```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
from datetime import datetime

class StudentPerformanceAnalyzer:
    def __init__(self):
        self.students_data = []
        self.df = None

    def add_student(self, student_id, name, subjects_grades):
        student_record = {
            'student_id': student_id,
            'name': name,
            'grades': subjects_grades,
            'gpa': self.calculate_gpa(subjects_grades)
        }
        self.students_data.append(student_record)

    def calculate_gpa(self, grades):
        grade_points = {'A+': 10, 'A': 9, 'B+': 8, 'B': 7, 'C+': 6, 'C': 5, 'D': 4, 'F': 0}
        total_points = sum(grade_points.get(grade, 0) for grade in grades.values())
        return total_points / len(grades) if grades else 0

    def load_from_csv(self, filename):
        try:
            self.df = pd.read_csv(filename)
            return True
        except Exception as e:
            print(f"Error loading CSV: {e}")
            return False

    def generate_performance_report(self):
        if not self.students_data:
            return "No data available"

        df_data = []
        for student in self.students_data:
            row = {'student_id': student['student_id'],
                   'name': student['name'],
                   'gpa': student['gpa']}
            for subject, grade in student['grades'].items():
                row[subject] = grade
            df_data.append(row)

        self.df = pd.DataFrame(df_data)

        stats = {
            'total_students': len(self.students_data),
            'average_gpa': np.mean([s['gpa'] for s in self.students_data]),
            'highest_gpa': max([s['gpa'] for s in self.students_data]),
            'lowest_gpa': min([s['gpa'] for s in self.students_data]),
            'gpa_std': np.std([s['gpa'] for s in self.students_data])
        }
        return stats

    def create_visualizations(self):
        if self.df is None or self.df.empty:
            return False

        plt.figure(figsize=(12, 8))
        plt.subplot(2, 2, 1)
        gpas = [s['gpa'] for s in self.students_data]
        plt.hist(gpas, bins=10, edgecolor='black', alpha=0.7)
        plt.title('GPA Distribution')
        plt.xlabel('GPA')
        plt.ylabel('Number of Students')

        plt.subplot(2, 2, 2)
        if 'Math' in self.df.columns:
            grade_counts = self.df['Math'].value_counts()
            plt.pie(grade_counts.values, labels=grade_counts.index, autopct='%1.1f%%')
            plt.title('Math Performance Distribution')

        plt.tight_layout()
        plt.savefig('performance_analysis.png', dpi=300, bbox_inches='tight')
        plt.show()

        return True

class PerformanceGUI:
    def __init__(self, root):
        self.root = root
        self.analyzer = StudentPerformanceAnalyzer()
        self.setup_gui()

    def setup_gui(self):
        self.root.title("Student Performance Analysis System")
        self.root.geometry("800x600")

        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        title_label = ttk.Label(main_frame, text="Student Performance Analysis System", font=('Arial', 16, 'bold'))
        title_label.grid(row=0, column=0, columnspan=2, pady=10)

        ttk.Label(main_frame, text="Student ID:").grid(row=1, column=0, sticky=tk.W, pady=5)
        self.student_id_entry = ttk.Entry(main_frame, width=30)
        self.student_id_entry.grid(row=1, column=1, pady=5, padx=10)

        ttk.Label(main_frame, text="Student Name:").grid(row=2, column=0, sticky=tk.W, pady=5)
        self.student_name_entry = ttk.Entry(main_frame, width=30)
        self.student_name_entry.grid(row=2, column=1, pady=5, padx=10)

        subjects = ['Math', 'Science', 'English', 'History', 'Computer']
        self.grade_entries = {}
        for i, subject in enumerate(subjects):
            ttk.Label(main_frame, text=f"{subject} Grade:").grid(row=3+i, column=0, sticky=tk.W)
            grade_combo = ttk.Combobox(main_frame, values=['A+', 'A', 'B+', 'B', 'C+', 'C', 'D', 'F'])
            grade_combo.grid(row=3+i, column=1, pady=2, padx=10)
            self.grade_entries[subject] = grade_combo

        button_frame = ttk.Frame(main_frame)
        button_frame.grid(row=8, column=0, columnspan=2, pady=20)

        ttk.Button(button_frame, text="Add Student", command=self.add_student).pack(side=tk.LEFT, padx=5)
        ttk.Button(button_frame, text="Load CSV", command=self.load_csv).pack(side=tk.LEFT, padx=5)
        ttk.Button(button_frame, text="Generate Report", command=self.generate_report).pack(side=tk.LEFT, padx=5)
        ttk.Button(button_frame, text="Create Charts", command=self.create_charts).pack(side=tk.LEFT, padx=5)
```

```
        self.results_text = tk.Text(main_frame, height=15, width=80)
        self.results_text.grid(row=9, column=0, columnspan=2, pady=10)

        scrollbar = ttk.Scrollbar(main_frame, orient="vertical", command=self.results_text.yview)
        scrollbar.grid(row=9, column=2, sticky="ns")
        self.results_text.configure(yscrollcommand=scrollbar.set)

    def add_student(self):
        student_id = self.student_id_entry.get()
        student_name = self.student_name_entry.get()

        if not student_id or not student_name:
            messagebox.showerror("Error", "Please enter student ID and name")
            return

        grades = {}
        for subject, entry in self.grade_entries.items():
            grade = entry.get()
            if grade:
                grades[subject] = grade

        if not grades:
            messagebox.showerror("Error", "Please enter at least one grade")
            return

        self.analyzer.add_student(student_id, student_name, grades)
        messagebox.showinfo("Success", f"Student {student_name} added successfully!")

        self.student_id_entry.delete(0, tk.END)
        self.student_name_entry.delete(0, tk.END)
        for entry in self.grade_entries.values():
            entry.set('')

    def load_csv(self):
        filename = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
        if filename:
            if self.analyzer.load_from_csv(filename):
                messagebox.showinfo("Success", "CSV file loaded successfully!")
            else:
                messagebox.showerror("Error", "Failed to load CSV file")

    def generate_report(self):
        stats = self.analyzer.generate_performance_report()
        if isinstance(stats, str):
            self.results_text.delete(1.0, tk.END)
            self.results_text.insert(tk.END, stats)
            return

        report = f"""
STUDENT PERFORMANCE ANALYSIS REPORT
==================================
Generated on: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}
SUMMARY STATISTICS:
- Total Students: {stats['total_students']}
- Average GPA: {stats['average_gpa']:.2f}
- Highest GPA: {stats['highest_gpa']:.2f}
- Lowest GPA: {stats['lowest_gpa']:.2f}
- GPA Standard Deviation: {stats['gpa_std']:.2f}

DETAILED STUDENT DATA:
"""
        for student in self.analyzer.students_data:
            report += f"\nStudent ID: {student['student_id']}\n"
            report += f"Name: {student['name']}\n"
            report += f"GPA: {student['gpa']:.2f}\n"
            report += f"Grades: {student['grades']}\n"
            report += "-" * 40 + "\n"

        self.results_text.delete(1.0, tk.END)
        self.results_text.insert(tk.END, report)

    def create_charts(self):
        if self.analyzer.create_visualizations():
            messagebox.showinfo("Success", "Charts created and saved as 'performance_analysis.png'")
        else:
            messagebox.showerror("Error", "No data available for visualization")

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