Assignment 1

Segregation of Data in MetaData and MainText [get_sentences(FilePointer)]

[This step is repeated in almost every task so exclusively mentioning it]

I have split the dataset into two parts: MetaData and MainText
So, MetaData parts contain all the headers like from: to lines: message-id: etc
While the MaintText is rest of the text excluding MetaData.

- Read the file using readlines() command, now all the lines separated by "\n" become a distinct element of a list.
- Traverse this list and find the first element(represents a line) which contains only "\n" and nothing else.
- This is the point of the split between the metadata and the main text.[I analysed the text files and found this pattern]
- Now everything in the List before this element is MetaData and everything after this is my MainText.
- This is implemented in the function **get_sentences(FilePointer)** which takes input as a pointer of a file.
- This function returns three things:
    - **MetadataList**:- List containing all the lines of metadata
    - **MainText**:- A string that contains all the text except MetaData
    - **SentencesList**:- List containing the sentences of MainText
        - This is done by passing the whole MainText string into sent_tokenize( ) of nltk.tokenize
- I have used the Metadata only in task 3 and task 8
- Other than this every task has **different preprocessing steps**, which are mentioned in the **methodology section of each task.**

Task 1

[Implemented as ques1( )]

Assumptions and Notes:

- ➔ Words are either a combination of alphabets or a combination of digits.
  Eg:- 910, hello, raw are words but h123abc is not a word
- ➔ MetaData contains new sentences on every new line.
- ➔ The number of words is only counted for MainText(No MetaData).
- ➔ Contraction of two words into a single word will be considered as two words.
  Eg: -**He'd** is a contraction of two words **He** and **would**, so counted as two words. More examples of the contraction of words are present.

Methodology and Preprocessing Steps:

- ➔ Segregate the data into Metadata and MainText using **get_sentences(FilePointer)**
- ➔ For Sentences :
    - ◆ Printing the number of sentences for MetaData and MainText separately.
    - ◆ For MetaData it will be the number of lines for the MetaDataList returned from **get_sentences(FilePointer)**.
    - ◆ For sentences in MainText, we have SentencesList returned from **get_sentences(FilePointer)** which has already tokenized the sentence using sent_tokenize( ). We can simply print its length.
- ➔ For Words :
    - ◆ Once we have the MainText [without metadata], tokenize the string into words using word_tokenize()
    - ◆ After tokenization, remove all the punctuation marks from all the tokens.
    - ◆ Traverse the Tokens list after removing punctuation
        - ● Check current if it has a non-zero length and is a combination of either alphabets only or digits only.
        - ● I have used regex for comparing if a token is made up of alphabets only ("^[A-Za-z]+$") or if it is made up of digits only("^[0-9]+$")
        - ● If it matches either of the regex, increase the count of the words
    - ◆ Finally print the number of words.

Task 2

[Implemented as ques2( )]

Assumptions and Notes:

- ➔ The number of words starting with vowels or consonants are only counted for MainText(No MetaData).

Methodology and Preprocessing Steps:

- ➔ Segregate the data into Metadata and MainText using **get_sentences(FilePointer)**.
- ➔ Tokenize the MainText using word_tokenize( ).
- ➔ Remove the punctuation marks from all the tokens.
- ➔ Traverse the token list:
    - ◆ For tokens/words starting with Vowels :
        - ● Use a regex ("^[aeiou][a-z]*$") to compare every token (converted in lower case), if it matches then append it in the list of vowel words.
    - ◆ A similar procedure for consonants as stated above.(regex is "^[b-dfghj-np-tv-z][a-z]*$")
- ➔ Finally, print the length of both lists containing vowels and consonants.

Task 3

[Implemented as ques2( )]

Assumptions and Notes:

- An email -id is of the form “XXXX@YYYYY” where XXXX and the YYYY part must contain at least one alphabet or digit.
- As mentioned in the clarifications, I have also included the articles and the message-id as emails.

