Australian Meteorological Association Inc

Monana

THE OFFICIAL PUBLICATION OF THE AUSTRALIAN METEOROLOGICAL ASSOCIATION INC

COVID-19 Easing

May 2020 Edition

No one submitted an article for this edition, so if you had submitted just a ½ page article, you certainly would have won the prize.

The next PWS Edition of the Monana magazine is due to be published on the 31st July 2020. Please submit all PWS items for publication by the 24th July 2020 to: ***monana@ameta.org.au***

But don't forget to also send non-PWS items for inclusion in the more general edition of Monana.

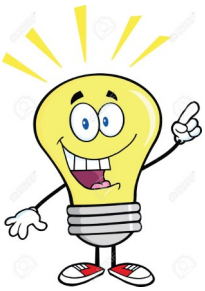
The South Australian Government has commenced its Roadmap to Recovery and hopefully restrictions will continue to ease. However, this will not result in everything returning to normal in the short term as social distancing will still be required in the months to come.

When considering restarting membership meetings, the Committee will need to balance the desire to "return to normal" against welfare concerns for the membership. Quite a few of our members fit into the susceptible age brackets—including myself.

At this stage, meetings of up to 20 people (with 4sqm per person) are being permitted. It is also likely that the success of that relaxation will not be known until later in July when it will become obvious if a spike in infections has occurred.

If things go well with this easing, the AMetA is planning for a return to regular meetings in August with the Annual General Meeting (AGM), unless official advice recommends otherwise.

Unfortunately, the BOM meeting room is unlikely to be available, so the Committee is looking for an alternative location in or close the Adelaide CBD (Central Business District). If a physical meeting is not possible, holding a "virtual meeting" is being investigated.



On the brighter side, social isolation has given me time to consider how I can use all this data that I'm getting from my weather station and its sensors. Apart from the weather readings, the weather station returns information about the state of the batteries in the remote sensors. Even people bored with self-isolation don't keep a constant vigil on these readings to see if there is anything wrong with batteries, but this is a perfect task for a computer. I'm currently playing with a program that will let me know if any battery anomalies are detected. The trouble is, I'm not quite sure what some of the readings mean yet as the product documentation is sketchy.

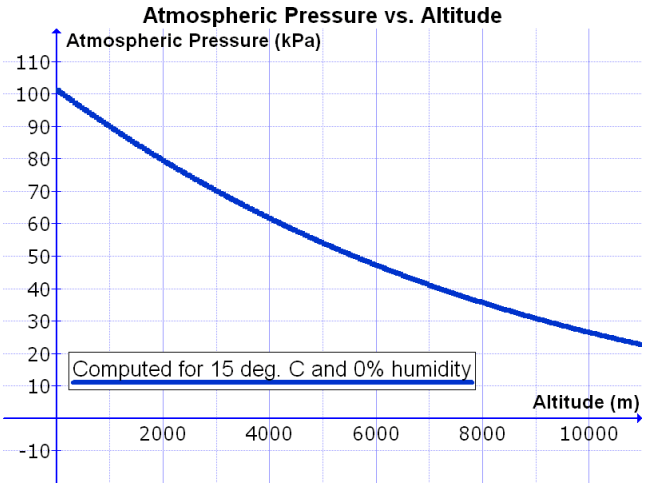
Don't forget that the Monana article competition is running for the rest of the year and is your chance to win either an Arduino or an AMetA published book to pass the hours of your self-isolation and social-distancing.

Even if your ½ to 1 page weather-related story has nothing to do with weather stations, sending your article to the Monana email address means it will end up in the most appropriate edition of the magazine. I'll certainly look forward to reading it. ***Hoping to see YOUR contribution to the Monana magazine soon***.

Keep Happy, Keep Safe,
Mark Little, President, AMetA

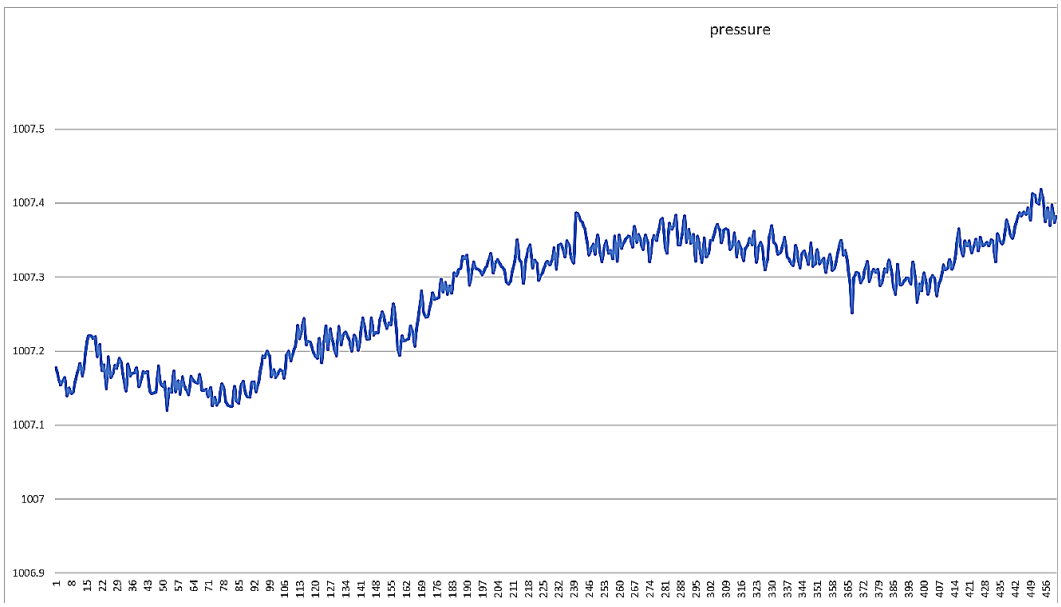
Reading Atmospheric Pressure

By Mark Little



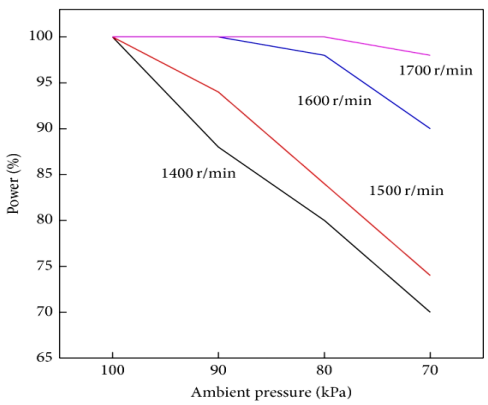
This project for beginners is both something that is in just about every weather station but different, at the same time. It is a sensor that reads atmospheric pressure, but with a twist. It measures the atmospheric pressure in a higher resolution that you would normally expect. Although the BOM normally uses and publishes pressure to a resolution of 0.1 hPa, the BMP180 is accurate to 0.01 hPa. As a bonus it can provide the temperature to 0.1°C as well.

