In [ ]:

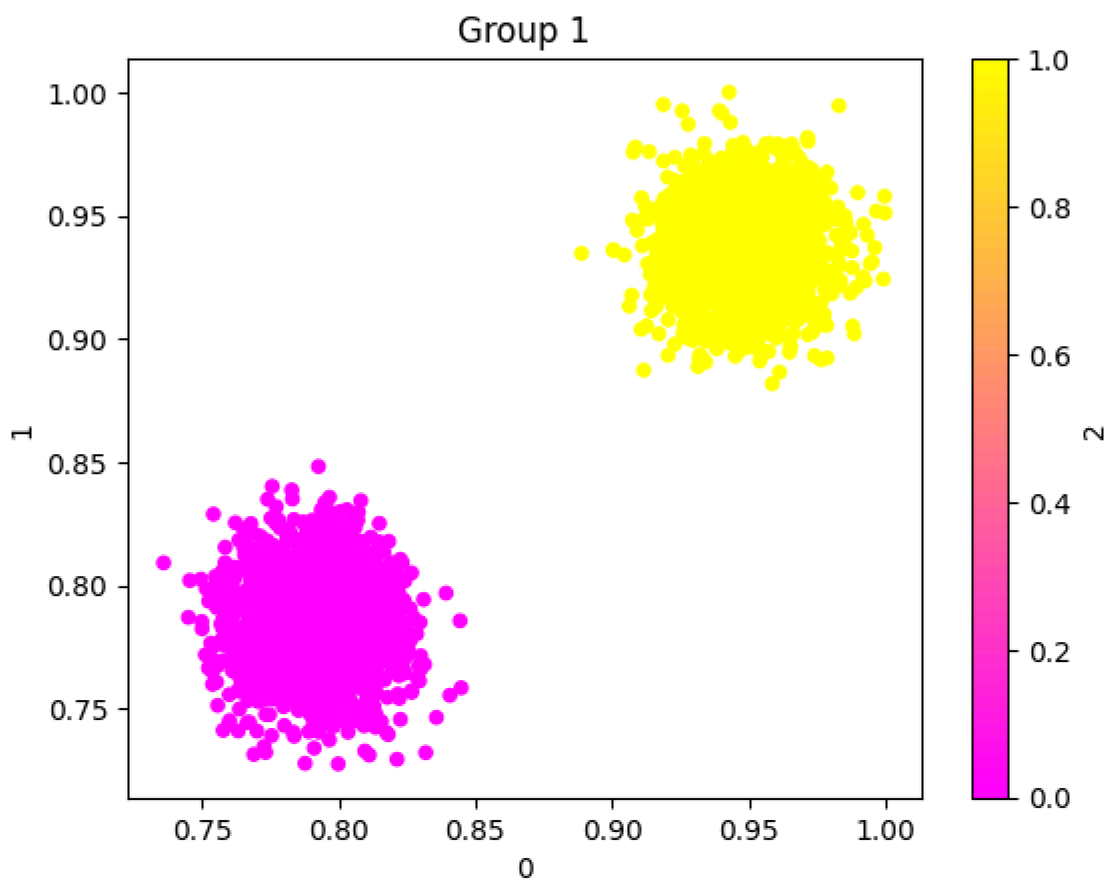
```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
# I'm just using this library function to make the matrix look nice, all calculatio
from sklearn.metrics import ConfusionMatrixDisplay
```

