Nhi Nguyen & Michelle Purnama (Student Leader)

TD WS2 Guideline: Text Analytics with Python

Before we start

- Make sure you download Anaconda and try launching Jupyter Notebook beforehand.
- The installation instructions for TD WS 2 can be found **here**.

Let's Start!

- Let's start by **creating a Jupyter Notebook file named "TD WS2"** in Anaconda
- We will start by **installing** NLTK (Natural Language Toolkit). NLTK is a powerful Python package that provides a set of diverse natural language algorithms.

```
!pip install nltk
```

- After that, we will **import** and load **NLTK package** to our Jupyter Notebook
- **Notes:** The "**install**" statement puts the code somewhere that Python expects those kinds of things to be, and the "**import**" statement says "go look there for something named "abc xyz" now, and make the data available to me for use".

```
#Loading NLTK
import nltk
```

- Now, let's choose a paragraph we want to analyze and put it in our code. Since it's almost Halloween, let's go with a Halloween-themed short story.

The *Mummified Mom* Story:
Last year my daughter and her children were invited to a Halloween party. Her older son wanted to go as Count Dracula; her daughter, as a ballerina; her younger son, as the cabin boy in Treasure Island. Then my daughter donned her own costume, wrapping herself in strips of white sheeting. At the party she collapsed, exhausted, on the sofa.

"And who are you?" someone asked her.

"I am a tired mummy," my daughter said.

- Okay, let's put this story in our code. We use triple quotes (""" """) to make sure Jupyter Notebook knows the start and the end of the string.

```
text = """Last year my daughter and her children were invited to a Halloween party. Her older son wanted to go as Count Dracula; her daughter, as a ballerina; her younger son, as the cabin boy in Treasure Island. Then my daughter donned her own costume, wrapping herself in strips of white sheeting. At the party she collapsed, exhausted, on the sofa.

"And who are you?" someone asked her.

"I am a tired mummy," my daughter said.
"""

print(text)
```

PHASE 1: Tokenization - Breaking Paragraphs Into Smaller Pieces

1.1) Tokenization

- The first step in text analytics is tokenization. Token is a single entity that is the building blocks for sentence or paragraph.
- This is the process of breaking down a text paragraph into smaller chunks of text such as words or sentences.

Sentence Tokenization = Breaking down text paragraph into **sentences**

- In order to do tokenization, we need to download the **punkt** module.
- Punkt Sentence Tokenizer = divides a text into a list of sentences

```
nltk.download('punkt')
```

- Let's write some code that will execute sentence tokenization by importing the **sent_tokenize class** from **nltk.tokenize module**.

```
from nltk.tokenize import sent_tokenize

tokenized_text1=sent_tokenize(text)
print(tokenized_text1)
```

- You will notice that the original paragraph will be broken down into sentences. Hence, the given text is tokenized into sentences.

Word Tokenization = Breaking down text paragraph into **words**

- Now, instead of using **sent_tokenize class**, let's try to use **word_tokenize class** from **nltk.tokenize module**.

```
from nltk.tokenize import word_tokenize

tokenized_text2=word_tokenize(text)
print(tokenized_text2)
```

1.2) Removing Punctuation

- Notice how punctuation (period, comma, semi-colon, exclamation point, etc.) are tokenized as individual words? We want to eliminate punctuation in our analysis as those are considered as noise in the text, which won't yield a meaningful insight!
- To filter out those punctuation from our given text, we use **for loop** in Python and **isalpha()** method
- **isalpha()** method returns "True" if all characters in the string are alphabets, Otherwise, It returns "False".

```
remove_punct=[]
for word in tokenized_text2:
   if word.isalpha():
      remove_punct.append(word)
```

- Let's print out the **remove_punct** variable that we just created. You should see that the punctuations are gone now!

```
print(remove_punct)
```

1.3) Frequency Distribution

- Think of **Frequency Distribution** as counting words in a given text. It will allow you to see which words are used the most in the paragraph. Let's import its module and apply it to our remove_punct.

```
from nltk.probability import FreqDist
freqdist = FreqDist(remove_punct)
print(freqdist)
```

- You will get something similar to this: **<FreqDist with 56 samples and 75 outcomes>**
- Now, let's see the most three common words!

```
freqdist.most_common(3)
```

- **[('her', 5), ('daughter', 4), ('my', 3)]** → So from here we can see that "**her**" is the most frequent word which appears 5 times in the story!
- How about the most 5 common words?

