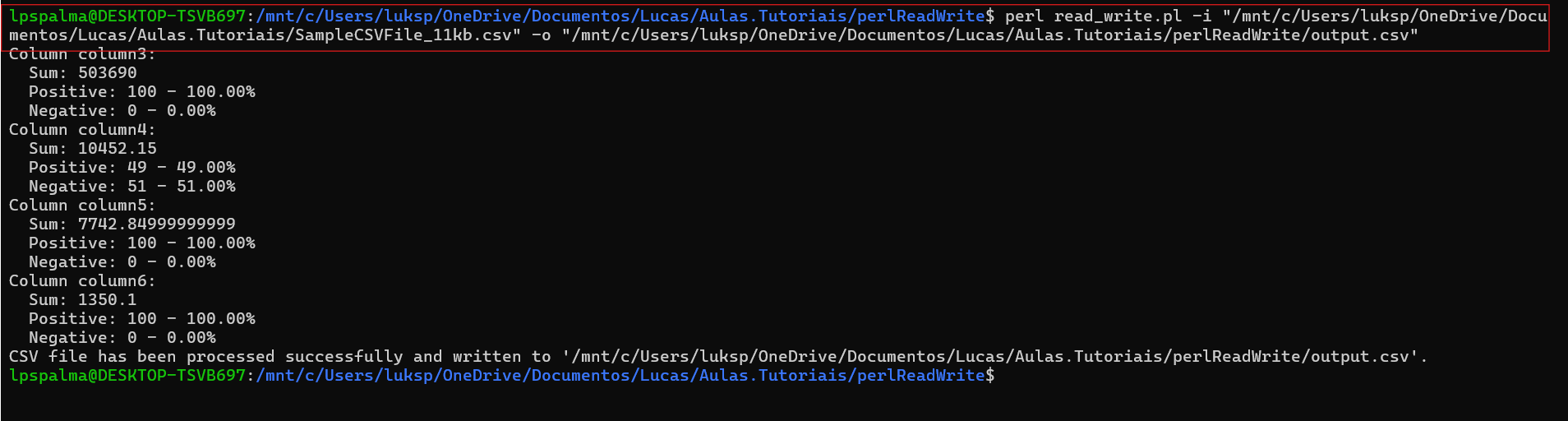
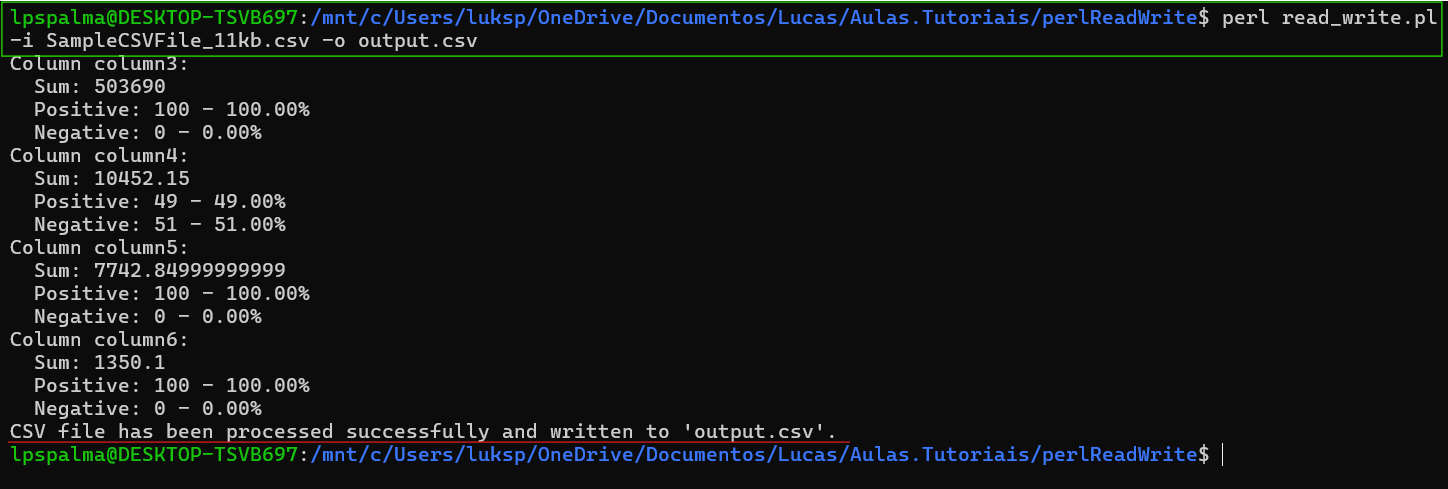
**Evidences**

