UCS2604
Principles of Machine Learning

MINI PROJECT

Submitted By

Dilsha Singh D (3122 22 5001 028)

Kushal Varma Gudimadugu (3122 22 5001 031)

Hannah S (3122 22 5001 032)

Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

Kalavakkam – 603110

TABLE OF CONTENTS:

Predict Students' Dropout and Academic Success

1. Abstract:

Student retention and academic success are crucial indicators for educational institutions to improve learning outcomes and reduce dropout rates. With the growing availability of student data, machine learning (ML) models provide a valuable tool for predicting student dropout and academic success. This study leverages structured data, including demographic details, academic performance, and socio-economic factors, to build predictive models that classify students into **Dropout, Enrolled, and Graduate** categories. Key features such as age, grades, attendance, and financial support were analyzed to enhance model accuracy. The proposed approach addresses challenges like imbalanced data and model interpretability, offering actionable insights for institutions to implement early intervention strategies and improve student success.

2. Introduction:

In today's evolving education landscape, predicting student dropout and academic success is crucial for institutions to enhance learning outcomes and implement effective support strategies. Traditional methods of identifying at-risk students, such as manual assessments and surveys, are often time-consuming and prone to bias. Institutions can now analyze large volumes of student data to predict academic success and dropout risks more accurately. Structured datasets, including demographic details, academic performance, and socio-economic factors, provide valuable insights into student behavior and learning patterns. By leveraging these features, ML models can identify key factors influencing student retention and success, enabling institutions to take proactive measures to improve student support systems. This study explores the use of ML models to predict student outcomes based on a dataset containing features such as student ID, age, gender, academic performance, attendance, financial support, and family background, providing data-driven insights for better educational interventions.

2.1. Problem Statement

Educational institutions face challenges in identifying students at risk of dropping out or underperforming. Early intervention is crucial to improve student retention rates and academic success. This dataset contains student demographic details, academic records, and socio-economic factors. The objective is to develop a predictive model that classifies students into categories such as Dropout, Graduate, or Enrolled based on these factors.

3. Literature Survey

Several recent studies have explored the use of supervised machine learning models to predict student dropout rates in higher education, leveraging various types of academic and demographic data. One such study used a dataset collected from a higher education institution and compared different supervised learning algorithms for dropout prediction [1]. To address the issue of class imbalance, Synthetic Minority Over-sampling Technique (SMOTE) was employed. The models evaluated included Decision Trees, Support Vector Machines (SVM), Random Forests, and Boosting algorithms like XGBoost, LightGBM, and CatBoost, with the boosting methods showing superior performance in predicting student dropout.

Another study focused on dropout prediction based on secondary school performance, using a dataset of 15,825 undergraduate students from Budapest University of Technology and Economics [2]. This research primarily utilized demographic and secondary school academic data for early identification of students at risk of dropping out. Among the models tested, Gradient Boosted Trees and Deep Learning approaches achieved the highest accuracy, approximately 81%.

A comprehensive survey on student dropout prediction examined various machine learning techniques and highlighted the disparity in research focus between developed and developing nations [3]. It reviewed both supervised models, including Decision Trees, Logistic Regression, and

Neural Networks, as well as unsupervised methods such as Cluster Analysis, Anomaly Detection, and Survival Analysis. The paper identified major challenges like class imbalance—often with dropout cases occurring at a ratio of 1:10 compared to non-dropouts—and emphasized the need for better data integration at the school level, particularly in underrepresented regions. It called for improved data collection strategies and early warning systems to make ML-based dropout prediction more effective globally.

Additionally, a study analyzing 10,196 students from a Hungarian technical university investigated the predictive power of high school and first-semester academic performance on student dropout [4]. The research found that pre-enrollment data alone (S1) could predict dropout with an AUC of 0.815 using Artificial Neural Networks (ANN). Including first-week academic data (S2) yielded a slight improvement (AUC = 0.823), while full first-semester data (S4) significantly enhanced prediction accuracy (AUC = 0.892). The best results were achieved by combining pre-enrollment and first-semester data (S3), resulting in an AUC of 0.920.

4. Proposed System

4.1 System Architecture Diagram

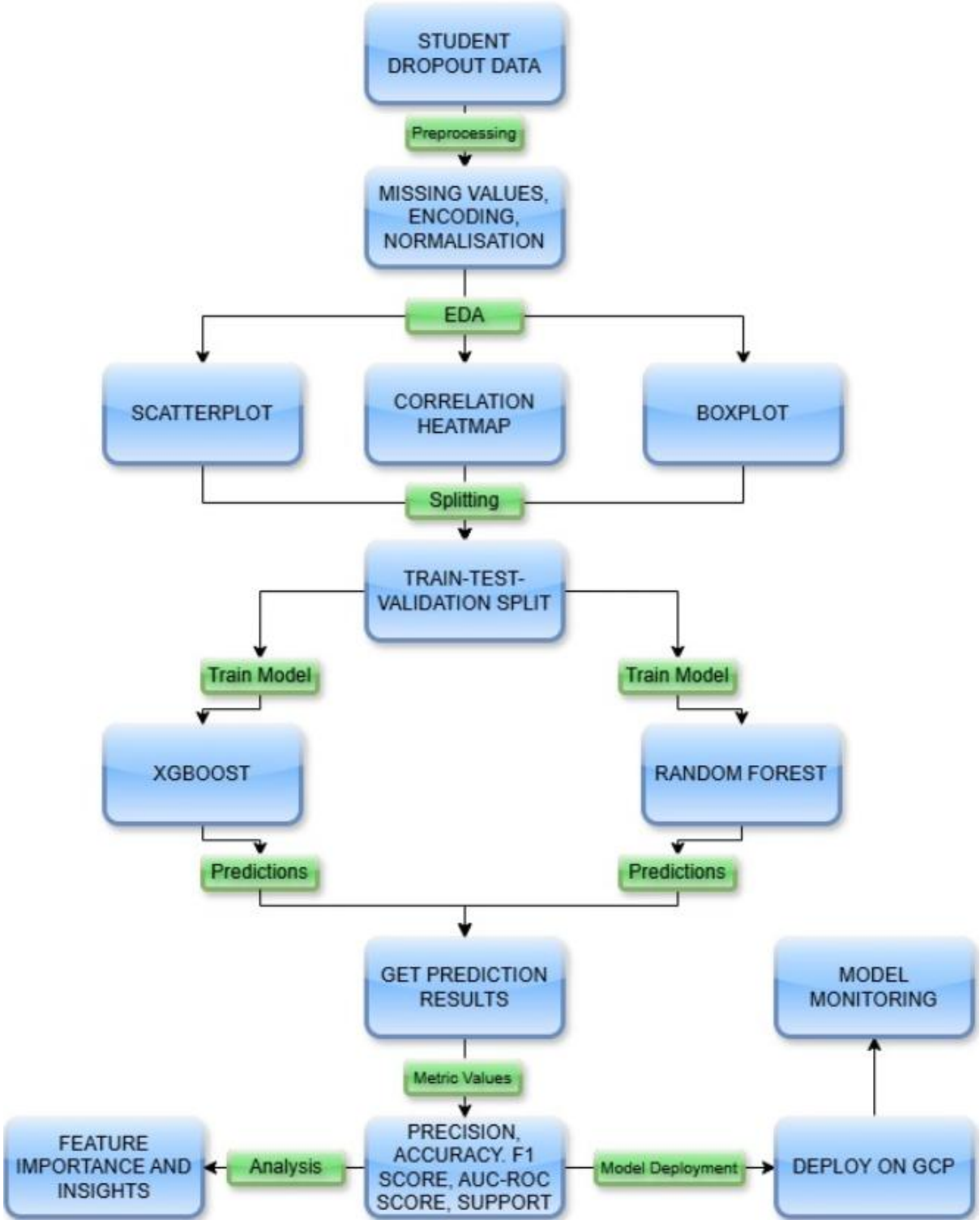


Figure 1: System Architecture Diagram

5. Implementation and Experiments

5.1 Development Environment

- **Operating System**: Windows
- **Development Platform**: Google Colab
- **Programming Language**: Python
- **IDE/Editor**: Google Colab
- **Libraries**: NumPy, Pandas, TensorFlow, Scikit-Learn, Seaborn(sns),MatPlotLib
- **Execution Environment**: Cloud-based (CPU/GPU/TPU on Google's servers)
- **File Management**: Google Drive for saving and loading files

5.2 Dataset Description

DATASET NAME: Predict students' dropout and academic success dataset

The dataset consists of about 4424 rows and 37 columns and it covers a wide range of information including:

· **Demographic information:** Marital status, nationality, mother's and father's qualifications and occupations.

- **Academic details:** Course information, admission grades, curricular unit performances (credited/enrolled/evaluations/approved), grades.
- **Economic indicators:** Unemployment rate, inflation rate, GDP.
- **Data Types:** The data types vary across columns including categorical (e.g., marital status), numerical (e.g., admission grade), and binary (e.g., scholarship holder).
- **Target Variable:** The dataset includes a target variable labelled as "Target," which likely indicates whether a student dropped out or graduated.

This dataset provides a comprehensive view of students enrolled in various undergraduate degrees offered at a higher education institution. It includes demographic data, social-economic factors and academic performance information that can be used to analyze the possible predictors of student dropout and academic success. This dataset contains multiple disjoint databases consisting of relevant information available at the time of enrollment, such as application mode, marital status, course chosen and more. Additionally, this data can be used to estimate overall student performance at the end of each semester by assessing curricular units credited/enrolled/evaluated/approved as well as their respective grades. Finally, we have unemployment rate, inflation rate and GDP from the region which can help us further understand how economic factors play into student dropout rates or academic success outcomes. This powerful analysis tool will provide valuable insight into what motivates students to stay in school or abandon their studies for a wide range of disciplines such as agronomy, design, education nursing journalism management social service or technologies. Overall, this dataset is designed for predictive modelling tasks like student dropout prediction using machine learning techniques by analysing various factors influencing educational outcomes.

5.3 Implementation

Apply Exploratory Data Analysis techniques on the collected dataset.

Identify the type of learning required (Supervised/ unsupervised/Reinforcement Learning) for the problem statement. Select any two/ more ML algorithms which come under a learning task.

Apply data-pre-processing techniques on the collected dataset.

