Project 2: How Good (Positive and Patriotic) is Australia?

Submission deadline: 5:00 pm, Friday 23rd October 2020

Value: 20% of CITS1401

To be completed individually.

You should construct a Python 3 program containing your solution to the following problem and submit your program electronically on Moodle. No other method of submission is allowed. Your program will be automatically tested on Moodle. Remember your first two checks against the tester on Moodle will not have any penalty. However any further check will carry 10% penalty per check.

You are expected to have read and understood the University's guidelines on academic conduct. In accordance with this policy, you may discuss with other students the general principles required to understand this project, but the work you submit must be the result of your own effort. Plagiarism detection, and other systems for detecting potential malpractice, will therefore be used. Besides, if what you submit is not your own work then you will have learnt little and will therefore, likely, fail the final exam.

You must submit your project before the submission deadline listed above. Following UWA policy, a late penalty of 5% will be deducted for each day (or part day), after the deadline, that the assignment is submitted. No submissions will be allowed after 7 days following the deadline except approved special consideration cases.

Context:

For this project, imagine for a moment that you have successfully completed your UWA course and recently taken up a position for the Department of Prime Minister and Cabinet in Canberra with the Australian Federal Government. At first you were quite reluctant to leave Perth to move 'over east' and, more generally, wondered what use a new graduate with a heavy focus on computing, programming and data could be to this department. Regardless, the opportunity to gain experience in the 'real world' was too good, and although it is not quite your own multi-million dollar technology start-up, there was no way you weren't taking up the offer.

Your first few weeks of orientation was a mostly blur. However, one thing you noticed was that any time you mentioned your skills in programming, and with Python[1] in particular, to any senior bureaucrat, or even some of the savvier politicians, their eyes seemed to 'light up' and they suddenly became much more interested in whatever you

1 Actually their eyes are more likely to light up if / when you mention your skills in data science and machine learning and big data, for all of which Python is basically the foundational tool for.

were saying to them. After reflecting on these experiences, maybe there would be some even more interesting opportunities for you in the near future?

However, for now you decide to put aside these, as it's not like the work that you have been doing already has not been interesting, and this is what you need to focus on for today. At an early morning meeting with your immediate supervisor, you were told that the Government is very interested in reducing its spend on trying to understand what (and how) the Australian population currently thinks about it. Instead of spending millions of dollars calling randomised groups of Australian residents every quarter to ask about their opinions on various Government services, many senior bureaucrats have wondered for a while now whether there was any way to use the masses of freely available data on the internet to provide similar insights at a fraction of the cost.

It is within this context that your supervisor has asked you to develop a program, as a proof-of-concept, to demonstrate that it is possible to provide some of these insights at a much lower cost. At your meeting your supervisor noted that, for the proof-of-concept stage, the use of any 'live' internet data will not be possible without approval from the legal team (as well as possibly many others). This seemed like quite an obstacle until you thought back to one of your early Python units (maybe this one?) and remembered that there is an open source, freely available corpus collection of billions of recently crawled websites called the Common Crawl (http://commoncrawl.org/). More specifically the Common Crawl corpus consists of tens of thousands of files saved in a certain format (the WARC format, see below), each of which contains the raw HTML of tens of thousands of web pages from a web 'crawl' performed in the recent past. Being open source this data is free for you to use so with it you can immediately begin building your proof-of-concept.

The Project:

As your program is to be a proof-of-concept, both you and your supervisor decided that its scope should be kept as narrow as possible (but, of course, it must be broad enough so that it can successfully demonstrate some really good insights). For this reason, it was decided that your program is to focus only on providing four insights only:

1. How 'positive' is Australia generally?
2. How 'positive' does Australia feel towards their Government specifically?
3. How 'patriotic' is Australia compared with two other major English speaking countries – UK and Canada?
4. What are the most referred-to websites (domains) by all Australian websites (your team may want to use this information in the future to better understand how 'influential' each Australian web result is to your insights, i.e. highly-referred to web domains should be counted as more influential, and lowly-referred to web domains should be counted as less influential).

As outlined in the 'context' section, in order to generate these insights (which will be discussed in greater detail later in this document), your program will need to examine

the raw HTML from large quantities of Australian web pages, and such information is available in WARC format from the Common Crawl.

