THINKFUL

Text Data Acquisition and Ingestion

DATA SCIENCE

Warm Up

- What are some sources where we can get text data?

Agenda

- Acquiring Text Data
- Text Data Sources
- Considerations for Text Data Acquisition
- Acquiring Data via Web Scraping
- Acquiring Data from RSS Feeds
- Acquiring Data from APIs
- Creating a Corpus for NLP

Acquiring Text Data

- A Natural Language Processing project begins with acquiring the right kind of text data and building a custom corpus that contains structural and contextual features from the domain in which you are working.
- Different domains use different language (vocabulary, acronyms, common phrases, etc.), so a domain-specific corpus can be analyzed and modeled better than one that contains documents from different domains.
- For example, if you are working in the financial domain, your models should be able to recognize stock symbols, financial terms, and company names, which means that the documents in your corpus need to contain these entities.

Text Data Sources

Web Scraping

RSS Feeds

APIs

Text Files

Text Data Sources

- Files - straightforward if plain text files; more complex otherwise.
- APIs - usually structured JSON format which is relatively easy to parse; often requires authentication and sometimes payment.
- RSS - standard XML format used primarily by blogs and news sites, which sometimes only provide summary of content.
- Web Scraping - HTML format usually with some styling, which can be inconsistent across different websites.

Considerations for Text Data Acquisition

- One important thing to consider when acquiring text data is the amount of structure that the source of your data will have.

- Some data sources, such as APIs, have significantly more consistent structure (ex. JSON) than other data sources, such as web pages.

- More structured data sources are generally preferable, but often there are fewer of them available than unstructured data sources.

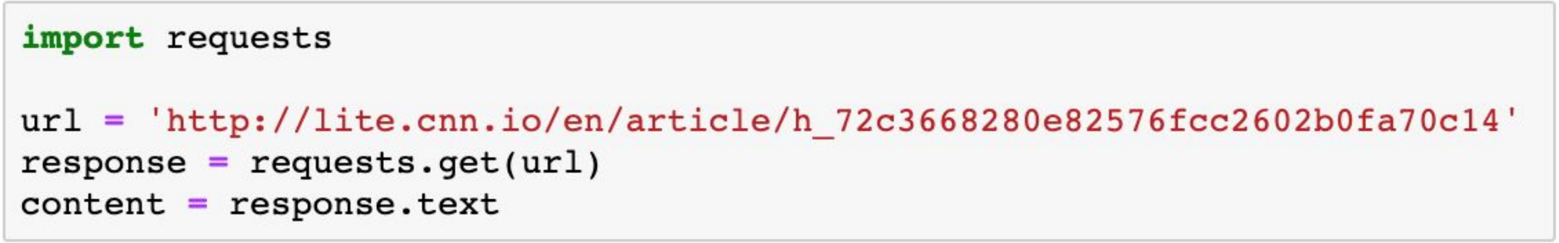
- Trade-off between structure and availability of information.

Web Scraping

- Web scraping refers to the automated extraction of specific information from a web page - often a page’s text content but may also include headers, published date, links, etc.
- Most web pages are formatted as HTML, but there are a range of layouts and styles that make web scraping across multiple sites challenging.
- Additionally, ensure when web scraping to rate limit and abide by the allowances in a website’s robots.txt file.

