In [1]:
```
import pandas as pd
import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

In [2]:
```
# Load dataset (Label = 0,1 | Article = headline of the news)
df1 = pd.read_csv('dataset1.csv', usecols=['Label', 'Article'], low_memory=F
df2 = pd.read_csv('dataset2.csv', usecols=['Label', 'Article'], low_memory=F

# Combine the datasets into one dataframe
df = pd.concat([df1, df2], ignore_index=True)

# Preview the combined dataset
print(df.head())  # Check first few rows to ensure data is loaded correctly
print(df.shape)   # Check the shape (rows, columns) of the dataset
```

```
  Label                                            Article
0     0  Ayon sa TheWrap.com, naghain ng kaso si Krupa,...
1     0  Kilala rin ang singer sa pagkumpas ng kanyang ...
2     0  BLANTYRE, Malawi (AP) -- Bumiyahe patungong Ma...
3     0  Kasama sa programa ang pananalangin, bulaklak ...
4     0  Linisin ang Friendship Department dahil dadala...
(25668, 2)
```

In [3]:
```
# Filter out rows where 'Label' is not numeric
df = df[df['Label'].apply(lambda x: str(x).isdigit())]

# Convert 'Label' to integer
df['Label'] = df['Label'].astype(int)
```

In [4]:
```
# Define a list of Tagalog stopwords
tagalog_stopwords = set([
    'ako', 'at', 'ng', 'sa', 'mga', 'ngayon', 'para', 'ito', 'ni', 'na', 'si
    'hindi', 'may', 'kay', 'kayo', 'kung', 'ngunit', 'o', 'isa', 'pala', 'ya
    'maging', 'dahil', 'tungkol', 'pagsasabi'
])

# Function to clean text
def clean_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove special characters, numbers, and punctuation
    text = re.sub(r'\W', ' ', text)
    text = re.sub(r'\s+', ' ', text)

    # Remove stopwords
    text = ' '.join([word for word in text.split() if word not in tagalog_st

    return text

# Apply the cleaning function to the 'Article' column
```

```
df['cleaned_article'] = df['Article'].apply(clean_text)

# Preview cleaned data
print(df[['Article', 'cleaned_article']].head())
```

```
                                             Article  \
0  Ayon sa TheWrap.com, naghain ng kaso si Krupa,...
1  Kilala rin ang singer sa pagkumpas ng kanyang ...
2  BLANTYRE, Malawi (AP) -- Bumiyahe patungong Ma...
3  Kasama sa programa ang pananalangin, bulaklak ...
4  Linisin ang Friendship Department dahil dadala...

                                     cleaned_article
0  ayon thewrap com naghain kaso si krupa 35 noon...
1  kilala rin ang singer pagkumpas kanyang kamay ...
2  blantyre malawi ap bumiyahe patungong malawi s...
3  kasama programa ang pananalangin bulaklak pags...
4  linisin ang friendship department dadalawin ka...
```

In [5]:
```
# Split the dataset into features (X) and target (y)
X = df['cleaned_article']  # Features (the cleaned articles)
y = df['Label']            # Target (labels: real or fake)

# Split into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Check the split data
print(f"Training samples: {len(X_train)}, Testing samples: {len(X_test)}")
```

```
Training samples: 20527, Testing samples: 5132
```

In [6]:
```
# Convert the text data into TF-IDF vectors
tfidf = TfidfVectorizer(max_features=5000)

# Fit and transform the training data
X_train_tfidf = tfidf.fit_transform(X_train)

# Transform the test data
X_test_tfidf = tfidf.transform(X_test)
```

In [7]:
```
# Initialize the Logistic Regression model
model = LogisticRegression(max_iter=2000)  # Increase max_iter for larger da

# Train the model
model.fit(X_train_tfidf, y_train)
```

Out[7]:
```
▾ LogisticRegression
LogisticRegression(max_iter=2000)
```

In [8]:
```
# Make predictions on the test set
y_pred = model.predict(X_test_tfidf)
```

In [9]:
```
# Check the length of y_test and y_pred to ensure there is no mismatch
print(f"Length of y_test: {len(y_test)}")
```

```
print(f"Length of y_pred: {len(y_pred)}")

# Evaluate the model using classification report
print(classification_report(y_test, y_pred, zero_division=0))
```

```
Length of y_test: 5132
Length of y_pred: 5132
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      3312
           1       0.88      0.76      0.82      1820

    accuracy                           0.88      5132
   macro avg       0.88      0.85      0.87      5132
weighted avg       0.88      0.88      0.88      5132
```

In [10]:

```
#Logistic Regression
# Class 0 (Real News):
# Precision: 0.88 – 88% of the articles predicted as real news were actually
# Recall: 0.95 – 95% of all actual real news articles were correctly identif
# F1-Score: 0.91 – A good balance between precision and recall for real news

# Class 1 (Fake News):
# Precision: 0.88 – 88% of the articles predicted as fake news were actually
# Recall: 0.76 – The model correctly identified 76% of all fake news article
# F1-Score: 0.82 – A good balance between precision and recall for fake news

# Accuracy: 0.88 – The model correctly classified 88% of all articles in the
```

In [13]:

```
from imblearn.over_sampling import SMOTE
```

In [14]:

