Report 3

Data Analytics and Azure

Please note: All of the datasets and results are available on [my GitHub page](#).

Version 1.0 RTM
Compiled on May 6th 2019

Introduction

While I have played around with many types of data, this set of 300 MB data that I analysed proved to be unusually complex. Here's my thoughts on this, and a review on Azure[1] (specifically virtual machines).

Motivation

I've been looking for quite some time for full CBSE results, and my interest in it only grew after seeing some excellent articles about the non-normal distribution of CBSE scores (one example is at http://www.thelearningpoint.net/home/examination-results-2013/cbse-2004-to-2014-bulls-in-china-shops).

When I picked up this bunch of data from here, I was naturally quite curious to analyse 1 million students' results scraped from the web (thanks to it being very easy to access anyone's CBSE Class 12 results till 2016).

But it took longer than expected and hit quite a few interesting roadblocks.

Part 1 - *Preparation*

The data was presented as 8 CSV files, and I needed it in one. That was quickly done thanks to a nifty Windows command:

```
copy *.csv cbse.csv
```

The above method appends all csv files in that folder and puts it into one csv file. It's not limited by extension – any will work as long as it is text based (so no Word docx files).

The next step was to import this into Excel. Now a question that I was asked a few times was – *why* Excel? Why not Python (which is quite good in this)? There were a few reasons:

- I was most familiar with Excel – and wanted to see how it would handle this.
- I was already midway in the process before someone suggested Python to me.
- It's the only one I know which can present the data in such a powerful way. Conditional formatting is one of the best ways to achieve this – and isn't slow either:

---

[1] It's an unbiased review (should mention this due to my MSP affiliation with Microsoft)

Anyway, that step went smoothly. In particular, the total number of rows was just under 1.003 million, which is just under Excel's limit of 1.04 million.

Naturally I then started to calculate percentiles (as I usually do). But that's for another section

Part 2 – Processing

This section just didn't go as planned.

At the start

To start with, when finding the percentiles, I realised that quite a few of the data were malformed. For example, I realised (at the beginning) that

- quite a few of the total scores were appended with P: for example, 078 P.
- Some others had just A (for absent)
- Others had F appended on them: for example, 022 F or 042 FT (for a student who failed in theory in a subject which had both theory and practical components)
- Very few results were nonsense
- Some results were duplicated

While most of them are normal tasks to be done (aka data cleaning), I didn't do it **before** starting to calculate the percentiles – and that was a mistake which ended up compounding the time.

Another decision that I made was to use Excel's less used binary file format (.xlsb) rather than the common .xlsx format. Initially I did it mainly because of speed and space (as .xlsb files take up significantly less space – 300 MB vs 450 MB – and hence load and save faster), but it allowed me to add VBA code, something which I wasn't planning to initially but proved to be valuable later.

Excel's *dependency tree*

To understand why the above case was an issue, it's important to consider the concept of *dependency trees*. In a nutshell, that is the method adopted by Excel to avoid having to recalculate everything when a change is detected. Excel keeps a track of which cells are 'dirty' – i.e, which cells needs to be recalculated. While this recalculation is usually kept on, Excel sometimes takes a lot of time to calculate when many cells needs to be recalculated, which is why it includes an option to switch recalculation off. When that happens, Excel uses the dependency tree to keep track of the cells to be recalculated (when can happen manually, or when saved if so enabled).
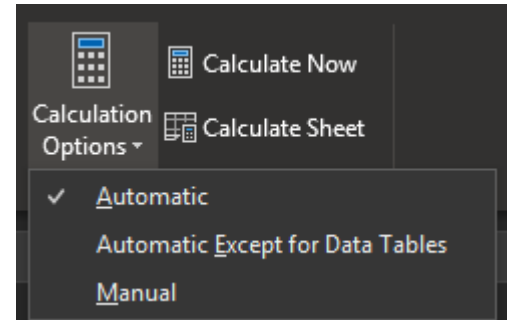
In my case, calculating one million cells' percentiles took around 3 hours on my quad-core laptop, which was clearly too much. So, I started by switching recalculation off, and then filling all the cells with the percentile calculation formula. But then that step itself took around 1 ½ hours and then Excel basically hanged till I forced it to calculate. The dependency tree was simply too large for Excel to handle. This also meant that I could not, for instance, combine multiple steps (like finding five columns' percentiles and totalling them *before* asking Excel to calculate them) into one.

