Worksheet2 Output:

```
df = pd.read_csv("/content/drive/MyDrive/Week2Workshop2/mnist_dataset.csv") # changed to read_csv and the correct file name
# Step 2: Dataset Information
print("Dataset Preview:")
print(df.head())  # Show first 5 rows
print("\nDataset Information:")
print(df.info())  # Summary of dataset
```

```
Dataset Preview:
   label  pixel_0  pixel_1  pixel_2  pixel_3  pixel_4  pixel_5  pixel_6  \
0      5        0        0        0        0        0        0        0
1      0        0        0        0        0        0        0        0
2      4        0        0        0        0        0        0        0
3      1        0        0        0        0        0        0        0
4      9        0        0        0        0        0        0        0

   pixel_7  pixel_8  ...  pixel_774  pixel_775  pixel_776  pixel_777  \
0        0        0  ...          0          0          0          0
1        0        0  ...          0          0          0          0
2        0        0  ...          0          0          0          0
3        0        0  ...          0          0          0          0
4        0        0  ...          0          0          0          0

   pixel_778  pixel_779  pixel_780  pixel_781  pixel_782  pixel_783
0          0          0          0          0          0          0
1          0          0          0          0          0          0
2          0          0          0          0          0          0
3          0          0          0          0          0          0
4          0          0          0          0          0          0

[5 rows x 785 columns]
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Columns: 785 entries, label to pixel_783
dtypes: int64(785)
memory usage: 359.3 MB
None
```

Softmax Test Case:

This test case checks that each row in the resulting softmax probabilities sums to 1, which is the fundamental property of softmax.

```
# Example test case
z_test = np.array([[2.0, 1.0, 0.1], [1.0, 1.0, 1.0]])
softmax_output = softmax(z_test)

# Verify if the sum of probabilities for each row is 1 using assert
row_sums = np.sum(softmax_output, axis=1)

# Assert that the sum of each row is 1
assert np.allclose(row_sums, 1), f"Test failed: Row sums are {row_sums}"

print("Softmax function passed the test case!")
```

```
Softmax function passed the test case!
```

Test Function for Prediction Function:

The test function ensures that the predicted class labels have the same number of elements as the input samples, verifying that the model produces a valid output shape.

```
# Define test case
X_test = np.array([[0.2, 0.8], [0.5, 0.5], [0.9, 0.1]])  # Feature matrix (3 samples, 2 features)
W_test = np.array([[0.4, 0.2, 0.1], [0.3, 0.7, 0.5]])  # Weights (2 features, 3 classes)
b_test = np.array([0.1, 0.2, 0.3])  # Bias (3 classes)

# Expected Output:
# The function should return an array with class labels (0, 1, or 2)

y_pred_test = predict_softmax(X_test, W_test, b_test)