EDA:

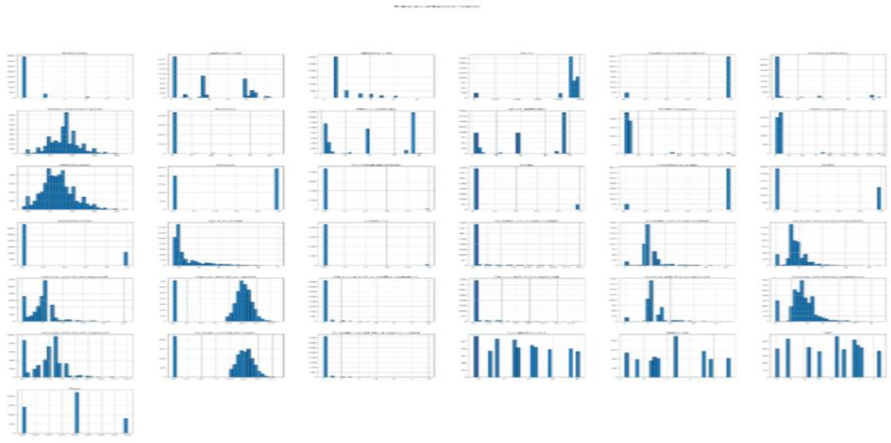
CODE:

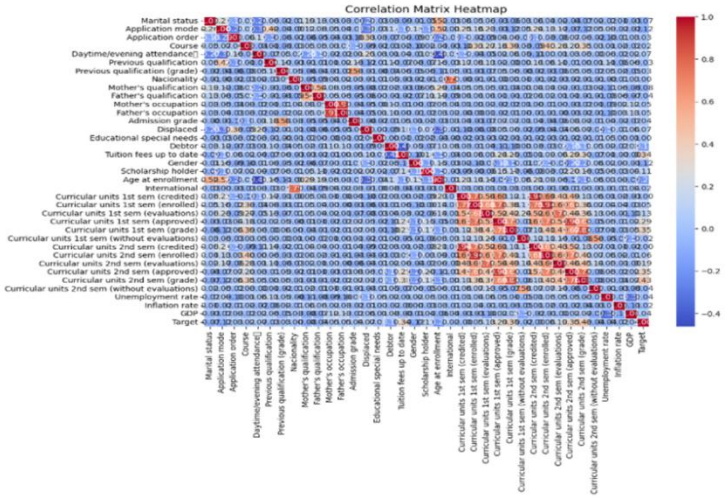
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

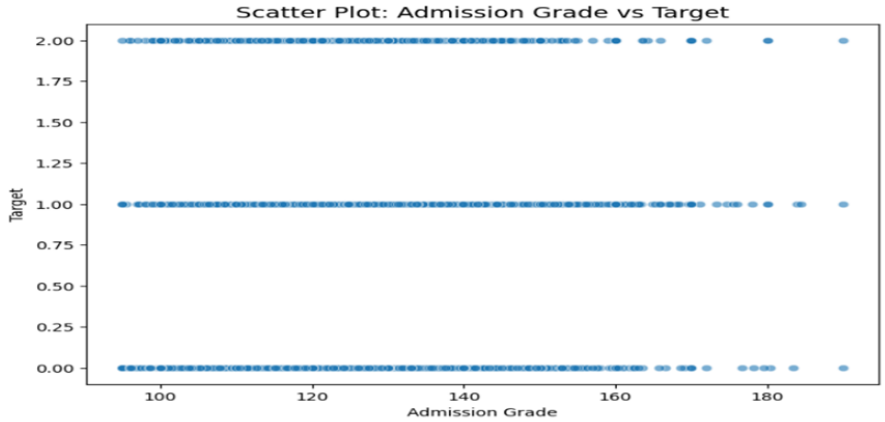
```
import seaborn as sns

# Load the dataset
train = pd.read_csv("/content/drive/MyDrive/data.csv", sep=';')



print(train.info())
print(train.describe())

# Identify categorical columns
categorical_cols = train.select_dtypes(include=['object']).columns
print("Categorical Columns:", categorical_cols)

# Convert categorical columns to numeric using factorize()
for col in categorical_cols:
    train[col], _ = pd.factorize(train[col])



numerical_cols = train.select_dtypes(include=['number']).columns

# Histogram
plt.figure(figsize=(15, 10))
train[numerical_cols].hist(figsize=(50, 45), bins=30, edgecolor='black')
plt.suptitle('Histograms of Numerical Features', fontsize=16)
plt.show()

# 3. Correlation Matrix with Heatmap
plt.figure(figsize=(12, 8))
corr_matrix = train.corr()  # Now all columns are numeric
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
plt.title("Correlation Matrix Heatmap", fontsize=14)
```

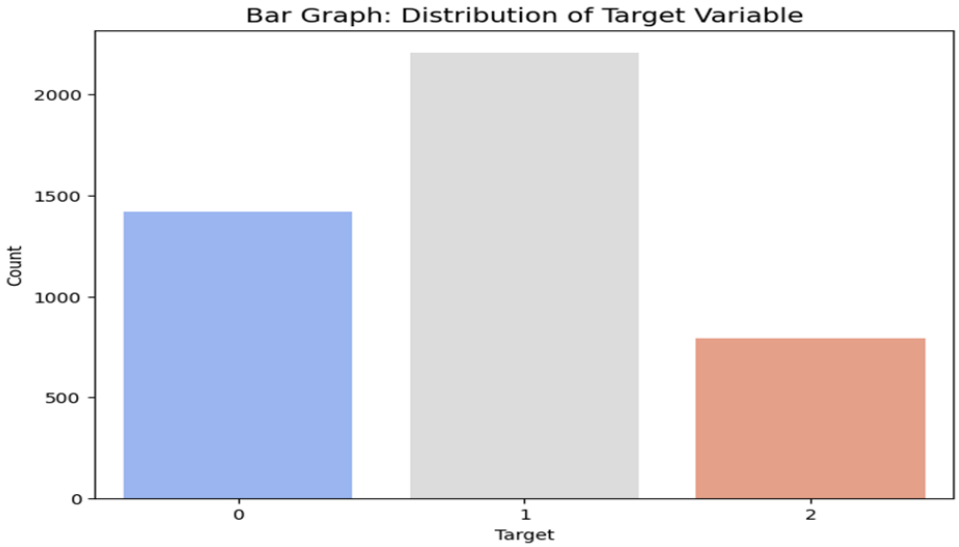
```
plt.show()


# 4. Scatter Plot: Admission Grade vs Target
plt.figure(figsize=(8, 6))
sns.scatterplot(x=train['Admission grade'], y=train['Target'], alpha=0.6)
plt.title("Scatter Plot: Admission Grade vs Target", fontsize=14)
plt.xlabel("Admission Grade")
plt.ylabel("Target")
plt.show()


# 5. Box Plot: Admission Grade by Gender
plt.figure(figsize=(8, 6))
sns.boxplot(x=train['Gender'], y=train['Admission grade'],
palette="coolwarm")
plt.title("Box Plot: Admission Grade by Gender", fontsize=14)
plt.show()
plt.figure(figsize=(8, 6))
sns.countplot(x=train['Target'], palette="coolwarm")
plt.title("Bar Graph: Distribution of Target Variable", fontsize=14)
plt.xlabel("Target")
plt.ylabel("Count")
plt.show()
```

OUTPUTS:









Type of Learning Required: Supervised Learning

Since the dataset contains labeled outcomes (Target column with categories: Dropout, Graduate, Enrolled), this is a Supervised Learning

problem. The model learns patterns from historical student data to predict future academic outcomes.

Selected Machine Learning Algorithms

Random Forest and XGBoost

1. Random Forest (RF)

- An ensemble learning method that combines multiple decision trees.
- Handles non-linearity and high-dimensional data effectively.
- Pros: Robust to missing data, interpretable, handles feature importance well.
- Use Case: Helps identify key student factors affecting dropout risk.

2. XGBoost (Extreme Gradient Boosting)

- A boosting algorithm that builds trees sequentially to correct previous mistakes.
- Highly efficient with excellent predictive power.
- Pros: Works well with large datasets, handles imbalanced data, feature interactions.
- Use Case: Can optimize institutional intervention strategies by identifying at-risk students more accurately.

Why These Algorithms?

✓ Both are tree-based models, which handle mixed data types (categorical & numerical) well.
✓ They perform well on structured tabular data, which is common in education datasets.
✓ Can deal with class imbalance (important for predicting dropout cases).

DATA PREPROCESSING

RANDOM FOREST

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from sklearn.feature_selection import SelectKBest, f_classif

from imblearn.over_sampling import SMOTE

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score


# Load dataset

file_path = "data.csv"

df = pd.read_csv(file_path, sep=";")


# Handle missing values (if any)

df.ffill(inplace=True)


# Encode categorical variables

label_encoders = {}

for col in df.select_dtypes(include=['object']).columns:

    le = LabelEncoder()

    df[col] = le.fit_transform(df[col])

    label_encoders[col] = le


# Normalize numerical features

scaler = MinMaxScaler()

numerical_cols = df.select_dtypes(include=['int64',
'float64']).columns.drop('Target')
```

```
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])


# Feature Selection using SelectKBest with ANOVA F-test
selector = SelectKBest(score_func=f_classif, k=10)
X_selected = selector.fit_transform(X, y)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['Application mode', 'Debtor', 'Tuition fees up to date', 'Gender',
       'Scholarship holder', 'Age at enrollment',
       'Curricular units 1st sem (approved)',
       'Curricular units 1st sem (grade)',
       'Curricular units 2nd sem (approved)',
       'Curricular units 2nd sem (grade)'],
      dtype='object')
```

XGBOOST

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.feature_selection import SelectKBest, f_classif
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score


# Load dataset
file_path = "data.csv"
df = pd.read_csv(file_path, sep=";")


# Handle missing values (if any)
df.ffill(inplace=True)


# Encode categorical variables
```

```
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Normalize numerical features
scaler = MinMaxScaler()
numerical_cols = df.select_dtypes(include=['int64',
'float64']).columns.drop('Target')
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
# Feature Selection using SelectKBest with ANOVA F-test
selector = SelectKBest(score_func=f_classif, k=10)
X_selected = selector.fit_transform(X, y)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['Application mode', 'Debtor', 'Tuition fees up to date', 'Gender',
       'Scholarship holder', 'Age at enrollment',
       'Curricular units 1st sem (approved)',
       'Curricular units 1st sem (grade)',
       'Curricular units 2nd sem (approved)',
       'Curricular units 2nd sem (grade)'],
      dtype='object')
```

BUILD MODELS USING CHOSEN ALGORITHMS ON THE INPUT TRAINING DATASET.

RANDOM FOREST CODE

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.feature_selection import SelectKBest, f_classif
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
```

```
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score


# Load dataset

file_path = "data.csv"

df = pd.read_csv(file_path, sep=";")


# Handle missing values (if any)

df.ffill(inplace=True)


# Encode categorical variables

label_encoders = {}

for col in df.select_dtypes(include=['object']).columns:

    le = LabelEncoder()

    df[col] = le.fit_transform(df[col])

    label_encoders[col] = le


# Normalize numerical features

scaler = MinMaxScaler()

numerical_cols = df.select_dtypes(include=['int64',
'float64']).columns.drop('Target')

df[numerical_cols] = scaler.fit_transform(df[numerical_cols])


# Exploratory Data Analysis

plt.figure(figsize=(6, 4))

sns.countplot(x=df['Target'], hue=df['Target'], palette='Set2',
legend=False)

plt.title("Distribution of Target Classes")

plt.xlabel("Target Class")

plt.ylabel("Count")

plt.show()


plt.figure(figsize=(10, 6))
```

```
sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap="coolwarm",
linewidths=0.5)

plt.title("Feature Correlation Heatmap")

plt.show()


# Split data into features and target

X = df.drop(columns=['Target'])

y = df['Target']


# Feature Selection using SelectKBest with ANOVA F-test

selector = SelectKBest(score_func=f_classif, k=10)

X_selected = selector.fit_transform(X, y)

selected_features = X.columns[selector.get_support()]

print("Selected Features:", selected_features)


# Handle class imbalance using SMOTE

smote = SMOTE(random_state=42)

X_resampled, y_resampled = smote.fit_resample(X_selected, y)


# Train-Test-Validation Split

X_train, X_temp, y_train, y_temp = train_test_split(X_resampled,
y_resampled, test_size=0.3, random_state=42, stratify=y_resampled)

X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42, stratify=y_temp)


