```python
import pandas as pd
import re
import warnings
from pycaret.classification import *
import matplotlib.pyplot as plt
import pickle
from sklearn.preprocessing import StandardScaler

# ============================
# 1. Read data and clean column names
# ============================
def clean_column(col):
    return re.sub(r'[^\x00-\x7F]+','_', col)

dataset = pd.read_csv("Dataset-Domestic.csv", encoding='utf-8')

# Remove all non-ASCII characters to avoid encoding errors
dataset.columns = [clean_column(col) for col in dataset.columns]

# ============================
# 2. Ignore all warnings (using warnings library)
# ============================
warnings.filterwarnings("ignore", category=UserWarning)

# ============================
# 3. Initialize PyCaret classification environment
# ============================
classification_setup = setup(
    data=dataset,
    target='Target',
    train_size=0.85,
    n_jobs=1,              # Single-threaded to avoid joblib encoding
conflicts
    verbose=True           # Display processing logs
)

# ============================
# 4. Model comparison and hyperparameter tuning
# ============================
best = compare_models()
# Tune hyperparameters, optimizing for AUC
tuned_best = tune_model(
    best,
    optimize='AUC',  # Optimize AUC metric
    search_library="scikit-optimize",
    search_algorithm="bayesian",
    n_iter=100
)
```

```
# ============================
# 5. Prediction + Save evaluation plots
# ============================
predict_model(tuned_best)

# ============================
# 6. Adjust AUC plot text and numbers for clarity
# ============================
# Modify matplotlib parameters to increase font size and adjust image quality
plt.rcParams.update({'font.size': 16})  # Increase font size
plt.rcParams['figure.dpi'] = 300  # Increase image resolution

# Save images as PNG files
plot_model(tuned_best, plot='auc', save=True)
plot_model(tuned_best, plot='confusion_matrix', save=True)
plot_model(tuned_best, plot='feature', save=True)

# ============================
# 7. Start interactive model analysis (only available in notebook
environments)
# ============================
try:
    evaluate_model(tuned_best)
except:
    print("evaluate_model() needs to run in a notebook environment, skipped.")

# ============================
# 8. Save the trained best model and scaler
# ============================
# Save the tuned_best model using pickle
with open('best_model_domestic.pkl', 'wb') as f:
    pickle.dump(tuned_best, f)

# Save the scaler
scaler = StandardScaler()
X_train = dataset.drop(columns=['Target'])  # Drop target column
scaler.fit(X_train)

with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)



###AutoML_International
import pandas as pd
import re
import warnings
```

```
from pycaret.classification import *
import matplotlib.pyplot as plt
import pickle
from sklearn.preprocessing import StandardScaler

# ============================
# 1. Read data and clean column names
# ============================
def clean_column(col):
    return re.sub(r'[^\x00-\x7F]+','_', col)

dataset_Real = pd.read_csv("Dataset-Int-Real.csv")
dataset_Synthetic = pd.read_csv("Dataset-Int-Synthetic.csv")

# Remove all non-ASCII characters to avoid encoding errors
dataset_Real.columns = [clean_column(col) for col in dataset_Real.columns]
dataset_Synthetic.columns = [clean_column(col) for col in
dataset_Synthetic.columns]

# ============================
# 2. Ignore all warnings (using warnings library)
# ============================
warnings.filterwarnings("ignore", category=UserWarning)

# Setup for Classification
from pycaret.classification import *
classification_setup = setup(data=dataset_Synthetic, test_data=dataset_Real,
target='Target', index=False, train_size=0.85)

best = compare_models()

# Tune the best model
tuned_best = tune_model(best, search_library="scikit-optimize",
search_algorithm="bayesian", n_iter=100)

# Predictions and plot evaluation
predict_model(tuned_best)
plot_model(tuned_best, plot='auc')
plot_model(tuned_best, plot='confusion_matrix')
plot_model(tuned_best, plot='feature')

# ============================
# 7. Start interactive model analysis (only available in notebook
environments)
# ============================
try:
    evaluate_model(tuned_best)
except:
```

```
    print("evaluate_model() needs to run in a notebook environment, skipped.")

