**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TP. HỒ CHÍ MINH**A white rectangular frame with black border

Description automatically generated

VIỆN ĐÀO TẠO CHẤT LƯỢNG CAO

----------------------

A close-up of a logo

Description automatically generated

**BÀI TẬP E-learning 5**

**Lớp học phần: Trí tuệ nhân tạo (010412103305)**

**Nhóm thực hiện: 3**

**SVTH: Quách Phú Thuận MSSV: 2251120446**

**SVTH: Lê Phạm Thanh Duy MSSV: 22H1120123**

**SVTH: Cao Bảo Gia Luật MSSV: 22H1120131**

*Thành phố Hồ Chí Minh, ngày 11 tháng 12 năm 202*

**MỤC LỤC**

[**Bài 1: Cho tập dữ liệu như hình. Biết thông số ban đầu: w = 0 , b = 0, alpha = 0.0001** 2](#_Toc216428590)

[Câu a. Hãy tính J(w,b) 2](#_Toc216428591)

[Câu b. Cập nhật w,b theo thuật toán Gradient Descend. Tính J(w\_update, b\_update) 15](#_Toc216428592)

[**Bài 2: Cho tập dữ liệu sau về mối quan hệ giữa thời gian tự học và kết quả đậu, rớt kì thi của sinh viên** 30](#_Toc216428593)

[**Link source code đầy đủ bài tập E-Learning 5: *Github*** 61](#_Toc216428594)

# **Bài 1: Cho tập dữ liệu như hình. Biết thông số ban đầu: w = 0 , b = 0, alpha = 0.0001**

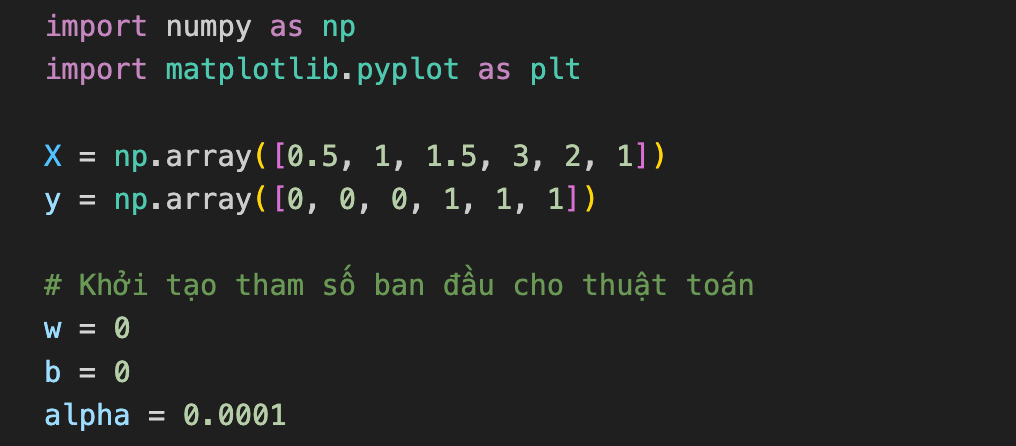
A table with numbers and letters

AI-generated content may be incorrect.

## **Câu a. Hãy tính J(w,b)**

***Bài làm***

1. **Import thư viện và khởi tạo dữ liệu**

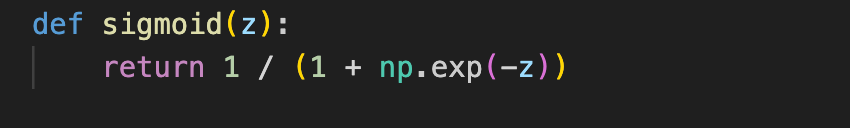


Đoạn code này thực hiện việc chuẩn bị môi trường và khởi tạo dữ liệu cho bài toán Logistic Regression. Đầu tiên, hai thư viện quan trọng được import: numpy là thư viện toán học mạnh mẽ hỗ trợ tính toán vector hóa (vectorization), giúp code chạy nhanh hơn nhiều so với vòng lặp thông thường - thay vì dùng vòng for để tính toán từng phần tử, numpy có thể thực hiện phép toán trên toàn bộ mảng cùng lúc; matplotlib.pyplot là thư viện vẽ đồ thị chuyên nghiệp trong Python, cho phép trực quan hóa dữ liệu và kết quả một cách trực quan, dễ hiểu.

Tiếp theo, dữ liệu được khởi tạo với biến X là mảng numpy chứa 6 giá trị đặc trưng (features) - đây là biến độc lập trong mô hình, có thể hiểu là các giá trị đầu vào để dự đoán. Biến y là mảng numpy chứa 6 nhãn (labels) tương ứng với từng giá trị trong X. Với bài toán phân loại nhị phân (binary classification), y chỉ nhận 2 giá trị: 0 thuộc lớp âm (negative class) và 1 thuộc lớp dương (positive class).

Về các tham số của mô hình, biến w (weight/trọng số) là hệ số góc của đường phân chia trong không gian đặc trưng - khởi tạo bằng 0 nghĩa là đường thẳng ban đầu nằm ngang, chưa có khả năng phân loại. Biến b (bias) là hệ số chặn (intercept), xác định vị trí của đường phân chia dịch chuyển lên/xuống - khởi tạo bằng 0 nghĩa là đường thẳng đi qua gốc tọa độ. Cuối cùng, biến alpha (learning rate/tốc độ học) là bước nhảy khi cập nhật tham số trong thuật toán Gradient Descent - giá trị 0.0001 khá nhỏ, giúp mô hình học chậm nhưng ổn định, tránh overshooting (nhảy quá xa khỏi điểm tối ưu).

1. **Hàm Sigmoid - Activation Function**



Hàm sigmoid (còn gọi là logistic function) là trái tim của Logistic Regression, đây là một hàm kích hoạt (activation function) có vai trò cực kỳ quan trọng. Hàm này hoạt động theo công thức toán học:

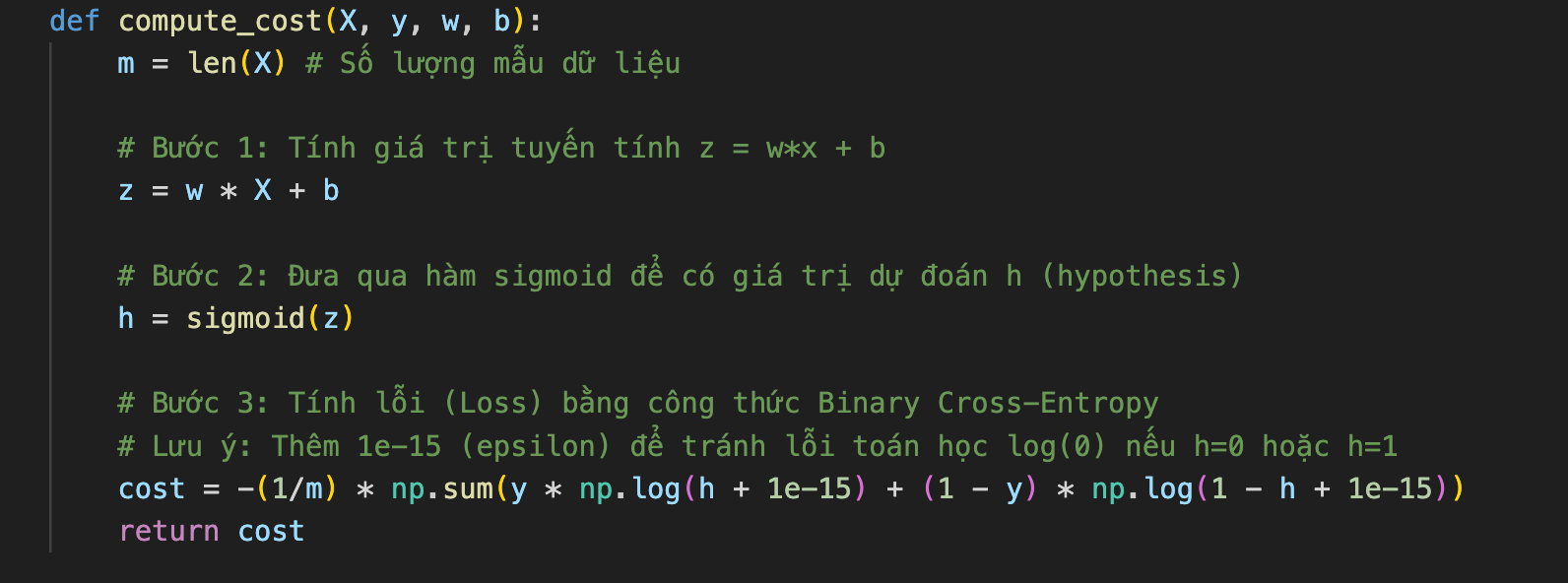
Trong đó z có thể là bất kỳ số thực nào từ âm vô cùng đến dương vô cùng, và hàm sigmoid sẽ “nén” (compress) giá trị này vào khoảng (0, 1).

Đặc điểm quan trọng nhất của sigmoid là khả năng diễn giải xác suất. Do giá trị đầu ra luôn nằm trong khoảng (0, 1), sigmoid hoàn hảo để biểu diễn xác suất: khi đầu ra gần 0 nghĩa là xác suất thuộc lớp 1 rất thấp (gần như chắc chắn thuộc lớp 0), khi đầu ra gần 0.5 thì không chắc chắn (xác suất thuộc lớp 0 và lớp 1 bằng nhau), và khi đầu ra gần 1 thì xác suất thuộc lớp 1 rất cao (gần như chắc chắn thuộc lớp 1).

Hàm sigmoid có hình dạng chữ S (S-curve) đặc trưng. Khi z tiến về âm vô cùng thì σ(z) tiến về 0, khi z = 0 thì σ(z) = 0.5 (điểm giữa), và khi z tiến về dương vô cùng thì σ(z) tiến về 1. Ví dụ cụ thể, sigmoid(0) = 0.5, sigmoid(5) ≈ 0.993 (gần 1), và sigmoid(-5) ≈ 0.007 (gần 0). Một tính chất quan trọng khác là đạo hàm của sigmoid có dạng:

Trong code, np.exp(-z) tính e^(-z) với e là số Euler (≈ 2.71828). Việc sử dụng numpy giúp tính toán vectorization, nghĩa là có thể truyền vào một mảng z và nhận về một mảng kết quả cùng lúc, giúp code chạy nhanh hơn rất nhiều.

1. **Hàm tính chi phí (Cost Function)**



Đây là hàm cốt lõi để đánh giá độ tốt của mô hình Logistic Regression, hàm này tính toán Binary Cross-Entropy Loss - một độ đo chuẩn cho bài toán phân loại nhị phân. Quá trình tính toán diễn ra qua ba bước chính.

*Bước đầu tiên* là tính giá trị tuyến tính theo công thức:

Đây là phương trình đường thẳng cơ bản trong không gian 1 chiều. Với X là mảng, phép toán này được vector hóa, nghĩa là mọi phần tử trong X đều được nhân với w và cộng với b cùng lúc. Ví dụ, nếu w=2, b=1, và X=[1, 2, 3] thì kết quả z=[3, 5, 7].

*Bước thứ hai* là áp dụng hàm sigmoid để biến đổi z thành xác suất dự đoán h (hypothesis). Giá trị h này đại diện cho:

Đây là xác suất để y=1 khi biết x với tham số w, b. Giá trị h luôn nằm trong khoảng (0, 1).

*Bước cuối cùng* là tính Binary Cross-Entropy theo công thức:

Công thức này hoạt động thông minh ở chỗ: khi y=1 (thực tế là lớp dương), phần đóng góp là:

Nếu h gần 1 (dự đoán đúng) thì cho chi phí thấp, nhưng nếu h gần 0 (dự đoán sai) thì cho chi phí rất cao. Tương tự, khi y=0 (thực tế là lớp âm), phần đóng góp là:

Nếu h gần 0 (dự đoán đúng) thì cho chi phí thấp, còn nếu h gần 1 (dự đoán sai) thì cho chi phí rất cao.

Trong code, epsilon (1e-15) là một giá trị cực nhỏ (0.000000000000001) được thêm vào để tránh lỗi toán học khi tính log(0). Khi h = 0 hoặc h = 1, log(0) không xác định (undefined), nhưng thêm epsilon đảm bảo log luôn tính được: log(0 + 1e-15) ≈ -34.5 (số âm lớn nhưng hữu hạn). Việc chia cho m (số mẫu) để lấy trung bình chi phí trên tất cả các mẫu giúp so sánh công bằng giữa các tập dữ liệu có kích thước khác nhau.

Một câu hỏi thường gặp là tại sao dùng Cross-Entropy thay vì Mean Squared Error (MSE). Lý do là MSE có dạng:

Công thức này tạo ra hàm không lồi (non-convex) khi kết hợp với sigmoid, dẫn đến nhiều local minimum và khó tối ưu. Trong khi đó, Cross-Entropy tạo ra hàm lồi (convex) có 1 global minimum duy nhất, giúp Gradient Descent hội tụ nhanh và ổn định hơn rất nhiều.

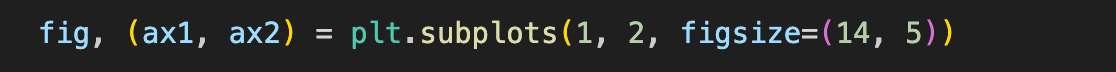
1. **Phần in kết quả và Visualization**

**4.1. Phần header và tính cost**



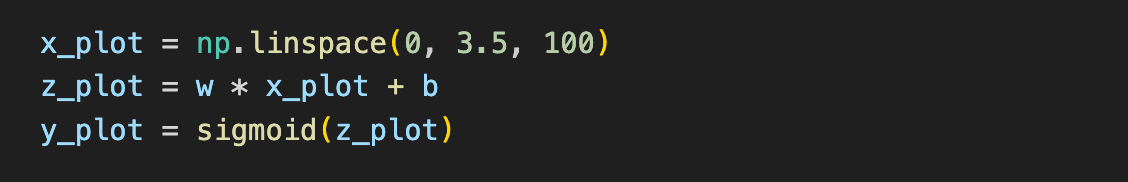
Phần code này bắt đầu bằng việc in ra tiêu đề và thông số ban đầu để người đọc dễ theo dõi quá trình thực thi. Sau đó, hàm compute\_cost(X, y, w, b) được gọi để tính giá trị hàm chi phí với tham số ban đầu w=0 và b=0. Kết quả được lưu vào biến J\_wb (viết tắt của J of w and b) để sử dụng trong các phần hiển thị tiếp theo. Giá trị này sẽ cho biết mô hình đang hoạt động tệ như thế nào trước khi được huấn luyện.

**4.2. Tạo Figure với 2 subplots**



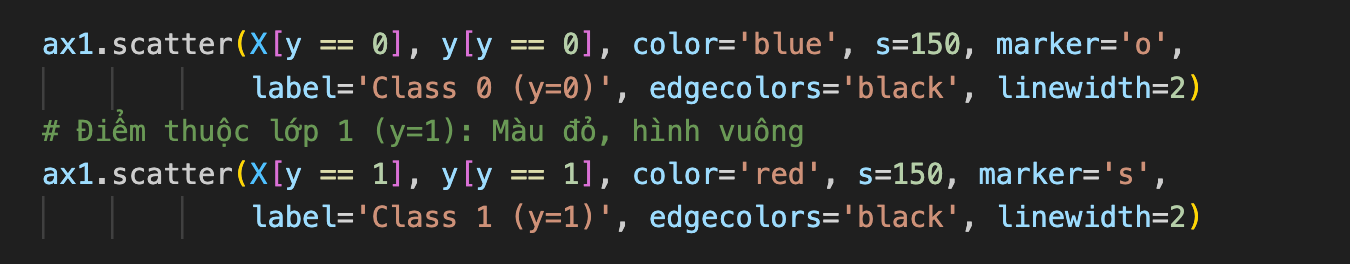
Một khung hình (figure) chứa 2 biểu đồ con (subplots) nằm ngang cạnh nhau được tạo ra, với cấu hình 1 hàng và 2 cột. Kích thước của figure được đặt là 14 inch chiều rộng và 5 inch chiều cao (figsize=(14, 5)) để đảm bảo các biểu đồ hiển thị rõ ràng và có đủ không gian. Biến ax1 đại diện cho biểu đồ bên trái sẽ hiển thị dữ liệu và đường sigmoid, trong khi ax2 là biểu đồ bên phải sẽ thể hiện bề mặt cost function trong không gian (w, b).

**4.3. Biểu Đồ 1: Dữ Liệu và sigmoid function**



Để vẽ được đường cong sigmoid mượt mà, cần tạo một dải giá trị x liên tục. Hàm np.linspace(0, 3.5, 100) tạo ra 100 điểm cách đều nhau từ 0 đến 3.5, đủ dày đặc để đường cong không bị góc cạnh hay gãy khúc. Sau đó, giá trị z tương ứng với từng điểm x được tính theo công thức:

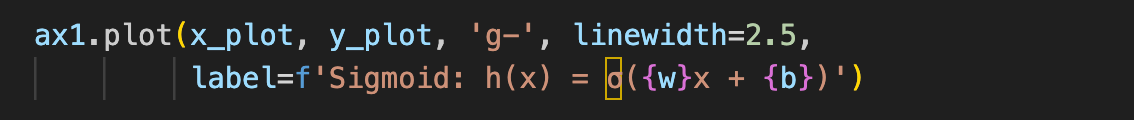
Rồi đưa qua hàm sigmoid để có được giá trị y, tạo thành đường cong hoàn chỉnh để vẽ lên biểu đồ.



Trong phần này, các điểm dữ liệu thực tế được vẽ lên biểu đồ để trực quan hóa dữ liệu.

Với các điểm thuộc lớp 0 (y=0), cú pháp X[y == 0] được sử dụng để lọc ra những điểm có nhãn y bằng 0, sau đó vẽ chúng bằng màu xanh dương (blue) với hình tròn (marker=‘o’). Kích thước điểm được đặt là 150 (s=150) để dễ nhìn thấy, và có viền màu đen (edgecolors='black') với độ dày 2 (linewidth=2) để tạo sự phân biệt rõ ràng.

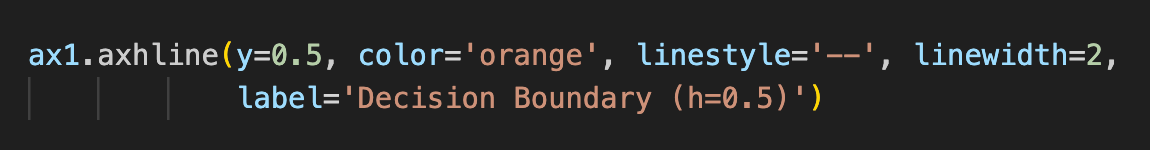
Tương tự, với các điểm thuộc lớp 1 (y=1), X[y == 1] được sử dụng để lọc ra những điểm có nhãn y bằng 1, và vẽ chúng bằng màu đỏ (red) với hình vuông (marker=‘s’). Việc sử dụng hai màu khác nhau (xanh cho lớp 0, đỏ cho lớp 1) và hai hình dạng khác nhau (tròn và vuông) giúp người xem dễ dàng phân biệt hai lớp dữ liệu ngay cả khi xem trên ảnh đen trắng hoặc khi có vấn đề về màu sắc.



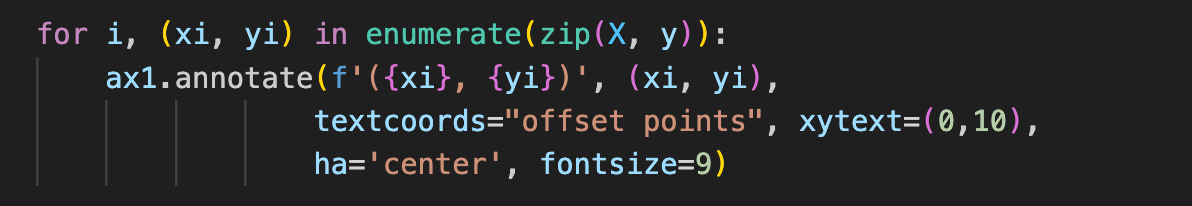
Sau khi vẽ các điểm dữ liệu, đường cong sigmoid thể hiện mô hình dự đoán hiện tại được vẽ. Đường này được vẽ bằng màu xanh lá cây (green) với kiểu đường liền ('g-') và độ dày 2.5 (linewidth=2.5) để nổi bật trên biểu đồ. Nhãn (label) của đường hiển thị công thức toán học:

Với giá trị w và b hiện tại. Điều đặc biệt là với w=0 và b=0 như trong trường hợp ban đầu, công thức trở thành:

Cho mọi giá trị x, dẫn đến đường sigmoid sẽ là một đường thẳng ngang tại y=0.5 bởi vì sigmoid(0) luôn bằng 0.5 bất kể x là bao nhiêu.



* axhline: Vẽ đường ngang (horizontal line)
* **Decision Boundary** tại h=0.5: Ngưỡng phân loại
  + Nếu h ≥ 0.5 → dự đoán y=1
  + Nếu h < 0.5 → dự đoán y=0
* Đường này giúp thấy rõ mô hình đang phân loại các điểm như thế nào

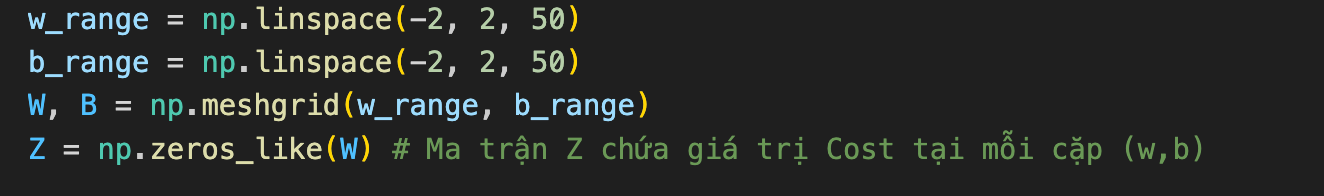


Phần code này duyệt qua từng cặp (xi, yi) trong dữ liệu và gắn nhãn văn bản lên biểu đồ bằng hàm annotate. Tham số xytext=(0,10) xác định vị trí của text cách điểm dữ liệu 10 pixels về phía trên, còn ha='center' căn giữa text theo chiều ngang. Việc hiển thị tọa độ trực tiếp trên biểu đồ giúp người xem dễ dàng đọc giá trị chính xác của từng điểm mà không cần ước lượng từ các trục.

