```
import tkinter as tk
from tkinter import ttk, filedialog
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from math import pi

df = pd.DataFrame()

import matplotlib
matplotlib.rcParams['figure.max_open_warning'] = 50

def load_csv():
    global df
    file_path = filedialog.askopenfilename(title="Select CSV file", filetypes=[("CSV files", "*.csv")])
    if file_path:
        try:
            df = pd.read_csv(file_path, nrows=60)
            print("CSV file loaded successfully.")
            print("DataFrame head:")
            print(df.head())
            update_plot()
        except Exception as e:
            print(f"Error loading CSV file: {e}")
    else:
        print("No file selected.")

def perform_linear_regression_all_students():
    print("Performing linear regression for all students")

    try:
        plt.cla()

        ax = plt.subplot(111)

        for student_name in df['StudentName'].unique():
            student_data = df[df['StudentName'] == student_name]
            if not student_data.empty:

                sns.regplot(x='AverageTotalMarks', y='FinalMark', data=student_data, label=student_name, ax=ax)

        plt.xlabel('AverageTotalMarks')
        plt.ylabel('FinalMark')
        plt.legend()
        update_plot()
    except Exception as e:
        print(f"Error performing linear regression: {e}")


def perform_bar_graph_all_students():
    print("Performing bar graph for all students")

    try:
        plt.cla()

        unique_students = df['StudentName'].unique()
        width = 0.35
        n_students = len(unique_students)

        for i, student_name in enumerate(unique_students):
            student_data = df[df['StudentName'] == student_name]
            if not student_data.empty:
                x_values = [pos + i * width for pos in range(len(student_data))]
                plt.bar(x_values, student_data['AverageTotalMarks'], width=width, label=student_name)

        plt.xlabel('Sample Index')
        plt.ylabel('AverageTotalMarks')
        plt.legend()
        update_plot()
    except Exception as e:
        print(f"Error performing bar graph: {e}")

def perform_pie_chart_all_students():
    print("Performing pie chart for all students")

    try:
        plt.cla()

        unique_students = df['StudentName'].unique()
        n_students = len(unique_students)
```

```
            student_counts = df['StudentName'].value_counts()
            plt.pie(student_counts, labels=student_counts.index, autopct='%1.1f%%', startangle=90)
            plt.title('Distribution of Students')
            update_plot()
        except Exception as e:
            print(f"Error performing pie chart: {e}")

    def perform_histogram_all_students():
        print("Performing histogram for all students")

        try:
            plt.cla()

            selected_columns = ['TotalMark1', 'TotalMark2', 'FinalMark']
            df[selected_columns].hist(bins=10, alpha=0.5, figsize=(10, 6))
            plt.suptitle('Histograms of Selected Variables')
            plt.show()
        except Exception as e:
            print(f"Error performing histogram: {e}")

    def perform_violin_plot_all_students():
        print("Performing violin plot for all students")

        try:
            plt.cla()

            sns.violinplot(data=df[['TotalMark1', 'TotalMark2', 'FinalMark']])
            plt.title('Violin Plot of Selected Variables')
            update_plot()
        except Exception as e:
            print(f"Error performing violin plot: {e}")

    def perform_line_graph_all_students():
        print("Performing line graph for all students")

        try:
            plt.cla()

            for student_name in df['StudentName'].unique():
                student_data = df[df['StudentName'] == student_name]
                if not student_data.empty:
                    plt.plot(student_data['FinalMark'], label=student_name)

            plt.xlabel('X-axis Label')
            plt.ylabel('Y-axis Label')
            plt.legend()
            update_plot()
        except Exception as e:
            print(f"Error performing line graph: {e}")

    def perform_dual_axis_chart_all_students():
        print("Performing dual-axis chart for all students")

        try:
            plt.cla()

            ax1 = plt.subplot(111)
            ax2 = ax1.twinx()

            ax1.plot(df['TotalMark1'], 'b-', label='TotalMark1')

            ax1.set_ylabel('TotalMark1', color='b')
            ax1.tick_params('y', colors='b')

            ax2.plot(df['TotalMark2'], 'r-', label='TotalMark2')
            ax2.set_ylabel('TotalMark2', color='r')
            ax2.tick_params('y', colors='r')

            plt.legend()
            update_plot()
        except Exception as e:
            print(f"Error performing dual-axis chart: {e}")

    def perform_linear_regression_all_students():
        print("Performing linear regression for all students")

        try:

            plt.cla()

            ax = plt.subplot(111)
            for student_name in df['StudentName'].unique():
                student_data = df[df['StudentName'] == student_name]
                if not student_data.empty:

                    sns.regplot(x='AverageTotalMarks', y='FinalMark', data=student_data, label=student_name, ax=ax)

            plt.xlabel('AverageTotalMarks')
            plt.ylabel('FinalMark')
            plt.legend()
            update_plot()
        except Exception as e:
            print(f"Error performing linear regression: {e}")


    def perform_heatmap_analysis():
        print("Performing heatmap analysis")

        try:
            plt.cla()

            heatmap_variables = ['TotalMark1', 'TotalMark2', 'FinalMark']

            correlation_matrix = df[heatmap_variables].corr()

            sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=.5)
            plt.title('Heatmap Analysis')
            update_plot()
        except Exception as e:
            print(f"Error performing heatmap analysis: {e}")


    def update_plot():
        canvas.draw()


    root = tk.Tk()
    root.title("Data Analysis Visualization")

    fig, ax = plt.subplots(figsize=(10, 5))
    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas_widget = canvas.get_tk_widget()
    canvas_widget.pack()

    load_csv_button = ttk.Button(root, text="Load CSV", command=load_csv)
    load_csv_button.pack(pady=10)

    linear_regression_button = ttk.Button(root, text="Linear Regression for All Students", command=perform_linear_regression_all_students)
    linear_regression_button.pack(pady=5)

    bar_graph_button = ttk.Button(root, text="Bar Graph for All Students", command=perform_bar_graph_all_students)
    bar_graph_button.pack(pady=5)

    pie_chart_button = ttk.Button(root, text="Pie Chart for All Students", command=perform_pie_chart_all_students)
    pie_chart_button.pack(pady=5)

    histogram_button = ttk.Button(root, text="Histogram for All Students", command=perform_histogram_all_students)
    histogram_button.pack(pady=5)

    violin_plot_button = ttk.Button(root, text="Violin Plot for All Students", command=perform_violin_plot_all_students)
    violin_plot_button.pack(pady=5)

    line_graph_button = ttk.Button(root, text="Line Graph for All Students", command=perform_line_graph_all_students)
    line_graph_button.pack(pady=5)

    dual_axis_chart_button = ttk.Button(root, text="Dual-Axis Chart for All Students", command=perform_dual_axis_chart_all_students)
    dual_axis_chart_button.pack(pady=5)

    heatmap_analysis_button = ttk.Button(root, text="Heatmap Analysis", command=perform_heatmap_analysis)
    heatmap_analysis_button.pack(pady=5)

    root.mainloop()
```