Mark as done

Description Submission view

Available from: Wednesday, 12 November 2025, 9:15 AM
Due date: Wednesday, 12 November 2025, 10:45 AM
Requested files: csvtool_p1.py, csvtool_p2.py, p1_input.txt, p2_input.txt, student_data_p1.csv, student_data_p2.csv, p1.py, p2.py (Download)
Maximum number of files: 9
Type of work: Individual work

Problem 1 - CSV Tool: Filter Rows

Description:

Your task is to write a Python function named `filter_rows` in the file **`csvtool_p1.py`**. This function will be the core logic for a tool that filters rows from a CSV file based on one or more conditions. The task of parsing condition strings has been handled for you; your function will receive neatly formatted tuples.

Function Specification: `filter_rows`

You must implement the function with this **exact** signature, using the types provided from the `typing` module *(already imported in your starter file)* in the starter code.

```
from typing import List, Dict, Any, Tuple

def filter_rows(file: str, conditions: List[Tuple[str, str, str]]) -> List[Dict[str, Any]]:
```

Parameters:

- `file`: The path to the CSV file (a string).
- `conditions`: A list of tuples. Each tuple represents a filter condition in the format `(column_name, operator, value)`. For example: `[('Age', '>=', '32'), ('Department', '==', 'HR')]`.

Returns:

- A list of dictionaries. Each dictionary represents a row that matches **all** specified conditions. The keys of the dictionary are the column headers from the CSV file. **Your function must `return` this list and should not use `print()`.**

Core Logic and File Handling:

- **If the provided CSV file is empty (contains no header and no data), your function should immediately return an empty list `[]`, regardless of any conditions provided.**
- For non-empty files, the first line is always the header.
- Supported operators are: `==`, `!=`, `>`, `<`, `>=`, `<=`.
- For operators `>`, `<`, `>=`, `<=`, you must attempt to convert both the value from the CSV and the value in the condition tuple to numbers (floats) for comparison.
- For operators `==` and `!=`, you must perform a direct string comparison.

Required Exceptions:
Your function must `raise` the following exceptions with precisely formatted messages. Note that these should only be raised for **non-empty** files.

- **File Not Found:** Raise a `FileNotFoundError`. Message: `"File <filename> not found"`.
- **Invalid Column:** Raise a `KeyError` if a column from a condition tuple does not exist in the CSV header. Message: `"Column <column_name> not found"`.
- **Type Mismatch for Comparison:** Raise a `TypeError` if a numeric comparison (e.g., `>`) is attempted on a value that cannot be converted to a number. Message: `"Cannot perform numeric comparison on non-numeric value <value> in column <column_name>"`. **Note that <value> here is value of the data row and not the value in the condition**

Examples

?

Assume `student_data_p1.csv` contains:

```
Name,Department,Salary,Age
Alice,Engineering,80000,32
Bob,Marketing,65000,28
Charlie,Engineering,95000,45
Diana,HR,55000,32
```

Example 1: Filter by Department
Test Command: `python3 csvtool_p1.py -filter student_data_p1.csv 'Department==Engineering'`
Your Function's Behavior: Must return a list containing two dictionaries for Alice and Charlie.
Final Output on Screen (printed by the provided tool):

```
[{'Name': 'Alice', 'Department': 'Engineering', 'Salary': '80000', 'Age': '32'}, {'Name': 'Charlie', 'Department': 'Engineering',
'Salary': '95000', 'Age': '45'}]
```

Example 2: Filter by Age and Salary
Test Command: `python3 csvtool_p1.py -filter student_data_p1.csv 'Age>=32' 'Salary<90000'`
Your Function's Behavior: Must return a list containing dictionaries for Alice and Diana.
Final Output on Screen (printed by the provided tool):

```
[{'Name': 'Alice', 'Department': 'Engineering', 'Salary': '80000', 'Age': '32'}, {'Name': 'Diana', 'Department': 'HR', 'Salary':
'55000', 'Age': '32'}]
```

Example 3: Invalid Column Error
Test Command: `python3 csvtool_p1.py -filter student_data_p1.csv 'Location==London'`
Your Function's Behavior: Must `raise KeyError("Column Location not found")`.
Final Output on Screen (printed to stderr by the tool):

```
KeyError: Column Location not found
```

How to Test Your Code Locally

The "Run" button executes the tests defined in `p1_input.txt`. You can create your own tests by modifying these files.