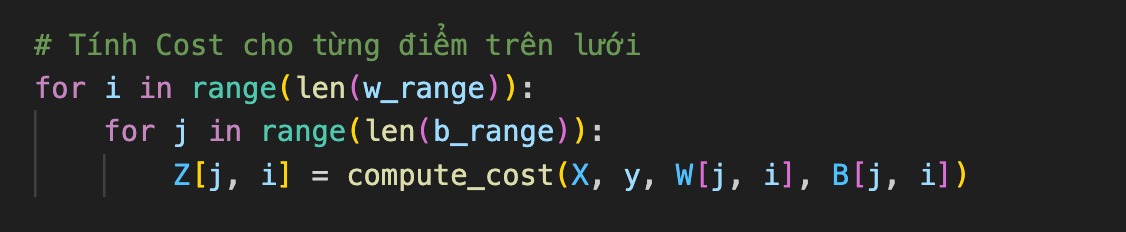


Phần trang trí này thiết lập các thuộc tính hiển thị cho biểu đồ. Nhãn trục x và y được đặt với font đậm (bold) và kích thước 13 để dễ đọc. Tiêu đề biểu đồ hiển thị công thức và giá trị J(w,b) với 8 chữ số thập phân, cung cấp thông tin chi tiết về hàm chi phí tại tham số hiện tại. Hàm legend hiển thị chú thích cho các đường và điểm, với loc='best' cho phép matplotlib tự động tìm vị trí tối ưu không bị che khuất dữ liệu. Lưới nền được bật với độ trong suốt 0.3 và kiểu đường gạch ngang, giúp người xem dễ ước lượng giá trị mà không làm loãng biểu đồ. Cuối cùng, set\_ylim và set\_xlim giới hạn phạm vi các trục để biểu đồ thoáng đãng, không bị sát mép khung hình.

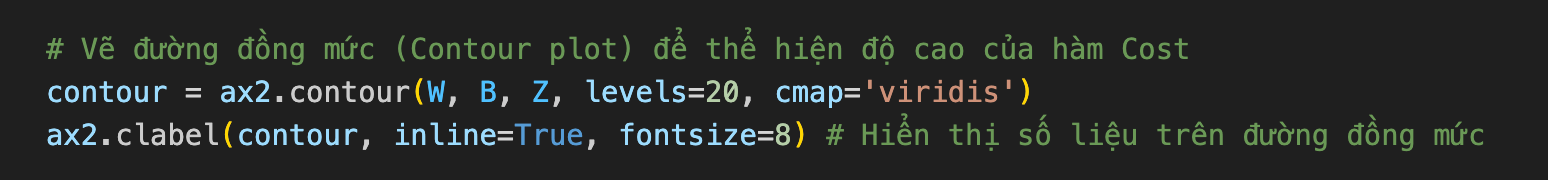
**4.4. Biểu Đồ 2: Mặt phẳng Cost Function**



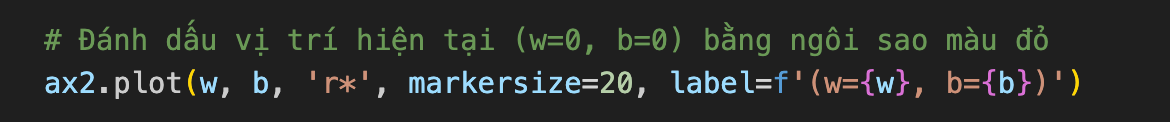
Để vẽ bề mặt Cost function trong không gian 2 chiều (w, b), trước tiên cần tạo lưới tọa độ (mesh grid). Hàm np.linspace(-2, 2, 50) tạo ra 50 giá trị cách đều nhau từ -2 đến 2 cho cả w và b, tạo thành dải giá trị khảo sát. Sau đó, hàm meshgrid kết hợp hai dải giá trị này thành lưới 2D, trong đó mỗi điểm trên lưới đại diện cho một cặp tham số (w, b) cụ thể. Ma trận Z được khởi tạo với kích thước 50×50 bằng zeros\_like(W) để lưu giá trị Cost tại từng điểm (w, b) trên lưới, tạo thành “bản đồ địa hình” của hàm chi phí.



Vòng lặp lồng nhau này duyệt qua tất cả 2500 cặp (w, b) trên lưới (50×50), tính giá trị Cost tại mỗi điểm bằng hàm compute\_cost, và lưu kết quả vào ma trận Z tại vị trí tương ứng. Quá trình này tạo ra “bản đồ địa hình” hoàn chỉnh của hàm Cost trong không gian tham số, trong đó các vùng trũng (giá trị thấp) đại diện cho các điểm tối ưu, còn các vùng cao (giá trị lớn) đại diện cho các tham số kém hiệu quả.



Hàm contour vẽ các đường đồng mức (contour lines) tương tự như đường bình độ trên bản đồ địa lý, trong đó mỗi đường nối các điểm có cùng giá trị Cost. Tham số levels=20 chỉ định vẽ 20 đường mức khác nhau để thể hiện sự thay đổi của Cost một cách chi tiết. Bảng màu viridis được chọn với gradient từ tím đậm (giá trị Cost cao) qua xanh lá sang vàng (giá trị Cost thấp), giúp người xem dễ dàng nhận biết các vùng tối ưu. Hàm clabel với inline=True hiển thị giá trị số trực tiếp trên các đường đồng mức với cỡ chữ 8, cung cấp thông tin định lượng chính xác về giá trị Cost tại từng vùng.



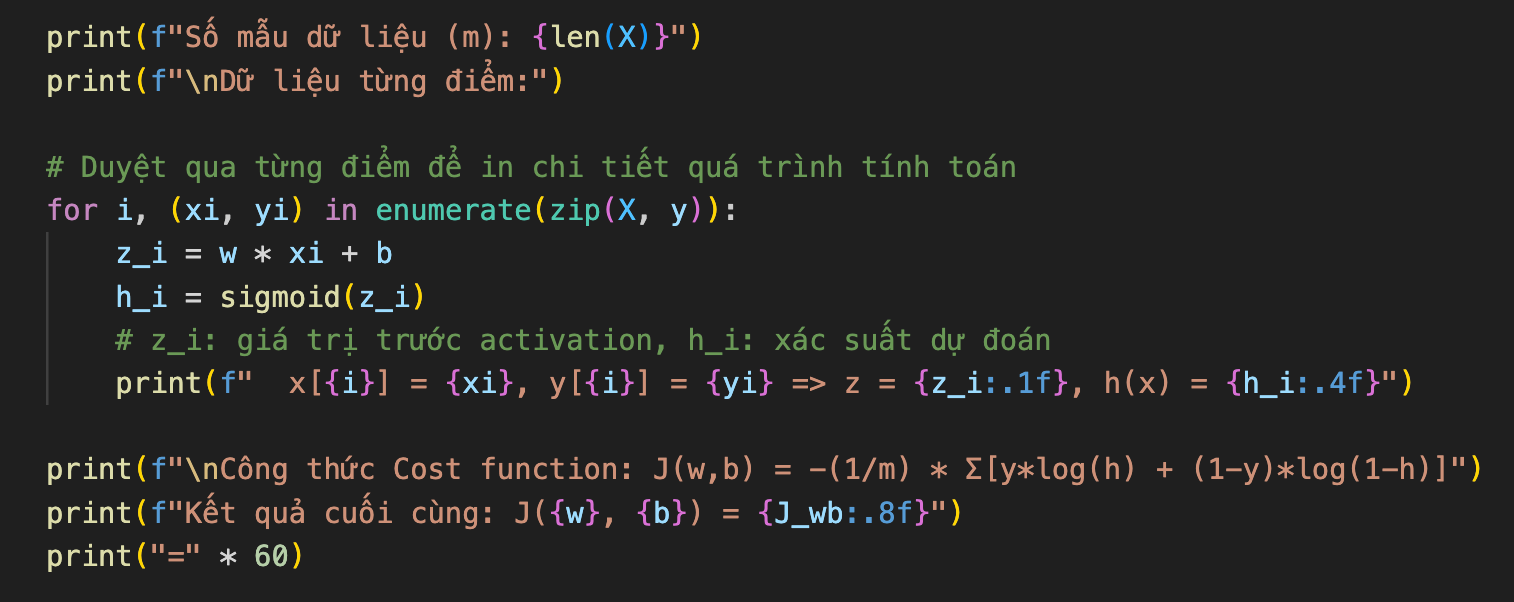
Lệnh này vẽ một ngôi sao đỏ kích thước lớn (markersize=20) tại vị trí (w=0, b=0) trên biểu đồ contour, đánh dấu rõ ràng điểm khởi đầu của tham số. Việc hiển thị vị trí hiện tại trên “bản đồ địa hình” Cost giúp người xem dễ dàng so sánh với vùng tối ưu (vùng trũng nhất có Cost thấp), từ đó hiểu được mô hình cần di chuyển theo hướng nào để cải thiện hiệu suất.

**4.5. Lưu và hiển thị**

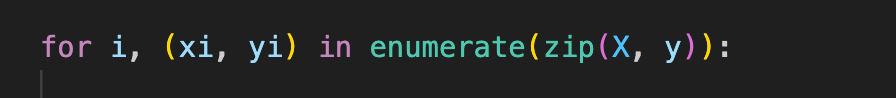


* tight\_layout(): Tự động điều chỉnh khoảng cách giữa các subplot để không bị chồng lấn
* savefig: Lưu hình với độ phân giải cao (300 DPI - chất lượng in ấn)
* bbox\_inches='tight': Cắt bỏ khoảng trắng thừa xung quanh
* show(): Hiển thị biểu đồ lên màn hình

**4.6. In chi tiết quá trình tính toán**



Phần cuối của chương trình in ra chi tiết quá trình tính toán để người đọc hiểu rõ từng bước. Đầu tiên, số lượng mẫu dữ liệu (m = 6) được in ra để biết quy mô tập dữ liệu. Tiếp theo, vòng lặp được sử dụng để duyệt qua từng cặp (xi, yi) trong dữ liệu:



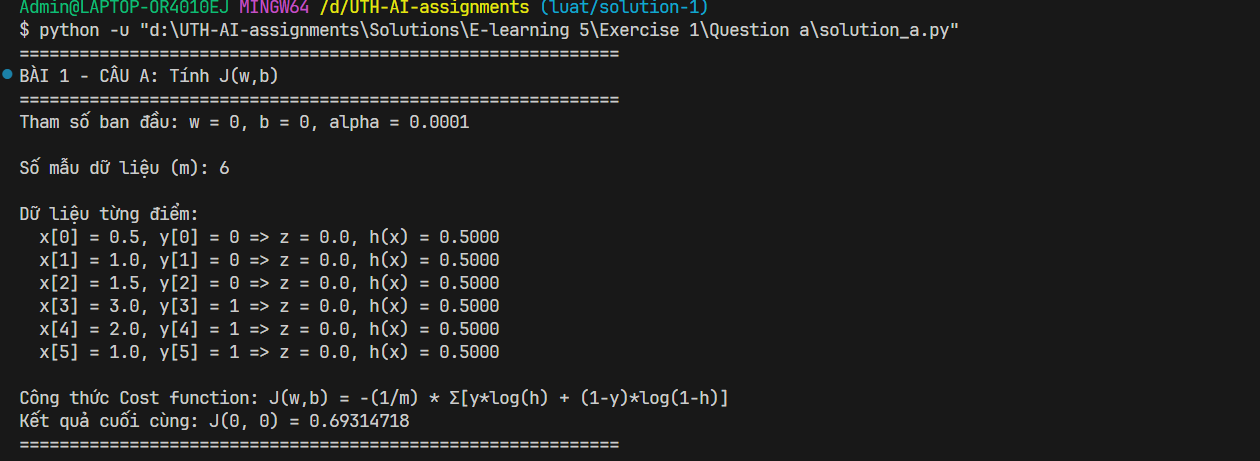
Hàm enumerate giúp có thêm chỉ số i để đánh số các điểm.

Với mỗi điểm dữ liệu, giá trị tuyến tính z\_i = w \* xi + b (giá trị trước khi qua activation function) và h\_i = sigmoid(z\_i) (xác suất dự đoán sau khi qua sigmoid) được tính. Kết quả được in ra với .1f cho z (1 chữ số thập phân) và .4f cho h (4 chữ số thập phân) giúp người đọc thấy rõ quá trình biến đổi từ dữ liệu đầu vào sang xác suất dự đoán.

Cuối cùng, công thức Cost function được hiển thị dưới dạng ký hiệu toán học:

Để người đọc hiểu rõ phương pháp tính, rồi in kết quả cuối cùng J(0, 0) với 8 chữ số thập phân để đảm bảo độ chính xác cao.

1. **Output và kết quả**
   1. **Console Output**



* 1. **Phân Tích Kết Quả**

**5.2.1. Giá trị J(0, 0) = 0.69314718**

Đây là giá trị hàm chi phí khi mô hình chưa được huấn luyện với w=0 và b=0. Giá trị này thể hiện mô hình đang dự đoán xác suất 0.5 cho mọi điểm, tức là dự đoán hoàn toàn ngẫu nhiên (50-50). Điều thú vị là J(0,0) gần bằng ln(2) ≈ 0.693147, và đây không phải trùng hợp ngẫu nhiên.

**Giải thích toán học:**

Khi h = 0.5 cho mọi điểm, hàm chi phí được tính như sau:

Kết quả này đại diện cho chi phí tối đa của một mô hình phân loại nhị phân khi dự đoán hoàn toàn ngẫu nhiên (50-50), phản ánh việc mô hình chưa học được bất kỳ thông tin hữu ích nào từ dữ liệu.

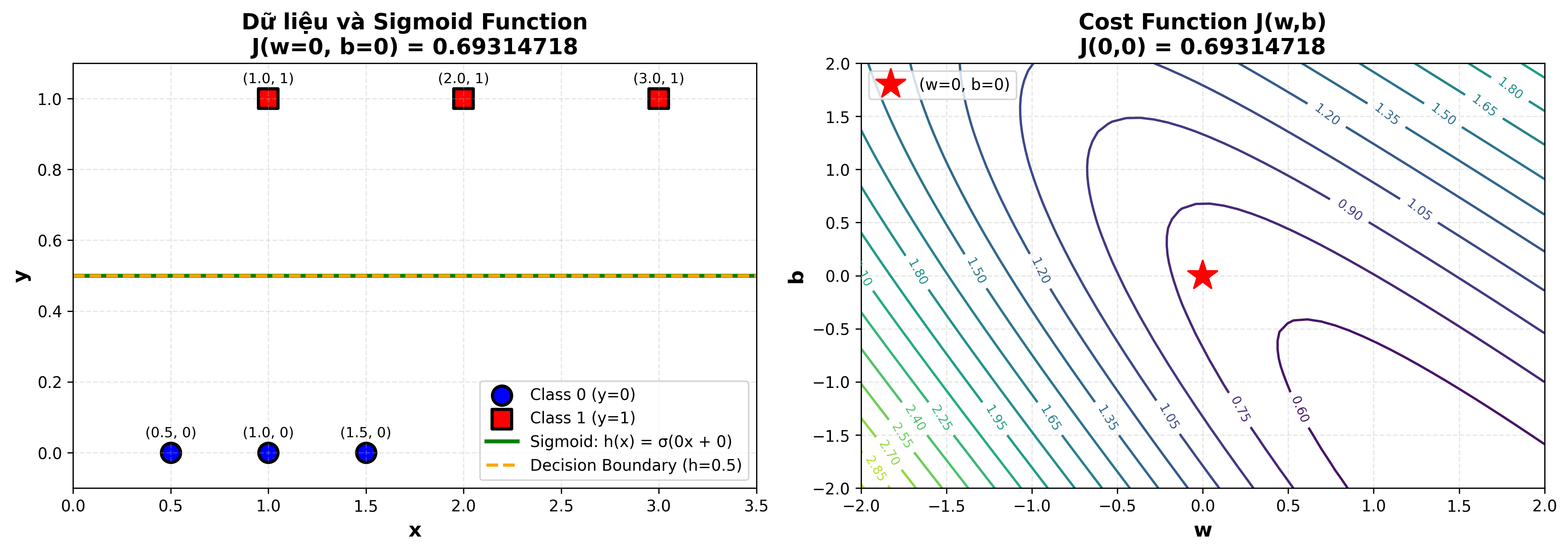
**5.2.2. Tại sao h(x) = 0.5 cho mọi x?**

Với tham số ban đầu w=0 và b=0, giá trị tuyến tính z luôn bằng 0 bất kể giá trị x là bao nhiêu:

Khi đưa qua hàm sigmoid:

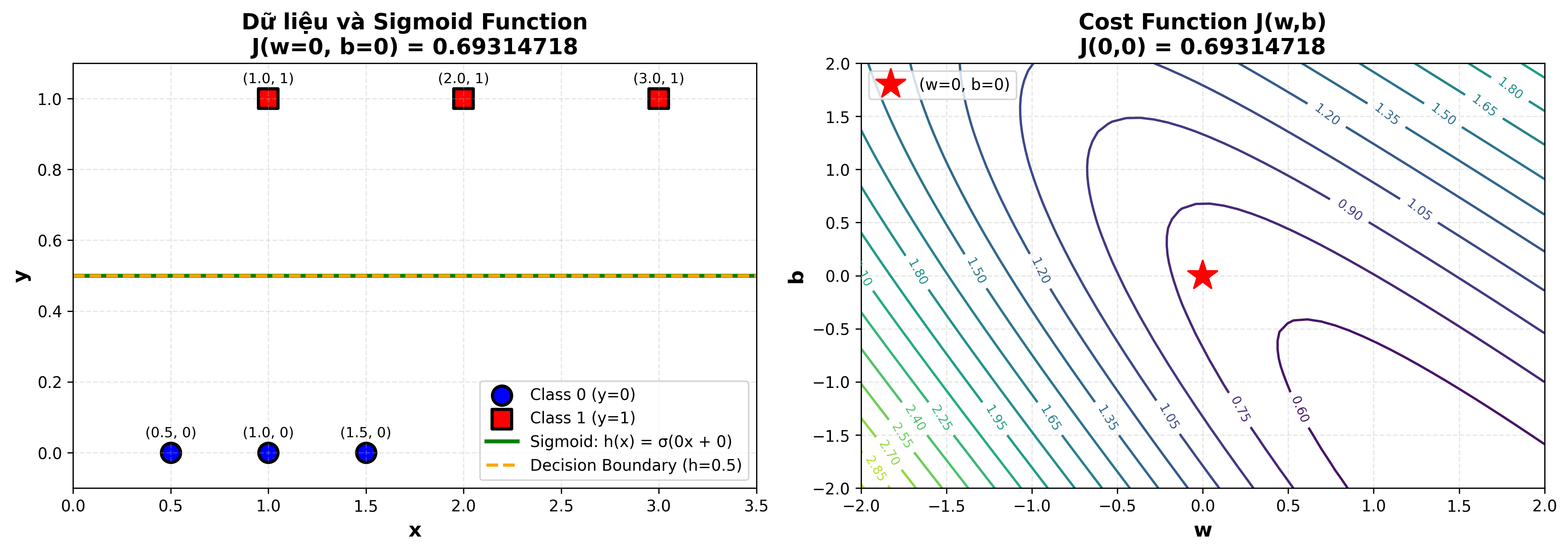
Kết quả này dẫn đến đường sigmoid là một đường thẳng ngang tại y=0.5, hoàn toàn không có khả năng phân loại vì mọi điểm dữ liệu đều nhận được xác suất giống nhau.

**5.2.3. Biểu đồ 1: Dữ liệu và Sigmoid**



Biểu đồ bên trái thể hiện sự phân bố dữ liệu và đường dự đoán của mô hình. Các điểm xanh (y=0) tập trung ở phía trái với giá trị x nhỏ, trong khi các điểm đỏ (y=1) nằm ở phía phải với giá trị x lớn hơn. Đường sigmoid màu xanh lá hiện tại là một đường ngang tại mức 0.5 do w=0 và b=0, trùng với đường quyết định (decision boundary) màu cam gạch ngang. Nhận xét quan trọng là mô hình hiện tại hoàn toàn không phân loại được gì vì tất cả điểm đều nhận xác suất dự đoán giống nhau là 0.5, không phản ánh sự khác biệt giữa hai lớp dữ liệu.