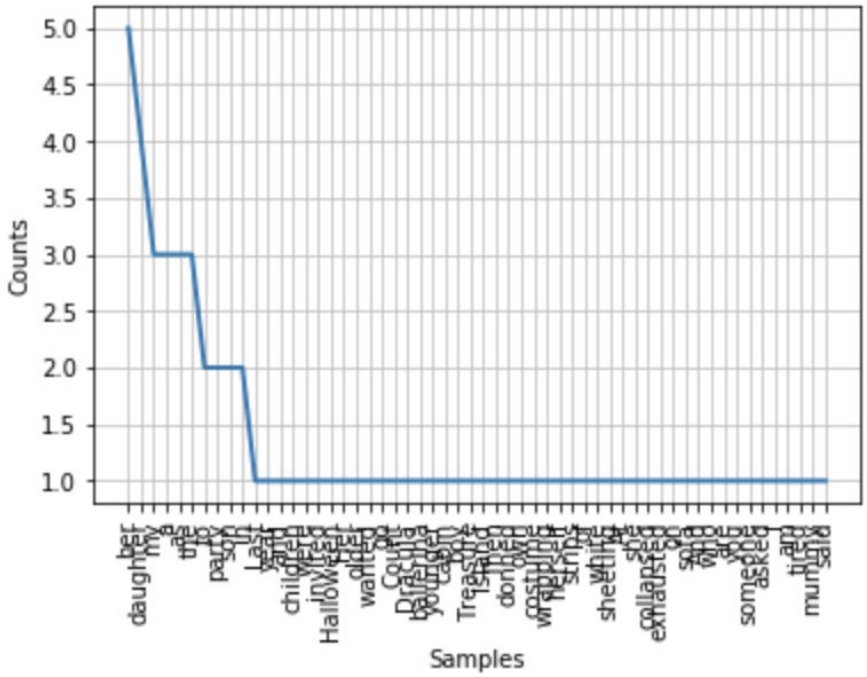
```
freqdist.most_common(5)
```

- Should we plot it on a graph for better data visualization?

```
# Frequency Distribution Plot

import matplotlib.pyplot as plt
freqdist.plot()
plt.show()
```

- We will get a graph that looks like this.



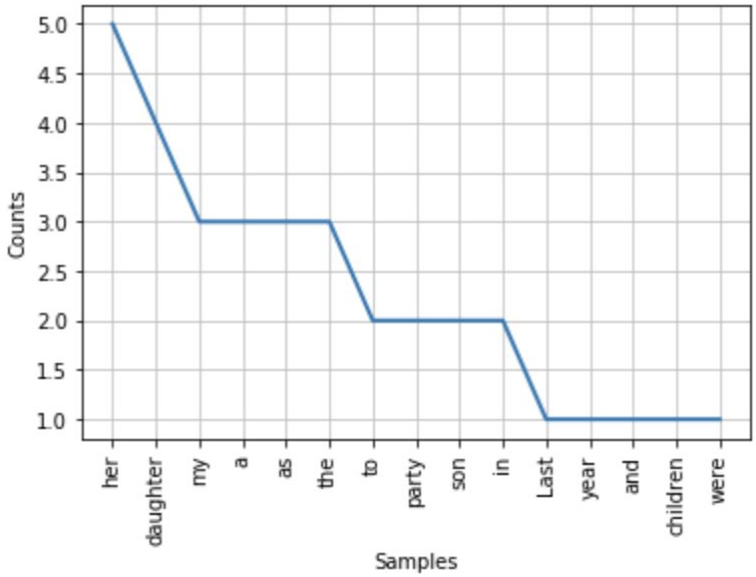
- This is a very clustered graph since the graph includes every single word in this graph as the x-values. Let's include only the first 15 words by adding number 15 in the plot() function.

```
# Frequency Distribution Plot

freqdist.plot(15,cumulative=False)
```

```
plt.show()
```

- You will get a graph that looks like this:



- When we do **cumulative = True**, we will have a cumulative frequency chart, which is calculated by adding each frequency from a frequency distribution table to the sum of its predecessors. Cumulative graphs are usually used to showcase trends, number, progress, or rate over a period of time.
- In our case, since we want to compare each value to one another, we do not need **a cumulative chart**.

1.3) Exercise

- Now that you understand how to code in Python to break down paragraphs and identify the most frequent words, let's try to do it with a different paragraph!
- We recommend that you **create a new Jupyter Notebook file named "Exercise"** for the exercise to avoid confusion!
- Choose any story from this link Funny Halloween Stories and try it out yourself!
- You can copy and paste all the codes that we have learned so far to the new Jupyter Notebook.
- Remember that you have to download the packages, modules, and classes again since this is for a new **"Exercise"** notebook.