So, if the BOM doesn't normally use atmospheric pressure to a resolution of 0.01 hPa, why would we as enthusiasts be interested in it? The first involves a bit of old-time aviation where pilots used a barometer to determine how high they were flying. It is known that the atmospheric pressure decreases nearly exponentially with height, so if I know the pressure at ground level, I can measure the pressure as I ascend, and calculate my altitude. This requires an better understanding of the pressure at Mean Sea Level to set their altimeter to improve the estimation of the height, because as we know, the atmospheric pressure changes due to the local weather conditions. We can see how it works by using the example chart above. If at a particular location, the pressure at Mean Sea Level (MSL) is 1000 hPa (100 kPa on the chart) and we are reading 90 kPa on our barometer, we would be about 1000 metres above MSL. Of course that could be the height of a local hill, so we still need to know where the actual ground level is as well to be safe. It should be noted that the temperature and the absolute humidity affect the calculations as indicated on the graph above. The barometer takes the temperature variations into account by measuring the temperature. The absolute humidity is not taken into account as its effect is smaller.



So, why else would we want to know the atmospheric pressure to 10 times the normal pressure resolution?

Normal weather predictions work on the large scale relatively slow pressure changes and 0.1 hPa gives enough resolution for that. However, like everything else in the world, nothing is that simple. In addition to these relative slow changes in pressure, the atmosphere is also subject to small changes in pressure known as atmospheric micro-oscillations. These micro-oscillations can have a period of less than a minute to periods of five (5) to ten (10) minutes. The graph above shows that there are low amplitude ripples in the atmospheric pressure. If the resolution was only 0.1 hPa, that ripple would be masked and only the general rise pressure trend would be obvious. To study these trends would require the readings to be stored and then analysed to extract the micro-oscillations.

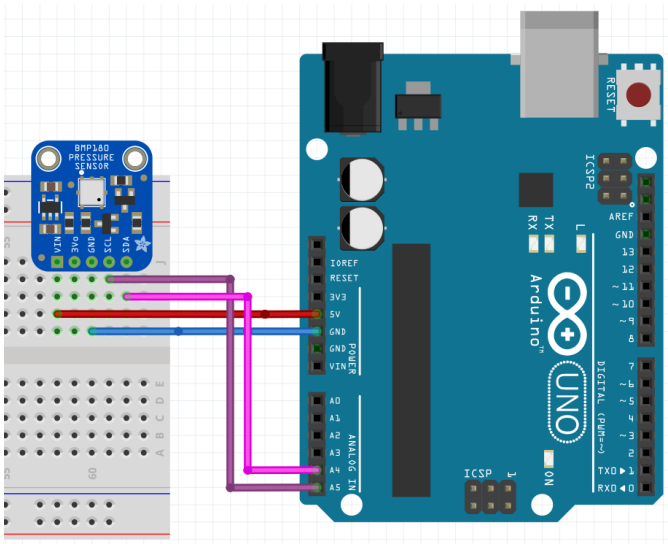


The first step in doing that is to get a sensor with an accuracy of 0.01 hPa or better and read it. Luckily, this can be done easily and cheaply using a chip that was primarily designed for the automotive market. Unlike a plane, the purpose of this sensor is not to make sure that the driver knows how high they are above the ground. The engine management system is another matter. As the

car goes up in altitude such as a trip to the mountains, the atmospheric pressure will drop, meaning there is less oxygen available for combustion, requiring changes to the engine timing and fuel delivery.

The barometric sensor that will be used for this project is the Bosh BMP180 developed for the automotive market. A number of companies have boards that carry this sensor. Prices can be variable, so it pays to shop around. The photo of the BMP180 board on the left is sold by Core Electronics, while the diagram shows the connections for an Adafruit board, also sold by Core Electronics (not an endorsement of them, by the way), but each board will have the at least same four pins, so it is easy to connect up any of them.

Since this is an introductory project, we aren't going to do anything fancy with the readings from the sensor, but the wider AMetA project to extend Personal Weather Stations and the Analogue Barometer project do support this device.

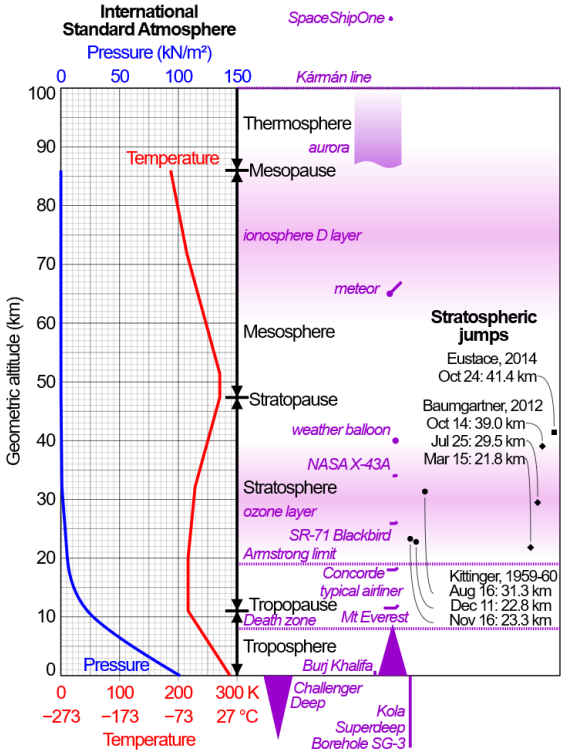


Arduino UNO	Adafruit BMP180
5V—+5V output pin	**VIN**—Supply input pin
GND—Ground (Earth) pin	**GND**—Ground (Earth) pin
A5—Multi-purpose pin set as SCL signal by program	**SCL**—Serial Clock Input Pin
A4—Multi-purpose pin set as SDA signal by program	**SDA**—Serial Data Output Pin

What will you need for this project? It requires a computer (Windows, Linux or Mac) that runs the Arduino Integrated Development Environment (IDE), a USB cable, a BMP180 sensor, an Arduino UNO, four wires and a breadboard. If you started dusting off the crumbs from the breadboard in the kitchen, you have the wrong idea of the type of breadboard we will be using.

