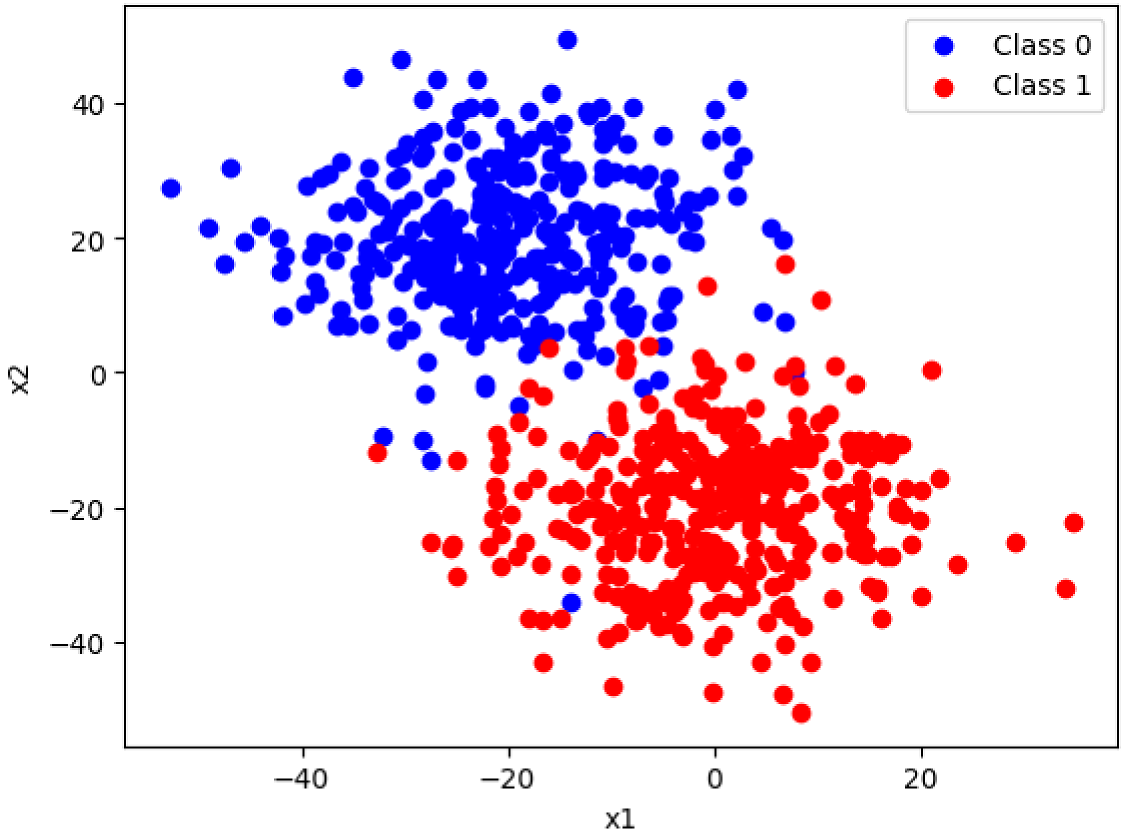
In [17]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_sc
```

In [3]:
```
# The data file is being read in this cell
df = pd.read_csv("Logistic-Regression-data-2-class-v0.csv")
```

In [4]:
```
print(df.head())
```

```
          x1         x2  yclass
0 -12.304702   3.499240       0
1 -21.302900  17.983794       0
2  -6.320254  29.639092       0
3   2.259775  26.227155       0
4 -14.777150  19.536615       0
```

In [5]:
```
X = df[['x1', 'x2']]  # Features 'x1' and 'x2'
y = df['yclass']  # Target variable 'yclass'

