CS-C2120 Programming Studio A Project Report

1. Personal Information

- Student's name: Quan Tran
- Student number: 100644052
- Degree Program: Bachelor's degree in Data Science
- Year of Studies: 2022-2025
- Date: 04/03/2024

2. General Description

This report outlines a proposed application—a visual data representation application—being developed as an interactive dashboard aimed at assisting the user in representing data and deriving insights from it. In the app "DataViz Pro," salient points of data would be highlighted using the "cards," which included important numbers and different modes of data visualizations like scatter plots, column charts, pie charts, and line charts. The app supports data input from a variety of sources: CSV files, URLs, and direct keyboard input, which means the user can easily be versatile in data input and presentation.

One of the major aims of this project was to address the challenge that is presented by data visualization through a friendly interface, which does not compromise the depth at which analysis could be carried out by the users. Packed with tools to enable aesthetic customization, "DataViz Pro" aims at friendly data analysis to users while maintaining an aesthetic approach.

This project is deemed demanding given the scope of the program.

3. User Interface

Launching the Application

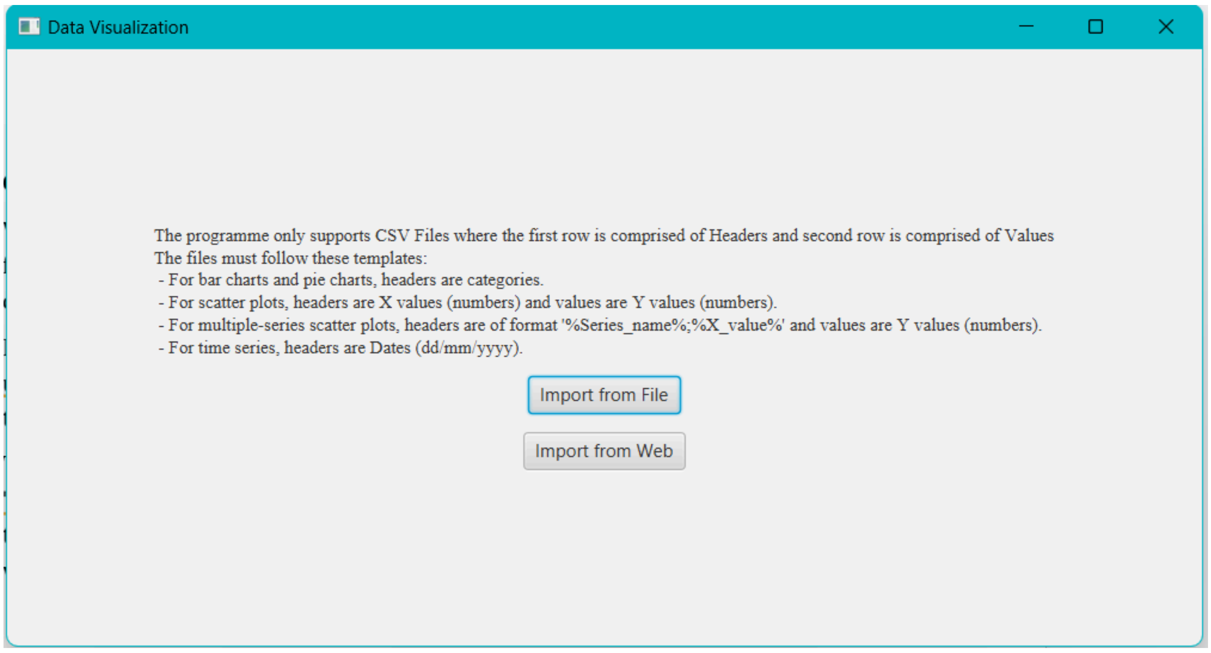
"DataViz Pro" opens with a simple decision window—either create a new dashboard or open an existing one. This is to give a smooth workflow to the user, in which he is guided into the data visualization in a continuous way.

