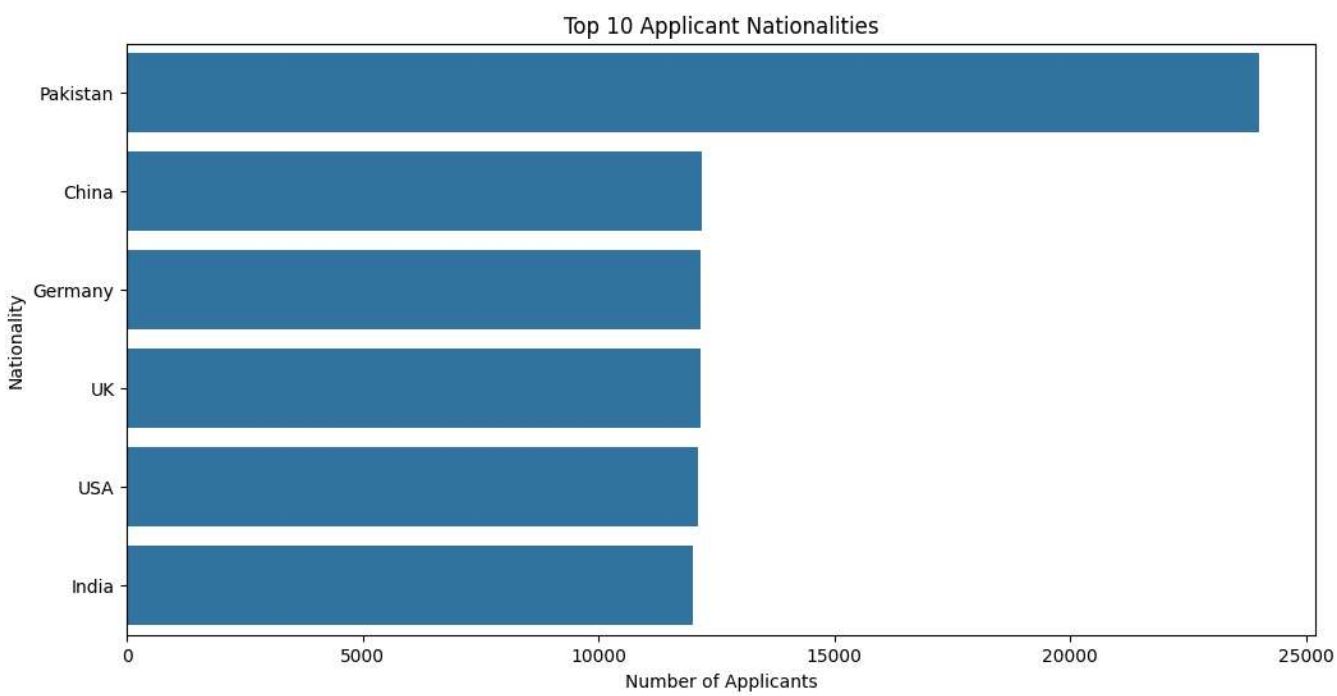
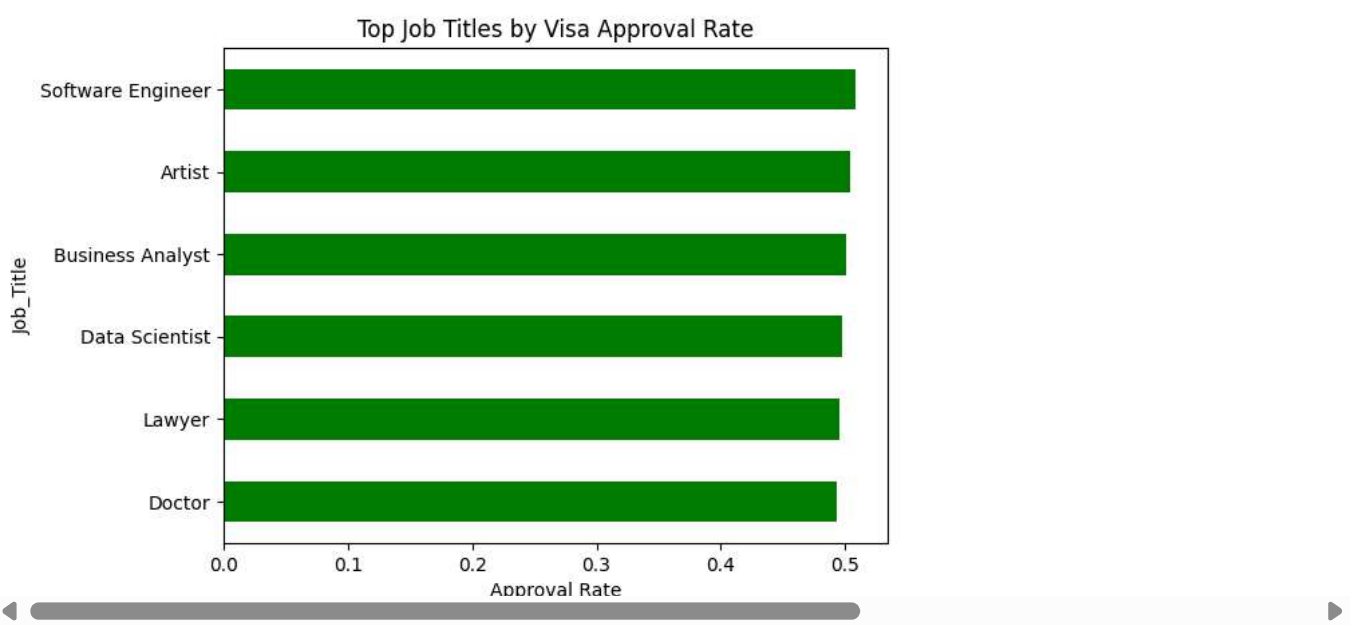
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_recall_curve
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
from collections import Counter