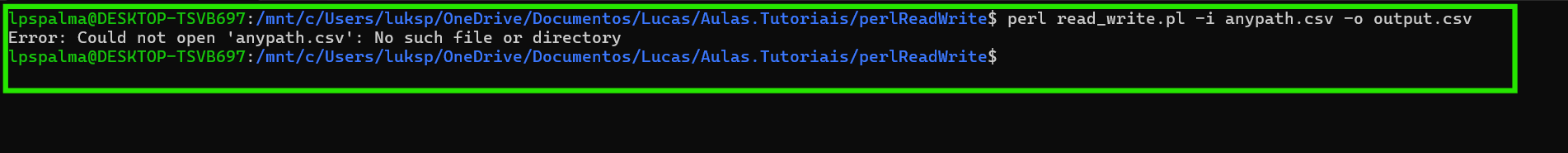
Task completed using Perl 5, some requirements were not clear so I decided to add more functionalities to make sure that I achieve more than the minimum requirements requested.

1. Input and output file paths and names should be passed as script parameters rather then hard coded in the code.

Example: <script\_name>.pl -i <Orignal CSV file> -o <Output CSV file>

To achieve this requirement I am using Getopt::Long to handle script parameters (-i for input file and -o for output file), which allows flexible input/output file paths.

* Running command using path to input file different from where the script is and writing the output file in another folder and receiving successful result.
* Running command using path to input file in the same folder of the script and writing the output file in the same folder, receiving successful result.
* Running command using a path that doesn’t have a valid file and showing how to handle the error.



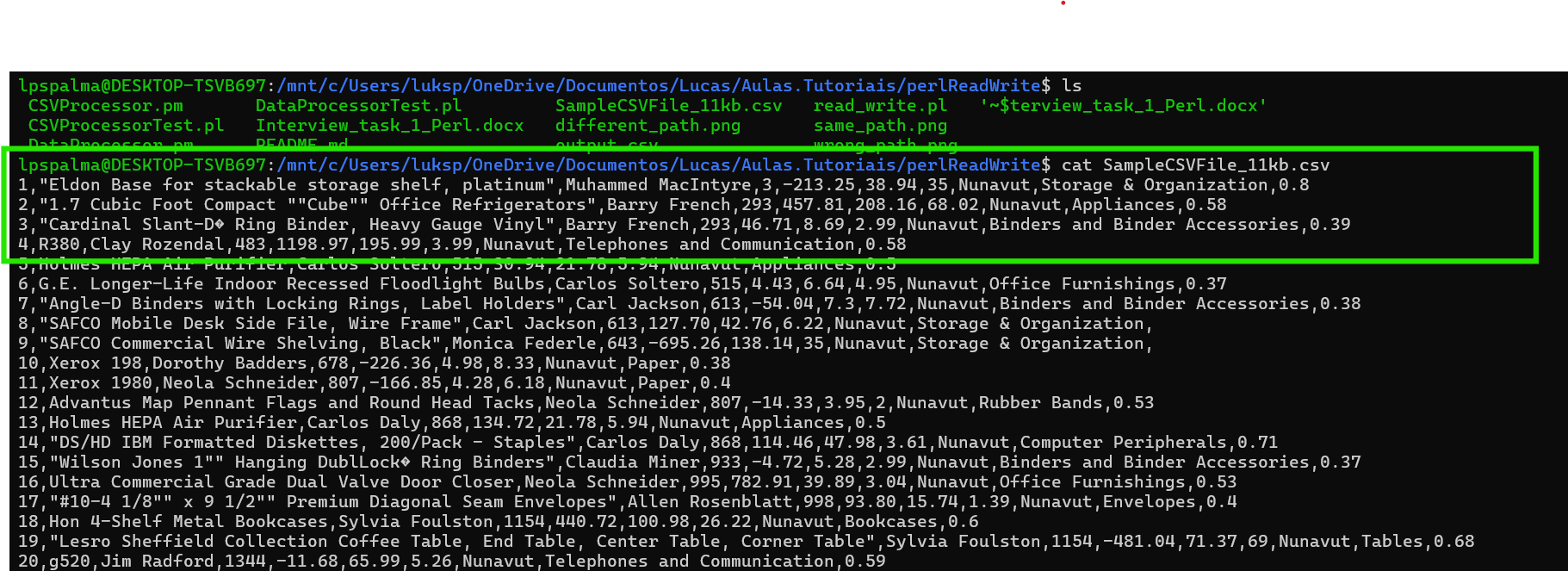
1. The output CSV file must contain the column headers which are missing from the original file, as well as the original data.

