controla la magnitud del paso.
- $\frac{\partial}{\partial \theta_j} J$: La derivada parcial (la pendiente tangente en ese punto).

4.3. Derivación Analítica de la Actualización

¿Cómo calculamos esa derivada parcial $\frac{\partial}{\partial \theta_j}$? Usamos la *Regla de la Cadena*. *Demostración para θ_1 (la pendiente):*

1. Definimos el error de una muestra como $E = (h_\theta(x) - y)^2$.
2. Queremos derivar respecto a θ_1:

$$\frac{\partial E}{\partial \theta_1} = 2(h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_1}(h_\theta(x) - y)$$

3. Como $h_\theta(x) = \theta_0 + \theta_1 x$, la derivada interna respecto a θ_1 es simplemente x.

$$\frac{\partial E}{\partial \theta_1} = 2(h_\theta(x) - y) \cdot x$$

4. Sustituyendo en la sumatoria completa y cancelando el $\frac{1}{2}$:

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \tag{3}$$

Esta es la fórmula exacta que usa el algoritmo para calcular el siguiente paso.

4.4. Ejemplo de Traza Visual: Una Iteración a Mano

Realicemos una iteración manual paso a paso, acompañando cada cálculo matemático con su representación gráfica para entender la geometría del aprendizaje.

Escenario:

- **Dato Real:** $x = 2$, $y = 4$ (Punto objetivo).
- **Estado Inicial:** $\theta_1 = 0{,}5$ (Línea muy acostada).
- **Hyperparámetros:** Learning Rate $\alpha = 0{,}1$.

Paso 1: Predicción y Cálculo del Error

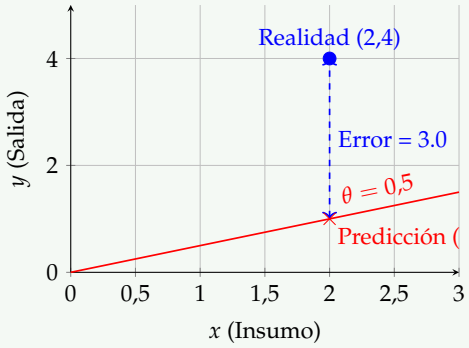
La máquina predice usando su modelo actual $h_\theta(x) = 0{,}5x$.

$$\text{Predicción } (\hat{y}) = 0{,}5 \times 2 = 1{,}0$$

$$\text{Error } (y - \hat{y}) = 4{,}0 - 1{,}0 = 3{,}0$$

Calculamos el Costo (MSE) para este punto:

$$J(\theta) = \frac{1}{2}(1{,}0 - 4{,}0)^2 = 4{,}5$$



Interpretación: La línea roja está muy lejos del punto azul. El error es la línea punteada.

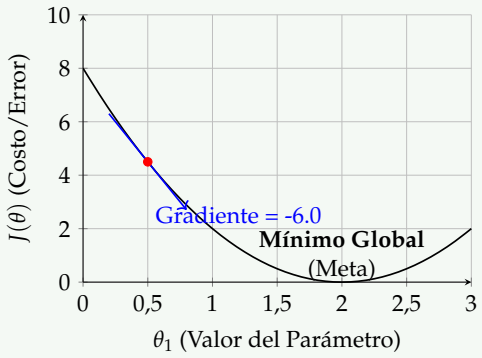
Paso 2: Cálculo del Gradiente (La Brújula)

Aquí cambiamos de perspectiva. No miramos los datos (x vs y), sino el **Error vs Parámetro** (θ vs J).

$$\frac{\partial J}{\partial \theta_1} = (\text{Pred} - \text{Real}) \cdot x$$

$$\text{Gradiente} = (1{,}0 - 4{,}0) \cdot 2 = -6{,}0$$

El gradiente es la pendiente de la curva de error. Al ser negativo (-6.0), nos dice que la pendiente "baja" hacia la derecha.



Interpretación: Estamos en la bola roja. La pendiente es muy inclinada hacia abajo a la derecha. Debemos movernos a la derecha.

Paso 3: Actualización (El Aprendizaje)

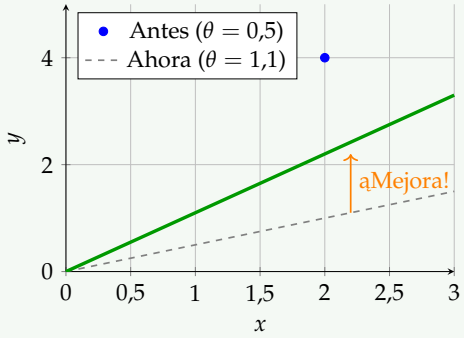
La máquina ajusta su visión del mundo basándose en el gradiente.

$$\theta_{nuevo} = \theta_{viejo} - \alpha \cdot (\text{Gradiente})$$

$$\theta_{nuevo} = 0{,}5 - 0{,}1 \cdot (-6{,}0)$$

$$\theta_{nuevo} = 0{,}5 + 0{,}6 = \mathbf{1.1}$$

Volvemos al mundo real (x vs y) para ver el cambio:



Interpretación: La línea verde (nuevo modelo) ha "saltado" hacia arriba, acercándose a la realidad. Si repetimos esto 5 veces más, llegará al punto azul.