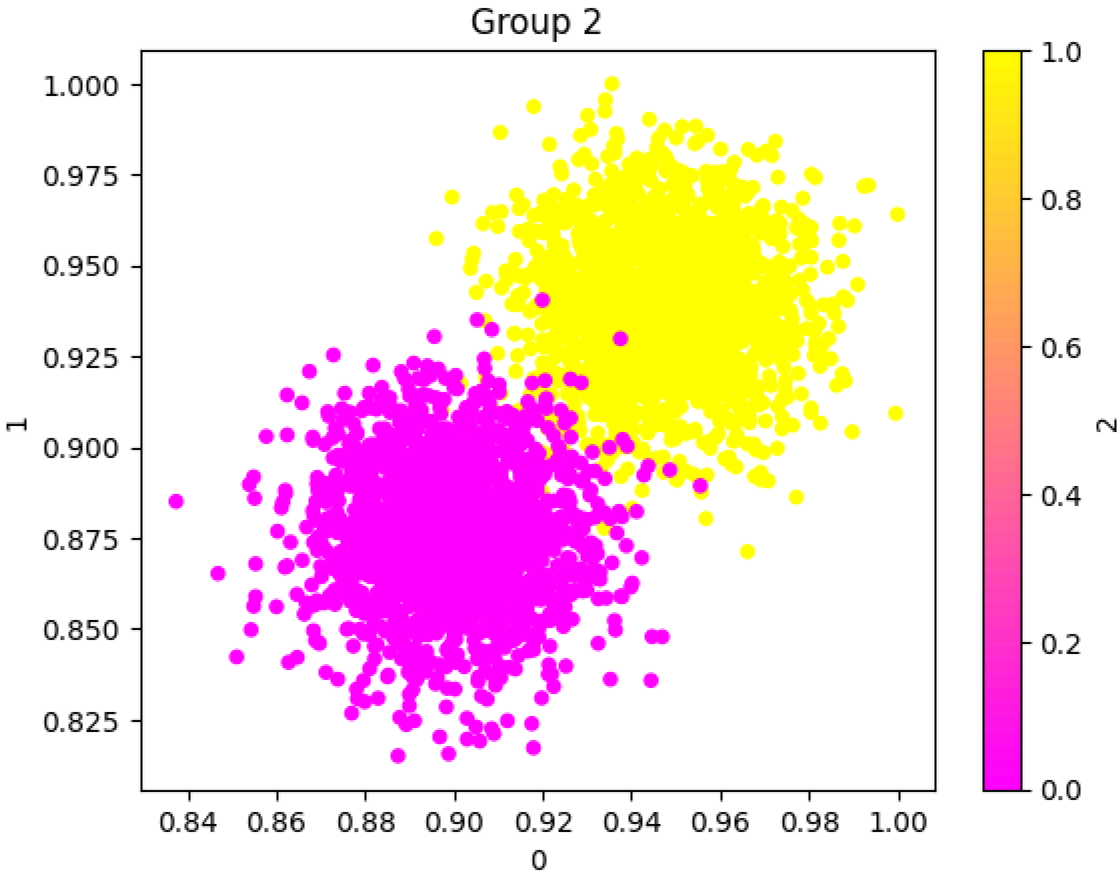
In [ ]:

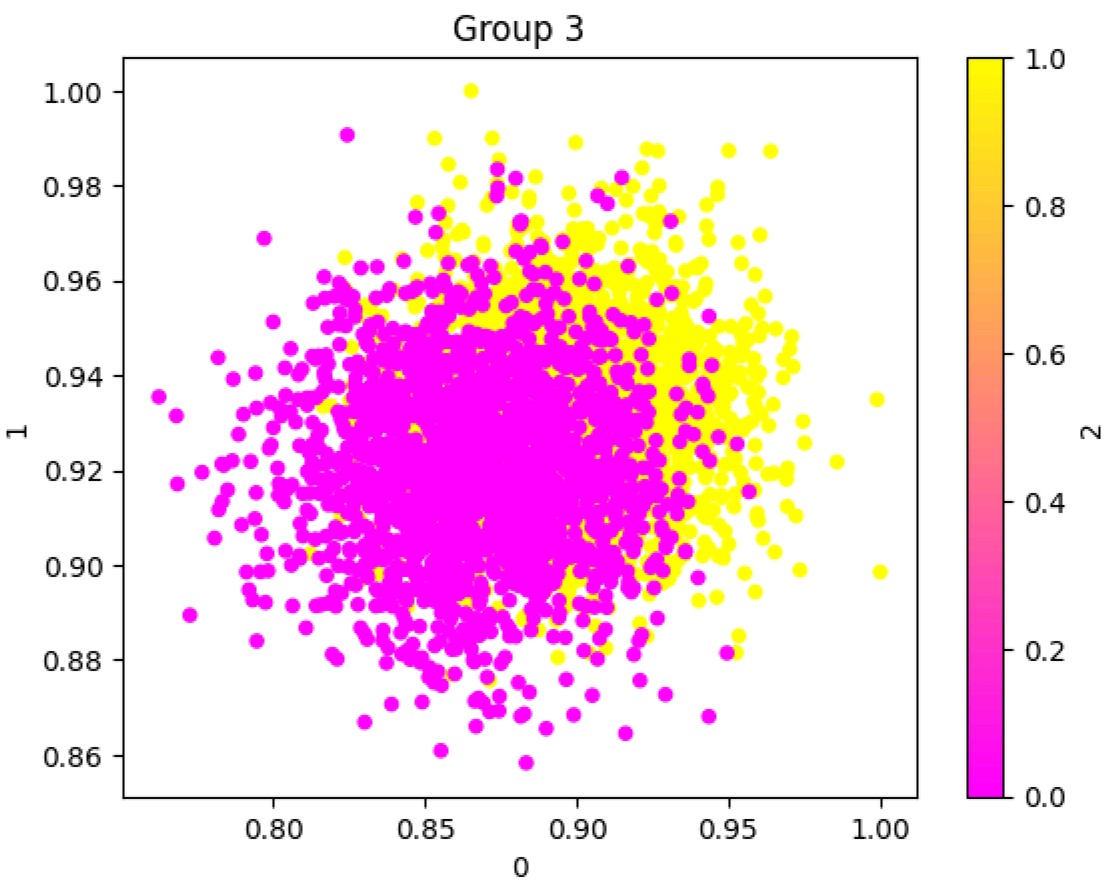
```python
# download all of the datasets into a list of dataframes
group_names = ['groupA', 'groupB', 'groupC']
datasets = []
for name in group_names:
    datasets.append(pd.read_csv(f"./Project1_Data/{name}.txt", header=None))
    # normalize each column 0 and 1
    for i in range(2):
        datasets[-1][i] = datasets[-1][i]/max(datasets[-1][i])
```

