LNMIIT
The LNM Institute of
Information Technology

NLP PROJECT REPORT

Submitted by

TextTech Titans

Mohit Jain	**21ucs131**
Rohan Choundiye	**21ucs172**
Sahil Gupta	**21ucs175**
Patel Shubh Dipakkumar	**21ucs201**

GitHub Link For Code:

https://github.com/Sahil26007/NLP-round1

ACKNOWLEDGEMENT

We would especially like to express our gratitude to our project mentor, Dr. Sakthi Balan, for providing us with the opportunity to work on this wonderful text analysis project(NLP Project Round 1). We were able to do a great deal of research for this project, and it was a great learning experience.

TABLE OF CONTENTS

TITLE

ABSTRACT

We used the Novel A Man Called Ove as a guide for this project's text preparation, POS tagging, and other techniques. In the discipline of natural language processing, text analytics is an essential component. It was interesting to work on this assignment because of the book's simple writing style and ease of comprehension. The Natural Language Tool Kit, or NLTK, was used for all of the preprocessing, tokenization, and tagging that we performed. Our expertise of a few text mining subfields was improved by the project, which also provided an explanation of the book's frequency distribution and Word Cloud. The plotted graphs in the report also reveal a great deal about the input text, which is taken from the book and the author's writing style, including the terminology he used frequently, key phrases, conclusive words, etc. This project can also be used to comprehend the vocabulary, word frequency, and text file difficulty of a certain file.

1. LITERATURE REVIEW

Numerous researchers worked on NLP, creating systems and tools that facilitate understanding of the field. NLP is a good field for research because of tools like Sentiment Analyzer, Parts of Speech (POS) Taggers, Chunking, Named Entity Recognition (NER), and Semantic Role Labelling.

2. INTRODUCTION

In this project, we imported a text file from a book and used NLP techniques to perform text analysis on it. We performed POS tagging on the imported text file, tokenization and lemmatization, frequency analysis, and frequency distribution analysis on the text file. We are now going to visualize the data, which will be an excellent learning opportunity for NLP. We have chosen a fiction novel.

BOOK NAME - A MAN CALLED OVE

We used NLTK, a set of Python-written modules and tools for statistical and symbolic natural language processing of English, to meet the project's aims. Furthermore, we have included a few functions and modules to do other operations, such as preprocessing, tokenizing, removing stop words, etc. We also created word clouds and displayed the text file's frequency distribution after applying these processes to the text file.

3. Python Libraries and Modules Used

Nltk.stem package: Interfaces used to remove morphological affixes from words, leaving only the word stem.

Nltk.corpus : The modules in this package provide functions that can be used to read corpus files in a variety of formats.

Collection modules : Provide different types of container data types**.**

Nltk.probability : A probability distribution specifies how likely it is that an experiment will have any given output.

Nltk.tokenizer package : Tokenizer divides strings into a list of substrings.

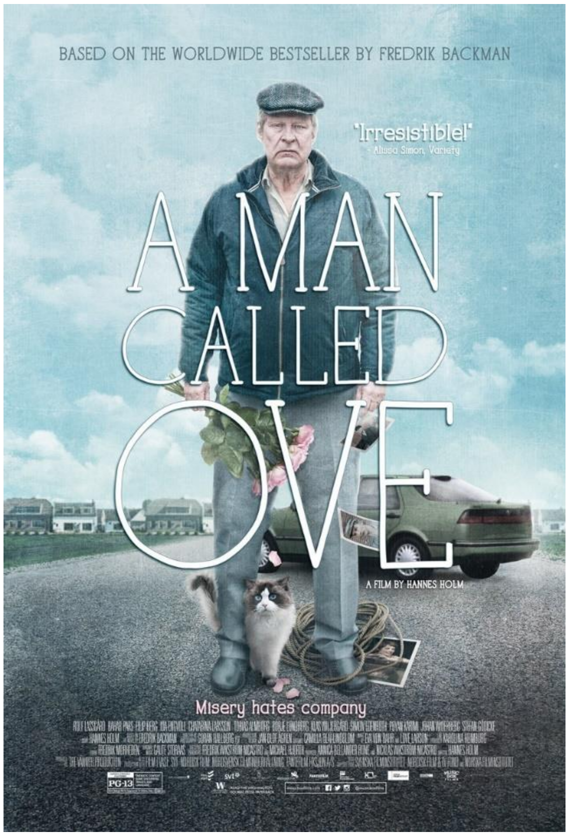
Wordcloud package : Provides modules to create wordcloud in python.

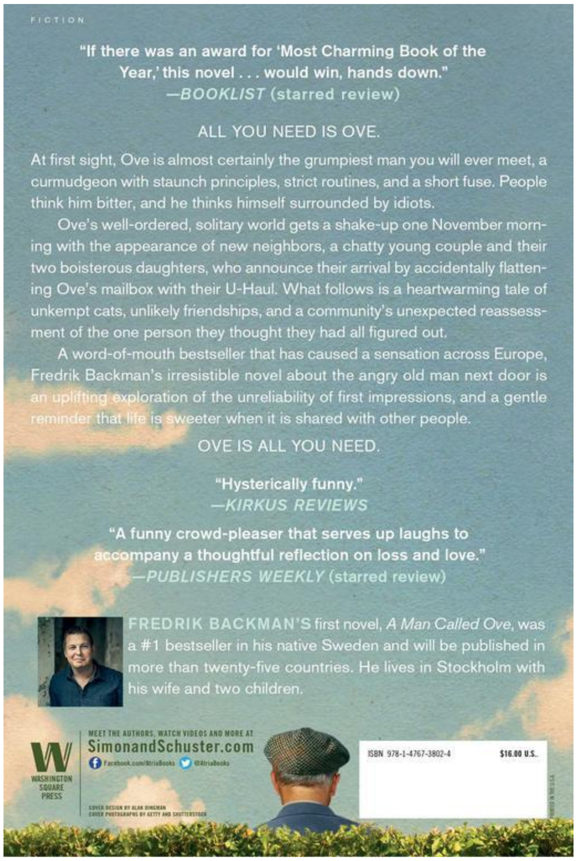
4. METHODOLOGY

Downloading Book:

We have downloaded this book from Google with the following URL
https://dx35vtwkllhj9.cloudfront.net/sodapictures/a-man-called-ove/downloads/A%20Man%20Called%20Ove%20Extract.pdf

The downloaded form of the book is in PDF.So We convert PDF to text files using online tools.




FICTION

"If there was an award for 'Most Charming Book of the Year,' this novel . . . would win, hands down."
—*BOOKLIST* (starred review)

ALL YOU NEED IS OVE.

At first sight, Ove is almost certainly the grumpiest man you will ever meet, a curmudgeon with staunch principles, strict routines, and a short fuse. People think him bitter, and he thinks himself surrounded by idiots.

Ove's well-ordered, solitary world gets a shake-up one November morning with the appearance of new neighbors, a chatty young couple and their two boisterous daughters, who announce their arrival by accidentally flattening Ove's mailbox with their U-Haul. What follows is a heartwarming tale of unkempt cats, unlikely friendships, and a community's unexpected reassessment of the one person they thought they had all figured out.

A word-of-mouth bestseller that has caused a sensation across Europe, Fredrik Backman's irresistible novel about the angry old man next door is an uplifting exploration of the unreliability of first impressions, and a gentle reminder that life is sweeter when it is shared with other people.

OVE IS ALL YOU NEED.

"Hysterically funny."
—*KIRKUS REVIEWS*

"A funny crowd-pleaser that serves up laughs to accompany a thoughtful reflection on loss and love."
—*PUBLISHERS WEEKLY* (starred review)

FREDRIK BACKMAN'S first novel, *A Man Called Ove*, was a #1 bestseller in his native Sweden and will be published in more than twenty-five countries. He lives in Stockholm with his wife and two children.

W
WASHINGTON
SQUARE
PRESS

MEET THE AUTHORS, WATCH VIDEOS AND MORE AT
SimonandSchuster.com
Facebook.com/AtriaBooks @AtriaBooks

COVER DESIGN BY ALAN DINGMAN
COVER PHOTOGRAPHS BY GETTY AND SHUTTERSTOCK

ISBN 978-1-4767-3802-4 $16.00 U.S.