- Great, from now on, you will be able to plug in any paragraph you want to analyze!

STORY: Trick or Tryst?
text = """Desperate for a Halloween costume to wear to a party, my 43-year-old daughter had an inspired idea. She put on a slinky black dress, fishnet stockings and balanced a small tabletop on her head. On it was a lamp, a champagne glass and an ashtray with two cigarette butts. She went as a one-night stand. And won first prize."""

```
from nltk.tokenize import sent_tokenize
tokenized_text1=sent_tokenize(text)
print(tokenized_text1)

from nltk.tokenize import word_tokenize
tokenized_text2=word_tokenize(text)
print(tokenized_text2)

remove_punct=[]
for word in tokenized_text2:
    if word.isalpha():
        remove_punct.append(word)

print(remove_punct)


from nltk.probability import FreqDist
freqdist = FreqDist(remove_punct)
print(freqdist)


import matplotlib.pyplot as plt
freqdist.plot(15)
plt.show()
```

PHASE 2: Stopwords Removal

2.1) Stopwords

- Great! Before we start this Phase, **make sure you switch back to your first notebook**, the **"TD WS2"** notebook.
- Stopwords are considered as noise in the text. Text may contain stopwords such as **is**, **am**, **are**, **this**, **a**, **an**, **the**, etc.
- In NLTK for removing stopwords, you need to create a list of stopwords and filter out your list of tokens from these words.
- Let's create a list of stopwords for English language

```
nltk.download('stopwords')

from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)
```

- Let's print out the Tokenized Sentence that we created in Phase 1 after removing the punctuations

```
print("Tokenized Sentence:",remove_punct)
```

- We will get something like this:

```
Tokenized Sentence: ['Last', 'year', 'my', 'daughter', 'and', 'her', 'children', 'were', 'invited', 'to', 'a', 'Hallo
ween', 'party', 'Her', 'older', 'son', 'wanted', 'to', 'go', 'as', 'Count', 'Dracula', 'her', 'daughter', 'as', 'a',
'ballerina', 'her', 'younger', 'son', 'as', 'the', 'cabin', 'boy', 'in', 'Treasure', 'Island', 'Then', 'my', 'daughte
r', 'donned', 'her', 'own', 'costume', 'wrapping', 'herself', 'in', 'strips', 'of', 'white', 'sheeting', 'At', 'the',
'party', 'she', 'collapsed', 'exhausted', 'on', 'the', 'sofa', 'And', 'who', 'are', 'you', 'someone', 'asked', 'her',
'I', 'am', 'a', 'tired', 'mummy', 'my', 'daughter', 'said']
```

- To filter out those stopwords from our given text, we use for loop in Python

```
remove_swords=[]
for w in remove_punct:
   if w not in stop_words:
     remove_swords.append(w)
```

- Let's print out the **remove_swords** var that we just created.

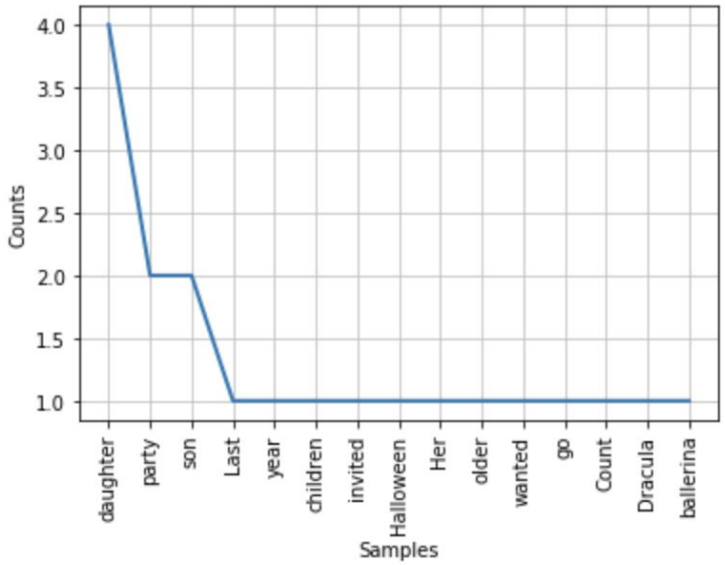
```
print("Tokenized Sentence:",remove_swords)
```

