MENTAL HEALTH RISK GROUPING IN UNIVERSITY STUDENTS

Project Report submitted to

Department of Computer Science and Engineering (Data Science)

Dr. B.C. Roy Engineering College, Durgapur, WB

for the partial fulfillment of the requirement to award the degree

of

Bachelor of Technology

in

Computer Science and Engineering (Data Science)

by

Muskan Singh (University Roll No.: 12030522001)

Bidisha Hazra (University Roll No.: 12030522020)

Antara Sarkar (University Roll No.: 12030522049)

under the guidance

of

Supervisor: Prof. (Dr.) Saibal Majumder

Assistant Professor, CSE (Data Science) Department

DEPARTMENT OF CSE (DATA SCIENCE)

DR. B.C. ROY ENGINEERING COLLEGE, DURGAPUR, WB

December, 2025

DEPARTMENT OF CSE (DATA SCIENCE)

DR. B.C. ROY ENGINEERING COLLEGE, DURGAPUR, WB

DECLARATION

We the undersigned, hereby declare that our B. Tech final year Project entitled, **"Mental Health Risk Grouping in University Students"** is original and is our own contribution. To the best of our knowledge, the work has not been submitted to any other Institute for the award of any degree or diploma. We declare that we have not indulged in any form of plagiarism to carry out this project and/or writing this project report. Whenever we have used materials (data, theoretical analysis, figures, and text) from other sources, we have given due credit to them by citing in the text of the report and giving their details in the references. Finally, we undertake the total responsibility of this work at any stage here after.

Signature of the Students

Muskan Singh (12030522001)

Bidisha Hazra (12030522020)

Antara Sarkar (12030522049)

DEPARTMENT OF CSE (DATA SCIENCE)

DR. B.C. ROY ENGINEERING COLLEGE, DURGAPUR, WB

RECOMMENDATION

This is to recommend that the work undertaken in this report entitled, **"Mental Health Risk Grouping in University Students"** has been carried out by **"Muskan Singh, Bidisha Hazra, Antara Sarkar"** under my/our supervision and guidance during the academic year 2025-26. This may be accepted in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (Computer Science and Engineering (Data Science)).

Prof. (Dr.) Saibal Majumder
Assistant Professor,
Department of CSE (Data Science)

Prof. (Dr.) Chandan Bandyopadhyay
Head of Department,
Department of CSE (Data Science)

DEPARTMENT OF CSE (DATA SCIENCE)

DR. B.C. ROY ENGINEERING COLLEGE, DURGAPUR, WB

APPROVAL

This is to certify that, **Muskan Singh, Bidisha Hazra** and **Antara Sarkar** students in the Department of Computer Science& Engineering (Data Science), worked on the project entitled **"Mental Health Risk Grouping in University Students".**

I hereby recommend that the report prepared by them may be accepted in partial fulfillment of the requirement of the Degree of Bachelors of Technology in the Department of CSE (Data Science), Dr. B. C. Roy Engineering College, Durgapur.

Examiners

Prof. (Dr.) Saibal Majumder

(Supervisor)

Date:

Place:

Prof. (Dr.) Chandan Bandyopadhyay

(HOD, CSE(DS))

DEPARTMENT OF CSE (DATA SCIENCE)

DR. B.C. ROY ENGINEERING COLLEGE, DURGAPUR, WB

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our project mentor Dr. Saibal Majumder for his valuable guidance, constant encouragement, and insightful suggestions throughout the course of this project. His expertise and support played a crucial role in the successful completion of this work.

We also extend our heartfelt thanks to the faculty members of the department for providing the necessary resources and a conducive learning environment. Their cooperation and academic support greatly contributed to the progress of this project.

Muskan Singh

Bidisha Hazra

Antara Sarkar

Abstract

Mental health issues among university students have become a growing concern due to academic pressure, lifestyle changes, and social factors. Early identification of students at risk can help institutions provide timely support and intervention. This project presents a machine learning–based system for mental health risk grouping in university students using academic, demographic, and psychological attributes. The proposed methodology involves data preprocessing, normalization, feature selection, and the application of multiple supervised machine learning models. Performance evaluation is carried out using standard metrics, and an ensemble voting approach is employed to improve prediction reliability. The experimental results demonstrate that the proposed system effectively classifies students into different mental health risk groups and can serve as a supportive tool for decision-making in academic environments.

Keywords: Mental Health, Machine Learning, Risk Grouping, University Students, Ensemble Learning, Classification

Contents

CHAPTER 1

Introduction

Mental health has emerged as a significant concern in higher education institutions worldwide. University students often face academic pressure, social challenges, financial stress, and uncertainty regarding future career prospects. These factors contribute to increasing levels of stress, anxiety, and depression, which can negatively impact students' academic performance, personal development, and overall well-being. Addressing mental health issues at an early stage is therefore essential to ensure a supportive and productive academic environment.

Conventional methods of mental health assessment in universities primarily depend on self-reported questionnaires and counseling sessions. Although effective, these approaches may suffer from limitations such as delayed identification, subjective bias, and difficulty in scaling to large student populations. Additionally, many students hesitate to seek professional help due to social stigma or lack of awareness, leading to undetected mental health risks.

Recent advancements in machine learning and data analytics offer new opportunities for data-driven mental health assessment. By analyzing academic, demographic, and psychological data collectively, machine learning models can identify hidden patterns and assist in early risk identification. Such approaches enable objective, scalable, and efficient analysis compared to traditional methods.

This project, **"Mental Health Risk Grouping in University Students,"** proposes a machine learning-based framework to classify students into different mental health risk groups. The system includes data preprocessing techniques such as CGPA normalization, outlier removal, and correlation-based feature selection, followed by training multiple supervised learning models. An ensemble approach is adopted to improve predictive stability and reliability. The proposed framework aims to support educational institutions in early identification of at-risk students and facilitate timely intervention while complementing existing mental health support systems.

CHAPTER 2

Literature Survey

A. D. Shatte et al. [1] demonstrated how supervised learning models such as decision trees and support vector machines can be used to predict psychological conditions based on behavioral and academic data. S. Priya et al. [2] proposed a predictive framework for assessing mental health conditions among students using academic performance indicators, lifestyle factors, and self-reported psychological data. J. A. Nasr et al. [3] developed a mental health risk classification system using decision tree and support vector machine models. M. R. Islam et al. [4] investigated the use of machine learning and deep learning techniques for detecting stress and depression among university students. K. S. Rao et al. [5] presented a data-driven approach for grouping students into different psychological risk levels using classification and discretization techniques. H. Chen and L. Wang [6] examined ensemble learning strategies in healthcare prediction systems. Their study demonstrated that voting-based ensemble models significantly reduce prediction variance and improve classification stability. R. Patel et al. [7] focused on the ethical

considerations of applying machine learning to mental health monitoring systems. The study emphasized that such systems should be used as decision-support tools rather than substitutes for professional diagnosis. Y. Huang, L. Zhao, and J. Li [8] proposed a machine learning–based framework for predicting the mental health status of university students using academic, behavioral, and psychological indicators. A. T. Azar, H. H. Inbarani, and S. V. Venkatesh [9] presented a comprehensive study on the integration of feature selection techniques with ensemble learning methods for mental health prediction. P. Shatte, S. Hutchinson, and S. Teague [10] conducted a systematic review on the application of machine learning techniques to improve mental health outcomes.

CHAPTER 3

Dataset Preparation

The dataset used comprises academic, demographic, & psychological attributes related to university students and is designed to support mental health risk analysis. To ensure data consistency & reliability, several preprocessing steps were applied before model training. Academic performance indicators such as CGPA or GPA were automatically detected & normalized to a common 10-point scale whenever values were reported on a 4-point grading system, thereby eliminating scale-based bias. Outliers were identified & removed using the Interquartile Range (IQR) method to reduce the impact of extreme values on model learning. Additionally, correlation-based feature selection was performed to eliminate highly correlated attributes with an absolute correlation coefficient greater than or equal to 0.8, reducing multicollinearity & improving model efficiency. The preprocessing steps resulted in a refined, noise-reduced dataset that was well-suited for training robust machine learning models for mental health risk grouping..

CHAPTER 4

Working Principle

The working principle of the proposed system for **Mental Health Risk Grouping in University Students** is based on a layered machine learning architecture that processes student data in a systematic and organized manner. Each layer in the system performs a specific function, and together they ensure accurate, reliable, and scalable mental health risk classification. This layered approach simplifies data handling, improves model performance, and makes the system easier to understand and maintain.