# Validate output shape
assert y_pred_test.shape == (3,), f"Test failed: Expected shape (3,), got {y_pred_test.shape}"

# Print the predicted labels
print("Predicted class labels:", y_pred_test)
```

```
Predicted class labels: [1 1 0]
```

```
import numpy as np

# Define correct predictions (low loss scenario)
y_true_correct = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])  # True one-hot labels
y_pred_correct = np.array([[0.9, 0.05, 0.05],
                           [0.1, 0.85, 0.05],
                           [0.05, 0.1, 0.85]])  # High confidence in the correct class

# Define incorrect predictions (high loss scenario)
y_pred_incorrect = np.array([[0.05, 0.05, 0.9],  # Highly confident in the wrong class
                             [0.1, 0.05, 0.85],
                             [0.85, 0.1, 0.05]])

# Compute loss for both cases
loss_correct = loss_softmax(y_pred_correct, y_true_correct)
loss_incorrect = loss_softmax(y_pred_incorrect, y_true_correct)

# Validate that incorrect predictions lead to a higher loss
assert loss_correct < loss_incorrect, f"Test failed: Expected loss_correct < loss_incorrect, but got {loss_correct:.4f} >= {loss_incorrect:.4f}"

# Print results
print(f"Cross-Entropy Loss (Correct Predictions): {loss_correct:.4f}")
print(f"Cross-Entropy Loss (Incorrect Predictions): {loss_incorrect:.4f}")
```

```
Cross-Entropy Loss (Correct Predictions): 0.4304
Cross-Entropy Loss (Incorrect Predictions): 8.9872
```

```python
import numpy as np

# Example 1: Correct Prediction (Closer predictions)
X_correct = np.array([[1.0, 0.0], [0.0, 1.0]])  # Feature matrix for correct predictions
y_correct = np.array([[1, 0], [0, 1]])  # True labels (one-hot encoded, matching predictions)
W_correct = np.array([[5.0, -2.0], [-3.0, 5.0]])  # Weights for correct prediction
b_correct = np.array([0.1, 0.1])  # Bias for correct prediction

# Example 2: Incorrect Prediction (Far off predictions)
X_incorrect = np.array([[0.1, 0.9], [0.8, 0.2]])  # Feature matrix for incorrect predictions
y_incorrect = np.array([[1, 0], [0, 1]])  # True labels (one-hot encoded, incorrect predictions)
W_incorrect = np.array([[0.1, 2.0], [1.5, 0.3]])  # Weights for incorrect prediction
b_incorrect = np.array([0.5, 0.6])  # Bias for incorrect prediction

# Compute cost for correct predictions
cost_correct = cost_softmax(X_correct, y_correct, W_correct, b_correct)

# Compute cost for incorrect predictions
cost_incorrect = cost_softmax(X_incorrect, y_incorrect, W_incorrect, b_incorrect)

# Check if the cost for incorrect predictions is greater than for correct predictions
assert cost_incorrect > cost_correct, f"Test failed: Incorrect cost {cost_incorrect} is not greater than correct cost {cost_correct}"

# Print the costs for verification
print("Cost for correct prediction:", cost_correct)
print("Cost for incorrect prediction:", cost_incorrect)

print("Test passed!")
```