A step-by-step example of creating a custom test:

1. **Objective:** Test a simple filter on a small, custom CSV file.
2. **Step 1: Modify `student_data_p1.csv`**. Change its content to:
   ```
   Name,Status
   Alice,Active
   Bob,Inactive
   ```
3. **Step 2: Modify `p1_input.txt`**. Change its content to a single test command that filters for active users:
   ```
   1
   python3 csvtool_p1.py -filter student_data_p1.csv 'Status==Active'
   ```
4. **Step 3: Click "Run"**. The runner script (`p1.py`) will execute your command. The expected final output on the screen should be:
   ```
   Executing: python3 csvtool_p1.py -filter student_data_p1.csv 'Status==Active'

   --- Tool Output ---
   [{'Name': 'Alice', 'Status': 'Active'}]
   -------------------
   ```

Notes:

- You will write your code in **`csvtool_p1.py`**. Do not modify any other files.
- You **CANNOT** use Python's built-in `csv` module.
- **You can assume that all input CSV files are well-formed. This means every row in a given file will have the same number of columns as its header.**

Problem 2 - CSV Tool: Group and Aggregate

Description:

In the file **`csvtool_p2.py`**, your task is to write a Python function named `group_by`. This function will group rows from a CSV by a specific column and then perform an aggregation (like sum or average) on another column for each group. The result will be saved to a new CSV file.

Function Specification: `group_by`

You must implement the function with this **exact** signature.

```
def group_by(file: str, keycol: str, aggcol: str, op: str, out: str) -> None:
```

Parameters:

- `file`: The path to the source CSV file.
- `keycol`: The name of the column to group the rows by.
- `aggcol`: The name of the column whose values will be aggregated.
- `op`: The aggregation operation to perform. Must be one of: `"sum"`, `"avg"`, `"min"`, `"max"`.
- `out`: The path for the destination (output) CSV file.

Returns / Output:

- Your function should **not print anything** and should **not return any value**. Its job is to create the output file or raise an exception.
- The output CSV file must have exactly two columns, with the header being `<keycol>,<aggcol>`.
- The aggregated value in the output file must always be a **float**. For example, the sum of integers 100 and 120 should be written as `220.0`.
- For the **`avg`** operation, the result in the output file must be formatted to exactly **two decimal places**.
- **Special Case:** If the input file contains no data rows (i.e., it is empty or only contains a header), your function should not raise an error. It should create an output file containing only the header row `<keycol>,<aggcol>` (e.g., `Region,Revenue`). If the file contains only a header, **the column names should still be validated**.

Required Exceptions:

- **File Not Found:** Raise a `FileNotFoundError`. Message: `"File <filename> not found"`.
- **Invalid Column:** Raise a `KeyError` if `keycol` or `aggcol` do not exist in the CSV header. Message: `"Column <column_name> not found"`.
- **Invalid Operation:** Raise a `ValueError` if `op` is not one of the allowed operations. Message: `"Invalid operation: <op>"`.
- **Type Mismatch for Aggregation:** Raise a `TypeError` if a numeric operation is attempted on a column with non-numeric data. Message: `"Cannot perform <op> on non-numeric value <value> in column <aggcol>"`.
- **Permission Denied:** Raise a `PermissionError` if writing to the output file is not permitted. Message: `"Permission denied: cannot write to <out_filename>"`.

Examples

For the following examples, assume `sales_data_p2.csv` contains:

```
Region,Product,Revenue,UnitsSold
North,Widget,5000,100
South,Gadget,7500,150
North,Gadget,8000,120
West,Widget,4000,80
South,Widget,6000,140
```

Example 1: Sum Revenue by Region

Test Command: `python3 csvtool_p2.py -groupby sales_data_p2.csv Region Revenue sum region_revenue.csv`
Your Function's Behavior: Must create the `region_revenue.csv` file. The order of rows in the output file does not matter.
Content of `region_revenue.csv`:

```
Region,Revenue
North,13000.0
South,13500.0
West,4000.0
```