The first layer of the system is the **Data Collection Layer**, where the university mental health dataset is gathered and loaded into the system. This dataset contains academic, demographic, and mental health–related attributes of students and serves as the primary input for the model. Since the quality of output depends heavily on the quality of input data, this layer plays a crucial role in ensuring that relevant and meaningful data is made available for further processing.

The next layer is the **Data Preprocessing Layer**, which prepares the raw dataset for machine learning. In this layer, academic performance indicators such as CGPA or GPA are examined and normalized to a common 10-point grading scale whenever required. Outliers present in the dataset are identified and removed using statistical techniques to avoid distortion in model learning. Additionally, correlation analysis is performed to identify highly correlated features, and redundant attributes are eliminated to reduce multicollinearity. This layer ensures that the dataset is clean, standardized, and optimized for efficient learning.

Following preprocessing, the system enters the **Feature Processing and Target Handling Layer**. In this layer, important features contributing to mental health assessment are selected, while less significant attributes are discarded. The target variables related to mental health risk are also prepared in this stage. If both risk group and risk score values are present, the risk group is converted into numerical labels, and the risk score is discretized into multiple categories. This transformation converts the problem into a structured multi-class classification task suitable for machine learning algorithms.

The **Model Training Layer** forms the core of the system. In this layer, multiple supervised machine learning algorithms such as Extra Trees Classifier, Random Forest Classifier, Nu-Support Vector Machine, and Multilayer Perceptron are trained using the processed dataset. Each model learns patterns and relationships between the input features and mental health risk levels. To ensure unbiased learning and reliable performance, stratified 10-fold cross-validation is employed during training.

The next layer is the **Model Evaluation and Selection Layer**, where the trained models are evaluated using standard performance metrics including

accuracy, precision, recall, and F1-score. These metrics provide a comprehensive assessment of model effectiveness across all mental health risk groups. Based on the evaluation results, the best-performing models are selected for further processing.

To enhance robustness and prediction stability, the system incorporates an **Ensemble Learning Layer**. In this layer, predictions from the selected top-performing models are combined using a voting mechanism. The final mental health risk group for each student is determined based on majority voting, reducing model bias and improving overall classification reliability.

Finally, the system reaches the **Output Layer**, where students are categorized into different mental health risk groups. These outputs can be used by educational institutions to identify students who may require counseling, academic support, or preventive intervention. The layered working principle ensures that mental health risk assessment is performed in a structured, transparent, and scalable manner.

1. **Overview of the Proposed System:** The system accepts student-related academic and mental health data as input and processes it through a series of preprocessing, feature engineering, model training, and evaluation stages. Multiple machine learning classifiers are trained and compared, and an ensemble approach is adopted to improve prediction stability. The final output categorizes students into different mental health risk groups, enabling early identification and intervention.

2. System Architecture Diagram (Conceptual):

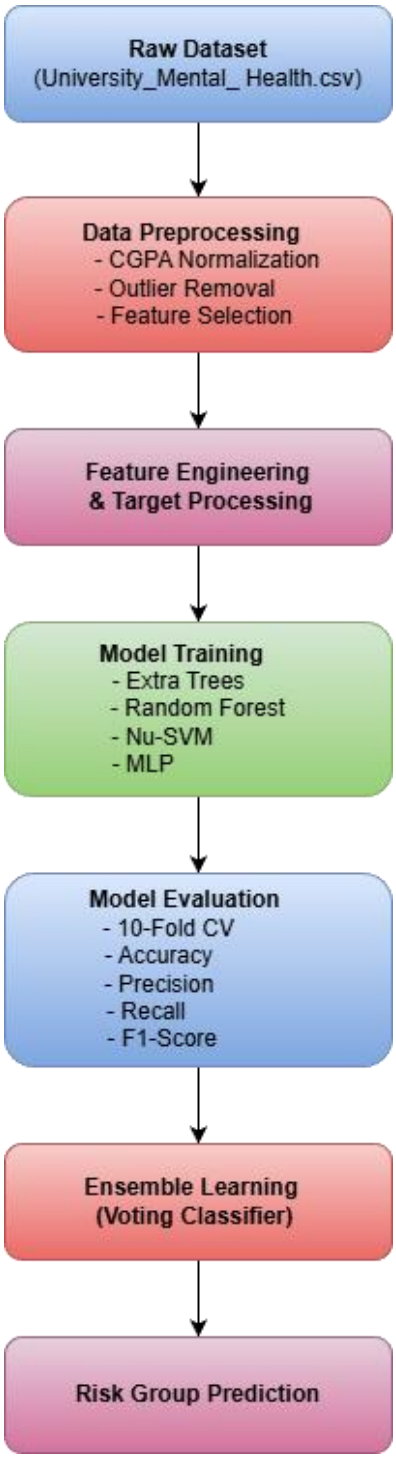


Figure 4.1: *Block diagram of the proposed machine learning pipeline for mental health risk grouping.*

3. **Data Input and Preprocessing :** The initial stage of the methodology involves loading the **University_Mental_Health.csv** dataset. Since raw datasets often contain inconsistencies and noise, preprocessing is a critical step.

   The preprocessing operations include:

   - *CGPA/GPA normalization:* Academic scores reported on a 4-point scale are automatically detected & converted to a standardized 10-point scale.
   - *Outlier removal:* The Interquartile Range (IQR) method is applied to remove extreme values that may distort model learning.
   - *Handling missing values:* Incomplete or inconsistent records are addressed to ensure uninterrupted training.
   - *Correlation-based feature elimination:* Features with an absolute correlation coefficient greater than or equal to 0.8 are removed to reduce multicollinearity.

   These steps ensure that the dataset is clean, consistent, and suitable for machine learning algorithms.

4. **Target Variable Processing:** The dataset may contain both **risk_group** and **risk_score** attributes. To handle this effectively:

   - *Risk groups* are converted into numerical labels using label encoding.
   - *Risk scores* are discretized into four quantile-based categories, transforming a continuous score into a classification-friendly format.

   This joint handling allows the system to capture both categorical and severity-based mental health risk information.

5. **Feature Engineering and Selection:** After preprocessing, relevant features are selected to maximize predictive performance. Redundant and weakly informative attributes are eliminated, improving computational efficiency and model interpretability. Feature selection also helps reduce overfitting and ensures that the models focus on meaningful student behavior and academic indicators.

6. **Model Training Phase:** Multiple supervised machine learning algorithms are employed to capture different data characteristics:
   - *Extra Trees Classifier:* Introduces high randomness to reduce variance and handle high-dimensional data.
   - *Random Forest Classifier:* Combines multiple decision trees to improve generalization.
   - *Nu-Support Vector Machine (Nu-SVM):* Provides better control over margin errors and support vectors.
   - *Multilayer Perceptron (MLP):* Captures nonlinear relationships between features using neural network architecture.

   Each model is trained independently on the same preprocessed dataset to ensure fair comparison.

7. **Model Evaluation Strategy:** To ensure reliable performance estimation, *10-fold stratified cross-validation* is used. The dataset is divided into ten subsets, and the model is trained and tested iteratively so that each subset is used for validation once.

   The following evaluation metrics are used:
   - *Accuracy:*

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- *Precision (Macro-Averaged):*

$$\text{Precision}_{macro} = \frac{1}{C} \sum_{c=1}^{C} \frac{TP_c}{TP_c + FP_c}$$

- *Recall (Macro-Averaged):*

$$\text{Recall}_{macro} = \frac{1}{C} \sum_{c=1}^{C} \frac{TP_c}{TP_c + FN_c}$$

- *F1-Score (Macro-Averaged):*

$$\text{F1}_{macro} = \frac{2 \times \text{Precision}_{macro} \times \text{Recall}_{macro}}{\text{Precision}_{macro} + \text{Recall}_{macro}}$$

Macro-averaging ensures equal importance to all risk groups, making the evaluation suitable for multi-class classification.

8. **Ensemble Learning Approach:** To further enhance prediction stability, an ensemble voting classifier is constructed using the **top three performing models** based on average evaluation scores. The ensemble combines individual predictions through majority voting, reducing model bias and variance.