That was the reason my mistake of not cleaning the data thoroughly before running data analytics proved to be costly. If **one** cell on **one** subject total was changed, that would instantly mean that

- All percentile cells in that subject column will need to be recalculated
- The average marks – and their percentile cells – will need to be recalculated
- The percentile averages – and *their* percentiles – will need to be recalculated.

Out of the five columns to be recalculated, three of them take around three hours, and the remaining two (the ones which do not calculate percentiles of something themselves) take a negligible amount of time. If *any* of the data were malformed (i.e, letters or otherwise uncalculatable), then *all* of the cells in the percentile column will return #VALUE!, thus making the data unusable until the data is corrected and recalculated.

Speed

Some of what Excel did was reasonably fast for the complexity of the data, while others took a puzzlingly long amount of time.

For instance, Excel was quite fast for things like conditional formatting – even when it was dependent on the percentile.

But then it took a *lot* of them just to filter data – which was particularly bothersome when trying to clean the data in one go. While *Find and Replace* was definitely very helpful (and took no more than a few minutes – plus it had some indication of progress via the active cell), it did not suffice for cases whether the condition is not static. If I tried to filter the data, it would take a lot of time (even hours at times) to filter it, and then recalculate – even though the data isn't deleted, only hidden! That seems to be a bug or (un)intentional design feature of some sort ([this](#) and [this](#)), and personally this makes no sense.

And my main pet peeve is that this filtering – or slowdown by the dependency tree – is not multi-threaded. This meant that my CPU was often running well below its potential at times – and even with Turbo Boost (which can be inconsistent or even non-existent depending on the environment), it took much more time that it would have taken if Excel was fully multi-threaded and not just for calculations.

The main irritation was that the core percentile calculations were quite slow. It took 3 hours to find the percentile of one million rows. Even though that is about 100 rows/sec (which *looks* fast), Excel could be fast. This meant that the finale (which was basically a worksheet recalculation due to the changes made) took **32** hours on 3 threads. That's long…

VBA scripts

With my facing the prospects of another long wait if I had to filter and modify cells all the time (which, as said in the previous section, can become very slow), I wondered whether this can be shortened up using Excel's Visual Basic for Applications (VBA). The benefits for this approach would be the fact that it's possible to modify multiple cells' values or even unrelated tasks in sequence without the risk of Excel becoming unusable, as they can be embedded – or called – from the same function.
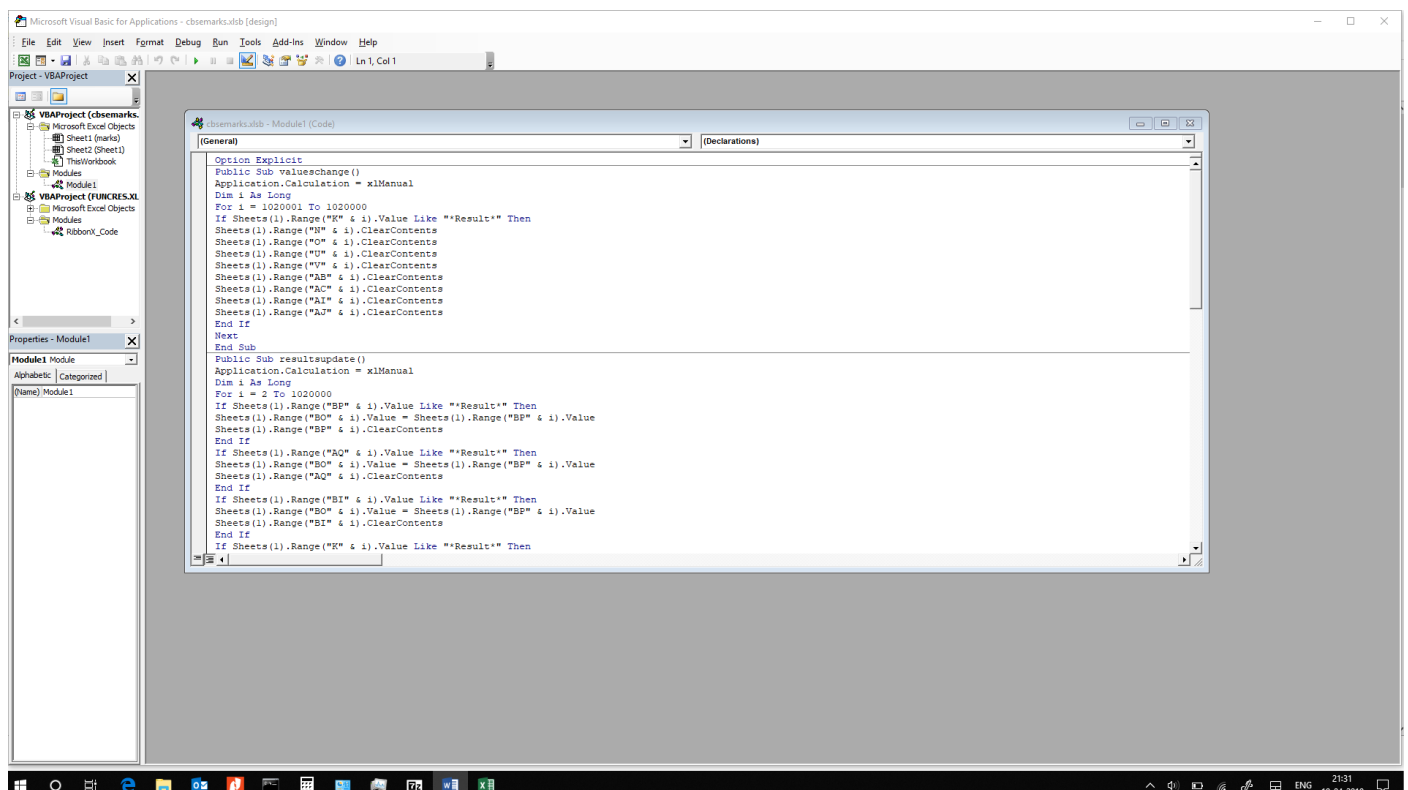
However, while still really helpful,

