CIS2201-1819

University of Huddersfield

Cyber security assignment report

Adam Birch

Table of Contents

[Important notes: 1](#_Toc7608309)

[System Specification - 1](#_Toc7608310)

[Requirements - 1](#_Toc7608311)

[Interface and User Interactions 1](#_Toc7608312)

[Software Functions 2](#_Toc7608313)

[System Design – 3](#_Toc7608314)

[Main.cs Class Diagram 3](#_Toc7608315)

[Frequency functionality Activity Diagrams 4](#_Toc7608316)

[Selecting a CSV Sequence Diagram 5](#_Toc7608317)

[Wireframe Interface Designs 6](#_Toc7608318)

[Implementation notes – 6](#_Toc7608319)

[Testing – 11](#_Toc7608320)

[Evaluation - 13](#_Toc7608321)

# Important notes:

* The file selected for testing MUST have the extension .csv, else it won’t appear in the OpenFileDialog options.
* The program can be closed using the [X] in the corner at any time (Providing that the program isn’t currently processing a request) safely, as it releases all resources before terminating.
* The program DOES NOT alter the files it is provided with, any file given can be read only and should never be at risk of corruption or harm.
* Some of the code shown is during development. Some code may have different layouts or additional comments to that documented here.

# System Specification -

I chose to create a desktop application to meet the criteria for this assignment. I will use C# within Microsoft Visual Studio 2017 for the implementation of the solution.

I have no experience with C# and very little experience with Visual Studio – however I have secured a placement and this is the environment I will be working in, so I thought it would be good practice for the placement.

## Requirements -

The application will have many requirements to be classed as a success:

### Interface and User Interactions

* GUI Interface –
  + Visual Studio allows for easy creation of Windows Form Applications natively. These can be used to display various forms and layouts dependant on the current context.
* Allow the user to select a CSV file –
  + C# has an OpenFileDialog object that can browse your system through the standard file explorer, and can be given constraints about the files that can be selected.
  + As I will allocate a CSV constraint, any file selected MUST have the .csv extension or it will not appear within the list of available files of the current directory.
* A CSV should be displayed in a table –
  + The data will be read into a DataTable for processing and manipulation and displayed using a DataGridView to display the DataTable within the Windows Form.
* The user can select a column to process –
  + The user will be able to select a cell within the DataGridView, which then processes the selected cell to get this column, and then programmatically selects all the other cells within that column.
* The user can sort the data within a column –
  + The selected column is used as an index. A dedicated function will use that index and the way the action was triggered to sort that column either Ascending or Descending.
  + The action will likely be controlled via a pair of buttons – Ascending and Descending.
* The data can be collected to show the frequency of a piece of data –
  + The user will be able to press a button, which isolates the selected row, and display: all unique data items, the row number of the first appearance of that item, the frequency (total number) of occurrences of that item, and a percentage of that item compared to all items.

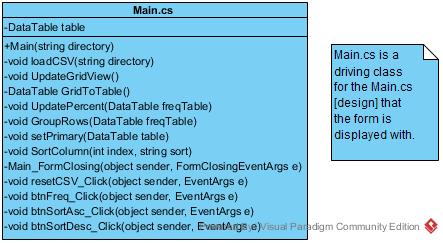
### Software Functions

* The software can sort the data within a column –
  + I will make use of the DataGridView inbuilt Sort function as this is well tested and supported within the community and has been well optimised.
  + This well optimised method helps to reduce the run time – especially with larger collections of data.
* The software can produce a table of frequencies for a given column –
  + The software can take the index of a selected column –
    - The DataGridView has a CurrentCell method that can be used to get metadata for the selected cell.
  + The software can retrieve the data from the selected column –
    - A new DataTable will be created, and a new row created for each row in the original persistent DataTable.
    - Each value is retrieved in turn and stored in the new table.
    - This will be done for both the data and row numbers.
  + Duplicate values can be removed and counted –
    - Each row of the data column will be compared to the rows above it.
    - If a duplicate is found the row will be removed and the frequency of the matching row will increment.
    - If no match is found, the current row is also unique and will be compared to the rows below it.
  + The percentage of an items occurrence of the total items can be calculated –
    - The frequency and total number of rows in the original tables will be used to perform a basic percentage calculation.
* The software can analyse the data to find potential threats –
  + Use of some kind of analysis tool such as Jenks’ analysis or other SIEM standards.