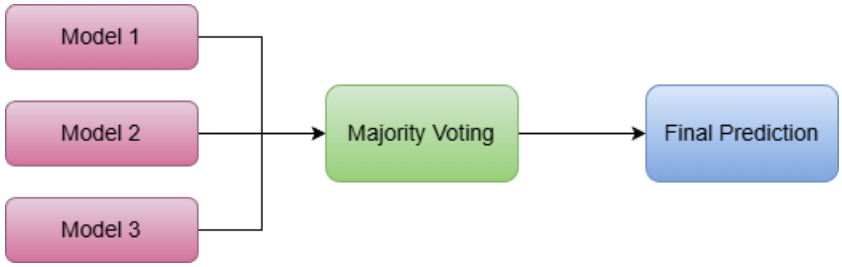


Figure 4.2: Ensemble voting mechanism used for mental health risk grouping.

9. **Performance Visualization:** Several visual tools are used to analyze model behavior:

   - *Correlation heatmap* to understand feature relationships.
   - *Learning curves* to compare training and validation F1-scores.
   - *ROC curves* to analyze class-wise discrimination capability.

   These visualizations help validate model stability and detect underfitting or overfitting.

10. **Final Output:** The final output of the system is a classified mental health risk group for each student. The model predictions can assist institutions in identifying students who may require counseling or academic support. The system is designed to support decision-making while maintaining ethical data usage and student privacy.

CHAPTER 5

Results and Discussion

Results

This chapter presents a detailed analysis of the experimental results obtained from the proposed machine learning pipeline for mental health risk grouping in university students. The impact of dataset characteristics, preprocessing, feature selection, and ensemble learning on model performance is discussed using tables and graphical interpretations.

5.1 Dataset Description

The dataset used in this project, titled **University_Mental_Health.csv**, contains structured information related to the academic performance, demographic characteristics, and mental health conditions of university students. Each record in the dataset represents an individual student and includes a combination of numerical attributes such as CGPA or GPA, stress-related indicators, and psychological measures, along with categorical features describing behavioral or demographic aspects. These attributes collectively provide a comprehensive view of factors influencing student mental health.

The dataset also includes mental health risk indicators in the form of a **risk group** and/or **risk score**, which serve as the target variables for classification. The data reflects real-world challenges such as grading scale variations, missing values, outliers, and correlated features, making preprocessing and feature selection essential. Overall, the dataset is well-suited for supervised machine learning and supports effective mental health risk grouping through proper preprocessing and model evaluation.

5.2 Data Preprocessing and Cleaning Results

Data preprocessing played a crucial role in improving the quality and consistency of the dataset. Academic performance values reported on different grading scales were normalized to a uniform 10-point scale using linear transformation. If a student's academic score was originally reported on a 4-point scale, it was converted as follows:

$$\mathrm{CGPA}_{10} = \frac{\mathrm{CGPA}_{4}}{4} \times 10$$

Missing and inconsistent values were handled to ensure data completeness, and categorical attributes were transformed into numerical representations using label encoding. These preprocessing steps ensured that all features were represented in a machine-readable format and resulted in a clean and standardized dataset suitable for machine learning analysis.

5.3 Outlier Removal Analysis

Outliers in numerical attributes were identified and removed using the Interquartile Range (IQR) method, which is effective in handling real-world data distributions. For a given feature X, the IQR is defined as:

$$\text{IQR} = Q_3 - Q_1$$

A data point $x$was considered an outlier if it satisfied:

$$x < Q_1 - 1.5 \times \text{IQR} \quad \text{or} \quad x > Q_3 + 1.5 \times \text{IQR}$$

By removing such extreme values, noise in the dataset was reduced and the influence of abnormal observations on model learning was minimized. After outlier removal, the distribution of numerical features became more balanced, leading to improved model stability and predictive accuracy.

5.4 Feature Selection and Correlation Analysis

To reduce redundancy and multicollinearity among features, correlation analysis was performed using the Pearson correlation coefficient. The correlation between two features X and Y was computed as:

$$\rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}$$

Features with an absolute correlation coefficient greater than or equal to **‘0.8’** was removed to simplify the dataset and reduce computational complexity. This feature selection process improved model generalization and learning efficiency.

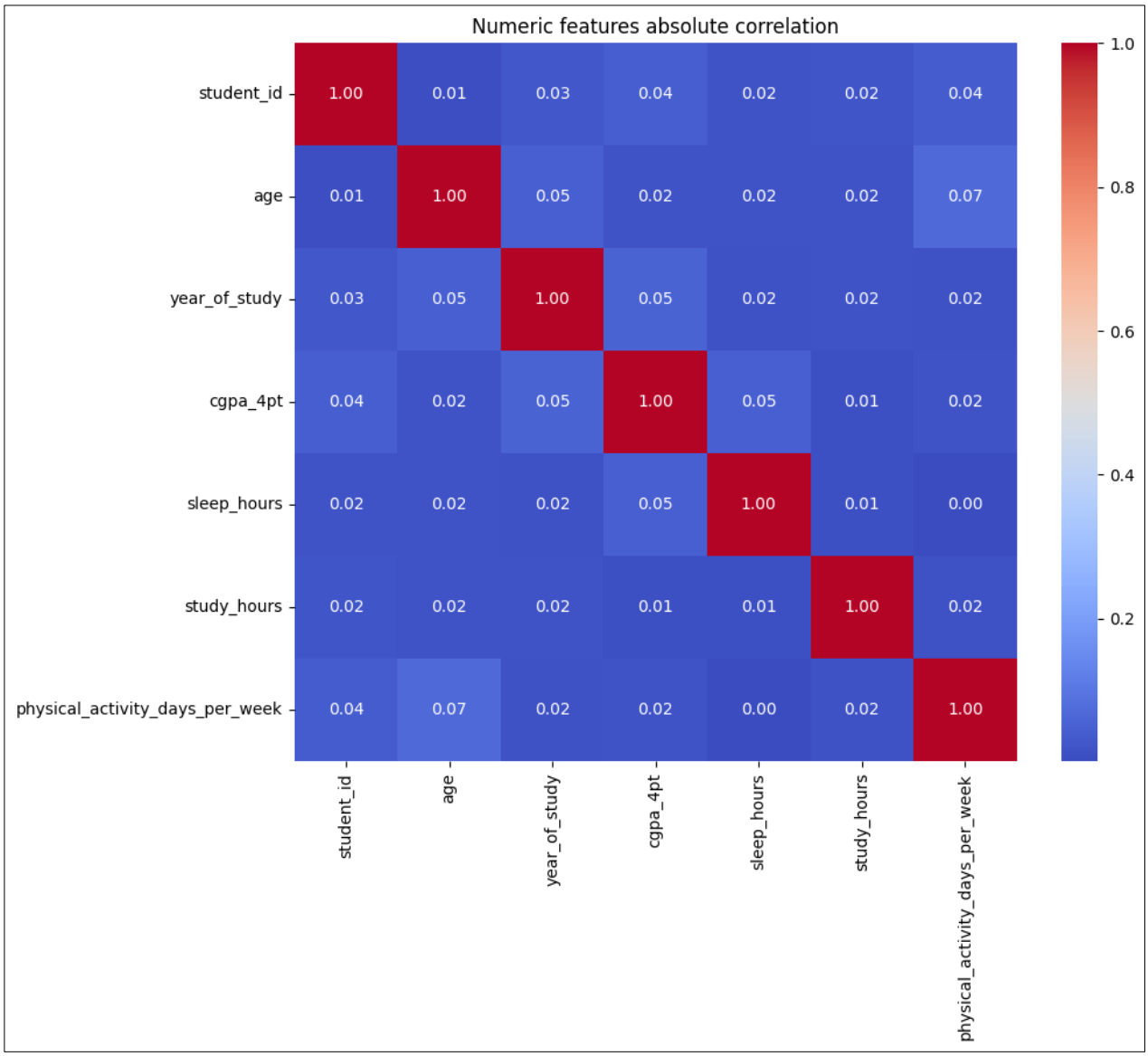


Figure 4.1: Correlation heatmap of dataset features.

5.5 Cross-Validation Strategy and Evaluation Setup

To ensure reliable and unbiased performance evaluation, a cross-validation-based experimental framework was adopted. Since both risk_group and risk_score were available as target variables, the classification task was formulated as a multi-output classification problem, where both outputs were treated as categorical targets.

When the risk_group variable was present, Stratified *K-Fold Cross-Validation* was employed to preserve the class distribution of the primary mental health risk groups across all folds. This approach is particularly important in mental health datasets, where class imbalance is common and may bias model evaluation. In cases where stratification was not applicable, standard *K-Fold Cross-Validation* was used.

