Logistic Regression From Scratch - Python Code

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

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

def initialize_parameters(n_features):
    W = np.zeros((n_features, 1))
    b = 0.0
    return W, b

def compute_cost_and_predictions(X, y, W, b):
    m = X.shape[0]
    Z = np.dot(X, W) + b
    A = sigmoid(Z)
    eps = 1e-15
    A = np.clip(A, eps, 1 - eps)
    cost = -(1 / m) * np.sum(y * np.log(A) + (1 - y) * np.log(1 - A))
    return cost, A

def compute_gradients(X, y, A):
    m = X.shape[0]
    dZ = A - y
    dW = (1 / m) * np.dot(X.T, dZ)
    db = (1 / m) * np.sum(dZ)
    return dW, db

def train_logistic_regression(X, y, learning_rate=0.01, num_iterations=1000):
    m, n = X.shape
    y = y.reshape(m, 1)
    W, b = initialize_parameters(n)
    cost_history = []

    for i in range(num_iterations):
        cost, A = compute_cost_and_predictions(X, y, W, b)
        dW, db = compute_gradients(X, y, A)
        W -= learning_rate * dW
        b -= learning_rate * db
        cost_history.append(cost)

    return W, b, cost_history

def predict(X, W, b, threshold=0.5):
    Z = np.dot(X, W) + b
    A = sigmoid(Z)
    return (A >= threshold).astype(int).flatten()

# Generate dataset
X, y = make_classification(
    n_samples=500,
    n_features=5,
    n_informative=3,
    n_redundant=0,
    class_sep=1.5,
    random_state=42
)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Train custom logistic regression
W, b, cost_history = train_logistic_regression(X_train, y_train, learning_rate=0.01, num_iterations=1000

# Predictions
y_pred = predict(X_test, W, b)

# Metrics
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Custom Model Metrics:")
print("Accuracy:", acc)
print("Precision:", prec)
print("Recall:", rec)
print("F1 Score:", f1)

# Sklearn comparison
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred_sk = model.predict(X_test)

acc_sk = accuracy_score(y_test, y_pred_sk)
prec_sk = precision_score(y_test, y_pred_sk)
rec_sk = recall_score(y_test, y_pred_sk)
f1_sk = f1_score(y_test, y_pred_sk)

print("\nSklearn Model Metrics:")
print("Accuracy:", acc_sk)
print("Precision:", prec_sk)
print("Recall:", rec_sk)
print("F1 Score:", f1_sk)

# Save cost history plot
plt.plot(range(len(cost_history)), cost_history)
plt.xlabel("Iteration")
plt.ylabel("Cost")
plt.title("Cost vs Iteration")
plt.savefig("/mnt/data/cost_plot.png")
```