# Load data
df = pd.read_excel('/content/Visa_Prediction_Dataset_Updated.xlsx')

# Clean Visa_Status
df = df[df['Visa_Status'].isin(['Approved', 'Rejected'])]
df['Visa_Status'] = df['Visa_Status'].map({'Approved': 1, 'Rejected': 0})

# Fill missing values
num_cols = df.select_dtypes(include=['int64', 'float64']).columns
df[num_cols] = df[num_cols].fillna(df[num_cols].median())
cat_cols = df.select_dtypes(include='object').columns
for col in cat_cols:
    df[col] = df[col].fillna(df[col].mode()[0])
```

Data Visualizations

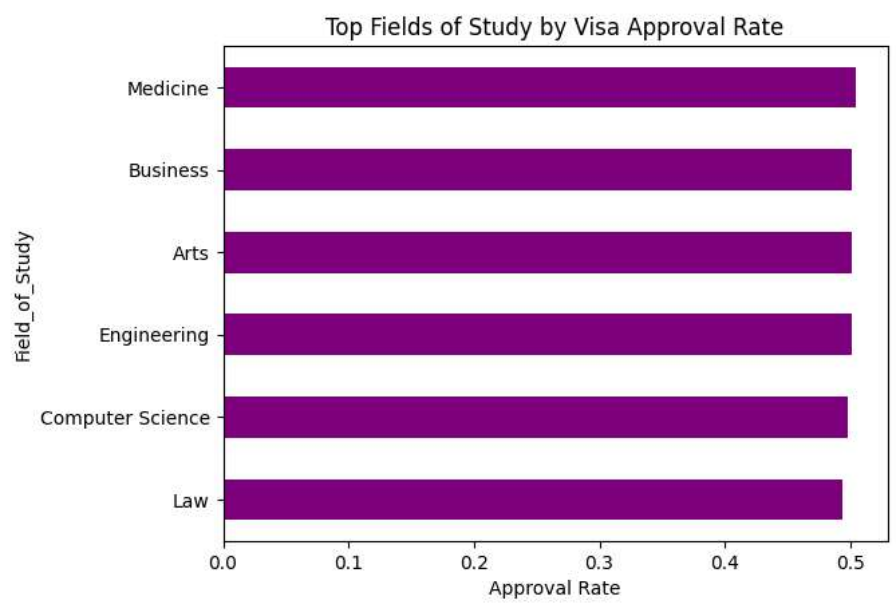
```
#Top Nationalities Applying for Visa
plt.figure(figsize=(12, 6))
sns.countplot(data=df, y='Nationality', order=df['Nationality'].value_counts().head(10).index)
plt.title('Top 10 Applicant Nationalities')
plt.xlabel('Number of Applicants')
plt.ylabel('Nationality')
plt.show()
```



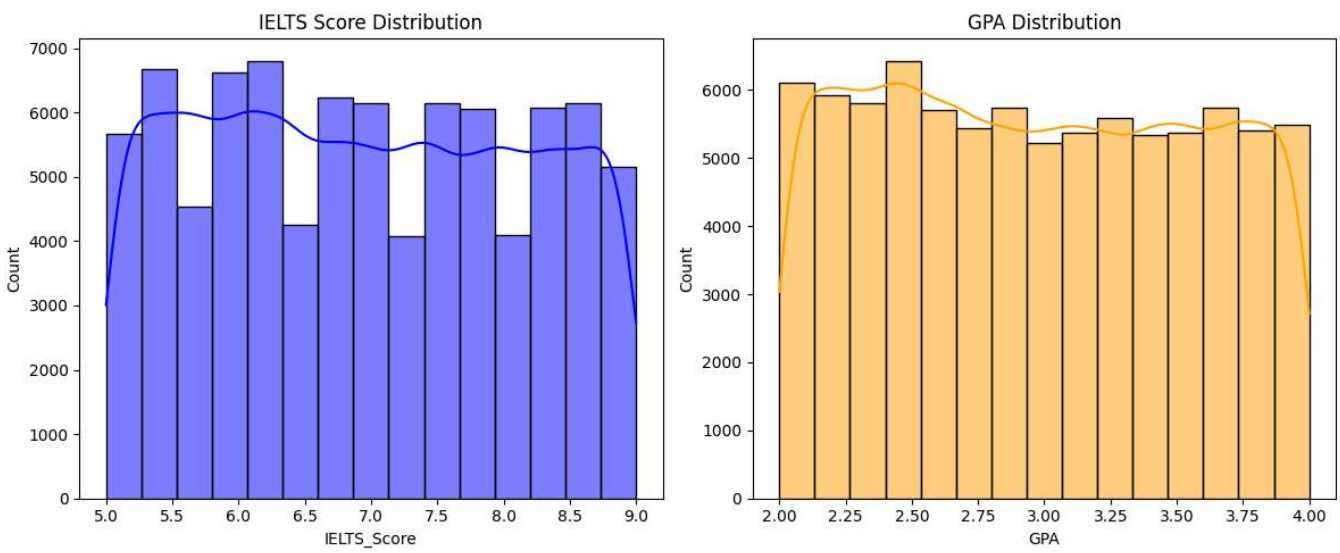

