```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
```

```
# ----------------------------
# 1. LOAD DATA
# ----------------------------
df = pd.read_csv("Visa_Status_Prediction_Dataset.csv")   # change filename accordingly
```

```
# ----------------------------
# 2. DROP UNUSED COLUMNS
# ----------------------------
df.drop(columns=["case_id"], inplace=True)
```

```
# ----------------------------
# 3. HANDLE MISSING VALUES
# ----------------------------
# Categorical columns
categorical_cols = ["continent", "education_of_employee", "region_of_employment", "unit_of_wage"]
for col in categorical_cols:
    df[col] = df[col].fillna("Unknown")
```

```
# Binary Y/N columns
binary_cols = ["has_job_experience", "requires_job_training", "full_time_position"]
for col in binary_cols:
    df[col] = df[col].fillna("N")
```

```
# Numeric columns
numeric_cols = ["no_of_employees", "yr_of_estab", "prevailing_wage"]
for col in numeric_cols:
    df[col] = df[col].fillna(df[col].median())
```

```
# ----------------------------
# 4. CONVERT Y/N TO 0/1
# ----------------------------
for col in binary_cols:
    df[col] = df[col].map({"Y": 1, "N": 0})
```

```
# ----------------------------
# 5. FEATURE ENGINEERING
# ----------------------------
# Company age from establishment year
df["company_age"] = 2025 - df["yr_of_estab"]
```

```
# Normalize wages (optional)
# Idea: Convert Hour → Year equivalent (assuming 2080 hours/year)
df.loc[df["unit_of_wage"] == "Hour", "prevailing_wage"] *= 2080
```

```
# Education Ordinal Encoding
education_order = {
    "High School": 1,
    "Bachelor's": 2,
    "Master's": 3,
    "Doctorate": 4,
    "Unknown": 0
}
df["education_level"] = df["education_of_employee"].map(education_order)
```

```
# ----------------------------
# 6. LABEL ENCODE OTHER CATEGORICAL VARIABLES
# ----------------------------
label_encode_cols = ["continent", "region_of_employment", "unit_of_wage"]

le = LabelEncoder()
for col in label_encode_cols:
    df[col] = le.fit_transform(df[col])
```

```
# ----------------------------
# 7. ENCODE TARGET VARIABLE
# ----------------------------
# Convert case_status into binary: Certified = 1, Denied = 0
df["case_status"] = df["case_status"].map({
    "Certified": 1,
    "Denied": 0
})
```

```
# Remove rows with other statuses (optional)
df = df[df["case_status"].isin([0, 1])]
```

```
# ----------------------------
# 8. SAVE CLEANED DATASET
# ----------------------------
df.to_csv("visa_data_cleaned.csv", index=False)

print("Preprocessing complete! Saved as visa_data_cleaned.csv")
df.head()
```

```
Preprocessing complete! Saved as visa_data_cleaned.csv
```

| | continent | education_of_employee | has_job_experience | requires_job_training | no_of_employees | yr_of_estab | region_of_employ |
|---|---|---|---|---|---|---|---|
| **0** | 1 | High School | 0 | 0 | 14513 | 2007 | |
| **1** | 1 | Master's | 1 | 0 | 2412 | 2002 | |
| **2** | 1 | Bachelor's | 0 | 1 | 44444 | 2008 | |
| **3** | 1 | Bachelor's | 0 | 0 | 98 | 1897 | |
| **4** | 0 | Master's | 1 | 0 | 1082 | 2005 | |

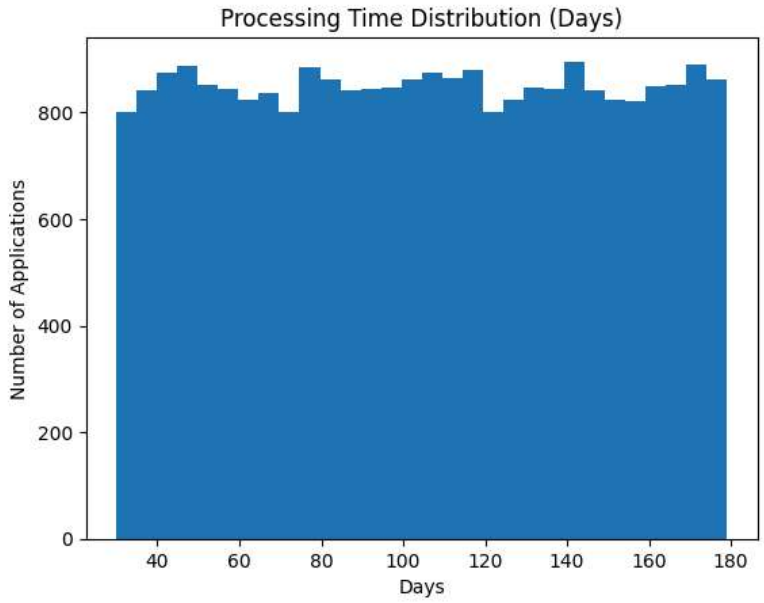
Next steps: Generate code with df | New interactive sheet

```
### Milestone2
```

```
import numpy as np

np.random.seed(42)

# Simulated processing time (days)
df["processing_time_days"] = np.random.randint(30, 180, size=len(df))

# Simulated application month (seasonality)
df["application_month"] = np.random.randint(1, 13, size=len(df))
```

