Step-by-Step Guide: Data Processing and Model Training

1. Introduction

This document outlines the step-by-step process of data loading, preprocessing, model training, and evaluation using Python and scikit-learn.

---

2. Prerequisites

Ensure you have the following installed:

- Python (>=3.7)
- Pandas
- NumPy
- Scikit-learn
- Joblib

Install dependencies using:

```
pip install pandas numpy scikit-learn joblib
```

---

3. Load Dataset

- The dataset (FAOSTAT_data.csv) is loaded into a Pandas DataFrame.

```
import pandas as pd
df = pd.read_csv('FAOSTAT_data.csv')
```

4. Data Cleaning & Preprocessing

- Remove missing values:

```
df.dropna(inplace=True)
```

- Select relevant columns:

```
df = df[['Area', 'Item', 'Year', 'Element', 'Value']]
```

- Reshape the data using a pivot table:

```
df = df.pivot_table(index=['Area', 'Item', 'Year'],
columns='Element', values='Value').reset_index()
```

- Rename necessary columns:

```
column_mapping = {'Area harvested': 'Area_Harvested',
'Production': 'Production'}
df.rename(columns={k: v for k, v in column_mapping.items() if k in
df.columns}, inplace=True)
```

- Filter only relevant columns and drop any remaining missing values:

```
df = df[['Area', 'Item', 'Year', 'Area_Harvested',
'Production']].dropna()
```

- Compute 'Yield':

```
if 'Production' in df.columns and 'Area_Harvested' in df.columns:
    df['Yield'] = df['Production'] / df['Area_Harvested']
```

- Ensure required columns are present:

```
required_cols = ['Area_Harvested', 'Yield', 'Production']
missing_cols = [col for col in required_cols if col not in df.columns]
if missing_cols:
    raise ValueError(f"Missing expected columns: {missing_cols}")
```

- Validate dataset is not empty:

```
if df.shape[0] == 0:
    raise ValueError("Dataset is empty after preprocessing. Ensure filtering criteria are correct.")
```

---

5. Splitting Data for Model Training

- Define input (X) and output (y):

```
from sklearn.model_selection import train_test_split
X = df[['Area_Harvested', 'Yield', 'Year']]
y = df['Production']
```

- Split into training and testing sets:

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

---

6. Feature Scaling

- Standardize feature values:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

---

7. Model Training

- Train a Random Forest Regressor:

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)
```

- Save the trained model and scaler:

```
import joblib
joblib.dump(model, 'crop_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
```

---

8. Model Evaluation

- Make predictions and evaluate performance:

```
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score
y_pred = model.predict(X_test)
print(f'MAE: {mean_absolute_error(y_test, y_pred)}')
print(f'MSE: {mean_squared_error(y_test, y_pred)}')
print(f'R2 Score: {r2_score(y_test, y_pred)}')
```

---

9. Conclusion

This script loads and processes agricultural data, trains a Random Forest model to predict crop production, and evaluates its performance using standard metrics. The trained model and scaler are saved for future use.

Step-by-Step Explanation of Streamlitapp.py

1️⃣ Importing Required Libraries

```
import streamlit as st
import pandas as pd
import numpy as np
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
```

- **streamlit**: Used for building the web application.
- **pandas**: Handles data manipulation and analysis.
- **numpy**: Provides support for large, multi-dimensional arrays.
- **joblib**: Loads pre-trained machine learning models.
- **matplotlib & seaborn**: Used for data visualization.

2️⃣Loading and Preprocessing Data

```
class CropDataProcessor:
    def __init__(self, data_path, model_path, scaler_path):
        self.data_path = data_path
        self.model = joblib.load(model_path)
        self.scaler = joblib.load(scaler_path)
        self.df = self.load_and_preprocess_data()
```

- A **class CropDataProcessor** is defined to handle data loading and preprocessing.
- **Joblib** loads the pre-trained machine learning model and the scaler.

load_and_preprocess_data Method:

```
   def load_and_preprocess_data(self):
        df = pd.read_csv(self.data_path)
        df.dropna(inplace=True)
        df = df[['Area', 'Item', 'Year', 'Element', 'Value']]
        df = df.pivot_table(index=['Area', 'Item', 'Year'],
columns='Element', values='Value').reset_index()
```

- Reads the dataset from a CSV file.

- Drops missing values.
- Keeps only relevant columns.
- **Pivot table restructuring** helps format the data for easier analysis.

```
    if 'Area harvested' in df.columns:
        df.rename(columns={'Area harvested': 'Area_Harvested'},
inplace=True)
    if 'Production' in df.columns:
        df.rename(columns={'Production': 'Production'},
inplace=True)
```

- Renames columns for consistency.

