

Academic Declaration

I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software.

Providing computational access to the Polytechnic Magazine (1879 to 1960)

Abstract

This project unlocks new ways of accessing a digitised collection of historic magazines held by the University of Westminster Archive. The project evaluated and implemented options for Optical Character Recognition (OCR) of the documents, corrected the OCR'd texts, ingested the texts into a Pandas DataFrame, added metadata and conducted pre-processing of the texts, and undertook a series of Natural Language Processing (NLP) tasks including frequency analysis, named entity recognition and topic modelling. Outputs of the project include: two public Jupyter notebooks shared via Google Colab that enable users with little or no experience of computational techniques to conduct their own digital research on the corpus; a DataFrame containing the underlying corpus, pre-processed text and metadata; the corpus itself as corrected plain text documents; and a series of Jupyter notebooks that were used to prepare the text, and experiment with NLP techniques to prepare the public notebooks. The project was undertaken in support of the University Archive's aspiration to improve the level of digital access to its collections, and it is anticipated that the public notebooks, in particular, will be a valuable resource for researchers and for introducing our students to new methods of working with historical texts.

Contents

1. Introduction.....	5
1.1 Context	5
1.2 Problem Statement	5
2. Literature Review.....	6
3. The Data: the Polytechnic Magazine files	7
4. Optical Character Recognition and initial text correction	8
4.1 Methods	8
4.2 Solution.....	11
4.3 Testing, Results and Discussion	12
4.4 Initial Text Correction	16
5. Natural Language Processing.....	17
5.1 Methods	17
5.2 Solution.....	18
5.3 Testing, Results and Discussion	25
6. Public Resources.....	33
6.1 Methods	34
6.2 Solution.....	35
5.3 Testing, Results and Discussion	41
7. Future work	46
8. Conclusion.....	46
References	48

1. Introduction

This project facilitates new forms of access to a digitised collection of historic magazines, the University of Westminster's Polytechnic Magazine collection. There were three principal areas of development for the solution: Optical Character Recognition (OCR) and initial text correction; text pre-processing and Natural Language Processing (NLP) experimentation; and development of the public resources. After an elaboration on the reasons for the project and a literature review, each of these development phases will be described in turn.

1.1 Context

There is a growing recognition in the archives domain that we must fully exploit the affordances of born-digital and digitised historical collections. Trevor Owens warns us to avoid 'screen essentialism' and to consider the informational qualities of digital objects beyond their representation on the screen (Owens, 2018, 46). The National Archives' digital strategy aspires to 'offer the ability for users to compute over all open digital records for research purposes' (The National Archives, 2017, 7). Such aspirations reflect the recognition that finding and retrieving individual documents can be an inadequate response to the large quantity of digital material now available to researchers (Winters and Prescott, 2019). Nevertheless, with some exceptions, where digital material is made available, it tends to be through a website aimed at replicating the experience of viewing physical documents, and this is largely the case with Westminster's Polytechnic Magazine collection (University of Westminster, 2011). The Polytechnic Magazine was the in-house magazine of the Regent Street Polytechnic, one of the predecessor institutions of the University of Westminster. Since 2011, a digitised run of the magazine covering the years 1879 to 1960 has been made available by the University Archive via a dedicated website. This has proved an invaluable resource for academic researchers, family historians and university staff.

1.2 Problem Statement

While providing search and viewing functionality, the existing Polytechnic Magazine website does not fully take advantage of the digitised nature of the collection. With 1,725 issues and more than 16 million words, describing over 80 years of the institution's history, the volume of material makes it difficult to analyse systematically.

In their current form the documents can only be accessed one at a time as PDFs for traditional close reading and computational analysis is impossible. The Optical Character Recognition (OCR) that was conducted during the original project of 2011, does not support easy extraction for computational analysis as it has not preserved the layout and order of sentences. It is hoped that opening up the collection to computational analysis will allow researchers to unearth previously unidentified themes in the material and lead to future research and experimentation. This will support the University Archive's aspiration to improve our level of digital access as measured by the Digital Preservation Coalition's Rapid Assessment Model, which sets out provision of 'Advanced resource discovery and access tools' as a criterion for organisations aspiring to meet the highest level of digital access (The Digital Preservation Coalition, 2021, 21). However, given the different levels of digital capability found in archive users, any solution to genuinely increase the University Archive's provision of digital access must support users with relatively limited digital skills as well as catering to more digitally proficient researchers.

2. Literature Review

In order to get the text of the documents into an environment that can be used for computational analysis, it was first necessary to extract the text from the existing PDF files. Walker, Lund and Ringger (2010) caution that OCR errors can impact the results of topic modelling, so accuracy during this step was important. Selection of OCR methods for initial investigation was informed by discussions from the Internet Archive (Kahle, 2020a and 2020b) and the OpenNews project (Han and Hickman, 2019). Tesseract workflows were adapted from the Programming Historian (Akhlaghi 2021 and Mähr 2020). The OCR element of the project lends itself to quantitative evaluation, with research suggesting that an accuracy of 90% is desirable for subsequent NLP (van Strien et al., 2020). It is also noted that there is scope for increasing accuracy post-OCR by correcting common errors (O'Hara, 2013). Once the text has been extracted, computational analysis can begin. In exploring potential research techniques my approach has been informed by Blaney, Milligan, Steer and Winters (2021), who provide an introduction to digital history methods, and by Ian Milligan's work on topic modelling web archives (Milligan, 2019). Andrew Salway and James Baker (2020) have used corpus linguistics to investigate catalogue data, as has

Christopher Day (2020) at The National Archives. Both pieces of work provide excellent examples of what can be achieved with computational techniques, when only museum or archive catalogue data is available. It is therefore expected that working with digitised records themselves can yield further insights. In terms of technical implementations, Bird, Klein and Loper (2009) provide the definitive introduction to NLP with the NLTK library. There are also several examples of large-scale digitisation and computational analysis projects working with historical newspapers, such as Living With Machines (The Alan Turing Institute, n.d.). Smaller scale examples include Monika Barget's series at the Leibniz Institute of European History's Digital Humanities lab (Barget 2020). Numerous workflows and tutorials have been examined for this project including Havens (2020), Li (2018), Prabhakaran (2018), Rahmani (n.d.) and Zhao (2020). Zhao's *Natural Language Processing in Python Tutorial* has proved particularly helpful, and several of the project workflows have been adapted from her work. For the NLP phase, evaluation will be based on a mixture of concrete (e.g. has the text been imported properly?) and more subjective (e.g. do the topics identified fit our knowledge of the collection?) criteria. In terms of making computational analysis available to researchers, examples of open computational access to records are rare, with some notable exceptions such as the Archives Unleashed project (Archives Unleashed, 2021) and the GLAM Workbench (Sherratt, 2021). Both projects have pioneered the use of Jupyter notebooks to make computational analysis open to non-specialists, an approach this project will take up. Success in the final stage will be difficult to measure during the life of the project, since it is hoped to result in longer-term engagement by outside researchers, however having live public notebooks with worked examples and explanatory text will be regarded as a positive end point.

3. The Data: the Polytechnic Magazine files

The starting point for this project was the 2,224 PDFs that were created during the University Archive's 2011 Polytechnic Magazine digitisation project. The magazines cover the period from 1879 to 1960 and therefore reflect a wide range of typographical styles and layouts. In terms of content, as well as news of the Polytechnic's activities, the magazine regularly featured personal news of members, reports from sports and social clubs, letters and commentary on current affairs and religious matters. As well as the main run of Polytechnic Magazines (from 1888 – 1960) there were also

examples of the precursor magazine Home Tidings, as well as various special issues such as holiday supplements. Various subsidiary artefacts were also included in the original set of PDFs, including separate PDFs containing portraits of prominent figures in the history of the Polytechnic, lists of prizes awarded, artefacts relating to how the magazines were arranged in bound volumes, and separate PDFs of front and back covers, these were appraised and considered to be outside the core content of the project and discarded, leaving 1,725 issues and core supplements. The quality of the scanning, while high for the time, is not perfect, with some cases of skewing, and it reflects the original rather thin letter forms.

4. Optical Character Recognition and initial text correction

The first challenge for opening up the magazines to computational analysis was to extract the text in a machine-readable format. This section details the different methods of approaching this problem that were considered; the implementation of the chosen solution: a Python script for bulk processing batches of files using ImageMagick and Tesseract; the evaluation of the OCR'd files; and how the files were initially corrected following processing.

4.1 Methods

Several potential solutions to this problem were considered. It was initially thought to use the existing OCR'd text embedded in the PDFs, and the Python libraries, pyPDF and PDFPlumber were evaluated for this purpose. However, when applied to sample files, the text extracted by both libraries was found to have run together text from two columns into one, potentially compromising future analysis. Rather than an issue with the libraries themselves, this was found to reflect problems with the underlying OCR in the PDF files, as text extraction via Adobe Acrobat resulted in the same issues. This issue is illustrated in the two images below. In Figure 1, we can see two columns of text, as they appear in the original scanned PDF.

In and Around the Poly

To the Front. The event of the month has been the departure of the 12th London Regiment for France. They were inspected on Thursday, December 17th, and departed on Wednesday the 23rd, crossing to France on the 24th and spending their Christmas in or near Havre.

In this regiment are four members of the Poly amongst the officers and some 400 amongst the men. They carry with them our most heartfelt and affectionate wishes and prayers for their welfare and success. We look to them with envious and proud expectations, knowing that they will not fail, but will amply justify and uphold the honour of the Poly and of the country on whose behalf they are offering their lives.

General Sir Francis Lloyd inspected the 12th London Battalion preparatory to their departure for the Front. The inspection took place in the grounds of Roehampton House where the Battalion

paraded. There are about 400 Poly boys in the Battalion, No. 1 Company being exclusively composed of our members, and are officered amongst others by Captain Arbuthnot and Second Lieutenant Ricketts. Major Hoare is a member of our Governing Body and Captain Studd is a son of our President. The Battalion presented a very smart appearance and seemed to give great pleasure to the Inspecting Officer.

At the conclusion of the official inspection, Mrs. Quintin Hogg, accompanied by Major Hoare, the Hon. T. H. W. Pelham and Mr. and Mrs. Robert Mitchell, inspected the Poly boys. Mrs. Hogg made a short speech at the end of the inspection wishing them all God-speed and a safe return. Need less to say, her visit was much appreciated by the members of the Poly Company.

It was a deep regret to Mr. and Mrs. Studd that they could not be present at this historic ceremony. Influenza is no respecter of persons and an old enemy of our President and having again got him in its grip it refused to release him even for the inspection of the 12th London Regiment.

Figure 1: Original text

In Figure 2, we see a selection of the same text extracted from the PDF with Adobe Acrobat DC, the text has been highlighted to illustrate the issue. The text highlighted in grey and the unhighlighted text should be in two different paragraphs, whereas instead they have been run together.

paraded. There are about 400 Poly boys in the Battalion, No. 1 Company being exclusively composed of our members,

In and A round the Poly

and are officered amongst others by Captain Arbuthnot and Second Lieutenant Ricketts. Major Hoarc is a member of To the Front. The event of the month has been the departure our Governing Body and Captain Studd is a son of our President.

of the 12th London Regiment for France. The Battalion presented a very smart appearance and seemed They were inspected on Thursday, December 17th, and departed to give great pleasure to the Inspecting Officer, on Wednesday the 23rd, crossing to Trance on the 24th and <r> <s-o spending their Christmas in or near Havre. At the conclusion of the official inspection, Mrs. Quintin

In this regiment are four members of the Poly amongst Hogg, accompanied by Major Hoare, the Hon. T. H. W. the officers and some 400 amongst the men. They carry Pelham and Mr. and Mrs. Robert Mitchell, inspected the with them our most heartfelt and affectionate wishes and Poly boys. Mi's. Hogg made a short speech at the end of the prayers for their welfare and success. We look to them with inspection wishing them all God-speed and a safe return . Need envious and proud expectations, knowing that they will not less to say, her visit was much appreciated by the members

fail, but will amply justify and uphold the honour of the of the Poly Company, Poly and of the country on whose behalf they are offering

Figure 2: Text extracted with Adobe Acrobat.

As it was felt that this would impact future analysis, it was decided to re-OCR the text. The following OCR approaches were initially tested: Adobe Acrobat, Abbey FineReader, OCRmyPDF, Tesseract v5, Google Document AI and Transkribus. The methods were evaluated with a simple visual check, since it was clear which methods were reproducing the layout error. While no option returned perfect results, Google Document AI and Tesseract v5 were selected for further consideration, since they appeared to offer the best character recognition (Google Document AI) and best layout

(Tesseract). Abbey FineReader yielded similar results but was discounted on grounds of cost. Like several of the other solutions, Google Document AI was not able to adequately retain the layout of the text, displaying similar behaviour to that illustrated above. However, it was included in the shortlist on the basis that it could potentially be paired with LayoutParser (Shen, et al., 2021), and if so, it would offer the best character recognition combined with a correct layout. LayoutParser was tested using a notebook adapted from the LayoutParser GitHub (Shen, n.d.). However, the tests resulted in further issues, which, while less severe than the running together of text, demonstrated that LayoutParser was not able to maintain the correct order of the paragraphs, as shown in Figure 3 below¹.

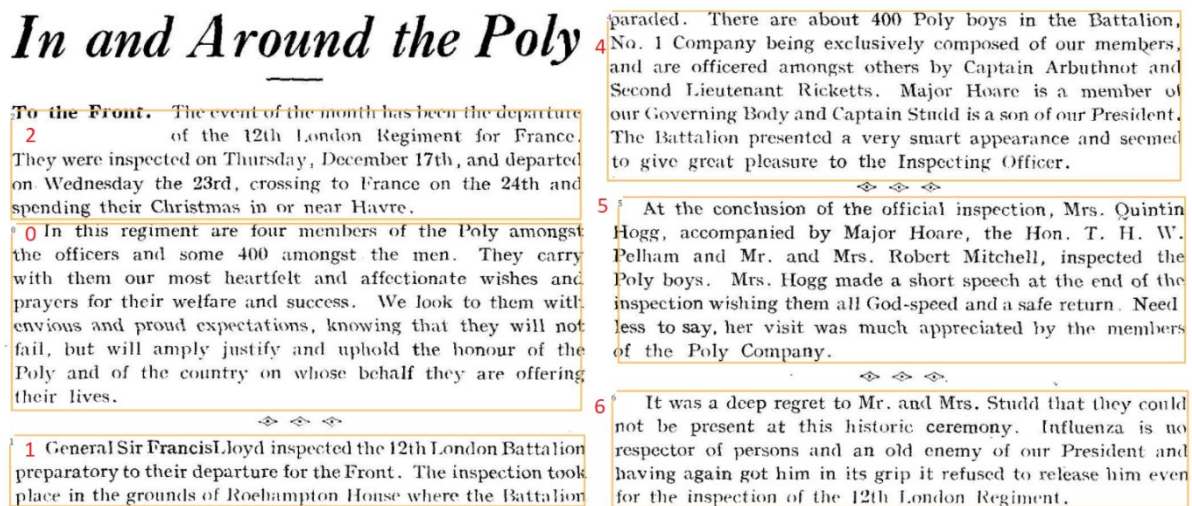


Figure 3: Incorrect paragraph order generated by LayoutParser.

