Jupyter Notebook for Week 4

NLP with Python Chapter 1, sections 4-8; begin Chapter 2

Student Name: Muhammad Rakibul Islam

- Date:
- Assignment Due:
- Instructor: Lisa Rhody
- Methods of Text Analysis, Spring 2023

```
import nltk
nltk.download("book")
```

```
[nltk_data] Downloading collection 'book'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/abc.zip.
[nltk_data]    | Downloading package brown to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown.zip.
[nltk_data]    | Downloading package chat80 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/chat80.zip.
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cmudict.zip.
[nltk_data]    | Downloading package conll2000 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2000.zip.
[nltk_data]    | Downloading package conll2002 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2002.zip.
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/dependency_treebank.zip.
[nltk_data]    | Downloading package genesis to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/genesis.zip.
[nltk_data]    | Downloading package gutenberg to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/gutenberg.zip.
[nltk_data]    | Downloading package ieer to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/ieer.zip.
[nltk_data]    | Downloading package inaugural to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/inaugural.zip.
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/movie_reviews.zip.
[nltk_data]    | Downloading package nps_chat to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/nps_chat.zip.
[nltk_data]    | Downloading package names to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/names.zip.
[nltk_data]    | Downloading package ppattach to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/ppattach.zip.
[nltk_data]    | Downloading package reuters to /root/nltk_data...
[nltk_data]    | Downloading package senseval to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/senseval.zip.
[nltk_data]    | Downloading package state_union to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/state_union.zip.
[nltk_data]    | Downloading package stopwords to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/stopwords.zip.
[nltk_data]    | Downloading package swadesh to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/swadesh.zip.
[nltk_data]    | Downloading package timit to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/timit.zip.
[nltk_data]    | Downloading package treebank to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/treebank.zip.
[nltk_data]    | Downloading package toolbox to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/toolbox.zip.
[nltk_data]    | Downloading package udhr to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/udhr.zip.
[nltk_data]    | Downloading package udhr2 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/udhr2.zip.
[nltk_data]    | Downloading package unicode_samples to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/unicode_samples.zip.
[nltk_data]    | Downloading package webtext to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/webtext.zip.
```

```
from nltk.book import *
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

Back to Python: Making Decisions and Taking Control

So far, our little programs have had some interesting qualities: the ability to work with language, and the potential to save human effort through automation. A key feature of programming is the ability of machines to make decisions on our behalf, executing instructions when certain conditions are met, or repeatedly looping through text data until some condition is satisfied. This feature is known as control, and is the focus of this section.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

4.1 Conditionals

Python supports a wide range of operatores such as < and >=, for testing the relationship bewteen values. The full set of these *relational operators* is shown in Table 4.1.

Table 4.1:

Numerical Comparison Operators

Operator	Relationship
<	less than
<=	less than or equal to
==	equal to (note this is two "=" signs, not one)
!=	not equal to
>	greater than
>=	greater than or equal to

figure4_1.png

We can use these to select different words from a sentence of news text. Here are some examples -- only the operator is changed from one line ot the next. They all use `sent7`, the first sentence from `text7` (*Wall Street Journal*). As before, if you get an error saying that `sent7` is undefined, you need to first type `from nltk.book import *`

```
sent7
```

```
['Pierre',
 'Vinken',
 ',',
 '61',
 'years',
 'old',
 ',',
 'will',
 'join',
 'the',
 'board',
 'as',
 'a',
 'nonexecutive',
```

```
 'director',
 'Nov.',
 '29',
 '.']
```

```
[w for w in sent7 if len(w) < 4]
```

```
[',', '61', 'old', ',', 'the', 'as', 'a', '29', '.']
```

```
[w for w in sent7 if len(w) <= 4]
```

```
[',', '61', 'old', ',', 'will', 'join', 'the', 'as', 'a', 'Nov.', '29', '.']
```

```
[w for w in sent7 if len(w) == 4]
```

```
['will', 'join', 'Nov.']
```

```
[w for w in sent7 if len(w) != 4]
```

```
['Pierre',
 'Vinken',
 ',',
 '61',
 'years',
 'old',
 ',',
 'the',
 'board',
 'as',
 'a',
 'nonexecutive',
 'director',
 '29',
 '.']
```

There is a common pattern to all of these examples: `[w for w in text if _condition_ ]`, where *condition* is a Python "test" that yields either true or false. In the cases shown in the previous code example, the condition is always a numerical comparison. However, we can also test various properties of words, using the functions listed in 4.2.

***Table 4.2**:*

Some Word Comparison Operators

Function	Meaning
`s.startswith(t)`	test if `s` starts with `t`
`s.endswith(t)`	test if `s` ends with `t`
`t in s`	test if `t` is a substring of `s`
`s.islower()`	test if `s` contains cased characters and all are lowercase
`s.isupper()`	test if `s` contains cased characters and all are uppercase
`s.isalpha()`	test if `s` is non-empty and all characters in `s` are alphabetic
`s.isalnum()`	test if `s` is non-empty and all characters in `s` are alphanumeric
`s.isdigit()`	test if `s` is non-empty and all characters in `s` are digits
`s.istitle()`	test if `s` contains cased characters and is titlecased (i.e. all words in `s` have initial capitals)

Here are some examples of these operators being used to select words from our texts: words ending with *-ableness*; words containing *gnt*; words having an initial capital; and words consisteng entirely of digits.

```
sorted(w for w in set(text1) if w.endswith('ableness'))
```

```
['comfortableness',
 'honourableness',
 'immutableness',
 'indispensableness',
 'indomitableness',
 'intolerableness',
 'palpableness',
 'reasonableness',
 'uncomfortableness']
```

```
sorted(term for term in set(text4) if 'gnt' in term)
```

```
['Sovereignty', 'sovereignties', 'sovereignty']
```

```
sorted(item for item in set(text6) if item.istitle())
```

```
['A',
 'Aaaaaaaaah',
 'Aaaaaaaah',
 'Aaaaaah',
 'Aaaah',
 'Aaaaugh',
 'Aaagh',
 'Aaah',
 'Aaauggh',
 'Aaaugh',
 'Aaauugh',
 'Aagh',
 'Aah',
 'Aauuggghhh',
 'Aauuugh',
 'Aauuuuugh',
 'Aauuuves',
 'Action',
 'Actually',
 'African',
 'Ages',
 'Aggh',
 'Agh',
 'Ah',
 'Ahh',
 'Alice',
 'All',
 'Allo',
 'Almighty',
 'Alright',
 'Am',
 'Amen',
 'An',
 'Anarcho',
 'And',
 'Angnor',
 'Anthrax',
 'Antioch',
 'Anybody',
 'Anyway',
 'Apples',
 'Aramaic',
 'Are',
 'Arimathea',
 'Armaments',
 'Arthur',
 'As',
 'Ask',
 'Assyria',
 'At',
 'Attila',
 'Augh',
 'Autumn',
 'Auuuuuuuugh',
 'Away',
 'Ay',
 'Ayy',
 'B',
```

```
sorted(item for item in set(sent7) if item.isdigit())
```

```
['29', '61']
```

We can also create more complex conditions if **c** is a condition then `not c` is also a condition. If we have two conditions, then we can combine them to form a new condition using conjunction and disjunction.

4.2 Operating on Every Element

Previously, we saw some examples of counting items other than words. Let's take a closer look at the notation we used:

```
[len(w) for w in text1]
```

```
[1,
 4,
 4,
 2,
 6,
 8,
 4,
 1,
 9,
 1,
 1,
 8,
 2,
 1,
 4,
 11,
 5,
 2,
 1,
 7,
 6,
 1,
 3,
 4,
 5,
 2,
 10,
 2,
 4,
 1,
 5,
 1,
 4,
 1,
 3,
 5,
 1,
 1,
 3,
 3,
 3,
 1,
 2,
 3,
 4,
 7,
 3,
 3,
 8,
 3,
 8,
 1,
 4,
 1,
 5,
 12,
 1,
 9,
```

```
[w.upper() for w in text1]
```

```
['[',
 'MOBY',
 'DICK',
 'BY',
 'HERMAN',
 'MELVILLE',
 '1851',
 ']',
 'ETYMOLOGY',
 '.',
 '(',
 'SUPPLIED',
 'BY',
 'A',
 'LATE',
 'CONSUMPTIVE',
 'USHER',
 'TO',
 'A',
 'GRAMMAR',
 'SCHOOL',
 ')',
```

```
 'THE',
 'PALE',
 'USHER',
 '--',
 'THREADBARE',
 'IN',
 'COAT',
 ',',
 'HEART',
 ',',
 'BODY',
 ',',
 'AND',
 'BRAIN',
 ';',
 'I',
 'SEE',
 'HIM',
 'NOW',
 '.',
 'HE',
 'WAS',
 'EVER',
 'DUSTING',
 'HIS',
 'OLD',
 'LEXICONS',
 'AND',
 'GRAMMARS',
 ',',
 'WITH',
 'A',
 'QUEER',
 'HANDKERCHIEF',
 ',',
 'MOCKINGLY'
```

These expressions have the form [f(w) for ...] or [w.f() for ...], where f is a function that operates on a word to compute its length, or to convert it to uppercase. For now, you don't need to understand the difference between the notations f(w) and w.f(). Instead, simply learn this Python idiom which performs the same operation on every element of a list. In the preceding examples, it goes through each word in text1, assigning each one in turn to the variable w and performing the specified operation on the variable.

Note: The notation just described is called a "list comprehension." This is our first example of a Python idiom, a fixed notation that we use habitually without bothering to analyze each time. Mastering such idioms is an important part of becoming a fluent Python programmer.

Let's return to the question of vocabulary size, and apply the same idiom here:

```
len(text1)
```

```
260819
```

```
len(set(text1))
```

```
19317
```

```
len(set(word.lower() for word in text1))
```

```
17231
```

Now that we are not double-counting words like This and this, which differ only in capitalization, we've wiped 2,000 off the vocabulary count! We can go a step further and eliminate numbers and punctuation from the vocabulary count by filtering out any non-alphabetic items:

```
len(set(word.lower() for word in text1 if word.isalpha()))
```

```
16948
```

This example is slightly complicated: it lowercases all the purely alphabetic items. Perhaps it would have been simpler just to count the lowercase-only items, but this gives the wrong answer (why?).

Don't worry if you don't feel confident with list comprehensions yet, since you'll see many more examples along with explanations in the following chapters.

4.3 Nested Code Blocks

Most programming languages permit us to execute a block of code when a **conditional expression**, or if statement, is satisfied. We already saw examples of conditional tests in code like `[w for w in sent7 if len(w) < 4]`. In the following program, we have created a variable called word containing the string value 'cat'. The if statement checks whether the test `len(word) < 5` is true. It is, so the body of the if statement is invoked and the print statement is executed, displaying a message to the user. Remember to indent the print statement by typing four spaces.

```
word = 'cat'
if len(word) < 5:
    print('word length is less than 5')
```

```
 word length is less than 5
```

When we use the Python interpreter we have to add an extra blank line after the print line in order for it to detect that the nested block is complete.

If we change the conditional test to `len(word) >= 5`, to check that the length of word is greater than or equal to 5, then the test will no longer be true. This time, the body of the if statement will not be executed, and no message is shown to the user:

```
if len(word) >=5:
    print('word length is greater than or equal to 5')
```

An if statement is known as a control structure because it controls whether the code in the indented block will be run. Another control structure is the for loop. Try the following, and remember to include the colon and the four spaces:

```
for word in ['Call', 'me', 'Ishmael', '.']:
    print(word)
```

```
 Call
 me
 Ishmael
 .
```

This is called a loop because Python executes the code in circular fashion. It starts by performing the assignment `word = 'Call'`, effectively using the word variable to name the first item of the list. Then, it displays the value of word to the user. Next, it goes back to the for statement, and performs the assignment `word = 'me'`, before displaying this new value to the user, and so on. It continues in this fashion until every item of the list has been processed.

Looping with Conditions

Now we can combine the `if` and `for` statements. We will loop over every item of the list, and print the item only if it ends with the letter l. We'll pick another name for the variable to demonstrate that Python doesn't try to make sense of variable names.

```
sent1 = ['Call', 'me', 'Ishmael', '.']
for xyzzy in sent1:
    if xyzzy.endswith('l'):
        print(xyzzy)
```

```
 Call
 Ishmael
```

You will notice that `if` and `for` statements have a colon at the end of the line, before the indentation begins. In fact, all Python control structures end with a colon. The colon indicates that the current statement relates to the indented block that follows.

We can also specify an action to be taken if the condition of the `if` statement is not met. Here we see the `elif` (else if) statement, and the `else` statement. Notice that these also have colons before the indented code.

```
for token in sent1:
    if token.islower():
        print(token, 'is a lowercase word')
    elif token.istitle():
        print(token, 'is a titlecase word')
    else:
        print(token, 'is punctuation')
```

```
Call is a titlecase word
me is a lowercase word
Ishmael is a titlecase word
. is punctuation
```

As you can see, even with this small amount of Python knowledge, you can start to build multiline Python programs. It's important to develop such programs in pieces, testing that each piece does what you expect before combining them into a program. This is why the Python interactive interpreter is so invaluable, and why you should get comfortable using it.

Finally, let's combine the idioms we've been exploring. First, we create a list of `cie` and `cei` words, then we loop over each item and print it. Notice the extra information given in the `print` statement: `end=' '`. This tells Python to print a space (not the default newline) after each word.

```
tricky = sorted(w for w in set(text2) if 'cie' in w or 'cei' in w)
for word in tricky:
    print(word, end=' ')
```

```
ancient ceiling conceit conceited conceive conscience conscientious conscientiously deceitful deceive deceived deceiving defi
```

5 Automatic Natural Language Understanding

This part of the book is about challenges still confronting Natural Language Processing. Go to https://www.nltk.org/book/ch01.html to read section 5: word sense disambiguation, pronoun resolution, generating language output, machine translation, spoken dialog systems, textual entailment, and limitations of NLP. Then return to the next question. Then answer the following question:

What are the similarities and differences between textual editing and textual criticism and the preparation of textual data for natural language understanding? How are some of the challenges confronting textual editing similar to the challenges that confront natural language processing?

Response:

I would say that all three require an understanding of language in the first place. Like you have to know a lagnuage at first to be able to edit, criticize and prepare it for natural language understanding.

As for differences in textual editing you simply working on creating an understandable and meanginful version of a text. Criticism involves trying to build a more analaytical approach to understanding the text. Finally, prepaaration of textual data for natural language understaning (as the words literally say) is the process of preparing the text in a way that computers are able to analyze the text in a way that can be useful for us to conduct further analysis and build tools with them.

As for some of the similarities of the challenges confronting textual editing and natural language processing, first of all both require you to understand a language in the first place. If for example I do not know Hindi at all, I porbably will not be able to edit or do NLP on a Hindi text. Also, language can be as I like to call it mischevious because the same words can have different meanings when expressedly in a different context and based on who is writing that. In that case both editing and NLP would face the challenge if you do not understand the contenxt of the text. Also the fact that language has dialects and variations can pose a problem. For example I speak Bengali and I know that there are different dialects of Bengali and in each even the same word can have two completely different meanings so that would be challenging.

8 Exercises

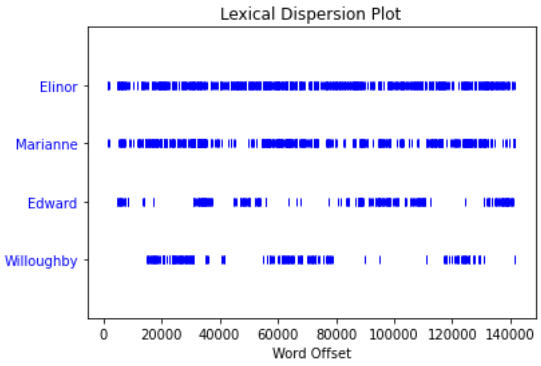
Produce a dispersion plot of the four main proagonists in *Sense and Sensibility*: Elinor, Marianne, Edward, and Willoughby. What can you observe about the different roles played by the males and females in this novel? Can you identify the couples?

Response:

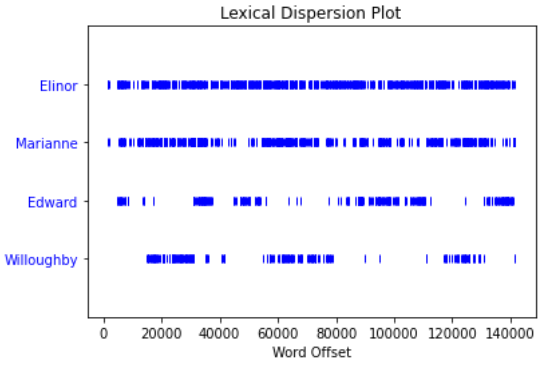
From the dispersion plot it can be concluded that the female characters lead the story overall more than the male characters.

It seems that Elinor and Edward are a couple and WIlloughby and Marianne are also a couple. I come to this conclusion trying to match up which characters co-exist most in in the dispersion plot.

```
#first way
from nltk.book import *
list = ['Elinor', 'Marianne', 'Edward', 'Willoughby']
text2.dispersion_plot(list)
```




```
#second way
text2.dispersion_plot(["Elinor", "Marianne", "Edward", "Willoughby"])
```

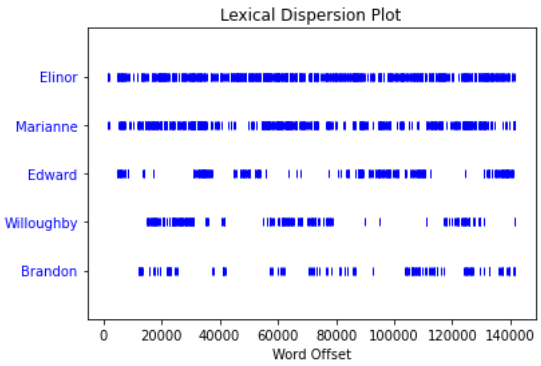


How would you add Colonel Brandon to this plot?

Response:

The following is a code of how to do that

```
text2.dispersion_plot(["Elinor", "Marianne", "Edward", "Willoughby", "Brandon"])
```



How could you see the words each of the characters are most likely to share in common?

```
# This is formatted as code
```

Response:

By using the code like we do below: thetextbeingstudied.common_contexts(['character1', 'character2'])

```
text2.common_contexts(['Elinor', 'Edward'])
```

```
said_, replied_, ._, ,_, and_, to_, ._was ._' ."_was ,_," on_' and_'
._could dear_, to_. of_' but_had ._had of_, ,_was
```

```
text2.common_contexts(['Elinor', 'Willoughby'])
```

```
said_, replied_, ._, ,_, and_, to_, ._was ._' and_' ._could dear_,
to_. cried_, of_' ._had ,_; of_, to_; in_' to_'
```

```
text2.common_contexts(['Marianne', 'Edward'])
```

```
said_, and_, ,_, for_, of_' ._was to_, ._' of_, ._, ._had that_was
."_was and_' in_' to_. ,_," ,_was ,_?" on_,
```

```
text2.common_contexts(['Marianne', 'Willoughby'])
```

```
said_, and_, ,_, cried_, for_, of_' ._was to_, ._' of_, ._, ._had
that_was and_' in_' to_. but_, on_, of_. ,_;
```

```
text2.common_contexts(['Marianne', 'Elinor'])
```

```
said_, and_, ,_, cried_, of_' ._was to_, ._' of_, ._, ._had that_was
."_was and_' in_' cried_. to_. but_, ,_," said_;
```

```
text2.similar('Elinor')
```

```
marianne she he edward willoughby it i lucy you they her that him and
this herself me there them "
```

```
text2.similar('Willoughby')
```

```
elinor marianne edward it lucy you her he she i herself ferrars him
this them that there me dashwood jennings
```

Question: What kinds of research questions might the literary textual critic approach digital text collections with? How might the digital environment help or hinder the creation of digital editions? Likewise, what kinds of questions are possible when the "dimentionality" of the text ar reduced? How can this be helpful, or not, to the literary scholar?

Response:

I think a literaly textual critic can approach a digital text collection with so many questions that it is hard to write a definitive list. In fact the book in section 5 lists out such questions like word sense disambiguation which aims to "work out which sense of a word was intended in a given context" or pronoun resolution.

These are already in the text itself so I would not repeat them. However, there are the other questions that I personally think a researcher can approach a text with like the ones below:

You can do genre analysis. Like can you find patterns within texts of a certain genre and use pattern identification to group together texts that belong to the same genre.

Maybe you can also figure out who the author of a particular set of text is if certain word similarities can be figured out as a pattern in different texts?

Maybe you can question the historical context of text(s) where a particular theme or topic being present can be indicative of the historical context it is being written in. For example poetry written in Bangladesh during its liberation war can have themes of war, suffering and the desire to be independent be topics that are present which indicates the historical context it was written in?

This list could go on and could very well vary based on the person asking the question but the above are some that comes to my head (beside what we read in section 5 and have done in the sets of code above).

As for how the digital environment might help or hinder the creation of digital editions, the first thing that comes to my mind is knowledge of the specific technology. If you cannot use the technology in the first place you wouldn't be able to create the digital edition.

Another thing is that technology is ever evolving and isn't permanent or stagnant. The technology you are using today might be obsolete completely tomorrow or the versions might have changed to a point that this isn't properly supported anymore so there is the need to be able to

keep up with those changes and make the necessary updates over time.

Speaking from personal experience, there is also the difficulty of navigating the issues of licensing and copyright for both the technology being used and the content being shared.

Recently I have also realized that digital editions of text need to be accessible so for example people with disabilities can access it.

I do have a basic understanding of dimensionality in data analaysis but have not yet fully grasped it in text analysis and I am trying to read up on the topic to understand better.