# 1. Create a scatter plot to visualize the data
plt.scatter(X[y == 0]['x1'], X[y == 0]['x2'], c='blue', label='Class 0')
plt.scatter(X[y == 1]['x1'], X[y == 1]['x2'], c='red', label='Class 1')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()
```



Most of the points with same class are clustered together. That is points with class 1 are in the lower half plane whereas points with class 0 are in the upper half of the plane. The plot seems to be divided about a straight line.

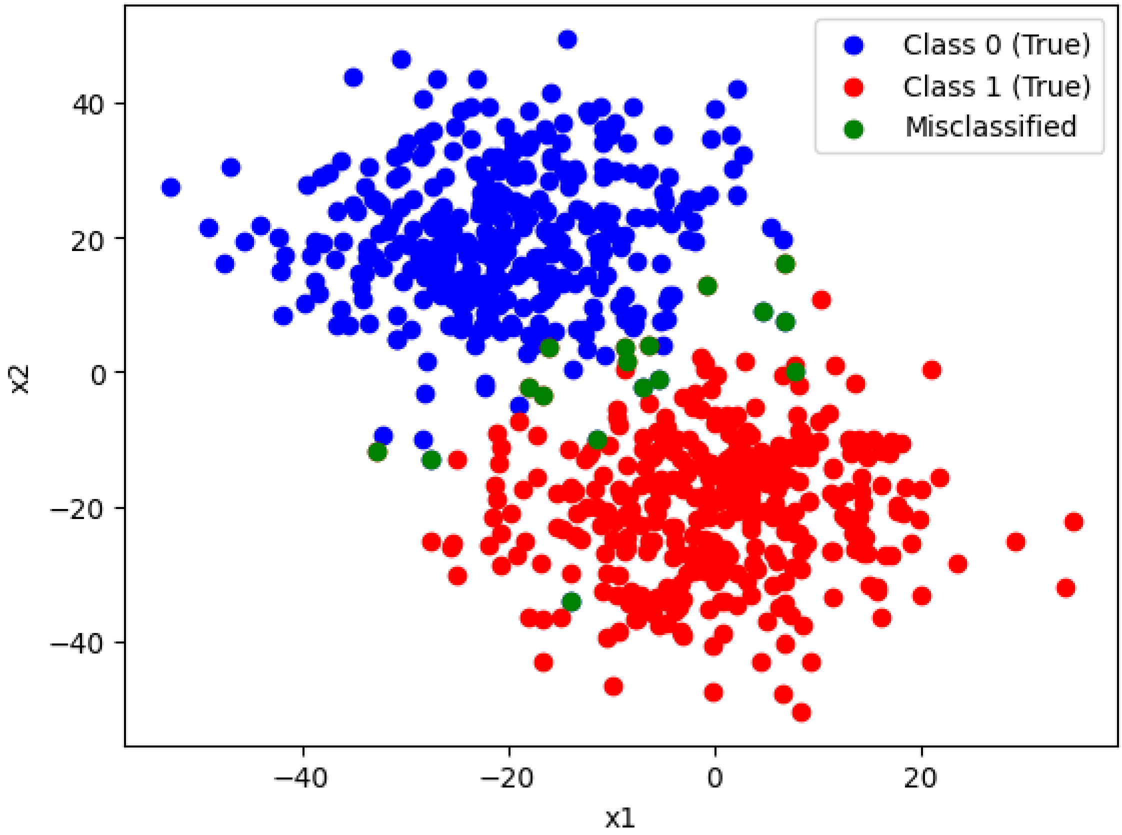
In [6]:
```
logist_regr = LogisticRegression()
logist_regr.fit(X, y)
```

Out[6]:
```
▾ LogisticRegression
LogisticRegression()
```

In [7]:
```
pred = logist_regr.predict(X)
logist_regr.score(df[['x1','x2']], df['yclass'])
```

Out[7]:
```
0.9763888888888889
```

In [8]:
```
plt.scatter(X[y == 0]['x1'], X[y == 0]['x2'], c='blue', label='Class 0 (True)')
plt.scatter(X[y == 1]['x1'], X[y == 1]['x2'], c='red', label='Class 1 (True)')
plt.scatter(X[pred != y]['x1'], X[pred != y]['x2'], c='green', label='Misclassifie
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()
```



Majority of the wrongly classified points are at the border where the two cluster meet. Also another discrepancy arises in the classification of the blue point which was present amongst the cluster of red points. The model ends up classifying it as a red point as it is present deep in the cluster of the red points.

The score of the model is 0.9763888888888889, which is a good and accurate score

In [10]:
```
confusion_matrix_result = confusion_matrix(y, pred)
precision = precision_score(y, pred)
recall = recall_score(y, pred)
f1 = f1_score(y, pred)