Tesseract v5 was the other OCR system selected for further consideration. Tesseract is a command-line application that works with TIFF files, it was therefore necessary to implement a workflow that converted the PDFs to TIFFs before processing. ImageMagick was selected to do this, using a workflow adapted from Akhlaghi (2021). During conversion, ImageMagick was configured to do some limited pre-processing of the images to increase OCR accuracy. ImageMagick also had the advantage of creating multi-page TIFFs from the PDFs, which could be processed by Tesseract as a single file, simplifying the workflow. As initial results of this workflow were positive, yielding good layout detection and character recognition it was decided to focus on implementing and testing a Tesseract-based OCR solution.

¹ N.B. This was subsequently shown to be due to an issue with the code in the layout parser tutorial workbook which has since been corrected.

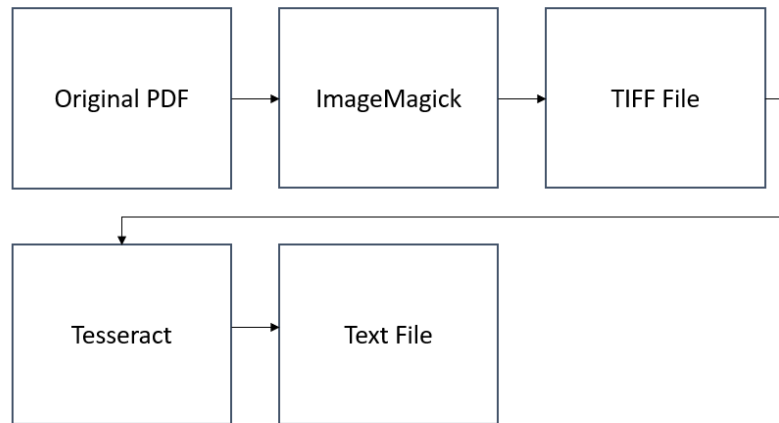


Figure 4: High level ImageMagick-Tesseract workflow

As combining ImageMagick and Tesseract results is a relatively slow process (taking over 3 minutes to process a 30-page file), it was necessary to automate the OCR workflow so that files could be processed in batches overnight. This was done with a simple Python script using the OS module, discussed below.

4.2 Solution

To manage the OCR process, a list of all the PDF files to process was generated. The `filelist.csv` would become a key document of the project, used to keep track of the OCR and subsequent testing process and, with added metadata, ultimately be ingested into the Python DataFrame. Before further work was carried out, the directory and subdirectories containing the PDFs were placed under version control using Git. The PDFs were divided into batches of up to 200 files each and added to folders corresponding to their batch. The Python script used to manage OCR was relatively simple, though it went through several refinements in response to issues that arose during the OCR process. The script used the OS module. It was designed to be placed in the folder with a batch of PDFs, and when run, to generate a file list, then use a for loop to check whether each file was a PDF. For each detected PDF file the script would then use the `os.system` method to first run ImageMagick to convert the PDF into a TIFF, then run tesseract on the TIFF to create the final text file. The script would then delete the TIFF (to avoid disk space issues due to the file size of TIFFs), before moving on to the next file in the list.

```

#import os module
import os

files = os.listdir()
print (files)
for file in files:
    if file.endswith('.pdf'):
        targetfile=file+'.tiff'
        print(targetfile)
        os.system('Magick -density 300 ' +(file) +' -depth 8 -strip
        -background white -alpha off ' +(targetfile))
        os.system('tesseract ' +(targetfile) +' ' +(targetfile))
        os.remove(targetfile)

```

Figure 5 Early version of the OCR script

Over the course of the OCR process, several minor enhancements were made to the script (see 0.1.OCRv6.py). Firstly, the OS module was used to add a shutdown command when the final file had been processed. Then an option to ask the user whether to invoke the shutdown command at the end of the script was added. At the end of each batch, the output text files were manually compared to the file list, and it became apparent that there were occasional missing files. It was determined that these were cases where the original files had spaces in the filename. In response to this issue error handling was added to the script, with the script outputting a log of all completed files and another of any files where errors were encountered. Finally, counters were added to the script, to display the number of PDFs that had been processed, so that the user could monitor progress.

```

for pdffile in files:
    try:
        if pdffile.endswith('.pdf'): #if file is a PDF
            targetfile=pdffile+'.tiff'
            os.system('Magick -density 300 ' +(pdffile) +' -depth 8 -strip -background white -alpha off ' +(targetfile))#pre-p
            os.system('tesseract ' +(targetfile) +' ' +(targetfile))#OCR tiff with Tesseract
            os.remove(targetfile)#delete the tiff as we have completed OCR and no longer need it (and tiffs are huge!)
            with open('log.txt', 'a') as f:
                print(pdffile, file=f)#add file name of process PDF to log file
            worked = worked + 1
            print ('Processed', worked, 'of', totalpdfs)
    except:
        with open ('error.txt', 'a') as f:
            print(pdffile, 'was not successfully processed', file=f)#if there is an error in the above steps, add name of affe

```

Figure 6: Excerpt of OCR script v6, showing error handling and logging

4.3 Testing, Results and Discussion

The OCR phase was completed in twelve days, taking approximately 24 hours to process each batch of up to 200 files. This was longer than anticipated, though the basic automation implemented by the script meant that other work could continue

while the files were being processed. The result was a set of machine-readable plain text files. As suggested in the Literature Review, a minimum OCR accuracy rate of 90% is recommended for carrying out NLP (van Strien et al., 2020). The standard method of measuring the accuracy of text or speech recognition is Word Error Rate (WER), which uses Levenshtein distance to measure the difference between a correct text and one that is an output of an OCR or speech recognition system (Thoma 2013). Several existing Python scripts are available to handle this, and one was adapted to work with the project files (Manovisut 2020).

In and A Around(round) the Poly To the Front . The event of the month has been the departure of the 12th London Regiment(Kegiment) for France . They were inspected on Thursday , December 17th , and departed on Wednesday the 23rd , crossing to France(Vrance) on the 24th and spending their Christmas in or near Havre . In this regiment are four members of the Poly amongst the officers and some 400 amongst the(amongst the) men . They carry with them our most heartfelt and affectionate wishes and prayers for their welfare and success . We look to them with envious and proud expectations , knowing that they will not fail , but will amply justify and uphold the honour of the Poly and of the country on whose behalf they are offering their lives . Qe KS General Sir Francis Lloyd(PrancisLJoyd) inspected the 12th London Battalion preparatory to their departure for the Front . The inspection took place in the grounds of Roehampton House where the Battalion paraded . There are about 400 Poly boys in the Battalion , No(Na) . 1 Company being exclusively composed of our members , and are officered amongst others by Captain Arbuthnot and Second Lieutenant Ricketts . Major Hoare is a member of our Governing Body and Captain Studd is a son of our President . The Battalion presented a very smart appearance and seemed(secmed) to give great pleasure to the Inspecting Officer . At the conclusion of the official inspection , Mrs. Quintin Hogg , accompanied by Major Hoare , the Hon . T. H. W. Pelham and Mr. and Mrs. Robert Mitchell , inspected the Poly boys . Mrs. Hogg made a short speech at the end of the inspection wishing them all God-speed and a safe return . Need less to say , her visit was much appreciated by the members of the Poly Company . It(Tt) was a deep regret to Mr. and Mrs. Studd that they could not be present at this historic ceremony . Influenza is no respecter of persons and an old enemy of our President and having again got him in its grip it refused to release him even for the inspection of the 12th London Regiment .(.)

Figure 7: Example of visual output of the WER script, highlighting errors

However, in order to calculate the WER it was necessary to manually correct files for comparison with the OCR'd version and this was very time consuming. It was therefore decided to only correct a limited sample of the files for comparison. This presented the problem of how to ensure that a sample was representative of the whole range of files. While appraising the PDF files it had been noted that there were several different layouts and typographical styles employed by the magazine over the years and these variations were categorised and recorded in the file list. Some examples of these different layouts are shown below.

styles, roughly in proportion to how commonly they occurred. To work out which variants to target, the number of pages for each issue was calculated using the Exiftool metadata extraction application, this was output to a text file and added to the file list. It was then possible to generate the following table showing the number of pages per layout type.

Layout type	Sum of Pages	Count of Files	Files tested
PM Mid, Monthly, 2 column, standard	10398	521	3
PM Mid, 2 column	5407	435	1
PM Later, cover, 2 column	4515	131	1
PM Early, similar to HT + HT later	6382	402	1
PM Mid, 3 column	1942	125	1
HT early	807	42	1
PM Mid, Monthly, 2 column, elaborate	395	24	0
(blank)	103	13	0
Q, 3 column	50	9	0
irregular columns, lots of illustrations	24	1	0

Figure 10: Incidence of different layout types and the number of files of each type selected for WER testing.

It was decided to test at least one example from each layout type that had more than 500 pages and to perform additional samples for the most common type. From each selected file, the first and middle pages were selected for manual correction and comparison, to provide a representative selection from within the issue. It is recognised that this is a very small sample size, but this reflects the very time-consuming process of preparing the files for testing. The results of the testing are shown below.

Filename	Layout	Date	Volume	Issue	Pages	WER
HT0020020	HT early	1881-02	2	20	20	18.596311475409
HT0140307	PM Early, similar to HT	1889-05-30	14	307	16	1.4604810996563
HT0220508	PM Mid, 3 column	1893-04-05	22	508	20	1.8233618233618
HT0330792	PM Mid, 2 column	1898-09-14	33	792	10	0.4602991944764
HT0450001	PM Mid, Monthly, 2 column, standard	1905-03	45	1	19	1.2793914246196
HT0660002	PM Mid, Monthly, 2 column, standard	1926-02	66	2	26	3.4849951597289
HT0870003	PM Mid, Monthly, 2 column, standard	1947-03	87	3	24	2.3895727733526
HT0970003	PM Later, cover, 2 column	1957-03	97	3	40	1.9788918205804

Figure 11: Outcome of WER testing

The resulting mean WER was 3.93, giving an accuracy rate of 96.07%. Though it was a cause of concern that the example of Home Tidings from the 'HT early' category was significantly higher than the mean, at 18.59. On investigation of this file, it seems that the majority of the discrepancy was caused by the order of paragraphs being different than in the original document, though crucially without the sentences from separate paragraphs being run together, as had been observed with other OCR systems. An additional comparison was conducted with this file using Google Document AI, where the WER was found to be considerably higher at 75.66 (giving an accuracy rating of just 24.34%). It was therefore decided that, although results were disappointing with this file, the Tesseract workflow still provided the best available option. Finally, the entire collection of OCR'd documents was subject to a quick manual review to ensure they were complete, at which point the date, volume and issue number of each issue was also added to the file list CSV. This process revealed that one volume of issues had been missing from the originally supplied files, these were downloaded from the Polytechnic Magazines website and OCR'd as a final batch.

4.4 Initial Text Correction

As a result of errors observed during the OCR process, it was decided to do some basic error correction to improve the overall accuracy of the text. This was conceived as a separate step before the subsequent data cleaning, since at this point the objective was not to provide normalised text for analysis using Python but to correct common errors in the text files themselves, so that they could be used directly by researchers either in their own analysis pipelines or using off-the-shelf text analysis applications. For this reason, it was decided to make corrections directly in the text files. However, in order to prioritise the most commonly occurring errors, it was necessary to do a basic frequency analysis of the text. Therefore, the text files were read into Python, the text was tokenized and subject to a frequency analysis with NLTK (see 0.1InitialDataCleaning.ipynb). The 2000 most common terms were reviewed, and where errors were identified, these were added to a spreadsheet, which had been set up to generate the relevant bash commands to find instances of the error using grep and, where a replacement term was entered, to generate the command to replace the error using sed. The top 100 errors were manually reviewed and updated in this way,

then the 600 next most common terms were reviewed more quickly and in cases that were obviously incorrect they were updated in bulk using a bash script. This workflow is displayed below.

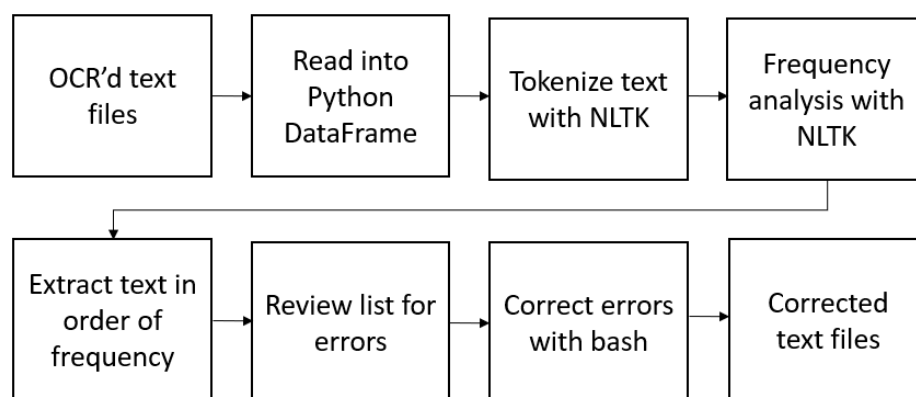


Figure 12 High-level workflow for initial error correction

At the conclusion of this section of the project, the 1,725 text files had been corrected and were ready for further processing and to be made available directly to researchers.

5. Natural Language Processing

The next stage of the project had two main objectives: firstly, to perform data cleaning and normalisation on the text to prepare it for further analysis. This would yield the next major output of the project: the pickled Pandas DataFrame containing the text, the metadata, and different instances of text processing such as tokenisation. Secondly, to use the prepared text to conduct some initial NLP analysis, this step to be used to form the basis of the public notebooks to be developed in the final stage of the project.

5.1 Methods

As discussed in the Literature Review section there are a wide variety of options available for working with text in Python. Broadly speaking, the Natural Language Processing pipeline was envisaged as follows: ingestion of texts into a DataFrame, data cleaning to include tokenization, removal of stop words and punctuation; followed by some exploratory analysis, making sure the text had been processed correctly and could be explored with tools such as NLTK; then to look in more detail at some NLP techniques to use as a proof of concept to be used in the public notebooks, with Topic

Modelling and Named Entity Recognition being the two initially selected. It was anticipated that each step in the workflow would form a different Jupyter notebook, with data being passed between the steps in the form of a pickled DataFrame.