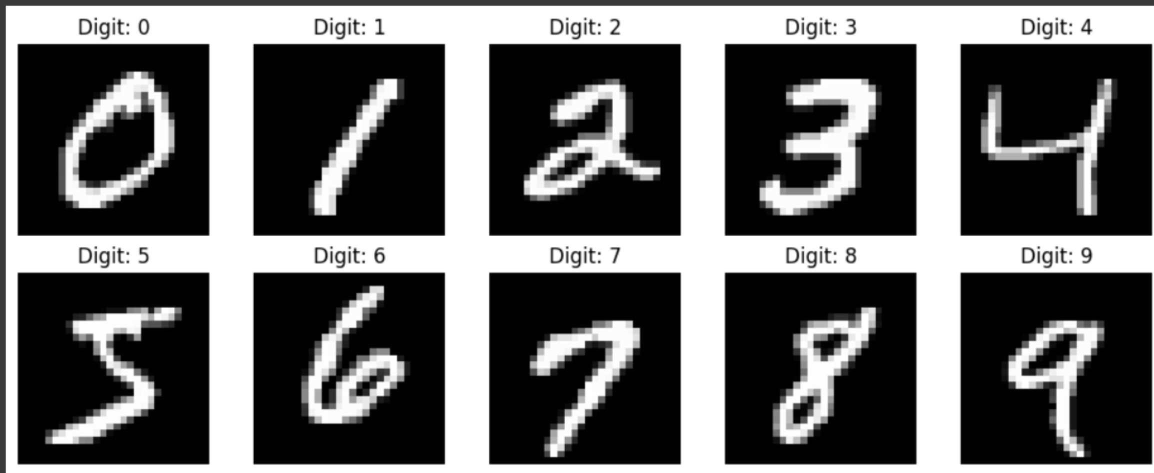
```
Cost for correct prediction: 0.0006234364133349324
Cost for incorrect prediction: 0.2993086135944 6115
Test passed!
```

```python
import numpy as np

# Define a simple feature matrix and true labels
X_test = np.array([[0.2, 0.8], [0.5, 0.5], [0.9, 0.1]])  # Feature matrix (3 samples, 2 features)
y_test = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])  # True labels (one-hot encoded, 3 classes)

# Define weight matrix and bias vector
W_test = np.array([[0.4, 0.2, 0.1], [0.3, 0.7, 0.5]])  # Weights (2 features, 3 classes)
b_test = np.array([0.1, 0.2, 0.3])  # Bias (3 classes)

# Compute the gradients using the function
grad_W, grad_b = compute_gradient_softmax(X_test, y_test, W_test, b_test)

# Manually compute the predicted probabilities (using softmax function)
z_test = np.dot(X_test, W_test) + b_test
y_pred_test = softmax(z_test)

# Compute the manually computed gradients
grad_W_manual = np.dot(X_test.T, (y_pred_test - y_test)) / X_test.shape[0]
grad_b_manual = np.sum(y_pred_test - y_test, axis=0) / X_test.shape[0]

# Assert that the gradients computed by the function match the manually computed gradients
assert np.allclose(grad_W, grad_W_manual), f"Test failed: Gradients w.r.t. W are not equal.\nExpected: {grad_W_manual}\nGot: {grad_W}"
assert np.allclose(grad_b, grad_b_manual), f"Test failed: Gradients w.r.t. b are not equal.\nExpected: {grad_b_manual}\nGot: {grad_b}"

# Print the gradients for verification
print("Gradient w.r.t. W:", grad_W)
print("Gradient w.r.t. b:", grad_b)
```

```python
print("Test passed!")
```

```
Gradient w.r.t. W: [[ 0.1031051   0.01805685 -0.12116196]
 [-0.13600547  0.00679023  0.12921524]]
Gradient w.r.t. b: [-0.03290036  0.02484708  0.00805328]
Test passed!
```

```python
csv_file_path = "/content/drive/MyDrive/Week2Workshop2/mnist_dataset.csv"  # Path to saved dataset
X_train, X_test, y_train, y_test = load_and_prepare_mnist(csv_file_path)
```



