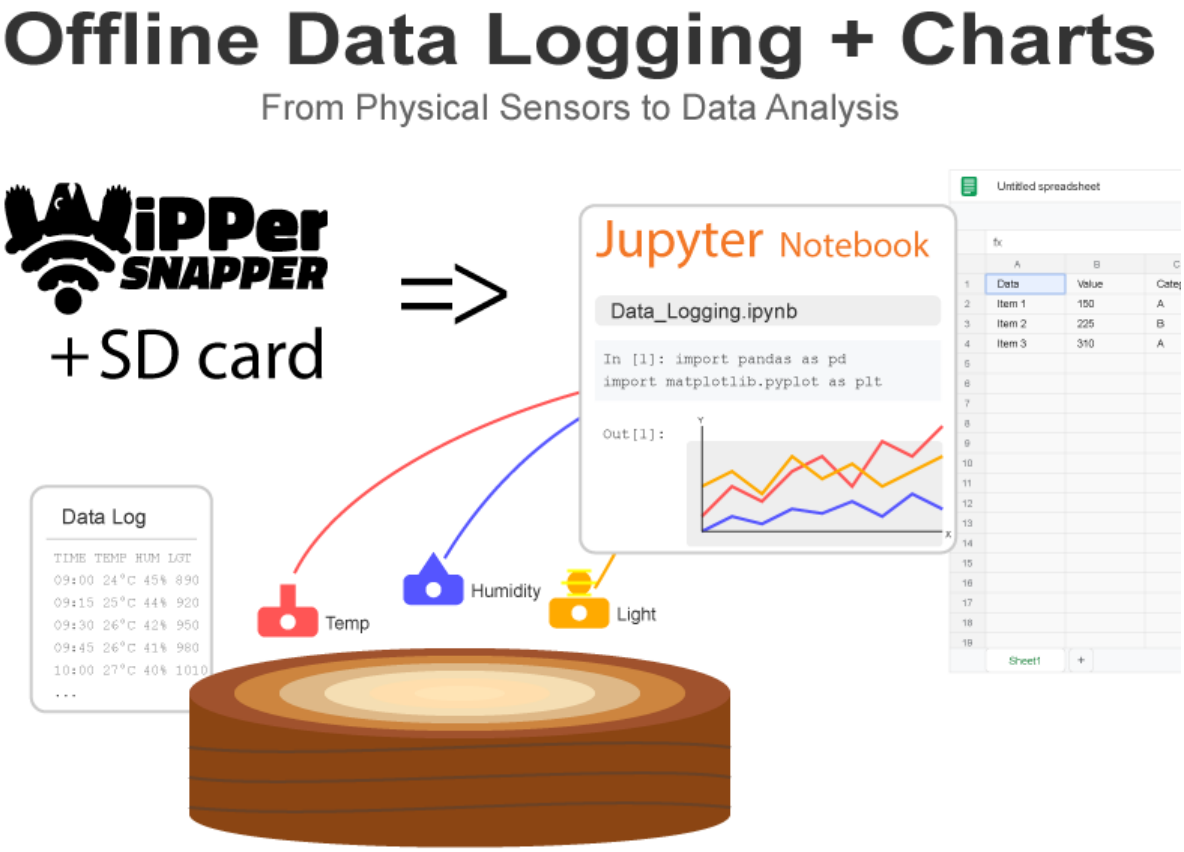
Plotting Offline Data - JSONL to CSV files, filters and graphs

Created by Tyeth Gundry

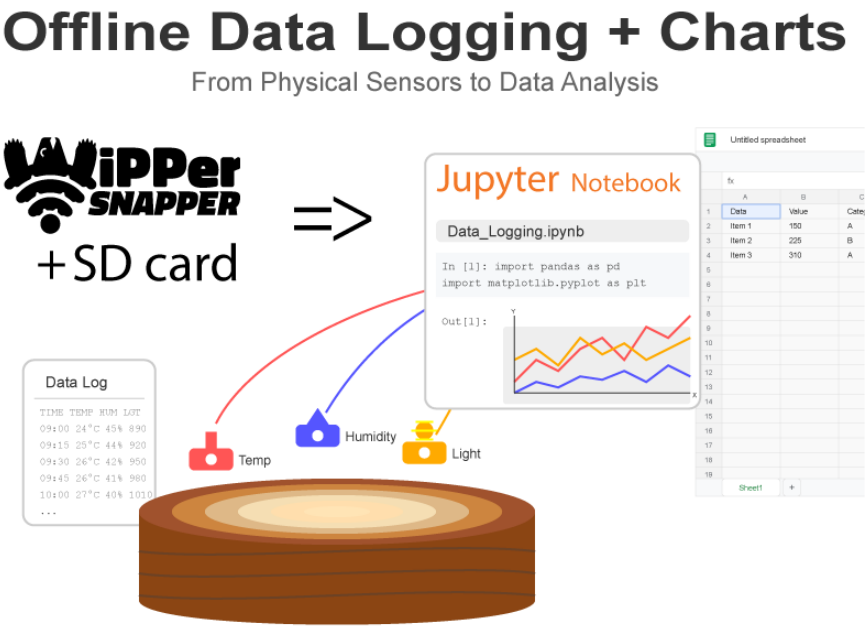


https://learn.adafruit.com/plotting-offline-data-jsonl-to-csv-files-filters-and-graphs

Last updated on 2025-04-02 04:24:47 PM EDT

Table of Contents

Overview



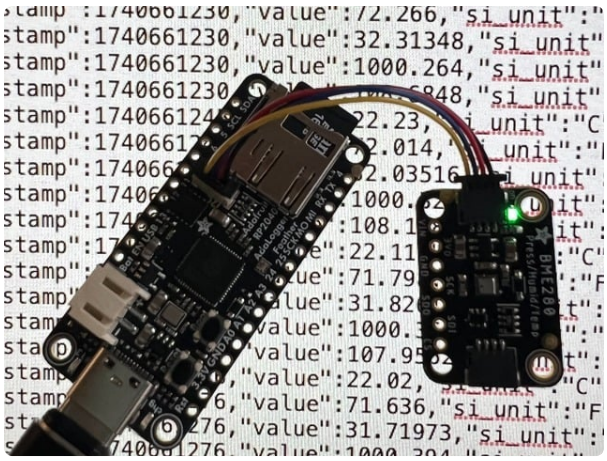
In this guide, you'll discover how to take your data files from a text format like JSONL, CSV, or even TSV into your favourite data analysis software, and start on that scientific charting adventure.

If you already have your data file(s) - feel free to skip to the next page, or keep reading and see where the example data has come from (hint: it's getting even easier to log data these days)!

Just remember, before removing memory sticks, SD cards, or USB drives from data loggers, it's usually best to shut down the data logging device and **remove the power before removing the memory storage device**. On computers like Windows, use Eject to ensure it's safe to remove cards and drives.

Data Loggers - Adafruit WipperSnapper Offline-Mode

Below demonstrates using data collected using Wippersnapper Offline-mode, which turns your Microcontroller board plus connected SD card, optional real-time clock, and bundles of Analog and Digital components and I2C sensors, into an offline data logging superstar. There's a whole separate guide about it you can check out here:

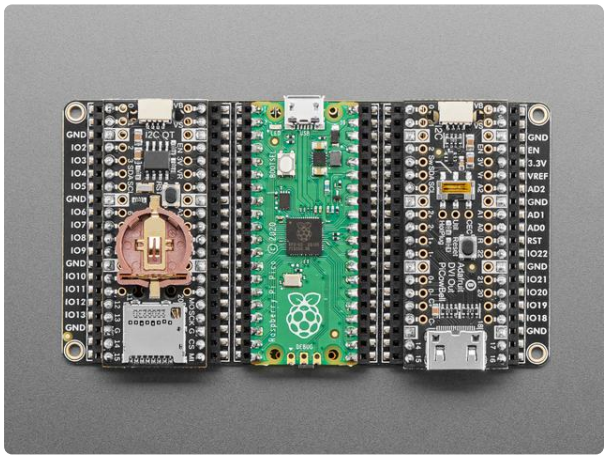


No-Code Offline Data Logger with WipperSnapper
By Brent Rubell
Overview

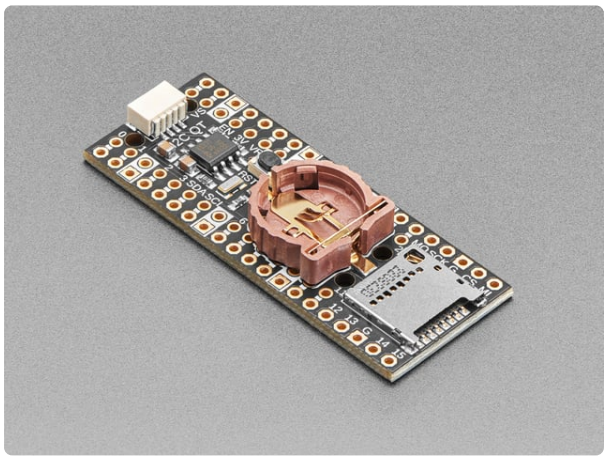
https://learn.adafruit.com/no-code-offline-data-logging-with-wippersnapper/overview

The next page will briefly outline the assembly of an example setup, which, with minor alteration, could be used for many data logging projects.

Parts



Adafruit Proto Tripler PiCowbell for Pico and PicoW
The Adafruit Proto Tripler PiCowBell is intended to be treated like a mini solder-less proto plate to simplify programming and sensor or display...
https://www.adafruit.com/product/5967



Adafruit PiCowbell Adalogger for Pico - MicroSD, RTC & STEMMA QT
Ding dong! Hear that? It's the PiCowbell ringing, letting you know that the new Adafruit PiCowbell Adalogger is in stock and ready to assist your
https://www.adafruit.com/product/5703

1 x Raspberry Pi Pico H - Pico with Headers Soldered
https://www.adafruit.com/product/5525
Any of the Raspberry Pi Pico variants will do (RP2040/RP235x)

1 x Lithium Ion Battery Pack - 3.7V 4400mAh https://www.adafruit.com/product/354

A nice large battery should keep you running for days!

---

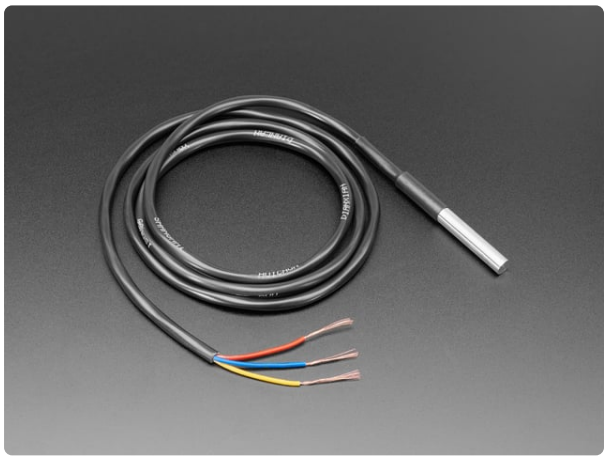
1 x SD/MicroSD Memory Card - 16GB Class 10 - Adapter Included https://www.adafruit.com/product/2693

Any micro SD card can be used, up to 32GB

There will be three "weatherproof" sensors connected through cable glands, and one breakout board version of a temperature sensor inside the waterproof container.

