```
# Data handling
import pandas as pd        # For loading and manipulating dataset
import numpy as np         # For numerical operations

# Text preprocessing
import re                  # For removing punctuation using regex
import string              # For handling string operations

# Stopwords
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

# Feature extraction
from sklearn.feature_extraction.text import TfidfVectorizer  # Convert text to numerical features

# Train-test split
from sklearn.model_selection import train_test_split

# Naive Bayes model
from sklearn.naive_bayes import MultinomialNB

# Evaluation metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_ma

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load dataset (Change path if needed)
# Assuming the CSV is comma-separated and has a header.
# The actual text content appears to be in a column named 'text'.
# The actual label content appears to be in a column named 'label'.
data = pd.read_csv("/content/news.csv", sep=',')

# Select the 'text' and 'label' columns and rename 'text' to 'message'
data = data[['text', 'label']].rename(columns={'text': 'message'})

# Display first 5 rows
print("First 5 Samples:\n")
print(data.head())

# Dataset size
print("\nDataset Shape:", data.shape)

# Class distribution
print("\nClass Distribution:\n")
print(data['label'].value_counts())
```

```
First 5 Samples:

                                             message label
0  Daniel Greenfield, a Shillman Journalism Fello...  FAKE
1  Google Pinterest Digg Linkedin Reddit Stumbleu...  FAKE
2  U.S. Secretary of State John F. Kerry said Mon...  REAL
3  — Kaydee King (@KaydeeKing) November 9, 2016 T...  FAKE
4  It's primary day in New York and front-runners...  REAL

Dataset Shape: (6335, 2)
```

```
Class Distribution:

label
REAL    3171
FAKE    3164
Name: count, dtype: int64
```

```
# Function to clean text
def preprocess_text(text):

    # Handle non-string inputs (e.g., NaN values)
    if not isinstance(text, str):
        return ""

    # Convert to lowercase
    text = text.lower()

    # Remove punctuation
    text = re.sub(f"[{string.punctuation}]", "", text)

    # Remove stopwords
    words = text.split()
    words = [word for word in words if word not in ENGLISH_STOP_WORDS]

    return " ".join(words)

# Apply preprocessing
data['clean_message'] = data['message'].apply(preprocess_text)

print("\nCleaned Sample:\n")
print(data[['message','clean_message']].head())

print("\nDataFrame Info after adding 'clean_message' column:\n")
data.info()
```

```
Cleaned Sample:

                                             message  \
0  Daniel Greenfield, a Shillman Journalism Fello...
1  Google Pinterest Digg Linkedin Reddit Stumbleu...
2  U.S. Secretary of State John F. Kerry said Mon...
3  — Kaydee King (@KaydeeKing) November 9, 2016 T...
4  It's primary day in New York and front-runners...

                                       clean_message
0  daniel greenfield shillman journalism fellow f...
1  google pinterest digg linkedin reddit stumbleu...
2  secretary state john f kerry said monday stop ...
3  — kaydee king kaydeeking november 9 2016 lesso...
4  primary day new york frontrunners hillary clin...

DataFrame Info after adding 'clean_message' column:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6335 entries, 0 to 6334
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   message        6335 non-null   object
 1   label          6335 non-null   object
 2   clean_message  6335 non-null   object
dtypes: object(3)
```

```
memory usage: 148.6+ KB
```

```
# Initialize TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Convert text into numerical feature matrix
X = vectorizer.fit_transform(data['clean_message'])

# Target variable
y = data['label']

print("Feature Matrix Shape:", X.shape)
print("Sample Feature Names:", vectorizer.get_feature_names_out()[:10])
```

```
Feature Matrix Shape: (6335, 84068)
Sample Feature Names: ['00' '000' '0000' '0000000031' '000000031' '0000035' '00001400' '00006'
 '00011' '00017b2908ff9fa45188d243fd49aaeeb2dhrcofficecom']
```

```
# Split data into training and testing sets
# Using 80% for training and 20% for testing, with a random state for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (5068, 84068)
X_test shape: (1267, 84068)
y_train shape: (5068,)
y_test shape: (1267,)
```

```
# Initialize Multinomial Naive Bayes
model = MultinomialNB()

# Train model
model.fit(X_train, y_train)

print("Model Parameters:\n", model.get_params())
```

```
Model Parameters:
 {'alpha': 1.0, 'class_prior': None, 'fit_prior': True, 'force_alpha': True}
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for Logistic Regression
# Using 'liblinear' solver for 'l1' penalty which is good for small datasets
param_grid_log_reg = {
    'C': [0.1, 1, 10, 100],
    'penalty': ['l1', 'l2']
}