A Quick debugging Step:

```python
# Assert that X and y have matching lengths
assert len(X_train) == len(y_train), f"Error: X and y have different lengths! X={len(X_train)}, y={len(y_train)}"
print("Move forward: Dimension of Feture Matrix X and label vector y matched.")
```

```
Move forward: Dimension of Feture Matrix X and label vector y matched.
```

Train the Model:

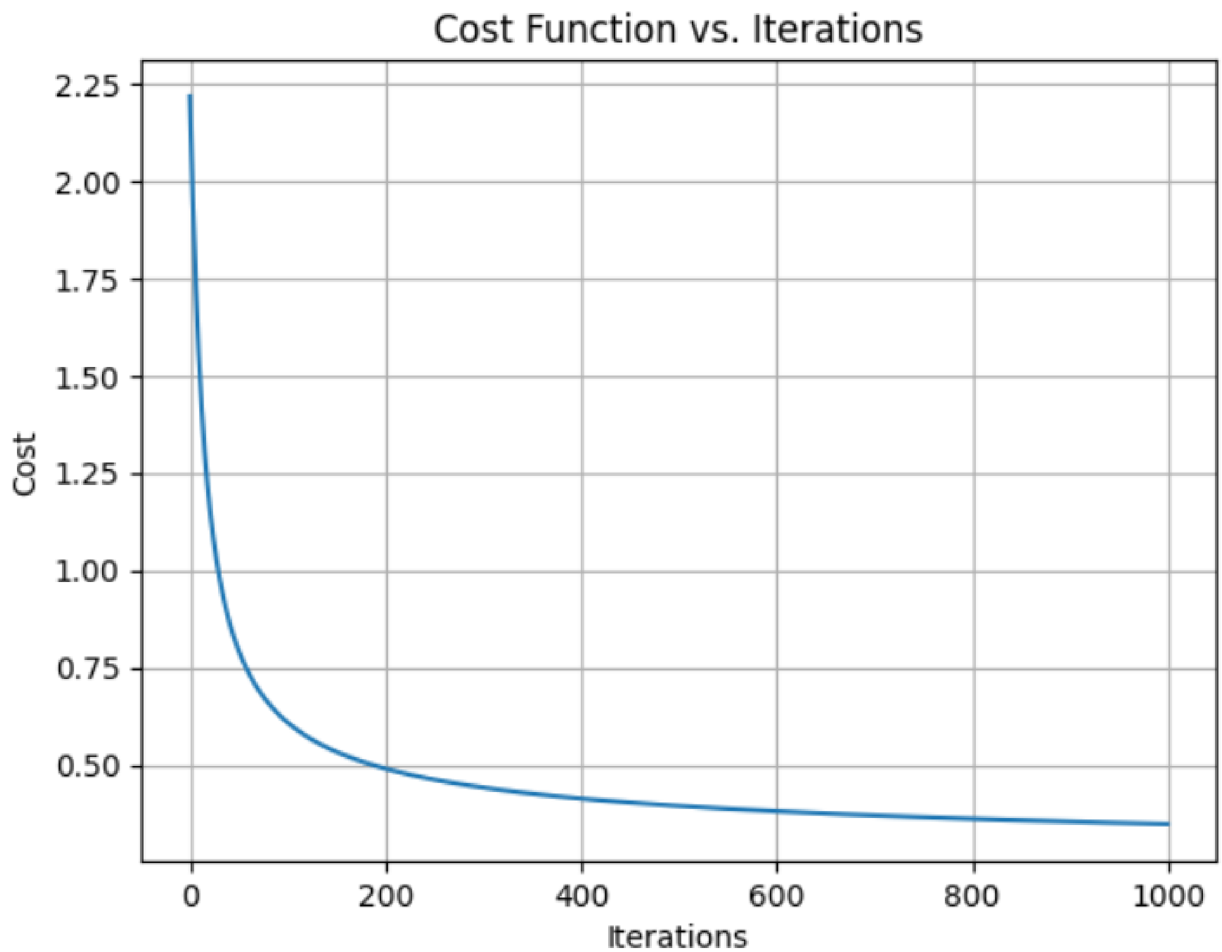
```python
print(f"Training data shape: {X_train.shape}")
print(f"Test data shape: {X_test.shape}")
```

```
Training data shape: (48000, 784)
Test data shape: (12000, 784)
```

```python
from sklearn.preprocessing import OneHotEncoder

# Check if y_train is one-hot encoded
if len(y_train.shape) == 1:
    encoder = OneHotEncoder(sparse_output=False)  # Use sparse_output=False for newer versions of sklearn
    y_train = encoder.fit_transform(y_train.reshape(-1, 1))  # One-hot encode labels
    y_test = encoder.transform(y_test.reshape(-1, 1))  # One-hot encode test labels

# Now y_train is one-hot encoded, and we can proceed to use it
d = X_train.shape[1]  # Number of features (columns in X_train)
c = y_train.shape[1]  # Number of classes (columns in y_train after one-hot encoding)

# Initialize weights with small random values and biases with zeros
W = np.random.randn(d, c) * 0.01  # Small random weights initialized
b = np.zeros(c)  # Bias initialized to 0

# Set hyperparameters for gradient descent
alpha = 0.1  # Learning rate
n_iter = 1000  # Number of iterations to run gradient descent

# Train the model using gradient descent
W_opt, b_opt, cost_history = gradient_descent_softmax(X_train, y_train, W, b, alpha, n_iter, show_cost=True)

# Plot the cost history to visualize the convergence
plt.plot(cost_history)
plt.title('Cost Function vs. Iterations')
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.grid(True)
```

```python
plt.grid(True)
plt.show()
```