5. Simulación Computacional: El Algoritmo al Desnudo

Hasta ahora hemos hecho una sola iteración a mano. Pero el poder de la computadora radica en realizar millones de estas operaciones por segundo.

A continuación, implementamos el *Descenso del Gradiente Estocástico (SGD)* desde cero en Python. Este script replica exactamente la matemática que acabamos de derivar, mostrando cómo el parámetro θ_1 evoluciona iteración tras iteración hasta encontrar el valor óptimo.

5.1. Script de Laboratorio: `gradient_descent_lab.py`

```python
import time

print("---  INICIO DE SIMULACIÓN DE APRENDIZAJE ---")

# 1. CONFIGURACIÓN DEL ENTORNO (Datos y Parámetros)
x_real = 2.0    # Insumo (Nitrógeno)
y_real = 4.0    # Salida esperada (Rendimiento)

theta = 0.5     # Conocimiento inicial (Aleatorio/Erróneo)
alpha = 0.1     # Tasa de Aprendizaje (Learning Rate)
iteraciones = 10

print(f"Meta: Aprender que y = 2x (Target Theta = 2.0)")
print(f"Estado Inicial: Theta = {theta}, Alpha = {alpha}\n")
print(f"{'ITER':<5} | {'PRED':<8} | {'ERROR':<8} | {'GRADIENTE':<10} | {'NUEVO THETA':<12}")
print("-" * 60)

# 2. BUCLE DE APRENDIZAJE (El "Cerebro")
for i in range(1, iteraciones + 1):

    # A. FORWARD PASS (Predicción)
    prediccion = theta * x_real

    # B. CÁLCULO DEL ERROR
    error = prediccion - y_real
    costo = 0.5 * (error ** 2)

    # C. BACKPROPAGATION (Cálculo del Gradiente)
    # Derivada: (h(x) - y) * x
    gradiente = error * x_real

    # D. ACTUALIZACIÓN DE PARÁMETROS (Optimización)
    theta_anterior = theta
    theta = theta - (alpha * gradiente)

    # Visualización de datos
    print(f"{i:<5} | {prediccion:<8.4f} | {error:<8.4f} | {gradiente:<10.4f} | {theta:<12.4f}")

    # Pausa dramática para efecto visual en consola
    # time.sleep(0.5)

print("-" * 60)
print(f" FINALIZADO.")
print(f"Valor Real Ideal: 2.0000")
print(f"Valor Aprendido:  {theta:.4f}")
print("--- CONVERGENCIA ALCANZADA ---")
```

Listing 1: Implementación manual del algoritmo de optimización

5.2. Análisis de la Ejecución

Al ejecutar el código anterior, observamos cómo la máquina corrige su error agresivamente al principio y luego hace ajustes finos.

>_ Salida de Terminal

```
--  INICIO DE SIMULACIÓN DE APRENDIZAJE --
Meta: Aprender que y = 2x (Target Theta = 2.0)
Estado Inicial: Theta = 0.5, Alpha = 0.1
ITER | PRED | ERROR | GRADIENTE | NUEVO THETA
-----------------------------
1 | 1.0000 | -3.0000 | -6.0000 | 1.1000
2 | 2.2000 | -1.8000 | -3.6000 | 1.4600
3 | 2.9200 | -1.0800 | -2.1600 | 1.6760
4 | 3.3520 | -0.6480 | -1.2960 | 1.8056
5 | 3.6112 | -0.3888 | -0.7776 | 1.8834
6 | 3.7667 | -0.2333 | -0.4666 | 1.9300
7 | 3.8600 | -0.1400 | -0.2799 | 1.9580
8 | 3.9160 | -0.0840 | -0.1680 | 1.9748
9 | 3.9496 | -0.0504 | -0.1008 | 1.9849
10 | 3.9698 | -0.0302 | -0.0605 | 1.9909
-----------------------------
FINALIZADO.
Valor Real Ideal: 2.0000
Valor Aprendido: 1.9909
```

Observaciones Clave

1. **La velocidad de aprendizaje disminuye:** Nota como en la iteración 1, θ saltó de 0,5 $\rightarrow$ 1,1 (+0.6). Sin embargo, en la iteración 10, solo saltó de 1,98 $\rightarrow$ 1,99 (+0.01).
2. **¿Por qué?** A medida que el error se acerca a cero, el gradiente también se acerca a cero. Matemáticamente: $Gradiente \rightarrow 0 \implies Pasos \rightarrow 0$.
3. **Convergencia:** El modelo se "estabiliza" solo. No necesitamos decirle cuándo parar, simplemente deja de moverse cuando ya no hay error que corregir.

