🔍 Step-by-Step Implementation Guide

This breakdown ensures a complete, professional workflow using **Python** for your predictive policing system. We cover:

- Dataset Collection
- Library Imports
- Data Preprocessing
- Model Building
- Model Evaluation
- Result Visualization
- Interactive Dashboard Development

✅ Step 1: Dataset Collection

You've identified three vital data categories:

- 📊 **Historical Data**: Leverage the UNODC Victimization Survey (2003–2008) to establish baseline patterns (e.g., burglary, theft, assault).
- 📰 **Recent Data**: Use current crime statistics such as the AllAfrica 2023 report and request up-to-date data from:
  - Rwanda National Police (RNP)
  - National Institute of Statistics of Rwanda (NISR)
- 📡 **Proxy Data (if needed)**: Include alternative sources such as:
  - Mobile phone usage
  - Social media trend data
  - Demographic and economic factors

🔧 **Action Plan**:

- Download/export UNODC reports (CSV or Excel).
- Request detailed structured data from RNP/NISR (including fields like `crime_type`, `timestamp`, `location`).
- Compile into a unified dataset:
  ✅ Example file: `rwanda_crime_data.csv`
  ✅ Columns: `crime_type`, `location`, `latitude`, `longitude`, `date`, `frequency`

🧰 Step 2: Importing Necessary Libraries

Use Python's ecosystem for data processing, modeling, visualization, and dashboarding.

📦 **Required Libraries**:

```
# Run once to install:
# pip install pandas numpy scikit-learn folium matplotlib seaborn streamlit

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 accuracy_score, classification_report
from sklearn.cluster import KMeans
import folium
import matplotlib.pyplot as plt
import seaborn as sns
import streamlit as st
```

💡 **Why these libraries?**

- `pandas`, `numpy`: for efficient data handling.
- `scikit-learn`: offers preprocessing tools, ML algorithms, and metrics.
- `folium`: generates interactive leaflet maps.
- `matplotlib`, `seaborn`: support insightful plotting.
- `streamlit`: powers your live web dashboard.

🧹 Step 3: Dataset Preprocessing

Ensure your data is clean, consistent, and model-ready.

🔄 **Steps**:

1. **Load Dataset**
2. **Handle Missing Data**
3. **Encode Categorical Values**
4. **Feature Engineering**
5. **Normalize/Scale Data**

🧪 **Code Example**:

```
# Load the data
df = pd.read_csv('rwanda_crime_data.csv')
print(df.head())
print(df.info())

# Handle missing values
df['crime_type'].fillna('Unknown', inplace=True)
df['location'].fillna('Unknown', inplace=True)
```

```
df['frequency'].fillna(df['frequency'].mean(), inplace=True)
df.dropna(subset=['latitude', 'longitude', 'date'], inplace=True)

# Encode categories
le_crime = LabelEncoder()
le_location = LabelEncoder()
df['crime_type_encoded'] = le_crime.fit_transform(df['crime_type'])
df['location_encoded'] = le_location.fit_transform(df['location'])

# Convert and extract time features
df['date'] = pd.to_datetime(df['date'])
df['month'] = df['date'].dt.month
df['hour'] = df['date'].dt.hour

# Normalize numeric values
scaler = StandardScaler()
df[['frequency', 'latitude', 'longitude']] =
scaler.fit_transform(df[['frequency', 'latitude', 'longitude']])
```

🧠 **Explanation**:

- Convert categories into numbers (`LabelEncoder`).
- Extracting `month`, `hour` allows seasonal/time-based predictions.
- `StandardScaler` ensures numerical consistency (especially for clustering algorithms like KMeans).

🤖 Step 4: Creating the Models

Two ML goals:

- **Classification**: Predict type of crime based on patterns.
- **Clustering**: Discover geographic hotspots using unsupervised learning.

