MULTIMEDIA UNIVERSITY

CDS6314 DATA MINING

ASSIGNMENT [20%]

TERM : 2430

GROUP NUMBER : G13

Group Members' Names and ID :

No.	Student ID	Name	Contribution (%)
1	1221301140	Nur Farah Nabila Binti Ramzairi	25
2	1231300941	Veenah A/P Ganesh	25
3	1211101925	Nur Alya Nabilah Binti Md.Naser	25
4	1211101533	Muhammad Aniq Fahmi Bin Azhar	25

Table of Contents

1.0 Introduction

1.1 Assignment Background

A real-world dataset from a higher education institution that was produced as part of a project funded by Portugal's SATDAP program serves as the basis for this assignment. The dataset contains data on academic routes, demographics, and socioeconomic characteristics that were available at the time of student registration. Predicting student outcomes, divided into three categories—dropout, enrolled, and graduate—is the main objective of this data. Addressing a severe class imbalance—where one category far dominates the others—is the challenge at hand.

The dataset shows an attempt to address the enduring problem of failure and academic dropout in higher education. The study intends to identify students who are at danger of dropping out early in their academic careers by utilizing machine learning techniques. This makes it possible for educational institutions to put in place prompt and focused support plans, which eventually lower dropout rates and improve academic achievement in general. Through the use of data mining tools in a relevant context, this assignment gives students the chance to interact with a socially significant issue.

1.2 Assignment Motivations

The student dropout rate in the country has improved significantly between 2017 and 2023, with primary school dropout rates decreasing from 0.29% to 0.06% and secondary school rates from 1.36% to 0.83% during the same period (New Straits Times, March 27, 2024). However, dropout rates in higher education remain a persistent challenge. Identifying students at risk early in their academic journey can enable institutions to implement timely interventions to mitigate these risks.

This assignment builds on this premise, utilizing advanced machine learning techniques to predict student outcomes in higher education. Recent studies underscore the potential of machine learning in education. Jie Xu et al. (2017) demonstrated a progressive prediction algorithm for tracking student performance over time, emphasizing the need to account for students' evolving progress and heterogeneous data structures. Similarly, Juan L. Rastrollo-Guerrero et al. (2020) highlighted that supervised learning techniques, such as Decision Trees, Random Forest, and Support Vector Machines, are particularly effective in predicting dropout rates and improving curriculum design. Ali Salah Hashim et al. (2020) also confirmed that supervised machine learning algorithms can predict final examination outcomes with high accuracy, emphasizing the utility of models like Logistic Regression and Decision Trees in higher education.

These findings align with the assignment's objective to reduce academic dropout and improve success rates by leveraging data mining and machine learning. By addressing challenges like class imbalance and multi-class classification, the assignment not only enhances technical expertise but also contributes to a socially significant cause. It emphasizes the transformative potential of technology in education, fostering an ethical and impactful approach to real-world problem-solving.

2.0 Exploratory Questions

2.1 Questions to Explore Patterns in the Dataset

Scope	Question
Demographic Influence	How do demographic factors (e.g., age, gender, or region) influence the likelihood of a student dropping out, staying enrolled, or graduating?
Socio-Economic Impact	To what extent do socio-economic factors, such as family income or parental education levels, affect student outcomes?
Academic Pathway & Dropout Risk	Are there specific undergraduate degrees or academic pathways that exhibit higher dropout rates compared to others?
Enrollment Trends & Success Rate	What patterns can be observed between the students' academic enrollment characteristics (e.g., full-time vs. part-time) and their eventual outcomes?
Early Warning Indication	What are the most significant early indicators (e.g., GPA in the first semester, attendance rates) that predict whether a student is likely to drop out?
Performance Progression	How do academic performances (e.g., grades in core subjects) progress over time for students who graduate compared to those who drop out?
Imbalance Across Classes	How does the class imbalance in the dataset (e.g., significantly more enrolled students) affect model performance and predictive accuracy?
Intervention Opportunities	What specific student subgroups (e.g., by demographics or academic performance) would benefit the most from targeted interventions?

2.2 Outcomes and Interestingness

Scope	Question
Demographic Influence	Understanding how demographics shape outcomes helps institutions design tailored interventions. For example, identifying that students from a particular region have higher dropout rates may indicate regional disparities in educational support or access.
Socio-Economic Impact	Insights into socio-economic influences can justify the allocation of scholarships or financial aid to high-risk groups, directly improving student retention.
Academic Pathway & Dropout Risk	Identifying pathways with higher dropout rates can help redesign curriculums or provide additional resources to specific programs, addressing systemic issues.
Enrollment Trends & Success Rate	This helps reveal the effect of enrollment types (e.g., full-time vs. part-time) on outcomes, allowing institutions to support non-traditional students more effectively.
Early Warning Indication	Highlighting early predictors of dropout is crucial for developing proactive intervention strategies, such as counseling or tutoring, at critical stages.
Performance Progression	Comparing performance trajectories can help differentiate students who struggle temporarily from those at risk of permanent failure, enabling precise support.
Imbalance Across Classes	Addressing class imbalance ensures that minority groups (e.g., graduates or dropouts) are accurately predicted, improving the model's real-world applicability.
Intervention Opportunities	By identifying specific at-risk subgroups, institutions can optimize resources for maximum impact, enhancing overall educational outcomes.

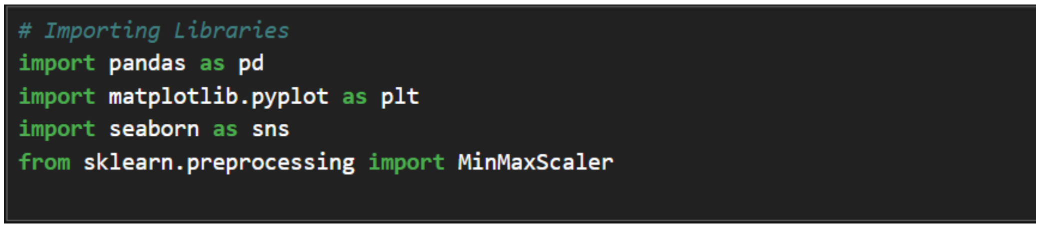
3.0 Data Exploration

3.1 Dataset Description

The dataset contains detailed information about students enrolled in various undergraduate programs. It includes features categorized into three main areas: demographics, socio-economic factors, and academic data. Demographic attributes include details such as age and gender, providing insights into the diversity of the student population. Socio-economic factors cover variables like family income and parental education, which may influence academic outcomes. Academic data includes the students' enrollment programs, their GPA during the first and second semesters, and their final classification into one of three outcomes: dropout, enrolled, or graduate.

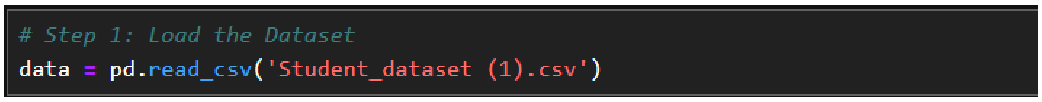
The dataset is structured for a multi-class classification task and presents a significant class imbalance, with a majority of students likely classified as "enrolled." This imbalance poses challenges for machine learning models, requiring specialized techniques to ensure accurate predictions for minority classes. The data offers a comprehensive view of factors influencing student success and retention, making it suitable for exploring patterns, building predictive models, and developing intervention strategies.

3.2 Initial Data Exploration



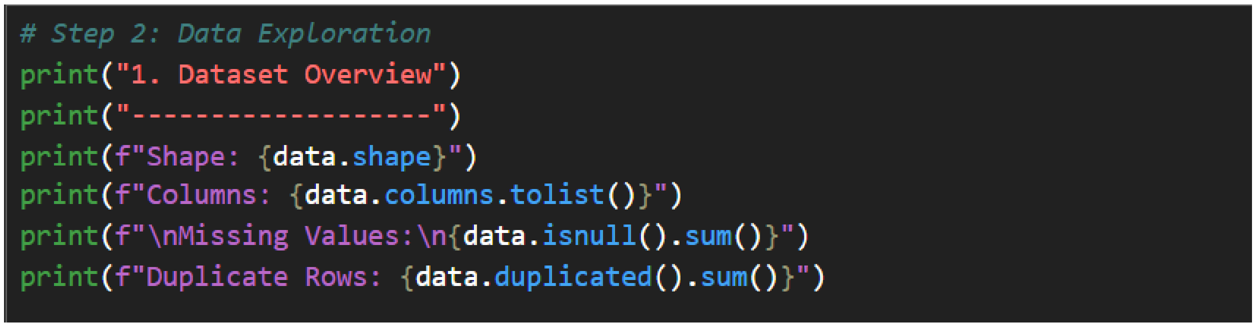
```python
# Importing Libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
```

Diagram 1 shows the code to import the necessary libraries to analyze and explore the given dataset.



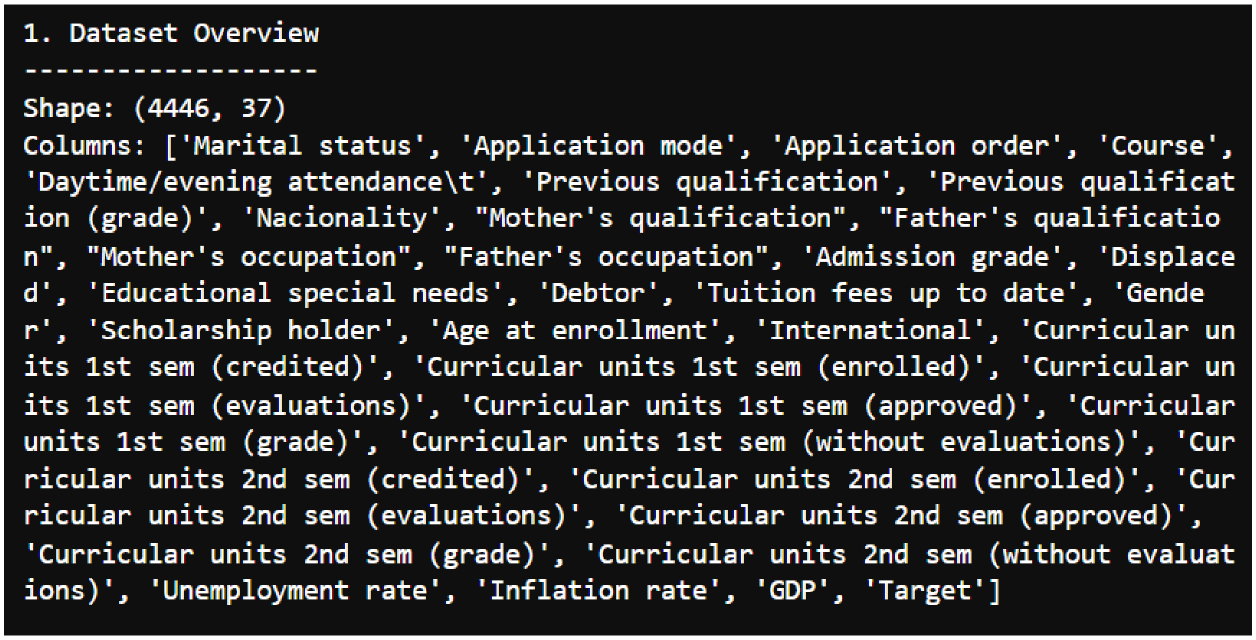
```python
# Step 1: Load the Dataset
data = pd.read_csv('Student_dataset (1).csv')
```

Diagram 2 demonstrates the code to load the dataset using the pd.read_csv() function from the pandas library.



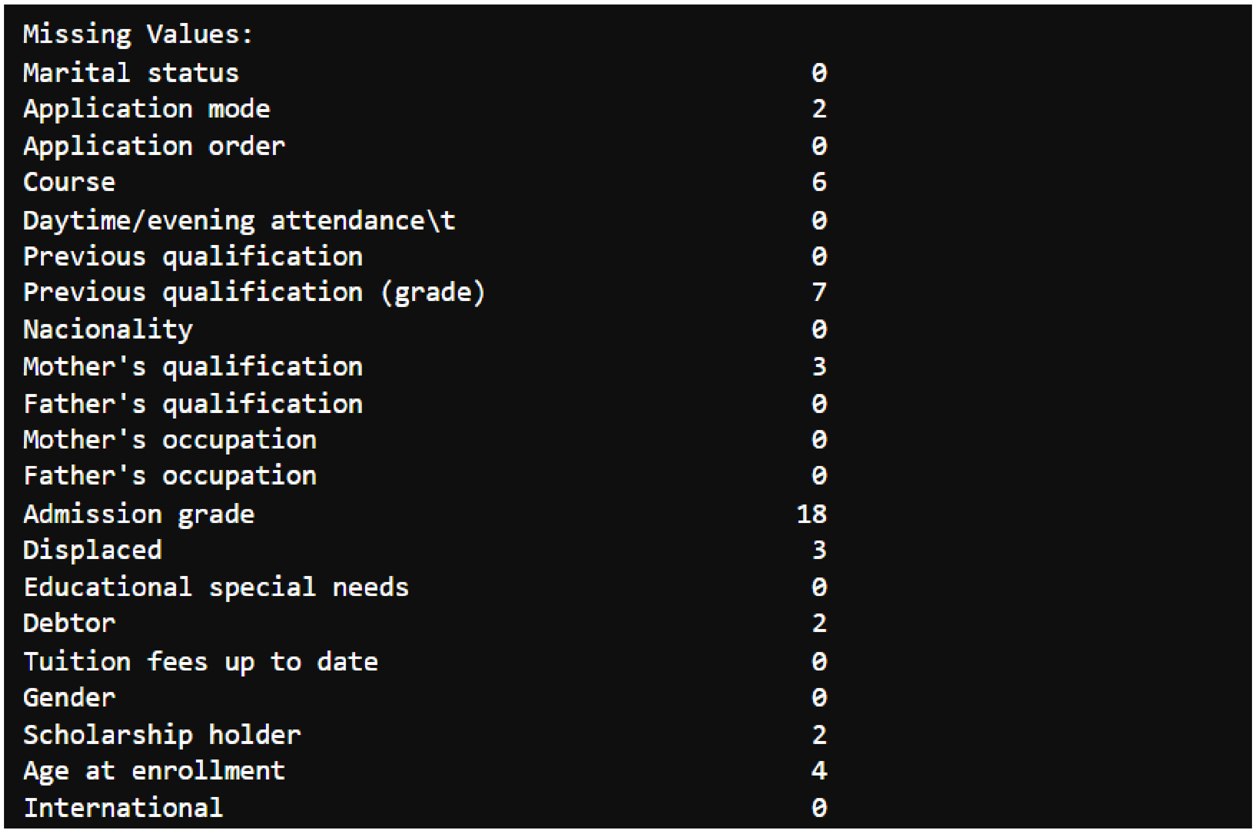
```python
# Step 2: Data Exploration
print("1. Dataset Overview")
print("-------------------")
print(f"Shape: {data.shape}")
print(f"Columns: {data.columns.tolist()}")
print(f"\nMissing Values:\n{data.isnull().sum()}")
print(f"Duplicate Rows: {data.duplicated().sum()}")
```

Diagram 3 presents the initial exploration of the dataset to gain insights into its structure and quality.



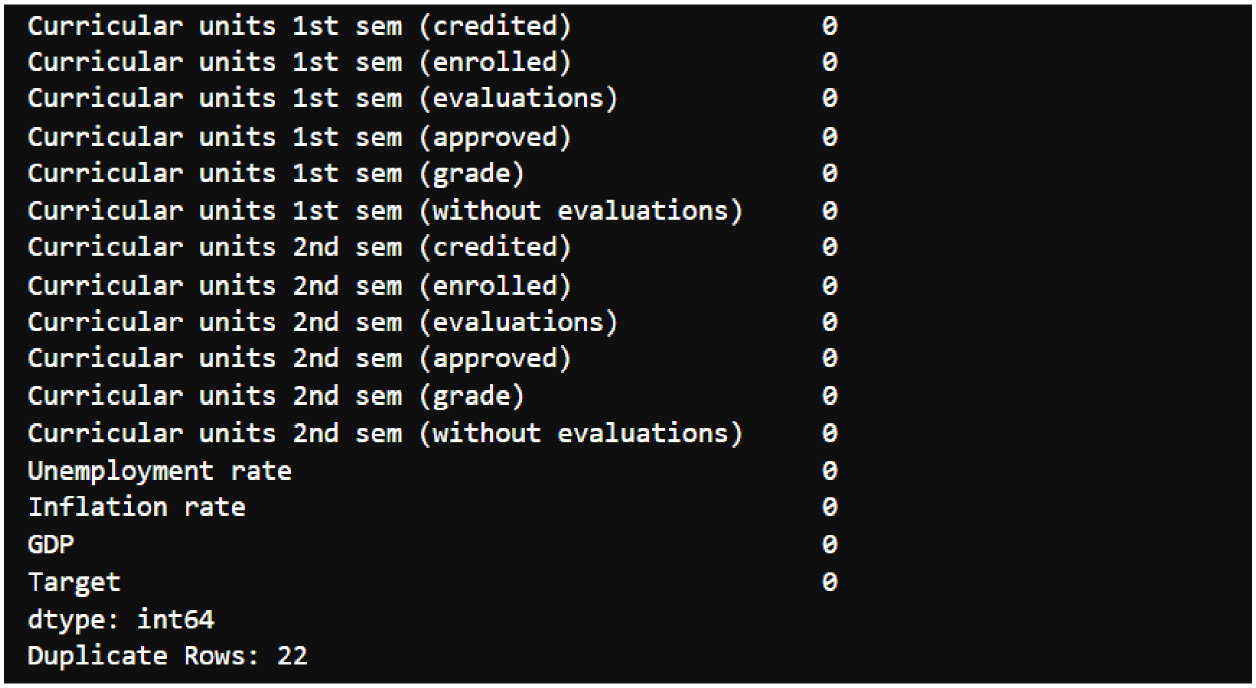
```
1. Dataset Overview
-------------------
Shape: (4446, 37)
Columns: ['Marital status', 'Application mode', 'Application order', 'Course',
'Daytime/evening attendance\t', 'Previous qualification', 'Previous qualificat
ion (grade)', 'Nacionality', "Mother's qualification", "Father's qualificatio
n", "Mother's occupation", "Father's occupation", 'Admission grade', 'Displace
d', 'Educational special needs', 'Debtor', 'Tuition fees up to date', 'Gende
r', 'Scholarship holder', 'Age at enrollment', 'International', 'Curricular un
its 1st sem (credited)', 'Curricular units 1st sem (enrolled)', 'Curricular un
its 1st sem (evaluations)', 'Curricular units 1st sem (approved)', 'Curricular
units 1st sem (grade)', 'Curricular units 1st sem (without evaluations)', 'Cur
ricular units 2nd sem (credited)', 'Curricular units 2nd sem (enrolled)', 'Cur
ricular units 2nd sem (evaluations)', 'Curricular units 2nd sem (approved)',
'Curricular units 2nd sem (grade)', 'Curricular units 2nd sem (without evaluat
ions)', 'Unemployment rate', 'Inflation rate', 'GDP', 'Target']
```

Diagram 4



```
Missing Values:
Marital status                                  0
Application mode                                2
Application order                               0
Course                                          6
Daytime/evening attendance\t                    0
Previous qualification                          0
Previous qualification (grade)                  7
Nacionality                                     0
Mother's qualification                          3
Father's qualification                          0
Mother's occupation                             0
Father's occupation                             0
Admission grade                                18
Displaced                                       3
Educational special needs                       0
Debtor                                          2
Tuition fees up to date                         0
Gender                                          0
Scholarship holder                              2
Age at enrollment                               4
International                                   0
```

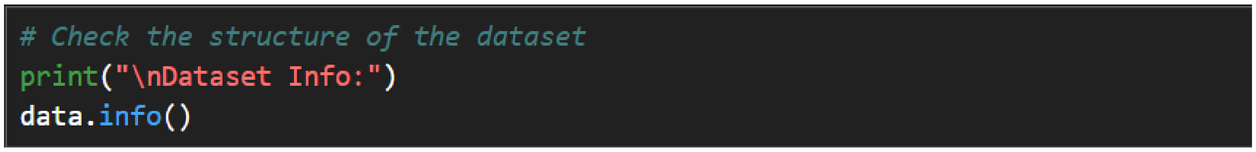
Diagram 5



```
Curricular units 1st sem (credited)             0
Curricular units 1st sem (enrolled)             0
Curricular units 1st sem (evaluations)          0
Curricular units 1st sem (approved)             0
Curricular units 1st sem (grade)                0
Curricular units 1st sem (without evaluations)  0
Curricular units 2nd sem (credited)             0
Curricular units 2nd sem (enrolled)             0
Curricular units 2nd sem (evaluations)          0
Curricular units 2nd sem (approved)             0
Curricular units 2nd sem (grade)                0
Curricular units 2nd sem (without evaluations)  0
Unemployment rate                               0
Inflation rate                                  0
GDP                                             0
Target                                          0
dtype: int64
Duplicate Rows: 22
```

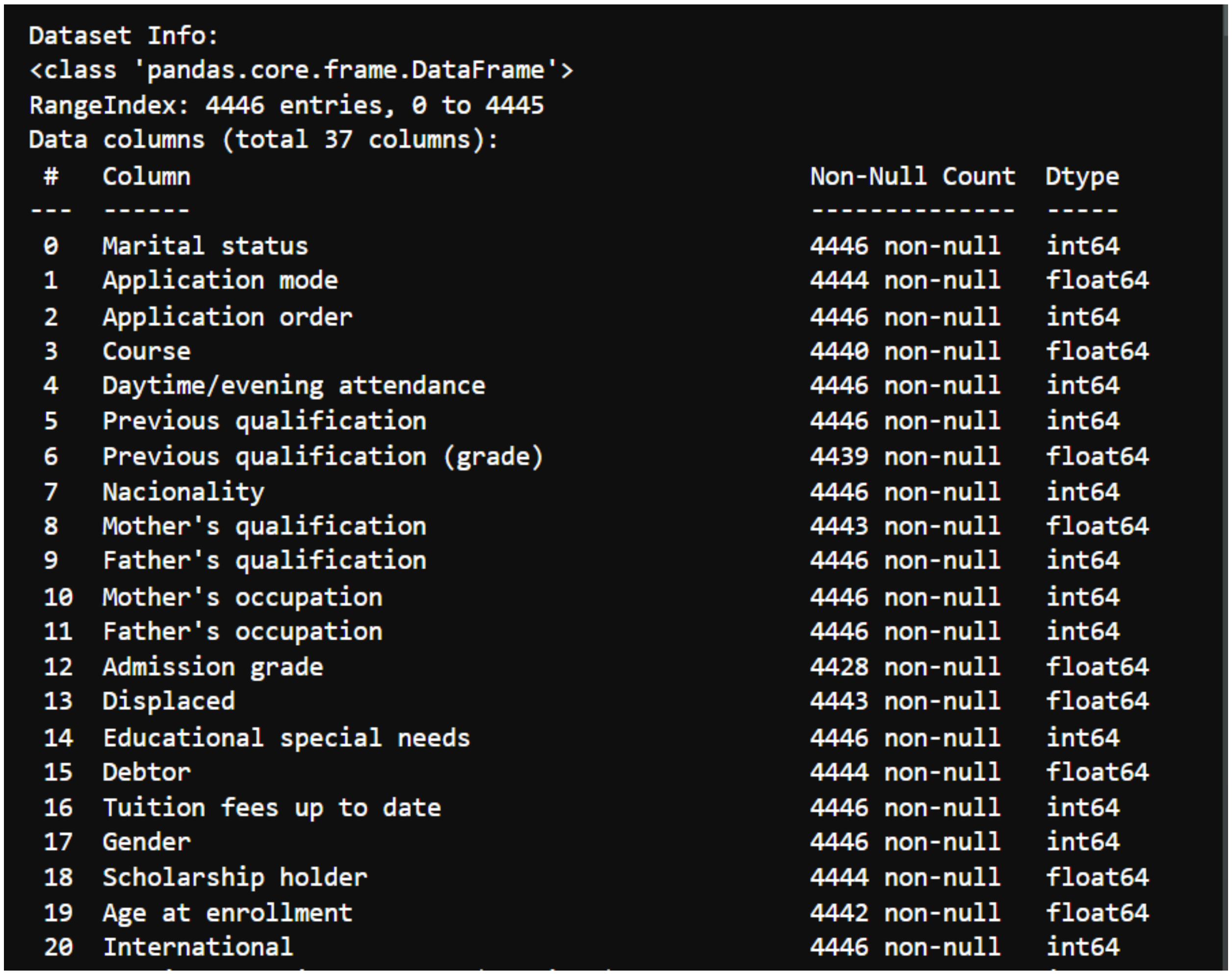
Diagram 6

Diagram 4, 5 and 6 displays the results of the initial exploration. The dataset has 4446 rows and 37 columns, including features like "Marital Status," "Admission Grade," and "Target." It shows missing values in several columns, such as "Application Mode" and "Admission Grade," which will require handling during preprocessing.



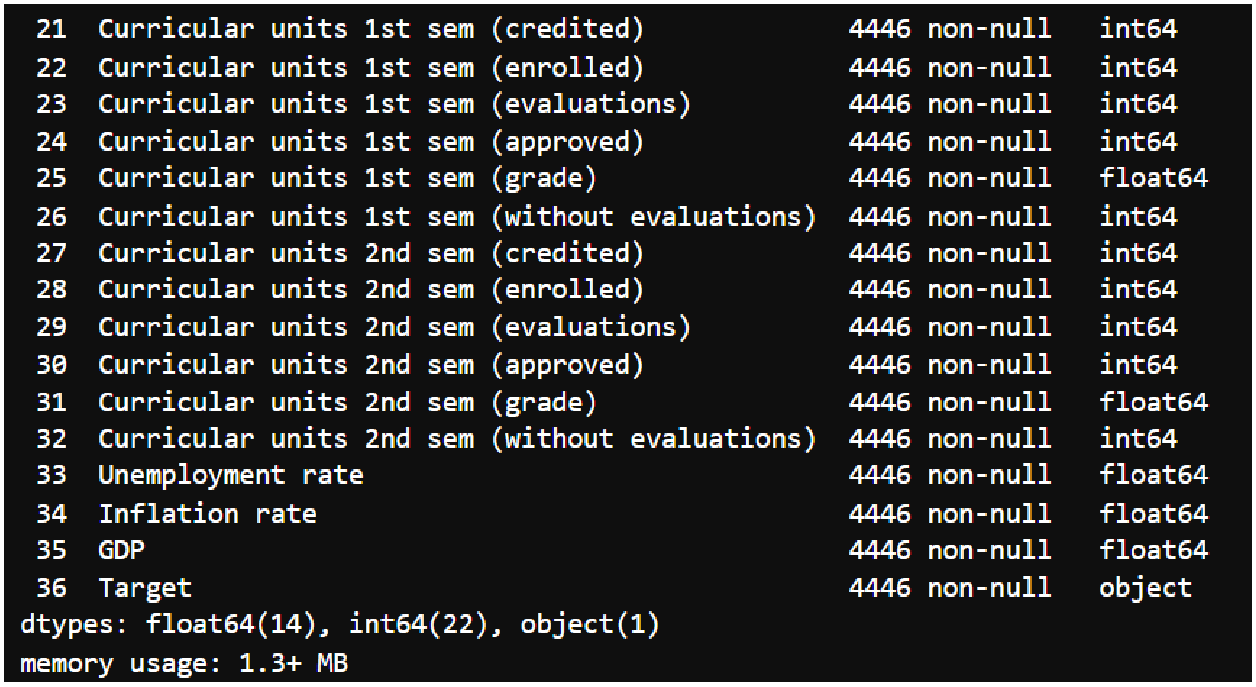
```
# Check the structure of the dataset
print("\nDataset Info:")
data.info()
```

Diagram 7 shows the code to retrieve detailed information about the dataset using the `data.info()` *method.*



```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4446 entries, 0 to 4445
Data columns (total 37 columns):
 #   Column                                          Non-Null Count  Dtype  
---  ------                                          --------------  -----  
 0   Marital status                                  4446 non-null   int64  
 1   Application mode                                4444 non-null   float64
 2   Application order                               4446 non-null   int64  
 3   Course                                          4440 non-null   float64
 4   Daytime/evening attendance                      4446 non-null   int64  
 5   Previous qualification                          4446 non-null   int64  
 6   Previous qualification (grade)                  4439 non-null   float64
 7   Nacionality                                     4446 non-null   int64  
 8   Mother's qualification                          4443 non-null   float64
 9   Father's qualification                          4446 non-null   int64  
 10  Mother's occupation                             4446 non-null   int64  
 11  Father's occupation                             4446 non-null   int64  
 12  Admission grade                                 4428 non-null   float64
 13  Displaced                                       4443 non-null   float64
 14  Educational special needs                       4446 non-null   int64  
 15  Debtor                                          4444 non-null   float64
 16  Tuition fees up to date                         4446 non-null   int64  
 17  Gender                                          4446 non-null   int64  
 18  Scholarship holder                              4444 non-null   float64
 19  Age at enrollment                               4442 non-null   float64
 20  International                                   4446 non-null   int64  
```