Creating a New Dashboard

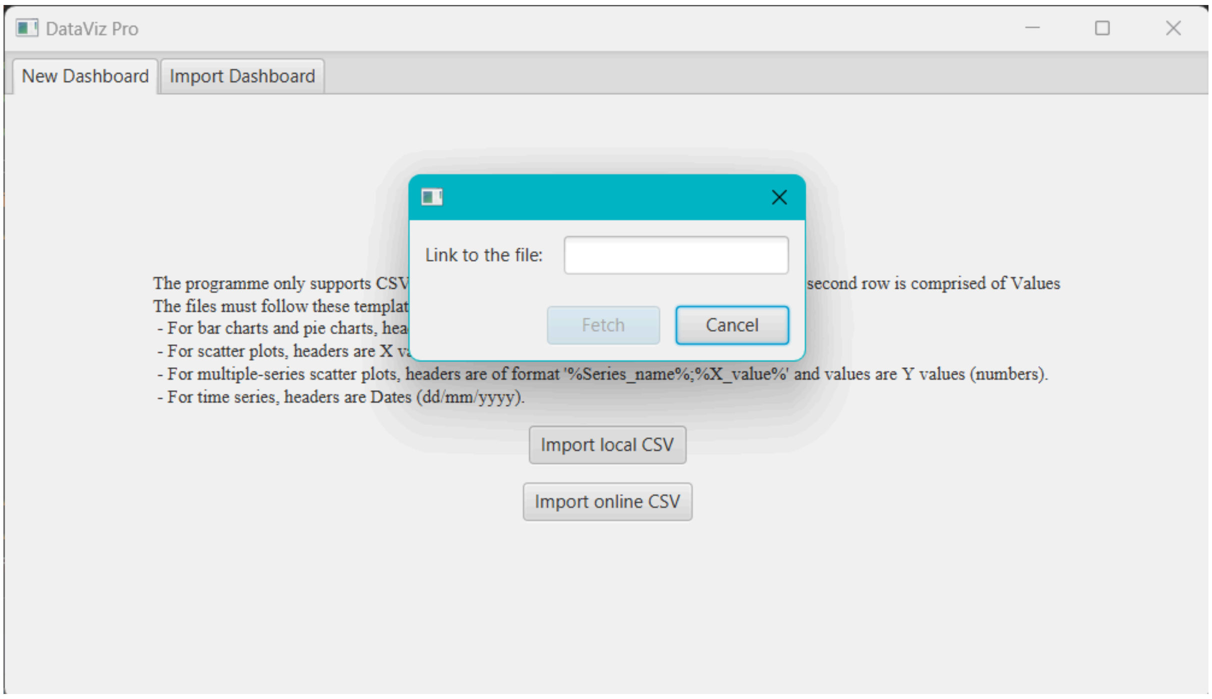
When creating a new dashboard, the user has the option of importing the data from a CSV file or URL. This kind of flexibility avails to the users for easy working of data either available locally or fetched from online sources.

Import from a CSV File: If clicked, a file browser window will be presented to the user through which he shall navigate to and select the CSV file that he would like to import.

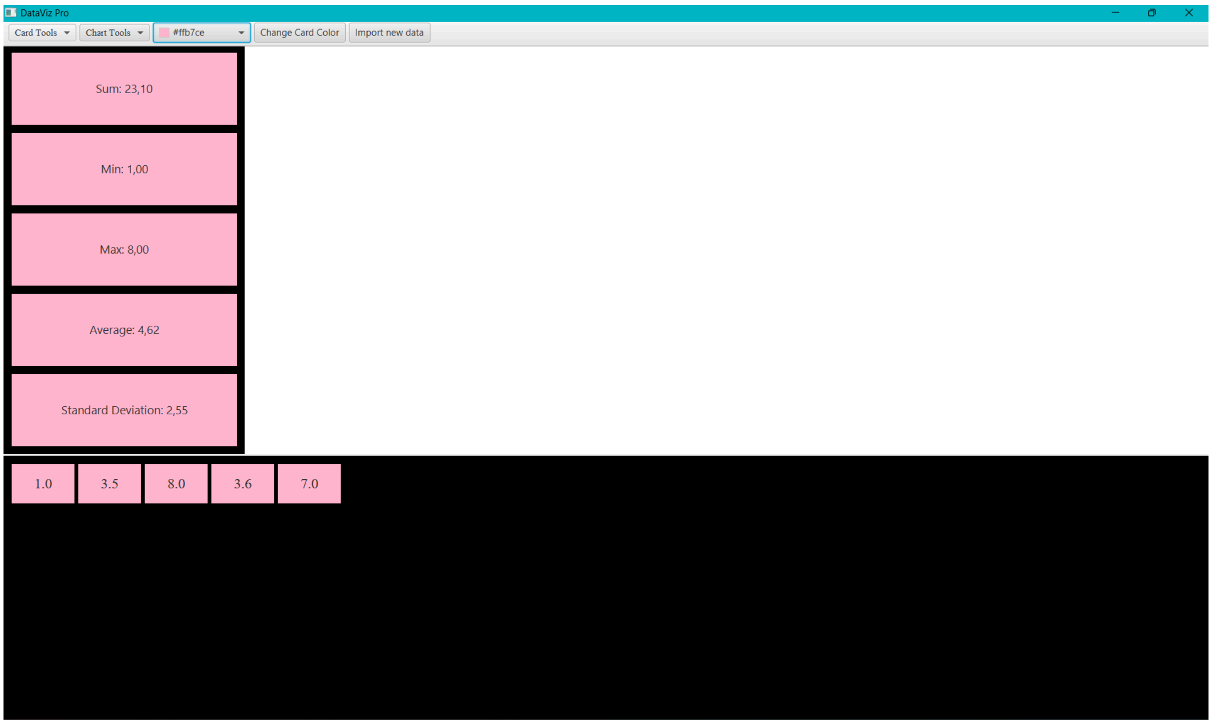
The URL where one wants to import data is done by the user after clicking on "Import via URL," and a dialog window is opened that lets one enter the URL of the data source. The application then fetches the data directly from the specified web address.



The programme only supports CSV Files where the first row is comprised of Headers and second row is comprised of Values
The files must follow these templates:
- For bar charts and pie charts, headers are categories.
- For scatter plots, headers are X values (numbers) and values are Y values (numbers).
- For multiple-series scatter plots, headers are of format '%Series_name%;%X_value%' and values are Y values (numbers).
- For time series, headers are Dates (dd/mm/yyyy).