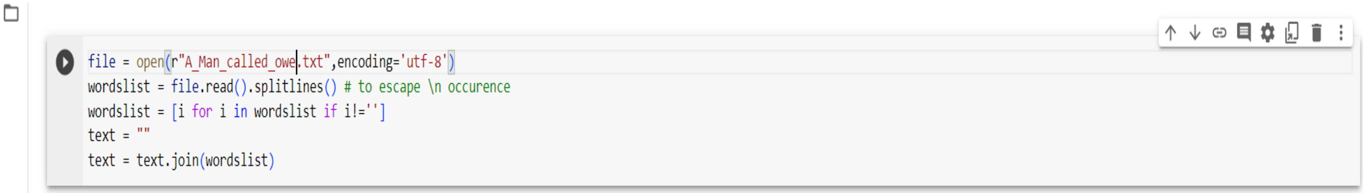
Above are the cover photos of the novel.

Importing the text:

In this step, we created a function (txt_file_to_string) to read the text imported from the book and convert it into a string for processing in Python. This function takes as input the path of the file to be read and returns the content of the file in string format.

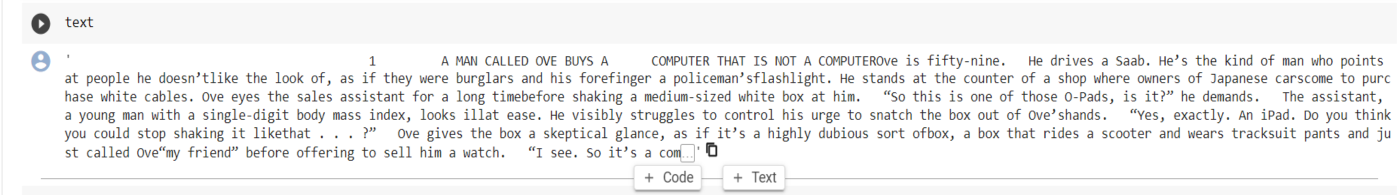
Text Before Preprocessing:

Code:



```
file = open(r"A_Man_called_owe.txt",encoding='utf-8')
wordslist = file.read().splitlines() # to escape \n occurence
wordslist = [i for i in wordslist if i!='']
text = ""
text = text.join(wordslist)
```

Image:



```
text
```

```
'                                    1         A MAN CALLED OVE BUYS A      COMPUTER THAT IS NOT A COMPUTEROve is fifty-nine.   He drives a Saab. He’s the kind of man who points
at people he doesn’tlike the look of, as if they were burglars and his forefinger a policeman’sflashlight. He stands at the counter of a shop where owners of Japanese carscome to purc
hase white cables. Ove eyes the sales assistant for a long timebefore shaking a medium-sized white box at him.   “So this is one of those O-Pads, is it?” he demands.   The assistant,
a young man with a single-digit body mass index, looks illat ease. He visibly struggles to control his urge to snatch the box out of Ove’shands.   “Yes, exactly. An iPad. Do you think
you could stop shaking it likethat . . . ?”   Ove gives the box a skeptical glance, as if it’s a highly dubious sort ofbox, a box that rides a scooter and wears tracksuit pants and ju
st called Ove“my friend” before offering to sell him a watch.   “I see. So it’s a com...'
```

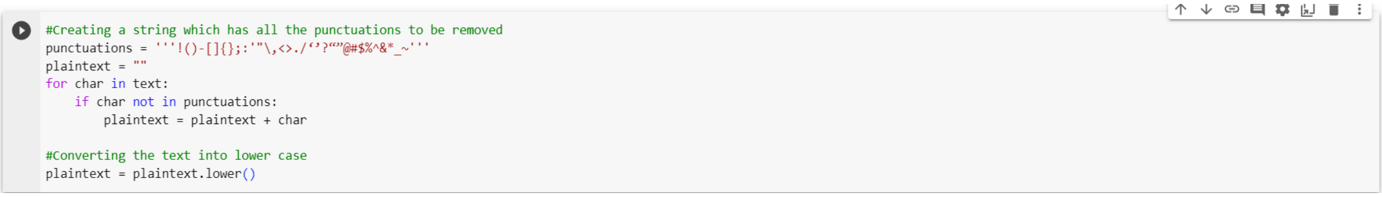
Text Pre-Processing :

1. Prefix and suffixes were removed in order to focus on the text of the eBook. The book from the website had extra prefixes and suffixes attached in its.txt file, in addition to the eBook's contents.

2. Lowercase - We used the string function string to change our string to plaintext. Lower().

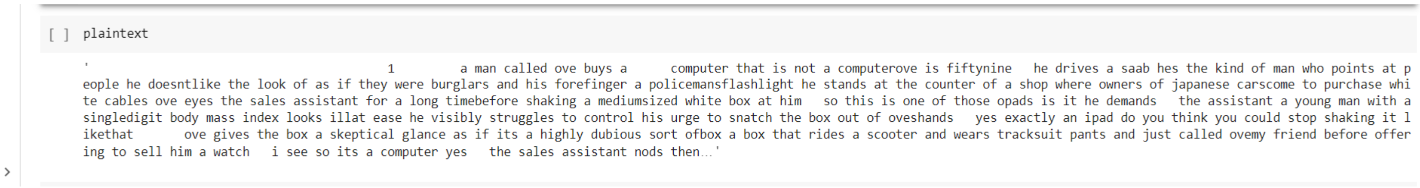
3. Removal Of Punctuations - We removed all the punctuation using regular expressions in two steps by first replacing everything other than word and whitespace characters with an empty string and then replacing _ (underscore, which is considered part of a word in Python) by the empty string.

Code:



```
#Creating a string which has all the punctuations to be removed
punctuations = '''!()-[]{};:'"\,<>./‘’?“”@#$%^&*_~'''
plaintext = ""
for char in text:
    if char not in punctuations:
        plaintext = plaintext + char

#Converting the text into lower case
plaintext = plaintext.lower()
```

Image:



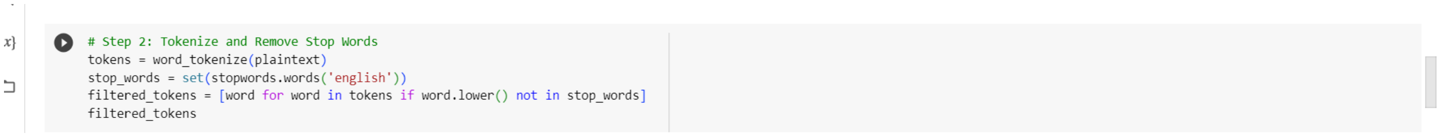
```
[ ] plaintext

'                                        1         a man called ove buys a       computer that is not a computerove is fiftynine    he drives a saab hes the kind of man who points at p
eople he doesntlike the look of as if they were burglars and his forefinger a policemansflashlight he stands at the counter of a shop where owners of japanese carscome to purchase whi
te cables ove eyes the sales assistant for a long timebefore shaking a mediumsized white box at him   so this is one of those opads is it he demands   the assistant a young man with a
singledigit body mass index looks illat ease he visibly struggles to control his urge to snatch the box out of oveshands   yes exactly an ipad do you think you could stop shaking it l
ikethat       ove gives the box a skeptical glance as if its a highly dubious sort ofbox a box that rides a scooter and wears tracksuit pants and just called ovemy friend before offer
ing to sell him a watch   i see so its a computer yes   the sales assistant nods then...'
```

Tokenization and Removing stop words:

We tokenized the string into single words using word tokenize() function imported from NLTK and stored them.

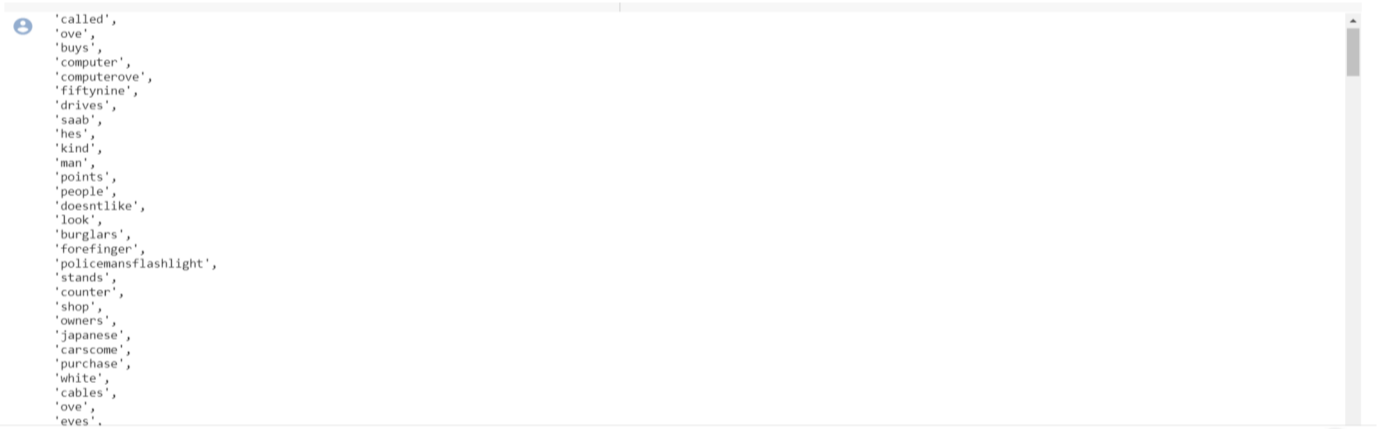
After tokenization, we remove the stop words from the text. We used the nltk.corpus stop words from English and removed them from the text.