```
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)  # Training set vectorized
X_test_tfidf = tfidf.transform(X_test)        # Test set vectorized
```

In [15]:

```
# Apply SMOTE to balance the dataset
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_tfidf, y_t
```

In [16]:

```
# Train the Logistic Regression model on resampled data
model = LogisticRegression(max_iter=2000)
model.fit(X_train_resampled, y_train_resampled)
```

Out[16]:

```
▾ LogisticRegression
LogisticRegression(max_iter=2000)
```

In [17]:

```
# Make predictions on the test set
y_pred = model.predict(X_test_tfidf)
```

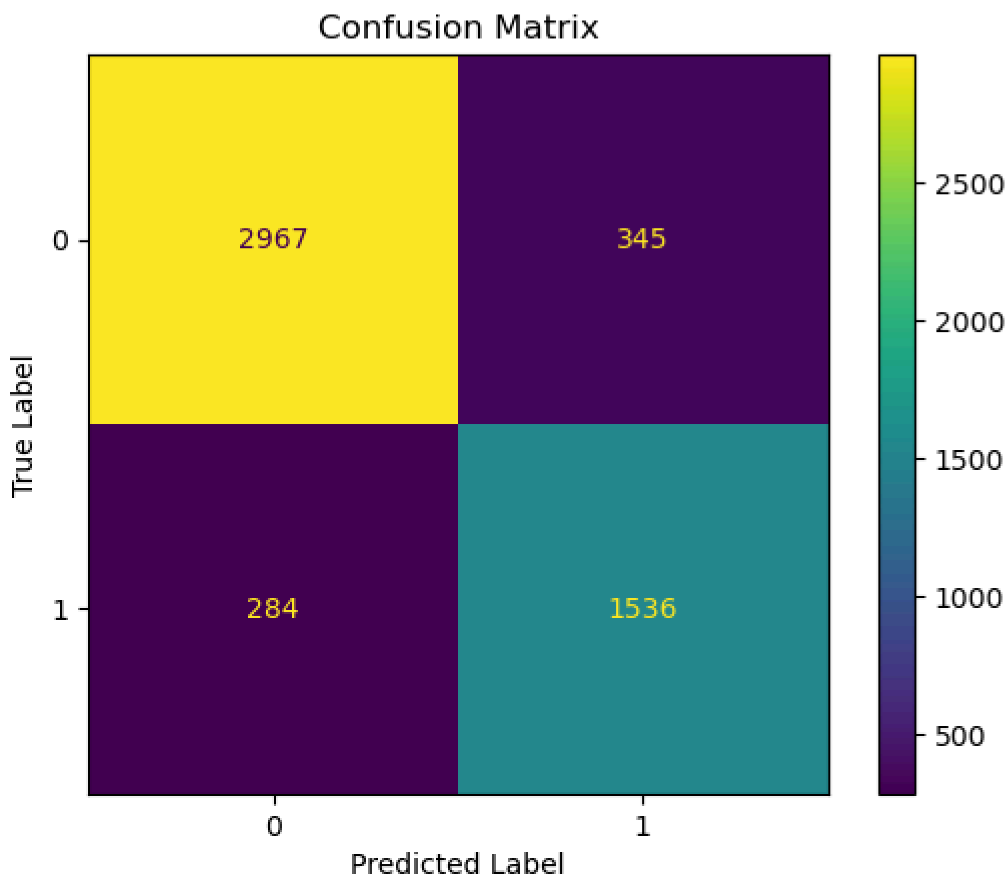
In [18]:

```
# Evaluate the model's performance
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred, zero_division=0))
```

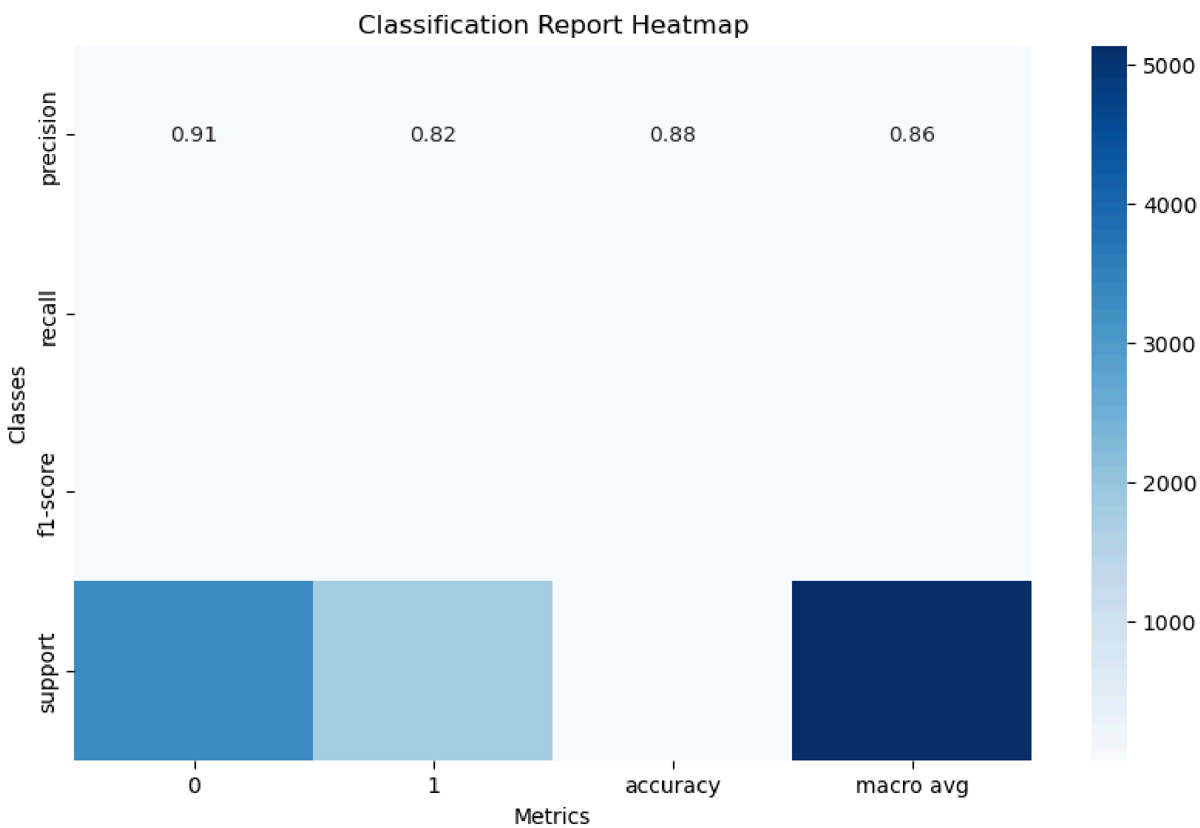
```
              precision    recall  f1-score   support