To achieve this requirement, I ensure the output file contains headers (column0, column1, column2) which are added if missing from the original file.

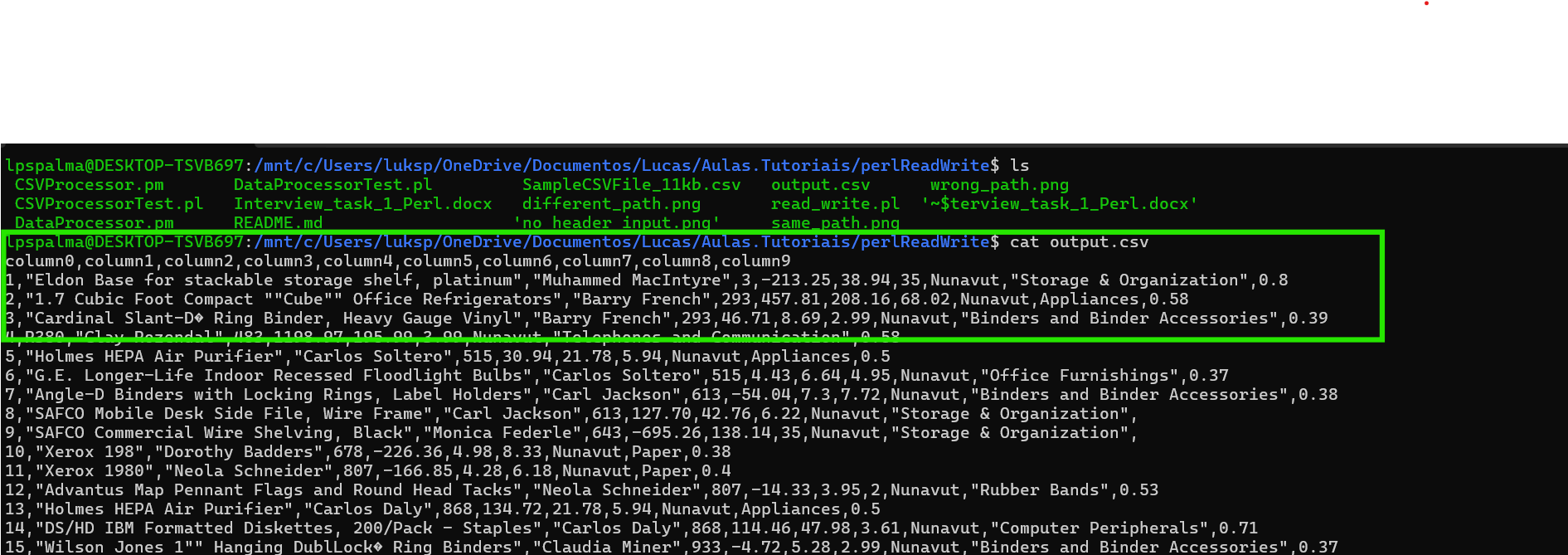
Since the headers name was not provided, I decided to use generic column names

my @header = map { "column**$\_**" } (0 .. $max\_columns - 1);