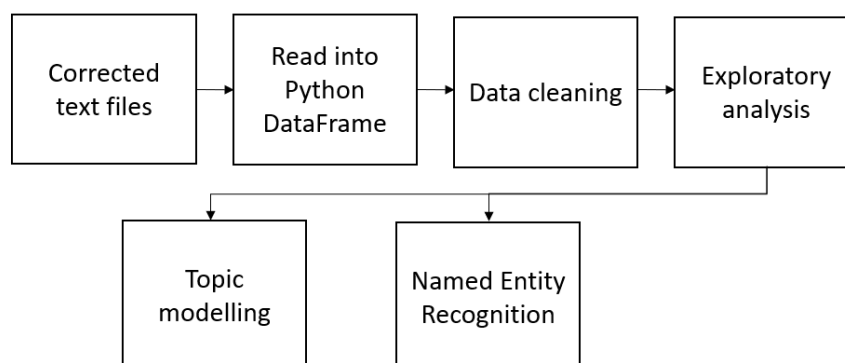


Figure 13: High level NLP pipeline

It was decided that a Pandas DataFrame would be the best way to store the corpus, with a row for each document and columns to contain metadata and processed text. Since the file list CSV already existed and contained metadata, it seemed the logical choice to use this as the basis for the DataFrame, and then insert the text from the OCR'd files into it. This could then be extended with columns for processed text. Although this could result in a very large DataFrame, it was decided that, as far as practicable, it would be worth preserving each step as a separate column so that potential future researchers could choose which processing steps they would like to have applied to the text. From reviewing workflows and tutorials from Havens (2020), Li (2018), Prabhakaran (2018), Rahmani (n.d.) and Zhao (2020) it was determined that the re and NLTK Python modules could be used for data cleaning and initial analysis, while Genism would be employed for topic modelling using Latent Dirichlet Allocation and SpaCY for Named Entity Recognition.

5.2 Solution

Text ingestion: 1.DataIngest.ipynb

The existing file list CSV containing metadata was loaded as a DataFrame using Pandas. The os module was used to list the OCR'd files. A for loop was used to open each file in turn and write the contents to a 'Text' column and the filename to an 'OCR filename' column.

```

#read each of the files in turn, adding them to the dataframe by index - index number will be based on 'index' counter
index = -1
for file in files:
    index = index+1
    with open (file, encoding='utf8') as f:
        content = f.read()
        df2.at[index, 'Text'] = content
        df2.at[index, 'OCR filename'] = file #added this so it will include filename of OCR'd text, to cross-check with original PDF filename
print(df2)

```

Figure 14: Adding the text to the DataFrame

Data Cleaning: 2.DataCleaning.ipynb

Several rounds of text cleaning and normalisation were then undertaken by applying a lambda function to the DataFrame, using the re module to remove punctuation, and change all text to lower case, this was adapted from Zhao (2020).

```

#I looked at examples from throughout the run of magazines and found
#these examples of other things to remove: "" _ ' \n\ ' @ ° ' »£$
def clean_text_round2(text):
    text = re.sub('[\"'\"_\"°\"£$]', '', text)
    text = re.sub('\n', ' ', text)
    return text

round2 = lambda x: clean_text_round2(x)

data_clean = pd.DataFrame(data_clean.Text.apply(round2))
data_clean

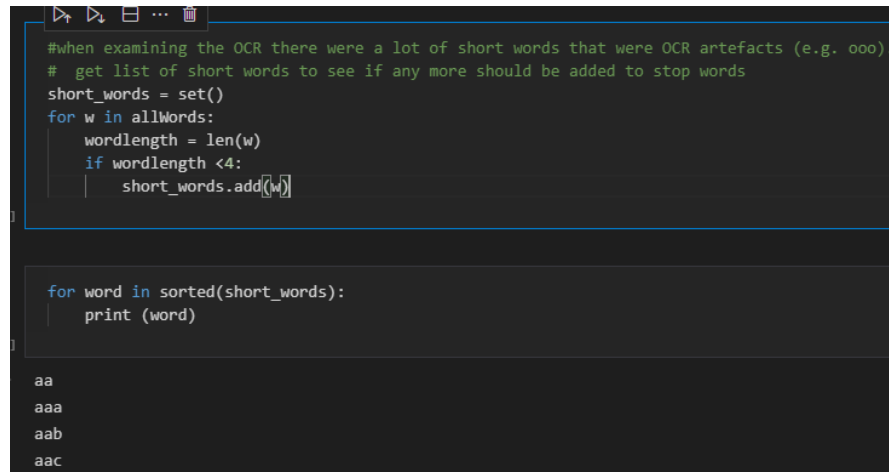
```

	Text
0	the polytechnic magazine christian workers ...
1	the polytechnic magazine regent street chris...
2	the polytechnic magazine christian workers ...

Figure 15 Data cleaning function

NLTK was then used to tokenize the text and the cleaned and tokenized text was added to a new column in the DataFrame ('tokenized_words'). NLTK was then used to perform a frequency analysis of the entire corpus. This was done as a way of getting an overview of the corpus to identify priorities for further cleaning. At this point NLTK's English stop word list was applied to the tokenized text to remove common words with little semantic value, the frequency analysis was then applied again to confirm the results of the stop words being removed. A further set of stop words that had been identified during the OCR checking were then removed from the tokenized text. It had also been noted during OCR that there appeared to be some examples of short 'words' that were in fact OCR artefacts. To identify these, all words shorter than four

characters were extracted into a list. Then, using a process adapted from Bird, Klein and Loper (2009), these were compared with an English dictionary from NLTK, and non-dictionary words were exported to a text file for review. Having reviewed the file, the decision was taken to remove all the identified words from the tokenized text.



```
#when examining the OCR there were a lot of short words that were OCR artefacts (e.g. ooo).
# get list of short words to see if any more should be added to stop words
short_words = set()
for w in allWords:
    wordlength = len(w)
    if wordlength < 4:
        short_words.add(w)

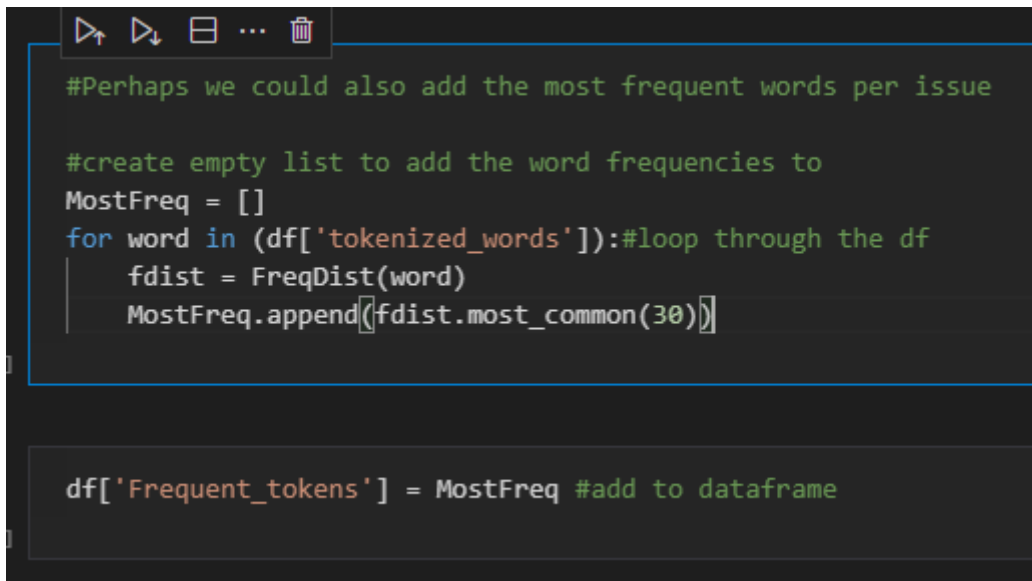
for word in sorted(short_words):
    print (word)

aa
aaa
aab
aac
```

Figure 16 Identifying short words for checking

Exploratory data analysis: 3.ExploratoryAnalysis.ipynb

NLTK was then used to perform another frequency analysis. It was noted that there were many single letter words present in the corpus, probably as a result of initials appearing in the original text (for example those of J.K. Studd, the president of the Polytechnic for more than 40 years). These were removed from the tokenized text. The len method was then used to establish the number of words per issue, this was added as a new column called 'Wordcount_tokens'. Seaborn was used to plot the number of words with the number of pages per issue. The most frequent words per issue were then added to the DataFrame using NLTK's FreqDest.

A screenshot of a Jupyter Notebook interface. The top part shows a code cell with the following Python code:

```
#Perhaps we could also add the most frequent words per issue

#create empty list to add the word frequencies to
MostFreq = []
for word in (df['tokenized_words']):#loop through the df
    fdist = FreqDist(word)
    MostFreq.append([fdist.most_common(30)])
```

The bottom part of the image shows a separate line of code in a different cell:

```
df['Frequent_tokens'] = MostFreq #add to dataframe
```

Figure 17: Adding most frequent tokens for each issue

In order to look at change over time, the 'Date' column was converted to a datetime and set as the index of the DataFrame. This was used to plot changes in word count and number of pages over time. The word count divided by the number of pages was added as a new column in the DataFrame. It was then noted that further frequency analysis would be valuable and so it was decided to continue this in an additional notebook. As a final step in the exploratory analysis, the entire unprocessed text was extracted into a list to demonstrate that concordances and dispersion plots could be generated with NLTK.

Frequency Analysis: 4.Frequency.ipynb

To continue the frequency analysis, it was decided to update the existing 'Frequent_tokens' column with a version containing the full list of frequent terms for each issue. Each row of the 'Frequent_tokens' column consisted of a list of tuples with the term and number of times it appears (e.g. [(love, 57), (god, 31) etc...]). To display a frequency by keyword that could be plotted over time, a for loop was used to traverse the column, with a loop within it that would go through the list of tuples, checking whether the first element of the tuple was the keyword, and if so add the frequency number to a list (adding '0' if the term did not appear).

```

#search for a keyword, and generate list of frequencies across issues
listoffreqs = []#create empty list to hold frequencies
keyword='war'#enter keyword to search for here
found = False #set boolean as false

for dflevel in df['Frequent_tokens']: #loop through each row
    found = False
    for lstlevel in dflevel:#within each row loop through list of tuples
        if lstlevel[0] == keyword:
            found = True
            listoffreqs.append(lstlevel[1])#if keyword found append the frequency r
    if found == False:
        listoffreqs.append(0)#if keyword not found append 0 to the list of frequenc

df['freq'] = listoffreqs#add list of frequencies as new column to df

```

Figure 18: Frequency by keyword

The list of frequencies was then added to the DataFrame, so it could be plotted against the Date index. As it was recognised that this method for generating the frequency data was rather cumbersome, a small function was developed that could be applied to the DataFrame to more efficiently generate the data for plotting.

```

#function to more efficiently implement the selection and processing of frequency based on keyword
def getFreq(list, keyword):
    for tup in list:
        if tup[0] == keyword:
            return tup[1]
    # if it's gone through the whole list and not found it return zero
    return 0

df[keyword] = df['Frequent_tokens'].apply([getFreq, args=[keyword]])

```

Figure 19: Function for frequency by keyword

Lemmatization: 5.Lemmatization.ipynb

Before proceeding with Topic Modelling, it was decided to lemmatize the text. This was Implemented using WordNetLemmatizer from NLTK.

```
#function to lemmatize text
def doLem(text):
    wordlist = []
    for word in text:
        word = lemmatizer.lemmatize(word)
        wordlist.append(word)
    return wordlist

df['Lemmatized_tokens'] = df['tokenized_words'].apply(doLem) #calling lemmatization function
```

Figure 20: Lemmatization

After initial topic modelling, it was decided to return to this notebook and generate bigrams and trigrams from the lemmatized text, to further improve results.

Topic Modelling: 6.TopicModelling.ipynb

To undertake topic modelling, the lemmatized tokens were converted to a string. Several different workflows were tried, ultimately settling on an adapted version of Zhao (2020). Topic modelling was undertaken with Gensim which requires a document-term matrix, and a dictionary of the location of the terms within the matrix. The document-term matrix was created using Sklearn's CountVectorizer and the string of lemmatized tokens. When creating the document-term matrix, the min_df and max_df values are used to specify words to be used in the matrix: with min_df being the number of documents a word must occur in to be counted and the max_df being the maximum number of documents that a word can occur in to be counted (Rahmani, n.d.). For this initial proof of concept, the values of min_df 10 and max_df 0.5 (50%) were selected, with the view that these would be parameters that could be selected by the user in the public notebooks.

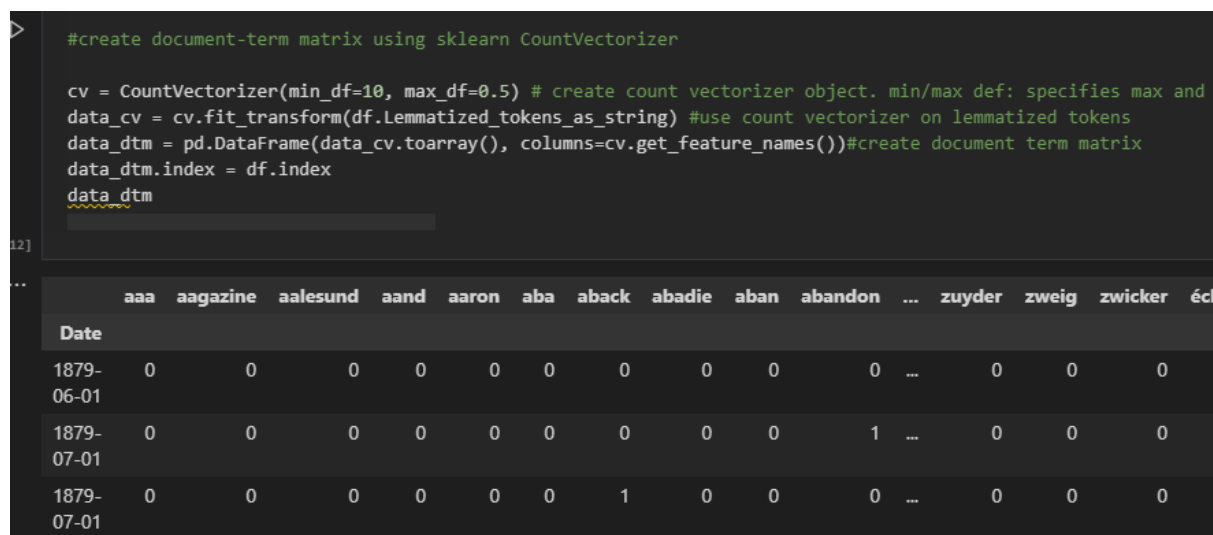


Figure 21: Creating the document-term matrix

The dictionary was then created with Gensim.