```
#Visa Approval Rate by Job Title (Professionals)
job_visa = df.groupby('Job_Title')['Visa_Status'].mean().sort_values(ascending=False).head(10)
job_visa.plot(kind='barh', color='green')
plt.title('Top Job Titles by Visa Approval Rate')
plt.xlabel('Approval Rate')
plt.gca().invert_yaxis()
plt.show()
```




```
#Visa Approval Rate by Field of Study (Students)
field_visa = df.groupby('Field_of_Study')['Visa_Status'].mean().sort_values(ascending=False).head(10)
field_visa.plot(kind='barh', color='purple')
plt.title('Top Fields of Study by Visa Approval Rate')
plt.xlabel('Approval Rate')
plt.gca().invert_yaxis()
plt.show()
```

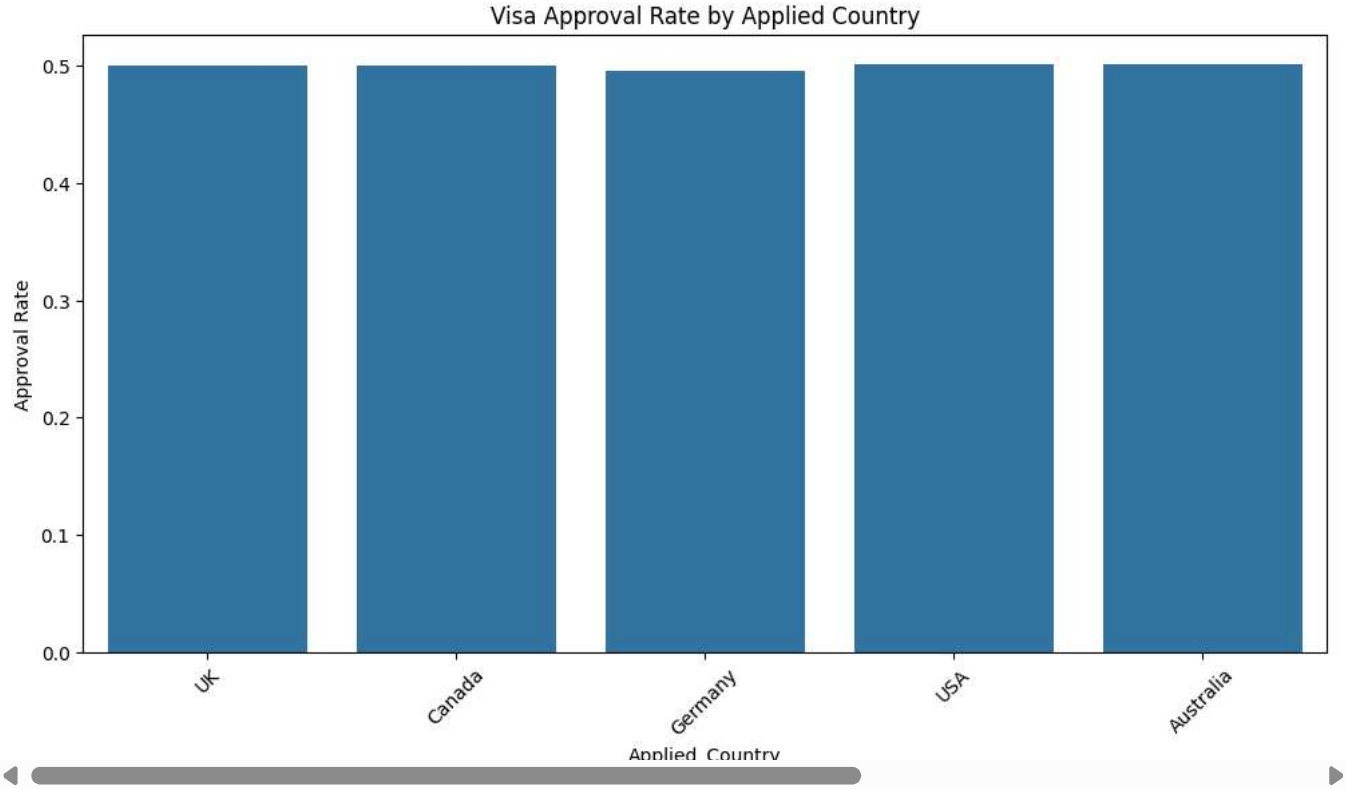



```
#Distribution of IELTS Score and GPA (Students)
fig, axs = plt.subplots(1, 2, figsize=(12, 5))
sns.histplot(df['IELTS_Score'], kde=True, bins=15, ax=axs[0], color='blue')
axs[0].set_title('IELTS Score Distribution')

sns.histplot(df['GPA'], kde=True, bins=15, ax=axs[1], color='orange')
axs[1].set_title('GPA Distribution')