Input file with no Header

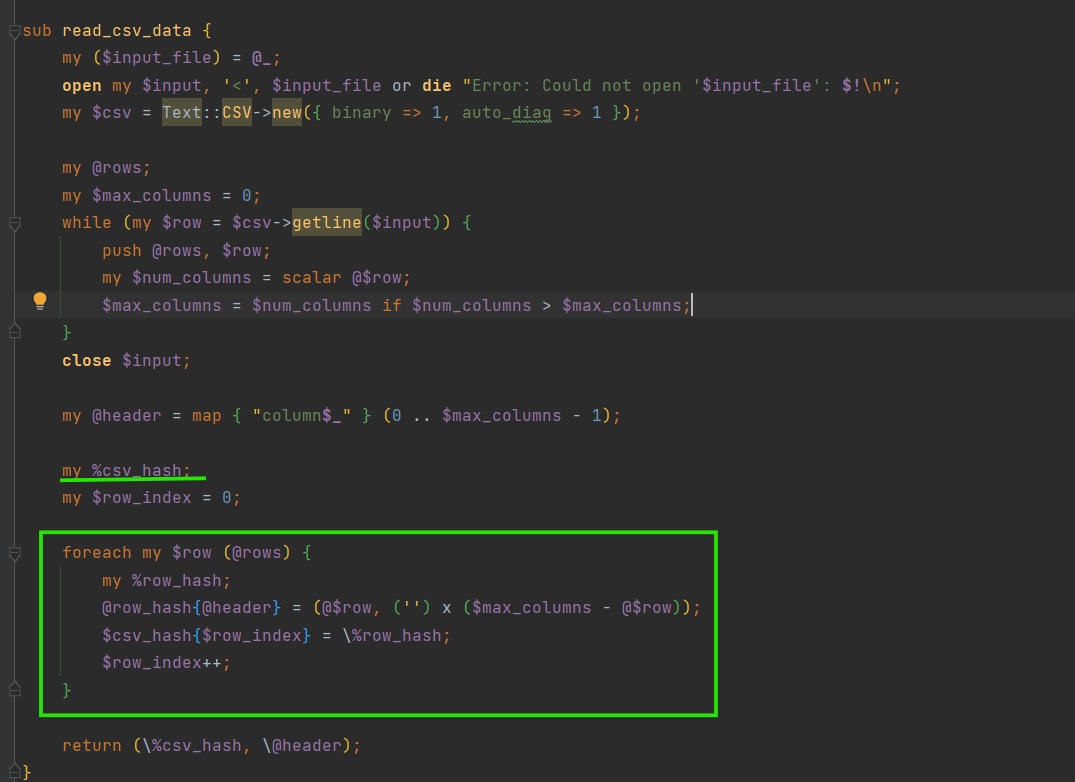


Output file with new header added



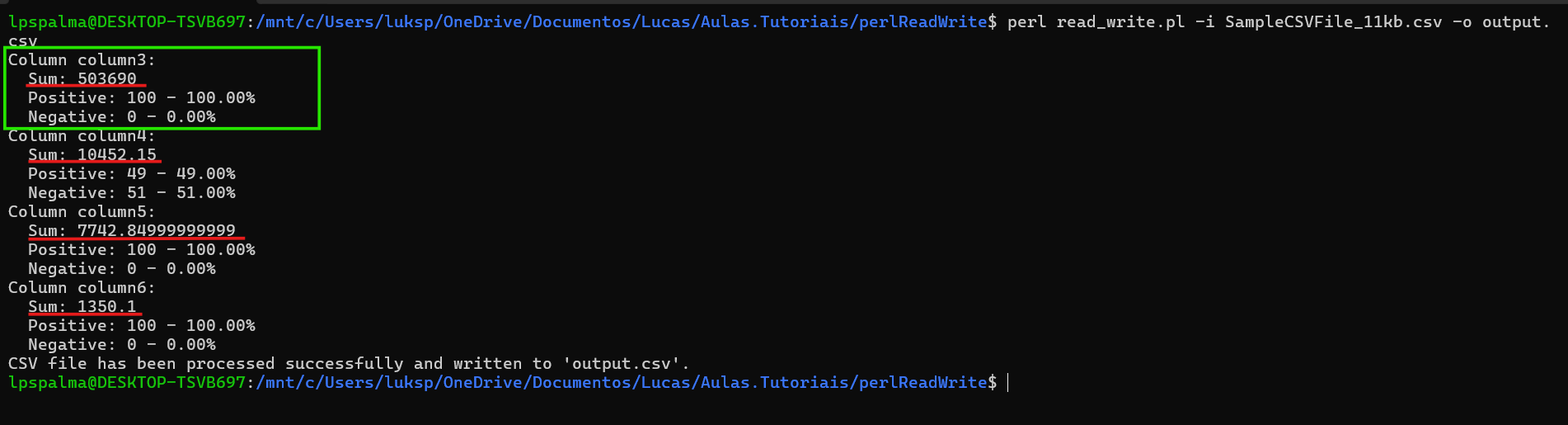
1. The data in the original CSV must be parsed and stored inside a hash.

