Project 2 Hamlet Data Mining

By:
Isha Chidrawar
Yash Patel

Regex

Our regular expression that we used was:

```
re.findall("[A-Z][(?<=I) ]?[a-z][A-Za-z,' \n]*[.|?|!]", hamlet)
```

To split this apart we will start on why we picked re.findall. The alternative was re.split, which divides the string based on the regex. The issue we ran across while utilizing this was that we needed to preserve the punctuation in the sentences but split removed it. The other issue we encountered is how strange it responded with new line characters where there were a number of empty strings in our final output. We discovered that findall, which only matches based on the regular expression, would be simpler to use.

```
[A-Z][(?<=I) ]?[a-z][A-Za-z,' \n]*[.|?|!]
```

Above is our regular expression broken up into colors for easier readability.

```
[A-Z]
```

This part says that we should start our sentences by first looking for a capital letter. We know that all sentences start with capital letters so this was the set that we wanted to look for first.

```
[a-z]
```

We will skip the next part and get to this part. Essentially this checks if the next letter is a lower case letter. All 2 or more letter words will have the second letter be a lower-case letter so we check for that. This removes all the names that are in all caps which removes most of the lines.

```
[(?<=I) ]?
```

There is one situation in which a sentence doesn't start with a capital letter followed by a lower case letter. That would be when the sentence starts with "I ". "I" is a common word that is at the beginning of sentences and it is unique since it is a one-letter word. It is also followed by a space that breaks our rule above. So we made this conditional that checks that if there is a space preceded by capital I then we will still include it in our regex. This can break if one of the characters' names ends in the letter I which we checked for and none of them do.

```
[A-Za-z,' \n]*
```

This is the most important aspect of our regular expression. For words, we look for all upper and lower case letters. We also look for the character ', which is used in contractions. We include blank spaces between words. For larger sentences, there is also a comma. If a sentence is longer than a line, "\n" is tested. Outside of the set, the * is used to account for any number of repetitions.

```
[.|?|!]
```

This is the end of our regular expression. This just says that our sentence should end in a period or question mark or exclamation point. We decided that these are the only ways a sentence will end in the text.

To make the sentences easier to read, we replaced the '\n' in the sentences with a space. The '\n' was included if a sentence was on more than one line. This made the sentences look more coherent.

Bigrams

When it came time to compute the bigrams, we had to alter the phrases in order to discover the bigrams more easily. First, we substituted a space for any special characters. This implies that only capital and lowercase characters remain. Then, to avoid duplication while locating our bigrams, we converted each line to a lowercase value. Then we calculated all of the bigrams for each line and concatenated them. We then filtered that table to only contain the (Noun, Verb) and (Pronoun, Verb) combination. We printed the top ten most often used bigrams.

	bigram	freq
3	(it, is)	12
8	(i, am)	9
11	(i, do)	7
14	(what, is)	7
15	(i, have)	6
19	(you, are)	6
31	(i, think)	5
43	(you, have)	4
58	(they, exit)	4
86	(it, was)	3

Here, the most common bigram is (it, is) which makes sense because "it is" would be a common pairing in dialogue. Most of the frequent pairings include "i" which also makes sense in dialogues.