# Week 3 – ML Model Development (Logistic Regression From Scratch)

This week, we train a machine learning model using the cleaned dataset.
We implement **Logistic Regression from scratch using NumPy** to understand how the algorithm works internally and compare its performance with Scikit-Learn's LogisticRegression.

## Student Information

**Name: Vishal Baraiya**
**Enrollment No.: 23010101014**
**Roll No.: C3-635**
**Course:** Machine Learning & Deep Learning Project

---

# Objectives of Week 3

- Train a machine learning model on the cleaned dataset.
- Implement Logistic Regression from scratch using NumPy.
- Evaluate the model using accuracy, precision, recall, F1 score, and ROC-AUC.
- Compare the scratch model with Scikit-Learn's LogisticRegression.
- Save the trained model weights and bias for Week-4 Flask deployment.

# 1. Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_sc
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
```

# 2. Load Dataset

In [2]:
```
df = pd.read_csv("../data/processed/clean_cardio.csv")
df.head()
```

Out[2]:

| | age | gender | height | weight | ap_hi | ap_lo | cholesterol | gluc | smoke | alco | a |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 18393 | 2 | 168 | 62.0 | -0.998967 | -0.138532 | 1 | 1 | 0 | 0 | |
| **1** | 20228 | 1 | 156 | 85.0 | 0.797881 | 0.919885 | 3 | 1 | 0 | 0 | |
| **2** | 18857 | 1 | 165 | 64.0 | 0.198932 | -1.196948 | 3 | 1 | 0 | 0 | |
| **3** | 17623 | 2 | 169 | 82.0 | 1.396830 | 1.978301 | 1 | 1 | 0 | 0 | |
| **4** | 17474 | 1 | 156 | 56.0 | -1.597916 | -2.255364 | 1 | 1 | 0 | 0 | |

# 3. Define Features & Target

## We select the relevant features for modeling.

The target variable is **cardio** (0 = No disease, 1 = Disease).

In [3]:

```
feature_cols = [
    # Core numeric (scaled)
    'age_years', 'ap_hi', 'ap_lo', 'bmi',
    'cholesterol', 'gluc',

    # Binary lifestyle (NOT scaled)
    'smoke', 'alco', 'active',

    # Interaction terms (scaled)
    'smoke_age', 'smoke_bmi',
    'alco_age', 'alco_bmi'
]

X = df[feature_cols].values
y = df['cardio'].values
```

# 4. Split Dataset

In [4]:

```
x_train, x_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

x_train.shape, x_test.shape
```

Out[4]:

```
((54844, 13), (13712, 13))
```

## Choosing the Appropriate Algorithm

The target variable **cardio** is binary (0 = no disease, 1 = disease).
Since we want to predict the probability of a patient having cardiovascular disease based on

multiple numerical and categorical features, this becomes a **binary classification problem**.

For this type of problem, **Logistic Regression** is one of the most suitable and widely used algorithms because:

- It models the probability of a binary outcome.
- It works well with medical datasets and interpretable features.
- It handles linear decision boundaries effectively.
- It allows us to analyze feature importance easily.
- It performs well when data is properly scaled (as in Week 2).

Therefore, we choose **Logistic Regression** as the appropriate algorithm and implement it **from scratch using NumPy**, as required.

# 5. Logistic Regression From Scratch

We implement:

- Sigmoid function
- Loss (Binary Cross-Entropy)
- Gradient computation
- Weight & bias update

In [5]:
```
class LogisticRegressionScratch:

    def __init__(self, lr=0.01, n_iters=2000):
        self.lr = lr
        self.n_iters = n_iters

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, x, y):
        n_samples, n_features = x.shape
        self.weights = np.zeros(n_features)
        self.bias = 0
        self.loss_history = []

        pos_weight = np.sum(y == 0) / np.sum(y == 1)

        for _ in range(self.n_iters):
            linear_model = np.dot(x, self.weights) + self.bias
            y_pred = self.sigmoid(linear_model)

            error = y_pred - y
            weighted_error = error * (y * pos_weight + (1 - y))