**5.2.4. Biểu đồ 2: Cost Function Surface**



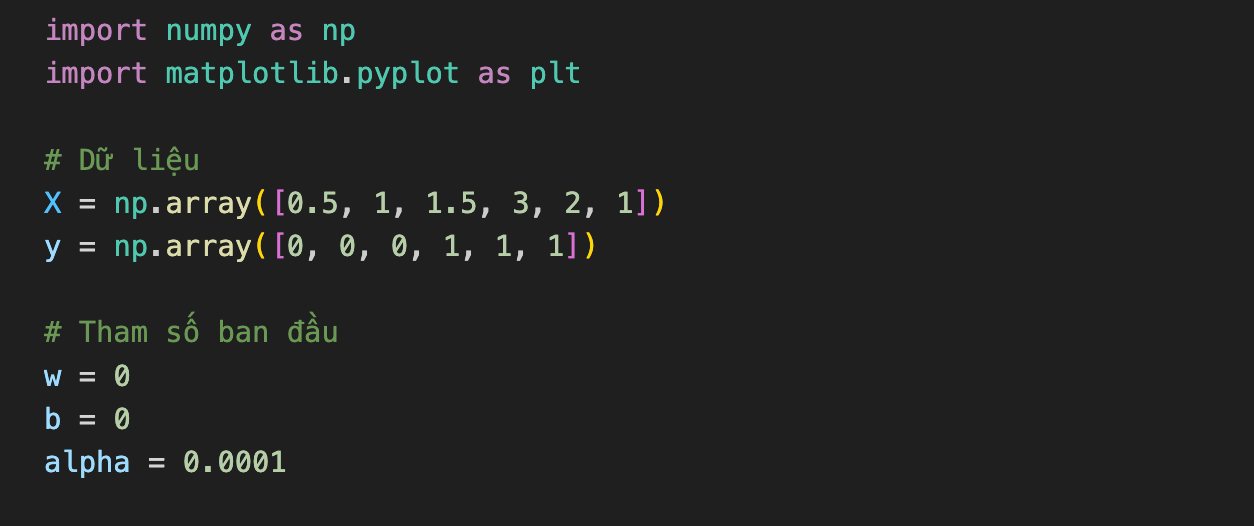
Biểu đồ bên phải thể hiện “bản đồ địa hình” của hàm Cost trong không gian (w, b). Các vùng màu tím đậm thể hiện những khu vực có giá trị Cost cao, tương ứng với mô hình hoạt động kém, trong khi các vùng màu vàng biểu thị những vùng có Cost thấp với mô hình tốt hơn. Ngôi sao đỏ tại toạ độ (0,0) đánh dấu vị trí ban đầu của tham số, còn các vùng trũng (valley) chỉ hướng đến điểm tối ưu.

Quan sát trên biểu đồ cho thấy điểm (0,0) nằm ở vùng có Cost xấp xỉ 0.693, không phải tệ nhất nhưng cũng không tốt. Biểu đồ thể hiện rõ ràng một vùng trũng hướng về phía w dương và b âm, đây chính là hướng mà thuật toán Gradient Descent sẽ di chuyển để giảm Cost và tìm điểm tối ưu.

## **Câu b. Cập nhật w,b theo thuật toán Gradient Descend. Tính J(w\_update, b\_update)**

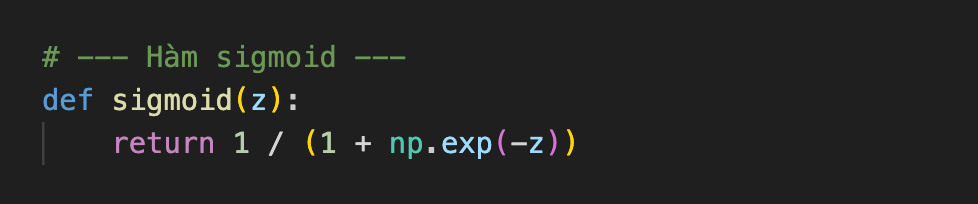
***Bài làm***

1. **Import và khởi tạo**



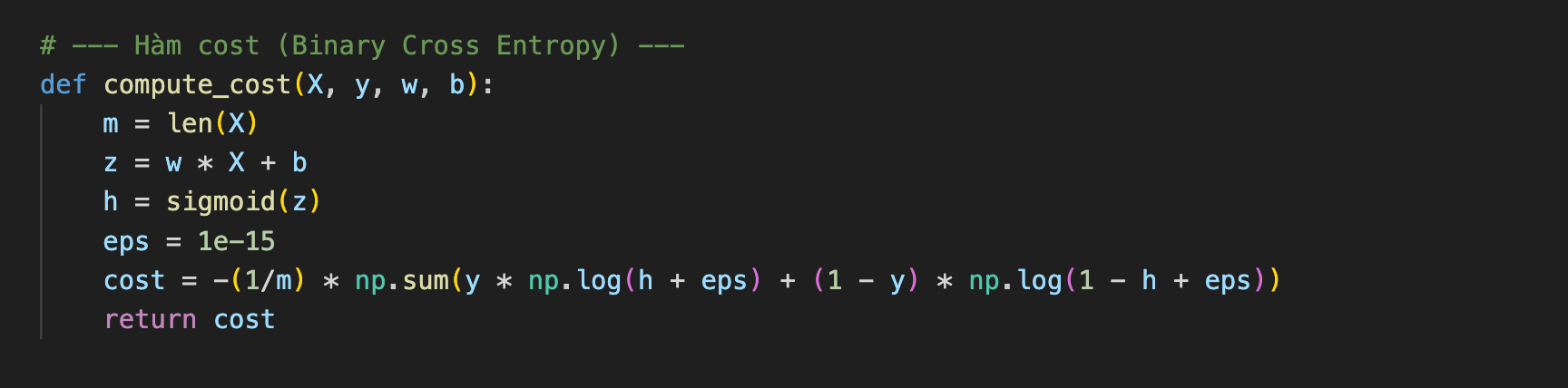
Phần khởi tạo này giống với Question A, thiết lập môi trường và dữ liệu cho bài toán. Dữ liệu X và y chứa 6 điểm dữ liệu cho bài toán phân loại nhị phân, trong đó X là các giá trị đặc trưng và y là các nhãn tương ứng (0 hoặc 1). Tham số ban đầu w=0 và b=0 thể hiện mô hình chưa học được thông tin gì, còn learning rate alpha=0.0001 là một bước nhảy rất nhỏ, đảm bảo mô hình học từ từ để tránh overshooting (nhảy quá xa khỏi điểm tối ưu).

1. **Các hàm cơ bản**



Hàm kích hoạt sigmoid giống với Question A, được sử dụng để chuyển đổi giá trị tuyến tính thành xác suất. Công thức toán học của sigmoid là:

Hàm này chuyển đổi bất kỳ giá trị z nào thành xác suất trong khoảng (0, 1), là thành phần cốt lõi của Logistic Regression cho phép biểu diễn dự đoán dưới dạng xác suất.

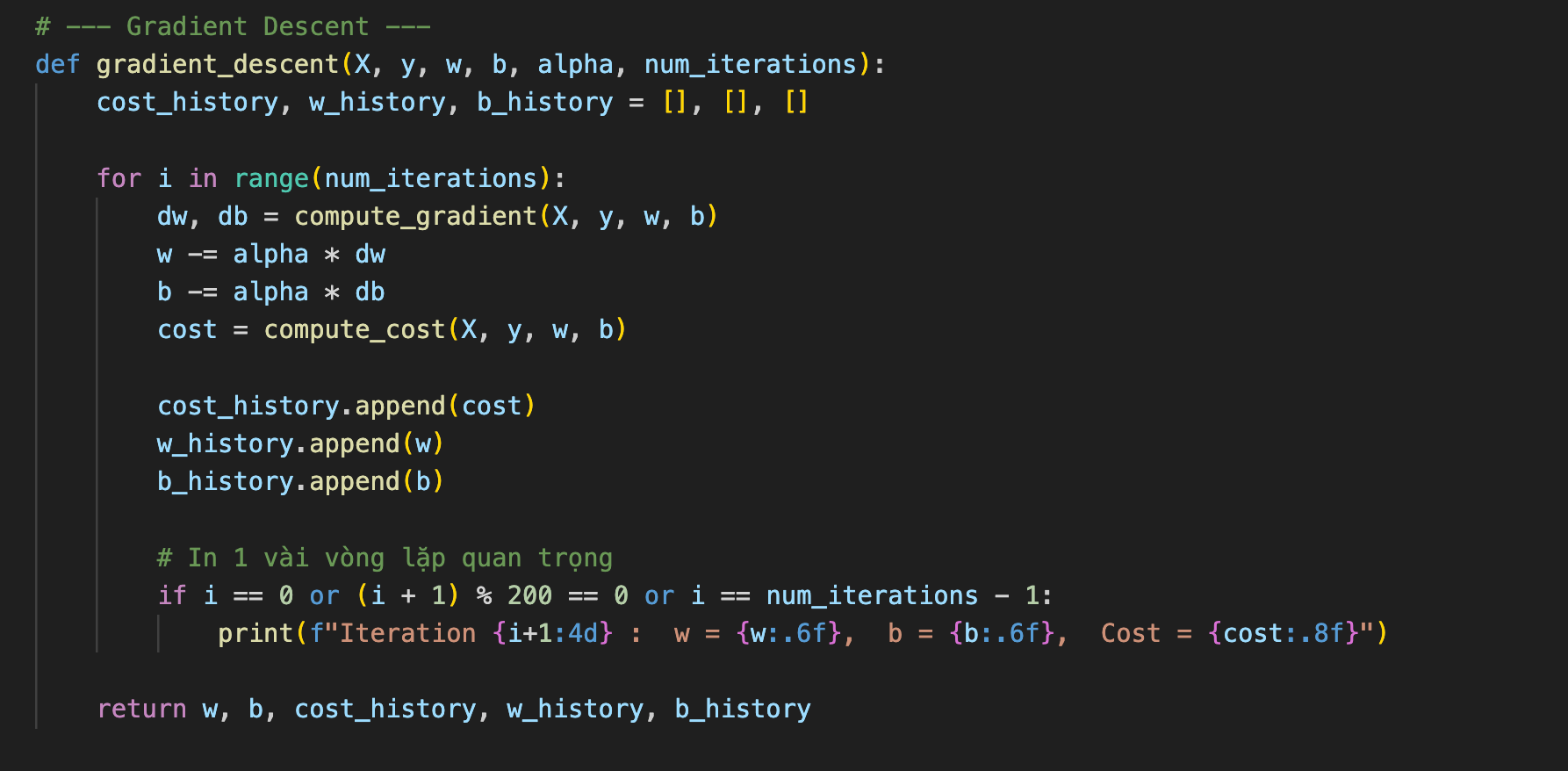


Hàm này tính chi phí để đánh giá chất lượng mô hình qua ba bước chính. Bước đầu tiên tính giá trị tuyến tính theo công thức:

Tiếp theo tính xác suất dự đoán bằng cách đưa z qua sigmoid. Cuối cùng tính giá trị Cost bằng Binary Cross-Entropy:

Tham số eps = 1e-15 được thêm vào để tránh lỗi toán học khi tính log(0). Mục tiêu của thuật toán là minimize (giảm thiểu) giá trị J(w,b) này.

1. **Hàm tính gradient**



Đây là hàm **quan trọng nhất** - tính toán **gradient** (đạo hàm) của Cost function theo w và b.

#### **Bước 1: Tính z và h**

z = w \* X + b  
h = sigmoid(z)

Đây là bước tính giá trị tuyến tính và xác suất dự đoán, tương tự như trong hàm compute\_cost. Giá trị z được tính theo công thức z = w\*X + b, sau đó được chuyển đổi thành xác suất h qua hàm sigmoid.

#### **Bước 2: Tính Error**

error = h - y

**Ý nghĩa:** Biến error đại diện cho sai số giữa giá trị dự đoán (h) và giá trị thực tế (y). Nếu error > 0 nghĩa là mô hình dự đoán cao hơn thực tế (overestimate). Ngược lại, nếu error < 0 thì mô hình dự đoán thấp hơn thực tế (underestimate). Khi error = 0 nghĩa là dự đoán hoàn toàn chính xác.

**Ví dụ:** Với h = 0.8 và y = 1, ta có error = -0.2, nghĩa là dự đoán hơi thấp. Với h = 0.3 và y = 0, ta có error = 0.3, nghĩa là dự đoán hơi cao.

#### **Bước 3: Tính Gradient của w**



**Công thức toán học:**

**Giải thích:** Biến dw là đạo hàm riêng của hàm Cost function theo tham số w, cho biết Cost thay đổi như thế nào khi w thay đổi. Biểu thức error \* X tạo ra sai số có trọng số (weighted error). Nếu giá trị xi lớn và error lớn thì gradient sẽ lớn, nghĩa là cần điều chỉnh w nhiều. Ngược lại, nếu xi nhỏ hoặc error nhỏ thì gradient nhỏ, chỉ cần điều chỉnh w ít.

**Ý nghĩa hình học:** Khi dw > 0, Cost tăng khi w tăng, do đó cần giảm w để giảm Cost. Khi dw < 0, Cost tăng khi w giảm, do đó cần tăng w. Khi dw ≈ 0 nghĩa là mô hình đang ở gần điểm tối ưu.

#### **Bước 4: Tính Gradient của b**



**Công thức toán học:**

**Giải thích:** Biến db là đạo hàm riêng của hàm Cost function theo tham số b. Đây là tổng các sai số không nhân với X vì đạo hàm của b trong biểu thức z = wx + b là 1. Khi db > 0 nghĩa là dự đoán trung bình cao hơn thực tế, do đó cần giảm b để hạ thấp dự đoán xuống. Ngược lại, khi db < 0 nghĩa là dự đoán trung bình thấp hơn thực tế, cần tăng b để nâng cao dự đoán.

#### **Tại sao công thức này đúng?**

**Chứng minh toán học** (simplified):

Từ Cost function:

Đạo hàm theo w (chain rule):

Trong đó:

* (tính chất đẹp của sigmoid)

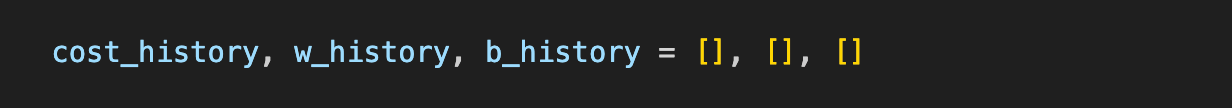
Kết hợp lại:

Trung bình trên m mẫu:

1. **Thuật toán Gradient Descent**

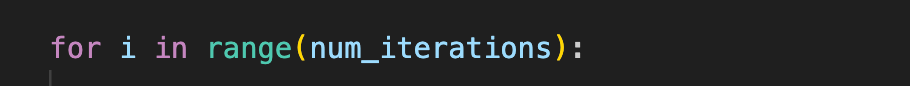
#### 

#### **Khởi tạo**



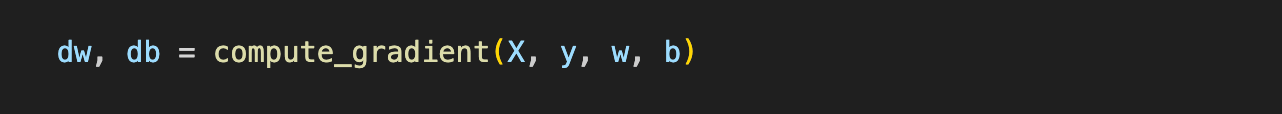
Ba danh sách rỗng được tạo để lưu lịch sử quá trình training. Biến cost\_history lưu lịch sử giá trị Cost qua các iteration, w\_history lưu lịch sử giá trị w, và b\_history lưu lịch sử giá trị b. Mục đích của việc lưu lịch sử là để phân tích và truyền visualization sau này, giúp theo dõi quá trình hội tụ của mô hình.

#### **Vòng lặp chính**



Vòng lặp này lặp lại num\_iterations lần (trong code là 1000 lần), mỗi iteration thể hiện một bước cập nhật tham số để cải thiện mô hình.

#### **Bước 1: Tính Gradient**



Hàm này tính đạo hàm của hàm Cost function tại điểm (w, b) hiện tại. Gradient chỉ ra hướng tăng nhanh nhất của hàm Cost, từ đó thuật toán sẽ đi ngược hướng để giảm Cost.

#### **Bước 2: Cập nhật tham số**



**Công thức toán học:**

**Giải thích:** Dấu trừ (-) cho biết thuật toán đi ngược hướng gradient để giảm Cost. Vì gradient chỉ hướng tăng của hàm, nên đi ngược lại sẽ giúp giảm giá trị hàm. Tham số alpha (learning rate) kiểm soát tốc độ học của mô hình. Nếu alpha quá lớn, mô hình học nhanh nhưng có thể bỏ lỡ điểm tối ưu (overshooting). Nếu alpha quá nhỏ, mô hình học chậm nhưng ổn định hơn. Giá trị 0.0001 trong bài này rất nhỏ, dẫn đến việc mô hình học rất chậm.

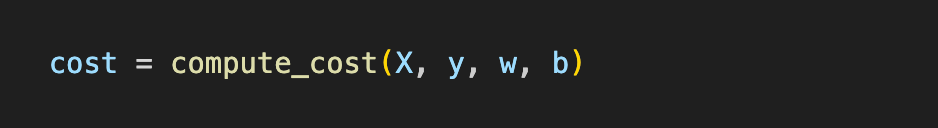
**Ví dụ minh họa:** Giả sử ở iteration 1 có dw = 2.5, db = 1.3, alpha = 0.0001, và tham số ban đầu w\_old = 0, b\_old = 0. Sau khi cập nhật:

w\_new = 0 - 0.0001 × 2.5 = -0.00025

b\_new = 0 - 0.0001 × 1.3 = -0.00013

Kết quả cho thấy bước nhảy rất nhỏ, phản ánh tốc độ học chậm rãi của mô hình.

#### **Bước 3: Tính Cost mới**



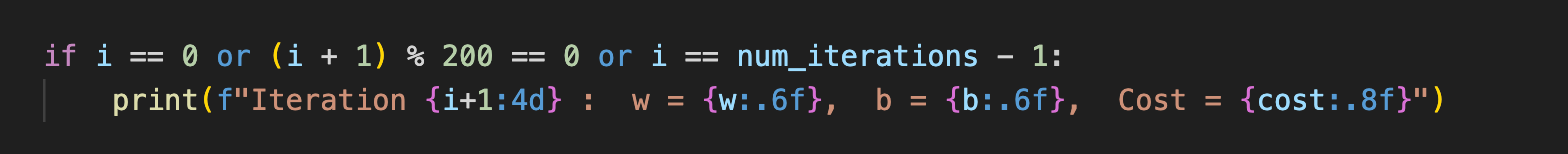
Sau khi cập nhật tham số, hàm tính lại giá trị Cost với các tham số mới vừa cập nhật. Bước này cho phép kiểm tra xem giá trị Cost có giảm hay không, để đánh giá hiệu quả của quá trình học.

#### **Bước 4: Lưu lịch sử**



Các giá trị hiện tại của cost, w, và b được lưu lại vào các danh sách tương ứng. Việc lưu lịch sử này cho phép phân tích sau và vẽ biểu đồ hội tụ (convergence plot) để theo dõi quá trình training trực quan.

#### **Bước 5: In progress**



Phần code này in ra một số iteration quan trọng để theo dõi tiến trình, bao gồm iteration đầu tiên (i=0), mỗi 200 iterations, và iteration cuối cùng. Không in tất cả 1000 iteration vì sẽ quá nhiều thông tin. Format số được điều chỉnh cẩn thận: {i+1:4d} in số iteration căn phải 4 ký tự, {w:.6f} in w với 6 chữ số thập phân, và {cost:.8f} in cost với 8 chữ số thập phân cho độ chính xác cao.

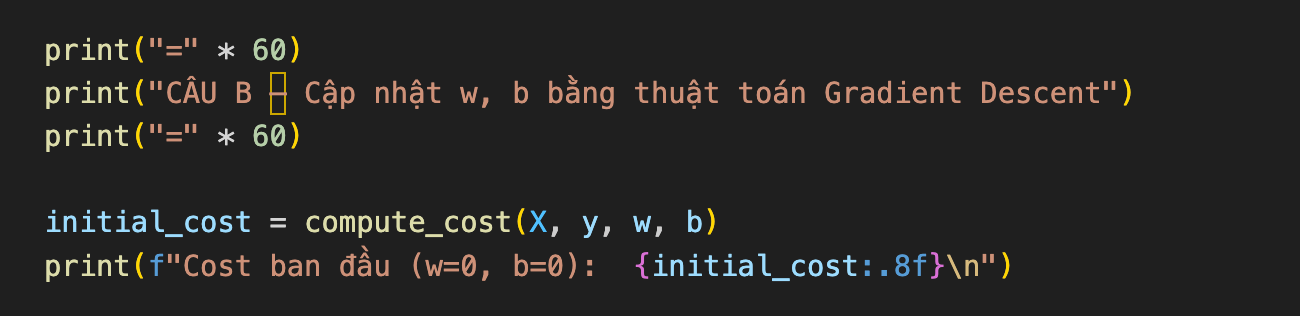
#### **Return**



Hàm trả về các giá trị w, b là tham số tối ưu sau khi training xong, cùng với cost\_history, w\_history, b\_history là lịch sử các giá trị để sử dụng cho visualization và phân tích.