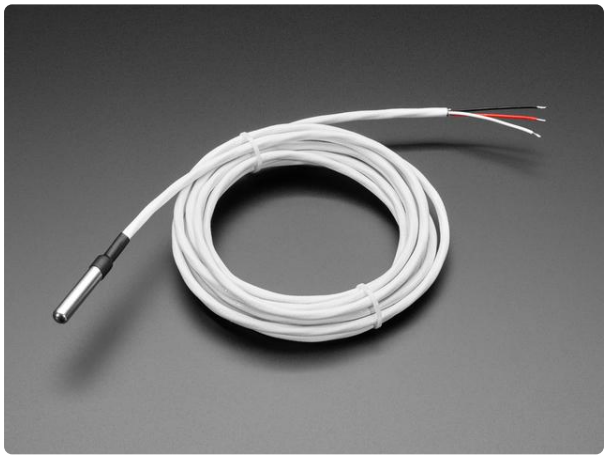
- Two “weatherproof” DS18B20 temperature sensors will be connected to pins D13 and D14 using Stemma cables connected to DS18B20 chainable breakouts for their convenient screw terminals.
- A weatherproof mesh enclosed SHT30 (temperature and relative-humidity) will connect via I2C.
- An MCP9808 temperature sensor, which will be inside the container, will also connect via I2C using a StemmaQT cable (JST-SH 4-pin).

These are the remaining components used:



Waterproof 1-Wire DS18B20 Digital temperature sensor

This is a pre-wired and waterproofed (with heat shrink) version of a 1 Wire DS18B20 sensor. Handy for when you need to measure something far away, or in wet conditions. While the...
https://www.adafruit.com/product/381



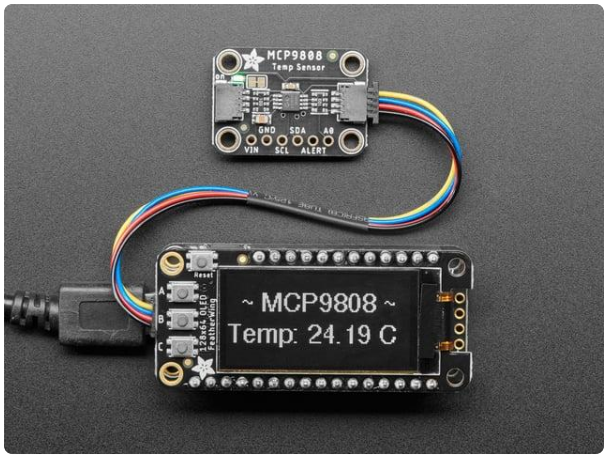
High Temp Waterproof DS18B20 Digital Temperature Sensor

This is a pre-wired and waterproofed version of the DS18B20 sensor made with a PTFE wire cable. Handy for when you need to measure something far away, or in wet conditions. This sensor...
https://www.adafruit.com/product/3846

SHT-30 Mesh-protected Weather-proof Temperature/Humidity Sensor

Take your next outdoor sensor project to the next level with this SHT-30 based temperature/humidity sensor. The sensor includes a dual-use sensor module from Sensirion in a sintered...

https://www.adafruit.com/product/4099



Adafruit MCP9808 High Accuracy I2C Temperature Sensor Breakout

The MCP9808 digital temperature sensor is one of the more accurate/precise we've ever seen, with a typical accuracy of ±0.25°C over the sensor's -40°C to...

https://www.adafruit.com/product/5027

2 x Adafruit Chainable DS18B20 Extender Breakout - STEMMA JST PH 2mm https://www.adafruit.com/product/5971

A no-solder solution for the DS18b20 sensor

---

2 x STEMMA JST PH 2mm 3-Pin to Male Header Cable - 200mm https://www.adafruit.com/product/3893

For connecting the extenders to your main board

---

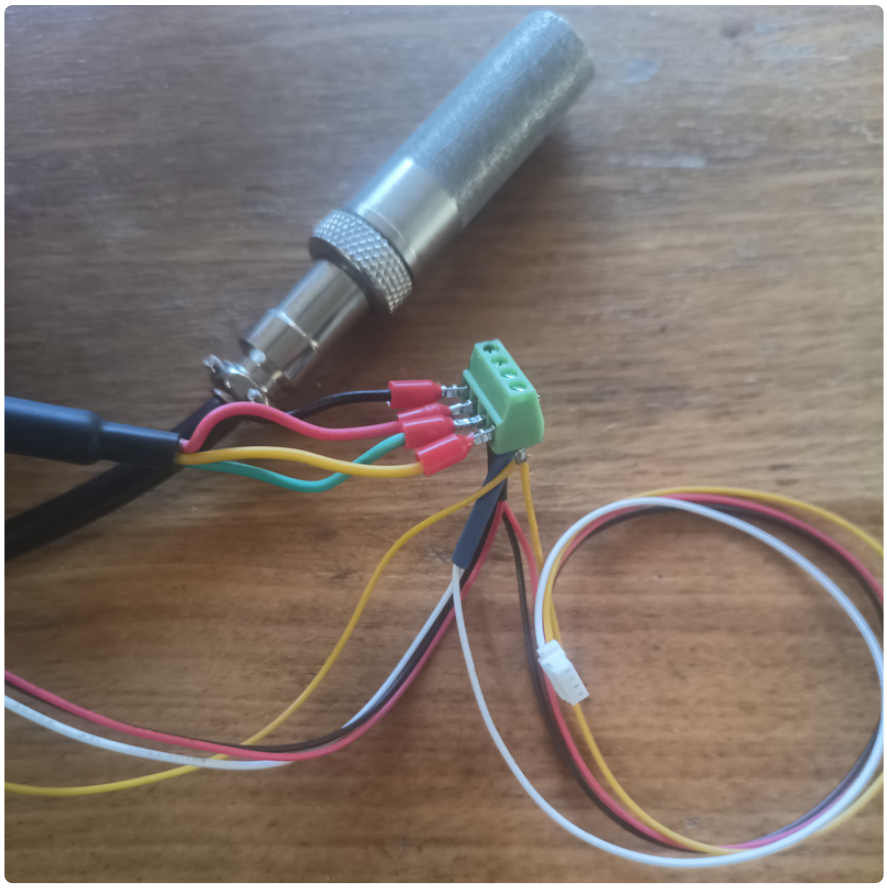
1 x STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long https://www.adafruit.com/product/4210

Any length of STEMMA QT cable is fine

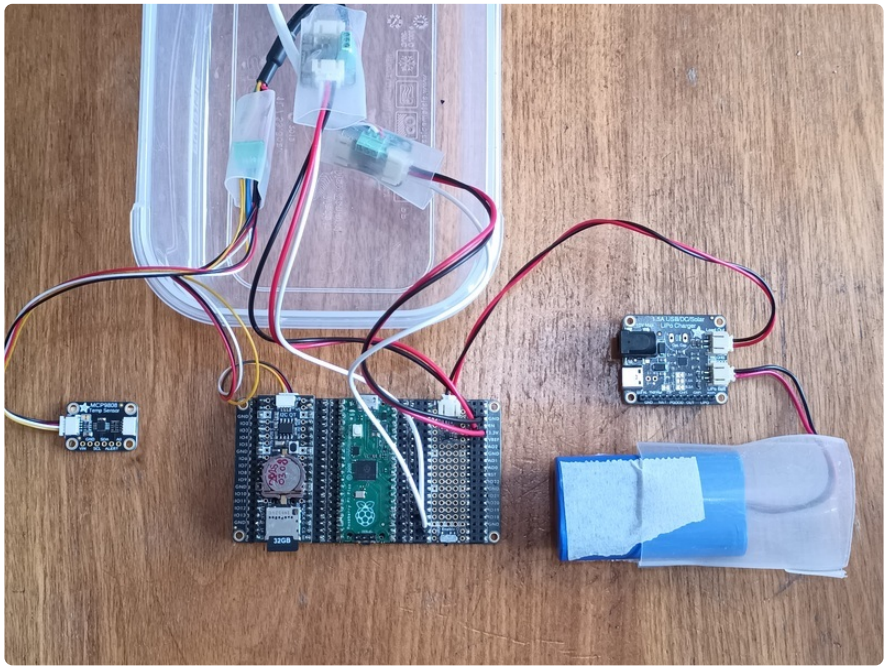
It's worth mentioning that the weatherproof SHT30 comes with individual wires (and ferrules) instead of the convenient Stemma QT connector.

You can connect it any way you know how, like solder it directly to a Stemma QT cable.
Those additional parts needed to attach the SHT30 are not mentioned in the list above.

I found it best to solder a Stemma QT cable to a 4 position screw terminal, so I can easily attach the loose wires of the SHT30.



Assembly



Keeping it simple...

This is a simple proof of concept, so in the name of simplicity, pre-soldered components are used when possible.

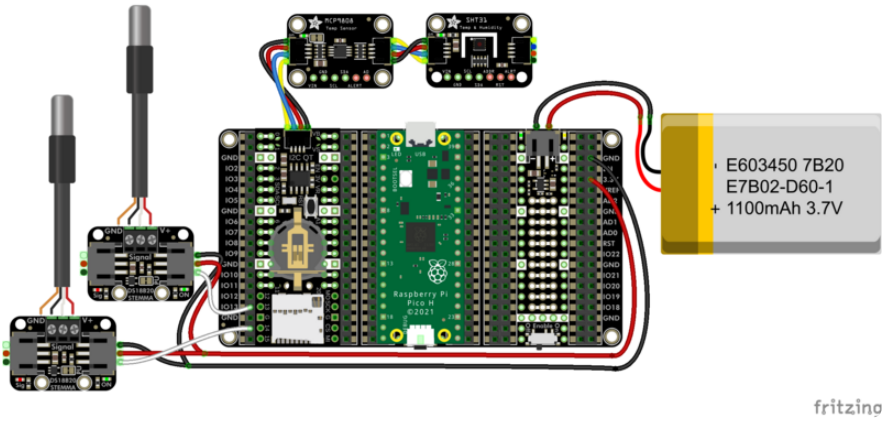
It will consist of an Adafruit Proto Tripler PiCowbell (for Pico and Pico W) as a baseplate and to easily add battery, power, and reset switches, duplicate sockets for each pin, and of course, mount the main boards. (It also comes in a Doubler (http://adafru.it/5906) variant)

Stacked on top is a Raspberry Pi RP2040 Pico with headers, paired with an Adafruit Picowbell Adalogger (providing the micro SD connector, and PCF8523 real-time clock with battery backup). Attached is also a large capacity LiPo (Lithium Polymer) battery.

There will be three "weatherproof" sensors connected through cable glands, and one breakout board version of a temperature sensor inside the waterproof container.