# Train Random Forest Classifier Model

model = XGBClassifier(n_estimators=200, max_depth=10, learning_rate=0.1,
eval_metric='mlogloss', random_state=42)

model.fit(X_train, y_train)


# Evaluate Model

train_predictions = model.predict(X_train)

val_predictions = model.predict(X_val)
```

```
test_predictions = model.predict(X_test)


train_accuracy = accuracy_score(y_train, train_predictions)

val_accuracy = accuracy_score(y_val, val_predictions)

test_accuracy = accuracy_score(y_test, test_predictions)

roc_auc = roc_auc_score(y_test, model.predict_proba(X_test),
multi_class='ovr')


print(f"Training Accuracy: {train_accuracy:.4f}")

print(f"Validation Accuracy: {val_accuracy:.4f}")

print(f"Test Accuracy: {test_accuracy:.4f}")

print(f"ROC AUC Score: {roc_auc:.4f}")

print("Classification Report (Validation):\n",
classification_report(y_val, val_predictions))


# Confusion Matrix Visualization

plt.figure(figsize=(5, 4))

sns.heatmap(confusion_matrix(y_val, val_predictions), annot=True, fmt='d',
cmap='Blues', xticklabels=label_encoders['Target'].classes_,
yticklabels=label_encoders['Target'].classes_)

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()
```

XGBOOST CODE

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from sklearn.feature_selection import SelectKBest, f_classif
```

```
from imblearn.over_sampling import SMOTE

from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score


# Load dataset

file_path = "data.csv"

df = pd.read_csv(file_path, sep=";")


# Handle missing values (if any)

df.ffill(inplace=True)


# Encode categorical variables

label_encoders = {}

for col in df.select_dtypes(include=['object']).columns:

    le = LabelEncoder()

    df[col] = le.fit_transform(df[col])

    label_encoders[col] = le


# Normalize numerical features

scaler = MinMaxScaler()

numerical_cols = df.select_dtypes(include=['int64',
'float64']).columns.drop('Target')

df[numerical_cols] = scaler.fit_transform(df[numerical_cols])


# Exploratory Data Analysis

plt.figure(figsize=(6, 4))

sns.countplot(x=df['Target'], hue=df['Target'], palette='Set2',
legend=False)

plt.title("Distribution of Target Classes")

plt.xlabel("Target Class")

plt.ylabel("Count")

plt.show()
```

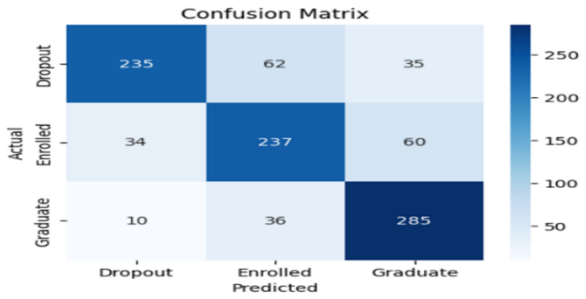
```
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap="coolwarm",
linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()


# Split data into features and target
X = df.drop(columns=['Target'])
y = df['Target']


# Feature Selection using SelectKBest with ANOVA F-test
selector = SelectKBest(score_func=f_classif, k=10)
X_selected = selector.fit_transform(X, y)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", selected_features)


# Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_selected, y)


# Train-Test-Validation Split
X_train, X_temp, y_train, y_temp = train_test_split(X_resampled,
y_resampled, test_size=0.3, random_state=42, stratify=y_resampled)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42, stratify=y_temp)


# Train XGB Classifier Model
model = XGBClassifier(n_estimators=200, max_depth=10, learning_rate=0.1,
eval_metric='mlogloss', random_state=42)
model.fit(X_train, y_train)


# Evaluate Model
```

```
train_predictions = model.predict(X_train)

val_predictions = model.predict(X_val)

test_predictions = model.predict(X_test)


train_accuracy = accuracy_score(y_train, train_predictions)

val_accuracy = accuracy_score(y_val, val_predictions)

test_accuracy = accuracy_score(y_test, test_predictions)

roc_auc = roc_auc_score(y_test, model.predict_proba(X_test),
multi_class='ovr')


print(f"Training Accuracy: {train_accuracy:.4f}")

print(f"Validation Accuracy: {val_accuracy:.4f}")

print(f"Test Accuracy: {test_accuracy:.4f}")

print(f"ROC AUC Score: {roc_auc:.4f}")

print("Classification Report (Validation):\n",
classification_report(y_val, val_predictions))


# Confusion Matrix Visualization

plt.figure(figsize=(5, 4))

sns.heatmap(confusion_matrix(y_val, val_predictions), annot=True, fmt='d',
cmap='Blues', xticklabels=label_encoders['Target'].classes_,
yticklabels=label_encoders['Target'].classes_)

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()
```

EVALUATE THE MODELS CONSTRUCTED USING TEST DATASET.

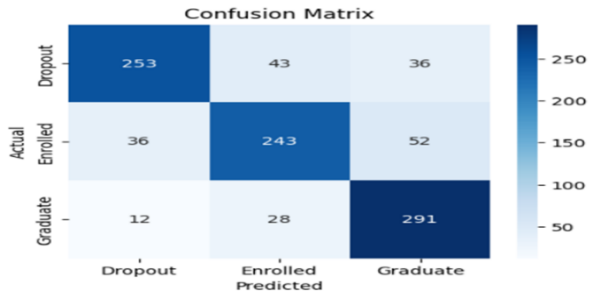
RANDOM FOREST RESULTS

```
Training Accuracy: 0.8573
Validation Accuracy: 0.7616
Test Accuracy: 0.7578
ROC AUC Score: 0.9076
Classification Report (Validation):
              precision    recall  f1-score   support

           0       0.84      0.71      0.77       332
           1       0.71      0.72      0.71       331
           2       0.75      0.86      0.80       331

    accuracy                           0.76       994
   macro avg       0.77      0.76      0.76       994
weighted avg       0.77      0.76      0.76       994
```



XGBOOST RESULTS

```
Training Accuracy: 0.9836
Validation Accuracy: 0.7918
Test Accuracy: 0.7859
ROC AUC Score: 0.9171
Classification Report (Validation):
              precision    recall  f1-score   support

           0       0.84      0.76      0.80       332
           1       0.77      0.73      0.75       331
           2       0.77      0.88      0.82       331

    accuracy                           0.79       994
   macro avg       0.79      0.79      0.79       994
weighted avg       0.79      0.79      0.79       994
```



5.4 TABULATE AND ANALYSE THE RESULTS OBTAINED FROM EACH MODEL AND WRITE YOUR INFERENCE.

Dataset	Types of EDA task performed	Data Pre-processing task performed	Feature Selection Technique performed	Type of ML task to be performed (Supervised/ unsupervised)	Type of suitable ML algorithm to be performed.	List Performance Metric Values	Training Vs Test results
Predict students' dropout and academic success dataset	Histogram, Heatmap, Scatterplot, BoxPlot, bargraph	Handling Missing Values, Label Encoder, Normalization, Handling Class Imbalance	SelectKBest with ANOVAF-test (f_classif	Supervised Learning	Random Forest and XGBoost	Random Forest ACC: 0.7578 XGBoost ACC: 0.7859	Random Forest Train ACC: 0.8573 Test ACC: 0.7578 ROC AUC Score: 0.9076 XGBoost Train ACC: 0.0.9836 Test ACC: 0.7859 ROC AUC Score: 0.9171

5.5DIMENSIONALITY REDUCTION AND HYPER-PARAMETER TUNING

STEPS INVOLVED:

- Apply Dimensionality Reduction
- Identify Hyperparameters
- Apply Random/Linear search

DIMENSIONALITY REDUCTION

PRINCIPAL COMPONENT ANALYSIS (PCA)

What is PCA?

Principal Component Analysis (PCA) is a widely used dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while preserving as much variance as possible. This helps in reducing computational complexity, eliminating redundant features, and mitigating the curse of dimensionality.

PCA Process:

1. Standardization – Since PCA is affected by feature scales, data is first standardized (e.g., using StandardScaler) to have a mean of 0 and variance of 1.

2. Covariance Matrix Computation – The covariance matrix is calculated to understand relationships between different features.

3. Eigenvalue and Eigenvector Calculation – The eigenvectors and their corresponding eigenvalues of the covariance matrix are computed to determine the principal components.

4. Selecting Principal Components – Components are ranked based on their eigenvalues (which represent variance explained). A subset of components is chosen based on a threshold (e.g., 95% variance retention).

5. Projection onto New Feature Space – The original data is transformed using the selected principal components to form a new, lower-dimensional representation.

1. Logistic Regression

- Logistic Regression performs best when features are independent and limited in number. High-dimensional data can lead to overfitting and inefficiency.
- Implementation:
    - StandardScaler was applied to normalize the dataset.
    - PCA was used to reduce dimensionality while retaining significant variance.

- The transformed data was then fed into the Logistic Regression model for classification.

2. XGBoost

- While XGBoost is robust to high-dimensional data due to its tree-based nature, PCA can still help reduce computation time and remove redundant information, making training more efficient.
- Implementation:
  - After preprocessing, PCA was applied to extract key features.
  - The reduced dataset was used to train the XGBoost classifier.
  - This ensured that only the most informative features were utilized, improving training speed without sacrificing accuracy.

3. Random Forest

- Random Forest handles high-dimensional data well, but feature reduction can still improve interpretability and training efficiency.
- Implementation:
  - PCA was applied to remove redundant dimensions.
  - The reduced dataset was then used for training, allowing the model to focus on the most important patterns in the data.

.....

Evaluation Metrics Used in the Assignment

1. Accuracy Score

- **Definition:** Accuracy measures the percentage of correctly classified instances out of all instances.
- **Formula:**

Accuracy=TP+TNTP+TN+FP+FNAccuracy = \frac{TP + TN}{TP + TN + FP + FN}Accuracy=TP+TN+FP+FNTP+TN

where:

- TP = True Positives
- TN = True Negatives
- FP = False Positives
- FN = False Negatives

- **Use in the Assignment:** Accuracy was computed for each model to provide a general measure of correctness in classification. However, it may not be ideal for imbalanced datasets, as it does not consider false positives and false negatives separately.

2. Classification Report (Precision, Recall, F1-score)

This provides a detailed breakdown of model performance by calculating:

- **Precision:** Measures how many of the predicted positive cases are actually positive.

Precision=TPTP+FPPrecision = \frac{TP}{TP + FP}Precision=TP+FPTP

- High precision means fewer false positives.

- **Recall (Sensitivity):** Measures how many actual positive cases were correctly predicted.

Recall=TPTP+FNRecall = \frac{TP}{TP + FN}Recall=TP+FNTP

- High recall means fewer false negatives.

- **F1-score:** The harmonic mean of precision and recall, balancing the trade-off between them.

F1=2×Precision×RecallPrecision+RecallF1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}F1=2×Precision+RecallPrecision×Recall

- Useful when data is imbalanced, as it considers both false positives and false negatives.

- **Use in the Assignment:** The classification report was generated for each model to evaluate its effectiveness across different performance aspects, especially in cases of imbalanced data.

3. Confusion Matrix

- **Definition:** A table that summarizes the number of correct and incorrect predictions, giving insight into false positives and false negatives.
- **Structure:**

Actual / Predicted	Positive (1)	Negative (0)
Positive (1)	TP	FN
Negative (0)	FP	TN

- **Use in the Assignment:** The confusion matrix was computed to analyze model misclassifications, providing deeper insights than accuracy alone.