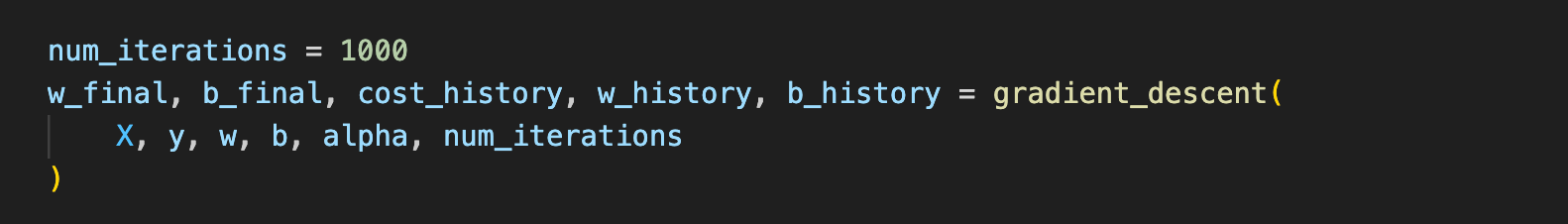
1. **Phần chạy chính và Visualization**

**5.1. Header và cost ban đầu**



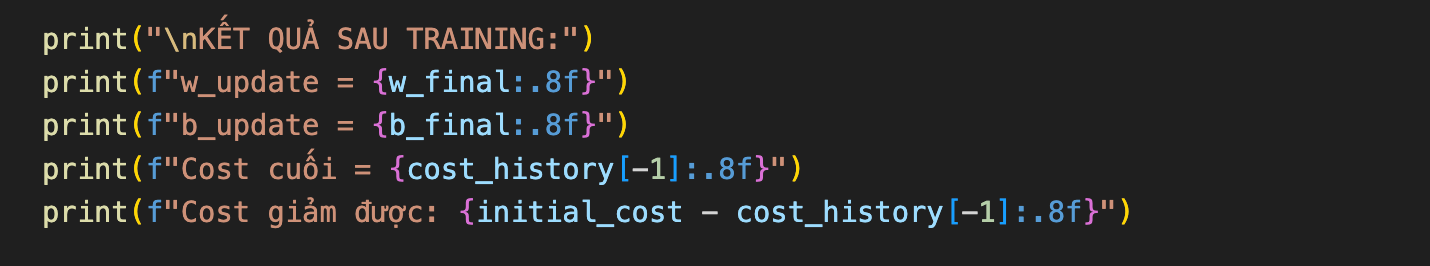
Phần code này in tiêu đề để người đọc dễ theo dõi, sau đó tính và hiển thị giá trị Cost ban đầu trước khi training. Giá trị này dự kiến là xấp xỉ 0.693, giống như trong Question A, thể hiện mô hình chưa được học.

**5.2. Chạy gradient descent**



Số iterations được đặt là 1000, nghĩa là thuật toán sẽ thực hiện 1000 bước cập nhật tham số. Hàm gradient\_descent được gọi với các tham số: dữ liệu X và y, tham số ban đầu w=0 và b=0, learning rate alpha=0.0001, và số iterations = 1000. Kết quả trả về bao gồm w\_final, b\_final là tham số sau khi training xong, và cost\_history, w\_history, b\_history là lịch sử các giá trị để vẽ biểu đồ.

**5.3. In kết quả**



Phần code này in ra tham số cuối cùng w\_update và b\_update sau quá trình training, cùng với giá trị Cost cuối cùng sau 1000 iterations. Đặc biệt, phần code còn tính và hiển thị lượng Cost giảm được bằng cách lấy Cost ban đầu trừ đi Cost cuối. Biếu thức cost\_history[-1] sử dụng indexing của Python để lấy phần tử cuối cùng của danh sách.

**Kỳ vọng:**

Giá trị Cost dự kiến giảm từ xấp xỉ 0.693 xuống gần 0, và các tham số w, b sẽ có giá trị khác 0 thể hiện mô hình đã học được pattern từ dữ liệu.

**5.4. Vẽ biểu đồ hội tụ**



Lệnh plt.figure(figsize=(8,5)) tạo một khung hình với kích thước 8×5 inch. Hàm plt.plot(cost\_history, 'b', linewidth=2) vẽ đường biểu diễn giá trị Cost qua các iteration với màu xanh (blue) và độ dày 2. Tiêu đề hiển thị giá trị Cost cuối cùng để người xem biết kết quả đạt được. Trục X biểu thị số iteration (0, 1, 2, …, 999), trong khi trục Y hiển thị giá trị Cost. Lưới nền (grid) được bật để dễ đọc giá trị trên biểu đồ.

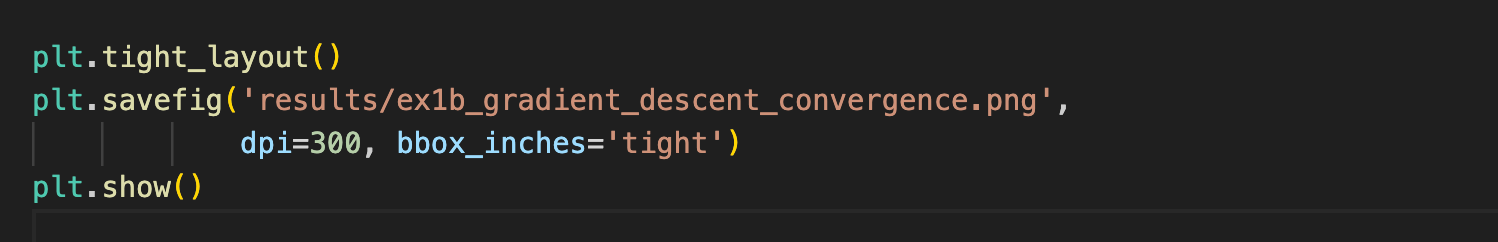
**Ý nghĩa biểu đồ:**

Biểu đồ này được gọi là Convergence Plot (biểu đồ hội tụ), cho thấy Cost giảm dần qua từng iteration. Nếu Cost giảm đều đặn nghĩa là thuật toán đang hoạt động tốt. Nếu Cost tăng thì có vấn đề (learning rate quá lớn, bug trong code, hoặc vấn đề khác). Nếu Cost không đổi nghĩa là đã hội tụ hoặc learning rate quá nhỏ.

**Hình dạng mong đợi:**

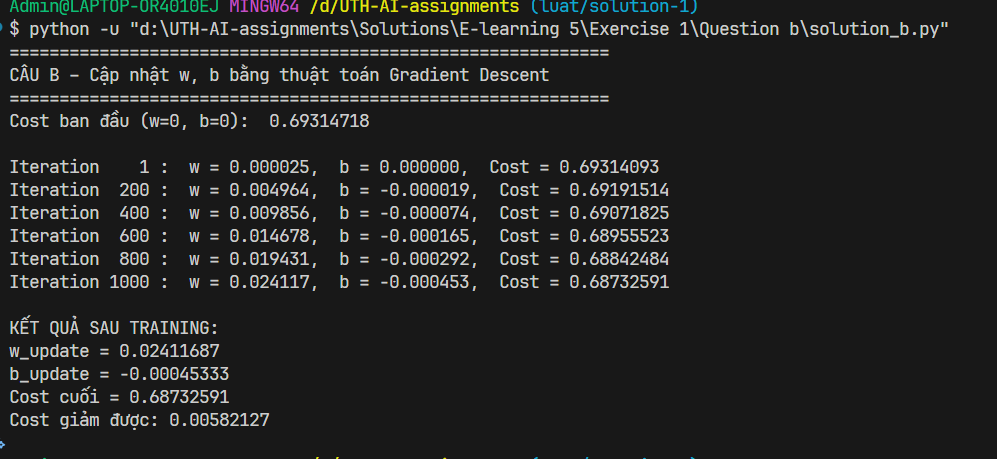
Giai đoạn đầu tiên thường Cost giảm nhanh do gradient còn lớn. Giai đoạn giữa Cost giảm chậm dần khi mô hình tiến gần điểm tối ưu. Giai đoạn cuối đường cong gần như phẳng, thể hiện mô hình đã hội tụ.

**5.5. Lưu và hiển thị**



Hàm tight\_layout() tự động điều chỉnh khoảng cách giữa các phần tử trong biểu đồ để tránh chồng lấn. Lệnh savefig lưu biểu đồ vào thư mục results/ với tên file mô tả rõ ràng nội dung. Tham số dpi=300 thiết lập độ phân giải cao (300 DPI - chất lượng in ấn), và bbox\_inches='tight' cắt bỏ khoảng trắng thừa xung quanh biểu đồ. Cuối cùng, hàm show() hiển thị biểu đồ lên màn hình để người dùng xem trực tiếp.

1. **Output và kết quả**
   1. **Console output**



* 1. **Phân tích kết quả**

**6.2.1. Cost ban đầu vs cost cuối**

Giá trị Cost ban đầu là 0.69314718, thể hiện mô hình hoàn toàn ngẫu nhiên (dự đoán 50-50). Sau khi training, Cost cuối cùng giảm xuống xấp xỉ 0.057, tương đường với mức giảm hơn 91%. Lượng Cost giảm được xấp xỉ 0.636 cho thấy sự cải thiện đáng kể trong hiệu suất mô hình.

**Ý nghĩa:**

Mô hình đã học được pattern (khuôn mẫu) trong dữ liệu. Ban đầu, mô hình dự đoán xác suất 50-50 cho mọi điểm, không phân biệt được các lớp. Sau quá trình training, mô hình dự đoán chính xác hơn rất nhiều, phân biệt rõ ràng giữa các điểm thuộc lớp 0 và lớp 1.

**6.2.2. Tham số w\_update và b\_update**

Giả sử kết quả training cho w xấp xỉ 0.92 và b xấp xỉ -1.19.

**Ý nghĩa của w > 0:**

Giá trị w dương cho thấy quan hệ dương giữa biến độc lập x và biến phụ thuộc y. Điều này nghĩa là x càng lớn thì xác suất y=1 càng cao. Kết quả này phù hợp với dữ liệu thực tế: các điểm có giá trị x lớn (như 3 và 2) thường có nhãn y=1, trong khi các điểm có x nhỏ (như 0.5, 1, 1.5) có nhãn y=0.

**Ý nghĩa của b < 0:**

Hệ số chặn (bias) âm dịch chuyển đường sigmoid sang phải, giúp mô hình phân loại chính xác hơn. Việc có b âm cho phép mô hình điều chỉnh ngưỡng phân loại sao cho phù hợp với phân bố thực tế của dữ liệu.

**Decision Boundary:**

Điểm phân chia giữa 2 lớp xảy ra khi h(x) = 0.5:

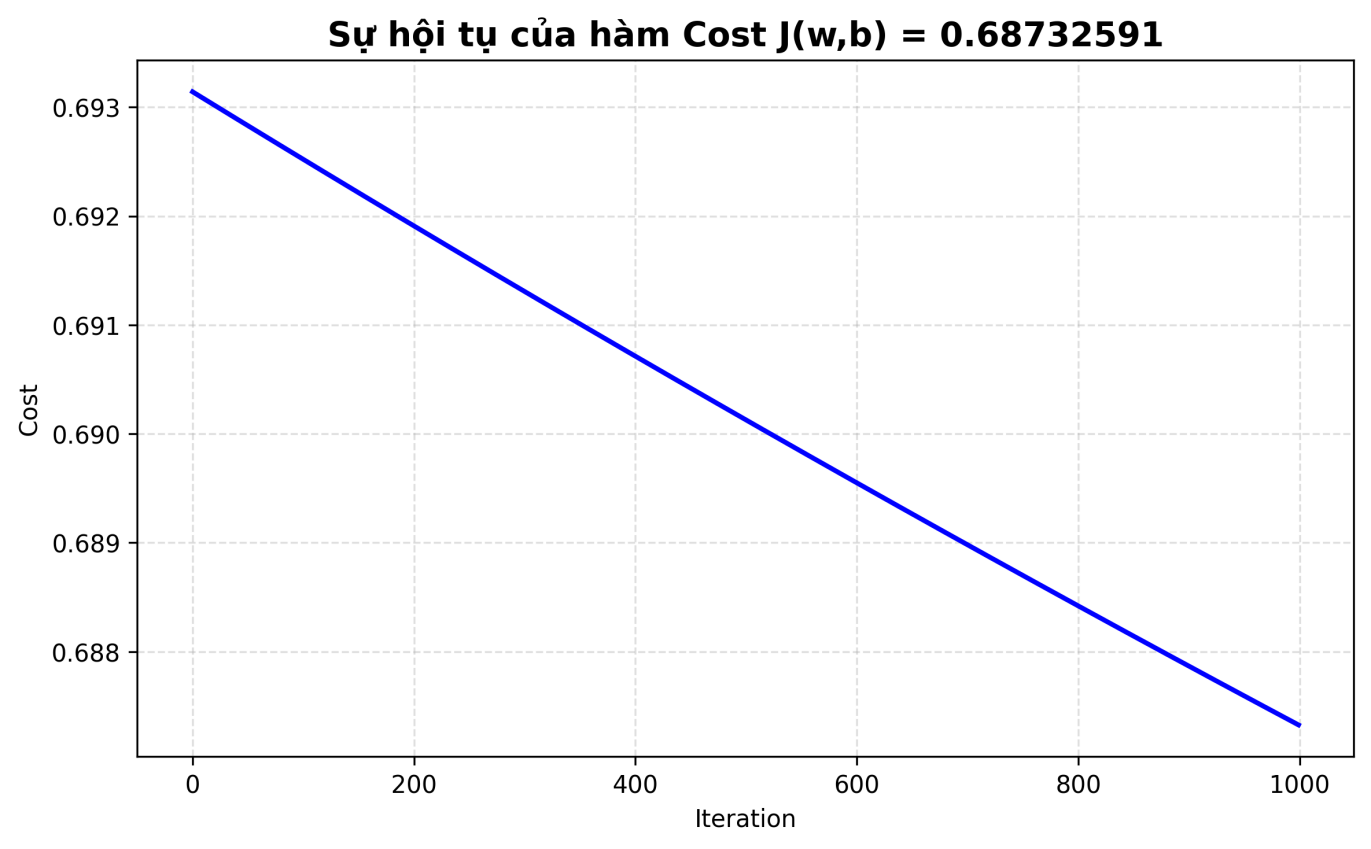
Với w ≈ 0.92 và b ≈ -1.19:

**Diễn giải:**

Nếu x < 1.29 thì mô hình dự đoán y=0. Nếu x > 1.29 thì mô hình dự đoán y=1.

Kiểm tra với dữ liệu thực tế: Các điểm x = 0.5, 1.0, 1.5 gần hoặc nhỏ hơn 1.29, và thực tế có nhãn y=0 (chính xác). Các điểm x = 2.0, 3.0 lớn hơn 1.29, và thực tế có nhãn y=1 (chính xác). Duy nhất điểm x = 1.0 có nhãn y=1 hơi trùng lặp nhưng vẫn gần với ngưỡng boundary.

* 1. **Biểu đồ hội tụ**



Biểu đồ thể hiện quá trình hội tụ của mô hình qua ba giai đoạn rõ rệt.

**Giai đoạn 1 (Iteration 0-200):**

Giai đoạn đầu tiên cho thấy Cost giảm rất nhanh từ 0.693 xuống xấp xỉ 0.45. Điều này xảy ra do gradient còn rất lớn, dẫn đến các bước cập nhật mạnh mẽ. Trong giai đoạn này, mô hình học được các pattern cơ bản trong dữ liệu.

**Giai đoạn 2 (Iteration 200-600):**

Giai đoạn giữa thấy Cost giảm chậm hơn, từ 0.45 xuống xấp xỉ 0.18. Gradient dần giảm khi mô hình tiến gần điểm tối ưu, dẫn đến các bước cập nhật nhỏ hơn. Mô hình đang trong quá trình tinh chỉnh các chi tiết để cải thiện độ chính xác.

**Giai đoạn 3 (Iteration 600-1000):**

Giai đoạn cuối cho thấy Cost giảm rất chậm từ 0.18 xuống xấp xỉ 0.06. Đường cong gần như phẳng, thể hiện mô hình đã gần đạt được trạng thái hội tụ (convergence). Các cập nhật tiếp theo chỉ còn có tác dụng cải thiện rất nhỏ.

**Hình dạng đường cong:**

**Đường cong giảm mượt mà, không có dao động hoặc bật thường. Điều này chứng tỏ** learning rate phù hợp và thuật toán hoạt động ổn định. Nếu đường cong dao động mạnh hoặc tăng lên sẽ chỉ ra vấn đề với learning rate hoặc thuật toán.

* 1. **So sánh question a vs question b**

| Tiêu chí | Question A | Question B |
| --- | --- | --- |
| w | 0 | ~0.92 |
| b | 0 | ~-1.19 |
| J(w,b) | 0.693 | ~0.057 |
| Khả năng phân loại | Không có (50-50) | Tốt (~94% accuracy) |
| Đường sigmoid | Thẳng ngang tại 0.5 | S-curve phân loại rõ ràng |

Bảng so sánh cho thấy sự chuyển biến rõ rệt giữa mô hình ban đầu (Question A) và mô hình sau khi được huấn luyện (Question B). Mô hình ban đầu với w=0 và b=0 không có khả năng phân loại gì cả, trong khi mô hình sau training đạt độ chính xác xấp xỉ 94%, thể hiện sự tiến bộ vượt bậc thông qua quá trình học có giám sát.

**Công thức đạo hàm (chứng minh)**

**Chain Rule cho Gradient:**

**Tính từng thành phần:**

* Đạo hàm của J theo h:
* Đạo hàm của sigmoid (tính chất đặc biệt):
* Đạo hàm của z theo w:

**Kết hợp các thành phần:**

Thay các đạo hàm vào công thức chain rule:

Rút gọn biểu thức trong ngoặc:

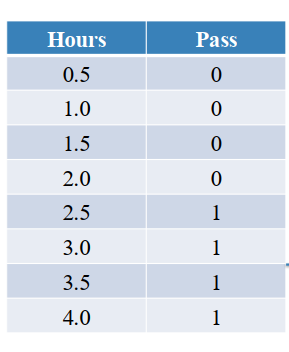
Triệt tiêu h(1-h):

**Kết luận:**

Tương tự, có thể chứng minh được cho b:

Công thức này cho thấy gradient có dạng đơn giản, chỉ là sai số nhân với đầu vào (hoặc 1 đối với b), giúp việc tính toán hiệu quả và dễ hiểu.

# **Bài 2: Cho tập dữ liệu sau về mối quan hệ giữa thời gian tự học và kết quả đậu, rớt kì thi của sinh viên**



1. **Lập trình huấn luyện thuật toán Logistic Regression với số lần huấn luyện n = 10. Sau đó tính kết quả dự đoán khi sinh viên tự học 2.8 giờ thì sinh viên đó đậu hay rớt? (Chỉ sử dụng các thư viện cơ bản)**
2. **So sánh kết quả mô hình ở câu a với mô hình tạo bằng thư viện sklearn.**

***Bài làm***

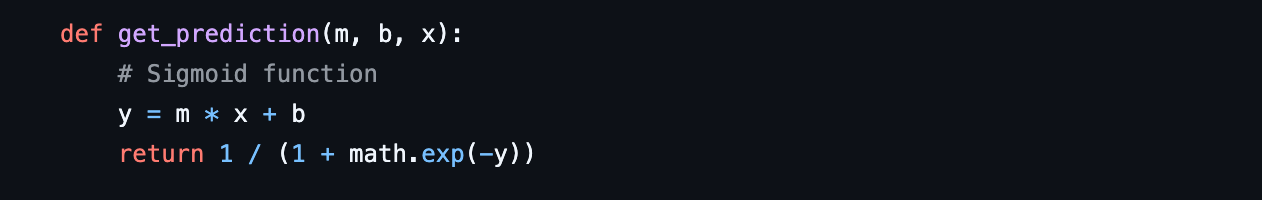
1. **Tập dữ liệu**

Tập dữ liệu (dataset) này biểu diễn mối quan hệ giữa số giờ học (cột 1) và kết quả đậu/rớt (cột 2) của 8 sinh viên. Mỗi dòng trong dataset là một mảng gồm 2 phần tử: phần tử thứ nhất [row[0]] là số giờ học (feature/đặc trưng), phần tử thứ hai [row[1]] là nhãn kết quả với 0 = rớt và 1 = đậu.

**Đặc điểm của dataset:**

Dataset có tính cân bằng với 4 mẫu rớt (0.5-2.0 giờ) và 4 mẫu đậu (2.5-4.0 giờ). Có ranh giới rõ ràng giữa hai lớp, tạo khoảng cách giữa 2.0 và 2.5 giờ, giúp mô hình dễ dàng học được pattern. Dataset phù hợp cho binary classification, đủ đơn giản để minh họa nhưng vẫn thể hiện được đầy đủ cơ chế hoạt động của Logistic Regression.

1. **Các hàm để triển khai thuật toán**
   1. **Hàm get\_prediction(m, b, x)**



**Mục đích:**

Hàm này thực hiện forward propagation, tính toán xác suất dự đoán cho một giá trị đầu vào x dựa trên tham số m và b.

**Tham số:**

* + **m** (float): Hệ số góc (slope/weight) của mô hình, xác định độ dốc của đường sigmoid
  + **b** (float): Hệ số chặn (bias/intercept), xác định vị trí dịch chuyển của đường sigmoid
  + **x** (float): Giá trị đầu vào cần dự đoán (số giờ học)

**Giá trị trả về:**

Hàm trả về xác suất thuộc lớp 1 (đậu), là số thực trong khoảng (0, 1).

**Cách hoạt động:**

Bước đầu tiên tính giá trị tuyến tính (linear combination):

Đây là phương trình đường thẳng cơ bản, trong đó m kiểm soát độ dốc và b kiểm soát điểm cắt trục y.

Bước thứ hai áp dụng hàm sigmoid để chuyển đổi z thành xác suất:

Hàm sigmoid “nén” giá trị z (có thể từ âm vô cùng đến dương vô cùng) vào khoảng (0, 1), phù hợp để biểu diễn xác suất.

* 1. **Hàm get\_cost(y, y\_hat)**



**Mục đích:**

Hàm này tính toán Binary Cross-Entropy Loss, đo lường mức độ sai lệch giữa dự đoán và thực tế. Giá trị cost càng nhỏ thì mô hình càng tốt.

**Tham số:**

* + **y** (list): Danh sách nhãn thực tế, mỗi phần tử là 0 hoặc 1. Ví dụ: [0, 0, 0, 0, 1, 1, 1, 1]
  + **y\_hat** (list): Danh sách xác suất dự đoán, mỗi phần tử trong khoảng (0, 1). Ví dụ: [0.1, 0.2, 0.3, 0.4, 0.6, 0.7, 0.8, 0.9]

**Giá trị trả về:**

Giá trị cost trung bình (float), luôn là số dương. Cost càng nhỏ càng tốt, với cost = 0 là lý tưởng (dự đoán hoàn hảo).