- Two “weatherproof” DS18B20 temperature sensors will be connected to pins D13 and D14 using Stemma cables connected to DS18B20 chainable breakouts for their convenient screw terminals.
- A weatherproof mesh enclosed SHT30 (temperature and relative-humidity) will connect via I2C.
- An MCP9808 temperature sensor, which will be inside the container, will also connect via I2C using a Stemma QT cable (JST-SH 4-pin).

The circuit diagram looks like this, which is available on the Downloads page of this guide:
(Note that the SHT30 is represented with a Stemma QT version in this diagram)



Logging Data

The Trial Run

My plan is to do some allotment greenhouse monitoring, but as an initial waterproofing test, and to indulge a curiosity, I plan to see differing freeze rates of hot and cold "foods" placed into a freezer.
I'm only doing a brief test in my freezer as my setup isn't suitable, but it does use desiccant for protection.

The DS18B20s will be poked into a pair of food containers that conveniently have silicone edged breather holes.

There will be 500g of water in each container, or 500ml but measured on weighing scales, one at room temperature and the other from a recently boiled kettle. This will make any future calculations easier.

The amount doesn't really matter to me, but hey if you keep a record of the details then it's actual science!

> It's import to note that most components won't tolerate condensing humidity without some form of conformal coating, alternatively a suitable heater or a replaceable dehumidifying substance (desiccant). Waterproofing is a whole other topic, but it's a good start to get an IP rated container with suitable ports (Adafruit sells some in the shop), use sealed components and wiring, and use cable glands to safely pass wires through the walls of the container.

SD Cards and Config files

This will become a more automatically setup in the future, which will be making it almost too easy, but for now after installing the offline data logging software (https://adafru.it/1agD) on your device, it requires configuring with your SD cards Clock Select (CS) pin and RTC (real-time clock) model in the **config.json** file.

You can always find the real-time clock (RTC) model and the SD card chip select pin information on the Pinout page from your boards learn guide in the Adafruit Learning System, just search for your product like 'pico adalogger' then go to the main guide. If you get stuck, then pop over to discord () or the forums (https://adafru.it/dYq).

Here's the beginning of the config file I used,. Yours will be similar, but you can optionally have no RTC defined:

```
{
    "exportedFromDevice": {
        "referenceVoltage": 3.3,
        "totalGPIOPins": 32,
        "totalAnalogPins": 6,
        "sd_cs_pin": 17,
        "rtc": "PCF8523",
        "statusLEDBrightness": 0.1
    },
    "components": [
    ]
}
```

The sensors and other **components** then need configuring with their pin names or i2c addresses, timings (polling period), and which sensor properties to record.

If your setup matches mine then this config will work nicely for you, otherwise look at the main guide for further information on configs and supported boards and components (https://adafru.it/1agt).

```
{
    "exportedFromDevice": {
        "referenceVoltage": 3.3,
```

```
        "totalGPIOPins": 32,
        "totalAnalogPins": 6,
        "sd_cs_pin": 17,
        "rtc": "PCF8523",
        "statusLEDBrightness": 0.1
    },
    "components": [
        {
            "name": "mcp9808",
            "componentAPI": "i2c",
            "i2cDeviceName": "mcp9808",
            "period": 30,
            "i2cDeviceAddress": "0x18",
            "i2cDeviceSensorTypes": [
                {"type": "ambient-temp-fahrenheit"},
                {"type": "ambient-temp"}
            ]
        },
        {
            "componentAPI": "ds18x20",
            "name": "DS18B20: Temperature Sensor",
            "sensorTypeCount": 2,
            "sensorType1": "object-temp-fahrenheit",
            "sensorType2": "object-temp",
            "pinName": "D13",
            "sensorResolution": 12,
            "period": 30
        },
        {
            "componentAPI": "ds18x20",
            "name": "DS18B20: Temperature Sensor",
            "sensorTypeCount": 2,
            "sensorType1": "object-temp-fahrenheit",
            "sensorType2": "object-temp",
            "pinName": "D14",
            "sensorResolution": 12,
            "period": 30
        },
        {
            "name": "Weatherproof SHT30",
            "componentAPI": "i2c",
            "i2cDeviceName": "sht30_mesh",
            "period": 30,
            "i2cDeviceAddress": "0x44",
            "i2cDeviceSensorTypes": [
                {
                    "type": "ambient-temp"
                },
                {
                    "type": "ambient-temp-fahrenheit"
                },
                {
                    "type": "relative-humidity"
                }
            ]
        }
    ]
}
```

config.json

https://adafru.it/1agE

Testing and Results

Ensure your RTC has a battery inserted (usually a CR1220 3 volt battery, but double check on your product page), and with the config correct, the microSD inserted, and for now the USB cable connected to the main board, connect the other end of that USB cable to your computer.

You want to reboot the device and verify the serial monitor output shows the sensors are logging, before unplugging the USB cable. If the RTC is not already set, then it will be set to a default value of the date and time that the offline data logger software was released.

At that point, if a battery is still connected, the device will remain logging unless switched off. Alternatively take it to the final resting place and power it up.

Come back when the experiment is done and unplug the power source (or use the switch) before removing the microSD card.

Let's go analyze those results on the computer! (I've attached them here in case you're curious)

log_1742475983.zip

https://adafru.it/1agF

Convert the Data File

It's likely your data file is not in the optimal format for data analysis. That's okay, it might be optimal for other purposes, like ensuring data validity, writing clean records with flexible columns (JSONL (https://adafru.it/1aft)), or taking up less space (CSV).

Sometimes you need more flexibility in recording extra fields for some data points and not others, or receive your data from an API that only returns a list of JSON Records with one per line, that's where `.jsonl` comes in.

Unfortunately, only a few things (usually paid) support directly importing JSONL files for data analysis, and WipperSnapper's offline mode writes to JSONL. Often it's best to convert the JSON records into a new format purely for data analysis. That might be `.csv` or `.xlsx` or some database format, just bear in mind that most formats have various limitations.

`.csv` files are great in theory, but as a "file format," it has disadvantages, mainly that it took shape in the smelting pot of industrial insanity, otherwise known as undefined behaviour or, more specifically, incompatible non-standardised usage (even before the internet).

These days, we have a bit more of a consensus since most companies want compatibility with other languages and systems.

The UK government has a guide on how it believes best compatibility can be achieved (mainly, don't use them unless appropriate). [Take a look if you are interested (https://adafru.it/1agG)](https://adafru.it/1agG), or know that ideally it should be UTF-8 encoded, but you can assume at minimum each value is separated by the field separator symbol (comma). Each row is separated by a line break (new line per row), and the file can optionally have a header row (recommended). That's about it.

Fortunately, .csv is so common that all spreadsheet and data analysis software can import it, assuming your data is "clean" (compatible and without errors). Often, after import, you still need to set data types for each field.

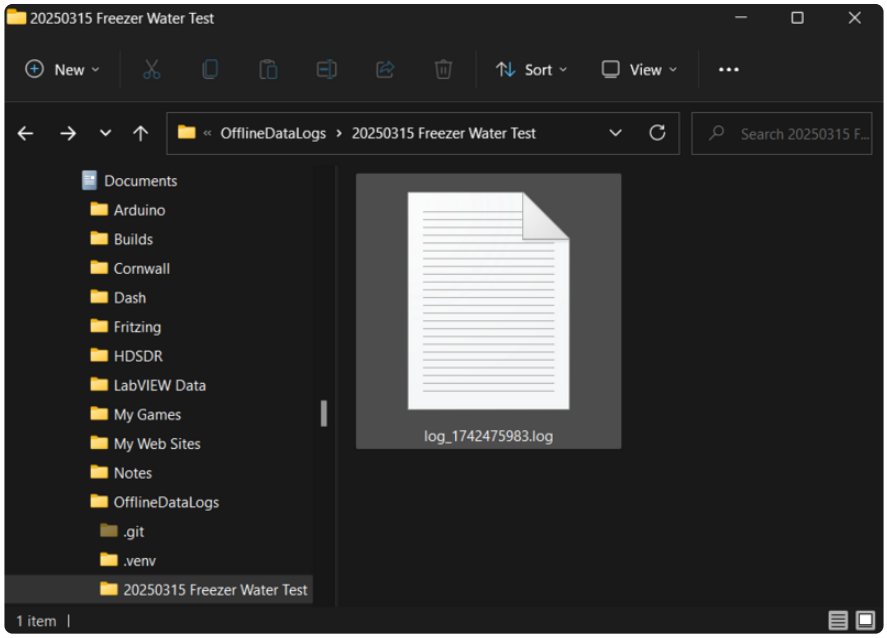
Similarly `.tsv` files (TSV = tab-separated values) use Tab ( ) to separate each field instead of a comma.

Export the Logging Files

Let's convert the offline data logger data from its native JSONL format to CSV for maximum compatibility.

First, copy the `.log` files from the SD card to your computer, it's usually safer and quicker to work with, rather than trying to repeatedly read and write them directly on the SD card.

It's good to be organised at this point. Have a folder structure for your projects and/or data files, and if necessary, rename the log files to help this goal further. I usually do this with my config and secrets files too.



Using an Online Converter for JSONL to CSV

Using an online service is often quickest, unless you already have a spreadsheet package installed on your computer, in which case use that instead. We'll show both, but start with this simple service, which is free for converting files up to 5 MB, no account needed:

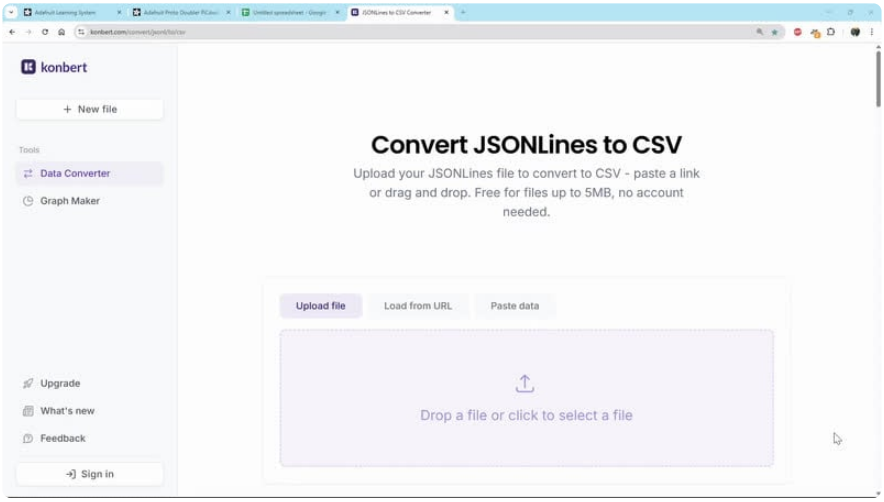
https://konbert.com/convert/jsonl/to/csv (https://adafru.it/1afv)

Simply select your `.log` file and it will upload and interpret as JSONL, then show you a preview of the data. If you use the converter via a different page or for other file formats you may find it won't accept a **.log** file as the file extension is unrecognised. Using the JSONL to CSV link directly **does** work!

Click the **Convert** button, which you'll find if you scroll down to the output section beneath the table of data.

A Download button appears with the file size indicated, click to receive your CSV file.