# Initialize Logistic Regression model
# Set solver to 'liblinear' as it supports both 'l1' and 'l2' penalties
# and is good for relatively small datasets.
log_reg = LogisticRegression(solver='liblinear', random_state=42, max_iter=1000)

# Initialize GridSearchCV
grid_search_log_reg = GridSearchCV(log_reg, param_grid_log_reg, cv=5, scoring='accuracy', verbose
# Fit GridSearchCV to the training data
grid_search_log_reg.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters for Logistic Regression:", grid_search_log_reg.best_params_)
print("Best cross-validation accuracy for Logistic Regression:", grid_search_log_reg.best_score_)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for Logistic Regression: {'C': 100, 'penalty': 'l2'}
Best cross-validation accuracy for Logistic Regression: 0.9327170842168184
```

```
best_log_reg_model = grid_search_log_reg.best_estimator_
y_pred_tuned_log_reg = best_log_reg_model.predict(X_test)

accuracy_tuned_log_reg = accuracy_score(y_test, y_pred_tuned_log_reg)
precision_tuned_log_reg = precision_score(y_test, y_pred_tuned_log_reg, pos_label="FAKE")
recall_tuned_log_reg = recall_score(y_test, y_pred_tuned_log_reg, pos_label="FAKE")
f1_tuned_log_reg = f1_score(y_test, y_pred_tuned_log_reg, pos_label="FAKE")

print("Accuracy (Tuned Logistic Regression):", accuracy_tuned_log_reg)
print("Precision (Tuned Logistic Regression):", precision_tuned_log_reg)
print("Recall (Tuned Logistic Regression):", recall_tuned_log_reg)
print("F1-Score (Tuned Logistic Regression):", f1_tuned_log_reg)

print("\nClassification Report (Tuned Logistic Regression):\n")
print(classification_report(y_test, y_pred_tuned_log_reg))

cm_tuned_log_reg = confusion_matrix(y_test, y_pred_tuned_log_reg)

plt.figure(figsize=(5,4))
sns.heatmap(cm_tuned_log_reg, annot=True, fmt='d', cmap='Blues',
            xticklabels=best_log_reg_model.classes_,
            yticklabels=best_log_reg_model.classes_)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix for Tuned Logistic Regression")
plt.show()
```

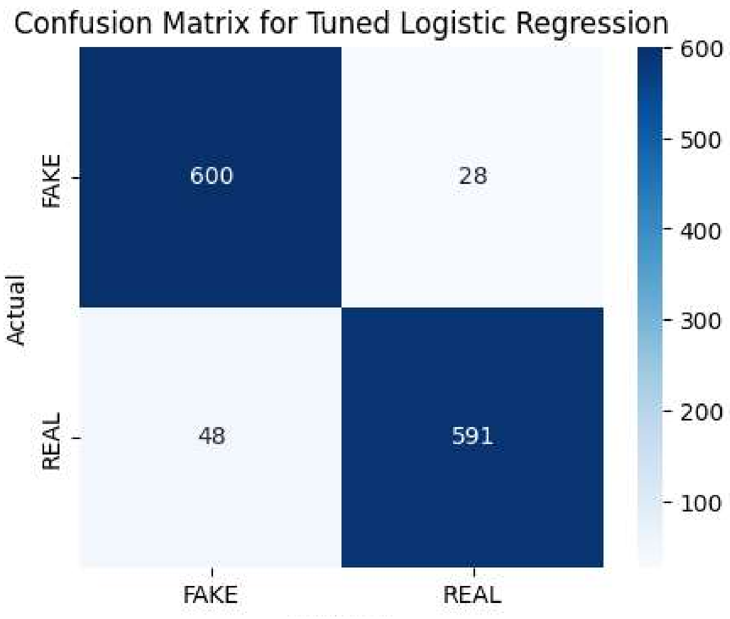
```
Accuracy (Tuned Logistic Regression): 0.9400157853196527
Precision (Tuned Logistic Regression): 0.9259259259259259
Recall (Tuned Logistic Regression): 0.9554140127388535
F1-Score (Tuned Logistic Regression): 0.9404388714733543

Classification Report (Tuned Logistic Regression):

              precision    recall  f1-score   support

        FAKE       0.93      0.96      0.94       628
        REAL       0.95      0.92      0.94       639

    accuracy                           0.94      1267
   macro avg       0.94      0.94      0.94      1267
weighted avg       0.94      0.94      0.94      1267
```


Confusion Matrix for Tuned Logistic Regression
Actual
FAKE
REAL
FAKE
REAL