After the user input a .csv file:



Sum: 23,10

Min: 1,00

Max: 8,00

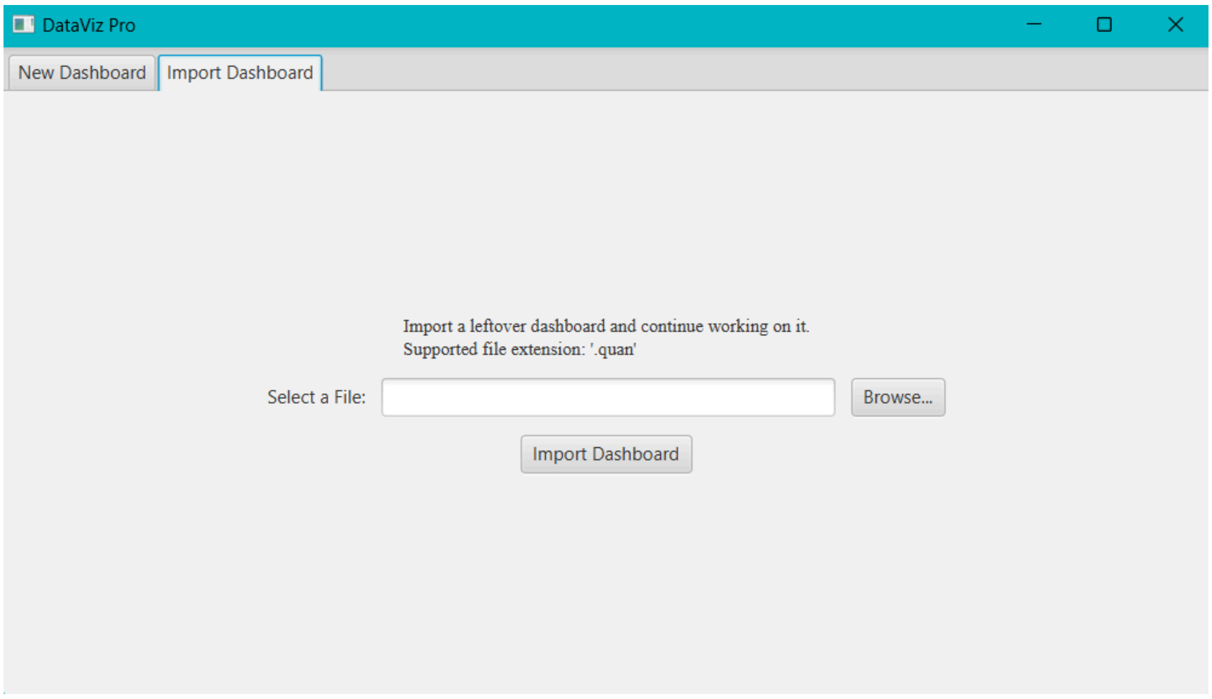
Average: 4,62

Standard Deviation: 2,55

1.0 3.5 8.0 3.6 7.0

Opening an Existing Dashboard

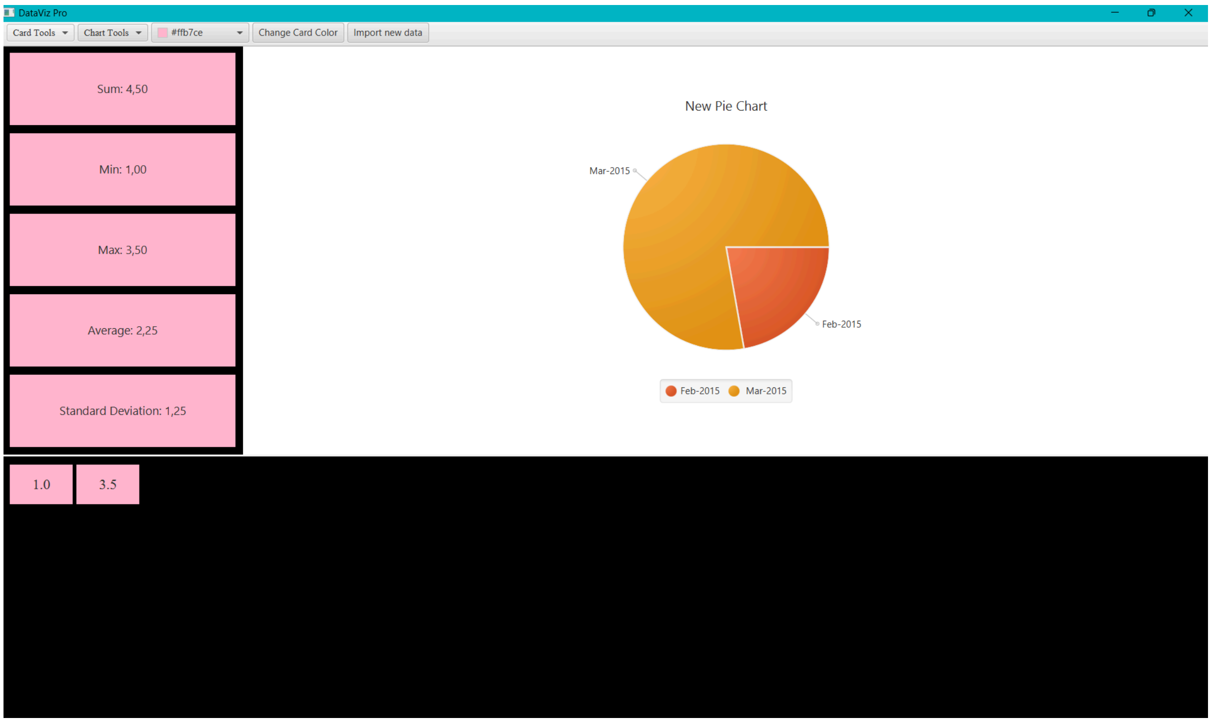
The program supports opening an existing dashboard from the custom file format ".quan".



