In [1]:
```
import pandas as pd
import numpy as np
```

In [19]:
```
data=pd.read_csv("loan_data.csv")
```

In [20]:
```
data.head(10)
```

Out[20]:

mp_exp	person_home_ownership	loan_amnt	loan_intent	loan_int_rate	loan_percent_income	cb_person_cred_hist_length	credit_
0	RENT	35000.0	PERSONAL	16.02	0.49	3.0	
0	OWN	1000.0	EDUCATION	11.14	0.08	2.0	
3	MORTGAGE	5500.0	MEDICAL	12.87	0.44	3.0	
0	RENT	35000.0	MEDICAL	15.23	0.44	2.0	
1	RENT	35000.0	MEDICAL	14.27	0.53	4.0	
0	OWN	2500.0	VENTURE	7.14	0.19	2.0	
1	RENT	35000.0	EDUCATION	12.42	0.37	3.0	
5	RENT	35000.0	MEDICAL	11.11	0.37	4.0	
3	RENT	35000.0	PERSONAL	8.90	0.35	2.0	
0	OWN	1600.0	VENTURE	14.74	0.13	3.0	

In [21]:
```
data["person_gender"] = data["person_gender"].map({"female": 0, "male": 1})
```

In [22]:
```
data["person_home_ownership"] = data["person_home_ownership"].map({"RENT": 0, "OWN": 1, "MORTGAGE": 2})
```

In [23]:
```
data["loan_intent"] = data["loan_intent"].astype("category").cat.codes
```

In [24]:
```
data["previous_loan_defaults_on_file"] = data["previous_loan_defaults_on_file"].map({"No": 0, "Yes": 1}
```

In [25]:
```
data["person_education"]=data["person_education"].map({"High School":0,"Bachelor":1,"Associate":2,"Mast
```

In [26]:
```
data.head()
```

Out[26]:

	person_age	person_gender	person_education	person_income	person_emp_exp	person_home_ownership	loan_amnt	loan_i
0	22.0	0	3.0	71948.0	0	0.0	35000.0	
1	21.0	0	0.0	12282.0	0	1.0	1000.0	
2	25.0	0	0.0	12438.0	3	2.0	5500.0	
3	23.0	0	1.0	79753.0	0	0.0	35000.0	
4	24.0	1	3.0	66135.0	1	0.0	35000.0	

In [38]:
```
X = data.drop(columns=["loan_status"]).values
y = data["loan_status"].values
```

In [39]:
```
X = (X - X.mean(axis=0)) / X.std(axis=0)
```

In [47]:
```
print(X)
```

```
[[ 1.         -0.95353824 -1.11006918 ... -0.73910854 -1.41981408
  -1.01603973]
 [ 1.         -1.11896309 -1.11006918 ... -0.99686317 -2.5499748
   0.98421348]
 [ 1.         -0.45726369 -1.11006918 ... -0.73910854  0.04741211
  -1.01603973]
 ...
 [ 1.          0.8661351   0.90084476 ...  1.06517387  0.70171569
  -1.01603973]
 [ 1.          0.2044357   0.90084476 ...  0.03415535 -0.5672367
  -1.01603973]
 [ 1.         -0.62268854  0.90084476 ... -0.73910854 -0.09137955
  -1.01603973]]
```

In [117]:
```
X = np.nan_to_num(X, nan=0.0)
```

In [118]:
```
X = np.c_[np.ones(X.shape[0]), X]
```

In [119]:
```
print(X)
```

```
[[ 1.          1.          1.         ... -0.73910854 -1.41981408
  -1.01603973]
 [ 1.          1.          1.         ... -0.99686317 -2.5499748
   0.98421348]
 [ 1.          1.          1.         ... -0.73910854  0.04741211
  -1.01603973]
 ...
 [ 1.          1.          1.         ...  1.06517387  0.70171569
  -1.01603973]
 [ 1.          1.          1.         ...  0.03415535 -0.5672367
  -1.01603973]
 [ 1.          1.          1.         ... -0.73910854 -0.09137955
  -1.01603973]]
```

In [120]:
```
weights = np.random.rand(X.shape[1]) * 0.01
```

In [121]:
```
print(weights)
```

```
[9.73293959e-03 3.20575190e-03 3.07566815e-04 1.59825115e-04
 5.04270893e-03 7.72329144e-03 2.05834952e-03 3.14878740e-03
 3.42535751e-03 6.14117126e-05 2.82232640e-03 8.58051685e-03
 6.89979154e-03 1.48616742e-04 9.60726633e-03 4.47146721e-03]
```

In [122]:
```
#Sigmoid ffunction
def sigmoid(z):
    z = np.clip(z, -500, 500)
    return 1 / (1 + np.exp(-z))
```

In [123]:
```
def compute_loss(y, y_pred):
    y_pred = np.clip(y_pred, 1e-10, 1 - 1e-10)
    return -np.mean(y * np.log(y_pred) + (1 - y) * np.log(1 - y_pred))
```

In [124]:
```
def gradient_descent(X, y, weights, learning_rate, epochs):
    for epoch in range(epochs):
        # Predictions
        z = np.dot(X, weights)
        y_pred = sigmoid(z)

        # gradient
        gradient = np.dot(X.T, (y_pred - y)) / y.size

        # Updating weights
        weights -= learning_rate * gradient

        # loss
        loss = compute_loss(y, y_pred)

        if epoch % 100 == 0:
            print(f"Epoch {epoch}: Loss = {loss}")

    return weights
```

In [125]:
```
print(weights)
```

```
[9.73293959e-03 3.20575190e-03 3.07566815e-04 1.59825115e-04
 5.04270893e-03 7.72329144e-03 2.05834952e-03 3.14878740e-03
 3.42535751e-03 6.14117126e-05 2.82232640e-03 8.58051685e-03
 6.89979154e-03 1.48616742e-04 9.60726633e-03 4.47146721e-03]
```

In [126]:
```
learning_rate = 0.01
epochs = 1000
weights = gradient_descent(X, y, weights, learning_rate, epochs)
```

```
Epoch 0: Loss = 0.6958624742520735
Epoch 100: Loss = 0.49807332068751403
Epoch 200: Loss = 0.42273844309389186
Epoch 300: Loss = 0.38291286369407096
Epoch 400: Loss = 0.35749717696396843
Epoch 500: Loss = 0.3395575963685377
Epoch 600: Loss = 0.32611981293086373
Epoch 700: Loss = 0.3156492224848377
Epoch 800: Loss = 0.30725328470936125
Epoch 900: Loss = 0.3003701927747551
```

In [127]:
```
def predict(X, weights):
    return (sigmoid(np.dot(X, weights)) >= 0.5).astype(int)

y_pred = predict(X, weights)
```

In [128]:
```
print(y_pred)
```

```
[1 0 1 ... 0 1 1]
```

In [129]:
```
accuracy = np.mean(y_pred == y)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 87.94%
```

In [ ]: