```
[2]: # Import necessary
libraries import pandas
as pd import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,
confusion_matrix import matplotlib.pyplot as plt

# Load dataset (assuming 'creditcard.csv' is the name of the file)
data =
pd.read_csv("C:/5th_semester/machine_learning/lab/lab_90_cmlp/creditc
ard. ↪csv")

# Separate features and labels
X = data.drop(columns=['Class']) # Features y =
data['Class'] # Labels (1 for fraud, 0 for
legitimate)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2,␣ ↪random_state=42, stratify=y)

# Standardize the feature data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

#custom MLP classifier

```
[3]: class CustomMLP:
    def __init__(self, input_size, hidden_size, output_size, lr=0.01,
 iterations=1000):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.lr = lr
        self.iterations = iterations
```

```
        # Initialize weights and biases
        self.W1 = np.random.randn(input_size, hidden_size)
        self.b1 = np.zeros((1, hidden_size))
        self.W2 = np.random.randn(hidden_size, output_size)
        self.b2 = np.zeros((1, output_size))
        self.cost_history = []

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

    def sigmoid_derivative(self, z):
        s = self.sigmoid(z)
        return s * (1 - s)

    def forward(self, X):
        self.z1 = np.dot(X, self.W1) + self.b1
        self.a1 = self.sigmoid(self.z1)
        self.z2 = np.dot(self.a1, self.W2) + self.b2
        self.a2 = self.sigmoid(self.z2)
        return self.a2

    def compute_cost(self, Y, output):
        m = Y.shape[0]
        cost = (-1 / m) * np.sum(Y * np.log(output) + (1 - Y) * np.log(1 -␣
↪output))
        return cost

    def backward(self, X, Y):
        m = X.shape[0]
        dz2 = self.a2 - Y
        dW2 = (1 / m) * np.dot(self.a1.T, dz2)
        db2 = (1 / m) * np.sum(dz2, axis=0)

        dz1 = np.dot(dz2, self.W2.T) * self.sigmoid_derivative(self.z1)
        dW1 = (1 / m) * np.dot(X.T, dz1)
        db1 = (1 / m) * np.sum(dz1, axis=0)

        # Update parameters
        self.W1 -= self.lr * dW1
        self.b1 -= self.lr * db1
        self.W2 -= self.lr * dW2
        self.b2 -= self.lr * db2

    def train(self, X, Y):
        for i in range(self.iterations):
            output = self.forward(X)
            cost = self.compute_cost(Y, output)
```

```
            self.cost_history.append(cost)
            self.backward(X, Y)
            if i % 100 == 0:
                print(f"Iteration {i} - Cost: {cost}")

    def predict(self, X):
        output = self.forward(X)
        return (output > 0.5).astype(int)

# Prepare labels in the right shape for binary classification
y_train_reshaped = y_train.values.reshape(-1, 1)
y_test_reshaped = y_test.values.reshape(-1, 1)

# Instantiate and train the model
input_size = X_train.shape[1]
hidden_size = 10  # You can experiment with different sizes
output_size = 1
mlp = CustomMLP(input_size, hidden_size, output_size, lr=0.01, iterations=1000)
mlp.train(X_train, y_train_reshaped)

# Predictions and evaluation for Custom MLP
y_pred_custom = mlp.predict(X_test)
accuracy_custom = accuracy_score(y_test_reshaped, y_pred_custom)
print("Custom MLP Test Accuracy:", accuracy_custom)
```

```
Iteration 0 - Cost: 0.6176973833528581
Iteration 100 - Cost: 0.29137128881018126
Iteration 200 - Cost: 0.17555123004219159
Iteration 300 - Cost: 0.12245497640382032
Iteration 400 - Cost: 0.09332469440154544
Iteration 500 - Cost: 0.07532259239228982
Iteration 600 - Cost: 0.0632450131081464
Iteration 700 - Cost: 0.054646455796337305
Iteration 800 - Cost: 0.04824512989573158
Iteration 900 - Cost: 0.04331117643135821
Custom MLP Test Accuracy: 0.9983322214809873
```

#implementing by library

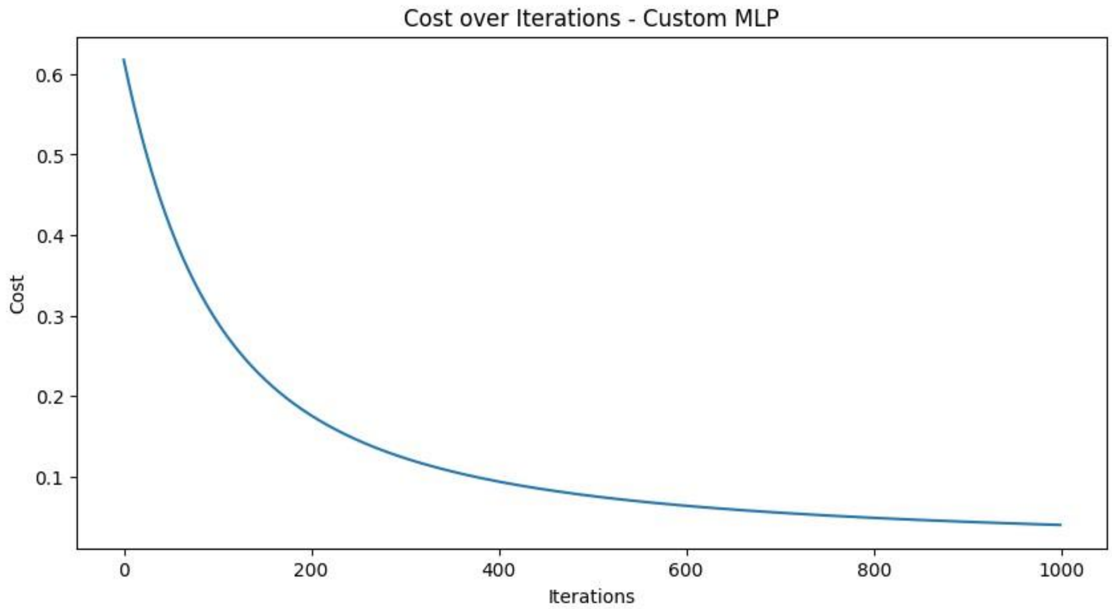
[4]:
```
from sklearn.neural_network import MLPClassifier

