Project 3: Indexing Final Report

1.
First part of the code contains the api methods
Second part contains the driver code for each query (term and phrase)

2.
In this project I have created an API with the following functions:

```
def create_inverted_index(data): function to create the inverted index
def number_of_documents_containing_word(inverted_index, word): function returning the number of documents that contains a particular word
def plays_from_scenes(scenes): function that converts and array of scene id's to an array of play id's
def consecutive_words_scenes(inverted_index, word1, word2): function that checks if two words are consecutive, that is, in the order "word1 word2"
def phrase_detect(inverted_index, sen, fname): function that returns the documents containing a particular phrase. This does its task by checking if each pair of consecutive words are present and then checking if those pairs are next to each other.
def stats(data): function that returns statistics of the data given. Like average scene length, smallest scene, largest scene, smallest play, largest play.

Below these I've implemented code to fit the information need of each term and phrase based query using the above functions.
```

3.
Libraries Used:

- JSON Library for handling the JSON file given to us
- Matplotlib to plot the graphs

4.

Let's say we have a query called "polar bears". If we just use count as a feature for our algorithm, it will check how frequently polar occurs, and how frequent bear occurs. So in the situation that a website has a page talking about "polar coordinates" and another page talking about "bears" (as in support or carry), this page will be returned as it will have many references of the both words "polar" and "bear". But this is not correct.

A fix for this is positional features as we have implemented in this project. We must check if polar and bear occur in the document and also check where they occur. This is to see if "polar" and "bears" occur next to each other. This will help us get the right pages for the query "polar bears".

5.
My runtimes are as follows:
term0: 15.971
term1: 2.0981 e-05
term2: 1.431 e-05
term3: 6.29 e-05

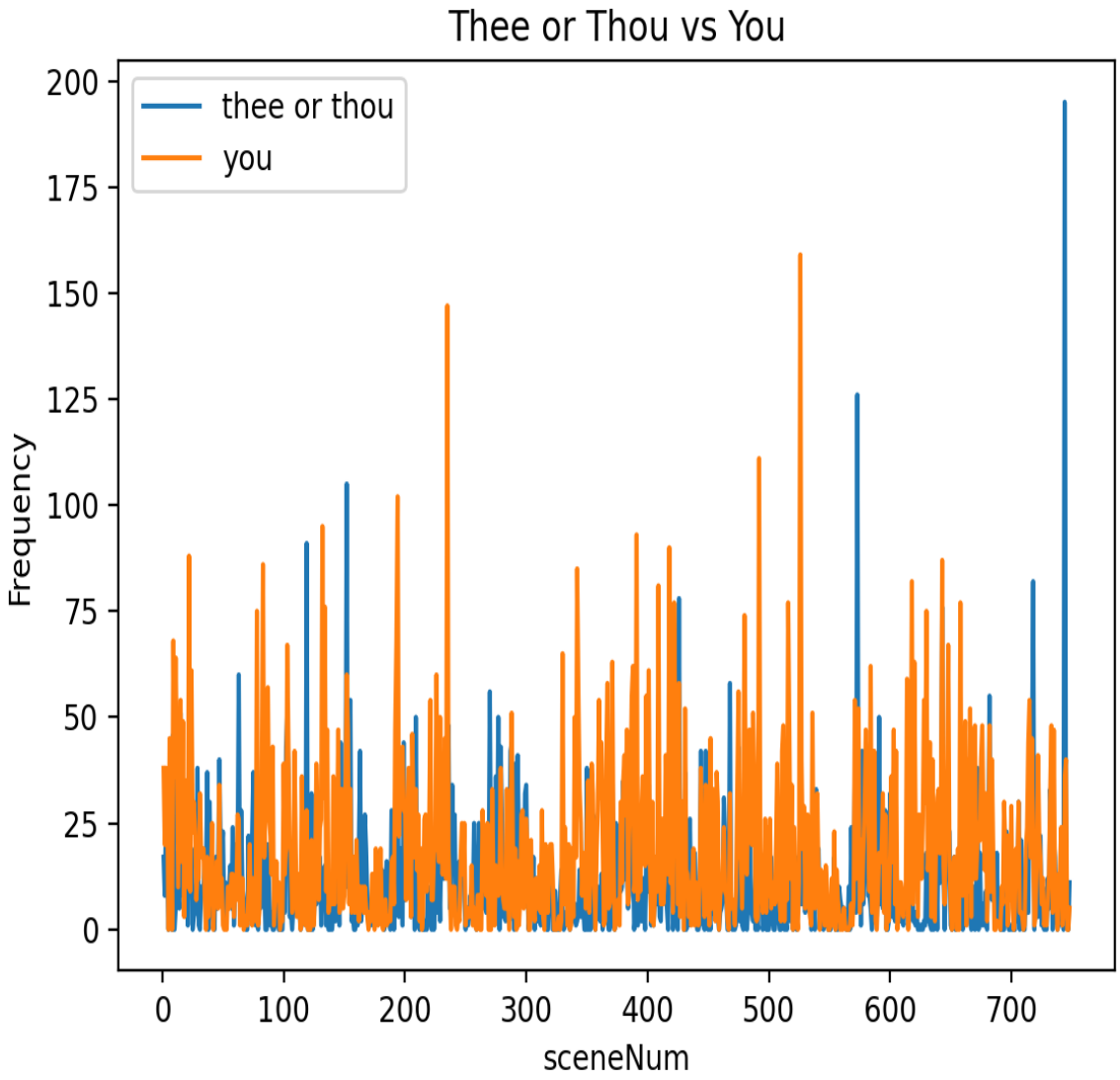
phrase0: 0.0026
phrase1: 0.0023
phrase2: 0.00059

In term based queries my term 0 is taking very long to run as I made my implementation quite complicated. I created an inverted index for each document and then ran a loop to obtain the counts of each word that we are looking for. Since this is a counting comparison task, I implemented it this way and it turned out to be very inefficient. It however returns the right outputs and provides us with a clean graph.

Among the phrase based queries, 'wherefore art thou romeo' takes the most time as I'm checking for the documents which have "wherefore art" then "art thou" then "thou romeo" after which I am checking if art thou romeo. Then I am filtering this to the documents where "wherefore art thou romeo" come together using the positional attributes that I recorded. This method seems to be inefficient compared to matching posting windows or intersecting posting lists method.

6.
Average Scene Length: **1202.8663101604277**
Smallest Scene: **antony_and_cleopatra:2.8**
Largest Scene: **loves_labors_lost:4.1**
Smallest Play: **comedy_of_errors**
Largest Play: **hamlet**

7. (Graph in next page)


Thee or Thou vs You
thee or thou
you
Frequency
sceneNum
0
25
50
75
100
125
150
175
200
0
100
200
300
400
500
600
700