            dw = (1/n_samples) * np.dot(x.T, weighted_error)
            db = (1/n_samples) * np.sum(weighted_error)
```

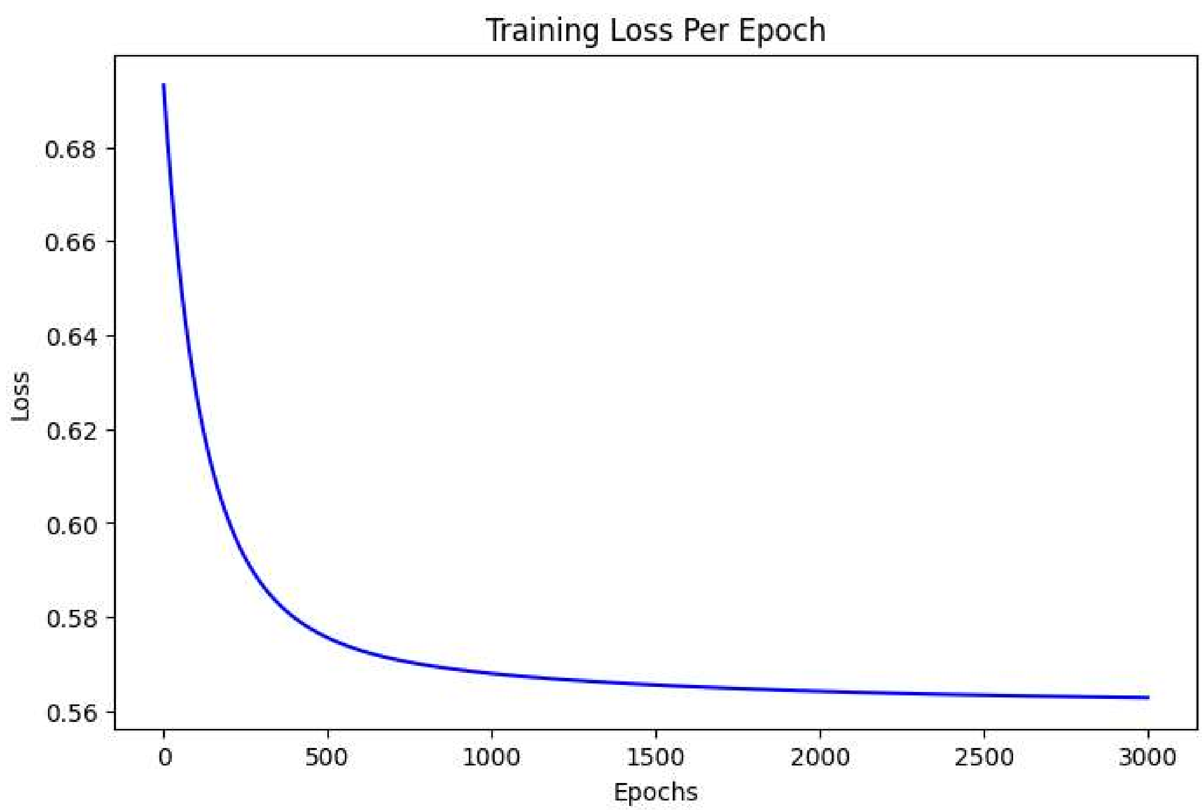
```
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

            loss = -np.mean(
                y*np.log(y_pred + 1e-8) + (1 - y)*np.log(1 - y_pred + 1e-8)
            )
            self.loss_history.append(loss)

    def predict_proba(self, x):
        return self.sigmoid(np.dot(x, self.weights) + self.bias)

    def predict(self, x, threshold=0.5):
        return (self.predict_proba(x) >= threshold).astype(int)
```

In [6]:
```
model_scratch = LogisticRegressionScratch(lr=0.01, n_iters=3000)
model_scratch.fit(x_train, y_train)
print("Training complete!")
```

```
Training complete!
```

# 6. Plot Training Loss

In [7]:
```
plt.figure(figsize=(8,5))
plt.plot(model_scratch.loss_history, color='blue')
plt.title("Training Loss Per Epoch")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```