Acquiring Text Data via Web Scraping

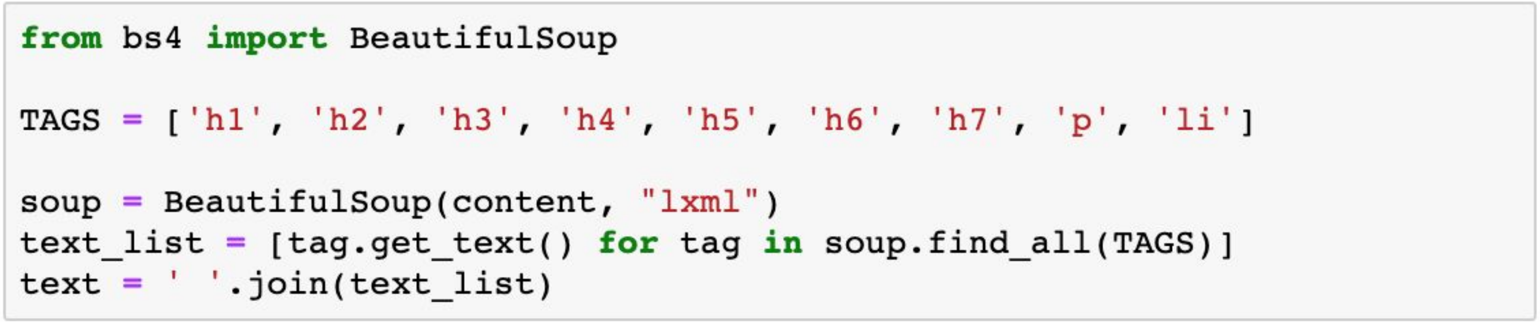
- We can scrape the content from a page using the *requests* library in Python.
- First, we need to specify the URL of the page we want to scrape.
- Then we call the `get` method and pass it the URL.
- Finally, we call the `text` method to extract the HTML content.



```python
import requests

url = 'http://lite.cnn.io/en/article/h_72c3668280e82576fcc2602b0fa70c14'
response = requests.get(url)
content = response.text
```

Cleaning Scraped HTML

- Strip away HTML elements and just keep the text. There are multiple ways to do this - below is an example using *BeautifulSoup*.
- First we define the HTML tags that we know contain text.
- Then we parse the content with BeautifulSoup, retrieve the text from all those tags, and combine the results together into a single chunk of text.



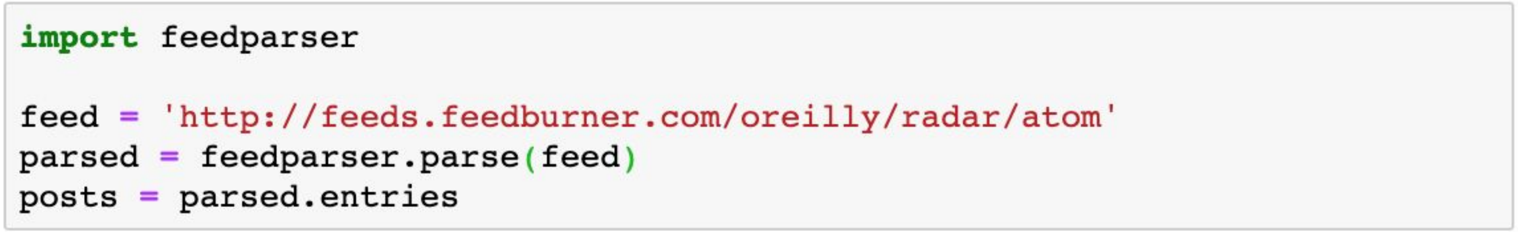
```python
from bs4 import BeautifulSoup

TAGS = ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'p', 'li']

soup = BeautifulSoup(content, "lxml")
text_list = [tag.get_text() for tag in soup.find_all(TAGS)]
text = ' '.join(text_list)
```

RSS (Really Simple Syndication)

- Standardized XML format for syndicated text data that is primarily used by blogs, news sites, and other online publishers who publish multiple documents (posts, articles, etc.) using the same general layout.
- Formatted more consistently than text on a regular web page - as a content feed or series of documents arranged in the order they were published.
- Content owners whose revenue depends heavily on serving advertisements have an incentive to display only summary text via RSS to encourage readers to visit their website to view both the full content and the ads.

Acquiring Text Data From RSS Feeds

- Python's *feedparser* library is great for retrieving and parsing RSS feeds.
- After importing feedparser and obtaining the URL for the RSS feed we want to parse, we simply call feedparser's `parse` method and pass it the feed URL.
- From there, we can obtain the posts by calling the `entries` method and then parse each entry to get the exact piece of content we need.