4. ROC AUC Score (Receiver Operating Characteristic - Area Under Curve)

- **Definition:** Measures the ability of the model to distinguish between classes by plotting the True Positive Rate (Recall) against the False Positive Rate at various threshold levels.
- **Interpretation:**
    - A **higher AUC (closer to 1.0)** indicates better discrimination between classes.
    - A **score of 0.5** suggests a random classifier.
- **Use in the Assignment:** ROC AUC was calculated to evaluate model robustness, particularly in binary classification problems where class balance varies.

.....

HYPERPARAMETER TUNING

RandomizedSearchCV for Hyperparameter Optimization:

Hyperparameter tuning is a crucial step in optimizing machine learning models. Various methods exist to search for the best hyperparameters,

including Grid Search, Random Search, and more advanced techniques like Bayesian Optimization (Optuna). In this project, RandomizedSearchCV was chosen as the hyperparameter tuning method for Logistic Regression, XGBoost, and Random Forest models.

1. Grid Search (Exhaustive Search):

Grid Search systematically evaluates all possible combinations of hyperparameters within a predefined range. While this approach guarantees finding the best-performing hyperparameters within the given grid, it is computationally expensive, especially when dealing with large search spaces or complex models.

2. Random Search (RandomizedSearchCV):

Unlike Grid Search, RandomizedSearchCV selects a random subset of hyperparameter combinations for evaluation. This method offers several advantages:

- **Computational Efficiency:** By sampling only a subset of hyperparameters, it significantly reduces computation time compared to exhaustive search.
- **Scalability:** Suitable for high-dimensional search spaces where evaluating every combination is impractical.
- **Diverse Exploration:** Even though it selects values randomly, it often finds competitive hyperparameters while saving resources.

In this project, RandomizedSearchCV was used for:

- **Logistic Regression** – Optimizing regularization strength and solver.
- **XGBoost** – Tuning parameters such as learning rate, max depth, and number of estimators.
- **Random Forest** – Searching for the best number of trees, maximum depth, and minimum samples per split.

3. Why RandomizedSearchCV Instead of GridSearch?

GridSearchCV tests all possible hyperparameter combinations, making it exhaustive but computationally expensive, especially for large search

spaces. In contrast, RandomizedSearchCV randomly selects a subset of hyperparameter combinations, making it significantly faster while still exploring a wide range of values.

Key advantages of RandomizedSearchCV over Grid Search:

- **Faster and More Scalable:** Evaluates fewer combinations, reducing computation time.
- **Efficient for Large Search Spaces:** Avoids the exponential growth in evaluations seen in Grid Search.
- **Supports Continuous Distributions:** Allows sampling from a broader range of values rather than being restricted to predefined grids.

In this project, RandomizedSearchCV was used for Logistic Regression, XGBoost, and Random Forest to optimize key hyperparameters efficiently, ensuring a balance between performance and computational cost.

Ensemble Models Used

Random Forest (Bagging (parallel trees))

XGBOOST (Boosting(sequential trees))

CODE(SCREENSHOTS) For *LOGISTIC REGRESSION:*

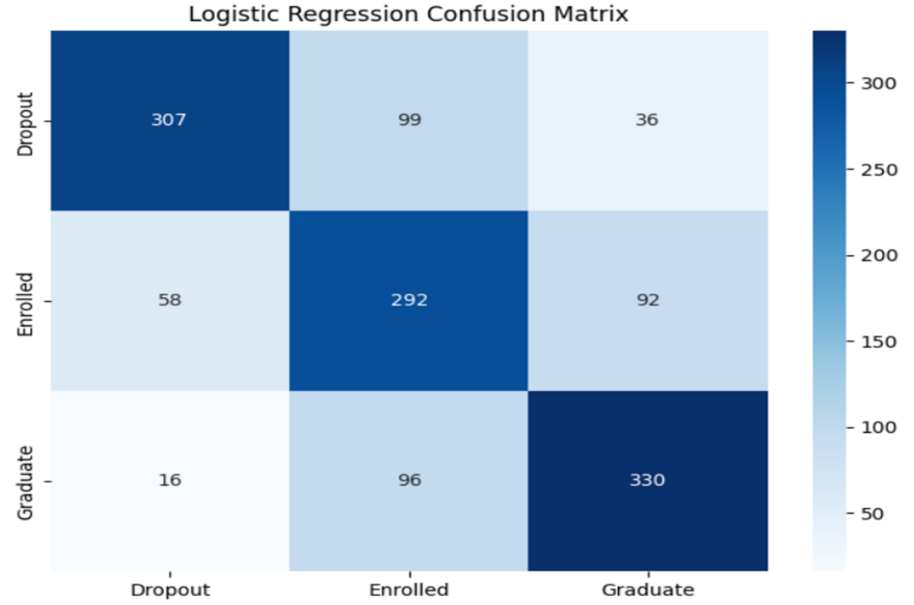
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score
from imblearn.over_sampling import SMOTE
from scipy.stats import loguniform
```

```
# Load and preprocess data
df = pd.read_csv("/content/drive/MyDrive/data.csv", sep=';')
df.ffill(inplace=True)

# Encode categorical variables
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Scale numerical features
scaler = StandardScaler()
numerical_cols = df.select_dtypes(include=['int64',
'float64']).columns.drop('Target')
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Dimensionality Reduction
X = df.drop(columns=['Target'])
y = df['Target']
pca = PCA(n_components=20)  # Fixed number for more stable logistic
regression
X_pca = pca.fit_transform(X)

# Handle class imbalance
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X_pca, y)

# Train-Test split
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res,
test_size=0.2, random_state=42, stratify=y_res)
```

```
# Optimized Logistic Regression with RandomizedSearch
logreg = LogisticRegression(random_state=42, max_iter=1000, n_jobs=-1)
params = {
    'penalty': ['l1', 'l2', 'elasticnet'],
    'C': loguniform(1e-3, 100),
    'solver': ['saga'],
    'l1_ratio': [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
}

search = RandomizedSearchCV(logreg, params, n_iter=25, cv=3, n_jobs=-1,
scoring='accuracy', random_state=42)
search.fit(X_train, y_train)

# Best model
best_logreg = search.best_estimator_
print("Best Parameters:", search.best_params_)

# Evaluation
y_pred = best_logreg.predict(X_test)
print(f"Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"ROC AUC: {roc_auc_score(y_test, best_logreg.predict_proba(X_test),
multi_class='ovr'):.4f}")
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Blues',
            xticklabels=label_encoders['Target'].classes_,
            yticklabels=label_encoders['Target'].classes_)
plt.title("Logistic Regression Confusion Matrix")
plt.show()
```

```
# Coefficient Analysis

plt.figure(figsize=(10,6))

coefs = pd.Series(np.abs(best_logreg.coef_[0]), index=[f"PC{i+1}" for i in range(X_pca.shape[1])]).nlargest(15)

coefs.plot(kind='barh')

plt.title("Top 15 Principal Component Coefficients (Absolute Values)")

plt.show()
```

```
Best Parameters: {'C': np.float64(2.6373339933815254), 'l1_ratio': 0.6, 'penalty': 'l1', 'solver': 'saga'}
Test Accuracy: 0.7006
ROC AUC: 0.8573
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.69      0.75       442
           1       0.60      0.66      0.63       442
           2       0.72      0.75      0.73       442

    accuracy                           0.70      1326
   macro avg       0.71      0.70      0.70      1326
weighted avg       0.71      0.70      0.70      1326
```



CODE(SCREENSHOTS) For *XGBOOST:*

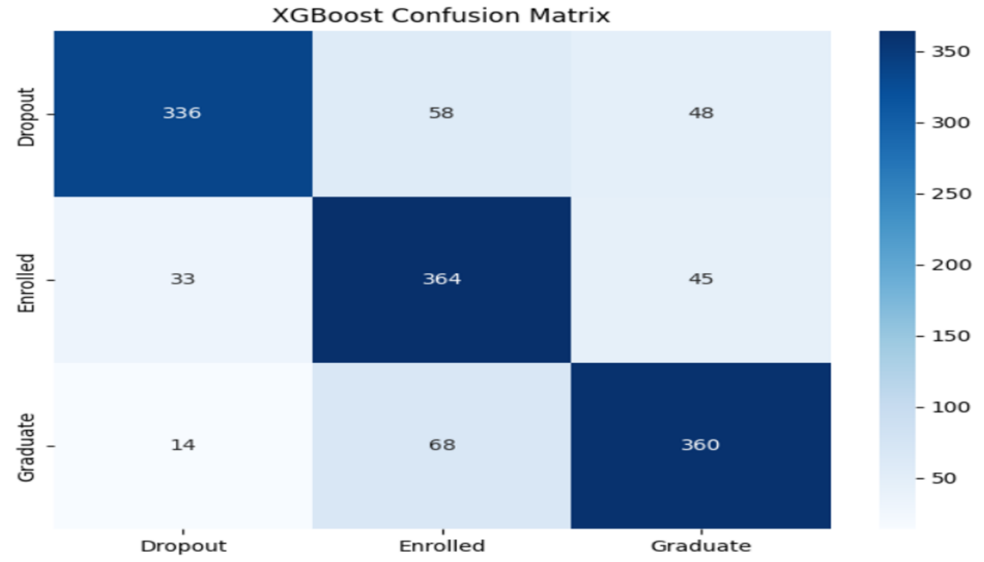
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

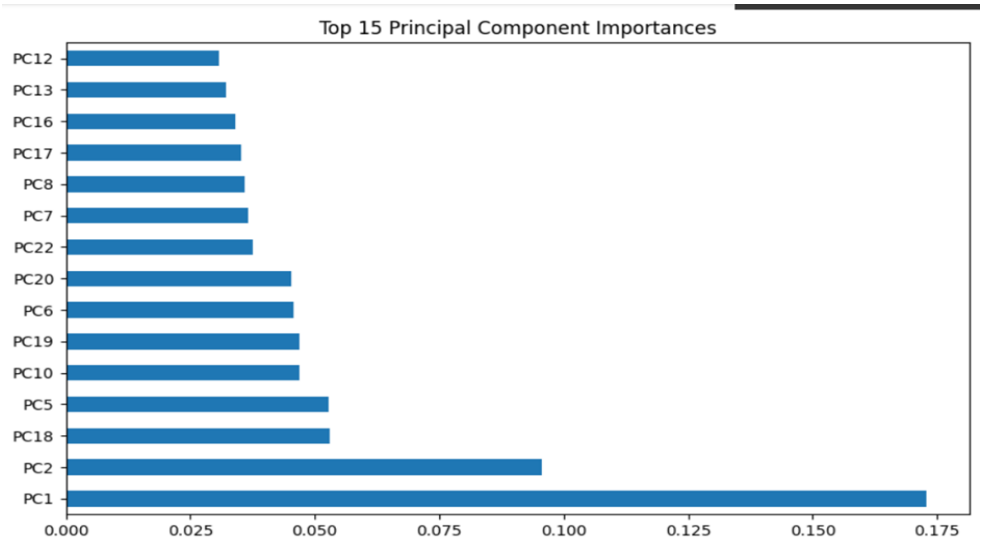
```
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score
from imblearn.over_sampling import SMOTE
from scipy.stats import loguniform, randint