There is also the Studio button where you can explore your data, filter and even ask AI to (try to) make a graph.



With our converted file in hand, that's it for this stage! Now we can move on to exploring the data in one of the next guide pages.

For an alternative solution to this online converter, we'll talk later about importing the JSONL data directly into a Jupyter Notebook, and using some Python to manipulate and evaluate the data (code provided for you, it'll be easy).

There's also a nice hacky offline Python script I came up when I had a very large data file (36MB) as a first test, along with a column for the component names added into the logs, but it comes with the following disclaimer:

BETA Territory:

WipperSnapper offline mode is an early experiment, so the data format may change and become more helpful in the future. For now, the priority is minimising space used on the SD card, so to add back extra useful information, there is a loosely maintained Python script to create extra columns for the component names, by using the **config.json** if available, along with converting the data to `.csv` or `.xlsx` (Excel format).

The basic usage instructions follow, but for more information please see the GitHub repository: https://github.com/tyeth/Adafruit-Wippersnapper-offline-mode-JSONL-data-converter

Ideally, clone the repository (https://adafru.it/1agH) or download as a zip and extract the files. Then install the requirements with the normal Python package manager pip: `pip install -r requirements.txt`

Then you can run the script from the folder where you extracted/cloned it. First provide the .log files, and optionally the **config.json** and **wipper_boot_log.txt** to fill in the extra information automatically.

For example to merge all .log files from any subfolders (recursively) into one .csv, the command would be:

`python ./json_to_xlsx.py -r --merged --csv` [The script defaults to generating Excel files]

For more information, please check this section of the file which lists the optional arguments:

https://github.com/tyeth/Adafruit-Wippersnapper-offline-mode-JSONL-data-converter/blob/main/jsonl_to_xlsx.py#L4-L29 (https://adafru.it/1agI)

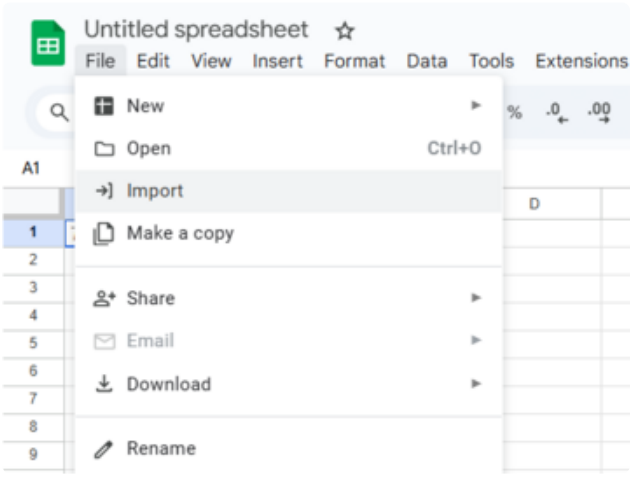
Alternatively to avoid installing anything you may wish to try, the Jupyter Notebook page of this guide will show you some online platforms that both convert the data and use Python code for charts.

Google Sheets

To use Google Sheets, first upload the data. You will probably want to start a new Spreadsheet for this importing process, rather than opening an existing one, although

at the import stage you get the choice to replace the current contents, add a new sheet to the existing spreadsheet, or create a new spreadsheet document.

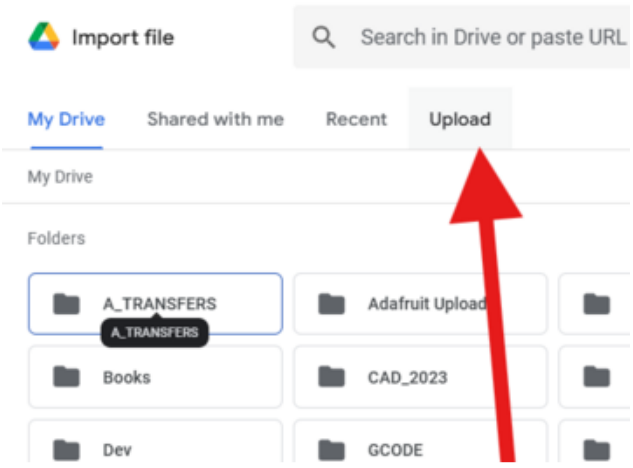
Sign in to your Google account and then visit Sheets and click "create new Spreadsheet", or visit here:
https://docs.google.com/spreadsheets/create (https://adafru.it/1agJ)



Once the spreadsheet loads, bring up the **File Menu**, then select the **Import** option.

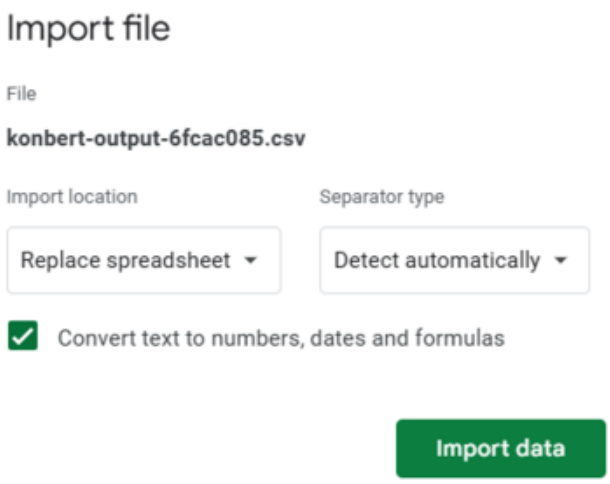
At this point you will need to have your data ready in a suitable format for import, ideally in .csv or .xlsx format.

If not then use the Guides "Table of Contents" to jump back a page in this guide and convert your data file before continuing.



Selecting the **Upload** option will allow you to drag-and-drop files or select them using the File Open dialog.

Alternatively you may have your files already in My Drive or via Shared with Me.



You'll be presented with the option to Import the file you've selected / uploaded. It will automatically detect the separator as comma.

You may wish to alter the **Import Location**, so that you can select an alternative destination from this list:

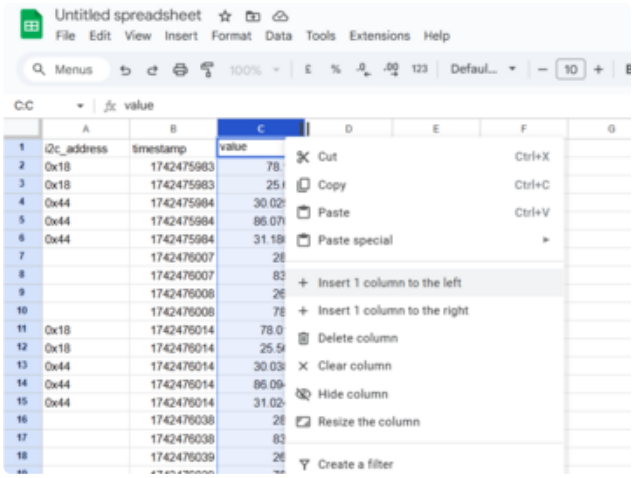
Create new spreadsheet
Insert new sheet(s)
Replace spreadsheet
Append to current sheet
Replace data at selected cell

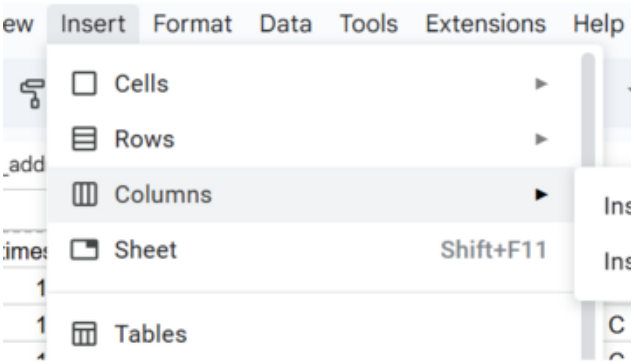
Leave the checkbox checked to allow automatically converting text to numbers, however if it causes problems then untick that option and try again.

You will be presented with your new data, rejoice!

Now it's time to do a bit of simple time conversion and then some sanity checking.

Converting UNIX Timestamps to friendly date and times





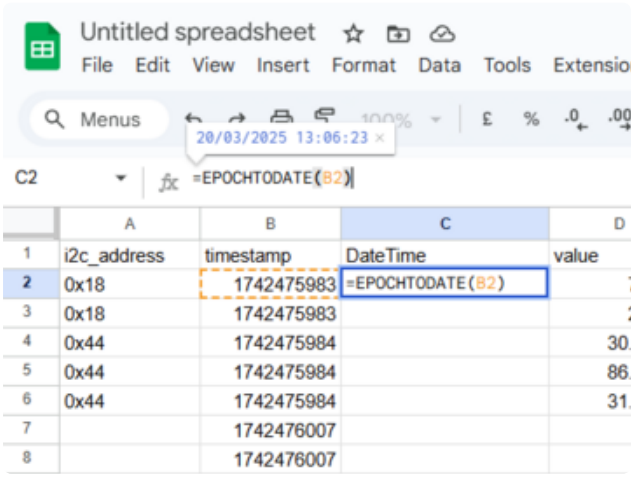
Start with adding a column between timestamp and value columns, by right clicking on the value column header (column C in screenshots) and choosing **Insert 1 column to the left**.

You can find the same option under the Insert menu on the Menu toolbar, then the Columns sub menu.

Add a column title like DateTime, and then move down one cell, out of the header row, to the first data row in that column.

This is where a formula can be used to convert the timestamps from UNIX Epoch time (https://adafru.it/INA) to a normal spreadsheet Date Time value.

You can enter the formula in the cell body, or use the formula bar above the spreadsheet to more easily see long entries.

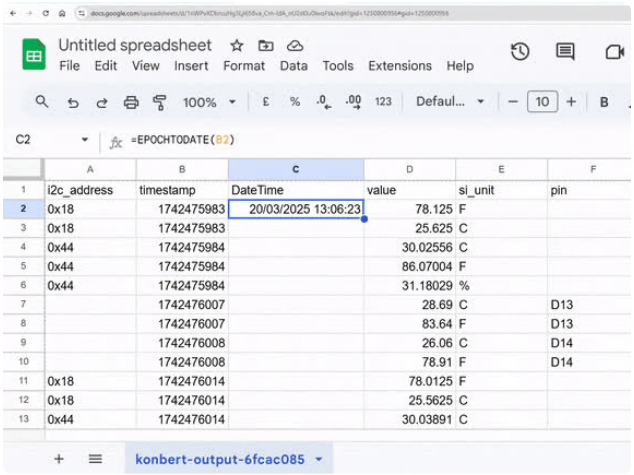


In the first empty cell in the new column, enter the equals symbol (=) to begin a formula, then the word EPOCH and accept the suggestion of EPOCHTODATE.

Then your cursor will be positioned inside the brackets of the EPOCHTODATE function in the formula bar.

Enter the address of the timestamp cell to your left, e.g. B2, or click on it to automatically populate the cell address into the formula.

Hitting enter or return will complete the formula and move the focus to the next cell. You will want to then reselect the updated cell showing the converted date time value.



