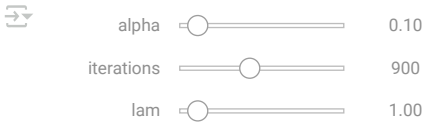
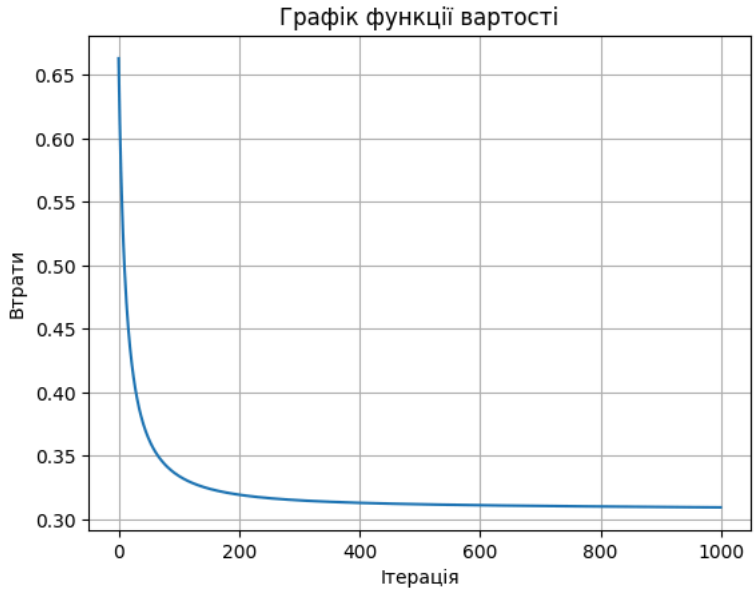
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
from ipywidgets import interact, FloatSlider, IntSlider



url = "https://archive.ics.uci.edu/ml/machine-learning-databases/credit-screening/crx.data"
columns = [f"feature_{i}" for i in range(15)] + ["class"]
data = pd.read_csv(url, names=columns, na_values='?')
data.dropna(inplace=True)

for i in range(15):
    if data[f"feature_{i}"].dtype == 'object':
        le = LabelEncoder()
        data[f"feature_{i}"] = le.fit_transform(data[f"feature_{i}"])

X = data.iloc[:, :-1].values
y = (data['class'] == '+').astype(int).values

# Масштабування
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Розділення даних
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



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

def compute_cost(X, y, theta, lam=0):
    m = len(y)
    h = sigmoid(X @ theta)
    cost = (-y @ np.log(h) - (1 - y) @ np.log(1 - h)) / m
    reg = lam / (2 * m) * np.sum(np.square(theta[1:]))
    return cost + reg

def gradient(X, y, theta, lam=0):
    m = len(y)
    h = sigmoid(X @ theta)
    grad = (X.T @ (h - y)) / m
    grad[1:] += (lam / m) * theta[1:]
    return grad

def gradient_descent(X, y, alpha, num_iters, lam=0):
    theta = np.zeros(X.shape[1])
    cost_history = []

    for _ in range(num_iters):
        grad = gradient(X, y, theta, lam)
        theta -= alpha * grad
        cost_history.append(compute_cost(X, y, theta, lam))

    return theta, cost_history



# Додавання одиничного стовпця
X_train_bias = np.c_[np.ones(X_train.shape[0]), X_train]
X_test_bias = np.c_[np.ones(X_test.shape[0]), X_test]

theta_opt, cost_history = gradient_descent(X_train_bias, y_train, alpha=0.1, num_iters=1000, lam=1)

y_pred_train = sigmoid(X_train_bias @ theta_opt) >= 0.5
y_pred_test = sigmoid(X_test_bias @ theta_opt) >= 0.5

print("Train Accuracy:", accuracy_score(y_train, y_pred_train))
print("Test Accuracy:", accuracy_score(y_test, y_pred_test))
print("Precision:", precision_score(y_test, y_pred_test))
print("Recall:", recall_score(y_test, y_pred_test))
print("F1 Score:", f1_score(y_test, y_pred_test))
```

```
Train Accuracy: 0.8908045977011494
Test Accuracy: 0.8396946564885496
Precision: 0.7741935483870968
Recall: 0.8727272727272727
```

```
F1 Score: 0.8205128205128205
```

```
def train_and_plot(alpha, iterations, lam):
    theta, cost = gradient_descent(X_train_bias, y_train, alpha, iterations, lam)
    plt.plot(cost)
    plt.title("Графік функції вартості")
    plt.xlabel("Ітерація")
    plt.ylabel("Втрати")
    plt.grid(True)
    plt.show()

interact(train_and_plot,
         alpha=FloatSlider(min=0.001, max=1, step=0.01, value=0.1),
         iterations=IntSlider(min=100, max=2000, step=100, value=1000),
         lam=FloatSlider(min=0, max=10, step=0.1, value=1))
```






```
train_and_plot
def train_and_plot(alpha, iterations, lam)
```

```
<no docstring>
```