In case the image isn't clear enough, the wires are as follows for connecting the Adafruit board to the Arduino—check the datasheet of the device you purchase to be sure.

The first thing to do is to install the Arduino IDE on your computer, then connect the Arduino to your computer using a suitable USB cable. This cable not only communicated with the Arduino, it supplies power to the Arduino from the computer. There isn't enough space to step though the process of using the Arduino IDE, that would have normally be done at a meeting with face-to-face contact (a current COVID19 no-no). However, here is a 35 minute Facebook presentation that will give you the basics of using the Arduino IDE. The presenter is a bit over the top for me, but he covers what is needed.



To read the data from the BMP180 sensor, you need to install drivers which do the low level control of the device for you. Fortunately, it is easy to install these drivers onto your computer for use. Taking the easy way out again , I have opted to simply point you to a Facebook presentation that not only shows you how to install these device drivers, but also tells you where to load the example source code and run it. While you are watching this video, take time to notice that the presenter ponders why the pressure and height is not as stable as he assumed they would be. It could be that the sensor is noisy, or perhaps the previous page provides a clue as to why small variations in pressure (and hence height) can be seen.

You should also remember that if you run this program to estimate the height of your station, you need to get the current Mean Sea Level pressure where you are and insert that value into your source code instead of

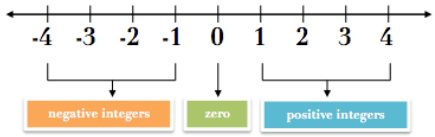
"SENSORS_PRESSURE_SEALEVELHPA". The default value is the Mean Sea Level (MSL) pressure used in the "International Standard Atmosphere" and is 1013.25 hPa.

After you have watched the pressure, temperature and altitude scrolling down the monitor page for a while, it is time to think about what we can quickly and easily do to make some use of this data. The easiest way to start analysing the data from the sensor is to put into a spreadsheet so that it can be examined in a tabular or graphical form. The easiest way to do this is to get the data in the text-based CSV (Comma Separated Values) format. The line below gives an example of the sensor data in CSV format (not actual readings, just a format example).

1,1010.34,27.3,9.5

The first value "1" is a reading counter, the second value "1010.34" is the atmospheric pressure in hectopascals, the third "27.3" is the temperature in Celsius, and finally the height in metres.

Except for the counter, the information is already being displayed, so it is a simple matter of changing the way that information is displayed to get it into the CSV format. The counter is not currently available in the program so it has to be added. Here we get into a bit of programming. The counter is simply a number that gets increased by one each time a CSV line is created. In the computer language being used C++, there are different types of numbers.



In this example, we will be using an "integer" which simply means that we can only use whole numbers. Unlike a pencil and paper, numbers stored in a computer are in a "bucket" that can only contain a limited quantity of numbers. For example, a small number stored in one byte of memory can have no more than 256 different values. If that includes numbers below zero, that range is –128 to +127. If we don't use negative numbers, the 256 values can be 0 to 255. The standard Arduino UNO integer is held in two bytes, so an unsigned integer can contain the values 0 to 65,535 which is big enough for this example. If that was not the case, there are larger integers, called a "long".

In the program source, we need to declare that we want a variable called "counter", and we need to say what type we want, so in this case it would be "unsigned int counter;". The semicolon (;) tells the computer that this is the end of the instruction. If we have a really big command or we want pretty formatting, we could put in on multiple lines because it will be seen as one instruction until it reaches the semi-colon.

```
1
2
3  var global = 10;
4
5  function fun() {
6
7      var local = 5;
8
9  }
10
```

global variable

local variable

In the C++ language is that if a variable like counter is declared inside a block of code bounded by a "{" and its associated "}", it only exists during the time that part of the code is being run. This is called a "local variable". One that is declared outside of any brackets at all will exist for the entire time the program is running and is called a "global" value. As the Code Academy site explains, whether a variable is global or local is called its "scope". We want the counter's scope to be global.

```
Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);

/* Output Reading Counter */
unsigned int counter = 0;
```

Making a global variable is simply a matter of adding it under the declaration of bmp at line 30 of the source code. Although the value will default to zero, it is good practice to explicitly set it to 0. Depending on what you are doing, it could be any valid unsigned integer value.

```
/* Output the Counter and automatically increment its value */
Serial.print( counter++ );
Serial.print( ',' );

/* Output the atmospheric pressue in hPa */
Serial.print( event.pressure );
Serial.print( ',' );

/* Output the Temperature in Celsius */
float temperature;
bmp.getTemperature(&temperature);
Serial.print( temperature );
Serial.print( ',' );

/* Output the Estimated Height in metres */
/* NOTE: For best results, use the current MSL pressure */
/* from your weather station or a nearby BOM station    */
float seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;
Serial.println(bmp.pressureToAltitude(seaLevelPressure,
                                      event.pressure));
```

Now we need to change the output format, so delete everything from line 91 to 128 inclusive. This is after "if (even. Pressure){" and before "}else{", just in case your file is slightly different. Now add the lines to the left in their place.

Notice the term "counter++". Adding the ++ after the name means "Use the number and then add 1 to the value of the number". If it was "--", 1 would be subtracted after use. Putting the "++" or "--" before the name would mean "add (or subtract) 1 BEFORE using the number".