You will usually find the software suggests filling in the same formula for the remainder of the columns empty cells, accept this if offered.
Alternatively you can double click on the cell marker, positioned in the bottom right corner of the cell, to fill the cells beneath automatically up to the length of cells in the adjacent column next to it. Usually completing the formula with Ctrl + Enter will also fill the range of cells with the formula.

You can easily confirm that the cells were filled to the end by using the Ctrl key combined with down or up arrow keys to navigate to the end or beginning of the current block of data (but it doesn't jump over gaps in the data).

Lastly I added an additional column for Component Name, and set its formula to use the `CONCATENATE` function, with the `i2c_address` cell and the `pin` cell as the arguments (you can add more if you wish).

It might make sense for you to also include the `si_unit`, and I2C multiplexer information if present.

My formula for the first data row in the Component Name column was `=CONCATENATE(A2, F2)` and then that was used to populate the entire column, giving me one column to distinguish between sensors.

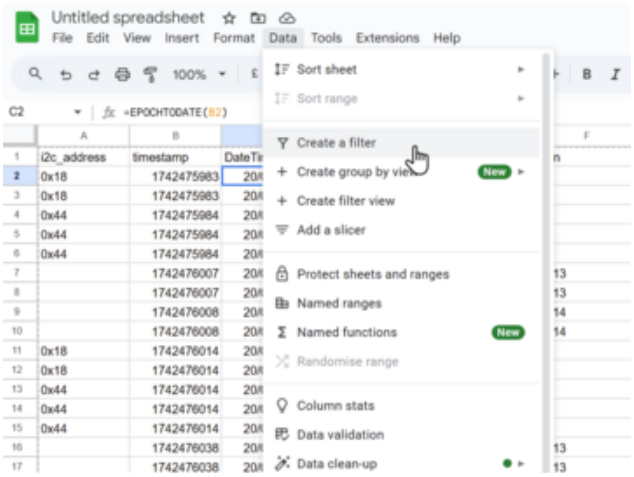
Filtering Data (and identifying problems)

So you've got your lovely data, but after a quick scroll through, you've spotted some problematic values, maybe some NAN's (Not A Number), zeros, or minus ones (only you can know what abnormal means for your experimental data).

Don't panic, it's simple to filter data and temporarily remove (filter out) the unwanted values.

Filtering can also be used to simply split a dataset that may be composed of multiple types of data (like ours has humidity and temperature which won't graph very well together).

Let's setup a filter for our data to only include the values with the units matching 'C' for Celsius, and to only include the I2C address for one sensor, the SHT30 at I2C address 0x44, so we want to filter out blanks and 0x18, as that was the main board enclosure internal temperature sensor (MCP9808).

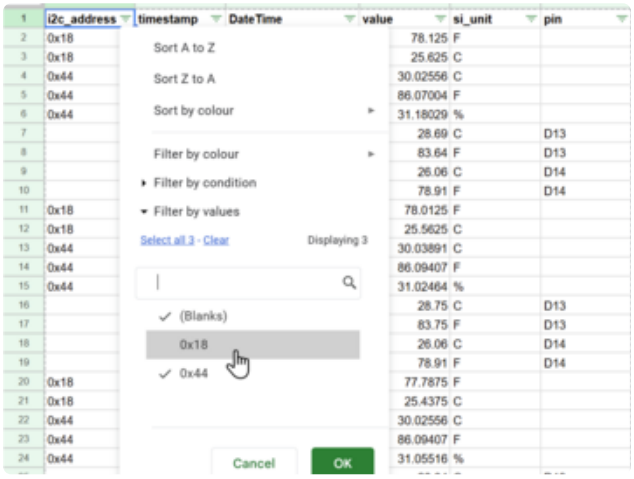


Usually it's enough to have the focus on a cell anywhere in the dataset and choose to add a filter, but for superstitious reasons I prefer to be in the header row and ideally on the column with unbroken data (like the timestamp column) to ensure the software grabs all the data rows correctly. You can also pre-select the data first before creating a filter (holding Ctrl key, or Apple key, and pressing A).

Goto the **Data** menu in the menu toolbar, and then select the option to **Create a filter.** The same option is available through the right click / context menu.

Now each column has a green filtering symbol next to the title, it looks a bit like an upside-down triangle made of the base line, middle line, and a dot.

Here's where you can remove (filter out) based on certain conditions, or directly (de)select values.



Clicking the green arrow / triangle will bring up the filter drop-down menu for that column.

Once a column has a filter applied then the green triangle becomes a green funnel showing data is being filtered in that column.

Here the `i2c_address` column has a filter option showing the 3 possible values, blank (when a component uses 'pin' instead), or 0x18 for the MCP9808, and 0x44 for the SHT30. Untick the 0x18 value and untick blanks too.

Do the same to filter the `si_unit` column to be 'C' only (for temperature in degrees Celsius).

Charting a path to victory, or how to graph the right things

Now select all the data or move to the first cell in the header row, and choose **Insert Chart** from the **Insert menu** (or right click / context menu).

It's not ready yet, but after changing a few things like the horizontal X axis to be the Date Time column, and only the values column for the Data Series, things start to come together.

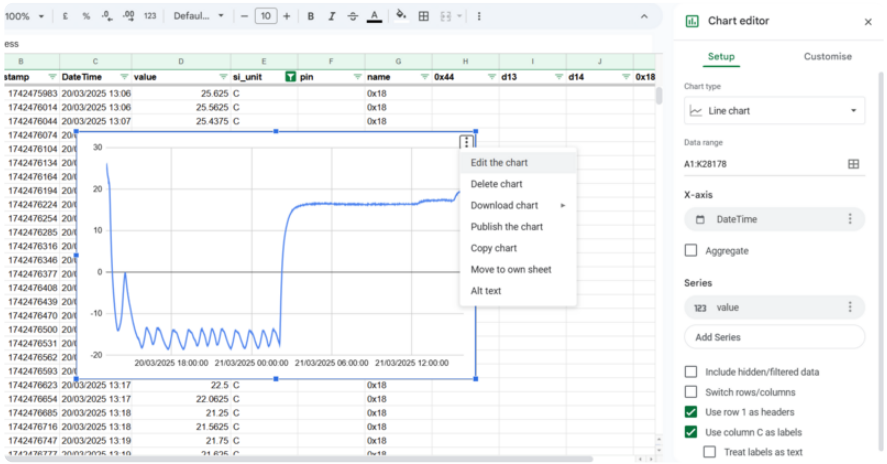
Start with selecting the **chart style** / type, a **line chart** is often a good starting choice. Then confirm the **data range** is correct, you may wish to remove the value and then use the 4 boxes icon to bring up the suggestions, as the first suggestion in that list is usually the right one. You can also use a comma separated list of ranges.

Doing the same with the **X axis** (removing suggestion) and using the **Add X-axis** button brings up the column names, making it easier to pick the `DateTime` column.

Now add the values. Choosing **Add Series** and selecting the `value` column should produce a recognisable graph. The final checkboxes are your choice, I like to **Use row 1 as headers**, but be sure to have the final checkbox unchecked, captioned **Treat labels as text** or the dates will be incorrectly handled.

You may wish to set up the tick marks and min/maximum values for each axis, this is available on the second tab labelled **Customise**.

To get back to this setup, use the triple dots menu in the top right of the chart area. It also allows downloading and moving the chart to its own worksheet.



If you change the filter, then the graph updates, but you'll notice it doesn't display well with more than one sensor. This is because it wants a separate column for each data series. We'll cover that in a minute, but to make life easy and get individual graphs for each sensor, you can create a new sheet and use a formula to include only filtered data from Sheet1 (or whatever the worksheet containing the imported data is named).

Filter formula for a new sheet of just one sensor

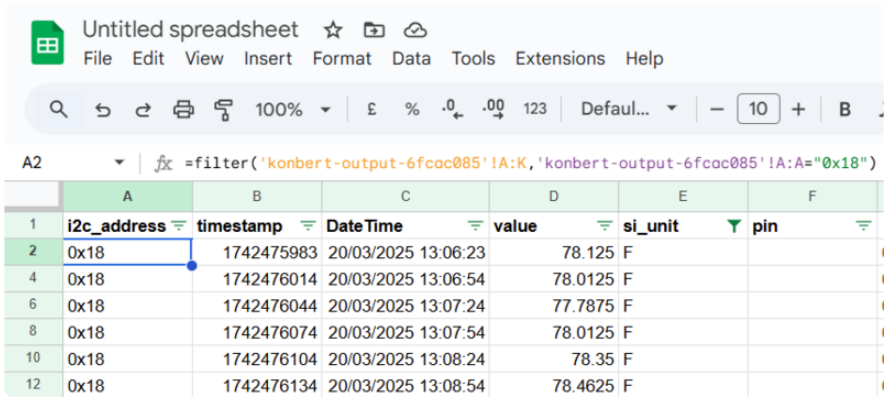
After inserting a new worksheet using the Plus icon in the bottom left corner, or from the menu toolbar, go to the second cell, A2, and enter the beginning of the formula `=FILTER(`. The function takes the data range as the first argument, and then filter criteria for any remaining arguments.

So we want the data worksheet name in single quotes (') followed by an Exclamation mark (!) and then the columns to include (A:F) for the first argument. Then a comma to separate the first and second arguments.

Then the filtering criteria, again the worksheet name in quotes followed by exclamation mark and then the first cell or whole column for `i2c_address` (A:A), and then equals and then the value 0x18 in double quotation marks (") and then if we wanted to filter the `si_unit` too, we would do another criteria but for column E. Finally, a closing bracket finishes the function. It should look similar to this:

```
=filter('konbert-output-6fcac085'!A:J,'konbert-output-6fcac085'!A:A="0x18")
```

Now you'll notice after hitting enter it takes a moment, then all the data from the other sheet appears, but filtered by the criteria in your formula. It has its own filtering / grouping options without affecting the original table of data. Unfortunately it doesn't copy the header row, so you'll want to copy that from the original data source worksheet.



