Coding & English Lit: Natural Language Processing in Python

This beginner-level tutorial shows you how to use Python and the Natural Language Toolkit (NLTK) to analyze texts imported from URLs and downloaded from files using Wuthering Heights *(the book and the Kate Bush song).*

Let’s say I’m a high schooler writing a lit crit essay on *Wuthering Heights*. For those of you who didn’t read the book in high school, the plot is basically that there is a young woman named Cathy from a wealthy family who lives on the moors.

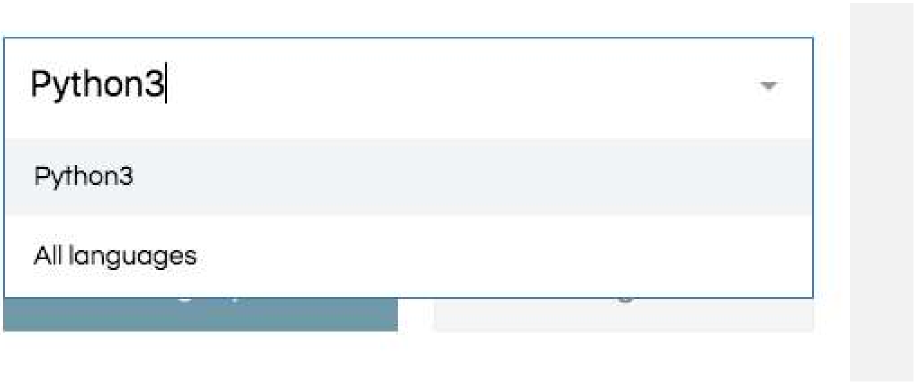
There is another rich dude on the moors, Edgar Linton. Cathy is supposed to marry him. But obviously, she is way more into Heathcliff, the abandoned child her family took in when she was also a child. Heathcliff used to be like a brother, but now he is more like a dark and handsome rogue.

Unfortunately, Heathcliff has a lot of issues, so things don't really work out, and it is a very tortured tale.

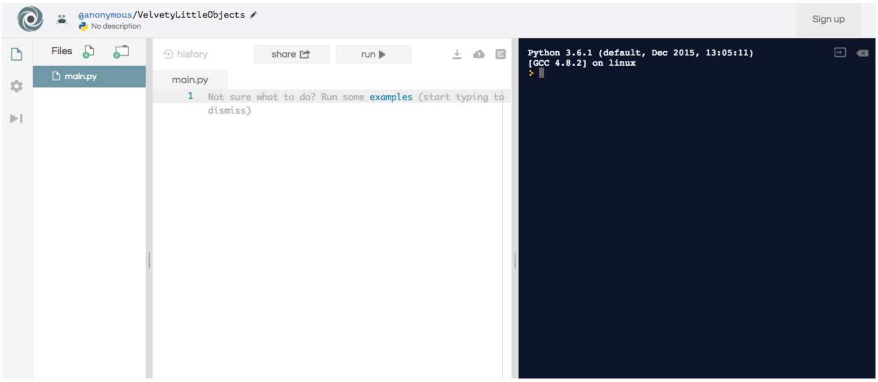
But anyway, let's say I'm a high schooler writing a lit crit essay about *Wuthering Heights*. Obviously the question is: how can I leverage Python programming and the Natural Language Toolkit (NLTK) to my advantage, to write the best lit crit essay ever?

Getting started

The most beginner-friendly Python development environment is probably repl.it (you don't even have to download anything!). If you're comfortable, feel free to use a Python workspace on Cloud9 — it's faster. Otherwise, though, go to repl.it and search for "Python3" when it asks you to type in a language.