           0       0.91      0.90      0.90      3312
           1       0.82      0.84      0.83      1820

    accuracy                           0.88      5132
   macro avg       0.86      0.87      0.87      5132
weighted avg       0.88      0.88      0.88      5132
```

In [20]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

# Generate the confusion matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)

# Add labels and title
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



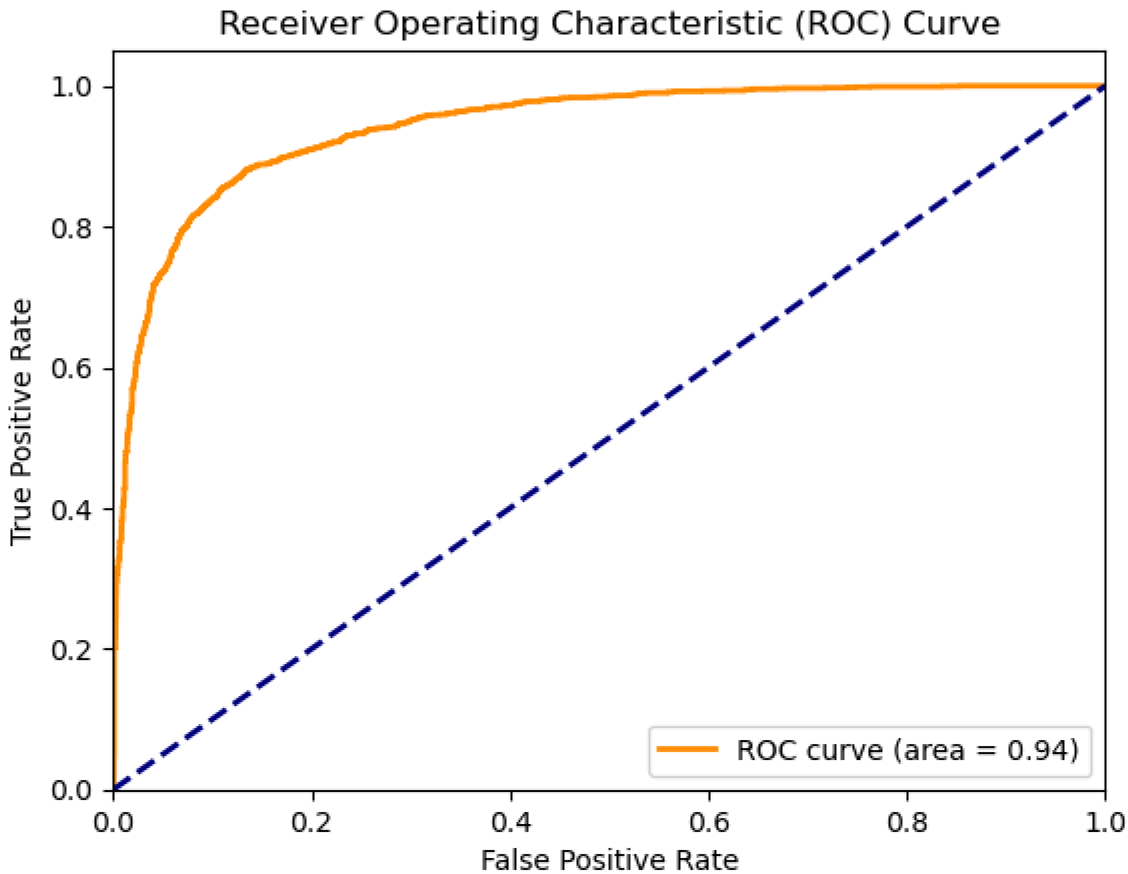
In [21]:
```
import seaborn as sns
import pandas as pd
from sklearn.metrics import classification_report