	A	B	C	D	E	F
1	i2c_address	timestamp	DateTime	value	si_unit	pin
2	0x18	1742475983	20/03/2025 13:06:23	78.125	F	
4	0x18	1742476014	20/03/2025 13:06:54	78.0125	F	
6	0x18	1742476044	20/03/2025 13:07:24	77.7875	F	
8	0x18	1742476074	20/03/2025 13:07:54	78.0125	F	
10	0x18	1742476104	20/03/2025 13:08:24	78.35	F	
12	0x18	1742476134	20/03/2025 13:08:54	78.4625	F	

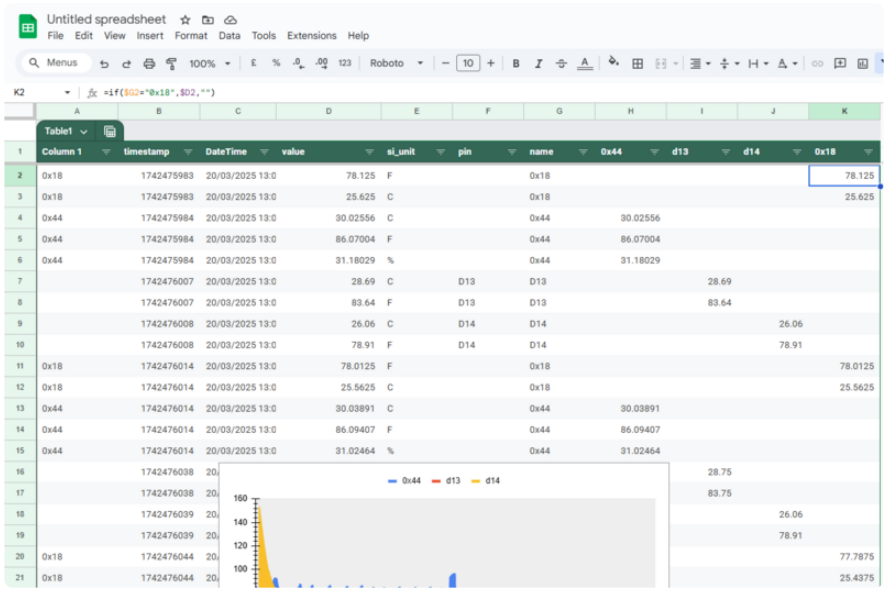
Doing this for each sensor might be wise, but there's also another option...

It's a horizontal world - using horizontal data for combined charts

By adding a new column for each component to the original table, it's pretty easy to have a formula that only prints the value column data (E) if the component name (A) matches and otherwise an empty string ("").

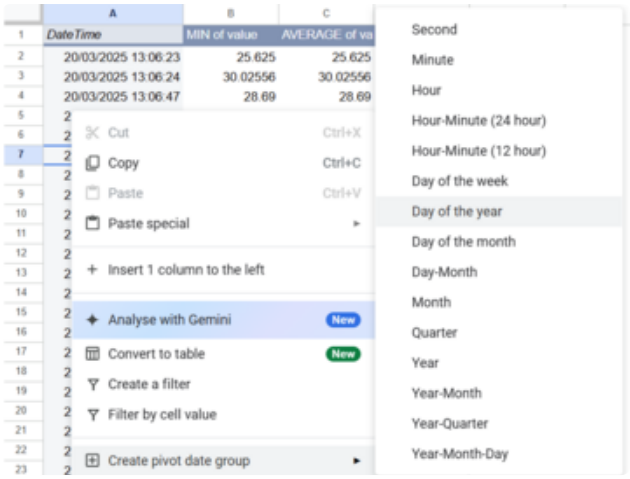
```
=if($A2="0x44",$E2,"")
```

Here the dollar ($) means don't change the column as you move/copy this formula, so $A2 will still refers to the I2C address, and $E2 will point at the value column, when the formula is used to fill in both the 0x18 and 0x44 columns (obviously adjusting the check for ="0x44").



That then gives you the value field copied to the new sensor column for I2C address 0x44 (SHT30). Do similar for the other components, but update the column for pin components if needed. You can now graph a combined dataset by selecting each Series as one of the new columns, instead of using the `values` column.

Taking it further



It's worth mentioning at this point you can convert the data into a table, which gives the ability to save views (filters), along with grouping the data by columns. Most helpfully, you can also use table names as references in the formula bar.

Note that a table also easily converts to a pivot table, which is used to aggregate data, e.g. average/max/min temp grouped by (per) hour. You can find **Pivot table** under the **Insert menu**. Then from your pivot table use the right click / context menu in the date column to choose **Create pivot date group**.

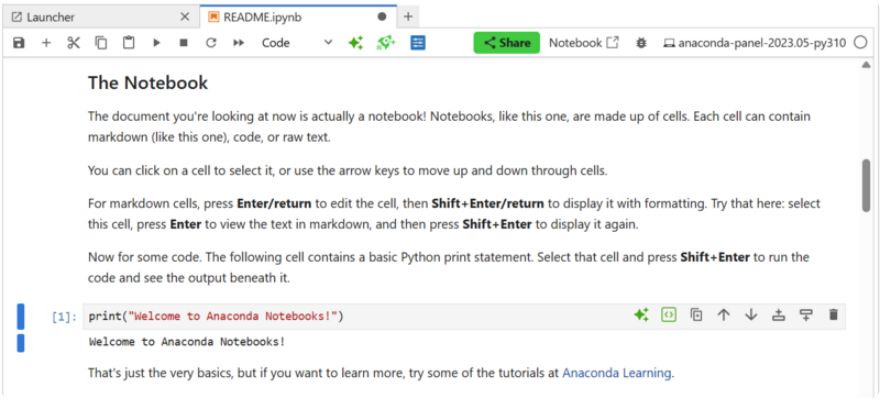
Lastly don't be afraid to use the A.I. assistant built into Google workspaces (or your PC/browser), they all understand spreadsheet formulas and can offer simple solutions that otherwise might take you a while to find.

Got bigger ideas for your data? No problem, it's easy to use it inside a data analysis notebook, which we'll show on the next page, or via the **Google colab platform** (with generous free compute allowance), they even have a handy example notebook (https://adafru.it/1agK), and browser extension (https://adafru.it/1agL), or take your newly organised data off to another site entirely.

Jupyter Notebooks

What is a Jupyter notebook (https://adafru.it/1agM)? It is a collection of "notes" (cells) in Markdown (https://adafru.it/Cdf) format, code, results, and media, commonly used to tweak and display information and for presenting complex data analysis.

It's also the way you interact with a programming backend system (kernel) to run the code from those notebook cells, but setting up a kernel can sometimes turn out to be a pain, so you might be relieved to know you don't have to!



These days it's all about online tools, install nothing manually, and here are a few suggestions for you. But there are also local editor options included, so everyone should find something suitable for their needs, otherwise leave guide feedback, or post questions on the forums (https://adafru.it/dYq) / Discord ().

As a first play, it's worth visiting this page of officially recommended online notebook platforms:

https://jupyter.org/try (https://adafru.it/1agN)

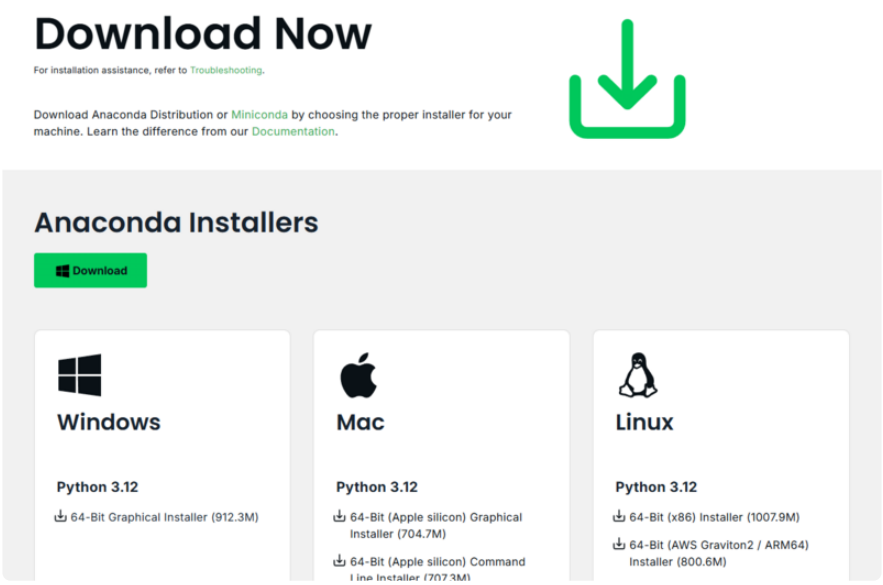
Also note that although it's traditional to use a Python kernel with Jupyter Notebooks (originally called IPython notebooks), many other programming languages are supported via additional kernels.
See the Jupyter page for more info: https://jupyter.org/try#kernels (https://adafru.it/1agN)

Python first, but let's talk about another kind of snake... Anaconda

If you're new to all this, the Jupyter Project recommends installing Anaconda (https://adafru.it/1agO). Anaconda is a collection of data science tools, installable for all platforms (https://adafru.it/1agP), that includes everything required for Jupyter Notebooks (like Python and Jupyter, and the IPython kernel). It also comes with more sophisticated versions like Jupyter Lab, and support for running notebooks in VS Code and other editors.

They also have a cloud platform, if you wish to try that first, at anaconda.cloud (https://adafru.it/1agQ), which has a free plan that comes with a small but useful AI chatbot allowance.

Navigate to the Anaconda downloads page (https://adafru.it/1agp), select to skip registration, then **select your operating system**, and **download the installer including Python 3.12+**.

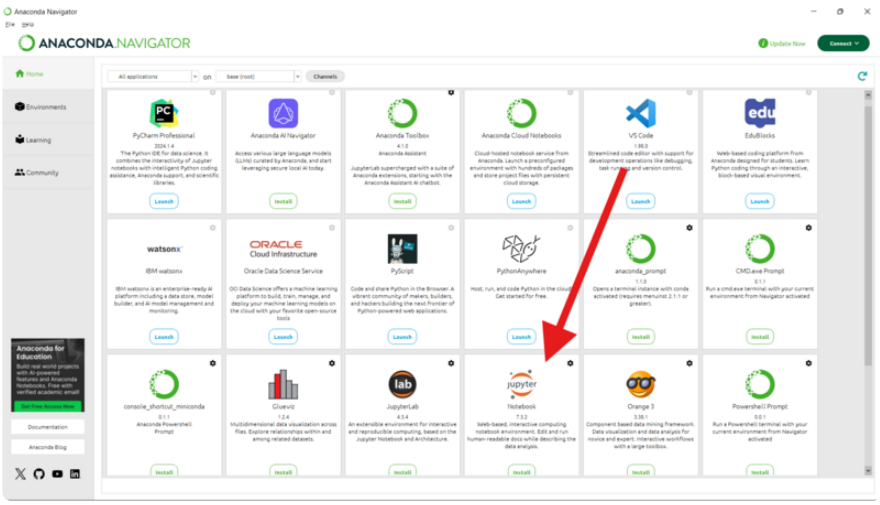


Install the version of Anaconda you downloaded by following the provided instructions.

Launching Jupyter Notebook

Once Anaconda is installed, **open** the Anaconda Navigator Application.

You will be presented with a panel of icons to launch different software options. You will see the option to launch Jupyter Notebooks from the bundled launcher.