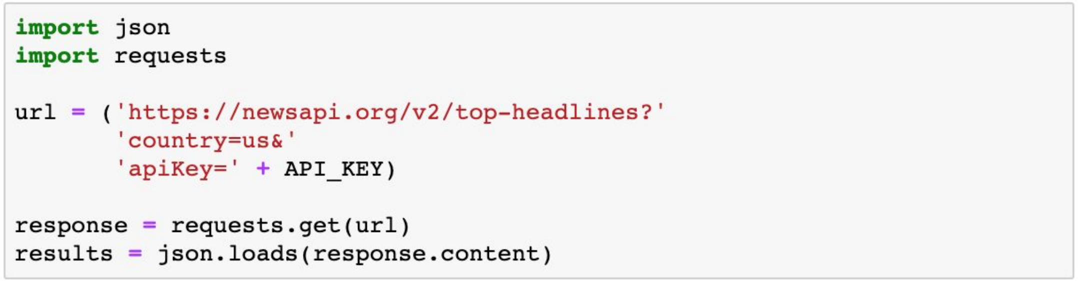
```
import feedparser

feed = 'http://feeds.feedburner.com/oreilly/radar/atom'
parsed = feedparser.parse(feed)
posts = parsed.entries
```

Application Programming Interfaces

- APIs are a great source for text data contained within applications. Application owners will often create APIs so that their applications can talk to other applications.
- An API is a set of programmatic instructions for accessing software applications, and the data that comes from APIs typically contains some sort of structure (such as JSON).
- This structure makes working with API data preferable to crawling websites and scraping content off of web pages.

Acquiring Text Data From APIs

- In Python, you can use the *requests* library to make API calls.
- You need to specify the API URL and sometimes provide an API key or other credentials in order to retrieve the data.
- We can then pass the call to the `get` method and receive a response, which we can then retrieve the contents of by calling the `content` method.
- If content returned is in bytes, we can use json.loads to convert to JSON.



```python
import json
import requests

url = ('https://newsapi.org/v2/top-headlines?'
       'country=us&'
       'apiKey=' + API_KEY)

response = requests.get(url)
results = json.loads(response.content)
```

Text Files

- The goal of data acquisition methods is to create a corpus of plain text files that contain just the text you want to work with.
- Web scraping will get you HTML, RSS will get you XML, and APIs will get you JSON - all of which you will need to parse, extract only the text content you want for each document, and save them to separate text files on disk.
- If you are starting out with text data in plain text files, your data acquisition work is already done and you can move on to ingesting the data via a corpus reader and building your corpus.

Creating A Corpus

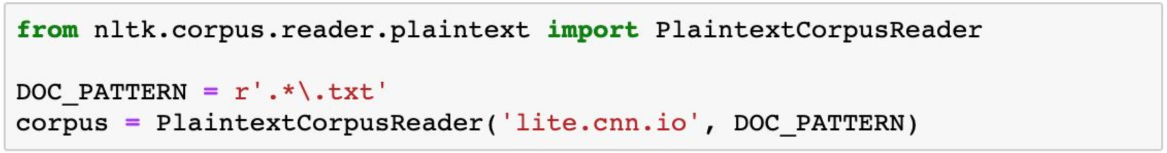
- A corpus is a collection of related documents that contain natural language.
- The simplest and most common method of organizing and managing a text-based corpus is to store individual documents in a file system on disk.
- Organizing the corpus into subdirectories allows it to be categorized by meta information such as date, source, subtopic, etc.

Creating A Corpus

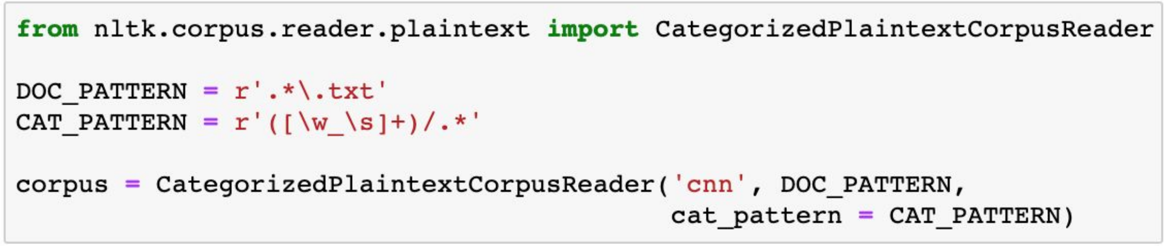
- By maintaining each document as its own file, corpus readers can quickly ingest different subsets of documents and processing can be parallelized, with each process ingesting a different subset of documents.
- Text corpora have the tendency to grow very quickly, so how you organize the documents on disk is important and should be thought about carefully.