```
    df = df[['Area', 'Item', 'Year', 'Area_Harvested',
'Production']].dropna()
```

- Ensures only relevant columns remain and drops any missing values.

```
    if 'Production' in df.columns and 'Area_Harvested' in
df.columns:
        df['Yield'] = df['Production'] / df['Area_Harvested']
    return df
```

- **Calculates Yield**: Yield = Production / Area Harvested.

3️⃣ Streamlit App Title

```
data_processor = CropDataProcessor('FAOSTAT_data.csv',
'crop_model.pkl', 'scaler.pkl')
```

```
df = data_processor.df

st.title("Crop Production Analysis & Prediction")
```

- Initializes the CropDataProcessor and loads the dataset.
- Displays the application title.

4️⃣ Crop Type Distribution Analysis

```
st.subheader("1. Crop Distribution Analysis")
crop_counts = df['Item'].value_counts()
```

- Displays a subheading for crop distribution.
- Counts occurrences of each crop.

```
fig, ax = plt.subplots(figsize=(10, 5))
sns.barplot(x=crop_counts.index[:10], y=crop_counts.values[:10], ax=ax)
ax.set_title("Top 10 Most Cultivated Crops")
ax.set_ylabel("Count")
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
st.pyplot(fig)
```

- **Plots a bar chart** for the **top 10 most cultivated crops** using Seaborn.

5️⃣ Geographical Distribution

```
st.subheader("Geographical Crop Distribution")
selected_crop = st.selectbox("Select Crop", df['Item'].unique())
```

```
crop_area_counts = df[df['Item'] ==
selected_crop]['Area'].value_counts()
```

- Displays crop distribution per region.
- Uses a dropdown (selectbox) for crop selection.

6️⃣ Yearly Trends in Agriculture

```
st.subheader("Yearly Trends in Agriculture")
yearly_trends = df.groupby("Year")[["Area_Harvested", "Yield",
"Production"]].mean().reset_index()
```

- Aggregates **yearly averages** of harvested area, yield, and production.

```
fig, ax = plt.subplots(figsize=(10, 5))
sns.lineplot(data=yearly_trends, x="Year", y="Area_Harvested",
label="Area Harvested", ax=ax)
sns.lineplot(data=yearly_trends, x="Year", y="Yield", label="Yield",
ax=ax)
sns.lineplot(data=yearly_trends, x="Year", y="Production",
label="Production", ax=ax)
ax.set_title("Yearly Trends: Area Harvested, Yield, and
Production")
st.pyplot(fig)
```

- **Plots trends over years** for area harvested, yield, and production.

7️⃣ Growth Analysis

```
analysis_type = st.radio("Analyze Growth by:", ["Item", "Area"])
selected = st.selectbox(f"Select {analysis_type}",
df[analysis_type].unique())
```

- Users can analyze growth by **crop or region**.

8️⃣ Environmental Relationships

```
st.subheader("Environmental Relationships")
fig, ax = plt.subplots(figsize=(10, 5))
sns.scatterplot(data=df, x="Area_Harvested", y="Yield",
alpha=0.5)
ax.set_title("Impact of Area Harvested on Yield")
st.pyplot(fig)
```

- **Plots a scatterplot** to visualize **the relationship between area harvested and yield**.

9️⃣ Production Prediction

```
st.subheader("Production Prediction")
region = st.selectbox("Select Region", df['Area'].unique(),
key="pred_region")
crop = st.selectbox("Select Crop", df['Item'].unique(),
key="pred_crop")
year = st.number_input("Enter Year",
min_value=int(df['Year'].min()), max_value=int(df['Year'].max()) +
10, step=1)
area_harvested = st.number_input("Enter Area Harvested",
min_value=0.0, step=0.1)
```

- **User inputs required parameters** for production prediction.

```
if st.button("Predict Production"):
    try:
        input_data = pd.DataFrame([[region, crop, year, area_harvested]], columns=['Area', 'Item', 'Year', 'Area_Harvested'])
        input_data = pd.get_dummies(input_data).reindex(columns=data_processor.scaler.feature_names_in_, fill_value=0)
        input_scaled = data_processor.scaler.transform(input_data)
        prediction = data_processor.model.predict(input_scaled)
        st.write(f"Predicted Production: {prediction[0]:,.2f} tons")
    except Exception as e:
        st.error(f"Error in prediction: {e}")
```

- **Encodes categorical features** to match the trained model.
- **Scales input data** using the preloaded scaler.
- **Predicts production** using the trained model.
- Displays the prediction or an error message if an issue occurs.

This step-by-step guide breaks down each line of code in Streamlitapp.py, explaining its purpose and functionality. 🚀