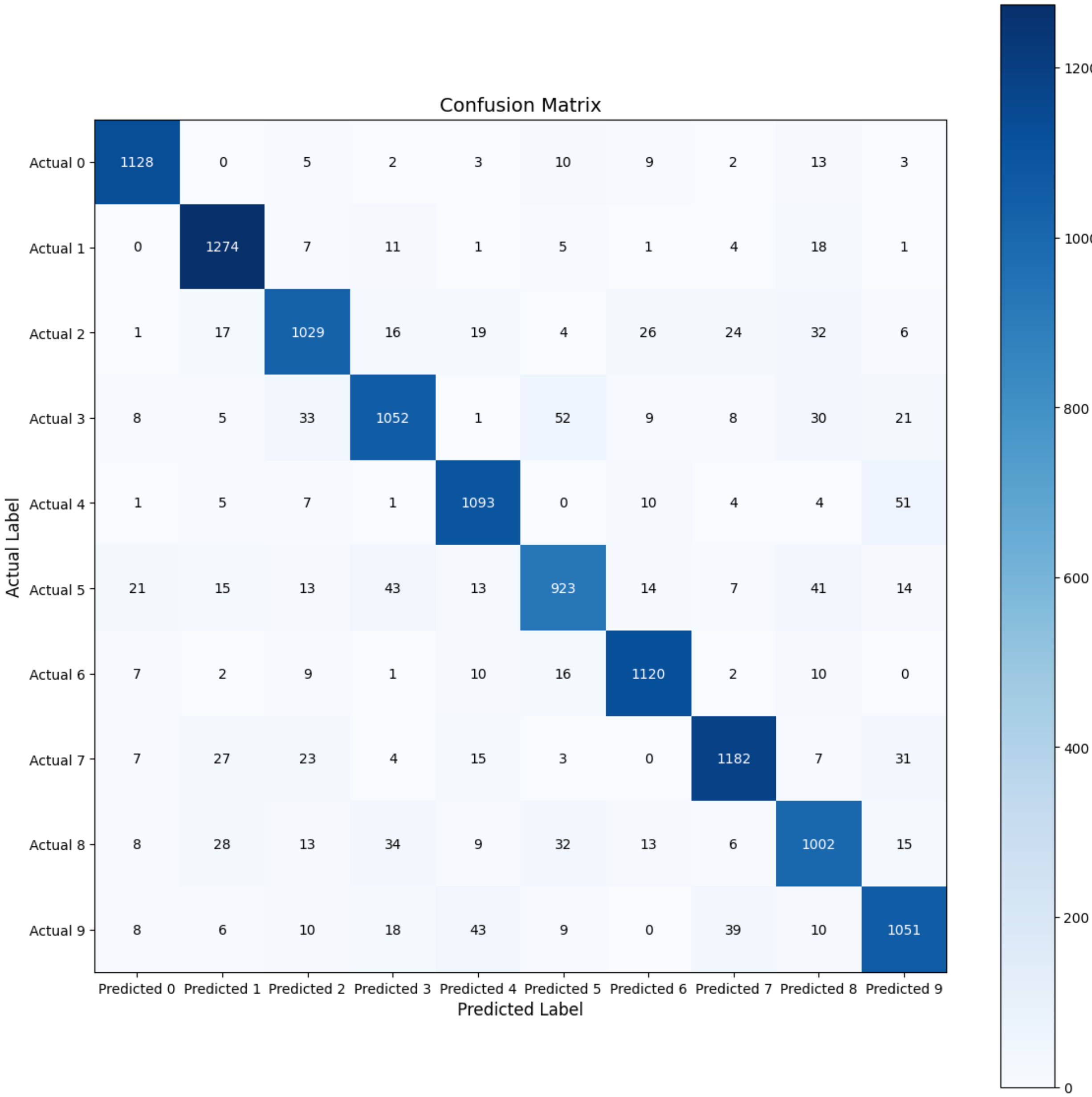
```
Iteration 0: Cost = 2.2172
Iteration 100: Cost = 0.6081
Iteration 200: Cost = 0.4898
Iteration 300: Cost = 0.4411
Iteration 400: Cost = 0.4129
Iteration 500: Cost = 0.3940
Iteration 600: Cost = 0.3802
Iteration 700: Cost = 0.3695
Iteration 800: Cost = 0.3608
Iteration 900: Cost = 0.3537
```


