# Programming Environment

## Programming Language:

C++ (Standard: C++17):

The core application, including data loading, preprocessing, the Random Forest machine learning model (decision trees, ensemble logic), prediction, recommendation engine, and the command-line interface (CLI), is implemented in C++. C++17 was chosen for its modern features, performance capabilities, and control over memory management, suitable for building a computational application from fundamental components.

## External Software Packages/Libraries and Their Roles:

**Standard C++ Library:** Extensively used for core functionalities:

**iostream:** For console input and output (CLI interactions).

**vector**: For dynamic arrays to store datasets, features, labels, and collections of objects string,

**sstream:** For string manipulation and parsing CSV data.

**fstream**: For file input/output (loading the dataset, reading/writing history).

**algorithm:** For functions like std::sort, std::shuffle, std::min, std::max, std::iota, std::transform.

**random:** For random number generation

**map:** For counting class occurrences

**cmath:** For mathematical functions like std::sqrt.

**limits:** For std::numeric\_limits

**iomanip:** For output formatting

**chrono, ctime:** For generating timestamps for the prediction history.

(No External Machine Learning Libraries for Core RF): Notably, for the Random Forest algorithm itself (decision tree construction, splitting logic, ensemble management), no external pre-built machine learning libraries were used in this C++ phase. The decision tree and random forest logic were implemented from scratch as per the project's C++ focus.

# User's Guide

This guide demonstrates how a user interacts with the Command Line Interface (CLI) of the Behavioral Optimization & Mental Wellness System to achieve their requirements.

## Pre-requisite:

The C++ application (depression\_classification) has been compiled and is ready to run in a terminal/console environment. **The cleaned\_student\_data.csv file must be in the same directory as the executable, or its path correctly specified in the C++ code.**

## Execution Steps:

## Step 1: Compile the main.cpp file to get the C++ application

Navigate to you project dir (can be done by running the following command)

git clone <https://github.com/DiwBhat/Depression-Classification.git>

Then cd into the cpp\_port derictory

cd cpp\_port

Run the following command to get the application

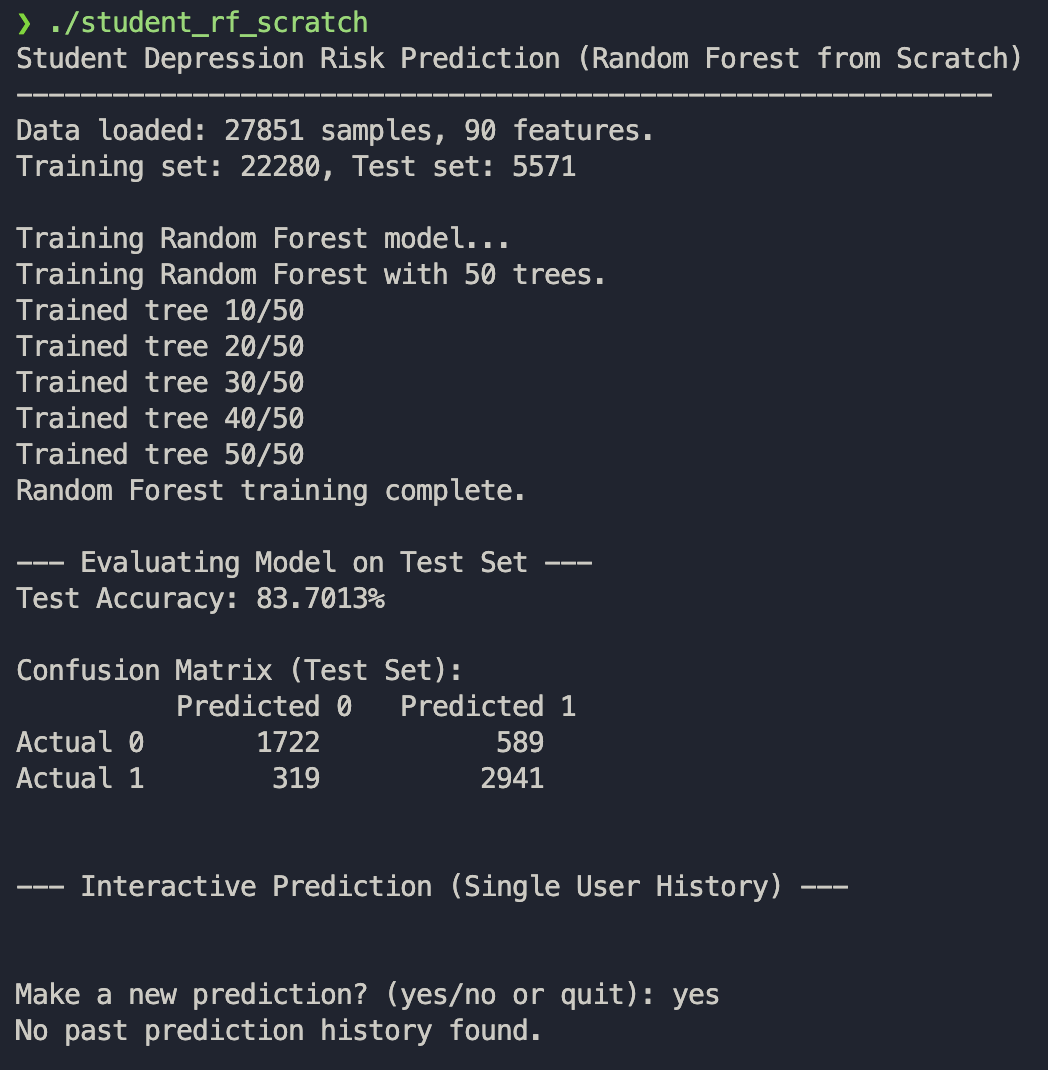
g++ -std=c++17 -Wall -Wextra -O2 -o depression\_classification main.cpp