"float" is a number type that means "floating point". It is a number type that can store fractions, as well as very small and very

small numbers by including an exponent part in the value.

Probably the hardest thing for some neophytes to grasp is the "&temperature" value. The "&" means that the program is looking for the address of the memory location holding the temperature variable, rather than the value of the temperature. Normally when a variable like temperature is passed to a function, it is the only value that is passed. When just the value is passed, the function cannot alter the original value, because it doesn't know where it is stored. By passing a reference (the address of where it is stored), the other function can update the value held in the temperature's storage location.

Finally, the last point to cover in this article is the difference between "Serial.print" and Serial.println". The print command will print what is contained within its brackets as will the println command, but will cause the display to go to the start of a new line when it is finished.

These comments don't explain all of the lines of code, but I have found that the best way to learn about programming is to jump right in and learn the bits you need to know as you go. After all, most of us are using this code to get the meteorological readings they can provide, rather than just programming for programming's sake.

After compiling the modified code and uploading it, have a look at the serial monitor, you should see the start-up message as before, but the readings should be displayed in the format discussed above. You can now use "cut and paste" to capture the readings from the serial monitor. Each reading will be just over 1 second apart.

Now, if you have trouble getting your modifications to work, you can get a working copy here(*). If you can't get your software working, you can drop me an email outlining your problems and I will see if I can help.

(*) Access to this members and guests only area requires the username ***ameta*** and the password ***ELpyxUXl*** (only valid for 2020).

Why Have A Sky Camera?

By Mark Little

I often get asked why I have a sky camera, so I show them some images and time-lapse movies. When people first look at an the images, they sometimes forget that wide angle lens distorts the image and think that the pole holding my weather station is way off vertical. In return, I ask them if they also think that my roof is as curved as it seems in the image. ☺

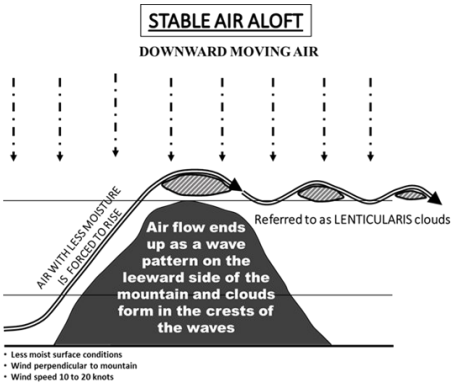
They also quickly find that there is a bit of a darker patch to the left of the weather station pole and wonder what it is. That spot is a defect and what got me the camera at a very cheap price and it doesn't really affect what I want the camera for. Sometimes people also tell me that it is a waste because the sky is black for half the time (on average). As we will see later, because the sky is dark does not mean that the camera doesn't see anything.

Quite a few weather sites allow images to be uploaded, but due to sheer number of sites that upload images, to save space the images are resized to a quite small image and if they created a time lapse movie, it looks like an image per hour which makes it very hard to see what is going on. On the other hand, I save even the movie images at full camera resolution (1920 x 1080 pixels) with the time between frames is one minute, but 100 days takes up 19 GBytes of storage (69GB per year).

The examples in this article use my time lapse movie for the 10th April 2020. It will be slow to load if everyone is reading their magazine at the same time, because the website is just running on my home computer. The movie will initially start out in black & white as it is in infrared mode in the morning, go to colour during the day, then back to infrared mode at night.

Use your viewer to move the video to 01:55 (07:55:01 local time on the top of the picture) and let it run. Look at the sky near the end of the shed wall in the lower right hand side of the picture. You will see the clouds begin to move across the screen towards the middle left of the picture. By 01:57 (07:57:01 local time), you should start to see a pattern forming in the cloud as it moves. Since the camera is pointing South, the clouds are coming from the South West. Looking at the database that holds the weather station readings, gives a surface wind of 218° (SW) so this is consistent with how a low cloud might be expected to move.

As the cloud advances, it becomes obvious that the cloud ripples are travelling with the cloud rather than being in the same location as the cloud passes. I'm not a meteorologist, so my interpretation of what this means may be way off-base, but as I understand it, there are two common ways to get waves in the clouds.



The first is that air encounters some sort of barrier like a mountain or hill and causes vertical waves. When the air in the wave is lifted to where air temperature drops below the dewpoint, a cloud forms. As the air drops and the air temperature rises above the dewpoint, the cloud evaporates. This can give rises to the smooth lenticular (lens-shaped) clouds that are in a stable position. The shape and longevity of these clouds can give rise to their identification as a UFO (Unidentified Flying Object). Click on the images for more information.

The other way is where two bodies of fluid move across each other and at the boundary line between them, waves start to form as the bodies moving at different speeds and densities interact. This produces the same vertical effect as the before, but the turbulence is not anchored to a fixed obstruction, so the waves move with the air flow. The image to the right is part of a 02:09 (08:08 local time) frame in the time-lapse movie hyperlinked on the previous page and meets the criteria for this sort of cloud where the structure known as Kelvin-Helmholtz waves or billow clouds.

If you are looking for more information on identifying clouds, the Bureau Of Meteorology has some information on cloud types, or you can go the World Meteorological Organization International Cloud Atlas.

Well, as you can see from that diversion into deciding what clouds were being seen by my sky cameras, scanning through the time-lapse movies can bring up some other interesting observations, well at least during the day. But is the darkness of night really wasted? Well, no it isn't. Let's skip forward in the time lapse movie to 13:10 (19:10 local time). In April, it certainly dark by that time, but looking at the image, clouds are still visible.

As you will notice, the image has gone from colour to black & white. This is because the camera has changed to Infrared mode as it was originally designed to be a security camera. The camera emits infrared light. The metal of the shed wall and the pole on the left reflects Infrared so they are bright in the image.

Clearly though the clouds are too far away to be reflecting the light from a few LEDs, they must either be emitting light themselves, or reflecting it from a much larger source. When you look at the clouds in the city or looking at the clouds over a township from the country and you often see the orange glow of the street lights reflected from the low clouds. However, because the camera is in infra-red mode, the clouds are probably visible because heat radiated from the city/earth is being absorbed by the clouds and re-radiated as infra-red radiation that the camera can detect. This means that using the time-lapse movie you can still see the direction the low clouds are travelling even at night.

