```
# --- 1. Install and Import Libraries (Corrected Import) ---
# Install NLTK (already satisfied, but kept for completeness)
!pip install nltk

import pandas as pd
import numpy as np
import re
# FIX: Corrected typo from 'model_model_selection' to 'model_selection'
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import nltk
from nltk.corpus import stopwords
import warnings
warnings.filterwarnings('ignore')

# NLTK Download Logic:
try:
    # Check if 'stopwords' is available
    stopwords.words('english')
except LookupError:
    # Download 'stopwords' if not found
    print("NLTK 'stopwords' not found. Downloading...")
    nltk.download('stopwords')

# Define stopwords set
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

# --- 2. Define Dataset URL ---
# Using a widely known fake news dataset hosted on a stable raw data source.
DATASET_URL = 'https://raw.githubusercontent.com/mfj05/Fake-News-Detection/main/news.csv'

# --- 3. Load Data with Error Handling and Fallback ---

print("\n--- Loading Data ---")
df = None # Initialize df

try:
    # Attempt to load the combined dataset
    df = pd.read_csv(DATASET_URL)
    print("Dataset successfully loaded from URL.")

    # Data Cleaning specific to this dataset
    df = df.iloc[:, 1:].sample(frac=1, random_state=42).reset_index(drop=True)
    df['label'] = df['label'].map({'FAKE': 0, 'REAL': 1})
    df.dropna(inplace=True)

except Exception as e:
    # FALLBACK MECHANISM: Create a small dummy DataFrame if URL loading fails
    print(f"URL loading failed ({type(e).__name__}: {e}). Using dummy DataFrame for demonstration.")
    data = {
        'title': [
            "Aliens Land: Government Hides Cure",
            "Stock Market Surges on Tech Earnings",
            "New Virus Discovered on Moon Rock",
            "Local Council Approves New Park Construction"
        ],
        'text': [
            "A spaceship landed in the US, and a secret group is suppressing the news about a universal cure.",
            "Major indices closed higher today following strong quarterly reports from leading technology companies.
            "Scientists claim a new strain of bacteria was found on a sample returned from a lunar mission, posing n
            "The town's board voted 5-2 to proceed with the planned green space project after a public consultation.
        ],
        'label': [0, 1, 0, 1]  # 0 for Fake, 1 for Real
    }
    df = pd.DataFrame(data)

# --- Continue Preprocessing with defined 'df' ---

print(f"Dataset Loaded. Total Articles: {len(df)}")
print(f"First 5 rows of the data:\n{df[['title', 'label']].head()}")
print(f"\nLabel Distribution:\n{df['label'].value_counts()}")

# --- 4. Data Cleaning and Feature Engineering ---

def clean_text(text):
    """
    Function to clean and preprocess the text.
    """
```

```
        text = str(text)

        # Remove all characters except letters a-z and A-Z
        text = re.sub(r'[^a-zA-Z\s]', '', text, re.I | re.A)

        # Convert to lowercase
        text = text.lower()

        # Tokenize and remove stop words
        tokens = text.split()
        tokens = [word for word in tokens if word not in stop_words]

        return ' '.join(tokens)

    # Combine 'title' and 'text' for the classification feature
    df['full_text'] = df['title'] + ' ' + df['text']
    df['cleaned_text'] = df['full_text'].apply(clean_text)

    # Define feature (X) and target (y)
    X = df['cleaned_text']
    y = df['label']

    print("\nText Cleaning Complete.")

    # --- 5. Data Splitting & Feature Extraction (TF-IDF) ---
    # Split the dataset into training and testing sets (80% train, 20% test)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    # TF-IDF Vectorizer
    # Note: min_df=5 might be too high for the dummy dataset, but is fine for the full set.
    tfidf_vectorizer = TfidfVectorizer(
        stop_words='english',
        max_df=0.8,
        min_df=1 # Changed to 1 to work with the small dummy data if needed
    )

    # Fit and transform the training data
    tfidf_train = tfidf_vectorizer.fit_transform(X_train)

    # Transform the testing data
    tfidf_test = tfidf_vectorizer.transform(X_test)

    print(f"\nTF-IDF Matrix Shape (Train): {tfidf_train.shape}")

    # --- 6. Model Training (Passive-Aggressive Classifier) ---
    pac = PassiveAggressiveClassifier(max_iter=50, random_state=42)

    # Train the model
    pac.fit(tfidf_train, y_train)

    print("\nPassive-Aggressive Classifier Trained.")

    # --- 7. Model Evaluation ---
    y_pred = pac.predict(tfidf_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"\nModel Accuracy on Test Set: {round(accuracy * 100, 4)}%")

    # Generate a confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred, labels=[0, 1])
    print("\nConfusion Matrix (Rows=Actual, Columns=Predicted):")
    print("        Predicted Fake (0) | Predicted Real (1)")
    print("Actual Fake (0):", conf_matrix[0])
    print("Actual Real (1):", conf_matrix[1])

    # --- 8. Live Prediction Function ---

    def predict_news_authenticity(news_title, news_text):
        """
        Takes a new article's title and text, cleans it, vectorizes it, and predicts its label.
        """
        # 1. Combine and Clean
        article_to_clean = news_title + ' ' + news_text
        cleaned_article = clean_text(article_to_clean)

        # 2. Vectorize using the trained TF-IDF vectorizer
        article_vector = tfidf_vectorizer.transform([cleaned_article])

        # 3. Predict
        prediction = pac.predict(article_vector)[0]
```

```
    # 4. Return result
    return "REAL NEWS ✅" if prediction == 1 else "FAKE NEWS ❌"

# Example Tests:
print("\n--- Testing on Custom Articles ---")

# Test 1: Highly suspicious claim (likely FAKE)
fake_title = "Secret Society Controls World's Weather"
fake_text = "A leaked memo suggests that an elite group meets weekly to decide where and when to trigger major storm
print(f"'{fake_title}' -> Prediction: {predict_news_authenticity(fake_title, fake_text)}")

# Test 2: Standard news report (likely REAL)
real_title = "Local Firefighters Extinguish Warehouse Blaze"
real_text = "Emergency services responded quickly to a three-alarm fire late last night at an industrial complex. No
print(f"'{real_title}' -> Prediction: {predict_news_authenticity(real_title, real_text)}")
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.2.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)

--- Loading Data ---
URL loading failed (HTTPError: HTTP Error 404: Not Found). Using dummy DataFrame for demonstration.
Dataset Loaded. Total Articles: 4
First 5 rows of the data:
                                    title  label
0           Aliens Land: Government Hides Cure      0
1         Stock Market Surges on Tech Earnings      1
2            New Virus Discovered on Moon Rock      0
3  Local Council Approves New Park Construction      1

Label Distribution:
label
0    2
1    2
Name: count, dtype: int64

Text Cleaning Complete.

TF-IDF Matrix Shape (Train): (3, 42)

Passive-Aggressive Classifier Trained.

Model Accuracy on Test Set: 0.0%

Confusion Matrix (Rows=Actual, Columns=Predicted):
        Predicted Fake (0) | Predicted Real (1)
Actual Fake (0): [0 0]
Actual Real (1): [1 0]

--- Testing on Custom Articles ---
'Secret Society Controls World's Weather' -> Prediction: FAKE NEWS ❌
'Local Firefighters Extinguish Warehouse Blaze' -> Prediction: REAL NEWS ✅
```