# Get the classification report as a dictionary
```

```
report = classification_report(y_test, y_pred, output_dict=True)

# Convert the classification report to a DataFrame
df_report = pd.DataFrame(report).transpose()

# Plot a heatmap of the classification report
plt.figure(figsize=(10, 6))
sns.heatmap(df_report.iloc[:-1, :].T, annot=True, cmap="Blues", fmt='.2f')

# Add labels and title
plt.title('Classification Report Heatmap')
plt.xlabel('Metrics')
plt.ylabel('Classes')
plt.show()
```



In [22]:
```
from sklearn.metrics import roc_curve, roc_auc_score

# Compute ROC curve and ROC area for class 1 (Fake News)
fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test_tfidf)[:
roc_auc = roc_auc_score(y_test, model.predict_proba(X_test_tfidf)[:, 1])

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

# Add labels and title
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



In [23]:
```
# ROC Curve:
# Interpretation: The ROC curve shows the true positive rate (recall) agains
# Your model has an AUC (Area Under the Curve) of 0.94, which indicates a st
# between real and fake news.

# Classification Report Heatmap:
# Interpretation: This heatmap visually represents the precision, recall, ar
# (class 0 and class 1), as well as overall accuracy.
# - Precision for class 0 (real news) is 0.91, and for class 1 (fake news) i
# - Recall for class 0 is 0.90, and for class 1 it's 0.84.
# This heatmap provides a quick overview of the model’s strengths and areas

# Confusion Matrix:
# Interpretation: The confusion matrix shows how many samples of each class
# - True Positives (Bottom Right): 1536 fake news articles were correctly cl
# - True Negatives (Top Left): 2967 real news articles were correctly classi
# - False Positives (Top Right): 345 real news articles were wrongly classif
# - False Negatives (Bottom Left): 284 fake news articles were wrongly class
# This matrix helps you understand where the model is performing well and wh
```

In [24]:
```
# Evaluate the model on the training set
y_train_pred = model.predict(X_train_tfidf)
```

```
# Evaluate the model on the test set
y_test_pred = model.predict(X_test_tfidf)

# Print the classification report for both sets
print("Training Set Performance:")
print(classification_report(y_train, y_train_pred, zero_division=0))

print("Test Set Performance:")
print(classification_report(y_test, y_test_pred, zero_division=0))
```

```
Training Set Performance:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93     13091
           1       0.86      0.90      0.88      7436

    accuracy                           0.91     20527
   macro avg       0.90      0.91      0.91     20527
weighted avg       0.91      0.91      0.91     20527

Test Set Performance:
              precision    recall  f1-score   support

           0       0.91      0.90      0.90      3312
           1       0.82      0.84      0.83      1820

    accuracy                           0.88      5132
   macro avg       0.86      0.87      0.87      5132
weighted avg       0.88      0.88      0.88      5132
```

In [25]:

```
# Analysis of Training and Test Set Performance:

# The training set performance is slightly better than the test set, but the
# - Training Accuracy: 0.91
# - Test Accuracy: 0.88
# The difference in precision, recall, and F1-scores between the training ar
# that the model is generalizing well and is not memorizing the training dat

# Overfitting Check:
# If the model were overfitting, we would expect the performance on the trai
# than the test set. Here, the difference is relatively small, indicating nc

# Underfitting Check:
# If the model were underfitting, it would perform poorly on both the traini
# However, the model achieves high accuracy on both sets (91% on training ar
# which suggests that it is capturing the underlying patterns in the data we

# Conclusion:
# The model is neither overfitting nor underfitting. It is generalizing well
```

In [26]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Initialize Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

```python
# Train the model on the resampled data (after SMOTE)
rf_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test_tfidf)

# Evaluate the Random Forest model
print("Random Forest Model Performance:")
print(classification_report(y_test, y_pred_rf, zero_division=0))
```

```
Random Forest Model Performance:
              precision    recall  f1-score   support

           0       0.90      0.91      0.91      3312
           1       0.84      0.82      0.83      1820

    accuracy                           0.88      5132
   macro avg       0.87      0.87      0.87      5132
weighted avg       0.88      0.88      0.88      5132
```

In [27]:
```python
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# Initialize Support Vector Classifier
svm_model = SVC(kernel='linear', probability=True, random_state=42)

# Train the model on the resampled data (after SMOTE)
svm_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred_svm = svm_model.predict(X_test_tfidf)

# Evaluate the SVM model
print("SVM Model Performance:")
print(classification_report(y_test, y_pred_svm, zero_division=0))
```

```
SVM Model Performance:
              precision    recall  f1-score   support

           0       0.91      0.89      0.90      3312
           1       0.80      0.84      0.82      1820

    accuracy                           0.87      5132
   macro avg       0.86      0.86      0.86      5132
weighted avg       0.87      0.87      0.87      5132
```