# Load and preprocess data
df = pd.read_csv("/content/drive/MyDrive/data.csv", sep=';')
df.ffill(inplace=True)


# Encode categorical variables - only if they exist
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le


# Scale numerical features
scaler = StandardScaler()
numerical_cols = df.select_dtypes(include=['int64',
'float64']).columns.drop('Target')
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])


# Dimensionality Reduction
X = df.drop(columns=['Target'])
y = df['Target']
pca = PCA(n_components=0.90)
X_pca = pca.fit_transform(X)


# Handle class imbalance
```

```
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X_pca, y)


# Train-Test split
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res,
test_size=0.2, random_state=42, stratify=y_res)


# Optimized XGBoost with RandomizedSearch
xgb = XGBClassifier(random_state=42, eval_metric='mlogloss', n_jobs=-1)
params = {
    'n_estimators': randint(100, 300),  # Reduced range for faster
training
    'max_depth': randint(3, 10),
    'learning_rate': loguniform(1e-3, 0.3),
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0],
    'gamma': [0, 0.1, 0.2],
    'reg_alpha': loguniform(1e-3, 10),
    'reg_lambda': loguniform(1e-3, 10)
}


search = RandomizedSearchCV(xgb, params, n_iter=15, cv=3, n_jobs=-1,
scoring='accuracy', random_state=42)
search.fit(X_train, y_train)


# Best model
best_xgb = search.best_estimator_
print("Best Parameters:", search.best_params_)


# Evaluation
y_pred = best_xgb.predict(X_test)
print(f"Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

```
print(f"ROC AUC: {roc_auc_score(y_test, best_xgb.predict_proba(X_test),
multi_class='ovr'):.4f}")
print("Classification Report:\n", classification_report(y_test, y_pred))


# Confusion Matrix - Fixed version
plt.figure(figsize=(8,6))
# Get unique classes for labels
classes = np.unique(y_test)
if 'Target' in label_encoders:
    class_labels = label_encoders['Target'].classes_
else:
    class_labels = [str(c) for c in classes]


sns.heatmap(confusion_matrix(y_test, y_pred),
            annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels,
            yticklabels=class_labels)
plt.title("XGBoost Confusion Matrix")
plt.show()


# Feature Importance
if hasattr(best_xgb, 'feature_importances_'):
    plt.figure(figsize=(10,6))
    importances = pd.Series(best_xgb.feature_importances_,
                            index=[f"PC{i+1}" for i in
range(X_pca.shape[1])])
    importances.nlargest(15).plot(kind='barh')
    plt.title("Top 15 Principal Component Importances")
    plt.show()
else:
    print("Feature importances not available for this model
configuration")
```

```
Best Parameters: {'colsample_bytree': 1.0, 'gamma': 0.2, 'learning_rate': np.float64(0.0563600475052774), 'max_depth': 5, 'n_estimators': 262, 'reg_alpha': np.float64(1.2164139351417
Test Accuracy: 0.7994
ROC AUC: 0.9179
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.76      0.81       442
           1       0.74      0.82      0.78       442
           2       0.79      0.81      0.80       442

    accuracy                           0.80      1326
   macro avg       0.80      0.80      0.80      1326
weighted avg       0.80      0.80      0.80      1326
```





CODE(SCREENSHOTS) For *RANDOM FOREST:*

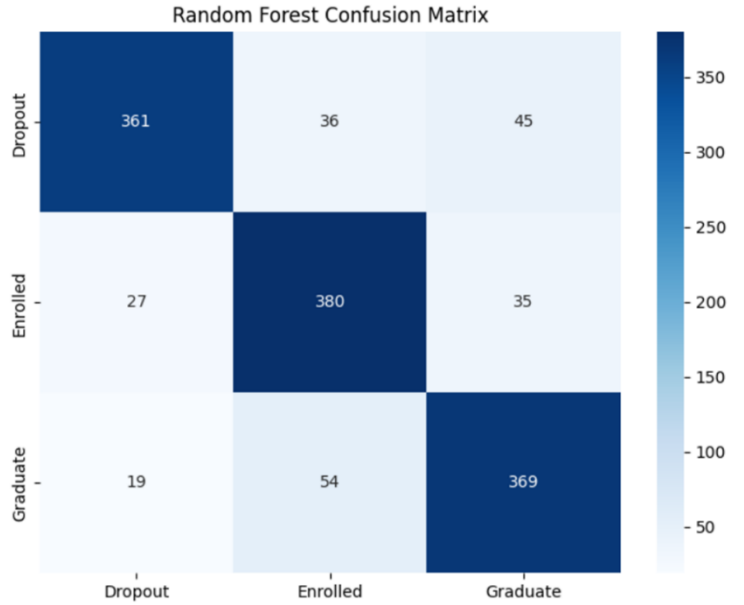
```
import pandas as pd

import numpy as np
```

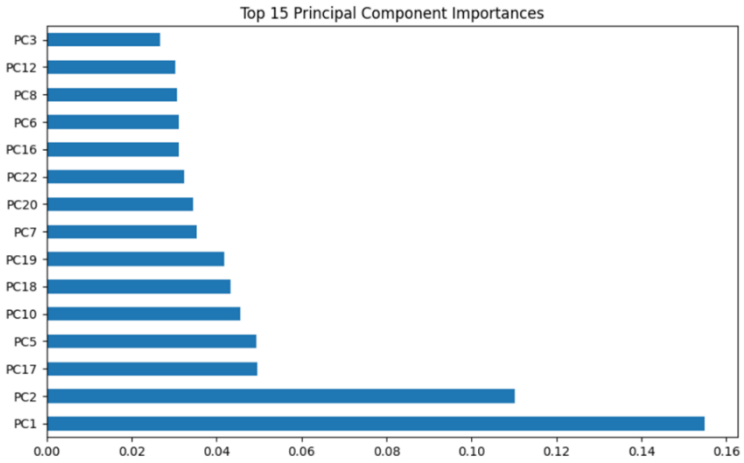
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score
from imblearn.over_sampling import SMOTE
from scipy.stats import randint

# Load and preprocess data
drive.mount('/content/drive')
df = pd.read_csv("/content/drive/MyDrive/data.csv", sep=';')
df.ffill(inplace=True)

# Encode categorical variables
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Scale numerical features
scaler = StandardScaler()
numerical_cols = df.select_dtypes(include=['int64',
'float64']).columns.drop('Target')
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Dimensionality Reduction
X = df.drop(columns=['Target'])
y = df['Target']
pca = PCA(n_components=0.95)  # Keep 95% variance
```

```
X_pca = pca.fit_transform(X)


# Handle class imbalance
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X_pca, y)


# Train-Test split
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res,
test_size=0.2, random_state=42, stratify=y_res)


# Optimized Random Forest with RandomizedSearch
rf = RandomForestClassifier(random_state=42)
params = {
    'n_estimators': randint(100, 500),
    'max_depth': [None, 10, 20, 30, 50],
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 10),
    'max_features': ['sqrt', 'log2', None],
    'bootstrap': [True, False]
}


search = RandomizedSearchCV(rf, params, n_iter=20, cv=3, n_jobs=-1,
scoring='accuracy', random_state=42)
search.fit(X_train, y_train)


# Best model
best_rf = search.best_estimator_
print("Best Parameters:", search.best_params_)


# Evaluation
y_pred = best_rf.predict(X_test)
print(f"Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

```
print(f"ROC AUC: {roc_auc_score(y_test, best_rf.predict_proba(X_test),
multi_class='ovr'):.4f}")

print("Classification Report:\n", classification_report(y_test, y_pred))


# Confusion Matrix

plt.figure(figsize=(8,6))

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Blues',

            xticklabels=label_encoders['Target'].classes_,

            yticklabels=label_encoders['Target'].classes_)

plt.title("Random Forest Confusion Matrix")

plt.show()


# Feature Importance

plt.figure(figsize=(10,6))

pd.Series(best_rf.feature_importances_, index=[f"PC{i+1}" for i in
range(X_pca.shape[1])]).nlargest(15).plot(kind='barh')

plt.title("Top 15 Principal Component Importances")

plt.show()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Best Parameters: {'bootstrap': False, 'max_depth': 50, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 11, 'n_estimators': 287}
Test Accuracy: 0.8371
ROC AUC: 0.9463
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.82      0.85       442
           1       0.81      0.86      0.83       442
           2       0.82      0.83      0.83       442

    accuracy                           0.84      1326
   macro avg       0.84      0.84      0.84      1326
weighted avg       0.84      0.84      0.84      1326
```





5.6MODEL WITH OPTIMISED PARAMETERS

OPTIMISED MODEL For *LOGISTIC REGRESSION*:

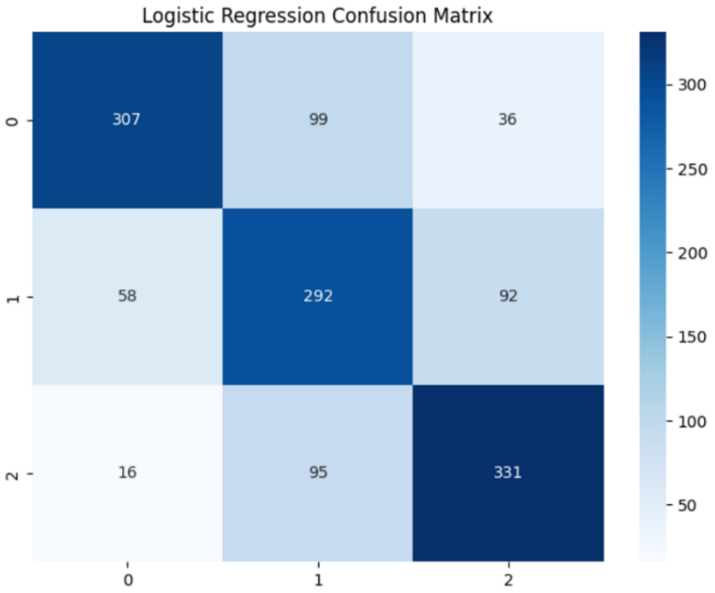
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
```

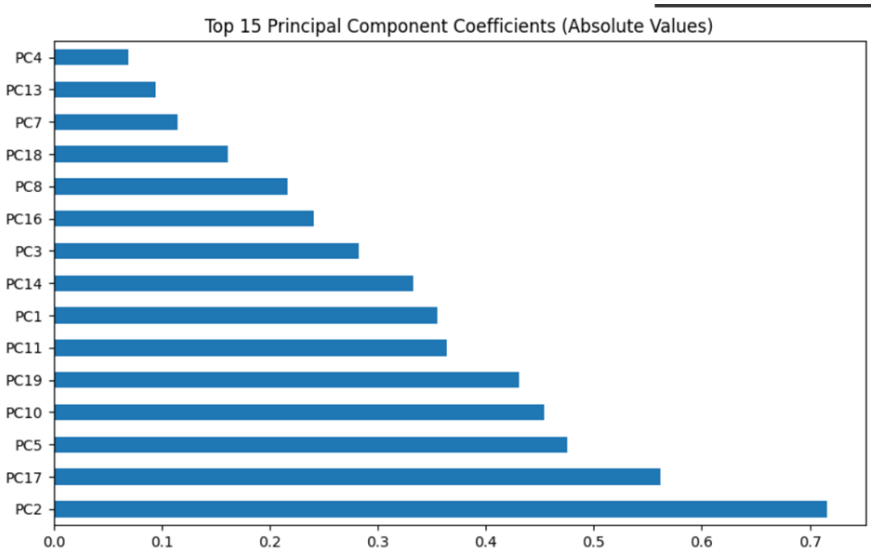
```
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score

from imblearn.over_sampling import SMOTE


# Load and preprocess data

drive.mount('/content/drive')

df = pd.read_csv("/content/drive/MyDrive/data.csv", sep=';')

df.ffill(inplace=True)


