Code

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
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 accuracy_score, precision_score, recall_score,
f1_score

nltk.download('stopwords')
nltk.download('wordnet')

# Load the datasets
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Text preprocessing function
def preprocess_text(text):
    if isinstance(text, float):
        return ""
    text = re.sub(r'\W', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    text = text.lower()

    lemmatizer = WordNetLemmatizer()
    tokens = text.split()
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in
set(stopwords.words('english'))]

    return ' '.join(tokens)

# Apply preprocessing to the datasets
train_data['clean_text'] = train_data['crimeaditionalinfo'].apply(preprocess_text)
test_data['clean_text'] = test_data['crimeaditionalinfo'].apply(preprocess_text)
```

```
# Feature extraction using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(train_data['clean_text']).toarray()
X_test = vectorizer.transform(test_data['clean_text']).toarray()
y_train = train_data['category']

# Model training with Logistic Regression
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(test_data['category'], y_pred)
precision = precision_score(test_data['category'], y_pred, average='weighted')
recall = recall_score(test_data['category'], y_pred, average='weighted')
f1 = f1_score(test_data['category'], y_pred, average='weighted')

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")

# Save preprocessed data and predictions
train_data.to_csv('preprocessed_train.csv', index=False)
test_data['predictions'] = y_pred
test_data.to_csv('predictions_test.csv', index=False)

# Save preprocessed data with sub-category and category
preprocessed_data = test_data[['crimeaditionalinfo', 'clean_text', 'category',
'predictions']]
preprocessed_data.to_csv('final_preprocessed_data.csv', index=False)
```

Key Steps in the Code

1. **Importing Libraries**:

   Core Python libraries like pandas, numpy, and re are used for data manipulation and regular expression processing.

   Natural Language Toolkit (nltk) is used for stopword removal and lemmatization.

   Scikit-learn libraries are used for feature extraction (TF-IDF), training the model (Logistic Regression), and evaluating its performance.

2. **Dataset Loading**:

   Training (train.csv) and testing (test.csv) datasets are loaded using pandas.

3. **Text Preprocessing**:

   A custom function preprocess_text is defined to:

      Remove special characters using regex.

      Normalize whitespaces and convert text to lowercase.

      Remove stopwords (common words like *"the"*, *"is"*) using the NLTK stopwords.words list.

      Lemmatize words using WordNetLemmatizer to reduce them to their base form (e.g., *"running"* → *"run"*).

Preprocessing is applied to the crimeaditionalinfo column in both datasets, creating a new clean_text column.

4. **Feature Extraction**:

   Text data is transformed into numerical features using **TF-IDF Vectorization**:

   The TfidfVectorizer converts the cleaned text into a matrix of TF-IDF features with a maximum of 5000 features.

5. **Model Training**:

   A Logistic Regression model is trained using the TF-IDF features (X_train) and the labels (y_train) from the training dataset.

   The model uses 10,000 iterations for optimization to ensure convergence.

6. **Predictions and Evaluations**:

   Predictions (y_pred) are made on the test data (X_test).

   The model's performance is evaluated using the following metrics:

   Accuracy: The ratio of correct predictions to total predictions.

   Precision: How many of the predicted categories are actually correct (weighted average across all categories).

   Recall: The proportion of actual categories correctly identified by the model.

F1-Score: Harmonic mean of precision and recall, providing a balanced measure of model performance.

7. **Saving Results**:

   The preprocessed data from the training set is saved as preprocessed_train.csv.

   The predictions on the test set are added as a new column (predictions) and saved as predictions_test.csv.

   A consolidated file (final_preprocessed_data.csv) containing both original and preprocessed text, actual categories, and predictions is also saved.

Model Evaluation Output

The reported metrics indicate how well the model performed on the test set:

Accuracy: 75.57% — The model correctly predicted the category for about 76% of the test cases.

Precision: 71.55% — On average, 71.55% of the model's predictions were correct across all categories.

Recall: 75.57% — The model successfully identified 75.57% of the actual categories in the dataset.

F1-Score: 72.26% — A balance between precision and recall, reflecting good overall performance.

Key Observations

Strengths:

The preprocessing effectively cleans the text for feature extraction.

The Logistic Regression model performs reasonably well with balanced metrics across precision, recall, and F1-score.

Limitations:

Accuracy is only 75.57%, indicating room for improvement. Advanced models (e.g., SVMs, neural networks) or parameter tuning could improve performance.

TF-IDF vectorization with a fixed max_features might limit the model's ability to capture all useful patterns in the text.

Potential Improvements:

Use hyperparameter tuning for the Logistic Regression model (e.g., grid search for the regularization parameter).

Explore advanced vectorization techniques like word embeddings (Word2Vec, GloVe) for better feature extraction.

Balance the dataset if it is imbalanced, as this can skew precision/recall metrics.