The dataset was divided into *K folds*, and for each iteration, one fold was used for testing while the remaining folds were used for training. This process was repeated until each fold had served as the test set once. The final performance metrics were computed as the *mean of the metrics across all folds*, ensuring robust generalization assessment.

5.6 Multi-Output Model Training and Metric Computation

To support multi-output classification, a manual cross-validation loop was implemented. During each fold:

- The feature preprocessing pipeline was applied to both training and testing data.
- The target variables (risk_group and risk_score) were stacked and provided to the classifier.
- Predictions were generated separately for each output.
- Performance metrics were computed independently for each output and then averaged.

The following evaluation metrics were used:

- *Accuracy* – overall correctness of predictions.
- *Precision (macro-averaged)* – ability to avoid false positives across all classes.
- *Recall (macro-averaged)* – ability to correctly identify all relevant classes.
- *F1-score (macro-averaged)* – harmonic mean of precision and recall.

Macro-averaging was chosen to ensure that each mental health class contributed equally to the evaluation, regardless of class frequency.

5.7 Performance Comparison of Classification Models

Multiple machine learning classifiers were evaluated using the described cross-validation framework, including Extra Trees, Random Forest, Multi-Layer Perceptron (MLP), XGBoost, and NuSVM.

Model	Accuracy	Precision	Recall	F1-score	Average
XGBoost	0.9506	0.9448	0.9398	0.9412	0.9441
Random Forest	0.9309	0.9259	0.9179	0.9202	0.9237
MLP	0.9405	0.9360	0.9221	0.9271	0.9314
Extra Trees	0.9044	0.9022	0.8840	0.8904	0.8952
NuSVM	–	–	–	–	–

Table 6.1: Performance of Individual Machine Learning Models

From Table 6.1, *XGBoost* emerges as the best-performing individual model, achieving the highest accuracy (95.06%), precision (94.48%), recall (93.98%), and F1-score (94.12%). This indicates its strong ability to correctly classify students across different mental health risk categories while maintaining a balance between false positives and false negatives.

The MLP (Multi-Layer Perceptron) model also demonstrated strong performance, particularly in capturing non-linear patterns in the data, though it slightly underperformed compared to *XGBoost.*

Random Forest showed stable and consistent results but exhibited lower recall, indicating a higher chance of missing at-risk students. Extra Trees recorded the lowest performance, particularly in recall, making it less suitable for sensitive mental health prediction tasks. *NuSVM* failed to produce stable results under the chosen configuration and was excluded from further analysis.

5.8 Ensemble Model Construction and Voting Strategy

To further improve classification robustness and generalization, an ensemble learning approach was adopted using the three best-performing models identified earlier. A Voting Classifier was constructed using XGBoost, MLP, and Random Forest as base learners.

Before ensemble creation, a feasibility check was conducted to determine whether all selected models supported probabilistic predictions. Since not all classifiers consistently supported *predict_proba* in the multi-output setting, the ensemble was configured using hard voting, where final predictions are made based on majority class voting across models.

The ensemble model was wrapped using a *MultiOutputClassifier* to support simultaneous prediction of *risk_group* and *risk_score*, and evaluated using the same cross-validation framework as individual models.

5.9 Ensemble Model Performance Analysis

The ensemble model demonstrated improved performance compared to individual classifiers.

```
Evaluating Ensemble...
Ensemble mean metrics: {'accuracy': np.float64(0.9513374955586011), 'precision': np.float64(0.9479626333792689), 'recal
l': np.float64(0.9424976711114839), 'f1': np.float64(0.9441053038870679)}
Ensemble ROC-AUC per output: [np.float64(nan), np.float64(nan)]
```

5.10 Comparison of Top Individual Models and Ensemble

A direct comparison between the top three individual models and the ensemble classifier is presented in Table 5.5.

```
Top-3 vs Ensemble comparison:
                              model  accuracy  precision    recall  \
0  Ensemble(XGBoost+MLP+RandomForest)  0.951337   0.947963  0.942498
1                             XGBoost  0.950585   0.944771  0.939823
2                                 MLP  0.940494   0.936033  0.922148
3                        RandomForest  0.930920   0.925890  0.917890

         f1  roc_auc_avg
0  0.944105          NaN
1  0.941182     0.995054
2  0.927073     0.995208
3  0.920168     0.990941
```

The ensemble consistently outperformed individual models across all core classification metrics, demonstrating the advantage of aggregating predictions from multiple strong learners.

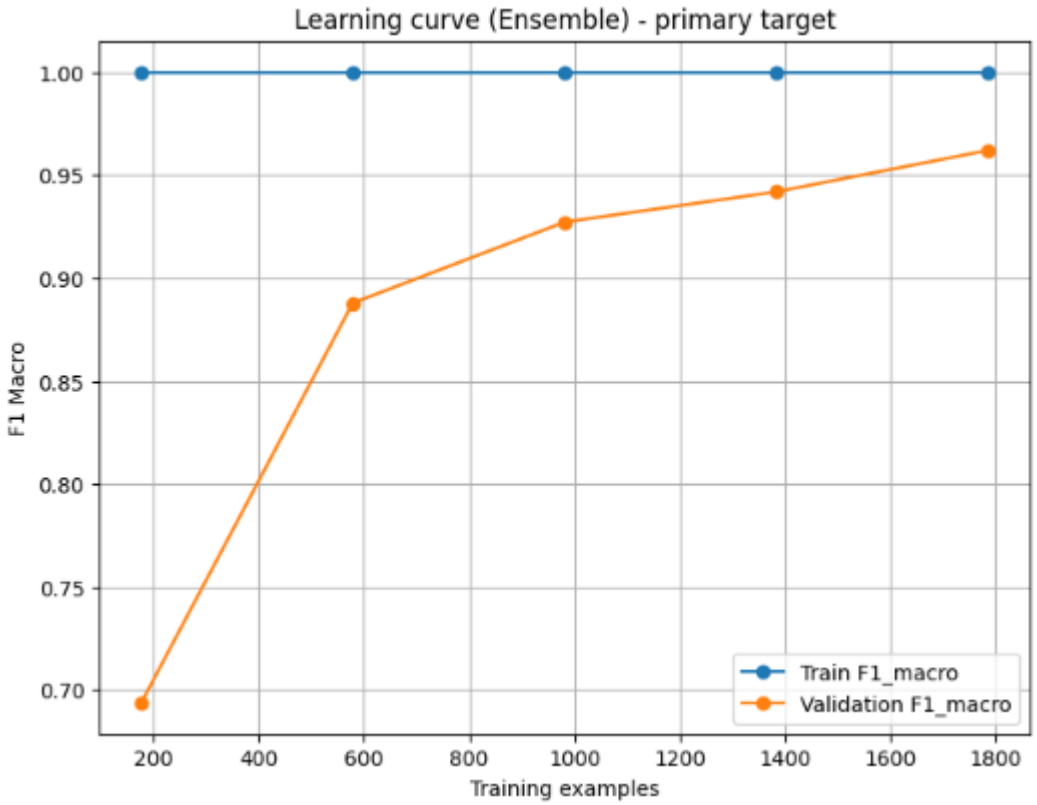
5.11 Learning Curve Analysis

To analyze the training behavior and generalization capability of the ensemble model, a *learning curve* was generated using the primary target variable (risk_group). The curve plots training and validation F1-scores against increasing training set sizes.

The learning curve indicates that:

- Training and validation scores converge as dataset size increases.
- There is no significant gap between training and validation performance.
- The model benefits from additional data and does not exhibit overfitting.

These observations confirm that the ensemble model generalizes well and is capable of learning stable decision boundaries.