Code:



```
# Step 2: Tokenize and Remove Stop Words
tokens = word_tokenize(plaintext)
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
filtered_tokens
```

Output Image:



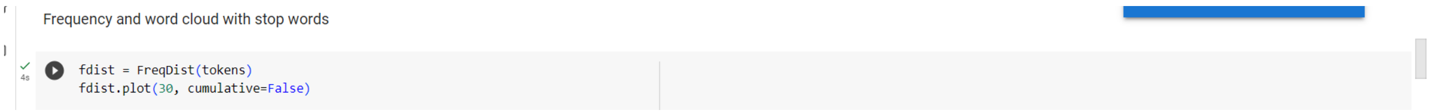
```
'called',
'ove',
'buys',
'computer',
'computerove',
'fiftynine',
'drives',
'saab',
'hes',
'kind',
'man',
'points',
'people',
'doesntlike',
'look',
'burglars',
'forefinger',
'policemansflashlight',
'stands',
'counter',
'shop',
'owners',
'japanese',
'carscome',
'purchase',
'white',
'cables',
'ove',
'eves',
```

Plotting Frequency distribution of tokens :

● Following the text's tokenization, we imported the FreqDist() function from the nltk.probability module. This function is useful for probability computations as it calculates the frequency at which each experiment's outcome happens. By providing the tokenized and lemmatized lists to the aforementioned function, we were able to save the frequency distribution of the two strings in the FreqDist object.

We imported the function figure() from the matplotlib.pyplot module, which is used to build a figure object, in order to plot the frequency distribution graph. The figure object is understood to include the entire figure. The frequency distribution graph was then plotted:

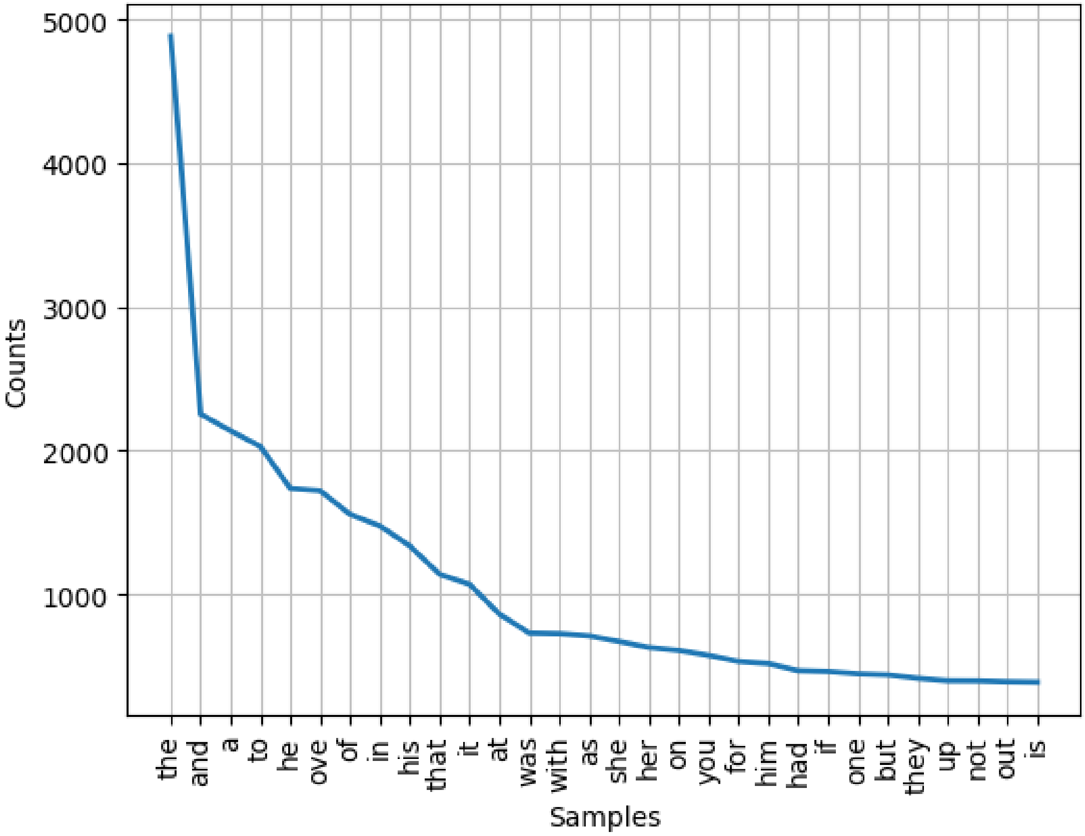
Code without removing stopwards:



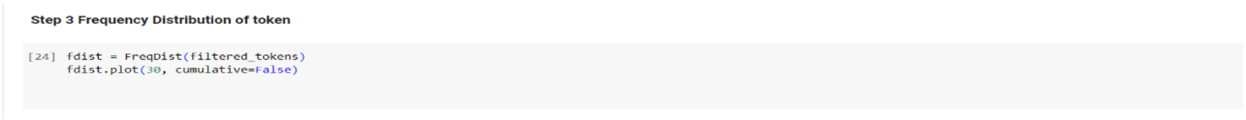
Frequency and word cloud with stop words

```
fdist = FreqDist(tokens)
fdist.plot(30, cumulative=False)
```

Output Image :



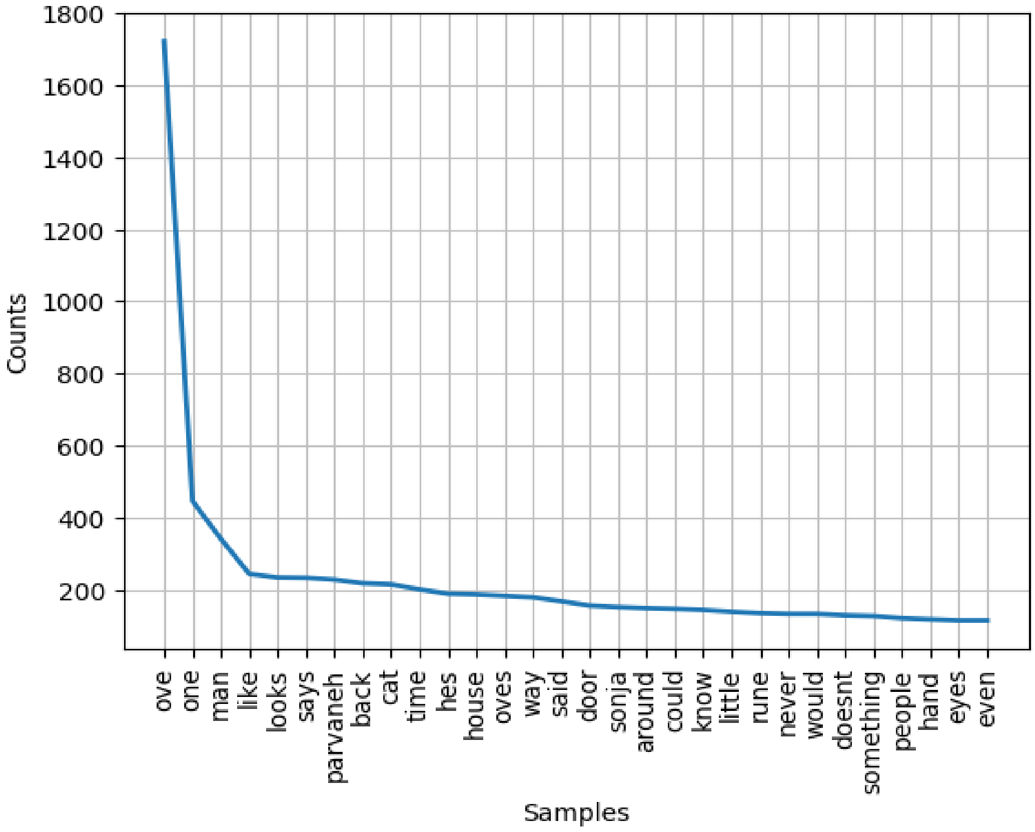
Code with removing stopwards:



Step 3 Frequency Distribution of token

```
[24] fdist = FreqDist(filtered_tokens)
     fdist.plot(30, cumulative=False)
```

Output Image :



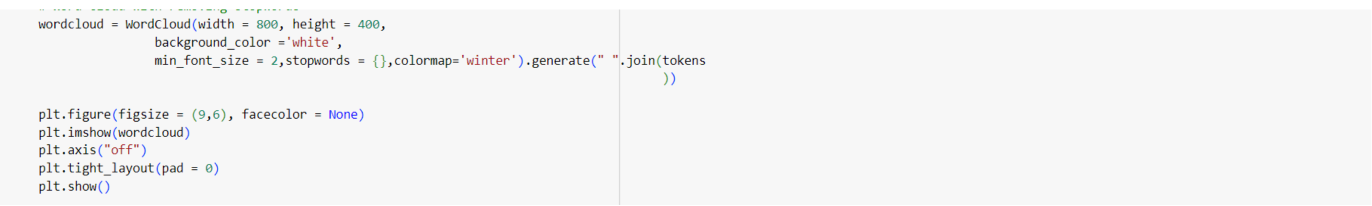
Inference :

- We observe that words like ove, one, man and like are now amongst the words that appear bigger as their frequency is higher relatively after removal of stop words.
- It is giving the visual representation of the most frequent words in the book “A Man called Ove”after removal of stop words.

Creating Word Cloud:

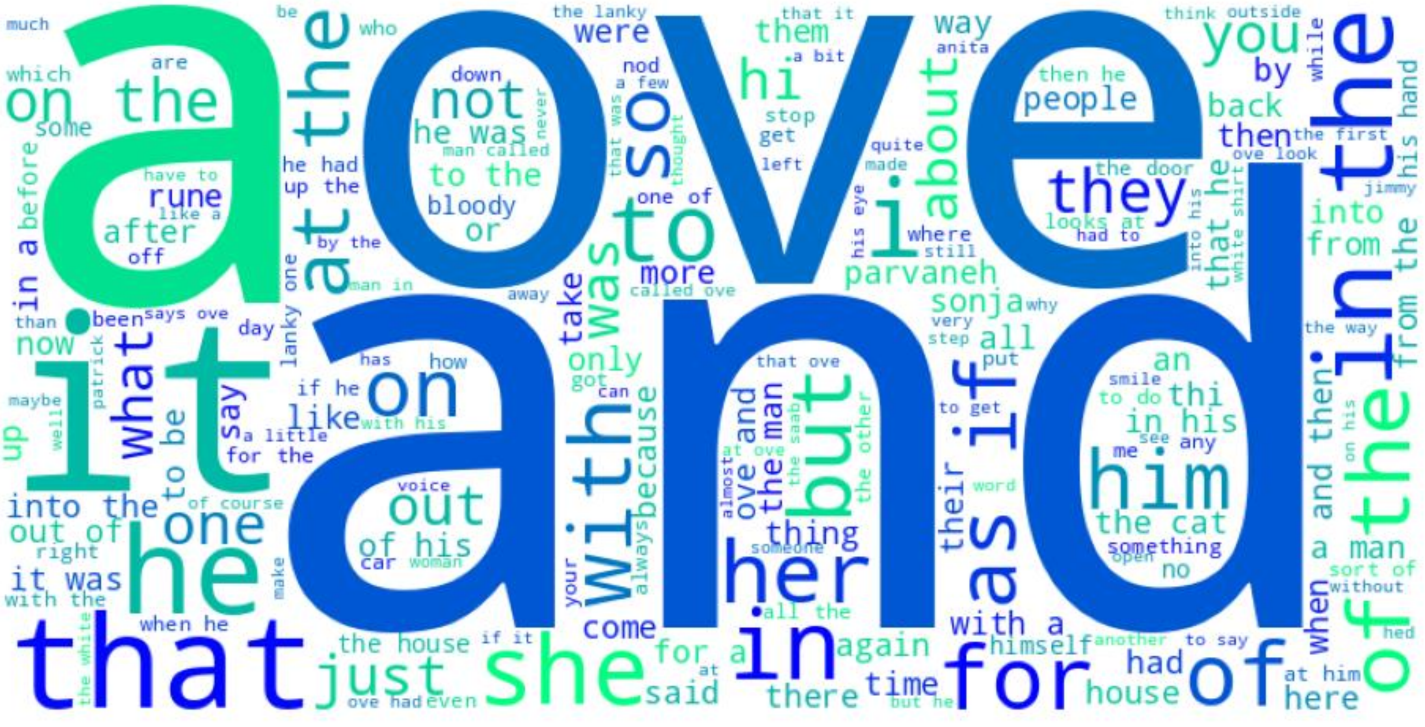
We used Counter, which was imported from the Collections module, to create a word cloud. Next, a word cloud was loaded from the word cloud module. Next, we visualized the word cloud using the plt.figure(), plt.show(), and plt.axis() functions. The word cloud is given below:

Code without removing stopwords:

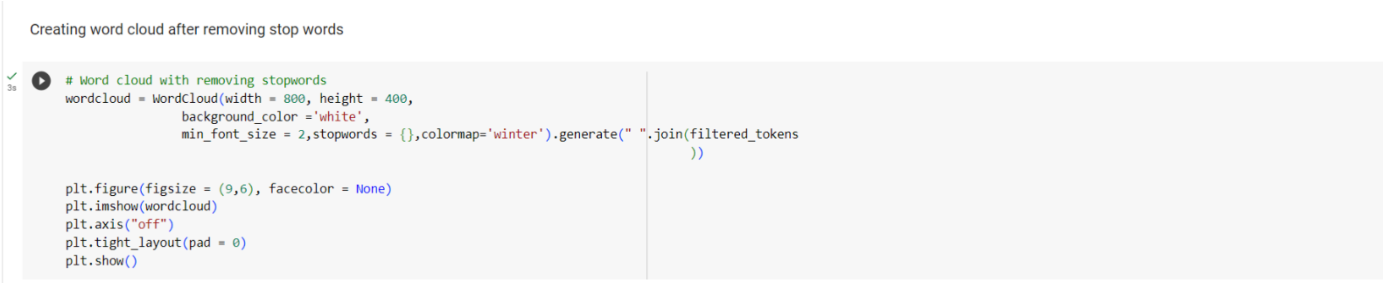


```
wordcloud = WordCloud(width = 800, height = 400,
                background_color ='white',
                min_font_size = 2,stopwords = {},colormap='winter').generate(" ".join(tokens
                                                                    ))


plt.figure(figsize = (9,6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

Image:



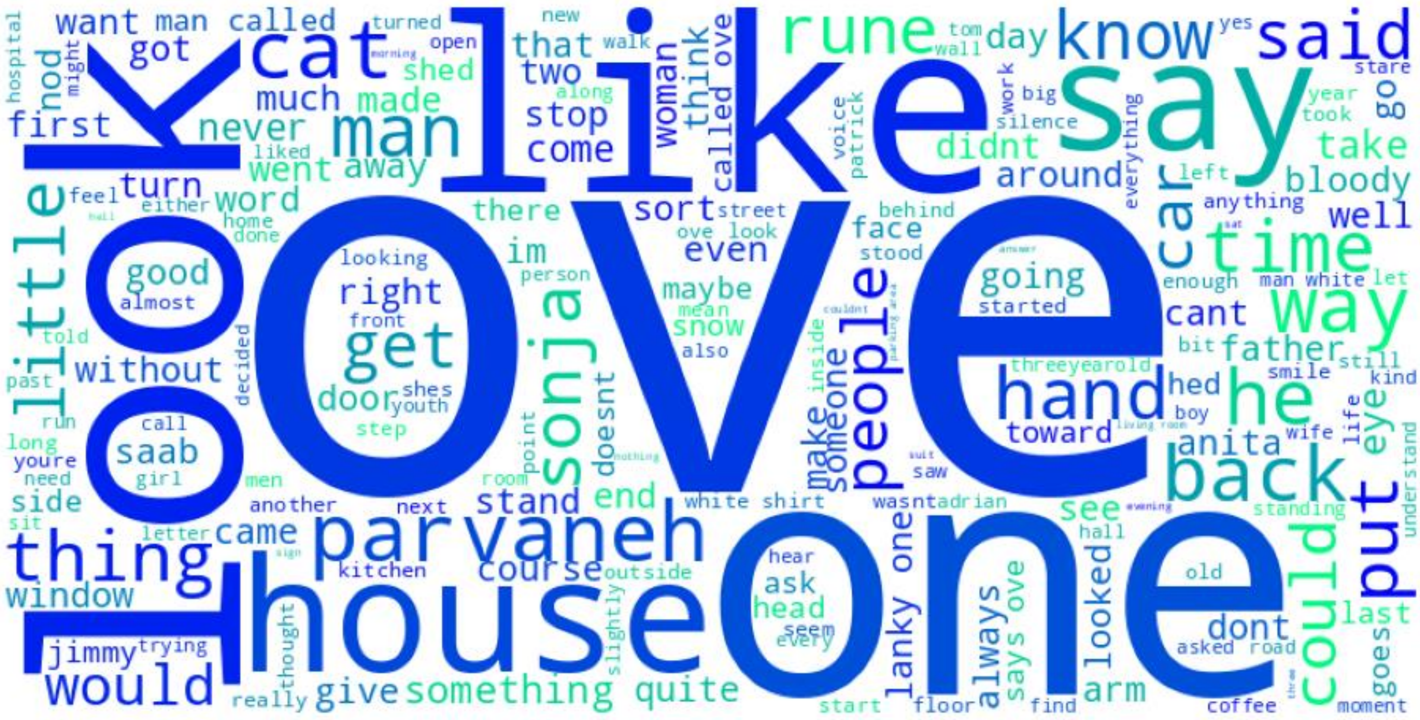
Code without removing stopwords:



Creating word cloud after removing stop words