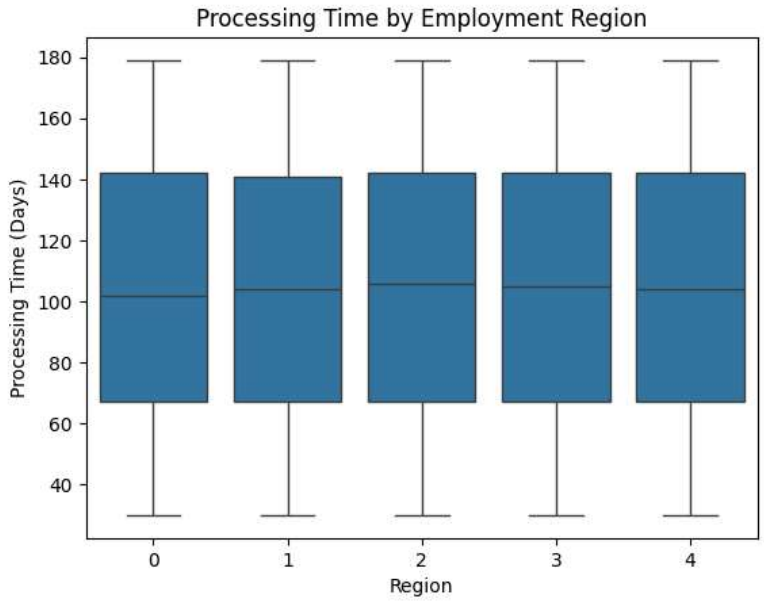
```
### Processing Time Distribution
```

```
plt.figure()
plt.hist(df["processing_time_days"], bins=30)
plt.title("Processing Time Distribution (Days)")
plt.xlabel("Days")
plt.ylabel("Number of Applications")
plt.show()
```



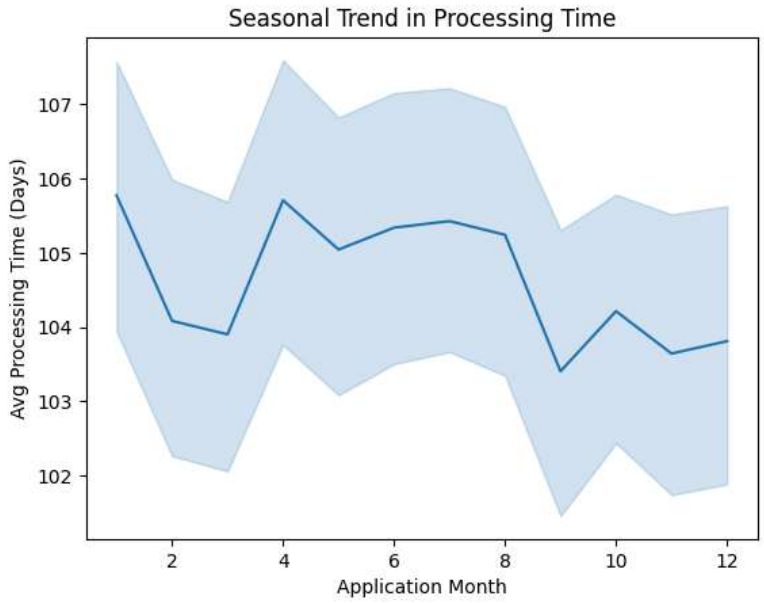

```
### Processing Time Across Regions
```

```
plt.figure()
sns.boxplot(x="region_of_employment", y="processing_time_days", data=df)
plt.title("Processing Time by Employment Region")
plt.xlabel("Region")
plt.ylabel("Processing Time (Days)")
plt.show()
```