Cost Function vs. Iterations
Cost
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0
200
400
600
800
1000
Iterations

```
# Predict on the test set
[21] y_pred_test = predict_softmax(X_test, W_opt, b_opt)

     # Evaluate accuracy
     y_test_labels = np.argmax(y_test, axis=1)  # True labels in numeric form

     # Evaluate the model
     cm, precision, recall, f1 = evaluate_classification(y_test_labels, y_pred_test)

     # Print the evaluation metrics
     print("\nConfusion Matrix:")
     print(cm)
     print(f"Precision: {precision:.2f}")
     print(f"Recall: {recall:.2f}")
     print(f"F1-Score: {f1:.2f}")

     # Visualizing the Confusion Matrix
     fig, ax = plt.subplots(figsize=(12, 12))
     cax = ax.imshow(cm, cmap='Blues')  # Use a color map for better visualization

     # Dynamic number of classes
     num_classes = cm.shape[0]
     ax.set_xticks(range(num_classes))
     ax.set_yticks(range(num_classes))
     ax.set_xticklabels([f'Predicted {i}' for i in range(num_classes)])
     ax.set_yticklabels([f'Actual {i}' for i in range(num_classes)])

     # Add labels to each cell in the confusion matrix
     for i in range(cm.shape[0]):
         for j in range(cm.shape[1]):
```

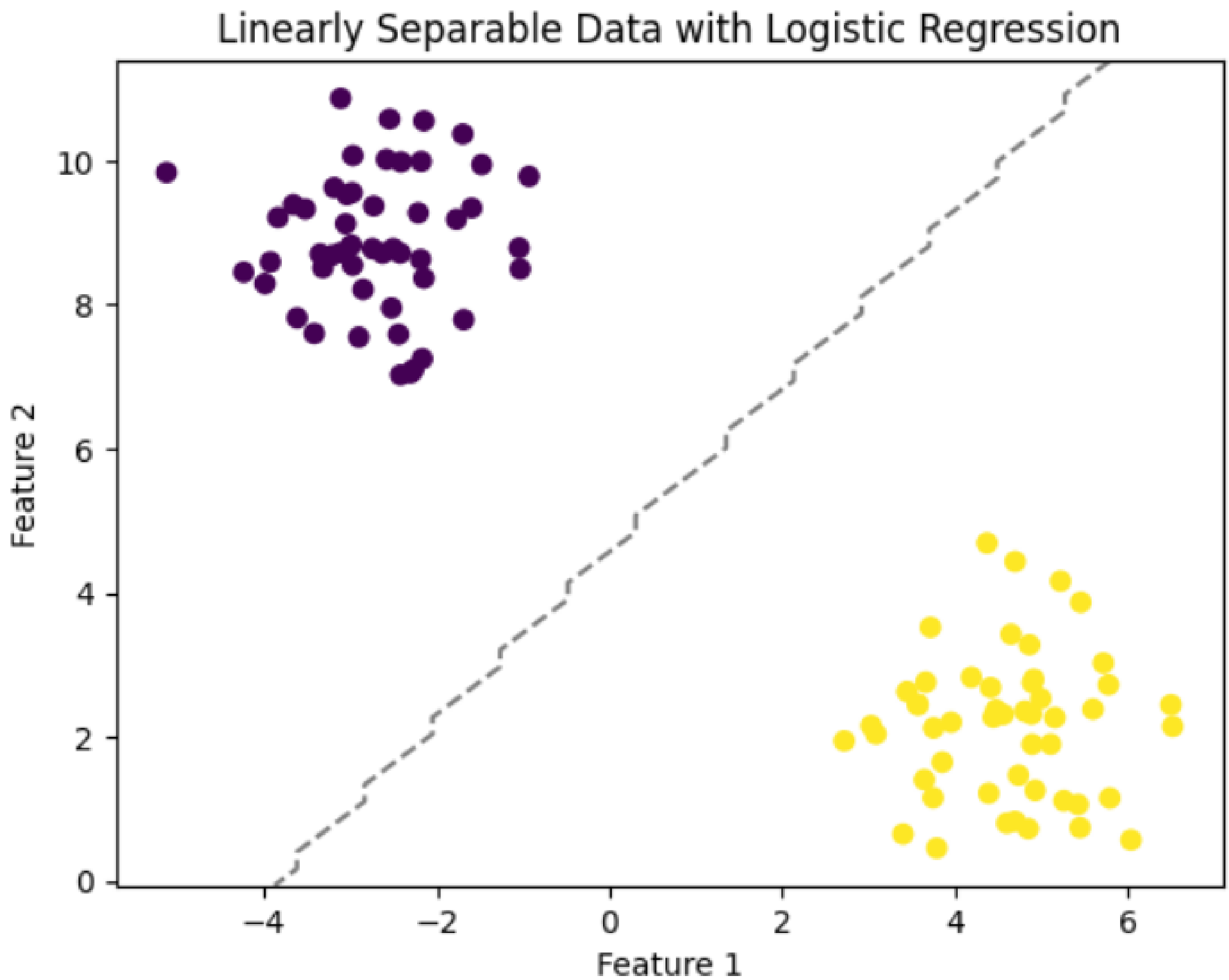
```python
    for j in range(cm.shape[1]):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='white' if cm[i, j] > np.max(cm) / 2 else 'black')

# Add grid lines and axis labels
ax.grid(False)
plt.title('Confusion Matrix', fontsize=14)
plt.xlabel('Predicted Label', fontsize=12)
plt.ylabel('Actual Label', fontsize=12)

# Adjust layout
plt.tight_layout()
plt.colorbar(cax)
plt.show()
```

```
Confusion Matrix:
[[1128    0    5    2    3   10    9    2   13    3]
 [   0 1274    7   11    1    5    1    4   18    1]
 [   1   17 1029   16   19    4   26   24   32    6]
 [   8    5   33 1052    1   52    9    8   30   21]
 [   1    5    7    1 1093    0   10    4    4   51]
 [  21   15   13   43   13  923   14    7   41   14]
 [   7    2    9    1   10   16 1120    2   10    0]
 [   7   27   23    4   15    3    0 1182    7   31]
 [   8   28   13   34    9   32   13    6 1002   15]
 [   8    6   10   18   43    9    0   39   10 1051]]
Precision: 0.90
Recall: 0.90
F1-Score: 0.90
```