Example 2: Average UnitsSold by Product

Test Command: `python3 csvtool_p2.py -groupby sales_data_p2.csv Product UnitsSold avg product_avg_units.csv`
Your Function's Behavior: Must create the `product_avg_units.csv` file. Note how the average for "Widget" ((100+80+140)/3 = 106.666...) is correctly rounded and formatted to two decimal places.
Content of `product_avg_units.csv`:

```
Product,UnitsSold
Gadget,135.00
Widget,106.67
```

Example 3: Invalid Column Error

Test Command: `python3 csvtool_p2.py -groupby sales_data_p2.csv Country Revenue sum country_totals.csv`
Your Function's Behavior: Must `raise KeyError("Column Country not found")`.
Final Output on Screen (printed to stderr by the tool):

```
KeyError: Column Country not found
```

Example 4: Handling a File with No Data Rows

Scenario: A file named `header_only.csv` exists and contains only the line `Region,Revenue`.
Test Command: `python3 csvtool_p2.py -groupby header_only.csv Region Revenue sum totals.csv`

Your Function's Behavior: The operation is valid, but there is no data to aggregate. The function must create an output file with only the header.
Content of `totals.csv`:

```
Region,Revenue
```

How to Test Your Code Locally

When you click the "Run" button, the `p2.py` script executes the tests from `p2_input.txt`. After your function runs, the script will show you a **preview** of the file that was created. This is how you verify your aggregation worked correctly.

A step-by-step example of creating a custom test:

1. **Objective:** Test if your function can correctly sum sales for different departments.
2. **Step 1: Modify `student_data_p2.csv`**. Change its content to:
```
Dept,Sales
HR,100
Eng,200
HR,150
```
3. **Step 2: Modify `p2_input.txt`**. Set up a single command to group these sales into a new file called `dept_sales.csv`:
```
1
python3 csvtool_p2.py -groupby student_data_p2.csv Dept Sales sum dept_sales.csv
```
4. **Step 3: Click "Run"**. Since a successful aggregation produces no direct output, the runner script helps by showing a preview of the resulting file. The complete output you see on screen should be:
```
Executing: python3 csvtool_p2.py -groupby student_data_p2.csv Dept Sales sum dept_sales.csv

--- Output from your program ---
[No output is produced]
------------------------------

======= PREVIEW of created file: 'dept_sales.csv' =======
Dept,Sales
Eng,200.0
HR,250.0
================ END OF PREVIEW ===============
```

Notes:

- You will write your code in **`csvtool_p2.py`**. Do not modify any other files.
- You **CANNOT** use Python's built-in `csv` module.
- **You can assume that all input CSV files are well-formed. This means every row in a given file will have the same number of columns as its header.**

Requested files

csvtool_p1.py

```python
#!/usr/bin/env python3

# You may add supporting functions, but do not change the signature of filter_rows.
# You are NOT allowed to use the 'csv' module, 're' module, or 'collections' module.

from lab_utils import run_csvtool_p1_main
from typing import List, Dict, Any, Tuple

# ==================================================================================
# ================== WRITE YOUR FUNCTION IN THE SPACE BELOW =======================
#
# Refer to the Problem 1 description for the EXACT specifications.
#
# ==================================================================================




# ==================================================================================
# ================== DO NOT MODIFY THE CODE BELOW THIS LINE =======================
# ==================================================================================

if __name__ == '__main__':
    # The run_csvtool_p1_main function (in lab_utils) handles parsing the command-line
    # arguments into tuples before calling your filter_rows function.

    # This will fail until you have defined the filter_rows function.
    run_csvtool_p1_main(filter_rows_func=filter_rows)
```

csvtool_p2.py

```python
#!/usr/bin/env python3

# You may add supporting functions, but do not change the signature of group_by.
# You are NOT allowed to use the 'csv' module, 're' module, or 'collections' module.

from lab_utils import run_csvtool_p2_main
from typing import List, Dict, Any, Tuple

# ===================================================================================
# =================== WRITE YOUR FUNCTION IN THE SPACE BELOW ========================
#
# Refer to the Problem 2 description for the EXACT specifications.
#
# ===================================================================================






# ===================================================================================
# =================== DO NOT MODIFY THE CODE BELOW THIS LINE ========================
# ===================================================================================

if __name__ == '__main__':
    # This will fail until you have defined the group_by function.
    run_csvtool_p2_main(group_by_func=group_by)
```