plt.tight_layout()
plt.show()
```

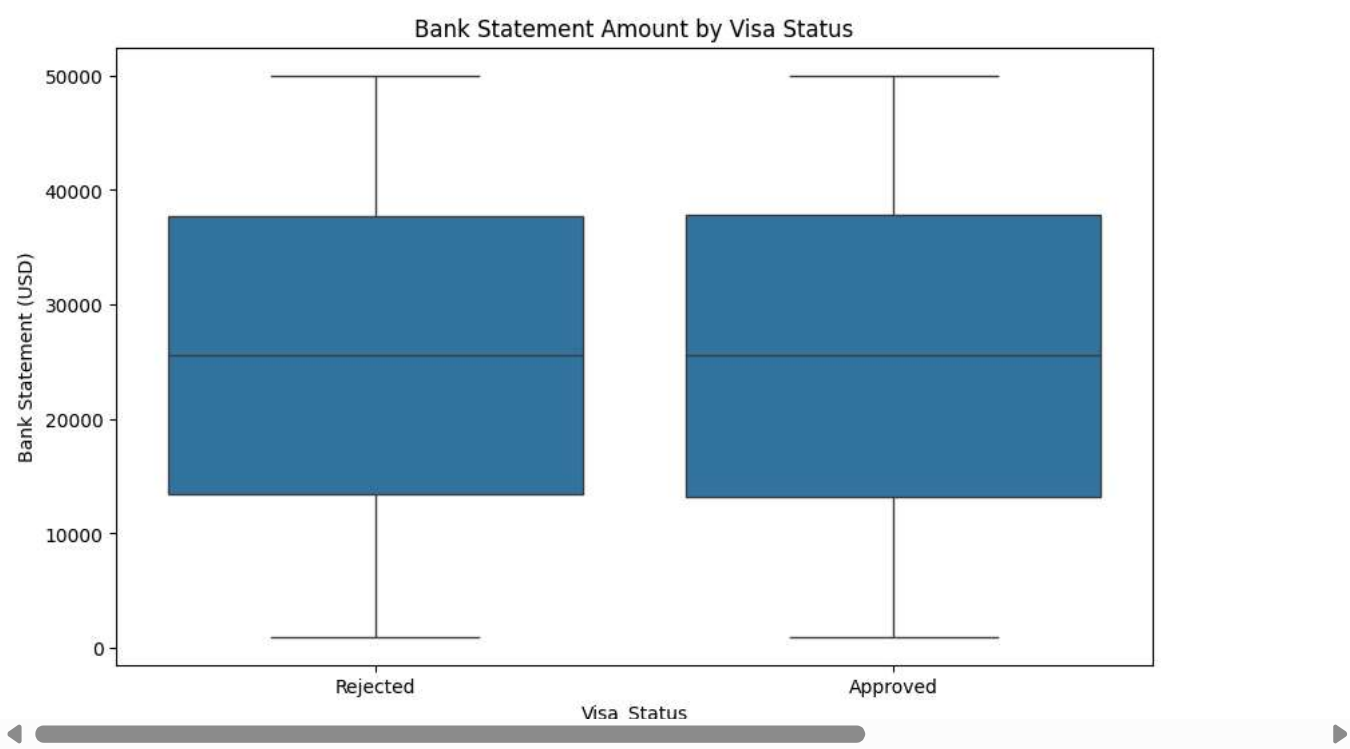



```
#Applied Country vs Visa Outcome
plt.figure(figsize=(12, 6))
sns.barplot(x='Applied_Country', y='Visa_Status', data=df, ci=None)
plt.title('Visa Approval Rate by Applied Country')
plt.ylabel('Approval Rate')
plt.xticks(rotation=45)
plt.show()
```

```
<ipython-input-31-e2cf16d00613>:3: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='Applied_Country', y='Visa_Status', data=df, ci=None)
```




```
#Bank Statement vs Visa Approval
plt.figure(figsize=(10, 6))
sns.boxplot(x='Visa_Status', y='Bank_Statement_Amount', data=df)
plt.xticks([0, 1], ['Rejected', 'Approved'])
plt.title('Bank Statement Amount by Visa Status')