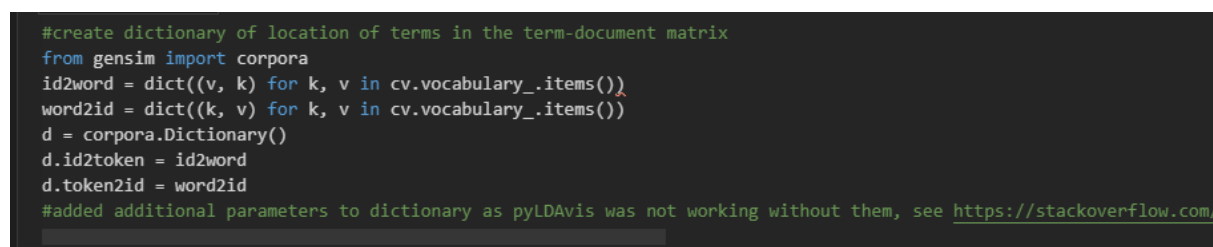


Figure 22: Creating the dictionary

Gensim's Latent Dirichlet Allocation model requires the user to specify in advance the number of topics in the model and the number of passes at training the model against the corpus. Several different combinations of topics and passes were experimented with, again with the view that these options could be selected by the user in the public version of the notebook.

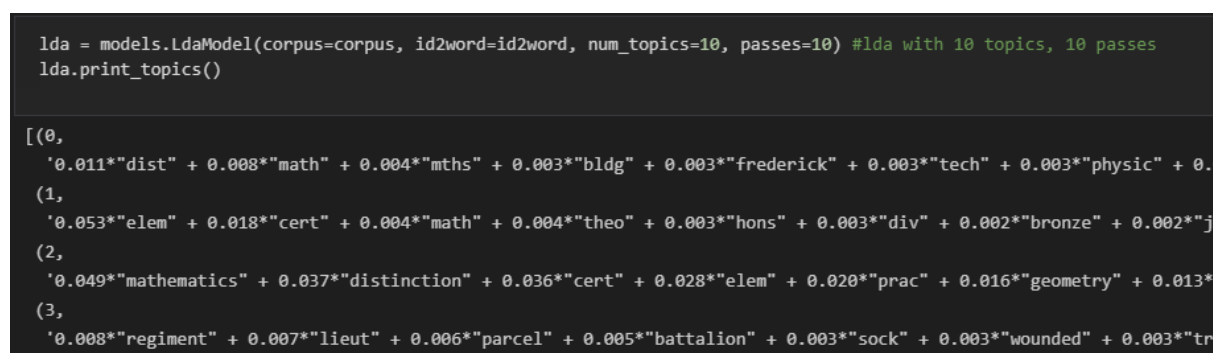


Figure 23: LDA model of the corpus, with 10 topics and 10 passes selected

The resulting topic models were then explored with PyLDAvis (Mabey, 2015).

Named Entity Recognition

Given the time that had been expended on Topic Modelling it was decided to deemphasise the Named Entity Recognition part of the project at this stage, with a view to incorporating it in the final phase if possible.

5.3 Testing, Results and Discussion

The pipeline that was implemented, diverged somewhat from the original plan, in three keys ways. Firstly, the exploratory analysis led to additional work on a new notebook focusing on frequency analysis. However, the additional time that was spent on this, together with the topic modelling phase, meant that the exploration of Named Entity Recognition was postponed to the final phase. Finally, the workflow in practice was less linear than planned, with additional text correction, and normalisation (such as lemmatization and the creation of bigrams and trigrams) taking place throughout as the results were refined.

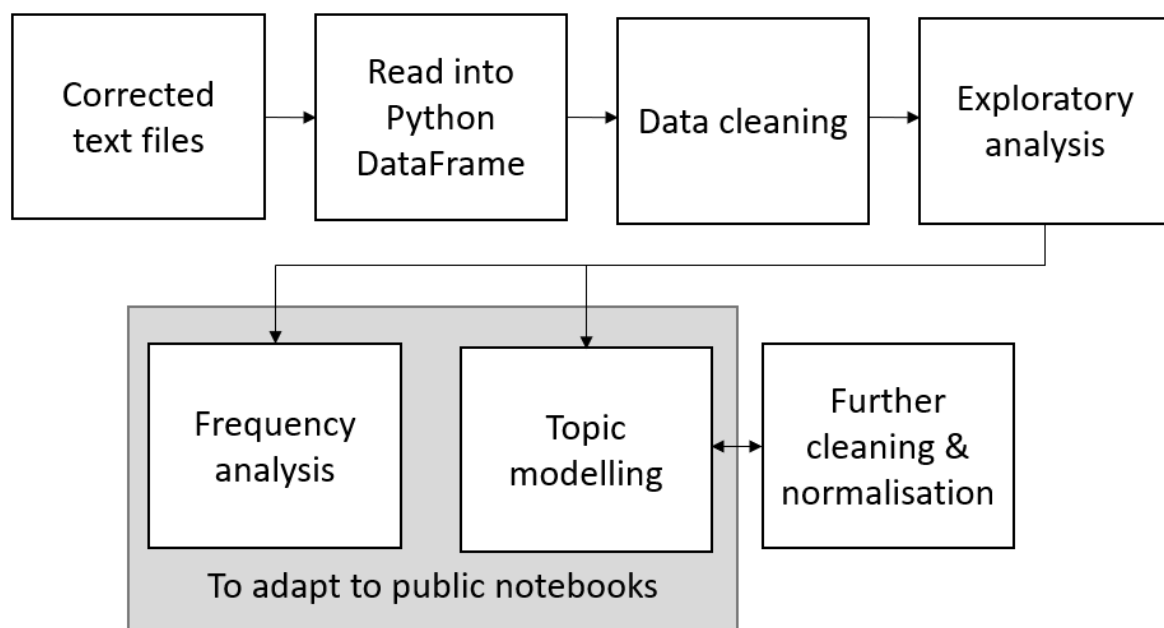


Figure 24: Actual workflow

Testing took place at each stage of the process. On ingest the DataFrame was printed to check that the files had been ingested correctly. On the initial ingest it was found that the OCR file names, and the existing metadata were not lining up as anticipated, so the sort order of the file list CSV was changed and it was ingested again, correcting the issue. The DataFrame was then checked to ensure there were no unexpected

empty fields. During each stage of data cleaning and normalisation the results were checked both by printing the text of individual issues, and by performing frequency analyses to ensure the entire corpus looked as expected.

During the exploratory analysis section, Seaborn was used to plot the data to check it. Plotting word counts vs number of pages unsurprisingly demonstrated a clear linear relationship between the two. This confirmed that the word count and pages columns were as expected.

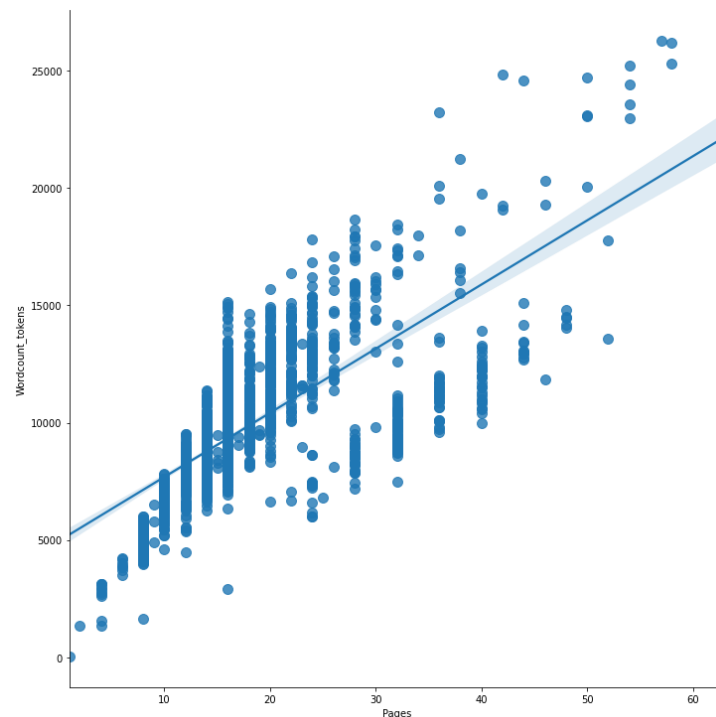


Figure 25: Number of words vs number of pages in the corpus

The plots of word counts and number of pages over time served as further confirmation that the data had been arranged correctly, as, for example, the notable decline in word counts during the Second World War made sense in the context of the historical events of the time.

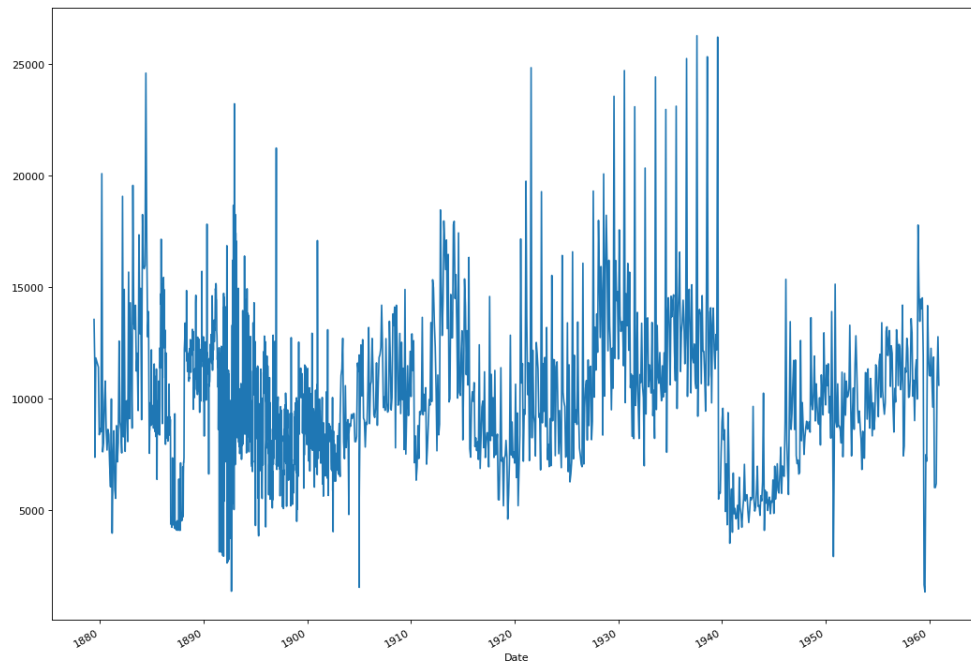


Figure 26: Word count over time

At the end of the exploratory analysis, accessing the unnormalized corpus was tested by running concordances and dispersion plotting using the test case of Mary Glen-Haig, an Olympic fencer who was a member of the Polytechnic. The output of these was found to match with our understanding of her sporting career and role at the Polytechnic.

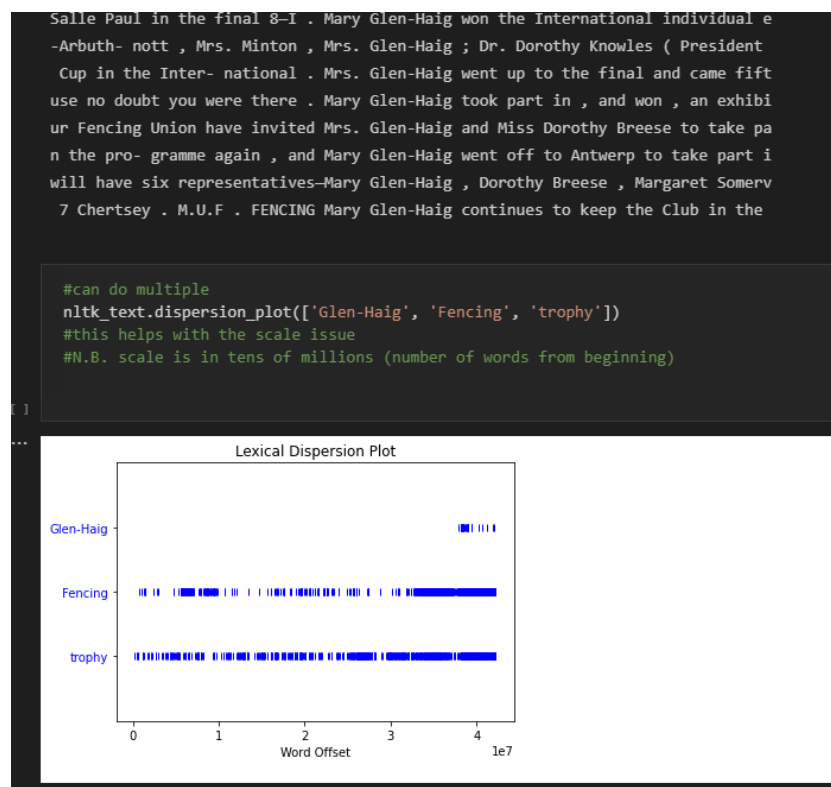


Figure 27: Concordances and lexical dispersion plots for Glen-Haig

As discussed, it was decided to spend some more time on developing a frequency analysis notebook, particularly the ability to plot frequency of keywords over time. This functionality was tested with various keywords that were found to fit with known historical information, for example with large spikes in references to 'war' during the period of the two World Wars.

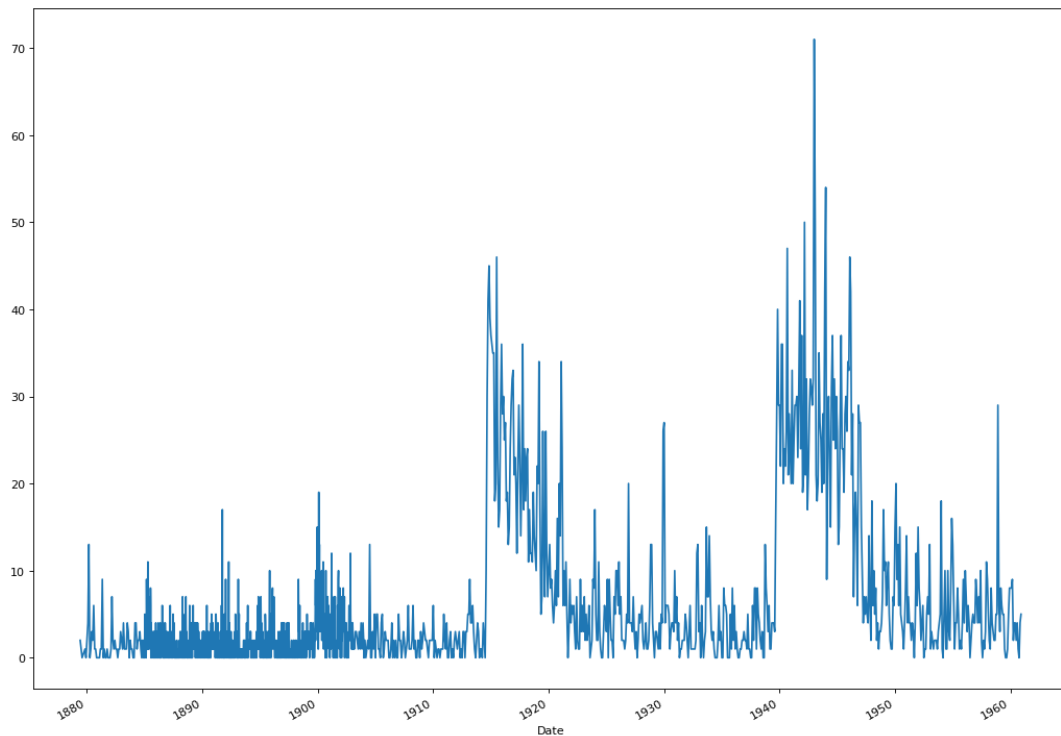


Figure 28: Frequency plot for 'War'

Plotting multiple search terms was also found to be possible by adapting the script, though as the data was quite noisy this could sometimes be hard to read so smoothing was applied using a rolling mean.