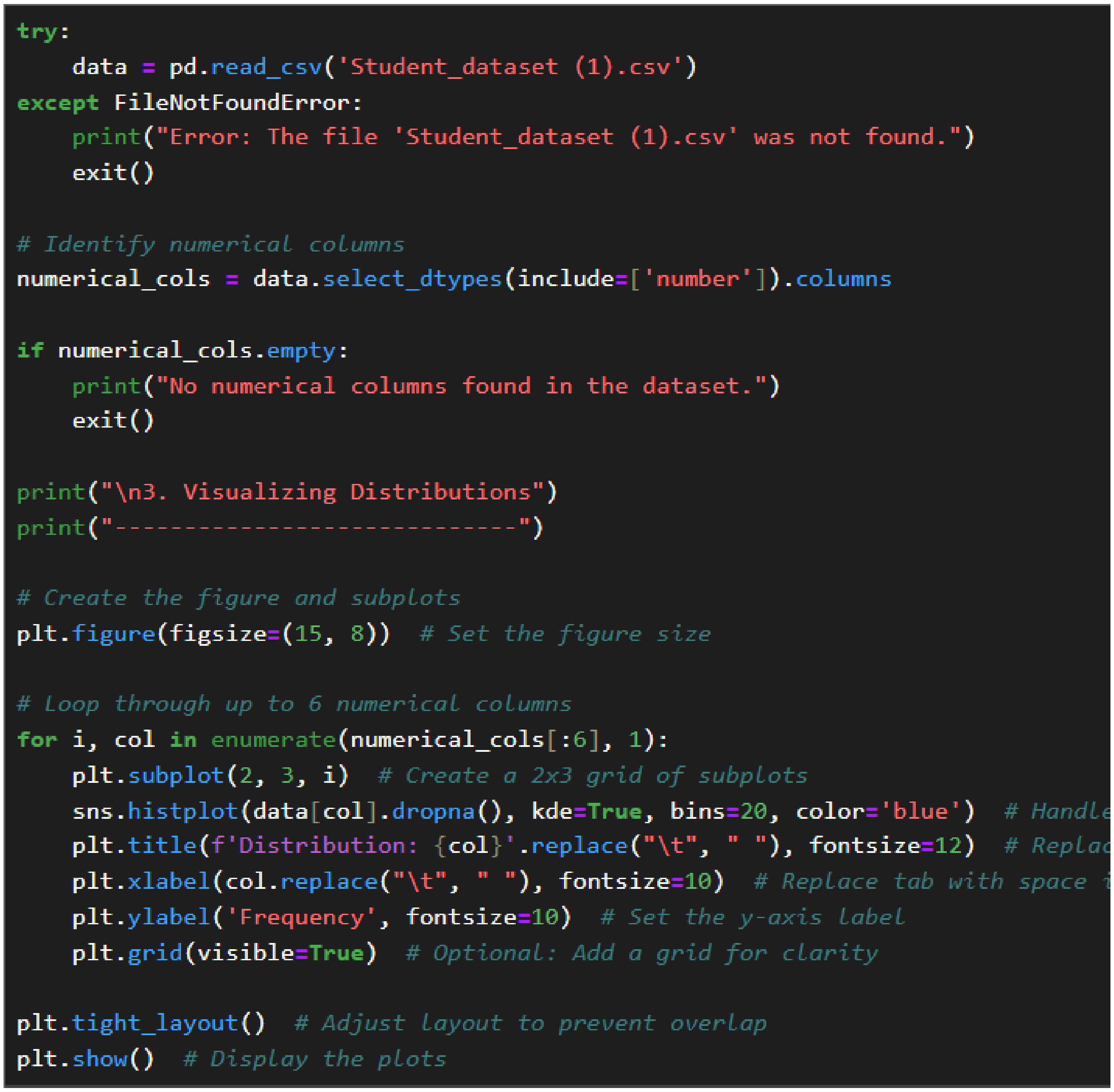
Diagram 8



```
 21  Curricular units 1st sem (credited)             4446 non-null   int64
 22  Curricular units 1st sem (enrolled)             4446 non-null   int64
 23  Curricular units 1st sem (evaluations)          4446 non-null   int64
 24  Curricular units 1st sem (approved)             4446 non-null   int64
 25  Curricular units 1st sem (grade)                4446 non-null   float64
 26  Curricular units 1st sem (without evaluations)  4446 non-null   int64
 27  Curricular units 2nd sem (credited)             4446 non-null   int64
 28  Curricular units 2nd sem (enrolled)             4446 non-null   int64
 29  Curricular units 2nd sem (evaluations)          4446 non-null   int64
 30  Curricular units 2nd sem (approved)             4446 non-null   int64
 31  Curricular units 2nd sem (grade)                4446 non-null   float64
 32  Curricular units 2nd sem (without evaluations)  4446 non-null   int64
 33  Unemployment rate                               4446 non-null   float64
 34  Inflation rate                                  4446 non-null   float64
 35  GDP                                             4446 non-null   float64
 36  Target                                          4446 non-null   object
dtypes: float64(14), int64(22), object(1)
memory usage: 1.3+ MB
```

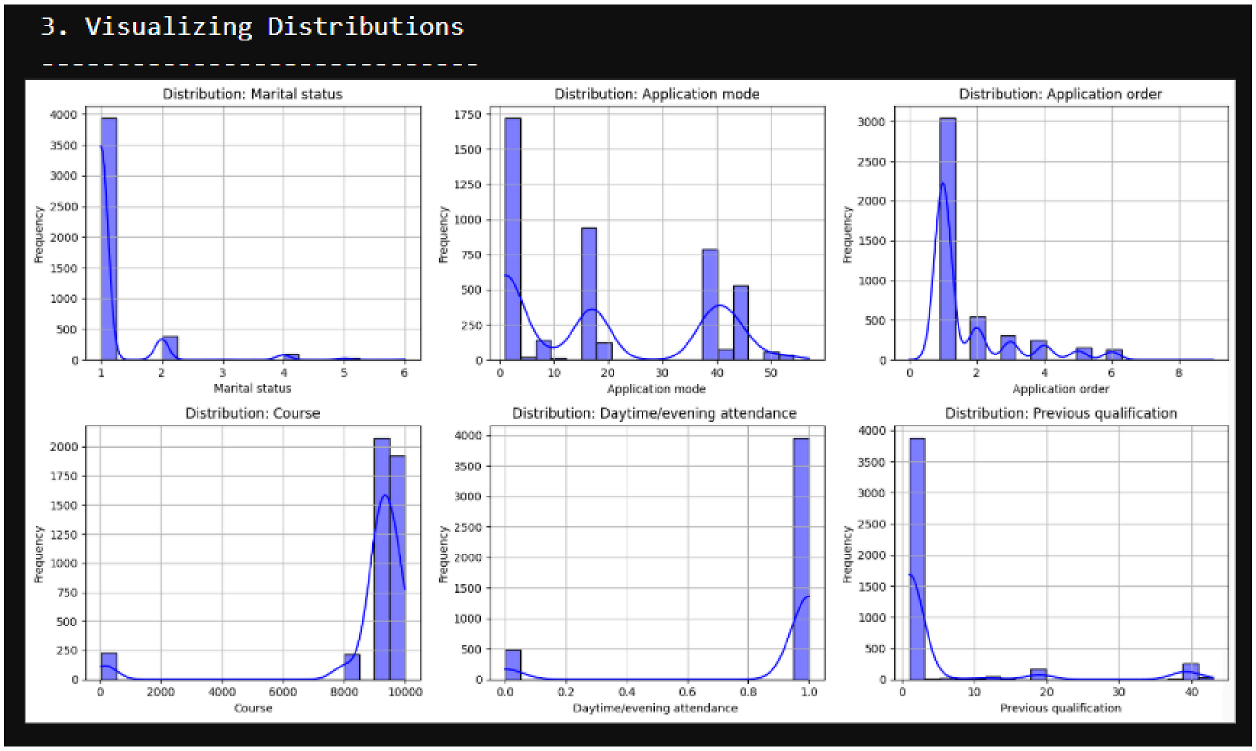
Diagram 9

Diagram 8 and 9 provides the output of the data.info() method. It reveals that the dataset consists of 37 columns, with a mix of integer, float, and object data types. While most columns are complete, some, like "Application Mode" and "Admission Grade," have missing values.



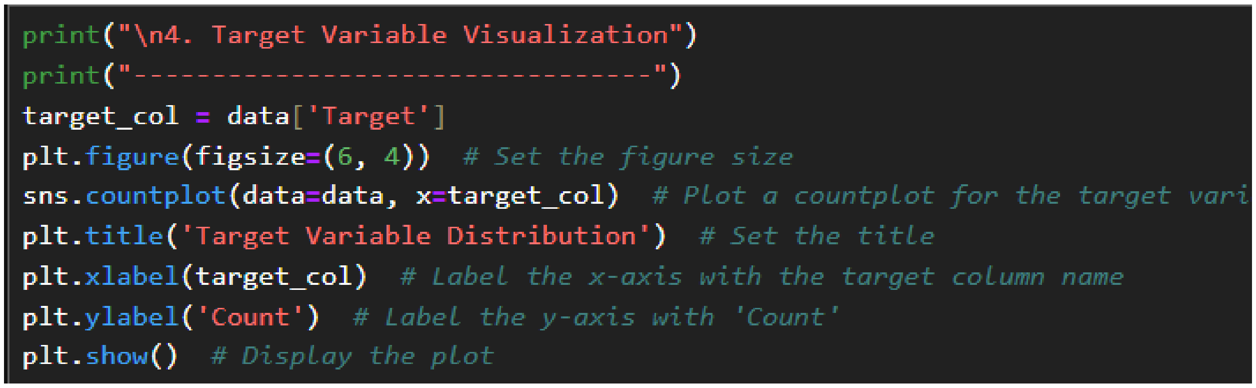
```python
try:
    data = pd.read_csv('Student_dataset (1).csv')
except FileNotFoundError:
    print("Error: The file 'Student_dataset (1).csv' was not found.")
    exit()

# Identify numerical columns
numerical_cols = data.select_dtypes(include=['number']).columns

if numerical_cols.empty:
    print("No numerical columns found in the dataset.")
    exit()

print("\n3. Visualizing Distributions")
print("-----------------------------")

# Create the figure and subplots
plt.figure(figsize=(15, 8))  # Set the figure size

# Loop through up to 6 numerical columns
for i, col in enumerate(numerical_cols[:6], 1):
    plt.subplot(2, 3, i)  # Create a 2x3 grid of subplots
    sns.histplot(data[col].dropna(), kde=True, bins=20, color='blue')  # Handle
    plt.title(f'Distribution: {col}'.replace("\t", " "), fontsize=12)  # Replac
    plt.xlabel(col.replace("\t", " "), fontsize=10)  # Replace tab with space i
    plt.ylabel('Frequency', fontsize=10)  # Set the y-axis label
    plt.grid(visible=True)  # Optional: Add a grid for clarity

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()  # Display the plots
```

Diagram 10 illustrates the process of visualizing the distributions of numerical columns in the dataset. The code begins by loading the dataset and handling potential errors, such as a missing file. It identifies numerical columns in the dataset using the select_dtypes method.



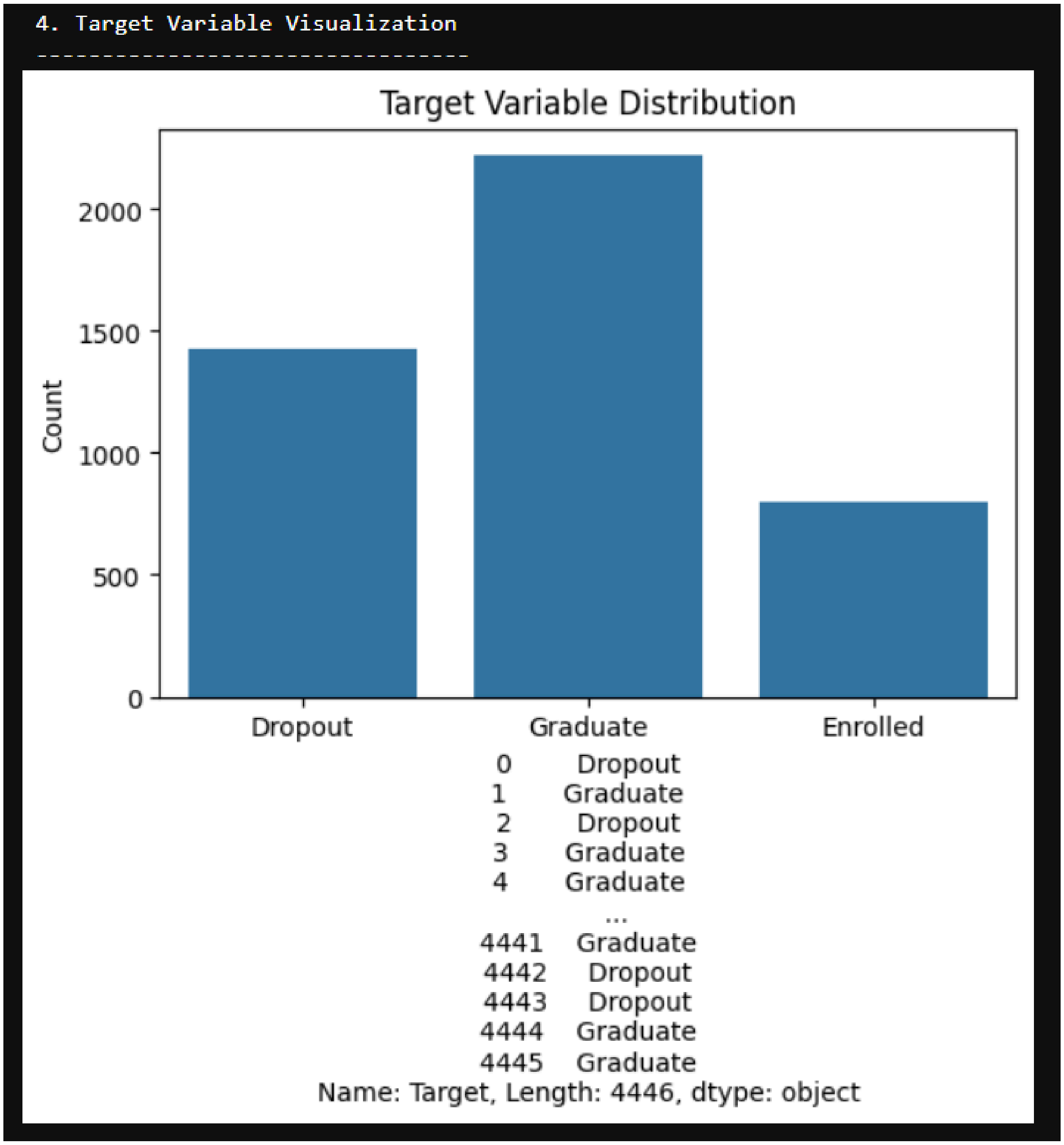
```
3. Visualizing Distributions
------------------------------
```

Diagram 11 showcases the visualization distribution of the data.



```
print("\n4. Target Variable Visualization")
print("--------------------------------")
target_col = data['Target']
plt.figure(figsize=(6, 4))  # Set the figure size
sns.countplot(data=data, x=target_col)  # Plot a countplot for the target vari
plt.title('Target Variable Distribution')  # Set the title
plt.xlabel(target_col)  # Label the x-axis with the target column name
plt.ylabel('Count')  # Label the y-axis with 'Count'
plt.show()  # Display the plot
```

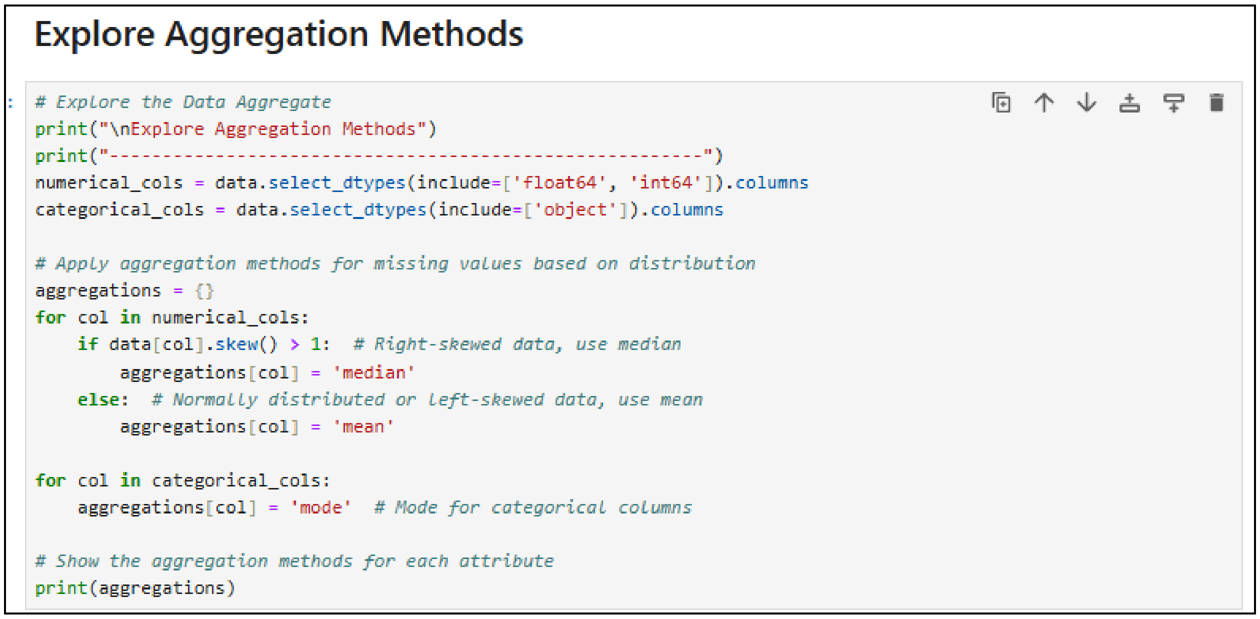
Diagram 12 shows the process of visualizing the target variable in the dataset. The code uses the sns.countplot function to create a count plot, displaying the frequency of each unique value in the target column.



```
4. Target Variable Visualization
--------------------------------
```

```
0        Dropout
1       Graduate
2        Dropout
3       Graduate
4       Graduate
          ...
4441    Graduate
4442     Dropout
4443     Dropout
4444    Graduate
4445    Graduate
Name: Target, Length: 4446, dtype: object
```

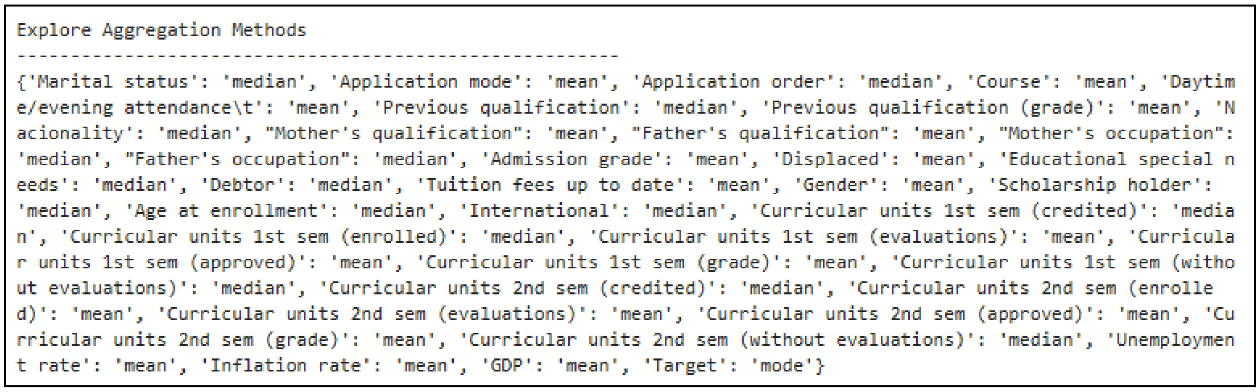
Diagram 13 illustrates the target variable's distribution using a count plot to understand its categorical composition. This sequence of diagrams demonstrates how each process leads to its respective visualization, facilitating better data exploration and interpretation.



Explore Aggregation Methods

```
# Explore the Data Aggregate
print("\nExplore Aggregation Methods")
print("--------------------------------------------------------")
numerical_cols = data.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = data.select_dtypes(include=['object']).columns

# Apply aggregation methods for missing values based on distribution
aggregations = {}
for col in numerical_cols:
    if data[col].skew() > 1:  # Right-skewed data, use median
        aggregations[col] = 'median'
    else:  # Normally distributed or left-skewed data, use mean
        aggregations[col] = 'mean'

for col in categorical_cols:
    aggregations[col] = 'mode'  # Mode for categorical columns

# Show the aggregation methods for each attribute
print(aggregations)
```

Diagram 14 analyzes the dataset to determine the best method for handling missing values. It separates numerical and categorical columns, then checks the skewness of numerical data to decide whether to use the mean (for normal or left-skewed data) or the median (for right-skewed data) for imputation. For categorical columns, it suggests using the mode.

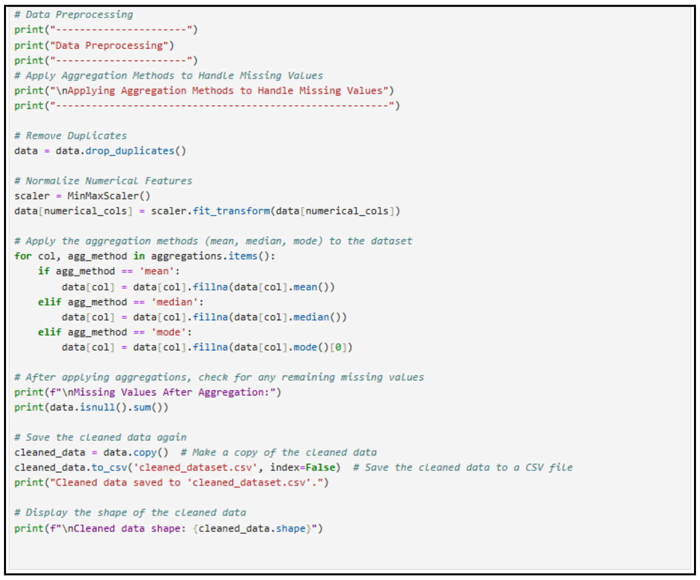


```
Explore Aggregation Methods
--------------------------------------------------------
{'Marital status': 'median', 'Application mode': 'mean', 'Application order': 'median', 'Course': 'mean', 'Daytim
e/evening attendance\t': 'mean', 'Previous qualification': 'median', 'Previous qualification (grade)': 'mean', 'N
acionality': 'median', "Mother's qualification": 'mean', "Father's qualification": 'mean', "Mother's occupation":
'median', "Father's occupation": 'median', 'Admission grade': 'mean', 'Displaced': 'mean', 'Educational special n
eeds': 'median', 'Debtor': 'median', 'Tuition fees up to date': 'mean', 'Gender': 'mean', 'Scholarship holder':
'median', 'Age at enrollment': 'median', 'International': 'median', 'Curricular units 1st sem (credited)': 'media
n', 'Curricular units 1st sem (enrolled)': 'median', 'Curricular units 1st sem (evaluations)': 'mean', 'Curricula
r units 1st sem (approved)': 'mean', 'Curricular units 1st sem (grade)': 'mean', 'Curricular units 1st sem (witho
ut evaluations)': 'median', 'Curricular units 2nd sem (credited)': 'median', 'Curricular units 2nd sem (enrolle
d)': 'mean', 'Curricular units 2nd sem (evaluations)': 'mean', 'Curricular units 2nd sem (approved)': 'mean', 'Cu
rricular units 2nd sem (grade)': 'mean', 'Curricular units 2nd sem (without evaluations)': 'median', 'Unemploymen
t rate': 'mean', 'Inflation rate': 'mean', 'GDP': 'mean', 'Target': 'mode'}
```

Diagram 15 creates and displays a list of these aggregation methods for each column, helping to ensure appropriate handling of missing data based on its distribution.

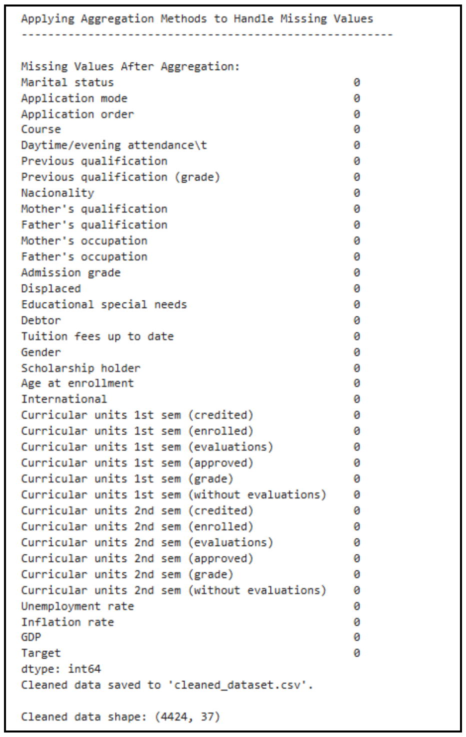
4.0 Data Preprocessing

4.1 Data Cleaning



```python
# Data Preprocessing
print("---------------------")
print("Data Preprocessing")
print("---------------------")
# Apply Aggregation Methods to Handle Missing Values
print("\nApplying Aggregation Methods to Handle Missing Values")
print("-------------------------------------------------------")

# Remove Duplicates
data = data.drop_duplicates()

# Normalize Numerical Features
scaler = MinMaxScaler()
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])

# Apply the aggregation methods (mean, median, mode) to the dataset
for col, agg_method in aggregations.items():
    if agg_method == 'mean':
        data[col] = data[col].fillna(data[col].mean())
    elif agg_method == 'median':
        data[col] = data[col].fillna(data[col].median())
    elif agg_method == 'mode':
        data[col] = data[col].fillna(data[col].mode()[0])

# After applying aggregations, check for any remaining missing values
print(f"\nMissing Values After Aggregation:")
print(data.isnull().sum())

# Save the cleaned data again
cleaned_data = data.copy()  # Make a copy of the cleaned data
cleaned_data.to_csv('cleaned_dataset.csv', index=False)  # Save the cleaned data to a CSV file
print("Cleaned data saved to 'cleaned_dataset.csv'.")

# Display the shape of the cleaned data
print(f"\nCleaned data shape: {cleaned_data.shape}")
```

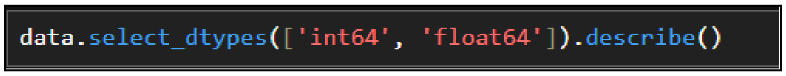
Diagram 16 performs data preprocessing to clean and standardize the dataset. It starts by removing duplicate entries to ensure data consistency. Next, it normalizes numerical features using the MinMaxScaler to bring all numerical values into the same scale. It then fills missing values in the dataset using pre-determined aggregation methods (mean for normally distributed data, median for skewed numerical data, and mode for categorical data). After handling missing values, it checks for any remaining missing data. Finally, the cleaned dataset is saved to a CSV file, and its shape (number of rows and columns) is displayed for verification.



```
Applying Aggregation Methods to Handle Missing Values
--------------------------------------------------------

Missing Values After Aggregation:
Marital status                                    0
Application mode                                  0
Application order                                 0
Course                                            0
Daytime/evening attendance\t                      0
Previous qualification                            0
Previous qualification (grade)                    0
Nacionality                                       0
Mother's qualification                            0
Father's qualification                            0
Mother's occupation                               0
Father's occupation                               0
Admission grade                                   0
Displaced                                         0
Educational special needs                         0
Debtor                                            0
Tuition fees up to date                           0
Gender                                            0
Scholarship holder                                0
Age at enrollment                                 0
International                                     0
Curricular units 1st sem (credited)               0
Curricular units 1st sem (enrolled)               0
Curricular units 1st sem (evaluations)            0
Curricular units 1st sem (approved)               0
Curricular units 1st sem (grade)                  0
Curricular units 1st sem (without evaluations)    0
Curricular units 2nd sem (credited)               0
Curricular units 2nd sem (enrolled)               0
Curricular units 2nd sem (evaluations)            0
Curricular units 2nd sem (approved)               0
Curricular units 2nd sem (grade)                  0
Curricular units 2nd sem (without evaluations)    0
Unemployment rate                                 0
Inflation rate                                    0
GDP                                               0
Target                                            0
dtype: int64
Cleaned data saved to 'cleaned_dataset.csv'.

Cleaned data shape: (4424, 37)
```

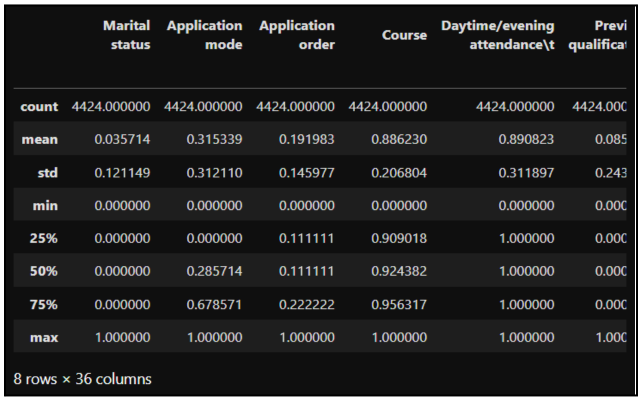
Diagram 17 displayed the cleaned data after applying aggregation methods.

4.2 Describing the Data Types



```
data.select_dtypes(['int64', 'float64']).describe()
```

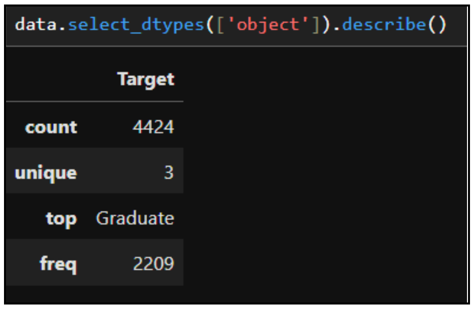
Diagram 18 selects all numerical columns (with data types int64 and float64) from the dataset and generates descriptive statistics for them using the .describe() method.



	Marital status	Application mode	Application order	Course	Daytime/evening attendance\t	Previ qualifica
count	4424.000000	4424.000000	4424.000000	4424.000000	4424.000000	4424.000
mean	0.035714	0.315339	0.191983	0.886230	0.890823	0.085
std	0.121149	0.312110	0.145977	0.206804	0.311897	0.243
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.000000	0.000000	0.111111	0.909018	1.000000	0.000
50%	0.000000	0.285714	0.111111	0.924382	1.000000	0.000
75%	0.000000	0.678571	0.222222	0.956317	1.000000	0.000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000

8 rows × 36 columns

Diagram 19 provides an overview of key metrics like count, mean, standard deviation, minimum, maximum, and percentiles (25th, 50th, and 75th) for each numerical column, helping to understand the dataset's distribution and identify any anomalies or patterns.