plt.ylabel('Bank Statement (USD)')
plt.show()
```



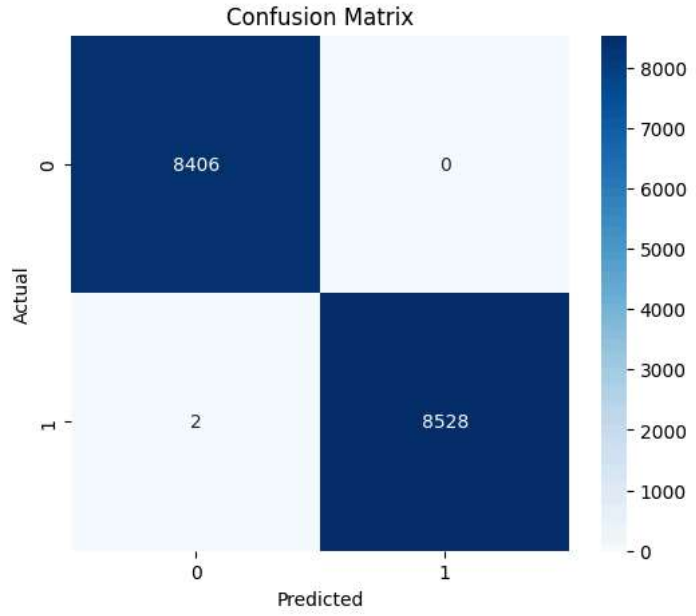

```
# Define categorical features
categorical_cols = [
    'Gender', 'Nationality', 'Current_Country', 'Highest_Education', 'Field_of_Study',
    'Job_Title', 'Industry', 'Visa_Type', 'Applied_Country',
    'Previous_Travel_History', 'Sponsorship_Type'
]

selected_features = [
    'Age', 'Gender', 'Nationality', 'Current_Country', 'GPA', 'Highest_Education',
    'Field_of_Study', 'IELTS_Score', 'Job_Title', 'Years_Work_Experience', 'Industry',
    'Expected_Salary_USD', 'Visa_Type', 'Applied_Country', 'Previous_Travel_History',
    'Annual_Income_USD', 'Bank_Statement_Amount', 'Sponsorship_Type'
]