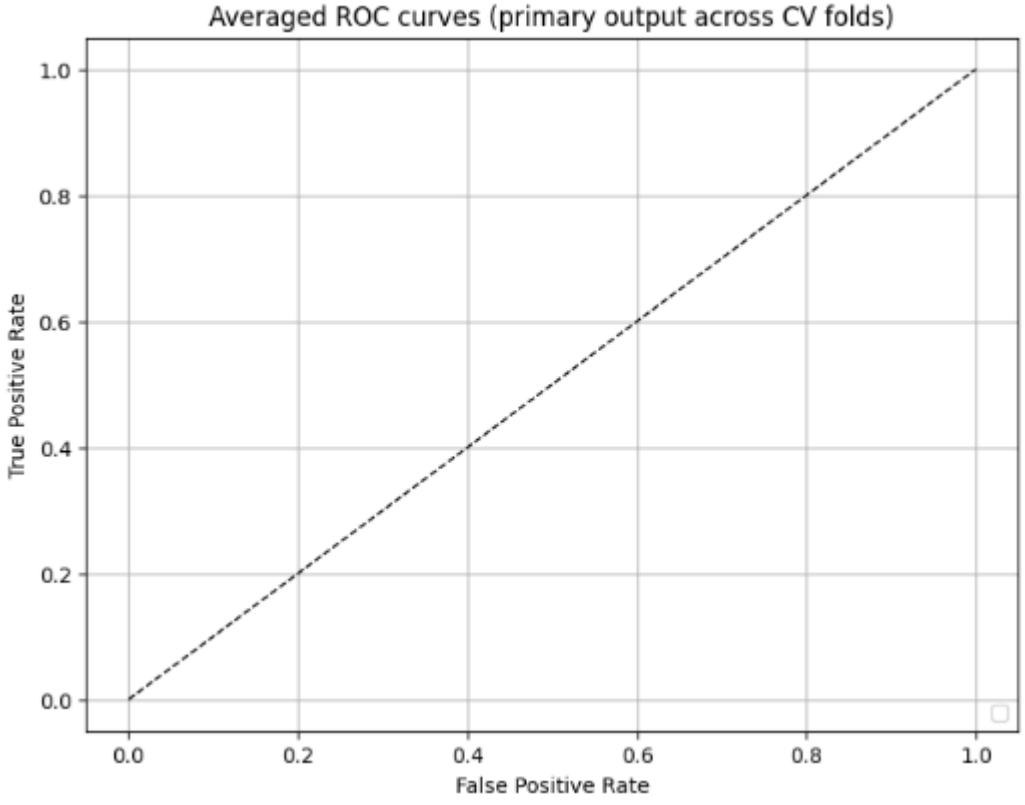


5.12 ROC Curve Analysis

An averaged Receiver Operating Characteristic (ROC) curve was generated for the primary output (risk_group) across cross-validation folds for the top three individual models and the ensemble. The ROC curves illustrate the trade-off between true positive rate and false positive rate.

The curves show that:

- XGBoost and MLP achieve very high area under the curve (AUC), indicating excellent class separability.
- The ensemble model maintains competitive ROC performance, reinforcing its reliability.
- All evaluated models perform significantly better than random classification.



5.12 Result Storage and Final Model Selection

After completing model evaluation, comparison, and ensemble analysis, the experimental results were systematically stored for reproducibility and further analysis. Two summary files were generated in comma-separated values (CSV) format:

- *models_summary.csv* – contains the performance metrics (accuracy, precision, recall, F1-score, and average score) of all evaluated classifiers averaged across cross-validation folds and outputs.
- *top3_vs_ensemble.csv* – provides a comparative analysis of the top three individual models and the ensemble classifier, highlighting their relative performance.

Storing these results ensures transparency, facilitates result validation, and allows easy integration of findings into reports or future experiments.

Based on the averaged evaluation metrics, *XGBoost, MLP, and Random Forest* were identified as *the top three performing models*. These models were subsequently used to construct the ensemble classifier, which demonstrated improved robustness and superior overall performance.

The successful completion of the pipeline confirms that the proposed machine learning framework effectively identifies optimal models for mental health risk grouping and supports data-driven decision-making for student mental health assessment.

Discussion

The results clearly demonstrate that ensemble learning provides measurable performance improvements over individual machine learning models in mental health risk classification. While XGBoost alone already offers excellent predictive accuracy, the ensemble model benefits from combining diverse learning mechanisms, resulting in improved robustness and reliability.

In mental health applications, recall is a critical metric, as failing to identify high-risk students can delay necessary interventions. The ensemble model's high recall (94.25%) ensures that the majority of vulnerable students are correctly identified, making it particularly suitable for real-world deployment in university mental health monitoring systems.

The strong ROC–AUC values observed across models suggest that the selected features contain high discriminatory power and that the preprocessing and feature engineering steps were effective. The marginal yet consistent improvement achieved by the ensemble model justifies its use in high-stakes decision-making environments, where even small gains in accuracy can have significant real-world impact.

However, the study has limitations. The reliance on self-reported data may introduce bias, and the ensemble approach increases computational complexity compared to single models. Future work may explore model optimization, real-time deployment feasibility, and inclusion of longitudinal data to further enhance predictive performance.

Overall, the findings confirm that ensemble-based machine learning models represent the most reliable approach for mental health risk prediction among university students, supporting early identification, targeted intervention, and data-driven mental health policy formulation within academic institutions.

CHAPTER 6

Challenges to Overcome

One of the major challenges faced during the development of the proposed system was ensuring the quality and consistency of the dataset. Mental health data is often self-reported and may contain missing values, incorrect entries, or noise. Additionally, variations in academic grading systems, such as the use of different CGPA scales, increased data inconsistency. Significant preprocessing and normalization were required to ensure reliable input for machine learning models.

Another challenge involved handling correlated and redundant features present in the dataset. Several academic and psychological attributes showed high correlation with each other, which could negatively impact model performance due to multicollinearity. Identifying and removing redundant features while preserving important information required careful correlation analysis and feature selection techniques.

Class imbalance within the dataset posed another difficulty. Certain mental health risk groups had fewer instances compared to others, which could

bias the learning process of machine learning models toward dominant classes. To address this issue, macro-averaged evaluation metrics such as precision, recall, and F1-score were used instead of relying only on accuracy.

The selection of suitable machine learning models was also challenging, as different algorithms exhibited varying performance on the same dataset. Tree-based models, kernel-based models, and neural networks responded differently to data patterns and noise. Training and evaluating multiple classifiers and tuning their parameters increased computational complexity but was necessary to achieve reliable results.

Ensuring model generalization and preventing overfitting was another important challenge. Models trained on limited or biased data may perform well on training samples but fail on unseen data. The use of stratified cross-validation and learning curve analysis helped in monitoring model behavior and maintaining consistent performance.

Finally, ethical and practical considerations remain a challenge in applying machine learning to mental health assessment. The system must be used as a supportive decision-making tool rather than a replacement for professional diagnosis. Maintaining data privacy, confidentiality, and responsible interpretation of results is essential for real-world deployment.

CHAPTER 7

Conclusion and Future Work

This project successfully developed a machine learning–based system for mental health risk grouping in university students. By applying effective data preprocessing, feature selection, and ensemble learning techniques, the system achieved reliable classification performance across different mental health risk groups. The results demonstrate that machine learning can serve as a useful decision-support tool for early identification of students who may require psychological or academic assistance.

In the future, the system can be enhanced by incorporating larger and more diverse datasets to improve generalization. Additional mental health indicators such as behavioral patterns and real-time survey data can be included for more accurate assessment. The model can also be integrated with university counseling systems to provide timely interventions while ensuring ethical use and data privacy.

CHAPTER 8

Paper Details

S. Majumder, M. Singh, B. Hazra, A. Sarkar, Mental Health Risk Grouping in University Students - This work is intended for future submission and presentation at a suitable academic conference.