- It took quite long to execute the command – and that is with calculation explicitly set to manual mode. Unfortunately, cell changing still takes a lot of time, and the issue of Excel becoming unusable still persists.
- VBA is purely singly threaded, based on archaic Visual Basic 6, which itself is not inherently multi-threadable, unlike VB.NET. This meant that running the macro would hang Excel and VBA till it completes, and there is no way to parallelise execution of several functions (like modify values in column B and C).
- There is no way to monitor the progress of the script – trying to print messages to command window (`Debug.Print`) is useless as the whole VBA window hangs when executing the script (through this is true for nearly all single-threaded operations in Excel)

Overall, it was helpful despite the drawbacks, and I think I could have saved some time by using this method earlier than I did. But it's still slow in modifying the cells' values.

I did run into a strange issue, wherein VBA (and Excel) would randomly close when running one of my macro scripts with no trace of what could have happened – when reopened, it was like you've done

nothing. AutoRecover does not work either. This turned out to be a memory issue as Excel would cross 3 GB (maximum 32-bit limit) and then crash (which is still a bug as the correct things would be to inform the user that no memory is available to complete the operation, which *does* happen in normal cases).

Part 3 – *Azure*

I'm now taking a short detour to give a short review of Microsoft Azure, which was used while working with my data. Only my experience with virtual machines will be covered here, as that was my primary focus on using Azure at that time.

Motivation

At that time, I was mainly processing the data on my laptop. While my laptop is fairly fast (quad core i5), it's still only a quad core. I needed something faster. Additionally, Excel would take up most – if not all – of the resources, and I did not want my laptop to be running all night.

The reason I picked Azure as the cloud platform was simple – it's the only one for which I have a substantial amount of credit every month ($150), and hence could run without having to worry about my credit running out or having to pay.

Creation and choices

Creating a virtual machine was straightforward – I go to the Azure portal – and select the option to create a resource. Once I selected the virtual machine option from there, I was presented with several options to configure from:

While there are several options to choose from, I think it could have been better:

- One of the main reasons for choosing a virtual machine is the flexibility that comes – one should be able to 'build your own PC' by allowing the user to choose *exactly* what he need. What we have here is many cryptic "VM sizes" that, while providing a reasonable variety of choices, is still 'tied in' to an extent. For example, if I need 16 compute cores, I need to pair it with over 64 GB of RAM while I can

do just well with 8 or 16 GB of RAM. Indeed, users who need that much power will *most likely* be having far more RAM than that, but the point still stands.