# Split early to prevent data leakage
X = df[selected_features]
y = df['Visa_Status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Label encode using train data only
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    X_train.loc[:, col] = le.fit_transform(X_train[col].astype(str))
    X_test.loc[:, col] = le.transform(X_test[col].astype(str))
    label_encoders[col] = le



# Scale
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train_scaled, y_train)

print("✅ SMOTE applied. Class distribution:", Counter(y_train_bal))
```

```
✅ SMOTE applied. Class distribution: Counter({0: 33924, 1: 33924})
```

```
# Train
model = RandomForestClassifier(n_estimators=200, random_state=42)
model.fit(X_train_bal, y_train_bal)

# Evaluate
y_probs = model.predict_proba(X_test_scaled)[:, 1]
y_pred = (y_probs > 0.5).astype(int)

print("=== Classification Report ===")
print(classification_report(y_test, y_pred))
```

```
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")

# Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      8406
           1       1.00      1.00      1.00      8530

    accuracy                           1.00     16936
   macro avg       1.00      1.00      1.00     16936
weighted avg       1.00      1.00      1.00     16936

Accuracy: 0.9999
```




```
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)
f1_scores = 2 * (precision * recall) / (precision + recall + 1e-10)
best_idx = np.argmax(f1_scores)
best_threshold = thresholds[best_idx]
print(f"🔍 Best Threshold for F1-score: {best_threshold:.2f}")
```

```
🔍 Best Threshold for F1-score: 0.67
```

```
cv_scores = cross_val_score(model, X_train_bal, y_train_bal, cv=5, scoring='f1')
print(f"Cross-Validated F1 Scores: {cv_scores}")
print(f"Mean F1 Score: {cv_scores.mean():.4f}")
```

```
Cross-Validated F1 Scores: [0.9999263  0.99985259 0.9999263  1.         0.99985259]
Mean F1 Score: 0.9999
```

```
def encode_or_default(value, col_name):
    le = label_encoders[col_name]
    if value not in le.classes_:
        value = le.classes_[0]
    return le.transform([value])[0]

   # return le.transform([value])[0] if value in le.classes_ else le.transform([le.classes_[0]])[0]

def predict_user_input():
    user_type = input("Student or Professional? ").strip().lower()
    age = int(input("Age: "))
    gender = encode_or_default(input("Gender: "), 'Gender')
    nationality = encode_or_default(input("Nationality: "), 'Nationality')
    current_country = encode_or_default(input("Current Country: "), 'Current_Country')
    visa_type = encode_or_default(input("Visa Type: "), 'Visa_Type')
    applied_country = encode_or_default(input("Applied Country: "), 'Applied_Country')
    travel_history = encode_or_default(input("Previous Travel History: "), 'Previous_Travel_History')
```

```
    sponsorship = encode_or_default(input("Sponsorship Type: "), 'Sponsorship_Type')
    bank_statement = int(input("Bank Statement (USD): "))

    # Type-based inputs
    gpa = ielts = education = field = job = industry = salary = experience = income = 0

    if user_type == 'student':
        gpa = float(input("GPA: "))
        education = encode_or_default(input("Highest Education: "), 'Highest_Education')
        field = encode_or_default(input("Field of Study: "), 'Field_of_Study')
        ielts = float(input("IELTS Score: "))
    else:
        job = encode_or_default(input("Job Title: "), 'Job_Title')
        industry = encode_or_default(input("Industry: "), 'Industry')
        salary = int(input("Expected Salary (USD): "))
        experience = int(input("Years of Work Experience: "))
        income = int(input("Annual Income (USD): "))

    # Final input vector
    sample = np.array([[age, gender, nationality, current_country, gpa, education, field,
                        ielts, job, experience, industry, salary, visa_type, applied_country,
                        travel_history, income, bank_statement, sponsorship]])

    sample_scaled = scaler.transform(sample)
    probability = model.predict_proba(sample_scaled)[0][1]
    prediction = "Approved" if probability > best_threshold else "Rejected"

    print(f"\n🔮 Prediction: {prediction}")
    print(f"📈 Approval Probability: {probability * 100:.2f}%")

if __name__ == "__main__":
    predict_user_input()
```

```
Student or Professional? Student
Age: 22
Gender: male
Nationality: pak
Current Country: pak
Visa Type: student
Applied Country: uk
Previous Travel History: yes
Sponsorship Type: university
Bank Statement (USD): 55000
GPA: 3
Highest Education: bs
Field of Study: IT
IELTS Score: 7

🔮 Prediction: Approved
📈 Approval Probability: 81.00%
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Star
  warnings.warn(
```