Methodology and Preprocessing Steps:

- Extract the whole text from the input file.
- Tokenize it using word_tokenize( ) and store the tokens in a list.
- Traverse the tokens list and find “@” symbol.
- If a particular token is “@” and both the token before it and after it contains atleast one-digit/alphabet, then Token[i-1]+ Token[i] + Token[i+1] will be a valid email-address. Where Token[i]=”@”
- Append the valid email address to a list and a set.
- Print all the email address present in the list and print all the distinct email address present in the doc.

Task 4

[Implemented as ques4( )]

Assumptions and Notes:

- Using only the MainText (No MetaData) for this task.
- Input word does not contain any punctuation marks and is only a combination of alphabets or a combination of digits.
- No spaces allowed in a word.
- Printing the original sentence (before preprocessing) if there is a match.
- If “100 is my favourite number” is in text file, and input word is “Hundred” then it will match this case and the answer list will contain this sentence.

Methodology and Preprocessing Steps:

- Take the input from user
  InputFile and InputWord
- Convert the InputWord into lower case.
- Check if the InputWord represents a number if it does convert it into InputNumber.
- Retrieve SentencesList of MainText using get_sentences(FilePointer).
- For every Sentence in SentenceList
  - Replace characters “\n” and “\t” with single space.
  - Convert it into lower case.
  - Now, Tokenize the using word_tokenize( ) and store it in a List.
  - Remove Punctuations from every token in the tokens list.

    - Now traverse the token list till you find a token with non-zero length. This is the first word in the sentence.
    - Compare the first word of the sentence with InputWord and InputNumber (If Inputword represents a number), if it matches then increase the count and store this sentence in an AnswerList.
- After the traversal, print the count and display the sentences in the AnswerList.

Task 5

[Implemented as ques5( )]

Assumptions and Notes:

- Using only the MainText(No MetaData) for this task.
- Input word does not contain any punctuation marks and is only a combination of alphabets or a combination of digits.
- No spaces allowed in a word.
- Printing the original sentence (before preprocessing) if there is a match.
- If “my favourite number is 12” is in text file, and input word is “twelve” then it will match this case and the answer list will contain this sentence.

Methodology and Preprocessing Steps:

- Take the input from user
  InputFile and InputWord
- Convert the InputWord into lower case.
- Check if the InputWord represents a number if it does convert it into InputNumber.
- Retrieve SentencesList of MainText using **get_sentences(FilePointer)**.
- For every Sentence in SentenceList
    - Replace characters “\n” and “\t” with single space.
    - Convert it into lower case.
    - Now, Tokenize the using word_tokenize( ) and store it in a List.
    - Remove Punctuations from every token in the tokens list.
    - Now traverse the token list from the end, till you find a token with non-zero length.
    - This is the last word in the sentence.
    - Compare the last word of the sentence with InputWord and InputNumber (If Inputword represents a number), if it matches then increase the count and store this sentence in an AnswerList.
- After the traversal, print the count and display the sentences in the AnswerList.

Task 6

[Implemented as ques6( )]

Assumptions and Notes:

- Using only the MainText(No MetaData) for this task.
- Input word does not contain any punctuation marks and is only a combination of alphabets or a combination of digits.
- No spaces allowed in a word.
- Printing the original sentence (before preprocessing) if there is a match.
- I have currently implemented two methods:
    - Not using Stemming/Lemmatization on the preprocessed text and input word.
    - Using Porter stemmer for stemming the input file and the input word, and then matching the stemmed words.[Currently Commented out, but those comments can be removed to see the output.]
- If “100 is my favourite number” is in text file, and input word is “Hundred” then it will match this case and the answer list will contain this sentence.

