Name Entity Recognition using Python

import tkinter as tk

from tkinter import filedialog, messagebox, ttk

import nltk

from nltk import word\_tokenize, pos\_tag, ne\_chunk

from collections import Counter

nltk.download('punkt')

nltk.download('maxent\_ne\_chunker')

nltk.download('words')

tkinter: This GUI library in python used to creating the application's graphical user interface.

filedialog: Is a one of the Part of tkinter used for loading file as input to the program it work on file dialog functionality.

messagebox: Is a Part of tkinter for displaying popup window on screen as a message boxes.

ttk: Themed Tkinter for enhanced widgets.It helps to create button, label, text box, check box, radio button, list box, frames, etc…

nltk: Natural Language Toolkit it perform natural language processing like ( tokenization, stemming, Name entity recognitions, POS tag, chunking.)

word\_tokenize, pos\_tag, ne\_chunk: NLTK functions for tokenization, part-of-speech tagging, and named entity chunking.

Counter: A class from the collections module for counting occurrences in a collection.

entity\_count = Counter() and word\_count = Counter(): Initialize two Counter objects, entity\_count and word\_count, to count occurrences of named entities and individual words

class NERApp:

def \_\_init\_\_(self, master):

self.master = master

master.title("Named Entity Recognition, Word Count, and POS Tagging")

self.create\_widgets()

class NERApp: This line declares for creating class **NERApp**.

def \_\_init\_\_(self, master): The \_\_init\_\_ method is a special method, also known as the constructor.

It is automatically called when an instance of the class is created.

The self parameter refers to the instance of the class, and master is a parameter passed when creating an instance.

self.master = master: This line stores the reference to the main window (master) as an attribute of the NERApp instance.

master.title("Named Entity Recognition"): Sets the title of the main window to "Named Entity Recognition” This is the title that appears at the top of the application window.

self.create\_widgets(): Calls the create\_widgets method. This method is responsible for setting up the graphical user interface (GUI) components of the application.

def create\_widgets(self):

# Text input area

self.text\_area = ttk.Entry(self.master, width=80)

self.text\_area.grid(row=0, column=0, columnspan=3, padx=10, pady=10)

def create\_widgets(self): This method is responsible for creating and configuring the GUI components of the application.

self.text\_area = ttk.Entry(self.master, width=80): Creates an entry widget (ttk.Entry) named self.text\_area. An entry widget allows the user to input or display a single line of text. width=80 sets the width of the entry to 80 characters.

self.text\_area.grid(row=0, column=0, columnspan=3, padx=10, pady=10): Places the entry widget in the main window using the grid geometry manager. It specifies that the widget should be in the first row (row=0), first column (column=0), span three columns (columnspan=3), and have padding of 10 pixels on the x-axis (padx=10) and y-axis (pady=10).

# Buttons

self.browse\_button = tk.Button(self.master, text="Browse", command=self.load\_file, bg="green")

self.browse\_button.grid(row=1, column=0, padx=10, pady=5)

self.process\_button = tk.Button(self.master, text="Process", command=self.process\_text, bg="blue")

self.process\_button.grid(row=1, column=1, padx=10, pady=5)

Creating Two buttons browse and process button to load the input and process it. Two buttons (self.browse\_button and self.process\_button) are created. The "Browse" button triggers the load\_file method, and the "Process" button triggers the process\_text.

# Status bar

self.status\_bar = tk.Label(self.master, text="", bd=1, relief=tk.SUNKEN, anchor=tk.W)

self.status\_bar.grid(row=2, column=0, columnspan=3, sticky=tk.W + tk.E, padx=10, pady=5)

self.status\_bar = tk.Label(self.master, text="", bd=1, relief=tk.SUNKEN, anchor=tk.W):

Creates a Label widget named self.status\_bar.

self.master: Specifies that the label belongs to the main window.

text="": Initializes the text content of the label as an empty string.

bd=1: Sets the border width of the label to 1 pixel.