# System Design –

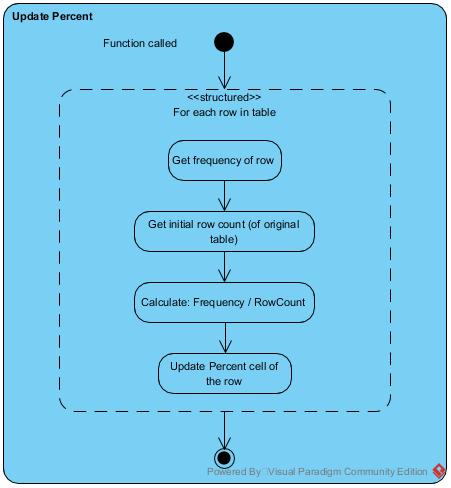
I have compiled a selection of UML diagrams to describe and detail many critical sections of the application:

## Main.cs Class Diagram



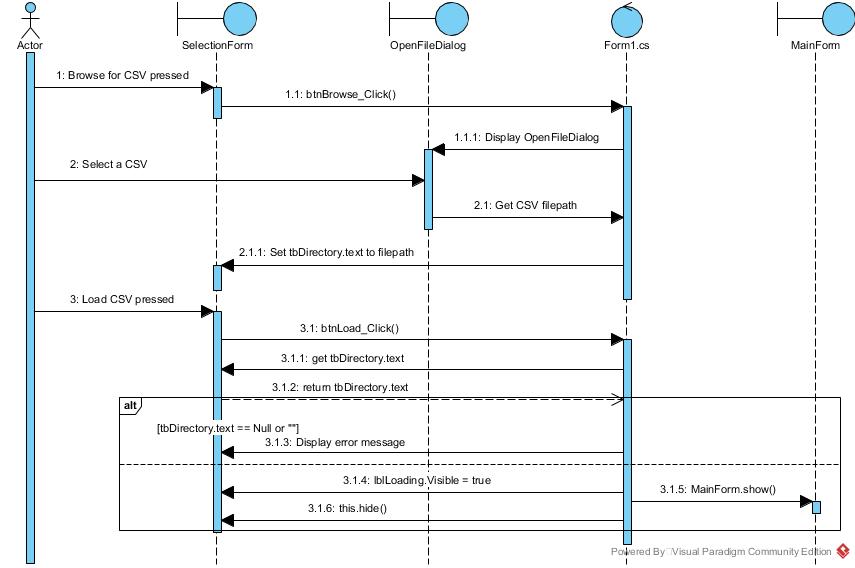
This diagram outlines the functions defined within the Main.cs file. The functions below Main\_FormClosing are all listeners that perform an action when the user interacts with the interface, be it through usage or from clicking on the [X] to close the application.

## Frequency functionality Activity Diagrams



These activity diagrams all contain the functionality to allow the user to check the frequency of rows based on the data within the selected column, and the percentage of all rows that each unique row is. This allows the user to quickly see and sort the rows to see which are the most common and uncommon values found within that row of the CSV provided.

## Selecting a CSV Sequence Diagram



This sequence diagram details the way the user interacts with the initial form to select a CSV file from a directory on the machine. There are parameters set for the OpenFileDialog such that only CSV files can be loaded, and other basic error checking that there is a directory selected before trying to read the file.

## Wireframe Interface Designs

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

These basic wireframe designs outline the initial concept for interfaces. The first is the initial form that allows the user to select a CSV from a directory, while the second is the form that displays the data in a grid with possible functions displayed in a ribbon style interface above. The buttons change when different data is being displayed, outlining the possible actions for the given data.

