LING 450: ASSIGNMENT 1
Group Members- Lovely, Rachel, Antanila
Approach 1 & 2: Antanila
Date: October 28th, 2022

Approach # 1: Regular Expressions (one page)

At first, we import spacy, a natural language process (nlp) object which is a function to analyze any given text. Then we import **re** 'regular expression' which is a function that matches a particular string with the regular expression, usually it comes back as the same. The second code with **open ("")** as **f** is telling the system to return a file which we are going to read and analyze. When returning file, we input encoding in order to encode or decode the file since some files have to be read as a specific encoding type in this case, we inputted **encoding ='uft-8'**. The code **'r'** is for reading the file that exists.

def is used to define a function in this case find_sents is returning sentences from the text that is being retrieved by the file above. Then we create a doc object by processing a string of text with the nlp object and in this case, we have chosen the file "data/5c1dbe1d1e67d78e2797d611.txt". We attempt to call the **read()** method for the variable file outside the **with** statement. The code will return with a list of sentences from the document. Then we use spacy.load to load **en_core_web_sm** is a trained pipeline which enables spaCy to predict linguistics attributes in context such as syntactic dependencies, named entities and parts of speech tags. However, this code is not going to process the text, it will only pull sentences from the text. The **list (doc.sents)** is a property used in the document **"data/5c1dbe1d1e67d78e2797d611.txt"** we are analyzing to iterate over the sentences.

findall() function is extremely useful as it returns a list of strings containing all matches. The code set the pattern to '"**(.+?)**"', where the single quotes represent the text body, while the double quotes represent the quotes within the text. We have the parenthesis which creates a capture group and **.*?** is a non-greedy moderator that extracts the quotes, not the text between the quotes. We observed that direct quotes in the file which ends with **.".** punctuation seems to be returned in the output.

The output is somewhat what we imagined. The sentences are extracted from the file we have pulled. It has specifically returned direct quotes from the text. There were some few encoding errors which we noticed after taking a deeper look at the file. After we reviewed the the data in the actual notepad that is stored in my desktop, we noticed that the output did not capture all the quotes. For example, some direct codes which are questions, and the code did not return those interrogative direct quotes. Moreover, there is no further information on who said the quotes. So, we believe this may be an encoding error which is not reading the file thoroughly.

After processing the steps five times using five different texts files, we observed significant changes. We noticed that the code returned shorter quotes, even one word like 'I'm Back' (Text 3). TEXT 6 displayed 'elected' and 'whole country' in the output, which is not a direct quote, more like a short phrase. We think this happened due to a punctuation error. Again, we came across some encoding issues where many of the quotes did not seem to be returned with the output. In TEXT 3 and TEXT 5, the code returned names like '**Gasol**' and '**Audrey**' which is a bit surprising since no names were being extracted from the other text files. We believe, this output occurred due to punctuations errors or a file readability error.

Approach # 2: Using spaCy's Matcher (one page)

Spacy's matcher produces a rule based matching engine and it operates based on tokens which can be accessed within the text. All matchers are processed using patterns which are already defined by the matcher. In this approach we first import the **Matcher** from **spacy.matcher**. Then we initialize the matcher **Matcher(nlp.vocab)** with the shared vocabulary using the document the code will operate on. This approach will process the texts, then find quotes and speakers from the data. Similarly like Approach 1 we had to load the text files using the proper encoding file and we input **text = f.read ()** to order the code to read the text. Then we convert the text into a **spacy doc** in order to apply patterns or attributions. For instance, now we call **matcher.add()** with a **pattern** and an **ID**. In this case we are inputting **pattern_n**, **part-of-speech tag (POS)** more specifically **Proper_Nouns** so that we can find out who the speakers are and every other quote. We want to look at the **proper nouns** in the text.

The matcher returns a list of **(match, doc[match[1]:match[2]])** for instance **(3232560085755078 826, 44, 45) Clark**. Additionally, the **match id (POS, PROPN)** uses the **hash value** of a word string to get the string value of all the proper nouns from the documents. We read in the chapter that spaCy uses a hash function to calculate the hash based on a word string and hashes cannot be reversed. The second output **44, 45** is processing the string of the match id and maps to the span of the original document.

The output is some what we imagined. We were still a bit surprised because we thought that the code will only return names of persons, but we observed that it returned different entities like countries, cities, institutions etc. It did just capture all sorts of nouns. Furthermore, we observed that the match does not seem to return anything else except the assigned labels.

In the next step '**finding quotes**,' we input a simple pattern to extract different attributions in single quotes. **ORTH**, **OP** are token attributes and the available token pattern key in this case **pattern_q** corresponds to a number of token attributes. In the code, **Orth** stands for the exact word for word of a token and **OP** is operator or quantifier to dictate how often to match a token pattern. This process looks at all the words in the data and then we input **matches_q.sort key+ lamba** to find the one line quotes from the text. We noticed that the quotes that returned are different from approach 1, so we are guessing that the code is analysing and processing every other second quote in the text. We have implemented the spaCy matcher step five times with different texts. The observations displayed that some of the single patterns could not extract single quotes. The output returned short phrases, or single words rather than a full quotation. Compared to approach 1 all the matcher output was different which means that the code did not process similar quotes from the loops but looked at every other second word syntax.

.