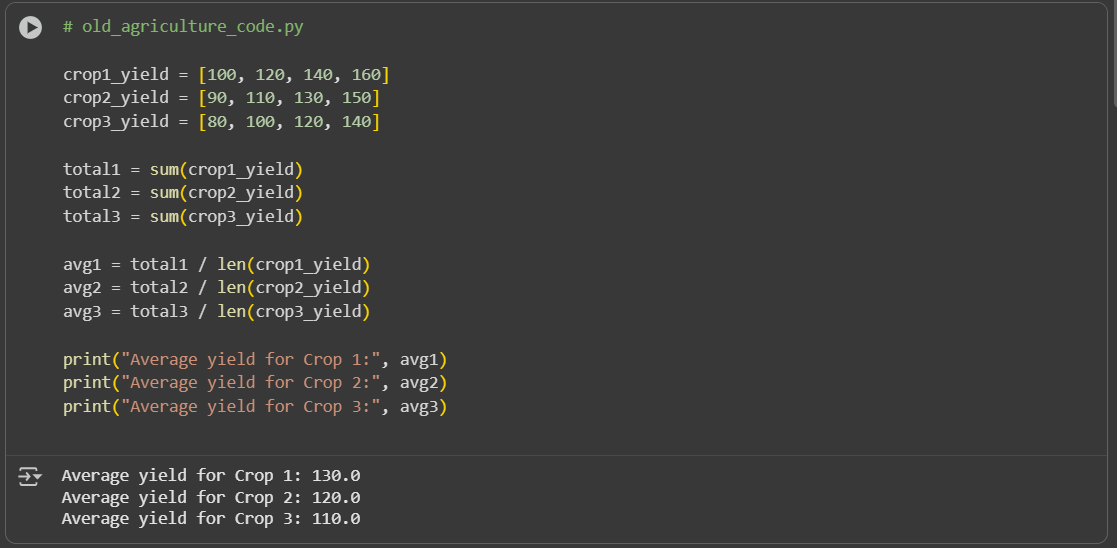
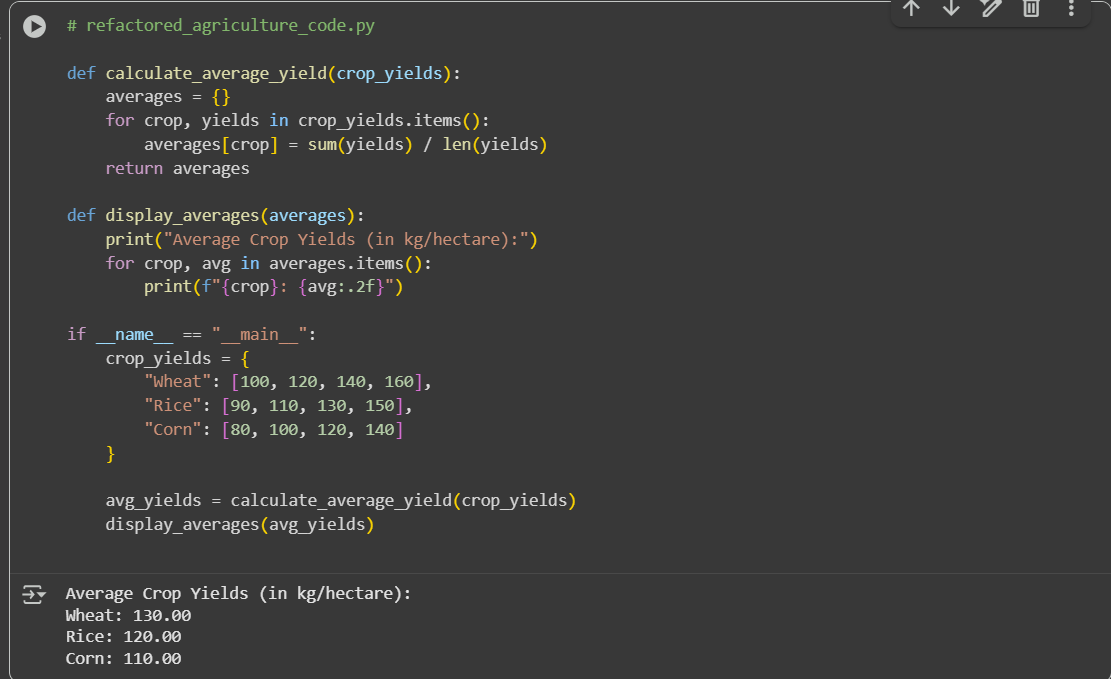
LAB TEST-3