In [ ]:

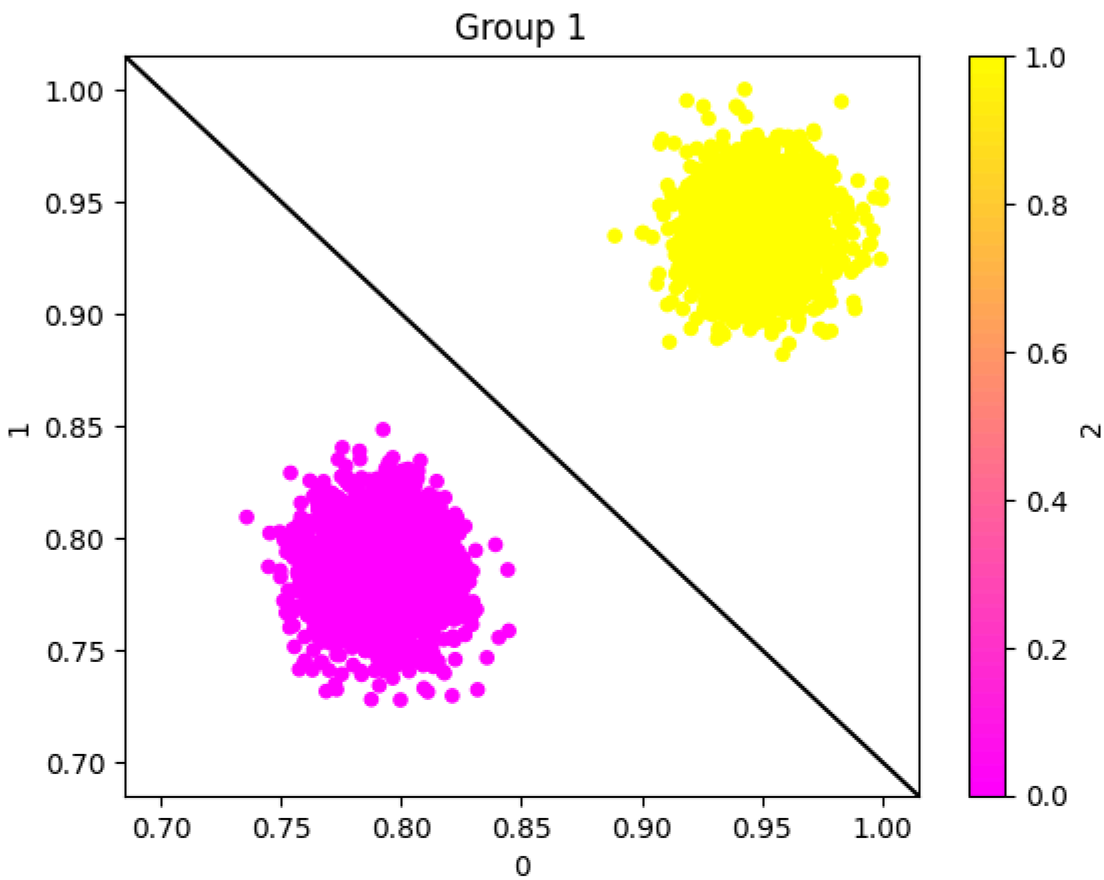
```python
# plot all 3 datasets
for i in range(3):
    datasets[i].plot.scatter(x=0, y=1, c=2, cmap="spring")
    plt.title(f"Group {i+1}")
    plt.savefig(f"group{i+1}_vis.png")
```