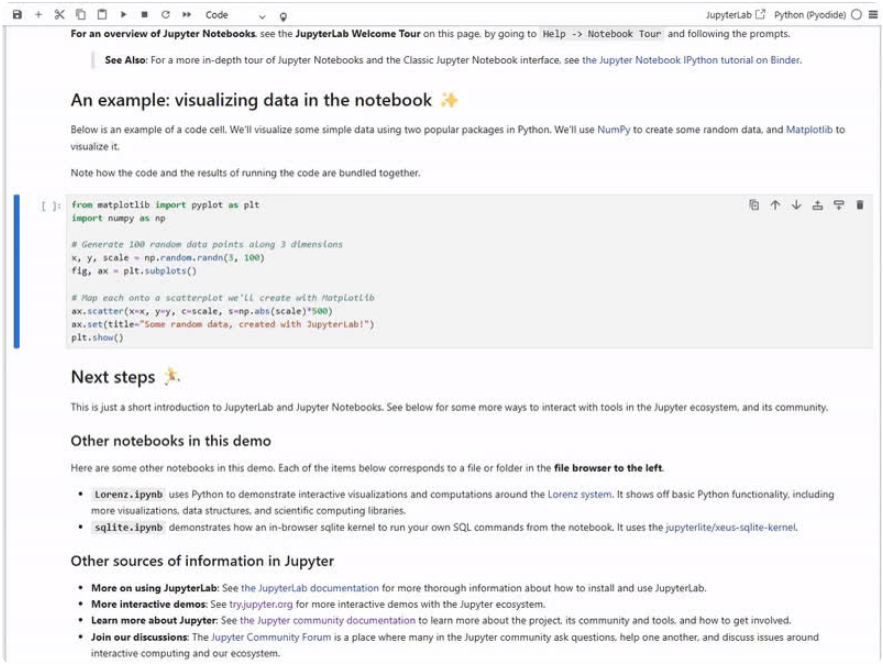
Launching Jupyter notebooks will give you a web page to interact with the notebooks, but notice that there are a lot of other options included in the launcher (some of which come with AI assistance).

Using a Notebook and adding the data

When you first load a notebook, the cells may already have notes (text and images) and code cells, possibly with the results of the code cells presented beneath them. That's because notebooks allow saving results too!

Assuming you have some code that has not been run since being updated (or had the results cleared / not saved), then after selecting the cell row (left hand blue bar indicates the current cell), click the **play icon** or choose **Run Cell(s)** from the menu toolbar which will ask the kernel to run the code and present the results.

Notice the Fast Forward icon which means restart kernel and run all cells. This sometimes has problems so doing the individual cells with the play button is most reliable, try combining with Select all cells.



Great, now that you understand how to execute the code cells, we can begin our data exploration journey.

Offline Data Jupyter Notebook (zip)

https://adafru.it/1agR

Download the provided notebook above, unzip its contents, and open in your Jupyter Notebook platform, Usually this requires uploading the notebook file first if using an online platform, otherwise just open the notebook file (with the `.ipynb` file extension).

First we need to add the data file, usually by uploading a CSV file or equivalent (like a JSONL formatted .log file), but you can also just include anything you like as code (so in this case we'll use dummy data instead of a file), or possibly fetch a file from a URL (if your notebook platform allows that and most do).

You can use your **own data file** here. If you want to follow along with the example data file then jump back to page one of this guide, the Overview page, and download the log file provided in a downloadable zip file (there is also a matching **config.json** for download just above).

After uploading the data, you need to add some code in a cell to import the necessary Python libraries. It's also a good time to install any extra requirements, using the syntax `!pip install X` where X is the library name.

We're going to install some later, but for now we'll import a couple of libraries that come bundled with Jupyter Notebooks to show some of the basics.

The main two number and data manipulation libraries are Numpy and Pandas. It's common to give them a shorter alias instead of their full name, so the import statements in the code cell will look like this:

```
# Comments begin with a hash symbol in Python
```

```
import numpy as np
import pandas as pd
```

Next to get a library to provide a more visual output instead of just printed text, so a graphing or visualisation library is imported (there are many alternatives to matplotlib):

```
import matplotlib
```

Finally, to explore our data... You'll need to appreciate that **pandas** operates on what's called Data Frames (think of them as rows), and is a wizard at manipulating and understanding data. It works in partnership with **numpy**, which provides support for mathematical routines and complexity along with optimised algorithms and its own more efficient data types to represent the standard Python data types. This is why later you will see some of data becomes `np.nan` (NaN = Not a Number) instead of blank values or empty strings ("").

In the provided notebook, there is a section in the code cell at the top that includes some fake data (the lines after `records =` that look like JSON data). The code then checks if those records are commented out, and uses the log file instead (specified by the `log_file_path` variable), or whether to use the fake data. If you don't have any data file then uncomment the records lines to include the sample data.

```
[1]: import pandas as pd
import json
import os
records = None
# Path to your log file, update and comment out `records` beneath
log_file_path = 'log_1742475983.log'  # Replace with your actual file path

## Example dataset, comment out or delete to instead load log_file_path
records = [
    {"i2c_address":"0x18","timestamp":1742475983,"value":78.125,"si_unit":"F"},
    {"i2c_address":"0x18","timestamp":1742475983,"value":25.625,"si_unit":"C"},
    {"i2c_address":"0x44","timestamp":1742475984,"value":30.02556,"si_unit":"C"},
    {"i2c_address":"0x44","timestamp":1742475984,"value":86.07004,"si_unit":"F"},
    {"i2c_address":"0x44","timestamp":1742475984,"value":31.18029,"si_unit":"%"},
    {"timestamp":1742476007,"pin":"D13","value":28.69,"si_unit":"C"},
    {"timestamp":1742476007,"pin":"D13","value":83.64,"si_unit":"F"},
    {"timestamp":1742476008,"pin":"D14","value":26.06,"si_unit":"C"},
    {"timestamp":1742476008,"pin":"D14","value":78.91,"si_unit":"F"},
    {"i2c_address":"0x18","timestamp":1742476014,"value":78.0125,"si_unit":"F"},
    {"i2c_address":"0x18","timestamp":1742476014,"value":25.5625,"si_unit":"C"},
    {"i2c_address":"0x44","timestamp":1742476014,"value":30.03891,"si_unit":"C"},
    {"i2c_address":"0x44","timestamp":1742476014,"value":86.09407,"si_unit":"F"},
    {"i2c_address":"0x44","timestamp":1742476014,"value":31.02464,"si_unit":"%"},
    {"timestamp":1742476038,"pin":"D13","value":28.75,"si_unit":"C"},
    {"timestamp":1742476038,"pin":"D13","value":83.75,"si_unit":"F"},
    {"timestamp":1742476039,"pin":"D14","value":26.06,"si_unit":"C"},
    {"timestamp":1742476039,"pin":"D14","value":78.91,"si_unit":"F"}
]
```

Start by converting the data to a Pandas DataFrame. There are some handy methods to automatically import a CSV using `pd.read_csv(filename)`, or JSON/JSONL file using `pd.read_json(filename, lines=True)`, and it converts them automatically into data frame objects, so take advantage when you can.

It's not too hard to convert things manually, and that will be done for JSONL data to ensure that each line is valid and alert you to which lines are broken, if detected (possibly if the logger's batteries were depleted).

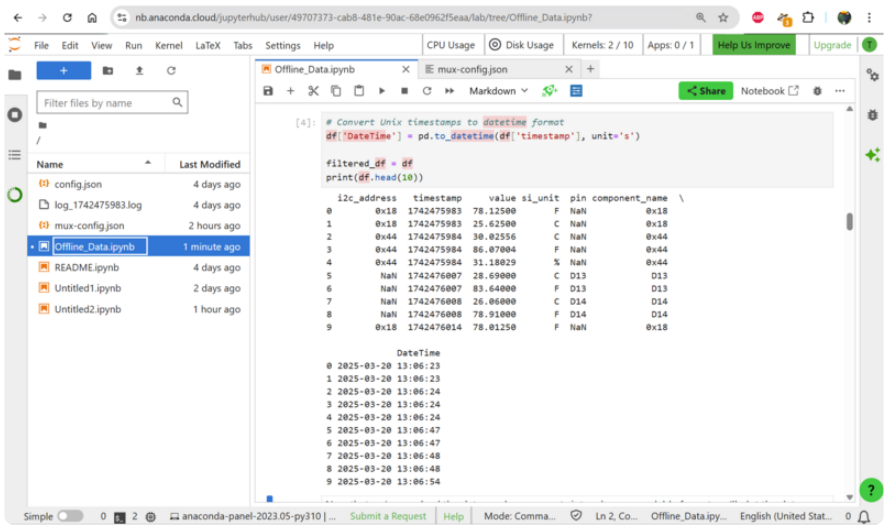
```
# If not using fake data above, then load from file:
if records is None:
    # Read the JSONL file
    records = []
    with open(log_file_path, 'r') as file:
        for (i,line) in enumerate(file):
            try:
                # Parse each line as JSON to avoid any erroneous lines
                record = json.loads(line.strip())
                records.append(record)
            except json.JSONDecodeError:
                # Skip lines that aren't valid JSON
                print(f"Skipping invalid JSON line #{i}: {line[:50]}...")
                continue


# Create DataFrame from the records
df = pd.DataFrame(records)

# Display the first 10 rows to verify
print(df.head(10))
```

```
  i2c_address   timestamp     value si_unit  pin
0        0x18  1742475983  78.12500       F  NaN
1        0x18  1742475983  25.62500       C  NaN
```

Now that the data is in DataFrames, you can start slicing it up or adding extra columns. The whole dataframe collection is treated as a single object at times, where using the indexing notation `dataframe_object[index]` will refer to whole columns.

It can also refer to a collection of testable items at other times, where the syntax is used to filter / test the objects before returning a subset.

Both will be later, but for now just the first syntax will be used to convert the timestamp column, and then to display the first ten rows:



Great, now let's plot some data, but what data to plot... (Surely it cannot be displayed all at once!)

It's worth looking briefly at the code cells above the timestamp conversion, as there the components (sensors) are given a Component Name column ("component_name") which allows you to easily disambiguate (separate) the different data streams by sensor.

If the **config.json** file is provided, then the components will receive full sensor names, otherwise the automatically generated names will be based on the Pin or I2C address information.

You can see in the image above that the component names are just using the automatic versions listing pin name etc, because no **config.json** file was provided for the initial run.

Since the cells were executed, a correct config file has been uploaded, but the cells/ notebook will need to be executed again to update with the new values.

You can use the `component_names` to filter the data, even updating the graphs automatically!

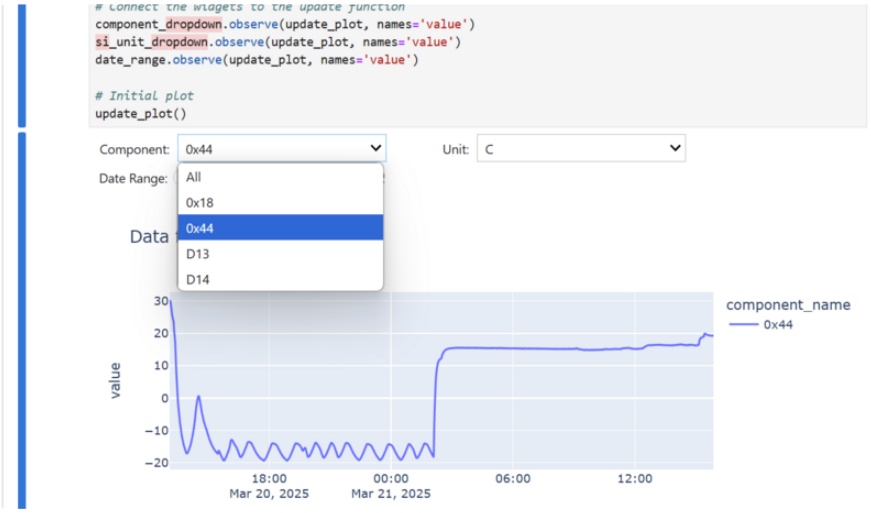
Self-adjusting Charts - Can't we have interactive widgets?

