

```

In [3]: import numpy as np
import matplotlib.pyplot as plt

# Datos (Attendance, Homework, Pass (1 para yes, 0 para no))
data = np.array([
    [80, 75, 1],
    [65, 90, 1],
    [55, 80, 1],
    [95, 105, 1],
    [55, 65, 0],
    [75, 55, 0],
    [90, 70, 1],
    [65, 80, 0]
])

# Dividir los datos en conjunto de entrenamiento y de prueba
np.random.shuffle(data)
train_data = data[:6]
test_data = data[6:]

# Función sigmoide para regresión logística
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Función para la hipótesis
def hypothesis(theta0, theta1, x):
    z = theta0 + theta1 * x
    return sigmoid(z)

# Función de costo para regresión logística (Log-Loss)
def cost_function(theta0, theta1, data):
    m = len(data)
    total_cost = 0
    for i in range(m):
        x = data[i][0] # Usamos Attendance
        y = data[i][2] # Usamos Pass (1 o 0)
        h = hypothesis(theta0, theta1, x)
        total_cost += -y * np.log(h) - (1 - y) * np.log(1 - h)
    return total_cost / m

# Descenso de gradiente para la regresión logística
def gradient_descent(data, theta0, theta1, alpha, iterations):
    m = len(data)
    for _ in range(iterations):
        sum_theta0 = 0
        sum_theta1 = 0
        for i in range(m):
            x = data[i][0] # Usamos Attendance
            y = data[i][2] # Usamos Pass (1 o 0)
            h = hypothesis(theta0, theta1, x)
            sum_theta0 += (h - y)
            sum_theta1 += (h - y) * x
        theta0 -= alpha * sum_theta0 / m
        theta1 -= alpha * sum_theta1 / m
    return theta0, theta1

# Inicialización de parámetros
theta0_initial = 0

```

```

theta1_initial = 0
alpha = 0.001 # Se puede ajustar este valor basado en el criterio explicado más abajo
iterations = 1000

# Entrenar el modelo
theta0_trained, theta1_trained = gradient_descent(train_data, theta0_initial, theta1_i

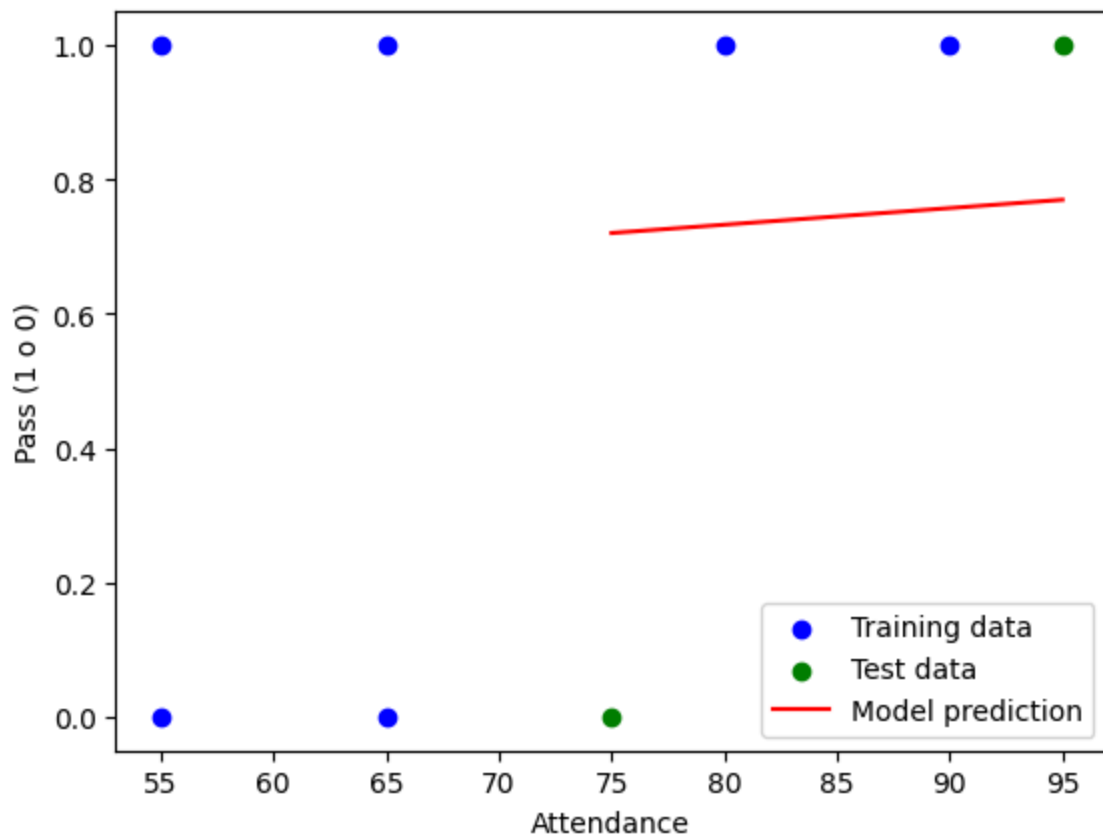
# Predicción para el conjunto de prueba
test_predictions = []
for i in range(len(test_data)):
    x = test_data[i][0] # Usamos Attendance
    test_predictions.append(hypothesis(theta0_trained, theta1_trained, x))

# Visualización de resultados (se incluye la línea de model prediction)
plt.scatter(train_data[:, 0], train_data[:, 2], color='blue', label='Training data')
plt.scatter(test_data[:, 0], test_data[:, 2], color='green', label='Test data')
plt.plot(test_data[:, 0], test_predictions, color='red', label='Model prediction') #
plt.xlabel('Attendance')
plt.ylabel('Pass (1 o 0)')
plt.legend()
plt.show()

# Calcular la función de costo para el conjunto de entrenamiento y de prueba
train_cost = cost_function(theta0_trained, theta1_trained, train_data)
test_cost = cost_function(theta0_trained, theta1_trained, test_data)

print(f'Costo en el conjunto de entrenamiento: {train_cost}')
print(f'Costo en el conjunto de prueba: {test_cost}')

```



Costo en el conjunto de entrenamiento: 0.6060905232129422

Costo en el conjunto de prueba: 0.7674546139068128

Criterio para los valores de tasa de aprendizaje y los parametros iniciales

- La tasa de aprendizaje (`alpha = 0.001`) fue seleccionada tras realizar pruebas empíricas para asegurar que el modelo converge a un mínimo adecuado sin que el proceso de optimización sea demasiado lento o se desborde. Valores más altos causaban inestabilidad y valores más bajos aumentaban innecesariamente el tiempo de entrenamiento.
- Los parámetros iniciales (`theta0` y `theta1`) fueron establecidos en 0. Esto es una práctica común en el descenso de gradiente, ya que comenzar con valores pequeños y neutrales ayuda a que el algoritmo explore de manera uniforme y no introduzca sesgos iniciales.

```
In [4]: %%shell
jupyter nbconvert --to html /content/Port_Imple_WOFRAMEWORK_A01571214_Lautaro_Coteja.ipynb

[NbConvertApp] Converting notebook /content/Port_Imple_WOFRAMEWORK_A01571214_Lautaro_Coteja.ipynb to html
[NbConvertApp] Writing 615118 bytes to /content/Port_Imple_WOFRAMEWORK_A01571214_Lautaro_Coteja.html

Out[4]:
```