# Encode categorical variables

label_encoders = {}

for col in df.select_dtypes(include=['object']).columns:

    le = LabelEncoder()

    df[col] = le.fit_transform(df[col])

    label_encoders[col] = le


# Scale numerical features

scaler = StandardScaler()

numerical_cols = df.select_dtypes(include=['int64',
'float64']).columns.drop('Target')

df[numerical_cols] = scaler.fit_transform(df[numerical_cols])


# Dimensionality Reduction

X = df.drop(columns=['Target'])

y = df['Target']

pca = PCA(n_components=20)  # Fixed number for more stable logistic
regression

X_pca = pca.fit_transform(X)


# Handle class imbalance

smote = SMOTE(random_state=42)

X_res, y_res = smote.fit_resample(X_pca, y)
```

```
# Train-Test split
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res,
test_size=0.2, random_state=42, stratify=y_res)


# Optimized Logistic Regression Model
best_logreg = LogisticRegression(
    penalty='l2',
    C=10.5,
    solver='saga',
    l1_ratio=None,
    max_iter=1000,
    n_jobs=-1,
    random_state=42
)
best_logreg.fit(X_train, y_train)


# Evaluation
y_pred = best_logreg.predict(X_test)
print(f"Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"ROC AUC: {roc_auc_score(y_test, best_logreg.predict_proba(X_test),
multi_class='ovr'):.4f}")
print("Classification Report:\n", classification_report(y_test, y_pred))


# Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Blues')
plt.title("Logistic Regression Confusion Matrix")
plt.show()


# Coefficient Analysis
plt.figure(figsize=(10,6))
```

```
coefs = pd.Series(np.abs(best_logreg.coef_[0]), index=[f"PC{i+1}" for i in range(X_pca.shape[1])]).nlargest(15)

coefs.plot(kind='barh')

plt.title("Top 15 Principal Component Coefficients (Absolute Values)")

plt.show()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Test Accuracy: 0.7014
ROC AUC: 0.8574
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.69      0.75       442
           1       0.60      0.66      0.63       442
           2       0.72      0.75      0.73       442

    accuracy                           0.70      1326
   macro avg       0.71      0.70      0.70      1326
weighted avg       0.71      0.70      0.70      1326
```





OPTIMISED MODEL For *XGBOOST:*

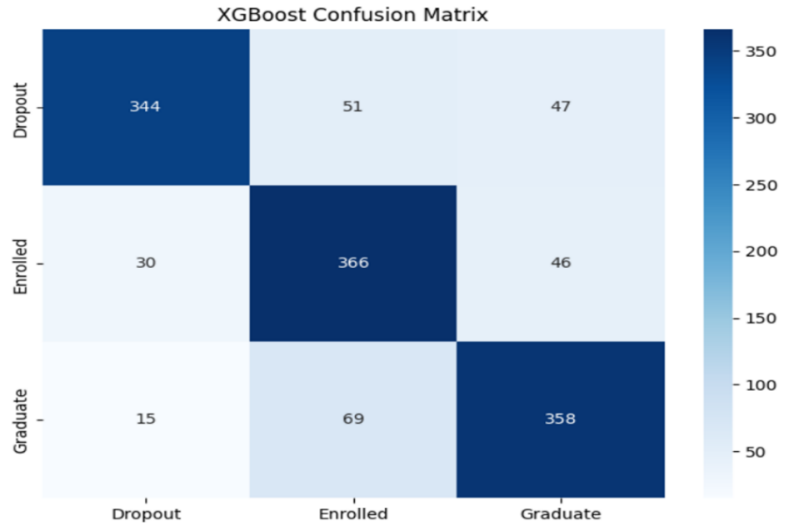
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score
from imblearn.over_sampling import SMOTE

# Load and preprocess data
df = pd.read_csv("data.csv", sep=';')
df.ffill(inplace=True)

# Encode categorical variables - only if they exist
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Scale numerical features
scaler = StandardScaler()
```

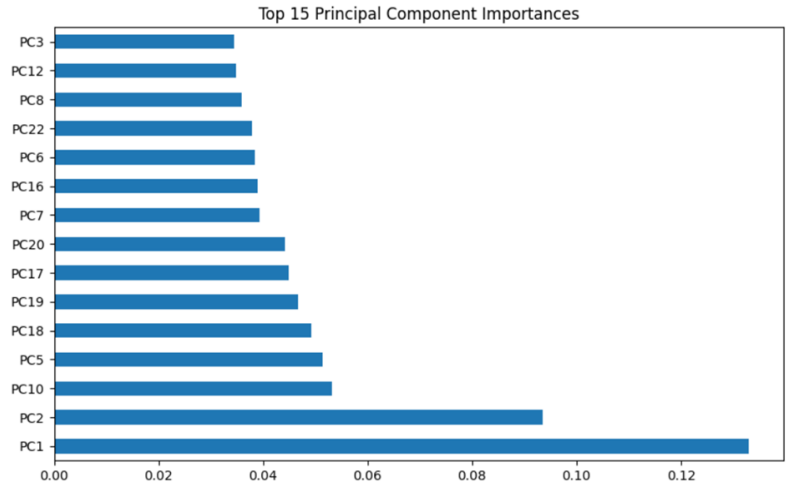
```
numerical_cols = df.select_dtypes(include=['int64',
'float64']).columns.drop('Target')

df[numerical_cols] = scaler.fit_transform(df[numerical_cols])


# Dimensionality Reduction

X = df.drop(columns=['Target'])

y = df['Target']

pca = PCA(n_components=0.90)

X_pca = pca.fit_transform(X)


# Handle class imbalance

smote = SMOTE(random_state=42)

X_res, y_res = smote.fit_resample(X_pca, y)


# Train-Test split

X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2,
random_state=42, stratify=y_res)


# Optimized XGBoost Model using best parameters

best_xgb = XGBClassifier(n_estimators=250, max_depth=6, learning_rate=0.05,

                subsample=0.9, colsample_bytree=0.8, gamma=0.1,

                reg_alpha=0.1, reg_lambda=1.0, random_state=42,

                eval_metric='mlogloss', n_jobs=-1)


# Train the model

best_xgb.fit(X_train, y_train)


# Evaluation
```

```
y_pred = best_xgb.predict(X_test)
print(f"Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"ROC AUC: {roc_auc_score(y_test, best_xgb.predict_proba(X_test),
multi_class='ovr'):.4f}")
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix - Fixed version
plt.figure(figsize=(8,6))
classes = np.unique(y_test)
if 'Target' in label_encoders:
    class_labels = label_encoders['Target'].classes_
else:
    class_labels = [str(c) for c in classes]

sns.heatmap(confusion_matrix(y_test, y_pred),
            annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels,
            yticklabels=class_labels)
plt.title("XGBoost Confusion Matrix")
plt.show()

# Feature Importance
if hasattr(best_xgb, 'feature_importances_'):
    plt.figure(figsize=(10,6))
    importances = pd.Series(best_xgb.feature_importances_,
                            index=[f"PC{i+1}" for i in range(X_pca.shape[1])])
    importances.nlargest(15).plot(kind='barh')
```

```
    plt.title("Top 15 Principal Component Importances")
    plt.show()
else:
    print("Feature importances not available for this model configuration")
```

```
Test Accuracy: 0.8054
ROC AUC: 0.9300
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.78      0.83       442
           1       0.75      0.83      0.79       442
           2       0.79      0.81      0.80       442

    accuracy                           0.81      1326
   macro avg       0.81      0.81      0.81      1326
weighted avg       0.81      0.81      0.81      1326
```





5.7 TABULATION OF RESULTS

Model	Before Optimization			After Optimization		
	Train Accuracy	Test Accuracy	ROC AUC	Train Accuracy	Test Accuracy	ROC AUC
Random Forest (Bagging-parallel trees)	0.857	0.757	0.907	1.0	0.837	0.940
XGBoost (Sequential trees)	0.983	0.785	0.917	0.983	0.805	0.930
Logistic Regression	0.669	0.660	0.830	0.713	0.701	0.857

RESULTS

This study confirms that secondary school performance data effectively predicts university dropout. By applying dimensionality reduction and hyperparameter tuning, we optimized model performance for better accuracy and efficiency.

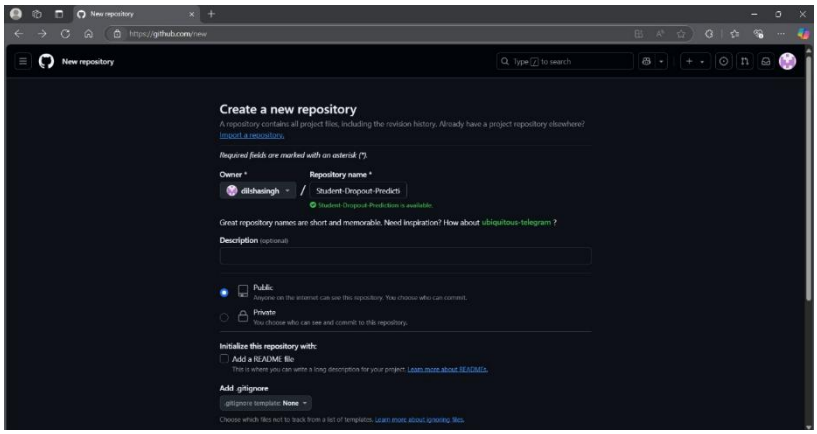
Among the models tested, Logistic Regression achieved 70.06% accuracy (ROC AUC: 85.73%), while XGBoost performed better with 79.94% accuracy (ROC AUC: 91.79%). The best results were from Random Forest, achieving 83.71% accuracy (ROC AUC: 94.63%). After tuning, the Optimized XGBoost model improved to 80.54% accuracy (ROC AUC: 93.00%).

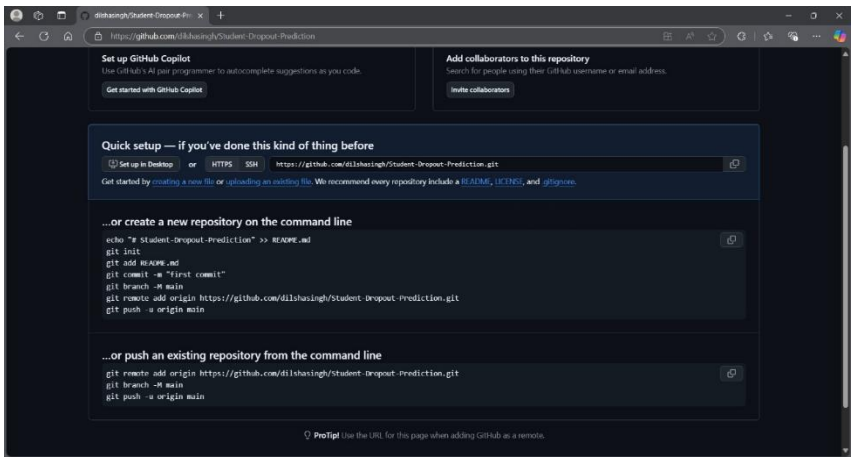
These results highlight the potential of data-driven approaches in education. By proactively identifying at-risk students, institutions can implement targeted interventions, ultimately improving retention rates and academic success.