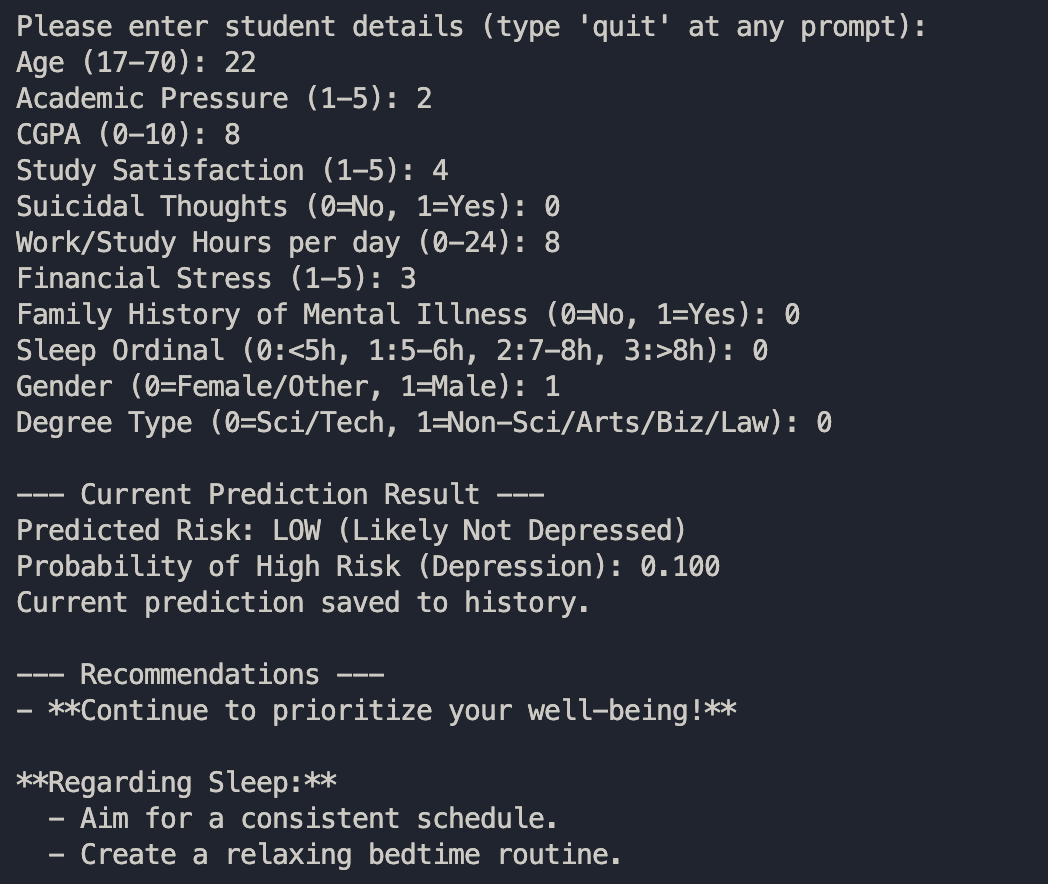
Step 2: Run the C++ application

