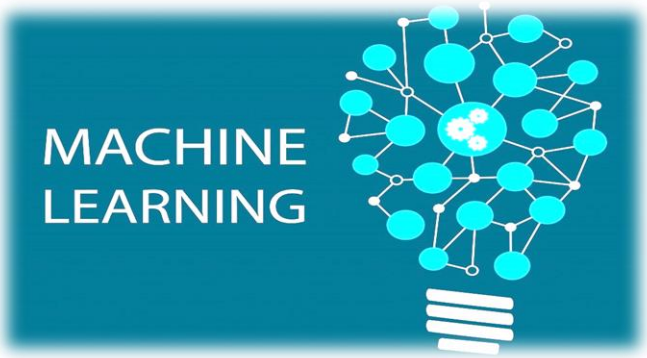
LEBANESE INTERNATIONAL UNIVERSITY

الجامعة اللبنانية الدولية

Project in Machine Learning-CSCI441



-Prepared by: Hiba Shaito-Mahmoud Zahra
-Presented to: Dr. Ali Ballout
-Project submitted in the context of the course Csci441
- "Machine Learning" –
Lebanese International University

Contents:

Introduction:

Student depression is an increasingly prevalent issue, with factors such as academic pressure, financial stress, and lifestyle challenges significantly affecting the mental health of young individuals. These pressures can lead to a decline in academic performance, reduced study satisfaction, and, in severe cases, suicidal thoughts. Addressing this issue requires a proactive approach to identify and support students at risk.

This project leverages data analysis and machine learning to predict depression levels among students based on key variables like academic pressure, CGPA, financial stress, family history, and other lifestyle factors. The dataset was carefully preprocessed, involving tasks such as normalizing educational degrees into numeric values through degree mapping and handling categorical variables. A classification model was developed to analyze these factors and predict depression risks, with its performance assessed using metrics like accuracy, confusion matrices, and classification reports.

By identifying patterns and highlighting at-risk individuals, this project not only provides a technical solution but also emphasizes the importance of early detection and intervention in combating student depression. It serves as a step toward fostering better mental health support systems in educational environments, enabling more informed decisions and effective policy-making.

Problem:

To predict the likelihood of depression among students based on factors such as academic pressure, CGPA, financial stress, and family history, enabling early identification and intervention.

Predicting Student Depression Using Machine Learning: A Data-Driven Approach

In real life, predicting student depression using machine learning involves collecting data from surveys, student records, or mental health assessments, covering factors like academic performance, socio-economic status, and mental well-being. Platforms like Kaggle provide relevant datasets to train models.

The process includes:

1. **Data Collection:** Gathering relevant student data.
2. **Data Preprocessing**: Cleaning and encoding data.
3. **Model Training:** Using machine learning algorithms to create predictive models.
4. **Model Evaluation:** Assessing model performance.
5. **Deployment:** Using the model for real-time predictions, enabling early interventions.

This approach can help educational institutions identify at-risk students and provide targeted support.

Dataset Sourced from Kaggle for Analyzing Student Depression

The data used in this project was obtained from Kaggle, a popular platform known for providing access to diverse datasets for machine learning and data analysis projects. Kaggle offers a wealth of publicly available datasets across various domains, making it an ideal resource for sourcing data to explore and develop predictive models. The dataset used here specifically focuses on student depression, allowing us to analyze various factors contributing to mental health issues among students.

Dataset Overview

The dataset used in this project was sourced from Kaggle and contains 27,901 rows and 18 features. Each row represents an individual data point, while the 18 features capture various aspects related to student depression, including factors such as academic pressure, study satisfaction, financial stress, and family history. This dataset provides a comprehensive view of the factors that may contribute to mental health issues among students, enabling the development of predictive models to analyze and understand student depression.

During data preprocessing, we encountered columns with a **standard deviation of 0**. Standard deviation is a measure of the amount of variation or dispersion in a set of values. A standard deviation of 0 indicates that all the values in that column are identical, meaning there is no variability or diversity in the data for that feature. Since such columns do not provide useful information for the model (as they do not contribute to distinguishing between different data points), these features were dropped from the dataset. This step helped in refining the dataset for better model performance and more meaningful analysis.

Tools Used

In this project, we utilized two powerful tools to perform data analysis and build the predictive model: Google Colab and VS Code.

- **Google Colab**: Google Colab is a cloud-based platform that allows you to run Python code in an interactive environment. It provides a convenient way to write and execute Python code, especially for data science projects, without needing to install any software locally. Colab also provides free access to GPUs and TPUs, making it ideal for resource-intensive tasks. We used Google Colab to load the dataset, perform initial data analysis, and run machine learning models.

- **VS Code (Visual Studio Code)**: Visual Studio Code is a lightweight and powerful code editor that provides an excellent environment for writing Python scripts. It has various extensions, such as Python and Jupyter, which make coding and debugging efficient. While Colab was used for heavy processing, we used VS Code for writing scripts, organizing code, and managing project files.

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a crucial step in the data science process that involves analyzing and visualizing data to understand its structure, patterns, and relationships. The goal of EDA is to gain insights into the dataset before applying any complex models. It helps identify potential issues such as missing values, outliers, or incorrect data types, and provides a better understanding of the data's distribution.