Import a leftover dashboard and continue working on it.
Supported file extension: '.quan'

Select a File:

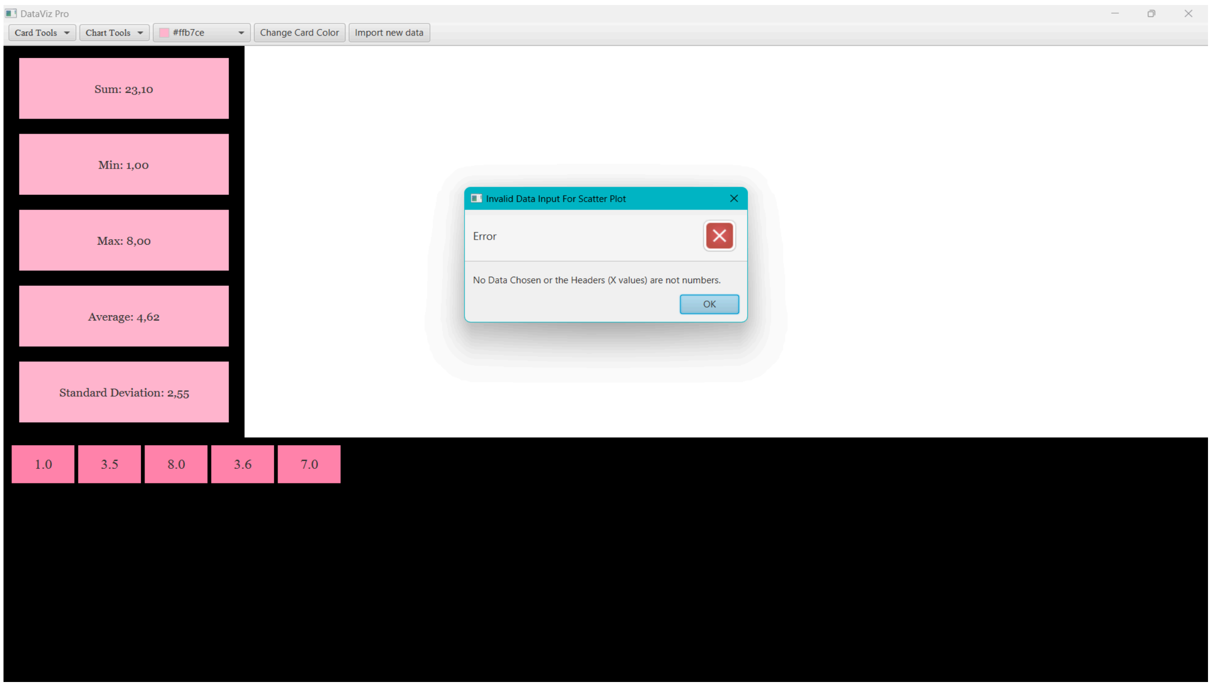
After the user inputs a .quan file with a saved Time Series:



Navigating Errors

It is developed in a way so that the users can be warned by the warning about a probable problem with their data. In the case when the data import is wrong and does not match, the warning window gives a clear notice immediately. Once the data is successfully imported and verified, the main dashboard workspace becomes available to the user. Here, an interactive environment will reside, and the magic happens with the visualizations. The user goes on to further discover the data, do different types of visualizations using tools, and then make changes to the dashboard's appearance to their preference.

An example of when the .csv file does not comply with the template:



No Data Chosen or the Headers (X values) are not numbers.