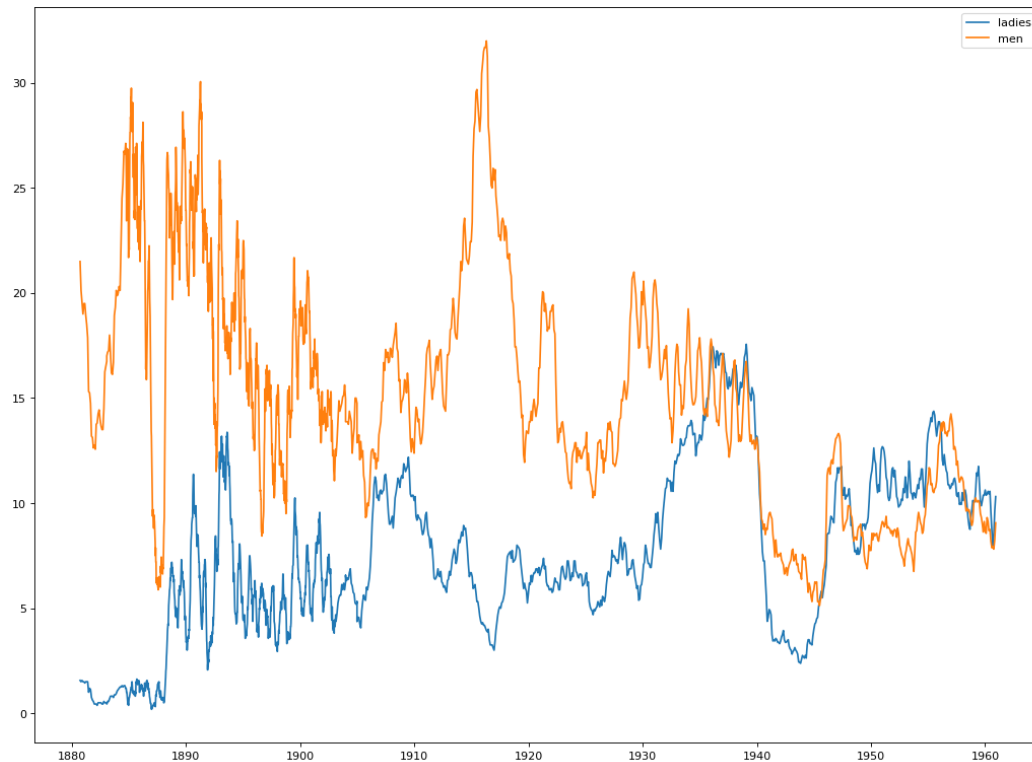


Figure 29: Smoothed frequency plot for 'ladies' and 'men'

Overall, the information gleaned from the frequency analysis notebook fit well with our understanding of the history of the Polytechnic.

Topic modelling proved to be one of the most time-consuming parts of the project. As well as the relative complexity of implementation, testing also took time as each topic model took at least five minutes to run, and often considerably longer. Topic modelling using workflows adapted from Prabhakaran (2018) and Rahmani (n.d.) was attempted but did not yield useable results. If more time had been available, it would have been desirable to spend further time debugging these workflows to come to a clear conclusion as to why they failed.

```

1 [(0,
2   '0.012*class" + 0.009*member" + 0.009*poly" + 0.007*club" + 0.006*time" '
3   '+ 0.006*first" + 0.005*last" + 0.005*cert" + 0.005*work" + 0.005*day"'),
4   (1,
5     '0.004*member" + 0.003*club" + 0.003*give" + 0.003*poly" + 0.002*time" '
6     '+ 0.002*man" + 0.002*year" + 0.002*hold" + 0.002*follow" + '
7     '0.002*last"'),
8     (2,
9       '0.002*member" + 0.002*poly" + 0.002*class" + 0.002*year" + 0.002*club" '
10      '+ 0.001*give" + 0.001*well" + 0.001*time" + 0.001*second" + '
11      '0.001*first"'),
12      (3,
13        '0.013*club" + 0.012*member" + 0.009*poly" + 0.008*team" + 0.008*year" '
14        '+ 0.008*time" + 0.007*first" + 0.006*well" + 0.005*play" + 0.005*new"'),
15        (4,
16          '0.012*member" + 0.007*last" + 0.006*give" + 0.006*time" + 0.006*man" + '
17          '0.005*well" + 0.005*club" + 0.005*evening" + 0.005*first" + '
18          '0.005*poly"'),
19          (5,
20            '0.116*pass" + 0.053*class" + 0.041*ist" + 0.029*first" + 0.027*year" + '
21            '0.017*distinction" + 0.012*second" + 0.006*mathematic" + 0.006*dist" + '
22            '0.005*design"'),
23            (6,
24              '0.009*man" + 0.006*member" + 0.005*give" + 0.005*well" + 0.004*time" + '
25              '0.004*life" + 0.004*day" + 0.004*last" + 0.004*love" + 0.004*work"'),
26              (7,
27                '0.013*member" + 0.010*poly" + 0.010*club" + 0.007*year" + 0.007*give" '
28                '+ 0.006*time" + 0.006*work" + 0.005*great" + 0.005*first" + '
29                '0.005*well"'),
30                (8,
31                  '0.009*member" + 0.006*man" + 0.006*give" + 0.006*time" + 0.005*well" + '
32                  '0.005*first" + 0.004*work" + 0.004*place" + 0.004*play" + '
33                  '0.004*great"'),
34                  (9,
35                    '0.003*member" + 0.002*poly" + 0.002*time" + 0.002*man" + 0.002*work" + '
36                    '0.001*club" + 0.001*well" + 0.001*give" + 0.001*meeting" + 0.001*day"')])
37

```

Figure 30: Example of failed topic model yielding almost identical topics

However, given the limited time available for the project it was decided to concentrate on adapting the simplest workflow from Zhao (2020) which did seem to produce meaningful results. It had been anticipated that pyLDavis would be used to visualise and interpret the results, this required some changes to how the Gensim dictionary was constructed but was ultimately successful.

Latent Dirichlet Allocation requires the user to specify the number of topics being sought in the topic model and the number of times the algorithm is applied to the corpus. The model also does not identify the meaning of the topics, with this requiring human interpretation. Therefore, some time was spent experimenting with running models with different numbers of topics and passes. This met with mixed success, with a number of topics tried ranging from four to twenty. As expected, a larger number of topics tended to yield many small overlapping topics which were difficult to meaningfully interpret. Nevertheless, there were some distinctive themes throughout,

for example Topic 1 in the below contained words such as ‘wicket’, ‘innings’ and ‘bat’ along with names such as ‘Ogilvie’ and ‘Ravani’. As George Ogilvie was the founder of the Polytechnic cricket society and Ted Ravani one of its leading players, clearly this topic was centred around cricket.

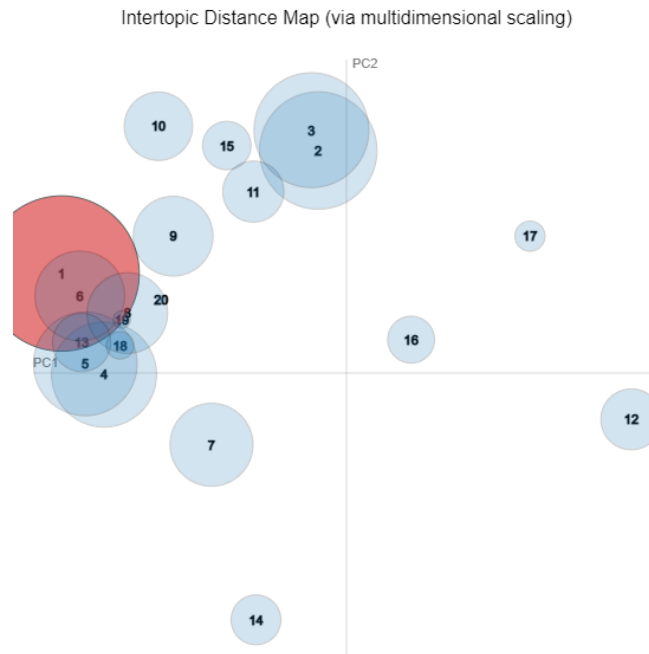


Figure 31 Examining a topic model of 20 topics with pyLDAvis

Larger numbers of topics also exposed other topics that were hidden with smaller numbers. One persistent theme that was usually present in models of five or more but absent from smaller models was war, unsurprising given the Polytechnic’s commitment to the war effort during the First and Second World Wars. One term that seems out of place in this topic is ‘sock’ but it in fact represents a key part of the Polytechnic’s war effort, with women members knitting socks that were sent to Polytechnic members serving at the front (McNally 2015). Similarly, there was almost always a topic containing words such as ‘distinction’, ‘certificate’ ‘elem’ (for elementary) along with the names of subjects taught at the Polytechnic, such as ‘mathematics’, ‘bookkeeping’ and ‘construction’.

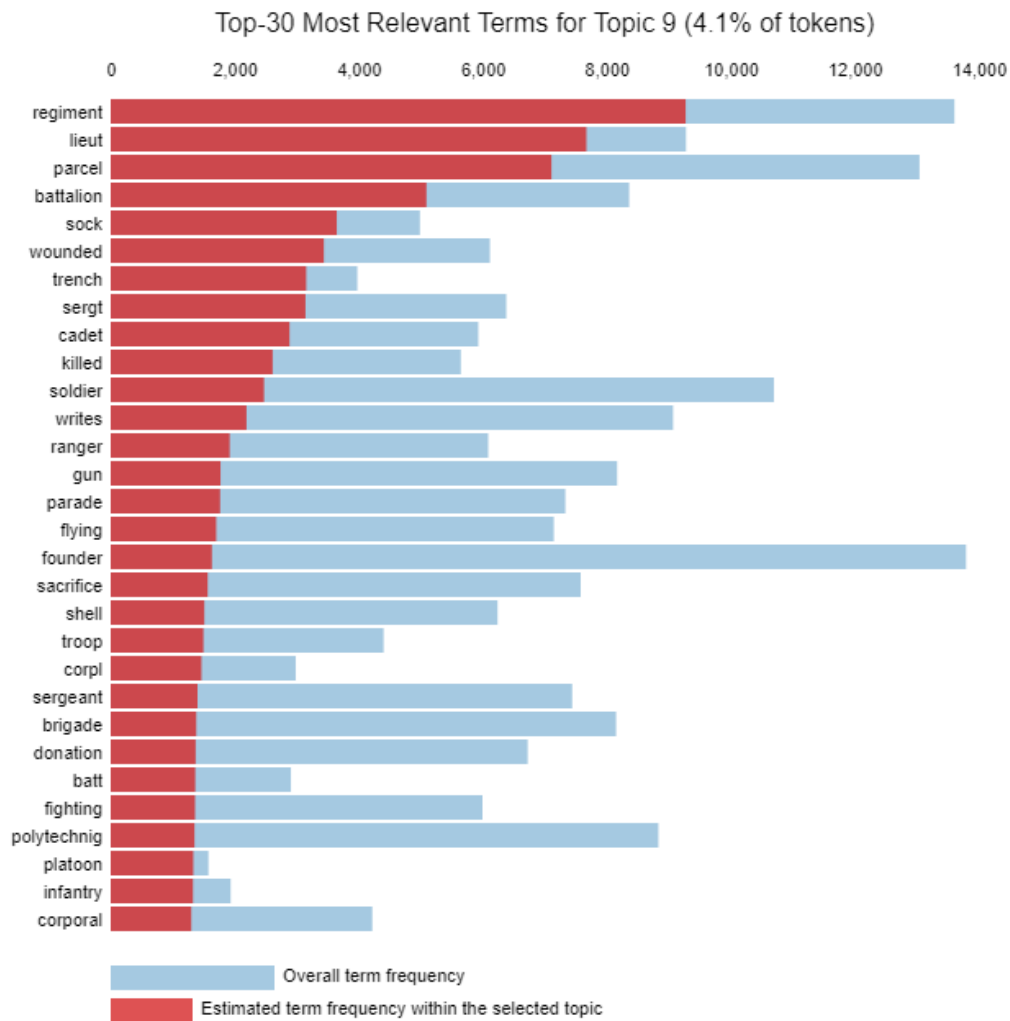


Figure 32: Examining the 'war' topic from a 10 topic LDA model

However, in all models, regardless of the number of topics, there were some topics which, were harder to distinguish, for example a persistent topic which seemed to combine terms that were common in other topics such as 'wicket' and 'geography' with more unusual words such as 'sleep', 'dream' and 'pray'.

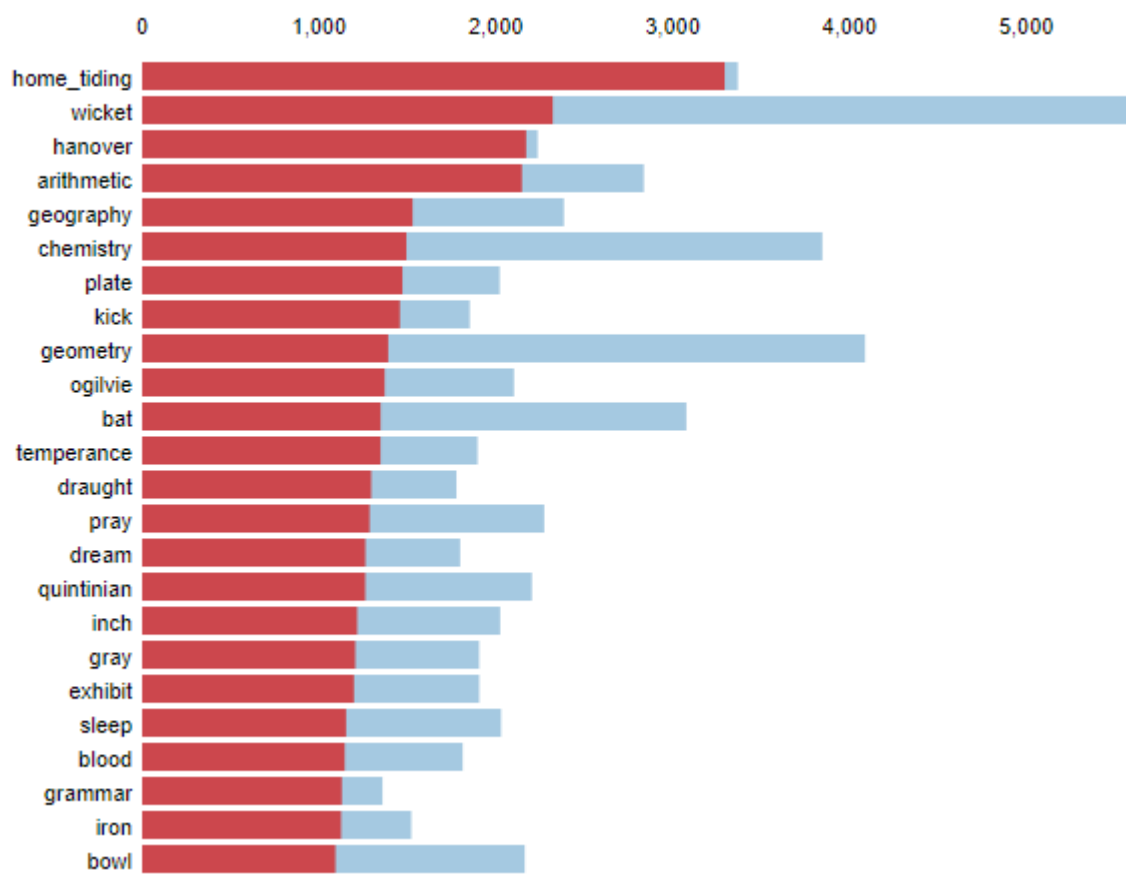


Figure 33: Example of a difficult to interpret topic

Nevertheless, it was felt that the topic modelling was sufficiently robust to be incorporated in the public notebooks, particularly as researchers would be able to use the parameters to experiment themselves and bring their own interpretation to the resulting topics.

