Overview and Initial Planning

The goal was to develop a program capable of parsing AWS VPC flow logs, applying tags to each row based on the given lookup table CSV, and generating 2 summary reports. To achieve this, the initial plan involved using only standard python libraries as per the requirements. I structured the program into 4 clearly defined functions:

- *load_tag_lookup()*: This method parses the lookup table and returns a dict to assign the dstport and protocol to each tag
- *process_flow_logs()*: This method parses the flow log, specifically looking at the dstport and protocol of each line. It looks up the tag, incrementing both the tag count and the dstport, protocol count
- *write_tag_counts()*: Writes the tag counter to a csv alphabetically, with untagged at the bottom
- *write_port_protocol_counts()*: Writes the port/protocol counts to a csv.

Dictionaries were used for constant-time lookups, and we used *defaultDict(int)* for the counting logic for a more clean and efficient counting method.

Function Development

load_tag_lookup()

The first step was to actually load the tag lookup table. I initially used csv.reader with hardcoded column indices, but decided to switch to csv.DictReader for more clarity. Since CSV files can be problematic due to BOMs and inconsistent whitespaces, I decided to incorporate some measures to handle these situations. Using methods like *.strip().lower()* and *.replace('\ufeff', '')*, the program normalized both headers and row keys. This also validated rows for missing or empty values and skipped any malformed entries alongside debug messages.

process_flow_logs()

In this function, I read each log line splitting each line by spaces and processing it into a list. Components of the log such as the version needed to be validated, and unvalidated versions were skipped. Initially, I misunderstood the formatting of the log format, extracting the srcport instead of the dstport. This was fixed by reading the AWS documentation more in depth, and

using the correct index for dstport. The program also uses the *PROTOCOL_MAP* defined at the top of the code to translate the common protocol numbers. We handle unknown protocols as well by just leaving the number as is.

write_tag_counts()

I decided to implement special logic to move the Untagged entries to the bottom of the output. This makes more meaningful matches appear first, making the output more readable. It was also alphabetized for consistency.

write_port_protocol_counts()

This function was essentially the same as the previous one, simply outputting a sorted dictionary into the CSV. Like the previous function, output was alphabetized.

Bugs and Edge Cases

Several issues were encountered during development, some of which were caused due to some quirks of CSVs. One major issue arose when a UTF-8 BOM in the CSV file caused an error for the tag field, despite the file appearing fine. I solved this by using the utf-8-sig encoding and sanitizing the headers. Another issue involved trailing spaces in the headers, which was fixed by stripping whitespace from the headers and row keys.

Several edge cases were accounted for. These ranged from duplicate log entries, malformed log entries, trailing or leading whitespaces, and case insensitivity. These are mostly handled with the code, and are additionally showcased as being handled through the test cases provided. Moreover, the program ensured that tags with no matches were still excluded correctly.

Final Design

The final design of the program is modular, readable, and robust. It handles several common pitfalls when parsing CSVs. The outputs also follow the structure as outlined in the problem statement. Multiple test cases were created and validated to ensure that the program runs correctly and that edge cases were handled.