4. Program Structure

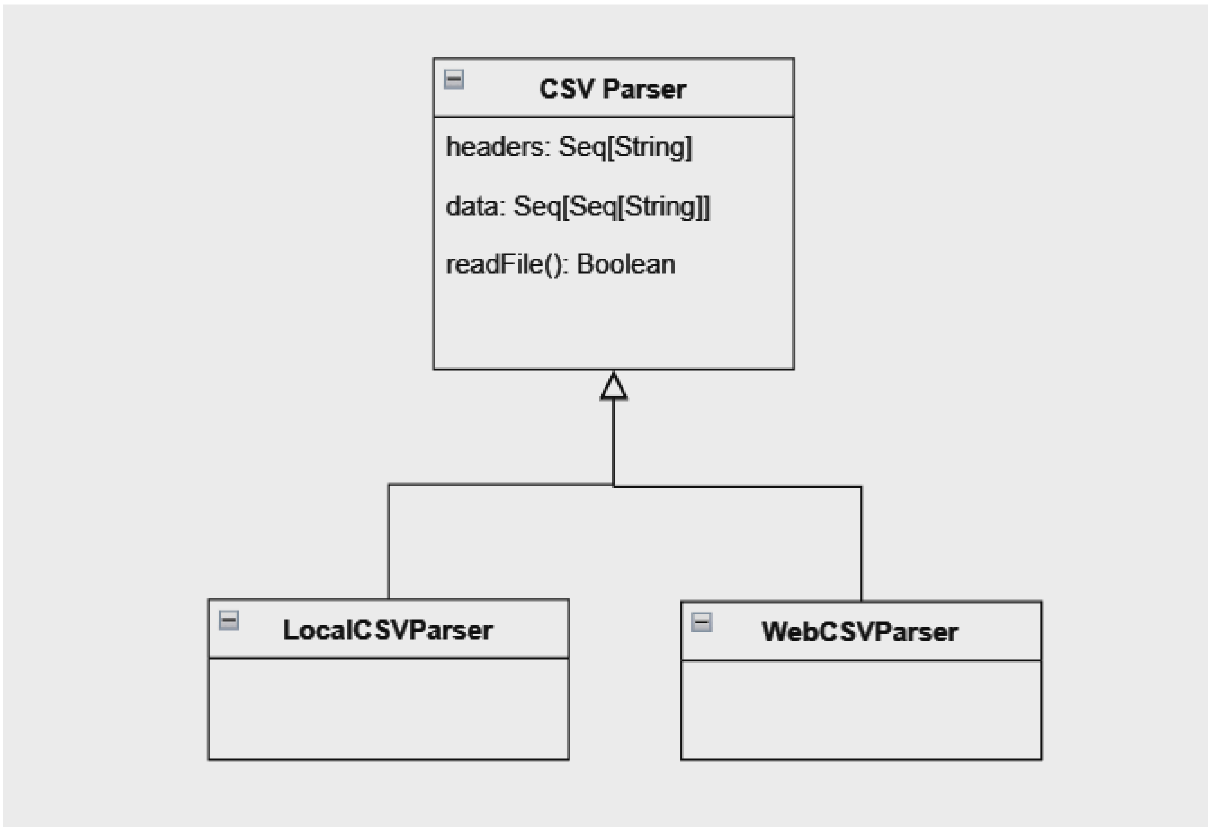
The Program Structure is designed with a clear and modular approach to ensure maintainability and scalability. This section outlines the division into main components, the realization of the class structure, and the relationships between classes. I will attach a UML diagram for the sake of clarity.

The application's architecture is segmented into five principal components, catering to distinct functionalities within the system:

1. File Handlers

   This component consists of three main classes:

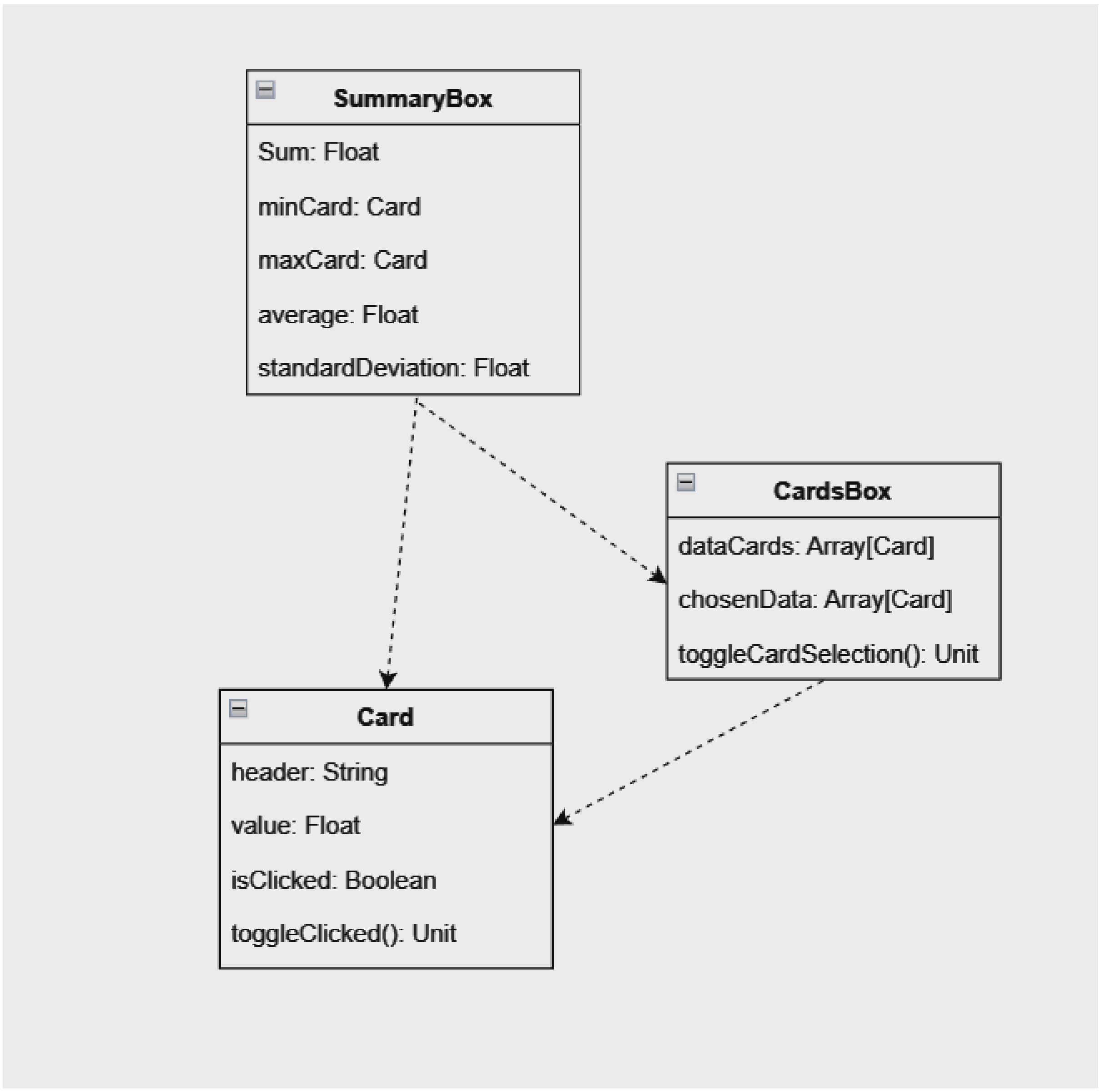
   - CSVParser: An abstract class with derived classes LocalCSVParser (for local file reading) and WebCSVParser (for reading files from URLs), handling the importation of new data.
   - DashboardExporter: Utilized for saving the current dashboard state into custom .quan format files, ensuring user's work can be persisted and retrieved.
   - DashboardImporter: Facilitates importing existing dashboards from .quan files, enabling users to review their work and continue working on it.