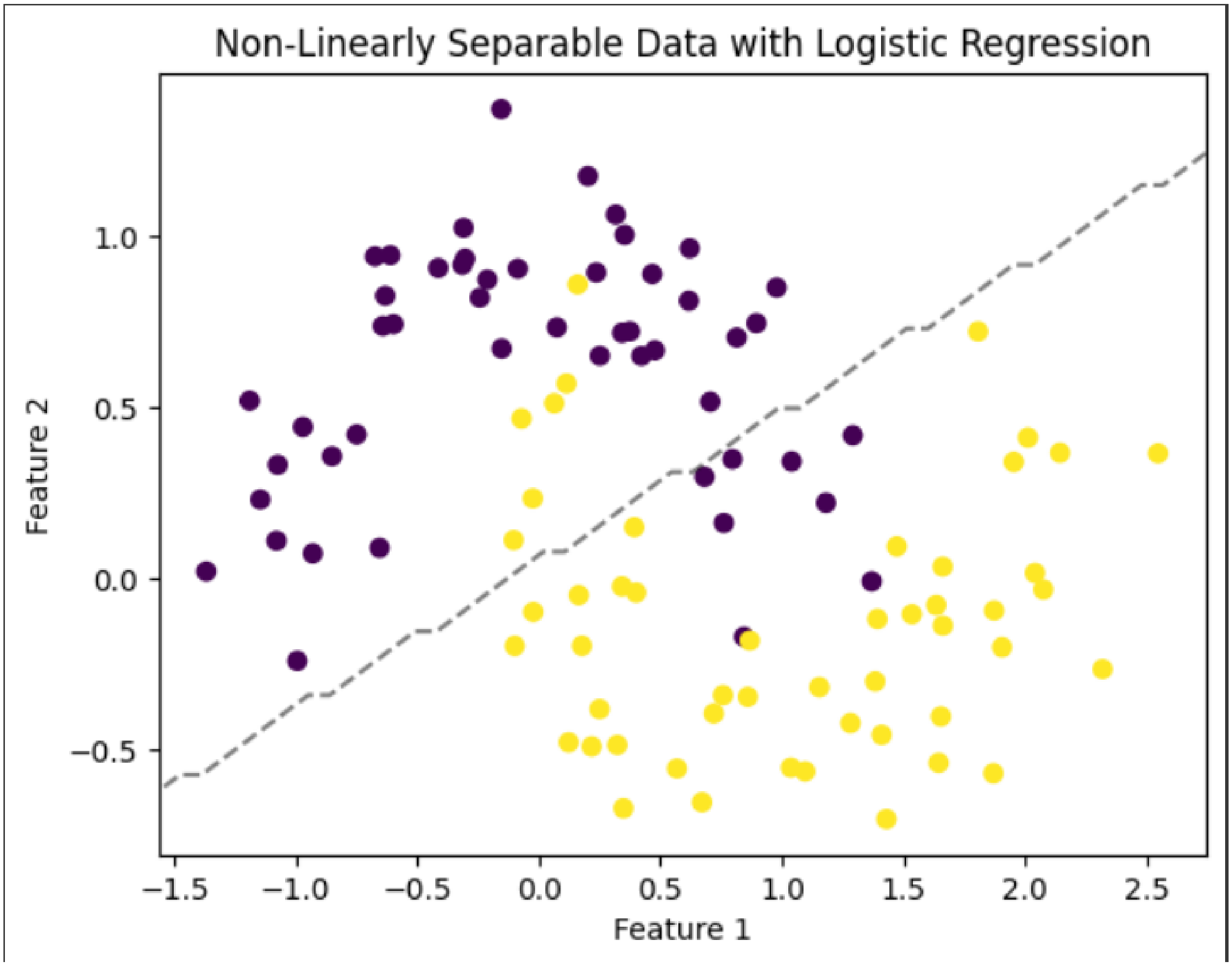

Confusion Matrix
Actual Label
Predicted Label
Actual 0
Actual 1
Actual 2
Actual 3
Actual 4
Actual 5
Actual 6
Actual 7
Actual 8
Actual 9
Predicted 0
Predicted 1
Predicted 2
Predicted 3
Predicted 4
Predicted 5
Predicted 6
Predicted 7
Predicted 8
Predicted 9
1128 0 5 2 3 10 9 2 13 3
0 1274 7 11 1 5 1 4 18 1
1 17 1029 16 19 4 26 24 32 6
8 5 33 1052 1 52 9 8 30 21
1 5 7 1 1093 0 10 4 4 51
21 15 13 43 13 923 14 7 41 14
7 2 9 1 10 16 1120 2 10 0
7 27 23 4 15 3 0 1182 7 31
8 28 13 34 9 32 13 6 1002 15
8 6 10 18 43 9 0 39 10 1051
1200
1000
800
600
400
200
0

[22]

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_blobs

# Generate linearly separable data
X, y = make_blobs(n_samples=100, centers=2, random_state=42)

# Create and train a logistic regression model
model = LogisticRegression()
model.fit(X, y)

# Plot the data points and decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 50),
                     np.linspace(ylim[0], ylim[1], 50))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, colors='k', levels=[0], alpha=0.5,
            linestyles=['--'])
plt.title('Linearly Separable Data with Logistic Regression')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```


Linearly Separable Data with Logistic Regression
Feature 2
Feature 1
10
8
6
4
2
0
−4
−2
0
2
4
6

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_moons

# Generate non-linearly separable data
X, y = make_moons(n_samples=100, noise=0.2, random_state=42)

# Create and train a logistic regression model
model = LogisticRegression()
model.fit(X, y)

# Plot the data points and decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 50),
                     np.linspace(ylim[0], ylim[1], 50))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, colors='k', levels=[0], alpha=0.5,
            linestyles=['--'])
plt.title('Non-Linearly Separable Data with Logistic Regression')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```


Non-Linearly Separable Data with Logistic Regression
Feature 2
Feature 1
1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5