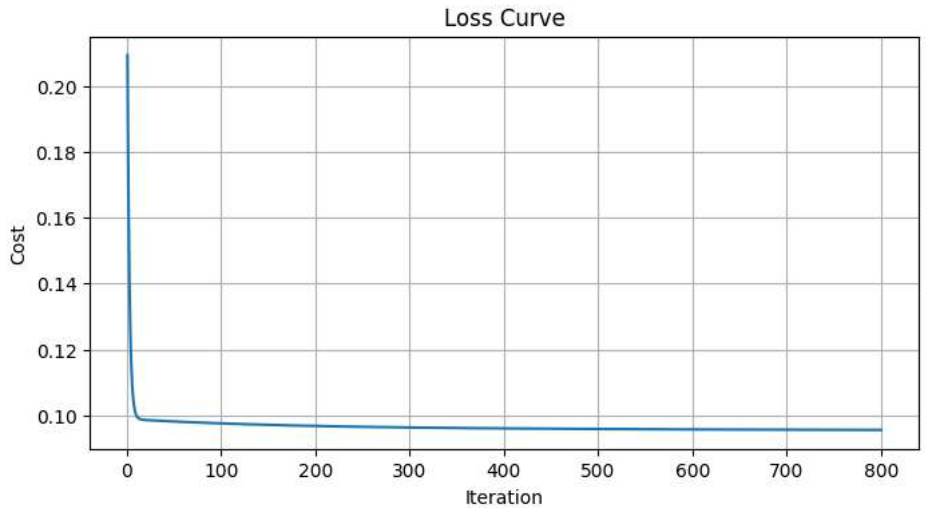
Practical_Assignment 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from datetime import datetime

df = pd.read_csv("/content/employee_promotions.csv")

df['date_of_birth'] = pd.to_datetime(df['date_of_birth'], format="%d-%m-%Y")
df['date_of_joining'] = pd.to_datetime(df['date_of_joining'], format="%d-%m-%Y")

today = datetime.today()

df['age'] = (today - df['date_of_birth']).dt.days / 365.25
df['years_service'] = (today - df['date_of_joining']).dt.days / 365.25

X = df[['gender', 'age', 'years_service']].values
y_cont = df['promoted'].values
y_class = y_cont.copy()
multiclass = False

m = X.shape[0]
X_bias = np.c_[np.ones((m, 1)), X]

def hypothesis(theta_array, x_row):
    return float(np.dot(x_row, theta_array))

def Cost_Function(theta_array, X, y, m):
    error = 0.0
    for i in range(m):
        error += (hypothesis(theta_array, X[i]) - y[i]) ** 2
    return error / (2 * m)

def Gradient_Descent(theta_array, X, y, m, alpha):
    summation = np.zeros(len(theta_array))
    for i in range(m):
        pred = hypothesis(theta_array, X[i])
        summation += (pred - y[i]) * X[i]
    new_theta = theta_array - (alpha / m) * summation
    return new_theta

def Training(X, y, alpha, iters):
    theta_array = np.zeros(X.shape[1])
    cost_values = []
    m = X.shape[0]
    for _ in range(iters):
        theta_array = Gradient_Descent(theta_array, X, y, m, alpha)
        cost_values.append(Cost_Function(theta_array, X, y, m))
    return theta_array, cost_values

alpha = 1e-4
iters = 800
theta_final, cost_values = Training(X_bias, y_cont, alpha, iters)

plt.figure(figsize=(8,4))
plt.plot(np.arange(1, len(cost_values)+1), cost_values, '-')
plt.xlabel('Iteration')
plt.ylabel('Cost')
plt.title('Loss Curve')
plt.grid(True)
plt.show()

y_pred_cont = np.array([hypothesis(theta_final, X_bias[i]) for i in range(m)])

thresh = 0.5
y_pred_class = (y_pred_cont >= thresh).astype(int)

print("Accuracy:", accuracy_score(y_class, y_pred_class))
print("Confusion Matrix:\n", confusion_matrix(y_class, y_pred_class))
print("Classification Report:\n", classification_report(y_class, y_pred_class))

X_train, X_test, ytrain_cont, ytest_cont, ytrain_class, ytest_class = train_test_split(
```

```
        X_bias, y_cont, y_class, test_size=0.2, random_state=42
    )

    theta_hold, cost_hold = Training(X_train, ytrain_cont, alpha, iters=400)
    ytest_pred_cont = np.array([hypothesis(theta_hold, X_test[i]) for i in range(len(X_test))])
    ytest_pred_class = (ytest_pred_cont >= thresh).astype(int)

    print("\nTest Accuracy:", accuracy_score(ytest_class, ytest_pred_class))
    print("Test Confusion Matrix:\n", confusion_matrix(ytest_class, ytest_pred_class))
    print("Test Classification Report:\n", classification_report(ytest_class, ytest_pred_class))
```




```
Accuracy: 0.72
Confusion Matrix:
 [[ 62  72]
 [ 12 154]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.46      0.60       134
           1       0.68      0.93      0.79       166

    accuracy                           0.72       300
   macro avg       0.76      0.70      0.69       300
weighted avg       0.75      0.72      0.70       300


Test Accuracy: 0.6833333333333333
Test Confusion Matrix:
 [[11 14]
 [ 5 30]]
Test Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.44      0.54        25
           1       0.68      0.86      0.76        35

    accuracy                           0.68        60
   macro avg       0.68      0.65      0.65        60
weighted avg       0.68      0.68      0.67        60
```