TP = confusion_matrix_result[1, 1]
```

```
FP = confusion_matrix_result[0, 1]
TN = confusion_matrix_result[0, 0]
FN = confusion_matrix_result[1, 0]

TPR = TP / (TP + FN)
FPR = FP / (FP + TN)

print("Confusion Matrix:")
print(confusion_matrix_result)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("True Positive Rate (TPR):", TPR)
print("False Positive Rate (FPR):", FPR)
```

```
Confusion Matrix:
[[352   8]
 [  9 351]]
Precision: 0.9777158774373259
Recall: 0.975
F1-score: 0.9763560500695411
True Positive Rate (TPR): 0.975
False Positive Rate (FPR): 0.022222222222222223
```

In [11]:

```
probabilities = logist_regr.predict_proba(X)
```

In [14]:

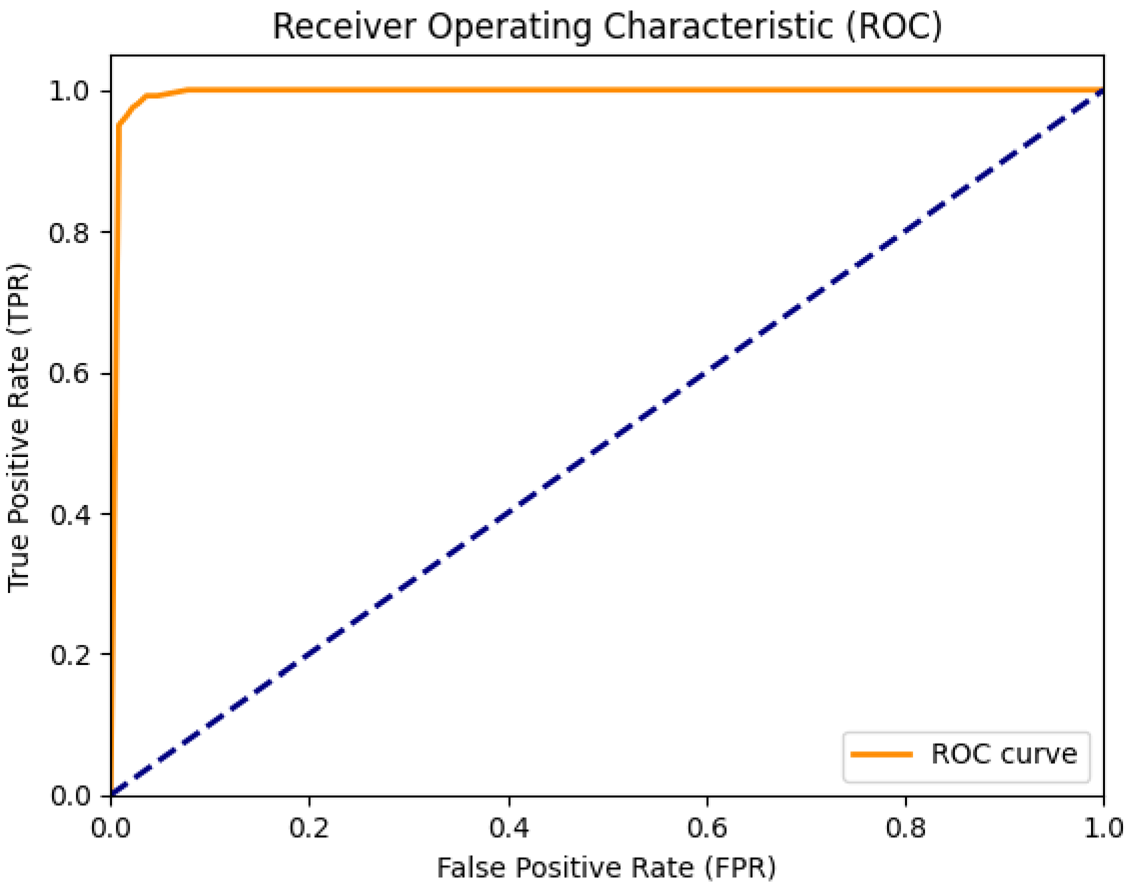
```
tpr_values = []
fpr_values = []
for i in range(0,11):
    # Classify observations based on the threshold
    threshold = i/10
    predicted_labels = (probabilities[:, 1] >= threshold).astype(int)

