Plan a Budget and Expense Tracker: a command-line application that allows a user to input their expenses and income. The script will save the data to a CSV file and display summaries, perhaps broken down by category.

Step 1: Install the following python libraries

- Import csv
- Import datetime
- Import pathlib

Step 2: Write the Python script

Create a new Python file (Budget and Expense Tracker.py) and complete the following tasks

- Data Storage:

CSV File: A CSV file will serve as the primary data storage. Each row in the CSV will represent a transaction (either income or expense) and will contain fields such as:

Date: The date of the transaction (e.g., YYYY-MM-DD).

Type: "Income" or "Expense".

Category: A user-defined category for the transaction (e.g., "Groceries", "Salary", "Rent").

Amount: The monetary value of the transaction.

Description: An optional text description of the transaction.

- Core Functionality:

Add Transaction:

Prompt the user for Date, Type, Category, Amount, and Description.

Validate input (e.g., amount is a number, date format is correct).

Append the new transaction as a row to the CSV file.

View Transactions:

Read all transactions from the CSV file.

Display them in a clear, tabular format, potentially allowing for sorting by date, category, or amount.

- Summarize Data:

Total Income/Expenses: Calculate and display the total income and total expenses for a given period (e.g., month, year, or all time).

Category Breakdown: Group transactions by category and display the total amount spent or earned in each category.

Balance: Calculate the current balance (total income - total expenses).

Delete Transaction (Optional but Recommended):

Allow users to delete a transaction, perhaps by providing a unique identifier or by selecting from a displayed list.

Edit Transaction (Optional but Recommended):

Allow users to modify details of an existing transaction.

- User Interface (Command-Line):

Main Menu: Present a clear menu with options like "Add Transaction", "View Transactions", "Summaries", "Exit".

Input Prompts: Use clear and concise prompts for user input.

Error Handling: Provide informative error messages for invalid input or file operations.

- Implementation Considerations:

Python Libraries: The csv module for CSV file handling and potentially datetime for date manipulation.

Modularity: Organize the code into functions for better readability and maintainability (e.g., add_transaction(), load_data(), save_data(), generate_summary()).

File Paths: Consider how the CSV file path will be managed (e.g., a default location, user-configurable).

Step 3: Interpret the output and learn from it. Play with the controls and note down what each action/step is achieving