During the EDA process, we typically perform tasks such as:

- → **Data Importing**: Loading the dataset into a working environment for analysis.
- → **Data Cleaning**: Identifying and handling missing or incorrect data.
- → **Statistical Summaries**: Reviewing measures like mean, median, standard deviation, and correlations.
- → **Data Visualization**: Plotting graphs like histograms, box plots, and scatter plots to visualize relationships between features.

By performing EDA, we ensure that the data is ready for modeling and further analysis

Importing Data with Pandas

```
import pandas as pd
df=pd.read_csv('Student Depression Dataset.csv')
```

- Before starting the analysis, it's important to first load the data into an environment where we can manipulate and analyze it. For this project, we used the **pandas** library to import and work with the dataset.
- The **pandas** library is a powerful and widely-used tool in Python for data manipulation and analysis. It provides flexible data structures like DataFrame and Series, making it easy to handle and analyze structured data. Pandas simplifies common tasks such as cleaning, transforming, and analyzing datasets efficiently.
- Functionality and Data Import
- One of the core features of pandas is its ability to handle a variety of file formats, such as **CSV**, **Excel**, and **SQL databases**. The most commonly used function for importing data is read_csv(), which reads data from CSV files and stores it in a DataFrame. This tabular structure, with rows and columns, makes it easy to work with and perform operations on the data.

Data Cleaning and Preparation

After importing the dataset, we performed essential data cleaning steps. We used head() to view the first few rows and checked the dataset's shape for accuracy. To identify missing data, we used isnull().sum() and removed any rows with null values. We also checked for duplicates using duplicated() and dropped unnecessary columns to ensure a clean dataset, making it ready for further analysis.

Data Encoding

After cleaning the dataset, we applied different encoding techniques to convert categorical features into numerical values for machine learning. For features with ordinal categories, such as "Degree," we used a mapping approach to assign numerical values. For binary features, such as "Suicidal Thoughts," we used label encoding, assigning 0 or 1 based on the categories. This encoding process ensures that the data is in a format suitable for model training and analysis.

Detailed Explanation of Data Encoding Techniques

1. Mapping

Mapping is a method of encoding categorical data by manually assigning numerical values to categories based on their meaning or hierarchy. This is particularly useful for ordinal categories, where the values have a meaningful order. For example, in a "Degree" feature, education levels can be mapped as:

- e.g.: class 12 → 1 (High School)
- e.g.: b.tech → 2 (Undergraduate)
- e.g.: m.tech → 3 (Postgraduate)
- e.g.: phd → 4 (Doctorate)

Mapping helps the model understand the inherent order or relationship between the categories. It is implemented in Python using a dictionary and the .map() method:

2. Ordinal Categories

These are categorical values with a natural order or ranking. For instance, education levels are ordinal because they represent increasing levels of a concept. Encoding ordinal categories numerically preserves this order, which is beneficial for certain machine learning algorithms.

3. Label Encoding

Label encoding is a simpler technique used to convert categorical variables into numbers, typically for binary or unordered categories. Each unique category is assigned a number. For example:

"Yes" → 1

"No" → 0

This is commonly used for binary features, such as "Suicidal Thoughts."

These encoding techniques ensure that categorical data is properly represented in numerical form, making it suitable for machine learning models.

Importance of Encoding in Machine Learning Models

Encoding categorical data into numerical formats is essential for building machine learning models, as most algorithms require numerical inputs to process the data. By using techniques like mapping and label encoding, we effectively convert text-based categories into a structured numerical format that models can interpret.

This step is particularly beneficial because it:

Retains Information: For ordinal features, mapping ensures that the numerical encoding reflects the natural order of categories, helping the model understand the relationships between them.

Simplifies Processing: Label encoding makes binary or nominal features easier to process without adding unnecessary complexity.

Improves Model Performance: Proper encoding reduces ambiguity, ensuring that the model can use all relevant data effectively to improve predictions.

In our project, encoding features like education levels and suicidal thoughts ensures that the data is optimized for analysis, enabling the model to capture meaningful patterns and relationships, ultimately enhancing its predictive capabilities.

Using Correlation to Identify Key Features

After encoding, the dataset consists entirely of numerical values, enabling the use of correlation analysis to study relationships between features. Correlation measures the strength and direction of the linear relationship between two variables.

This step is crucial because:

Feature Selection: It helps identify the most important features that have a strong relationship with the target variable (depression).

Redundancy Reduction: Highly correlated independent variables can be identified, allowing us to remove redundant features that do not add significant value to the model.

Highly correlated redundant variables are features that provide similar information, leading to overfitting and multicollinearity, and one can be removed to simplify the model.

Insights: Correlation analysis provides insights into how various factors interact with each other and their influence on the target variable.

By analyzing the correlation matrix, we can focus on features with the highest relevance to depression while removing or deprioritizing features that contribute little to the predictive performance of the model. This step ensures a cleaner, more efficient dataset for model training.