Then it should take you to a blank Python editor.



```
Not sure what to do? Run some examples (start typing to
dismiss)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
```

NLTK doesn't come with Python by default, so first we need to import it, as well as a few other tools we need. Add this code to your editor (the white part of the screen):

```
import nltk
from urllib import request
from nltk.sentiment.vader import
SentimentIntensityAnalyzer

nltk.download('punkt')
```

This code imports NLTK, as well as a sentiment analyzer (get excited for that) and then downloads a specific package called "punkt" that helps with tokenizing (breaking down texts into words, sentences, etc). The second line, where we import `request` from `urllib`, will allow us to grab text off the internet and process that, too.

Getting text from the internet

Project Gutenberg hosts the texts of many old books for free, and luckily for us, *Wuthering Heights* is located at this URL:

```
https://www.gutenberg.org/files/768/768.txt
```

Let’s store the URL in a variable. Add this line to your code:

```
url =
"https://www.gutenberg.org/files/768/768.txt"
```

A variable is just a container for a value — in this case, the URL. Now, whenever we say `url`, the computer knows we are referring to that web address. The quotation marks around the URL indicate that we are storing it as a **string**, a word or phrase bounded by quotation marks.

OK, now we are going to do some magic. We are going to use Python to open the webpage, read the contents, and use NLTK to tokenize it into a list of a million separate little words so we can analyze it better.

Add these lines to your code:

```
response = request.urlopen(url)
raw = response.read().decode('utf8')
tokens = nltk.word_tokenize(raw)
text = nltk.Text(tokens)
```

First, we open the webpage and store it in a variable called `response`. Then, we read the contents and convert it into UTF-8 (friendly to both computers and humans) and store that in the `raw` variable. Then, we tokenize the raw text and store that in `tokens`. Finally, we convert the `tokens` to an NLTK-friendly format and call it `text`.

Now we can do stuff to our `text` variable, which holds all of *Wuthering Heights* in a computer-friendly format!

Concordance

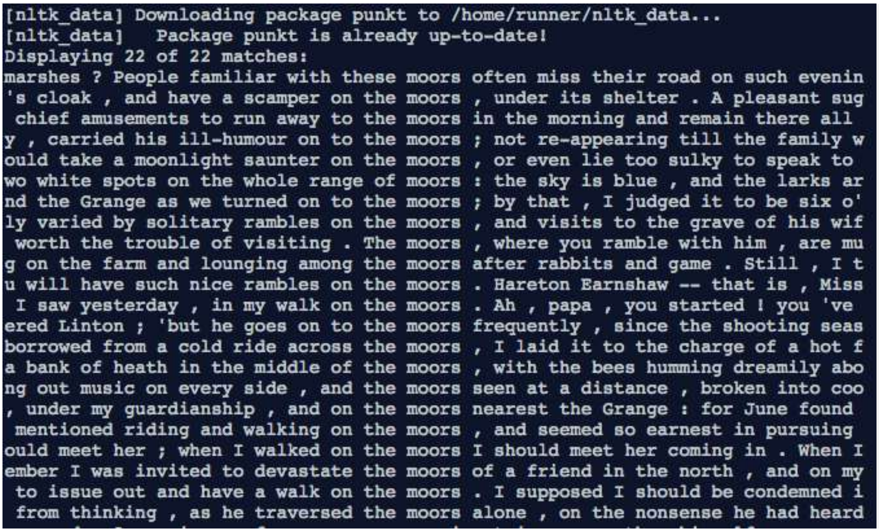
OK, let's say I'm writing an essay about how the moors are actually a personification of Heathcliff, because they are so rugged and wild. I'll want to examine each instance of the word "moors" in *Wuthering Heights*.

I can do that by just running this command:

```
text.concordance("moors")
```

Add the above line to your code and then press the "Run" button above your editor.