    # Calculate the confusion matrix
    confusion_matrix_result = confusion_matrix(y, predicted_labels)

    # Calculate True Positive Rate (TPR) and False Positive Rate (FPR)
    TP = confusion_matrix_result[1, 1]
    FP = confusion_matrix_result[0, 1]
    TN = confusion_matrix_result[0, 0]
    FN = confusion_matrix_result[1, 0]

    TPR = TP / (TP + FN)
    FPR = FP / (FP + TN)

    tpr_values.append(TPR)
    fpr_values.append(FPR)
    # Print the results for the current threshold
    print(f"Threshold: {threshold:.1f}")
    print("Confusion Matrix:")
    print(confusion_matrix_result)
    print("True Positive Rate (TPR):", TPR)
    print("False Positive Rate (FPR):", FPR)
    print()
```

```
Threshold: 0.0
Confusion Matrix:
[[  0 360]
 [  0 360]]
True Positive Rate (TPR): 1.0
False Positive Rate (FPR): 1.0

Threshold: 0.1
Confusion Matrix:
[[332  28]
 [  0 360]]
True Positive Rate (TPR): 1.0
False Positive Rate (FPR): 0.07777777777777778

Threshold: 0.2
Confusion Matrix:
[[343  17]
 [  3 357]]
True Positive Rate (TPR): 0.9916666666666667
False Positive Rate (FPR): 0.04722222222222222

Threshold: 0.3
Confusion Matrix:
[[347  13]
 [  3 357]]
True Positive Rate (TPR): 0.9916666666666667
False Positive Rate (FPR): 0.03611111111111111

Threshold: 0.4
Confusion Matrix:
[[350  10]
 [  7 353]]
True Positive Rate (TPR): 0.9805555555555555
False Positive Rate (FPR): 0.027777777777777776

Threshold: 0.5
Confusion Matrix:
[[352   8]
 [  9 351]]
True Positive Rate (TPR): 0.975
False Positive Rate (FPR): 0.022222222222222223

Threshold: 0.6
Confusion Matrix:
[[353   7]
 [ 11 349]]
True Positive Rate (TPR): 0.9694444444444444
False Positive Rate (FPR): 0.019444444444444445

Threshold: 0.7
Confusion Matrix:
[[354   6]
 [ 13 347]]
True Positive Rate (TPR): 0.9638888888888889
False Positive Rate (FPR): 0.016666666666666666

Threshold: 0.8
Confusion Matrix:
[[357   3]
 [ 18 342]]
True Positive Rate (TPR): 0.95
False Positive Rate (FPR): 0.008333333333333333

Threshold: 0.9
```

```
Confusion Matrix:
[[357   3]
 [ 25 335]]
True Positive Rate (TPR): 0.9305555555555556
False Positive Rate (FPR): 0.008333333333333333

Threshold: 1.0
Confusion Matrix:
[[360   0]
 [360   0]]
True Positive Rate (TPR): 0.0
False Positive Rate (FPR): 0.0
```

In [15]:

```
# Plot the ROC curve
plt.figure()
plt.plot(fpr_values, tpr_values, color='darkorange', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```



From the ROC curve, we can see that the area under the curve that is AUC is comparable to the area of the square shown in the figure. Therefore, it is a very good model with accurate predictions.

In [18]:

```
roc_auc = roc_auc_score(y, pred)

# Print the AUC value
print("AUC:", roc_auc)
```

```
AUC: 0.976388888888889
```