Calculating Correlation and Visualizing with Heatmaps

Correlation is calculated using statistical methods that measure the linear relationship between two variables. In Python, the corr() function from pandas is commonly used to compute correlation values for a dataset. The result is a correlation matrix, which provides pairwise correlation values for all numerical features. These values range from -1 to 1, where:

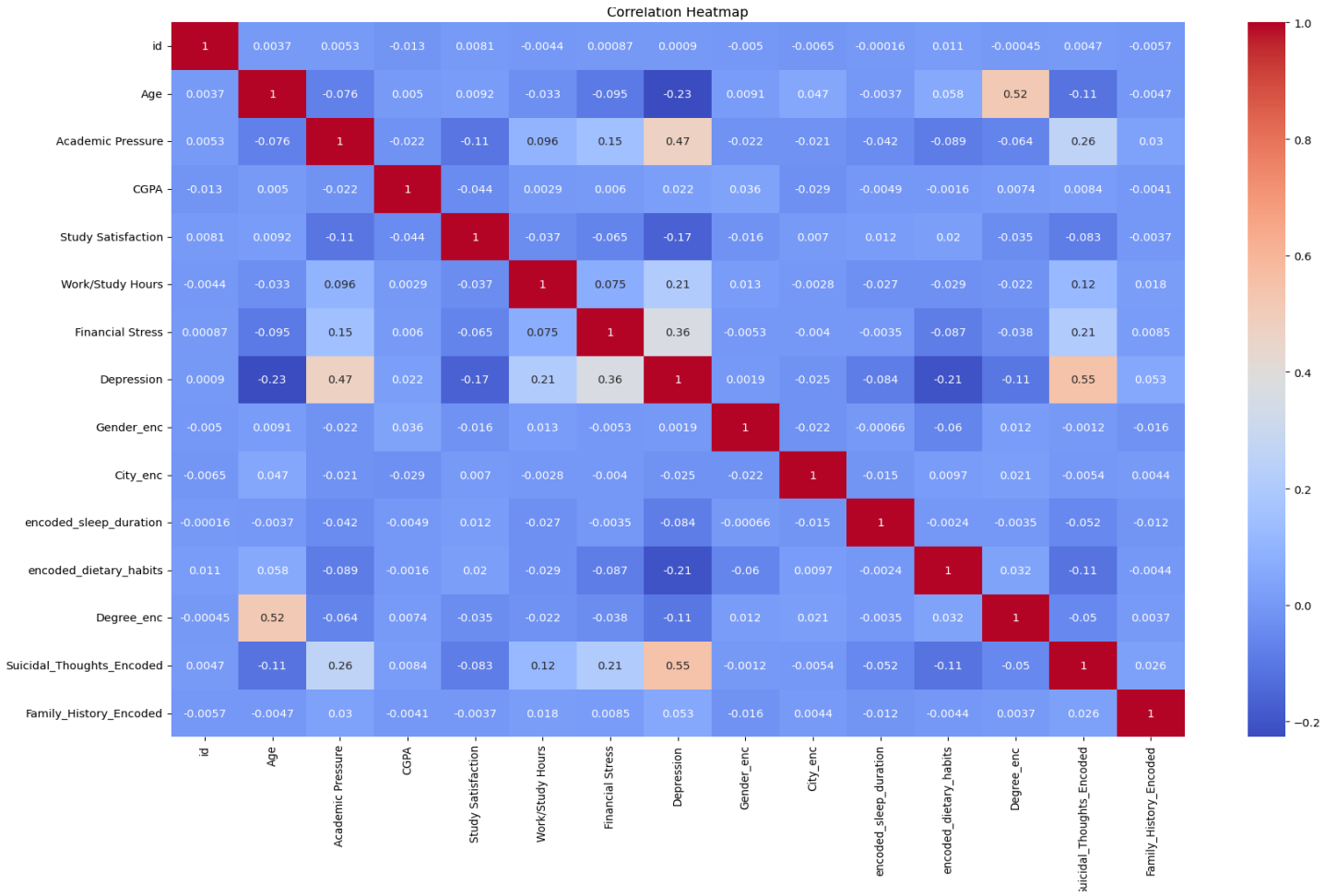
1 indicates a perfect positive correlation.

-1 indicates a perfect negative correlation.

0 indicates no correlation.

To visualize this matrix effectively, we use **heatmaps**, a graphical representation of data where correlation values are represented by colors. The seaborn library, a powerful Python library for data visualization, provides the heatmap() function, which allows us to easily plot correlation matrices. Heatmaps make it easier to identify patterns and relationships at a glance.

For example, features with strong correlations to the target variable can be quickly highlighted, enabling us to select the most relevant predictors for our model. Seaborn also allows customization of heatmaps, such as annotating values or adjusting color palettes, making the visualizations both informative and aesthetically pleasing.