6. Public Resources

The objective of the final stage of the project was to make resources available to users. This was to be accomplished in two main ways: firstly, by creating public Jupyter notebooks. The public notebooks would require no prior knowledge of digital analysis of texts and would serve as an introduction to our students and researchers to the corpus and some of the techniques that could be used with it. Secondly, the corrected OCR text files and the pickled DataFrame that contained the corpus, along with metadata and pre-processed text, was to be made available so that advanced users could incorporate them into their own analysis.

6.1 Methods

The first question to deal with was how to host the public Jupyter notebooks. While nbviewer offers an attractive static presentation of notebooks, it does not offer the user the opportunity to execute the code in the notebook. Binder offered a potential alternative but required more work on configuration than Google Colab, and it was not considered practical in the time available. Colab also offers the ability to include forms which provide an easy way for users to enter parameters, and this was seen as a big advantage in making the work more accessible. The significant drawback with using Colab as the main platform for disseminating the notebooks is in terms of sustainability, given that Google could potentially withdraw the product in future. To mitigate this risk, versions of the public notebooks were also stored on GitHub.

For the first public notebook it was envisaged that users would first install the modules and load the corpus into the notebook and then be able to choose from a variety of frequency analysis techniques to experiment with. As there had not been time during the second phase of the project to explore Named Entity Recognition, it was decided to add a short exploration of this technique to the first public notebook.

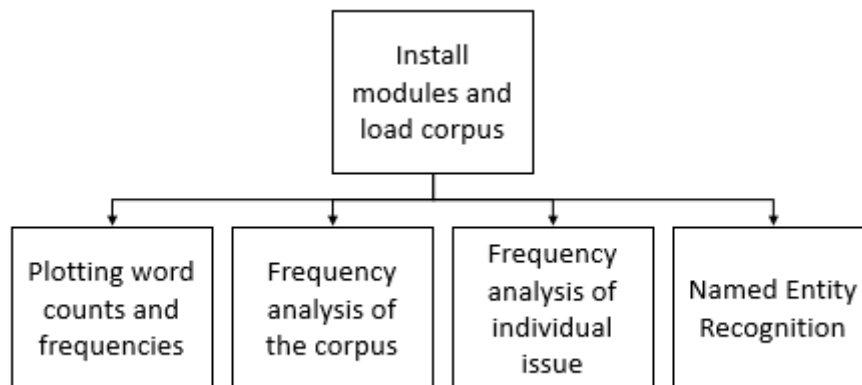


Figure 34: Workflow for the first public notebook

The second notebook would focus on Topic Modelling, allowing users to set the parameters for the document-term matrix and the topic model and then visualise their model with PyLDAvis.

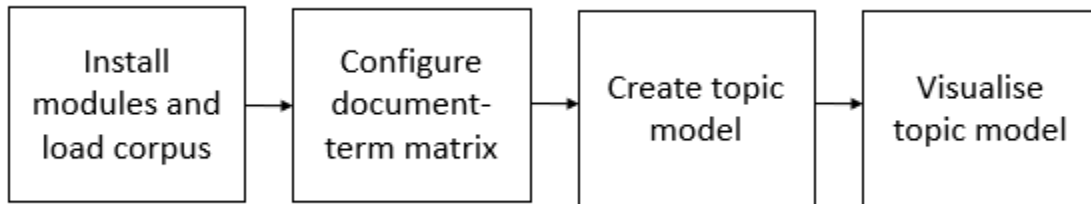


Figure 35: Workflow for the second public notebook

In addition to the public notebooks, it was envisaged that a GitHub page would be created to hold the other public outputs: the DataFrame and the OCR'd text files. In order to make it clear how the public notebooks and the DataFrame had been created, especially the text processing steps that had been taken, it was also decided to make available the notebooks used in the second phase of the project, labelling them as 'Preparatory notebooks'.

6.2 Solution

To access the corpus in the notebooks, the pickled DataFrame containing it was uploaded to Google Drive, it could then be downloaded to Colab using gdown and the pickled file opened as a DataFrame.

```

1 #@title Download corpus and open as a dataframe { display-mode: "form" }
2 #installing gdown to enable the notebook to download the corpus file from google drive
3 ! pip install gdown -q
4 ! gdown - id <file_id>
5 #downloading corpus file from google drive
6 import gdown
7 url = "https://drive.google.com/uc?id=1RrOn2iOr-H54K3n9QjMkwiY-sY_Pf2mY"
8 output = "corpus_8.pkl"
9 gdown.download(url, output, quiet=False)
10 #Opening the corpus as a dataframe, this is a data structure in Python that is a bit 1
11 #Each line represents one issue of the Polytechnic magazine.
12 #The rows contain metadata such as the date and volume of the issue, as well as extrac
13 with open('corpus_8.pkl', "rb") as fh:
14     df = pickle.load(fh)
15
/bin/bash: -c: line 0: syntax error near unexpected token `newline'
/bin/bash: -c: line 0: `gdown - id <file_id>'
Downloading...
From: https://drive.google.com/uc?id=1RrOn2iOr-H54K3n9QjMkwiY-sY_Pf2mY
To: /content/corpus_8.pkl
509MB [00:03, 149MB/s]

```

Figure 36: Accessing the corpus in Colab

Plotting word counts and frequencies in the first notebook was based on the work that had been done during the Exploratory Analysis and Frequency Analysis sections of the second phase of the project. The principal difference being that the public notebooks would enable users to specify the parameters for the plots themselves. Two

main plots were implemented, the first allows users to plot word counts, page counts and words per page over time. This plot used the existing DataFrame columns for those values, plotting them against the Date column using bokeh.

```

1 #@title Word count, page count and words per page over time { display-mode: "form" }
2
3 smoothing = 38 #@param {type:"slider", min:0, max:100, step:1}
4 Graph = "Word count" #@param ["Word count", "Page count", "Words per page"]
5 DateRangeTool = "No" #@param ["Yes", "No"]
6 Date_Range_Tool = False
7 if DateRangeTool == 'Yes':
8     Date_Range_Tool = True
9 #@markdown Select the graph you would like to display. Smoothing may make it easier to see
10
11
12 if Graph == 'Word count':
13     if smoothing > 0: #applies smoothing if selected
14         df['Wordcount_smoothed'] = df['Wordcount_tokens'].rolling(window=smoothing).mean()
15         df['Wordcount_smoothed'].plot(figsize=(1080, 607), range_tool=Date_Range_Tool) #number of
16     else:
17         df['Wordcount_tokens'].plot(figsize=(1080, 607), range_tool=Date_Range_Tool)
18
19 if Graph == 'Page count':
20     if smoothing > 0: #applies smoothing if selected
21         df['Pages_smoothed'] = df['Pages'].rolling(window=smoothing).mean()
22         df['Pages_smoothed'].plot(figsize=(1080, 607), range_tool=Date_Range_Tool)
23     else:
24         df['Pages'].plot(figsize=(1080, 607), range_tool=Date_Range_Tool)
25
26 if Graph == 'Words per page':
27     if smoothing > 0: #applies smoothing if selected
28         df['Words_per_page_smoothed'] = df['Words_per_page'].rolling(window=smoothing).mean()
29         df['Words_per_page_smoothed'].plot(figsize=(1080, 607), range_tool=Date_Range_Tool)
30     else:
31         df['Words_per_page'].plot(figsize=(1080, 607), range_tool=Date_Range_Tool)
32

```

Figure 37: Code for the first plot

The second plot was designed to work in a similar way to the Google books Ngram viewer (Google, n.d.), allowing users to specify keywords and plot their frequency in the corpus over time. This plot searches for the keywords in the existing Frequent_tokens column in the DataFrame and creates a new DataFrame containing these values to plot against the Date column using bokeh. Bokeh was selected over Seaborn (which had been used earlier) as it allowed for greater user interactivity, such as the option to filter by date range. Both plots also enable the user to select the level of smoothing they desired and apply this as a rolling mean.

```

9
10 #instead of adding 'freq' to existing df we will create a new one this w
11 freqDf = pd.DataFrame(index=df.index)#creates new df that is blank except
12
13 #gets search terms from the user
14 #@title Word frequency over time
15 keyword1 = "fencing" #@param {type:"string"}
16 keyword1 = keyword1.lower()
17 keyword2 = "harriers" #@param {type:"string"}
18 keyword2 = keyword2.lower()
19 keyword3 = "football" #@param {type:"string"}
20 keyword3 = keyword3.lower()
21 smoothing = 41 #@param {type:"slider", min:0, max:100, step:1}
22 DateRangeTool = "No" #@param ["Yes", "No"]
23 Date_Range_Tool = False
24 if DateRangeTool == 'Yes':
25     Date_Range_Tool = True
26
27 #@markdown Enter one or more keywords to display how frequently they occur
28
29
30 #applies getFreq function to create new dataframe columns based on the fr
31 def addColumn(keyword):
32     freqDf[keyword] = df['Frequent_tokens'].apply(getFreq, args=[keyword])
33     if smoothing > 0: #applies smoothing if selected
34         freqDf[keyword] = freqDf[keyword].rolling(window=smoothing).mean()
35
36 #calls the add column function with the keyword entered by the user
37 if keyword1 != "":
38     addColumn(keyword1)
39 if keyword2 != "":
40     addColumn(keyword2)
41 if keyword3 != "":
42     addColumn(keyword3)
43

```

Figure 38: Excerpt from the code for the second plot

The second section uses the Frequency Distribution object created during the second phase of the project to allow the user to search for individual terms and see how frequently they occur in the corpus, with a counter displaying the search term's rank.

```

found = False
for item in new_words_all:
    counter = counter+1 #counter to determine rank
    if item[0] == search_term: #if item found, write the item name and value in the freq dist to the parameter
        searchkey=item[0]
        searchvalue=item[1]
        rank = counter
        print ("The term [" ,searchkey, "] occurs", searchvalue,"times","it is the",(p.ordinal(rank)),"most frequent")
        found = True

```

Figure 39: Finding and displaying frequency of search term

Following on from this, users can print a list of the most common terms in the corpus with an option to also filter out the remaining top terms from the list.

```
1 #@title List the words in the corpus, in order of frequency
2 #@markdown Enter the number of words you would like to see. For example, to see the top 100 words, enter
3 TotalWords = 50#@param {type:"integer"}
4 #@markdown To remove the most common words from the list, enter a number in the FilterCommonWords field.
5 FilterCommonWords = 10 #@param {type:"integer"}
6 words_all = (fdist.most_common(TotalWords))
7 wordsdisplayed = (TotalWords - FilterCommonWords)
8 print('Displaying',wordsdisplayed,'words.',FilterCommonWords,'words have been filtered from the list.')
9 trimmedList_all = words_all[FilterCommonWords:]
10 print(trimmedList_all)
```

Figure 40: Displaying list of top words in the corpus

The filtered words are then used in subsequent cells to plot word clouds and a bar chart of the results.

The next section applies similar techniques to a single, user-selected, issue of the magazine. In the first cell, the user selects the date from a dropdown, the selected date is converted to a datetime object and matched with the Date column in the DataFrame to select the relevant issue of the magazine. Metadata held in the DataFrame, such as the issue type, issue and volume number (if applicable) and number of pages and words is then displayed, along with a parameterised URL to enable the user to view the scanned copy of the issue on the original Polytechnic Magazine website.

```
1 #@title Select an issue to work with { display-mode: "form" }
2 IssueDate = "1879-06-01" #@param [{"1879-06-01", "1879-07-01", "1879-07-01", "1879-08-01", "1879-11-01", "1879-11-18", "1896-11-25", "1896-12-02", "1896-12-09", "1896-12-16", "1896-12-22", "1896-12-29", "1896-12-30", "1936-02-01", "1936-03-01", "1936-04-01", "1936-05-01", "1936-06-01", "1936-07-01", "1936-08-01", "1936-09-01"}]
3 #@markdown Note that early editions of Home Tidings were published monthly and the day of publication is not
4 #convert selected string to date
5 date_time_str = IssueDate
6 date_time_obj = datetime.strptime(date_time_str, '%Y-%m-%d')
7
8 print('You have selected:')
9 if str(df.at[date_time_obj, 'Issue']) == 'nan':
10 | print('An issue of', df.at[date_time_obj, 'Type'])
11 else:
12 | print('Issue', df.at[date_time_obj, 'Issue'], 'of Volume', (int(df.at[date_time_obj, 'Volume'])), 'of the', df.at[date_time_obj, 'Page'])
13 print('This issue was', df.at[date_time_obj, 'Pages'], 'pages long, and the version in the filtered corpus contains', df.at[date_time_obj, 'Words'], 'words')
14 polyURL = 'https://polymags.westminster.ac.uk/PDFFiles/' + (df.at[date_time_obj, 'Filename'])
15 print('You can read the full digitised copy of this issue at', polyURL)
```

Figure 41: Selecting an issue to work with

The rest of the section then enables the user to filter out common words and displays a word cloud and bar chart in the same way as described above, but only for the selected issue. The final section of the first notebook provides a basic introduction to Named Entity Recognition using the issue selected above. The first part of this section

uses SpaCY to run NER on the document and displaCY to display the text with the identified entities highlighted.

```

1 #@title Identify and tag people and places in your selected issue
2 #@markdown This section uses [Named Entity Recognition](https://spacy.io/usage/linguistic-features) to
3 #@markdown entities such as people and places within your chosen issue. When you run this cell, the tex
4 #@markdown will be shown below with coloured tags to inticate entites.
5 #@markdown Note that you must have selected an issue in the section above to run this section.
6 FileContent = (df.at[date_time_obj,'Text'])
7
8 import spacy
9 from spacy import displacy
10
11 nlp = spacy.load("en_core_web_sm")
12
13 doc = nlp(FileContent)
14 colors = {"ORG": "linear-gradient(90deg, #aa9cfc, #fc9ce7)"}
15 options = {"ents": ["ORG"], "colors": colors}
16 displacy.render(doc, style="ent", jupyter=True)

```

Figure 42: Running NER with SpaCY

Finally, the user is given the option to display a filtered list of identified entities of common types (people, places, and organisations), with a view that this could be used for further work, such as plotting places to a map, and a proof of concept of this is linked to.

```

FileContent = (df.at[date_time_obj,'Text'])
#https://spacy.io/usage/linguistic-features#named-entities
nlp = spacy.load("en_core_web_sm")
#do nlp on selected text
doc = nlp(FileContent)
#create list of entities and lables
listofents = []
for ent in doc.ents:
    listofents.append([ent.text, ent.label_])

#filter list so it only includes selected entity type

listOfNEs = []

#geographical locations
if Ntype == 'Places':
    for item in listofents:
        if item[1] == 'LOC':
            listOfNEs.append(item)
        elif item[1] == 'GPE':
            listOfNEs.append(item)

```

Figure 43: Generating filtered list of entities

The second notebook focused on topic modelling, drawing on the work that had been developed during the topic modelling section of the second project phase. The first cells import libraries and download the corpus. As the version of the pickle library that is used in Google Colab was found to be incompatible with that used in PyLDAvis the

corpus is loaded in the form of a CSV rather than a pickle. The CSV is then read as a DataFrame.

```
1 #@title Download the Polytechnic Magazine corpus
2 #load corpus
3 import gdown
4 url = "https://drive.google.com/uc?id=1855DjLlxVI-k3vexxdgEIDtck\_PjvV3v"
5 output = "corpus_11.csv"
6 gdown.download(url, output, quiet=False)
7
8 df = pd.read_csv('corpus_11.csv')
```

Downloading...