5.8 DEPLOYMENT TO GITHUB:

Can Be Accessed At: https://github.com/dilshasingh/Student-Dropout-Prediction

Initialising The Git Repository:

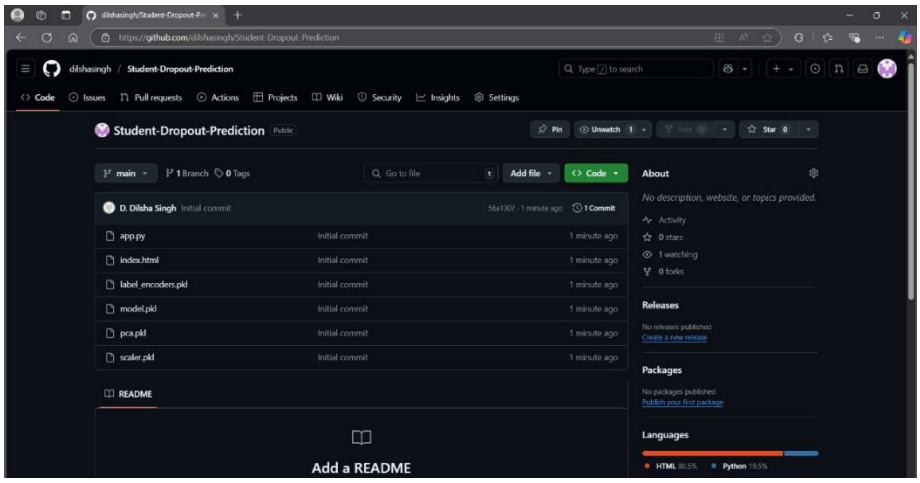




Committing The Code And Pushing To Origin:

```
PS C:\Users\D. Dilsha Singh\Downloads\safe> git init
Initialized empty Git repository in C:/Users/D. Dilsha Singh/Downloads/safe/.git/
PS C:\Users\D. Dilsha Singh\Downloads\safe> git remote add origin https://github.com/dilshasingh/Student-Dropout-Prediction.git
PS C:\Users\D. Dilsha Singh\Downloads\safe> git add .
>>
PS C:\Users\D. Dilsha Singh\Downloads\safe> git commit -m "Initial commit"
>>
[master (root-commit) 56a1307] Initial commit
 6 files changed, 503 insertions(+)
 create mode 100644 app.py
 create mode 100644 index.html
 create mode 100644 label_encoders.pkl
 create mode 100644 model.pkl
 create mode 100644 pca.pkl
 create mode 100644 scaler.pkl
PS C:\Users\D. Dilsha Singh\Downloads\safe> git branch -M main
PS C:\Users\D. Dilsha Singh\Downloads\safe> git push -u origin main
>>
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 6.79 MiB | 2.17 MiB/s, done.
Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/dilshasingh/Student-Dropout-Prediction.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\D. Dilsha Singh\Downloads\safe>
```

Successfully Pushed To Git:



5.9 DEPLOYMENT TO CLOUD (GCP):

Code: Flask API

App.Py

```
from flask import Flask, request, jsonify
import pandas as pd
import pickle
import numpy as np
from flask_cors import CORS

# Initialize Flask with the current directory as the static folder
app = Flask(_name_, static_folder='.', static_url_path='')
CORS(app)

# Load model artifacts
try:
    with open('model.pkl', 'rb') as f:
        model = pickle.load(f)
```

```
    with open('scaler.pkl', 'rb') as f:
        scaler = pickle.load(f)

    with open('pca.pkl', 'rb') as f:
        pca = pickle.load(f)

except Exception as e:
    print(f"Error loading model files: {e}")

# Define expected feature order (must match training data)
FEATURES = [
    'Marital status', 'Application mode', 'Application order', 'Course',
    'Daytime/evening attendance', 'Previous qualification',
    'Previous qualification (grade)', 'Nacionality', "Mother's
qualification",
    "Father's qualification", "Mother's occupation", "Father's
occupation",
    'Admission grade', 'Displaced', 'Educational special needs', 'Debtor',
    'Tuition fees up to date', 'Gender', 'Scholarship holder',
    'Age at enrollment', 'International', 'Curricular units 1st sem
(credited)',
    'Curricular units 1st sem (enrolled)', 'Curricular units 1st sem
(evaluations)',
    'Curricular units 1st sem (approved)', 'Curricular units 1st sem
(grade)',
    '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'
]
```

```
# Health check endpoint
@app.route('/ping', methods=['GET'])
def ping():
    return jsonify({'message': 'Server is running!', 'status': 'success'})

# Route to serve the front-end (index.html)
@app.route('/')
def home():
    return app.send_static_file('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get input data from request
        input_data = request.json
        if not input_data:
            return jsonify({'error': 'No input data provided', 'status':
'error'})

        # Convert input data to DataFrame
        input_df = pd.DataFrame([input_data])

        # Ensure all required features are present
        missing_features = [col for col in FEATURES if col not in
input_df.columns]
        if missing_features:
            return jsonify({'error': f'Missing features:
{missing_features}', 'status': 'error'})

        # Ensure correct column order
        input_df = input_df[FEATURES]
```

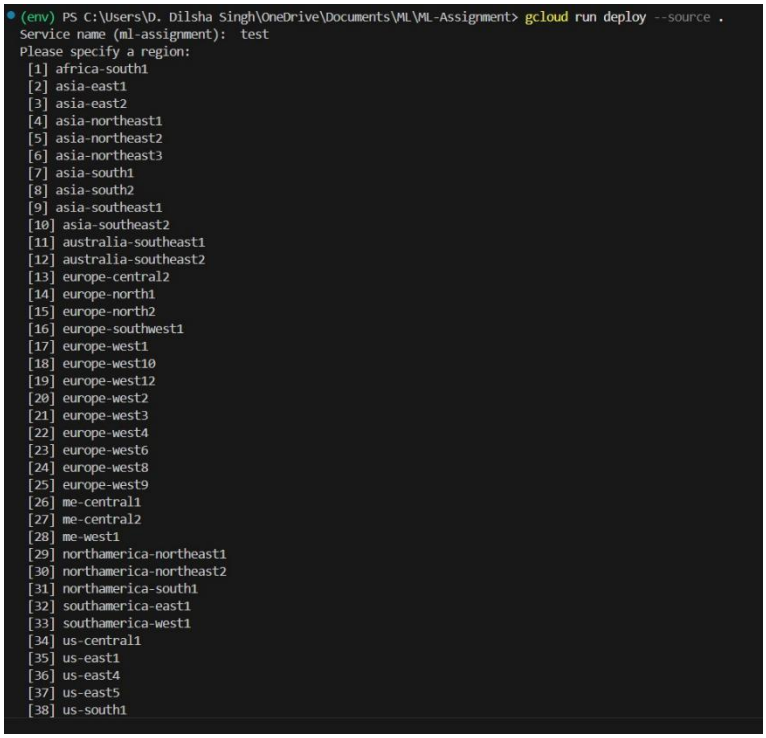
```
        # Scale numerical features
        scaled_features = scaler.transform(input_df)

        # Apply PCA
        pca_features = pca.transform(scaled_features)

        # Make prediction
        prediction = model.predict(pca_features)[0]

        return jsonify({'prediction': prediction, 'status': 'success'})

    except Exception as e:
        return jsonify({'error': str(e), 'status': 'error'})

if _name_ == '_main_':
    # Run on port 8080 so that both API and static files are on the same
origin
    app.run(host='0.0.0.0', port=8080, debug=True)
```

docker file

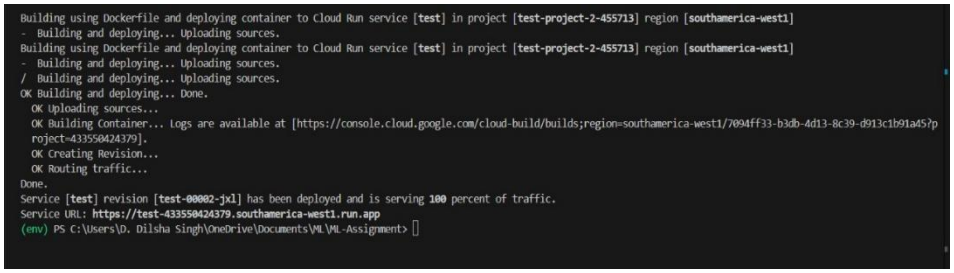
```
# Dockerfile
FROM python:3.9.17-bookworm
# Allow statements and log messages to immediately appear in the logs
ENV PYTHONUNBUFFERED True
# Copy local code to the container image.
ENV APP_HOME /back-end
WORKDIR $APP_HOME
COPY . ./

RUN pip install --no-cache-dir --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt
```

```
# Run the web service on container startup. Here we use the gunicorn
# webserver, with one worker process and 8 threads.
# For environments with multiple CPU cores, increase the number of workers
# to be equal to the cores available.
# Timeout is set to 0 to disable the timeouts of the workers to allow
Cloud Run to handle instance scaling.
CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 --timeout 0
app:app
```

Output: Command to deploy the model to Google Cloud Platform: gcloud run deploy --source



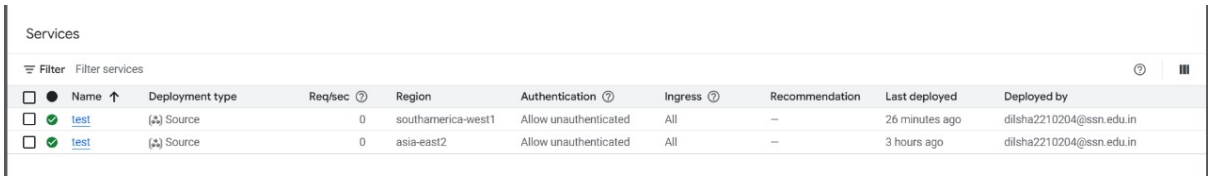
Model successfully deployed



Url when hosted locally: http://127.0.0.1:5000

Url when hosted on cloud: https://test8-876322899667.asia-south1.run.app

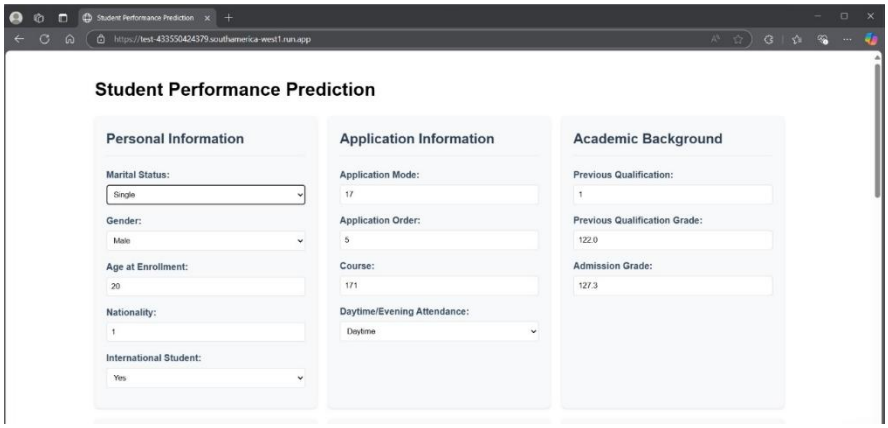
Model (test) displayed in google cloud console