Corpus Ingestion

- Once our corpus is properly organized, we can ingest it with an NLTK CorpusReader object very easily. Below are examples using NLTK's PlainTextCorpusReader and CategorizedPlainTextCorpusReader.



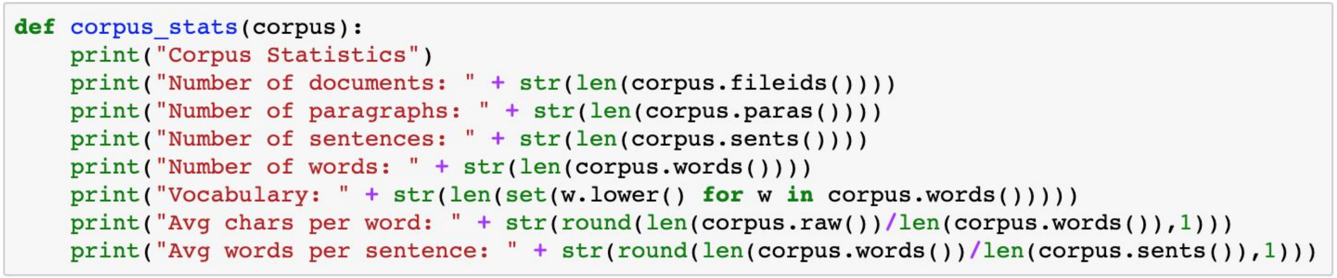
```
from nltk.corpus.reader.plaintext import PlaintextCorpusReader

DOC_PATTERN = r'.*\.txt'
corpus = PlaintextCorpusReader('lite.cnn.io', DOC_PATTERN)
```



```
from nltk.corpus.reader.plaintext import CategorizedPlaintextCorpusReader

DOC_PATTERN = r'.*\.txt'
CAT_PATTERN = r'([\w_\s]+)/.*'

corpus = CategorizedPlaintextCorpusReader('cnn', DOC_PATTERN,
                                          cat_pattern = CAT_PATTERN)
```

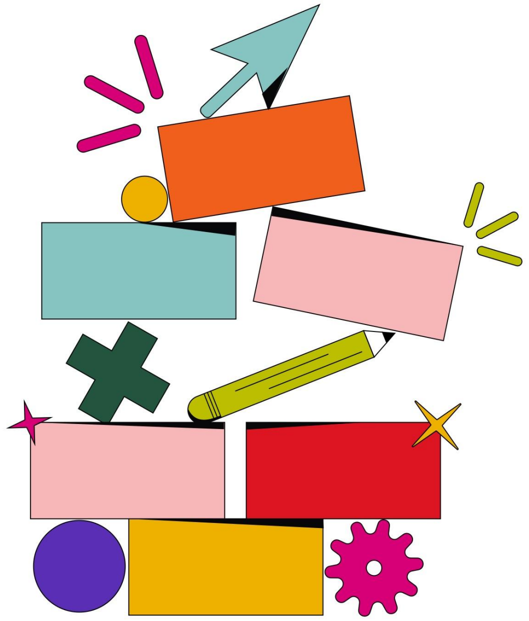
We Have A Corpus! Now What?

- Now that we have a corpus, we can start to look at some basic statistics that inform us about the content of the documents we have collected.
- Methods inside a CorpusReader object provide access to the documents (fileids), paragraphs (paras), sentences (sents), words (words), etc.
- Below is a function we can use to print out different corpus statistics.



```python
def corpus_stats(corpus):
    print("Corpus Statistics")
    print("Number of documents: " + str(len(corpus.fileids())))
    print("Number of paragraphs: " + str(len(corpus.paras())))
    print("Number of sentences: " + str(len(corpus.sents())))
    print("Number of words: " + str(len(corpus.words())))
    print("Vocabulary: " + str(len(set(w.lower() for w in corpus.words()))))
    print("Avg chars per word: " + str(round(len(corpus.raw())/len(corpus.words()),1)))
    print("Avg words per sentence: " + str(round(len(corpus.words())/len(corpus.sents()),1)))
```

Questions?



