Project 3
Team 2
results/assessment

A.) Trying out the functions used in lecture 8 acted as a warmup as it helped to initialize variables that would be used for the entirety of the project along with providing valuable insight.

Following the lecture, a VCorpus of the directory (28 files for each of the chapters) was initialized. Upon inspecting the VCorpus with inspect(), the MetaData along with the characters in each file were outputted.
Example:

```
[[22]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 12481
```

The str() function allowed us to see each of the metadata:

```
$ c7.txt :List of 2
 ..$ content: chr [1:459] "The Light of Knowledge" "" "" "After what seemed an eternity to the little sufferer he was able to" ...
 ..$ meta   :List of 7
 .. ..$ author       : chr(0)
 .. ..$ datetimestamp: POSIXlt[1:1], format: "2020-12-03 00:54:16"
 .. ..$ description  : chr(0)
 .. ..$ heading      : chr(0)
 .. ..$ id           : chr "c7.txt"
 .. ..$ language     : chr "en"
 .. ..$ origin       : chr(0)
 .. ..- attr(*, "class")= chr "TextDocumentMeta"
 ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
```

Afterwards, we converted the corpus into a Document term matrix and also inspected and used str() on it:

```
> inspect(TarzanDTM)
<<DocumentTermMatrix (documents: 28, terms: 11677)>>
Non-/sparse entries: 31388/295568
Sparsity           : 90%
Maximal term length: 22
Weighting          : term frequency (tf)
Sample             :
         Terms
Docs      and but for had his she that the was with
  c1.txt   98  28  40  34  54   4   55 240  56   32
  c11.txt 125  21  36  42  90   8   23 328  43   45
  c13.txt 159  20  20  55 105   8   33 367  55   39
  c17.txt  97  19  24  52  37  17   27 305  60   22
  c18.txt 105  35  43  36  38  14   36 232  45   21
  c19.txt 136  29  29  72  41  21   51 299  61   28
  c20.txt 153  30  26  65  66  95   63 279  63   32
  c27.txt 105  25  44  40  23  63   55 211  49   18
  c7.txt  126  26  29  37  93   3   35 373  56   37
  c9.txt  139  28  26  54  96   6   32 263  62   33
>
```

```
> str(TarzanDTM)
List of 6
 $ i       : int [1:31388] 1 1 1 1 1 1 1 1 1 1 ...
 $ j       : int [1:31388] 2 4 5 6 7 8 9 10 11 12 ...
 $ v       : num [1:31388] 1 1 1 1 1 1 1 1 1 1 ...
 $ nrow    : int 28
 $ ncol    : int 11677
 $ dimnames:List of 2
  ..$ Docs : chr [1:28] "c1.txt" "c10.txt" "c11.txt" "c12.txt" ...
  ..$ Terms: chr [1:11677] "_i_" "'ancient" "'appened" "'arf" ...
 - attr(*, "class")= chr [1:2] "DocumentTermMatrix" "simple_triplet_matrix"
 - attr(*, "weighting")= chr [1:2] "term frequency" "tf"
```

This provided more insight as to how many terms there were, and how many sparse entries there were. As you can see, the most common terms were usually “and”, and “the” which are stop words.

We also made a term document matrix, and used inspect() and str(). This showed the same data in a different format.

After subsetting the data. We retrieved the term frequency from the first chapter of the book. We displayed a dataframe of it to show the amount of occurrences for each word.

For analytical purposes we then cleaned the corpus by following along the slides of lecture 8. We converted all the terms to lowercase, removed numbers and punctuation, and removed stop words. After inspection, the amount of terms left were 7583 compared to 11677 in the original corpus. We then played around with finding words that occurred at least 4 times, and played around with the associations of the words such as “girl” where it was most associated with the word “structure”. We then removed sparse words where we removed words with a sparse value of 0.6. We were left with 409 words. We then sorted term frequencies for chapter 1 in the book in decreasing order. We found that the most frequent term was “Tarzan” followed by “Upon”, and “one”. Tokenizing sentences within chapters allowed us to analyze terms within each sentence. And we got sentiment for each phrase within a sentence with the function get_nrc_sentiment(). We also got the weights and TFidf scores with dfm_tfidf() and dfm_weight().

