**Journal – Marked Register Project**

**Background**

Everyone entitled to vote can (and legally should) register with their local council. Every October, they canvass the whole city and then compile the Register of Electors for the following year.

When you vote at a polling station, the clerk draws a horizontal line next to your name, so you can’t vote twice.

After the election, these papers are scanned and become known as the Marked Register. Political parties can buy a copy, in PDF format, which might inform how they communicate with electors.

Brighton & Hove Green Party currently uses volunteers to manually type the Electoral Numbers of everyone marked as having voted, onto a spreadsheet. Each page contains on average about a dozen marked voters. This data input process takes over 100 hours. I imagine there must be a few errors in such a tedious task, even done by dedicated and motivated volunteers.

**Structure of the Register**

The city comprises 21 wards, A to U. Each ward has several polling districts (each with its own polling station) named backwards from Z, eg Ward A comprises polling districts AT to AZ. The Register of Electors for each Polling District is sorted alphabetically by street and then by house number/name and flat number, and finally by name (surname then forename). Voters are numbered consecutively from 1, with later additions suffixed /1, /2, etc after the number of the person before them in the order of the register. An Elector Number could, for example, be AU-1309 or RZ-277/2.

On polling day, each polling station is given a paper copy of the register for that polling district. Some larger polling districts split the register into two parts, presumably to avoid queues building up. When you turn up to vote, an officer draws a line on the left of your name, to prevent multiple voting. There are 125 PDF files of Marked Registers.

**Data Preparation**

Because of the highly regulated nature of elections, there is no missing data 😊

**Input Data**

Neural Networks can take in PNG or JPG files

* PNG files retain more resolution (expect better classification)
* JPG files are smaller (expect lower demand on resources and quicker results)

There are only two, so I propose to try both and compare their performance.

In a post in the Brighton Data Forum Slack channel, my friend Oskar recommended ImageMagick as a good command line tool for converting image formats, however, I couldn’t get Anaconda to install it. After an hour on Stackoverflow, I gave up and found an alternative app, Win2PDF. I was surprised that the batch conversion took nearly three hours to produce JPG and PNG versions. This makes me question whether my humble PC will cope even with a very small trial run. It also converted single pages into separate files, eg AT.1, AT.2, etc. This will be helpful, as the first two pages were just a header and list of streets, which will play no role in the classification, so I’ll move them into a folder called unused:

move \*.1.jpg unused

Some Polling District registers contained a blank page at the end, so I deleted these manually. I counted them out of interest (there were 68.) As an afterthought, I was looking for a way to have identified these blank pages, so I sorted the backup directory by size and there was a clear cluster of 68 small files (15-28kB) distinct from the rest (60-319kB). The range of sizes of blank pages surprised me, but there was a very clear gap between them and the first page with data (on inspection, it contained a header, footer and one single voter.)

The only remaining ‘dirtiness’ in my image data was the fact that a few polling districts were scanned in portrait, with the majority in landscape. I used Windows Explorer to sort by dimensions, then selected all the landscape images and rotated them.

I was hoping to get access to the exact version of the database that was printed for polling day, which would’ve avoided the need for OCR. Unfortunately, that’s not available.

**Output Data**

The output data consists of two columns in a spreadsheet: *District* (two letters) and *Voter* *number* (a number up to 9999), containing the Electoral Numbers of everyone who voted. While the register uses 99/1, 99/2, etc for late additions, volunteers used a decimal point as a delimiter, in order to facilitate numerical sorting (ie into register order). I imagine having output data in the same order as the scanned input pages might make for more efficient training.

**First Steps with ChatGPT**

Having explored ChatGPT for a few weeks, I’m aware of its limitations (and it initially felt like cheating), but I found I was intuitively asking it to teach me how to do things myself, sometimes asking for a simpler suggestion, so I was able to understand the code. This gives me confidence in the code, and enables me to write my own comments. I now treat it as the new StackOverflow.

**Convolutional Neural Networks**

My Udemy course on data science only covered the basics of neural networks (with Keras/TF), so I needed to learn about CNNs for image recognition. I love Grant Sanderson’s (3blue1brown) visualisations of complex mathematical concepts, so I was delighted to discover he’d made a video called [But what is Convolution?](https://www.youtube.com/watch?v=KuXjwB4LzSA)

With that grounding, I set out to learn about CNNs. After a couple of hours of dry, academic lectures and amateur visualisations, I finally stumbled on an absolute gem, [A friendly introduction to Convolutional Neural Networks and Image Recognition](https://www.youtube.com/watch?v=2-Ol7ZB0MmU) by Luis Serrano, which took a simple-as-possible approach of sliding a 2x2 frame over a 3x3 1-bit image and clearly describing every possible outcome.

**The Jupyter Brick Wall**

My budget PC, which had successfully run Jupyter Notebooks locally, just refused to install several libraries. I spent whole evenings trawling Stack Overflow and asking ChatGPT to guide me (and learnt about PATH variables) but it just didn’t work and I put the project on ice. I had been using Anaconda (the number one python installation) and I also tried Jupyter itself.

Months later, a friend who builds PCs offered to help, so I took my PC up to York for the weekend. After two hours and much swearing about the quality of open source documentation, I finally had everything I thought I needed. I soon realised I also needed to install an OCR engine. I tried several but none would install, so I again put the project on ice.

**Moving to GCP**

Because I was processing data about real people, my concerns for security meant I daren’t use a cloud service. A friend in Brighton Data Forum finally persuaded me that cloud platforms are set up for security by default and that I would be fine if I didn’t actively compromise security. I had found AWS had too steep a learning curve, and I’d left something in Azure assigned (not even running) and lost £40, so I tried GCP. I was delighted to find an accessible, beginner-friendly platform, that didn’t demand my credit card for a free trial. I installed PyTesseract first time! With the help of Google Gemini, my new LLM friend, and with security and the reputation of the Green Party in mind, I uploaded the page of the Marked Register containing my own details and successfully extracted the text. The next step is to extract the bounding boxes…