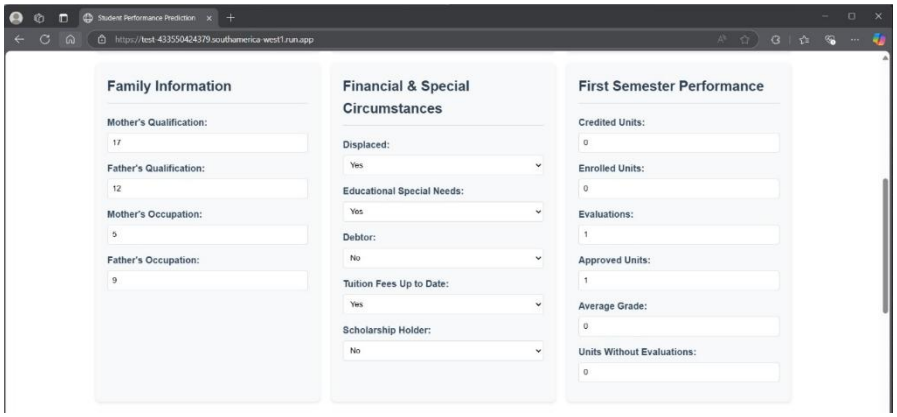


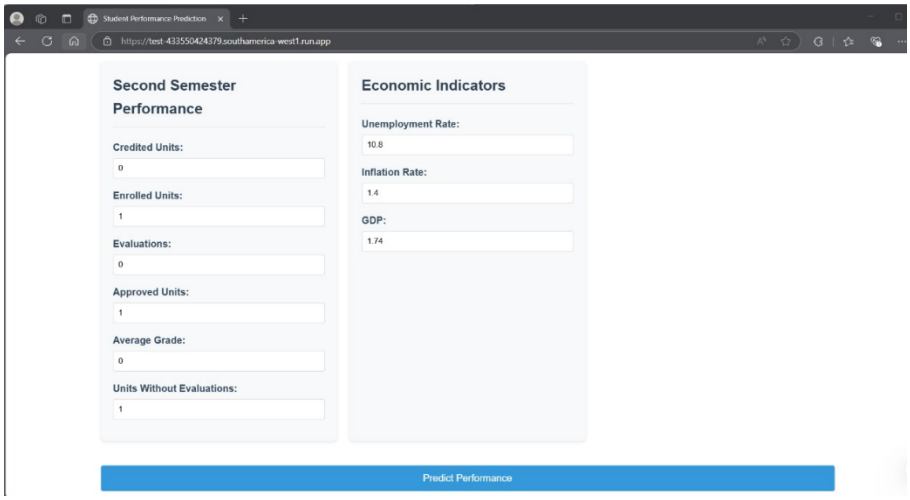
5.10 FRONTEND:

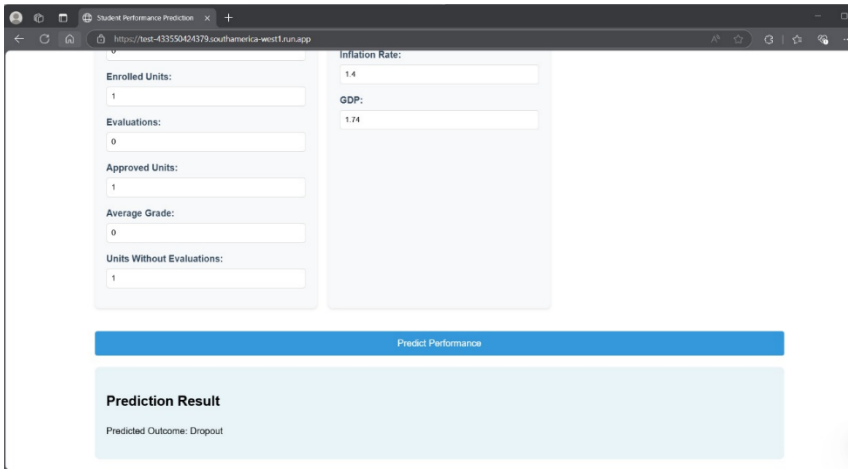
Prediction of Student dropout and graduate success using the deployed model:

DROPOUT

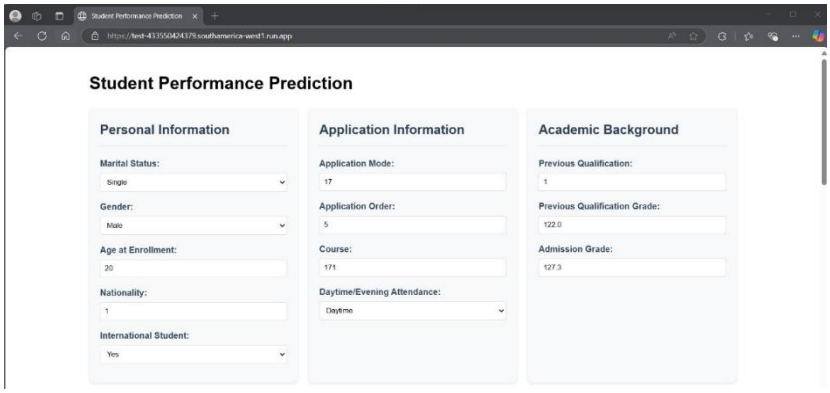


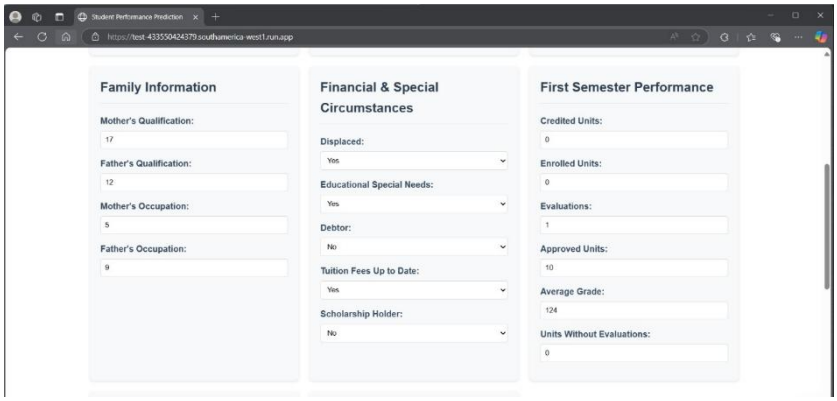


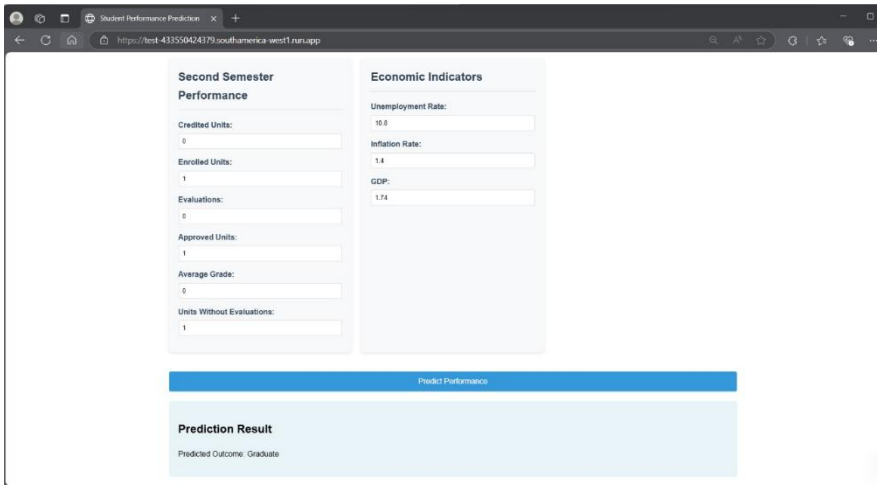




GRADUATE SUCCESS







6. IMPACT OF PROJECT ON HUMAN, SOCIAL AND SUSTAINABLE DEVELOPMENT

This project has the potential to significantly impact human, social, ethical, and sustainable development.

1.Human Development: By accurately predicting student dropouts, the project enables timely interventions, offering personalized support that enhances students' well-being and academic success, ultimately improving their future prospects.

2.Social Development: Reducing dropout rates contributes to a more educated society, fostering social mobility and reducing inequality. An educated populace is more likely to engage in civic activities, contributing to a stronger community fabric.

3.Ethical Development: Addressing the root causes of dropouts, such as financial constraints or lack of support, promotes equity in education. Data privacy and algorithmic fairness must be prioritized to avoid perpetuating biases.

4.Sustainable Development: Reducing dropouts means better utilization of educational resources, contributing to SDG 4 (Quality Education). An educated workforce is essential for sustainable economic growth and innovation. By implementing the findings responsibly, the project can lead to a more equitable, educated, and sustainable society.

7. Conclusion and Future work

7.1 Conclusion

This study successfully implemented and evaluated multiple machine learning models to predict student dropout and academic success. The results indicate that ensemble-based models, such as Random Forest and XGBoost, outperform traditional classifiers like Logistic Regression in terms of accuracy and predictive capability. While Logistic Regression provided a baseline accuracy of 70.06%, XGBoost and Random Forest achieved higher performance, with Random Forest yielding the best results at 83.71% accuracy (ROC AUC: 94.63%). The application of dimensionality reduction and hyperparameter tuning further improved model efficiency, with Optimized XGBoost reaching 80.54% accuracy (ROC AUC: 93.00%). This comparative analysis underscores the effectiveness of machine learning in educational analytics, enabling institutions to identify at-risk students and implement targeted interventions to enhance student retention and academic success.

7.2 Future Work

Even after deployment, there are several ways to enhance student dropout prediction and academic success analysis:

- **Continuous Model Improvement:** Regularly updating the model with new student data to improve accuracy and adapt to changing educational patterns.
- **Real-Time Monitoring:** Enhancing the system to analyze live student performance and engagement data for proactive interventions.
- **Improved Explainability:** Developing interpretable ML models to help educators and administrators understand key dropout risk factors.
- **Scalability Enhancements:** Optimizing model performance to handle larger datasets across multiple institutions efficiently.
- **Integration with Educational Systems:** Connecting the model with Learning Management Systems (LMS) and student support tools to automate early intervention strategies.

These enhancements will ensure long-term effectiveness, allowing institutions to make data-driven decisions and improve student retention and academic success.

8. References

[1] Villar, A., & de Andrade, C. R. V. (2024). Supervised machine learning algorithms for predicting student dropout and academic success: a comparative study. Discover Artificial Intelligence, 4(1).

[2] M. Nagy and R. Molontay, "Predicting Dropout in Higher Education Based on Secondary School Performance," 2018 IEEE 22nd International Conference on Intelligent Engineering Systems (INES), Las Palmas de Gran Canaria, Spain, 2018, pp. 000389–000394, doi: 10.1109/INES.2018.8523888.

[3] Matti Vaarma, Hongxiu Li, Predicting student dropouts with machine learning: An empirical study in Finnish higher education, Technology in Society, Volume 76, 2024, 102474, ISSN 0160-791X.

[4] Mduma, N., Kalegele, K. and Machuve, D. (2019). A Survey of Machine Learning Approaches and Techniques for Student Dropout Prediction. Data Science Journal, 18(1), p. 14. https://doi.org/10.5334/dsj-2019-014.