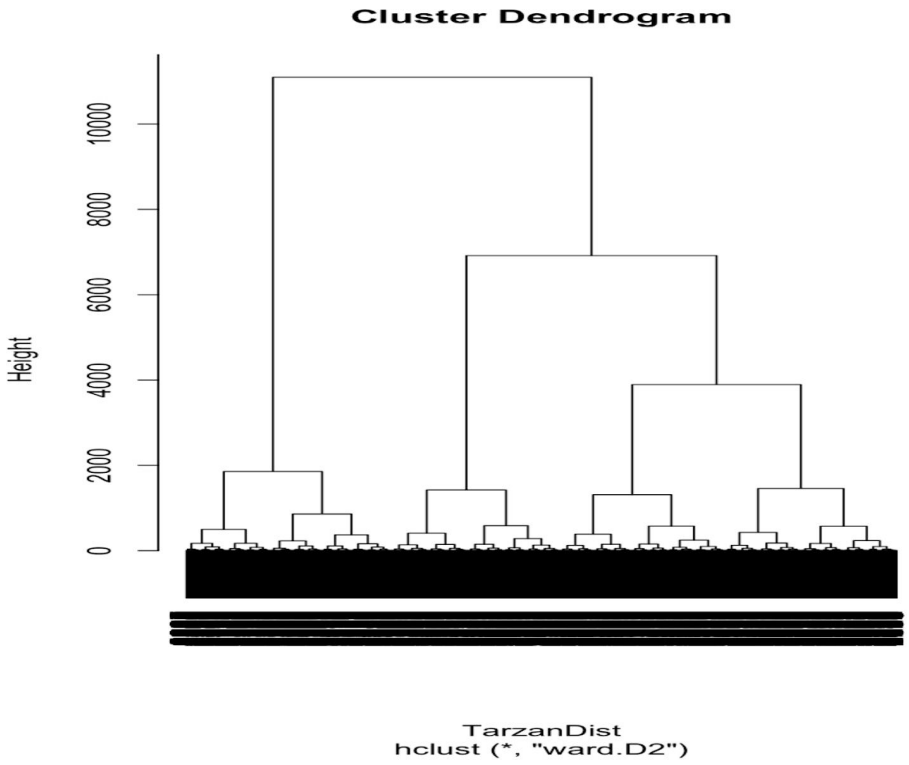
b.)

Longest words/sentences:

	Longest_Words	Word_Length	Longest_Sentences	Sentence_Length
1	playfellows--carefully	22	scene below him, wondering over every feature of thi...	72
2	society--civilization	21	hold with his right hand, which he realized was absol...	72
3	impossible--learning	20	drawn his revolver; the other sailors stood watching t...	72
4	absent-mindedness."	19	In agony the man watched, fearful to launch his spear...	72
5	action--determined,	19	may God bless him and keep him in safety in his wild ...	72
6	admiration--watched	19	Porter to his side, and the two argued in low tones for...	72
7	anthropoid--terkoz.	19	years, chanced to glance toward the harbor. Instantly...	72
8	grizzly--absolutely	19	last night. Oh, Miss Jane, you don't know what I have ...	72
9	north-northeasterly	19	dollars more from Robert Canler, and had given his n...	72
10	civilization--even	18	chest of tools, and the old sails which Black Michael h...	72

The words in all the chapters are put into an array. Then, they are ordered from the longest to the shortest words. Only the first ten words are saved into another array. For the longest sentences, a for loop iterates through all chapters and stores the sentences in descending order including only the ten longest sentences. Lastly, the words and sentences are placed in the data frame bDF.

c.) To create the dendrogram, stop words, numbers, punctuation, and sparse words were removed. The following plot is the dendrogram based on lecture 8.



d.) This part utilizes wordnet package to mark parts of the ten longest sentences from part b for nouns and verbs that are 5 or more characters long. First, the function tokenizes each sentence then uses the wordnet package to identify the noun and the verb using the ‘getTermFilter’ function. If the function is able to find the word’s synonym in the dictionary for the current word as it loops, it is added in the list of nouns or verbs. Lastly, the nouns and verbs are printed.

```
> nounsANDverbs()
[1] "Nouns in Each Sentence:"
[1] "scene"   "feature"
[1] "right"
[1] "revolver" "watching" "scene"
[1] "agony"  "launch" "spear"
[1] "safety" "savage" "jungle"
[1] "Porter"  "minutes"
[1] "years"  "glance" "harbor" "tears"
[1] "night"
[1] "Robert" "given"  "amount"
[1] "chest"   "Black"   "Michael"
[1] ""
[1] ""
[1] ""
```

```
[1] "Verbs in Each Sentence:"
[1] "feature"
[1] "right"
NULL
[1] "launch" "spear"
[1] "bless"  "savage"
[1] "Porter"
[1] "glance" "harbor"
[1] "contend"
[1] "amount"
[1] "Black"
```

e.) For this part, the filtered words from part c are further filtered. Using zipfR function, spc, we can see the number of nouns and verbs in all the chapters as well as the frequency of each type of verb in the entire document. The 'ItaRi.emp.vgc' shows the distinct verbs every 1000 words and how many times each distinct token appeared in the document thus far. Afterwards, we created a model by fitting the expected frequency spectrum to the observed spectrum. Based on the model, we were able to obtain the values of V and Vm at random sample sizes

```
>   head(TarzanFilteredSPC)
  m Vm
1 1 31
2 2 26
3 3 19
4 4 57
5 5 55
6 6 29
```

```
>   summary(TarzanFilteredSPC)
zipfR object for frequency spectrum
Sample size:     N  = 4085755
Vocabulary size: V  = 19271
Class sizes:     Vm = 31 26 19 57 55 29 39 37 ...
```