self.status\_bar.grid(row=2, column=0, columnspan=3, sticky=tk.W + tk.E, padx=10, pady=5):

# Named Entities Table

self.named\_entities\_tab = ttk.Frame(self.result\_notebook)

self.result\_notebook.add(self.named\_entities\_tab, text='Named Entities')

self.named\_entities\_tree = ttk.Treeview(self.named\_entities\_tab, columns=('Entity', 'Occurrences'), show='headings')

self.named\_entities\_tree.heading('Entity', text='Entity')

self.named\_entities\_tree.heading('Occurrences', text='Occurrences')

self.named\_entities\_tree.pack(fill='both', expand=True)

self.named\_entities\_tab = ttk.Frame(self.result\_notebook):

Creates a new tab (frame) named self.named\_entities\_tab within the result\_notebook. ttk.Frame is used for creating a themed frame.

self.result\_notebook.add(self.named\_entities\_tab, text='Named Entities'):

Adds the self.named\_entities\_tab to the result\_notebook (the notebook widget that holds multiple tabs).

Specifies the display text for this tab as 'Named Entities'.

self.named\_entities\_tree = ttk.Treeview(self.named\_entities\_tab, columns=('Entity', 'Occurrences'), show='headings'):

Creates a Treeview widget named self.named\_entities\_tree within the self.named\_entities\_tab.

columns=('Entity', 'Occurrences'): Specifies the columns in the treeview.

show='headings': Indicates that only the column headings should be shown initially.

self.named\_entities\_tree.heading('Entity', text='Entity') and self.named\_entities\_tree.heading('Occurrences', text='Occurrences'):

Sets the headings for the columns in the self.named\_entities\_tree. The first column is labeled 'Entity', and the second column is labeled 'Occurrences'.

self.named\_entities\_tree.pack(fill='both', expand=True):

def load\_file(self):

try:

file\_path = filedialog.askopenfilename(filetypes=[("Text files", "\*.txt"), ("All files", "\*.\*")])

if file\_path:

with open(file\_path, 'r', encoding='utf-8', errors='replace') as file:

text = file.read()

self.text\_area.delete(0, tk.END)

self.text\_area.insert(0, text)

self.update\_status("File loaded successfully.")

except Exception as e:

messagebox.showerror("Error", f"Error loading file: {str(e)}")

def load\_file(self):

Defines a method named load\_file within the NERApp class.

try: and except Exception as e:

text = file.read():

Reads the content of the opened file and stores it in the variable text.

self.text\_area.delete(0, tk.END):

Clears the existing content in the text\_area Entry widget from position 0 to the end.

self.text\_area.insert(0, text):

Inserts the content of the opened file (text) into the text\_area Entry widget at position 0.

self.update\_status("File loaded successfully."):

Calls the update\_status method to display a success message in the status bar, indicating that the file has been loaded successfully.

except Exception as e: and messagebox.showerror("Error", f"Error loading file: {str(e)}"):

If any exception occurs during the execution of the try block, it is caught in the except block.

Displays an error message using messagebox.showerror, indicating the nature of the error.

def display\_named\_entities(self, entity\_count):

self.named\_entities\_tree.delete(\*self.named\_entities\_tree.get\_children())

for entity, count in entity\_count.items():

self.named\_entities\_tree.insert('', 'end', values=(entity, count))

Defines a method named display\_named\_entities within the NERApp class.

Retrieves all child items (rows) from the named\_entities\_tree.

Uses the delete method to remove each item, effectively clearing the existing entries in the Named Entities table.

Iterates over the items in the entity\_count Counter using entity\_count.items().

'end' specifies that the new row should be inserted at the end of the table.

values=(entity, count) provides the values for the 'Entity' and 'Occurrences' columns in the new row.

root = tk.Tk()

app = NERApp(root)

# Run the application

root.mainloop()

root is a commonly used variable name for the main window.

The NERApp class likely contains the main logic and components of your Named Entity Recognition application.

The mainloop() function is a blocking function that keeps the application running until the user closes the window.

Output:













