Project name: Smart text editor

Group members:

1.Name: Janhavi Dadaso kamble

Roll no:COA253

2.Name: Sayali Vijay Jankar

Roll no:COA250

3.Name:Shravani Rajeshwar Jadhav

Roll no:COA248

CODE:

```
import tkinter as tk
from tkinter import filedialog, messagebox, simpledialog
from textblob import TextBlob
import speech_recognition as sr
import pyttsx3  # NEW IMPORT

class SmartTextEditor:
    def __init__(self, root):
        self.root = root
        self.root.title("Smart Text Editor")
        self.root.geometry("800x600")

        self.dark_mode = False
        self.engine = pyttsx3.init()  # Initialize TTS engine
        self.font_size = 13  # Default font size
```

```
        self.current_font = 'Consolas'

        # Text Area
        self.text_area = tk.Text(root, wrap='word', font=(self.current_font, self.font_size))
        self.text_area.pack(expand=1, fill='both')

        # Status bar
        self.status_bar = tk.Label(root, text="Words: 0 | Characters: 0", anchor='e')
        self.status_bar.pack(fill='x', side='bottom')

        # Bind text change to update counter
        self.text_area.bind('<KeyRelease>', self.update_status)

        # Menu
        self.create_menu()

    def create_menu(self):
        menu = tk.Menu(self.root)
        self.root.config(menu=menu)

        # File Menu
        file_menu = tk.Menu(menu, tearoff=0)
        file_menu.add_command(label="Open", command=self.open_file)
        file_menu.add_command(label="Save", command=self.save_file)
        file_menu.add_command(label="Save As", command=self.save_as_file)  # NEW "Save
As"
        file_menu.add_separator()
        file_menu.add_command(label="Exit", command=self.root.quit)
```

```
        menu.add_cascade(label="File", menu=file_menu)


        # Edit Menu
        edit_menu = tk.Menu(menu, tearoff=0)
        edit_menu.add_command(label="Auto-Correct", command=self.auto_correct_text)
        edit_menu.add_command(label="Clear", command=lambda: self.text_area.delete(1.0,
tk.END))
        edit_menu.add_command(label="Voice Typing", command=self.voice_typing)
        edit_menu.add_command(label="Read Aloud", command=self.read_aloud)  # NEW
        menu.add_cascade(label="Edit", menu=edit_menu)


        # View Menu (with Zoom In, Zoom Out, and Change Font)
        view_menu = tk.Menu(menu, tearoff=0)
        view_menu.add_command(label="Zoom In", command=self.zoom_in)  # NEW: Zoom In
        view_menu.add_command(label="Zoom Out", command=self.zoom_out)  # NEW: Zoom
Out
        view_menu.add_command(label="Change Font", command=self.change_font)  # NEW:
Change Font
        view_menu.add_command(label="Toggle Dark Mode",
command=self.toggle_dark_mode)
        menu.add_cascade(label="View", menu=view_menu)


        # NEW Help Menu
        help_menu = tk.Menu(menu, tearoff=0)
        help_menu.add_command(label="About", command=self.show_about)
        help_menu.add_command(label="Documentation",
command=self.show_documentation)
        menu.add_cascade(label="Help", menu=help_menu)
```

```
        # NEW Tools Menu
        tools_menu = tk.Menu(menu, tearoff=0)
        tools_menu.add_command(label="Word Count", command=self.show_word_count)
        tools_menu.add_command(label="Character Count", command=self.show_char_count)
        tools_menu.add_command(label="Find & Replace", command=self.find_and_replace)  # NEW Find & Replace
        menu.add_cascade(label="Tools", menu=tools_menu)

    def open_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
        if file_path:
            with open(file_path, 'r', encoding='utf-8') as file:
                content = file.read()
                self.text_area.delete(1.0, tk.END)
                self.text_area.insert(tk.END, content)
            self.update_status()

    def save_file(self):
        file_path = filedialog.asksaveasfilename(defaultextension=".txt",
                                                 filetypes=[("Text files", "*.txt")])
        if file_path:
            with open(file_path, 'w', encoding='utf-8') as file:
                content = self.text_area.get(1.0, tk.END)
                file.write(content)
            messagebox.showinfo("Saved", "File saved successfully!")

    # NEW: Save As method
    def save_as_file(self):
```

```
    file_path = filedialog.asksaveasfilename(defaultextension=".txt",
                                             filetypes=[("Text files", "*.txt")])
    if file_path:
        with open(file_path, 'w', encoding='utf-8') as file:
            content = self.text_area.get(1.0, tk.END)
            file.write(content)
        messagebox.showinfo("Saved", "File saved successfully as new file!")

def auto_correct_text(self):
