4 Scripts as below:

NOTE:

- **Libraries can be used: math, mathplotlib, os, pandas, numpy.**
- Dataset is available on attachment.

1. Digital Shakespeare: Pre-processing

This language model will be trained on Shakespeare's books. First, read in all the given books and remove all special sentences from it. The special sentences lies between < and >.

Inputs	**Files:** All the files in the dataset folders
Output	**Filename:** *cleaned.txt* Example in sample behaviour.
Details	Task: • Remove all special sentences (Special **sentences lies between** “<” and “>”). • Remove all the characters that are **not alphanumeric** excepts space.
Sample behaviour	

```
< Shakespeare -- A MIDSUMMER-NIGHT'S DREAM >
< from Online Library of Liberty (http://oll.libertyfund.org) >
< Unicode .txt version by Mike Scott (http://www.lexically.net) >
< from "The Complete Works of William Shakespeare" >
< ed. with a glossary by W.J. Craig M.A. >
< (London: Oxford University Press, 1916) >
<STAGE DIR>
<Scene.—Athens, and a Wood near it.>
</STAGE DIR>

<ACT 1>

<SCENE 1>
<Athens. The Palace of Theseus.>
<STAGE DIR>
<Enter Theseus, Hippolyta, Philostrate, and Attendants.>
</STAGE DIR>
<THESEUS>       <1%>
        Now, fair Hippolyta, our nuptial hour
        Draws on apace: four happy days bring in
        Another moon; but O! methinks how slow
        This old moon wanes; she lingers my desires,
        Like to a step dame, or a dowager
        Long withering out a young man's revenue.
</THESEUS>

<HIPPOLYTA>     <1%>
        Four days will quickly steep themselves in night;
        Four nights will quickly dream away the time;
        And then the moon, like to a silver bow
        New-bent in heaven, shall behold the night
        Of our solemnities.
</HIPPOLYTA>
```

2. Zipf's Law: Common Statistics

Zipf's law of word distribution states that the frequency of every word in a large corpus is inversely proportional to its rank in the frequency table. Let f_1 be the I^{th} largest frequency in the list that is f_1 is the frequency of most common word, f_2 is the frequency of second most common word and so on. Zipf's law states that f_1 is approximately equal to $^{a}/_{I}$ for some constant .

Inputs	**File:** *cleaned.txt*
Output	There are 3 outputs: 1. **File:** vocab.txt (Using the file that was created in the previous task.) 2. Graph of frequency against first 100 words in sorted vocab. 3. Graph of words that occur n time against word occurrence.
Details	There are 3 Task:
	1. Finding all the unique word in the cleaned file. Save all the clean words along with their frequencies, and sort (in descending order), based on word's frequency in file *vocab.txt* 1. *To get unique words convert all the text into lower case first and the find unique words.* 2. Plot a graph of frequencies against first 100 words from the sorted vocab file. 3. Count the number of words that occur once, twice , thrice till 250. Plot number of words that occur n times against word occurrence.

3. Bi-gram Model

A bi-gram language model is a probabilistic model where a sentence probability is decomposed into product of conditionals as follow:

$$p(x_1, x_2, x_3, \dots x_n) = \prod p(x_k \vee x_{k-1}) \text{---------------------------(equation 1)}$$

These probabilities are approximated from the corpus using the following equation.

$$p(x_k \vee x_{k-1}) = \frac{c(x_k, x_{k-1})}{c(x_{k-1})} \text{-------------------------------------(equation 2)}$$

- where **c** stands for **count** of the **words occurring together in the corpus.**

For the **first 1000 words** in the **vocab.txt** file, fill in the following table.

	word 1	word 2	word 3	……	word 1000
word 1	c(word1| word1)	c(word1| word2)	c(word1| word3)		c(word1| word1000)
word 2					
word 3					
……					
word 1000					

If you look at the numbers in your table you will see a **lot of zeros**. This is called **data sparsity problem** and causes a major issue by pushing probabilities to zero. A simple fix is to **smooth out the probabilities** by adding one to each count. Then our **new equation of probability** becomes

$$p(x_k \vee x_{k-1}) = \frac{c(x_k, x_{k-1}) + 1}{c(x_{k-1}) + V} \text{-----------------------------------(equation 3)}$$

- where **V** is the words in **vocabulary**.

Convert the counts into probabilities using equation 3 and **save them** in a file called **model**. You can save it in a file extension of your choice.

The **data structure to store such a table is also up to you**.
Advance packages such as pickle, nltk, spacy and advance data structures such as heaps should not be used.

4. Generate Statement

Given your model and a prompt, generate sentences of various lengths.

Inputs	**Input Sentences:** • This is a • What is the purpose • Move From here and
Output	**Suggested Word:** • man • because • so
Details	Chose the word with the highest probability at each location based upon the model. This is called **greedy decoding or inference.** For example for prompt number 1, the probability can be decomposed as p(this)p(is|this)p(a|is)p(..|a) where you chose word with the highest probability at p(..|a).
Sample behaviour	`Enter you stmtMove from here and` `Suggested word =  so`