In [29]:
```python
pip install lightgbm
```

```
Collecting lightgbmNote: you may need to restart the kernel to use updated p
ackages.

  Downloading lightgbm-4.5.0-py3-none-win_amd64.whl.metadata (17 kB)
Requirement already satisfied: numpy>=1.17.0 in c:\users\enchong\anaconda3\l
ib\site-packages (from lightgbm) (1.26.4)
Requirement already satisfied: scipy in c:\users\enchong\anaconda3\lib\site-
packages (from lightgbm) (1.11.4)
Downloading lightgbm-4.5.0-py3-none-win_amd64.whl (1.4 MB)
   ---------------------------------------- 0.0/1.4 MB ? eta -:--:--
    --------------------------------------- 0.0/1.4 MB 660.6 kB/s eta 0:00:0
3
   -------------- ------------------------- 0.5/1.4 MB 6.4 MB/s eta 0:00:01
   ----------------------------- ---------- 1.1/1.4 MB 8.4 MB/s eta 0:00:01
   ---------------------------------------- 1.4/1.4 MB 9.2 MB/s eta 0:00:00
Installing collected packages: lightgbm
Successfully installed lightgbm-4.5.0
```

In [30]:

```python
from lightgbm import LGBMClassifier
from sklearn.metrics import classification_report

# Initialize LightGBM model
lgbm_model = LGBMClassifier(random_state=42)

# Train the model on the resampled data (after SMOTE)
lgbm_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred_lgbm = lgbm_model.predict(X_test_tfidf)

# Evaluate the LightGBM model
print("LightGBM Model Performance:")
print(classification_report(y_test, y_pred_lgbm, zero_division=0))
```

```
[LightGBM] [Info] Number of positive: 13091, number of negative: 13091
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.364445 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 146241
[LightGBM] [Info] Number of data points in the train set: 26182, number of u
sed features: 4535
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.0000
00
LightGBM Model Performance:
              precision    recall  f1-score   support

           0       0.89      0.92      0.91      3312
           1       0.85      0.80      0.83      1820

    accuracy                           0.88      5132
   macro avg       0.87      0.86      0.87      5132
weighted avg       0.88      0.88      0.88      5132
```

In [31]:
```python
from sklearn.model_selection import GridSearchCV
from lightgbm import LGBMClassifier

# Set up the parameter grid for tuning
param_grid = {
    'num_leaves': [31, 50],
    'max_depth': [-1, 10, 20],
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [100, 200, 500]
}

# Initialize the LightGBM model
lgbm_model = LGBMClassifier(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=lgbm_model, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2)

# Train with grid search
grid_search.fit(X_train_resampled, y_train_resampled)

# Best parameters from grid search
print("Best Parameters from Grid Search:", grid_search.best_params_)

# Evaluate the best model
best_lgbm_model = grid_search.best_estimator_
y_pred_lgbm = best_lgbm_model.predict(X_test_tfidf)

# Evaluate performance
print(classification_report(y_test, y_pred_lgbm, zero_division=0))
```

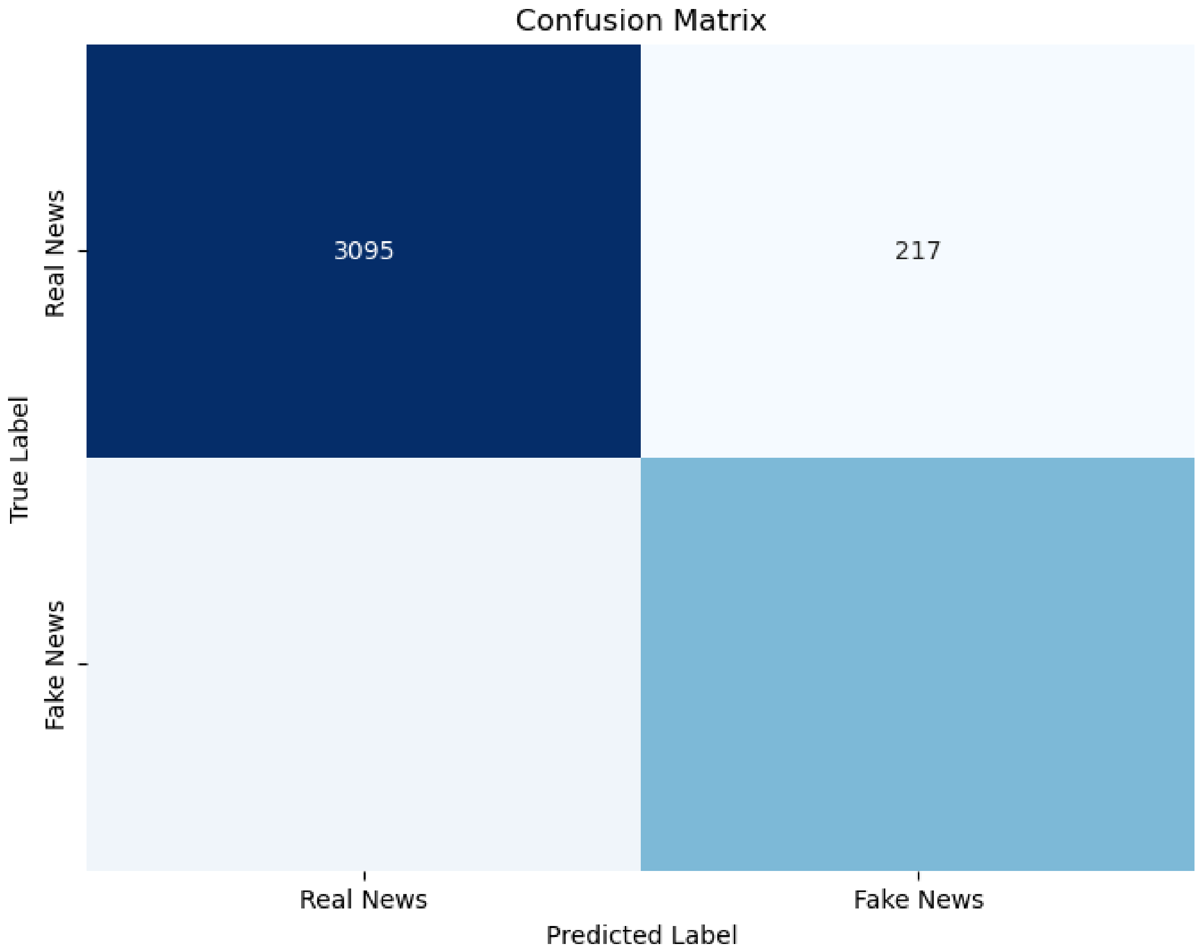
```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
[LightGBM] [Info] Number of positive: 13091, number of negative: 13091
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.275957 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 146241
[LightGBM] [Info] Number of data points in the train set: 26182, number of u
sed features: 4535
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.0000
00
Best Parameters from Grid Search: {'learning_rate': 0.05, 'max_depth': -1,
'n_estimators': 500, 'num_leaves': 50}
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      3312
           1       0.87      0.83      0.85      1820

    accuracy                           0.90      5132
   macro avg       0.89      0.88      0.89      5132
weighted avg       0.90      0.90      0.90      5132
```

In [34]:
```python
# LightGBM Model Performance After Hyperparameter Tuning (using GridSearchCV
# Best Parameters Found from Grid Search:
```

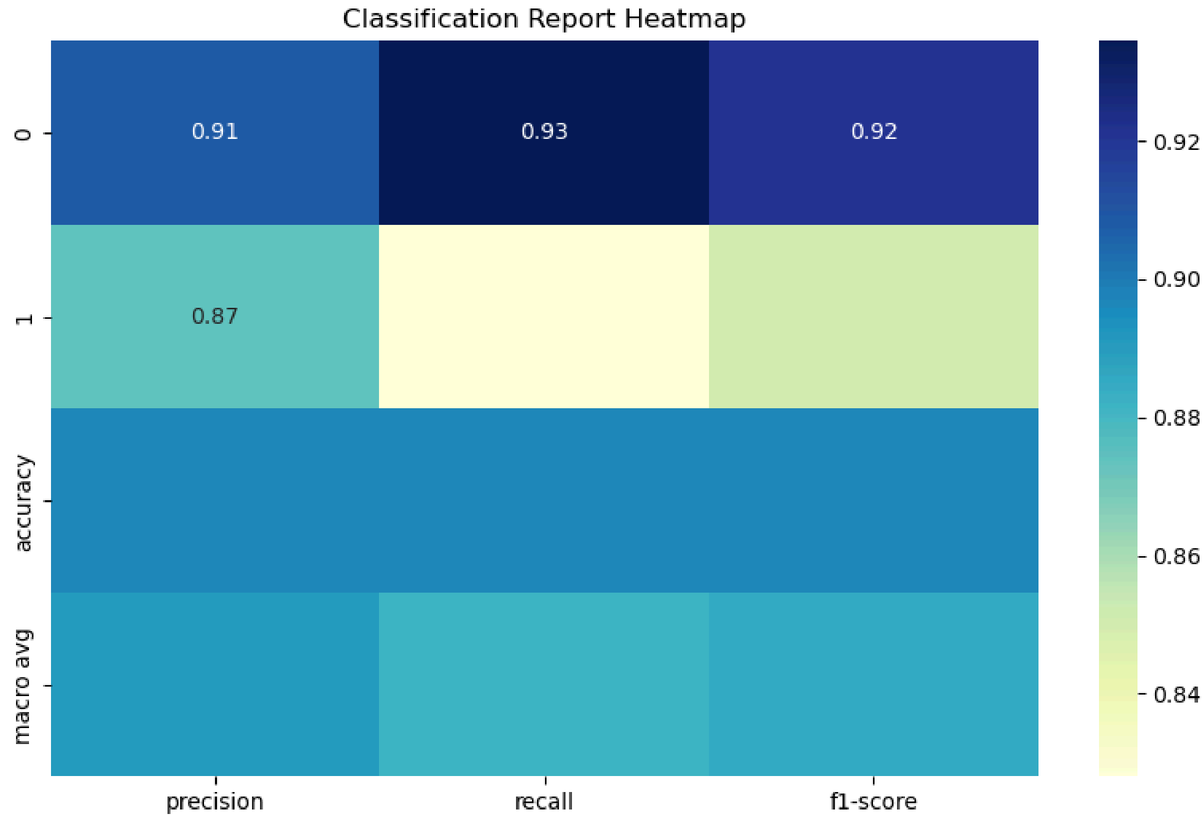
```
# - Learning Rate: 0.05
# - Max Depth: -1 (no depth limit)
# - Number of Estimators: 500
# - Number of Leaves: 50

# Performance Metrics:
# Class 0 (Real News):
# - Precision: 0.91 – 91% of articles predicted as real news were actually r
# - Recall: 0.93 – 93% of all actual real news articles were correctly ident
# - F1-Score: 0.92 – A good balance between precision and recall for real ne

# Class 1 (Fake News):
# - Precision: 0.87 – 87% of articles predicted as fake news were actually f
# - Recall: 0.83 – The model correctly identified 83% of all fake news artic
# - F1-Score: 0.85 – A strong balance between precision and recall for fake

# Overall Model Performance:
# - Accuracy: 0.90 – The model correctly classified 90% of all articles in t
# - Macro Average F1-Score: 0.88
# - Weighted Average F1-Score: 0.90

# Conclusion:
# The LightGBM model performed exceptionally well after hyperparameter tunin
# and a good balance between precision and recall for both real and fake new
```