The Common Crawl and WARC format:

The WARC (Web ARChive) format is a standard format for mass storage of large amounts of 'web pages' within a single file. The Common Crawl makes the results of their crawl freely available for download in this format (as well as the WAT and WET formats, which will not be used for this project). For this project we will use WARC files from the August 2020 crawl (https://commoncrawl.org/2020/08/august-2020-crawl-archive-now-available/). In order to access these files you need to download the "WARC files" list – which you can access by clicking on the "CC-MAIN-2020-34/warc.paths.gz" hyperlink in the table in the August 2020 crawl homepage.

Clicking on this link will download an archive, which, when opened, will contain a text file. Once you open the text file you can download any of the WARC files from the common crawl by appending https://commoncrawl.s3.amazonaws.com/ to the front of any of the lines of this file and pasting this full address into your browser.

A couple of notes about the Common Crawl WARC files as discussed so far:

- The file list and all Common Crawl WARC files are compressed using gzip. These files can be unzipped automatically if you are using Linux or Mac OSX. For Windows you will have to download a free application to do this - try 7-Zip: https://www.7-zip.org/.
- The Common Crawl WARC files are very large – approximately 900MB compressed and up to 5GB uncompressed. Each file contains approximately 45,000 individual crawl results.

Due to the size of the files above, this project has made available a massively cut down sample Common Crawl WARC file on LMS as well as Moodle server. It is expected you will use this file to get familiar with the format and for your (initial) testing of your project. However, your submission will be tested with other WARC files.

To start getting familiar with WARC files, it is recommended you download the sample file and open it in a text editor (for Windows, Wordpad performs better; you can also use Thonny). You will see that a WARC file consists of an overall file header, beginning with the text "WARC/1.0", and the next time you see this text is to describe either a request (`"WARC/1.0\r\nWARC-Type: request"`), a response (`"WARC/1.0\r\nWARC-Type: response"`) or possibly a metadata or other type of WARC category (e.g. `"WARC/1.0\r\nWARC-Type: metadata"`). For this project we are only interested in WARC responses (`"WARC/1.0\r\nWARC-Type: response"`), as these are the only categories that contains the raw HTML data of the web page we are analysing.[2]

Looking into more detail at WARC responses, you can see that these are further broken down into three sections, which are separated by blank lines. The first is the WARC

2 Note the use of '\r' with '\n' to signify a line ending in the WARC (and HTTP) headers. This is a standard line ending code for text files saved with Microsoft Windows and some other scenarios. You will need to account for this when processing these headers.

response header (beginning with "`WARC/1.0`"). The second is the HTTP header (usually beginning with "`HTTP/1.1 200`") and the third is the raw HTML data (usually but not necessarily beginning with "`<!DOCTYPE HTML>`"). For the purposes of this project, you can assume that the first block of text (before the first blank line) is the WARC header, the second block of text (after the first blank line) is always the HTTP header, and the third block of text (i.e. anything after the second blank line and before the next "`WARC/1.0`" heading) is the raw HTML that we need to analyse.

Taking into account the above, your program will need to be able to open a WARC file, discard or ignore the overall WARC file header, and then for each result:

1. Extract the URL from the WARC response header (this is stored in the line starting with "`WARC-Target-URI`")
2. Extract the "`Content-Type`" from the HTTP header. For this project we are only interested in responses that are of "`Content-Type: text/html`". Any other types of HTTP responses can be ignored.
3. Extract the raw HTML for this result and store it in a data structure so that it is associated with the URL you extracted (in point 1).

Extracting Raw Text from HTML:

If you were to have a look at the raw HTML you have extracted in detail, you would see that it doesn't quite (yet) look like nice words and sentences that you will be able to analyse to determine its "positivity" and "patriotism" as you are required to do for insights 1 - 3. In order to get your text to this point, you are going to have to perform some transformations on it, namely:

Removal of any HTML tags – any text between a '`<`' character and a '`>`' character you can assume is a HTML tag and needs to be removed before completing your analysis for insights 1, 2 and 3.

Removal of JavaScript code – before you remove your HTML tags above, you will also need to remove any text that is between the '`<script>`' and '`</script>`' tags (again only for completing insights 1 - 3). "Note that a "<script>" (or "</script>") tag can have any amounts of whitespace or other text between the "<script" (or "</script") portion and the final ">" character and valid script tag that must be removed.