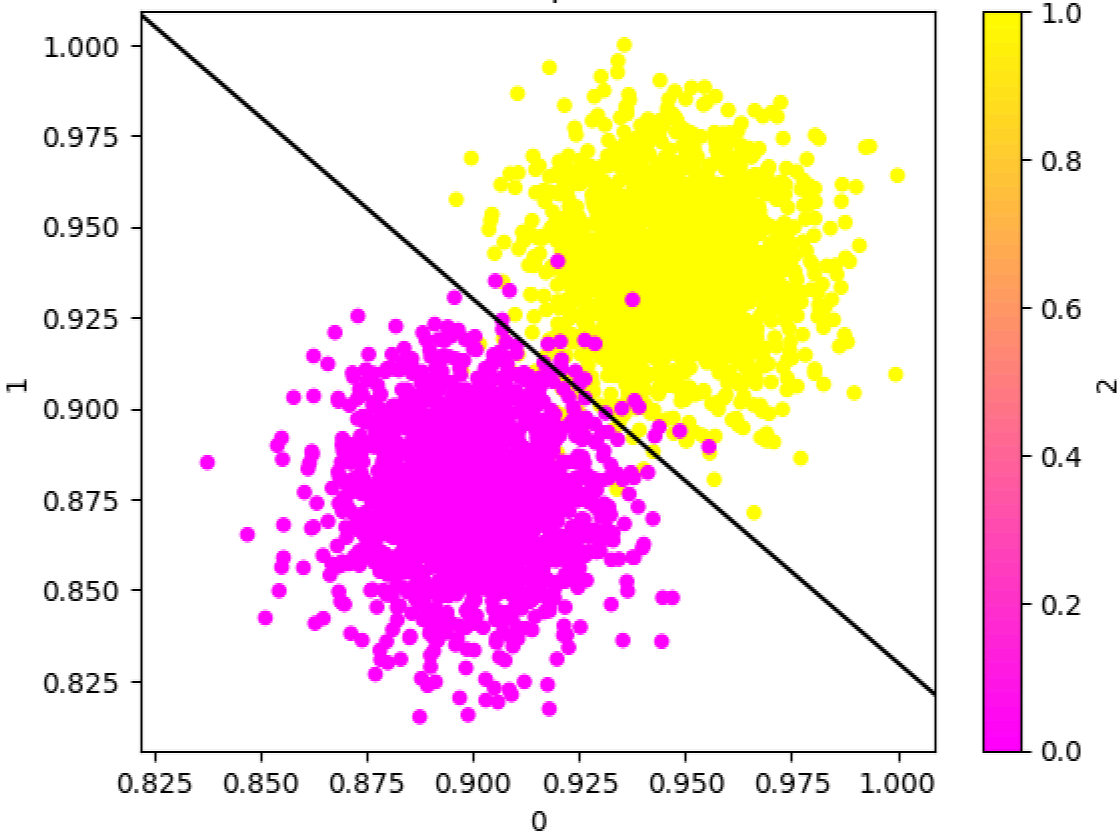

Group 2
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
1
0.84
0.86
0.88
0.90
0.92
0.94
0.96
0.98
1.00
0
1.0
0.8
0.6
0.4
0.2
0.0
2


Group 3
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
1
0.80
0.85
0.90
0.95
1.00
0
1.0
0.8
0.6
0.4
0.2
0.0
2