In [37]:

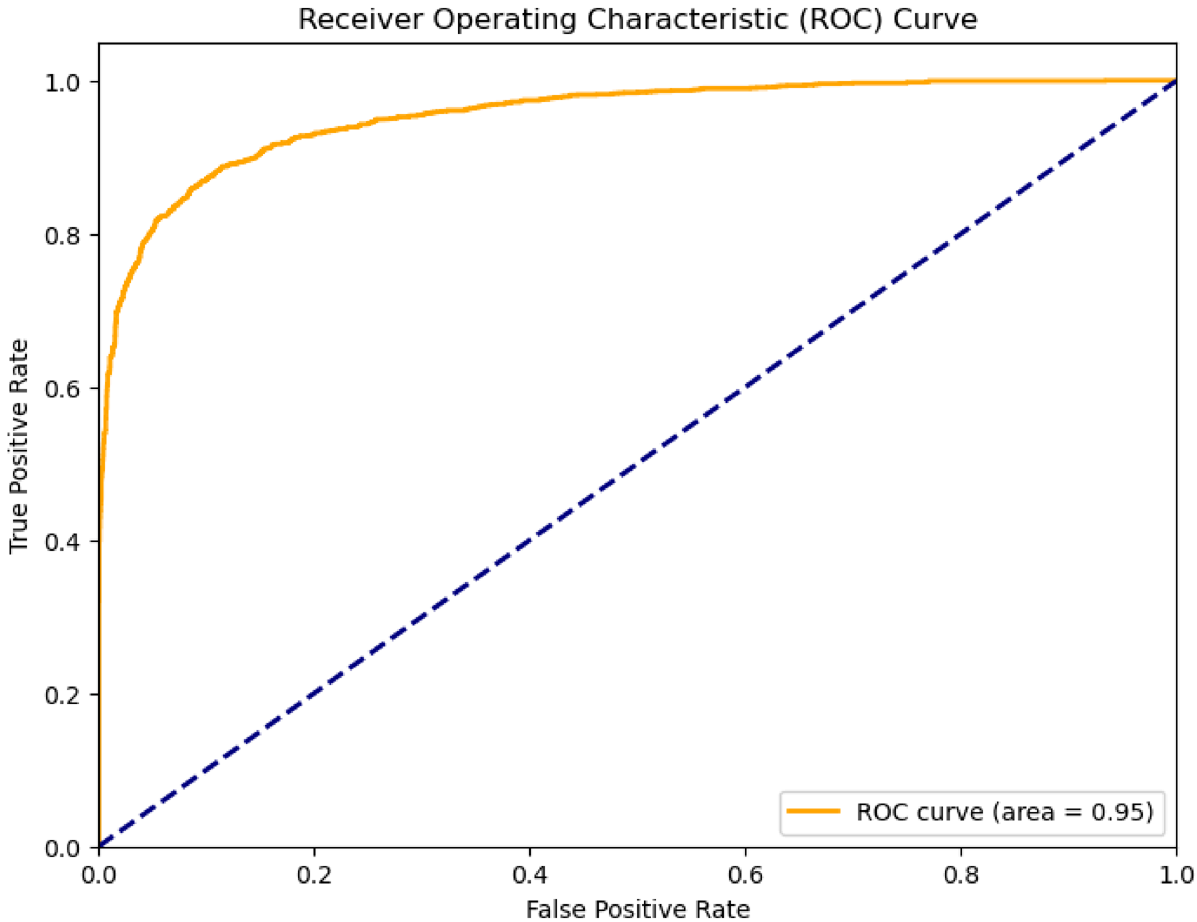
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_lgbm)

