```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

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 = data.dropna()

# Кодування категоріальних змінних
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}"])

# Поділ даних на ознаки та мітки
X = data.iloc[:, :-1].values
y = (data['class'] == '+').astype(int)

# Поділ на тренувальну та тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)



import numpy as np

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



def predict(X, theta):
    return sigmoid(np.dot(X, theta))



def cost_function(X, y, theta, lambda_):
    m = len(y)
    h = predict(X, theta)
    cost = (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
    reg_cost = (lambda_ / (2 * m)) * np.sum(np.square(theta[1:]))
    return cost + reg_cost



def gradient(X, y, theta, lambda_):
    m = len(y)
    h = predict(X, theta)
    grad = (1 / m) * np.dot(X.T, (h - y))
    grad[1:] += (lambda_ / m) * theta[1:]
    return grad



def gradient_descent(X, y, theta, alpha, iterations, lambda_):
    cost_history = []
    for _ in range(iterations):
        grad = gradient(X, y, theta, lambda_)
        theta -= alpha * grad
        cost_history.append(cost_function(X, y, theta, lambda_))
    return theta, cost_history



import ipywidgets as widgets
from IPython.display import display
import matplotlib.pyplot as plt

# Створення віджетів для взаємодії з користувачем
alpha_slider = widgets.FloatSlider(value=0.01, min=0.001, max=0.1, step=0.001, description='Alpha:')
iterations_slider = widgets.IntSlider(value=1000, min=100, max=5000, step=100, description='Iterations:')
lambda_slider = widgets.FloatSlider(value=0.1, min=0, max=1, step=0.1, description='Lambda:')

def update_model(alpha, iterations, lambda_):
    # ініціалізація параметрів
    theta_init = np.zeros(X_train.shape[1])

    # Застосування градієнтного спуску
    theta, cost_history = gradient_descent(X_train, y_train, theta_init, alpha, iterations, lambda_)

    # Візуалізація кривої вартості
    plt.plot(range(len(cost_history)), cost_history, label="Cost function")
    plt.xlabel('Iterations')
```

```
    plt.ylabel('Cost')
    plt.title('Learning curve')
    plt.show()

# Відображення віджетів
widgets.interactive(update_model, alpha=alpha_slider, iterations=iterations_slider, lambda_=lambda_slider)
```

Показати прихований результат

```
# Ініціалізація початкових значень для theta
theta_init = np.zeros(X_train.shape[1])

# Параметри для градієнтного спуску
alpha = 0.01  # Швидкість навчання
iterations = 1000  # Кількість ітерацій
lambda_ = 0.1  # Параметр регуляризації

# Виконання градієнтного спуску
theta_optimal, cost_history = gradient_descent(X_train, y_train, theta_init, alpha, iterations, lambda_)

# Оцінка моделі з використанням оптимальних параметрів theta
accuracy, precision, recall, f1 = evaluate_model(X_test, y_test, theta_optimal)

# Виведення результатів оцінки
print(f"Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1-score: {f1}")
```

Показати прихований результат

```
from sklearn.decomposition import PCA

# Використовуємо PCA для зменшення розмірності до 2 ознак
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)

# Ініціалізація параметрів
theta_init = np.zeros(X_train_pca.shape[1])

# Параметри для градієнтного спуску
alpha = 0.01  # Швидкість навчання
iterations = 1000  # Кількість ітерацій
lambda_ = 0.1  # Параметр регуляризації

# Навчання моделі на зменшених даних
theta_optimal, cost_history = gradient_descent(X_train_pca, y_train, theta_init, alpha, iterations, lambda_)

# Візуалізація межі прийняття рішень на зменшених даних
plot_decision_boundary(X_train_pca, y_train, theta_optimal)
```

Показати прихований результат