The Insights Themselves:

Some more details about what is required for each insight is below:

1. How 'positive' is Australia generally?

For this insight, both you and your supervisor are keen to understand how much Australian websites use 'positive' words compared to how much Australian websites use 'negative' words. It was decided that, for this insight, your program should produce a list with five items. The first and second items in this list are the total count of positive words and negative words respectively within the raw text for all Australian web pages

that were in the WARC file provided to your program. The third item of the list should be the ratio of positive words to negative, which can be calculated by dividing the former by the latter. The fourth and fifth items should be the average number of positive words and negative words respectively found in the typical Australian web page.

To assist you in this duty, your supervisor has provided you with a list of common positive English words, and a list of common negative English words. You can find these lists as text files on LMS and Moodle Server. For this project you can assume that any words that are not in either of these lists should not be included as part of the positive or negative counts.

Note in order to produce accurate results here, you will have to make sure that your program counts the appearance of any positive and negative words in your text regardless of the word's case (uppercase / lowercase or a combination of the both) and any punctuation at the start, end or within each word itself (e.g. commas, full stops, quotation marks, etc.) - "- in fact it is recommended you remove all punctuation from your text before performing this step (for now assume every non-alphanumeric character visible on a standard ANSI keyboard is a punctuation character that needs to be removed, if you have bought your computer in Australia or the US then it is very likely you will have an ANSI keyboard, if not then you can easily find out what punctuation an ANSI keyboard contains through a Google image search).

Note the above analysis should be performed on **Australian websites** only. For this project assume that a website will always an Australian website if, and only if its domain name ends in a `'.au'` (domain names are discussed in more detail in insight 4).

2. How 'positive' is Australia towards its Government?

As well as calculating how 'positive' Australia is in general, the second outcome of this project is to determine how 'positive' Australia is towards its Government. In order to determine this your program should examine every sentence that contains the word 'government' for any positive or negative words. You and your supervisor decided on the following rules for any sentences containing the word `'government'`:

- If the sentence has only one or more positive words then it should be counted as a 'positive' sentence.
- If the sentence contains one negative word then it should be counted as a 'negative' sentence **however** if the sentence contains two negative words then it should be counted as a 'positive' sentence (i.e. it is likely that the writer has used a double negative). If the sentence contains three or more negative words then it should be counted as a negative sentence.
- If the sentence contains a combination of positive and negative words (or no positive or negative words) then it should not be counted as either positive or negative.

As with insight 1 your results should be provided in a list with the first and second items being your raw positive and negative counts, and the third item being the ratio of positive to negative counts, and the fourth and fifth items being your average number of positive sentences and negative sentences per web page respectively.

Also as with insight 1, the above analysis should be performed on **Australian websites** only. In addition, the same directions with regards to the word's case and punctuation applies also, but you may wish to delay removing any sentence-ending punctuation characters (see below) to ensure you are able to split your raw text for the result into individual sentences first.

For this section you can assume that a sentence is any number of words ended by a sentence-ending character (a full stop, a question mark or an exclamation mark).

3. How 'patriotic' is Australia compared with the other major English speaking countries

For this insight you are required to determine how often the word "`australia`" appears in the raw text of your Australian websites compared with how often the other country's names appear in their web sites, specifically focussing on two other major English speaking countries – Canada and the United Kingdom (who both have their own unique TLDs):

- For Canada your program should determine how often the word "`canada`" appears in the raw text for any URLs whose domain name ends in "`.ca`".
- For the United Kingdom your program should determine how often the word "`uk`" and the phrases "`united kingdom`" and "`great britain`" appear in the raw text for any URLs whose domain name ends in "`.uk`".

All of the insights are to be calculated as percentages with the following formula:

(total number of occurrences of all words / phrases for the country) / (aggregate number of words of every web result's raw text for that country) * 100.

These percentages are then to be provided in a list in the order of [Australia, Canada, United Kingdom].

The same directions from insight 1 with regards to the word's case and punctuation removal applies to the words you will examine in this insight also.

4. Web domain links and counts

For every Australian web page in your WARC file, your program should count every domain name that it links to. A domain name is the part of a URL that refers to the root site, for example the domain name for the link "`https://www.google.com.au/example/testing.html`" is "`www.google.com.au`". For this project you can consider that a link in to web page only exists when it appears within a `<a ...>` tag that contains the "`href`" attribute within the page's raw html, for example,

"`<a href='https://www.google.com.au/example/testing.html'>`" or

'`<a href="https://www.google.com.au/example/testing.html">`' or

'`<a href=https://www.google.com.au/example/testing.html>`'.