# ============================
# 8. Save the trained best model and scaler
# ============================
# Save the tuned_best model using pickle
with open('best_model_international.pkl', 'wb') as f:
    pickle.dump(tuned_best, f)

# Save the scaler
scaler = StandardScaler()
X_train = dataset.drop(columns=['Target'])  # Drop target column
scaler.fit(X_train)

with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from ctgan import CTGAN
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import r2_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the dataset
real_data = pd.read_csv('Dataset-Int-Real.csv')

# Create a MinMaxScaler instance to scale the data to the [0, 1] range
scaler = MinMaxScaler()

# Scale the real data
scaled_real_data = scaler.fit_transform(real_data)

# Convert the scaled data back to a DataFrame to maintain consistency
scaled_real_data = pd.DataFrame(scaled_real_data, columns=real_data.columns)

# Names of the discrete columns (adjust according to your dataset)
discrete_columns = []

# Create a CTGAN model instance
ctgan = CTGAN(
    epochs=10000,
    verbose=True
)

# Train the CTGAN model using the scaled real data
ctgan.fit(scaled_real_data, discrete_columns)

# Generate synthetic data
scaled_synthetic_data = ctgan.sample(487)

# Inverse transform the synthetic data to restore it to the original data
range
synthetic_data = scaler.inverse_transform(scaled_synthetic_data)

# Convert to DataFrame
synthetic_data = pd.DataFrame(synthetic_data, columns=real_data.columns)

# Save the synthetic data to a CSV file
synthetic_data.to_csv('Dataset-int-synthetic.csv', index=False)
```

```python
# Plot the distribution of real and synthetic data
fig, axs = plt.subplots(nrows=len(real_data.columns), figsize=(10, 40))
kde_data = {}  # Store the KDE data for each column

for i, column in enumerate(real_data.columns):
    ax = sns.kdeplot(real_data[column], ax=axs[i], label='Real Data')
    sns.kdeplot(synthetic_data[column], ax=axs[i], label='Synthetic Data')

    # Get the KDE line data for both real and synthetic data
    real_kde_line = ax.lines[0]
    synthetic_kde_line = ax.lines[1]
    x_real = real_kde_line.get_xdata()
    y_real = real_kde_line.get_ydata()
    x_synthetic = synthetic_kde_line.get_xdata()
    y_synthetic = synthetic_kde_line.get_ydata()

    # Store the x, y data in the dictionary
    kde_data[column] = {
        'x_real': x_real,
        'y_real': y_real,
        'x_synthetic': x_synthetic,
        'y_synthetic': y_synthetic
    }

    axs[i].set_xlabel(column)
plt.show()

# Save the x, y data to CSV files
for column_name in kde_data.keys():
    real_xy = pd.DataFrame(
        {
            "x_real": kde_data[column_name]["x_real"],
            "y_real": kde_data[column_name]["y_real"],
        }
    )
    synthetic_xy = pd.DataFrame(
        {
            "x_synthetic": kde_data[column_name]["x_synthetic"],
            "y_synthetic": kde_data[column_name]["y_synthetic"],
        }
    )
    real_xy.to_csv(f"{column_name}_real_kde_data.csv", index=False)
    synthetic_xy.to_csv(f"{column_name}_synthetic_kde_data.csv", index=False)

# Create a new DataFrame for the real data, adding a 'Data_Type' column
real_data_labeled = real_data.copy()
real_data_labeled['Data_Type'] = 'Real Data'
```

```
# Create a new DataFrame for the synthetic data, adding a 'Data_Type' column
synthetic_data_labeled = synthetic_data.copy()
synthetic_data_labeled['Data_Type'] = 'Synthetic Data'

# Combine the real and synthetic data
combined_data = pd.concat([real_data_labeled, synthetic_data_labeled],
ignore_index=True)

# Save the combined data to a .csv file
combined_data.to_csv('Real_and_Synthetic_Data.csv', index=False)

# Plot the correlation matrix for real and synthetic data
fig, axs = plt.subplots(ncols=2, figsize=(20, 10))
sns.heatmap(real_data.corr(), cmap='coolwarm', annot=True, ax=axs[0])
axs[0].set_title('Real Data Correlation Matrix')
sns.heatmap(synthetic_data.corr(), cmap='coolwarm', annot=True, ax=axs[1])
axs[1].set_title('Synthetic Data Correlation Matrix')
plt.show()
```

```python
import streamlit as st
import pickle
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load models
model1 = pickle.load(open('best_model_domestic.pkl', 'rb'))
model2 = pickle.load(open('best_model_international.pkl', 'rb'))

# Load Scaler object (only once)
with open('scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)

# Page configuration
st.set_page_config(
    page_title='Evaluating and forecasting undergraduate dropouts',