```
# Word cloud with removing stopwords
wordcloud = WordCloud(width = 800, height = 400,
                background_color ='white',
                min_font_size = 2,stopwords = {},colormap='winter').generate(" ".join(filtered_tokens
                                                                                  ))


plt.figure(figsize = (9,6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

Image:



Inference :

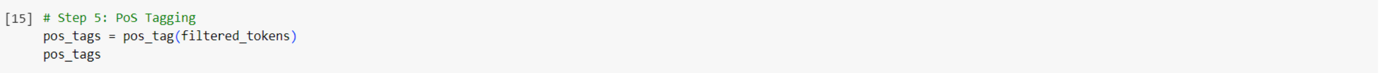
- It provides a graphic depiction of the terms that appear most frequently in the book.
- We may deduce that stop words are highly frequent in the text since we see that terms like the, and, of, and an are among those that seem bigger and have a high frequency in the previous plot. In fact, stop words make up a large portion of the word cloud overall..

POS Tagging:

Part-of-speech (POS) tagging in Natural Language Processing (NLP) is the process of assigning grammatical tags to each word in a sentence, indicating its part of speech (e.g., noun, verb, adjective, etc.).

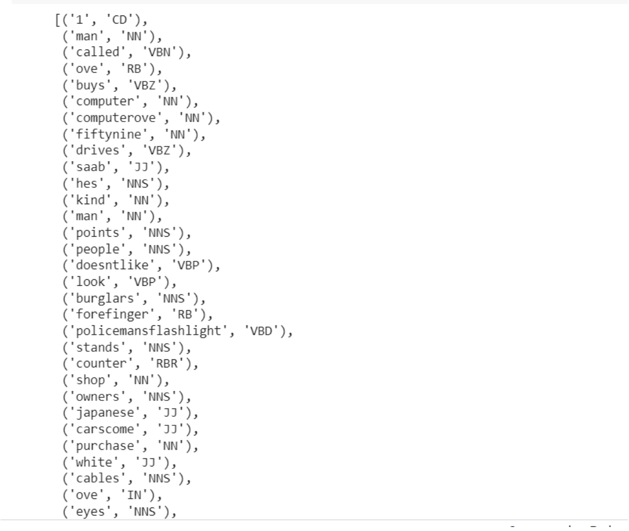
In this code we use average perceptron tagger to tag our tokens. The nltk.tag.AveragedPerceptronTagger is the default tagger as of NLTK version 3.1.

Code:



```
[15] # Step 5: PoS Tagging
     pos_tags = pos_tag(filtered_tokens)
     pos_tags
```

Image:



```
[('1', 'CD'),
 ('man', 'NN'),
 ('called', 'VBN'),
 ('ove', 'RB'),
 ('buys', 'VBZ'),
 ('computer', 'NN'),
 ('computerove', 'NN'),
 ('fiftynine', 'NN'),
 ('drives', 'VBZ'),
 ('saab', 'JJ'),
 ('hes', 'NNS'),
 ('kind', 'NN'),
 ('man', 'NN'),
 ('points', 'NNS'),
 ('people', 'NNS'),
 ('doesntlike', 'VBP'),
 ('look', 'VBP'),
 ('burglars', 'NNS'),
 ('forefinger', 'RB'),
 ('policemansflashlight', 'VBD'),
 ('stands', 'NNS'),
 ('counter', 'RBR'),
 ('shop', 'NN'),
 ('owners', 'NNS'),
 ('japanese', 'JJ'),
 ('carscome', 'JJ'),
 ('purchase', 'NN'),
 ('white', 'JJ'),
 ('cables', 'NNS'),
 ('ove', 'IN'),
 ('eyes', 'NNS'),
```

Frequency distribution of pos tags:

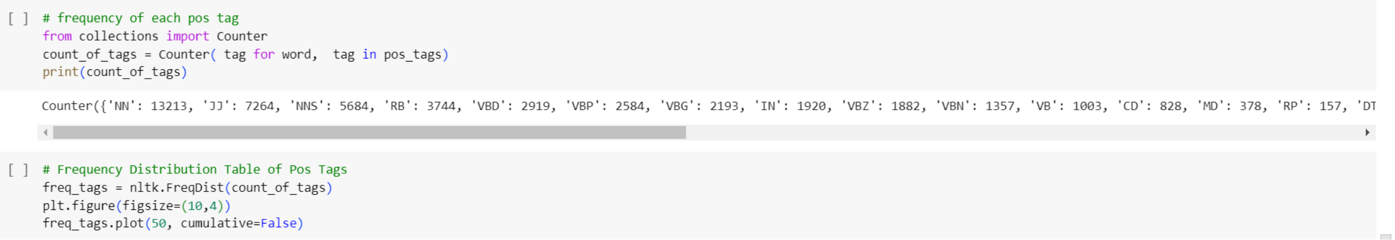
To create a frequency distribution of Part-of-Speech (POS) tags obtained from POS tagging using NLTK, we use Python's built-in library **collections.**
then creates a frequency distribution of the tags using the **Counter** class from the **collections** module.
we imported the FreqDist() function from the nltk.probability module. This function is useful for probability computations as it calculates the frequency at which each experiment's outcome happens.
To plot the frequency distribution of POS tags, we use Python's popular data visualization library, Matplotlib.

We imported the function figure() from the matplotlib.pyplot module, which is

used to build a figure object, in order to plot the frequency distribution graph.

The frequency distribution graph was then plotted:

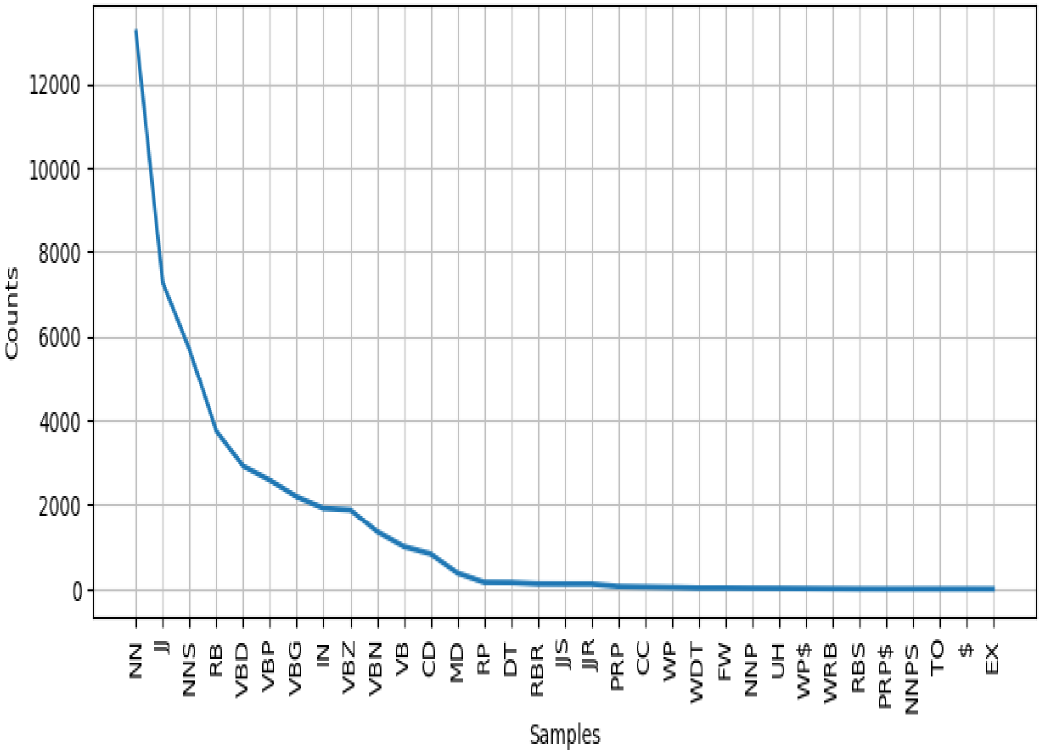
Code:



```
[ ] # frequency of each pos tag
    from collections import Counter
    count_of_tags = Counter( tag for word,  tag in pos_tags)
    print(count_of_tags)

    Counter({'NN': 13213, 'JJ': 7264, 'NNS': 5684, 'RB': 3744, 'VBD': 2919, 'VBP': 2584, 'VBG': 2193, 'IN': 1920, 'VBZ': 1882, 'VBN': 1357, 'VB': 1003, 'CD': 828, 'MD': 378, 'RP': 157, 'DT
```

```
[ ] # Frequency Distribution Table of Pos Tags
    freq_tags = nltk.FreqDist(count_of_tags)
    plt.figure(figsize=(10,4))
    freq_tags.plot(50, cumulative=False)
```