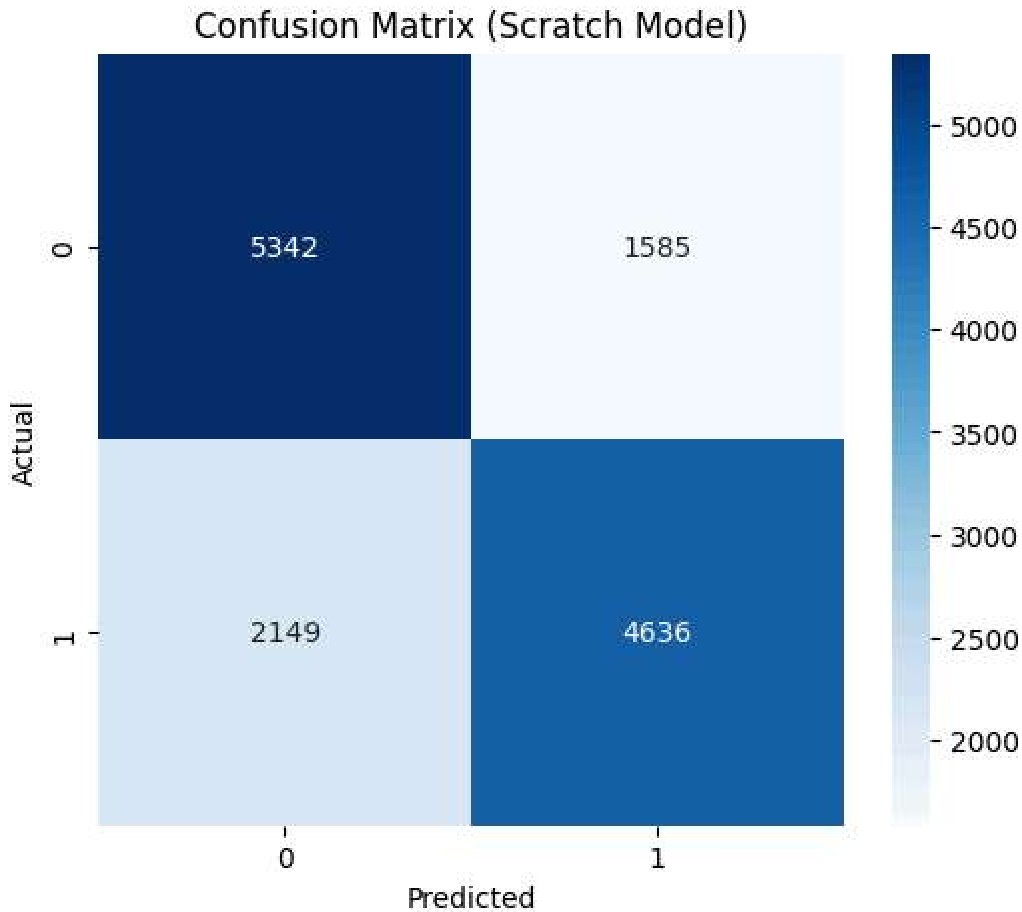
# 7. Evaluate Scratch Model

In [8]:
```
y_pred = model_scratch.predict(x_test)
y_proba = model_scratch.predict_proba(x_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
```

```
Accuracy: 0.727683780630105
Precision: 0.7452178106413759
Recall: 0.6832719233603537
F1 Score: 0.7129017376595418
```