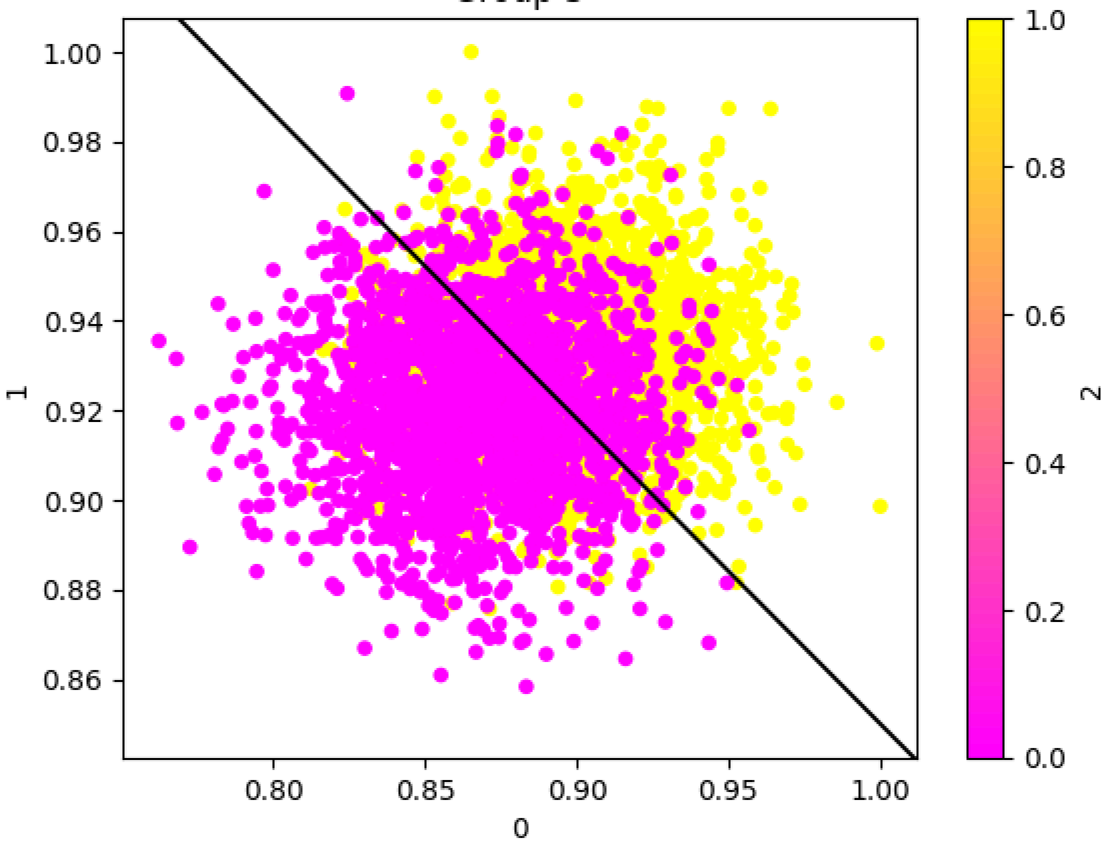
```
In [ ]: # the 2 points that define these lines (I came up with these points by eyeballing t
        pointsx1 = [1,1,1]
        pointsy1 = [0.7,0.83,0.85]
        pointsx2 = [0.7,0.83,0.78]
        pointsy2 = [1,1,1]
        for i, x1, y1, x2, y2 in zip(range(3), pointsx1, pointsy1, pointsx2, pointsy2):
            datasets[i].plot.scatter(x=0, y=1, c=2, cmap="spring")
            plt.axline(xy1=(x1, y1), xy2=(x2,y2), color='black', linestyle='-')
            plt.title(f"Group {i+1}")
            plt.savefig(f"group{i+1}_vis_with_line.png")
```



Group 2



Group 3



In [ ]:

```python
# using the points I have above, I will find the line equations that describe each
lines = []
for i in range(3):
    m = (pointsy2[i] - pointsy1[i]) / (pointsx2[i] - pointsx1[i])
    b = pointsy1[i] - (m * pointsx1[i])
    print(f"For group {i+1}: y = {m:.2f}x + {b:.2f}")
    lines.append([m,b])
```

```
For group 1: y = -1.00x + 1.70
For group 2: y = -1.00x + 1.83
For group 3: y = -0.68x + 1.53
```