Image:


Counts
12000
10000
8000
6000
4000
2000
0
NN
JJ
NNS
RB
VBD
VBP
VBG
IN
VBZ
VBN
VB
CD
MD
RP
DT
RBR
JJS
JJR
PRP
CC
WP
WDT
FW
NNP
UH
WP$
WRB
RBS
PRP$
NNPS
TO
$
EX
Samples

Plotting Bi-gram Probability Model (with stopwords):

First of all we find out chapter C by using index function the we find bigrams using bi_grams = list(bigrams(chapter_c)): This line generates a list of bigrams from the chapter_c, which is assumed to be a list of words. It uses the bigrams function from the NLTK library to create these pairs.

bigrams_frequency = nltk.FreqDist(bi_grams): This line calculates the frequency distribution of the bigrams, counting how often each bigram occurs in the text.

cfd = nltk.ConditionalFreqDist(bi_grams): A Conditional Frequency Distribution is created using the bigrams. It associates each condition (word1) with a frequency distribution (FreqDist) of possible events (word2).

unique_words = list(set(word for bigram in bigrams_frequency for word in bigram)): This line extracts all unique words from the bigrams. These unique words will be used as both conditions (word1) and events (word2) in the conditional frequency distribution and the bigram matrix.

bigram_matrix = pd.DataFrame(0, columns=unique_words, index=unique_words, dtype=float): A square DataFrame (matrix) is created, where the rows and columns correspond to unique words. The matrix is initialized with 0.0 for all elements.

word_freq = nltk.FreqDist(chapter_c): The frequency distribution of individual words is calculated, counting how often each word appears in the text.

conditional_freq = cfd[word1][word2]: This line retrieves the conditional frequency of the bigram (word1, word2) from the conditional frequency distribution.

first_word_count = word_freq[word1]: The frequency of the first word (word1) is obtained.

if conditional_freq > 0:: A check is performed to ensure that the bigram has occurred at least once in the text.

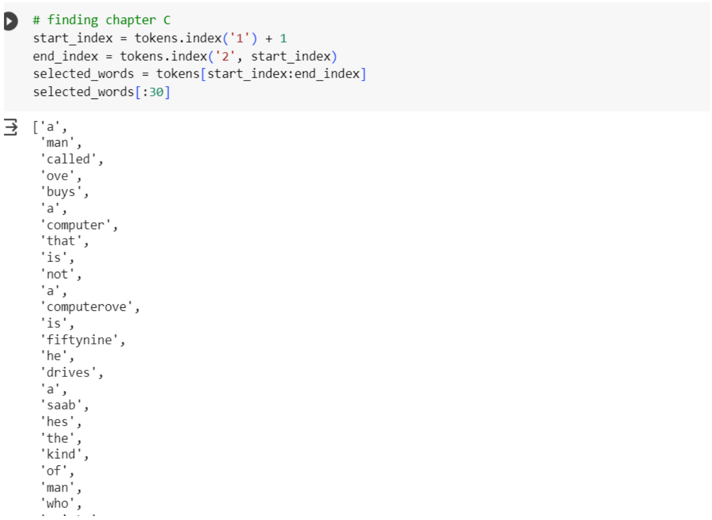
probability = float(conditional_freq) / first_word_count: The conditional probability of observing word2 given word1 is calculated by dividing the conditional frequency by the frequency of the first word.

bigram_matrix.at[word1, word2] = probability: The calculated probability is stored in the corresponding cell of the bigram matrix.

bigram_matrix.fillna(0.0, inplace=True): Any potential NaN (Not-a-Number) values in the bigram matrix are filled with 0.0.

The result is a bigram probability matrix in the form of a Pandas DataFrame. Each cell in the matrix represents the probability of observing a specific word (column) given the preceding word (row).

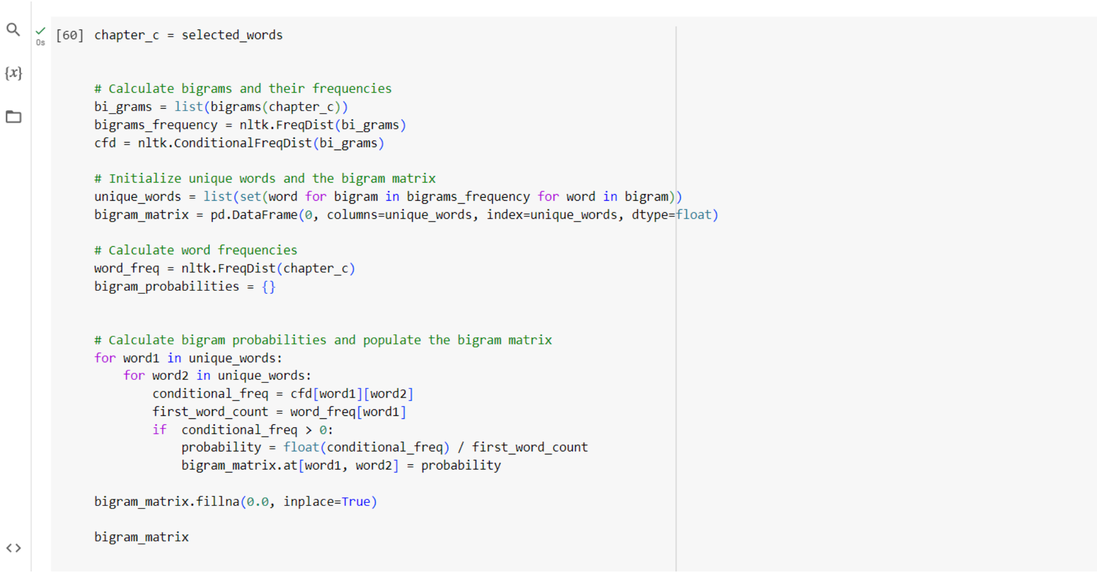
Code to find Chapter C:



```
# finding chapter C
start_index = tokens.index('1') + 1
end_index = tokens.index('2', start_index)
selected_words = tokens[start_index:end_index]
selected_words[:30]
```