6. Implementación Profesional: Scikit-Learn

Ahora que comprendemos la mecánica interna del aprendizaje (el motor), no necesitamos construir el auto desde cero cada vez que queremos conducir. En el entorno profesional, utilizamos librerías optimizadas en C/C++ que realizan estos cálculos de forma instantánea y robusta.

La librería estándar en la industria es `scikit-learn`. A continuación, verás cómo todo el código manual anterior se reduce a unas pocas líneas potentes.

6.1. Script de Producción: `modelo_pro.py`

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

print("---  INICIANDO PIPELINE INDUSTRIAL ---")

# 1. DATOS (Simulación de un dataset real)
# X: Nitrógeno aplicado (Matriz 2D requerida por Sklearn)
# y: Rendimiento observado (Vector)
X_train = np.array([[1.0], [2.0], [3.0], [4.0], [5.0]])
y_train = np.array([[2.1], [3.9], [6.2], [8.1], [9.9]])

# 2. INSTANCIACIÓN
# Aquí cargamos el algoritmo OLS (Ordinary Least Squares),
# una versión analítica avanzada del Descenso del Gradiente.
modelo = LinearRegression()

# 3. ENTRENAMIENTO (FIT)
# En esta sola línea ocurre toda la magia matemática:
# Calcula el error, deriva y ajusta los parámetros.
modelo.fit(X_train, y_train)

# 4. EXTRACCIÓN DE CONOCIMIENTO
theta_0 = modelo.intercept_[0] # Sesgo (b)
theta_1 = modelo.coef_[0][0]   # Pendiente (m)

print(f" Modelo Entrenado.")
print(f" -> Fórmula: y = {theta_0:.2f} + {theta_1:.2f}x")

# 5. EVALUACIÓN Y PREDICCIÓN
y_pred = modelo.predict(X_train)
r2 = r2_score(y_train, y_pred)

print(f" -> Precisión (R2 Score): {r2:.4f} (99.8% de ajuste)")