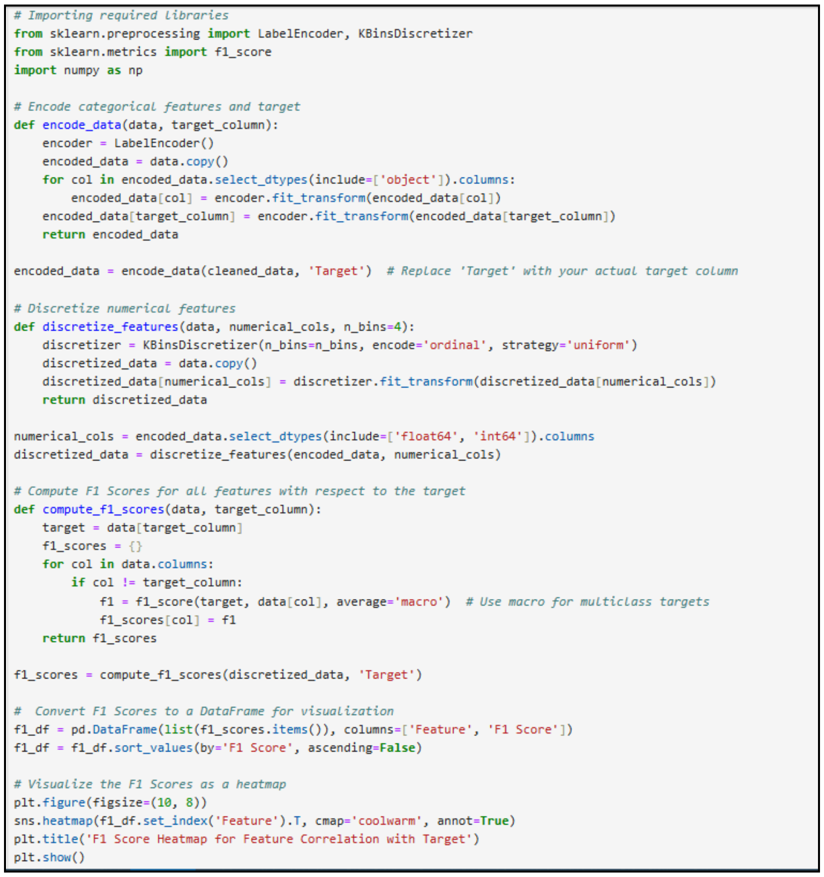


```
data.select_dtypes(['object']).describe()
```

	Target
count	4424
unique	3
top	Graduate
freq	2209

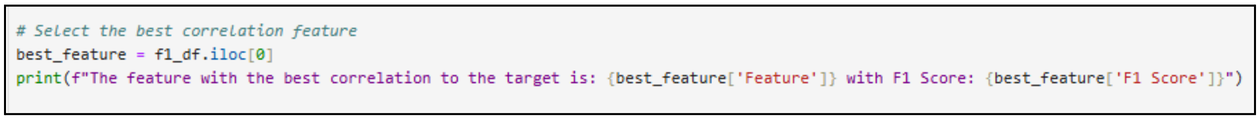
Diagram 20 selects all categorical columns (with data type object) from the dataset and generates summary statistics for them using the .describe() method. The output shows details for the categorical column "Target," including the total count of non-missing values (4424), the number of unique categories (3), the most frequent category ("Graduate"), and the frequency of this category (2209). This provides an overview of the categorical data distribution and helps identify dominant or frequent categories.

4.3 F1 Score Heatmap for Feature Correlation



```python
# Importing required libraries
from sklearn.preprocessing import LabelEncoder, KBinsDiscretizer
from sklearn.metrics import f1_score
import numpy as np

# Encode categorical features and target
def encode_data(data, target_column):
    encoder = LabelEncoder()
    encoded_data = data.copy()
    for col in encoded_data.select_dtypes(include=['object']).columns:
        encoded_data[col] = encoder.fit_transform(encoded_data[col])
    encoded_data[target_column] = encoder.fit_transform(encoded_data[target_column])
    return encoded_data

encoded_data = encode_data(cleaned_data, 'Target')  # Replace 'Target' with your actual target column

# Discretize numerical features
def discretize_features(data, numerical_cols, n_bins=4):
    discretizer = KBinsDiscretizer(n_bins=n_bins, encode='ordinal', strategy='uniform')
    discretized_data = data.copy()
    discretized_data[numerical_cols] = discretizer.fit_transform(discretized_data[numerical_cols])
    return discretized_data

numerical_cols = encoded_data.select_dtypes(include=['float64', 'int64']).columns
discretized_data = discretize_features(encoded_data, numerical_cols)

# Compute F1 Scores for all features with respect to the target
def compute_f1_scores(data, target_column):
    target = data[target_column]
    f1_scores = {}
    for col in data.columns:
        if col != target_column:
            f1 = f1_score(target, data[col], average='macro')  # Use macro for multiclass targets
            f1_scores[col] = f1
    return f1_scores

f1_scores = compute_f1_scores(discretized_data, 'Target')

#  Convert F1 Scores to a DataFrame for visualization
f1_df = pd.DataFrame(list(f1_scores.items()), columns=['Feature', 'F1 Score'])
f1_df = f1_df.sort_values(by='F1 Score', ascending=False)

# Visualize the F1 Scores as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(f1_df.set_index('Feature').T, cmap='coolwarm', annot=True)
plt.title('F1 Score Heatmap for Feature Correlation with Target')
plt.show()
```

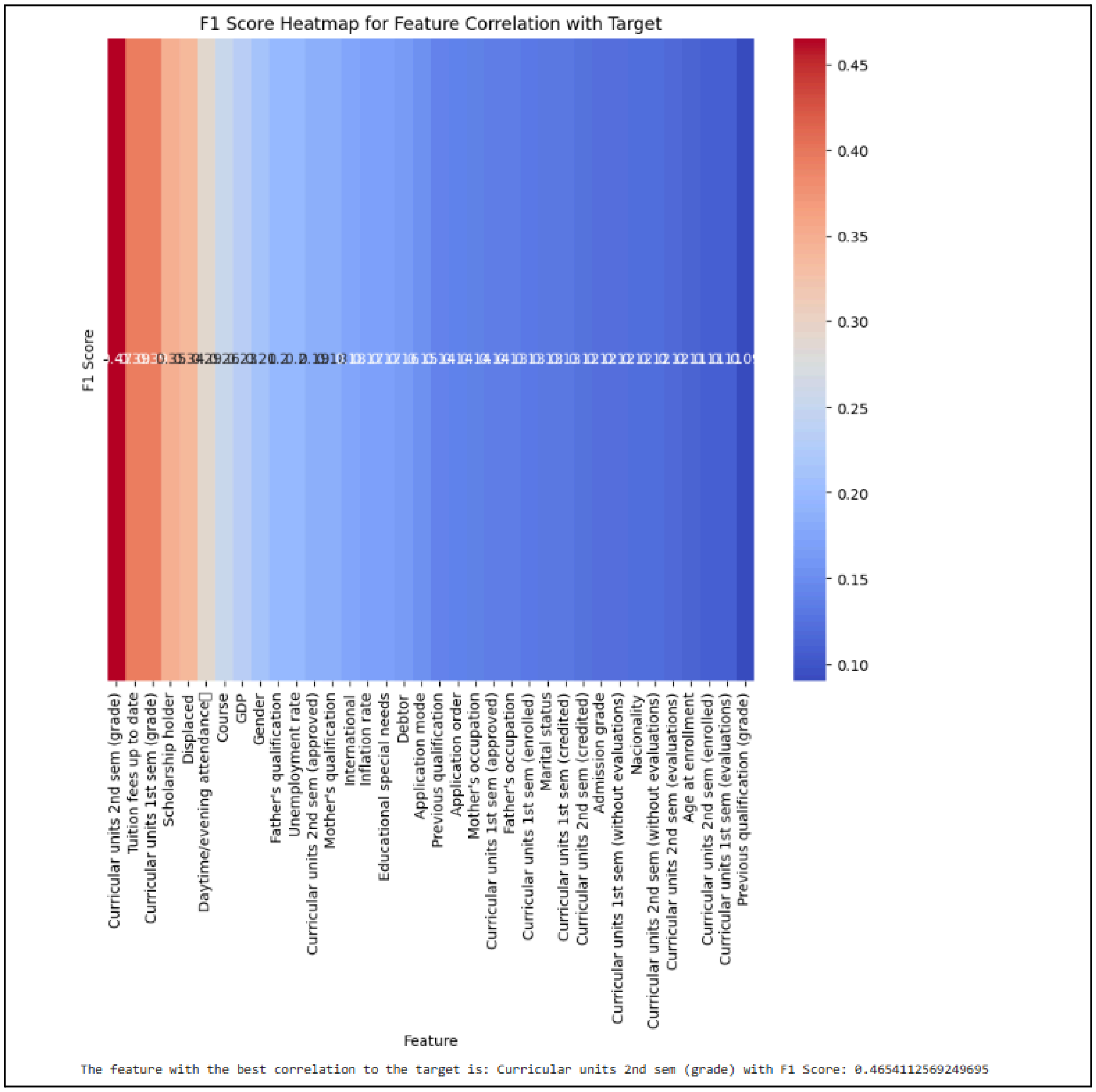
Diagram 21



```python
# Select the best correlation feature
best_feature = f1_df.iloc[0]
print(f"The feature with the best correlation to the target is: {best_feature['Feature']} with F1 Score: {best_feature['F1 Score']}")
```

Diagram 22

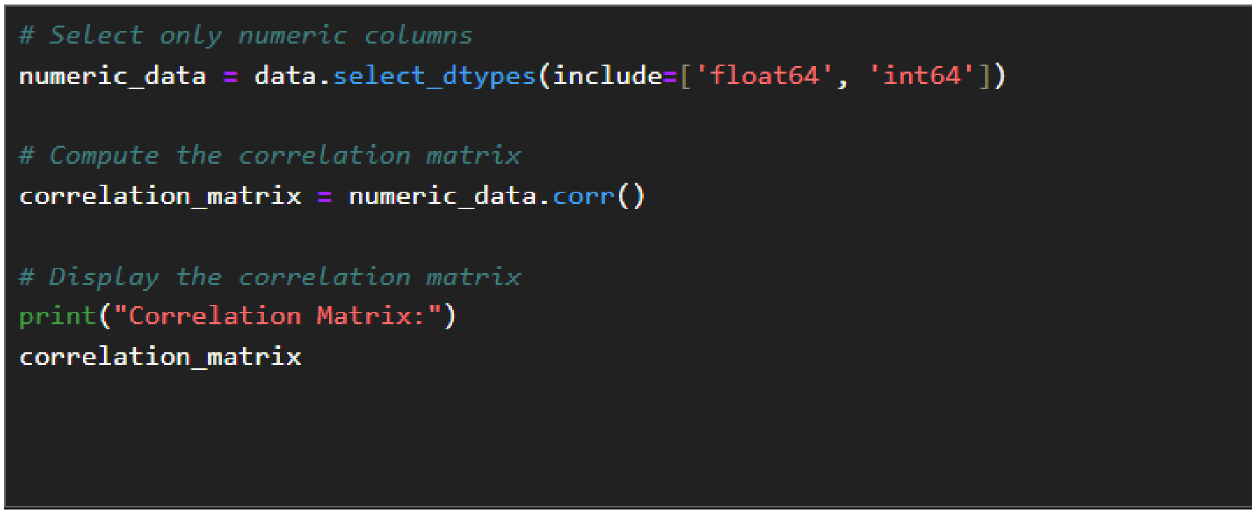
Diagram 21 & 22 evaluates the importance of features in predicting the target variable in the dataset. First, it encodes categorical features and the target variable into numerical format using LabelEncoder. Then, it discretized numerical features into bins using KBinsDiscretizer to handle continuous data. After that, it computes the F1 scores for each feature in relation to the target variable, which measures their correlation and predictive power. The F1 scores are stored in a DataFrame, sorted, and visualized using a heatmap for better understanding.



```
The feature with the best correlation to the target is: Curricular units 2nd sem (grade) with F1 Score: 0.4654112569249695
```

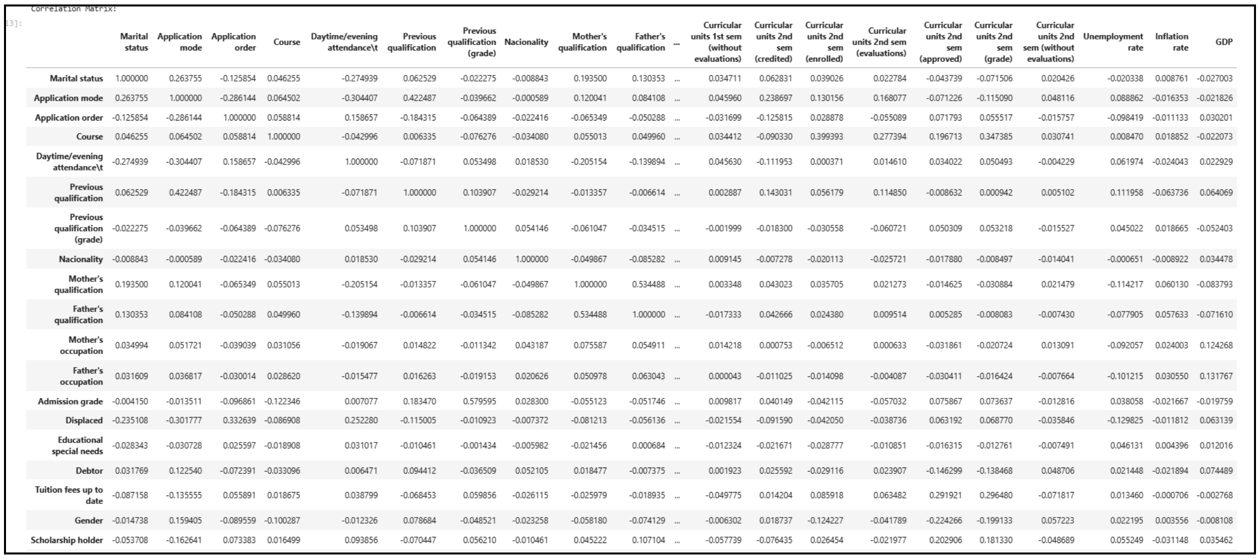
Diagram 23 feature with the highest correlation (best F1 score) is identified and displayed, helping to pinpoint the most impactful feature for the target prediction.

4.4 Correlation Matrix



```
# Select only numeric columns
numeric_data = data.select_dtypes(include=['float64', 'int64'])

# Compute the correlation matrix
correlation_matrix = numeric_data.corr()

# Display the correlation matrix
print("Correlation Matrix:")
correlation_matrix
```

Diagram 24 shows This code calculates and displays the correlation matrix for numerical attributes in a dataset. It begins by selecting only the numerical columns (float64 and int64 data types) from the dataset to focus on quantitative relationships. Then, it computes the correlation matrix..

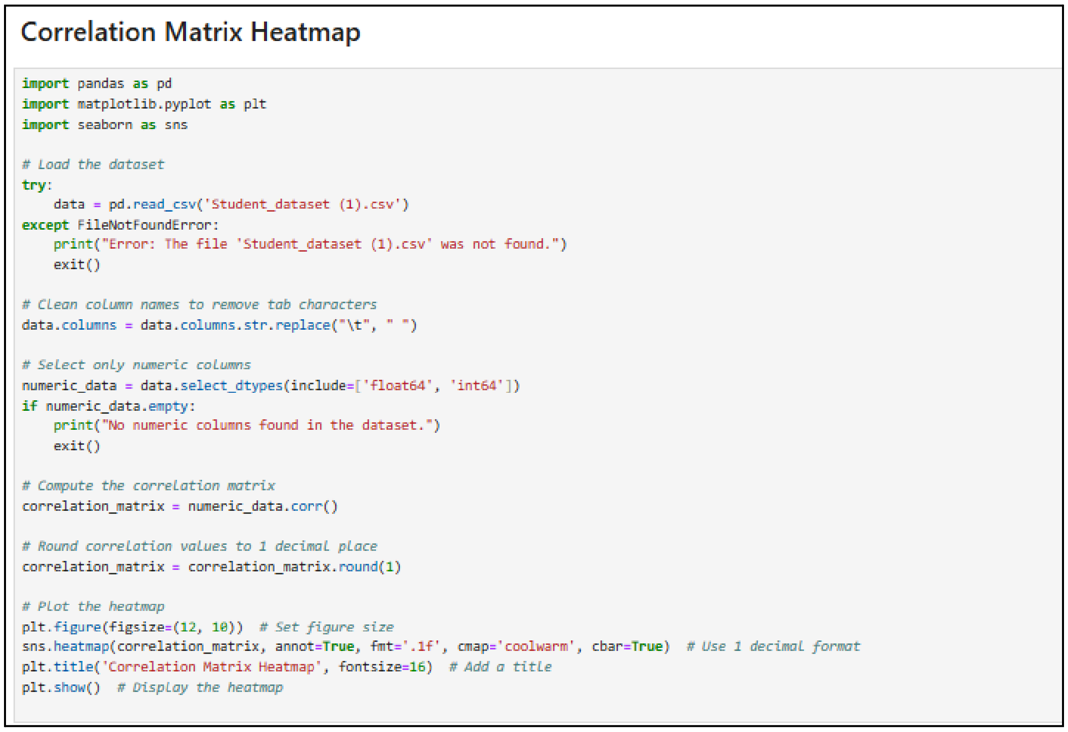


```
Correlation Matrix:
```

	Marital status	Application mode	Application order	Course	Daytime/evening attendance\t	Previous qualification	Previous qualification (grade)	Nacionality	Mother's qualification	Father's qualification	...	Curricular units 1st sem (without evaluations)	Curricular units 2nd sem (credited)	Curricular units 2nd sem (enrolled)	Curricular units 2nd sem (evaluations)	Curricular units 2nd sem (approved)	Curricular units 2nd sem (grade)	Curricular units 2nd sem (without evaluations)	Unemployment rate	Inflation rate	GDP
Marital status	1.000000	0.263755	-0.125854	0.046255	-0.274939	0.062529	-0.022275	-0.008843	0.193500	0.130353	...	0.034711	0.062831	0.039026	0.022784	-0.043739	-0.071506	0.020426	-0.020338	0.008761	-0.027003
Application mode	0.263755	1.000000	-0.286144	0.064502	-0.304407	0.422487	-0.039662	-0.000589	0.120041	0.084108	...	0.045960	0.238697	0.130156	0.168077	-0.071226	-0.115090	0.048116	0.088862	-0.016353	-0.021826
Application order	-0.125854	-0.286144	1.000000	0.058814	0.158657	-0.184315	-0.064389	-0.022416	-0.065349	-0.050288	...	-0.031699	-0.125815	0.028878	-0.055089	0.071793	0.055517	-0.015757	-0.098419	-0.011133	0.030201
Course	0.046255	0.064502	0.058814	1.000000	-0.042996	0.006335	-0.076276	-0.034080	0.055013	0.049960	...	0.034412	-0.090330	0.399393	0.277394	0.196713	0.347385	0.030741	0.008470	0.018852	-0.022073
Daytime/evening attendance\t	-0.274939	-0.304407	0.158657	-0.042996	1.000000	-0.071871	0.053498	0.018530	-0.205154	-0.139894	...	0.045630	-0.111953	0.000371	0.014610	0.034022	0.050493	-0.004229	0.061974	-0.024043	0.022929
Previous qualification	0.062529	0.422487	-0.184315	0.006335	-0.071871	1.000000	0.103907	-0.029214	-0.013357	-0.006614	...	0.002887	0.143031	0.056179	0.114850	-0.008632	0.000942	0.005102	0.111958	-0.063736	0.064069
Previous qualification (grade)	-0.022275	-0.039662	-0.064389	-0.076276	0.053498	0.103907	1.000000	0.054146	-0.061047	-0.034515	...	-0.001999	-0.018300	-0.030558	-0.060721	0.050309	0.053218	-0.015527	0.045022	0.018665	-0.052403
Nacionality	-0.008843	-0.000589	-0.022416	-0.034080	0.018530	-0.029214	0.054146	1.000000	-0.049867	-0.085282	...	0.009145	-0.007278	-0.020113	-0.025721	-0.017880	-0.008497	-0.014041	-0.000651	-0.008922	0.034478
Mother's qualification	0.193500	0.120041	-0.065349	0.055013	-0.205154	-0.013357	-0.061047	-0.049867	1.000000	0.534488	...	0.003348	0.043023	0.035705	0.021273	-0.014625	-0.030884	0.021479	-0.114217	0.060130	-0.083793
Father's qualification	0.130353	0.084108	-0.050288	0.049960	-0.139894	-0.006614	-0.034515	-0.085282	0.534488	1.000000	...	-0.017333	0.042666	0.024380	0.009514	0.005285	-0.008083	-0.007430	-0.077905	0.057633	-0.071610
Mother's occupation	0.034994	0.051721	-0.039039	0.031056	-0.019067	0.014822	-0.011342	0.043187	0.075587	0.054911	...	0.014218	0.000753	-0.006512	0.000633	-0.031861	-0.020724	0.013091	-0.092057	0.024003	0.124268
Father's occupation	0.031609	0.036817	-0.030014	0.028620	-0.015477	0.016263	-0.019153	0.020626	0.050978	0.063043	...	0.000043	-0.011025	-0.014098	-0.004087	-0.030411	-0.016424	-0.007664	-0.101215	0.030550	0.131767
Admission grade	-0.004150	-0.013511	-0.096861	-0.122346	0.007077	0.183470	0.579595	0.028300	-0.055123	-0.051746	...	0.009817	0.040149	-0.042115	-0.057032	0.075867	0.073637	-0.012816	0.038058	-0.021667	-0.019759
Displaced	-0.235108	-0.301777	0.332639	-0.086908	0.252280	-0.115005	-0.010923	-0.007372	-0.081213	-0.056136	...	-0.021554	-0.091590	-0.042050	-0.038736	0.063192	0.068770	-0.035846	-0.129825	-0.011812	0.063139
Educational special needs	-0.028343	-0.030728	0.025597	-0.018908	0.031017	-0.010461	-0.001434	-0.005982	-0.021456	0.000684	...	-0.012324	-0.021671	-0.028777	-0.010851	-0.016315	-0.012761	-0.007491	0.046131	0.004396	0.012016
Debtor	0.031769	0.122540	-0.072391	-0.033096	0.006471	0.094412	-0.036509	0.052105	0.018477	-0.007375	...	0.001923	0.025592	-0.029116	0.023907	-0.146299	-0.138468	0.048706	0.021448	-0.021894	0.074489
Tuition fees up to date	-0.087158	-0.135555	0.055891	0.018675	0.038799	-0.068453	0.059856	-0.026115	-0.025979	-0.018935	...	-0.049775	0.014204	0.085918	0.063482	0.291921	0.296480	-0.071817	0.013460	-0.000706	-0.002768
Gender	-0.014738	0.159405	-0.089559	-0.100287	-0.012326	0.078684	-0.048521	-0.023258	-0.058180	-0.074129	...	-0.006302	0.018737	-0.124227	-0.041789	-0.224266	-0.199133	0.057223	0.022195	0.003556	-0.008108
Scholarship holder	-0.053708	-0.162641	0.073383	0.016499	0.093856	-0.070447	0.056210	-0.010461	0.045222	0.107104	...	-0.057739	-0.076435	0.026454	-0.021977	0.202906	0.181330	-0.048689	0.055249	-0.031148	0.035462

Diagram 25 shows the Correlation Matrix

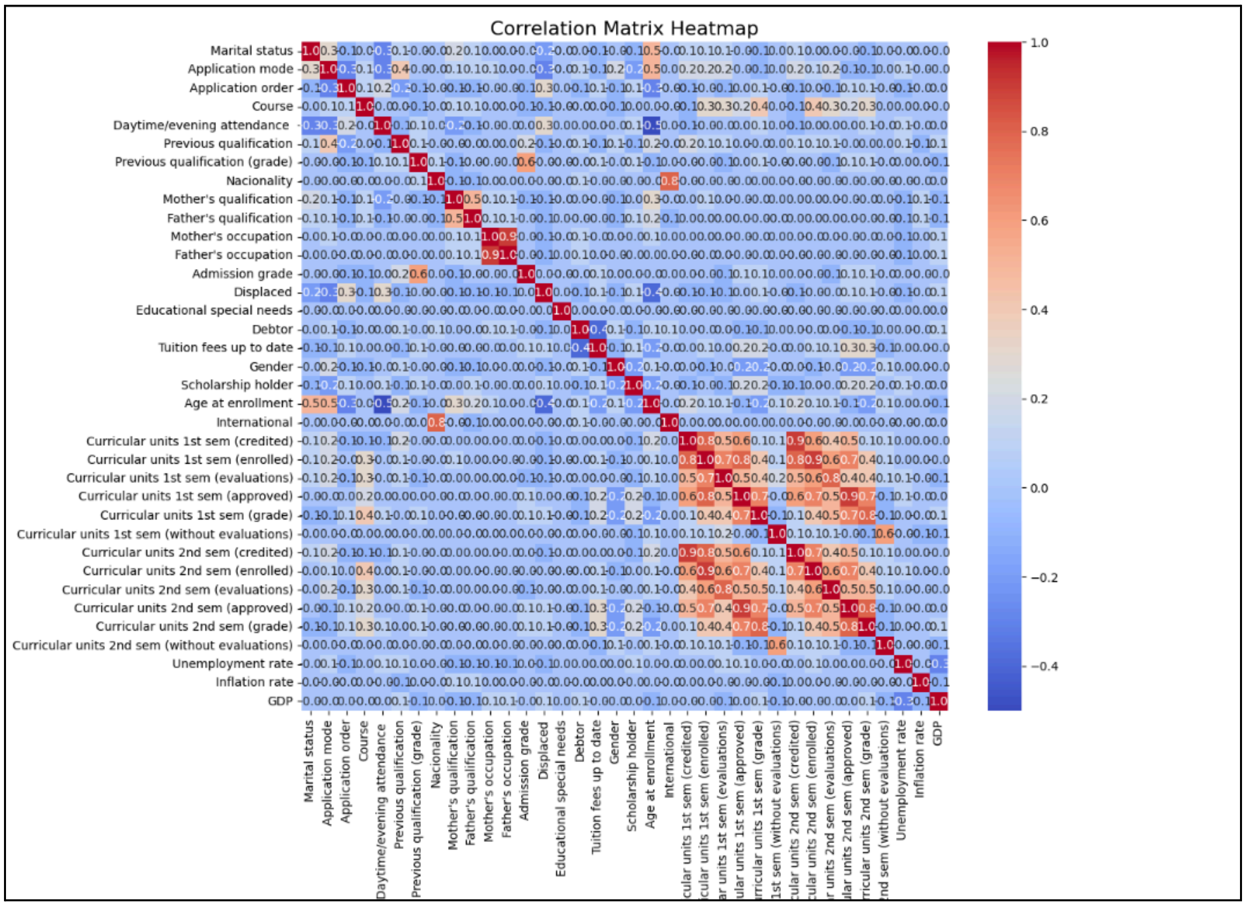
4.5 Correlation Matrix Heatmap



Correlation Matrix Heatmap

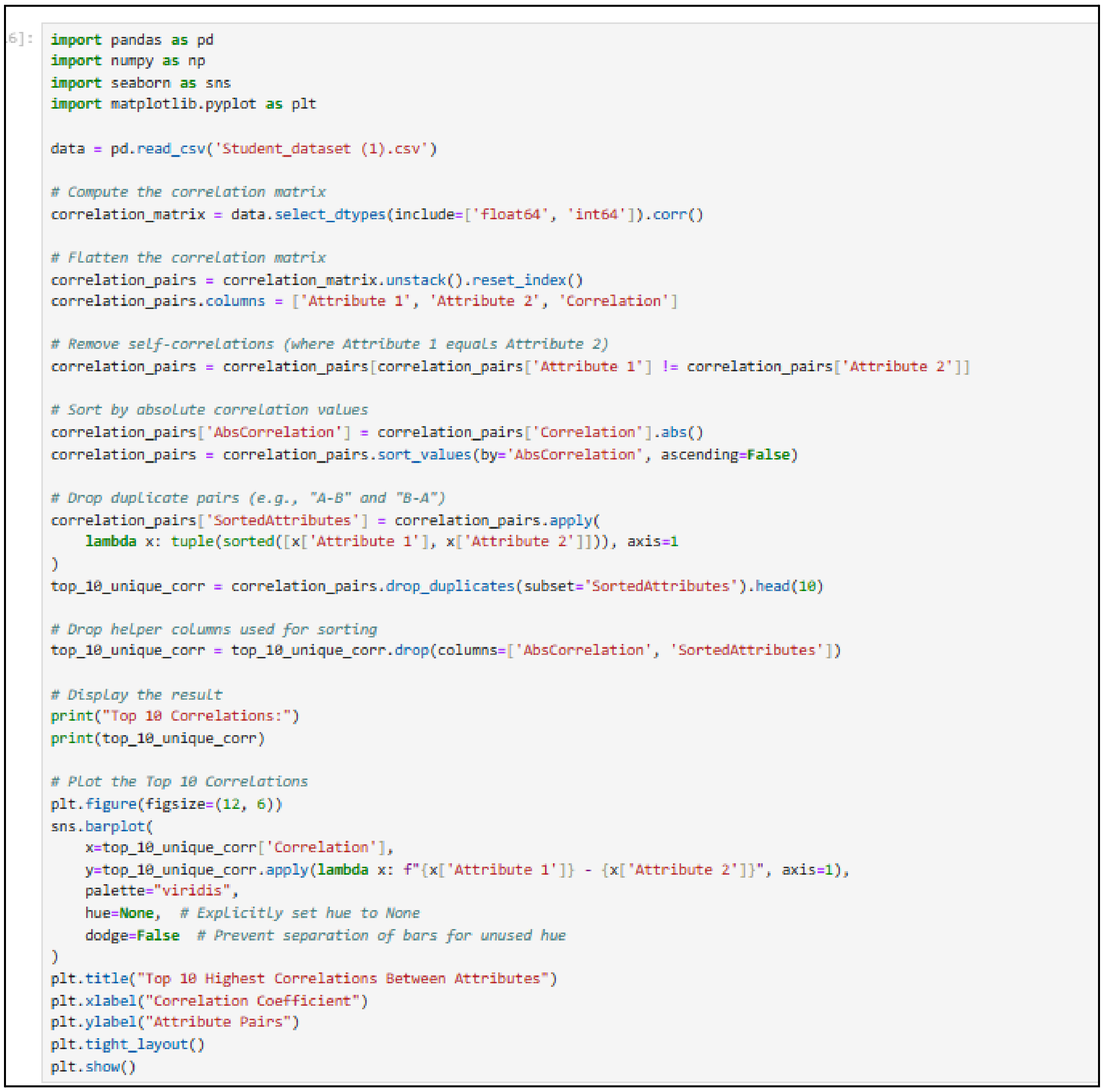
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
try:
    data = pd.read_csv('Student_dataset (1).csv')
except FileNotFoundError:
    print("Error: The file 'Student_dataset (1).csv' was not found.")
    exit()