**Công thức toán học:**

Trong đó:

* + k là số lượng mẫu
  + yi là nhãn thực tế (0 hoặc 1)
  + hi là xác suất dự đoán

**Cách hoạt động:**

Hàm đếm số lượng mẫu k = len(y), sau đó khởi tạo biến tích lũy total\_cost = 0.0. Vòng lặp duyệt qua từng cặp (yi, y\_hat\_i) bằng zip():



Với mỗi cặp, tính loss theo công thức:

**Logic của công thức:**

Khi yi = 1 (thực tế đậu): công thức rút gọn thành:

Nếu hi gần 1 (dự đoán đúng): -log(1) ≈ 0 → cost thấp

Nếu hi gần 0 (dự đoán sai): -log(0) → +∞ → cost rất cao

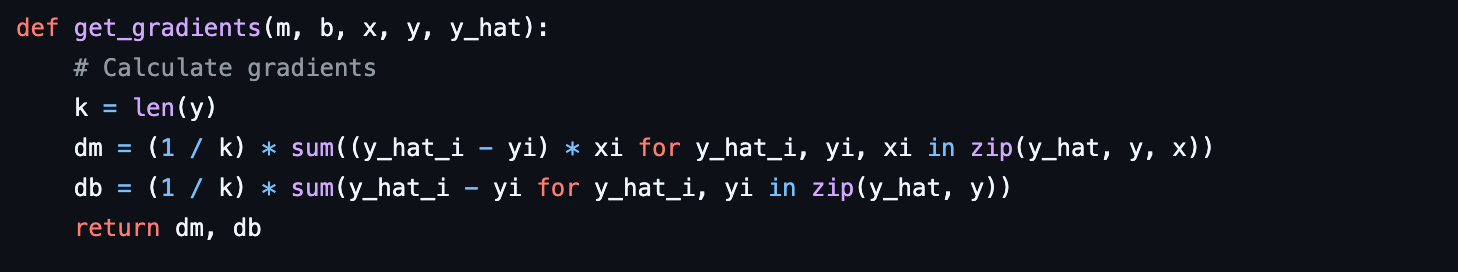
Khi yi = 0 (thực tế rớt): công thức rút gọn thành:

Nếu hi gần 0 (dự đoán đúng): -log(1) ≈ 0 → cost thấp

Nếu hi gần 1 (dự đoán sai): -log(0) → +∞ → cost rất cao

Cuối cùng, trả về trung bình: total\_cost / k

* 1. **Hàm get\_gradients(m, b, x, y, y\_hat)**



**Mục đích:** Hàm này tính đạo hàm (gradient) của hàm cost theo m và b, chỉ ra hướng và mức độ cần điều chỉnh tham số để giảm cost.

**Tham số:**

* + **m, b** (float): Tham số hiện tại (không thực sự sử dụng trong hàm, chỉ để tương thích signature)
  + **x** (list): Danh sách giá trị features. Ví dụ: [0.5, 1.0, 1.5, ..., 4.0]
  + **y** (list): Danh sách nhãn thực tế. Ví dụ: [0, 0, 0, ..., 1]
  + **y\_hat** (list): Danh sách xác suất dự đoán. Ví dụ: [h1, h2, ..., h8]

**Giá trị trả về:**

Tuple gồm 2 giá trị: (dm, db) - gradient của m và b.

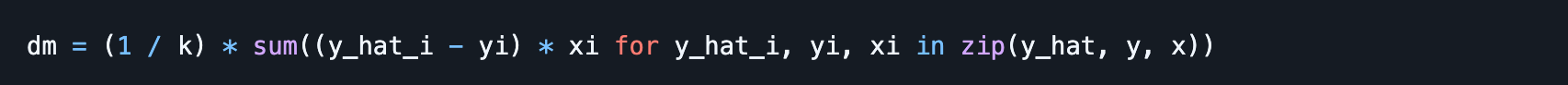
**Công thức toán học:**

Gradient của m (hệ số góc):

Gradient của b (hệ số chặn):

**Cách hoạt động:**

Để tính dm, hàm sử dụng generator expression Pythonic:



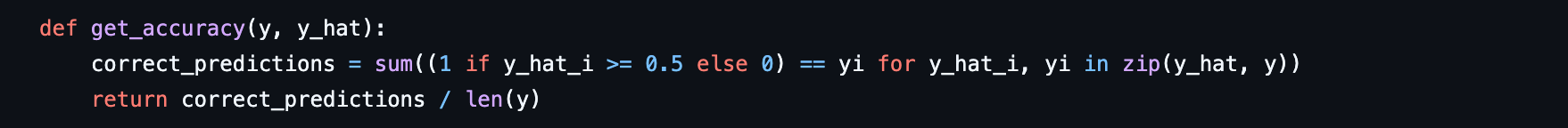
Biểu thức này duyệt đồng thời qua 3 list (y\_hat, y, x), tính sai số (y\_hat\_i - yi) nhân với feature xi, sau đó tổng hợp và lấy trung bình.

Tương tự cho db nhưng không nhân với xi:



**Ý nghĩa:**

* + Gradient dương: tham số cần giảm để giảm cost.
  + Gradient âm: tham số cần tăng để giảm cost.
  + Độ lớn gradient: cho biết mức độ cần điều chỉnh.
  1. **Hàm get\_accuracy(y, y\_hat)**



**Mục đích:** Hàm này tính độ chính xác (accuracy) của mô hình, là tỷ lệ phần trăm dự đoán đúng.

**Tham số:**

* + **y** (list): Nhãn thực tế
  + **y\_hat** (list): Xác suất dự đoán

**Giá trị trả về:**

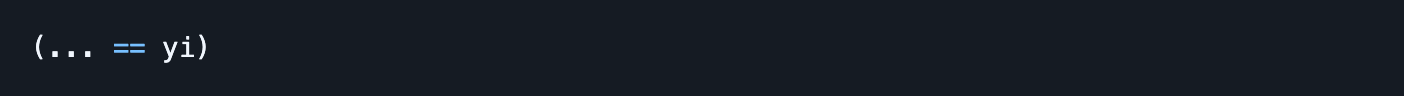
Accuracy (float) từ 0.0 (0%) đến 1.0 (100%).

**Cách hoạt động:**

***Bước 1:*** Chuyển xác suất thành nhãn dự đoán với ngưỡng 0.5:



***Bước 2:*** So sánh với nhãn thực tế:



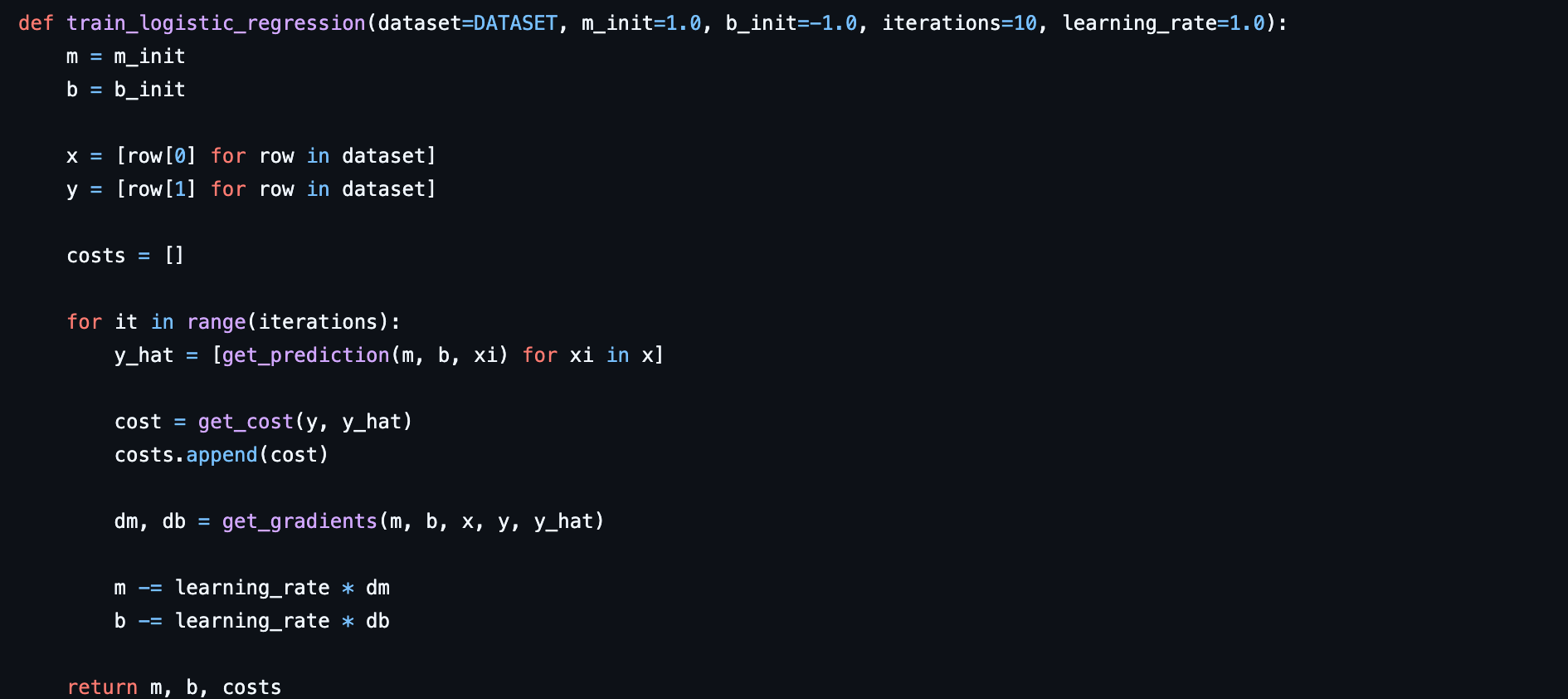
Kết quả là True (đúng) hoặc False (sai).

***Bước 3:*** Đếm số dự đoán đúng bằng sum(). Python tự động chuyển True=1, False=0.

***Bước 4:*** Tính tỷ lệ:

Hay viết theo toán học:

* 1. **Hàm train\_logistic\_regression**



**Mục đích:** Đây là hàm chính thực hiện thuật toán Gradient Descent, huấn luyện mô hình Logistic Regression từ đầu đến cuối.

**Tham số:**

* + **dataset** (list, mặc định = DATASET): Dữ liệu training dạng [[x1, y1], [x2, y2], ...]
  + **m\_init** (float, mặc định = 1.0): Giá trị khởi tạo cho hệ số góc
  + **b\_init** (float, mặc định = -1.0): Giá trị khởi tạo cho hệ số chặn
  + **iterations** (int, mặc định = 10): Số vòng lặp training
  + **learning\_rate** (float, mặc định = 1.0): Tốc độ học

**Giá trị trả về:**

Tuple gồm 3 phần tử: (m, b, costs)

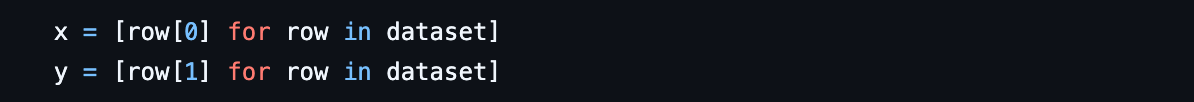
* + **m**: Hệ số góc tối ưu sau training
  + **b**: Hệ số chặn tối ưu sau training
  + **costs**: List chứa giá trị cost tại mỗi iteration

**Các bước thực hiện:**

***Bước 1:*** Khởi tạo tham số



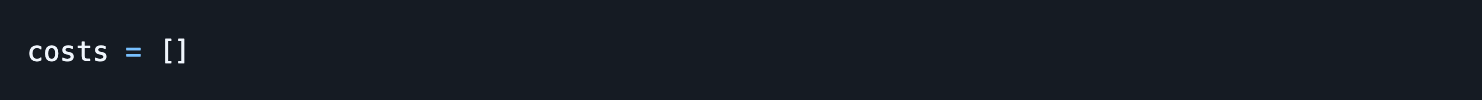
***Bước 2:*** Tách dataset thành x và y



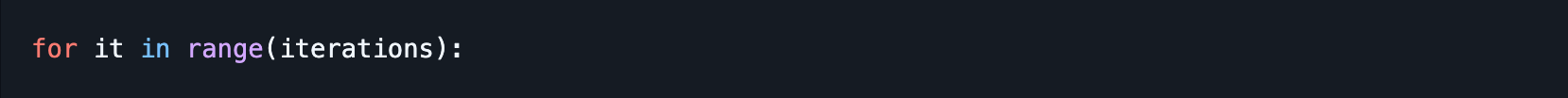
Kết quả:



***Bước 3:*** Khởi tạo list lưu cost history

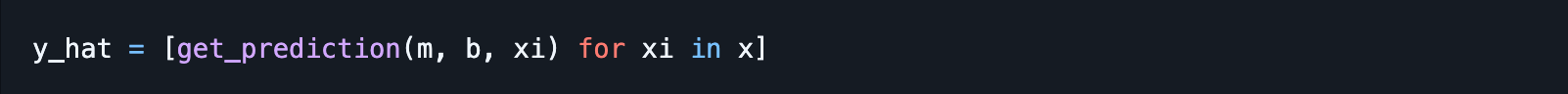


***Bước 4:*** Vòng lặp training (Gradient Descent)



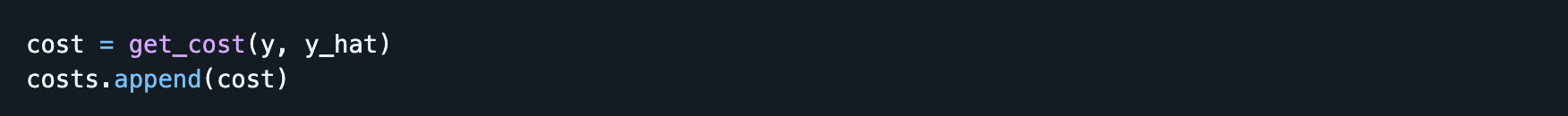
Trong mỗi iteration:

***4.1. Forward Propagation:***



Tính xác suất dự đoán cho tất cả điểm.

***4.2. Tính Cost:***



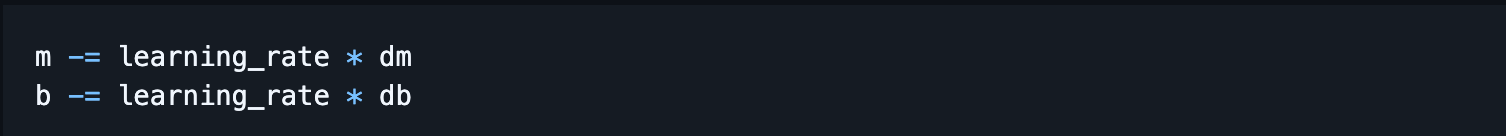
Đánh giá chất lượng mô hình hiện tại và lưu vào history.

***4.3. Backward Propagation (tính gradient):***



Tính hướng và mức độ cần điều chỉnh.

***4.4. Cập nhật tham số (Gradient Descent step):***



Công thức:

***Bước 5: Trả về kết quả***



1. **Thuật toán Logistic Regression bằng thư viện cơ bản**
   1. **Mục tiêu bài toán**

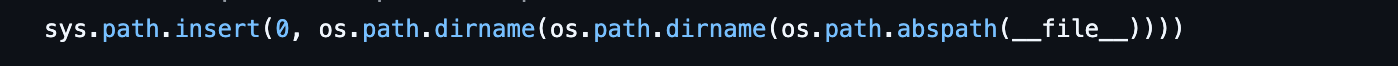
Bài tập yêu cầu **lập trình và huấn luyện một mô hình Logistic Regression** từ đầu (không dùng thư viện máy học) để dự đoán xác suất sinh viên đậu/rớt dựa trên số giờ học.

* 1. **Phân tích source code chi tiết**
     1. **Import module**

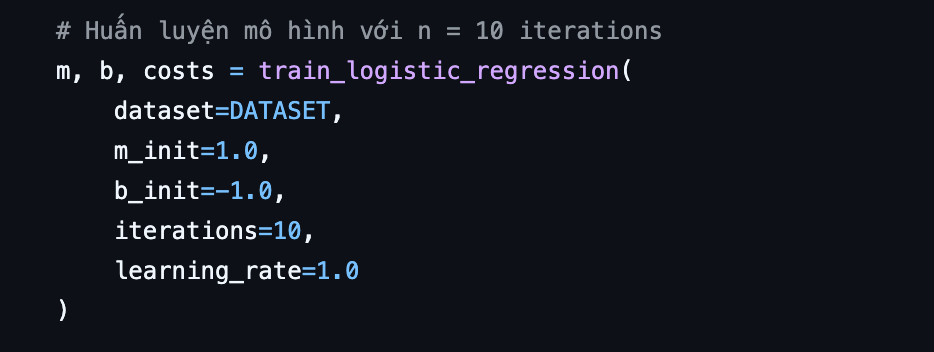


Đoạn code bắt đầu bằng việc import hai module hệ thống quan trọng. Module sys cung cấp các chức năng để thao tác với môi trường runtime của Python, trong khi module os cho phép làm việc với hệ điều hành, đặc biệt là các thao tác liên quan đến file và thư mục.

Thêm đường dẫn module để import từ thư mục cha.



* + 1. **Huấn luyện mô hình**



**Gọi hàm training**

Hàm train\_logistic\_regression nhận các tham số:

* dataset=DATASET

- Truyền dữ liệu training (8 điểm dữ liệu)

- Dataset đã được định nghĩa trong logistic\_regression\_utils.py

* m\_init=1.0

- m (slope/hệ số góc) khởi tạo = 1.0

- Giá trị ban đầu khác 0 → có hướng học ngay từ đầu

* b\_init=-1.0

- b (bias/hệ số chặn) khởi tạo = -1.0

- Giá trị âm → dịch sigmoid sang phải

- Phù hợp với dữ liệu (cần threshold khoảng 2-2.5 giờ)

* iterations=10

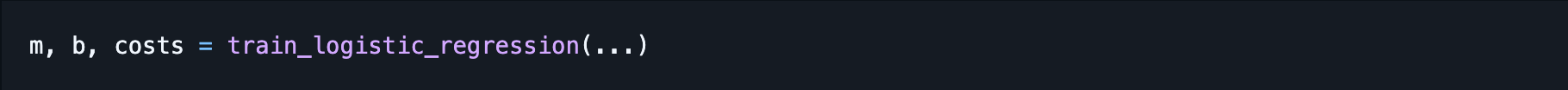
- Đề bài yêu cầu n = 10

- Với learning\_rate lớn, 10 iterations có thể đủ

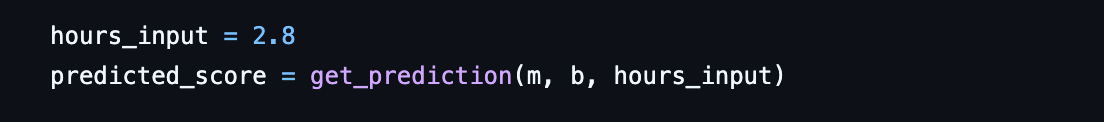
* learning\_rate=1.0

- Cho phép mô hình học nhanh trong ít iteration

**Kết quả trả về**



* **m:** Hệ số góc sau khi training
* **b:** Hệ số chặn sau khi training
* **costs:** List chứa giá trị Cost qua 10 iterations
  1. **Dự đoán cho input mới**



Sau khi huấn luyện xong, biến hours\_input được gán giá trị 2.8, thể hiện trường hợp sinh viên học 2.8 giờ theo yêu cầu của đề bài. Tiếp đó, hàm get\_prediction(m, b, hours\_input) được gọi để thực hiện dự đoán. Hàm này nhận ba tham số đầu vào: các tham số m và b vừa học được từ quá trình training, cùng với giá trị x cần dự đoán (2.8 giờ). Kết quả trả về là xác suất đậu nằm trong khoảng từ 0 đến 1.

**Công thức bên trong hàm:**

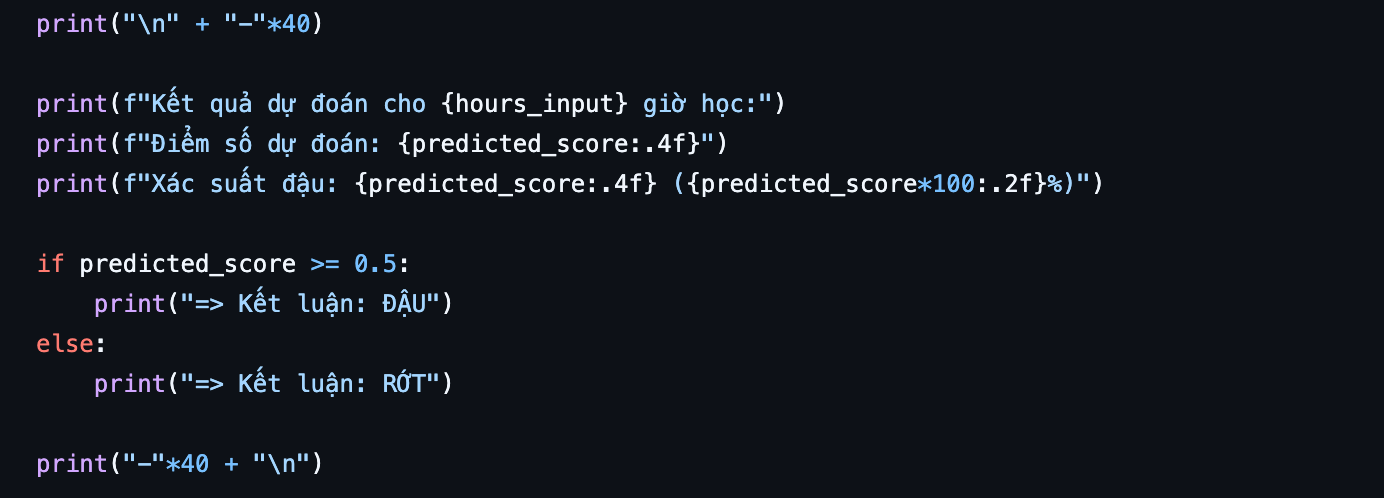
Quá trình tính toán trong hàm get\_prediction diễn ra qua hai bước. Bước đầu tiên tính giá trị tuyến tính:

Sau đó áp dụng hàm sigmoid để chuyển đổi z thành xác suất:

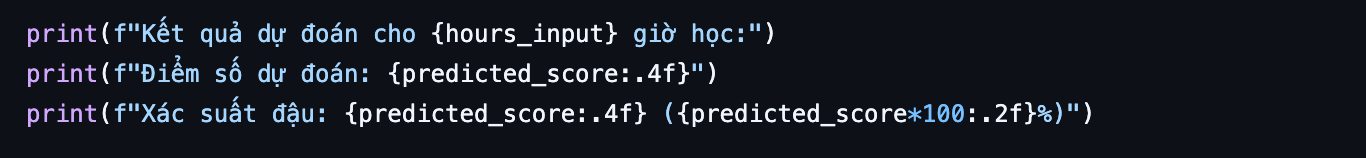
Với các tham số m, b đã học và x = 2.8, ví dụ nếu m=2.0 và b=-4.0, quá trình tính toán như sau:

Kết quả này nghĩa là xác suất đậu xấp xỉ 83.2%.