- https://azure.microsoft.com/en-gb/pricing/details/virtual-machines/series/ gives a general outline of what each series mean, but I did not know why exactly I should pick F series over A series (for example) for my particular use-case.
- Some very high configurations were locked out in my case, but that's a small nitpick (and can be removed by logging a support call)
- For some reason, I wasn't able to use Azure Hybrid Benefit if Windows 10 Pro was selected as the operating system, with an error message stating that "License Windows Client must be selected", which does not make sense as Windows 10 is indeed a client version (and the VS subscription does cover client versions of Windows). Even then, I did not know how much I would save with this option (up to 49%? That sure sounds promising, but I was doubtful that it would be that great in reality)

The rest of the process was quick and easy. The virtual machine was created in around two minutes, which is *very good* and a good reason to use virtual machines for short – but high – workloads as in my case.
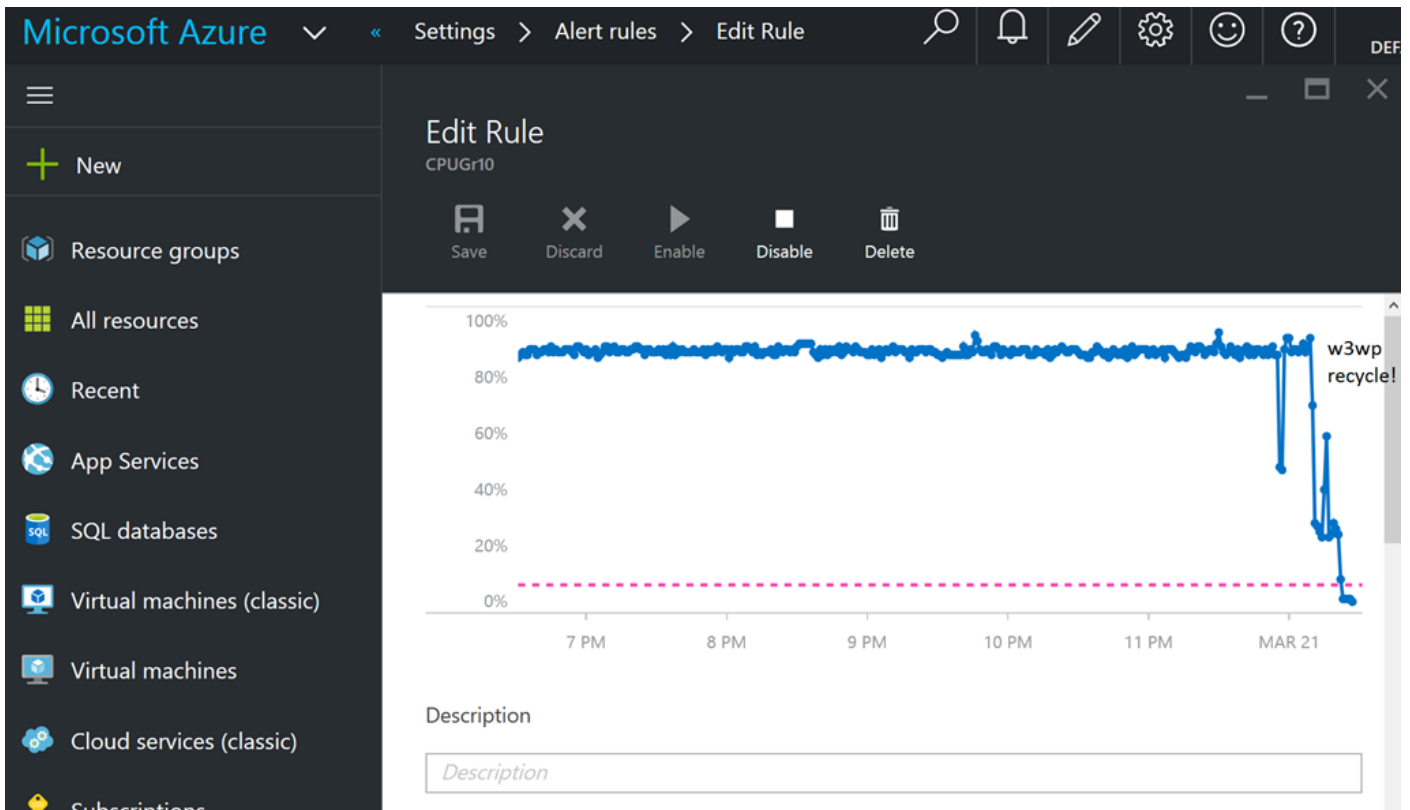
Experience with the virtual machine

Connecting to the virtual machine was quite easy using RDP and brought me to a freshly-installed Windows Server 2019 (the OS I chose) instance, from which I installed Office 365 (64-bit) and started working.

For most of the time, I was on a 16-core VM with 128 GB of RAM, and hence the utilisation was either 7% (one core used) or ~100% (fully loaded). And the RAM wasn't used much, which forms my argument on why Azure could have more flexibility.