From: https://drive.google.com/uc?id=1855DjLlxVI-k3vexxdgEIDtck_PjvV3v

To: /content/corpus_11.csv

966MB [00:16, 59.9MB/s]

Figure 44: Loading the corpus

The user is then able to set the parameters of the document-term matrix they would like to use for their topic model, crucially the MinDF and MaxDF parameters which determine the extent to which common and rare words in the corpus are included in the model, and the date range they would like to model. The document-term matrix is then created with Gensim using the selected parameters.

```
#create document-term matrix
cv = CountVectorizer(min_df=MinDf, max_df=MaxDf) #built count vectorizer with parameters from above
data_cv = cv.fit_transform(df_extractedDates)
dtm = pd.DataFrame(data_cv.toarray(), columns=cv.get_feature_names())
dtm.index = df_extractedDates.index

tdm = dtm.transpose()#transpose dtm

#put tdm into gensim corpus
sparse_counts = scipy.sparse.csr_matrix(tdm)
corpus = matutils.Sparse2Corpus(sparse_counts)

#create dictionary
id2word = dict((v, k) for k, v in cv.vocabulary_.items())
word2id = dict((k, v) for k, v in cv.vocabulary_.items())
d = corpora.Dictionary()
d.id2token = id2word
d.token2id = word2id
```

Figure 45: Creating the DTM

The user then selects the number of topics they would like the model to include and the number of passes of the algorithm to apply. The model is then created and the top words from the topics are displayed.


```

1 #@title Create the topic model
2 #@markdown This cell will create your topic model.
3
4 #@markdown Latent Dirichlet Allocation requires the user to specify the number of topics for the model to lo
5 NumberOfTopics = 5 #@param {type:"integer"}
6 #@markdown You can also specify how many times you would like the algorithm to pass over the document. A hig
7 NumberOfPasses = 10 #@param {type:"integer"}
8
9 #@markdown When you have finished setting your parameters run this cell to generate the topic model. Note th
10 #@markdown When the model has been generated, it will display the topics it has found and the words most clo
11
12 warnings.filterwarnings("ignore",category=DeprecationWarning)#suppresses deprecation warnings generated by p
13 lda = models.LdaModel(corpus=corpus, id2word=d, num_topics=NumberOfTopics, passes=NumberOfPasses, alpha = 'a
14
15 lda.print_topics()

```

Figure 46: Creating the topic model

In the final cell of the notebook the topic model is visualised using pyLDAvis, allowing the user to explore their model.

In addition to the public notebooks, a [GitHub page](#) was created to house the other elements of the public resources, provide contextual information and secondary materials such as the phase two notebooks. In the event the pickled DataFrame and the OCR'd files proved to be too large to easily store on GitHub so they were hosted on Google Drive, linked to from the GitHub page.

5.3 Testing, Results and Discussion

For each of the cells in the public notebook, a range of user-entered parameters were tested, and the functionality was found to work as expected. As in phase two, the word, page and words per page counts in the first notebook were in accordance with known historical trends. For example, the considerable drop in word and page counts during the Second World War. The very sharp drop in 1887 was also verified against the scanned PDFs, which showed a brief run of significantly shorter issues of the magazine in that period.

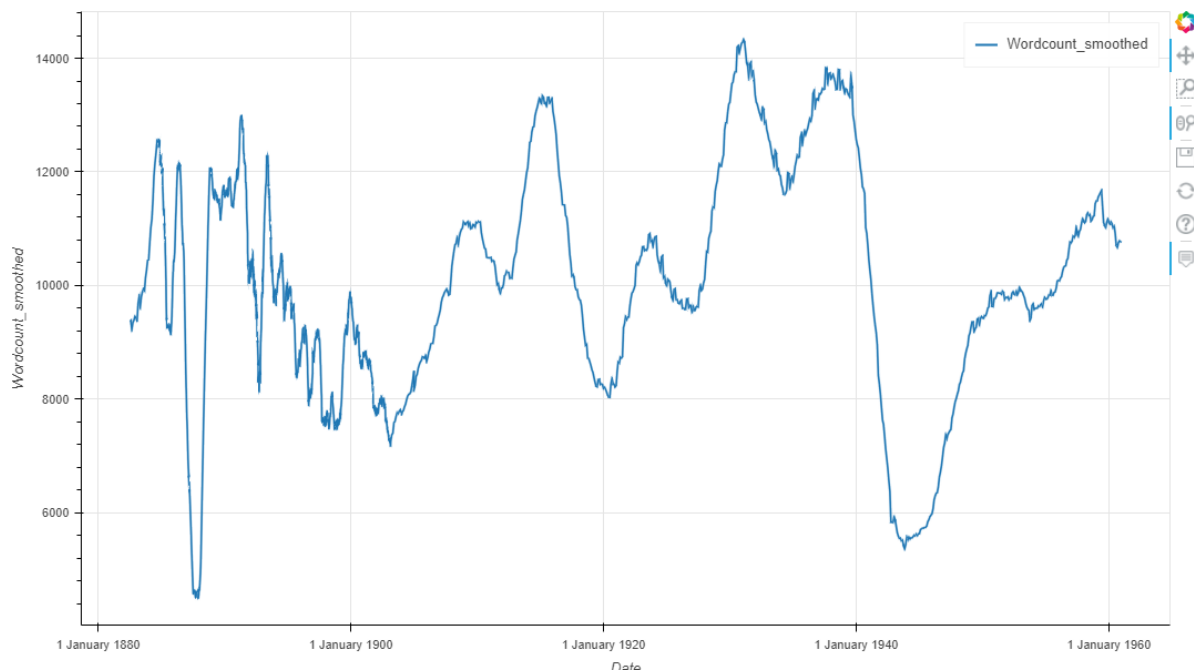


Figure 47: Word counts over time rendered by bokeh

Similarly, the frequency plot was tested with a selection of keywords. Terms relating to sports clubs showed an expected behaviour, with a significant drop-off in terms during both World Wars, and a significant rise in the interwar period.

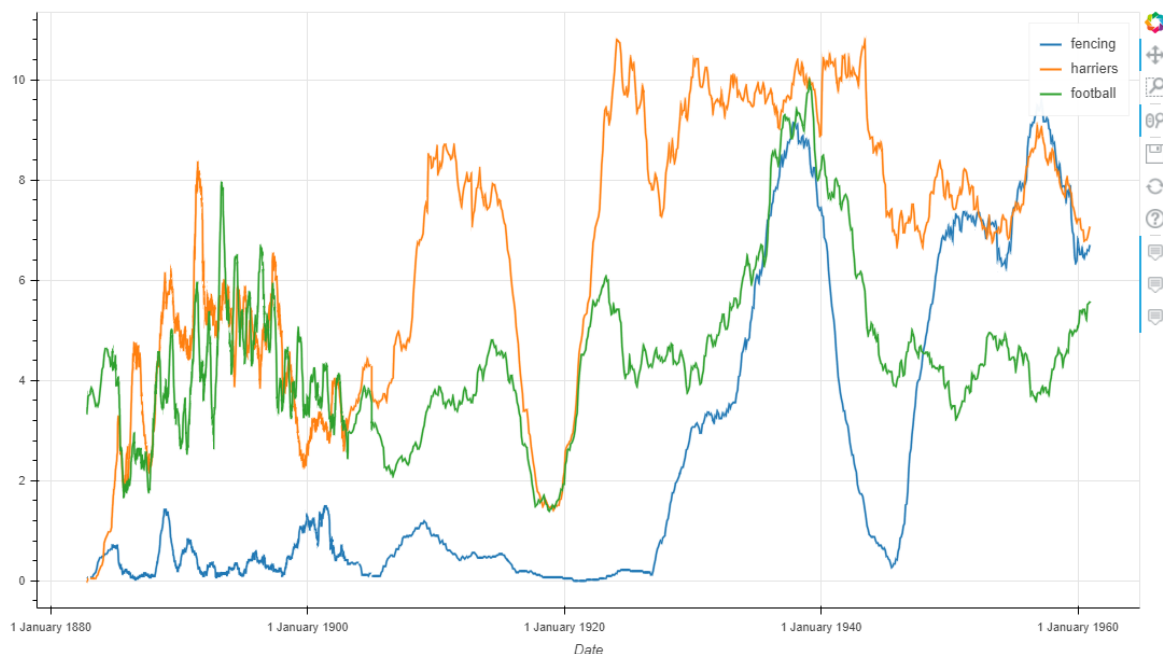


Figure 48: Testing the frequency plot with keywords based on sports clubs

The plot was also tested using the names of individuals associated with the Polytechnic. For example, the below chart accurately tracks the involvement of Quintin

Hogg's son and grandson, both know as Lord Hailsham, in the inter-war and post-war periods.

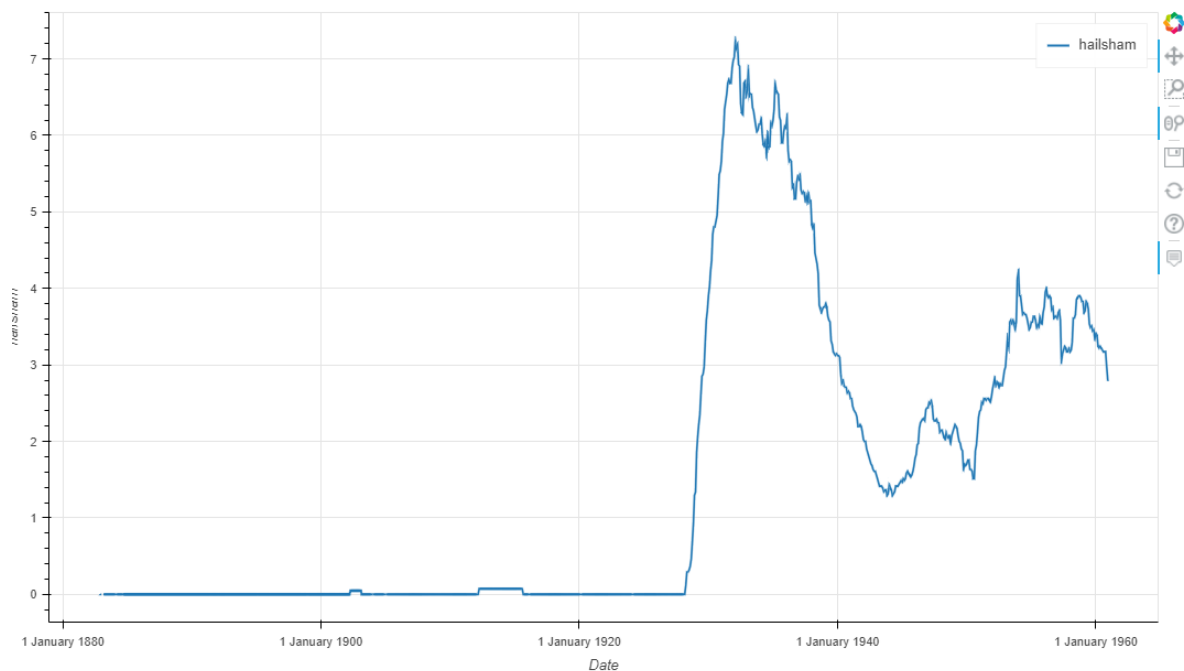


Figure 49: Frequency of 'Hailsham'

The overall frequency analysis functionality also worked as expected, displaying terms that we would expect to occur such as 'poly' and 'members' and filtering them out when the 'Filter common words' option was applied by the user.

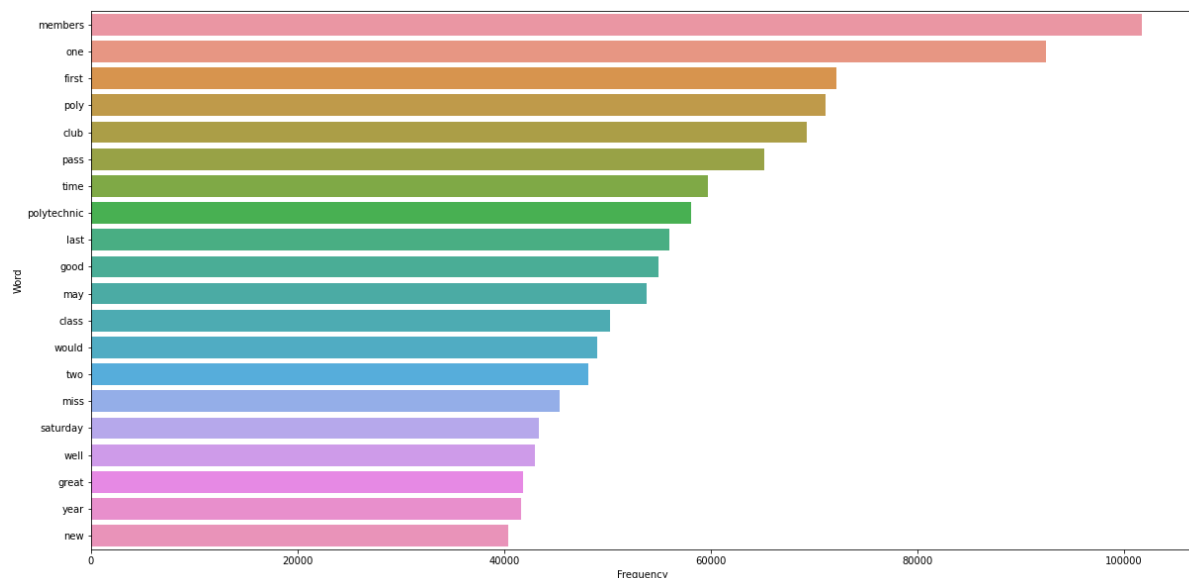


Figure 50: Unfiltered frequency analysis

Similarly, the results of frequency analysis for individual issues matches expectations, for example an analysis of the September 1914 issue includes prominent terms such

Implementing the topic modelling notebook was complicated due to compatibility issues between Colab and pyLDavis, however this was ultimately identified as being due the use of the pickle module and was mitigated by importing the corpus as a CSV. Topic modelling itself was implemented and tested in much the same way as during the second phase of the project and yielded similar results, with the important addition of the ability for users to set their own parameters.