You can! (https://adafru.it/1agM) There is a fantastic documentation page for the IPyWidgets project (now Jupyter-Widgets), which demonstrates the widgets in interactive notebooks using the Pyodide kernel (running Python in the browser), and has many example notebooks to demonstrate different usage.

Use the play icon to work through the notebooks one cell at a time.

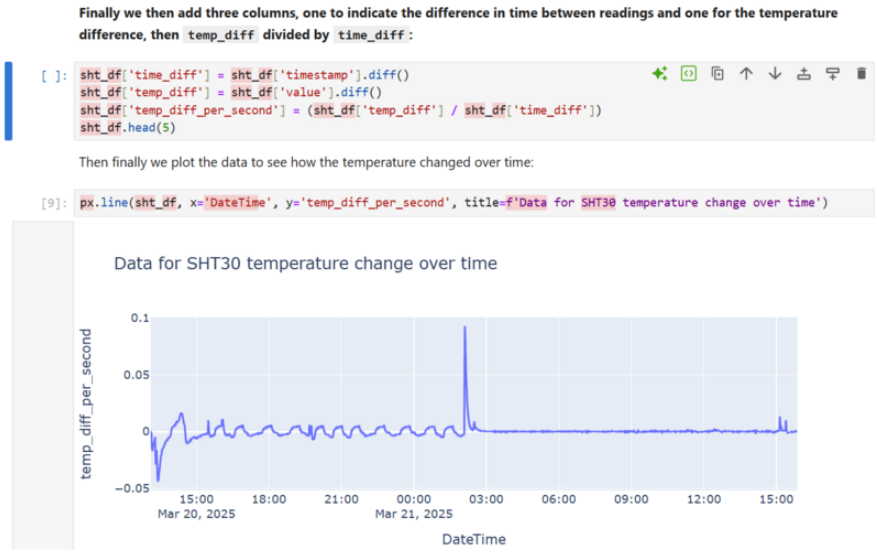
See https://ipywidgets.readthedocs.io/ (https://adafru.it/1agS) (I was impressed to see a Game Controller input widget (https://adafru.it/1agT))

Here the dropdown widget is used to allow selecting the sensor and measurement type / unit, along with a slider to filter the date range if desired:



Later in the notebook, there are a couple of cells that show how to filter a dataset, and then add a few columns for data analysis reasons (to show rate of change).

Have a quick look, and maybe it will inspire you to take your data a little further...



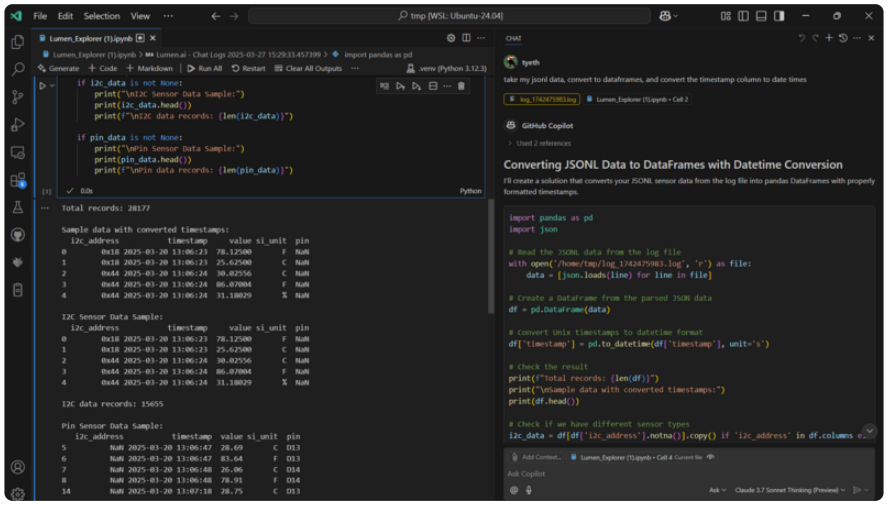
All of these examples were tested using locally installed Jupyter Notebooks, along with on the various try-it-now Jupyter Labs online platforms (using Jupyter-lite), and additionally in the Anaconda Cloud platform.

AI-Assisted Data Analysis and Alternative Editors

Editor preferences vary, and personally I'm more of a Visual Studio Code (VSCode) fan, which does require adding the official Jupyter extension to VSCode before use.

Visual Studio Code is my preferred choice because it supports editing most things flawlessly, and now has a free Copilot chat option that can do remarkably well at following your careful instructions, generating the code for each notebook cell with relative ease.

The key is to include the right (just enough) context by dropping in relevant files along with your written instructions into the Copilot chat panel. Similarly, provide it any error messages to have it attempt to fix them.



There are offline assistants, too, if you want the best of both worlds. They aren't as quick offline so it can be frustrating. I did take a quick look at lumen (https://adafru.it/1agU) which offers an offline supporting chat-based data analysis and visualisation platform. It also supports any model on huggingface.co (https://adafru.it/1agV) which is useful.

To better understand using notebooks in VS Code, I recommend reading the official guide (with helpful video):

https://code.visualstudio.com/docs/datascience/jupyter-notebooks (https://adafru.it/1agW)

Other Online Platforms: Google Colab

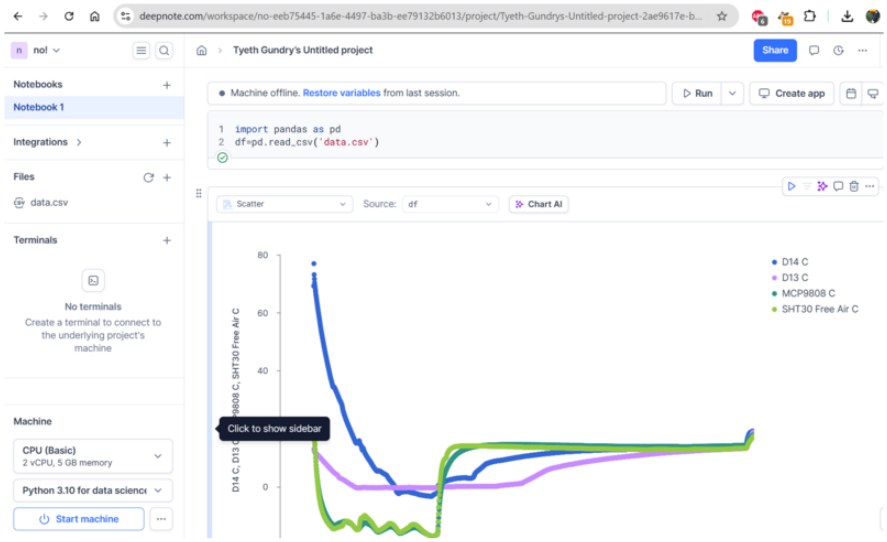
This is perhaps the oldest platform for freely hosted Jupyter Notebooks, Colab (https://adafru.it/1agX) (or 'Colaboratory') was first introduced in 2017 as a research project by Google. It comes with a **generous allowance of compute** per month, storage, memory, and **even GPU capable machines** for more powerful work. That means you can run some complex things that normally require expensive graphics cards, like machine learning / inference jobs, or photogrammetry (meshroom (https://adafru.it/1agY)). They also host a large collection of public notebooks and datasets.

Check out this interactive notebook (https://adafru.it/1agZ) that explains getting started, and links to further info for charts / visualisation (https://adafru.it/1ah0), importing data (https://adafru.it/1agK), and many many more things.

Another online platform - Deepnote.com (https://adafru.it/1ah1)

Worth a mention as I've been using Deepnote since 2021, and it has a great data visualisation and exploration tool built-in.

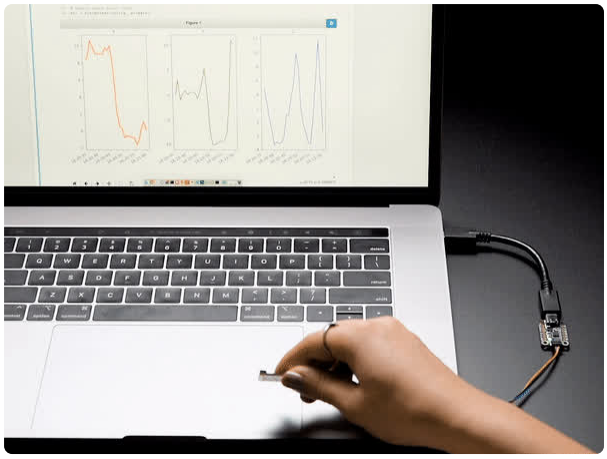
This allows one to explore the data in a dataframe without having to plot it first. Also facilitating pivot-tables (re-stratifying your data), and normal graphs. It additionally comes with an AI assistant that can help push you in a good starting direction with your data.



What about Live Data in notebooks?

Adafruit has a couple of great guides on using Jupyter Notebooks for data analysis, with live data flowing into the notebook from a sensor, and you might be able to use the same hardware you have now to do similar, if desired, (it's just Python code reading USB Serial data into the PC).

Check out the guide linked here which uses an MCP2221 to convert USB to General Purpose Input Output (GPIO) pins along with an I2C interface over Stemma QT, and the other related guide for running a CircuitPython kernel in a Jupyter Notebook (and using a CircuitPlayground Express to provide interactive data):



CircuitPython Libraries and Jupyter Notebook on any Computer with MCP2221
By Brent Rubell
Overview

https://learn.adafruit.com/jupyter-on-any-computer-with-circuitpython-libraries-and-mcp2221/overview

CircuitPython with Jupyter Notebooks
By Brent Rubell
Overview

https://learn.adafruit.com/circuitpython-with-jupyter-notebooks/overview

And that about wraps up this guide. Good luck on your future data exploration journeys!

Downloads

Fritzing Circuit Diagram (with included Frizting parts for all the circuit components):

Fritzing - Offline Pico DS18b20 Datalogger.zip

https://adafru.it/1ah2

Offline Wippersnapper **Config.json** file, with 2x DS18B20 at 30 second read intervals, and an MCP9808 temperature sensor plus an SHT30 for temperature and humidity connected over I2C:

The original file is still attached below, but now the code sample is embedded from the Learn repo on GitHub: https://github.com/adafruit/Adafruit_Learning_System_Guides/blob/main/Plotting_Offline_Data/config.json (https://adafru.it/1ah3)

config.json

https://adafru.it/1ah4

Recorded offline data, a zip containing the logging file, with .log extension, in JSONL format:
(used in examples)

log_1742475983.zip

https://adafru.it/1ah5

Jupyter Notebook file, used in examples for data manipulation and visualisation:
(uses jupyter-widgets and plotly libraries for extra interactive functionality, installed by notebook)

Offline Data Jupyter Notebook (zip)

https://adafru.it/1ah6