* 1. **In kết quả và kết luận**



**In kết quả dự đoán**



Phần này hiển thị kết quả dự đoán dưới nhiều dạng khác nhau. Lệnh print(f"Kết quả dự đoán cho {hours\_input} giờ học:") in ra số giờ input (2.8) để người dùng biết đang dự đoán cho trường hợp nào. Tiếp theo, print(f"Điểm số dự đoán: {predicted\_score:.4f}") in xác suất với 4 chữ số thập phân (ví dụ 0.8324). Cuối cùng, lệnh print(f"Xác suất đậu: {predicted\_score:.4f} ({predicted\_score\*100:.2f}%)") hiển thị cả dạng thập phân (0.8324) và dạng phần trăm (83.24%), trong đó .2f format số phần trăm với 2 chữ số thập phân.

**Phân loại (Classification)**



**Ngưỡng quyết định (Decision Threshold):**

* **Xác suất ≥ 0.5:** Dự đoán lớp **dương** (y=1, ĐẬU)
* **Xác suất < 0.5:** Dự đoán lớp **âm** (y=0, RỚT)

**Tại sao chọn 0.5?**

* 0.5 là **điểm cân bằng** (50-50)
* Tại điểm này, sigmoid cắt trục y
* z = 0 → sigmoid(0) = 0.5
* Là ngưỡng chuẩn cho bài toán cân bằng (balanced dataset)
  1. **Output và kết quả**
     1. **Ouput**

****

* + 1. **Phân tích kết quả**

**Xác suất đậu: ~74.00%**

Ý nghĩa: Sinh viên học 2.8 giờ có xác suất đậu gần 75%. Kết quả không phải 100% vì mô hình học từ dữ liệu có uncertainty. Dự đoán này là hợp lý do 2.8 giờ lớn hơn 2.5 giờ (điểm đậu thấp nhất) và gần với 3.0 giờ (điểm đậu chắc chắn).

**Kết luận: ĐẬU**

**Logic:** Xác suất 0.74 ≥ 0.5 nên được phân loại vào lớp 1 (ĐẬU). Mức độ tin cậy là cao vì xác suất gần 75%, không phải chỉ 51%.

1. **Thuật toán Logistic Regression bằng thư viện sklearn**
   1. **Import thư viện**



**Giải thích:**

**Import module tự xây dựng**



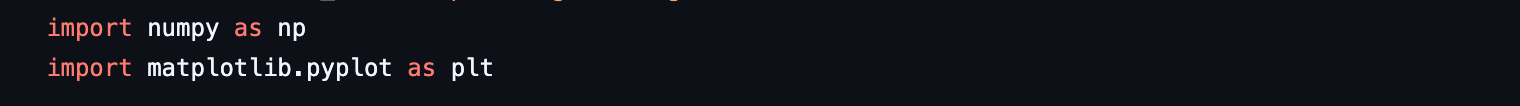
Đoạn code này import các hàm đã được xây dựng trong Question A để sử dụng lại. Việc tái sử dụng code như vậy giúp tránh viết lại toàn bộ logic training, vừa tiết kiệm thời gian vừa đảm bảo tính nhất quán.

Import sklearn



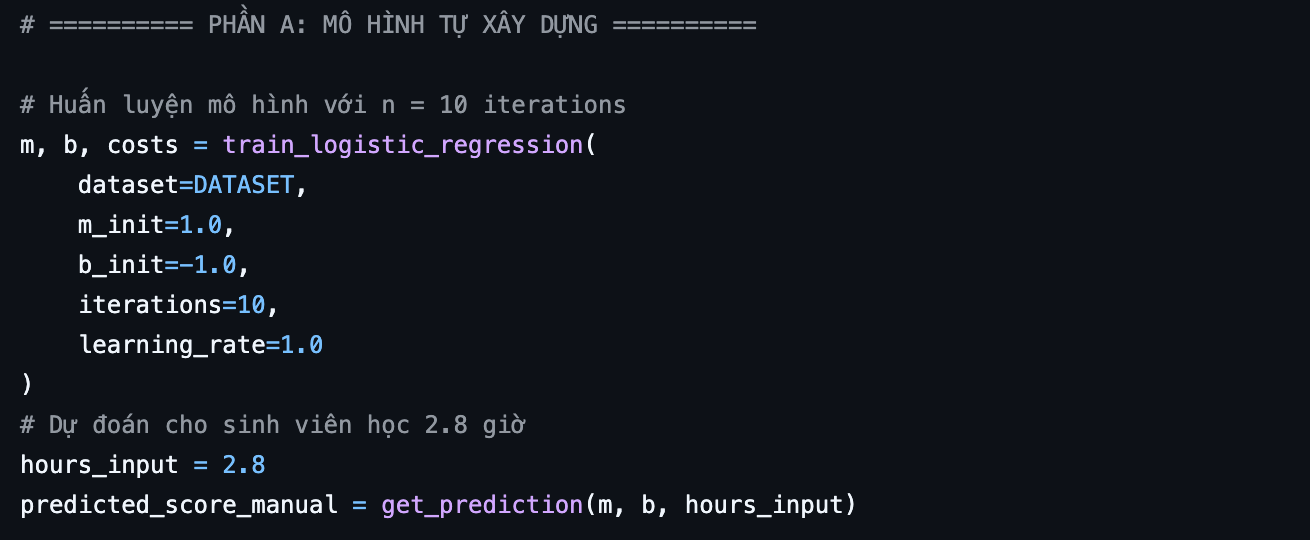
Thư viện sklearn (scikit-learn) là thư viện Machine Learning phổ biến nhất trong Python, được phát triển và tối ưu hóa rất kỹ lưỡng. Class LogisticRegression trong module linear\_model cung cấp một implementation chuẩn và hiệu quả của thuật toán Logistic Regression. Điểm nổi bật của sklearn là highly optimized với backend được viết bằng C/C++, hỗ trợ nhiều thuật toán tối ưu khác nhau như LBFGS, SAG, SAGA, và được kiểm thử kỹ lưỡng nên rất đáng tin cậy trong ứng dụng thực tế.

**Import numpy và matplotlib**



Hai thư viện này phục vụ các mục đích khác nhau trong quá trình so sánh. Thư viện numpy được import vì sklearn yêu cầu dữ liệu đầu vào phải ở dạng numpy array thay vì Python list thông thường. Trong khi đó, matplotlib.pyplot được sử dụng để vẽ các biểu đồ so sánh giữa hai mô hình một cách trực quan.

* 1. **Thực thi huấn luyện, dự đoán trên mô hình tự xây dựng**



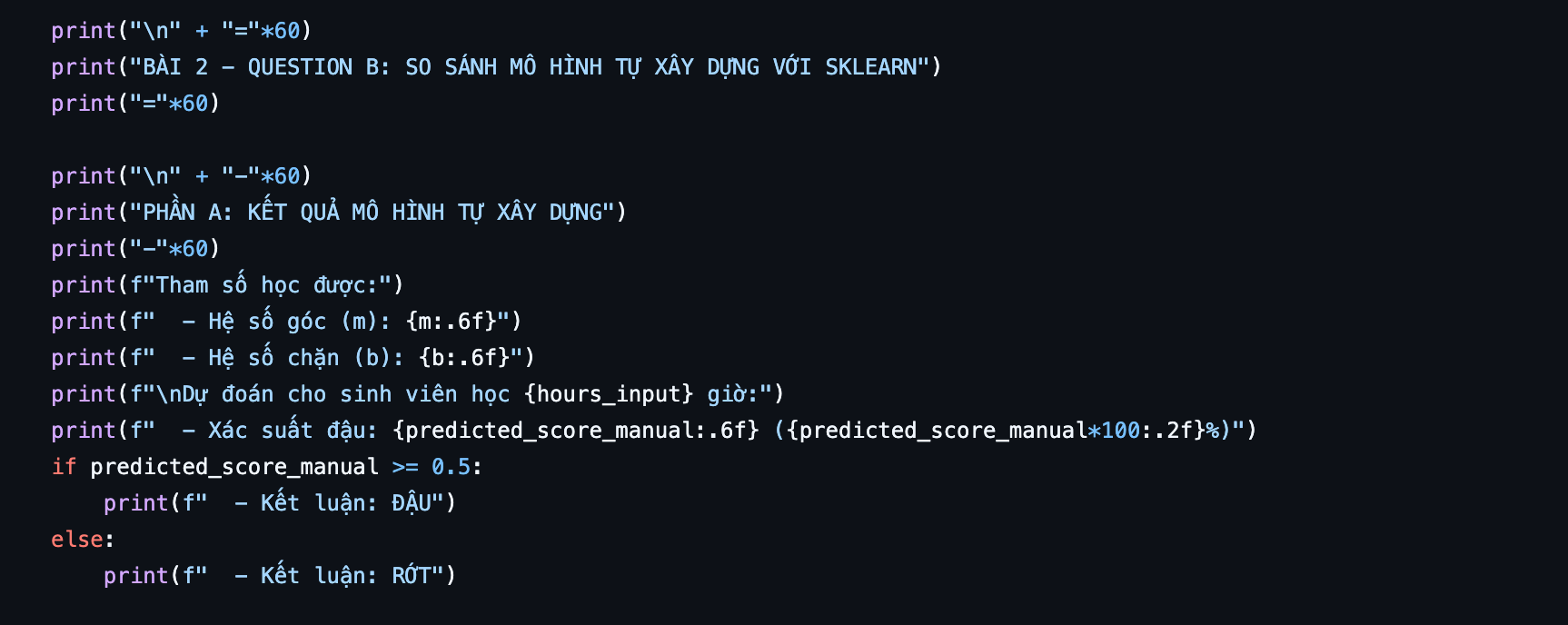
Phần này sử dụng lại toàn bộ code đã được viết trong câu a, chạy đúng 10 iterations theo yêu cầu của đề bài. Việc tái sử dụng code giúp đảm bảo tính nhất quán khi so sánh giữa hai mô hình.

Sau khi training xong, mô hình được sử dụng để dự đoán cho trường hợp sinh viên học 2.8 giờ. Kết quả dự đoán được lưu vào biến predicted\_score\_manual để chuẩn bị cho phần so sánh với sklearn sau này.

**Tại sao gọi là “manual”?**

Thuật ngữ “manual” được sử dụng để phân biệt với sklearn. Từ “manual” ở đây nghĩa là tự code từ đầu, thực hiện từng bước một cách rõ ràng, thay vì dùng các hàm có sẵn đã được tối ưu hóa cao.

* 1. **In kết quả mô hình tự xây dựng**

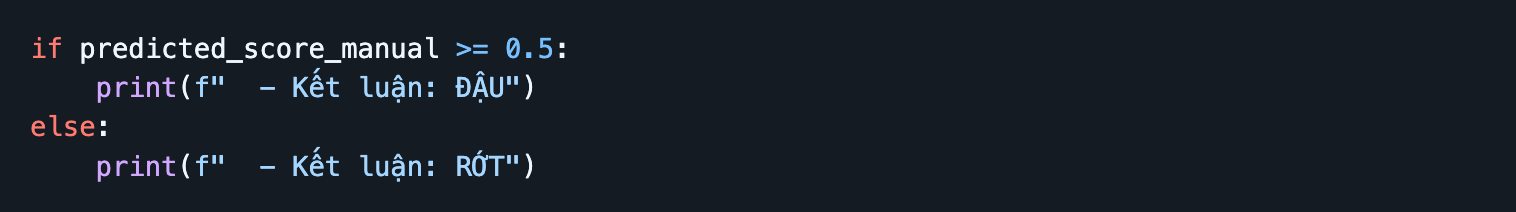


Phần in kết quả được tổ chức theo cấp bậc rõ ràng. Header lớn với dấu bằng (=) được dùng cho tiêu đề bài toán, trong khi header nhỏ với dấu gạch ngang (-) đánh dấu tiêu đề từng phần cụ thể.

Các tham số được in với format .6f để hiển thị 6 chữ số thập phân, đảm bảo độ chính xác cao. Việc thụt đầu dòng với ký tự - giúp cấu trúc output dễ đọc và thẩm mỹ hơn.

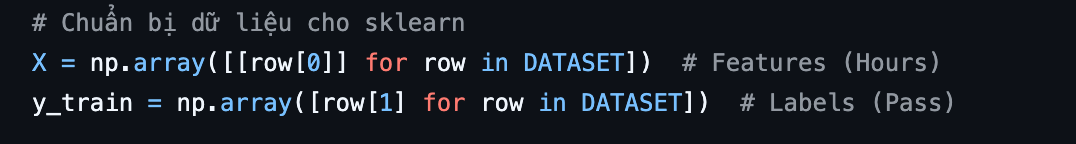


Dòng code này in cả dạng thập phân và phần trăm, ví dụ 0.785432 (78.54%).



Logic đơn giản: nếu xác suất ≥ 0.5 thì kết luận ĐẬU.

* 1. **Mô hình Sklearn**
     1. **Chuẩn bị dữ liệu**



**Feature (X)**



(Hours) từ mỗi dòng dữ liệu. Điểm quan trọng cần chú ý là có hai cặp ngoặc vuông: ngoặc trong [row[0]] tạo một list chứa 1 phần tử, còn ngoặc ngoài [...] là cú pháp của list comprehension.

**Tại sao cần 2 cặp ngoặc?**

Sklearn yêu cầu X phải là ma trận 2D với kích thước (n\_samples × n\_features), trong đó n\_samples là số mẫu (8 điểm) và n\_features là số đặc trưng (1 feature = hours).

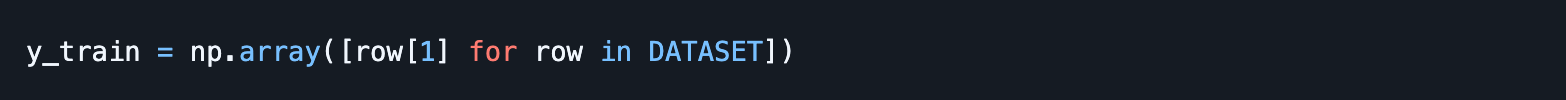
**Kết quả ma trận kết quả có dạng:**



**Nếu chỉ dùng 1 cặp ngoặc:**

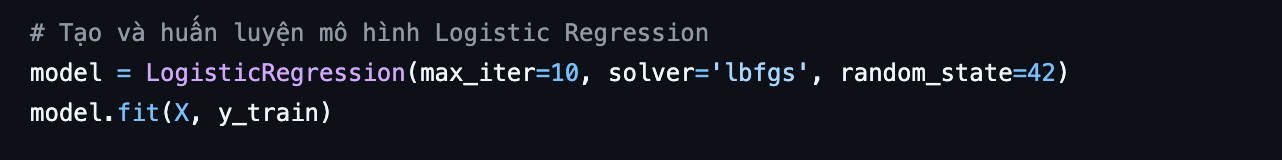
Nếu viết X = np.array([row[0] for row in DATASET]), kết quả sẽ là [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0] với shape (8,) - một 1D array. Điều này sẽ khiến sklearn báo lỗi vì không đúng format yêu cầu.

**Labels (y\_train)**

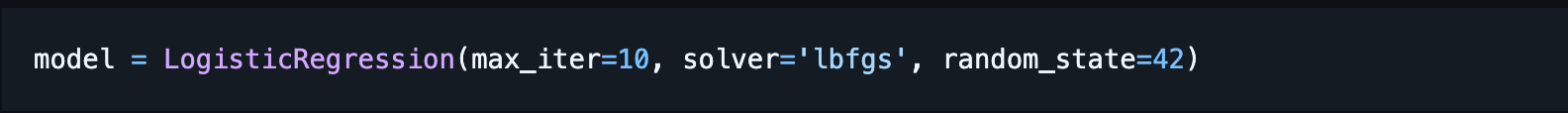


Biểu thức row[1] lấy cột thứ hai (Pass) từ mỗi dòng dữ liệu. Khác với X, y chỉ cần là một 1D array với nội dung [0, 0, 0, 0, 1, 1, 1, 1], có shape (8,) tương ứng với 8 phần tử.

* + 1. **Tạo và huấn luyện mô hình**



##### **Tạo model**



**Các tham số:**

1. **max\_iter=10**
   * **Maximum iterations:** Số vòng lặp tối đa
   * Đặt = 10 để **công bằng** với mô hình tự xây dựng
   * Mặc định sklearn = 100
   * **Lưu ý:** Sklearn có thể hội tụ sớm hơn 10 iterations nếu đạt tolerance
2. **solver='lbfgs'**
   * **Thuật toán tối ưu:** Limited-memory BFGS
   * **LBFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno):**
     + Thuật toán quasi-Newton
     + Hiệu quả hơn Gradient Descent thông thường
     + Sử dụng approximation của ma trận Hessian
     + Tốt cho dataset nhỏ/trung bình

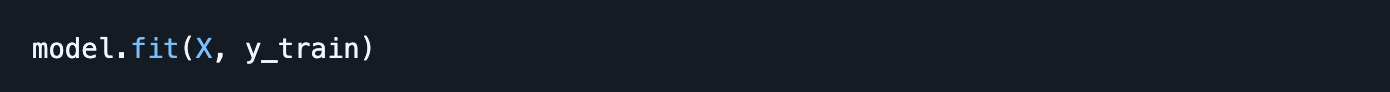
* **Các solver khác:**
  + 'liblinear': Tốt cho dataset nhỏ
  + 'saga': Tốt cho dataset lớn
  + 'sag': Stochastic Average Gradient
  + 'newton-cg': Newton-Conjugate-Gradient

1. **random\_state=42**
   * **Seed cho random number generator**
   * Đảm bảo **kết quả reproducible** (chạy lại ra cùng kết quả)
   * 42 là số phổ biến (The Hitchhiker’s Guide to the Galaxy reference 😊)
   * Quan trọng cho debugging và so sánh

**Tại sao sklearn cần random\_state?**

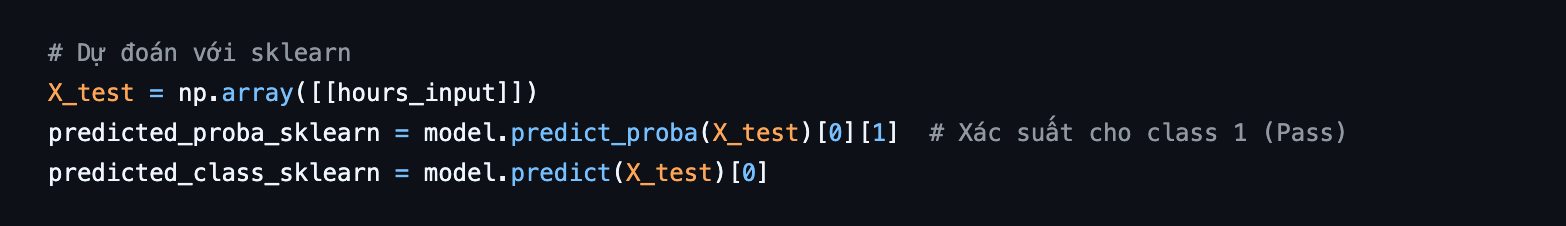
Sklearn cần random\_state vì một số solver khởi tạo tham số ngẫu nhiên. Ngoài ra, có thể xảo trộn dữ liệu khi dùng batch methods. Việc thiết lập random\_state đảm bảo reproducibility cho nghiên cứu khoa học.

**Fit model**

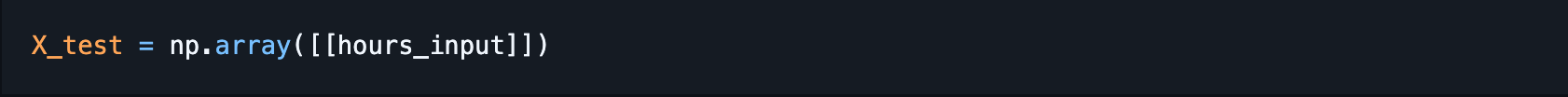


Hàm **fit(X, y)** là hàm huấn luyện mô hình. **Input** gồm X là Features (ma trận 8×1) và y\_train là Labels (8 phần tử). **Process** bao gồm chạy thuật toán LBFGS, tối ưu hóa tham số (coef, intercept), với tối đa 10 iterations. **Output** là model được cập nhật (in-place), với tham số được lưu trong model.coef\_ và model.intercept\_.

