code

```
import pandas as pd
import tkinter as tk
from tkinter import messagebox, filedialog
import threading
import time
import random  # Simulating speed sensor input

class BrakingSystem:
    def __init__(self):
        self.speed_limit = 3000  # Default speed limit (RPM)
        self.current_speed = 0
        self.is_running = True

    def upload_dataset(self, file_path):
        """
        Uploads a dataset to configure the speed limit.
        :param file_path: Path to the dataset file.
        """
        try:
            df =
pd.read_csv(r'C:\Users\Dhashna\Downloads\corrected_test_dataset.csv')
            if 'SpeedLimit' in df.columns:
                self.speed_limit = int(df['SpeedLimit'].iloc[0])
                print(f"Speed limit updated to {self.speed_limit} RPM.")
                messagebox.showinfo("Dataset Upload", f"Speed limit set to
```

```
{self.speed_limit} RPM.")
            else:
                raise ValueError("Dataset must contain a 'SpeedLimit' column.")
        except Exception as e:
            messagebox.showerror("Error", f"Failed to upload dataset: {e}")

    def calculate_speed(self):
        """
        Simulates speed calculation. Replace with real sensor data.
        """
        self.current_speed = random.randint(0, 5000)  # Simulated speed
        return self.current_speed

    def stop_blade(self):
        """
        Stops the blade if the speed exceeds the limit.
        """
        self.is_running = False
        print("Emergency stop! Blade stopped.")
        messagebox.showwarning("Emergency Stop", "Blade stopped due to speed
limit violation!")

    def monitor_speed(self, speed_label):
        """
        Continuously monitors the speed and activates the brake if necessary.
        """
```

```
        self.is_running = True
        while self.is_running:
            speed = self.calculate_speed()
            speed_label.config(text=f"Current Speed: {speed} RPM")
            print(f"Current Speed: {speed} RPM")
            if speed > self.speed_limit:
                self.stop_blade()
                break
            time.sleep(1)  # Simulate real-time monitoring

# Tkinter GUI
class BrakingSystemGUI:
    def __init__(self, braking_system):
        self.braking_system = braking_system
        self.root = tk.Tk()
        self.root.title("Ultrafast Electronic Braking System")
        self.root.config(bg="lightblue")  # Set background color
        self.root.geometry("500x350")  # Adjust window size
        self.root.iconbitmap('')  # Remove window icon by providing an empty string
        self.setup_gui()

    def setup_gui(self):
        # Speed Limit Display with larger font size
        self.speed_limit_label = tk.Label(self.root, text=f"Speed Limit:
{self.braking_system.speed_limit} RPM", font=("Arial", 18), bg="lightblue")
        self.speed_limit_label.pack(pady=10)
```

```
        # Current Speed Display with larger font size
        self.current_speed_label = tk.Label(self.root, text="Current Speed: 0 RPM",
font=("Arial", 18), bg="lightblue")
        self.current_speed_label.pack(pady=10)

        # Upload Dataset Button
        upload_button = tk.Button(self.root, text="Upload Dataset",
command=self.upload_dataset, font=("Arial", 14), width=25, height=3, bg="red",
fg="white")
        upload_button.pack(pady=20)

        # Start Monitoring Button
        start_button = tk.Button(self.root, text="Start Monitoring",
command=self.start_monitoring, font=("Arial", 14), width=25, height=3, bg="red",
fg="white")
        start_button.pack(pady=20)

        # Emergency Stop Button
        stop_button = tk.Button(self.root, text="Emergency Stop",
command=self.emergency_stop, font=("Arial", 14), width=25, height=3, bg="red",
fg="white")
        stop_button.pack(pady=20)

    def upload_dataset(self):
        file_path = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")])
```

```
        if file_path:
            self.braking_system.upload_dataset(file_path)
            self.speed_limit_label.config(text=f"Speed Limit:
{self.braking_system.speed_limit} RPM")

    def start_monitoring(self):
        # Ensure the previous monitoring session is stopped
        self.braking_system.is_running = False
        monitoring_thread =
threading.Thread(target=self.braking_system.monitor_speed,
args=(self.current_speed_label,))
        monitoring_thread.start()

    def emergency_stop(self):
        self.braking_system.is_running = False
        self.braking_system.stop_blade()

    def run(self):
        self.root.mainloop()

# Main Program
    if __name__ == "__main__":
    # Initialize the braking system
    braking_system = BrakingSystem()
```

```
# Start the GUI
gui = BrakingSystemGUI(braking_system)
gui.run()
```