# Plot confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False, xtic
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
plt.show()
```



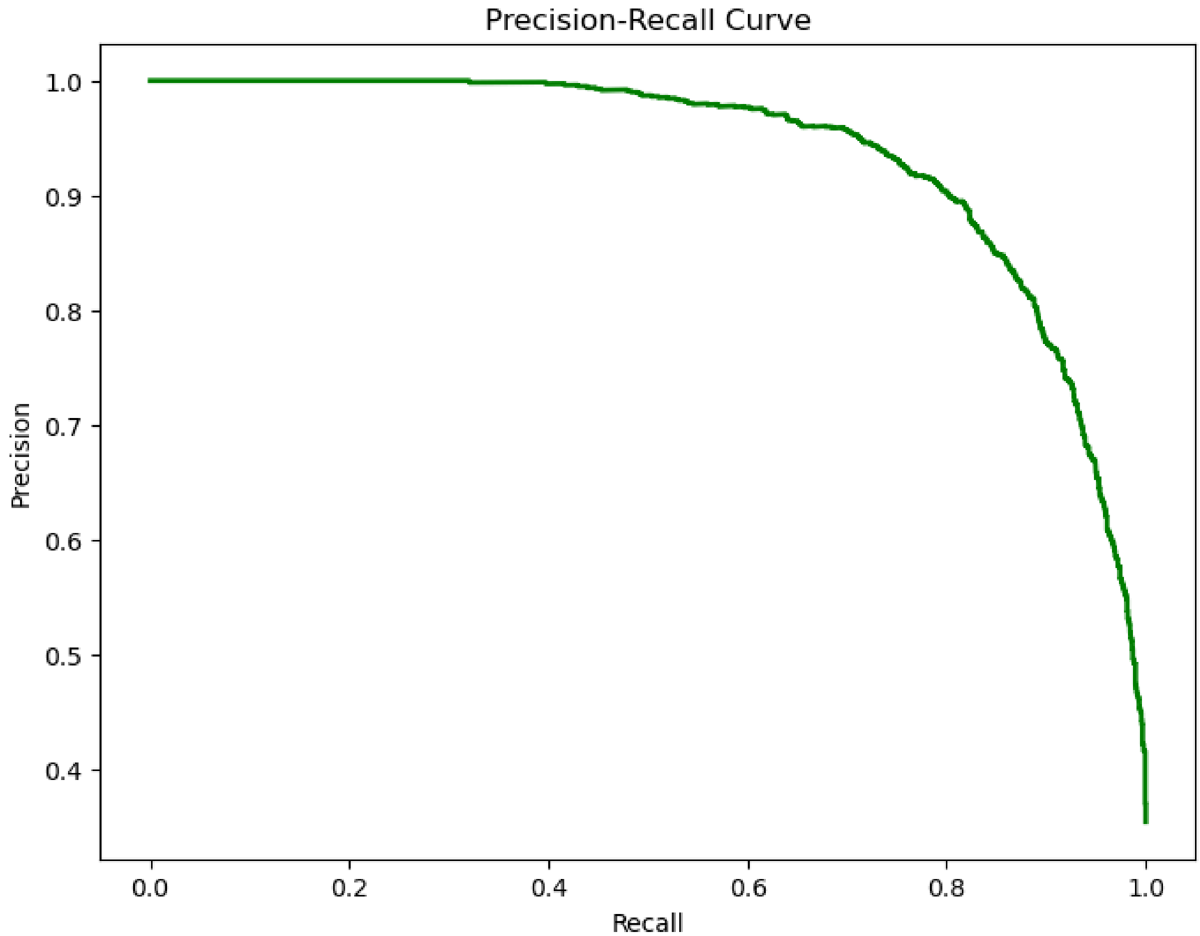
In [38]:
```
from sklearn.metrics import classification_report
import pandas as pd

# Get classification report as a dictionary
report = classification_report(y_test, y_pred_lgbm, output_dict=True)

# Convert the classification report into a DataFrame
df_report = pd.DataFrame(report).transpose()

# Plot classification report heatmap
plt.figure(figsize=(10,6))
sns.heatmap(df_report.iloc[:-1, :-1], annot=True, cmap="YlGnBu")
plt.title('Classification Report Heatmap')
plt.show()
```



In [39]:
```
from sklearn.metrics import roc_curve, auc

# Compute ROC curve and area for class 1 (Fake News)
fpr, tpr, thresholds = roc_curve(y_test, best_lgbm_model.predict_proba(X_tes
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, color='orange', lw=2, label='ROC curve (area = {:.2f})'.f
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')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



In [40]:
```
from sklearn.metrics import precision_recall_curve

# Compute precision-recall curve
precision, recall, _ = precision_recall_curve(y_test, best_lgbm_model.predic

# Plot precision-recall curve
plt.figure(figsize=(8,6))
plt.plot(recall, precision, color='green', lw=2)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()
```



In [42]:
```
# Make predictions on the resampled training set
y_train_pred_lgbm = best_lgbm_model.predict(X_train_resampled)

# Evaluate performance on the resampled training set
print("Training Set Performance:")
print(classification_report(y_train_resampled, y_train_pred_lgbm, zero_divis

# Make predictions on the test set
y_test_pred_lgbm = best_lgbm_model.predict(X_test_tfidf)

# Evaluate performance on the test set
print("Test Set Performance:")
print(classification_report(y_test, y_test_pred_lgbm, zero_division=0))
```

```
Training Set Performance:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     13091
           1       0.98      0.96      0.97     13091

    accuracy                           0.97     26182
   macro avg       0.97      0.97      0.97     26182
weighted avg       0.97      0.97      0.97     26182

Test Set Performance:
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      3312
           1       0.87      0.83      0.85      1820

    accuracy                           0.90      5132
   macro avg       0.89      0.88      0.89      5132
weighted avg       0.90      0.90      0.90      5132
```

In [35]:

```
import joblib

# Save the LightGBM model
joblib.dump(best_lgbm_model, 'lightgbm_model.pkl')

# To load the model later
# loaded_model = joblib.load('lightgbm_model.pkl')
```

Out[35]:

```
['lightgbm_model.pkl']
```