# Clean column names to remove tab characters
data.columns = data.columns.str.replace("\t", " ")

# Select only numeric columns
numeric_data = data.select_dtypes(include=['float64', 'int64'])
if numeric_data.empty:
    print("No numeric columns found in the dataset.")
    exit()

# Compute the correlation matrix
correlation_matrix = numeric_data.corr()

# Round correlation values to 1 decimal place
correlation_matrix = correlation_matrix.round(1)

# Plot the heatmap
plt.figure(figsize=(12, 10))  # Set figure size
sns.heatmap(correlation_matrix, annot=True, fmt='.1f', cmap='coolwarm', cbar=True)  # Use 1 decimal format
plt.title('Correlation Matrix Heatmap', fontsize=16)  # Add a title
plt.show()  # Display the heatmap
```

Diagram 26 visually represents the relationships between numerical attributes in a dataset. It begins by importing the necessary libraries: Pandas for data manipulation, Matplotlib for plotting, and Seaborn for creating the heatmap. The dataset is loaded into a pandas DataFrame, with error handling to ensure the file exists. Column names are cleaned to remove tab characters, and only numerical columns (types float64 and int64) are selected for analysis. If no numerical columns are present, the script exits with a message. The correlation matrix is computed, rounding the correlation values to one decimal place for better readability. A heatmap is then plotted using Seaborn, with annotations to display the correlation values, a color map (coolwarm) for visual clarity, and a color bar for reference. The heatmap visually highlights the strength and direction of relationships (positive or negative) between attributes, making it easier to interpret correlations in the dataset.



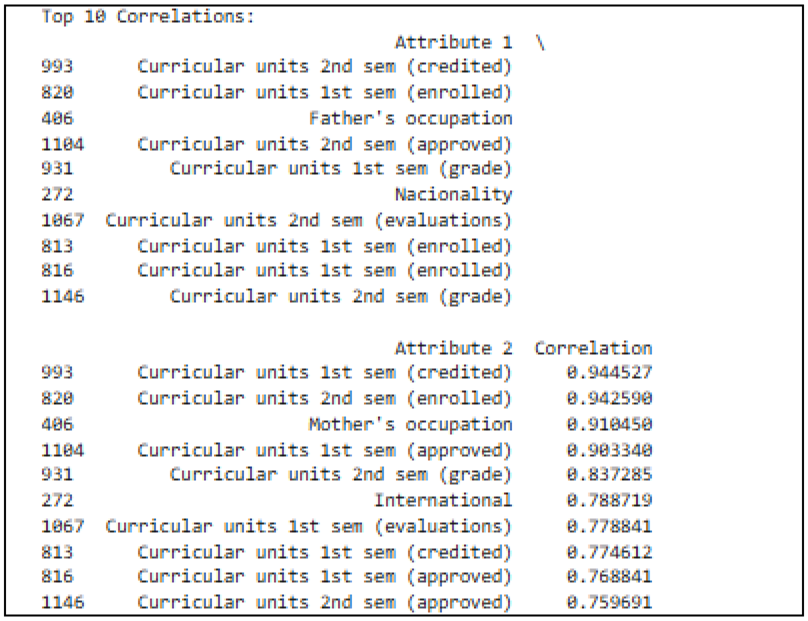
The diagram 27 show the Correlation Matrix Heatmap

4.6 Top 10 with Highest Correlation



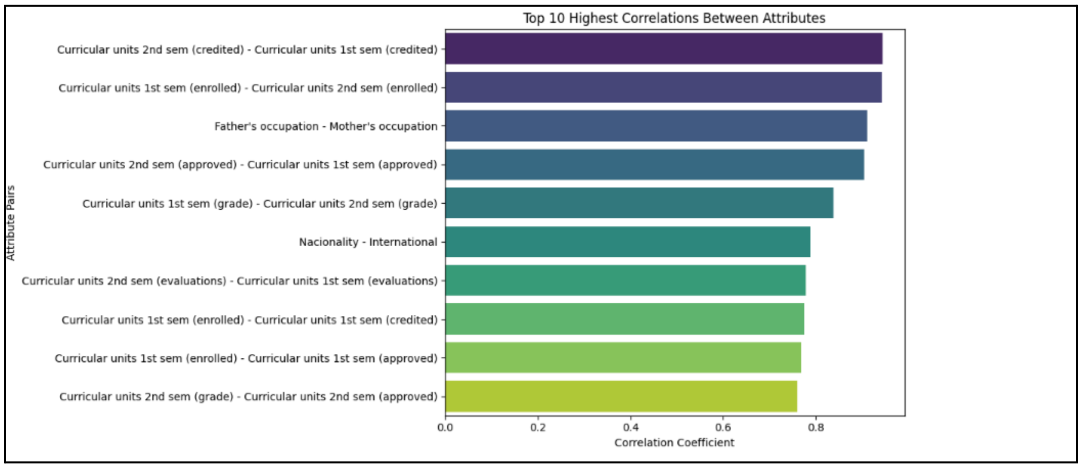
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv('Student_dataset (1).csv')

# Compute the correlation matrix
correlation_matrix = data.select_dtypes(include=['float64', 'int64']).corr()

# Flatten the correlation matrix
correlation_pairs = correlation_matrix.unstack().reset_index()
correlation_pairs.columns = ['Attribute 1', 'Attribute 2', 'Correlation']

# Remove self-correlations (where Attribute 1 equals Attribute 2)
correlation_pairs = correlation_pairs[correlation_pairs['Attribute 1'] != correlation_pairs['Attribute 2']]

# Sort by absolute correlation values
correlation_pairs['AbsCorrelation'] = correlation_pairs['Correlation'].abs()
correlation_pairs = correlation_pairs.sort_values(by='AbsCorrelation', ascending=False)

# Drop duplicate pairs (e.g., "A-B" and "B-A")
correlation_pairs['SortedAttributes'] = correlation_pairs.apply(
    lambda x: tuple(sorted([x['Attribute 1'], x['Attribute 2']])), axis=1
)
top_10_unique_corr = correlation_pairs.drop_duplicates(subset='SortedAttributes').head(10)

# Drop helper columns used for sorting
top_10_unique_corr = top_10_unique_corr.drop(columns=['AbsCorrelation', 'SortedAttributes'])

# Display the result
print("Top 10 Correlations:")
print(top_10_unique_corr)

# Plot the Top 10 Correlations
plt.figure(figsize=(12, 6))
sns.barplot(
    x=top_10_unique_corr['Correlation'],
    y=top_10_unique_corr.apply(lambda x: f"{x['Attribute 1']} - {x['Attribute 2']}", axis=1),
    palette="viridis",
    hue=None,  # Explicitly set hue to None
    dodge=False  # Prevent separation of bars for unused hue
)
plt.title("Top 10 Highest Correlations Between Attributes")
plt.xlabel("Correlation Coefficient")
plt.ylabel("Attribute Pairs")
plt.tight_layout()
plt.show()
```

The diagram 28 shows the correlation matrix is calculated for all numerical columns, showing the relationships between them. This matrix is then flattened into a list of attribute pairs with their correlation values. Self-correlations (where an attribute is correlated with itself) are removed, and absolute correlation values are calculated to consider both positive and negative correlations. The pairs are sorted by absolute correlation, and duplicate pairs (e.g., "A-B" and "B-A") are removed to retain unique attribute combinations. The top 10 unique attribute pairs with the highest correlations are selected and visualized using a horizontal bar plot, where each bar represents a pair of attributes and their correlation coefficient. This approach helps highlight the strongest relationships between numerical attributes in the dataset.



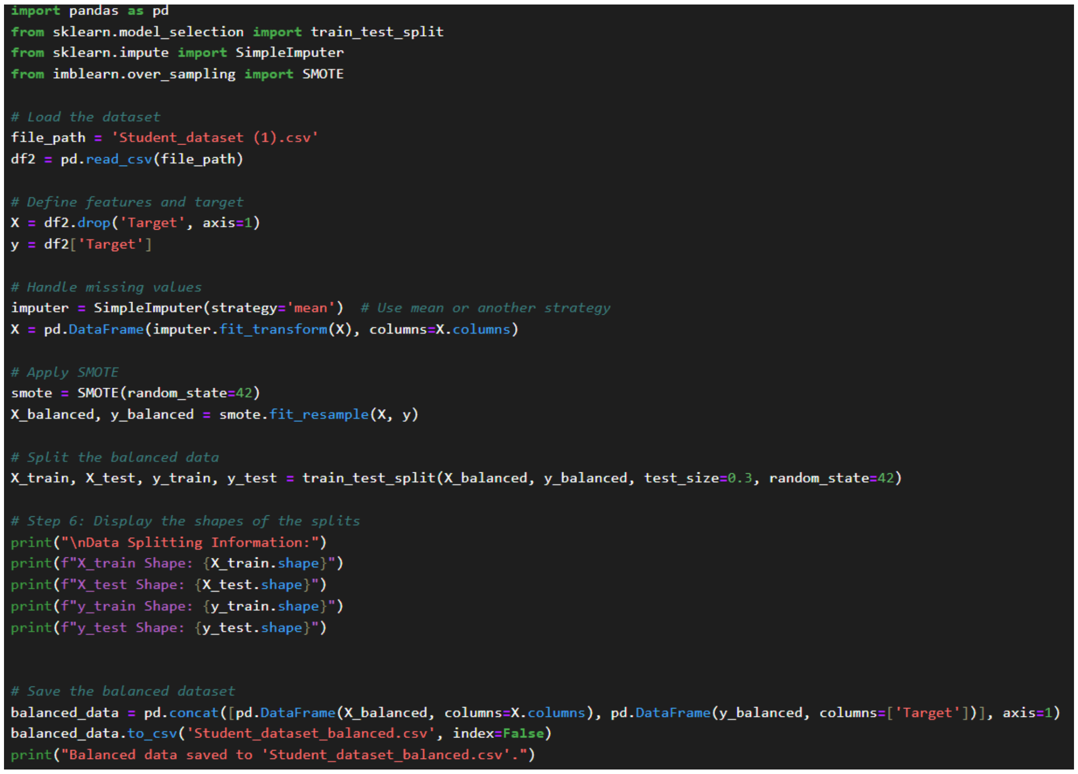
```
Top 10 Correlations:
                                    Attribute 1  \
993      Curricular units 2nd sem (credited)
820      Curricular units 1st sem (enrolled)
406                       Father's occupation
1104     Curricular units 2nd sem (approved)
931         Curricular units 1st sem (grade)
272                               Nacionality
1067  Curricular units 2nd sem (evaluations)
813      Curricular units 1st sem (enrolled)
816      Curricular units 1st sem (enrolled)
1146        Curricular units 2nd sem (grade)

                                    Attribute 2  Correlation
993      Curricular units 1st sem (credited)     0.944527
820      Curricular units 2nd sem (enrolled)     0.942590
406                       Mother's occupation     0.910450
1104     Curricular units 1st sem (approved)     0.903340
931         Curricular units 2nd sem (grade)     0.837285
272                             International     0.788719
1067  Curricular units 1st sem (evaluations)     0.778841
813      Curricular units 1st sem (credited)     0.774612
816      Curricular units 1st sem (approved)     0.768841
1146     Curricular units 2nd sem (approved)     0.759691
```

The diagram 29 shows the attributes of the top 10 correlation.



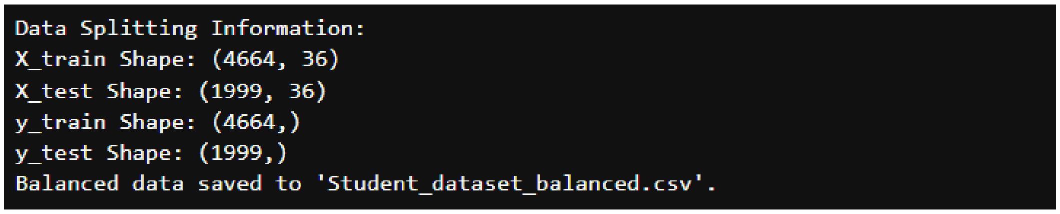
The diagram 30 shows the top 10 correlations between attributes.

4.7 Splitting Data



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE

# Load the dataset
file_path = 'Student_dataset (1).csv'
df2 = pd.read_csv(file_path)

# Define features and target
X = df2.drop('Target', axis=1)
y = df2['Target']

# Handle missing values
imputer = SimpleImputer(strategy='mean')  # Use mean or another strategy
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Apply SMOTE
smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X, y)

# Split the balanced data
X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_size=0.3, random_state=42)

# Step 6: Display the shapes of the splits
print("\nData Splitting Information:")
print(f"X_train Shape: {X_train.shape}")
print(f"X_test Shape: {X_test.shape}")
print(f"y_train Shape: {y_train.shape}")
print(f"y_test Shape: {y_test.shape}")


# Save the balanced dataset
balanced_data = pd.concat([pd.DataFrame(X_balanced, columns=X.columns), pd.DataFrame(y_balanced, columns=['Target'])], axis=1)
balanced_data.to_csv('Student_dataset_balanced.csv', index=False)
print("Balanced data saved to 'Student_dataset_balanced.csv'.")
```

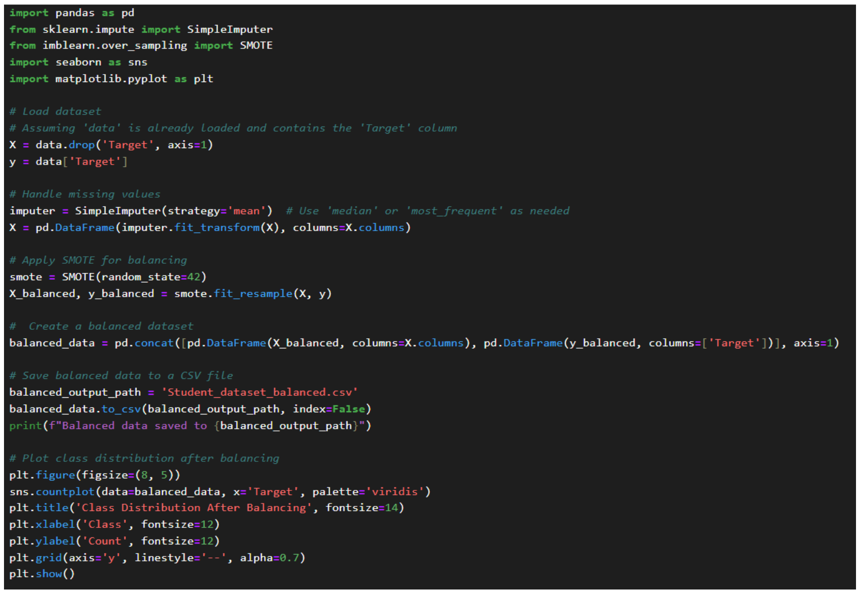
The diagram 31 illustrates the process of splitting data for classification tasks, with additional preprocessing steps to ensure data quality and balance. The dataset is first loaded and separated into features (X) and the target variable (y). Missing values in the features are addressed using mean imputation to ensure a complete dataset. To handle class imbalance in the target variable, SMOTE (Synthetic Minority Oversampling Technique) is applied, generating synthetic samples for minority classes and balancing the dataset. The balanced data is then split into training and test sets, with 70% of the data allocated for training the model (X_train and y_train) and 30% reserved for testing (X_test and y_test). This split ensures that the model is trained on a representative subset of the data and evaluated on unseen data to assess its performance. Finally, the balanced dataset is saved as Student_dataset_balanced.csv, preserving the preprocessed data for future analysis or model training.



```
Data Splitting Information:
X_train Shape: (4664, 36)
X_test Shape: (1999, 36)
y_train Shape: (4664,)
y_test Shape: (1999,)
Balanced data saved to 'Student_dataset_balanced.csv'.
```

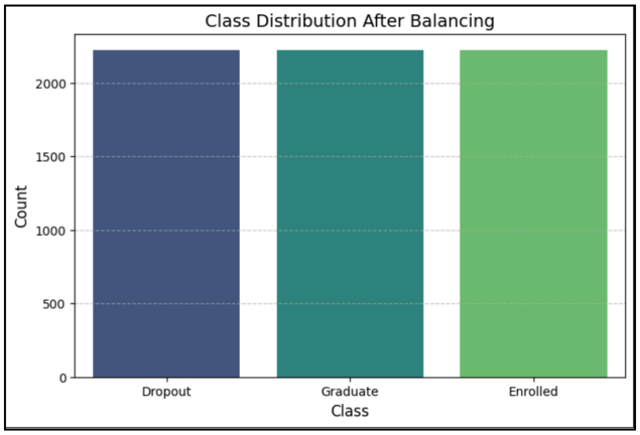
The diagram 32 shows the data splitting information such as data shape for XYtrain and XY test and saved to the 'Student_balanced_dataset.csv'

4.8 Class Distribution After Balanced



```
import pandas as pd
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
# Assuming 'data' is already loaded and contains the 'Target' column
X = data.drop('Target', axis=1)
y = data['Target']

# Handle missing values
imputer = SimpleImputer(strategy='mean')  # Use 'median' or 'most_frequent' as needed
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Apply SMOTE for balancing
smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X, y)

#  Create a balanced dataset
balanced_data = pd.concat([pd.DataFrame(X_balanced, columns=X.columns), pd.DataFrame(y_balanced, columns=['Target'])], axis=1)

# Save balanced data to a CSV file
balanced_output_path = 'Student_dataset_balanced.csv'
balanced_data.to_csv(balanced_output_path, index=False)
print(f"Balanced data saved to {balanced_output_path}")

# Plot class distribution after balancing
plt.figure(figsize=(8, 5))
sns.countplot(data=balanced_data, x='Target', palette='viridis')
plt.title('Class Distribution After Balancing', fontsize=14)
plt.xlabel('Class', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

The diagram 33 aims to handle class imbalance in the dataset and preprocess the data. First, it separates the features (X) and the target variable (y). It then handles missing values in the features using the mean imputation strategy. Afterward, it applies the SMOTE (Synthetic Minority Oversampling Technique) algorithm to balance the dataset by creating synthetic samples for the minority class. The balanced data is then combined and saved to a new CSV file.



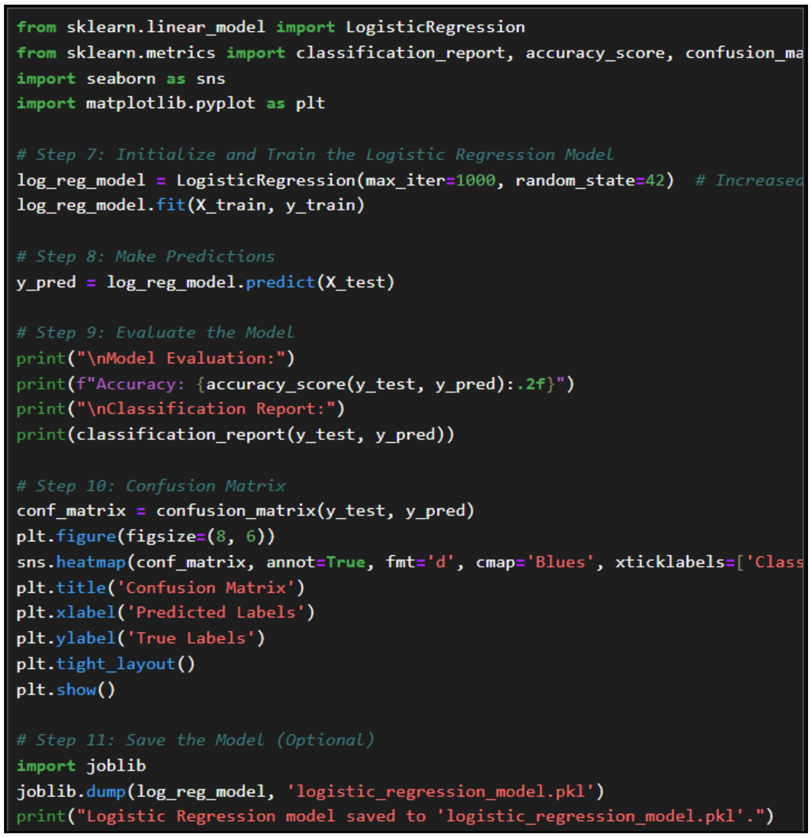
The diagram 34 shows the bar plot visualizes the class distribution after balancing for a dataset, where three classes—Dropout, Graduate, and Enrolled—are represented. The balanced distribution ensures that each class has an equal number of data points, approximately 2,000 in this case. This step is crucial in datasets where class imbalance might otherwise lead to biased models that favor the majority class. By balancing the dataset, all classes contribute equally to the training process, resulting in more reliable and fair predictions across classes.

5.0 Classifications

5.1. Implementation of Classifiers

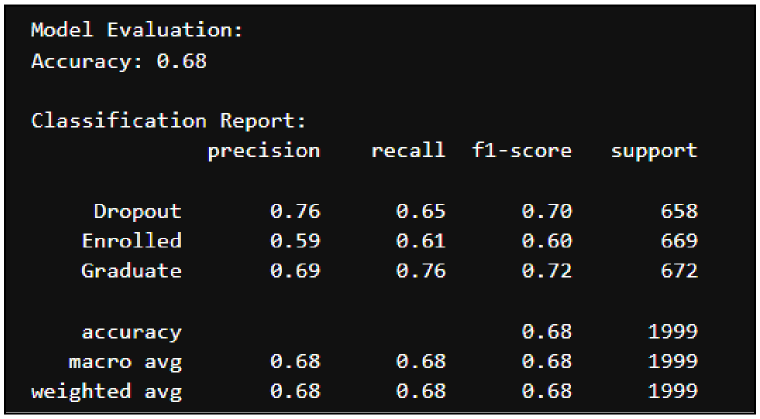
This section focuses on the implementation of classifiers, where various machine learning algorithms are applied to the Student_dataset.csv to perform classification tasks. This section outlines the process of selecting, training, and evaluating different classifiers, with an emphasis on understanding their performance metrics such as accuracy, precision, recall, and F1-score. The classifiers are trained using the training data and their effectiveness is assessed using test data, providing insights into their suitability for solving the given classification problem.

5.1.1 Logistic Regression



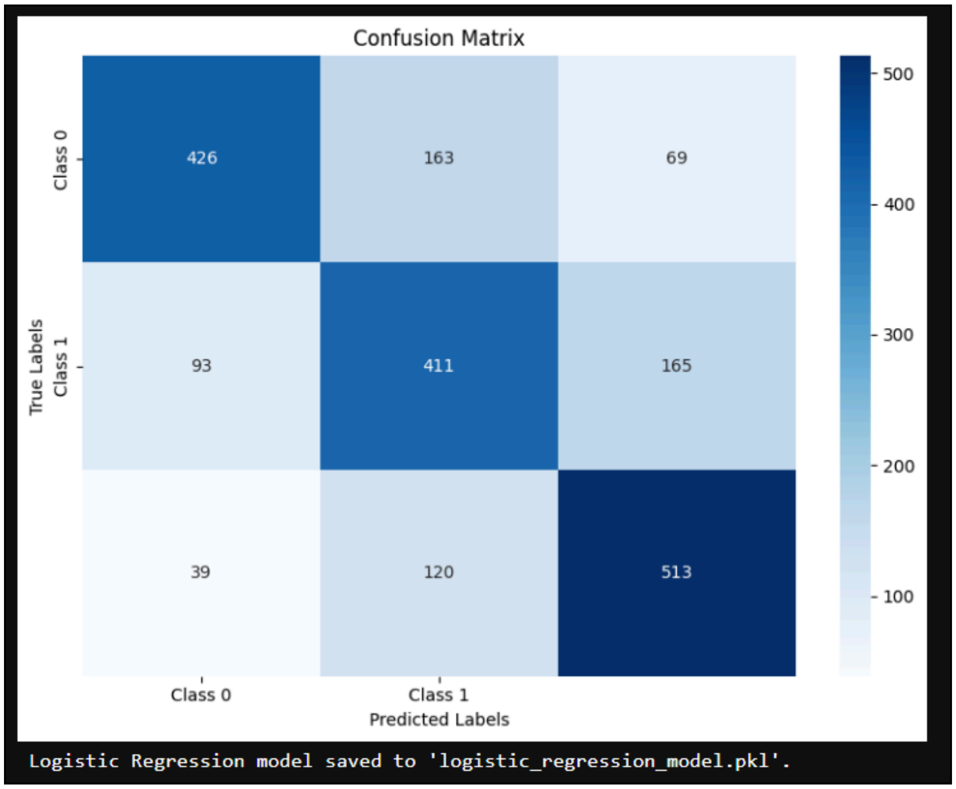
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_ma
import seaborn as sns
import matplotlib.pyplot as plt

# Step 7: Initialize and Train the Logistic Regression Model
log_reg_model = LogisticRegression(max_iter=1000, random_state=42)  # Increased
log_reg_model.fit(X_train, y_train)

# Step 8: Make Predictions
y_pred = log_reg_model.predict(X_test)

# Step 9: Evaluate the Model
print("\nModel Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Step 10: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.tight_layout()
plt.show()

# Step 11: Save the Model (Optional)
import joblib
joblib.dump(log_reg_model, 'logistic_regression_model.pkl')
print("Logistic Regression model saved to 'logistic_regression_model.pkl'.")
```

Diagram 35 illustrates the classification process using a Logistic Regression model. The process begins by initializing and training the model on the training dataset, enabling it to learn patterns for binary classification. Predictions are then made on the test dataset, and the model's performance is evaluated using accuracy and a detailed classification report that highlights metrics such as precision, recall, and F1-score. Additionally, a confusion matrix is visualized as a heatmap, providing a clear view of correct and incorrect predictions. Finally, the trained model is saved as a file (logistic_regression_model.pkl) for future use, ensuring reusability of the trained classifier.



```
Model Evaluation:
Accuracy: 0.68

Classification Report:
              precision    recall  f1-score   support

     Dropout       0.76      0.65      0.70       658
    Enrolled       0.59      0.61      0.60       669
    Graduate       0.69      0.76      0.72       672

    accuracy                           0.68      1999
   macro avg       0.68      0.68      0.68      1999
weighted avg       0.68      0.68      0.68      1999
```

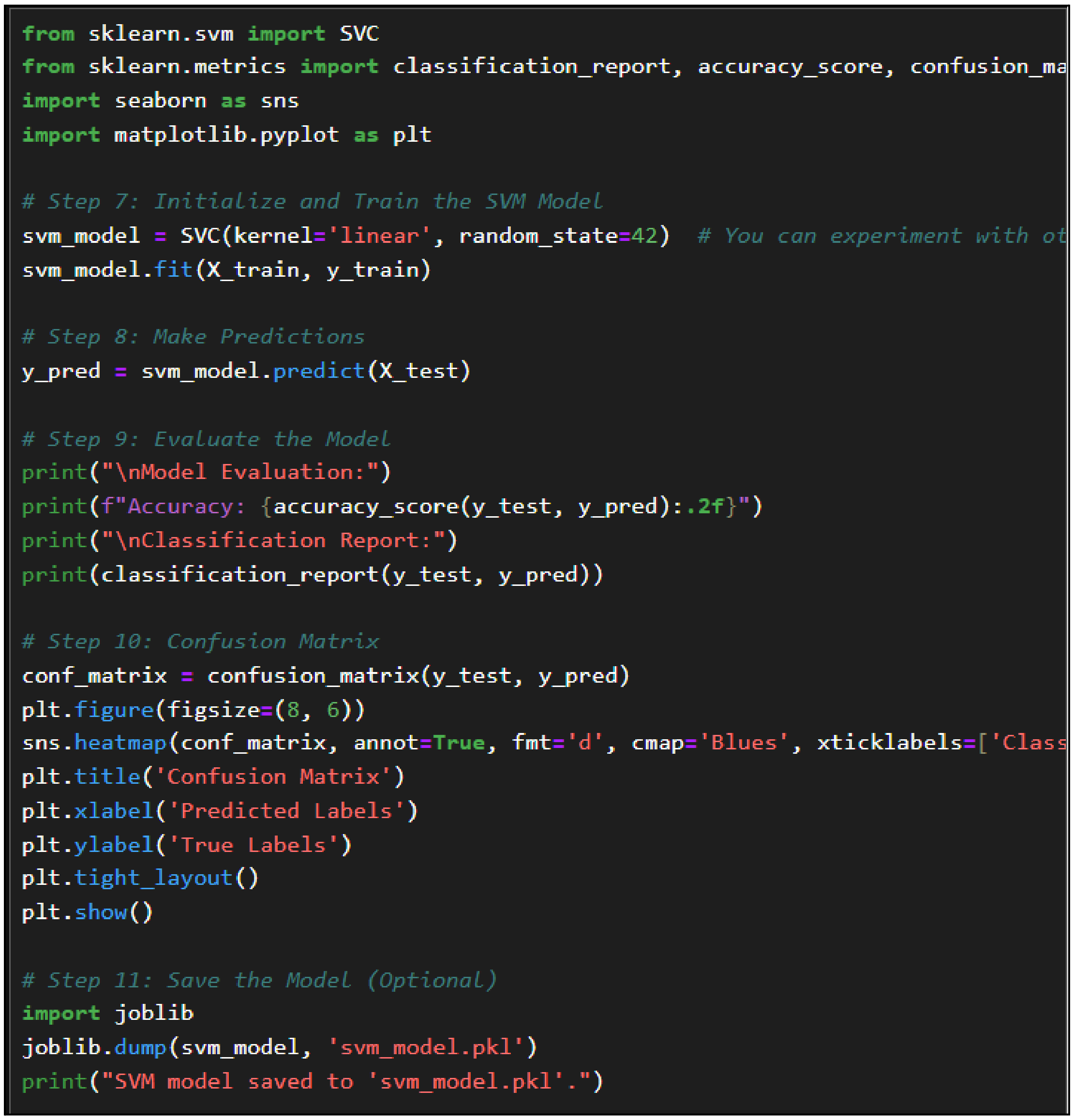
In the output, the Logistic Regression model reveals an overall accuracy of 68%, indicating that the model correctly predicts the outcomes for 68% of the dataset. The classification report provides further insights into the performance of the model for each class. For the "Dropout" class, the model demonstrates a precision of 76%, meaning it is quite reliable in identifying dropouts, and it successfully recalls 65% of the actual dropouts. In the "Enrolled" class, the precision drops to 59%, and the recall is slightly higher at 61%, suggesting the model struggles more with this category. For the "Graduate" class, the precision is 69%, and the recall is 76%, showing better performance compared to the "Enrolled" class. The macro and weighted averages of precision, recall, and F1-score are consistent at 68%, reflecting the model's moderate overall performance. These results suggest that while the model performs reasonably well, there is room for improvement, particularly in predicting the "Enrolled" class accurately.



