Extracting quotes and speakers from text

Designing an algorithm that extracts quotes in and out of quotation marks is a tricky task that would require a combination of rules, tokenization, regular expressions and named entity extraction. We would assume the text is in true-case and we would not encounter typical issues with named entity recognition which does not work properly with all upper or lowercase.

Whilst one could take a machine learning approach to this task and build a quotes classifier, and train based on a subset of quotes that would assign a probability that taken together composes a valid quote string. A pattern matching approach might be simpler to implement and less costly as training a ML classifier involves using a large corpus of texts whereas identifying a dictionary of quote indication phrases and break phrases might yield the same results. This would of course need to be tested to see how it works on test texts, if accuracy is high enough, in that case a ML approach might be required, however even with this approach the fuzziness of quotes without quotation marks isn't always easy to identify.

The core of the algorithm will hinge off identifying phrases such as "said that", "admitted that", "claimed".

Data Preparation:
Data would need to be ensured to be true-cased before running the algorithm and also that the paragraphs and sentences are appropriately punctuated as there will be rule-checks for end-of sentences. Also ensure the text is correctly encoded.

Quote and Speaker Algorithm Extraction:

- Tokenize text (paragraph or sentence) using nltk/spaCy/CoreNLP etc.
- Identify Named Entities using a named entity extractor such as nltk or spaCy.
- Rule check whether a quote indication phrase such as "said", "admitted" "claimed" is present in the paragraph from a dictionary of such phrases to indicate that there is a quote to extract from the the text. These will be pattern matched using **re** or a function to check against tokens.
- Use rules and index positioning of the named entity tokens and quote indication phrases, i.e.
    - Named Entity tokens indexes cannot be more than 2-3 words before or after a phrase like "said" and the named entity type would have to be a person. i.e. sentence[0:3] could be named entity and phrase could be sentence [4:5]
    - If the quote indication phrase occurs at the end of the sentence (check to see if a punctuation phrase occurs after quotation phrase), regular expression pattern matching for all text between quotation marks (with nested quote regex patterns too) before the quote indication phrase will occur. If the quote phrase does not occur at the end of the sentence then the following text in the pattern matched quotation marks will be assigned to the named entity.
        - Note: This will allow multiple sentence long quotes to be assigned to a named entity.
    - If a break word such as "adding" (another dictionary of phrases will be pattern matched) occurs after the quotation marks and another quote starts, the phrase will be checked against a dictionary of phrases and then the continuing text in the quotation marks will be assigned to the named entity previously identified.
- For quotes where there are no quotation marks, a similar process as above will be described but this is much harder to identify correctly and the algorithm will simply identify the text that comes before or after the quote indication phrase depending on where the quote indication phrase is relative to the sentence. It will only assign the text of that sentence from that phrase. This is a big drawback of the algo.

Test Data:
Text inputs of various different formats .csv, .txt, with and without quotation marks.

Major Drawback:
If a chunk of text doesn't perfectly fit the pattern, it's not going to be matched.