Project Overview:

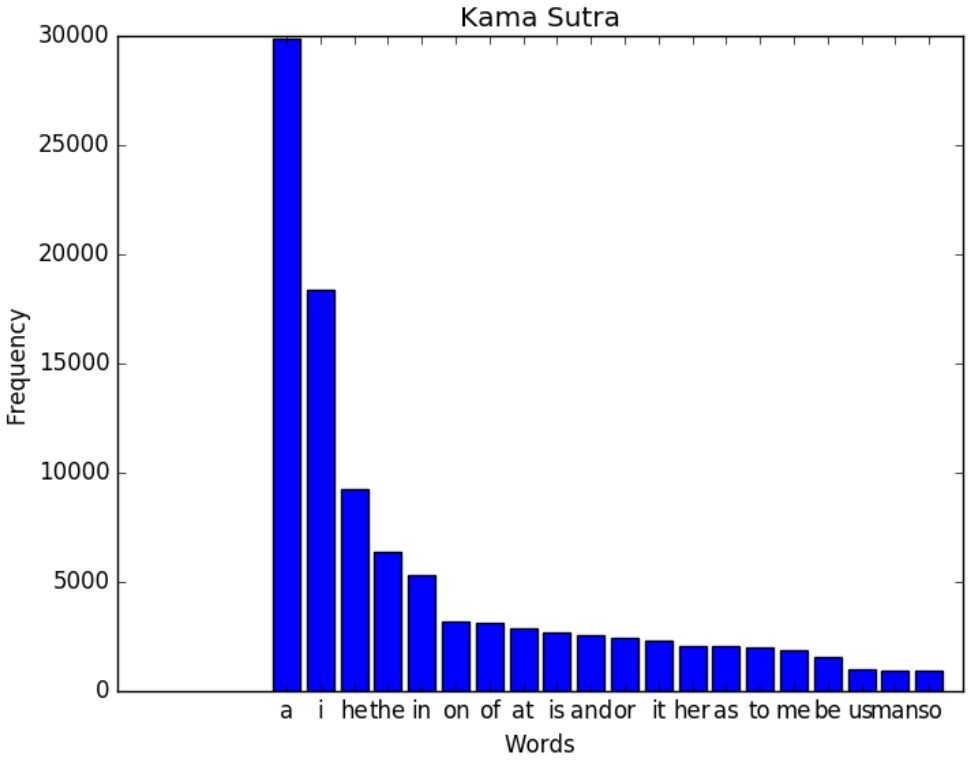
In this project I chose to explore Zipf's law. This states that for nearly any text, the frequency of any word is inversely proportional to that word's rank in the frequency table. This is shown really well when enormous samples of words are taken, such as the entirety of Wikipedia. However, there are still interesting results to be found when we look at word frequencies in novels. I have selected four texts (Mary Shelly's *Frankenstien*, Jane Austin's *Pride and Prejudice*, Charles Dicken's *A Tale of Two Cities*, and the *Kama Sutra*) to analyze for word frequency.

Implementation:

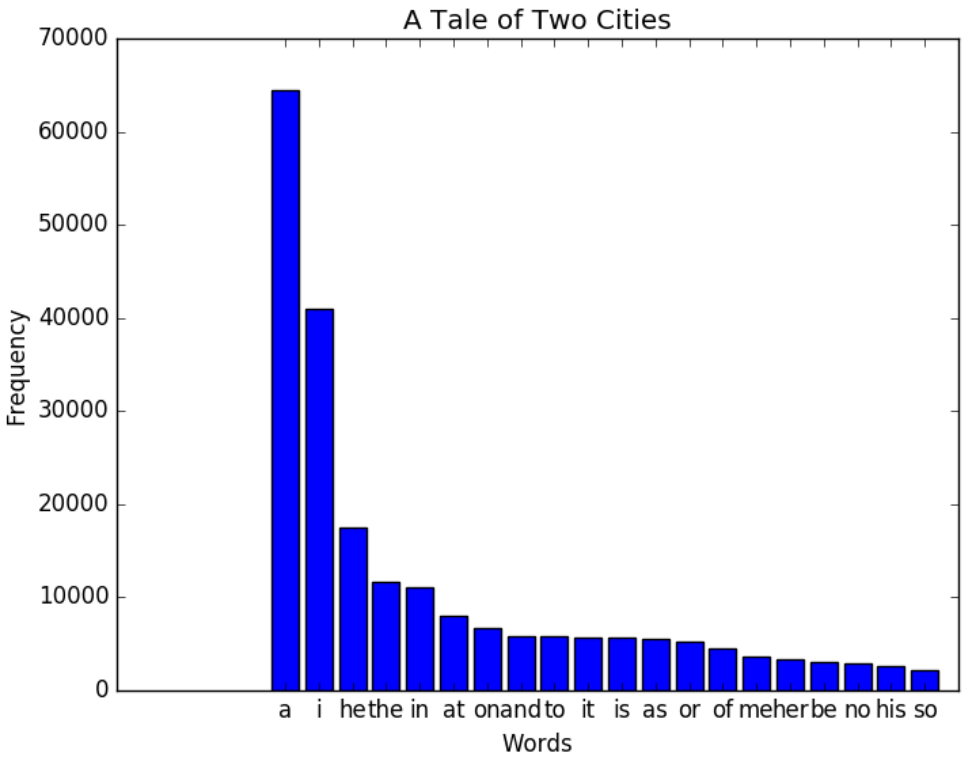
My code has two main components, one to import texts from Project Gutenberg, and another that actually runs frequency analysis. The first component is very simple (two lines) but implements two different modules to import text files from Project Gutenberg and save them as .txt files on my computer. It takes two inputs: the link for the text and the filename that you want to save it as. I implement the requests module so that I can ask Project Gutenberg for the information by giving it a link, and then I use the pickle module to dump the text into its own text module.

The second part is more complicated, but isn't anything too crazy. The only input is the text file that you want analyzed. First I import the text file and I import a list of the 1000 most frequently used English words. This was an important design decision because I could have had the code search for different string separated by a space in the text and then found the frequency of all of those different words, but I decided it was much more efficient and was overall not a terrible loss to accuracy to just import a list of the most commonly used words in the English language and assume that would probably contain the most common words in any given book. Next I use a dictionary to store data, and I run through each word in the list as the key, and use the count method to return a value of how many times that word is used. Now I have a great data structure for the data I want, except it isn't sorted. I found a sort method for dictionaries that will return a list of tuples with the data sorted by value. This means I have a list of tuples that is sorted with the most frequently used word as the first item in the list, and they are given in descending order. Finally I use matplotlib and pyplot to make a bar graph of the frequencies.

Results:



Here is the output of my function for the Kama Sutra. Although not neccesarily considered a literary classic, it had the most normal distribution of words, and demonstrated Zipf's law the best. We can see the most frequently used word is "a", then "I", then "he", then "the". The word "I" is used about half as often as "a", and the word "he" is used about a third as much, "the" a fourth, and so on. This is Zipf's law in action, that something as seemingly deterministic as language can be as predictive as the function 1/x.



Here is the frequency of *A Tale of Two Cities*. This graph also has a very nice distribution, but we can see that it actually follows the same trend as the *Kama Sutra*, and even has the exact same top five words. We would think that two completely different texts would have very different distributions, but in reality they follow distinct patterns.

Reflection:

This project went well in the fact that I got the results I was looking for and it runs pretty efficiently. I could do better to do actual synthesis with my information. I would like to take these graphs and this data and actually do predictive work with it as opposed to just observing what is happening.