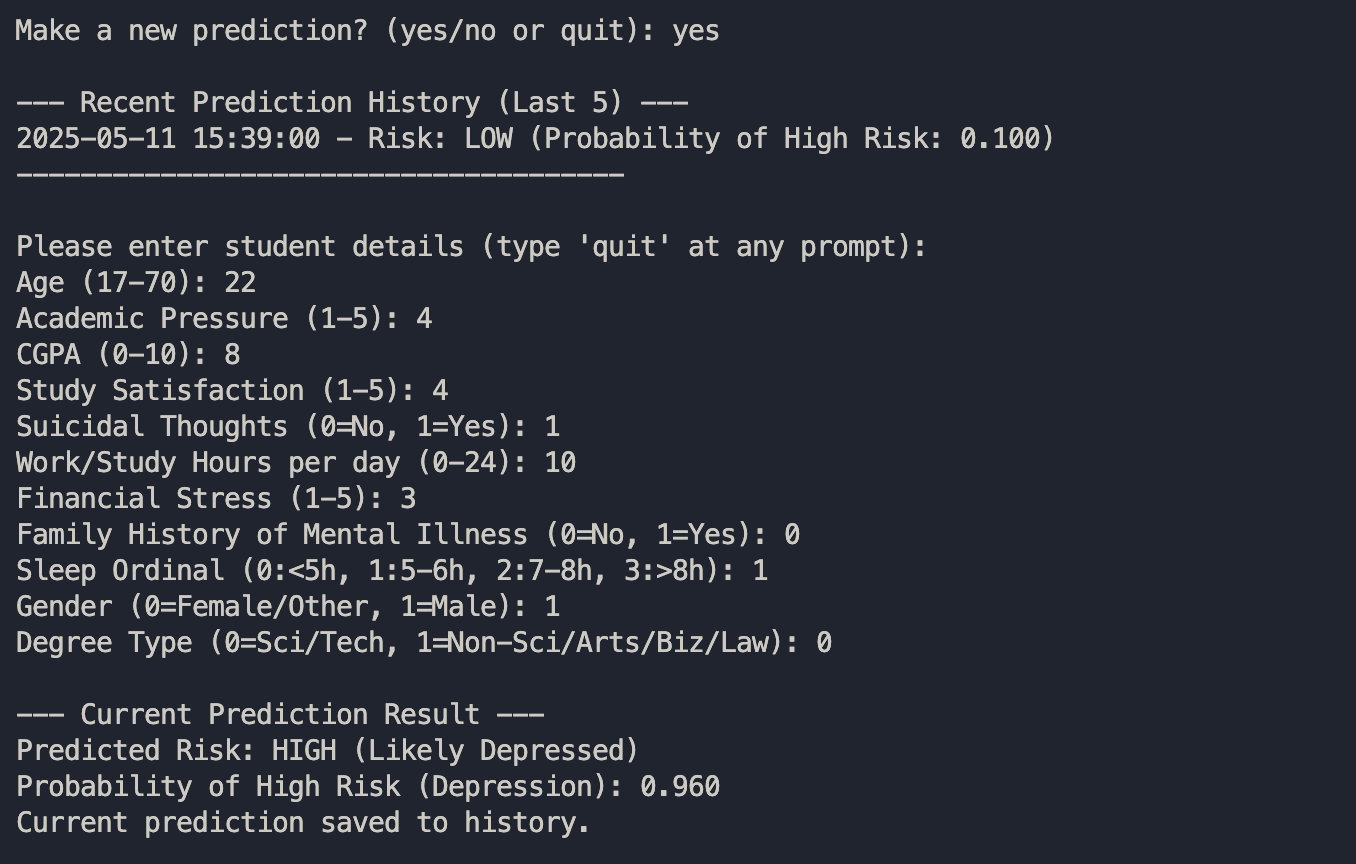
Run the following command to run the application