Methodology and Preprocessing Steps:

- Take the input from user
  InputFile and InputWord
- Convert the InputWord into lower case.
- Check if the InputWord represents a number if it does convert it into InputNumber.
- Retrieve SentencesList of MainText using **get_sentences(FilePointer)**.
- For every Sentence in SentenceList [Without stemming method]
    - Replace characters “\n” and “\t” with single space.
    - Convert it into lower case.
    - Now, Tokenize the using word_tokenize( ) and store it in a List.
    - Remove Punctuations from every token in the tokens list.
    - Now traverse the token list
        - Compare the current token with InputWord and InputNumber (If Inputword represents a number), if it matches then increase the count of words and flag this sentence to be stored.
    - If the sentence is flagged, store it in AnswerList and increase the count of Number of sentences.

- For every Sentence in SentenceList [Using porter stemmer]
    - Replace characters “\n” and “\t” with single space.
    - Convert it into lower case.
    - Now, Tokenize the using word_tokenize( ) and store it in a List.
    - Remove Punctuations from every token in the tokens list.
    - Now traverse the token list
        - Convert the current token into the base form using porter stemmer and the input world is also converted to base form using porter stemmer.

        - Compare the stemmed current token with stemmed InputWord and InputNumber (If Inputword represents a number), if it matches then increase the count of words and flag this sentence to be stored.
    - If the current sentence is flagged, store it in AnswerList and increase the count of Number of sentences.

- After the traversal, print the count of words, sentences and display the sentences in the AnswerList.

Task 7

[Implemented as ques7( )]

Assumptions and Notes:

- Using only the MainText(No MetaData) for this task.
- Any sentence ending with a "?" is a question.
- If some sentences end with a "?" but there are some punctuation marks after that like inverted commas or comma or ">" etc. then remove them and the sentence will be a question. Eg. **"How are you?"** is a question, even though it is ending with inverted commas.
- Printing the original sentence (before preprocessing) if there is a match.

Methodology and Preprocessing Steps:

- Take the input from user
  InputFile
- Retrieve SentencesList of MainText using **get_sentences(FilePointer)**.
- For every Sentence in SentenceList
    - Replace characters "\n" and "\t" with single space.
    - Convert it into lower case.
    - Remove Punctuations except [**? ! .**] from every token in the tokens list, as these mark the end of a sentence in English.
    - Now traverse the token list from the end, till you find a token with non-zero length.
    - This is the last non-zero length token of the sentence.
    - Compare it with "?" symbol, if it matches then increase the count and store this sentence in an AnswerList.
- After the traversal, print the count and display the sentences in the AnswerList.

Task 8

[Implemented as ques8( )]

Assumptions and Notes:

- Using all the text from the input file.
- Retrieving any time from the text, not just the one associated with a Date.
- Time has the format “HH:MM:SS”

Methodology and Preprocessing Steps:

- Take the input from user
  InputFile
- Retrieve MetaDataList and MainText using **get_sentences(FilePointer)**.
- Traverse the MetaDataList.[It contains Lines from metadata]
  - Tokenize each Line using word_tokenize( )
  - Now traverse the token list
    - Compare each token with the regex “^[0-2][0-9]:[0-6][0-9]:[0-6][0-9]$”.
    - If it matches check if it is a valid time in HH:MM:SS format then print it.
- Tokenize the MainText using word_tokenize( )
- Traverse this token list of the main text and apply a similar procedure as mentioned for tokens in metadalist.

Task 9

[Implemented as ques9( )]

Assumptions and Notes:

- Any word in all capital letters is an abbreviation.
- Abbreviations can also start from capital letters, end with a period and contain multiple uppercase, lowercase letters and periods in between.
- Using only MainText(No MetaData) for this.

Methodology and Preprocessing Steps:

- Take the input from user
  InputFile
- Retrieve MetaDataList and MainText using **get_sentences(FilePointer)**.
- Tokenize the MainText using word_tokenize( )
- Traverse this token list of main text :
  - Check if a token contains all capital letters through a regex.("^[A-Z][A-Z]*[A-Z]$")
  - If yes then store it in Answer List.
  - Check if a token contains this abbreviation through a regex.("^[A-Z][a-zA-Z.]*\.$")
  - If yes then store it in Answer List
- Display all the Abbreviations in AnswerList.