Explanation of DTC Software Application Task

James Pritchard

November 18, 2025

1 The Beginning

I thought I may as well write a little about the code to ensure clarity, just in case the comments in the code are not sufficient. As is probably clear, I will be using informal language since it should be (I hope) more interesting to read. Or perhaps I should say less boring.

I'll aim to make clear the structure of my code and why I structured it that way, as well as include some nice graphs, since why not.

The coding process I went through was documented on GitHub: `https://github.com/JamesPritch/Puzzles/tree/main/DTC`.

2 The General Outline

2.1 Introduction

I have never coded anything which has such a strong correlation with time before, so if I have made some rookie errors, I extend my humblest apologies. However, I think it has gone quite well, and I can say for certain that it was enjoyable to do something I haven't before.

The language I used was Python since I would have to pay to use MATLAB and R would be a bit left-field.

2.2 Structure of Code

After reading the brief for the task, I thought it would be simplest to write two pieces of software: one to generate data and save it[1] to file, and another to visualise some data which has been read from a file. I also decided that I wanted to output data into the file daily, since that enables a nice naming system for files (using a numerical value of the day) and it keeps visualisation simply by only graphing whole days at a time.

[1] Technically, them.

For the first file, my plan was to have one function to generate one data point at a time and another to save a whole load of datapoints to the file, which I would then use in various loops to achieve the desired effect. I decided to use a `np.array` to store the data since it can handle elements of different datatypes which was crucial here and is probably faster than using lists.

For the second file, I planned to have a function to read the `np.array` from the file and another to visualise it.

3 More Detail

3.1 Generation and Storing of Readings

The `np.array` for the sensor output is somewhat uninventively named `sensor_outputs`. I should note the time value is the number of seconds that have passed since midnight that day - a nice monotonically increasing sequence.

As I wrote the code to generate `sensor_outputs` with the function `generate_readings`, I outsourced the generation of each individual variable (temperature, humidity, pressure, and air quality) into four more functions (`generate_temp`, `generate_humi`, `generate_pres`, and `generate_airQ`) to make the code clearer and easier to test. It would also be possible to say that this method is more applicable to an example of plugging into a sensor if one imagines these four functions as four different readings/data streams/wires coming from some black box which is the sensor.

As for storage of readings, I wrote a function called `store_readings`. What a font of ingenuity.

3.1.1 Algorithm

The algorithm for this file goes as follows:

1. Set current day.
2. Initialise `sensor_outputs`.
3. While the number of seconds that have passed since midnight is less than $24 \cdot 60 \cdot 60$, call `generate_readings` every second.
4. Store the readings to file by calling `store_readings`.

In my code, these instructions repeat forever in a loop, however, that's fairly unrealistic. Equally, it wouldn't be hard to set this loop to run for a finite set of days.

3.1.2 Equations of Readings

Since I had free rein over what the simulated readings might look like, I decided to use some nice functions that I like. I started off sinusoidally and dictated `generate_temp` should generate data of the form:

$$y = \sin(2\pi t), \tag{1}$$

where y are temperature readings and t is time. Ensuring I had an initial condition of $y = 22.5$ (as given in the example sensor output), a variation of 20 Degrees from minima to maxima, and one period[2] was of the correct length T, I arrived at

$$y = 22.5 + 10 \sin\left(\frac{2\pi t}{T}\right). \tag{2}$$

Very similarly, `generate_humi` generates humidity data of the form

$$y = 45.2 + 10 \sin\left(\frac{\pi t}{T}\right), \tag{3}$$

which begins at 45.2, rises to 55.2, and lowers again to 45.2.

I spiced things up for pressure and air quality to keep things interesting, using exponentials and logarithms since they're too beautiful to not use.

For pressure, I wanted a linear drop from 1013.2 to 963.2 during the first third of the period[3] and the an exponential increase to return to the initial pressure. The linear drop for `generate_pres` is trivial:

$$y = 1013.2 - 50\frac{t}{T/3}, \tag{4}$$

and the exponential much more interesting:

$$y = 963.2 + \exp\left\{\frac{3}{2}\ln(51)\left(t - \frac{T}{3}\right) * \frac{1}{T}\right\} - 1. \tag{5}$$

For air quality, I effectively did the opposite, namely increasing in a logarithmic fashion by 30 until two thirds way through the period[4] and then a linear decrease to the initial value. The logarithmic function in `generate_airQ` is

$$y = 12 + \frac{30}{\ln(43/3)} \ln\left(20\frac{t+1}{T}\right), \tag{6}$$

and the linear function is

$$y = 42 - 30\frac{t - \frac{2}{3}T}{T/3}. \tag{7}$$

It it probably worth saying that in each of the four functions `generate_temp`, `generate_humi`, `generate_pres`, and `generate_airQ`, a small amount of randomness is added, and the value is then rounded to the correct number of decimal places before being returned.

[2] The period used here is one day, so one complete sine curve per day.

[3] From 00:00 to 08:00.