# Implementation notes –

|  |  |
| --- | --- |
| Issue | Solution |
| Trying to alter the data in the frequency column of a row would fail as the column was read only. | I needed to change the way I defined a column. Rather than creating a column within the method of adding a column, I needed to create a defined column first with several attributes and add it later. This was because the column was acting as an Expression column – where by the contents are function driven and not directly accessible to the program. See Figure 1 for an example. |
| The Percent calculation would always display 0% for any given input values. | Originally I was directly reading the values from the cells, casting them to Integers, and using maths on these values. However, using Integers for these values caused some unknown issue (The result was being stored as a double so shouldn’t have caused issues). I then defined the Frequency and Count[[1]](#footnote-1) values separately as Decimals, and defined perc (percent) before writing that perc value into the cell. See figures 2 and 3 to see a comparison. |
| The user was unable to select a column using the initial idea of getting a selected column. | By default, a DataGridView has several properties enabled and disabled based on the most common uses. This means that Sorting is set to Automatic and Cell Selection is set to Individual. I needed to programmatically change the properties so that the Sort is set to NotSortable and the selection is set to FullColumnSelect. As this is a long task undone and redone every time the DataGridView needs to be updated, I created the function seen in figure 4. |
| The change above broke the sort as the user is unable to use the inbuilt sort of the DataGridView. | I created a new function “SortColumn” which requires an index and sorting order as parameters. It goes through similar steps as the solution above to set the sort to Automatic, utilise the sort, and change it back to the NotSortable setting. This can be seen in figure 5. |
| Using columns with existing names doesn’t work. | Figure 6 shows the original implementation. The provided column headers are not used as each used column header is a character and the delaminating comma. I changed the way a header is found to use the ReadLine function of the StreamReader to that seen in figure 7. Figure 8 then shows the new result. |
| All cells are removed when only non-matching rows should be. | The implementation should remove all rows where a column value doesn’t match the cell selected. I had a table.Clear() function after updating the DataGridView to try and free up some memory (in testing I hit over 5GB of ram usage) but this was also clearing the DataGridView as the table stores the data even after the data is displayed. |

A close up of a screen

Description automatically generated

Figure

Figure 1 shows the definition of the DataTable. The “Row” and “selectedColumn[[2]](#footnote-2)” columns are defined within the Add method of the DataTable.Columns object. “Frequency” and “Percent” are both defined as DataColumns with their types and ReadOnly values explicitly defined before the column is added to the DataTable.

A picture containing indoor

Description automatically generated

Figure

Figure 2 shows the original implementation of the percentage calculation. The theory is correct and using constant decimal numbers would work (E.G. 12.3) but would never work in the current state as values are integers.

A picture containing indoor

Description automatically generated

Figure

Figure 3 shows the final implementation of the percentage calculation. It has the exact same theory and calculation as Figure 2 but performs extra steps in defining each variable individually before being used in the calculation. This would ensure that all data types are correct before attempting the calculation.

A screen shot of a social media post

Description automatically generated

Figure

Figure 4 shows the update function that is used to edit the contents of the DataGridView to show the data provided within a DataSource parameter. The source can only be bound if the DataGridView is in the default state (Automatic sort and Cell selection) but the settings for the functionality required needs the custom state (NotSortable and FullColumnSelect). The function toggles between the two, setting the new DataSource and refreshing the data being displayed.

A screenshot of a cell phone

Description automatically generated

Figure

Figure 5 shows the function to sort the DataGridView based on the currently selected column. The two functions - btnSortDesc\_Click and btnSortAsc\_Click are action listeners for the two buttons that allow the sort functions to be called. They both access the SortColumn function to sort. They pass the selected column as a parameter along with the string value of the sort to be used.

A screenshot of a computer

Description automatically generated

Figure

Figure 6 shows the provided headers being used as rows of data.

A screenshot of a cell phone

Description automatically generated

Figure

Figure 7 shows the functions used to use the given headers from the file. This throws an exception if duplicate names are used for the columns – default values will then be assigned to each column.

A screenshot of a cell phone

Description automatically generated

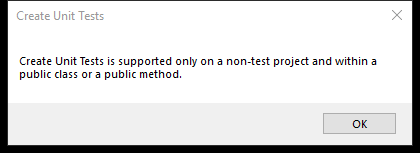
Figure

Figure 8 shows the given headers from Figure 6 with the new modifications to the code to allow the headers to work as expected.