The weights are the slopes and the treshholds are the intercepts times -1

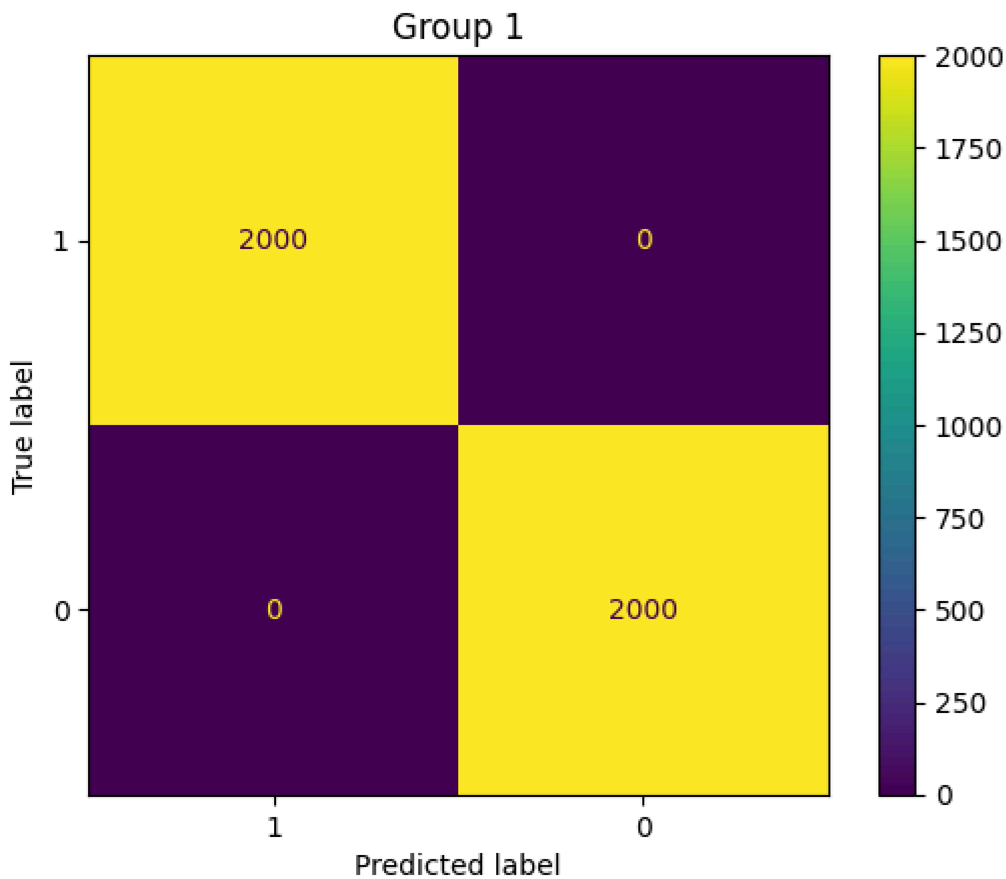
In [ ]:

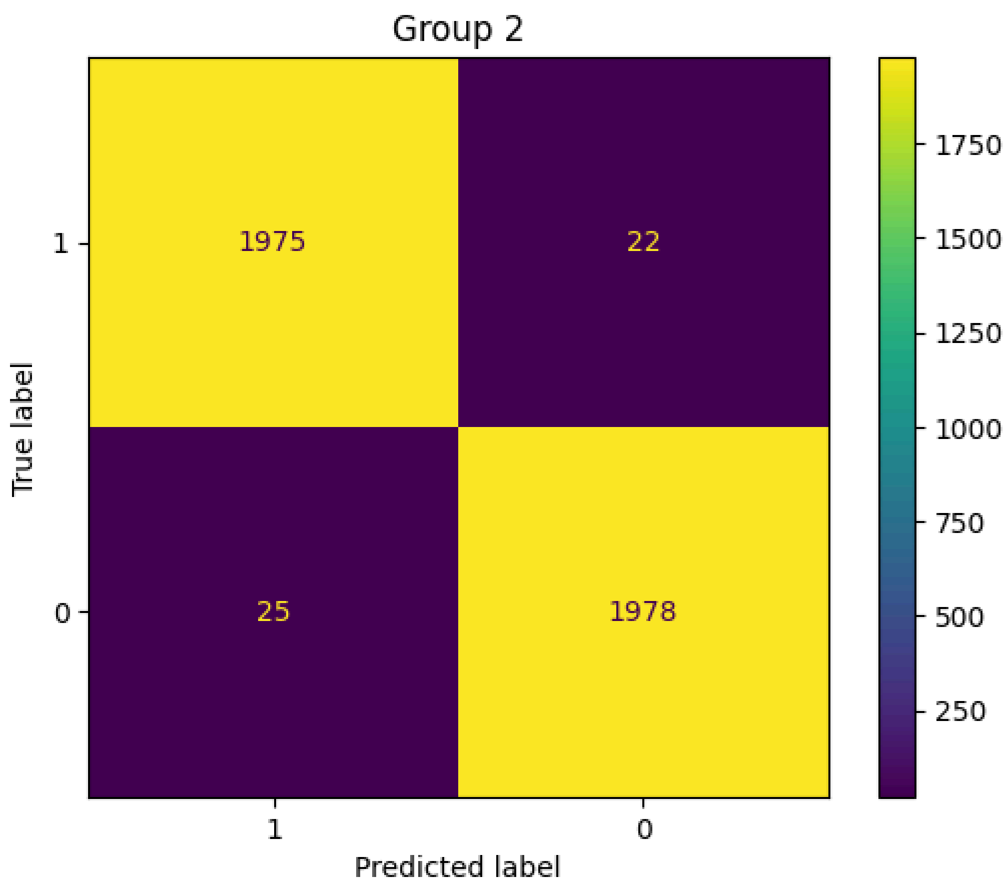
```python
# calculating the predicted category and the accurucy of it
for i in range(3):
    dataset = datasets[i]
    m, b = lines[i]
    # the value is predicted to be 1 if the point is above the line, else 0
    dataset['pred'] = (dataset[1] > m * dataset[0] + b).astype(int)
    # record if the values are in the right category or not
    dataset['correct'] = dataset['pred'] == dataset[2]
```