- As you can see from the result, we have two sentences. The **remove_swords** var in this phase has no stopwords that exist in the **remove_punct** var in Phase 1.
- Let's run the Frequency Distribution method again after we remove the stopwords.

```
freqdist_filtered = FreqDist(remove_swords)
print(freqdist_filtered)
```

- You will receive a result that looks like this: **<FreqDist with 38 samples and 43 outcomes>**
- Notice that in Phase 1, we have 56 samples and 75 outcomes. But now in Phase 2, we only have 38 samples and 43 outcomes. So we eliminated the stopwords!
- Let's graph it!

```
freqdist_filtered.plot(15,cumulative=False)
plt.show()
```



- Notice that the stopwords like "**her**" or "**my**" are gone in this graph.

2.2) Exercises

- Alright, remember the exercise you did for Phase 1? We will reuse that paragraph and try to eliminate stopwords from that text. Let's do it.
- Make sure to go back to the "**Exercise**" notebook.
- Since everyone chose different stories, we have made an Exercise Keys document that you can use to compare your results!

```
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))

print("Tokenized Sentence:",remove_punct)

remove_swords=[]
for w in remove_punct:
   if w not in stop_words:
      remove_swords.append(w)

print("Filtered Sentence:",remove_swords)

freqdist_filtered = FreqDist(remove_swords)
print(freqdist_filtered)

freqdist_filtered.plot(15)
plt.show()
```

PHASE 3: Lexicon Normalization

- After removing stopwords, there is still another type of noise in text. For example, the word "connection" or "connected" or "connecting" has a common root word "connect".
- This is where **lexicon normalization** comes in. It reduces derivationally related forms of a word to a common root word.
- There are two methods for Lexicon Normalization: Stemming and Lemmatization.

Stemming = a process of linguistic normalization, which reduces words to their word root word or chops off the derivational affixes.

- First, let's import **stemming module** and **class** in our code

```
# Stemming
from nltk.stem import PorterStemmer
#from nltk.tokenize import sent_tokenize, word_tokenize

ps = PorterStemmer()
```

- Now we will create the stemmed version of our text.
- Let's print out the Filtered Sentence from Phase 2 again

```
print("Filtered Sentence:",remove_swords)
```

- Similar to what we did when we removed stopwords, we will use another for loop. Now we can print out the Stemmed Sentence we just did

```
stemmed_text=[]
for w in remove_swords:
   stemmed_text.append(ps.stem(w))
print("Stemmed Sentence:",stemmed_text)
```

- You will see how every word has become lowercase. Some of the words go back to its root word correctly, but some do not. Hence, there are still limitations of Stemming, and that's why Lemmatization is developed!

Lemmatization = a better approach than stemming

- Lemmatization is usually more sophisticated than stemming. Stemming works on an individual word without any knowledge of the context.
- For example, The word "better" has "good" as its lemma. This will be missed by stemming because it requires a dictionary look-up.

- Let's import **lemmatization module and class** in our code
- In order to lemmatize, you need to create an instance of the **WordNetLemmatizer()** and call the **lemmatize()** function on a single word

```
nltk.download('wordnet')

from nltk.stem.wordnet import WordNetLemmatizer
lem = WordNetLemmatizer()

from nltk.stem.porter import PorterStemmer
```

```
stem = PorterStemmer()
```

- Let's test both Stemming and Lemmatization methods with the word "flying"

```
word = "flying"
print("Stemmed Word:",stem.stem(word))
print("Lemmatized Word:",lem.lemmatize(word,"v"))
```

- You will see that Lemmatization produces a better guess than Stemming: **fli** vs **fly**
- The "v" is to provide the tag as the second argument to lemmatize() to provide more accuracy as sometimes, the same word can have multiple lemmas based on the meaning/context

```
print(lem.lemmatize("stripes", 'v'))
# strip

print(lem.lemmatize("stripes", 'n'))
# stripe
```

PHASE 4: Understand POS Tagging

4.1) POS Tagging

- Next, we are moving on to POS Tagging, aka Part-of-Speech tagging.
- What is POS Tagging you ask? It is used to identify the grammatical group of a given word. Whether it is a NOUN, PRONOUN, ADJECTIVE, VERB, ADVERBS, etc. based on the context.
- Thus, it looks for relationships within the sentence and assigns a corresponding tag to the word.
- Let's use this sentence as an example: 'T' for Temple 'U', University! Fight, fight, fight, for the Cherry and the White, for the Cherry and the White, we will fight fight fight!
- Make sure to put them in triple quotes!

```
sentence = """ 'T' for Temple 'U', University! Fight, fight, fight, for the Cherry and the White, for the Cherry and the White, we will fight fight fight! """

print(sentence)
```