# Testing –

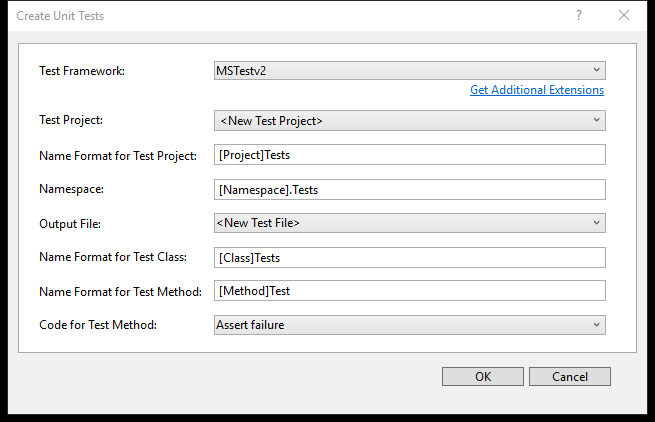
## Unit Testing

Unit testing is not supported for Microsoft Visual Studio projects that are implemented in the way that I have. When I try to create a unit test I get the following error:



As my methods are private this is not possible without changing to public.

If I change the functions to Public I get the following dialogue:



I do not fully understand this or how unit testing syntax is done within C#. Given more time I would look into this however this will unfortunately not be included.

## Functionality Testing

All screenshot evidence is drawn in a table below this table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | What is being tested | Expected outcome | Actual outcome | Pass / Fail | Evidence |
| 1 | Can the user open Form1? | The program will start and the form will appear. | The form opens successfully | Pass | Figure 9 |
| 2 | Can the user browse for a file? | The Browse button is pressed and the file explorer will open. | The file explorer opens as expected, only showing directories and csv files. | Pass | Figure 10 |
| 3 | Is a selected file’s directory added to the text box? | Selecting the file will write the file’s entire file path to the textbox. | The file path appears as expected. | Pass | Figure 11 |
| 4 | Can the user select no file to load? | A message will appear asking for a file. | The program crashed with a null value exception | FAIL | Figure 12 |
| 5 | Can the program load a file into a DataGridView? | A file will be selected, allocated column names and displayed. | After a short pause (~15 seconds) the data is loaded into a DataGridView on a new form. | Pass | Figure 13 |
| 6 | Can the user select a column? | A cell will be pressed and the whole column will be selected. | Clicking a cell or header selects the entire column. | Pass | Figure 14 |
| 7a | Can columns be sorted (Ascending)? | A column will be selected and sorted  L -> H. | The column is sorted in reverse alphabetical order. | Pass | Figure 15 |
| 7b | Can columns be sorted (Descending)? | A column will be selected and sorted  H -> L | The column is sorted in alphabetical order. | Pass | Figure 16 |
| 8 | Can a selected column be grouped on frequency? | A column will be selected and grouped by frequency. That column and row number will be displayed along with the frequency and percentage. | Column 4 was selected, each used cell data are shown along with the row number of the first encounter. | Pass | Figure 17 |
| 9a | Can data in the new table be sorted (Ascending)? | A column will be selected and sorted  L -> H. | The column is sorted in reverse alphabetical order. | Pass | Figure 18 |
| 9b | Can data in the new table be sorted (Descending)? | A column will be selected and sorted  H -> L | The column is sorted in alphabetical order. | Pass | Figure 19 |
| 10 | Can the original CSV be reloaded into view? | The Reset Data button will be pressed, and the original data will be displayed. | The data returns to its default state. | Pass | Figure 20 |
| 11a | Can the user change the selection mode (Cell) | While a column is selected, press the button to change type and a single cell can be selected. | A cell can be selected once the button is pressed. | Pass | Figure 21 |
| 11b | Can the user change the selection mode (Column) | While a cell is selected, press the button to change type and a whole column can be selected. | A column can be selected once the button is pressed. | Pass | Figure 22 |
| 12 | Can the user filter rows based on cell data? | A cell will be selected, and only rows with matching data will be displayed in the DataGridView. | Only tcp rows were displayed. | Pass | Figure 23 |
|  | Can the program close without errors? | The [X] will be pressed and the window will close after displaying a confirmation. | The message is displayed and no errors are thrown. | Pass | Figure |