The read\_csv\_data subroutine reads the CSV data into a hash (%csv\_hash), storing the parsed data.



1. The sum of the column 3 values must be printed out.

Although the script calculates sums, counts, and percentages for specified columns, it doesn't specifically print the sum of column 3 values. But the requirement was achieved since the script print the sum of the column 3 together with the other columns.

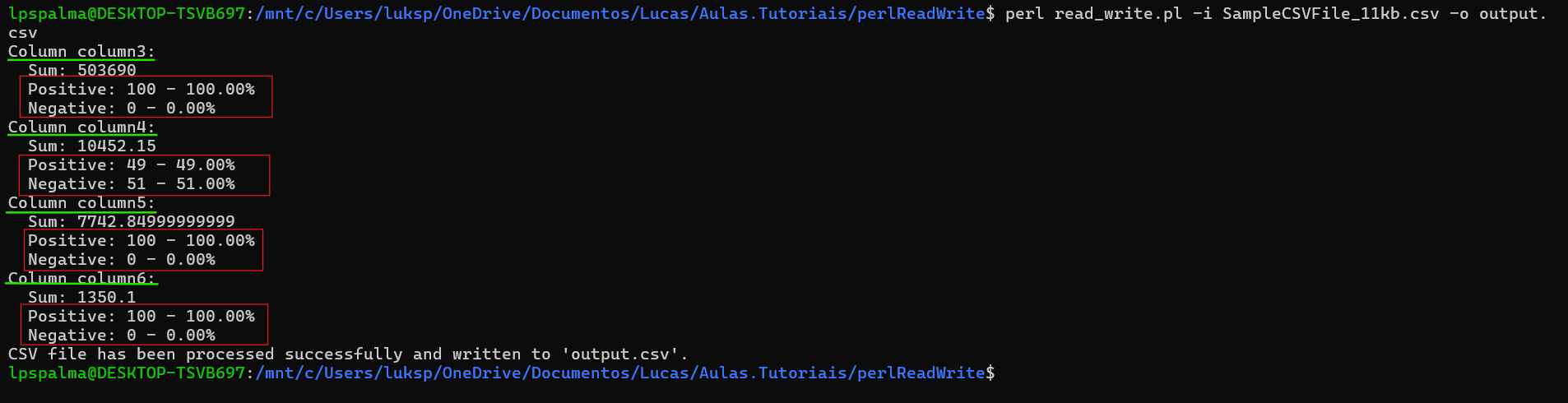


In the result we are able to find the sum of the columns 3 to 6, columns that have numerical values, when is empty, the script read the field as 0.

1. The count of negative and positive column values should also display as well as the percentage of the total number of values.

Example: Negative: 10 - 50%, Positive: 10 - 50%

The calculate\_data subroutine calculates counts and percentages for positive and negative values in specified columns, which meets this requirement.



**Error Handling and Clear Messages:** I've implemented error handling for file operations (die statements with clear error messages) and provided appropriate error messages. The error messages indicate file open failures (Could not open '<filename>' for writing: <error>).

**Documentation and Structure:** I added comments and structured subroutines (main, read\_csv\_data, calculate\_data, write\_output, etc.), which enhances readability and maintainability. Also, the Readme doc was added in the project detailing the execution, function details and test strategy.

**Test Plan:** Unit tests (Test::More) that cover positive, negative, and edge scenarios for both most part of the functions. These tests validate functionality and handle error cases effectively.