**Name: Sabatini H**

**Project :Crop Production Prediction**

**Data Visualization**

Visualizing data can help us understand patterns, trends, and relationships within the dataset. Here are some common ways to visualize the data in your df\_pivot DataFrame using matplotlib and seaborn.

Let's look at a few examples:

1. **Production Trend over Time for a Specific Crop and Area:** This will show how the production of a particular crop in a specific region has changed over the years.
2. **Top N Producing Areas for a Specific Crop in a Given Year:** This will highlight the regions with the highest production for a chosen crop in a particular year.
3. **Yield Distribution for a Specific Crop:** This can help understand the spread and central tendency of yield for a chosen crop across different areas and years.

**Crop production trends by region, crop type and year**

To analyze crop production trends by region, crop type, and year, we can leverage the df\_pivot DataFrame we created. This DataFrame is already structured to show production, area harvested, and yield for each combination of area, item (crop), and year.

We can use filtering and grouping techniques to extract the data needed for analysis. For visualization, line plots are excellent for showing trends over time. We can plot the production of a specific crop across different regions, or compare trends of different crops within a region.

Here are a few examples demonstrating how to achieve this:

1. **Compare Production Trends of a Specific Crop Across Multiple Regions:** This will show how the production of one crop varies over time in several selected areas.
2. **Compare Production Trends of Multiple Crops in a Specific Region:** This will allow you to see the production trends of different crops within a single area.
3. **Overall Production Trend for a Specific Crop Globally (or a Large Aggregate Area):** If 'World' or a similar aggregate area exists in your data, you can visualize the global trend.

Predictive modelling, develop model to predict crop yield or production for informed decision making

Building a predictive model for crop yield or production is a common application of machine learning in agriculture (as highlighted by sources like [1] and [2]). This involves using historical data, potentially combined with other factors like weather or soil conditions (though your current dataset primarily contains historical production data), to train a model that can forecast future values.

Here's a basic outline of the steps involved and a simple example using a linear regression model. Keep in mind that this is a simplified approach for demonstration, and a real-world model would involve more complex data preparation, feature engineering, and model selection.

**Steps:**

1. **Feature Engineering:** Create relevant features from your existing data. For time series data like this, previous years' values can be strong predictors.
2. **Data Splitting:** Divide your data into training and testing sets. The model will learn from the training data and be evaluated on the unseen testing data.
3. **Model Selection:** Choose a suitable model. Simple models like Linear Regression can be a starting point. More advanced models like Random Forests, Gradient Boosting (e.g., LightGBM, XGBoost), or even time series specific models (like ARIMA or LSTMs if you have enough data points per series) could be considered.
4. **Training:** Train the chosen model on the training data.
5. **Evaluation:** Assess the model's performance on the testing data using appropriate metrics (e.g., Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R-squared).
6. **Prediction:** Use the trained model to make predictions on future data.

Let's demonstrate with a simple Linear Regression model to predict 'Production\_tonnes' for a specific crop and area based on the 'Year'.

**Explanation:**

* We select a specific crop and area to focus on for this simple model.
* We filter the df\_pivot DataFrame to get the data for the chosen crop and area.
* We prepare the data, with 'Year' as the feature (input) and 'Production\_tonnes' as the target (output). We use values to get NumPy arrays and reshape 'Year' into a 2D array as required by scikit-learn.
* train\_test\_split divides the data into training and testing sets. random\_state ensures reproducibility of the split.
* We initialize a LinearRegression model and train it using the fit() method on the training data.
* We use the predict() method to make predictions on the test data.
* We evaluate the model using mean\_squared\_error, rmse, and r2\_score. Lower MSE and RMSE indicate better performance (less error), and an R-squared closer to 1 indicates that the model explains more of the variance in the target variable.
* We then demonstrate how to make predictions for future years.
* Finally, we plot the actual vs. predicted values to visually assess the model's fit and show the future predictions.

**Limitations of this simple model and next steps:**

* **Only uses Year:** This model only considers the year. Real-world crop yield is influenced by many factors like weather (rainfall, temperature), soil type, fertilizers, pesticides, technology, and government policies. Incorporating these features would likely improve accuracy.
* **Linearity Assumption:** Linear Regression assumes a linear relationship between the year and production, which might not always hold true over long periods due to technological advancements, climate change, etc.
* **Data Availability:** The model's performance is highly dependent on the amount and quality of historical data for the specific crop and area.
* **Overfitting:** With more complex models, you need to be careful about overfitting, where the model performs well on training data but poorly on unseen data. Techniques like cross-validation can help.
* **Time Series Nature:** This is a simple regression model applied to time series data. More sophisticated time series forecasting models might capture temporal dependencies better.

For a more robust predictive model, you would need to:

* **Gather more features:** Integrate data on weather, soil, input costs, etc.
* **Perform Feature Engineering:** Create lagged variables (e.g., production in the previous year), moving averages, or other time series features.
* **Explore different models:** Experiment with models like Random Forest Regressor, Gradient Boosting Regressor, or time series models.
* **Hyperparameter Tuning:** Optimize the parameters of your chosen model.
* **Cross-Validation:** Use cross-validation techniques to get a more reliable estimate of the model's performance.

**Streamlit Application**

Creating a Streamlit application involves building a separate Python script that uses the streamlit library. You would typically take the predictive modeling code you developed and integrate it into the Streamlit app structure.

Here's a basic structure of how you could create a Streamlit application to show predicted crop production based on the year, for a selected crop and region.

**Steps:**

1. **Install Streamlit:** If you haven't already, you'll need to install Streamlit.
2. **Create a New Python File:** Create a new .py file (e.g., crop\_predictor\_app.py).
3. **Import Libraries:** Import necessary libraries like streamlit, pandas, sklearn, numpy, and matplotlib.
4. **Load Data:** Load your data (the df\_pivot DataFrame) within the Streamlit app.
5. **Add Streamlit Widgets:** Use Streamlit's widgets (like selectbox, number\_input, slider) to allow the user to select the crop, region, and input the year for prediction.
6. **Implement Prediction Logic:** Based on the user's selections, filter the data, train the model (or load a pre-trained model), and make a prediction.
7. **Display Results:** Use Streamlit functions (like write, pyplot, line\_chart) to display the input values, the prediction, and potentially a visualization of the historical data and the prediction.
8. **Run the App:** Open your terminal, navigate to the directory where you saved the Python file, and run the command streamlit run your\_app\_file\_name.py

A screenshot of a computer

AI-generated content may be incorrect.

A graph with a line and a dotted line

AI-generated content may be incorrect.