THINKFUL

Summary

Brief review, should call back to the objective and make the direct connection for how the objective has now been achieved.

- The importance of acquiring domain-specific text to build your corpus.
- Sources and considerations for acquiring text data.
- How to acquire text data from web scraping, RSS feeds, and APIs.
- How to use corpus reader objects in NLTK to create a corpus.
- How to calculate some basic corpus statistics.

Assignment

- See Jupyter Notebook.

THINKFUL

Thank You

Text Data Acquisition and Ingestion

Warm Up

- What are some sources where we can get text data?

High Level Agenda

- Acquiring Text Data
- Text Data Sources
- Considerations for Text Data Acquisition
- Acquiring Data via Web Scraping
- Acquiring Data from RSS Feeds
- Acquiring Data from APIs
- Creating a Corpus for NLP

Acquiring Text Data

- A Natural Language Processing project begins with acquiring the right kind of text data and building a custom corpus that contains structural and contextual features from the domain in which you are working.
- Different domains use different language (vocabulary, acronyms, common phrases, etc.), so a domain-specific corpus can be analyzed and modeled better than one that contains documents from different domains.
- For example, if you are working in the financial domain, your models should be able to recognize stock symbols, financial terms, and company names, which means that the documents in your corpus need to contain these entities.

Text Data Sources

Web Scraping

RSS Feeds

APIs

Text Files

Text Data Sources

- Files - straightforward if plain text files; more complex otherwise.
- APIs - usually structured JSON format which is relatively easy to parse; often requires authentication and sometimes payment.
- RSS - standard XML format used primarily by blogs and news sites, which sometimes only provide summary of content.
- Web Scraping - HTML format usually with some styling, which can be inconsistent across different websites.

Considerations for Text Data Acquisition

- One important thing to consider when acquiring text data is the amount of structure that the source of your data will have.
- Some data sources, such as APIs, have significantly more consistent structure (ex. JSON) than other data sources, such as web pages.
- More structured data sources are generally preferable, but often there are fewer of them available than unstructured data sources.
- Trade-off between structure and availability of information.

Web Scraping

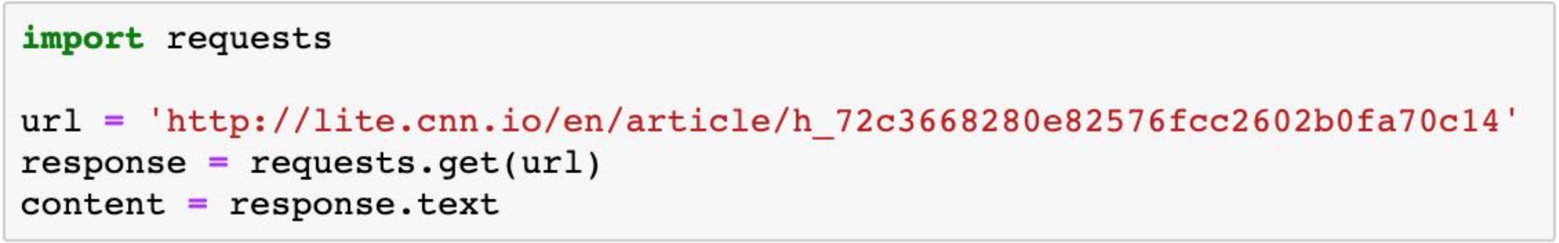
- Web scraping refers to the automated extraction of specific information from a web page - often a page's text content but may also include headers, published date, links, etc.

- Most web pages are formatted as HTML, but there are a range of layouts and styles that make web scraping across multiple sites challenging.

- Additionally, ensure when web scraping to rate limit and abide by the allowances in a website's robots.txt file.