📘 Model 1: Classification (Random Forest)
📍 Model 2: Clustering (KMeans)

🧪 **Code**:

```
# Features and target
X = df[['location_encoded', 'month', 'hour', 'frequency']]
y = df['crime_type_encoded']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
# K-Means clustering for hotspots
kmeans = KMeans(n_clusters=5, random_state=42)
df['cluster'] = kmeans.fit_predict(df[['latitude', 'longitude']])

print("Classification and clustering models ready.")
```

💡 **Notes**:

- Random Forest handles noise and imbalanced data well.
- KMeans helps visualize and categorize spatial crime clusters.

📊 Step 5: Model Evaluation

Use quantitative metrics for classification and visual analysis for clustering.

🧪 **Code**:

```
# Classification results
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
print("Detailed Report:\n", classification_report(y_test, y_pred,
target_names=le_crime.classes_))

# Cluster overview
print("Crime density per cluster:")
print(df['cluster'].value_counts())
```

📈 **Insights**:

- The `classification_report` shows how well your model predicts each crime category (precision, recall, F1-score).
- KMeans clusters are evaluated qualitatively via visualizations.

📍 Step 6: Visualizing Results

Translate data into intuitive graphics to uncover trends and hotspot zones.

📊 **Code**:

```
# Line plot for crime frequency
plt.figure(figsize=(10, 6))
df.groupby('month')['frequency'].sum().plot(kind='line')
plt.title('Crime Frequency by Month')
plt.xlabel('Month')
plt.ylabel('Frequency')
plt.savefig('crime_trends.png')
plt.show()
```

```
# Folium crime hotspot map
m = folium.Map(location=[-1.95, 30.05], zoom_start=8)  # Kigali center
colors = ['red', 'blue', 'green', 'purple', 'orange']
for idx, row in df.iterrows():
    folium.CircleMarker(
        location=[row['latitude'], row['longitude']],
        radius=5,
        color=colors[row['cluster']],
        fill=True,
        fill_color=colors[row['cluster']],
        fill_opacity=0.7
    ).add_to(m)
m.save('crime_hotspots.html')

# Crime type distribution chart
plt.figure(figsize=(10, 6))
sns.countplot(x='crime_type', data=df)
plt.title('Distribution of Crime Types')
plt.xticks(rotation=45)
plt.savefig('crime_types.png')
plt.show()
```

🧠 **Explanation**:

- Visuals show trends over months, crime type distributions, and hotspot clusters.
- Stakeholders gain insight into "when", "where", and "what" crimes occur most.

💻 Step 7: Interactive Dashboard with Streamlit

Deploy an interactive dashboard to make the system usable by decision-makers.

📁 **File**: Save as `dashboard.py`

🧪 **Code**:

```
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import folium
from streamlit_folium import folium_static

# Load preprocessed data
df = pd.read_csv('rwanda_crime_data.csv')

st.title("🛡️ Predictive Policing Dashboard - Rwanda")

# Sidebar filters
st.sidebar.header("Filter Options")
```

```
location_opt = st.sidebar.selectbox("Select Location",
df['location'].unique())
month_opt = st.sidebar.slider("Select Month", 1, 12, 1)

# Filtered data
filtered_df = df[(df['location'] == location_opt) & (df['month'] ==
month_opt)]

# Trend line
st.subheader("📈 Crime Trends")
fig, ax = plt.subplots()
filtered_df.groupby('month')['frequency'].sum().plot(kind='line', ax=ax)
st.pyplot(fig)

# Hotspot map
st.subheader("🗺️ Crime Hotspots")
m = folium.Map(location=[-1.95, 30.05], zoom_start=8)
for _, row in filtered_df.iterrows():
    folium.CircleMarker(
        location=[row['latitude'], row['longitude']],
        radius=5,
        color='red',
        fill=True,
        fill_opacity=0.7
    ).add_to(m)
folium_static(m)

# Crime type distribution
st.subheader("🔎 Crime Type Distribution")
fig2, ax2 = plt.subplots()
sns.countplot(x='crime_type', data=filtered_df, ax=ax2)
plt.xticks(rotation=45)
st.pyplot(fig2)
```

💡 **Final Notes**:

- Use `streamlit run dashboard.py` to launch the app.
- Make sure to include your dataset and trained models in the working directory.
- You can also deploy it on platforms like **Streamlit Cloud** or **Render** for broader access.