Set parameters for including common and rare words

These parameters will allow you to control what words are included in your topic model.

MinDf determines the number of documents a word must occur in to be included, while **MaxDf** is the the maximum number of documents that a word can c while a high **MaxDf** will allow more common words from across the corpus, potentially resulting in more general topics.

You can either enter a whole number (e.g. 10) or a decimal which will be understood as a percentage of the corpus (e.g. 0.5 = 50%) So a MinDf of 10 would (i.e. 50%) would mean that each word may not appear in more than half of the documents

For a further discussion of how this works, see Adel Rahmani's excellent notebook on [Topic Modelling of Australian Parliamentary Press Releases](#).

MinDf:

MaxDf:

[Show code](#)

Select the date range you would like to model

The digitised run of the Polytechnic Magazine covers the years 1879 to 1960. You can select a date range to model, or use the entire corpus by selecting 18

StartDate:

EndDate:

[Show code](#)

Figure 53: User interface for DTM creation

The ability to set a date range was added as a new feature for the public notebooks and this appears to work as expected, with, for example, the 'war' topic showing greater prevalence for the years 1914 - 45.

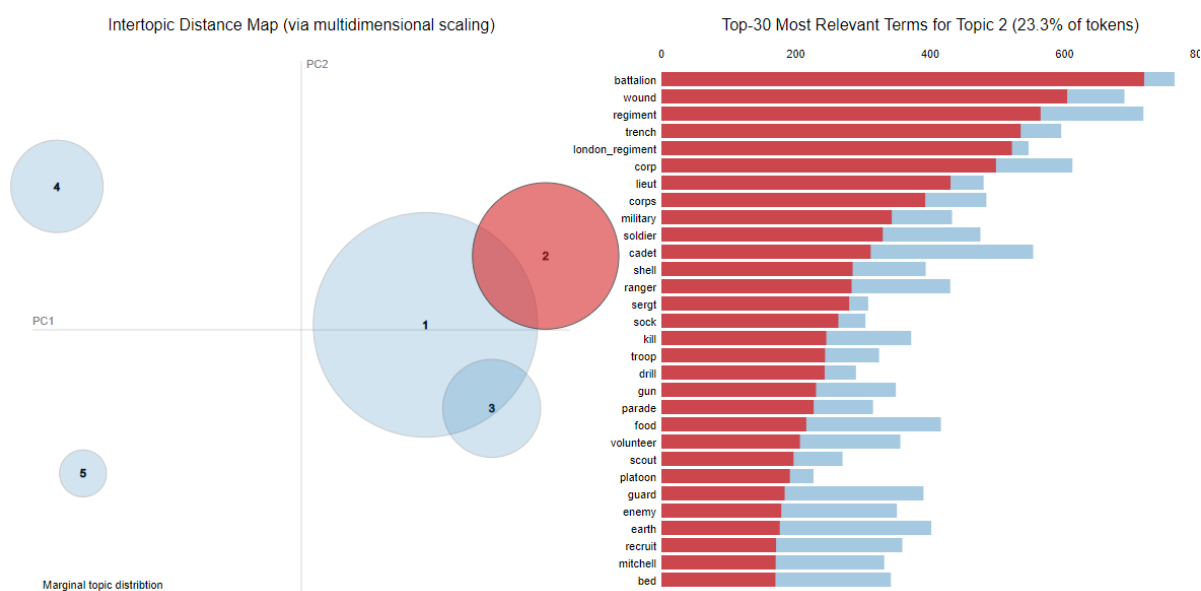


Figure 54: Topic model for 1914-45, with the 'war' topic highlighted

The notebooks were demonstrated to a member of University Archive staff, with some user feedback incorporated into the notebooks, such as the ability to filter out additional common words from the frequency analysis. Overall feedback from the demo was positive and the use of the notebooks as a teaching resource for history students will be explored. Finally, the other public resources were successfully hosted on Google Drive and [GitHub](#).

7. Future work

There are clearly some areas of the project that could benefit from further work. It is felt that the decision to include an additional notebook dealing with Frequency Analysis was justified by the results, and that the public notebook derived from it is a particularly useful starting point for potential researchers to access the corpus. However, this decision, combined with the time invested in OCR (when it had originally been hoped to use the existing OCR'd text from the PDFs), meant that there was less time available to work on the Topic Modelling and Named Entity Recognition elements of the project. Topic Modelling proved to be extremely time consuming and if time had been available to debug alternative workflows, it would have been possible to explore the results of the topic model in more detail. Future projects could benefit from, for example, modelling the prevalence of topics over time (as seen in Havens, 2020). In the event the Named Entity Recognition that was implemented was also rather basic, and although useful as a proof of concept, it is easy to see how future projects could take it further, for example by mapping places mentioned in the corpus over time. The quality of the NER itself could also probably be improved through training the model. There are also some concerns over the sustainability of aspects of the project outputs, especially that the public notebooks rely on being hosted by Google Colab. Future work could focus on migrating them to other hosting solutions such as Binder. While the OCR phase delivered a corpus that could be worked with, it is also an area that could be revisited as technology in this area continues to progress.

8. Conclusion

In conclusion this project has broadly achieved its objectives. The Polytechnic Magazine corpus was successfully transformed with OCR into a format that can be used for computational analysis. The Natural Language Processing experiments that

were undertaken demonstrate what is possible with the corpus and will hopefully be the first of many attempts to use computational techniques to examine this rich historical source. The public notebooks provide an accessible entry point into working computationally with the text and they will provide a useful introduction to our students and researchers. Finally, much has been learnt through the implementation of the project that can be applied to future work in bringing new techniques to bear on our historic collections.

References

The Alan Turing Institute, (n.d.), *Living with Machines*. Available online at <https://www.turing.ac.uk/research/research-projects/living-machines>

Always Already Computational: Collections as Data, (2017), *The Santa Barbara Statement on Collections as Data*. Available online at <https://collectionsasdata.github.io/statement/>

Archives Unleashed, (2021), *Welcome to the Archives Unleashed Project*, Available online at <https://archivesunleashed.org/>

Andrew Akhlaghi, (2021), 'OCR and Machine Translation', *The Programming Historian* 10. Available online at <https://doi.org/10.46430/phen0091>

Artefactual Systems, (n.d.), *AtoM 2.5 API Overview*. Available online at <https://www.accesstomemory.org/en/docs/2.5/dev-manual/api/api-intro/#api-intro>

Monica Barget, (2020), *Doing Digital History with Python III: topic modelling with Gensim, spaCY, NTLK and SciKit learn*. Available online at <https://dhlabs.hypotheses.org/1693>

Steven Bird, Ewan Klein and Edward Loper, (2009), *Natural Language Processing with Python*, Sebastopol, CA: O'Reilly

Jonathan Blaney, Sarah Milligan, Marty Steer and Jane Winters, (2021), *Doing Digital History*, Manchester: Manchester University Press

Christopher Day, (2020), *Computing Cholera: Topic modelling the catalogue entries of the General Board of Health*. Available online at <https://blog.nationalarchives.gov.uk/computing-cholera-topic-modelling-the-catalogue-entries-of-the-general-board-of-health/>

The Digital Preservation Coalition (2021), *Digital Preservation Coalition Rapid Assessment Model (version 2 - March 2021)*. Available online at <http://doi.org/10.7207/dpcram21-02>

Google, (n.d.), *Google Books Ngram Viewer*. Available online at <https://books.google.com/ngrams>

Ted Han and Amanda Hickman, (2019), *Our Search for the Best OCR Tool, and What We Found*. Available online at <https://source.opennews.org/articles/so-many-ocr-options/>

Lucy Havens, (2020), *Exploring Britain and UK Handbooks*. Available online at [https://data.nls.uk/wp-content/uploads/2020/10/Exploring Britain and UK Handbooks.html](https://data.nls.uk/wp-content/uploads/2020/10/Exploring_Britain_and_UK_Handbooks.html)

ImageMagick Studio, (2021), *ImageMagick*. Available online at <https://imagemagick.org/index.php>

Brewster Kahle, (2020a), *Can You Help us Make the 19th Century Searchable?* Available online at <https://blog.archive.org/2020/08/21/can-you-help-us-make-the-19th-century-searchable/>

Brewster Kahle, (2020b), *FOSS wins again: Free and Open Source Communities comes through on 19th Century Newspapers (and Books and Periodicals...)*. Available online at <https://blog.archive.org/2020/11/23/foss-wins-again-free-and-open-source-communities-comes-through-on-19th-century-newspapers-and-books-and-periodicals/>

Milan van Lange, (2019), *Integrating New Methods in Historical Research (part 2): Exploring Newspapers with Topic Modeling*, Available online at <https://niodbibliotheek.blogspot.com/2019/05/integrating-new-methods-in-historical-research-2.html>

Layout Parser Contributors, (2020), 'Model zoo' in *Layout Parser 0.2.0 documentation*. Available online at <https://layout-parser.readthedocs.io/en/latest/notes/modelzoo.html>

Susan Li, (2018), *Topic Modeling and Latent Dirichlet Allocation (LDA) in Python*. Available online at <https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24>

Ben Mabey, (2015), *pyLDavis documentation*, Available online at <https://pyldavis.readthedocs.io/en/latest/readme.html>

Moritz Mähr, (2020), 'Working with batches of PDF files', *The Programming Historian* 9. Available online at <https://doi.org/10.46430/phen0088>

Anna McNally. (2015), 'Discovering Knitting at the Regent Street Polytechnic, 1898-1948', *Cloth and Culture*, Volume 12, 2014 - Issue 1. DOI: <https://doi.org/10.2752/175183514x13916051793479>

Ian Milligan, (2019), *History in the Age of Abundance?* Montreal & Kingston, London, Chicago: McGill-Queen's University Press

Kiettiphong Manovisut, (2020), *python-word-error-rate*. Available online at <https://github.com/imalic3/python-word-error-rate>

Laura Turner O'Hara, (2013), "Cleaning OCR'd text with Regular Expressions," *The Programming Historian* 2. Available online at <https://doi.org/10.46430/phen0024>

Trevor Owens, (2018), *The Theory and Craft of Digital Preservation*. Baltimore: John Hopkins University Press.

Pdfplumber documentation, (n.d.), *pdfplumber 0.5.28*. Available online at <https://pypi.org/project/pdfplumber/>

Kim Pham, (2017), 'Web Mapping with Python and Leaflet', *The Programming Historian* 6. Available online at <https://doi.org/10.46430/phen0070>

PyPDF documentation, (n.d.), *pyPdf 1.13*. Available online at <https://pypi.org/project/pyPdf/>

Selva Prabhakaran, (2018), *Topic Modeling with Gensim (Python)*. Available online at <https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/>

Adel Rahmani, (n.d.), *Topic Modelling of Australian Parliamentary Press Releases*. Available online at <https://nbviewer.jupyter.org/github/adelr/trove-refugee/blob/master/Analyses.ipynb>

Radim Řehůřek, (2021), *Gensim documentation*, available online at https://radimrehurek.com/gensim/auto_examples/index.html

Andrew Salway and James Baker, (2020), 'Investigating Curatorial Voice with Corpus Linguistic Techniques', *Museum and Society*, 18 (2). Available online at <https://journals.le.ac.uk/ojs1/index.php/mas/article/download/3175/3138>

Shannon Shen (n.d.), *Deep Layout Parsing*. Available online at <https://github.com/Layout-Parser/layout-parser/blob/master/examples/Deep%20Layout%20Parsing.ipynb>

Zejiang Shen, Ruochen Zhang, Melissa Dell, Benjamin Charles Germain Lee, Jacob Carlson, Weining Li, (2021), 'LayoutParser: A Unified Toolkit for Deep Learning Based Document Image Analysis', *arXiv:2103.1534*. Available online at <https://arxiv.org/pdf/2103.15348.pdf>

Tim Sherratt, (2021), *Welcome to the GLAM Workbench*. Available online at <https://glam-workbench.net/>

Daniel van Strien, Kaspar Beelen, Mariona Coll Ardanuy, Kasra Hosseini, Barbara McGillivray, Giovanni Colavizza, (2020), 'Assessing the impact of OCR quality on downstream NLP tasks', *ICAART 2020 - Proceedings of the 12th International Conference on Agents and Artificial Intelligence*. Available online at <https://doi.org/10.17863/CAM.52068>

Leontien Talboom and David Underdown, (2019), 'Access is What we are Preserving': *But for Whom?* Available online at <https://www.dpconline.org/blog/access-what-we-are-preserving>

Tesseract contributors, (n.d.), *Tesseract Documentation*, Available online at <https://tesseract-ocr.github.io/tessdoc/#500x>

The National Archives, (2017), *Digital Strategy*. Available online at <https://www.nationalarchives.gov.uk/documents/the-national-archives-digital-strategy-2017-19.pdf>

Martin Thoma, (2013), *Word Error Rate Calculation*. Available online at <https://martin-thoma.com/word-error-rate-calculation/>

David Underdown, (2018), *Using the Discovery API to analyse catalogue data*. Available online at <https://blog.nationalarchives.gov.uk/using-the-discovery-api/>

Daniel D. Walker, William B. Lund, and Eric K. Ringger, (2010), 'Evaluating Models of Latent Document Semantics in the Presence of OCR Errors', *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 240–250. Available online at <https://www.aclweb.org/anthology/D10-1024.pdf>

University of Westminster, (2011), *Polytechnic Magazine Website*. Available online at <https://polymags.westminster.ac.uk/>

Jane Winters and Andrew Prescott, (2019), 'Negotiating the Born-Digital: A Problem of Search', *Archives and Manuscripts: Special Issue: After the Digital Revolution*, 47(3):391-403

Alice Zhao, (2020), *Natural Language Processing in Python Tutorial*. Available online at <https://github.com/adashofdata/nlp-in-python-tutorial>