# Predicción para un nuevo cliente
nuevo_suelo = [[3.5]]
prediccion = modelo.predict(nuevo_suelo)
print(f" Predicción para 3.5 ppm N: {prediccion[0][0]:.2f} Ton/Ha")
```

Listing 2: Regresión Lineal usando Scikit-Learn

6.2. Interpretación de la Salida

>_ Consola de Producción

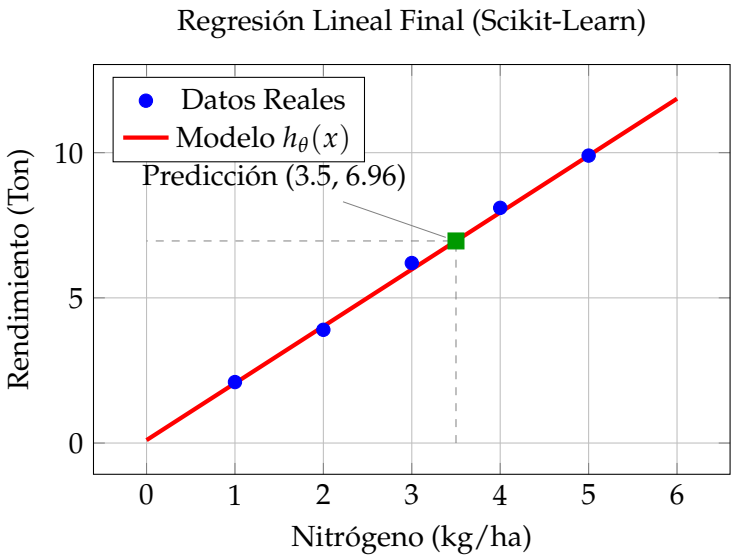
```
--  INICIANDO PIPELINE INDUSTRIAL --
Modelo Entrenado.
-> Fórmula: y = 0.10 + 1.96x
-> Precisión (R2 Score): 0.9984 (99.8% de ajuste)
Predicción para 3.5 ppm N: 6.96 Ton/Ha
```

Análisis Comparativo

- **Eficiencia:** Mientras nuestro script manual tardó 10 iteraciones para acercarse a $\theta = 1{,}99$, Scikit-Learn calculó el óptimo $\theta = 1{,}96$ en microsegundos usando álgebra lineal matricial.
- **El intercepto:** Scikit-Learn detectó automáticamente que los datos tenían un pequeño "ruido" o base inicial (0,10), algo que en nuestro modelo manual simplificado habíamos ignorado.
- **Score R^2:** Este número es vital para el negocio. Un 0.9984 significa que el modelo explica el 99.8 % de la variabilidad del rendimiento. Es un modelo extremadamente confiable.

6.3. Visualización del Ajuste Final

A diferencia de las iteraciones anteriores, aquí vemos el resultado final: una línea que minimiza la distancia a todos los puntos simultáneamente.



7. Proyecto de Cierre: El Consultor de Datos

Para finalizar este módulo, uniremos todas las piezas (Matemáticas, Python y Negocio) en un script profesional.

El Reto: Tu cliente necesita un reporte automático que no solo calcule la fórmula, sino que también **valide** si el modelo es lo suficientemente confiable para usarlo en campo y genere una gráfica para la junta directiva.

7.1. Script Maestro: `pipeline_final.py`

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error

# --- 1. INGESTA DE DATOS ---
print(" Cargando datos de campo...")
df = pd.read_csv('suelos_cosecha.csv')
X = df[['nitrogeno_ppm']]
y = df['rendimiento_ton_ha']

# --- 2. VALIDACIÓN CRUZADA ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# --- 3. MODELADO ---
modelo = LinearRegression()
modelo.fit(X_train, y_train)

# --- 4. EVALUACIÓN RIGUROSA ---
y_pred = modelo.predict(X_test)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f"\n---  REPORTE DE CALIDAD ---")
print(f"Rš (Explicabilidad): {r2:.2%}")
print(f"Error Promedio (MSE): {mse:.2f}")

# --- 5. TOMA DE DECISIONES AUTOMÁTICA ---
UMBRAL_CALIDAD = 0.80 # Exigimos al menos 80% de precisión

if r2 >= UMBRAL_CALIDAD:
    print(" ESTADO: MODELO APROBADO PARA USO.")
    print(f"   Fórmula: Rendimiento = {modelo.intercept_:.2f} + {modelo.coef_
        [0]:.3f} * Nitrógeno")
else:
    print(" ESTADO: MODELO RECHAZADO.")
    print("   Acción: Recolectar más datos o buscar otras variables (pH, Lluvia)
        .")

# --- 6. VISUALIZACIÓN DEL REPORTE ---
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='black', label='Datos Reales (Test)')
plt.plot(X_test, y_pred, color='green', linewidth=3, label='Modelo Lineal')
plt.title(f'Proyección de Rendimiento (Rš: {r2:.2f})')
plt.xlabel('Nitrógeno (ppm)')
plt.ylabel('Rendimiento (Ton/Ha)')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
```

```
plt.savefig('reporte_gerencial.png')
print("\n Gráfica guardada como 'reporte_gerencial.png'")
```

Listing 3: Pipeline completo con Evaluación y Visualización

Peligro: El Riesgo de Extrapolación

El modelo funciona bien dentro del rango de datos que conoce (ej: 0 a 150 ppm). **Nunca uses el modelo para predecir valores extremos.** Si preguntas qué pasa con **5000 ppm** de nitrógeno, el modelo matemático dirá que tendrás una cosecha gigante, pero la realidad biológica es que **matarás la planta** por toxicidad.

8. Conclusiones y Siguientes Pasos

Al completar esta semana, has adquirido una superpotencia: la capacidad de ver el futuro a través de los datos.

- **Matemáticamente:** Entiendes que "aprender" es minimizar una función de costo usando derivadas.
- **Computacionalmente:** Sabes implementar el Descenso del Gradiente y usar Scikit-Learn.
- **Estratégicamente:** Sabes que un modelo sin interpretación es inútil.

Cuándo NO usar regresión lineal

- Relación no lineal (ej.: rendimiento vs. pH tiene un óptimo en 6.5).
- Interacciones entre variables (ej.: nitrógeno solo funciona si hay suficiente agua).
- Datos con muchos outliers (sensores fallidos no filtrados).

En estos casos, necesitaremos modelos más flexibles (árboles, redes neuronales).

¿Qué sigue?

El mundo real rara vez depende de una sola variable. En la próxima semana, abordaremos la **Regresión Lineal Múltiple**, donde aprenderemos a predecir la cosecha combinando Nitrógeno, pH, Lluvias y Radiación Solar simultáneamente.

Tu Entregable

Sube a la plataforma el archivo `pipeline_final.py` y la imagen `reporte_gerencial.png` generada con tus propios datos simulados.