The encoding results reveal important relationships between various factors and depression:

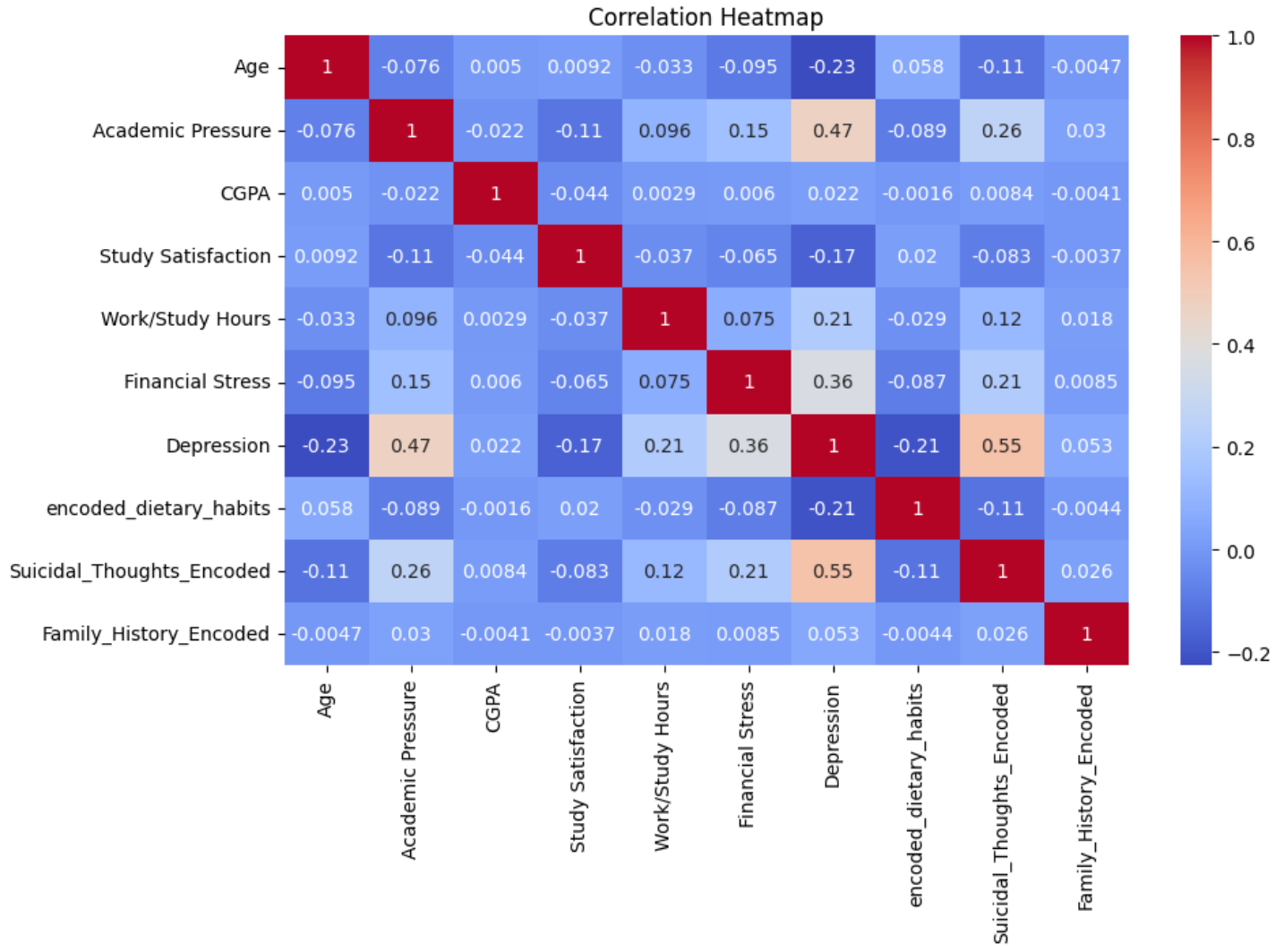
- **Suicidal Thoughts (0.546):** This strong positive correlation suggests that individuals with higher depression levels are more likely to experience suicidal thoughts.
- **Financial Stress (0.363)** and **Academic Pressure (0.474):** Both show moderate positive correlations, highlighting that financial and academic stress significantly contribute to depression.
- **Work/Study Hours (0.208):** This mild positive correlation indicates that a heavier workload may have a slight impact on depression.
- **Age (-0.226)** and encoded dietary habits **(-0.206):** Both exhibit negative correlations, implying that younger individuals and those with poor dietary habits may experience higher levels of depression.
- **CGPA (-0.274):** This weak-to-moderate negative correlation suggests that as CGPA increases, the likelihood of depression slightly decreases.

On the other hand, variables like id, Gender_enc, City_enc, encoded_sleep_duration, and Degree_enc show weak or no significant correlation with depression. These features may not contribute significantly to predicting depression in this context.

Correlation Matrix After Dropping Low Correlation Features:

After removing columns with weak or no significant correlation with depression, the correlation matrix now highlights only the most impactful features. By dropping these less relevant columns, we improve the focus on the factors that truly contribute to understanding the relationship with depression. This process helps in refining the model and making it more efficient by reducing the noise introduced by irrelevant variables.