Additionally, I had no idea whether Excel was taking advantage of Turbo Boost. Task Manager reports that the processor is always stuck at the base speed no matter what, which *might* make sense as it's a VM, but then the Azure guide is misleading in that it specifies the boost speed even though it can't probably be used. And it was an issue practically, as that meant that I was unable to exploit the power of Turbo Boost when I needed it the most (i.e, when the processor was only loaded on one core).

However, the Azure portal is very useful. I can view the CPU load of the virtual machine without having to log in to it, which gave me a useful indication as to what stage it was (still running the script, calculating etc).

(taken from http://cennest.com/weblog/2016/04/using-the-azure-support-portal-to-diagnose-a-cpu-spike/)

And this one feature saved me from having to log in to the virtual machine every time, for I could easily estimate what Excel was doing at that time:

- ~0% - process completed or Excel crashed
- ~$\frac{100}{n_{cores}}$% - Excel is computing something which involves only one core, for example, filtering. Or a VBA script is executing
- ~100% - Excel is calculating something which involves the use of multiple cores – mainly calculating percentiles

I can even set the frequency of the graph – one hour or one day?

Conclusion

Azure is powerful, no doubt on that. The ability to rapidly create virtual machines, and easily monitor the vital parameters of the virtual machines, was very much useful in my work, as shown above.

But I think there is room for improvement, particularly in sections relating to flexibility. While Azure is already significantly better in this regard than traditional on-premises solutions, the mere fact that everything is allocated virtually would make me think that it should be possible for infinitely large choices, which is not the case here.

Additionally, pricing is an interesting question. Fortunately, it was not something that I had to worry about with the substantial credit provided. But with some of the virtual machine configurations costing over 0.5 pounds per hour, it could be better, even if it's quite good already. Furthermore, shutting down the virtual machine isn't enough – one must delete it – otherwise charges for disks are applied – which is not something that would happen in on-premises.

Would I use it again? Yes.

Part 4 – *Finalising*

The R experiences

Once I had managed (finally!) to process the data in Excel, the next step was to graph it. Based on my experiences in the previous sections, I knew that the one program *not* to use would be Excel, as I was aware that creation of charts is single-threaded there.

I used the Microsoft R Client (and R-Studio when the former was unavailable)