p1_input.txt

```
2
python3 csvtool_p1.py -filter student_data_p1.csv Department==Engineering
python3 csvtool_p1.py -filter student_data_p1.csv Age>=32 Salary<90000
```

p2_input.txt

```
1
python3 csvtool_p2.py -groupby student_data_p2.csv Region Revenue sum region_revenue.csv
```

student_data_p1.csv

```
Name,Department,Salary,Age
Alice,Engineering,80000,32
Bob,Marketing,65000,28
Charlie,Engineering,95000,45
Diana,HR,55000,32
```

student_data_p2.csv

```
Region,Product,Revenue,UnitsSold
North,Widget,5000,100
South,Gadget,7500,150
North,Gadget,8000,120
West,Widget,4000,80
South,Widget,6000,140
```

p1.py

```python
import subprocess
import shlex

############################ Do Not Change This File #################################
def csvtool_caller(command: str) -> str:
    """Execute csvtool command."""
    try:
        args = shlex.split(command)
        result = subprocess.run(
            args, capture_output=True, text=True, timeout=2
        )
        if result.returncode != 0:
            return result.stderr.strip()
        return result.stdout.strip()
    except subprocess.TimeoutExpired:
        return "Error: Command timed out"
    except Exception as e:
        return f"Error: Failed to execute command - {str(e)}"

def solution(inp: str) -> str:
    """Wrapper function that the evaluator calls."""
    return csvtool_caller(inp)

def process_input(filename):
    lines = open(filename, 'r').readlines()
    lines = [line.strip() for line in lines if line.strip()]
    num_tests = int(lines[0])
    input_tests = [lines[t].strip() for t in range(1, num_tests + 1)]
    return input_tests

if __name__ == "__main__":
    try:
        Input = process_input('p1_input.txt')
        for command in Input:
            print(f"Executing: {command}\n")
            output = csvtool_caller(command)
            print(f"--- Tool Output ---")
            print(output if output else "[No output produced]")
            print("-------------------\n")
    except FileNotFoundError:
        print("p1_input.txt not found.")
```

p2.py

```python
import subprocess
import shlex
import os

############################ Do Not Change This File #################################
def csvtool_caller(command: str) -> str:
    """Execute csvtool command."""
    try:
        args = shlex.split(command)
        result = subprocess.run(
            args, capture_output=True, text=True, timeout=2
        )
        if result.returncode != 0:
            return result.stderr.strip()
        return result.stdout.strip()
    except subprocess.TimeoutExpired:
        return "Error: Command timed out"
    except Exception as e:
        return f"Error: Failed to execute command - {str(e)}"

def solution(inp: str) -> str:
    """Wrapper function that the evaluator calls."""
    return csvtool_caller(inp)

def process_input(filename):
    lines = open(filename, 'r').readlines()
    lines = [line.strip() for line in lines if line.strip()]
    num_tests = int(lines[0])
    input_tests = [lines[t].strip() for t in range(1, num_tests + 1)]
    return input_tests

if __name__ == "__main__":
    try:
        Input = process_input('p2_input.txt')
        for command in Input:
            print(f"Executing: {command}")

            program_output = csvtool_caller(command)

            print(f"\n--- Output from your program ---\n{program_output if program_output else '[No output is produced]'}\n-----------------

            try:
                parts = shlex.split(command)
                if '-groupby' in parts:
                    dest_filename = parts[-1]

                    print(f"======= PREVIEW of created file: '{dest_filename}' =======")
                    if os.path.exists(dest_filename):
                        with open(dest_filename, 'r') as f:
                            content = f.read()
                        print(content.strip() if content else "[ The file was created but is empty. ]")
                    else:
                        print(f"[ File '{dest_filename}' was not found. ]")
                    print("================ END OF PREVIEW ================\n")
            except Exception as e:
                print(f"[ Could not generate file preview due to an error: {e} ]\n")

    except FileNotFoundError:
        print("p2_input.txt not found. Cannot run local tests.")
    except Exception as e:
        print(f"An unexpected error occurred in the runner script: {e}")
```

VPL