At 13:14 (19:14 local time) in the movie, two vertical dots can be seen on the left hand side of the image. Although it is a cloudy night, if you let the movie run, you will see that those dots move up in an arc to the right. I haven't worked out what the stars are, but at some day I'm going to figure out the point around which they are rotating and that will be the South Celestial Pole.

From 13:30 (19:30 local time) in the time lapse movie, a strange bright horizontal light can be seen apparently coming out of the weather station. This did confuse me at first, but ultimately it wasn't a death ray or some sort of weird electrical discharge, it was just an insect with wings that are infrared reflective. Look at the wavy pattern and you can see the beating of the wings.

As I sat writing this, there was a thunderstorm outside and I hoped to catch a picture of a lightning bolt. Sadly, I didn't, but it made me want to try detecting lightning with a lightning detector sensor, but we don't really get that many storms.

Data Analysis with SQL (Part 1)

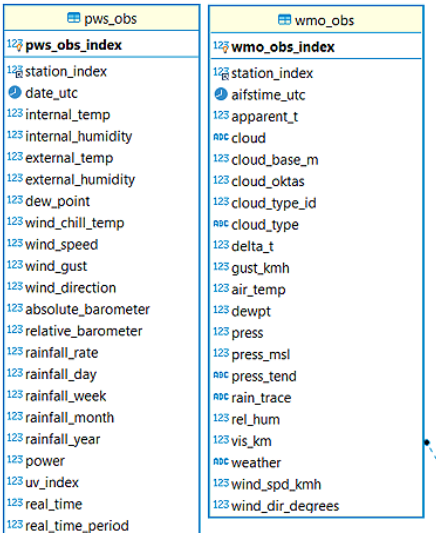
By Mark Little

First of all, what is "SQL"? It stands for "Structured Query Language" and it is used to insert, update and extract information from a relational database. The information from my weather station is stored in a database called "MySQL", actually, it is a public domain version called "MariaDB", but for the purpose of this article, they are equivalent. Examples given in this and subsequent articles are done using the public domain "DBeaver" Universal Database Manager that can be used with all popular databases, but the SQL statements will work in DBeaver or a computer program. If you are reading this sometime after 2020, the DBeaver mascot is currently wearing a mask for COVID-19 protection, not because it is a bandit. ☺

To make it easy for people to follow these examples, the data produced by the database queries is saved as a CSV file that may be imported into Excel to be plotted. Later articles will show how that data can be displayed on a web page, if required.

Any financial member of the AMetA can store the data from their weather station into the same database if they wish, but there are no service guarantees as database is just running on my home computer although it is available via the Internet—more on that in a later article.

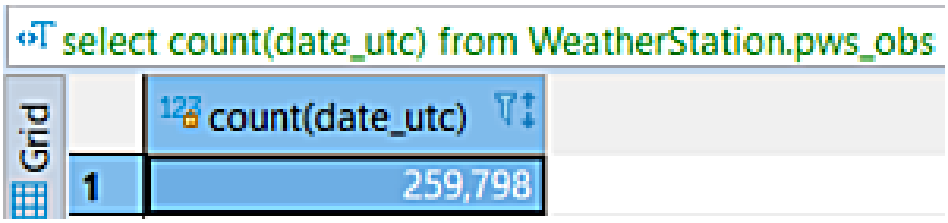
The first thing to understand the basics of how the data is stored in the database (without diving deeply into the relational bit at this stage). A database is normally divided into tables of related data. For example, in my WeatherStation database, there is a table that holds the data from my weather station (and potentially from the weather station of other members). There is also another table that holds data from the BOM's weather stations that is periodically downloaded from the BOM website. There are other tables, but we will ignore most of them for now.



The image to the left is a portion of the WeatherStation Database that holds all of the weather-related data. The pws_obs table contains the observations from my weather stations—only a portion of the table is shown, and the other table is the observations from the Bureau of Meteorology stations around my location. Let's start with an easy query. Let's find out how many observations there are in the pws_obs table.

Each item is known as a "field" and if this was a spreadsheet, they would correspond to the columns in a spreadsheet. A record in the database would correspond to a row in the spreadsheet. That is, each time an observation is inserted into the database, a new record is inserted and each field is filled with the reading corresponding to the field. This is like a spreadsheet where a row's columns would be filled with data associated with a column.

Let's start with a simple database query using DBeaver. The query and the result are shown below. Let's unpack the query to see what it is asking the database to provide. "select" is the verb that tells the database what to do. Other common verbs that should be select explanatory are "insert", "update" and "delete" - they will be dealt with in a later article. The preposition "from" is also rather self-explanatory and identifies where whatever is being selected comes from. In this case it is Weatherstation.pws_obs. This identifies that it is from the database *WeatherStation* and the database table *pws_obs*.



What is being selected in count (date_utc). What this means is that the databases needs to count the number of entries in the date_utc field and return that number. Since each observation records the time at which it occurred, this means that the database contains more than a quarter million observations. This may seem like a very large number, but if you consider that if the weather station sends out a reading every minute, that is 259,798 minutes since recording started. That is about 4,330 hours, or about 180 days. That is less than six months of records since a year is 365 or 366 days long! My weather station spits out a reading much faster than once a minute so that quarter million reading comes up much

quicker.

Let's say I want to plot a graph of the external temperature at the time at which the observation was taken. I need to get the date/time of the reading and the temperature at that time.

```
select date_utc, external_temp from WeatherStation.pws_obs
```

Results | Result - 2 | Result - 3 | pws_obs - 4

select date_utc, external_temp from WeatherStation.pws_ot | Enter a SQL expressio

	date_utc	external_temp
1	2019-12-14 17:02:08	12.7
2	2019-12-14 17:02:24	12.7
3	2019-12-14 17:02:40	12.7
4	2019-12-14 17:02:50	12.7

This time, you don't get to see all the results on screen, because there are over a quarter of a million results—a bit too many to fit on the screen. However you can see that the readings are coming at rate of about 5 per minute! That means that the "less than six months to get to a quarter million readings" reduces down to less than 5 weeks at this rate. This can be a real problem over a long period, especially if there are over a quarter of a million weather stations providing observations at that rate (Weather Underground). That would be over 657,450,000,000 readings per year. This sort of thing is called BIG DATA for good reason!