Here's how the correlation looks after dropping unnecessary columns:



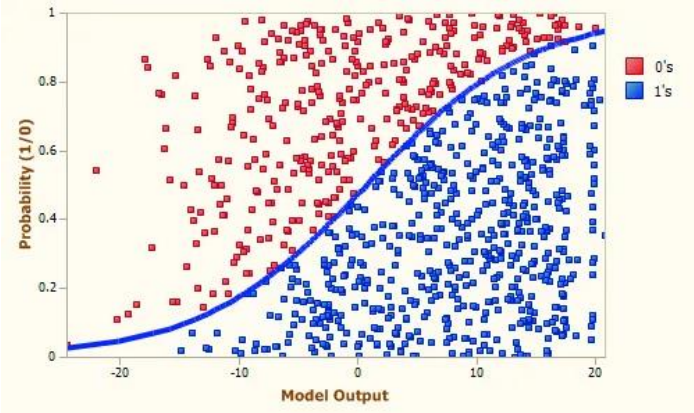
Data Splitting and Libraries:

After preparing and cleaning the dataset, the next step is to separate the target variable (in this case, depression) from the feature variables. The target variable represents what we are trying to predict, while the features are the various factors that might influence the outcome.

To perform this split, the **pandas** library is used to separate the target column, while the **scikit-learn** library provides the tools for splitting the data into training and testing sets. This is done using the train_test_split() function, which randomly divides the data into training and testing datasets, ensuring that the model can be trained on one portion and evaluated on another.

By default, the training set typically consists of 70-80% of the data, with the remaining 20-30% used for testing. This split ensures that the model is validated on unseen data, which is crucial for assessing its generalization performance.

Logistic Regression



Logistic Regression is a classification algorithm used when the target variable is categorical, typically binary (two classes). It is a linear model that predicts the probability of an instance belonging to one of the classes. In logistic regression, instead of predicting continuous values (like in linear regression), we predict probabilities that lie between 0 and 1. The output is then converted to a class label (0 or 1) based on a threshold, usually 0.5.

For example, in our case, the model predicts whether a student is likely to experience depression (1) or not (0) based on various features.

Logistic Regression Model:

The **Logistic Regression model** works by fitting a logistic curve (also known as the sigmoid function) to the data. This curve maps the predicted values to probabilities between 0 and 1.

Why Logistic Regression?

Logistic Regression was chosen for this project for several reasons:

- **Binary Classification Problem**: The problem we're solving—predicting whether a student is depressed or not—is a binary classification task. Logistic regression is designed specifically for this type of problem.
- **Interpretability**: Logistic regression provides coefficients that indicate the strength and direction of the relationship between each feature and the target variable. This helps in understanding how different factors (e.g., academic pressure, financial stress) contribute to the likelihood of depression.
- **Simplicity and Efficiency**: Logistic regression is relatively simple to implement and computationally efficient, making it a good starting point for binary classification tasks.
- **Probability Output**: Logistic regression not only gives a prediction but also provides the probability of the target being 1 (depression in this case), which can be useful for further analysis or decision-making.

Library Used: scikit-learn

For implementing Logistic Regression, we used the **scikit-learn** library, which is one of the most widely used libraries for machine learning in Python. It provides a simple and efficient implementation of various machine learning algorithms, including logistic regression.

To use logistic regression in scikit-learn, you need to import the LogisticRegression class from the sklearn.linear_model module. After training the model, you can use methods like predict() to make predictions and predict_proba() to obtain the predicted probabilities.

Logistic Model Evaluation and Accuracy Testing

After training the logistic regression model, the next crucial step is to evaluate its performance on unseen data. This is done by predicting the target variable for the test set and comparing the predicted values with the actual values. The model's accuracy is a common metric used to assess its performance. Accuracy represents the percentage of correctly predicted instances out of the total instances in the test set.

To measure the model's performance, various metrics such as accuracy, precision, recall, and F1-score can be used, depending on the problem's requirements. In this case, accuracy is a good starting point to understand how well the logistic regression model is performing in predicting student depression based on the provided features.

Key Metrics:

1.Accuracy (0.85 or 85.1%):

This is a strong result, meaning the model correctly predicts whether someone is experiencing depression 85% of the time. However, accuracy alone isn't always the best metric, especially if the dataset is imbalanced (i.e., more samples of one class than the other).

2.Confusion Matrix:

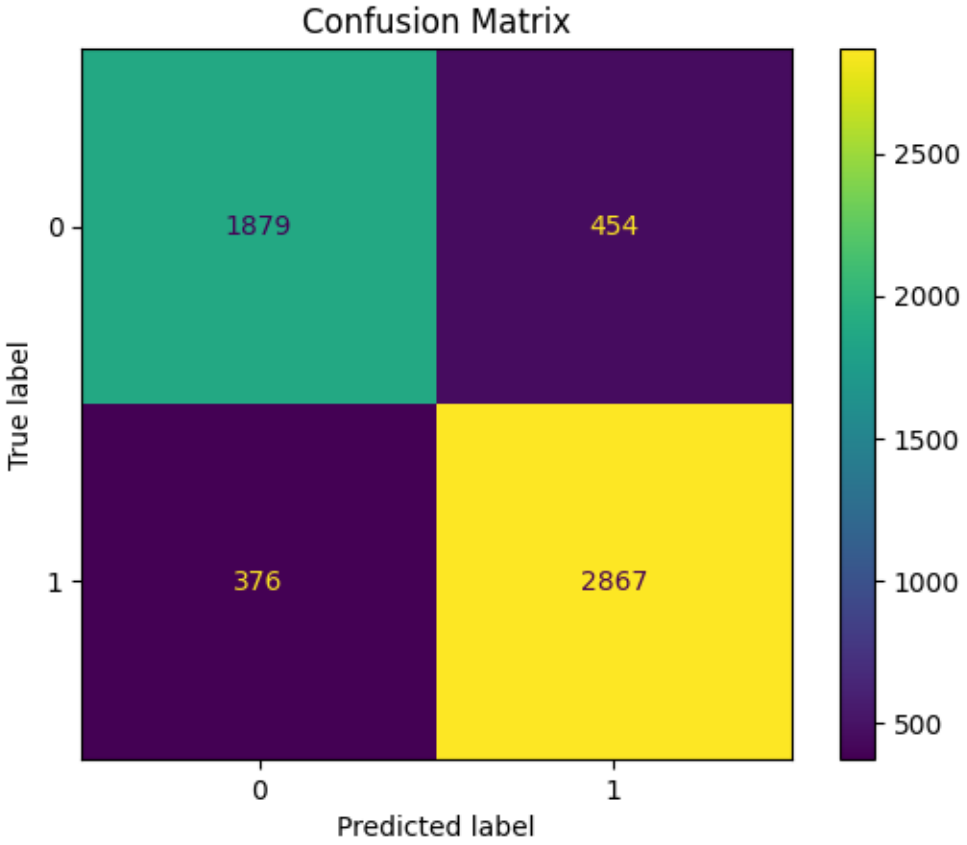
True Negatives (1879): Cases where no depression was predicted correctly.

False Positives (454): Cases where depression was predicted, but there was none.

True Positives (2867): Cases where depression was predicted and it was correct.

False Negatives (376): Cases where no depression was predicted, but the person actually had depression.

The relatively low number of false negatives is a good sign because missing depression cases is often more critical than misclassifying non-depression cases.



We plotted the confusion matrix using the model's predictions and true labels to visually assess its performance in distinguishing between depression and non-depression cases. We used ConfusionMatrixDisplay from the sklearn.metrics library.

3.Precision, Recall, F1-Score:

Precision:

→Class 0: 83% (When the model predicts no depression, it's correct 83% of the time.)

→Class 1: 86% (When the model predicts depression, it's correct 86% of the time.)

Recall:

→Class 0: 81% (The model captures 81% of actual non-depression cases.)

→Class 1: 88% (The model captures 88% of actual depression cases, which is excellent.)

F1-Score:

→Class 0: 82%

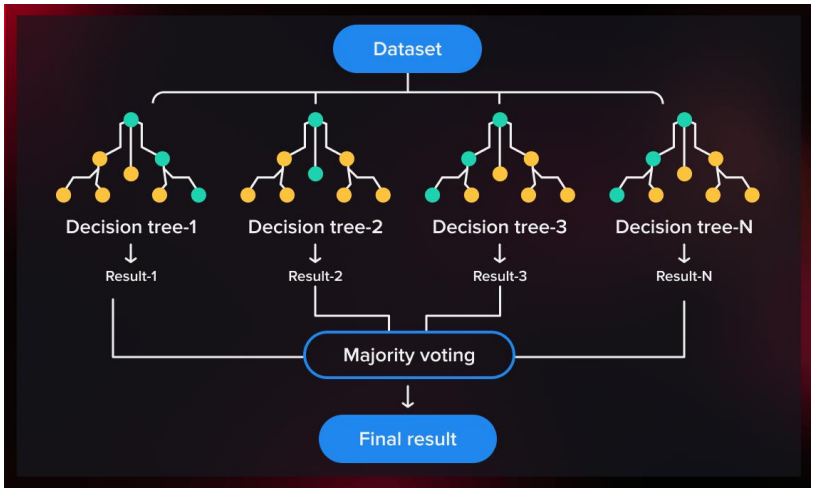
→Class 1: 87%

F1 combines precision and recall, so a high F1-score (especially for depression cases) is desirable.

4.Class Imbalance:

Before training the model, I calculated the class distribution in the dataset, where depression (Class 1) makes up 58.56% of the data and non-depression (Class 0) accounts for 41.44%. Given this imbalance, I took it into account during model training. Despite the imbalance, the model performs well, especially in detecting depression cases, with a high recall of 88%. This means it successfully identifies most depression cases while keeping false negatives relatively low (376). The precision, recall, and F1-scores reflect the model's ability to manage the class imbalance effectively.

Random Forest:



Random Forest is an ensemble learning method that builds multiple decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. Each tree is trained on a random subset of the data with random feature selections, which reduces overfitting and increases generalization.

How it Works?

