import pandas as pd

import re

from collections import Counter

from textstat import flesch\_kincaid\_grade, gunning\_fog, smog\_index

from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline

*# Step 1: Load and Preprocess Data*

**def** load\_and\_clean\_data(file\_path):

    """

    Load data from a CSV file and clean the text for analysis.

    """

    try:

        *# Load the CSV file*

        data = pd.read\_csv(file\_path, encoding='latin1')  *# Adjust encoding as needed*

        print("Data loaded successfully.")

        *# Clean text function*

        **def** clean\_text(text):

            text = re.sub(**r**'[^\w\s]', '', str(text))  *# Remove punctuation*

            text = re.sub(**r**'\s+', ' ', text)  *# Normalize whitespace*

            return text.lower()

        *# Apply cleaning*

        data['cleaned\_message'] = data['message'].apply(clean\_text)

        print("Text cleaning complete.")

        return data

    except Exception as e:

        print(**f**"Error loading or cleaning data: {e}")

        return None

*# Step 2: Analyze Text Complexity*

**def** add\_readability\_metrics(data):

    """

    Add readability metrics to the DataFrame.

    """

    try:

        data['flesch\_kincaid'] = data['cleaned\_message'].apply(flesch\_kincaid\_grade)

        data['gunning\_fog'] = data['cleaned\_message'].apply(gunning\_fog)

        data['smog\_index'] = data['cleaned\_message'].apply(smog\_index)

        print("Readability metrics added.")

    except Exception as e:

        print(**f**"Error calculating readability metrics: {e}")

    return data

*# Step 3: Analyze Repetition Patterns*

**def** add\_repetition\_score(data):

    """

    Calculate and add a repetition score for each message.

    """

    **def** repetition\_score(text):

        words = text.split()

        word\_counts = Counter(words)

        unique\_words = len(word\_counts)

        total\_words = len(words)

        return 1 - (unique\_words / total\_words) if total\_words > 0 else 0

    data['repetition\_score'] = data['cleaned\_message'].apply(repetition\_score)

    print("Repetition scores added.")

    return data

*# Step 4: Detect AI Probability using roberta-base-openai-detector*

**def** detect\_ai\_probability(data):

    """

    Use a pre-trained transformer model to predict AI probability for each message.

    Handles long messages and sequence mismatches.

    """

    *# Load the tokenizer and model*

    tokenizer = AutoTokenizer.from\_pretrained("roberta-base-openai-detector")

    model = AutoModelForSequenceClassification.from\_pretrained("roberta-base-openai-detector")

    classifier = pipeline("text-classification", model=model, tokenizer=tokenizer)

    **def** detect(message):

        try:

            *# Handle long messages by truncating to max length*

            prediction = classifier(message, truncation=True, max\_length=512)

            return prediction[0]['score']

        except Exception as e:

            print(**f**"Error processing message: {message}\n{e}")

            return 0.0

    data['ai\_probability'] = data['cleaned\_message'].apply(detect)

    print("AI probabilities added.")

    return data

*# Step 5: Aggregate Metrics into AI Rating*

**def** aggregate\_ai\_rating(data):

    """

    Combine various metrics into a single AI rating.

    """

    try:

        *# Normalize readability metrics*

        data['normalized\_readability'] = (data['flesch\_kincaid'] + data['gunning\_fog'] + data['smog\_index']) / 3

        *# Combine metrics into a single AI score*

        data['ai\_score'] = (data['normalized\_readability'] + data['repetition\_score'] + data['ai\_probability']) / 3

        *# Scale AI score to a range of 1 to 10, ensuring the lower bound is 1*

        data['ai\_rating'] = data['ai\_score'].apply(**lambda** x: max(1, round(x \* 10)))

        print("AI ratings aggregated.")

    except Exception as e:

        print(**f**"Error aggregating AI ratings: {e}")

    return data

*# Step 6: Save Results to File*

**def** save\_results(data, output\_file):

    """

    Save the processed data with AI ratings to a CSV file.

    """

    try:

        data.to\_csv(output\_file, index=False)

        print(**f**"Results saved to {output\_file}.")

    except Exception as e:

        print(**f**"Error saving results: {e}")

*# Main Function*

**def** main():

    *# File paths*

    input\_file = "messages.csv"  *# Replace with your file path*

    output\_file = "ai\_detected\_messages.csv"

    *# Process data*

    data = load\_and\_clean\_data(input\_file)

    if data is not None:

        data = add\_readability\_metrics(data)

        data = add\_repetition\_score(data)

        data = detect\_ai\_probability(data)

        data = aggregate\_ai\_rating(data)

        save\_results(data, output\_file)

*# Run the script*

if \_\_name\_\_ == "\_\_main\_\_":

    main()