R can load the data quicker than Excel, and creating the histogram took only a few seconds. But for some reason, the number of bins couldn't be more than 20 (this worked later though) …
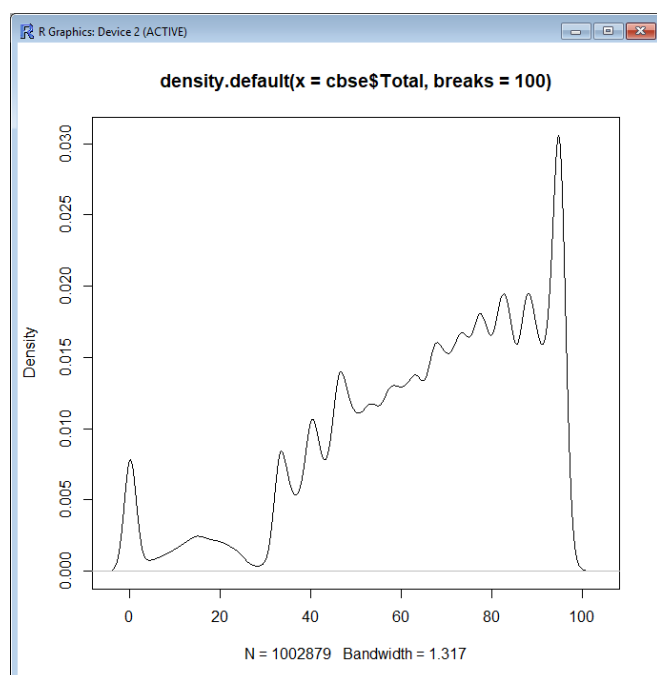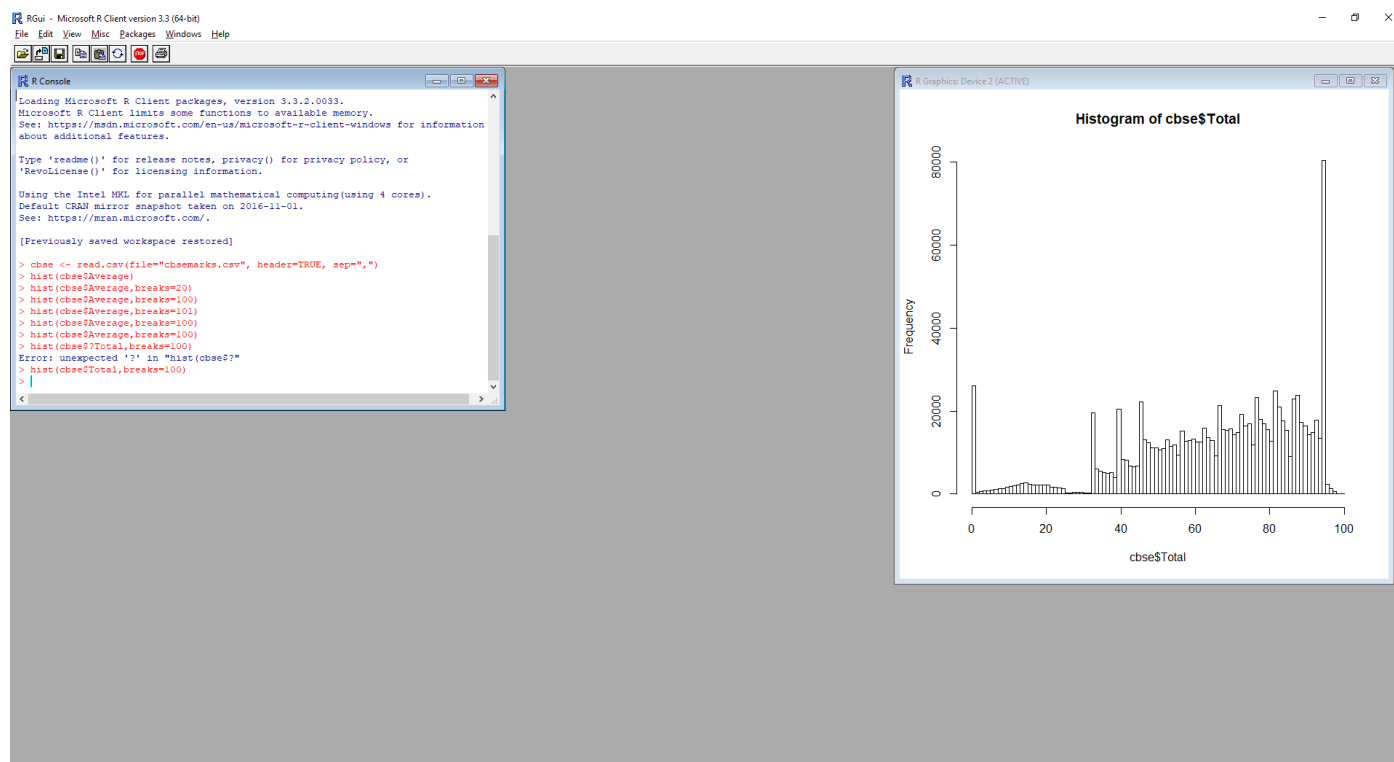
The density plot in R is very interesting, and provides a continuous-*ish* representation of the data:

However, it is slightly misleading – while the peak is indeed at 95, it is not as prominent – though still visible – in the density plot as compared to the histogram.

To sum up, R is useful. But to get the most of R would require learning its syntax, which would take some time for me as I hadn't used R before (This is, in essence, true with Python as well, and is one of the better reasons for using a GUI tool like Excel or IBM SPSS).

When looking on the lab machine's application list, I simply clicked the link for IBM SPSS for fun. And soon I realised that it could be potentially useful. And it indeed was.

It's quick. Slightly slower than R, but still massively faster than Excel. It's also stable, despite being written in Java (I mention this as Maple is another mathematical software written in Java and I'm told by people doing first year mathematics about its reliability – or lack thereof).