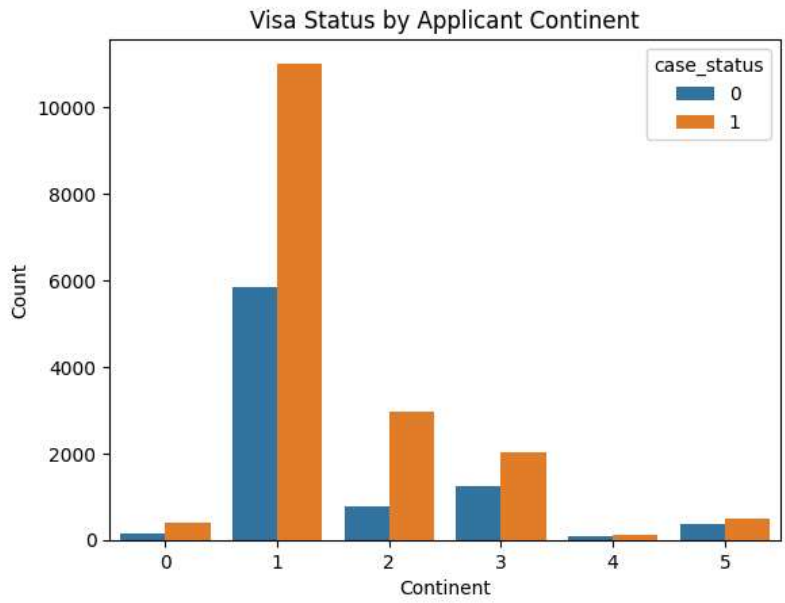

```
### Seasonal Trends (Month-wise)
```

```
plt.figure()
sns.lineplot(x="application_month", y="processing_time_days", data=df)
plt.title("Seasonal Trend in Processing Time")
plt.xlabel("Application Month")
plt.ylabel("Avg Processing Time (Days)")
plt.show()
```



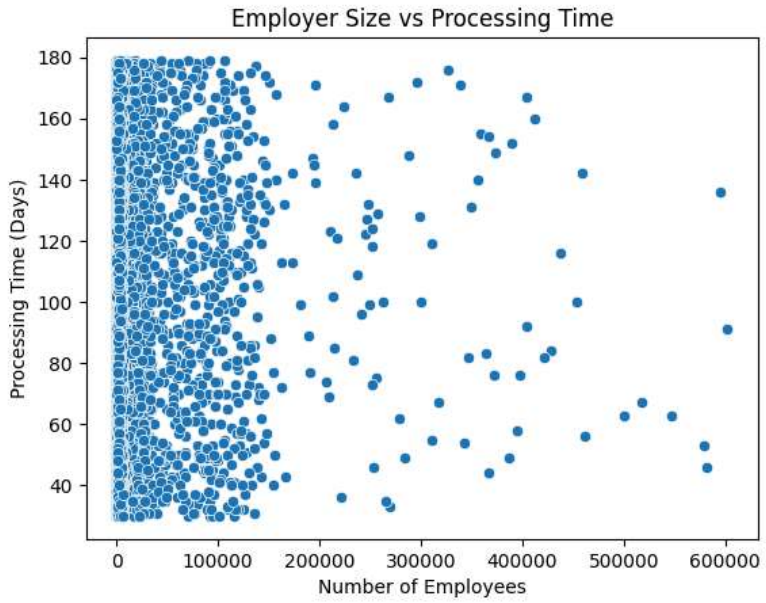

```
### Applicant Origin vs Visa Outcome
```

```
plt.figure()
sns.countplot(x="continent", hue="case_status", data=df)
plt.title("Visa Status by Applicant Continent")
plt.xlabel("Continent")
plt.ylabel("Count")
plt.show()
```




```
### Workload Indicators
```

```
plt.figure()
sns.scatterplot(x="no_of_employees", y="processing_time_days", data=df)
plt.title("Employer Size vs Processing Time")
plt.xlabel("Number of Employees")
plt.ylabel("Processing Time (Days)")
plt.show()
```

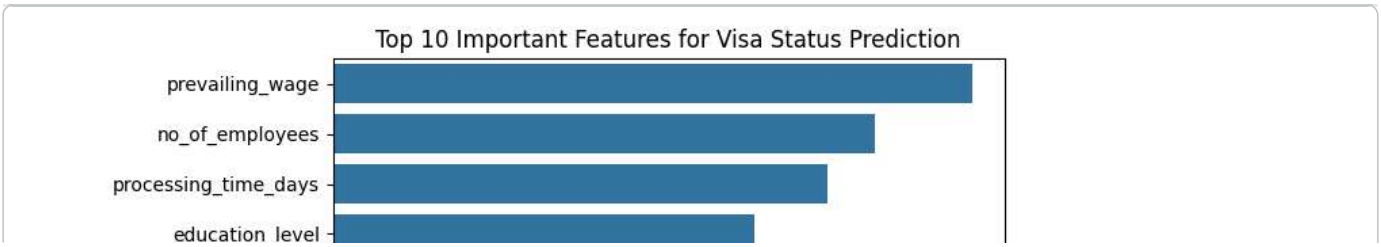



```
### Feature Importance Analysis
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Drop the original 'education_of_employee' column as it's now encoded in 'education_level'
X = df.drop(columns=["case_status", "education_of_employee", "unit_of_wage"])
y = df["case_status"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

importances = model.feature_importances_
features = X.columns

importance_df = pd.DataFrame({
    "Feature": features,
    "Importance": importances
}).sort_values(by="Importance", ascending=False)
```

```
### Plot Feature Importance
```

```
plt.figure()
sns.barplot(x="Importance", y="Feature", data=importance_df.head(10))
plt.title("Top 10 Important Features for Visa Status Prediction")
plt.show()
```


Top 10 Important Features for Visa Status Prediction
prevailing_wage
no_of_employees
processing_time_days
education_level