2. Data Display:

   This component consists of three main classes:

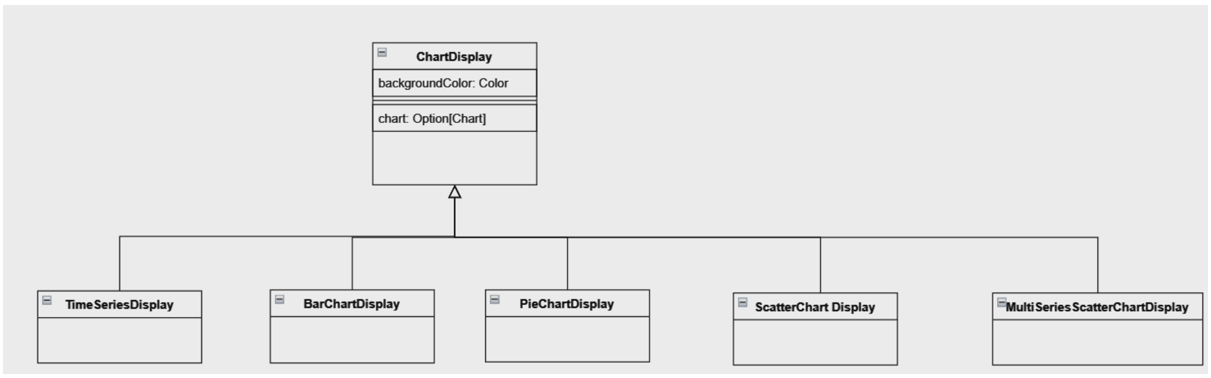
   - The Card class depicts each data point as a Card tile.
   - The CardsBox class shows all Cards created from the data points in the given input files
   - The SummaryBox class displays statistics such as Sum, Min, Max, Average, Standard Deviation



3. Charts Display:

This component consists of one main class and one main trait, coupled with five children:

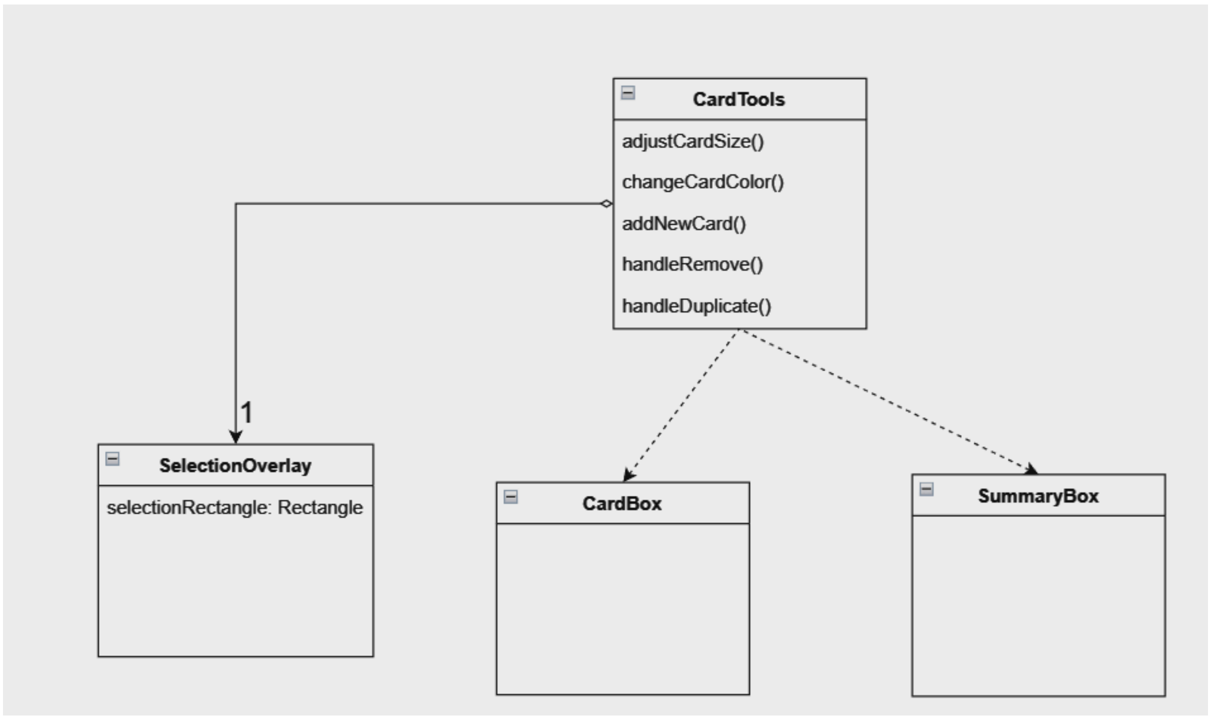
- The ChartDisplay trait, with subclasses BarChartDisplay, LineChartTimeSeriesDisplay, MultiSeriesScatterChartDisplay, Pie ChartDisplay, ScatterChartDisplay
- The ChartBox, the wrapper of the chart visualized