[4] From 08:00 to 23:59

3.1.3 Storing of Readings

Nothing really to report here, there are a few ways to do this but I chose one that makes the raw file quite readable. It's in CSV format. The file itself is called `sensor_output_x`, where `x` is the current day and is stored in a folder called `sensor_outputs`. This naming system means files will automatically order themselves chronologically, although finding a specific day would be a bit tedious.

3.2 Visualisation of Data

This file is a stand-alone piece of code that literally just reads a `np.array` from the file using a function called `read_file` and then graphs it using the most boringly named function to ever exist: `graph`. I won't even bother to write a subsubsection to say the algorithm, since it involves running one and then the other. No prizes on which goes first.

In fact, I'm not sure I can think of any subsubsection that's worth writing, it's all pretty expected: `read_file` reads from the file, and `graph` graphs four plots for each of temperature, humidity, pressure, and air quality, with sensible t- and y-axis labels. A line of best fit is added to all four graphs since it's nice to have one and shows outliers quite clearly. I used a simple polynomial fit which won't win awards but gets the job done.

I suppose it's worth noting this piece of code can be run at any time, but requires a separate console than the first piece of code. Since it seems very unlikely to run the code on the final second of the day, it makes most sense to me to set the value of `current_day` to be the value of the current day -1 (i.e. yesterday), and to have the default be to visualise yesterdays data. However, it would be just as simple to set the value for `current_day` by hand to access the data for whichever day is of interest.

Without further a-do, here are the plots:

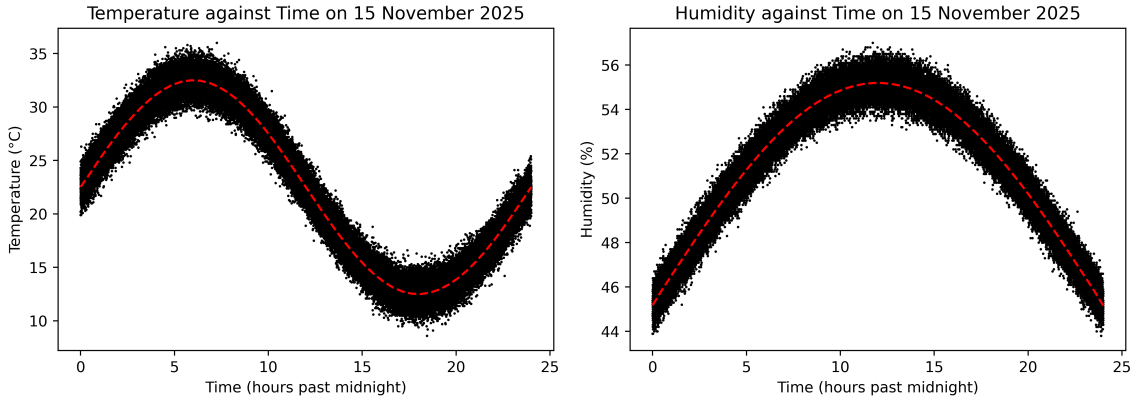

Figure 1: Temperature and humidity against time for an entire day, with data values in black and a line of best fit in red.

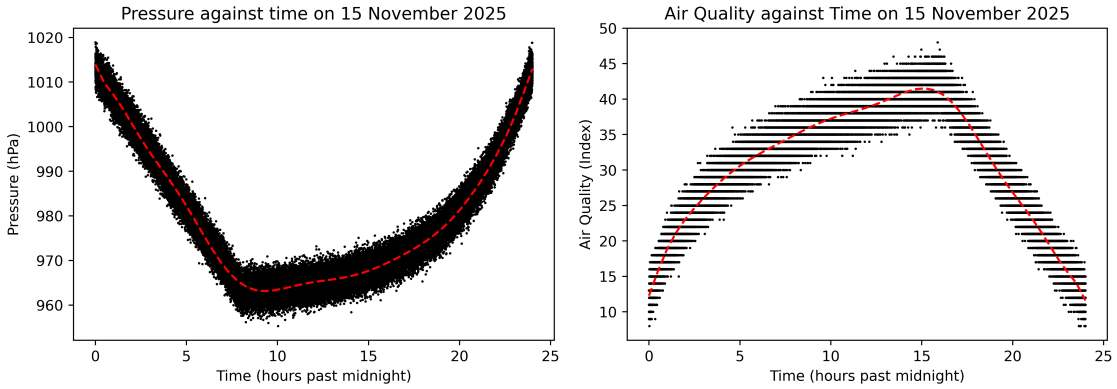

Figure 2: Pressure and air quality against time for an entire day, with data values in black and a line of best fit in red.

It all seems to work rather well.

4 Potential Improvements

- A better file naming system for the case of wanting to visualise the readings from a specific day in the past. By better I mean more easily readable to humans. Perhaps having a folder per year and then have each file named the number of days passed in that year and not since 1970.
- Data visualisation over longer time-scales than one day.
- It's possible that the saving of a file could take more than a second if running on a less powerful computer, which would mean missing a datapoint for the next day. It would be better to be able to record data with no stops at all.
- Highlighting points automatically that are a certain distance away from the mean.
- Backing up more regularly than daily to minimise data loss if something goes wrong.

5 The End

Fin.