Your program should examine each link that each of your Australian web pages refers to, extract only its domain name (that includes any subdomains, including '`www`'), and count the occurrences of these domain names across all of your Australian results. Any

links that do not start with a `'http://'` or a `'https://'` can be ignored. Your program should return the top 5 most occurring domain names and their counts in a list of tuples with the format: [(domain_name, aggregate_count), ...]. The tuples in the list should be in descending order by count. In case of tie in their counts, rank them in ascending alphabetical order.

Note URLs are case-insensitive, so all domain names (and indeed all text within `"<a ..."` tags) should always be converted to lower case before counting.

The Program Itself:

Your program should be written in Python and have the following `main()` signature (from where it will be called for testing and "demonstrations"):

```
def main (WARC_fname, positive_words_fname, negative_words_fname):
```

The input arguments to this function are:

- *`WARC_fname`* – the name of the WARC filename that your program will analyse
- *`positive_words_fname`* – the name of the filename containing the list of positive words[3] (one per line) that is to be inputted into your program.
- *`negative_words_fname`* – the name of the filename containing the list of negative words[4] (one per line) that is to be inputted into your program.

  (For windows, use Wordpad is recommended to view the files)

When called your program should return four lists representing the outputs of each of your insights (in the order of insight 1 to insight 4).

Your program should not modify any of the words within your lists of `positive_words` and `negative_words` (e.g. do not remove any punctuation from these words, even if this means that there will be no instances of this particular word counted). All sample files are provided on LMS and Moodle server in a zip file.

Of course, you are expected to structure your program using several 'helper functions' that are called from within your `main()` function (and / or helper functions within these helper functions) in order for your marker / colleagues to more easily understand your program.

For testing your program will be called by using the `main()` function. For example:

```
>>> gen_pos, gov_pos, pat, top_links = main("warc_sample_file.warc",
"positive_words.txt", "negative_words.txt")

>>> gen_pos

[1238, 586, 2.1126, 26.913, 12.7391]
```

3 This list is sourced from Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews." Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004), Aug 22-25, 2004, Seattle, Washington, USA, (online access: https://gist.github.com/mkulakowski2/4289437)

4 As above (online access: https://gist.github.com/mkulakowski2/4289441)

```
>>> gov_pos
[21, 13, 1.6154, 0.4565, 0.2826]
>>> pat
[0.3421, 0.3186, 0.0995]
>>> top_links
[('www.industryupdate.com.au',275),('religionsforpeaceaustralia.org.au',
183),('boundforsouthaustralia.history.sa.gov.au',148),('www.jcu.edu.au',
114),('blogs.geelongcollege.vic.edu.au',54)]
```

Additional Requirements:

Your program may be distributed to high-level bureaucrats, some of which may not have the same level of technical skills that you and your team have. To ensure your program is a success in these people's hands you will need to ensure that:

- All inputs to your `main()` function are validated to ensure they are valid and what your program expects. If they are not then your program should gracefully terminate.
- If a user was also to input any file name that cannot be found or opened then it should also gracefully terminate.
- As this is a proof-of-concept, the import of any Python modules is **strictly prohibited.** As with your previous project, the use of certain modules (e.g. warc or warcio) would be a perfectly sensible thing to do in a production setting (provided these modules have been vetted by your information security team for use in programs that may be exploring and/or collating sensitive Government data), these will again take away from the core activities of the project which are to become familiar with opening files, text processing and the use of inbuilt Python structures (which in turn are similar to basic structures from many other programming languages).
- Your program, of course, should be appropriately commented.

In addition to the above please note the following:

- WARC files will often contain characters that the default settings for your `open()` function call cannot handle. Therefore, you should open your WARC file in binary mode for example:

  ```
  warc_file_handler = open(WARC_fname, 'rb')
  ```

  And you should use the `read()` function to read in its content, followed immediately by a call to the `decode()` function to convert the file's contents to basic text, ignoring any decode errors for example:

  ```
  warc_text = warc_file_handler.read().decode('ascii', 'ignore')
  ```

- Do not assume that the input file names will end in `'.warc'`. File name suffixes are not mandatory in systems other than Microsoft Windows. Do not enforce that within your program that the file must end with a `'.warc'` or any other extension (or try

to add an extension onto the provided `WARC_filename` argument), doing so can easily lead to lost marks.