The first output shows how many times certain types of verbs occurred. For example, 31 ri-types of verbs appeared once, and 26 ri-types appeared twice and so forth. The picture next to it shows the details of the file including the total nouns, verbs, and sample size.

```
>   head(TarzanFilteredVGC)
     N   V V1
1 1000 140 62
2 2000 178 58
3 3000 201 60
4 4000 211 53
5 5000 224 61
6 6000 235 59
```

```
>   summary(TarzanFilteredVGC)
zipfR object for vocabulary growth curve
1400 samples for N = 1000 ... 1399898
Spectrum elements included up to m = 1
```

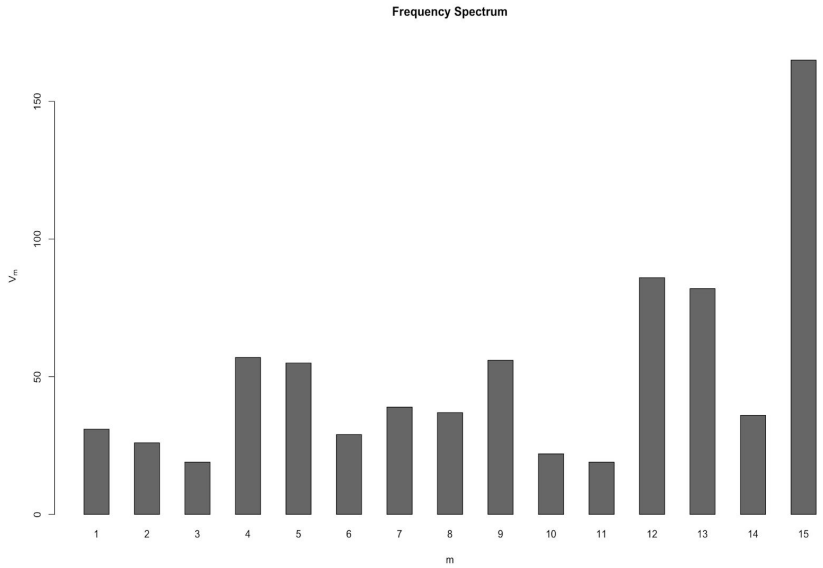
The picture on the left shows the number of each distinct type of token every 1000 tokens with V1 column indicating only a fraction of them appearing one time, twice, etc. The second picture summarizes the sample size and spectrum elements.

```
>   head(TarzanFilteredFsub.ItaRi.bin.vgc)
     N        V       V1
1 1000  966.8444  934.5129
2 2000 1870.6094 1747.5554
3 3000 2715.8619 2452.2892
4 4000 3506.8191 3060.5333
5 5000 4247.3758 3582.8973
6 6000 4941.1304 4028.9013
```