    original_text = self.text_area.get(1.0, tk.END)
    corrected_text = str(TextBlob(original_text).correct())
    self.text_area.delete(1.0, tk.END)
    self.text_area.insert(tk.END, corrected_text)
    self.update_status()

def voice_typing(self):
    recognizer = sr.Recognizer()
    mic = sr.Microphone()
    try:
        messagebox.showinfo("Voice Typing", "Speak now...")
        with mic as source:
            recognizer.adjust_for_ambient_noise(source)
            audio = recognizer.listen(source, timeout=5)

        text = recognizer.recognize_google(audio)
        self.text_area.insert(tk.INSERT, text + " ")
        self.update_status()
```

```
        except sr.UnknownValueError:
            messagebox.showerror("Error", "Could not understand audio")
        except sr.RequestError:
            messagebox.showerror("Error", "Speech recognition service is unavailable")
        except sr.WaitTimeoutError:
            messagebox.showwarning("Timeout", "No speech detected. Try again.")

    # NEW: Text-to-Speech (Read Aloud)
    def read_aloud(self):
        text = self.text_area.get("1.0", tk.END).strip()
        if text:
            self.engine.say(text)
            self.engine.runAndWait()
        else:
            messagebox.showinfo("No Text", "There's no text to read.")

    # NEW: Show About Information
    def show_about(self):
        messagebox.showinfo("About", "Smart Text Editor\nVersion 1.0\nDeveloped by Your
Name")

    # NEW: Show Documentation
    def show_documentation(self):
        messagebox.showinfo("Documentation", "For more information, visit:
\nwww.smarttexteditor.com")

    # NEW: Show Word Count
    def show_word_count(self):
        text = self.text_area.get("1.0", tk.END).strip()
```

```
        word_count = len(text.split())
        messagebox.showinfo("Word Count", f"Total words: {word_count}")

    # NEW: Show Character Count
    def show_char_count(self):
        text = self.text_area.get("1.0", tk.END).strip()
        char_count = len(text.replace(" ", ""))  # Remove spaces to get character count
        messagebox.showinfo("Character Count", f"Total characters (without spaces): {char_count}")

    # NEW: Find & Replace Feature
    def find_and_replace(self):
        find_window = tk.Toplevel(self.root)
        find_window.title("Find & Replace")

        # Add Find Label, Entry, Replace Label, and Entry
        tk.Label(find_window, text="Find:").pack(pady=5)
        find_entry = tk.Entry(find_window, width=40)
        find_entry.pack(pady=5)

        tk.Label(find_window, text="Replace with:").pack(pady=5)
        replace_entry = tk.Entry(find_window, width=40)
        replace_entry.pack(pady=5)

        def replace_text():
            find_text = find_entry.get()
            replace_text = replace_entry.get()
            content = self.text_area.get(1.0, tk.END)
```

```
            updated_content = content.replace(find_text, replace_text)
            self.text_area.delete(1.0, tk.END)
            self.text_area.insert(tk.END, updated_content)
            find_window.destroy()

        # Add Replace Button
        replace_button = tk.Button(find_window, text="Replace", command=replace_text)
        replace_button.pack(pady=10)

    # NEW: Zoom In (Increase font size)
    def zoom_in(self):
        self.font_size += 1
        self.text_area.config(font=(self.current_font, self.font_size))

    # NEW: Zoom Out (Decrease font size)
    def zoom_out(self):
        self.font_size -= 1
        self.text_area.config(font=(self.current_font, self.font_size))

    # NEW: Change Font
    def change_font(self):
        font_options = ['Consolas', 'Arial', 'Courier', 'Helvetica']
        font = simpledialog.askstring("Change Font", f"Select a font from: {',
'.join(font_options)}")
        if font in font_options:
            self.current_font = font
            self.text_area.config(font=(self.current_font, self.font_size))
```

```
    def update_status(self, event=None):
        text = self.text_area.get(1.0, tk.END)
        words = len(text.split())
        chars = len(text) - 1  # subtract one for the last newline
        self.status_bar.config(text=f"Words: {words} | Characters: {chars}")

    def toggle_dark_mode(self):
        if not self.dark_mode:
            self.text_area.config(bg="#2e2e2e", fg="white", insertbackground='white')
            self.status_bar.config(bg="#1e1e1e", fg="white")
        else:
            self.text_area.config(bg="white", fg="black", insertbackground='black')
            self.status_bar.config(bg="SystemButtonFace", fg="black")
        self.dark_mode = not self.dark_mode

if __name__ == "__main__":
    root = tk.Tk()
    app = SmartTextEditor(root)
    root.mainloop()
```

Output: