NLP Project Report

Under the Guidance of Dr. Sakthi Balan

Team Name: **LexicallyFlexible**

GitHub Link Repo:

https://github.com/RishabhJ01/nlp-project-2021

Team Members:

Rishabh Jain (19UCS176)

Tapomay Singh Khatri (19UCS064)

Vishnu Bhat (19UCS058)

INDEX

System Requirements:

1. Python 3.x
2. Libraries: NLTK, matplotlib, wordcloud, re, NLTK.corpus, json, ssl, urllib
3. Jupyter Notebook
4. OS: Windows / Linux / Mac OS

Books Used:

Book 1 :
Title: Pride and Prejudice
Author: Jane Austen
Link: https://www.gutenberg.org/files/1342/1342-0.txt

Book 2 :
Title: Gulliver's Travel
Author: Jonathan Swift
Link: https://www.gutenberg.org/files/829/829-0.txt

Project Round-1

Problem statement - 1 :

Import the text, let's call it data1 and data2.

We have imported Book-1 as data1 and Book-2 as data2. We have used **"*json*"** and **"*urllib*"** libraries to scrap the texts from the given links. Further we decoded it to convert the text to string.

Problem statement - 2 :

Perform simple text pre-processing.

1. Changing the text to lowercase.
2. Removing running sections.
3. Removing special characters.
4. Removing Chapter Headlines.
5. Removing digits.
6. Removing implicit next line characters - '\n'.

Problem statement - 3 :

Tokenize the texts.

Tokenization is basically dividing the text into smaller units called tokens. They can be either words, characters, or subwords. It helps in understanding the context and interpreting the meaning behind the sequence of tokens. For example “miserable animals presume*”,* will be tokenized into - [“*miserable*” , “*animals*” , “*presume*”].

1. ‘*data1*’ and ‘*data2*’ are tokenized using the ***nltk.word_tokenize*** function.
2. **‘*token1*’** contains tokens of ‘*data1*’ i.e. Book-1.
3. ‘***token2*’** contains tokens of ‘*data2*’ i.e. Book-2.

Inference:

So now the imported texts **‘*data1*’** and **‘*data2*’** will be tokenized as explained above using ***nltk.word_tokenize*** function. Tokens created are now stored in ‘*token1*’ for ‘*data1*’ and ‘*token2*’ for ‘*data2*’ .

Problem statement - 4 :

Analyze the frequency distribution of tokens separately.

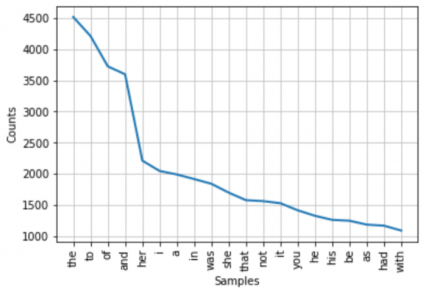
Formed the Frequency Distribution using FreqDist function of NLTK library and then plotted it.

Token1:

In [9]:
```
# Frequency Distribution for token 1
fdist1 = FreqDist(token1)
print(fdist1.most_common(20))
```

```
[('the', 4511), ('to', 4207), ('of', 3722), ('and', 3596), ('her', 2215), ('i', 2050), ('a', 1994), ('in', 1921), ('was', 1845), ('she', 1704), ('that', 1583), ('not', 1568), ('it', 1534), ('you', 1421), ('he', 1333), ('his', 1268), ('be', 1254), ('as', 1192), ('had', 1175), ('with', 1097)]
```

In [12]:
```
fdist1.plot(20)
```

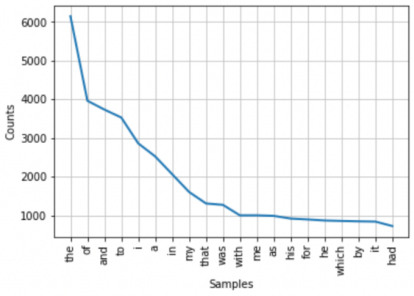


Token2:

In [10]:
```
# frequency Distribution for token 2
fdist2 = FreqDist(token2)
print(fdist2.most_common(20))
```

```
[('the', 6151), ('of', 3965), ('and', 3738), ('to', 3532), ('i', 2864), ('a', 2527), ('in', 2068), ('my', 1608), ('that', 1310), ('was', 1274), ('with', 1003), ('me', 1003), ('as', 989), ('his', 920), ('for', 897), ('he', 870), ('which', 858), ('by', 848), ('it', 841), ('had', 726)]
```

In [13]:
```
fdist2.plot(20)
```



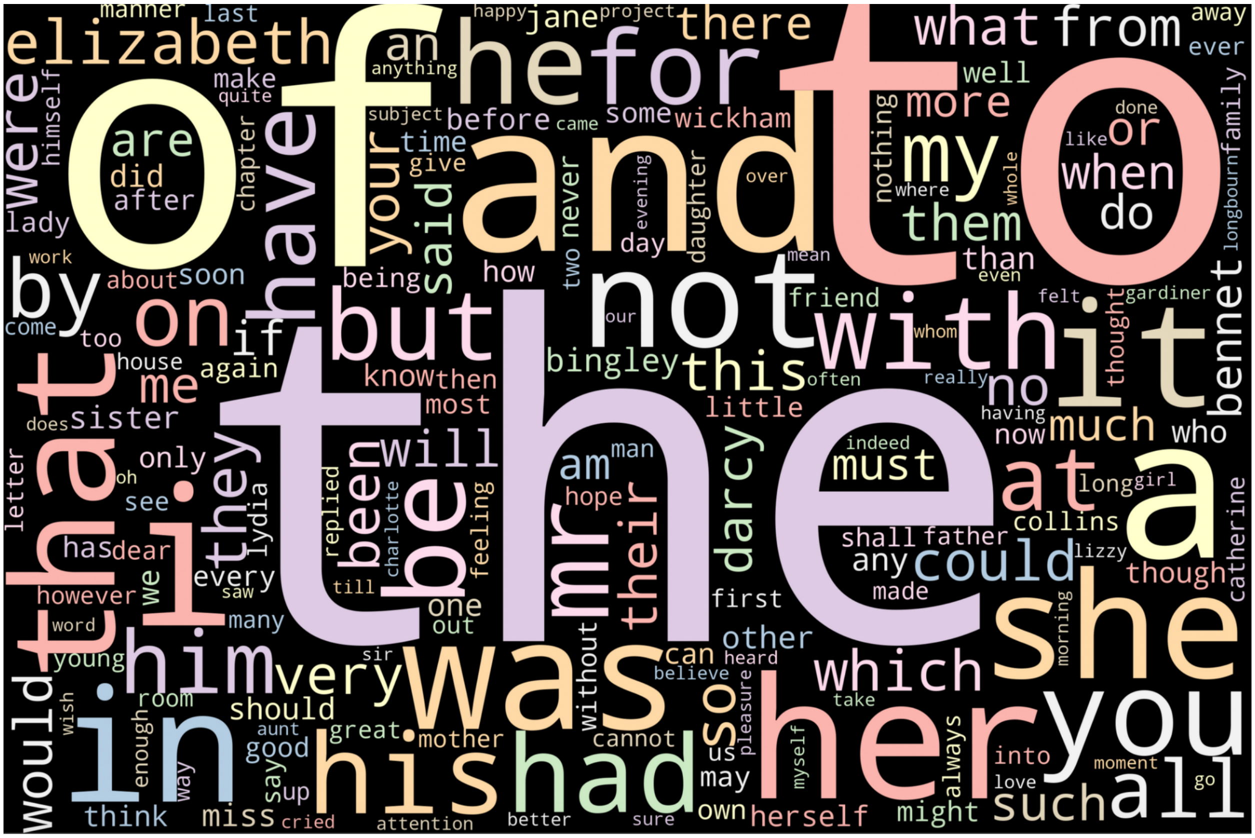
Problem statement - 5 :

Create a wordcloud before and after removing the stop words.

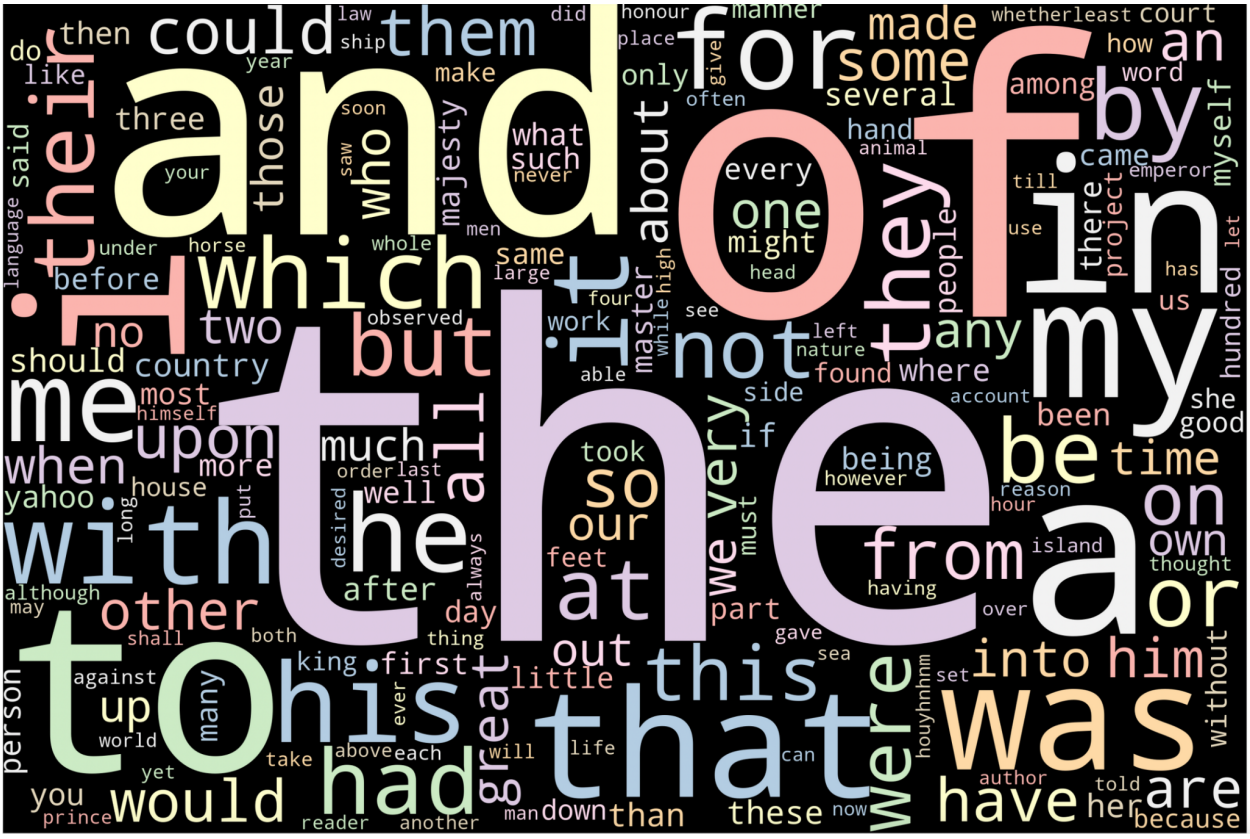
After analyzing the frequency distribution of the tokens, wordclouds are created using these tokens. A wordcloud is a visual representation of text data wherein each word is pictured with its frequency, i.e. higher the frequency, larger the size of the word. **Stopwords** - These are most commonly occurring words because of the grammar rules which hinder the visualization and useful computations.

Before removing stopwords :

Book-1 :


manner
last
happy
jane
project
there
what
from
away
elizabeth
of
an
he
for
to
ever
anything
well
make
quite
himself
were
subject
before
came
some
wickham
nothing
more
family
done
are
time
like
or
chapter
your
give
my
did
after
have
said
never
evening
daughter
over
where
whole
when
lady
and
do
longbourn
them
two
day
mean
than
work
about
soon
being
how
even
by
not
friend
felt
gardiner
bennet
but
our
with
it
whom
come
too
if
thought
house
again
bingley
this
often
really
me
know
then
no
does
sister
most
little
having
that
the
will
much
now
who
only
darcy
indeed
man
must
oh
they
am
at
a
letter
see
lydia
long
girl
hope
catherine
replied
been
be
collins
charlotte
feeling
mr
their
i
shall
father
has
dear
lizzy
any
could
every
though
we
however
saw
till
first
made
word
many
one
out
other
she
young
him
sir
can
morning
heard
very
was
believe
which
without
wish
room
so
take
should
pleasure
aunt
great
in
would
enough
mother
cannot
you
us
myself
good
into
always
her
say
way
moment
go
his
had
may
love
all
up
own
think
miss
such
cried
attention
better
sure
herself
might

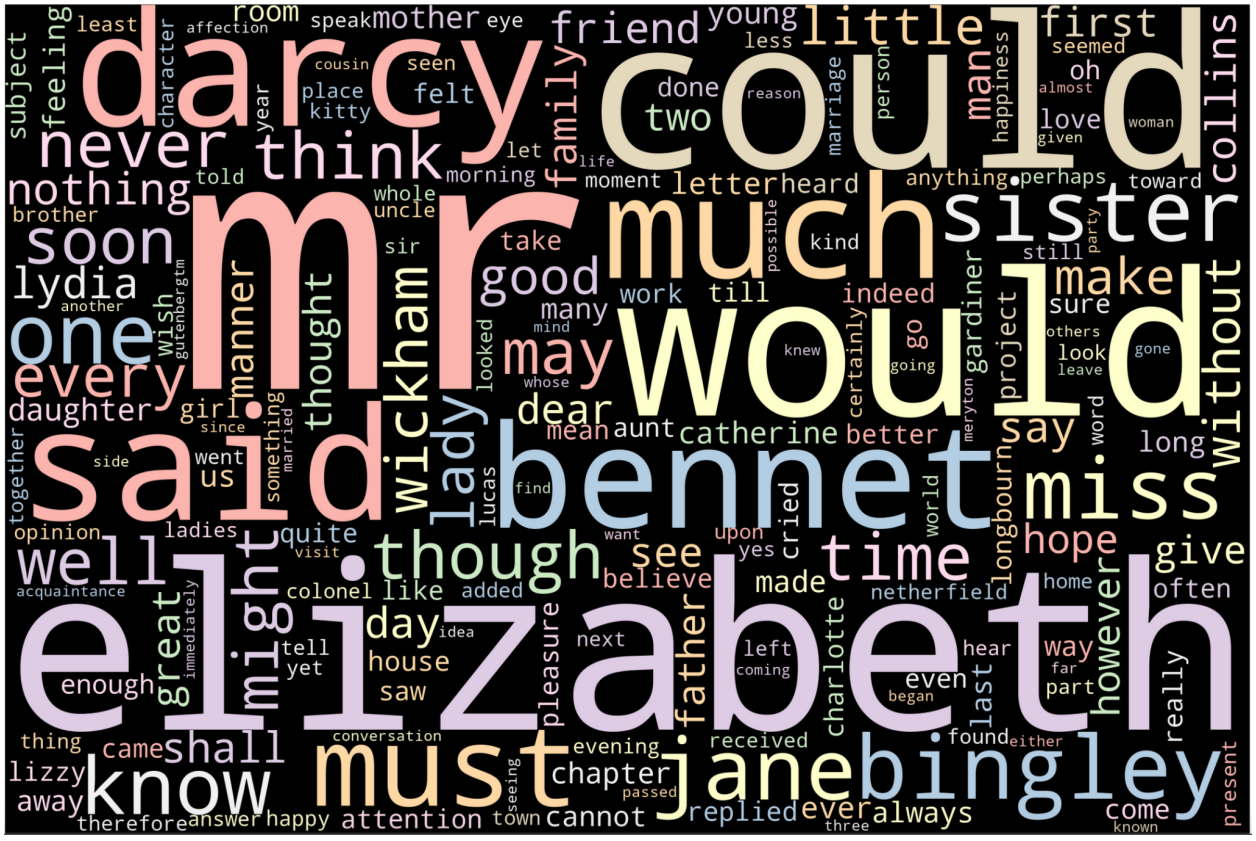
Book-2 :



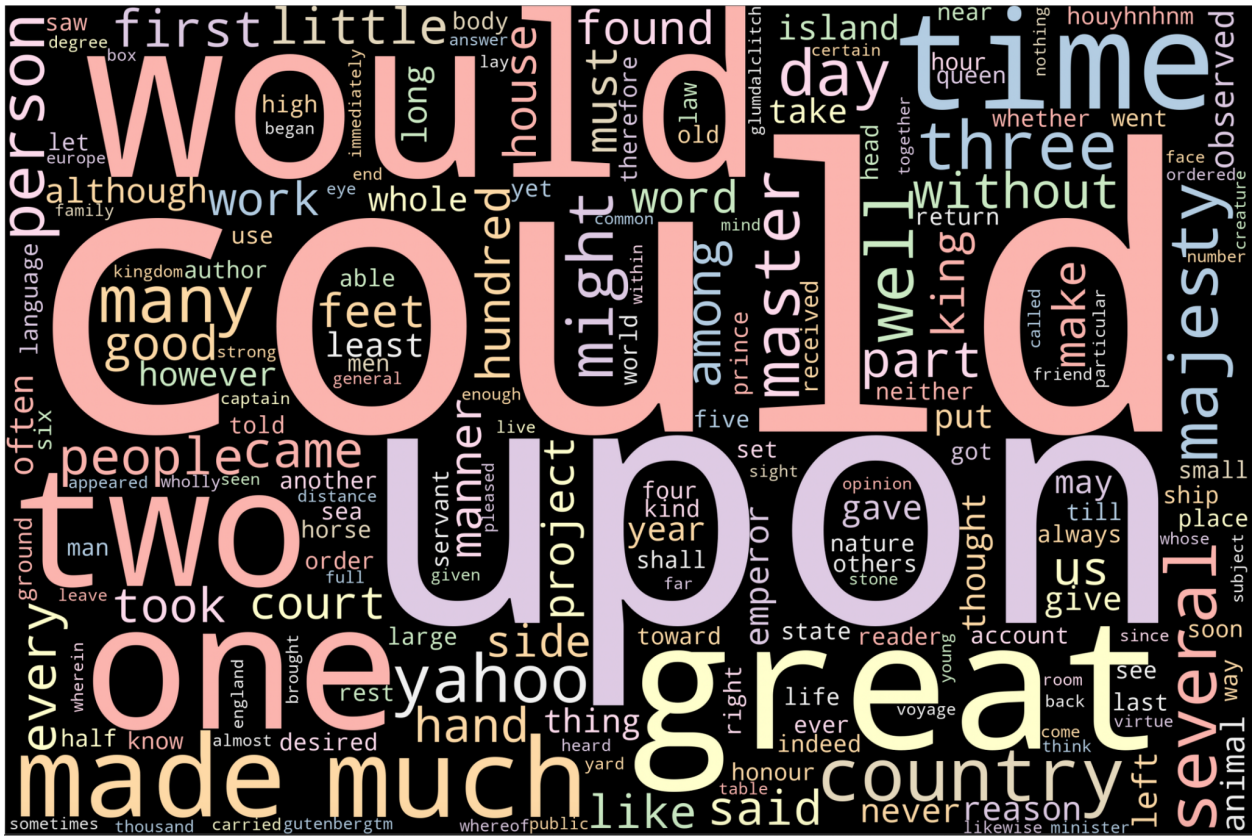
After removing stopwords :

NLTK library provides a list of the stopwords of the english language. So with the help of this list, we omitted out the stopwords from our tokens.

Book-1 :



Book-2 :



Inference :

As it can be clearly seen in above pictures of wordcloud before removing the stop words,the larger words are*‘and’*,*‘of’*,*‘the’*, *‘to’*, etc. These are the stopwords which occur very frequently (size of a word in a word cloud is determined according to its frequency) and these words are not essential for our analysis. So after removing the stopwords, we can ensure that other words which are in context with our books show up in the word cloud.

Problem statement - 6 :

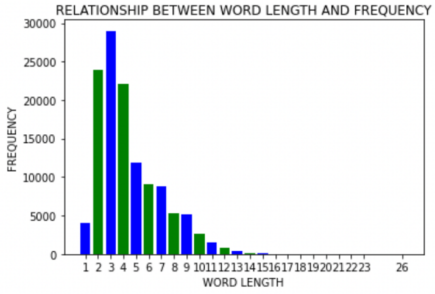
Evaluate the relationship between the word length and frequency after removing stopwords.

Firstly we created an empty list and stored the number of words for each length and then plotted it using Frequency Distribution and matplotlib.

Before Removing Stopwords:-

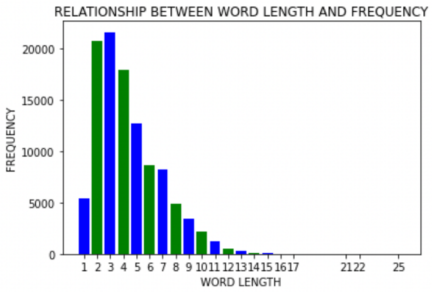
Book-1:

```
In [14]: plot_relationship(fdist1)
```



Book-2:

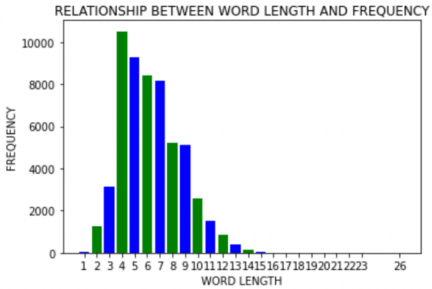
In [15]:
```
plot_relationship(fdist2)
```



After Removing Stopwords:-

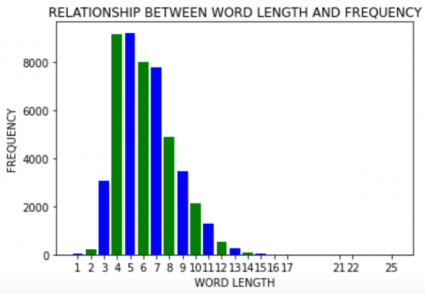
Book-1:

In [21]:
```
# For Data 1
plot_relationship(FreqDist(rem1))
```



Book-2:

In [22]:
```
# For Data 2
plot_relationship(FreqDist(rem2))
```



Inference :

Frequency of words of smaller length decreased significantly after removing the stopwords as generally these stopwords are of smaller length, e.g. - *'to', 'the',* etc.

Problem statement - 7 :

Do PoS Tagging using any of the four tagset studied in the class and get the distribution of various tags.

PoS stands for parts of speech. PoS tagging means to tag each word with its part of speech like noun, verb, adverb, adjective, etc. Though there are some third party libraries which can be used to do PoS tagging in just one line of code, there is a very fancy algorithm which is doing all the work behind this. Basically it uses *HMM (hidden markov model)* to tag each word. The crux of the algorithm is to find the most probable tagset for the given set of words among all the possible tagsets. It uses the *Viterbi* algorithm which is based on the dynamic programming paradigm to efficiently find the most probable tagset. There are many available tagsets which can be used for this purpose. For this project, we have used the *Treebank* tagset.*Tree bank* tagset includes *36* PoS tags like Coordinating Conjunction (CC), Determiner (DT) etc.

Note :- Below plots are for the 20 most occuring tags.

Book-1 :-

PoS tagging analysis

In [30]:
```
FreqDist(tags1)
```

Out[30]:
```
FreqDist({'NN': 17754, 'JJ': 10157, 'RB': 5250, 'VBD': 4328, 'NNS': 4103, 'VB': 2420, 'VBP': 2234, 'VBG': 2176, 'VBN': 2147, 'MD': 1932,
...})
```

Book-2 :-

PoS tagging analysis

In [31]:
```
FreqDist(tags2)
```

Out[31]:
```
FreqDist({'NN': 14486, 'JJ': 9178, 'NNS': 6175, 'VBD': 4104, 'RB': 3455, 'VBN': 1922, 'VBP': 1899, 'VBG': 1724, 'VB': 1596, 'IN': 1390,
...})
```