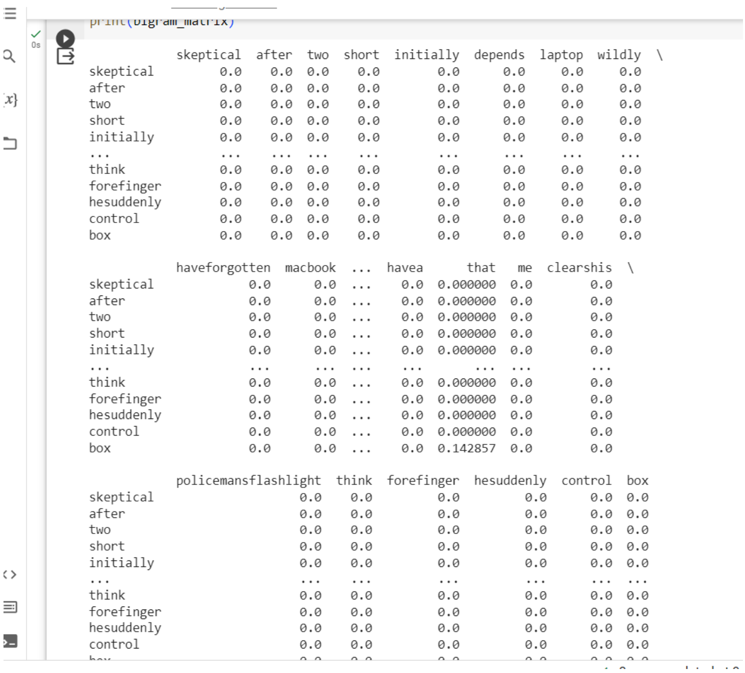
```
['a',
 'man',
 'called',
 'ove',
 'buys',
 'a',
 'computer',
 'that',
 'is',
 'not',
 'a',
 'computerove',
 'is',
 'fiftynine',
 'he',
 'drives',
 'a',
 'saab',
 'hes',
 'the',
 'kind',
 'of',
 'man',
 'who',
```

Code for Creating Matrix:



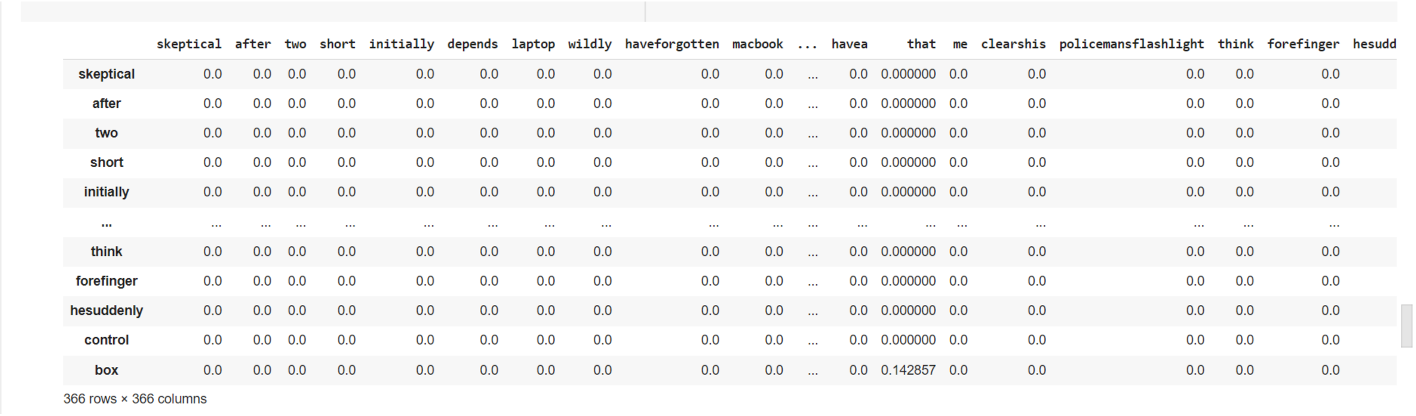
```
[60] chapter_c = selected_words


     # Calculate bigrams and their frequencies
     bi_grams = list(bigrams(chapter_c))
     bigrams_frequency = nltk.FreqDist(bi_grams)
     cfd = nltk.ConditionalFreqDist(bi_grams)

     # Initialize unique words and the bigram matrix
     unique_words = list(set(word for bigram in bigrams_frequency for word in bigram))
     bigram_matrix = pd.DataFrame(0, columns=unique_words, index=unique_words, dtype=float)

     # Calculate word frequencies
     word_freq = nltk.FreqDist(chapter_c)
     bigram_probabilities = {}


     # Calculate bigram probabilities and populate the bigram matrix
     for word1 in unique_words:
         for word2 in unique_words:
             conditional_freq = cfd[word1][word2]
             first_word_count = word_freq[word1]
             if  conditional_freq > 0:
                 probability = float(conditional_freq) / first_word_count
                 bigram_matrix.at[word1, word2] = probability

     bigram_matrix.fillna(0.0, inplace=True)

     bigram_matrix
```

Image:



```
print(bigram_matrix)
```

```
            skeptical  after  two  short  initially  depends  laptop  wildly  \
skeptical         0.0    0.0  0.0    0.0        0.0      0.0     0.0     0.0
after             0.0    0.0  0.0    0.0        0.0      0.0     0.0     0.0
two               0.0    0.0  0.0    0.0        0.0      0.0     0.0     0.0
short             0.0    0.0  0.0    0.0        0.0      0.0     0.0     0.0
initially         0.0    0.0  0.0    0.0        0.0      0.0     0.0     0.0
...               ...    ...  ...    ...        ...      ...     ...     ...
think             0.0    0.0  0.0    0.0        0.0      0.0     0.0     0.0
forefinger        0.0    0.0  0.0    0.0        0.0      0.0     0.0     0.0
hesuddenly        0.0    0.0  0.0    0.0        0.0      0.0     0.0     0.0
control           0.0    0.0  0.0    0.0        0.0      0.0     0.0     0.0
box               0.0    0.0  0.0    0.0        0.0      0.0     0.0     0.0

            haveforgotten  macbook  ...  havea      that   me  clearshis  \
skeptical             0.0      0.0  ...    0.0  0.000000  0.0        0.0
after                 0.0      0.0  ...    0.0  0.000000  0.0        0.0
two                   0.0      0.0  ...    0.0  0.000000  0.0        0.0
short                 0.0      0.0  ...    0.0  0.000000  0.0        0.0
initially             0.0      0.0  ...    0.0  0.000000  0.0        0.0
...                   ...      ...  ...    ...       ...  ...        ...
think                 0.0      0.0  ...    0.0  0.000000  0.0        0.0
forefinger            0.0      0.0  ...    0.0  0.000000  0.0        0.0
hesuddenly            0.0      0.0  ...    0.0  0.000000  0.0        0.0
control               0.0      0.0  ...    0.0  0.000000  0.0        0.0
box                   0.0      0.0  ...    0.0  0.142857  0.0        0.0

            policemansflashlight  think  forefinger  hesuddenly  control  box
skeptical                    0.0    0.0         0.0         0.0      0.0  0.0
after                        0.0    0.0         0.0         0.0      0.0  0.0
two                          0.0    0.0         0.0         0.0      0.0  0.0
short                        0.0    0.0         0.0         0.0      0.0  0.0
initially                    0.0    0.0         0.0         0.0      0.0  0.0
...                          ...    ...         ...         ...      ...  ...
think                        0.0    0.0         0.0         0.0      0.0  0.0
forefinger                   0.0    0.0         0.0         0.0      0.0  0.0
hesuddenly                   0.0    0.0         0.0         0.0      0.0  0.0
control                      0.0    0.0         0.0         0.0      0.0  0.0
box                          0.0    0.0         0.0         0.0      0.0  0.0
```



	skeptical	after	two	short	initially	depends	laptop	wildly	haveforgotten	macbook	...	havea	that	me	clearshis	policemansflashlight	think	forefinger	hesudd
skeptical	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
after	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
two	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
short	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
initially	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
think	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
forefinger	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
hesuddenly	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
control	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	
box	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.142857	0.0	0.0	0.0	0.0	0.0	

366 rows × 366 columns

Shannon Game for Bi-gram Model:

For playing a word-guessing game based on a provided starting word and bigram probabilities. Here's a breakdown of how this code works:

The play_shannons_game function takes two arguments: existing_string and bigram_probabilities.

It initializes an empty string s to store the current guessed sentence and then displays a series of introductory messages to welcome the player and provide instructions for the game.

It prompts the player to enter a starting word by using the input function and stores the player's input in the previous_word variable. The starting word is added to the sentence s.

The game enters a loop using while True, which means it will keep running until a return statement is executed or the player chooses to exit the game.

Inside the loop, the make_guess function is called with the previous_word and bigram_probabilities as arguments. This function should return a list of guesses for the next word based on the probabilities of word transitions.

For each guess in the list of guesses, the game asks the player whether the guess is correct. The player responds with "yes" or "no" using the input function.

If the player responds "yes," it means the guess was correct, so the previous_word is updated to the guessed word, and the guessed word is added to the sentence s. The current sentence is then displayed.