1. **Random Subsets**: The model creates multiple decision trees, each trained on a random subset of the training data (with replacement) — this is known as **bootstrapping**.
2. **Random Features**: When splitting a node, a random subset of features is considered for each tree, further reducing overfitting.
3. **Aggregation**: Each tree in the forest makes its own prediction, and the final output is determined by aggregating the predictions (majority vote for classification).
4. **Example Use Case**: Random Forest can be used for predicting customer churn (whether a customer will leave or stay) based on features like age, usage patterns, and account type.

Advantages:

- Handles large datasets with high dimensionality well.
- Robust to overfitting, especially with large datasets.
- Can capture non-linear relationships between features.
- Provides feature importance, helping to understand the influence of different features.

Disadvantages:

- Can be computationally expensive, especially for large datasets.
- Less interpretable than simpler models like logistic regression.

Why Random Forest?

Random Forest was chosen for this project for several reasons:

Handling Complex Relationships: Random Forest is an ensemble learning method that builds multiple decision trees to capture complex, non-linear relationships between features. Since student depression can be influenced by multiple interacting factors (e.g., academic performance, social environment, and personal well-being), Random Forest is well-suited for this task.

Robustness to Overfitting: Random Forest reduces the risk of overfitting by averaging the predictions of multiple decision trees. This is particularly useful when working with a large and complex dataset, as it helps generalize better to unseen data, improving model performance.

Feature Importance: Random Forest provides feature importance scores, which help identify which factors (such as stress levels, social support, or study habits) are most influential in predicting depression. This can provide valuable insights into the key drivers of depression and guide further analysis or interventions.

Handling High Dimensionality: Random Forest can handle high-dimensional datasets effectively, making it a good choice when there are many features that may influence the outcome. It performs well even when there is a mix of categorical and continuous variables.

Flexibility: Unlike logistic regression, which assumes a linear relationship between features and the target variable, Random Forest can capture non-linear relationships without the need for explicit feature engineering or transformation.

Random Forest Model Evaluation and Accuracy Testing

Key Metrics Analysis:

1.Accuracy (83.8%): The model achieves an accuracy of 83.8%, meaning it correctly classifies 83.8% of the test samples. While this is a solid result, relying solely on accuracy can be misleading if the dataset is imbalanced. Further analysis of precision, recall, and F1-score is essential for deeper insights.

2.Confusion Matrix:

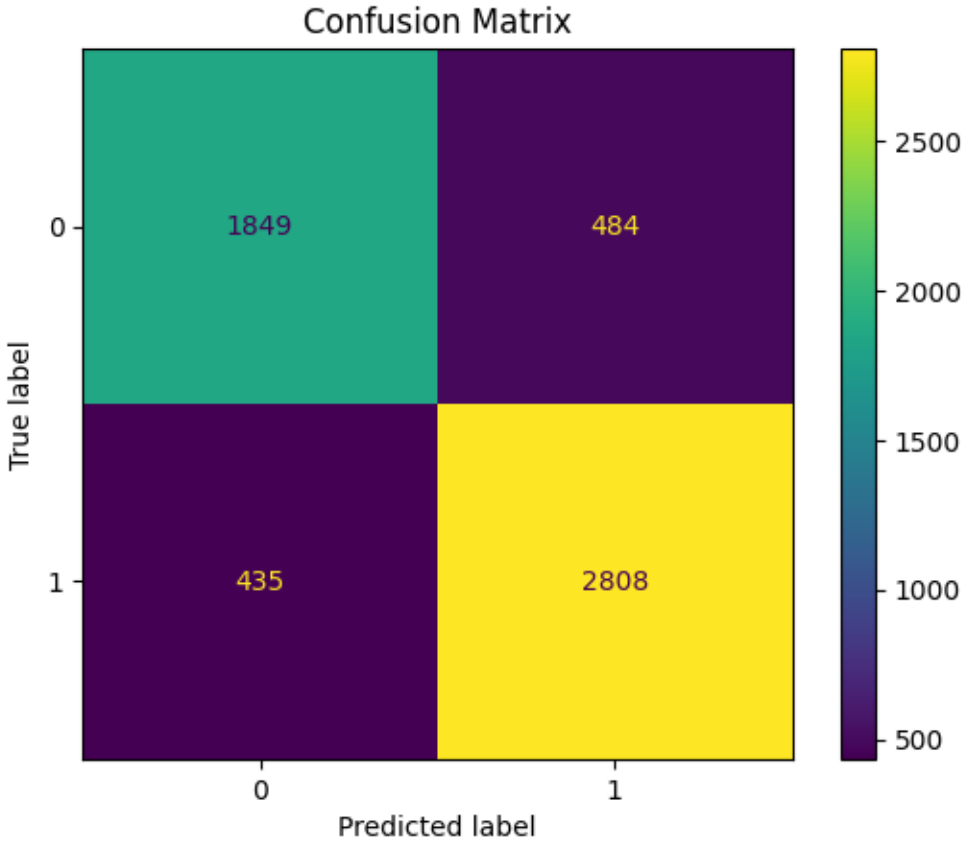
True Negatives (1866): The model correctly predicted 1866 cases where the actual class was 0 (non-target).

False Positives (467): The model incorrectly predicted 467 cases as class 1 (target) when they were actually class 0.