4. Tools

This component consists of two main classes:

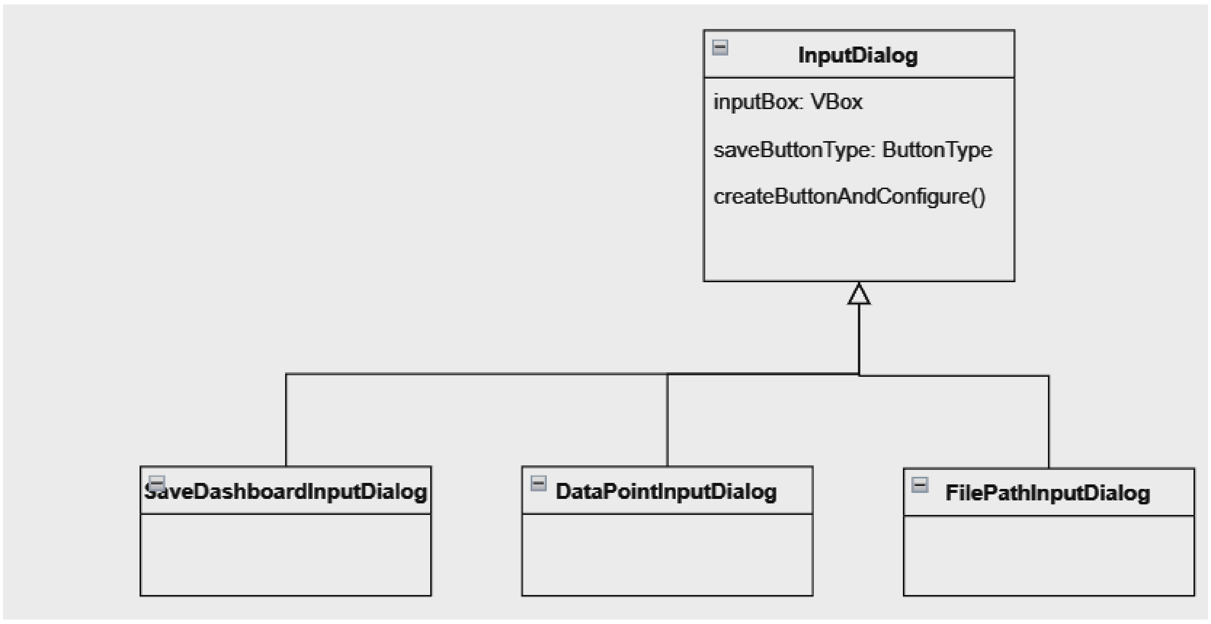
- The CardTools class provides tools for interacting with cards and summaries in the dashboard. This includes functionality to change card sizes, and colors, add new data, duplicate, remove, and perform rectangle selection on cards.
- The ChartTools class provides interactive tools for generating and managing charts in the dashboard, allowing users to create various types of charts, save the dashboard, and manipulate data visualization.



5. The User Prompts

This component consists of one trait with two children classes and multiple children of javafx.scene.control.Alert class:

- InputDialog trait is a generic trait for creating input dialogs in ScalaFX applications. It provides common functionality for different types of input dialogs.
- DataPointInputDialog is the dialog for local input to add a new data point, allowing the entry of a header and a numeric value, extending the InputDialog trait with specific configurations for handling user inputs.
- FilePathInputDialog is the dialog for web input, allowing the user to enter a URL string, extending the InputDialog trait with specific configurations for handling URL input.
- SaveDashboardInputDialog is the dialog for dashboard savings, allowing the user to choose the directory for the file to be saved.
- Alerts are warning windows popping up when there is an error.



6. The GUI and UI components:

   The GUI merges the remaining classes to create the User Interface Window.

The program architecture deviates considerably from the project plan. Since the project plan is just a tentative plan, I continuously improvise so that the program is more concise and scalable by following the DON'T-REPEAT-YOURSELF principle. So I decided to separate the program into many classes, each class handling a user interface element. However, I admit that this structure is composed of various unnecessary classes that are redundant. In contrast, there are also long methods/functions that can be refactored into classes.