- Again, to analyze the sentence, let's first break it down into smaller chunks.

```
tokens=nltk.word_tokenize(sentence)
print(tokens)
```

- Okay, let's apply POS Tagging to our sentence using **nltk.pos_tag()**

```
nltk.download('averaged_perceptron_tagger')

nltk.pos_tag(tokens)
```

- After we execute the code, here is the few lines of the result:

('for', 'IN'),
('the', 'DT'),
('Cherry', 'NNP'),
('and', 'CC'),
('the', 'DT'),
('White', 'NNP'),
('we', 'PRP'),
('will', 'MD'),
('fight', 'VB'),
('fight', 'RB'),
('fight', 'NN'),

- DT stands for determiner "the"
- NNP stands for singular proper noun since "Cherry" is capitalized.
- CC stands for coordinating conjunction "and"

- For more descriptions on each POS, check out this article here:
- https://medium.com/@gianpaul.r/tokenization-and-parts-of-speech-pos-tagging-in-pythons-nltk-library-2d30f70af13b

4.2) Exercises

- Okay, so for this exercise, let's pick one sentence from your Halloween story.
- Make sure to go back to the "**Exercise**" notebook.

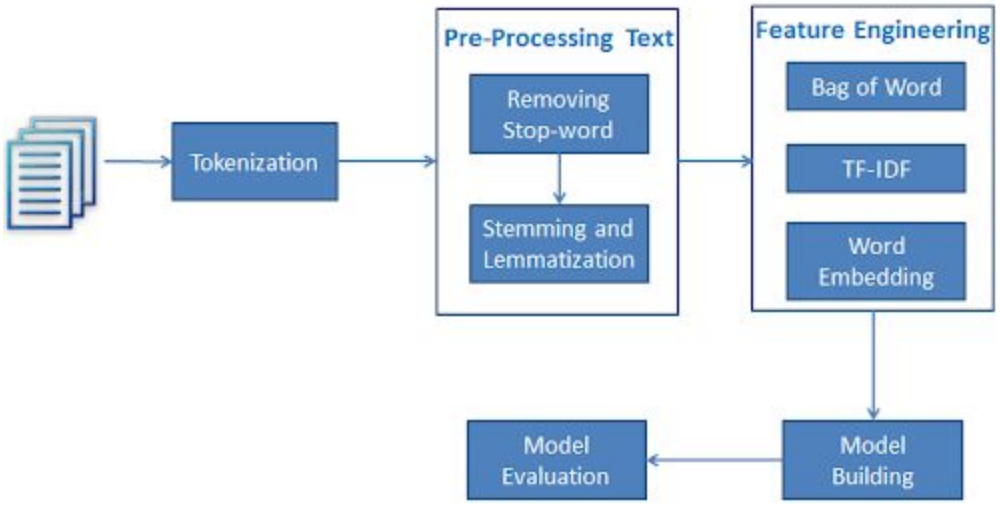
```
sentence = """It was Halloween night when a driver called our road-service dispatch office complaining that he was locked out of his car"""
tokens=nltk.word_tokenize(sentence)
print(tokens)

nltk.pos_tag(tokens)
```

PHASE 5: Sentiment Analysis

5.1) Text Classification

- Okay, moving on, let's go back to your "**TD WS 2**" notebook.
- Text classification is one of the important tasks of text mining.
- It has various applications in today's computer world such as spam detection, task categorization in CRM services, categorizing products on E-retailer websites, or classifying the content of websites for a search engine.



5.2) Performing Sentiment Analysis using Text Classification

- Uptill Phase 4, you have learned data pre-processing using NLTK. Now, you will get into a tiny bit of Sentiment Analysis using Text Classification.

- Monkey Learn also created a simple Sentiment Analysis test, check it out here: https://app.monkeylearn.com/main/classifiers/cl_pi3C7JiL/
- We have a sample dataset that you can use to see the ranking of different values based on sentiment analysis.
- The dataset is on "Sentiment Analysis of Movie, Reviews". It is also located in **TD WS 2 Folder for Students.** The dataset's name is **sentimentanalysis.tsv**
- Let's download it and examine it!

