Date: 10/09/2023

Homework 2 Report

Priyanshu A. Shrivastava
NetID: ps1276

Abstract:

In this homework, we are tasked to read a csv file and perform mathematical operations like the ones that we do in an excel file. We are supposed to test our program on various testcases and edge cases using jasmine testing framework.

Introduction:

Moving forward from homework 1 we will now read csv file and then implement mathematical function like SUM(), AVG() and SD() etc., and then test the program with jasmine testing framework. I have started by executing the first homework program initializing the node.js framework with “npm init -y”. I then installed jasmine in that directory with “npm install --save-dev jasmine” and then initialized jasmine with: “npx jasmine init”. This sets up the environment for programming and testing the required application.

Working:

We first start by taking csv file as input and converting parsing it as variable which is then processed to convert it in a matrix format, and we are making sure that row and columns aren't uneven:

```
//Convert csv to 2D array
function matrix(data) {
    //split with \n
    let csvData = data.split('\n');
    //remove empty row
    csvData = csvData.filter(row => row);
    //each row is split by ',' and trim leading or tailing whitespace
    csvData = csvData.map(row => row.split(',').map(cell => cell.trim()));

    //throws an error if num of columns is not same in each row
    if (!csvData.every(row => row.length === csvData[0].length)) {
        throw new Error('Incorrect CSV format');
    }

    return csvData;
}
```

Then we add the individual cell value inside the map so that it becomes easy to access cell value according to the function.

```
let dataMap = new Map();

function setCell(key, value) {
    dataMap.set(key.toUpperCase(), value);
}

function getCell(key) {
    return dataMap.get(key.toUpperCase());
}
```

```
//extract value from matrix and feed to the global dict "dataMap"
function createMap(filePath) {
    const content = fs.readFileSync(filePath, 'utf8');
    const rows = matrix(content);

    for (let rowIndex = 0; rowIndex < rows.length; rowIndex++) {
        for (let colIndex = 0; colIndex < rows[rowIndex].length; colIndex++) {
            const key = `${String.fromCharCode(65 + colIndex)}${rowIndex + 1}`;
            setCell(key, parseFloat(rows[rowIndex][colIndex]) || rows[rowIndex][colIndex]);
        }
    }
}
```

We then take the equation formula from user:

```
function getEquation() {
    const equation = readline.createInterface({
        input: process.stdin,
        output: process.stdout
    });
    equation.question('Enter the equation: ', (equationInput) => {  // renaming the parameter to avoid conflict
        try {
            const result = executeEquation(equationInput);
            console.log(`Result: ${result}`);
        } catch (error) {
            console.error(error);
        }
        equation.close();
    });
}
```

We then evaluate the formula by breaking it down and then accessing the map to extract the user required range of values for the specified mathematical function:

We perform the calculation based on the operation specified

```
    // Return calculated value based on the operation
    let avg = total / count;

    // If the calculation is standard deviation
    if (calc === "SD") {
        let squaredDifferencesSum = 0;

        // If it's a column range like A1:A3
        if (colstart === colend) {
            for (let row = rowstart; row <= rowend; row++) {
                const cellValue = parseFloat(getCell(`${colstart}${row}`)) || 0;
                squaredDifferencesSum += Math.pow(cellValue - avg, 2);
            }
        }
        // If it's a row range like A1:C1
        else if (rowstart === rowend) {
            for (let col = colstart.charCodeAt(0); col <= colend.charCodeAt(0); col++) {
                const cellValue = parseFloat(getCell(`${String.fromCharCode(col)}${rowstart}`)) || 0;
                squaredDifferencesSum += Math.pow(cellValue - avg, 2);
            }
        }

        return Math.sqrt(squaredDifferencesSum / count);
    }
    // For SUM operation
    else if (calc === "SUM") {
        return total;
    }
    // For AVG operation
    else if (calc === "AVG") {
        return avg;
    }
    else {
        throw new Error('Unsupported operation');
    }
}
```

Result:

We will check if the programming running properly by first running the main program “math.js” and then test it using by adding testcases to the “math_test.spec.js” program in the spec folder.
For test1.csv we get the following result:

```
(base) ryuk@Priyanshus-Air HW2 % node math.js
Enter the equation: =SUM(A1:A3)
Result: 15
(base) ryuk@Priyanshus-Air HW2 % node math.js
Enter the equation: =AVG(A1:C1)
Result: 2
(base) ryuk@Priyanshus-Air HW2 % node math.js
Enter the equation: =SD(B1:B3)
Result: 3.265986323710904
(base) ryuk@Priyanshus-Air HW2 %
```

In our testing environment we check:
-if the values in the csv are numeric
-if number of columns in each row are same
-test all mathematical operation with a test.csv file

For test1.csv

```
(base) ryuk@Priyanshus-Air HW2 % npm test

> hw2@1.0.0 test
> /Users/ryuk/Documents/Software/HW2/node_modules/jasmine/bin/jasmine.js

Enter the equation: Randomized with seed 31493
Started
.....


5 specs, 0 failures
Finished in 0.003 seconds
Randomized with seed 31493 (jasmine --random=true --seed=31493)
```

For test3.csv

```
(base) ryuk@Priyanshus-Air HW2 % npm test

> hw2@1.0.0 test
> /Users/ryuk/Documents/Software/HW2/node_modules/jasmine/bin/jasmine.js

Enter the equation: Randomized with seed 59248
Started
F.

Failures:
1) CSV Functions should contain only numeric values
  Message:
    Expected false to be true.
  Stack:
        at <Jasmine>
        at /Users/ryuk/Documents/Software/HW2/spec/math_test.spec.js:16:38
        at Array.forEach (<anonymous>)
        at /Users/ryuk/Documents/Software/HW2/spec/math_test.spec.js:15:17
        at Array.forEach (<anonymous>)
  Message:
    Expected false to be true.
  Stack:
        at <Jasmine>
        at /Users/ryuk/Documents/Software/HW2/spec/math_test.spec.js:16:38
        at Array.forEach (<anonymous>)
        at /Users/ryuk/Documents/Software/HW2/spec/math_test.spec.js:15:17
        at Array.forEach (<anonymous>)
  Message:
    Expected false to be true.
  Stack:
        at <Jasmine>
        at /Users/ryuk/Documents/Software/HW2/spec/math_test.spec.js:16:38
        at Array.forEach (<anonymous>)
        at /Users/ryuk/Documents/Software/HW2/spec/math_test.spec.js:15:17
        at Array.forEach (<anonymous>)

2 specs, 1 failure
Finished in 0.004 seconds
Randomized with seed 59248 (jasmine --random=true --seed=59248)
```