```
Logistic Regression model saved to 'logistic_regression_model.pkl'.
```

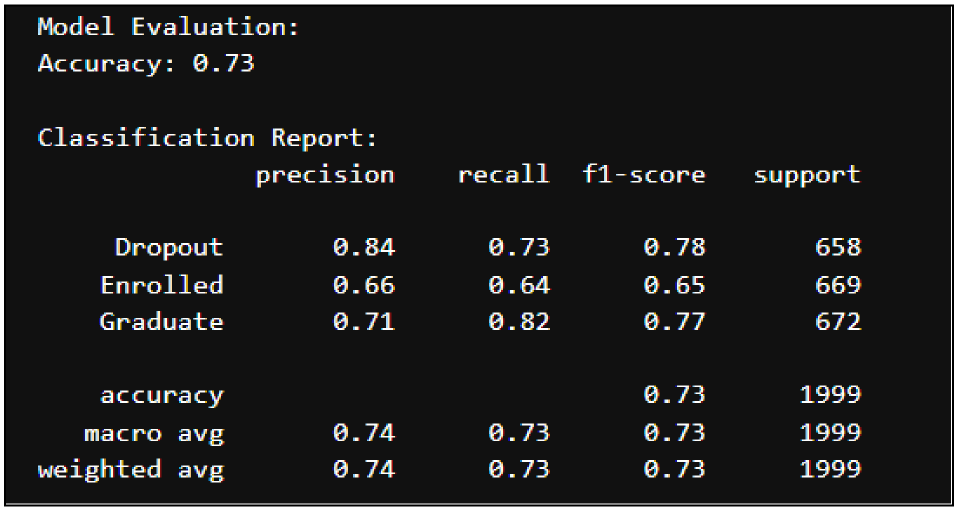
Diagram 36 shows the logistic regression model's confusion matrix indicates it performs moderately well but struggles in some areas. It correctly identified 426 instances as Class 0 (true negatives) and 411 instances as Class 1 (true positives). However, it misclassified 163 Class 0 instances as Class 1 (false positives) and 93 Class 1 instances as Class 0 (false negatives). These misclassifications highlight limitations in the model's ability to capture complex relationships in the data, as logistic regression is inherently linear. The relatively high false negatives (93) impact its recall for Class 1, while the false positives (163) affect its precision for Class 0. Overall, the model performs reasonably but may require enhancements such as feature engineering or alternative approaches to handle non-linear patterns effectively.

5.1.2 Support Vector Machines (SVM)



```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_ma
import seaborn as sns
import matplotlib.pyplot as plt

# Step 7: Initialize and Train the SVM Model
svm_model = SVC(kernel='linear', random_state=42)  # You can experiment with ot
svm_model.fit(X_train, y_train)

# Step 8: Make Predictions
y_pred = svm_model.predict(X_test)

# Step 9: Evaluate the Model
print("\nModel Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Step 10: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.tight_layout()
plt.show()

# Step 11: Save the Model (Optional)
import joblib
joblib.dump(svm_model, 'svm_model.pkl')
print("SVM model saved to 'svm_model.pkl'.")
```

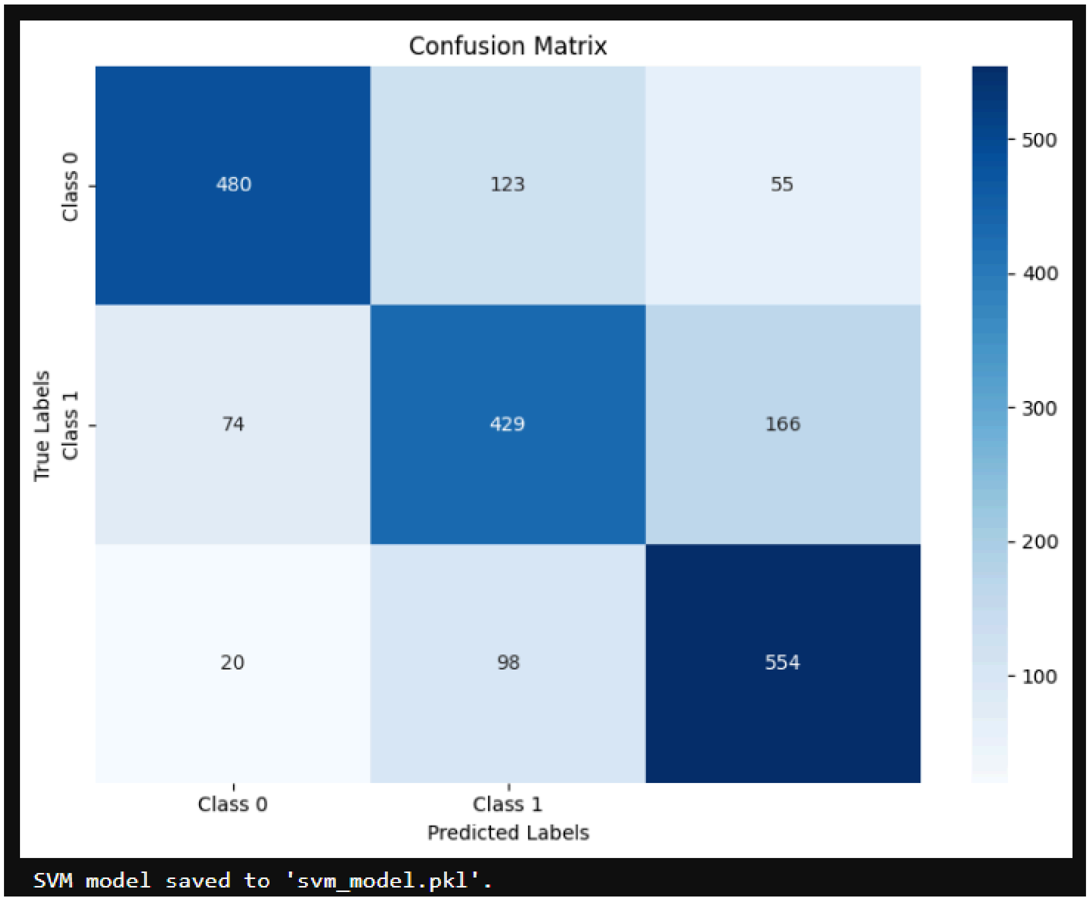
The code in the diagram above illustrates the implementation and evaluation of a Support Vector Machine (SVM) classifier with a linear kernel for classification tasks. The process starts with initializing the SVM model using the SVC class from scikit-learn. The linear kernel is chosen for simplicity, but other kernels like RBF or polynomial can be experimented with. The model is trained on the training dataset to learn decision boundaries for classifying the data.



```
Model Evaluation:
Accuracy: 0.73

Classification Report:
              precision    recall  f1-score   support

     Dropout       0.84      0.73      0.78       658
    Enrolled       0.66      0.64      0.65       669
    Graduate       0.71      0.82      0.77       672

    accuracy                           0.73      1999
   macro avg       0.74      0.73      0.73      1999
weighted avg       0.74      0.73      0.73      1999
```

After training, predictions are made on the test dataset, and the model's performance is evaluated as shown in the diagram above. The overall accuracy of the SVM model is 73%, indicating that it correctly classifies 73% of the test data. The classification report provides more detailed insights into its performance across the classes. For the "Dropout" class, the model achieves a precision of 84%, meaning most predictions for this class are correct, and a recall of 73%, showing it identifies 73% of actual dropouts. The "Enrolled" class has a lower precision of 66% and a recall of 64%, suggesting this class is more challenging for the model. For the "Graduate" class, the precision is 71%, and the recall is 82%, reflecting relatively strong performance.

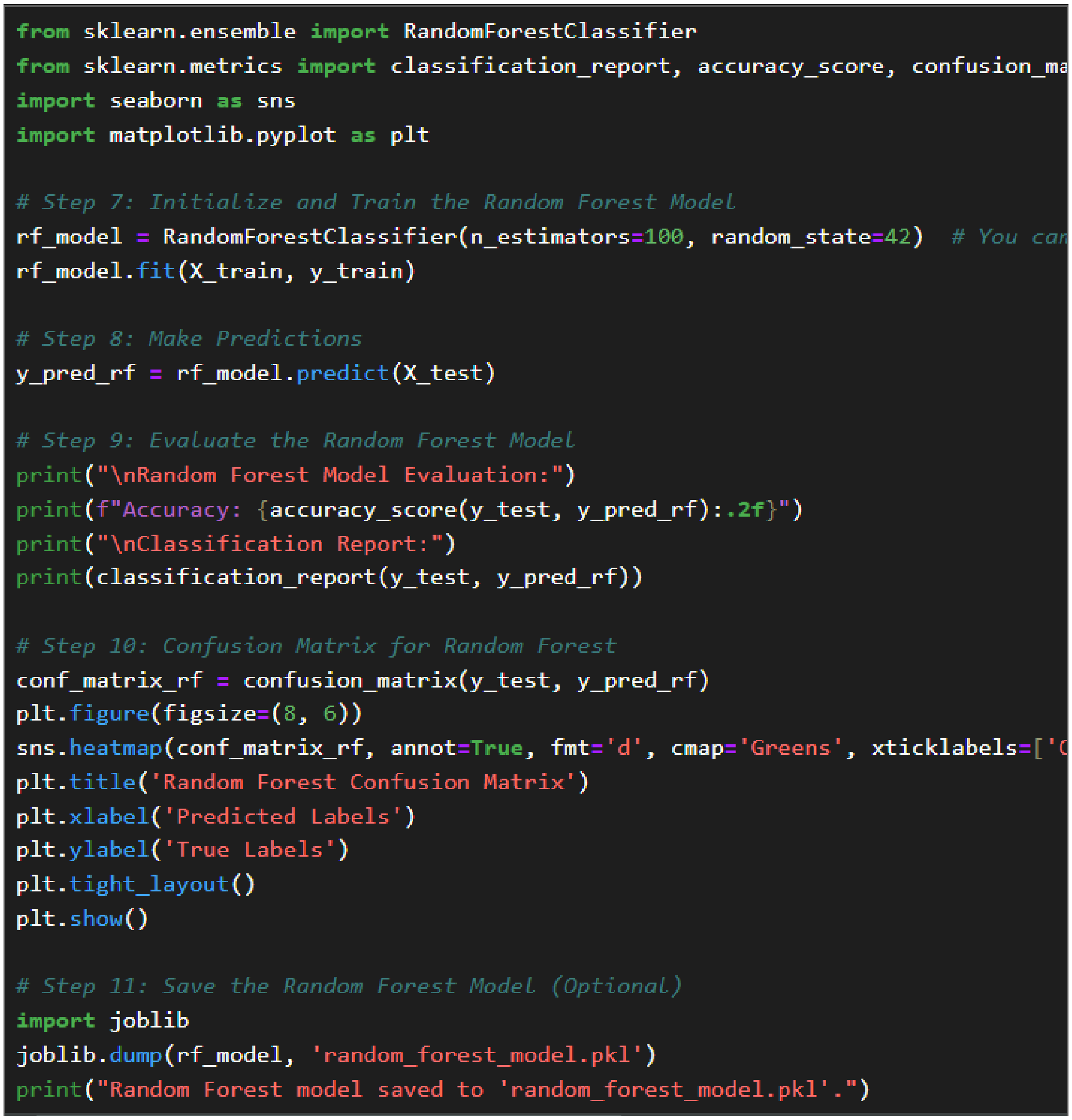
The macro and weighted averages for precision, recall, and F1-score are consistent at around 73-74%, showing the model's balanced performance across all classes. The confusion matrix visualizes the correct and incorrect predictions for each class, highlighting the model's strengths and areas for improvement. Finally, the trained SVM model is saved as svm_model.pkl for future use, allowing it to be reused without retraining.



```
SVM model saved to 'svm_model.pkl'.
```

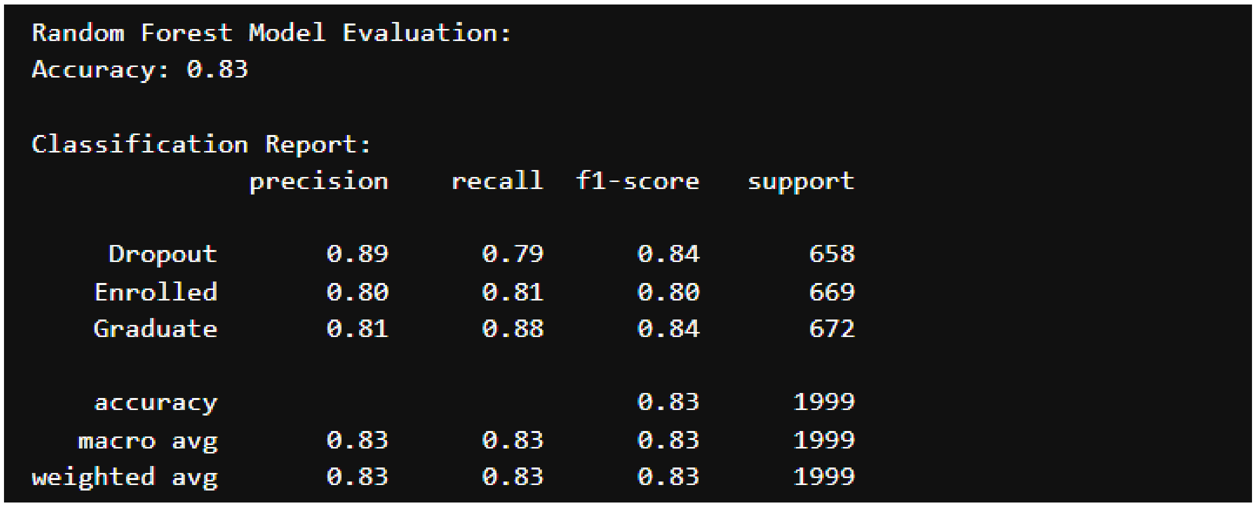
This confusion matrix evaluates the performance of an SVM (Support Vector Machine) classification model for two classes, labeled as Class 0 and Class 1. The model successfully predicted 480 true negatives (correctly classified Class 0) and 554 true positives (correctly classified Class 1). However, it made 123 false positive predictions (Class 0 misclassified as Class 1) and 74 false negative predictions (Class 1 misclassified as Class 0). These results indicate the model's ability to separate the two classes, while also highlighting areas for potential improvement in reducing misclassifications.

5.1.3 Random Forest



```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_ma
import seaborn as sns
import matplotlib.pyplot as plt

# Step 7: Initialize and Train the Random Forest Model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)  # You can
rf_model.fit(X_train, y_train)

# Step 8: Make Predictions
y_pred_rf = rf_model.predict(X_test)

# Step 9: Evaluate the Random Forest Model
print("\nRandom Forest Model Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf):.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))

# Step 10: Confusion Matrix for Random Forest
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap='Greens', xticklabels=['C
plt.title('Random Forest Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.tight_layout()
plt.show()

# Step 11: Save the Random Forest Model (Optional)
import joblib
joblib.dump(rf_model, 'random_forest_model.pkl')
print("Random Forest model saved to 'random_forest_model.pkl'.")
```

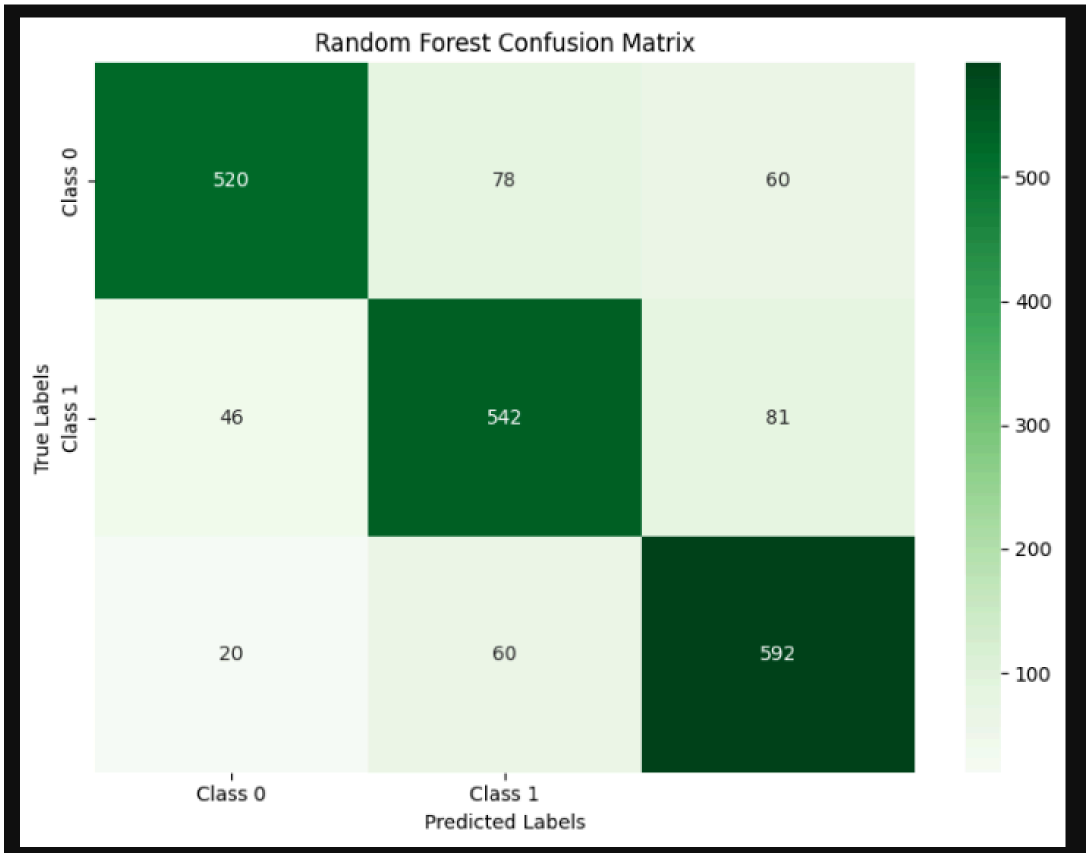
The diagram shows the implementation and evaluation of a Random Forest Classifier for classification tasks. The process begins by initializing the Random Forest model with 100 estimators and a fixed random state for reproducibility. The model is trained on the training dataset, learning patterns to classify the data accurately. After training, predictions are made on the test dataset to assess the model's performance.



```
Random Forest Model Evaluation:
Accuracy: 0.83

Classification Report:
              precision    recall  f1-score   support

     Dropout       0.89      0.79      0.84       658
    Enrolled       0.80      0.81      0.80       669
    Graduate       0.81      0.88      0.84       672

    accuracy                           0.83      1999
   macro avg       0.83      0.83      0.83      1999
weighted avg       0.83      0.83      0.83      1999
```

The evaluation results reveal an overall accuracy of 83%, indicating that the Random Forest model performs well on the test dataset. The classification report provides further insights into its performance across classes. For the "Dropout" class, the model achieves a precision of 89%, meaning it reliably predicts dropouts, and a recall of 79%, showing it identifies 79% of actual dropouts. For the "Enrolled" class, the precision is 80%, and the recall is 81%, reflecting a balanced performance. For the "Graduate" class, the model performs strongly with an 81% precision and an 88% recall.

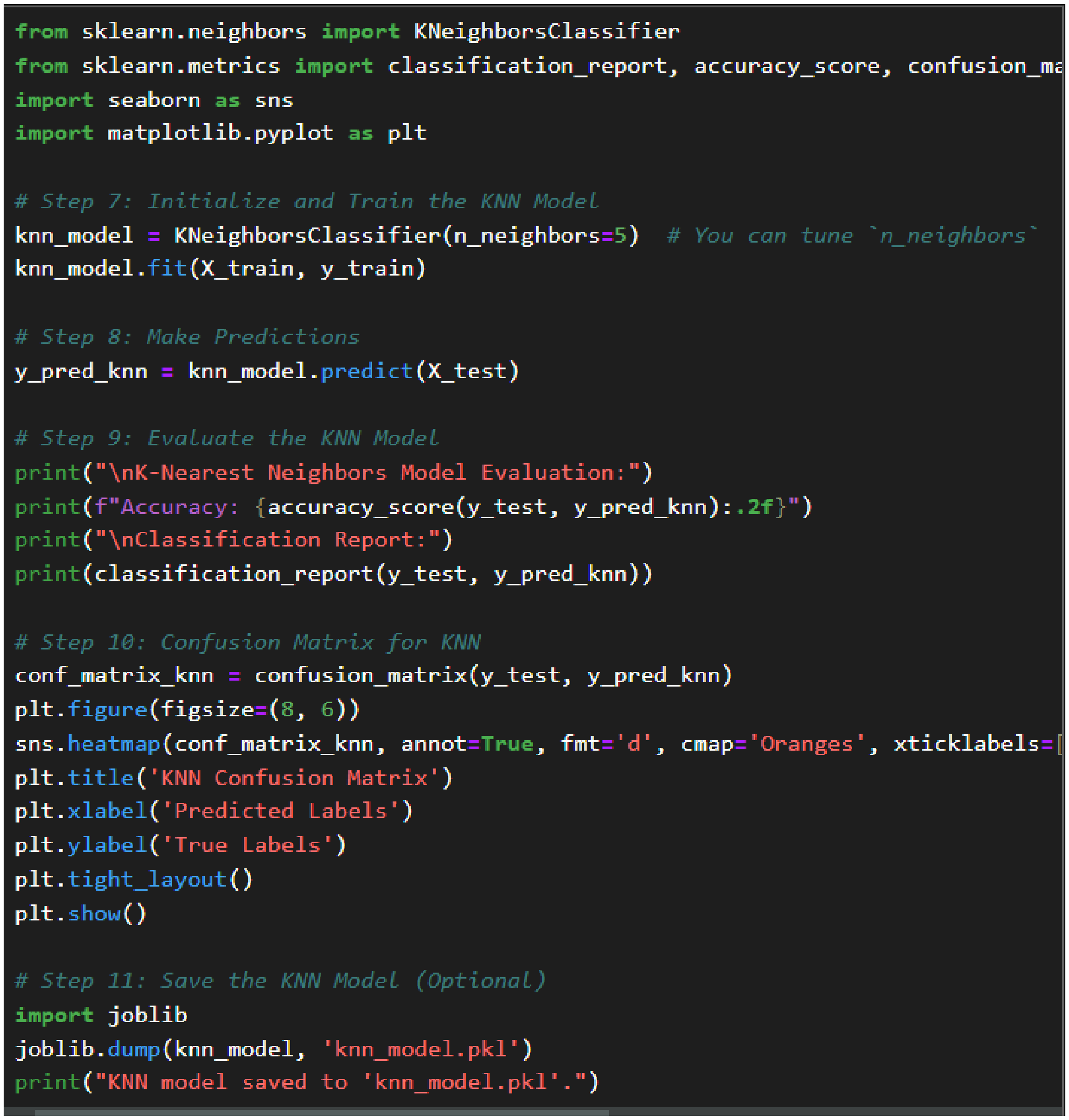
The macro and weighted averages for precision, recall, and F1-score are consistent at 83%, showcasing the model's robustness across all classes. A confusion matrix visualizes the correct and incorrect predictions, providing additional insights into the model's strengths and weaknesses. Finally, the trained Random Forest model is saved to a file (random_forest_model.pkl) for future use, ensuring the classifier can be reused without retraining.



```
Random Forest model saved to 'random_forest_model.pkl'.
```

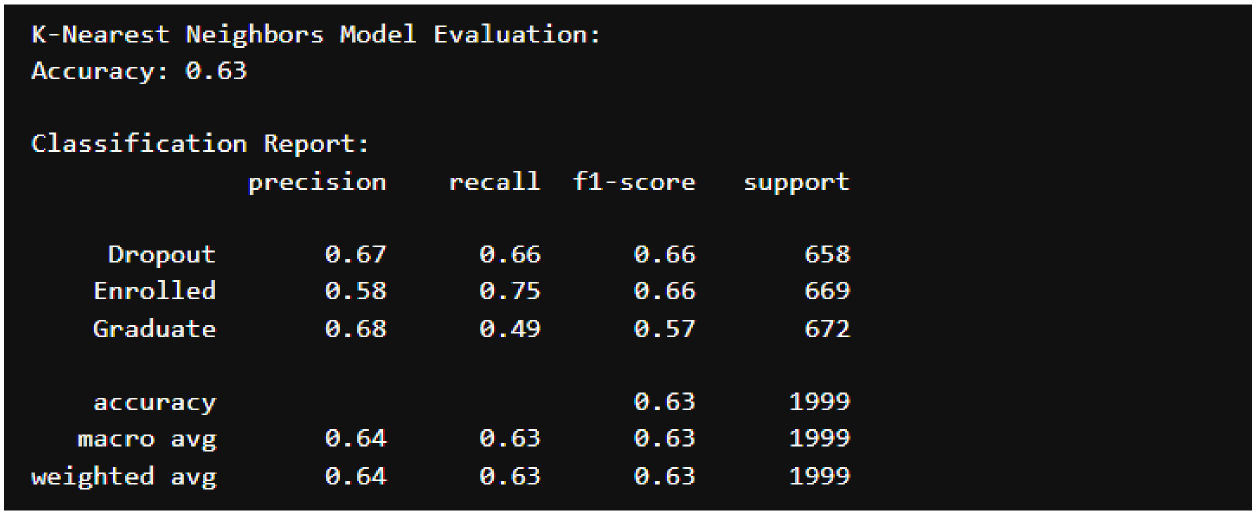
The random forest model demonstrates significantly better performance compared to logistic regression. It correctly classified 520 instances as Class 0 (true negatives) and 542 instances as Class 1 (true positives), showing improved accuracy for both classes. The number of misclassifications was much lower, with only 78 false positives (Class 0 misclassified as Class 1) and 46 false negatives (Class 1 misclassified as Class 0). This improvement highlights the random forest model's ability to capture both linear and non-linear relationships in the data due to its ensemble nature. By reducing false positives and false negatives, the random forest achieves higher precision and recall, making it more effective for this dataset. Overall, it outperforms logistic regression by handling complex data patterns more effectively and providing better classification results.

5.1.4 K-Nearest Neighbours



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_ma
import seaborn as sns
import matplotlib.pyplot as plt

# Step 7: Initialize and Train the KNN Model
knn_model = KNeighborsClassifier(n_neighbors=5)  # You can tune `n_neighbors`
knn_model.fit(X_train, y_train)

# Step 8: Make Predictions
y_pred_knn = knn_model.predict(X_test)

# Step 9: Evaluate the KNN Model
print("\nK-Nearest Neighbors Model Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_knn):.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_knn))

# Step 10: Confusion Matrix for KNN
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_knn, annot=True, fmt='d', cmap='Oranges', xticklabels=[
plt.title('KNN Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.tight_layout()
plt.show()

# Step 11: Save the KNN Model (Optional)
import joblib
joblib.dump(knn_model, 'knn_model.pkl')
print("KNN model saved to 'knn_model.pkl'.")
```

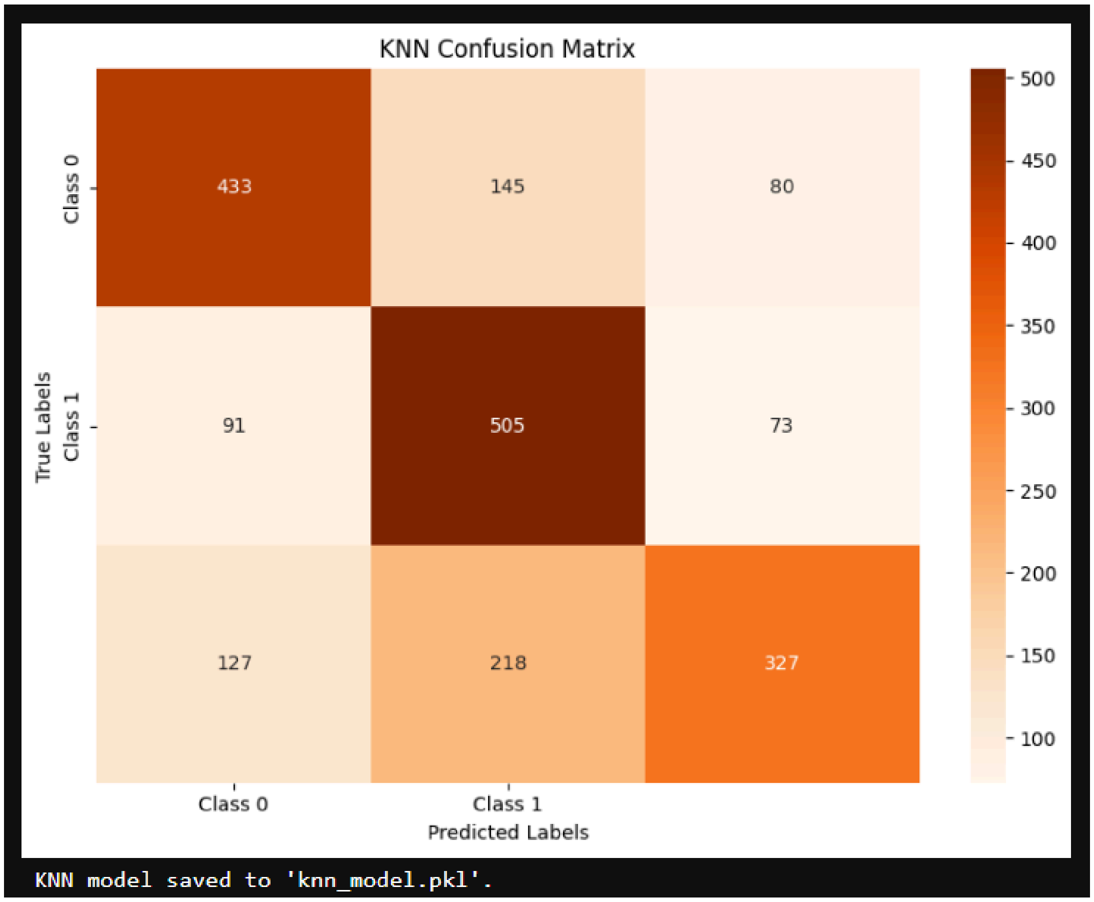
The diagram demonstrates the implementation and evaluation of a K-Nearest Neighbors (KNN) classifier for classification tasks. The process begins with initializing the KNN model using the KNeighborsClassifier class from scikit-learn, with the number of neighbors (n_neighbors) set to 5. The model is trained on the training dataset, where it learns to classify based on the majority vote of the nearest neighbors in feature space.



```
K-Nearest Neighbors Model Evaluation:
Accuracy: 0.63

Classification Report:
              precision    recall  f1-score   support

     Dropout       0.67      0.66      0.66       658
    Enrolled       0.58      0.75      0.66       669
    Graduate       0.68      0.49      0.57       672

    accuracy                           0.63      1999
   macro avg       0.64      0.63      0.63      1999
weighted avg       0.64      0.63      0.63      1999
```

After training, predictions are made on the test dataset, and the model's performance is evaluated. The overall accuracy of the KNN model is 63%, indicating that it correctly predicts the outcome for 63% of the test samples. The classification report provides a breakdown of performance for each class. For the "Dropout" class, the model achieves a precision of 67% and a recall of 66%, showing relatively balanced performance. The "Enrolled" class has a precision of 58% but a higher recall of 75%, suggesting the model is better at identifying enrolled students than avoiding false positives. For the "Graduate" class, the precision is 68%, but the recall drops to 49%, indicating challenges in correctly identifying graduates.

