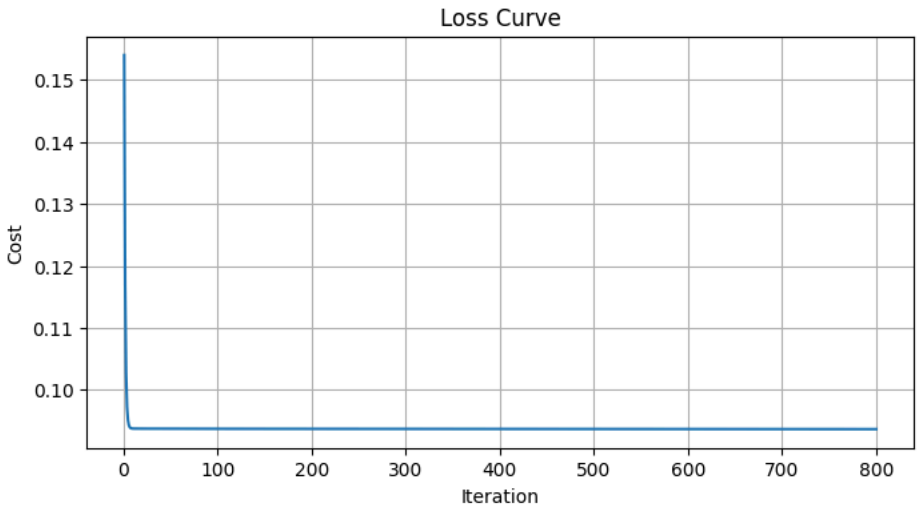
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files


df = pd.read_csv("/content/datset.csv")
df.head()
```

	gender	age	bmi	diabetes
0	0	80.0	25.19	0
1	0	54.0	27.32	0
2	1	28.0	27.32	0
3	0	36.0	23.45	0
4	1	76.0	20.14	0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
X = df.iloc[:, 0:3].values
y_cont = df.iloc[:, -1].values


unique_vals = np.unique(y_cont)
if unique_vals.size >= 3:
    y_class = pd.qcut(y_cont, q=3, labels=[0,1,2]).astype(int)
    multiclass = True
else:
    sorted_vals = sorted(unique_vals)
    mapping = {sorted_vals[i]: i for i in range(len(sorted_vals))}
    y_class = np.array([mapping[v] for v in y_cont])
    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 i 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)])

if multiclass:
    b1 = np.quantile(y_cont, 1/3)
    b2 = np.quantile(y_cont, 2/3)
    def convert_to_class_val(v):
        if v < b1:
            return 0
        elif v < b2:
            return 1
        else:
            return 2
    y_pred_class = np.array([convert_to_class_val(v) for v in y_pred_cont])
else:
    uniq = np.unique(y_cont)
    thresh = (uniq[0] + uniq[1]) / 2.0
    y_pred_class = (y_pred_cont >= thresh).astype(int)


from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

acc = accuracy_score(y_class, y_pred_class)
cm = confusion_matrix(y_class, y_pred_class)
report_str = classification_report(y_class, y_pred_class)


from sklearn.model_selection import train_test_split

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))])

if multiclass:
    ytest_pred_class = np.array([convert_to_class_val(v) for v in ytest_pred_cont])
else:
    ytest_pred_class = (ytest_pred_cont >= thresh).astype(int)

acc_hold = accuracy_score(ytest_class, ytest_pred_class)
cm_hold = confusion_matrix(ytest_class, ytest_pred_class)
report_hold = classification_report(ytest_class, ytest_pred_class)
```