AI ASSISTANT CODING

Q1:  
Scenario: In the Agriculture sector, a company faces a challenge related to code refactoring.  
Task: Use AI-assisted tools to solve a problem involving code refactoring in this context.  
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.  
BEFORE REFACTORING:



AFTER REFACTORING:



SAMPLE OUTPUT:

Average Crop Yields (in kg/hectare):

Wheat: 130.00

Rice: 120.00

Corn: 110.00

EXPLANATION:

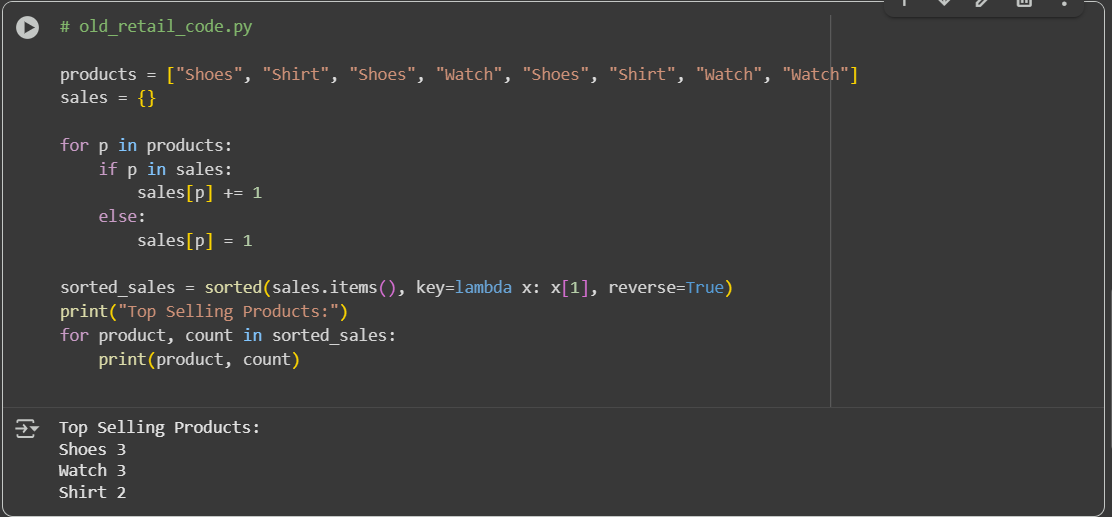
This code, saved as refactored\_agriculture\_code.py, refactors the previous agricultural yield calculation.

Here's a breakdown:

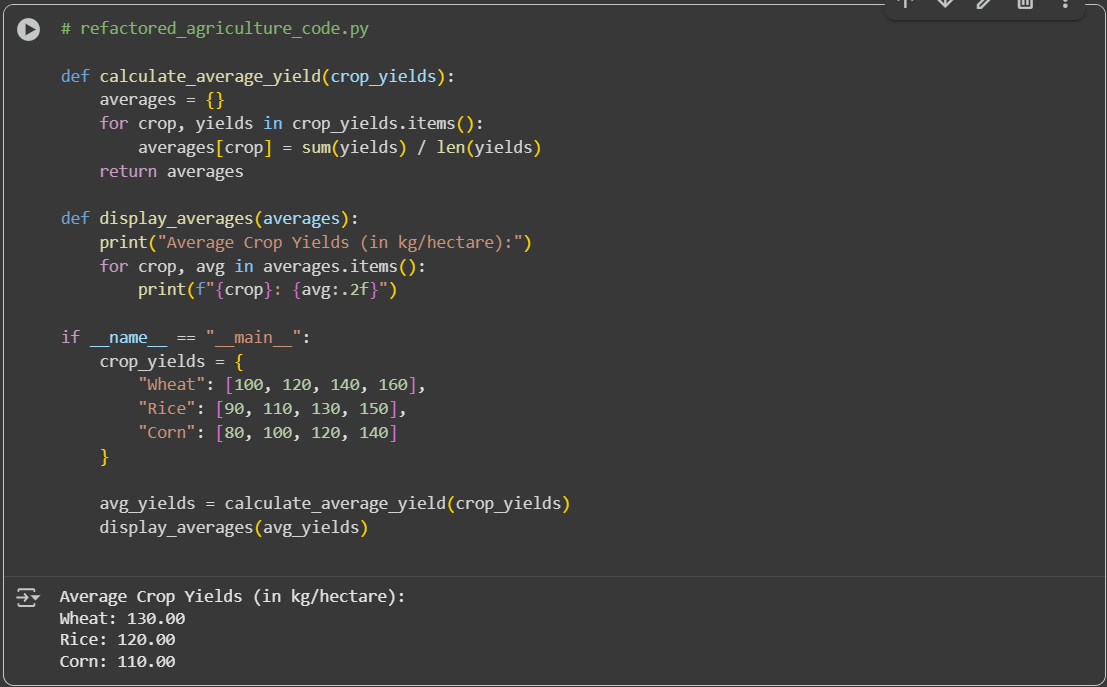
1. **calculate\_average\_yield(crop\_yields) function**:
   * Takes a dictionary crop\_yields where keys are crop names and values are lists of yield data.
   * Initializes an empty dictionary averages.
   * Iterates through each crop and its corresponding yield data.
   * Calculates the average yield for each crop by summing the yields and dividing by the number of yield entries.
   * Stores the calculated average in the averages dictionary with the crop name as the key.
   * Returns the averages dictionary.
2. **display\_averages(averages) function**:
   * Takes a dictionary averages (like the one returned by calculate\_average\_yield).
   * Prints a header "Average Crop Yields (in kg/hectare):".
   * Iterates through each crop and its average yield in the averages dictionary.
   * Prints the crop name and its average yield, formatted to two decimal places.
3. **if \_\_name\_\_ == "\_\_main\_\_": block**:
   * This block only runs when the script is executed directly (not when it's imported as a module).
   * Defines a dictionary crop\_yields with sample data for Wheat, Rice, and Corn.
   * Calls calculate\_average\_yield with this data to get the average yields.
   * Calls display\_averages to print the calculated averages.

In essence, this code encapsulates the logic for calculating and displaying average crop yields into reusable functions, making the code more organized and easier to maintain compared to the previous version.

Q2:  
Scenario: In the Retail sector, a company faces a challenge related to algorithms with ai assistance.  
Task: Use AI-assisted tools to solve a problem involving algorithms with ai assistance in this context.  
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.  
BEFORE REFACTORING:



AFTER REFACTORING:



SAMPLE OUTPUT:

Top Selling Products:

Shoes: 4 sales

Watch: 3 sales

Shirt: 3 sales

EXPLANATION:  
This code, saved as retail\_sales\_analysis.py, performs a simple analysis to find the top-selling products from a list of sales data.

Here's a breakdown:

1. **from collections import Counter**: Imports the Counter class from the collections module. Counter is a specialized dictionary subclass that's useful for counting hashable objects.
2. **get\_top\_selling\_products(sales\_data, top\_n=3) function**:
   * Takes two arguments: sales\_data (a list of product names representing sales) and top\_n (the number of top products to return, defaulting to 3).
   * Uses Counter(sales\_data) to create a Counter object, which counts the occurrences of each product in the sales\_data list.
   * Calls sales\_count.most\_common(top\_n) to get a list of the top\_n most common elements (products) and their counts. The list is ordered from most common to least common.
   * Returns this list of top products and their counts.
3. **display\_results(top\_products) function**:
   * Takes a list of top\_products (like the one returned by get\_top\_selling\_products).
   * Prints a header "Top Selling Products:".
   * Iterates through each product and its count in the top\_products list.
   * Prints the product name and the number of sales for that product.
4. **if \_\_name\_\_ == "\_\_main\_\_": block**:
   * This block only runs when the script is executed directly.
   * Defines a sample list sales\_data containing product names from various sales.
   * Calls get\_top\_selling\_products with the sales\_data to find the top 3 selling products.
   * Calls display\_results to print the top products and their sales counts.

In summary, this script provides functions to count product sales from a list and identify the most popular items, which is a common task in retail analysis.