Acquiring Text Data via Web Scraping

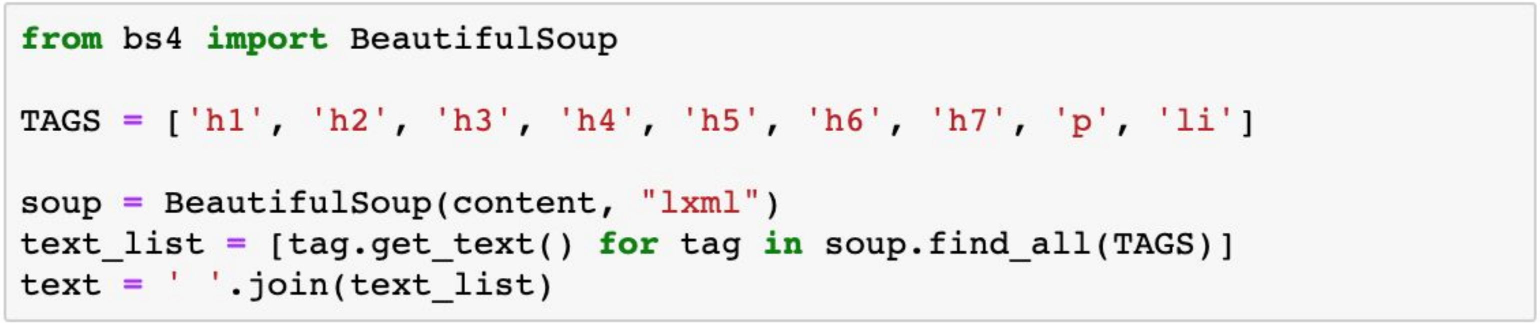
- We can scrape the content from a page using the *requests* library in Python.
- First, we need to specify the URL of the page we want to scrape.
- Then we call the `get` method and pass it the URL.
- Finally, we call the `text` method to extract the HTML content.



```python
import requests

url = 'http://lite.cnn.io/en/article/h_72c3668280e82576fcc2602b0fa70c14'
response = requests.get(url)
content = response.text
```

Cleaning Scraped HTML

- Strip away HTML elements and just keep the text. There are multiple ways to do this - below is an example using *BeautifulSoup*.
- First we define the HTML tags that we know contain text.
- Then we parse the content with BeautifulSoup, retrieve the text from all those tags, and combine the results together into a single chunk of text.



```python
from bs4 import BeautifulSoup

TAGS = ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'p', 'li']

soup = BeautifulSoup(content, "lxml")
text_list = [tag.get_text() for tag in soup.find_all(TAGS)]
text = ' '.join(text_list)
```

RSS (Really Simple Syndication)

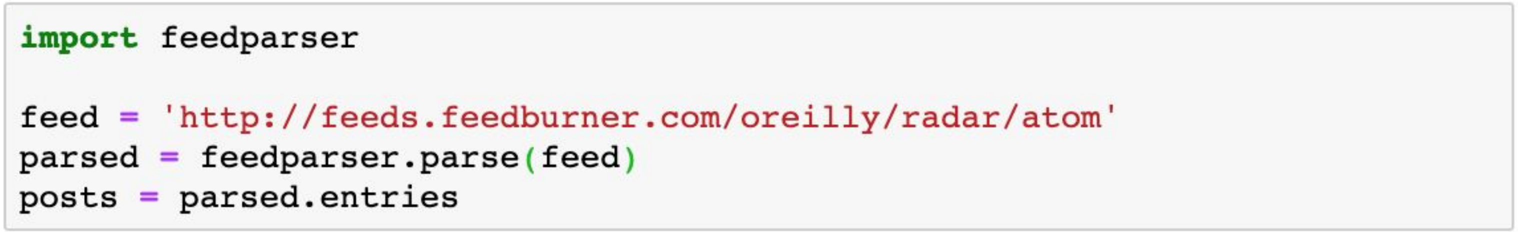
- Standardized XML format for syndicated text data that is primarily used by blogs, news sites, and other online publishers who publish multiple documents (posts, articles, etc.) using the same general layout.

- Formatted more consistently than text on a regular web page - as a content feed or series of documents arranged in the order they were published.