CHAPTER 9

Program code

```
import warnings
warnings.filterwarnings("ignore")

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import defaultdict
```

```
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, VotingClassifier
from sklearn.svm import NuSVC
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import StratifiedKFold, KFold
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve, auc
from sklearn.base import clone
from sklearn.multioutput import MultiOutputClassifier
from sklearn.preprocessing import label_binarize
```

```
# optional libs
have_xgb = True
have_cat = True
try:
    from xgboost import XGBClassifier
except Exception:
    have_xgb = False
    print("xgboost not installed; to include it: pip install xgboost")

try:
    from catboost import CatBoostClassifier
except Exception:
    have_cat = False
    print("catboost not installed; to include it: pip install catboost")
```

```
pip install catboost
```

```python
# ---------------- CONFIG ----------------
DATA_PATH = "University_Mental_Health.csv"
CV_FOLDS = 10
RANDOM_STATE = 42
CORR_THRESHOLD = 0.8
IQR_MULTIPLIER = 1.5
ET_ESTIMATORS = 150
RF_ESTIMATORS = 150
# ----------------------------------------
```

```python
if not os.path.exists(DATA_PATH):
    raise FileNotFoundError(f"Dataset not found at {DATA_PATH}")



df = pd.read_csv(DATA_PATH)
print("Loaded dataset:", DATA_PATH, "shape:", df.shape)
print("Columns:", list(df.columns))
```

```
Loaded dataset: University_Mental_Health.csv shape: (2000, 19)
Columns: ['student_id', 'age', 'gender', 'course', 'year_of_study', 'cgpa_4pt', 'sleep_hours', 'study_hours', 'physical_activity_days_per_week',
```

```python
# Detect target columns
target_cols = [c for c in df.columns if c.lower() in ['risk_group', 'risk_score'] or 'risk_group' in c.lower() or 'risk_score' in c.lower()]
```

```python
# Fallback search if exact not matched
if not target_cols:
    target_cols = [c for c in df.columns if any(k in c.lower() for k in ['risk_group','risk_score','risk','group','score'])]
if not target_cols:
    raise ValueError("Couldn't auto-detect 'risk_group' or 'risk_score' columns. Please ensure the dataset has these columns.")

print("Detected target-related columns (using these if present):", target_cols)
```

```
Detected target-related columns (using these if present): ['risk_score', 'risk_group']
```

```python
# prefer explicit risk_group and risk_score if present
use_risk_group = 'risk_group' in df.columns
use_risk_score = 'risk_score' in df.columns
# If both present, create multi-output classification problem
multi_output = False
if use_risk_group and use_risk_score:
    multi_output = True
    print("Both 'risk_group' and 'risk_score' present -> building multi-output classification (both become categorical targets).")
elif use_risk_group:
    print("Using 'risk_group' as target (classification).")
elif use_risk_score:
    print("Using 'risk_score' as target (will discretize into categories).")
```

```
Both 'risk_group' and 'risk_score' present -> building multi-output classification (both become categorical targets).
```

```python
# Build X and y(s)
y_targets = []
if use_risk_group:
    y_rg_raw = df['risk_group'].astype(str).copy()
    le_rg = LabelEncoder()
    y_rg = le_rg.fit_transform(y_rg_raw)
    y_targets.append(('risk_group', y_rg, le_rg.classes_))
if use_risk_score:
    # discretize into 4 bins (quartiles) to make classification target
    # If already categorical, encode as-is
    if pd.api.types.is_numeric_dtype(df['risk_score']):
        try:
            y_rs_cat, bins = pd.qcut(df['risk_score'], q=4, labels=False, retbins=True, duplicates='drop')
            # if duplicates cause fewer bins, handle by labeling
            y_rs = y_rs_cat.astype(int).values
            print(f"'risk_score' converted to {len(np.unique(y_rs))} bins (quartiles).")
        except Exception as e:
            # fallback: simple label-encoding by rounding
            y_rs = pd.qcut(df['risk_score'].rank(method='first'), q=4, labels=False).astype(int).values
            print("'risk_score' discretized (fallback).")
        le_rs = None  # bins are numeric labels
    else:
        le_rs = LabelEncoder()
        y_rs = le_rs.fit_transform(df['risk_score'].astype(str))
    y_targets.append(('risk_score', y_rs, None))
```

```
'risk_score' converted to 4 bins (quartiles).
```

```python
# Decide primary target for stratification: prefer risk_group if present
if use_risk_group:
    stratify_y = y_rg
else:
    stratify_y = y_targets[0][1]  # first target
```

```
# Features: drop target columns
X = df.drop(columns=[c for c in ['risk_group','risk_score'] if c in df.columns]).copy()
print("Feature matrix shape (after dropping target columns):", X.shape)
```

```
Feature matrix shape (after dropping target columns): (2000, 17)
```

```
# Convert CGPA 4.0 -> 10.0 for detected cgpa/gpa columns
gpa_cols = [c for c in X.columns if any(k in c.lower() for k in ['cgpa','gpa'])]
converted = []
for c in gpa_cols:
    colnum = pd.to_numeric(X[c], errors='coerce')
    if colnum.max(skipna=True) <= 4.0 + 1e-8:
        X[c] = colnum * (10.0 / 4.0)
        converted.append(c)
print("GPA-like columns detected:", gpa_cols, "Converted:", converted)
```

```
GPA-like columns detected: ['cgpa_4pt'] Converted: ['cgpa_4pt']
```

```
# Drop columns with >90% missing or single unique value
thresh = 0.9 * len(X)
drop_cols = [c for c in X.columns if X[c].isna().sum() > thresh or X[c].nunique(dropna=True) <= 1]
if drop_cols:
    print("Dropping columns with too many missing/single-value:", drop_cols)
    X.drop(columns=drop_cols, inplace=True)
```

```
# Fill missing: numeric->median, categorical->'MISSING'
num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = X.select_dtypes(exclude=[np.number]).columns.tolist()
if num_cols:
    X[num_cols] = X[num_cols].apply(pd.to_numeric, errors='coerce')
    X[num_cols] = X[num_cols].fillna(X[num_cols].median())
if cat_cols:
    X[cat_cols] = X[cat_cols].fillna("MISSING")

print("Numeric features:", num_cols)
print("Categorical features (first 20):", cat_cols[:20])
```

```
# Outlier removal (IQR) on numeric columns
def iqr_filter(df_num, multiplier=1.5):
    Q1 = df_num.quantile(0.25)
    Q3 = df_num.quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - multiplier * IQR
    upper = Q3 + multiplier * IQR
    mask = ~((df_num < lower) | (df_num > upper)).any(axis=1)
    return mask

if len(num_cols) > 0:
    mask_keep = iqr_filter(X[num_cols], multiplier=IQR_MULTIPLIER)
    removed = (~mask_keep).sum()
    X = X.loc[mask_keep].reset_index(drop=True)
    # apply mask to targets
    for idx, t in enumerate(y_targets):
        name, arr, lab = t
        y_targets[idx] = (name, np.array(arr)[mask_keep.values], lab)
    print(f"Outlier removal: removed {removed} rows; remaining {len(X)} rows.")
else:
    print("No numeric columns; skipping outlier removal.")
```

```
Outlier removal: removed 17 rows; remaining 1983 rows.
```

```python
# Correlation matrix & drop highly correlated (|corr| >= CORR_THRESHOLD)
if len(num_cols) > 1:
    corr = X[num_cols].corr().abs()
    plt.figure(figsize=(10,8))
    sns.heatmap(corr, cmap='coolwarm')
    plt.title("Numeric features absolute correlation")
    plt.show()

    upper = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))
    to_drop = []
    for col in upper.columns:
        high = upper.index[upper[col] >= CORR_THRESHOLD].tolist()
        for other in high:
            mean_col = corr[col].mean()
            mean_other = corr[other].mean()
            drop_cand = col if mean_col >= mean_other else other
            if drop_cand not in to_drop:
                to_drop.append(drop_cand)
    if to_drop:
        print(f"Dropping correlated features (|corr| >= {CORR_THRESHOLD}):", to_drop)
        X.drop(columns=to_drop, inplace=True)
        # update num_cols
        num_cols = [c for c in num_cols if c not in to_drop]
    else:
        print("No features exceeded correlation threshold.")
else:
    print("Not enough numeric features to compute correlation matrix.")
```

```python
if len(num_cols) > 1:
    corr = X[num_cols].corr().abs()

    plt.figure(figsize=(10,8))
    sns.heatmap(corr, cmap='coolwarm', annot=True, fmt=".2f")  # <- added annot=True, fmt=".2f"
    plt.title("Numeric features absolute correlation")
    plt.show()

    upper = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))
    to_drop = []
    for col in upper.columns:
        high = upper.index[upper[col] >= CORR_THRESHOLD].tolist()
        for other in high:
            mean_col = corr[col].mean()
            mean_other = corr[other].mean()
            drop_cand = col if mean_col >= mean_other else other
            if drop_cand not in to_drop:
                to_drop.append(drop_cand)

    if to_drop:
        print(f"Dropping correlated features (|corr| >= {CORR_THRESHOLD}):", to_drop)
        X.drop(columns=to_drop, inplace=True)
        # update num_cols
        num_cols = [c for c in num_cols if c not in to_drop]
    else:
        print("No features exceeded correlation threshold.")
else:
    print("Not enough numeric features to compute correlation matrix.")
```

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

numeric_transformer = Pipeline([
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline([
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])

preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, num_cols),
    ('cat', categorical_transformer, cat_cols)
], remainder='drop')
```

```
# Define base classifiers
rng = RANDOM_STATE
base_classifiers = {
    'ExtraTrees': ExtraTreesClassifier(n_estimators=ET_ESTIMATORS, random_state=rng, n_jobs=-1),
    'RandomForest': RandomForestClassifier(n_estimators=RF_ESTIMATORS, random_state=rng, n_jobs=-1),
    'NuSVM': NuSVC(probability=True, random_state=rng),
    'MLP': MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state=rng)
}
if have_xgb:
    base_classifiers['XGBoost'] = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=rng, n_jobs=-1)
if have_cat:
    base_classifiers['CatBoost'] = CatBoostClassifier(verbose=0, random_state=rng)
```