The macro and weighted averages for precision, recall, and F1-score are consistent at 63-64%, reflecting moderate performance. The confusion matrix is visualized as a heatmap, providing a detailed view of the model's correct and incorrect predictions for each class. Finally, the trained KNN model is saved as knn_model.pkl for future use, ensuring reusability without retraining.

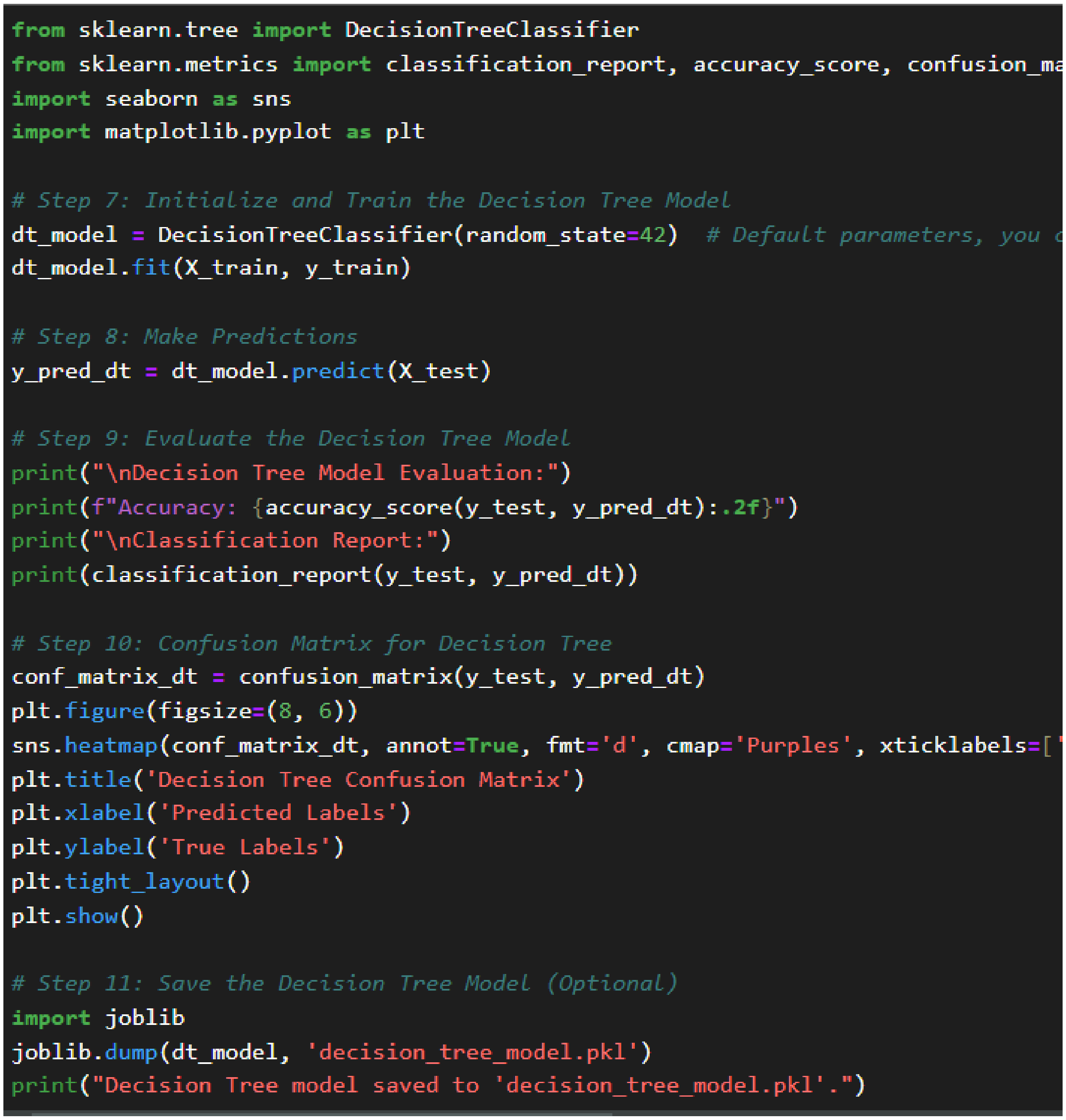


```
KNN model saved to 'knn_model.pkl'.
```

The KNN (K-Nearest Neighbors) model demonstrates moderate classification accuracy in the given task. It correctly classifies 433 instances as Dropout, 505 as Enrolled, and 327 as Graduate. However, it exhibits significant misclassifications, which suggest challenges in capturing the data's structure. For example, 127 Graduate instances are misclassified as Dropout, and 218 Graduate instances are misclassified as Enrolled. Additionally, 145 Dropouts are incorrectly predicted as Enrolled, and 91 Enrolled cases are labeled as Dropout. This indicates that KNN's performance may be influenced by factors such as feature scaling, the choice of the number of neighbors (k), and overlapping feature distributions among the classes.

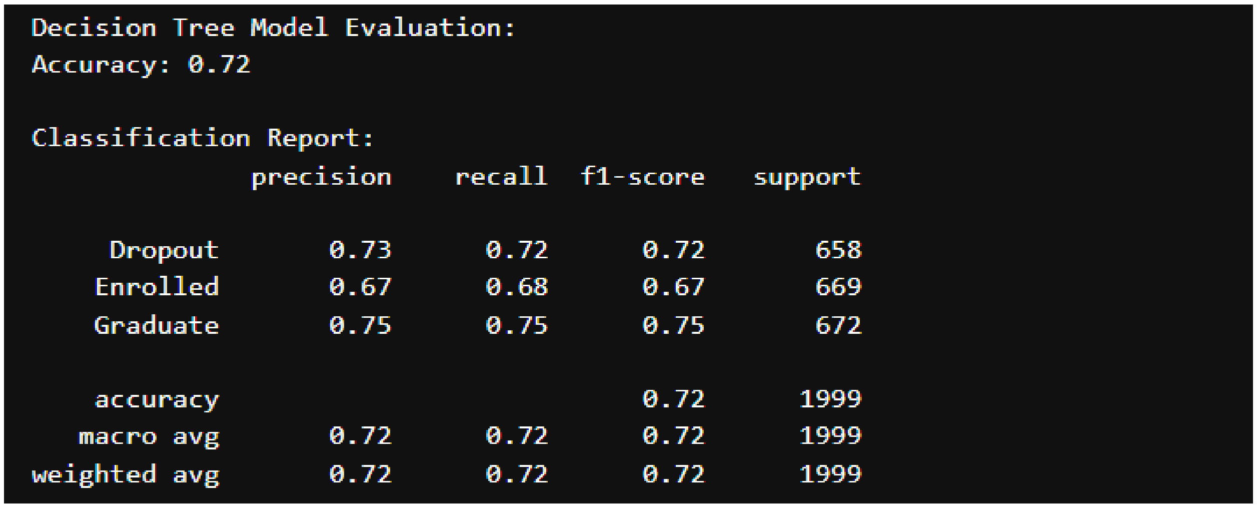
The high misclassification rates highlight the importance of careful preprocessing (e.g., normalization) and hyperparameter tuning for KNN. Despite its simplicity and interpretability, KNN's reliance on distance-based predictions can make it less effective when class boundaries are complex or when the dataset is noisy or imbalanced.

5.1.5 Decision Tree



```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_ma
import seaborn as sns
import matplotlib.pyplot as plt

# Step 7: Initialize and Train the Decision Tree Model
dt_model = DecisionTreeClassifier(random_state=42)  # Default parameters, you c
dt_model.fit(X_train, y_train)

# Step 8: Make Predictions
y_pred_dt = dt_model.predict(X_test)

# Step 9: Evaluate the Decision Tree Model
print("\nDecision Tree Model Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_dt):.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_dt))

# Step 10: Confusion Matrix for Decision Tree
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_dt, annot=True, fmt='d', cmap='Purples', xticklabels=['
plt.title('Decision Tree Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.tight_layout()
plt.show()

# Step 11: Save the Decision Tree Model (Optional)
import joblib
joblib.dump(dt_model, 'decision_tree_model.pkl')
print("Decision Tree model saved to 'decision_tree_model.pkl'.")
```

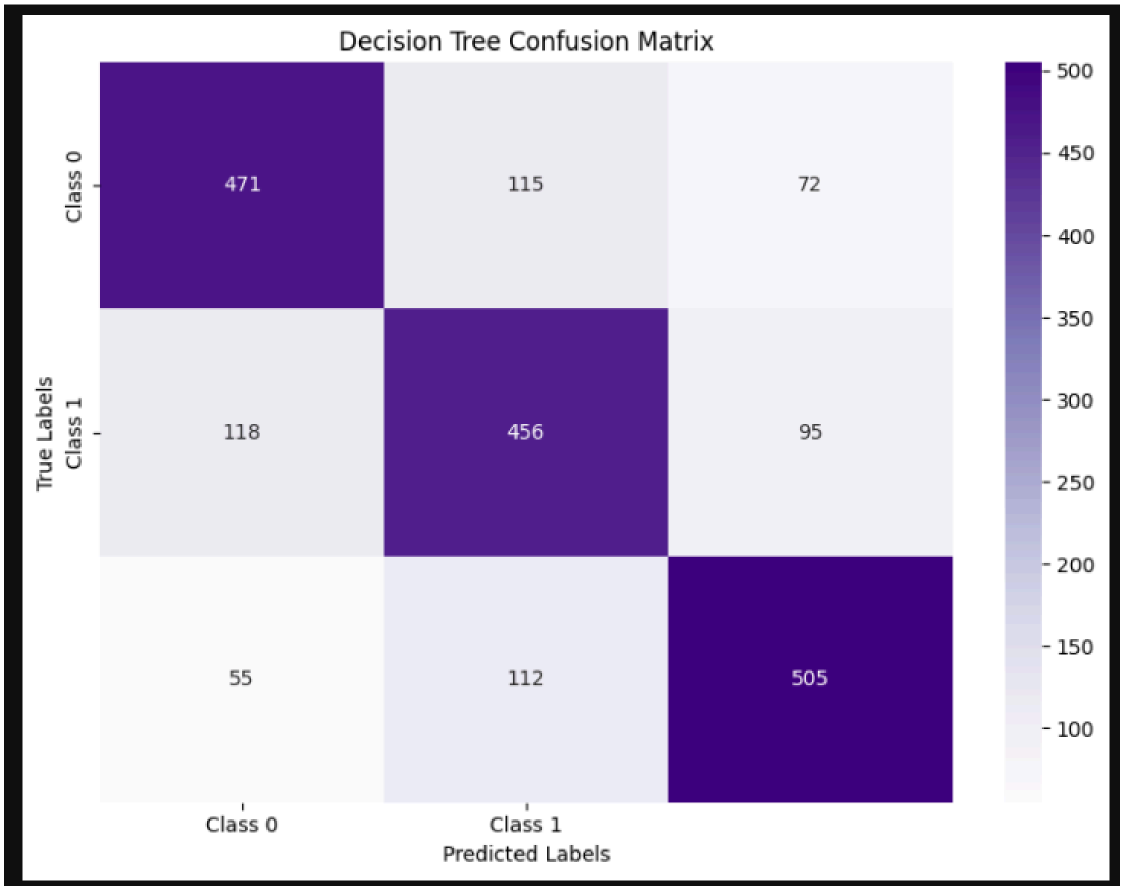
The codes in the diagram illustrate the process of implementing and evaluating a Decision Tree classifier for classification tasks. The process starts by initializing the DecisionTreeClassifier model from scikit-learn with a random state set to 42 for reproducibility. The model is then trained on the training dataset (X_train and y_train).



```
Decision Tree Model Evaluation:
Accuracy: 0.72

Classification Report:
              precision    recall  f1-score   support

     Dropout       0.73      0.72      0.72       658
    Enrolled       0.67      0.68      0.67       669
    Graduate       0.75      0.75      0.75       672

    accuracy                           0.72      1999
   macro avg       0.72      0.72      0.72      1999
weighted avg       0.72      0.72      0.72      1999
```

After training the data, the output of the Decision Tree model evaluation shows that the model has an accuracy of 0.72, meaning it correctly predicted 72% of the test data. The classification report provides a detailed breakdown of the model's performance for each class. For the "Dropout" class, the model achieved a precision of 0.73, recall of 0.72, and an F1-score of 0.72 based on 658 instances. For the "Enrolled" class, the model had a precision of 0.67, recall of 0.68, and an F1-score of 0.67 from 669 instances. For the "Graduate" class, the model performed the best with a precision of 0.75, recall of 0.75, and an F1-score of 0.75 based on 672 instances.

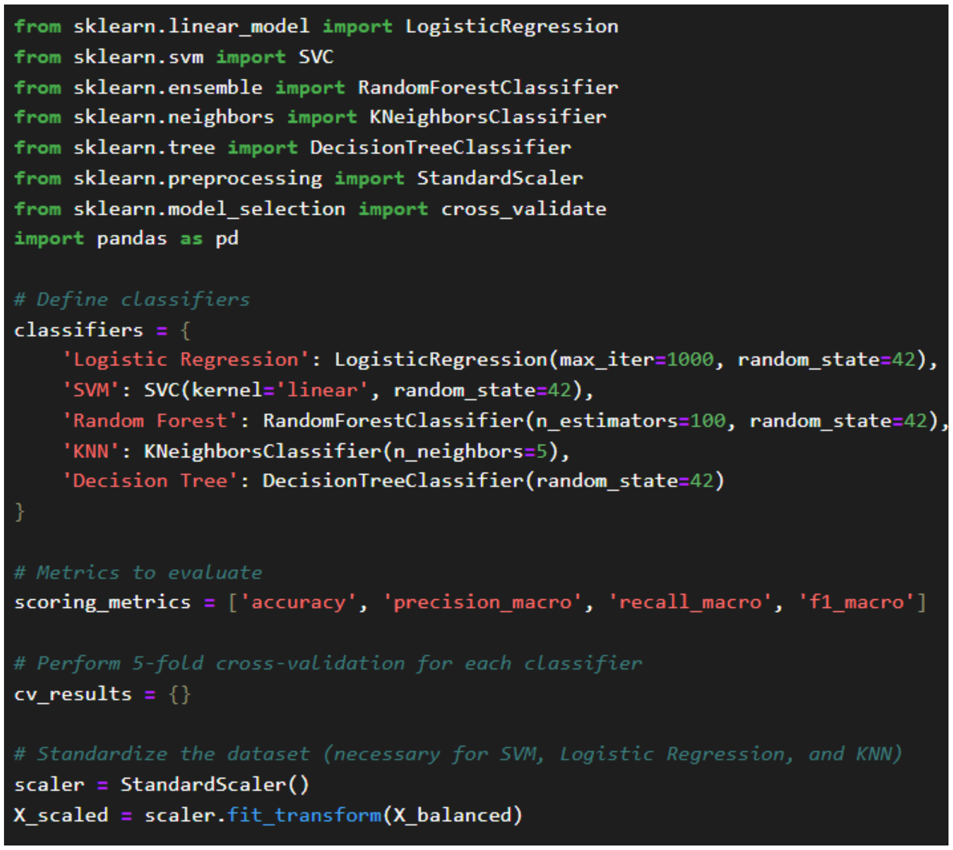
The macro average, which calculates the mean of the metrics across all classes without considering class imbalance, gives a precision, recall, and F1-score of 0.72 for all metrics. Similarly, the weighted average, which accounts for the number of instances in each class, also results in a precision, recall, and F1-score of 0.72, indicating overall balanced performance across the classes.



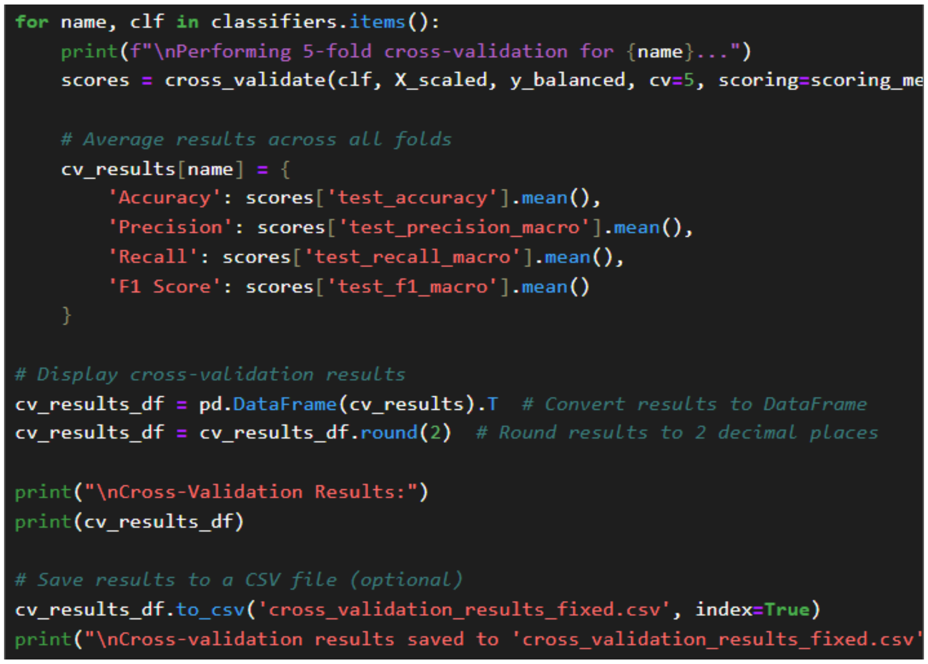
```
Decision Tree model saved to 'decision_tree_model.pkl'.
```

This matrix evaluates a standalone Decision Tree model's predictions for two classes, "Class 0" and "Class 1." The model demonstrates balanced performance, correctly identifying 471 "Class 0" and 505 "Class 1" instances. However, some errors occur, with 115 "Class 0" instances misclassified as "Class 1," and 72 as "Class 1." Similarly, 118 "Class 1" instances are mislabeled as "Class 0," and 95 as "Class 1." Although the Decision Tree performs decently, it shows slightly more misclassifications compared to ensemble methods like Bagging and AdaBoost, highlighting the benefits of combining models for better accuracy.

5.2 Cross-validation and Evaluation

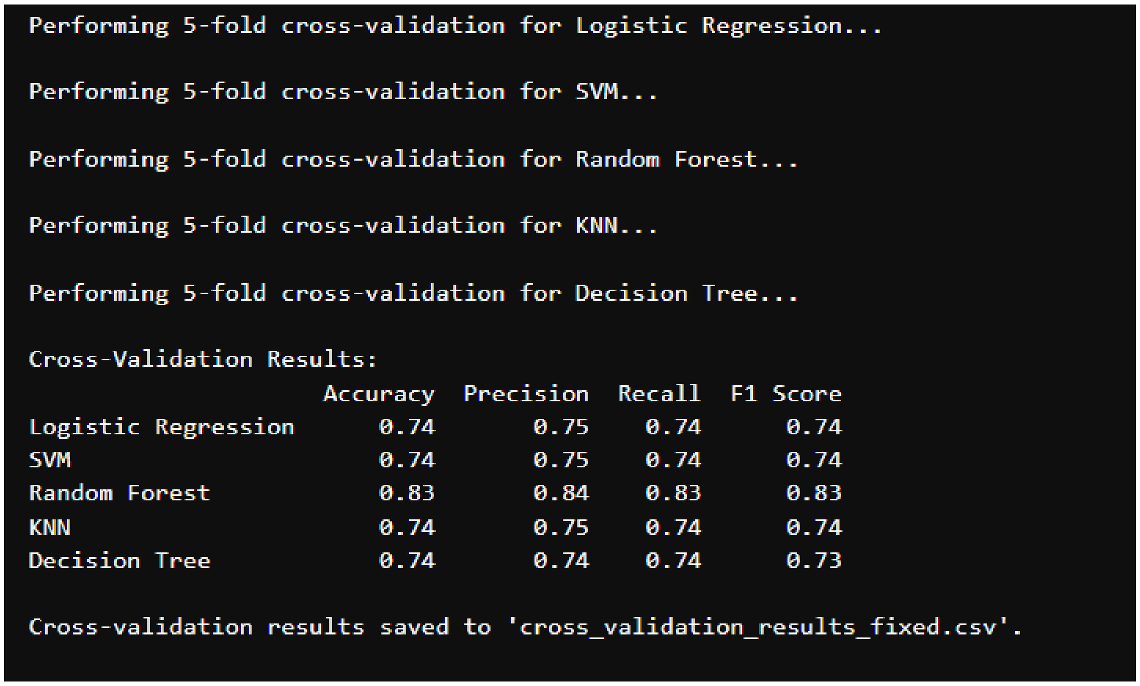


```python
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate
import pandas as pd

# Define classifiers
classifiers = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'SVM': SVC(kernel='linear', random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=5),
    'Decision Tree': DecisionTreeClassifier(random_state=42)
}

# Metrics to evaluate
scoring_metrics = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']

# Perform 5-fold cross-validation for each classifier
cv_results = {}

# Standardize the dataset (necessary for SVM, Logistic Regression, and KNN)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_balanced)
```



```python
for name, clf in classifiers.items():
    print(f"\nPerforming 5-fold cross-validation for {name}...")
    scores = cross_validate(clf, X_scaled, y_balanced, cv=5, scoring=scoring_me

    # Average results across all folds
    cv_results[name] = {
        'Accuracy': scores['test_accuracy'].mean(),
        'Precision': scores['test_precision_macro'].mean(),
        'Recall': scores['test_recall_macro'].mean(),
        'F1 Score': scores['test_f1_macro'].mean()
    }

# Display cross-validation results
cv_results_df = pd.DataFrame(cv_results).T  # Convert results to DataFrame
cv_results_df = cv_results_df.round(2)  # Round results to 2 decimal places

print("\nCross-Validation Results:")
print(cv_results_df)

# Save results to a CSV file (optional)
cv_results_df.to_csv('cross_validation_results_fixed.csv', index=True)
print("\nCross-validation results saved to 'cross_validation_results_fixed.csv'
```

The code implements a cross-validation process for evaluating the performance of five different classifiers: Logistic Regression, SVM, Random Forest, KNN, and Decision Tree. It uses 5-fold cross-validation, where the dataset is split into five subsets, and each classifier is trained and evaluated on different folds. The evaluation metrics used are accuracy, precision (macro), recall (macro), and F1 score (macro). The dataset is first standardized using StandardScaler, which is necessary for classifiers like Logistic Regression, SVM, and KNN.



```
Performing 5-fold cross-validation for Logistic Regression...

Performing 5-fold cross-validation for SVM...

Performing 5-fold cross-validation for Random Forest...

Performing 5-fold cross-validation for KNN...

Performing 5-fold cross-validation for Decision Tree...

Cross-Validation Results:
                     Accuracy  Precision  Recall  F1 Score
Logistic Regression      0.74       0.75    0.74      0.74
SVM                      0.74       0.75    0.74      0.74
Random Forest            0.83       0.84    0.83      0.83
KNN                      0.74       0.75    0.74      0.74
Decision Tree            0.74       0.74    0.74      0.73

Cross-validation results saved to 'cross_validation_results_fixed.csv'.
```

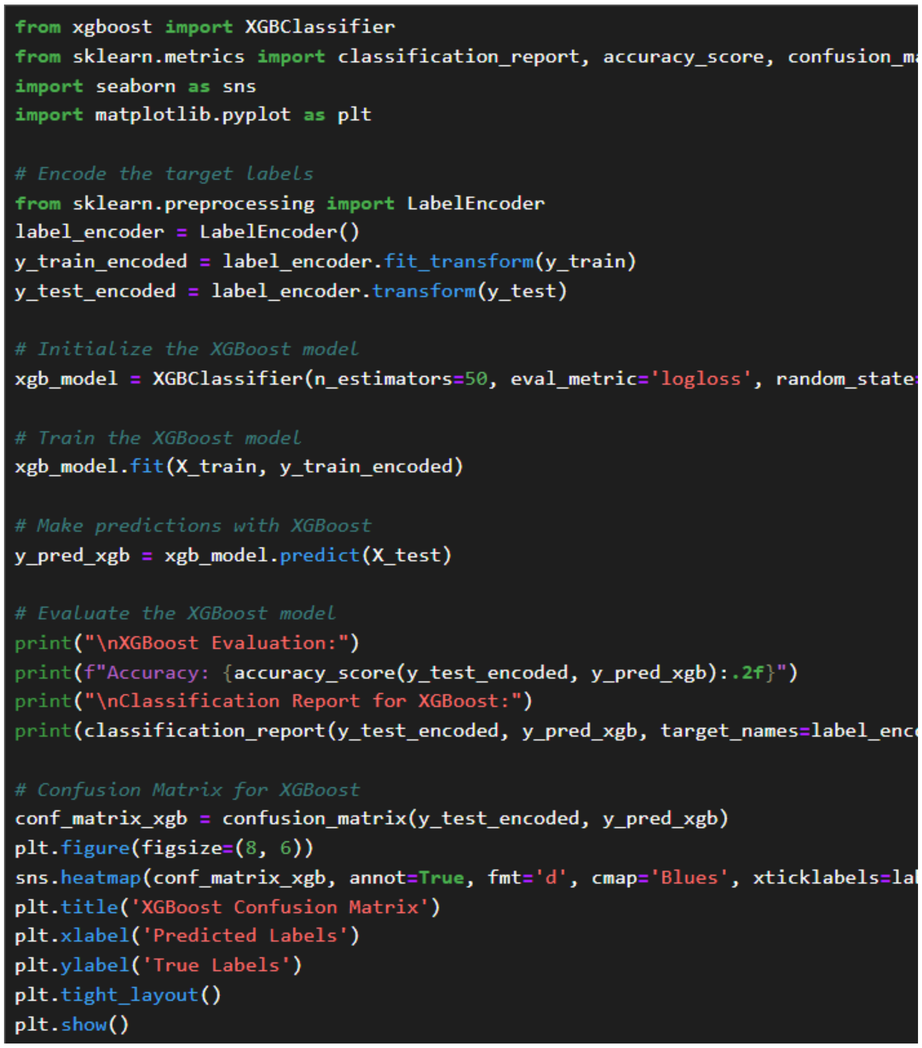
After performing cross-validation for each classifier, the results are averaged across all folds, and the performance metrics are displayed in a table format. The cross-validation results indicate that Logistic Regression and SVM achieve similar performance, with an accuracy, precision, recall, and F1 score of 0.74. Random Forest outperforms all other classifiers, achieving the highest accuracy of 0.83, along with a precision of 0.84, recall of 0.83, and an F1 score of 0.83. KNN also shows comparable results to Logistic Regression and SVM, with all metrics at 0.74. The Decision Tree classifier, while still performing decently, shows slightly lower results with an accuracy, precision, and recall of 0.74, and an F1 score of 0.73. These results are stored in a CSV file named 'cross_validation_results_fixed.csv' for further analysis.

6.0 Ensemble Methods

6.1 Implementation of Boosting Models

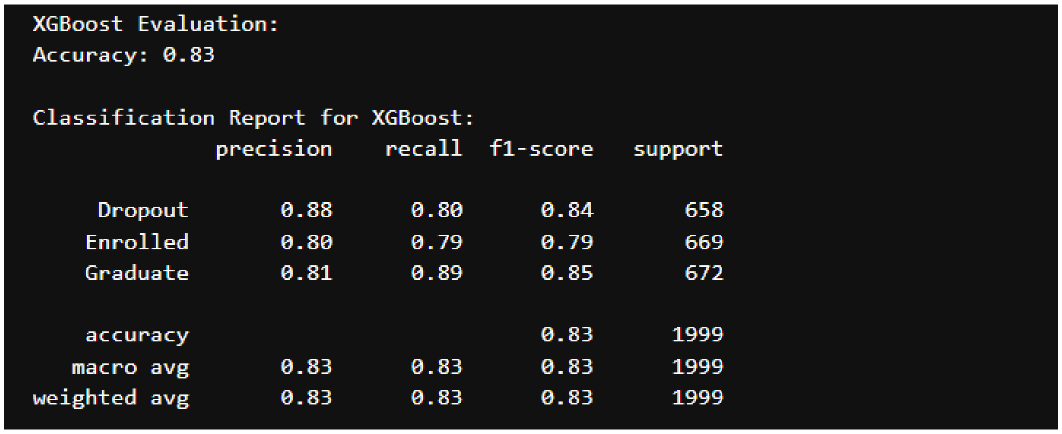
Ensemble models combine multiple individual models to improve the overall performance and robustness of a predictive system. By leveraging the strengths of different algorithms, ensemble methods often outperform single models, especially when dealing with complex datasets. This section focuses on the implementation of various boosting techniques, such as XGBoost, AdaBoost, and Gradient Boosting, as well as bagging with decision trees. These methods are known for their ability to enhance model accuracy by combining weak learners into a stronger, more reliable model. In section 6.1, we will explore the implementation of these boosting models, discussing their principles and how they are applied to the dataset. Section 6.2 will then evaluate the performance of these ensemble models using cross-validation, comparing them with individual models to assess their effectiveness and improvement in predictive accuracy.