    layout='wide',
    initial_sidebar_state='expanded'
)

# Page title and extra information
st.markdown("""
    <h1 style='color: darkblue;text-align: center; font-size: 36px;'>Warning
forecasts of domestic and international student dropouts</h1>
    <div style='text-align: center; font-size: 18px;'>Developed by Dr. Songbo
Wang et al., Hubei University of Technology.</div>
    <div style='text-align: center; font-size: 18px;'>Email:
Wangsongbo@hbut.edu.cn</div>
""", unsafe_allow_html=True)

st.sidebar.subheader('Input features for single prediction')
st.sidebar.markdown('<h2 style="font-size: 24px;">Select student type</h2>',
unsafe_allow_html=True)
# Radio button for selecting student type
student_type = st.sidebar.radio("", ('Domestic', 'International'))

# Use columns to divide into two parts
col1, col2 = st.sidebar.columns(2)

with col1:
    # Nationality slider based on the student type
    if student_type == 'Domestic':
        Nationality = 1
    elif student_type == 'International':
        Nationality = st.slider("Nationality", 2, 21, 10)
    Marital = st.slider('Marital status', 1, 6, 1)
    Mode = st.slider('Application order', 1, 18, 9)
    Order = st.slider("Application order", 1, 6, 2)
```

```
    Course = st.slider('Course type', 1, 17, 10)
    Attendance = st.slider('Daytime/evening attendance', 0, 1, 0)
    Qualification = st.slider("Previous qualification", 1, 17, 7)
    Mother_Q = st.slider("Mother qualification", 1, 29, 14)
    Father_Q = st.slider('Father qualification', 1, 34, 15)
    Mother_O = st.slider('Mother occupation', 1, 32, 18)
    Father_O = st.slider("Father occupation", 1, 46, 20)
    Displaced = st.slider('Displaced', 0, 1, 0)

with col2:
    Need = st.slider('Educational special need', 0, 1, 0)
    Debtor = st.slider("Debtor", 0, 1, 0)
    Fee = st.slider("Tuition fee", 0, 1, 0)
    Gender = st.slider("Gender", 0, 1, 0)
    Scholarship = st.slider('Scholarship', 0, 1, 0)
    Age = st.slider("Age", 17,70, 23)
    First = st.slider("1st semester approved course", 0, 26, 9)
    Second = st.slider("2nd semester approved course", 0, 20, 4)
    Unemployment = st.slider("Unemployment rate", 7.6, 16.2, 11.0)
    Inflation = st.slider("Inflation rate", -0.8, 3.7, 1.0)
    GDP = st.slider("GDP", -4.06, 3.51, 1.00)

if st.sidebar.button('Predict'):
    # Input features for prediction
    input_features = [[Marital, Mode, Order, Course, Attendance,
Qualification, Nationality,
                       Mother_Q, Father_Q, Mother_O, Father_O, Displaced,
Need, Debtor,
                       Fee, Gender, Scholarship, Age, First, Second,
Unemployment, Inflation, GDP]]

    # Create DataFrame
    new_data = pd.DataFrame(input_features, columns=['Marital', 'Mode',
'Order', 'Course', 'Attendance', 'Qualification',
                                                     'Nationality',
'Mother_Q', 'Father_Q', 'Mother_O', 'Father_O',
                                                     'Displaced', 'Need',
'Debtor', 'Fee', 'Gender', 'Scholarship',
                                                     'Age', 'First', 'Second',
'Unemployment', 'Inflation', 'GDP'])

    # Standardize the input features
    feature_values_scaled = scaler.transform(new_data.values)

try:
    col1, col2, col3 = st.columns([5, 1, 5])  # Left column for Domestic,
right column for International
```

```
    # Domestic Student Prediction (Left Column)
    with col1:
        prediction = model1.predict(feature_values_scaled)
        st.markdown("<h1 style='color: darkblue;font-size: 30px;'>Domestic
students result:</h1>", unsafe_allow_html=True)

        if prediction[0] == 1:
            st.markdown(f"<h2 style='font-size: 28px;'> Dropout! </h2>",
unsafe_allow_html=True)
        elif prediction[0] == 0:
            st.markdown(f"<h2 style='font-size: 28px; '>Graduate! </h2>",
unsafe_allow_html=True)

        # Display features and values for Domestic students
        rows = []
        for i in range(0, len(new_data.columns), 2):
            feature_name_1 = new_data.columns[i]
            value_1 = new_data.iloc[0, i]
            feature_name_2 = new_data.columns[i+1] if i+1 <
len(new_data.columns) else ''
            value_2 = new_data.iloc[0, i+1] if i+1 < len(new_data.columns)
else ''
            row = [f'{feature_name_1}: {value_1}', f'{feature_name_2}:
{value_2}']
            rows.append(row)

        rows_df = pd.DataFrame(rows, columns=['Line 1', 'Line 2'])
        st.dataframe(rows_df, use_container_width=True)

    with col3:
        # Check if Nationality is within the valid range for international