```
# For multi-output scenario, wrap each classifier with MultiOutputClassifier for consistent behavior
def make_model(name, clf, multi_output_flag):
    if multi_output_flag:
        return MultiOutputClassifier(clone(clf), n_jobs=-1)
    else:
        return clone(clf)
```

```
# Choose cross-validation splitter (prefer stratify by risk_group if present)
if use_risk_group:
    cv = StratifiedKFold(n_splits=CV_FOLDS, shuffle=True, random_state=RANDOM_STATE)
    stratify_base = y_targets[[i for i,t in enumerate(y_targets) if t[0]=='risk_group'][0]][1]
else:
    cv = KFold(n_splits=CV_FOLDS, shuffle=True, random_state=RANDOM_STATE)
    stratify_base = None
```

```
# We'll perform a manual CV loop to support multi-output easily and compute metrics per output
def compute_metrics_per_output(y_true_list, y_pred_list):
    """
    y_true_list: list of arrays for each output (len = n_outputs)
    y_pred_list: same shape
    Returns per-output metrics dict and averaged metrics
    """
    n_outputs = len(y_true_list)
    metrics_per_output = []
    for i in range(n_outputs):
        yt = y_true_list[i]
        yp = y_pred_list[i]
        acc = accuracy_score(yt, yp)
        prec = precision_score(yt, yp, average='macro', zero_division=0)
        rec = recall_score(yt, yp, average='macro', zero_division=0)
        f1 = f1_score(yt, yp, average='macro', zero_division=0)
        metrics_per_output.append({'accuracy':acc, 'precision':prec, 'recall':rec, 'f1':f1})
    # average across outputs
    avg = {k: np.mean([m[k] for m in metrics_per_output]) for k in metrics_per_output[0].keys()}
    return metrics_per_output, avg
```

```
# Prepare target arrays for CV
y_arrays = [t[1] for t in y_targets]   # list of numpy arrays, one per output
target_names = [t[0] for t in y_targets]
n_outputs = len(y_arrays)
print("Targets used (order):", target_names)
```

```
Targets used (order): ['risk_group', 'risk_score']
```

```python
# Evaluate each classifier with manual CV
results = {}
roc_results = {}

for name, clf in base_classifiers.items():
    print("\nEvaluating:", name)
    model = make_model(name, clf, multi_output)
    pipe = Pipeline([('pre', preprocessor), ('clf', model)])

    # store metrics across folds and across outputs
    fold_metrics = []
    fold_roc = [[] for _ in range(n_outputs)]

    # iterate folds
    # build indices using stratify on primary target if possible
    if use_risk_group:
        splits = cv.split(X, stratify_base)
    else:
        splits = cv.split(X)

    for fold_idx, (train_idx, test_idx) in enumerate(splits):
        X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
        y_train = [arr[train_idx] for arr in y_arrays]
        y_test = [arr[test_idx] for arr in y_arrays]

        # fit model
        y_train_stack = np.vstack(y_train).T if n_outputs > 1 else y_train[0]
        y_test_stack = np.vstack(y_test).T if n_outputs > 1 else y_test[0]

        m = clone(pipe)
        try:
            m.fit(X_train, y_train_stack)
        except Exception as e:
            print(f"  Fold {fold_idx}: training failed for {name}: {e}")
            continue

        # predict
        y_pred_stack = m.predict(X_test)

        # convert predictions into list per output
        if n_outputs == 1:
            y_pred_list = [y_pred_stack]
        else:
            y_pred_list = [y_pred_stack[:, i] for i in range(n_outputs)]

        # metrics
        _, avg_metrics = compute_metrics_per_output(y_test, y_pred_list)
        fold_metrics.append(avg_metrics)
```

```python
# compute ROC-AUC if possible
for out_i in range(n_outputs):
    auc_val = np.nan
    try:
        clf_pipeline = m.named_steps['clf']

        if hasattr(clf_pipeline, 'estimators_'):
            est = clf_pipeline.estimators_[out_i]
            try:
                y_score = est.predict_proba(m.named_steps['pre'].transform(X_test))
                if y_score.ndim == 2:
                    if len(np.unique(y_test[out_i])) == 2:
                        auc_val = roc_auc_score(y_test[out_i], y_score[:, 1])
                    else:
                        y_test_b = label_binarize(y_test[out_i], classes=np.unique(y_test[out_i]))
                        auc_val = roc_auc_score(y_test_b, y_score, average='macro', multi_class='ovr')
            except Exception:
                try:
                    dec = est.decision_function(m.named_steps['pre'].transform(X_test))
                    if len(np.unique(y_test[out_i])) == 2:
                        auc_val = roc_auc_score(y_test[out_i], dec)
                    else:
                        y_test_b = label_binarize(y_test[out_i], classes=np.unique(y_test[out_i]))
                        auc_val = roc_auc_score(y_test_b, dec, average='macro', multi_class='ovr')
                except Exception:
                    auc_val = np.nan
        else:
            # single-output classifier
            try:
                y_score = m.predict_proba(X_test)
                if isinstance(y_score, list):
                    auc_val = np.nan
                else:
                    if n_outputs == 1 and y_score.ndim == 2:
                        if len(np.unique(y_test[0])) == 2:
                            auc_val = roc_auc_score(y_test[0], y_score[:, 1])
            except Exception:
                auc_val = np.nan

    except Exception:
        auc_val = np.nan

    fold_roc[out_i].append(auc_val)
```

```
    # aggregate fold_metrics
    if fold_metrics:
        metrics_mean = {k: np.mean([fm[k] for fm in fold_metrics]) for k in fold_metrics[0].keys()}
    else:
        metrics_mean = {'accuracy': np.nan, 'precision': np.nan, 'recall': np.nan, 'f1': np.nan}

    results[name] = {'fold_metrics': metrics_mean}
    roc_results[name] = [np.nanmean(arr) if arr else np.nan for arr in fold_roc]

    print(f"  Mean metrics: {metrics_mean}")
    print(f"  Mean ROC-AUC per output: {roc_results[name]}")
```

```
# Build summary table and select top-3 by average of acc/prec/rec/f1
summary_rows = []

for name, res in results.items():
    mm = res['fold_metrics']  # <-- corrected key
    avg_metric = np.mean([mm['accuracy'], mm['precision'], mm['recall'], mm['f1']])
    summary_rows.append({
        'model': name,
        'accuracy': mm['accuracy'],
        'precision': mm['precision'],
        'recall': mm['recall'],
        'f1': mm['f1'],
        'avg': avg_metric
    })

summary_df = (
    pd.DataFrame(summary_rows)
    .sort_values('avg', ascending=False)
    .reset_index(drop=True)
)

print("\nSummary of evaluated classifiers (averaged across CV folds and outputs):")
print(summary_df)

top3 = summary_df.head(3)['model'].tolist()
print("\nTop-3 selected models for ensemble:", top3)
```

```
# Build Voting ensemble using the original base classifiers (wrapped similarly if multi-output)
estimators_for_voting = [(n, base_classifiers[n]) for n in top3]
```

```
# Determine whether soft voting possible: try to check predict_proba support by light fitting on small subset
def supports_predict_proba(clf):
    try:
        if hasattr(clf, "predict_proba"):
            return True
        else:
            return False
    except Exception:
        return False

all_support_proba = True
```

```
# quick local check on small sample
try:
    sample_X = X.iloc[:min(100, len(X))]
    sample_y = [arr[:len(sample_X)] for arr in y_arrays]
    sample_y_stack = np.vstack(sample_y).T if n_outputs > 1 else sample_y[0]
    for name, base in estimators_for_voting:
        c = clone(base)
        try:
            c.fit(preprocessor.fit_transform(sample_X), sample_y_stack)  # note: fit on transformed array if needed
            if not supports_predict_proba(c):
                all_support_proba = False
                break
        except Exception:
            all_support_proba = False
            break
except Exception:
    all_support_proba = False

voting_type = 'soft' if all_support_proba else 'hard'
print("Ensemble voting type:", voting_type)

voting = VotingClassifier(estimators=estimators_for_voting, voting=voting_type, n_jobs=-1)
if multi_output:
    voting_model = MultiOutputClassifier(voting, n_jobs=-1)
else:
    voting_model = voting
```