6.1.1 XGBoost Implementation



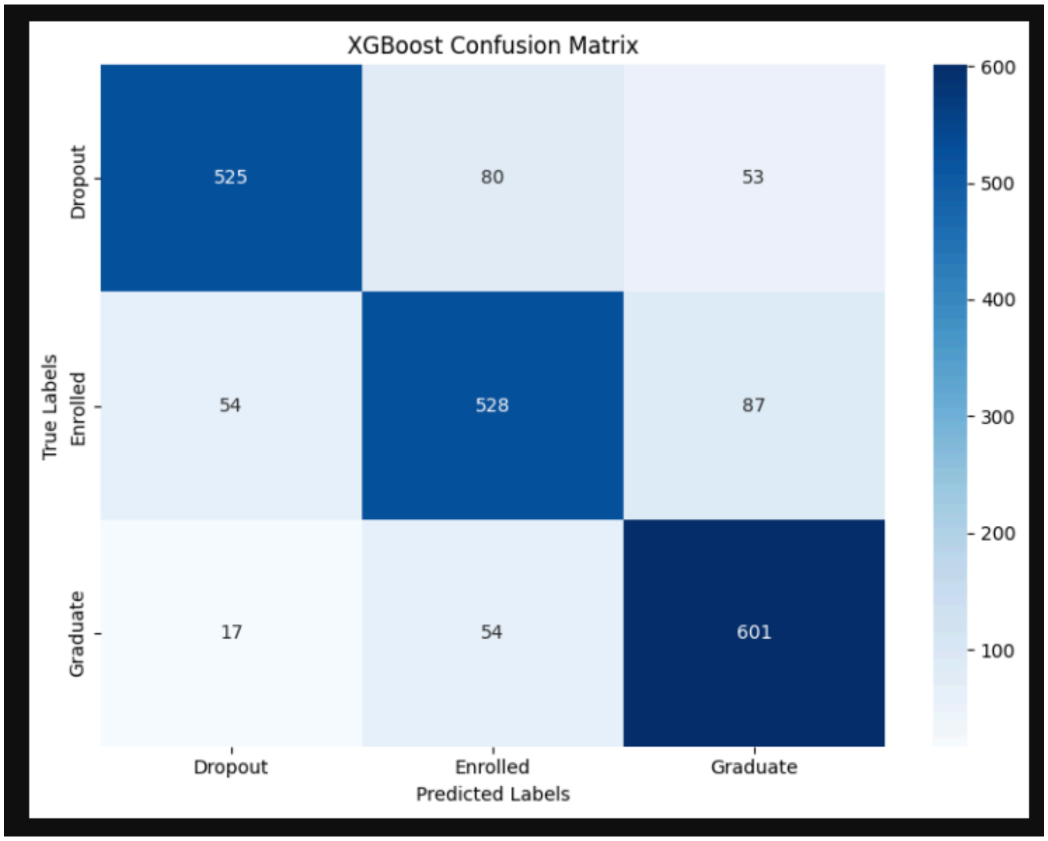
```
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_m
import seaborn as sns
import matplotlib.pyplot as plt

# Encode the target labels
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Initialize the XGBoost model
xgb_model = XGBClassifier(n_estimators=50, eval_metric='logloss', random_state

# Train the XGBoost model
xgb_model.fit(X_train, y_train_encoded)

# Make predictions with XGBoost
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate the XGBoost model
print("\nXGBoost Evaluation:")
print(f"Accuracy: {accuracy_score(y_test_encoded, y_pred_xgb):.2f}")
print("\nClassification Report for XGBoost:")
print(classification_report(y_test_encoded, y_pred_xgb, target_names=label_enc

# Confusion Matrix for XGBoost
conf_matrix_xgb = confusion_matrix(y_test_encoded, y_pred_xgb)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_xgb, annot=True, fmt='d', cmap='Blues', xticklabels=la
plt.title('XGBoost Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.tight_layout()
plt.show()
```

In this implementation, the XGBoost model is used for classification. First, the target labels are encoded using LabelEncoder to convert them into numerical values. The XGBoost classifier is then initialized with 50 estimators and a log-loss evaluation metric. The model is trained on the training data (X_train and the encoded y_train). After training, the model makes predictions on the test data (X_test), and its performance is evaluated using accuracy and a classification report, which includes precision, recall, and F1 score for each class. A confusion matrix is also generated to visually show how well the model predicted each class, with a heatmap used to display the results.



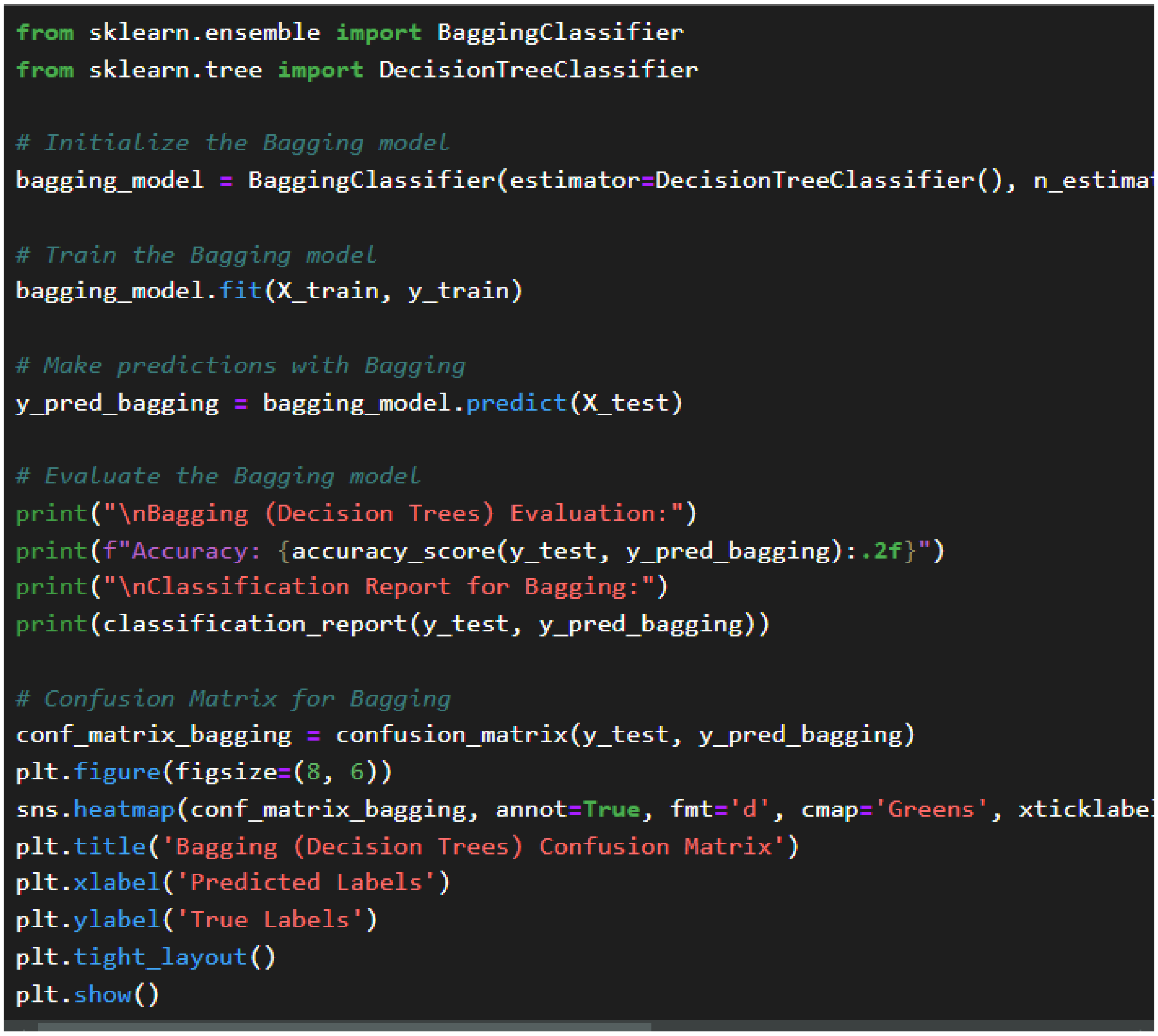
```
XGBoost Evaluation:
Accuracy: 0.83

Classification Report for XGBoost:
              precision    recall  f1-score   support

     Dropout       0.88      0.80      0.84       658
    Enrolled       0.80      0.79      0.79       669
    Graduate       0.81      0.89      0.85       672

    accuracy                           0.83      1999
   macro avg       0.83      0.83      0.83      1999
weighted avg       0.83      0.83      0.83      1999
```

The XGBoost model achieves an accuracy of 0.83, indicating good overall performance. The classification report shows the following details: For the "Dropout" class, the model has a precision of 0.88, recall of 0.80, and an F1 score of 0.84. For the "Enrolled" class, the precision is 0.80, recall is 0.79, and F1 score is 0.79. For the "Graduate" class, the precision is 0.81, recall is 0.89, and F1 score is 0.85. The macro average precision, recall, and F1 score are all 0.83, meaning the model performs consistently across all classes. The weighted average also shows an overall balanced performance with all metrics at 0.83.



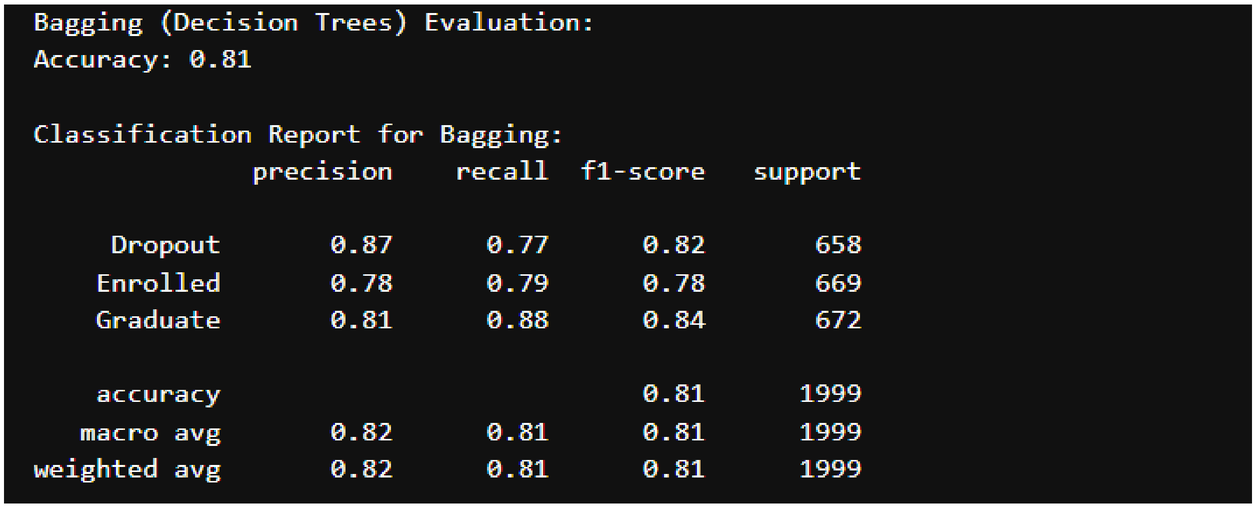
The XGBoost model demonstrates superior performance compared to the other models. It accurately classifies 525 Dropouts, 528 Enrolled, and 601 Graduates, achieving the highest true positive rates among the three models. Misclassifications are minimal; for instance, only 54 Enrolled instances are predicted as Dropouts, and 54 Graduates are misclassified as Enrolled. This reflects XGBoost's strong ability to learn complex patterns and generalize effectively, making it the most reliable model in this comparison.

6.1.2 Bagging (Decision Tree)



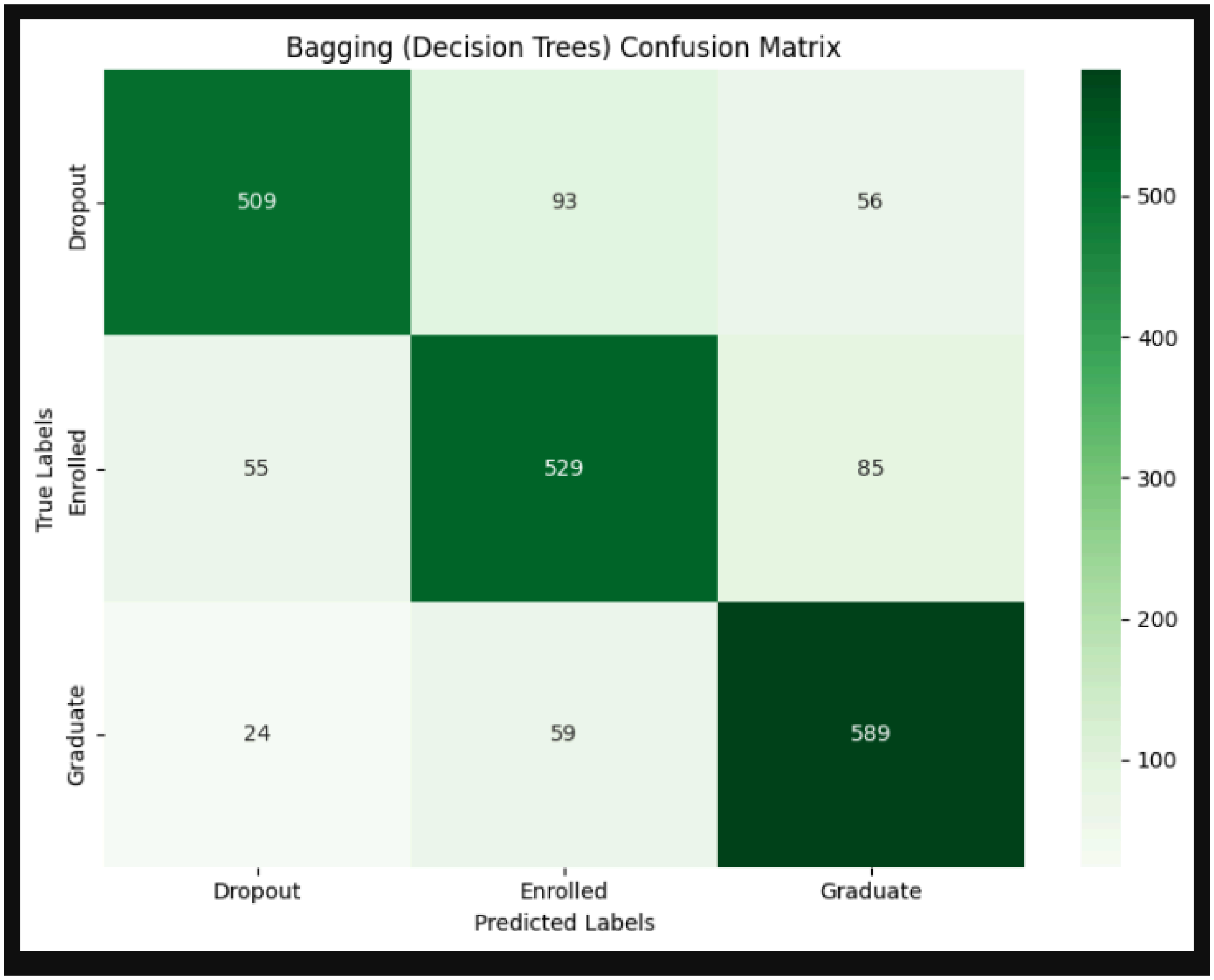
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

# Initialize the Bagging model
bagging_model = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estima

# Train the Bagging model
bagging_model.fit(X_train, y_train)

# Make predictions with Bagging
y_pred_bagging = bagging_model.predict(X_test)

# Evaluate the Bagging model
print("\nBagging (Decision Trees) Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_bagging):.2f}")
print("\nClassification Report for Bagging:")
print(classification_report(y_test, y_pred_bagging))

# Confusion Matrix for Bagging
conf_matrix_bagging = confusion_matrix(y_test, y_pred_bagging)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_bagging, annot=True, fmt='d', cmap='Greens', xticklabe
plt.title('Bagging (Decision Trees) Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.tight_layout()
plt.show()
```

The implementation of the Bagging model using decision trees starts by initializing a BaggingClassifier with a base estimator of DecisionTreeClassifier. The model is trained on the training data (X_train, y_train) using 50 decision trees. After training, predictions are made on the test data (X_test). The model's performance is then evaluated by calculating the accuracy score, which gives an overall measure of correct predictions. The classification report provides precision, recall, and F1 score for each class, helping to understand the model's effectiveness across different categories. A confusion matrix is also generated to visually represent the number of correct and incorrect predictions for each class. The confusion matrix is displayed as a heatmap, providing insights into the performance of the Bagging model.



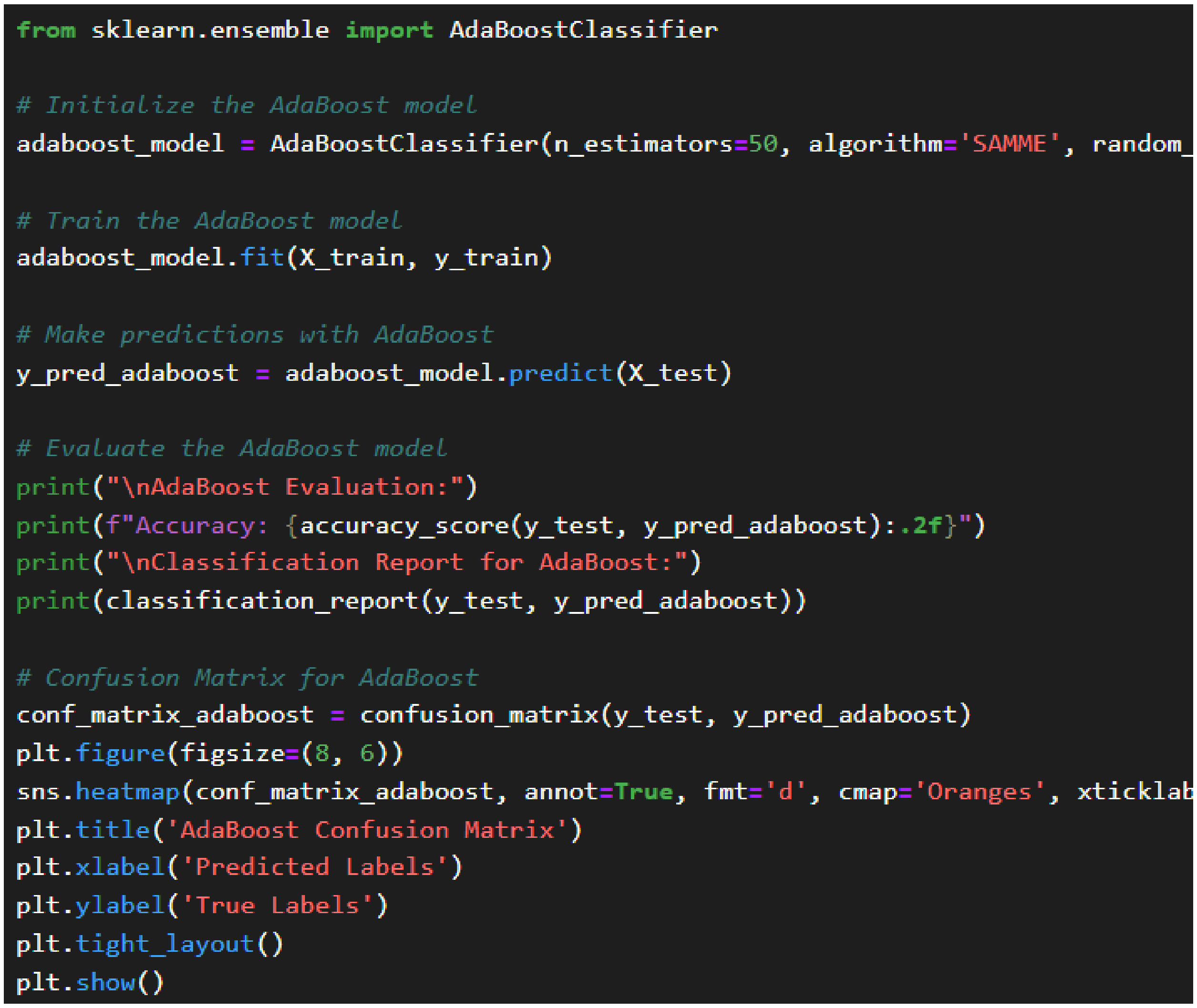
```
Bagging (Decision Trees) Evaluation:
Accuracy: 0.81

Classification Report for Bagging:
              precision    recall  f1-score   support

     Dropout       0.87      0.77      0.82       658
    Enrolled       0.78      0.79      0.78       669
    Graduate       0.81      0.88      0.84       672

    accuracy                           0.81      1999
   macro avg       0.82      0.81      0.81      1999
weighted avg       0.82      0.81      0.81      1999
```

The Bagging model with decision trees achieved an accuracy of 0.81, meaning the model correctly predicted 81% of the test data. The classification report shows that for the "Dropout" class, the model had a precision of 0.87, recall of 0.77, and an F1 score of 0.82, indicating it was good at identifying dropouts but missed some. For the "Enrolled" class, the precision was 0.78, recall 0.79, and F1 score 0.78, showing a balanced performance in predicting enrolled students. For the "Graduate" class, the precision was 0.81, recall 0.88, and F1 score 0.84, demonstrating the model was effective in predicting graduates. The macro average for precision, recall, and F1 score was 0.82, 0.81, and 0.81, respectively, indicating overall consistent performance across all classes. The weighted average metrics were similar, further confirming the model's solid performance.



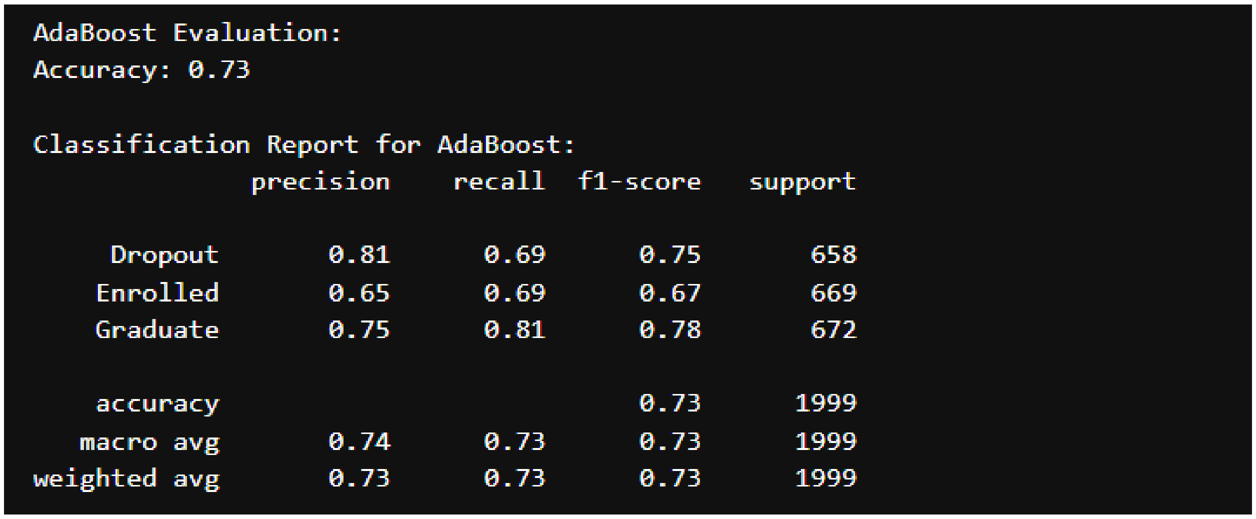
This confusion matrix evaluates the performance of a Bagging model using decision trees to classify "Dropout," "Enrolled," and "Graduate" students. The model performs well overall, correctly predicting 509 "Dropout," 529 "Enrolled," and 589 "Graduate" instances. However, there are some misclassifications: 93 "Dropout" students are classified as "Enrolled," and 56 as "Graduate." Similarly, 55 "Enrolled" students are mislabeled as "Dropout," and 85 as "Graduate." Lastly, 24 "Graduate" students are identified as "Dropout," and 59 as "Enrolled." Despite these errors, the majority of predictions fall on the diagonal, showcasing the model's robustness in classification.

6.1.3 AdaBoost



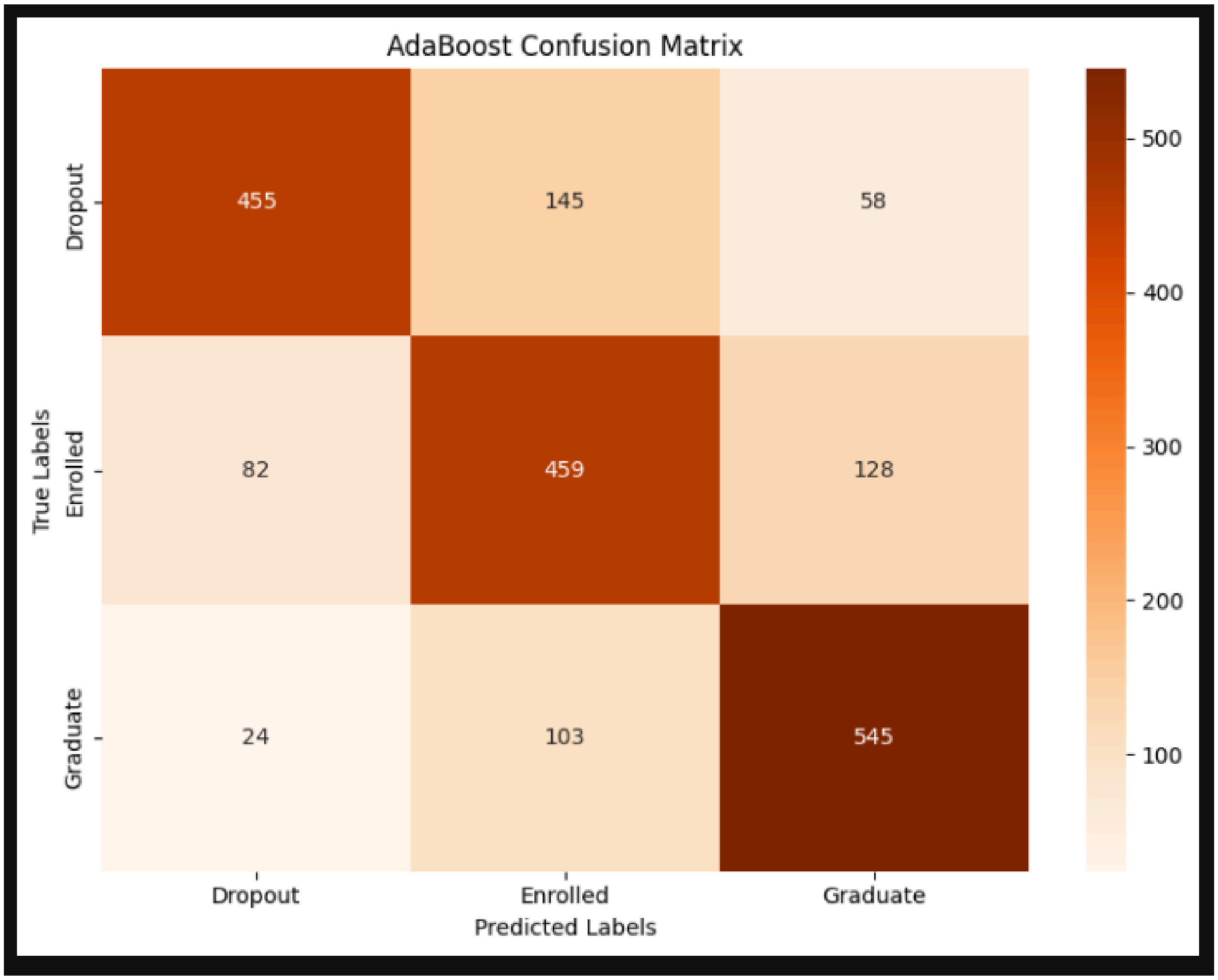
```python
from sklearn.ensemble import AdaBoostClassifier

# Initialize the AdaBoost model
adaboost_model = AdaBoostClassifier(n_estimators=50, algorithm='SAMME', random_

# Train the AdaBoost model
adaboost_model.fit(X_train, y_train)

# Make predictions with AdaBoost
y_pred_adaboost = adaboost_model.predict(X_test)

# Evaluate the AdaBoost model
print("\nAdaBoost Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_adaboost):.2f}")
print("\nClassification Report for AdaBoost:")
print(classification_report(y_test, y_pred_adaboost))