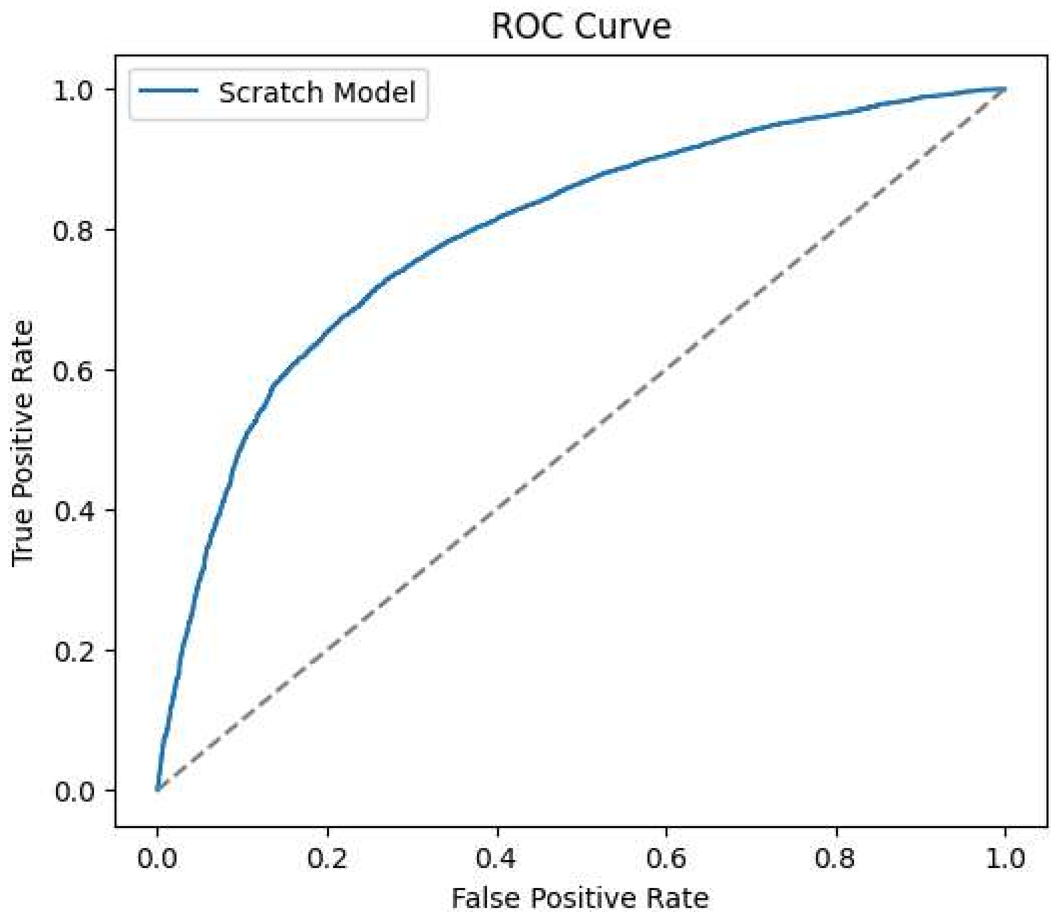
# 8. Confusion Matrix

In [9]:
```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix (Scratch Model)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```




# 9. ROC Curve

In [10]:
```
fpr, tpr, _ = roc_curve(y_test, y_proba)

plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label="Scratch Model")
plt.plot([0,1], [0,1], linestyle='--', color='gray')
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()

print("AUC Score:", roc_auc_score(y_test, y_proba))
```




```
AUC Score: 0.7931182532141964
```

# 10. Compare with sklearn LogisticRegression

In [11]:

```
sk_model = LogisticRegression(max_iter=5000)
sk_model.fit(x_train, y_train)

sk_pred = sk_model.predict(x_test)
sk_proba = sk_model.predict_proba(x_test)[:,1]

print("Accuracy:", accuracy_score(y_test, sk_pred))
print("Precision:", precision_score(y_test, sk_pred))
print("Recall:", recall_score(y_test, sk_pred))
print("F1 Score:", f1_score(y_test, sk_pred))
print("AUC:", roc_auc_score(y_test, sk_proba))
```

```
Accuracy: 0.7276108518086347
Precision: 0.7539127539127539
Recall: 0.6673544583640383
F1 Score: 0.7079978109608318
AUC: 0.794238664740271
```

## 10.1 Cross-validation & Hyperparameter Search

Evaluate `LogisticRegression` with stratified K-fold cross-validation and tune key hyperparameters (C, class_weight, solver).

In [14]:
```
# Stratified 10-fold CV accuracy for a baseline LogisticRegression
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
base_lr = LogisticRegression(max_iter=5000)
cv_scores = cross_val_score(base_lr, X, y, cv=cv, scoring="accuracy", n_jobs=-1)
print(f"CV accuracy: {cv_scores.mean():.3f} +/- {cv_scores.std():.3f}")

# Hyperparameter search over C and class_weight
param_grid = {
    "C": [0.01, 0.1, 1, 10],
    "class_weight": [None, "balanced"],
    "solver": ["lbfgs"],
    "penalty": ["l2"],
    "max_iter": [5000]
}

search = GridSearchCV(
    estimator=LogisticRegression(),
    param_grid=param_grid,
    cv=cv,
    scoring="accuracy",
    n_jobs=-1,
    verbose=0
)
search.fit(X, y)

best_lr = search.best_estimator_
print("Best params:", search.best_params_)
print(f"Best CV accuracy: {search.best_score_:.3f}")

# Evaluate the tuned model on the held-out test split
best_test_pred = best_lr.predict(x_test)
print("Test accuracy (best model):", accuracy_score(y_test, best_test_pred))
```

```
CV accuracy: 0.728 +/- 0.004
Best params: {'C': 0.01, 'class_weight': 'balanced', 'max_iter': 5000, 'penalty': 'l
2', 'solver': 'lbfgs'}
Best CV accuracy: 0.728
Test accuracy (best model): 0.7292882147024504
```

# 11. Save Model Parameters

These weights will be used in **Week-4 Flask App**.

In [13]:
```
import os
import numpy as np

os.makedirs("../models", exist_ok=True)

np.save("../models/logistic_weights.npy", model_scratch.weights)
np.save("../models/logistic_bias.npy", np.array([model_scratch.bias]))
```

```
print("Scratch model parameters saved successfully!")
```

```
Scratch model parameters saved successfully!
```

# Week 3 Completed Successfully

- Loaded cleaned dataset
- Performed train-test split
- Implemented **Logistic Regression from scratch** using NumPy
- Trained model with gradient descent
- Visualized loss curve
- Evaluated model using multiple metrics
- Drew confusion matrix & ROC curve
- Compared results with sklearn model
- Saved model weights for deployment

In [ ]:

```
```