```
print("\nEvaluating Ensemble...")

ensemble_pipe = Pipeline([('pre', preprocessor), ('clf', voting_model)])
fold_metrics = []
fold_roc = [[] for _ in range(n_outputs)]

if use_risk_group:
    splits = cv.split(X, stratify_base)
else:
    splits = cv.split(X)

for fold_idx, (train_idx, test_idx) in enumerate(splits):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train = [arr[train_idx] for arr in y_arrays]
    y_test = [arr[test_idx] for arr in y_arrays]

    y_train_stack = np.vstack(y_train).T if n_outputs > 1 else y_train[0]

    m = clone(ensemble_pipe)
    try:
        m.fit(X_train, y_train_stack)
    except Exception as e:
        print(f"  Ensemble training failed on fold {fold_idx}: {e}")
        continue

    y_pred_stack = m.predict(X_test)

    if n_outputs == 1:
        y_pred_list = [y_pred_stack]
    else:
        y_pred_list = [y_pred_stack[:, i] for i in range(n_outputs)]

    _, avg_metrics = compute_metrics_per_output(y_test, y_pred_list)
    fold_metrics.append(avg_metrics)

    # ROC-AUC attempts (per output)
    for out_i in range(n_outputs):
        auc_val = np.nan
        try:
            clf_pipeline = m.named_steps['clf']

            if hasattr(clf_pipeline, 'estimators_'):
                est = clf_pipeline.estimators_[out_i]
                try:
                    y_score = est.predict_proba(m.named_steps['pre'].transform(X_test))
                    if len(np.unique(y_test[out_i])) == 2:
                        auc_val = roc_auc_score(y_test[out_i], y_score[:, 1])
                    else:
                        y_test_b = label_binarize(y_test[out_i], classes=np.unique(y_test[out_i]))
                        auc_val = roc_auc_score(y_test_b, y_score, average='macro', multi_class='ovr')
                except Exception:
                    try:
                        dec = est.decision_function(m.named_steps['pre'].transform(X_test))
                        if len(np.unique(y_test[out_i])) == 2:
                            auc_val = roc_auc_score(y_test[out_i], dec)
                        else:
                            y_test_b = label_binarize(y_test[out_i], classes=np.unique(y_test[out_i]))
                            auc_val = roc_auc_score(y_test_b, dec, average='macro', multi_class='ovr')
                    except Exception:
                        auc_val = np.nan
            else:
                # single-output ensemble
                try:
                    y_score = m.predict_proba(X_test)
                    if n_outputs == 1 and y_score.ndim == 2:
                        if len(np.unique(y_test[0])) == 2:
                            auc_val = roc_auc_score(y_test[0], y_score[:, 1])
                        else:
                            auc_val = np.nan
                except Exception:
                    auc_val = np.nan

        except Exception:
            auc_val = np.nan

        fold_roc[out_i].append(auc_val)

# Aggregate metrics
ensemble_mean_metrics = {k: np.mean([fm[k] for fm in fold_metrics])
                         for k in fold_metrics[0].keys()} if fold_metrics else \
                        {'accuracy': np.nan, 'precision': np.nan, 'recall': np.nan, 'f1': np.nan}

ensemble_roc_mean = [np.nanmean(arr) if arr else np.nan for arr in fold_roc]

print("Ensemble mean metrics:", ensemble_mean_metrics)
print("Ensemble ROC-AUC per output:", ensemble_roc_mean)
```

```python
# Comparison table: top3 individuals vs ensemble
compare_rows = []

for m in top3:
    mm = results[m]['fold_metrics']  # <-- fixed key
    compare_rows.append({
        'model': m,
        'accuracy': mm['accuracy'],
        'precision': mm['precision'],
        'recall': mm['recall'],
        'f1': mm['f1'],
        'roc_auc_avg': np.nanmean(roc_results[m])
    })

compare_rows.append({
    'model': 'Ensemble(' + "+".join(top3) + ')',
    'accuracy': ensemble_mean_metrics['accuracy'],
    'precision': ensemble_mean_metrics['precision'],
    'recall': ensemble_mean_metrics['recall'],
    'f1': ensemble_mean_metrics['f1'],
    'roc_auc_avg': np.nanmean(ensemble_roc_mean)
})

compare_df = pd.DataFrame(compare_rows).sort_values('f1', ascending=False).reset_index(drop=True)
print("\nTop-3 vs Ensemble comparison:")
print(compare_df)
```

```python
# Learning curve (training vs validation) for the ensemble on primary target (risk_group if available)
# If multi-output, we take the first target (primary) for learning curve visuals
primary_target_for_learning = y_targets[0][1]  # first target array

# Build a pipeline for single-output learning curve (primary target)
single_output_pipe = Pipeline([
    ('pre', preprocessor),
    ('clf', voting if n_outputs == 1 else clone(voting))
])

from sklearn.model_selection import learning_curve

try:
    train_sizes, train_scores, test_scores = learning_curve(
        single_output_pipe, X, primary_target_for_learning,
        cv=cv, scoring='f1_macro', n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 5)
    )

    train_scores_mean = np.mean(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)

    plt.figure(figsize=(8,6))
    plt.plot(train_sizes, train_scores_mean, 'o-', label='Train F1_macro')
    plt.plot(train_sizes, test_scores_mean, 'o-', label='Validation F1_macro')
    plt.xlabel("Training examples")
    plt.ylabel("F1 Macro")
    plt.title("Learning curve (Ensemble) - primary target")
    plt.legend()
    plt.grid(True)
    plt.show()

except Exception as e:
    print("Could not compute learning curve:", e)
```

```
# Averaged ROC curves: top3 individuals + ensemble
mean_fpr = np.linspace(0,1,100)
plt.figure(figsize=(8,6))
models_to_plot = top3 + ['Ensemble']
for name in models_to_plot:
    tprs = []
    aucs = []
    # run CV again folds
    if use_risk_group:
        splits = cv.split(X, stratify_base)
    else:
        splits = cv.split(X)
    for train_idx, test_idx in splits:
        X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
        y_train = [arr[train_idx] for arr in y_arrays]
        y_test = [arr[test_idx] for arr in y_arrays]
        y_train_stack = np.vstack(y_train).T if n_outputs>1 else y_train[0]
        if name == 'Ensemble':
            pipe = clone(ensemble_pipe)
        else:
            m = make_model(name, base_classifiers[name], multi_output)
            pipe = Pipeline([('pre', preprocessor), ('clf', m)])
        try:
            pipe.fit(X_train, y_train_stack)
            # predict_proba for the primary output only (index 0)
            if multi_output:
                # extract the estimator for first output and call predict_proba on preprocessed X_test
                clf_wrap = pipe.named_steps['clf']
                if hasattr(clf_wrap, 'estimators_'):
                    est = clf_wrap.estimators_[0]
                    try:
                        X_test_tr = pipe.named_steps['pre'].transform(X_test)
                        y_score = est.predict_proba(X_test_tr)
                        if y_score.ndim == 2:
                            if len(np.unique(y_test[0]))==2:
                                fpr, tpr, _ = roc_curve(y_test[0], y_score[:,1])
                                tprs.append(np.interp(mean_fpr, fpr, tpr))
                                aucs.append(auc(fpr,tpr))
                            else:
                                # multiclass: compute per-class, then average
                                pass
                    except Exception:
                        continue
                else:
                    # single-output pipeline
                    try:
                        y_score = pipe.predict_proba(X_test)
                        if y_score.ndim==2 and len(np.unique(y_test[0]))==2:
                            fpr, tpr, _ = roc_curve(y_test[0], y_score[:,1])
                            tprs.append(np.interp(mean_fpr, fpr, tpr))
                            aucs.append(auc(fpr,tpr))
                    except Exception:
                        continue
            else:
                try:
                    y_score = pipe.predict_proba(X_test)
                    if y_score.ndim==2 and len(np.unique(y_test))==2:
                        fpr, tpr, _ = roc_curve(y_test, y_score[:,1])
                        tprs.append(np.interp(mean_fpr, fpr, tpr))
                        aucs.append(auc(fpr,tpr))
                except Exception:
                    continue
        except Exception:
            continue
    if tprs:
        mean_tpr = np.mean(tprs, axis=0)
        mean_auc = np.mean(aucs)
        plt.plot(mean_fpr, mean_tpr, label=f"{name} (AUC {mean_auc:.3f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Averaged ROC curves (primary output across CV folds)")
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```