|  |
| --- |
| Evidence |
| Select a CSV File  Figure 9 |
| Open  Figure 10 |
| A screenshot of a cell phone  Description automatically generated  Figure 11 |
| A screenshot of a computer screen  Description automatically generated  Figure 12 |
| Main  Figure 13 |
| Main  Figure 14 |
| Main  Figure 15 |
| Main  Figure 16 |
| Main  Figure 17 |
| Main  Figure 18 |
| Main  Figure 19 |
| Main  Figure 20 |
| Main  Figure |
| Main  Figure |
| Main  Figure |
| Close Application  Figure 24 |

## Fixing failed tests

|  |  |  |
| --- | --- | --- |
| Test Number | Fix | Evidence |
| 4 | Add a statement to catch if the field is empty. I used System.IO.File.Exists(tbDirectory.Text) to check that the data entered is that of a valid file (not directory) and a message is displayed if not. | A screenshot of a social media post  Description automatically generated  Figure |

# Evaluation –

I believe I have created a basic processing application that serves a basic outline of the task provided. The user is able to perform some basic processing to the CSV file they need and assess the frequency of data items in a column. Given more time I would include more complex functions to the application to create a more niche analysis tool.

I have learned much about C# doing this, and I think it has become my new favourite language as it is full of tools while staying efficient and well documented.

One of the greatest successes of this project (in my opinion) is the way I implemented the frequency calculation. I get the second row of data first, and compare it to the one above it. This reduces the time taken as all rows above the current one being checked are unique. If the current cell matches one above then the frequency of the one above will increment and the current row is removed, else the current row is also unique so go down a row. Like with most processes in this application – the smaller the dataset the faster it processes – However with this function, a larger dataset may be faster than a smaller one if there are fewer unique values (as there are less to compare to).

In hindsight, one way to improve this further would be to sort the column first so all the rows are collected together. Then count how many rows have the same data and remove all but one, changing the frequency of that last one to the number of rows there were. This would allow a single loop to be used, and a thread to be created for each unique set of data – allowing for a multi-threaded function to be created.

Another success was how I implemented a way to store the original data while displaying new data. By this I mean that the original CSV data is retained while a new table is displayed. The original data is stored in the DataTable “table” and persists throughout the runtime of the program. As the DataGridView displays the data provided in the DataSource parameter, I created an Update method which takes a DataTable as an argument. This provided table is then used as the table to be displayed even if the table is then destroyed later on in the runtime. This allows the program to quickly revert to the original data. This is also done as the DataGridView has several properties that need to be changed in order to allow data to be displayed and interacted with. Data cannot be assigned if Sorted is not set to Automatic, but FullColumnSelect cannot be set if Sorted is set to Automatic – meaning the user cannot select a column of data. This method sets the correct properties to set the data, then sets the properties to allow the user to interact as expected.

One of the greatest shortcomings of the application is the lack of in-depth analysis functions. This was mostly due to time constraints as mentioned before. With more time I would have used Jenks’ Analysis to help identify which elements should be investigated further. I could also include some UI customisations and themes to make the program nicer to use, and made it multi-threaded to process larger sets of data faster. I looked into this and it doesn’t seem too difficult to do – but requires a complete rework of the structure I created to read the CSV into the table.

I chose to implement a feature that allows the user to filter all rows to only show those where a column value matches the one selected. This can be useful if you want to compare different values used when the same operation is done. For example, if 200 http requests are logged with a 0 as a value and 2 have a 21, then the 21 values are strange and likely need to be looked into more. As the table is changed, the filtered data can also be grouped by frequency to analyse the percentage of data hits for a column after the filter has taken place.

In all, I believe I have made a good attempt at making an application to aid in the manual analysis of data, however I also think I could do much better if I had more time to implement a more niche solution.

1. Count is the total number of rows present in the original table, read from the CSV provided. [↑](#footnote-ref-1)
2. The name of the selectedColumn column changes based on the column selected by the user and is the same name as that in the original CSV file (or the ones generated if none are provided). [↑](#footnote-ref-2)