students
        if 2 <= Nationality <= 21:
            # Use model2 for International students
            prediction = model2.predict(feature_values_scaled)
            st.markdown("<h1 style='color: darkblue; font-size:
30px;'>International students result:</h1>", unsafe_allow_html=True)

            if prediction[0] == 1:
                st.markdown(f"<h2 style='font-size: 28px; '>Dropout! </h2>",
unsafe_allow_html=True)
            elif prediction[0] == 0:
                st.markdown(f"<h2 style='font-size: 28px;'>Graduate! </h2>",
unsafe_allow_html=True)

            # Display features and values for International students
            rows = []
```

```
            for i in range(0, len(new_data.columns), 2):
                feature_name_1 = new_data.columns[i]
                value_1 = new_data.iloc[0, i]
                feature_name_2 = new_data.columns[i+1] if i+1 <
len(new_data.columns) else ''
                value_2 = new_data.iloc[0, i+1] if i+1 < len(new_data.columns)
else ''
                row = [f'{feature_name_1}: {value_1}', f'{feature_name_2}:
{value_2}']
                rows.append(row)

            rows_df = pd.DataFrame(rows, columns=['Line 1', 'Line 2'])
            st.dataframe(rows_df, use_container_width=True)

        else:
            # Adjust the Nationality to 10 if it doesn't meet the criteria for
international
            Nationality = 10
            # Create the new data again with the modified Nationality
            input_features = [[Marital, Mode, Order, Course, Attendance,
Qualification, Nationality,
                               Mother_Q, Father_Q, Mother_O, Father_O,
Displaced, Need, Debtor,
                               Fee, Gender, Scholarship, Age, First, Second,
Unemployment, Inflation, GDP]]

            # Create DataFrame
            new_data = pd.DataFrame(input_features, columns=['Marital',
'Mode', 'Order', 'Course', 'Attendance', 'Qualification',
                                                             'Nationality',
'Mother_Q', 'Father_Q', 'Mother_O', 'Father_O',
                                                             'Displaced',
'Need', 'Debtor', 'Fee', 'Gender', 'Scholarship',
                                                             'Age', 'First',
'Second', 'Unemployment', 'Inflation', 'GDP'])

            # Standardize the input features
            feature_values_scaled = scaler.transform(new_data.values)

            # Perform prediction for the modified Nationality (if needed)
            prediction = model2.predict(feature_values_scaled)
            st.markdown("<h1 style=' color: darkblue;font-size:
30px;'>International students result:</h1>", unsafe_allow_html=True)

            if prediction[0] == 1:
                st.markdown(f"<h2 style='font-size: 28px;'>Dropout! </h2>",
unsafe_allow_html=True)
            elif prediction[0] == 0:
```

```
                st.markdown(f"<h2 style='font-size: 28px;'>Graduate! </h2>",
unsafe_allow_html=True)

            # Display features and values for Domestic students
            rows = []
            for i in range(0, len(new_data.columns), 2):
                feature_name_1 = new_data.columns[i]
                value_1 = new_data.iloc[0, i]
                feature_name_2 = new_data.columns[i+1] if i+1 <
len(new_data.columns) else ''
                value_2 = new_data.iloc[0, i+1] if i+1 < len(new_data.columns)
else ''
                row = [f'{feature_name_1}: {value_1}', f'{feature_name_2}:
{value_2}']
                rows.append(row)

            rows_df = pd.DataFrame(rows, columns=['Line 1', 'Line 2'])
            st.dataframe(rows_df, use_container_width=True)
except ValueError as e:
    st.error(f"N/A: {e}")
except Exception as e:
    st.error(f"error: {e}")
# Impact factors ranking display
st.markdown("<h2 style='color: darkblue;font-size: 24px;'>Impact factors
ranking:</h2>", unsafe_allow_html=True)
col1, col2, col3, col4, col5 = st.columns([4, 4, 2, 2, 2])

with col1:
    st.markdown("<p style='font-size: 20px;'>1. 2nd semester approved
course.</p>", unsafe_allow_html=True)
with col2:
    st.markdown("<p style='font-size: 20px;'>2. 1st semester approved
course.</p>", unsafe_allow_html=True)
with col3:
    st.markdown("<p style='font-size: 20px;'>3. Tuition fee.</p>",
unsafe_allow_html=True)
with col4:
    st.markdown("<p style='font-size: 20px;'>4. Course type.</p>",
unsafe_allow_html=True)
with col5:
    st.markdown("<p style='font-size: 20px;'>5. Debtor.</p>",
unsafe_allow_html=True)
```