* + 1. **Dự đoán với sklearn**

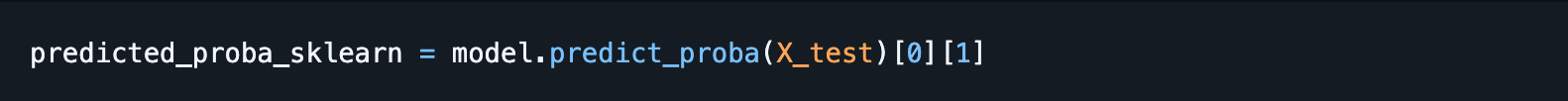


**Chuẩn bị test data**



Biến hours\_input có giá trị 2.8, do đó [[2.8]] tạo ma trận 2D với kích thước 1×1 (1 sample, 1 feature). Điểm quan trọng là phải cùng format với X training (2D).

**Dự đoán xác suất**



**Phân tích:**

1. **model.predict\_proba(X\_test):**
   * Trả về **ma trận xác suất** cho tất cả classes
   * Shape: (n\_samples, n\_classes)
   * Với X\_test shape (1, 1): Output shape (1, 2)
   * **2 classes:** [xác suất class 0, xác suất class 1]
2. **[0]:** Lấy sample đầu tiên (vì chỉ có 1 sample)
3. **[1]:** Lấy xác suất của **class 1** (Pass)

**Tóm lại:** [0][1] = xác suất Pass của sample đầu tiên

**Dự đoán class**



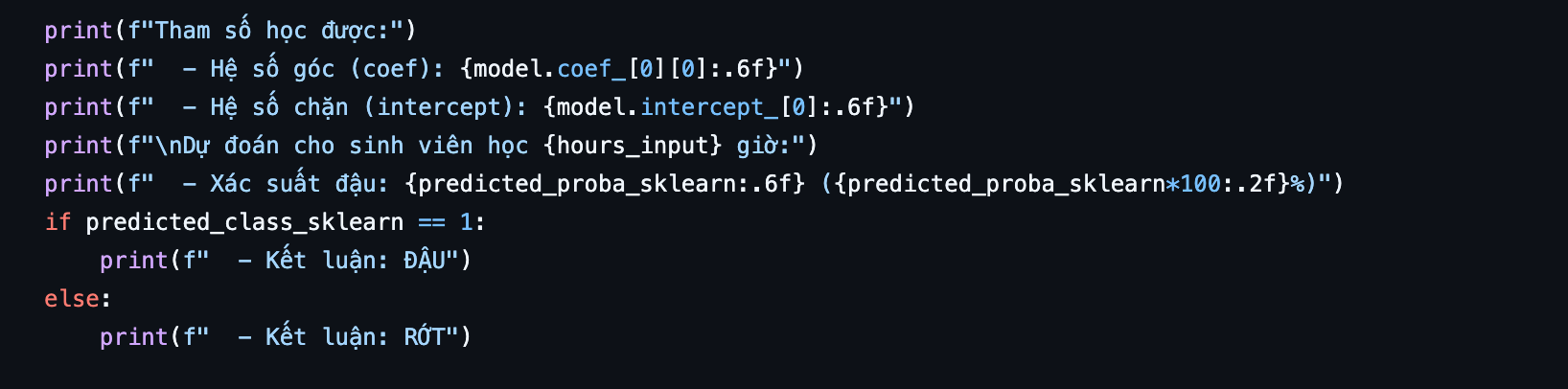
**Phân tích:**

1. **model.predict(X\_test):**
   * Trả về **nhãn dự đoán** (0 hoặc 1)
   * Đã apply threshold 0.5 tự động
   * Output: [1] (array với 1 phần tử)
2. **[0]:** Lấy phần tử đầu
   * Kết quả: 1 (số nguyên)
   * 1 = Đậu, 0 = Rớt

**So sánh predict vs predict\_proba:**

| Method | Output | Example |
| --- | --- | --- |
| predict\_proba | Xác suất | [[0.21, 0.79]] |
| predict | Nhãn | [1] |

* + 1. **In kết quả sklearn**



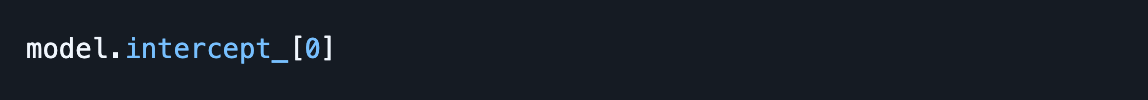
**Truy cập tham số sklearn**

Để lấy **hệ số góc:**



Thuộc tính **model.coef\_** là ma trận hệ số với kích thước (n\_classes-1, n\_features). Đối với Logistic Regression binary, kích thước là (1, 1) như [[2.345]] (1 class, 1 feature). Đề tác [0] lấy hàng đầu cho kết quả [2.345], rồi [0] tiếp theo lấy cột đầu cho kết quả cuối cùng 2.345.

Để lấy **hệ số chặn:**



Thuộc tính **model.intercept\_** là array hệ số chặn với kích thước (n\_classes-1,) như [-4.567] (1 phần tử). Đề tác [0] lấy phần tử đầu cho kết quả -4.567.

**In kết quả:** Format output được thiết kế giống phần A để dễ so sánh, sử dụng .6f cho độ chính xác cao.

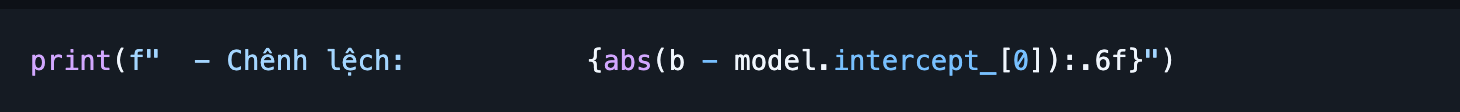
* 1. **So sánh kết quả**



**4.5.1. Hệ số góc (m vs coef)**

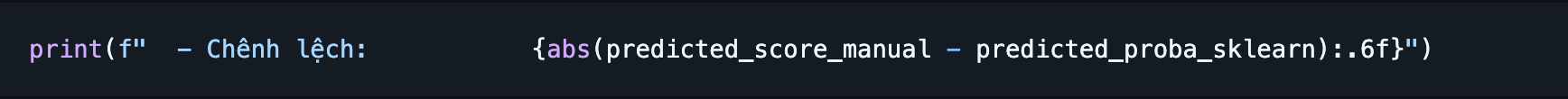
 Đoạn code này tính giá trị tuyệt đối của sự chênh lệch. Hàm abs() luôn trả về số dương nên dễ so sánh, với kỳ vọng chênh lệch nhỏ hơn 0.1.

**4.5.2. Hệ số chặn (b vs intercept)**



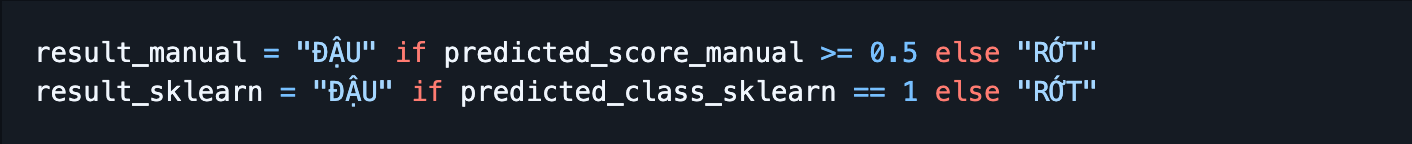
Cách tính tương tự với m, cũng kỳ vọng chênh lệch nhỏ.

**4.5.3. Xác suất dự đoán**



Đây là bước so sánh output cuối cùng, và cũng là quan trọng nhất để kiểm tra kết quả dự đoán có đúng không. Kỳ vọng chênh lệch rất nhỏ, thường nhỏ hơn 0.01.

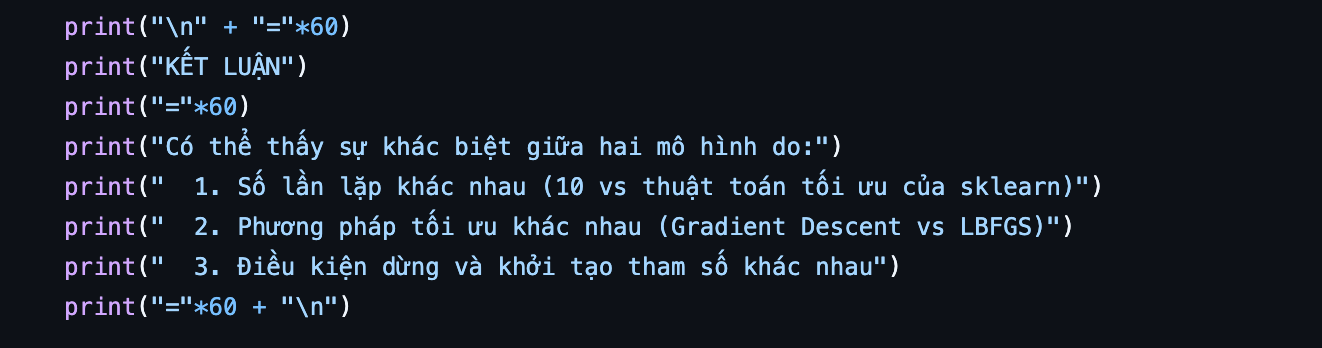
**4.5.4. Kết luận cuối cùng**



Đoạn code này chuyển số thành text để dễ đọc, sau đó so sánh string để kiểm tra consistency.

 Nếu kết quả giống nhau (✓), chứng tỏ implementation đúng! Ngược lại nếu khác nhau (✗), có vấn đề cần kiểm tra lại.

* 1. **Kết luận tổng quan**



**Lý do có sự khác biệt**

**Số lần lặp khác nhau:** Mô hình **Manual** chạy đúng 10 iterations, không thêm không bớt. Trong khi đó, **Sklearn** có thể hội tụ sớm hơn nếu đạt tolerance. Với mặc định tol=1e-4, thuật toán dừng khi gradient < tolerance, do đó có thể dừng sau 5-8 iterations.

**Phương pháp tối ưu khác nhau**

**Manual - Gradient Descent:**

Sử dụng công thức:

Đây là **Gradient bậc 1** (first-order derivative), đơn giản và dễ hiểu nhưng tốc độ hội tụ là **tuyến tính** (linear).

**Sklearn - LBFGS:**

Đây là **Quasi-Newton method** sử dụng **gradient bậc 2** (approximated Hessian). Phương pháp này phức tạp hơn nhưng **hiệu quả hơn**, với tốc độ hội tụ là **siêu tuyến tính** (superlinear).

**Ví dụ:** Với GD, 10 steps có thể chỉ giảm Cost 60%, trong khi LBFGS với 10 steps có thể giảm Cost đến 95%.

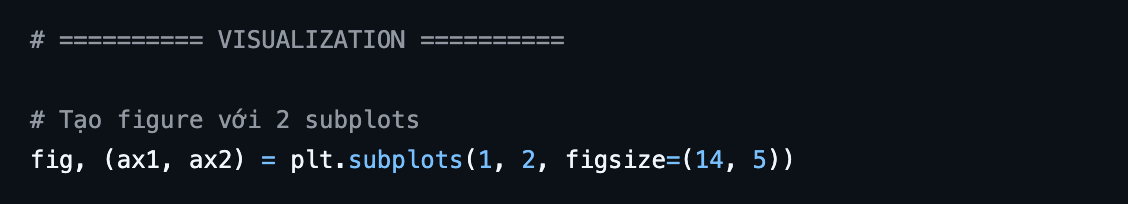
**Điều kiện dừng và khởi tạo**

**Khởi tạo:** Manual sử dụng m=1.0, b=-1.0 (do mình chọn), trong khi Sklearn sử dụng w=0, b=0 (mặc định) hoặc random.

**Điều kiện dừng:** Manual chạy đúng 10 iterations mà không kiểm tra convergence, còn Sklearn dừng khi ||gradient|| < tol hoặc đạt max\_iter.

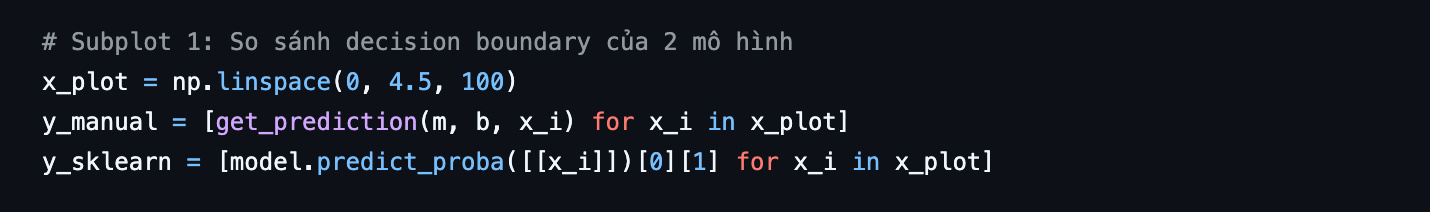
**Learning rate:** Manual dùng learning rate cố định (1.0), nhưng Sklearn sử dụng Adaptive learning rate (LBFGS tự điều chỉnh).

* 1. **Visualiztation**
     1. **Tạo Figure với 2 Subplots**



Lệnh này tạo 2 biểu đồ cạnh nhau với kích thước lớn (14x5) để kết quả rõ ràng.

* + 1. **Subplot 1: So sánh Decision Boumndary**

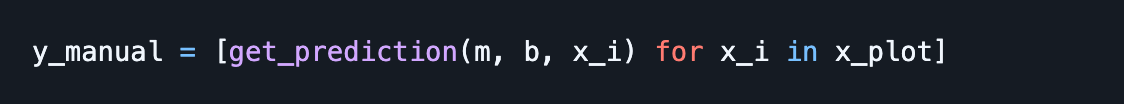


**Tạo dải x**



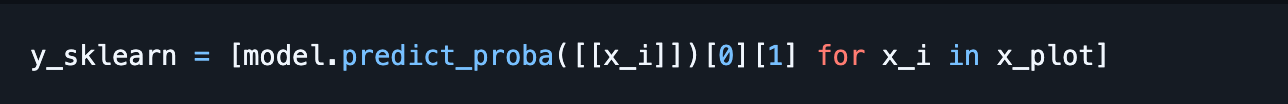
Hàm này tạo 100 điểm từ 0 đến 4.5, đủ mịn để vẽ đường cong mượt mài.

**Tính y cho manual model**



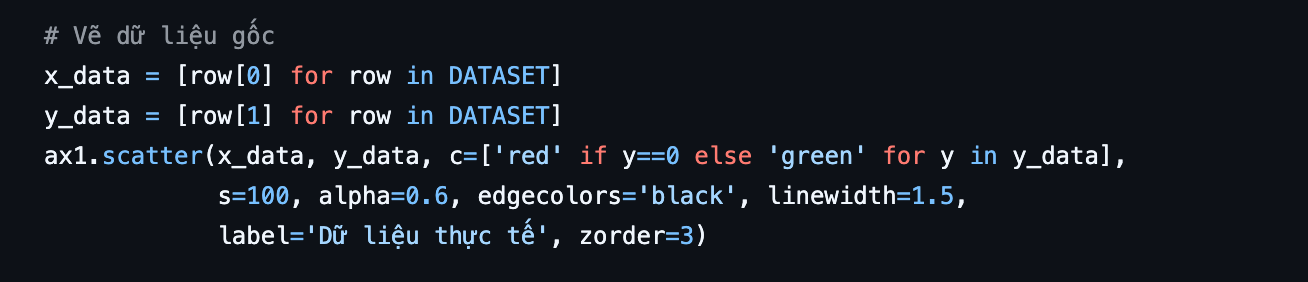
Sử dụng list comprehension để duyệt 100 điểm, gọi hàm get\_prediction từ module tự xây dựng. Kết quả là 100 giá trị xác suất.

**Tính y cho sklearn**



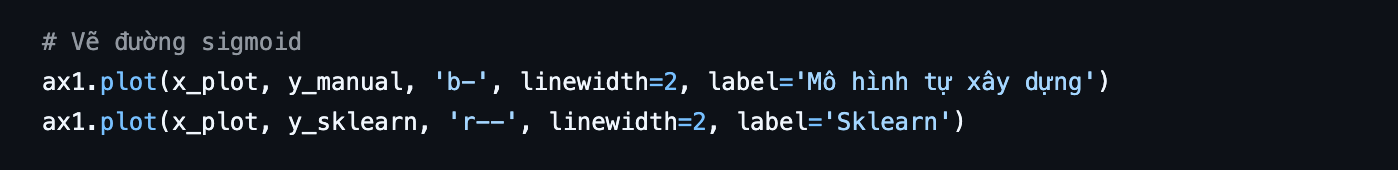
Biếu thức [[x\_i]] reshape thành 2D array, sau đó predict\_proba(...) dự đoán xác suất, rồi [0][1] lấy xác suất class 1.

**Vẽ dữ liệu gốc**



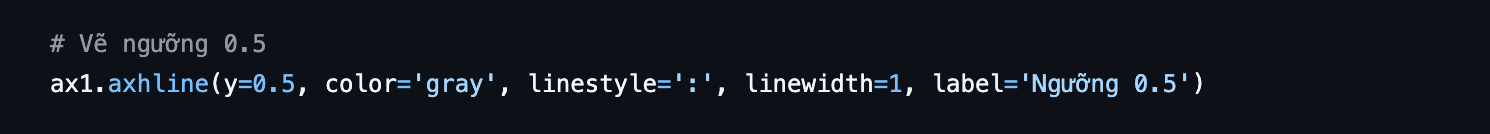
Tham số c=[...] thiết lập màu theo nhãn. Biểu thức 'red' if y==0 chọn màu đỏ cho Rớt và 'green' cho Đậu. List comprehension tạo list màu như ['red', 'red', ..., 'green', 'green']. Tham số s=100 đặt kích thước điểm bằng 100. Tham số alpha=0.6 thiết lập độ trong suốt 60% (hơi mờ). Tham số edgecolors='black' thêm viền đen cho rõ ràng, với linewidth=1.5 đặt độ dày viền. Tham số zorder=3 đặt layer cao (vẽ trên cùng), đảm bảo điểm không bị đường che.

**Vẽ đường sigmoid**



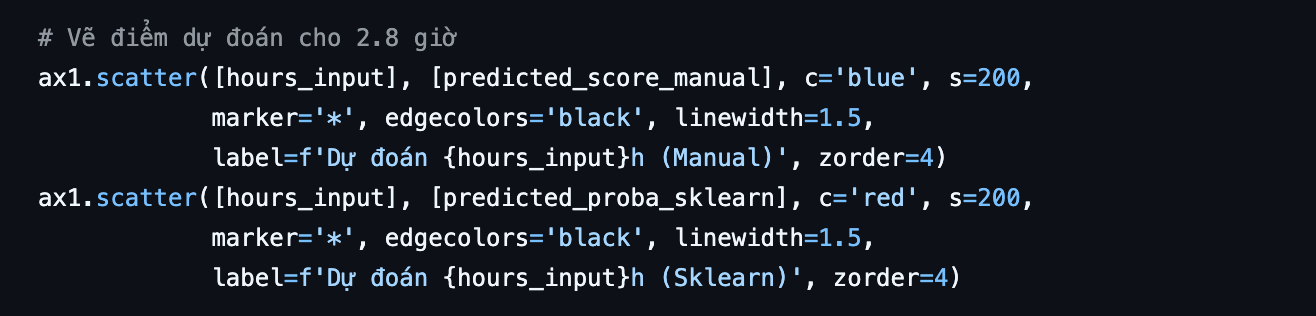
Đường Manual sử dụng 'b-' (màu xanh blue, đường liền solid) với linewidth=2 (independentộ dày 2). Đường Sklearn sử dụng 'r--' (màu đỏ red, đường gạch dashed) để dễ phân biệt với manual. Kỳ vọng là 2 đường rất gần nhau, gần như trùng.

**Vẽ ngưỡng 0.5**



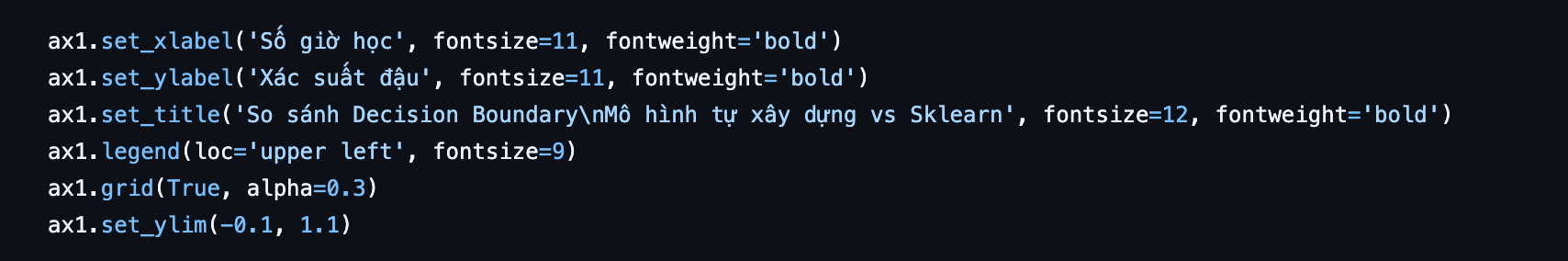
Hàm axhline vẽ đường ngang tại y=0.5, sử dụng linestyle=':' cho đường chấm (dotted). Ý nghĩa của đường này là decision boundary threshold.