To only ask for the latest data, I will put in the starting datetime myself. Using a start and end range is possible, but generating the query would be a bit more complex than required for this explanation. Also beware because the observations are recorded using UTC, rather than local time, if you are looking for data since a local datetime, you will get the wrong data—we will deal with calculating the correct datetime for queries using the local datetime in a later article.

```
select date_utc, external_temp from WeatherStation.pws_obs where date_utc >= '2020-04-02 13:35:58';
```

pws_obs

select date_utc, external_temp from WeatherStation.pws_ot | Enter a SQL expression to filter results (use Ctrl+Space)

	date_utc	external_temp
1	2020-04-02 13:35:58	17.778
2	2020-04-02 13:37:14	17.778
3	2020-04-02 13:38:42	17.778
4	2020-04-02 13:39:58	17.778
5	2020-04-02 13:41:14	17.722
6	2020-04-02 13:42:42	17.5
7	2020-04-02 13:43:58	17.389
8	2020-04-02 13:45:14	17.278
9	2020-04-02 13:46:42	17.222
10	2020-04-02 13:47:58	17.222

As you can see by comparing the results with the previous results that there are no results before "2020-04-02 13:35:58" which is 01:35:58 pm on the 2$_{nd}$ April 2020 (UTC).

Let's try something a bit different. Let's say to reduce the amount of observation data that needs to be archived, we decide that we only want to record the average temperature, maximum temperature and the minimum temperature for each day.

```
select date(date_utc) obs_date, avg(external_temp) avg_temp,
max(external_temp) max_temp, min(external_temp) min_temp from WeatherStation.pws_obs
where date(date_utc) >= '2020-02-01' group by date(date_utc);
```

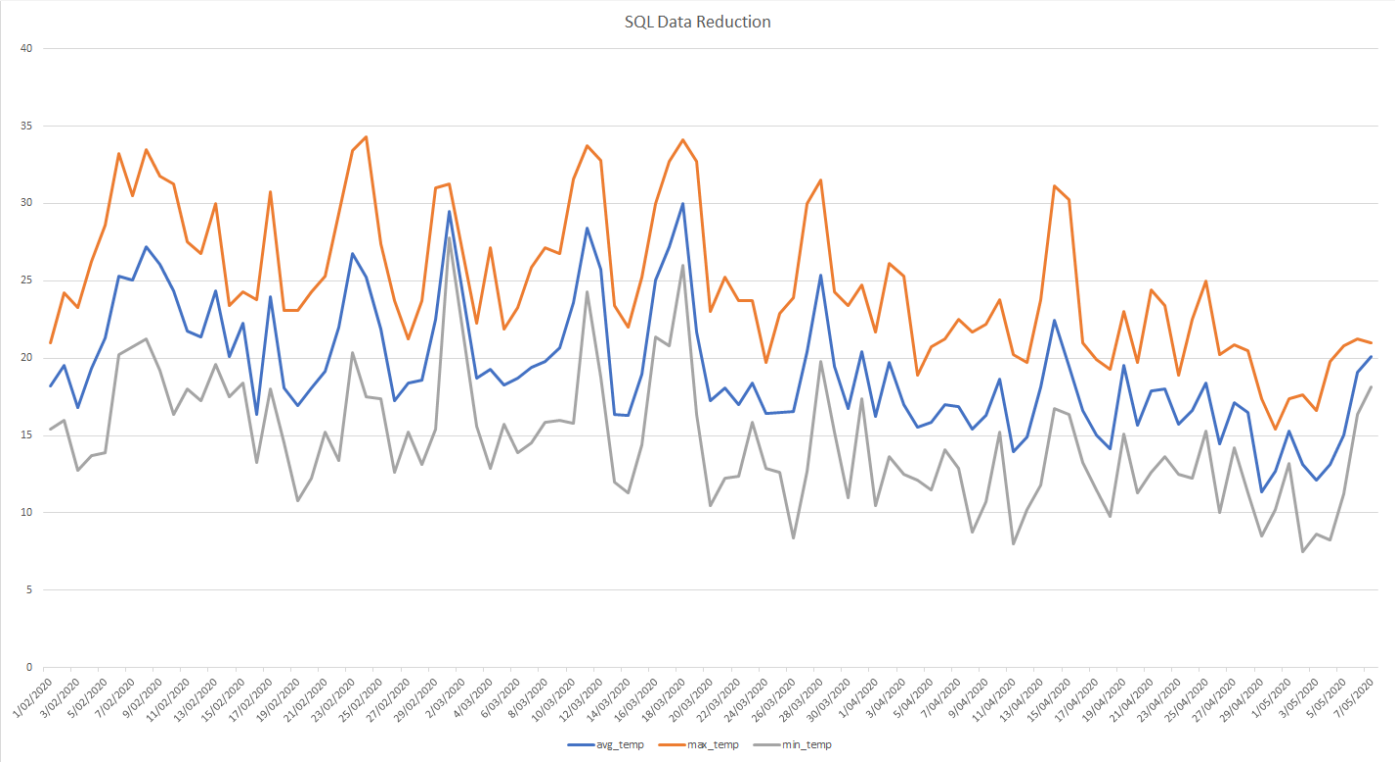
Result

select date(date_utc) obs_date, avg(external_temp) avg_ter | Enter a SQL expression to filter results (use Ctrl+Space)

	obs_date	avg_temp	max_temp	min_temp
1	2020-02-01	18.2178779548	21	15.3888998032
2	2020-02-02	19.5126542974	24.2222003937	16
3	2020-02-03	16.8215969779	23.2777996063	12.7777996063
4	2020-02-04	19.3564772039	26.2777996063	13.7222003937
5	2020-02-05	21.2950844611	28.6110992432	13.8888998032
6	2020-02-06	25.3339054292	33.2221984863	20.2222003937
7	2020-02-07	25.0325652689	30.5	20.7222003937
8	2020-02-08	27.1846902276	33.5	21.2777996063

There are a few differences in the query (on the next page) starting with the date() function. This is used to remove the time component of the date_utc. This means that it now refers to all times of that day, not just the hour, minute and second of that day. The use of the functions avg(), max() and min() is used to get the average value, the maximum value and the minimum values respectively. The other thing to note is that each function, there is a name and that is the name used for the result of that field. As you can see, the column is not identified as date(utc_date), but rather as obs_date. This allows the use of more understandable names in the results. The other new clause is "group by date(date_utc)". If that clause was removed, there would be only one row in the result. The obs_date would be the first date read and the average, maximum and minimum values would refer to the average, maximum and minimum of all results selected, not just for a single day.