# Instantiate and train the MLPClassifier mlp_lib =
MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000,
solver='adam', ␣
 ↪random_state=42)
mlp_lib.fit(X_train, y_train)
```

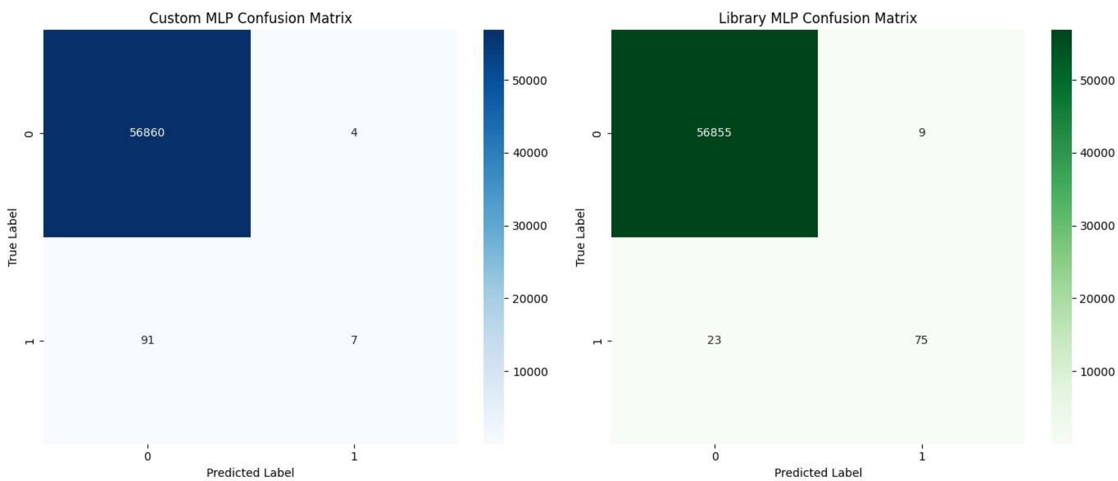
```
# Predictions and evaluation for Scikit-Learn MLP
y_pred_lib = mlp_lib.predict(X_test)
accuracy_lib = accuracy_score(y_test, y_pred_lib)
print("Library MLP Test Accuracy:", accuracy_lib)
```

```
Library MLP Test Accuracy: 0.9994382219725431
```

#visulization of cost over iteration and confusion matrix

[6]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Plot Cost History for Custom MLP Model
plt.figure(figsize=(10, 5))
plt.plot(mlp.cost_history)
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Cost over Iterations - Custom MLP ')
plt.show()

# Confusion Matrices for Custom MLP and Library MLP
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Custom MLP Confusion Matrix
sns.heatmap(conf_matrix_custom, annot=True, fmt="d", cmap="Blues", ax=axes[0])
axes[0].set_title("Custom MLP Confusion Matrix")
axes[0].set_xlabel("Predicted Label")
axes[0].set_ylabel("True Label")

# Library MLP Confusion Matrix
sns.heatmap(conf_matrix_lib, annot=True, fmt="d", cmap="Greens", ax=axes[1])
axes[1].set_title("Library MLP Confusion Matrix")
axes[1].set_xlabel("Predicted Label")
axes[1].set_ylabel("True Label")

plt.tight_layout()
plt.show()
```


Cost over Iterations - Custom MLP
Cost
Iterations
0.6
0.5
0.4
0.3
0.2
0.1
0
200
400
600
800
1000


Custom MLP Confusion Matrix
56860
4
91
7
True Label
Predicted Label
Library MLP Confusion Matrix
56855
9
23
75
True Label
Predicted Label
50000
40000
30000
20000
10000

```
[ ]:
```