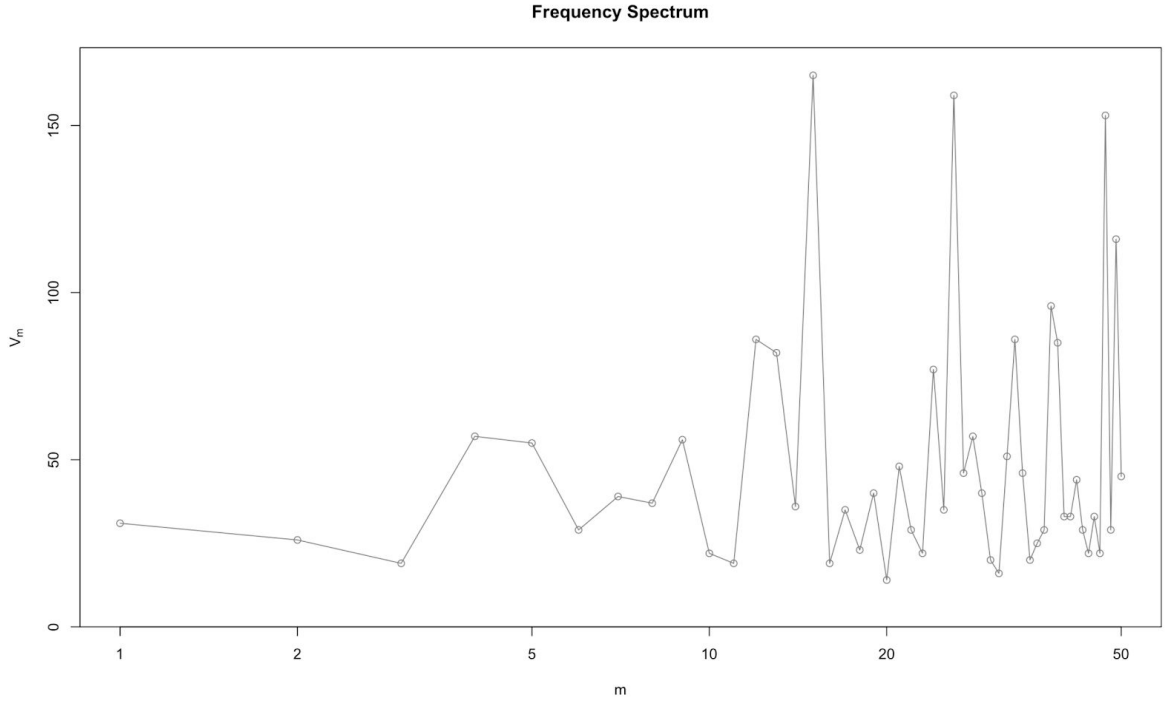
The output shows the vocabulary growth curve for a random sample taken from the frequency spectrum. The second column, V, shows the increasing sample size and the third column shows the number of spectrum elements to include for the VGC generated. The screen shot is of the first 6 classes.

```
>   summary(TarzanFilteredFZM)
finite Zipf-Mandelbrot LNRE model.
Parameters:
   Shape:          alpha = 0.03403301
   Lower cutoff:       A = 1.20772e-06
   Upper cutoff:       B = 0.001893328
 [ Normalization:      C = 412.504 ]
Population size: S = 4268.943
Sampling method: Poisson, approximations are allowed.

Parameters estimated from sample of size N = 4085755:
                     V    V1    V2    V3    V4    V5
   Observed: 19271.00 31.00 26.00 19.00 57.00 55.00 ...
   Expected: 19271.62  4.69 13.91 28.26 44.58 58.67 ...

Goodness-of-fit (multivariate chi-squared test):
         X2 df            p
   174.5822  3 1.304229e-37
```

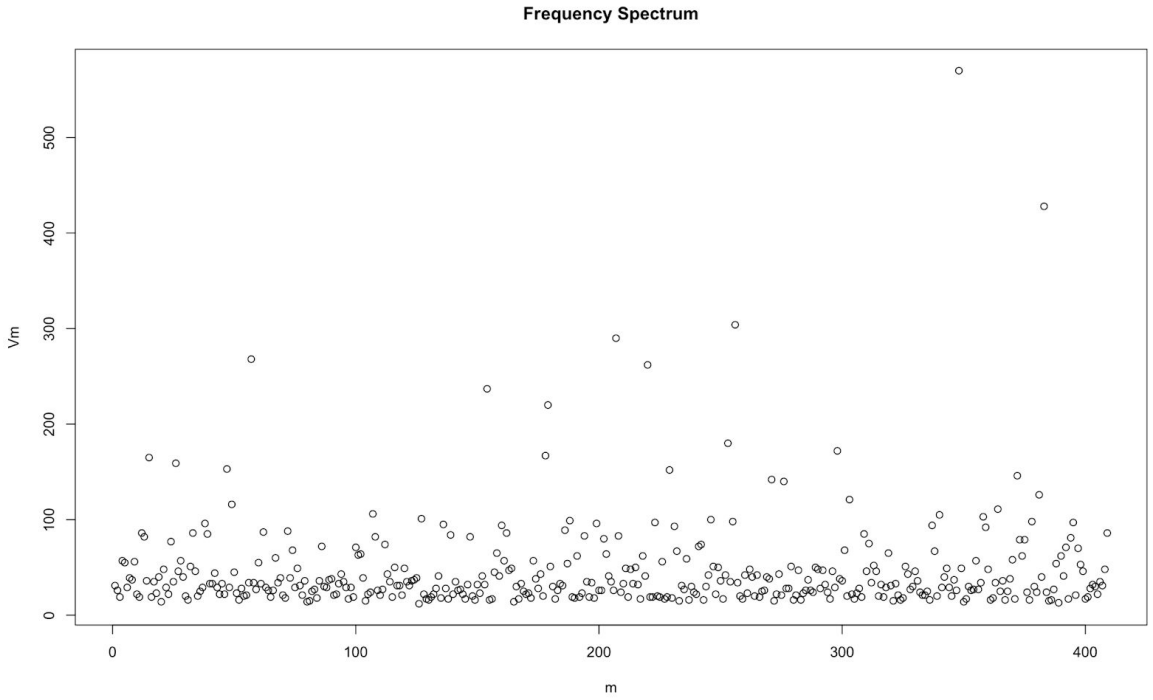
With a p-value of 1.304229e-37, the chi-squared test shows that the model is not a very good fit; thus, an accurate predictor for possible observations.