Now if we export those results into a CSV file, we can use Excel to quickly plot the results.



This daily data reduction method takes approximately 7,200 temperature readings taken per day into three reading, the average, the maximum and the minimum—a rather significant saving in space, but all of the information about weather events like temperatures changes during passing fronts that may have occurred during that day is lost. A change in the query can be used to get the three readings some other period such as every 5 minutes to get more details at the expense of more storage.

How data reduction is carried out depends on what sort of historical information needs to be preserved and that will vary from project to project. What sort of information do you think that you will want to be able to retrieve in five (5) years time. You may decide that you will archive all of the data for a year to an archive disk, but only keep the daily max, min on average online. Whatever you do is up to you.

Well, that's enough for now. In the next edition, we will look at how to combine results from different tables and how to sort data in various ways. Also to be discussed will be how to measure the average direction of the wind. There are different methods that vary based on what exactly is meant. That is, does the average wind direction calculation need to take into account the speed of the wind, or is the wind speed ignored. Until then think about how you would like to analyse your data and try to figure out how you would do it.

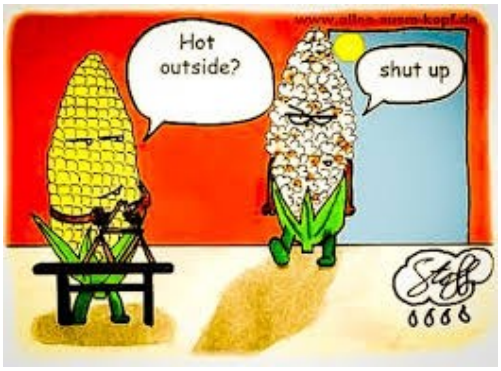
Who Wrote That Article?

By Mark Little

While my name in on quite a few articles, this can be misleading. Many articles are really a silent collaboration between quite a few people. For example, the barometer article (and the others) incorporate corrections and improvements from Beth, Dave and Bruce, amongst others. Apart from providing recognition to those who helped, it shows that you don't need to be an meteorological guru or an expert writer to contribute to your magazine—we will help you if you need a bit of help with your grammar and composition. It is you and your weather photos, projects and experiences we are interested in, not your grammar.

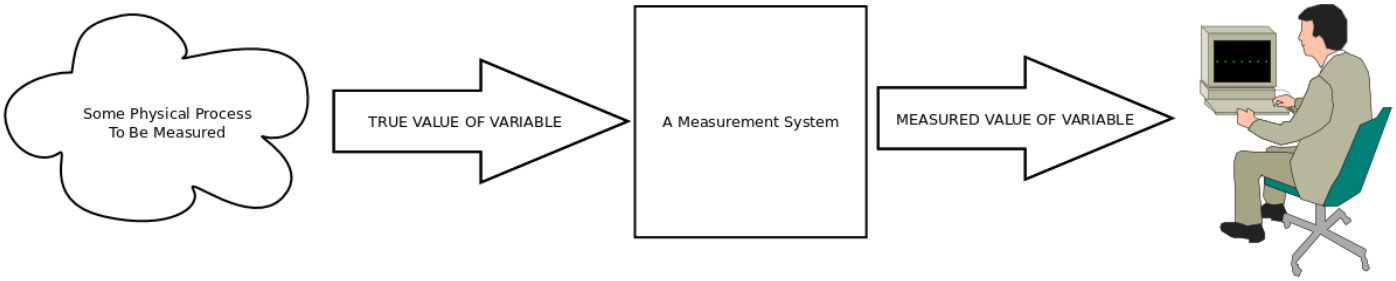
Errors & Inaccuracies

By Mark Little



When we are looking at our weather stations, the observations they produce and the results of our analysis., we need to think about "Nothing's Perfect" and "Never Let Perfect Get In the Way Of Good Enough". What that means is that nothing that we do will results in a 100% perfect representation of the atmosphere, but it also means that we should not waste our time trying to achieve that impossible task of perfect results when the purpose to which we put that information does not require that level of accuracy. For example, If I get asked "Is it hot outside?", I do not need to know the exact temperature. Just knowing that it is over 45°C would be good enough to give the absolutely correct answer of "Yes" (assuming the person asking is a normal human being).

The purpose of this series is to look at the entire process of measurement, reading the results, analysing and displaying the results, think about what is limiting our results and how they can be improved to meet our requirements and when it is time to question whether it is sensible to keep trying for improvements. Don't be put off by the fact that there will be quite a few formulas flung around. If that is not your thing, you can skip them and look at the practical examples that will be used to illustrate what the formulas are telling us to expect. But first let's make a basic model of what we are looking at.