- This data has 5 sentiment labels:
  - 0 - negative
  - 1 - somewhat negative
  - 2 - neutral
  - 3 - somewhat positive
  - 4 - positive
- Now that you have download the dataset, let's import it in our Jupyter Notebook
- In order to import it, we need to find the **file path.**
  - **For Mac Users**,
    - Click on the file **sentimentanalysis.tsv**
    - Then click this shortcut: **Command + Option + C**
    - Then click **Command + V** to paste the file path.
  - **For Windows Users**,
    - Hold the **Shift** key while right click the file
    - Choose "Copy as path" from the drop down menu
    - Then click **Ctrl + V** to paste the file path
- After you have the file path, replace **thefilepath** below with your file path.

```
import pandas as pd

data=pd.read_csv('thefilepath', sep='\t')
```

- For **Nhi's laptop**, the code above will be like this: data=pd.read_csv('**/Users/Nhi/Desktop/sentimentanalysis.tsv**', sep='\t')

- Okay, let's print the first few rows of this dataset out using **head()** method.

```
data.head()
```

- What about the first 10 rows instead?

```
data.head(10)
```

- Awesome! Now let's get a quick overview of this dataset

```
data.info()
```

- Here, Python tells us that we have a total of 4 columns. It includes the column names, data types (int64 or object), and the entries.
- The only column we want to focus for now is the **Sentiment** column.
- Let's count the total entries for each value of the Sentiment column.

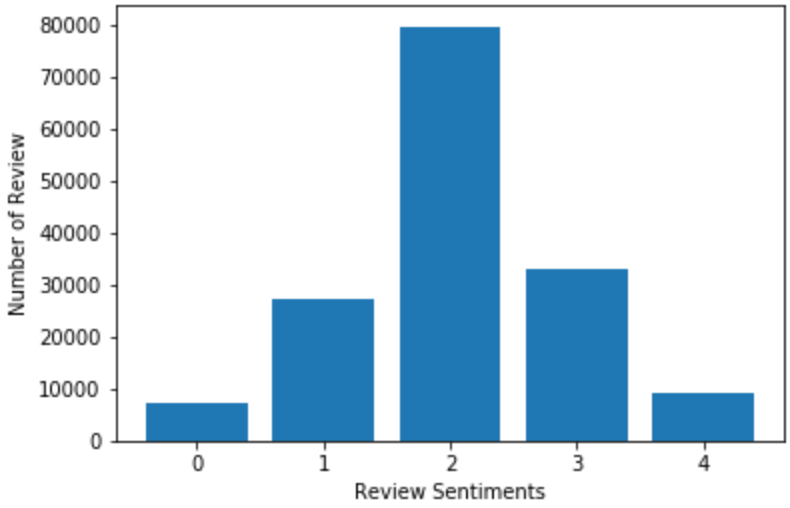
```
data.Sentiment.value_counts()
```

- Cool, from the result, we see that **"2" or "neutral" has the most entries.**
- For better visualization, let's graph it! We don't have to import matplotlib again since we already did it in earlier phases.

```
Sentiment_count=data.groupby('Sentiment').count()
print(Sentiment_count)
```

```
plt.bar(Sentiment_count.index.values, Sentiment_count['Phrase'])
plt.xlabel('Review Sentiments')
plt.ylabel('Number of Review')
plt.show()
```

- You should see this graph



- And that concludes our workshop today!
- To recap, we learned how to:
    - Break down paragraphs into smaller pieces like sentences or words.
    - Remove punctuation and stopwords to increase the accuracy of our analysis.
    - Use Stemming or Lemmatization to turn words to their base words.
    - Understand Part-of-Speech tagging.
    - Create simple graphs in Python
    - Scratch a bit of the surface of Sentiment Text Analysis!

LEARNING RESOURCES

- To read more about Text Analysis, go to this Monkey Learn article: https://monkeylearn.com/text-analysis/
- Put your new Python skills to the test and challenge yourself by doing this tutorial! https://www.dataquest.io/blog/tutorial-text-analysis-python-test-hypothesis
- To take a bootcamp course on Python: https://www.udemy.com/course/complete-python-bootcamp/

GIVE US YOUR FEEDBACK!

- Please use this link **http://bit.ly/TD-SAT2** to submit the attendance and feedback form for this TD Saturday Workshop 2.
- Thank you for joining Nhi and Michelle today! Have a great weekend :)

AIS Technical Development Workshop 2: Text Analytics

Date: Saturday, October 19, 11-1pm

Nhi Nguyen & Michelle Purnama (Student Leader)