Linear Regression is a fundamental supervised learning algorithm used for **regression**—predicting a **continuous numerical value**—by modeling a straight-line relationship between one or more input features and the output.

It's based on fitting the "line of best fit" to the data, which minimizes the sum of squared errors between the predicted values and the actual observed values.

## 1. Simple Linear Regression (SLR)

SLR uses only **one** independent variable (X) to predict the dependent variable (Y).

### **Theory and Equation**

The relationship is defined by:

y^​=β0​+β1​x

* **y^​**: The predicted value of the dependent variable.
* **x**: The independent variable (feature).
* **β0​ (Intercept)**: The value of y^​ when x is 0.
* **β1​ (Slope/Coefficient)**: The change in y^​ for every one-unit change in x.

### **Practical Example: Predicting Sales from Advertising Spend** 📊

**Scenario:** A company tracks its television advertising budget and resulting sales figures to quantify the return on investment.

|  |  |
| --- | --- |
| Input (X) | Output (Y) |
| TV Ad Spend ($) | Sales Volume |

### **Python Example (Simple Linear Regression using scikit-learn)**

Python

import numpy as np

from sklearn.linear\_model import LinearRegression

# Training Data

# X: Ad Spend (in thousands of dollars)

X = np.array([10, 20, 30, 40, 50]).reshape(-1, 1) # Must be 2D for scikit-learn

# y: Sales (in thousands of units)

y = np.array([25, 45, 65, 80, 105])

# 1. Initialize and Train Model

slr\_model = LinearRegression()

slr\_model.fit(X, y)

# 2. Get the Coefficients

intercept = slr\_model.intercept\_

slope = slr\_model.coef\_[0]

print(f"Intercept (β₀): {intercept:.2f}")

print(f"Slope (β₁): {slope:.2f}")

# Interpretation: Sales = 5.00 + 1.95 \* Ad\_Spend

# A $1k increase in ad spend predicts a 1.95k increase in sales.

# 3. Make a Prediction

# Predict sales for a $60,000 ad spend (60 in units of 1k)

new\_ad\_spend = np.array([[60]])

predicted\_sales = slr\_model.predict(new\_ad\_spend)

print(f"Predicted Sales for $60k Ad Spend: {predicted\_sales[0]:.2f}k units")

## 2. Multiple Linear Regression (MLR)

MLR uses **two or more** independent variables (X1​,X2​,…) to predict the dependent variable (Y).

### **Theory and Equation**

The relationship is defined by:

y^​=β0​+β1​x1​+β2​x2​+⋯+βn​xn​

* **βi​**: The coefficient for each independent variable xi​, representing the change in y^​ for a one-unit change in xi​, **holding all other variables constant**.

### **Practical Example: Predicting House Price** 🏠

**Scenario:** Predicting the sale price of a house based on multiple features.

|  |  |
| --- | --- |
| Input (Xi​) | Output (Y) |
| X1​: Square Footage | House Price ($) |
| X2​: Number of Bedrooms |  |
| X3​: Age of House (Years) |  |

### **Python Example (Multiple Linear Regression using scikit-learn)**

Python

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# 1. Create a sample dataset (simulating a DataFrame)

data = {

    'SqFt': [1500, 2000, 1200, 2500, 1800],

    'Bedrooms': [3, 4, 2, 4, 3],

    'Age': [10, 5, 20, 2, 15],

    'Price': [300000, 450000, 200000, 550000, 350000] # Target variable

}

df = pd.DataFrame(data)

# Define Features (X) and Target (y)

X = df[['SqFt', 'Bedrooms', 'Age']]

y = df['Price']

# Split data (standard practice, even for small examples)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 2. Initialize and Train Model

mlr\_model = LinearRegression()

mlr\_model.fit(X\_train, y\_train)

# 3. Get the Coefficients and Intercept

print(f"Intercept (β₀): {mlr\_model.intercept\_:.2f}")

print(f"Coefficients (β₁...βₙ): {mlr\_model.coef\_}")

print(f"Features: {list(X.columns)}")

# 4. Make Predictions and Evaluate

y\_pred = mlr\_model.predict(X\_test)

# Since the dataset is tiny, interpretation is illustrative:

# Price = β₀ + (β₁ \* SqFt) + (β₂ \* Bedrooms) + (β₃ \* Age)