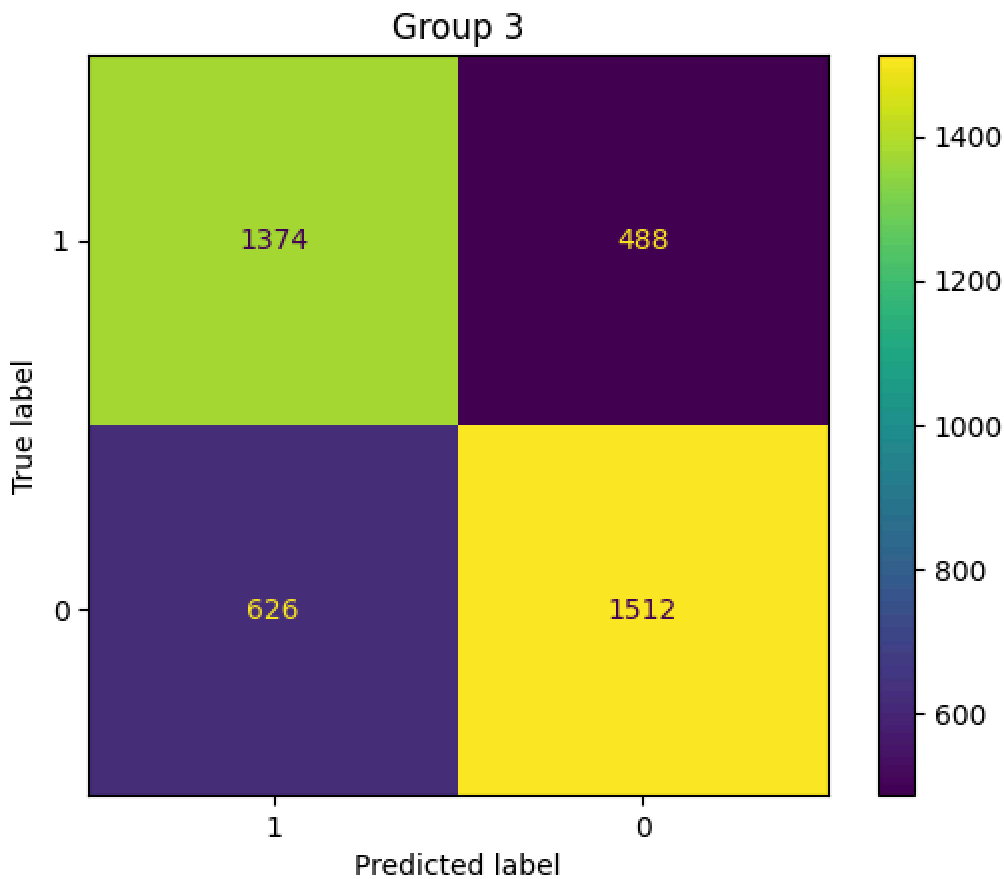
In [ ]:

```python
# generate the confusion matrix
i = 1
for dataset in datasets:
    true_pos = len(dataset[(dataset['correct']) & (dataset['pred'] == 1)])
    true_neg = len(dataset[(dataset['correct']) & (dataset['pred'] == 0)])
    false_pos = len(dataset[(~dataset['correct']) & (dataset['pred'] == 1)])
    false_neg = len(dataset[(~dataset['correct']) & (dataset['pred'] == 0)])
    matrix = np.array([[true_pos, false_pos], [false_neg, true_neg]])
    print(matrix)
    ConfusionMatrixDisplay(confusion_matrix=matrix, display_labels=[1,0]).plot()
    plt.title(f"Group {i}")
    plt.savefig(f"group{i}_confusion_mat.png")
    i += 1
```

```
[[2000    0]
 [   0 2000]]
[[1975   22]
 [  25 1978]]
[[1374  488]
 [ 626 1512]]
```