True Positives (2805): The model correctly identified 2805 cases as class 1 (target).

False Negatives (438): The model missed 438 cases, predicting them as class 0 when they were actually class 1.

The relatively low number of false negatives is promising, as correctly identifying positive cases (class 1) is often critical depending on the problem.



3.Precision, Recall, and F1-Score:

Class 0 (Non-Target):

Precision: 81% - When the model predicts class 0, it is correct 81% of the time. Recall: 80% - The model captures 80% of actual class 0 cases. F1-Score: 80% - A balance of precision and recall for class 0.

Class 1 (Target):

Precision: 86% - When the model predicts class 1, it is correct 86% of the time. Recall: 86% *- The model captures 86% of actual class 1 cases. *F1-Score: 86% - High F1 indicates a good balance of precision and recall for class 1. Overall, the model demonstrates better performance for class 1, which could be important if identifying these cases is a priority.

4. Class Imbalance: The dataset has 3243 samples of class 1 and 2333 samples of class 0. Despite the imbalance, the model handles both classes well, maintaining balanced precision and recall metrics.

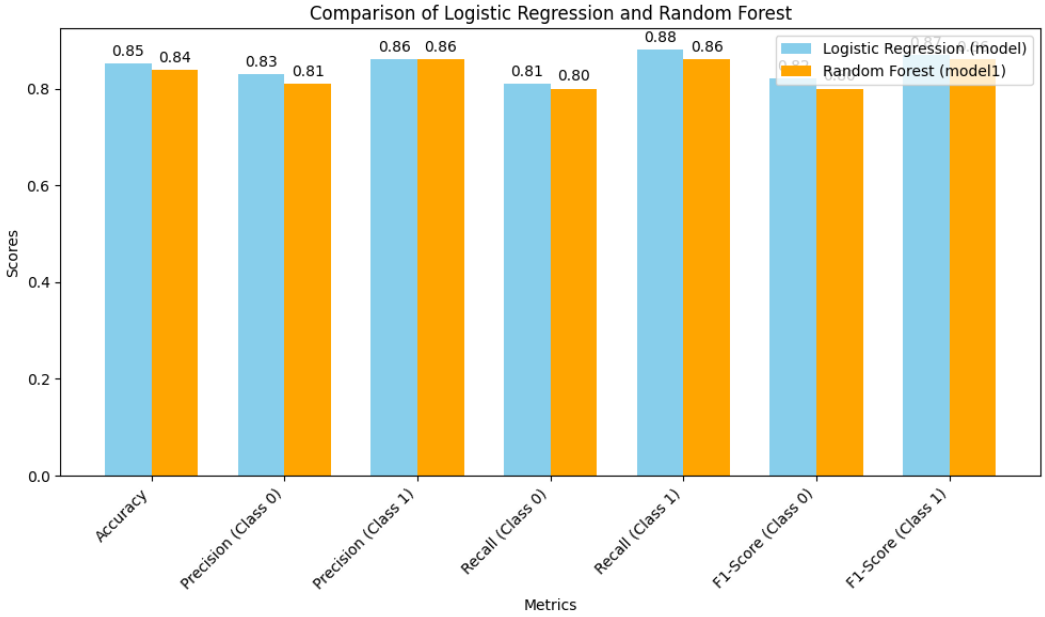
Comparison :

Accuracy: **Logistic Regression** achieved slightly higher accuracy (85.1%) compared to **Random Forest** (83.8%), indicating it is marginally better at making overall correct predictions.

Precision: Both models demonstrated similar precision for predicting **Class 1** (0.86), suggesting they are equally good at correctly identifying positive cases. **Logistic Regression** had a slightly better precision for **Class 0** (0.83 vs. 0.81), meaning it made fewer false positive predictions for negative cases.

Recall: **Logistic Regression** outperformed **Random Forest** in recall for **Class 1** (0.88 vs. 0.86), capturing more true positive cases. Both models showed similar recall for **Class 0** (0.80 vs. 0.81), indicating similar performance in capturing true negatives.

F1-Score: **Logistic Regression** consistently scored higher F1-scores for both classes, particularly for **Class 1** (0.87 vs. 0.86). This suggests that Logistic Regression strikes a slightly better balance between precision and recall.



Conclusion:

Logistic Regression marginally outperforms **Random Forest** in most key metrics, particularly in accuracy and recall for critical cases (Class 1). This makes Logistic Regression the better choice if minimizing the misclassification of positive cases is a priority. However, both models exhibit strong performance and can be considered depending on the specific needs of the project, such as interpretability or computational efficiency. These results were visualized using **matplotlib.pyplot** to provide clear comparisons of performance across different metrics.

What We Learned:

Through this project, we learned the importance of handling imbalanced data, ensuring data accuracy, and addressing data-related issues such as missing values and correlations to improve model performance. Additionally, we explored the value of evaluating models beyond accuracy, using metrics like precision, recall, and F1-scores to gain a deeper understanding of their effectiveness.

This project highlights the potential of machine learning in tackling real-world mental health challenges and emphasizes the critical role of proper data preprocessing in achieving reliable outcomes.