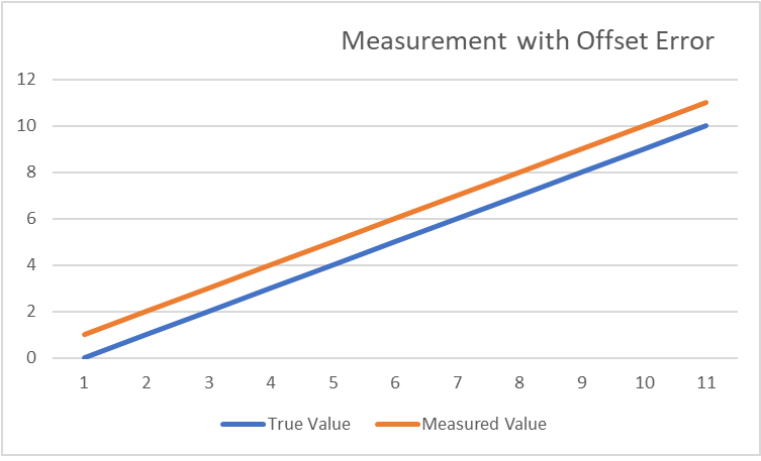


With all of our systems, there is something that we want to measure with some form of measurement system (sensor) and it gives a result that has some deviation from the real value that is caused by the measurement system. In a more shorthand way:

$$Variable_{as\ read} = Variable_{true\ value} + Error_{measurement}$$

Which means that the value that we read from measurement system is the real value plus whatever error the measurement system added. If your brain did not just explode, you have survived the first formula.

So now let's look at the types of error that we can expect to see. The first is called an "offset error" and as the name suggests the error means that each reading is offset by some constant value. Looking at my HP2550 weather station, the temperature, humidity, barometer and wind direction have an offset adjustment that can be used to correct any errors in the readings for those sensors. It is the nature of the sensor that any error is likely to be an offset error, if the sensor has not failed.



As you can see from this graph, the offset error can be easily removed by simply subtracting the value of the offset from the measured reading. If you *know* that this is an offset error, you can simply get one point where you know the real value and the value as read to calculate the offset that needs to be removed for any reading because the error is a constant that obeys this formula.

$$Variable_{as\ read} = Variable_{true\ value} + Error_{offset}$$

If you do not know what type of error this is, a much more complex analysis is required to determine the type of error that is involved, so that you can calculate the offset adjustment value.

Removing the error to get the true value is called "Calibration", so in this case calibration just consists of subtracting 1 from the value read to get the true value. If your station can't do it for you, you can make the adjustment in you head.

The next common type of measurement error is called a "Scale Error" and is common in sensors like the solar radiation sensor and the wind speed. Consider the solar radiation sensor that is sitting behind a clear plastic window to keep the weather off it. As the weather station ages, the plastic window will start to degrade and become less transparent. If we say that half of the light does not reach the sensor, then the reading will always be half what is should be reading. If we "scale" the readings by multiplying by 2, then we will have calibrated out the error resulting from the less than transparent window. Using the same format as formula as before, we would have a formula as below. Of course, the value "$Error_{scale}$" would be negative as the error causes the reading to be lower than it should be.

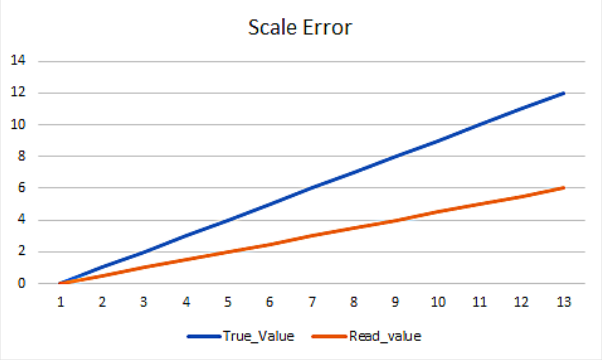
$$Variable_{as\ read} = Variable_{true\ value} + Error_{scale} * Variable_{true\ value}$$

In this case, we can simplify the equation by realising that group the values on the right hand side of the equation because they both contain "$Variable_{true\ value}$" to give

$$Variable_{as\ read} = (1 + Error_{scale}) * Variable_{true\ value}$$

Since "$1 + Error_{scale}$" is a constant, we can simplify the calculation by replacing it with another constant "$Scale_{factor}$" equal to "$1 + Error_{scale}$" .

$$Variable_{as\ read} = Scale_{factor} * Variable_{true\ value}$$



As you can see from a graph of a scale error, it looks quite different from the offset error as the readings meet at zero (0) and diverge the further away from zero that the true value gets.

To correct this error, it is simply a matter of dividing each reading by the $Scale_{factor}$ to get the true answer (provided that there are no other mechanisms in play). Let's check this out to see if that is correct. If the cloudy plastic reduces the light to the sensor by ½, then the error scale value must be –0.5 (because the reading went down). This makes the Scale factor (1-0.5) or 0.5. This means that the variable as read is half of what it should be as expected. To calibrate the reading we need to multiply the reading by 1 / scale factor. 1 divided by 0.5 is 2, so that means the variable as read which is half what it should be is doubled, giving the true value.

Now, lets look at a tricky one that is much more complex that my simple explanation implies. Instead of adding a constant offset, or a scale error, let's add some noise. We can attempt to reduce the effect of the noise by using signal averaging which exploits the fact that making a measurement many times, the signal component will tend to accumulate but the noise will be irregular and tend to cancel itself out over time. This implies that computing the average of many samples of a noisy signal will reduce the fluctuations and leave the signal more visible. This can be used to recover a digital signal which has an amplitude smaller than the noise.

However, it is not plain sailing. If we take a signal that has a noise level that is equal to the value of the signal that we are looking for and we average 4 samples, the level of the noise is expected to drop to half its previous value (not taking into account other factors). If we average 9 samples, we expect the noise to fall to 1/3 the value, 25 samples and it falls to 1/5 the value. By the time we get to 100 samples, it has only fallen to 1/10 the value and 400 samples is needed to get it down to 1/20 of the value.

$$SNR_{increase} = \sqrt{N_{samples}}$$

As we will see in a later edition, doing a lot of mathematical calculations tends to bring errors of its own that cannot be reduced by averaging. Even ignoring that, there are still problems with averaging a long stream of samples. The value that we are looking for is not steady, so all of the samples that are going to be averaged must be taken fast enough so that the changes in the signal that we want to measure are not masked by the averaging.

Well, that's enough for you to ponder in this edition. In the next PWS edition, the discussion will continue looking at errors that can be caused by sensors having a limited measurement resolution and the limits of computer mathematics.