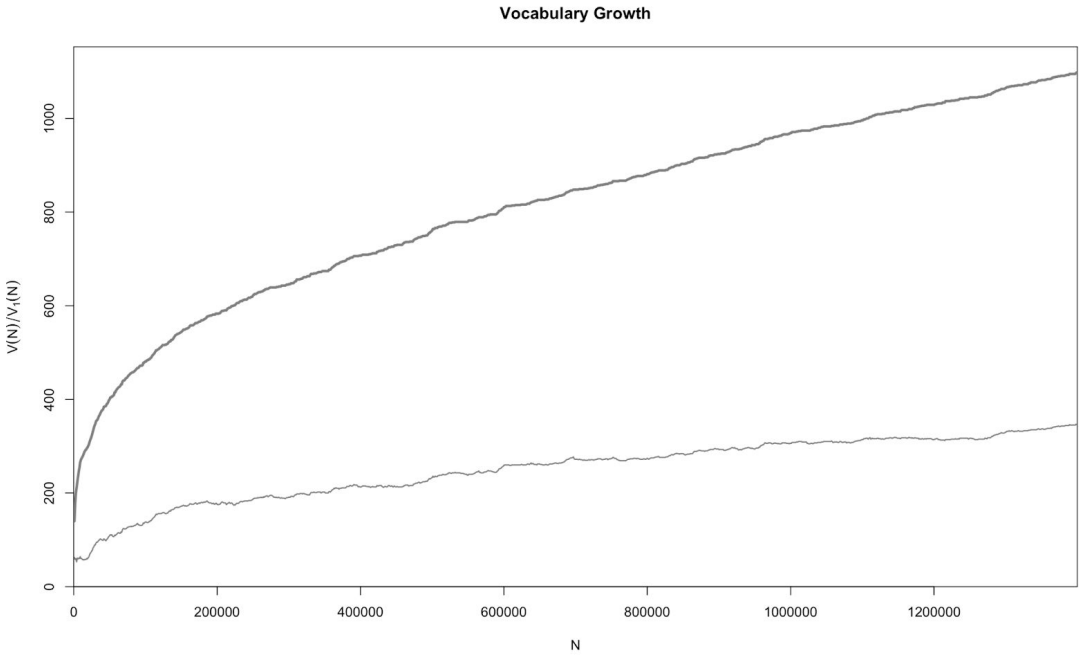
This frequency spectrum graph shows the first 15 frequency classes in the spectrum.



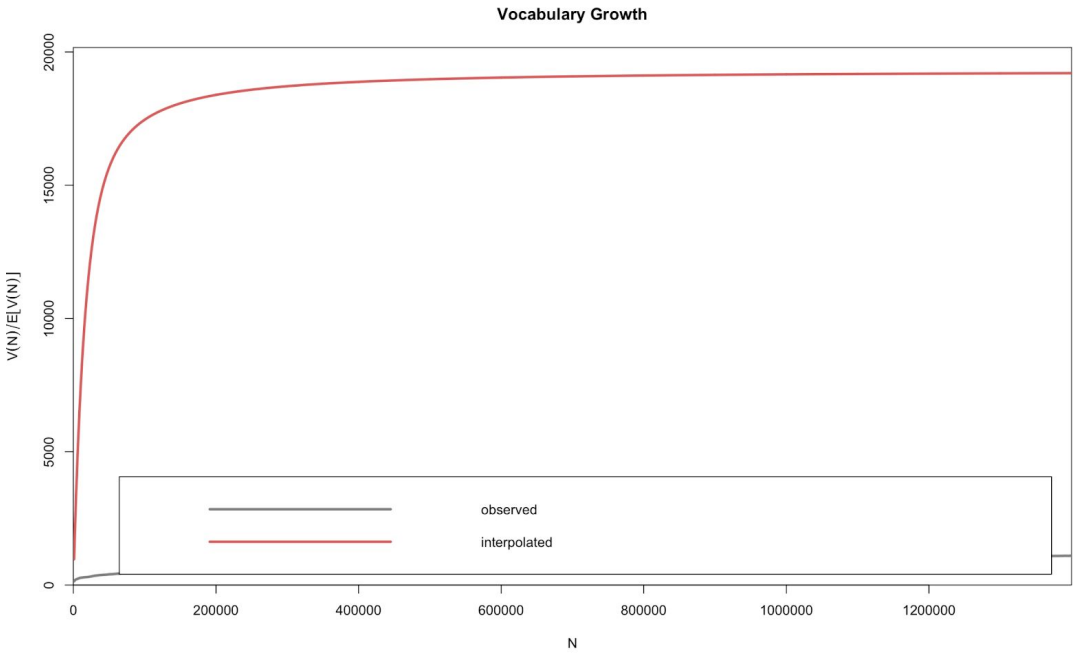
The frequency spectrum graph above shows the frequency spectrum if log x is passed in and shows the first 50 classes.