And more importantly, it has a powerful GUI:
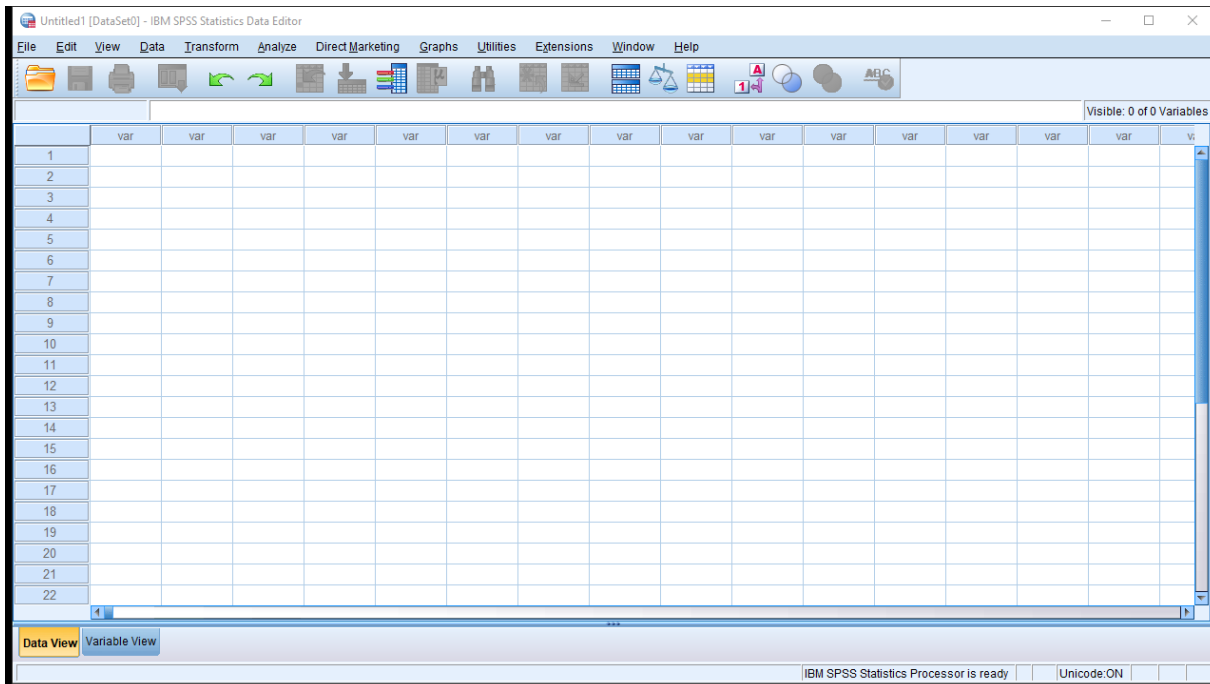
I can easily create charts and view statistical data easily, and while it is not a replacement for Excel in that it cannot save charts or workbooks with the main file, the charts and results are generated in a Jupyter notebook style, which makes it easy to view multiple results on the same page.