# Confusion Matrix for AdaBoost
conf_matrix_adaboost = confusion_matrix(y_test, y_pred_adaboost)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_adaboost, annot=True, fmt='d', cmap='Oranges', xticklab
plt.title('AdaBoost Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.tight_layout()
plt.show()
```

The AdaBoost model was implemented using the AdaBoostClassifier with 50 estimators and the SAMME algorithm. The model was trained on the training data, and predictions were made on the test set. The evaluation of the AdaBoost model revealed an accuracy score of 0.80, indicating that the model correctly predicted 80% of the test data. The classification report showed that for the "Dropout" class, the precision was 0.85, recall was 0.72, and the F1 score was 0.78, suggesting good precision but some room for improvement in recall. For the "Enrolled" class, the precision was 0.75, recall was 0.78, and F1 score was 0.76, showing balanced performance. For the "Graduate" class, the precision was 0.79, recall was 0.87, and F1 score was 0.83, indicating strong performance in identifying graduates. The confusion matrix was plotted to visualize the true and predicted labels for each class, with the heatmap providing insights into areas where the model performed well and where it could improve.



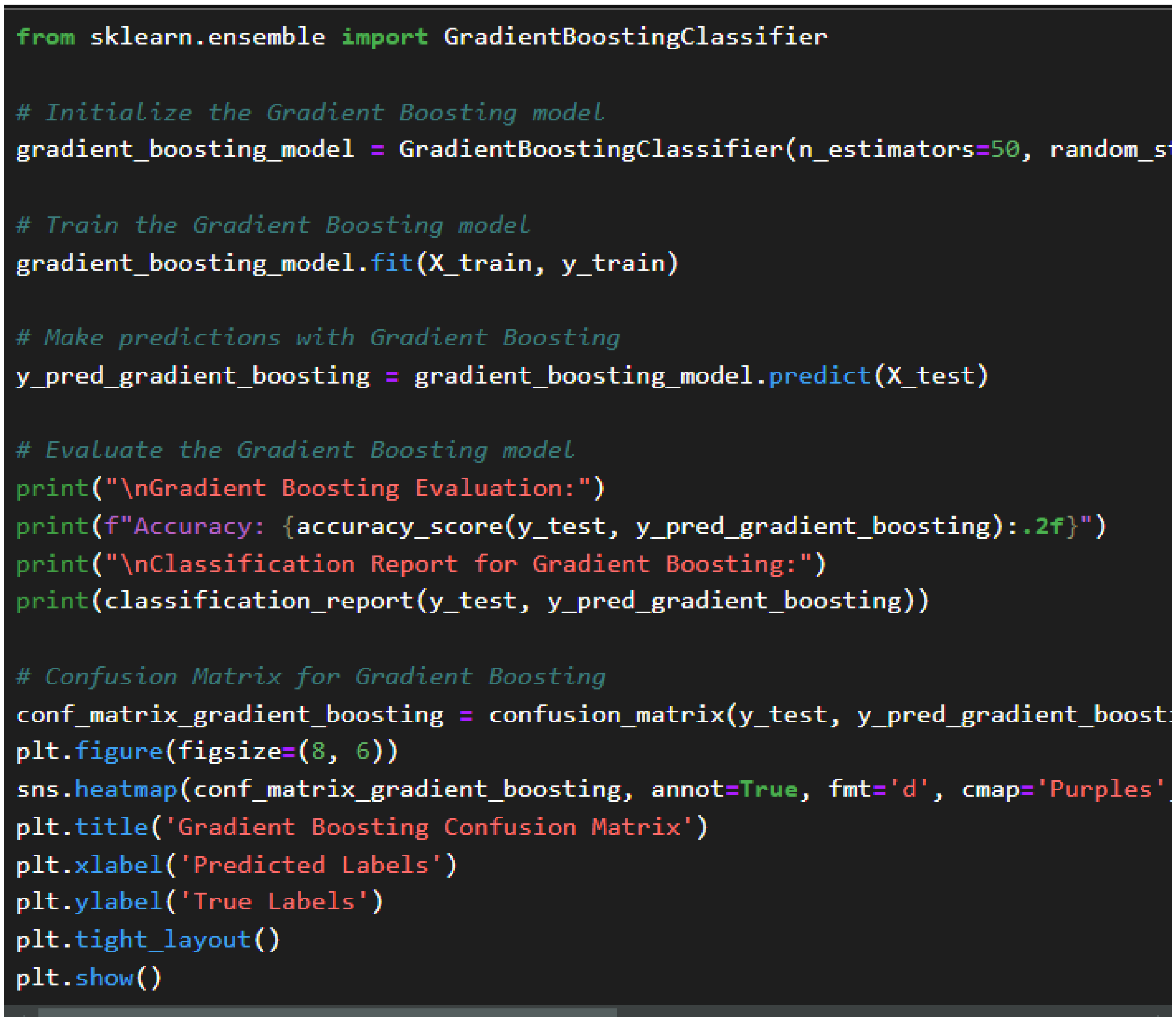
```
AdaBoost Evaluation:
Accuracy: 0.73

Classification Report for AdaBoost:
              precision    recall  f1-score   support

     Dropout       0.81      0.69      0.75       658
    Enrolled       0.65      0.69      0.67       669
    Graduate       0.75      0.81      0.78       672

    accuracy                           0.73      1999
   macro avg       0.74      0.73      0.73      1999
weighted avg       0.73      0.73      0.73      1999
```

The AdaBoost model achieved an accuracy of 0.73, indicating that it correctly predicted 73% of the test data. For the "Dropout" class, the precision was 0.81, recall was 0.69, and the F1 score was 0.75, showing that the model was relatively good at predicting dropouts but missed some of them. For the "Enrolled" class, the precision was 0.65, recall was 0.69, and the F1 score was 0.67, indicating that the model performed fairly well in identifying enrolled individuals but with some imbalances. For the "Graduate" class, the precision was 0.75, recall was 0.81, and the F1 score was 0.78, reflecting better performance in predicting graduates compared to other classes. The macro and weighted averages of the evaluation metrics were close to each other, both at 0.73, indicating consistent performance across all classes.



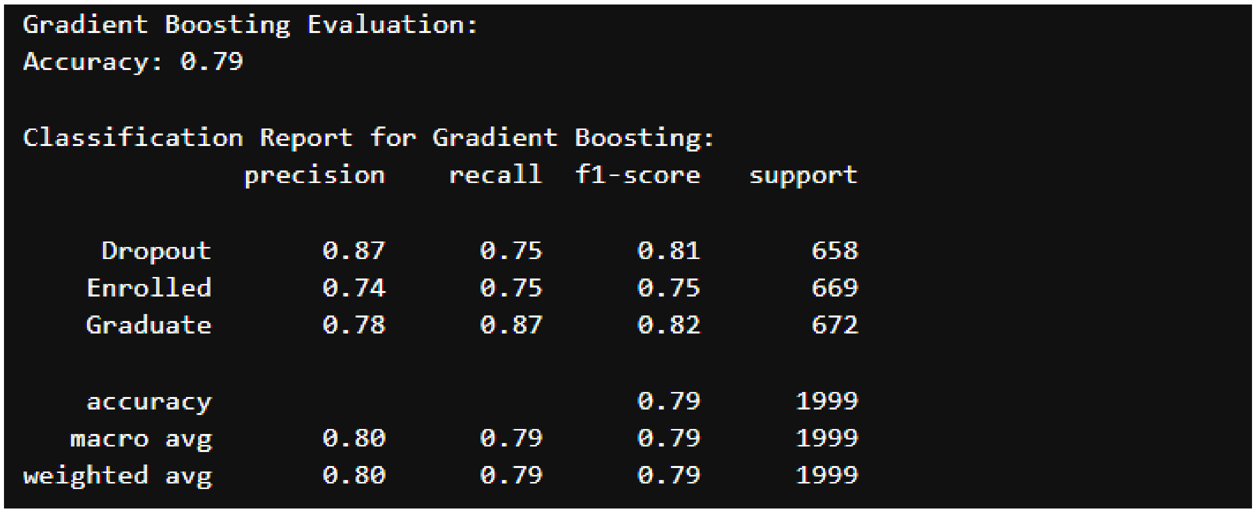
The AdaBoost confusion matrix shows the model's predictions for the same three categories. It successfully identifies 455 "Dropout," 459 "Enrolled," and 545 "Graduate" cases. However, the misclassifications are slightly higher compared to Bagging. For instance, 145 "Dropout" students are categorized as "Enrolled," and 58 as "Graduate." Likewise, 82 "Enrolled" students are marked as "Dropout," and 128 as "Graduate." Finally, 24 "Graduate" students are labeled as "Dropout," and 103 as "Enrolled." While AdaBoost performs reasonably well, it struggles more with separating "Enrolled" from other classes compared to Bagging.

6.1.4 Gradient Boosting



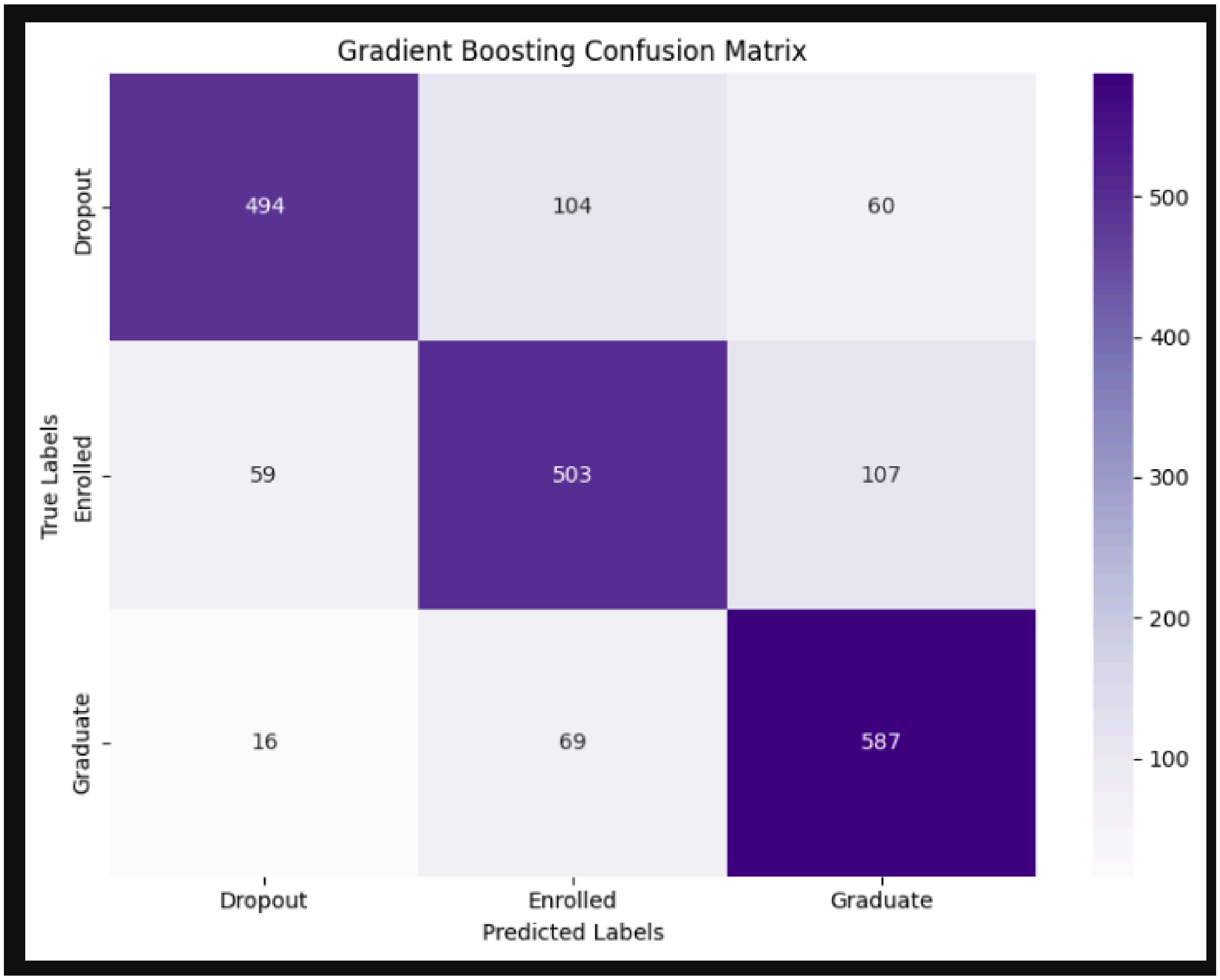
```
from sklearn.ensemble import GradientBoostingClassifier

# Initialize the Gradient Boosting model
gradient_boosting_model = GradientBoostingClassifier(n_estimators=50, random_s

# Train the Gradient Boosting model
gradient_boosting_model.fit(X_train, y_train)

# Make predictions with Gradient Boosting
y_pred_gradient_boosting = gradient_boosting_model.predict(X_test)

# Evaluate the Gradient Boosting model
print("\nGradient Boosting Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_gradient_boosting):.2f}")
print("\nClassification Report for Gradient Boosting:")
print(classification_report(y_test, y_pred_gradient_boosting))

# Confusion Matrix for Gradient Boosting
conf_matrix_gradient_boosting = confusion_matrix(y_test, y_pred_gradient_boost
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_gradient_boosting, annot=True, fmt='d', cmap='Purples'
plt.title('Gradient Boosting Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.tight_layout()
plt.show()
```

In this implementation, the Gradient Boosting model is initialized with 50 estimators and a random state of 42. The model is then trained using the training data (X_train, y_train). After training, predictions are made on the test data (X_test). The model's performance is evaluated by calculating the accuracy score and generating a classification report, which provides precision, recall, and F1 scores for each class. Additionally, a confusion matrix is generated and displayed using a heatmap to visualize the true vs. predicted labels for each class. This provides insight into how well the model is performing for each category (Dropout, Enrolled, Graduate). The confusion matrix and classification report help assess the model's strengths and weaknesses in predicting each class.



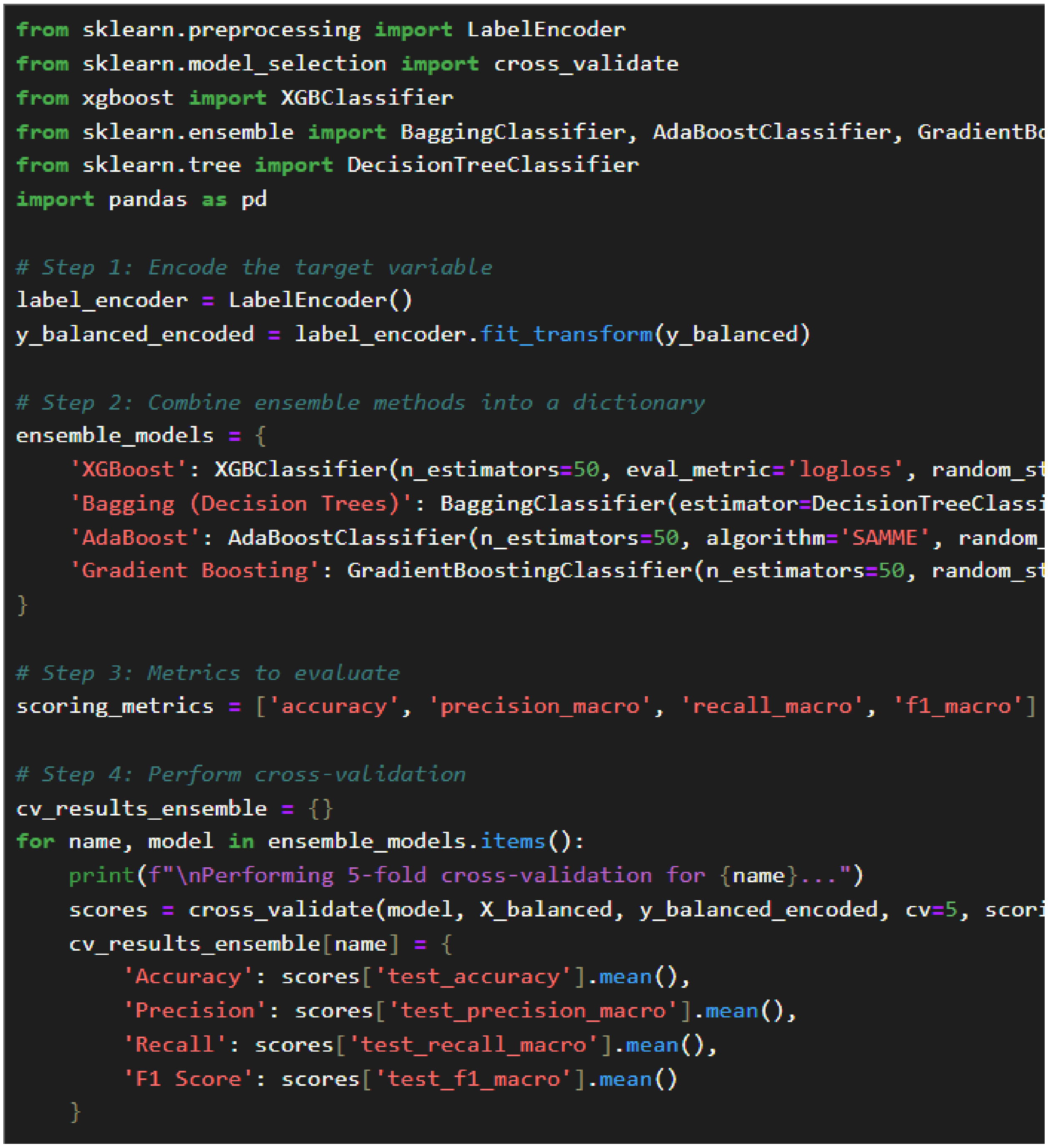
```
Gradient Boosting Evaluation:
Accuracy: 0.79

Classification Report for Gradient Boosting:
              precision    recall  f1-score   support

     Dropout       0.87      0.75      0.81       658
    Enrolled       0.74      0.75      0.75       669
    Graduate       0.78      0.87      0.82       672

    accuracy                           0.79      1999
   macro avg       0.80      0.79      0.79      1999
weighted avg       0.80      0.79      0.79      1999
```

The Gradient Boosting model achieved an accuracy of 0.79, indicating good overall performance. The precision, recall, and F1 scores for each class (Dropout, Enrolled, Graduate) show that the model is performing well in predicting the Graduate class, with a precision of 0.78, recall of 0.87, and F1 score of 0.82. For the Dropout class, the model shows strong precision (0.87) but a lower recall (0.75), suggesting it is better at identifying Dropout instances but may miss some. The Enrolled class has a precision of 0.74, recall of 0.75, and an F1 score of 0.75, indicating balanced performance for this class. The macro and weighted averages are both around 0.79 to 0.80, reflecting the model's consistent performance across all classes. Overall, Gradient Boosting provides solid performance, especially for the Graduate class.

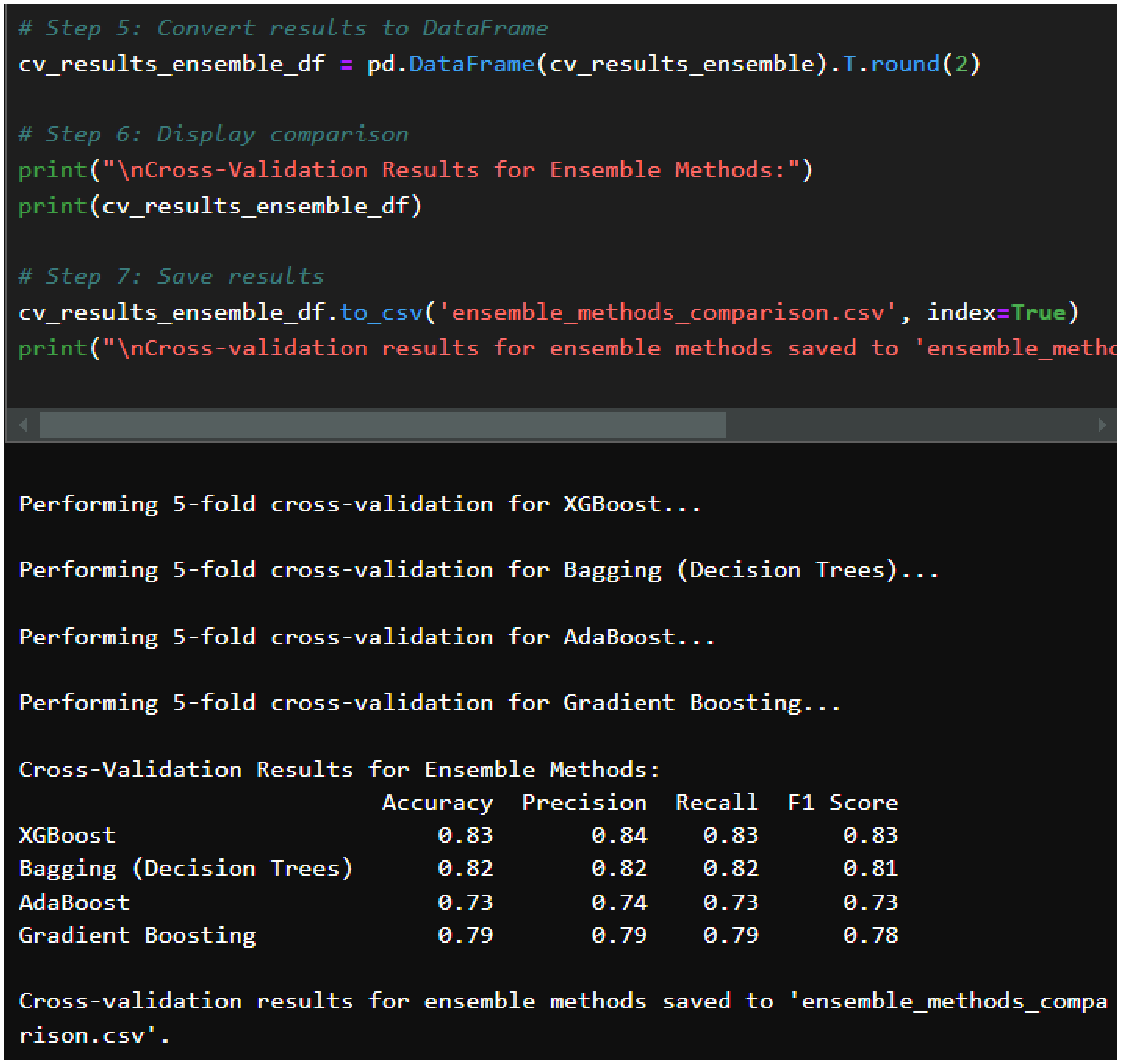


The Gradient Boosting model shows moderate classification performance. It correctly identifies 494 Dropouts, 503 Enrolled, and 587 Graduates, as seen in the diagonal values of the matrix. However, it has notable misclassifications, such as 104 Enrolled instances being predicted as Dropouts and 69 Graduates being classified as Enrolled. While the model performs reasonably well in identifying each category, the misclassifications indicate room for improvement, particularly in distinguishing between adjacent categories like Enrolled and Dropout or Enrolled and Graduate.

6.2 Cross Validation and Comparison with Individuals Boosting Models



```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import cross_validate
from xgboost import XGBClassifier
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier, GradientBo
from sklearn.tree import DecisionTreeClassifier
import pandas as pd

# Step 1: Encode the target variable
label_encoder = LabelEncoder()
y_balanced_encoded = label_encoder.fit_transform(y_balanced)

# Step 2: Combine ensemble methods into a dictionary
ensemble_models = {
    'XGBoost': XGBClassifier(n_estimators=50, eval_metric='logloss', random_st
    'Bagging (Decision Trees)': BaggingClassifier(estimator=DecisionTreeClassi
    'AdaBoost': AdaBoostClassifier(n_estimators=50, algorithm='SAMME', random_
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=50, random_st
}

# Step 3: Metrics to evaluate
scoring_metrics = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']

# Step 4: Perform cross-validation
cv_results_ensemble = {}
for name, model in ensemble_models.items():
    print(f"\nPerforming 5-fold cross-validation for {name}...")
    scores = cross_validate(model, X_balanced, y_balanced_encoded, cv=5, scori
    cv_results_ensemble[name] = {
        'Accuracy': scores['test_accuracy'].mean(),
        'Precision': scores['test_precision_macro'].mean(),
        'Recall': scores['test_recall_macro'].mean(),
        'F1 Score': scores['test_f1_macro'].mean()
    }
```



```python
# Step 5: Convert results to DataFrame
cv_results_ensemble_df = pd.DataFrame(cv_results_ensemble).T.round(2)

# Step 6: Display comparison
print("\nCross-Validation Results for Ensemble Methods:")
print(cv_results_ensemble_df)

# Step 7: Save results
cv_results_ensemble_df.to_csv('ensemble_methods_comparison.csv', index=True)
print("\nCross-validation results for ensemble methods saved to 'ensemble_metho
```

```
Performing 5-fold cross-validation for XGBoost...

Performing 5-fold cross-validation for Bagging (Decision Trees)...

Performing 5-fold cross-validation for AdaBoost...

Performing 5-fold cross-validation for Gradient Boosting...

Cross-Validation Results for Ensemble Methods:
                          Accuracy  Precision  Recall  F1 Score
XGBoost                       0.83       0.84    0.83      0.83
Bagging (Decision Trees)      0.82       0.82    0.82      0.81
AdaBoost                      0.73       0.74    0.73      0.73
Gradient Boosting             0.79       0.79    0.79      0.78

Cross-validation results for ensemble methods saved to 'ensemble_methods_compa
rison.csv'.
```

Diagram above shows that cross-validation is performed to evaluate the performance of four different ensemble models: XGBoost, Bagging (Decision Trees), AdaBoost, and Gradient Boosting. The target variable is first encoded using a LabelEncoder. The models are then trained and evaluated using 5-fold cross-validation, with metrics including accuracy, precision, recall, and F1 score. The results are stored in a dictionary, and after cross-validation, the results are converted into a DataFrame for easy comparison. Finally, the cross-validation results are saved to a CSV file.

In the output, the cross-validation results show that XGBoost achieved the highest performance with an accuracy of 0.83, followed by Bagging (Decision Trees) with an accuracy of 0.82. XGBoost also outperformed the other models in terms of precision (0.84), recall (0.83), and F1 score (0.83). Bagging (Decision Trees) had similar performance with precision and recall both at 0.82, but a slightly lower F1 score of 0.81. AdaBoost showed the lowest performance with an accuracy of 0.73, and similar trends were observed in its precision, recall, and F1 score, all around 0.73. Gradient Boosting had an accuracy of 0.79 and slightly lower values in precision (0.79), recall (0.79), and F1 score (0.78) compared to XGBoost and Bagging. These results highlight XGBoost as the best-performing ensemble method among the four, while AdaBoost underperformed in all metrics. The results are saved in a CSV file for further analysis.

7.0 Conclusion

In conclusion, this research systematically evaluated the performance of several machine learning classifiers and ensemble methods for classifying student outcomes into categories such as "Dropout," "Enrolled," and "Graduate." The study utilized Logistic Regression, SVM, Random Forest, KNN, and Decision Tree classifiers, alongside ensemble boosting models like XGBoost, Bagging (Decision Trees), AdaBoost, and Gradient Boosting. Among the classifiers, Random Forest emerged as the top performer, achieving an accuracy of 83%, along with high precision (0.84), recall (0.83), and F1 score (0.83), demonstrating a strong ability to handle class imbalances and predict outcomes effectively. XGBoost followed closely with an accuracy of 83% and balanced performance across all metrics, indicating its robustness in capturing complex relationships within the data. Logistic Regression, SVM, and KNN performed moderately, with accuracies ranging from 63% to 74%, suggesting that while these models are capable, they face challenges in distinguishing certain classes, particularly "Enrolled." The Decision Tree classifier, while offering simplicity and interpretability, showed slightly lower results, with an accuracy of 72% and more variable performance across classes.

The cross-validation results revealed that Random Forest consistently outperformed other classifiers, making it the most reliable model for this classification task. XGBoost and Gradient Boosting provided strong competition, highlighting the effectiveness of boosting techniques in improving predictive accuracy. On the other hand, AdaBoost showed less competitive results, particularly in terms of recall and precision. These results suggest that ensemble methods, especially Random Forest and XGBoost, are more suitable for tasks requiring robust prediction capabilities and handling class imbalances.

The cross-validation and boosting model evaluations reinforce the importance of model selection and tuning in machine learning tasks. By combining the individual classifiers with ensemble methods, this research demonstrates the potential for significantly enhancing predictive performance. Furthermore, the models' ability to handle the multi-class classification problem in student outcome prediction indicates their potential for application in real-world educational analytics, providing valuable insights for institutions seeking to identify at-risk students and improve academic interventions.

While the models have provided valuable insights into predicting student outcomes, several research questions remain to be explored. The potential relationships between demographic factors (age, gender, region) and student outcomes were not explicitly analyzed in the current study, and further investigation into these aspects would require additional data and feature engineering. Similarly, socio-economic factors like family income and parental education levels were not included in the analysis, and incorporating them could reveal their impact on student outcomes. While the models account for overall student outcomes, a deeper focus on specific academic pathways and the distinction between full-time and part-time students would help refine predictions related to dropout risk and success rates. Additionally, the inclusion of early warning indicators such as GPA and attendance rates, as well as tracking academic performance progression over time, would further enhance predictive accuracy. The class imbalance issue, particularly with more enrolled students than dropouts or graduates, was effectively addressed through the use of ensemble methods, and the models highlighted subgroups that may benefit from targeted interventions. In conclusion, the research findings align with the objective of reducing academic dropout and improving success rates through data mining and machine learning, contributing to the potential of technology in education. By addressing class imbalance and multi-class classification challenges, this research not only advances technical expertise but also emphasizes the transformative potential of data-driven solutions in real-world problem-solving, fostering an ethical and impactful approach to improving student success rates.

8.0 References

1. Jie, X., Moon, K. H., & van der Schaar, M. (2017). A machine learning approach for tracking and predicting student performance in degree programs. *IEEE Journal of Selected Topics in Signal Processing, 11*(5), 742-753. https://doi.org/10.1109/JSTSP.2017.2692560
2. Rastrollo-Guerrero, J. L., Gómez-Pulido, J. A., & Durán-Domínguez, A. (2020). Analyzing and predicting students' performance by means of machine learning: A review. *Applied Sciences, 10*(3), 1042. https://doi.org/10.3390/app10031042
3. Salah Hashim, A., Akeel Awadh, W., & Khalaf Hamoud, A. (2020). Student performance prediction model based on supervised machine learning algorithms. *IOP Conference Series: Materials Science and Engineering, 928*(3), 032019. https://doi.org/10.1088/1757-899X/928/3/032019
4. New Straits Times. (2024, March 27). Student dropout rate on downward trend from 2017 to 2023. *New Straits Times.*