- Ensure your program does NOT call the `input()` function at any time. Calling the `input()` function will cause your program to hang, waiting for input that automated testing system will not provide (in fact, what will happen is that if the marking program detects the call(s), and will not test your code at all which may result in zero grade).
- For the purposes of our testing your program should also not call the `print()` function at any time. If it has encountered an error state and is exiting gracefully then your program needs to return empty lists. At no point should you print the program's outputs instead of (or in addition to) returning them or provide a printout of the program's progress in calculating such outputs.
- For any of your outputs that is a ratio or an average, if ever the denominator is zero then your program should return `None` for this ratio rather than zero or an error-state.
- All of your outputs should be rounded to 4 decimal places (if they are required to be rounded). Rounding should only occur at the final step immediately before the value is saved into its final data structure (i.e. do not round off any values during any of your intermediate steps when calculating your outputs).
- If you wish to view the contents of the positive and negative words files in Microsoft Windows please open these files in WordPad or another text editor besides NotePad, as NotePad does not process text files that end in just `'\n'` correctly.

Submission:

Submit your solution on Moodle before the deadline. You are required to paste your code in the text box as well as load the same code as a python file. The name of the file must be your student_id.py. Read the submission guidelines on Moodle portal.

You need to contact unit coordinator if you have special considerations or you plan to be making a submission after the mentioned due date.

Marking Rubric:

Your program will be marked out of 40 (later scaled to be out of 20% of your final mark for CITS1401). 30 out of 40 marks will be awarded automatically based on how well your program completes a number of tests, reflecting normal use of the program, and also how the program handles various states including, but not limited to, different numbers of results in any WARC file and / or any error states. You need to think creatively what your program may face. Your submission will be graded by data files other than what has been provided. Therefore you need to be creative to look into corner or worst cases. There are some hidden tests in Moodle as well as the project may undergo further automated testing after the deadline.

10 out of 40 marks will be awarded manually after the deadline. They will be based on style (5/10) "the code is clear to read" and efficiency (5/10) "your program is well constructed and runs efficiently". For style, think about use of comments, sensible variable names, your name and student ID at the top of the program, etc. (Please look at your lecture notes, where this is discussed.)

Style Rubric:

0	Gibberish, impossible to understand
1 – 2	Style is quite poor
3 – 4	Style is adequate to good, with small lapses
5	Style is very good or excellent, your submission is very easy to read and follow

Your program will be traversing files of various sizes (possibly including very large warc files) so try to minimise the number of times your program looks at the same data items. You may think to use different data structures such as tuples, lists, or dictionaries.

Efficiency Rubric:

0	Code too incomplete to judge efficiency, or wrong problem tackled
1 – 2	Very poor or inferior efficiency, many lapses
3 – 4	Acceptable or good efficiency, with some or few lapses
5	Excellent efficiency, should have no issues with large WARC files, etc.

Efficiency lapses can include (but are not limited to):

- The use of more loops than necessary
- Inappropriate use of readline()
- Opening files more than once (and not closing the files you open)
- Use of try / except when the error can be caught and handled using an if statement instead
- Blocks of code and / or helper functions run / called more times than is necessary

Automated Moodle testing is being used so that all submitted programs are being tested the same way. However, there is randomness in the testing data. Sometimes it happens that there is one mistake in your program that means that no tests are passed or your program gives error for a test case resulting in failure to proceed with other test cases, and you will get zero grade. Remember there is penalty for re-submissions. So it is better to check your program thoroughly on Thonny before attempting to submit it on Moodle.

Your program is running well on the provided example file? Great! Now try downloading and running one or several WARC files from the Common Crawl (see the instructions under the heading "The Common Crawl and WARC Format" and on the Common Crawl website). The sample solution will open and process one of these files in approximately 45-90 seconds, so I would take this as a good guide to compare your program to when judging the efficiency of your solution. Note that opening and processing one of these large files can take 4 – 8 GB of memory (in addition to what your system is already using) to do so, if your computer is using much much more memory than this to do so then this is also a guide / hint that you could improve the efficiency of your solution.

Lastly, want some more smaller WARC files for you to test? Why not write a Python script to generate your own subset of WARC responses from a Common Crawl WARC File?