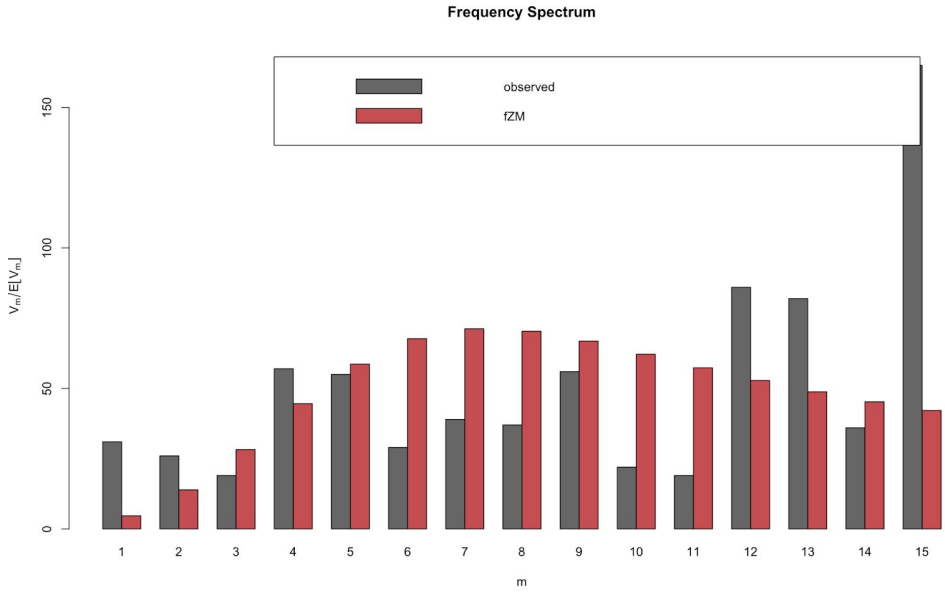
The scatter plot shows a different way to represent the frequency spectrum as seen in graph one.



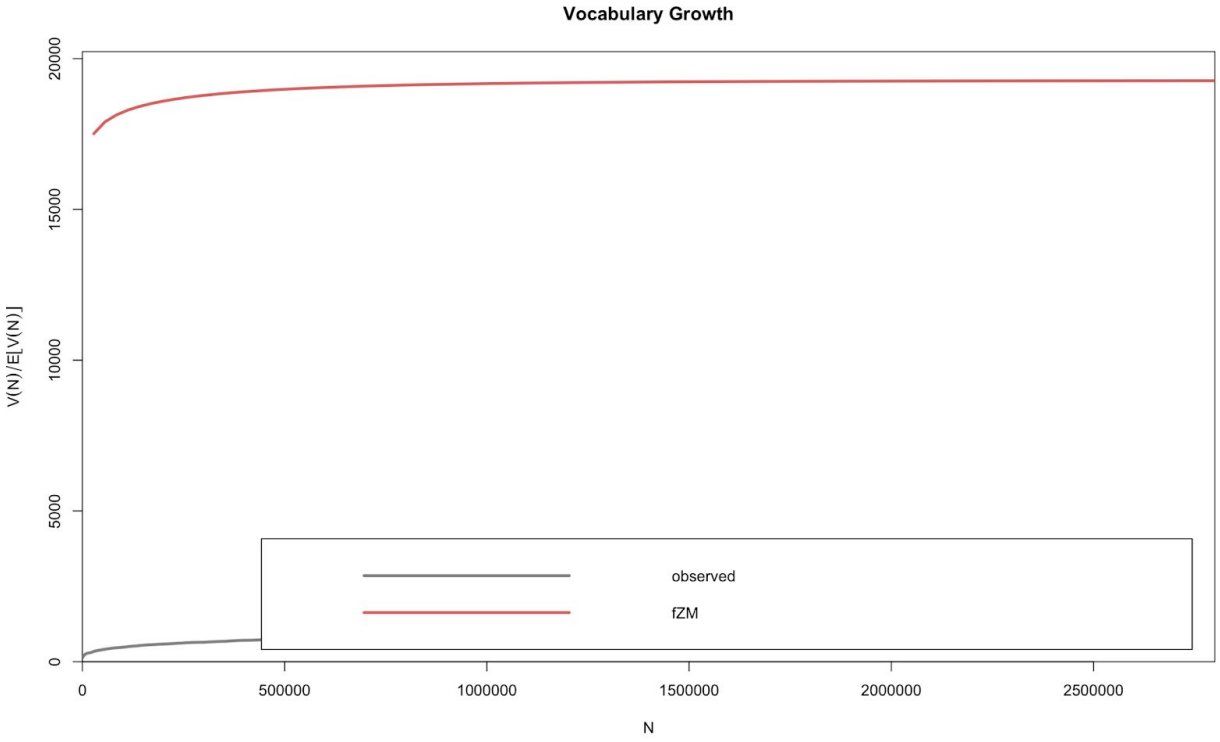
In the graph above, the bottom line is the VGC curve for a spectrum element of 1 meaning each type of word only shoes up once while the line above is the observed VGC