That was the one which finally enabled me to produce these beautiful histograms:
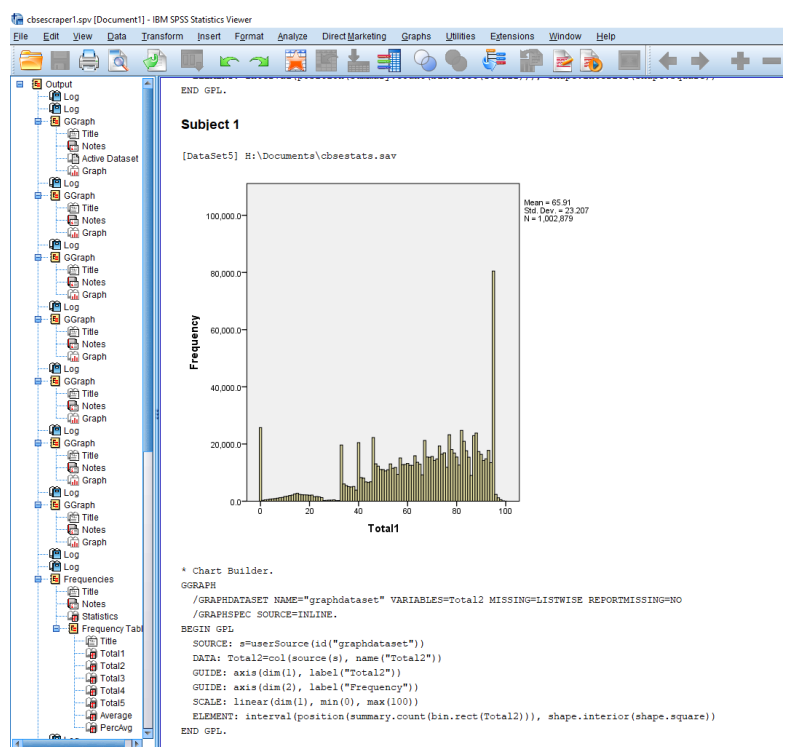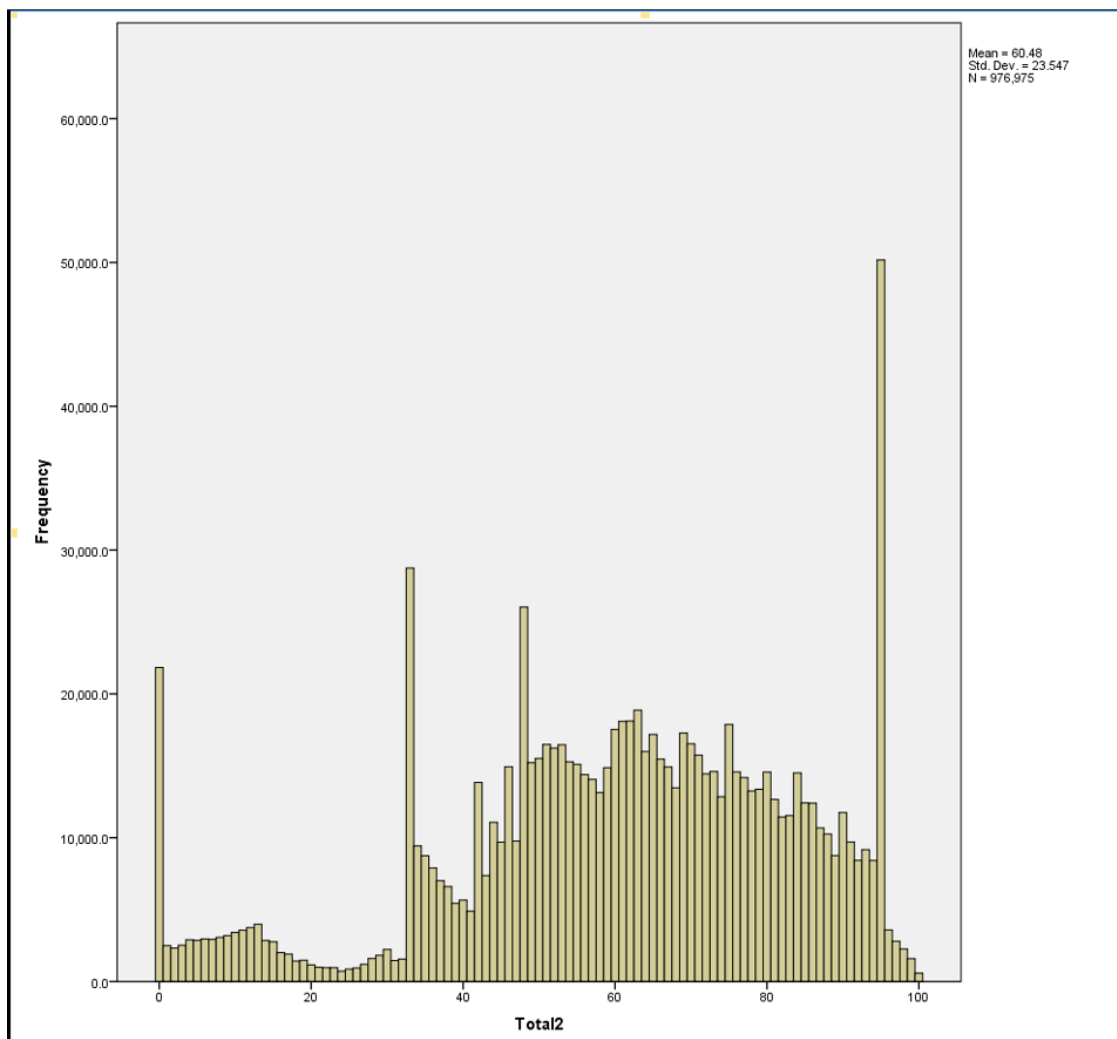
The title: "Subject 2 mark distribution. The peaks still stand.". In particular, the distribution is not normal, which is typically what would be expected from such a dataset.

Conclusion

This project was interesting. It was meant to be a routine data analytics exercise, but it ended up being far more time-consuming – and interesting. Excel was my go-to tool – it's powerful and easy to use – plus I was familiar with it. But it was also horribly slow for large datasets, and hence I turned to statistical tools like R, which gave me greater insight on how professional statistical tools are used.

Would I use Excel again? Yes. But I would be better prepared to use alternative tools (looking at you, Python) and potentially make use of Jupyter notebooks (or its Azure equivalent, Azure Databricks) instead.

But what about CBSE? Unfortunately, 2015 is the last year for which direct raw scraping of results would work. From 2016, they started demanding the school code and exam centre code, which means that a look-up table is needed, else we'll have $O(n^3)$ complexity, which is horrible, considering that $O(n)$ complexity is quite bad in itself. Worse, from 2019, they now require an *admit card code*, which is something that I have no idea of, particularly as I took the exams in 2018 and never had something like that.

Results

All of my datasets and results are available on my [GitHub profile](#).

As of writing, I have written a short write-up on the results obtained and the practice of moderation on [CBSE's Wikipedia page](#).