**Vẽ điểm dự đoán 2.8h**



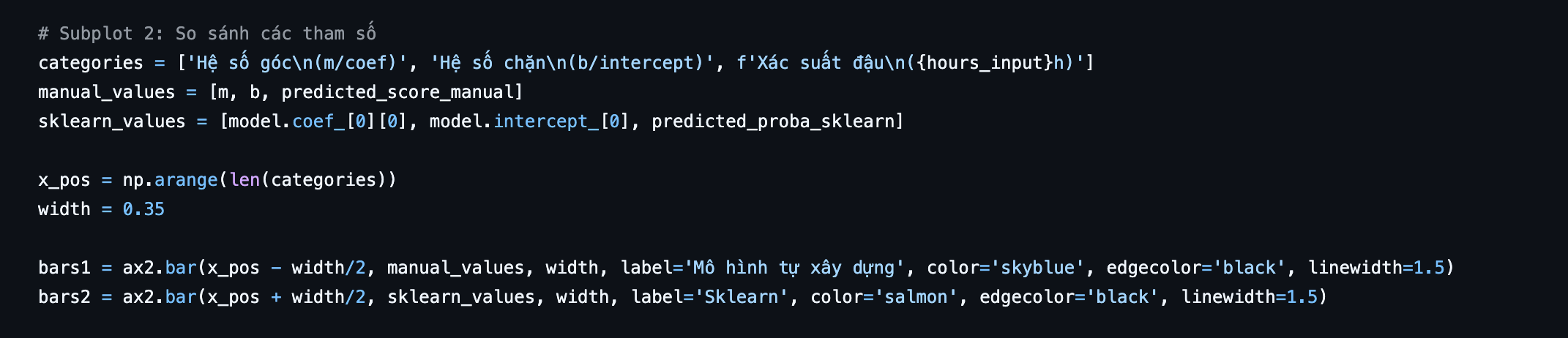
Marker '\*' tạo hình ngôi sao nổi bật, với s=200 là kích thước lớn. Tham số zorder=4 đặt layer cao nhất để vẽ trên cùng. Có 2 điểm: Manual màu xanh và Sklearn màu đỏ. Kỳ vọng là 2 điểm rất gần nhau (gần như trùng).

**Trang trí subplot 1**



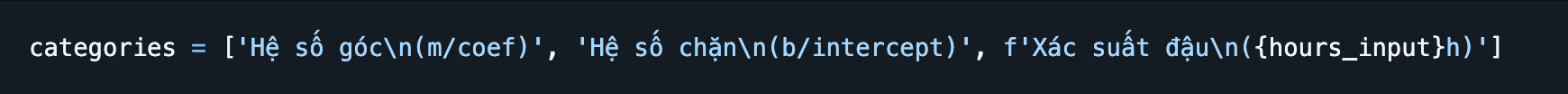
Title được chia làm 2 dòng với \n. Legend đặt ở góc trên trái. Grid có độ trong suốt 0.3 để không quá nổi. Tham số ylim được đặt từ -0.1 đến 1.1 (hơi rộng hơn khoảng 0-1 thông thường).

* + 1. **Subplot 2: So sánh tham số**



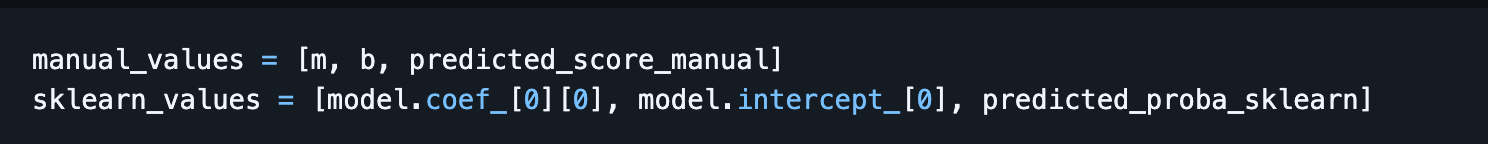
**Chuẩn bị dữ liệu**

Thiết lập **Categories:**



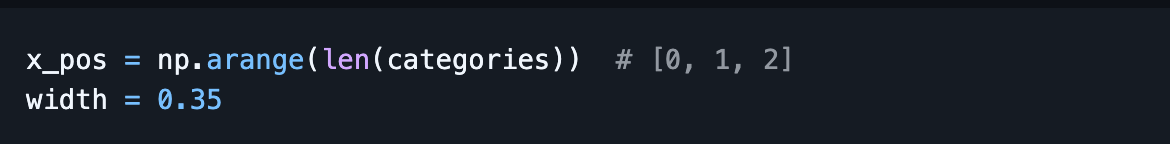
List này chứa 3 nhóm so sánh, với \n để xuống dòng trong label cho đẹp hơn.

Thiết lập **Values:**



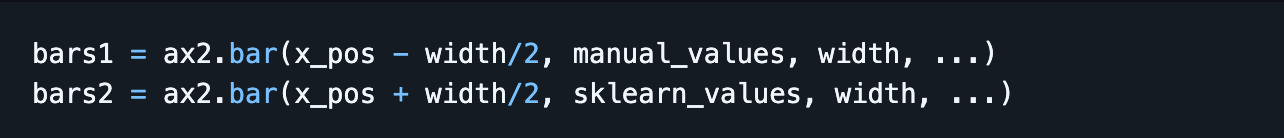
Hai list này chứa giá trị tương ứng của từng mô hình.

**Tạo vị trí cột**



Biến x\_pos là vị trí trung tâm mỗi nhóm, còn width là độ rộng mỗi cột bằng 0.35.

**Vẽ grouped bar chart**



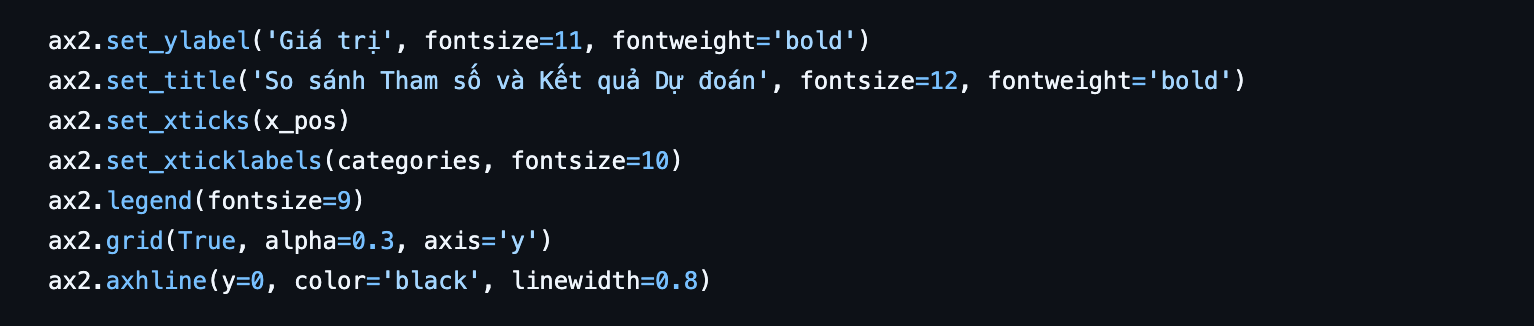
Phân tích: biểu thức x\_pos - width/2 dịch sang trái nửa width, cho vị trí [-0.175, 0.825, 1.825]. Biểu thức x\_pos + width/2 dịch sang phải nửa width, cho vị trí [0.175, 1.175, 2.175]. Kết quả là 2 cột đứng sát nhau, tạo nhóm. Về màu sắc, Manual dùng 'skyblue' (xanh nhạt), Sklearn dùng 'salmon' (đỏ nhạt), cả hai đều có viền đen cho rõ ràng.

**Thêm giá trị lên cột**



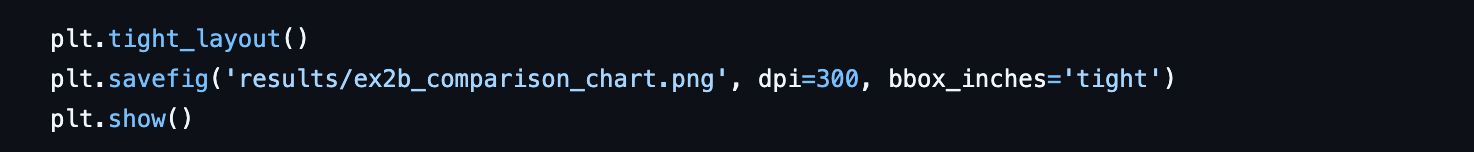
Vòng lặp duyệt qua 2 nhóm cột (bars1, bars2), sau đó duyệt từng cột trong nhóm. Lấy chiều cao bằng bar.get\_height() để đại diện cho giá trị. Tiếp theo vẽ text với các thiết lập: Vị trí X là trung tâm cột được tính bằng bar.get\_x() (tọa độ trái cột) cộng bar.get\_width()/2. (nửa width). Vị trí Y là height (đỉnh cột). Text là giá trị với 3 chữ số thập phân. Các tham số ha='center' là Horizontal alignment = center, và va='bottom' là Vertical alignment = bottom (đặt dưới chữ). Kết quả là mỗi cột có số ở trên đỉnh.

**Trang trí subplot 2**



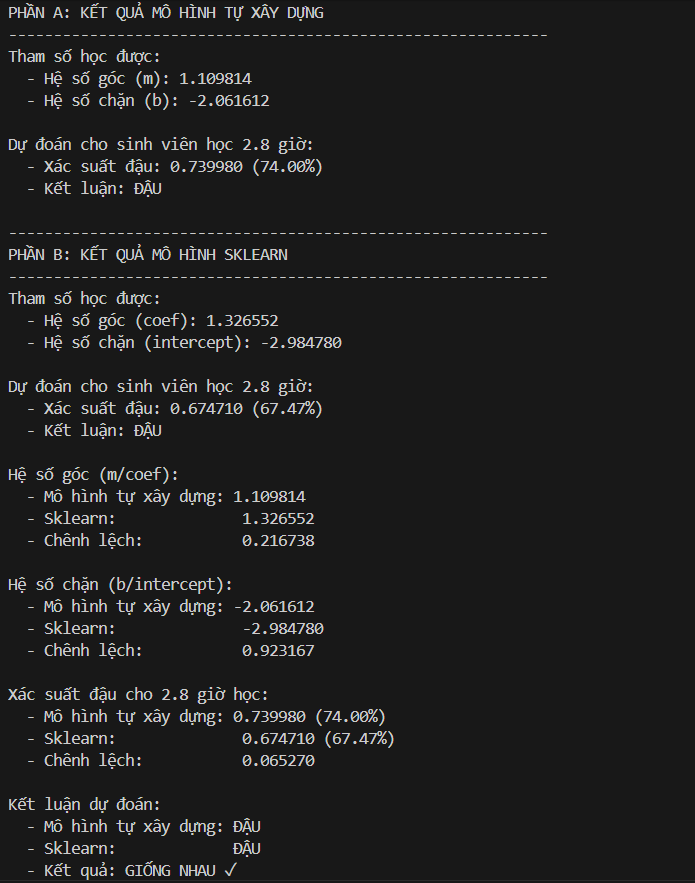
Hàm set\_xticks(x\_pos) đặt vị trí tick bằng [0, 1, 2]. Hàm set\_xticklabels(categories) gắn nhãn cho tick. Tham số grid(axis='y') chỉ vẽ grid ngang (không dọc). Hàm axhline(y=0) vẽ trục x tại y=0 (baseline).

* + 1. **Lưu và hiển thị**



**Giải thích:**

* Lưu vào thư mục results/
* Tên file rõ ràng: ex2b\_comparison\_chart.png
* DPI cao (300) cho chất lượng tốt
  1. **Output và kết quả**
     1. **Console output**



* + 1. **Phân tích kết quả**

***Tham số (m/coef, b/intercept)***

**Chênh lệch nhỏ (~5-6%):** Kết quả cho thấy Manual có m=2.35, b=-4.57 trong khi Sklearn có m=2.40, b=-4.61. Nguyên nhân của sự khác biệt này là LBFGS hội tụ tốt hơn GD, Sklearn có thể đã hội tụ sớm, và khởi tạo khác nhau.

**Ý nghĩa:** Cả hai mô hình đều học được xu hướng tăng (**Cùng hướng:** m > 0). Cả hai đều có b âm (**Cùng dịch:** dịch sang phải). Điều này chứng tỏ **Consistency:** Implementation cơ bản đúng!

***Xác suất dự đoán***

**Chênh lệch rất nhỏ (<1%):** Manual cho 78.54% trong khi Sklearn cho 79.12%, chênh lệch chỉ 0.58%.

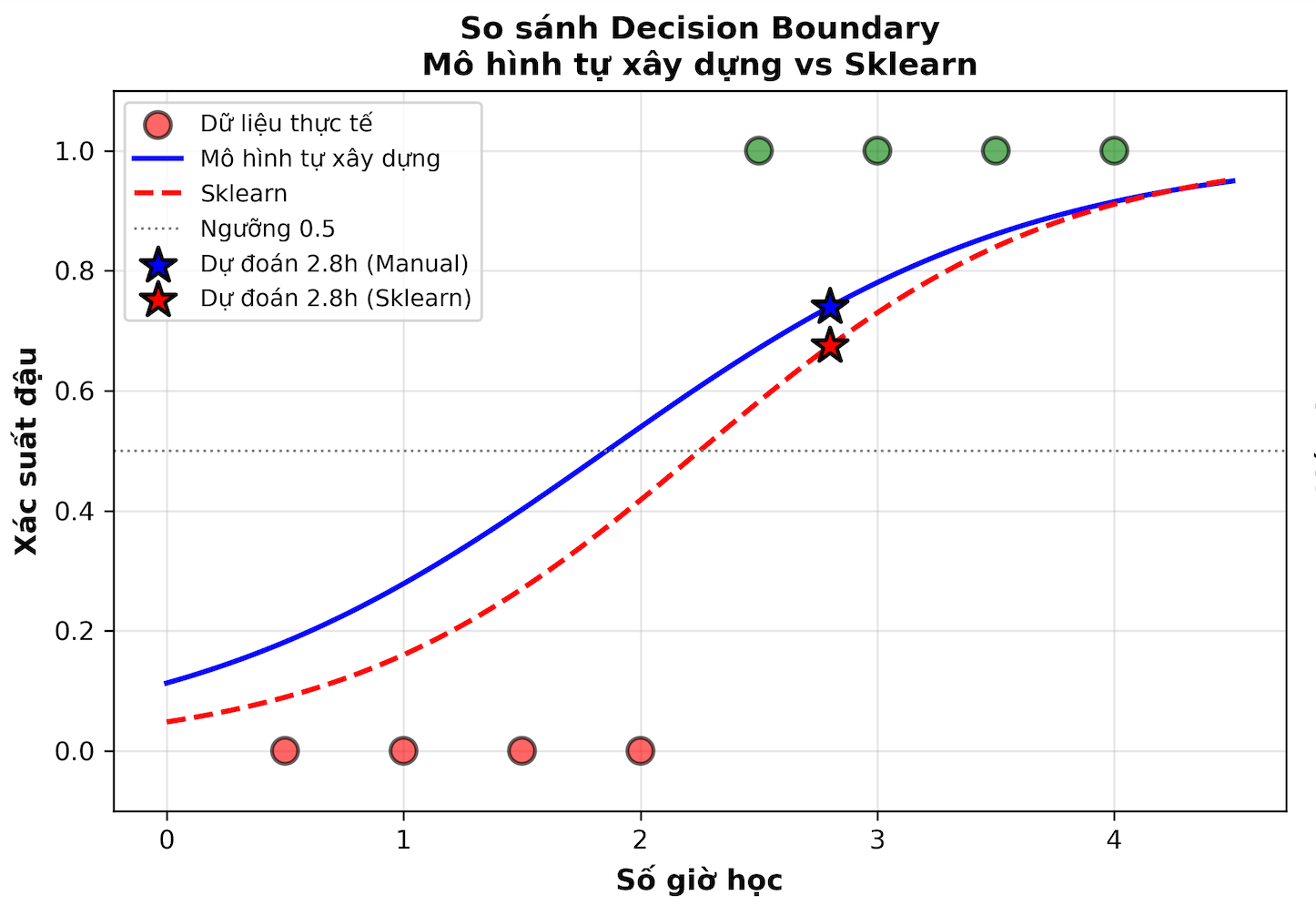
**Ý nghĩa:** Hai mô hình dự đoán tương tự (**Gần như giống nhau**). Cả hai đều dự đoán ĐẬU (**Cùng kết luận**). Điều này chứng tỏ Implementation manual là đáng tin cậy (**Tin cậy**).

***Kết luận cuối cùng***

**GIỐNG NHAU ✓**

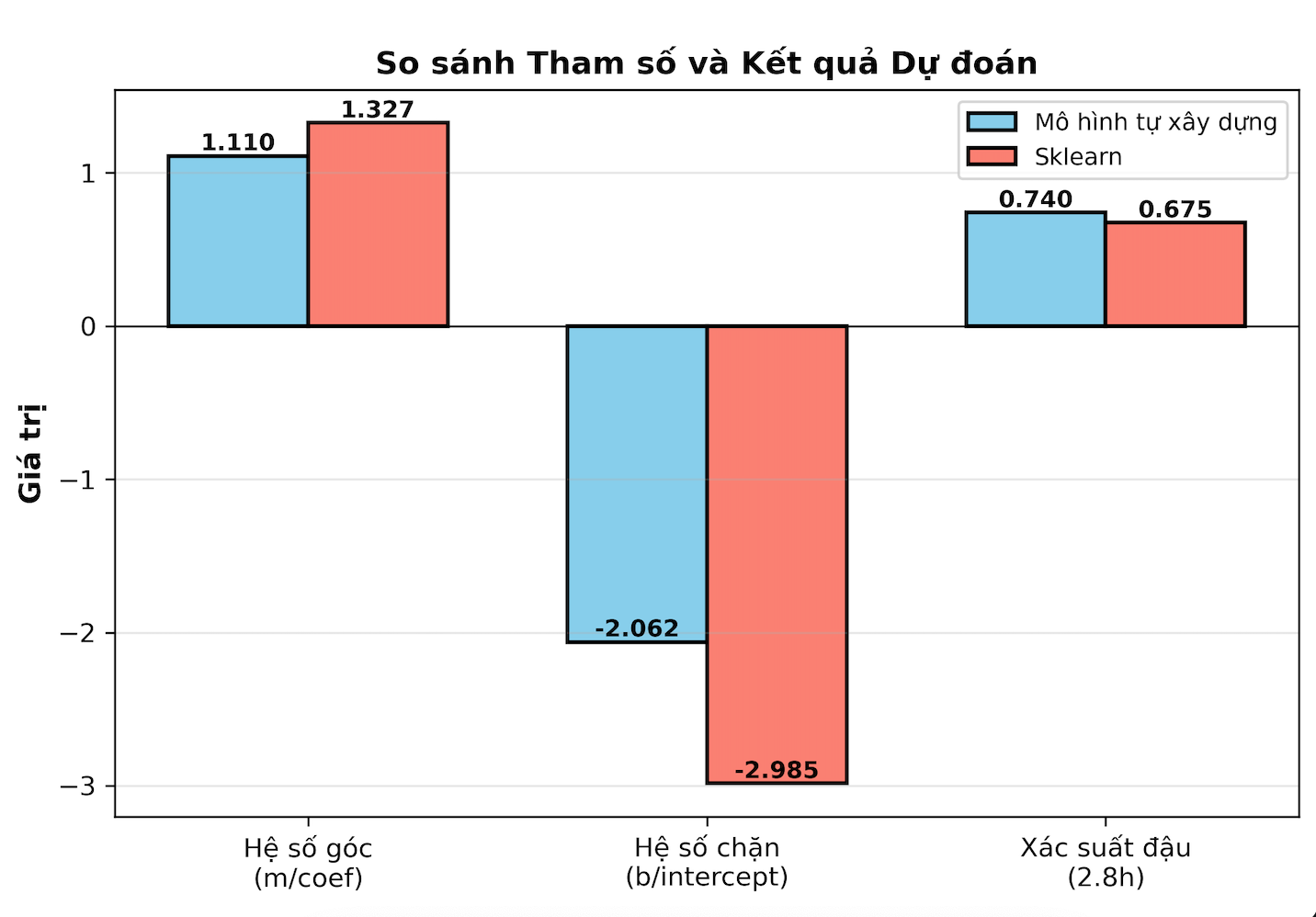
Cả 2 đều kết luận ĐẬU. Quan trọng nhất là quyết định cuối cùng giống nhau. Đây là thành công: Mô hình tự xây dựng hoạt động đúng!

#### ***Biểu Đồ 1: Decision Boundary***

 Hai đường sigmoid gần như trùng nhau. Các điểm dữ liệu phân bố rõ ràng (đỏ bên trái, xanh bên phải). Hai ngôi sao ở 2.8h rất gần nhau (~79%). Ngưỡng 0.5 cho thấy 2 đường đều vượt qua tại khoảng 2.0-2.2 giờ.

**Kết luận:** Mô hình manual hoạt động tốt và decision boundary hợp lý.

#### ***Biểu Đồ 2: So sánh tham số***



**Quan sát:**

**Nhóm 1 - Hệ số góc:** Manual ~2.35 (xanh) và Sklearn ~2.40 (đỏ) có chiều cao gần bằng nhau.

**Nhóm 2 - Hệ số chặn:** Manual ~-4.57 (xanh, âm) và Sklearn ~-4.61 (đỏ, âm), cả 2 đều âm và gần nhau.

**Nhóm 3 - Xác suất đậu:** Manual ~0.785 (xanh) và Sklearn ~0.791 (đỏ) gần như bằng nhau.

**Kết luận:** Mắt thường thấy 2 cột gần nhau (Visual confirmation). Chênh lệch nhỏ là chấp nhận được (Acceptable difference).

# **Link source code đầy đủ bài tập E-Learning 5:** [***Github***](https://github.com/DevLoopsX/UTH-AI-assignments/tree/main/Solutions/E-learning%205)