The graph above shows the model is not an accurate predictor of what can be observed with real data in the vocabulary growth curves especially as the sample size grows.



The graph shows that the fZM model is not an accurate predictor of VCGs especially as the spectrum size increases.



The graph above shows once again the fZM model is not an accurate predictor of VGCs.

f.) For this part, the function uses quanteda's method 'tokens' by creating an intermediary tokenizer that tokenizes 10 sentences and words that at least have six characters. Then, the function loops through each sentence and prints out the ngram chosen.

```
> bigramORtrigram(2)
[[1]]
[1] "wondering" "feature"

[[2]]
[1] "feature" "strange"

[[1]]
[1] "realized"   "absolutely"

[[2]]
[1] "absolutely"   "unassailable"

[[1]]
[1] "revolver" "sailors"

[[2]]
[1] "sailors"  "watching"
```

```
> bigramORtrigram(3)
[[1]]
[1] "wondering" "feature"   "strange"

[[1]]
[1] "realized"     "absolutely"   "unassailable"

[[1]]
[1] "revolver" "sailors"  "watching"

[[2]]
[1] "sailors"  "watching" "intently"

[[1]]
[1] "watched" "fearful" "launch"

[[2]]
```

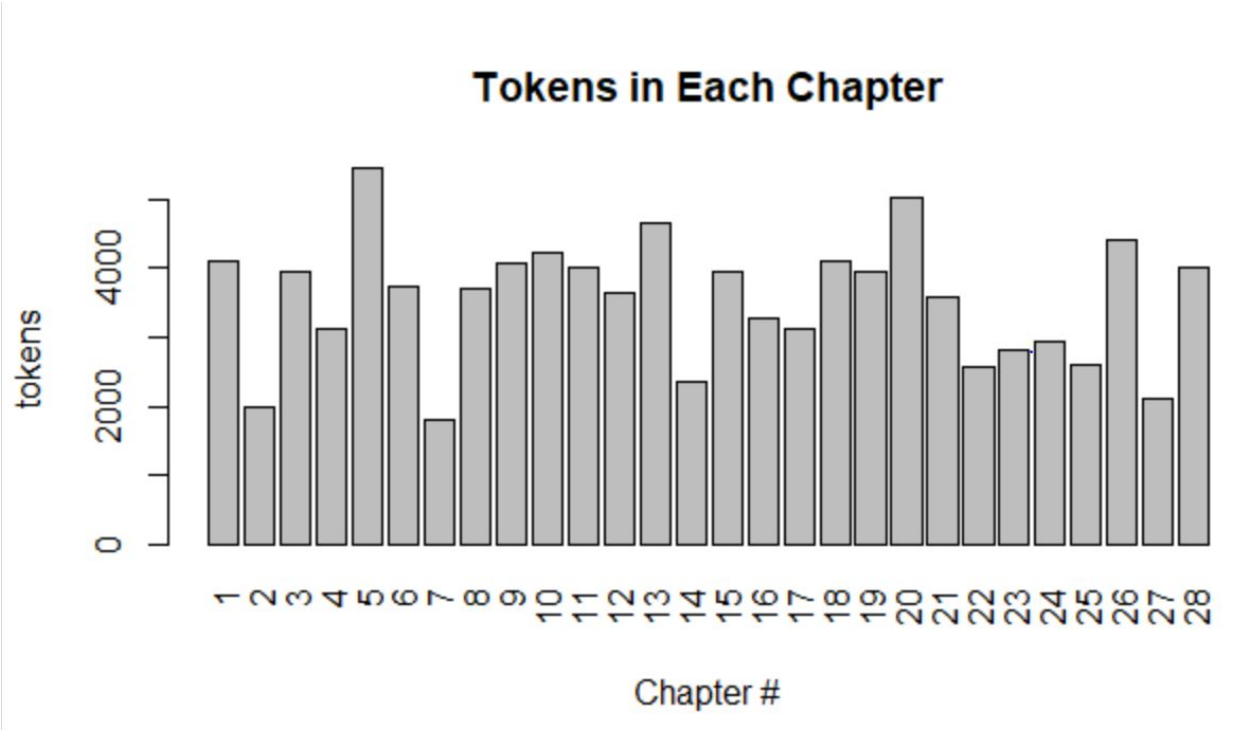
g.)
First function, findWordinPhrase, uses the stringi package to detect the specified word from the document. The method used was 'stri_detect' to detect the pattern, the word, from the document. The function utilizes a nested loop with the first one iterating through each chapter, and the second loop going through each phrase in every chapter. It also has a counter to keep track of the