Group 1
True label
Predicted label
2000
0
0
2000
1
0
1
0
2000
1750
1500
1250
1000
750
500
250
0


Group 2
True label
Predicted label
1975
22
25
1978
1
0
1
0
1750
1500
1250
1000
750
500
250



In [ ]:
```
# Calculating accuracy, error, TPR, TNR, FPR, FNR for all the datasets
metrics = []

for i, dataset in enumerate(datasets, start=1):
    TP = len(dataset[(dataset['pred'] == 1) & (dataset[2] == 1)])
    TN = len(dataset[(dataset['pred'] == 0) & (dataset[2] == 0)])
    FP = len(dataset[(dataset['pred'] == 1) & (dataset[2] == 0)])
    FN = len(dataset[(dataset['pred'] == 0) & (dataset[2] == 1)])
    total = len(dataset)

    accuracy = (TP + TN) / total
    error = (FP + FN) / total
    tpr = TP / (TP + FN) if (TP + FN) > 0 else 0
    tnr = TN / (TN + FP) if (TN + FP) > 0 else 0
    fpr = FP / (FP + TN) if (FP + TN) > 0 else 0
    fnr = FN / (FN + TP) if (FN + TP) > 0 else 0

    metrics.append({
        "Group": i,
        "Accuracy": accuracy,
        "Error": error,
        "TPR": tpr,
        "TNR": tnr,
        "FPR": fpr,
        "FNR": fnr
    })

# Put results in a table
```

```python
import pandas as pd
metrics_df = pd.DataFrame(metrics)
print(metrics_df)
```

```
   Group  Accuracy    Error     TPR    TNR    FPR     FNR
0      1   1.00000  0.00000  1.0000  1.000  0.000  0.0000
1      2   0.98825  0.01175  0.9875  0.989  0.011  0.0125
2      3   0.72150  0.27850  0.6870  0.756  0.244  0.3130
```

Part 1.6: Comparison of Results and Impact of Normalization

Comparison Across Datasets

The performance of the linear separator varied significantly between the three datasets:

- Group 1 achieved perfect classification with 100% accuracy. This means the separation line cleanly divided small and big cars without errors.
- **Group 2** also performed very well, with about 98.8% accuracy. There were very few misclassifications, reflected in the low error rate (1.2%).
- **Group 3** was much more challenging. Its accuracy dropped to about **72%**, and both false positives (≈24%) and false negatives (31%) were high compared to the other groups. This indicates significant overlap between the two car types in this dataset, making a simple linear separator much less effective.

These differences come from how the data was generated, some datasets are more linearly separable than others. Group 1 was the easiest to classify, Group 2 was nearly as good, and Group 3 highlighted the limitations of linear separation.

Why Normalization Helps

Normalization was essential because price (USD) and weight (pounds) originally exist on very different scales. Without normalization, price would dominate the decision boundary due to its larger numeric range. By scaling both features into the same [0,1] range, each contributes equally to the separation line.

This not only makes the classification results more balanced but also ensures that the neuron's weights and thresholds are meaningful and comparable across datasets.

Part B

Part 1: Neuron Definition

Weights:

- (w_X = -3)

- (w_Y = +3)
- (w_Z = +1)
  Threshold: (T = -1)

The net input is:

$$net = -3X + 3Y + Z$$

Output rule:

$$o = \begin{cases} 1, & \text{if } net \geq -1 \\ 0, & \text{if } net < -1 \end{cases}$$

X	Y	Z	¬X+Y	Inequality (net vs T)	Output
0	0	0	1	0 ≥ -1	1
0	0	1	1	1 ≥ -1	1
0	1	0	1	3 ≥ -1	1
0	1	1	1	4 ≥ -1	1
1	0	0	0	-3 ≥ -1 (false)	0
1	0	1	0	-2 ≥ -1 (false)	0
1	1	0	1	0 ≥ -1	1
1	1	1	1	1 ≥ -1	1

Boolean function implemented by this neuron:

$$f(X, Y, Z) = \neg X + Y$$

(equivalent to $X \Rightarrow Y$)

Part 2: Threshold Range

The threshold can take any value in the range:

$$-2 < T \leq 0.$$