The documentation for the project can be found in the directory \`target/scala-3.4.0/api\`

5. Algorithms:

The fundamental mathematical statistics in the program are the sum, mean, min, max, and standard deviation. I utilize the built-in functions to calculate the sum, min, and max statistics. However, for the average and the standard deviation, I derived custom methods to adapt to the program structure, based on the fundamental formulas:

$$mean = \frac{total\ of\ the\ values\ of\ the\ cards}{number\ of\ cards}$$

$$standard\ deviation = \sqrt{\frac{\sum_{value} (value - mean)^2}{total\ number\ of\ cards}}$$

6. Data Structures

Seq, Array, Buffer, and ObservableBuffer are mostly used in this the program. Seq is chosen because of its simplicity. Buffer is used to store the cards since it eases the modification of data points. Array is opted for other kinds of collections because it is one of the classes I am most familiar with. ObservableBuffer is utilized when there are necessities to store XYChart.Data objects.

7. Files and Internet access

For file imports, the only compatible file format is CSV files, and the data in the file must comply with these formats for the corresponding charts:

- Bar charts and Pie charts: The first row is the headers and the remaining rows are values (only numbers are accepted).

  Example:
  Quan, Minh, Linh, Trang, Thang
  0.5, 4.5, 6, 3.4, 3
  6, 4, 1.5, 5.5, 5

- Scatter chart: The first row is the x values and the second row is y values.
  Example:
  0.5, 4.5, 6, 3.4, 3
  6, 4, 1.5, 5.5, 5

- Multiple-series Scatter chart: The first row is the headers combined with x values and the second row is y values.
  Example:
  A:0.5, B:4.5, A:6, B:3.4, A:3
  6, 4, 1.5, 5.5, 5

- Time Series: The first row is the date of format dd/mm/yyyy and the second row is the values.
  Example:
  22/02/2003, 23/02/2003, 24/02/2003, 25/05/2003, 26/02/2003
  4, 5, 6, 7, 8

The program is capable of accepting files retrieved from the Internet through a provided URL string. This functionality involves the creation of a new URL object from the input string. Subsequently, an InputStreamReader is instantiated for this URL, and then passed to a buffered reader for file reading. The contents of the file are processed similarly to other CSV files, adhering to the specified format. It's essential to emphasize that files imported via URL must conform to the predefined format.

For exporting files, I've devised a proprietary file format named ".quan". In a .quan file, the initial line specifies the RGB values representing the color of the data cards within the stored dashboard. Following this, the second line indicates a numerical identifier for the type of chart featured on the dashboard. Subsequently, the succeeding line contain the selected data intended for visualization on the dashboard's chart. Upon loading this dashboard within the program, users will encounter the saved chart along with the chosen data cards, rendered in the color scheme utilized during the dashboard's initial saving. It's important to note that there are some errors with this feature that will be discussed later on.

8. Testing

The program is mainly tested by inputting various .csv files, both cleaned and erroneous, to see how the program responds.

For non-GUI components, I created some unit tests to check if the outputs match the expected results, as well as used the debugger to track the variables' values.

9. Known bugs and missing features

If there are some data points not selected when saving a dashboard, the saving/loading dashboards features are not working properly.

10. 3 best sides and 3 weaknesses

3 best sides:

1. Intuitive user interface.
2. The rectangle selection tool is working well.
3. The statistics can be automatically updated when there are changes to the data.

3 weaknesses:

1. The output pie chart is not standardized: it does not start from the top and the slices are not ordered.
2. Limitations in .csv templates supported
3. The source code is not 'DRY' in some places.

11. Deviations from the plan, realized process, and schedule

I took the course last year but could not complete the course project. So I decided to spend the summer studying the library documentation. Therefore, the implementation took less time than expected.

When I devised the project plan, I did not study the library documentation very well. Therefore, there were more classes and objects I had to implement than I previously thought.

The order of the progress: Building the utility functions of the program (CSV Parser) → building the layout and the UI → building the chart displays and tools → refactoring the code → debugging.

12. Final Evaluation

To be honest, I am not happy with the aesthetics of the program, as I do not see myself as an artistic person. I spent quite some time just to figure out the layout and choose the color palette. Some parts of the project are redundant and streamlined, but I expect that it will take a huge amount of time.

If I started again, I would devise the technical plan more rigorously as it would reduce significantly the amount of time I spend implementing the features.

13. References

Credits to Hung Nguyen for his great project that I used as a source to have a foundation knowledge on ScalaFX library: Hungreeee/Covid-19-Dashboard: Implement an interactive data-dashboard, displaying and visualizing time-series Covid-19 situation in Finland using ScalaFX. (github.com)

Credits to TA Hien Ta for her hints about how I can visualize the data points.

These are the demos from ScalaFX that I consulted to learn how to create charts from a data source: scalafx/scalafx-demos/src/main/scala/scalafx/scene/chart at master · scalafx/scalafx (github.com)

Comments on AI use: I utilized Github Copilot to help me write documentation for the classes and methods to generate Scala documentation.

14. Appendices

Here is the link to the project repository: Tran Quan / Data Dashboard · GitLab (aalto.fi)

The documentation can be found under the directory ‘target/scala-3.4.0/api’.

The executable file of the program can be found under the directory 'out\artifacts\CS_C2120_Quan_s_Data_Dashboard_jar'