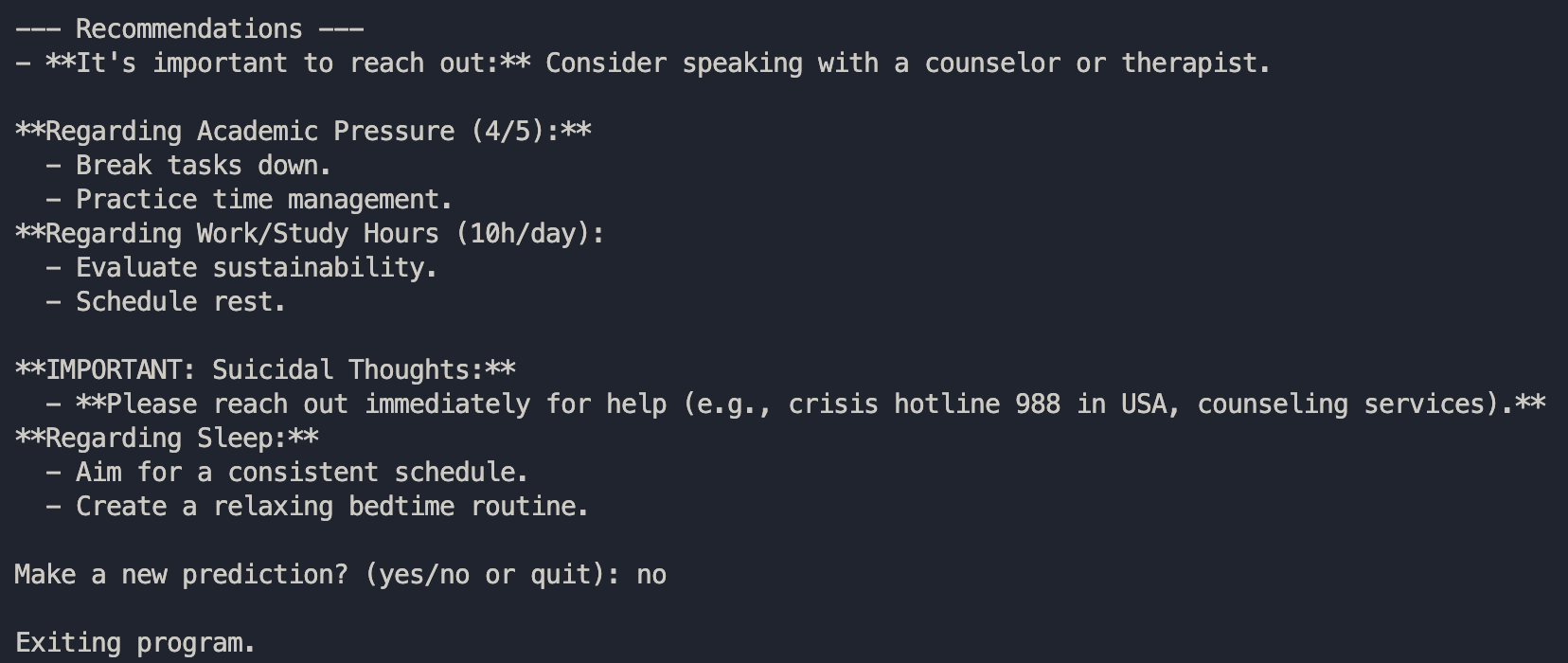
./depression\_classification

## Screenshot of a sample workflow









# Summary of Work Planned for Phase I

## Complete Work (Phase I):

* Data Loading & Basic Preprocessing: C++ module to load the data.
* Decision Tree Implementation: Core logic for a decision tree classifier built from scratch, including Gini impurity calculation, best split finding, and recursive tree construction with parameters for max depth and min samples per leaf.
* Random Forest Implementation: Ensemble logic built from scratch, managing multiple decision trees, and feature subsampling.
* Model Training: Functionality to train the Random Forest model.
* Prediction Logic: Class prediction based on majority voting from trees in the forest.
* Probability estimation for the positive class based on the proportion of tree votes.
* Basic Evaluation: Calculation and display of test set accuracy and a confusion matrix.
* Simplified CLI: A command-line interface for user interaction.
* Prompts for simplified, more intuitive user inputs.
* Displays prediction results (risk category and probability).
* Rule-Based Recommendation Engine: Generates basic recommendations based on the predicted risk level and specific user input values
* Single-User History Tracking: Saves prediction timestamps, probabilities, and risk levels to a local text file (risk\_history.txt) and displays recent history to the user.

## Incomplete Work for Future Improvement (Beyond Phase I Scope):

* Advanced Hyperparameter Tuning in C++: While the Random Forest has tunable parameters, a systematic hyperparameter optimization framework (like GridSearchCV or Bayesian Optimization) was not implemented in C++. Parameters are currently set manually.
* Robust Input Validation in CLI: The CLI input validation is basic. More comprehensive checks for data types, ranges, and formats could be added.
* Advanced Feature Importance Calculation: A from-scratch implementation for calculating feature importance specific to this Random Forest was not developed.
* Cross-Validation within C++ Training: The current C++ training trains on the whole training split. Implementing k-fold cross-validation within the C++ training loop for more robust model parameter assessment was not part of this phase.
* More Sophisticated Recommendation Engine: The rule-based engine is simple; it could be expanded with more nuanced rules or draw from a larger knowledge base.
* Error Handling and Logging: While some basic error handling exists, a more comprehensive error handling and logging mechanism could be implemented.
* No GUI: The interface is CLI-only.

# Planning for Phase II

Phase II could focus on enhancements and addressing some of the incomplete areas:

* GUI Development
* Enhanced Recommendation System
* Model Iteration and Improvement using Advanced Hyperparameter Tuning
* Feature Engineering & Selection
* Feedback Mechanism

# Location for GitHub repository for code sharing

The repo can be accessed via the following url:

<https://github.com/DiwBhat/Depression-Classification/cpp_port>