- Content owners whose revenue depends heavily on serving advertisements have an incentive to display only summary text via RSS to encourage readers to visit their website to view both the full content and the ads.

Acquiring Text Data from RSS Feeds

- Python's *feedparser* library is great for retrieving and parsing RSS feeds.
- After importing feedparser and obtaining the URL for the RSS feed we want to parse, we simply call feedparser's `parse` method and pass it the feed URL.
- From there, we can obtain the posts by calling the `entries` method and then parse each entry to get the exact piece of content we need.



```python
import feedparser

feed = 'http://feeds.feedburner.com/oreilly/radar/atom'
parsed = feedparser.parse(feed)
posts = parsed.entries
```

APIs (Application Programming Interfaces)

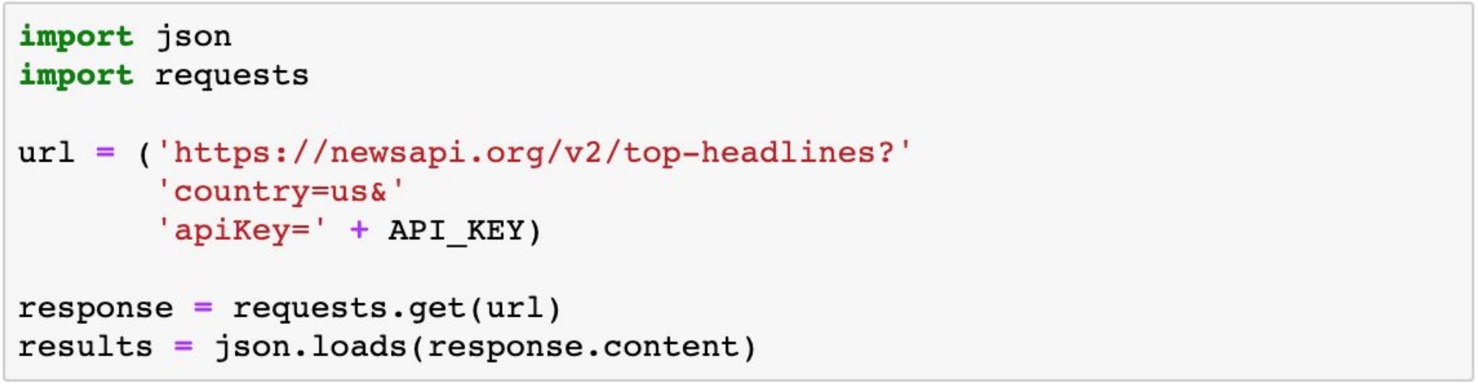
- APIs are a great source for text data contained within applications. Application owners will often create APIs so that their applications can talk to other applications.

- An API is a set of programmatic instructions for accessing software applications, and the data that comes from APIs typically contains some sort of structure (such as JSON).

- This structure makes working with API data preferable to crawling websites and scraping content off of web pages.

Acquiring Text Data From APIs

- In Python, you can use the *requests* library to make API calls.
- You need to specify the API URL and sometimes provide an API key or other credentials in order to retrieve the data.
- We can then pass the call to the `get` method and receive a response, which we can then retrieve the contents of by calling the `content` method.
- If content returned is in bytes, we can use json.loads to convert to JSON.



```python
import json
import requests

url = ('https://newsapi.org/v2/top-headlines?'
       'country=us&'
       'apiKey=' + API_KEY)

response = requests.get(url)
results = json.loads(response.content)
```

Text Files

- The goal of data acquisition methods is to create a corpus of plain text files that contain just the text you want to work with.

- Web scraping will get you HTML, RSS will get you XML, and APIs will get you JSON - all of which you will need to parse, extract only the text content you want for each document, and save them to separate text files on disk.

- If you are starting out with text data in plain text files, your data acquisition work is already done and you can move on to ingesting the data via a corpus reader and building your corpus.

