Best Performing Student Recognition System Using Machine Learning

In [39]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
import numpy as np
import tkinter as tk
from tkinter import filedialog, messagebox
from tkinter import ttk
```

In [40]:
```
# Function to load data from a CSV file
def load_data(file_path):
    df = pd.read_csv(file_path)
    return df
```

In [41]:
```
# Function to calculate scores based on given features
def calculate_scores(df):
    # Define features for the model
    features = ['gpa_sem1', 'gpa_sem2', 'core_courses_avg', 'hackathons
    # Standardize the features
    scaler = StandardScaler()
    features_scaled = scaler.fit_transform(df[features])

    # Create a Linear Regression model
    model = LinearRegression()

    # Simulated contribution scores for training the model
    np.random.seed(0)  # For reproducibility
    contribution_scores = np.random.rand(len(df)) * 100  # Random score

    # Train the model
    model.fit(features_scaled, contribution_scores)

    # Predict scores for each student
    df['predicted_score'] = model.predict(features_scaled)

    return df
```

In [42]:
```
# Function to get the top 3 students for each year
def get_top_students(df):
    top_students_by_year = df.groupby('year').apply(lambda x: x.nlarges
    return top_students_by_year[['year', 'student_id', 'predicted_score
```

In [43]:
```
# Function to handle the file upload and process
def process_csv():
    file_path = filedialog.askopenfilename(title="Select CSV File", fil
    if file_path:
        try:
            df = load_data(file_path)
            df_with_scores = calculate_scores(df)
            top_students = get_top_students(df_with_scores)
            display_top_students(top_students)
        except Exception as e:
            messagebox.showerror("Error", f"Failed to process file: {e}
```

In [44]:
```
# Function to display the top students with a gap between each year
def display_top_students(top_students):
    for row in tree.get_children():
        tree.delete(row)  # Clear previous results

    for year, group in top_students.groupby('year'):
        # Insert a row with the year
        tree.insert("", tk.END, values=(f"Year: {year}", "", ""))

        for i, (student_id, score) in enumerate(zip(group['student_id']
            tree.insert("", tk.END, values=("", student_id, f"{score:.2
```

```
        # Insert an empty row after the last student in each year
        tree.insert("", tk.END, values=("", "", ""))
```

In [45]:
```
# Create the main application window
app = tk.Tk()
app.title("Best Performing Student Recognition System")
app.geometry("700x550")
app.configure(bg="#f8f8f8")  # Light background color
```

In [46]:
```
# Create a frame for the content with light color and padding
frame = tk.Frame(app, bg="#E6E6FA", padx=30, pady=30, relief=tk.GROOVE,
frame.pack(padx=20, pady=20, fill=tk.BOTH, expand=True)
```

In [47]:
```
# Title label with modern font and padding
title_label = tk.Label(frame, text="Best Performing Students", font=("A
title_label.pack(pady=20)
```

In [48]:
```
# Create and place the upload button with a clear style
upload_button = ttk.Button(frame, text="Upload CSV File", command=proce
upload_button.pack(pady=10)
```

In [49]:
```
# Create Treeview widget to display the top students
columns = ('year', 'student_id', 'predicted_score')
tree = ttk.Treeview(frame, columns=columns, show="headings")
tree.heading('year', text='Year')
tree.heading('student_id', text='Student ID')
tree.heading('predicted_score', text='Predicted Score')
```

Out[49]: {}

In [50]:
```
# Add horizontal and vertical scrollbars to the Treeview widget
tree_scrollbar_y = ttk.Scrollbar(frame, orient="vertical", command=tree
tree_scrollbar_x = ttk.Scrollbar(frame, orient="horizontal", command=tr
tree.configure(yscrollcommand=tree_scrollbar_y.set, xscrollcommand=tree
tree_scrollbar_y.pack(side=tk.RIGHT, fill=tk.Y)
tree_scrollbar_x.pack(side=tk.BOTTOM, fill=tk.X)
```

In [51]:
```
# Pack the Treeview widget
tree.pack(pady=10, fill=tk.BOTH, expand=True)
```

In [52]:
```
# Apply a light and simple button style
style = ttk.Style()
style.configure("TButton", font=("Arial", 12), padding=10)
```

In [53]:
```
# Run the application
app.mainloop()
```

In [ ]:
```

```