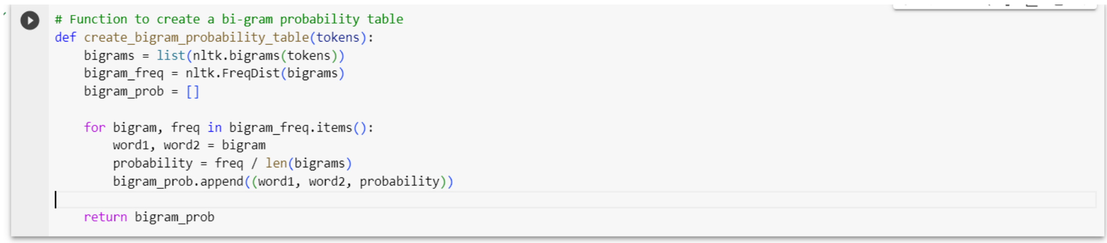
If the player responds "no," the loop continues to the next guess in the list.

If the player responds "exit," the game ends with a "Thanks for playing!" message, and the return statement exits the function.

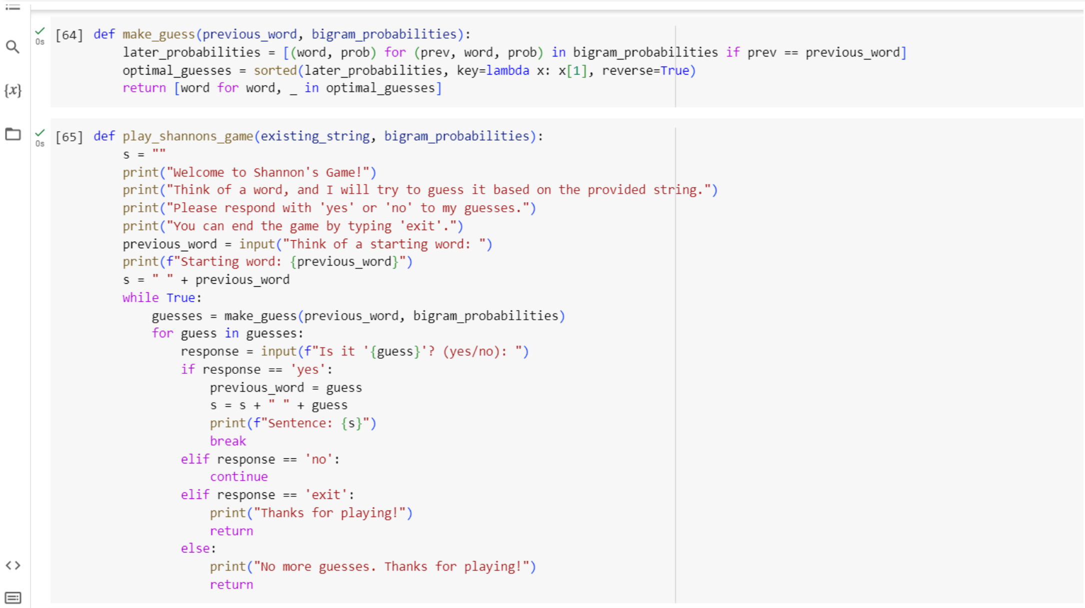
If the player responds with anything else, the game also ends with a message saying "No more guesses. Thanks for playing!"

Then we call a function play_shannons_game for chapter_other_than_c to generate a sentence.

Code:

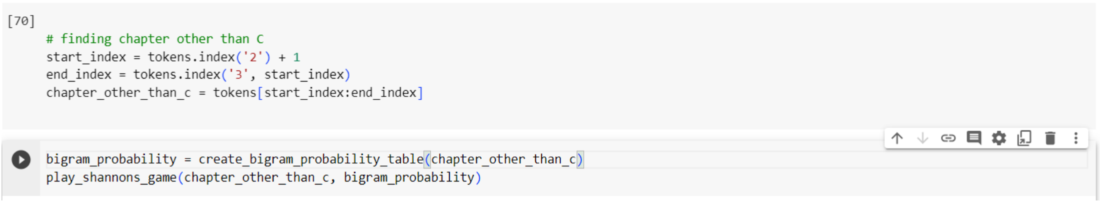


```
# Function to create a bi-gram probability table
def create_bigram_probability_table(tokens):
    bigrams = list(nltk.bigrams(tokens))
    bigram_freq = nltk.FreqDist(bigrams)
    bigram_prob = []

    for bigram, freq in bigram_freq.items():
        word1, word2 = bigram
        probability = freq / len(bigrams)
        bigram_prob.append((word1, word2, probability))

    return bigram_prob
```



```
[64] def make_guess(previous_word, bigram_probabilities):
         later_probabilities = [(word, prob) for (prev, word, prob) in bigram_probabilities if prev == previous_word]
         optimal_guesses = sorted(later_probabilities, key=lambda x: x[1], reverse=True)
         return [word for word, _ in optimal_guesses]
```

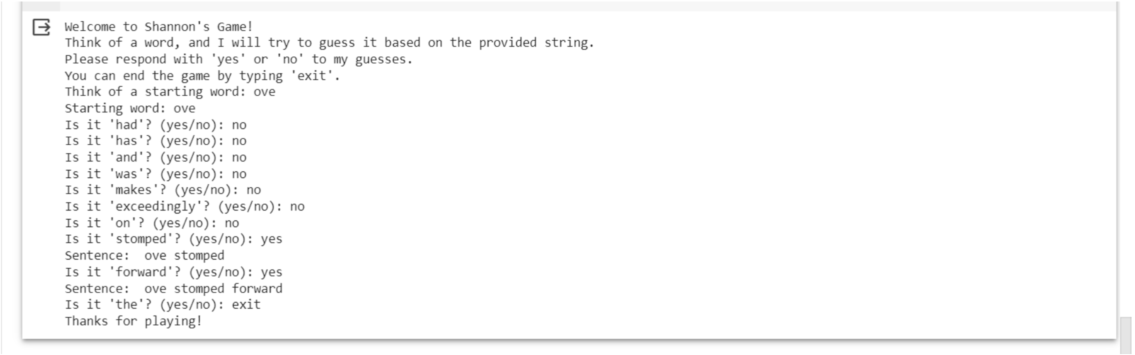
```
[65] def play_shannons_game(existing_string, bigram_probabilities):
         s = ""
         print("Welcome to Shannon's Game!")
         print("Think of a word, and I will try to guess it based on the provided string.")
         print("Please respond with 'yes' or 'no' to my guesses.")
         print("You can end the game by typing 'exit'.")
         previous_word = input("Think of a starting word: ")
         print(f"Starting word: {previous_word}")
         s = " " + previous_word
         while True:
             guesses = make_guess(previous_word, bigram_probabilities)
             for guess in guesses:
                 response = input(f"Is it '{guess}'? (yes/no): ")
                 if response == 'yes':
                     previous_word = guess
                     s = s + " " + guess
                     print(f"Sentence: {s}")
                     break
                 elif response == 'no':
                     continue
                 elif response == 'exit':
                     print("Thanks for playing!")
                     return
             else:
                 print("No more guesses. Thanks for playing!")
                 return
```



```
[70]
    # finding chapter other than C
    start_index = tokens.index('2') + 1
    end_index = tokens.index('3', start_index)
    chapter_other_than_c = tokens[start_index:end_index]
```

```
bigram_probability = create_bigram_probability_table(chapter_other_than_c)
play_shannons_game(chapter_other_than_c, bigram_probability)
```

Image:



```
Welcome to Shannon's Game!
Think of a word, and I will try to guess it based on the provided string.
Please respond with 'yes' or 'no' to my guesses.
You can end the game by typing 'exit'.
Think of a starting word: ove
Starting word: ove
Is it 'had'? (yes/no): no
Is it 'has'? (yes/no): no
Is it 'and'? (yes/no): no
Is it 'was'? (yes/no): no
Is it 'makes'? (yes/no): no
Is it 'exceedingly'? (yes/no): no
Is it 'on'? (yes/no): no
Is it 'stomped'? (yes/no): yes
Sentence:  ove stomped
Is it 'forward'? (yes/no): yes
Sentence:  ove stomped forward
Is it 'the'? (yes/no): exit
Thanks for playing!
```

From above image we can conclude that by using bi-gram probability model we can generate sentences with good accuracy.

5. CONCLUSION

Working on this project taught us a lot about cooperation and subject comprehension. On the surface, we were all familiar with all of the text analytics methods, but this project has increased our understanding of text processing using Python's natural language processing features.

We came to the conclusion that by utilizing built-in toolkits and Python libraries, this project illustrates the basic ideas of text preparation, POS tagging, tokenization, etc. The matplotlib-generated word clouds, bar charts, and graphs helped us better understand the findings and were helpful for the data's visual representation. All things considered, this was an excellent learning experience that motivated us to study more about NLP.

REFERENCES

- https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/

- https://www.mygreatlearning.com/blog/nltk-tutorial-with-python/#3