Creating a Corpus

- A corpus is a collection of related documents that contain natural language.
- The simplest and most common method of organizing and managing a text-based corpus is to store individual documents in a file system on disk.
- Organizing the corpus into subdirectories allows it to be categorized by meta information such as date, source, subtopic, etc.
- By maintaining each document as its own file, corpus readers can quickly ingest different subsets of documents and processing can be parallelized, with each process ingesting a different subset of documents.
- Text corpora have the tendency to grow very quickly, so how you organize the documents on disk is important and should be thought about carefully.

Corpus Ingestion

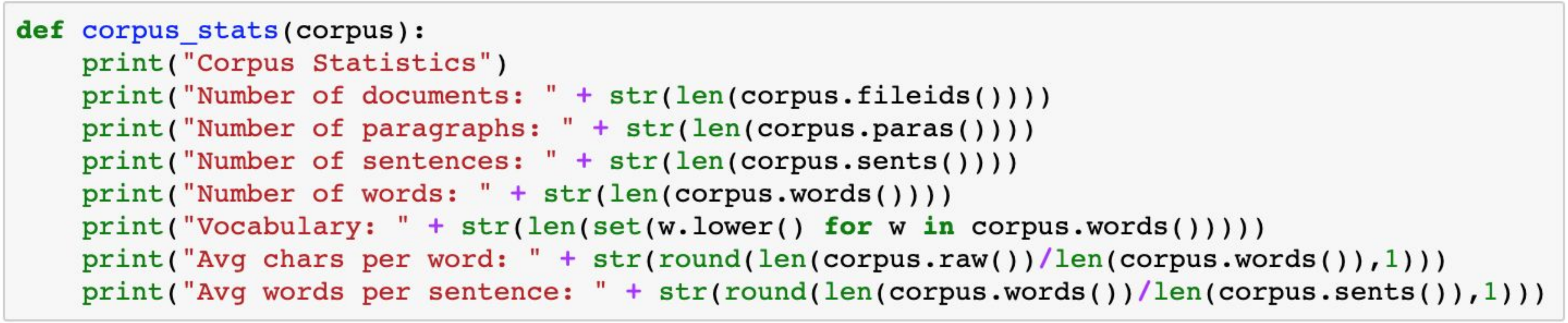
- Once our corpus is properly organized, we can ingest it with an NLTK CorpusReader object very easily. Below are examples using NLTK's PlainTextCorpusReader and CategorizedPlainTextCorpusReader.

```
from nltk.corpus.reader.plaintext import PlaintextCorpusReader

DOC_PATTERN = r'.*\.txt'
corpus = PlaintextCorpusReader('lite.cnn.io', DOC_PATTERN)
```

```
from nltk.corpus.reader.plaintext import CategorizedPlaintextCorpusReader

DOC_PATTERN = r'.*\.txt'
CAT_PATTERN = r'([\w_\s]+)/.*'

corpus = CategorizedPlaintextCorpusReader('cnn', DOC_PATTERN,
                                          cat_pattern = CAT_PATTERN)
```

We Have a Corpus! Now What?

- Now that we have a corpus, we can start to look at some basic statistics that inform us about the content of the documents we have collected.
- Methods inside a CorpusReader object provide access to the documents (fileids), paragraphs (paras), sentences (sents), words (words), etc.
- Below is a function we can use to print out different corpus statistics.



```python
def corpus_stats(corpus):
    print("Corpus Statistics")
    print("Number of documents: " + str(len(corpus.fileids())))
    print("Number of paragraphs: " + str(len(corpus.paras())))
    print("Number of sentences: " + str(len(corpus.sents())))
    print("Number of words: " + str(len(corpus.words())))
    print("Vocabulary: " + str(len(set(w.lower() for w in corpus.words()))))
    print("Avg chars per word: " + str(round(len(corpus.raw())/len(corpus.words()),1)))
    print("Avg words per sentence: " + str(round(len(corpus.words())/len(corpus.sents()),1)))
```

Questions?

Recap

In this session, we covered:

- The importance of acquiring domain-specific text to build your corpus.
- Sources and considerations for acquiring text data.
- How to acquire text data from web scraping, RSS feeds, and APIs.
- How to use corpus reader objects in NLTK to create a corpus.
- How to calculate some basic corpus statistics.

Assignment

- See Jupyter Notebook.