Be patient as the computer reads the ENTIRE TEXT OF *WUTHERING HEIGHTS* (faster than you, mind you). What a treasure trove of results!



```
[nltk_data] Downloading package punkt to /home/runner/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Displaying 22 of 22 matches:
marshes ? People familiar with these moors often miss their road on such evenin
's cloak , and have a scamper on the moors , under its shelter . A pleasant sug
 chief amusements to run away to the moors in the morning and remain there all
y , carried his ill-humour on to the moors ; not re-appearing till the family w
ould take a moonlight saunter on the moors , or even lie too sulky to speak to
wo white spots on the whole range of moors : the sky is blue , and the larks ar
nd the Grange as we turned on to the moors ; by that , I judged it to be six o'
ly varied by solitary rambles on the moors , and visits to the grave of his wif
 worth the trouble of visiting . The moors , where you ramble with him , are mu
g on the farm and lounging among the moors after rabbits and game . Still , I t
u will have such nice rambles on the moors . Hareton Earnshaw -- that is , Miss
 I saw yesterday , in my walk on the moors . Ah , papa , you started ! you 've
ered Linton ; 'but he goes on to the moors frequently , since the shooting seas
borrowed from a cold ride across the moors , I laid it to the charge of a hot f
a bank of heath in the middle of the moors , with the bees humming dreamily abo
ng out music on every side , and the moors seen at a distance , broken into coo
, under my guardianship , and on the moors nearest the Grange : for June found
 mentioned riding and walking on the moors , and seemed so earnest in pursuing
ould meet her ; when I walked on the moors I should meet her coming in . When I
ember I was invited to devastate the moors of a friend in the north , and on my
 to issue out and have a walk on the moors . I supposed I should be condemned i
 from thinking , as he traversed the moors alone , on the nonsense he had heard
```

Concordance of "moors"

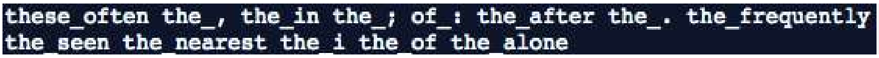
Already I can see that people go on a lot of long, solitary "rambles on the moors." And that even if people are familiar with the moors, they lose their way! Could this be a metaphor for how unpredictable and temperamental Heathcliff is? Thanks, NLTK!

Common Contexts

Common contexts allows us to see which other words are often used along with the word in question. If I want to see the common contexts of the word "moors," I can run:

```
text.common_contexts(["moors"])
```

The results:



```
these_often the_, the_in the_; of_: the_after the_. the_frequently
the_seen the_nearest the_i the_of the_alone
```

Common contexts of "moors"

Immediately, I see that moors is often used with "frequently" and "alone." It seems that the moors invite solitude… like Heathcliff.

In the original code, you'll notice that the word "moors" is in square brackets: `["moors"]`. Although `"moors"` itself is a **string** (a word or phrase bounded by quotation marks), it becomes part of a data structure called a **list**when it's surrounded by square brackets.

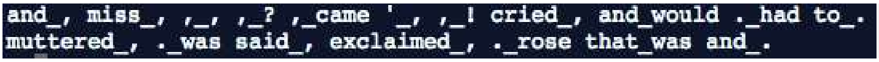
We can include multiple items in a list, such as:

```
["Cathy", "Heathcliff"]
```

And likewise, we can analyze the common contexts of multiple words. For instance, I can see how Cathy and Heathcliff are often used together if I run:

```
text.common_contexts(["Cathy", "Heathcliff"])
```

The results:



```
and_, miss_, ,_, ,_? ,_came '_, ,_! cried_, and_would ._had to_.
muttered_, ._was said_, exclaimed_, ._rose that_was and_.
```

Common contexts of “Cathy” and “Heathcliff”

Lots of emotion, apparently, since they “cried” and “exclaimed” to or about each other quite a bit and also used lots of punctuation: “?” and “!”. Also, look at that subjunctive “would.” Cathy's relationship with Heathcliff is full of subjunctives, of things that “would” have been if only he hadn’t had so many issues.

```
# text.concordance("moors")
```

This will spare the computer from constantly producing the concordance and allow you to keep your output screen clean.

Word frequency

OK, time for some quantitative analysis. Let’s see who Cathy really loves more, Heathcliff or Edgar Linton, by having the computer count how often each one is mentioned.

The code below counts how often Heathcliff is mentioned and stores the number in a variable called `count`:

```
count = text.count("Heathcliff")
```

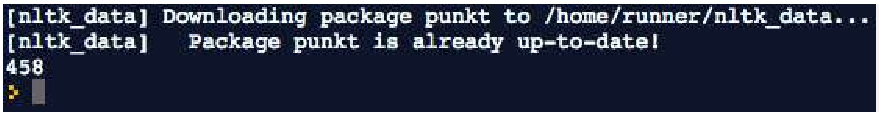
But if you run this code, does anything appear on your screen?

No! You've simply stored a number in a variable. Whereas `concordance` and `common_context` were designed by NLTK to print text automatically, we will need to use a `print` statement to see the value of our `count` variable here.

After calculating the `count`, add the following line to your code:

```
print(count)
```

Now you should see this output:



```
[nltk_data] Downloading package punkt to /home/runner/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
458
> 
```

Heathcliff is mentioned 458 times in Wuthering Heights!

OK, now YOU calculate how many times the word "Edgar" is mentioned in *Wuthering Heights*. Who does Cathy love more?

A little math

Let's suppose you wanted to find out what percentage of the book is made up of the word "Heathcliff." You already have the number of times the word “Heathcliff” occurs, stored in your `count` variable. Now you just need the number of words in the entire text.

Luckily, the `len` function in Python can calculate the length of a data structure. In this case, we can run `len` on our `text` variable (which holds every single tokenized word in *Wuthering Heights*) to get the number of words in the text.

This code stores the total number of words in *Wuthering Heights* in a variable called `total`:

```
total = len(text)
```

Now we can calculate the percentage of *Wuthering Heights* made up of the word “Heathcliff”!

```
percentage = count/total
```

Don’t forget to print the percentage if you want to see the answer.

Sentiment analysis

OK, now let’s analyze the sentiment of *Wuthering Heights*. Sentiment analysis reports the percentage of text that’s positive, negative, or neutral, and delivers a compound score between -1 (super negative) and 1 (super positive).

If you’re already thinking that *Wuthering Heights* is going to be pretty negative, you’re right!

First, we need to set up a few tools for our sentiment analysis. Add these lines to your code:

```
nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()
```

Now our sentiment analyzer is stored inside a variable called `sid`.

Let’s analyze the entire text of *Wuthering Heights* (even though we all know it’s going to be negative)! The tricky part here is that I can’t use my `text` variable, because it’s all the broken-up tokenized words, and the sentiment analyzer needs a **string**.

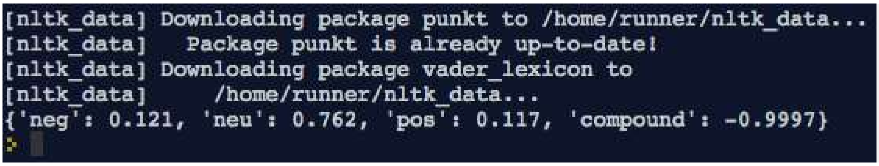
The `raw` variable from early in our tutorial stored the original decoded string of the *Wuthering Heights* text, so we're going to analyze that.

To get the polarity scores of the raw text, run this code:

```
scores = sid.polarity_scores(raw)
print(scores)
```

Again, be patient as the computer downloads what it needs and reads THE ENTIRE BOOK AND ANALYZES EVERY WORD. Seriously, you may want to get up and get a coffee while you're waiting. Like, leave your house and go to Starbucks and then come back.

Eventually, this should appear:



```
[nltk_data] Downloading package punkt to /home/runner/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /home/runner/nltk_data...
{'neg': 0.121, 'neu': 0.762, 'pos': 0.117, 'compound': -0.9997}
>
```

Sentiment analysis of Wuthering Heights

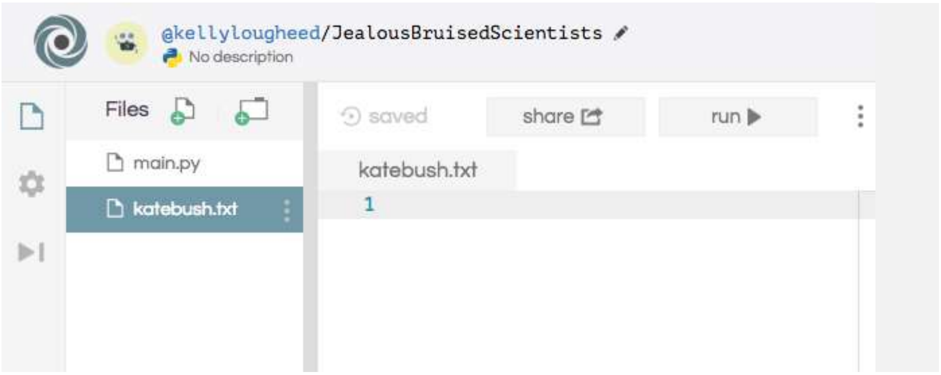
OK, a little negative and a little positive, but the compound score is pretty overwhelming negative. As we suspected!

Reading & analyzing files

OK, obviously it was super boring to wait for the computer to read and analyze an entire book, and we don't want to do that again.

Let's analyze something shorter. Now, *Wuthering Heights*is pretty famous as a book, but it is possibly even more famous as a song by Kate Bush with an epic music video.

I've put the lyrics here for you. Create a new file called **katebush.txt** in your repl:



Click on the file icon with the plus sign to create a new file and call it "katebush.txt"

Then paste the lyrics into that file.

Return to **main.py** (the original Python file we've been working in) and **for Pete's sake, comment out the sentiment analysis of the *Wuthering Heights* raw text** unless you want to be waiting around for a million years again:

```
# scores = sid.polarity_scores(raw)
# print(scores)
```

Instead, we're going to analyze the sentiment of Kate Bush's lyrics.

First, we need to use Python to get and read the file. This code will do the trick:

```
file1 = open("katebush.txt", "r")
lyrics = file1.read()
```

First, we open **katebush.txt** in "r" mode, which means "read," and store it in a variable called `file1`. Then we read the actual contents of the file and store it in a variable called `lyrics`.

What kind of variable is `lyrics`? If you said **string**, you are correct — `lyrics` contains all the words to the song, surrounded by quotation marks.

Lists and list operations

I could analyze the entire song by analyzing lyrics, but I think I would like to analyze it lyric-by-lyric. Instead of a big **string**, I want to convert `lyrics` into a **list**containing each individual line. Here's how I do that:

```
lyrics = lyrics.split("\n")
```

Now, instead of a string, I've split `lyrics` up by the character `\n`, which is computer code for a line break. Now instead of a variable like this:

```
lyrics = "Out on the wiley, windy moors\n We'd
roll and fall in green..."
```

I have a variable like this:

```
lyrics = ["Out on the wiley, windy moors",
"We'd roll and fall in green"]
```

Note: If I were dealing with a text passage inside of lyrics with line breaks, I could have broken up the text with NLTK's sentence tokenizer. Instead of using `split`, I would have done: `lyrics = nltk.sent_tokenizer(lyrics)`

Now I can loop through the entire **list** of lyrics and analyze each lyric separately!

The following loop goes through each lyric in the list of `lyrics`, calculates the polarity scores, and prints both the lyric and its scores:

```
for lyric in lyrics:
    lyric_scores = sid.polarity_scores(lyric)
    print(lyric)
    print(lyric_scores)
```

*Note: you could call `lyric` anything you wanted, like `l` or `line` or whatever. You're just creating a temporary name to refer to the current element of the **list** you're looping through.*

Try it out! And as always, be patient…

Surprise, all the lyrics range from neutral to negative! What a tortured tale.