COMP2811 - Example code, dataset: analysis & review

As part of the first week, I am reviewing the example code and Yorkshire water dataset we have been given

Example code – The Earthquake data viewer example

"cwk2_solution_final" has been provided for use in this coursework, the folder is base set of example code files which contain the structure of a similar project which was used to view Earthquake data alongside a CSV parser. We will modify this code to be used for our water project. The CSV parser present is a generic CSV parser written for C++ so does not need to be adapted. In the table below I have analyzed each file and written a description of what it is and what needs to be modified.

Filename	**Description & Required modifications**
`csv.hpp`	This file contains a generic CSV parser written for CPP which has been sourced from the GitHub repository at https://github.com/vincentlaucsb/csv-parser. As the CSV parser is a generic solution it does not need to be modified to suit our needs and should provide all the functionality required. I have also tested the speed of the CSV parser on our dataset which contains 140,733 lines of data. The parser printed every line of the file in ~7 seconds with a lot of that time likely due to the overhead of printf. This means that the CSV parser can be used without too much concern over speed. However, using a database may still make filtering the data faster.
`dataset.cpp`	This file implements the functions that are prototyped in the dataset header file. The functions allow us calling code to load a data set and operate analysis on the data such as finding min/max values and finding the mean of a certain selection of data. Our application will need to implement a similar structure, having a class that manages the database. However, we will need to modify the analysis functions to be relevant to our water dataset.
`dataset.hpp`	This file outlines prototypes to be used by `dataset.cpp`. The prototypes are implemented fully by the `dataset.cpp` code.
`main.cpp`	This file is the entry point for the entire application and contains the calls that create the application window and run the application logic. The main file should be kept clear of performing any logic and just used to call overarching functions.
`model.cpp`	This file contains the data / table model which is used by the Qt library to display our data. The model loads in data using

	dataset.cpp and then formats it for display such as deciding which text alignment to use and passing the data to Qt in a format that Qt can understand. As this code handles generating the tables and figures shown in the application, we will need to extend the functionality to show our larger and more complicated data set. The code also does not natively support graphs so this functionality will have to be written from scratch.
`model.hpp`	This file outlines prototypes to be used by `model.cpp`. The prototypes are implemented fully by the `model.cpp` code.
`quake.cpp`	This file holds the quake class where each instance of a quake class holds its positional and impact data as well as functions for displaying the quake's data in a string list format. The class is set up for earthquake data so we will need to completely modify this class to take our data points however the functionality to generate strings of the data is a useful starting point.
`quake.hpp`	This file outlines prototypes to be used by `quake.cpp`. The prototypes are implemented fully by the `quake.cpp` code. This file also contains constants for maximum and minimum values used for functions dealing with the positional data of the Earthquake.
`stats.cpp`	This file contains functions used by our Qt application for displaying analysis dialog boxes of the Earthquake data where analysis is datapoints such as mean, max and minimum magnitudes. The code tells Qt how to arrange the widgets used in the stats dialog. The code proves a very useful starting point for creating dialogs for our application, for example when the user wants more details on the health hazard of a specific chemical.
`stats.hpp`	This file outlines prototypes to be used by `stats.cpp`. The prototypes are implemented fully by the `stats.cpp` code.
`window.cpp`	This file contains the code for the whole application window for rendering such as the file menus and status bars as well as logic such as for opening a CSV file and setting the data location. This file contains the main logic of the application and is called to show by the file main.cpp. Much of the code is generic and so can be reused by our application once we have added the additional cards we need and modify the options to be related to our data set analysis needs, functions already present such as `createToolBar` and `addFileMenu` will be great starting points.
`window.hpp`	This file outlines prototypes to be used by `window.cpp`. The prototypes are implemented fully by the `window.cpp` code.

Water data set – Government water analysis for Yorkshire

We have been provided with the UK Environment Agency's water analysis dataset for locations in Yorkshire. This dataset is from 2024 and contains 17 columns, each column containing useful data pertaining to the location of the water being tested, the chemical being tested for, the chemical quantities, the reason for the measurement and more.

Our application needs to display this information to the user in separate manageable cards that show related data in a way that is easy to view and allows the user to gauge trends in the case of data over time. Based on the names of the columns it looks like the data is also natively translatable to JSON with each column being a group and characteristic for example "`sample.samplingPoint`". Before we write the application, we need to know exactly what the data in each row represents which I have logged in the table below:

Row name & number	**Purpose**
`"@id" A/1`	This is a URL pointing to the measurement data relating to this row, also in the URL's return which is a JSON object is links to other measurements at the same location for different chemicals along with their data.
`"sample.samplingPoint" B/2`	This is a URL that points to the sampling point i.e. location that the measurement is from. The values in this column will appear repeatedly very often as each sampling point has a lot of measurements.
`"sample.sapmplingPoint.notation" C/3`	The sampling point notation is the unique sampling point ID extracted from the `sample.SamplingPoint` URL
`"sample.samplingPoint.label"`	The sampling point label is the name of the sampling point's location e.g. "`MALHAM TARN`"
`"sample.sampleDataTime" E/5`	This is the time the sample was taken in the format `YYYY-MM-DDTHH:MM:SS`
`"determinand.label" F/6`	This is the name of the chemical being measured in the sample e.g. "`Pb Filtered`"
`"determinand.definition" G/7`	This is an expanded name with acronyms and elemental symbols expanded for the chemical being measured e.g. "`Lead Dissolved`" for "`Pb Filtered`"
`"determinand.notation" H/8`	This is a unique ID for the chemical being measured
`"resultQualifier.notation" I/9`	This is a qualifier for if the chemical is under the detection limit for example "<" means the chemical is below the limit of detection.

"result" J/10	This is how much of the chemical was measured in the sample, there is no unit label for this as it is in the next column
"codeResultInterpretation.interpretation" K/11	This is the interpretation of the data for example if it is exceeding the safety thresholds however this data is missing from the data set
"determinand.unit.label" L/12	This is the unit label for the previous column which shows how much of the chemical was measured in the sample, for example "ug/l"
"sample.sampledMaterialType.label" M/13	This is the material that is being sampled for example what type of water (Ground / Lake / etc)
"sample.isComplianceSample" N/14	Boolean value for if this sample was taken to check for compliance with laws
"sample.purpose.label" O/15	This is the reason the sample was taken in text. For example, if it was taken as part of general monitoring or to comply with EU regulations etc.
"sample.samplingPoint.easting" P/16	The easting coordinate for the location of the sample
"sample.samplingPoint.northing" Q/17	The northing coordinate for the location of the sample