1. Correctness

What makes a CSV parser “correct”? We're not asking for additional input-output pairs here, but fairly precise, natural-language descriptions. Put another way, what kinds of general *properties* should your tests be checking about your CSV parser?

General properties that my parser should be checking about my CSV are the datatypes and structure of the CSV. Instead of blindly transcribing each line of the csv onto a data structure, my parser should also be checking for consistency in the types of data in the CSV - meaning that it would throw an error if a numerical column suddenly had a string appear. Also, it should check for structural abnormalities in the CSV, such as if the CSV is empty or if there are blank lines. Then, the parser shouldn’t continue and should instead throw an error.

2. Random, On-Demand Generation

Suppose we gave you a function that randomly produced CSV data on demand. You could then call this class from your testing code. How might you use this source of random data to expand the power of your testing?

If random CSVs were generated, then I would have a wider variety of datatypes and structures to pass into my tests, so I could make sure that my parser was functional for ANY kind of CSV, including those with, perhaps ten columns and a different datatype for each column. In short, this random generation would enable me to test my parser on a more diverse collection of CSVs, which is important for making sure that my parser works in any scenario.

3. Overall experience, Bugs encountered and resolved

In what ways did this sprint differ from prior programming assignments you’ve done? Did anything surprise you? Did you encounter any bugs during your work on this sprint? If yes, what were they and how did you fix them? If not, how do you think that you managed to avoid them?

At first, this sprint felt like tight-rope walking without a net because there was very little of how we should do something and more of what we should accomplish. However, I realized that the copilot component was my net, and I was able to use it to fill the gaps in my comprehension of the assignment. I was also able to use copilot when I ran into a bug where my test for my schema kept failing because I was passing an object to a schema that expected a tuple. By ascertaining from copilot that a schema took in arrays, not objects, I was able to fix my code to pass in an array. Also, when I tried to test the default behavior of the parser - creating a string[][] when no schema was passed in- I kept getting an error saying that two arguments were

expected but only one was provided. From copilot I learned syntactically that I needed a ? in the method header to make the schema optional, which exemplifies my use of copilot in general to help me with syntax in typescript, a language quite novel to me.