**Project Research Question**

**How effectively can machine learning models predict student dropout based on demographic attributes, academic performance, and study habits, and what are the key factors influencing dropout rates?**

**Project Name (Title)**

**Predicting Student Dropout Using Machine Learning: An Analysis of Academic and Demographic Factors**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import LabelEncoder

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix

# Data Generation

np.random.seed(42)

n = 20000

student\_id = np.arange(1, n+1)

gender = np.random.choice(['Male', 'Female'], size=n)

age = np.random.randint(18, 25, size=n)

attendance = np.random.randint(50, 100, size=n)

study\_hours = np.random.randint(1, 8, size=n)

course = np.random.choice(['BSc', 'BBA', 'BCom', 'BTech','DTSC','AIR','LAW','MBBS','CA'], size=n)

gpa = np.round(np.random.uniform(4.0, 10.0, size=n), 2)

dropout = np.random.choice(['yes', 'no'], size=n, p=[0.15, 0.85])

# DataFrame

df = pd.DataFrame({

    'student\_id': student\_id,

    'gender': gender,

    'age': age,

    'attendance': attendance,

    'study\_hours': study\_hours,

    'course': course,

    'gpa': gpa,

    'dropout': dropout

})

# First Data Overview

print("First data overview:")

print(df.head())

print("\nMissing Values:")

print(df.isnull().sum())

# Boxplots for GPA and Attendance

sns.boxplot(x=df['gpa'], color='lightblue')

plt.title('Boxplot of GPA')

plt.show()

sns.boxplot(x=df['attendance'], color='lightgreen')

plt.title('Boxplot of Attendance')

plt.show()

# Gender Distribution

print("\nGender distribution:")

print(df['gender'].value\_counts())

sns.countplot(x='gender', data=df, palette='pastel')

plt.title('Gender distribution')

plt.show()

# Course Distribution

print("\nCourse distribution:")

print(df['course'].value\_counts())

sns.countplot(x='course', data=df, palette='pastel')

plt.title('Course distribution')

plt.show()

# Summary Statistics

print("\nAge Summary:")

print(df['age'].describe())

print("\nGPA Summary:")

print(df['gpa'].describe())

print("\nAttendance Summary:")

print(df['attendance'].describe())

# Histograms

sns.histplot(df['gpa'], kde=True, color='orange')

plt.title('GPA Distribution')

plt.show()

sns.histplot(df['attendance'], kde=True, color='red')

plt.title('Attendance Distribution')

plt.show()

# Bivariate Analysis: Gender vs Dropout

sns.countplot(x='gender', hue='dropout', data=df)

plt.title('Gender vs Dropout')

plt.show()

# Study Hours vs GPA

sns.scatterplot(x='study\_hours', y='gpa', data=df)

plt.title('Study Hours vs GPA')

plt.show()

# Dropout vs GPA

sns.boxplot(y='gpa', x='dropout', data=df, palette='pastel')

plt.title('GPA vs Dropout')

plt.show()

# Multivariate Analysis: Pairplot

sns.pairplot(df[['age', 'attendance', 'study\_hours', 'gpa', 'gender']], hue='gender', palette='pastel')

plt.suptitle('Pairplot by Gender', y=1.02)

plt.show()

# Correlation Heatmap

correlation = df[['age', 'attendance', 'study\_hours', 'gpa']].corr()

sns.heatmap(correlation, annot=True)

plt.title('Correlation Heatmap')

plt.show()

# Encode Categorical Variables

le\_gender = LabelEncoder()

df['gender\_encoded'] = le\_gender.fit\_transform(df['gender'])

le\_course = LabelEncoder()

df['course\_encoded'] = le\_course.fit\_transform(df['course'])

le\_dropout = LabelEncoder()

df['dropout\_encoded'] = le\_dropout.fit\_transform(df['dropout'])

# Features and Target

X = df[['age', 'attendance', 'study\_hours', 'gpa', 'gender\_encoded', 'course\_encoded']]

y = df['dropout\_encoded']

# Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Logistic Regression

lr\_model = LogisticRegression(max\_iter=200)

lr\_model.fit(X\_train, y\_train)

lr\_pred = lr\_model.predict(X\_test)

print("\n🔷 Logistic Regression Results:")

print("Accuracy:", accuracy\_score(y\_test, lr\_pred))

print("Classification Report:\n", classification\_report(y\_test, lr\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, lr\_pred))

# Decision Tree

dt\_model = DecisionTreeClassifier(random\_state=42)

dt\_model.fit(X\_train, y\_train)

dt\_pred = dt\_model.predict(X\_test)

print("\n🔷 Decision Tree Results:")

print("Accuracy:", accuracy\_score(y\_test, dt\_pred))

print("Classification Report:\n", classification\_report(y\_test, dt\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, dt\_pred))

# Random Forest

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

rf\_pred = rf\_model.predict(X\_test)

print("\n🔷 Random Forest Results:")

print("Accuracy:", accuracy\_score(y\_test, rf\_pred))

print("Classification Report:\n", classification\_report(y\_test, rf\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, rf\_pred))

# Model Comparison

models = ['Logistic Regression', 'Decision Tree', 'Random Forest']

accuracies = [accuracy\_score(y\_test, lr\_pred),

              accuracy\_score(y\_test, dt\_pred),

              accuracy\_score(y\_test, rf\_pred)]

comparison\_df = pd.DataFrame({'Model': models, 'Accuracy': accuracies})

print(comparison\_df)

# Feature Importances (Random Forest)

importances = rf\_model.feature\_importances\_

feature\_names = X.columns

sns.barplot(x=importances, y=feature\_names, palette='pastel')

plt.title('Feature Importances (Random Forest)')

plt.show()