# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

# Suppress warnings

warnings.filterwarnings("ignore", category=FutureWarning)

# Step 1: Load the dataset

file\_path = '/content/balanced\_student\_dataset\_manual (6) (1).xlsx' # Update this with the correct path

college\_student\_dataset = pd.read\_excel('/content/balanced\_student\_dataset\_manual (6) (1).xlsx')

# Step 2: Define features and target

features = ['Age at enrollment', 'Previous qualification (grade)', 'Admission grade', 'GPA',

'Attendance percentage', 'Course', 'International student', 'Tuition fees status',

'Scholarship holder', 'Debtor']

target = 'Dropout'

# Convert target variable to binary (Dropout = 1, Non-Dropout = 0)

y = college\_student\_dataset[target]

X = college\_student\_dataset[features]

# Step 3: Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

test\_size=0.2, random\_state=42, stratify=y)

# Step 4: Define the preprocessing pipeline with feature scaling and one-hot encoding

numeric\_features = ['Age at enrollment', 'Previous qualification (grade)', 'Admission grade',

'GPA', 'Attendance percentage']

categorical\_features = ['Course', 'International student', 'Tuition fees status',

'Scholarship holder', 'Debtor']

preprocessor = ColumnTransformer(

transformers=[('num', StandardScaler(), numeric\_features),

('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_features)])

# Preprocess the data

X\_train\_processed = preprocessor.fit\_transform(X\_train)

X\_test\_processed = preprocessor.transform(X\_test)

# Step 5: Use class weights in Logistic Regression and Random Forest to handle any class imbalance

logistic\_model = LogisticRegression(class\_weight='balanced', random\_state=42)

rf\_model = RandomForestClassifier(class\_weight='balanced', random\_state=42)

# Step 6: Create pipelines

logistic\_pipeline = Pipeline(steps=[('classifier', logistic\_model)])

rf\_pipeline = Pipeline(steps=[('classifier', rf\_model)])

# Step 7: Fit the models on the training data

logistic\_pipeline.fit(X\_train\_processed, y\_train)

rf\_pipeline.fit(X\_train\_processed, y\_train)

# Step 8: Predictions

# Adjusted threshold for Logistic Regression

threshold = 0.7

y\_pred\_logistic\_adjusted = (logistic\_pipeline.predict\_proba(X\_test\_processed)[:, 1] > threshold).astype(int)

# Random Forest Predictions

y\_pred\_rf = rf\_pipeline.predict(X\_test\_processed)

# Step 9: Print the classification report for Logistic Regression and Random Forest

print("Logistic Regression Classification Report:")

print(classification\_report(y\_test, y\_pred\_logistic\_adjusted, zero\_division=0))

print("\nRandom Forest Classification Report:")

print(classification\_report(y\_test, y\_pred\_rf, zero\_division=0))

print("=======================================")

# Step 10: Create a DataFrame for predictions

predictions\_df = pd.DataFrame(X\_test.copy())

predictions\_df['Logistic Probabilities'] = logistic\_pipeline.predict\_proba(X\_test\_processed)[:, 1]

predictions\_df['RF Probabilities'] = rf\_pipeline.predict\_proba(X\_test\_processed)[:, 1]

# Calculate the mean probability

predictions\_df['Mean Probability'] = predictions\_df[['Logistic Probabilities', 'RF Probabilities']].mean(axis=1)

# Step 11: Select 5 high-risk students and 5 low-risk students for combination examples

top\_high\_risk\_students = predictions\_df[predictions\_df['Mean Probability'] > threshold].nlargest(5, 'Mean Probability')

low\_risk\_students = predictions\_df[predictions\_df['Mean Probability'] < 0.3].nsmallest(5, 'Mean Probability')

# Combine both at-risk and low-risk students into a single DataFrame

combined\_examples = pd.concat([top\_high\_risk\_students, low\_risk\_students], axis=0)

# Step 12: Display the combined 10 examples (5 at risk, 5 not at risk)

print(combined\_examples[['Logistic Probabilities', 'RF Probabilities', 'Mean Probability']])

# Step 13: Visualize the top 10 students (5 at risk and 5 not at risk) based on Mean Probability

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

sns.barplot(x='Mean Probability', y=combined\_examples.index, data=combined\_examples, palette='Blues\_d')

plt.title('Top 10 Students (High Risk and Low Risk) Based on Mean Probability')

plt.xlabel('Mean Probability of Dropout')

plt.ylabel('Student Index')

plt.tight\_layout() # Adjust layout to ensure nothing is cut off

# Show the plot

plt.show()

# Optional: Save the combined examples to a CSV for reference

combined\_examples\_path = '/path/to/save/combined\_student\_examples.csv' # Update path as needed

combined\_examples.to\_csv(combined\_examples\_path, index=False)