occurrences of the specified word. Every time the word us detected, the function prints out the sentence.

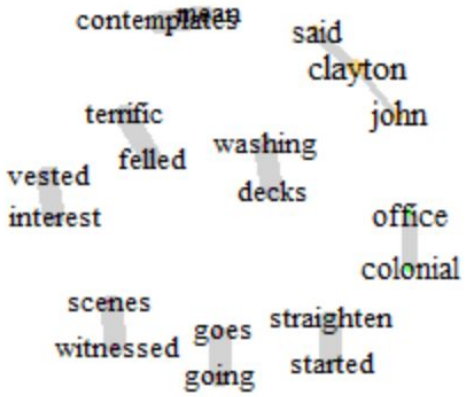
```
[1] "turning   path tarzan dropped quietly   ground   "
[1] "surprised tarzan greatly    black warrior  ruined "
[1] "     may tarzan   ruin good meat   "
[1] " day tarzan followed kulonga hovering     trees like"
[1] "tarzan thought much   wondrous method  slaying   swung"
[1] " crouched tarzan   apes"
[1] "tarzan   apes  swing quietly   wake"
[1] "fifty feet higher   thus tarzan blazed  forest trails  marked"
[1] " kulonga continued  journey tarzan closed     traveled"
[1] " moment  delayed   tarzan  anxious  ascertain "
[1] "tarzan  directly  kulonga   made  discovery   forest"
[1] "tarzan must act quickly   prey   gone  tarzans life"
[1] " quickly  tarzan   apes drag back  prey  kulongas cry"
[1] " alarm  throttled   windpipe  hand  hand tarzan drew "
[1] "tarzan climbed   larger branch drawing  still threshing victim"
[1] "tarzan examined  black minutely    never seen  "
[1] " tarzan   apes  hungry    meat meat   kill"
[1] "Number of sentences with the word 'tarzan': 613"
>
```

Second function, TarzanWordCloud, also uses the stringi package and word cloud package. The method ‘stri_detect’ from the stringi package was used as well as the ‘wordcloud’ from the word cloud package. The function uses a nested loop that loops through each chapter and phrase, like the first function, and if the word is found in the phrase, then the phrase is appended to a list. Then, all the words in the list are tokenized using the ‘tokenize_word’ method. The words are placed in a new list after removing the white spaces. Methods ‘termFreq’ and ‘names’ are used to determine the frequency of each word then print the name of each word. Lastly, the word cloud is generated and initialized by the words, their frequency, and color.

The third function plots the number of tokens for each chapter. The function utilizes the ‘tokenize_word’ from the quanteda library. The function uses a nested for loop to iterate through each chapter and tokenize each word while removing white space. A bar plot is generated with the chapter vs number of tokens for each chapter.



The fourth function is a combination of methods from the corpus package. The ‘create_tcorpus’ method to create a corpus. The ‘semnet_window’ method is called to create a semantic network based on the co-occurrences of tokens from the chapters. The ‘plot_semnet’ method is used to display the network as shown in the screenshot below.



discussion/conclusion:

This project allowed us to explore different packages to analyze the story. In order to be efficient, we learned to split the large document into separate documents and filter out sparse words and white spaces to make meaningful conclusions or observations from the text. This project also taught us the importance of transforming unstructured data, which is how most of the information in our world is stored, into structured data. In terms of extracting a meaningful observation, the function ‘termFreq’ allows us to see the frequency of a term in a document in which we can determine the importance of a word. For example, the word tarzan appears more than 600 times, so we can see it is a critical component of the story, essentially the core of the story.

Another important takeaway is the functions ‘semnet_window’ and ‘plot_semnet’ that generate the semantic network, so we can see what words are present usually in pairs. Having tools like this in text analysis lets us see relationships between words and patterns that we normally could not with our eyes. Using text analysis tools is an asset when analyzing large documents to derive patterns from unstructured data and natural language.