# Week 8: Working with Text Data from APIs

Fill out the cell below with your information.

# Student Name: Muhammad Rakibul Islam

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

## Objectives

This week you'll be working with the availablility of APIs and the data they provide, as well as learing about some of the steps that you need to take while cleaning and preparing text data for analysis.

In this notebook, you will:

- ingest data into a notebook from an API;
- use Pandas dataframes to find and organize data;
- uncover some of the challenges of working with data, including variation, multiple words with similar word stems, words with similar meanings, stopword control, and more;
- Consider the relationship between the data you are working with, the forms of "data cleaning" or "data scrubbing" that most text analysis piplines use, and the challenges that those methods present a feminist critical approach.

# Getting Started

As always, we need to begin by importing Python libraries that we know we will need to use. By this point, perhaps you are already familiar with some of them and know why we use them. Others may be less familiar to you. See if you can read through the list of imports and identify what each package does and why we will need it.

```
import nltk
import numpy as np
import pandas as pd
```

```
import urllib
import pprint
```

## Access data using an API

In the following exercise, you will import data from the Chronicling America API. You will set parameters for what content and keywords to pull in, then you will send the request to the server. After you import the data, you'll organize and clean up the JSON format--in other words, when you get your search results, it will come packaged in a file format, called JSON. We will ingest the JSON file, turn it into a dictionary, and then turn part of that dictionary into a Pandas Dataframe. All we're doing when we turn text data into a dataframe is organizing the metadata and the files into a format that can be used and acted upon in order to do other kinds of analysis.

To work with APIs, we will need to import a new library called "requests."

```
# Make the Requests module available
import requests
```

## What are APIs?

APIs are a set of routines that allow you to build from and interact with a software application. APIs make it possible for 2 software programs to work with each other. We will use APIs to pull data from applications. Many applications, like Twitter, Instagram, LinkedIn, and other applications have APIs.

In the following example, you are going to use the requests library to make an http request to the Open Movie Database (OMDb). We are going to use a security protocol called an API Key and request that the API return results to a query about the movie *The Princess Bride*.

First, go to the API Key generator on the OMDb website. Click the radio button next to FREE. Enter your email address, first, and last name, and then in the "use" section, you can write: "Completing an assignment for class." Then click Submit. It usually takes just a few moments for a confirmation email to arrive in your email box. Be sure to click on the second link in the email first to validate your key. The email will include a sequence of characters and numbers that you will use in this exercise. Once you have set up a key, you can begin the rest of the activity.

In order to retrieve data from the OMDb API using your API Key, we need to make a request using a communication protocol that is common on the internet: http. Essentially, what we will do is create a variable called `url` in which we will store an http request that is sent to the internet address www.omdbapi.com/. What follows the address, beginning with a question mark, is a query string. Query strings are not part of the URL syntax, but it tells the API (in this case) what your key is,and

then what information you would like to retrieve. In this case, the information is all the information included in the record with the title *The Princess Bride*.

Notice that the URL cannot contain spaces. Therefore, we insert the percent sign `%` where a space in the title might go. Alternately, we could put a + sign between each word.

```
url = 'http://www.omdbapi.com/?apikey=2359b80d&t=the%princess%bride'
```

```
# Create a variable movies and use the get method in requests to read in the
# response from the URL.
movies = requests.get(url)
# What datatype is the variable movies?
type(movies)
```

```
requests.models.Response
```

The result of the `requests.get()` method is specific to the requests library. In order to use the file, though, we need to convert the data from its current format into a JSON file. We do that by taking movies and applying the `.json' function.

```
# Take movies and turn it into json.
json_data = movies.json()
type(json_data)
```

```
dict
```

When you check the data type now, you will discover that the json file is saved as a dictionary, which is to say a series of "keys" and "values" saved in pairs.

Finally, we need to create a for loop so that we can go through the API results and print out the keys and their associated values. So, for every key and it's associated value in the json_data object, we look at each item in the dictionary and print the key, then a : and then the associated values.

```
for key, value in json_data.items():
  print(key + ':', value)
```

```
Title: The Princess Bride
Year: 1987
Rated: PG
Released: 09 Oct 1987
Runtime: 98 min
Genre: Adventure, Comedy, Family
Director: Rob Reiner
Writer: William Goldman
Actors: Cary Elwes, Mandy Patinkin, Robin Wright
Plot: A bedridden boy's grandfather reads him the story of a farmboy-turned-pira
```

```
Language: English
Country: United States
Awards: Nominated for 1 Oscar. 7 wins & 10 nominations total
Poster: https://m.media-amazon.com/images/M/MV5BYzdiOTVjZmQtNjAyNy00YjA2LTk5ZTAt
Ratings: [{'Source': 'Internet Movie Database', 'Value': '8.0/10'}, {'Source': '
Metascore: 77
imdbRating: 8.0
imdbVotes: 433,206
imdbID: tt0093779
Type: movie
DVD: 18 Jul 2000
BoxOffice: $30,857,814
Production: N/A
Website: N/A
Response: True
```

We know that we can search by title in the API because of the documentation on the OMDb website. Look under Usage and Parameters. In fact, the OMDb site includes examples, so that you can do a search and find the query string you need to get the result you are looking for. All the search parameters are listed here.

Another way to search for a particular movie is with its item ID in the IMDB database. You can find the item identifier by looking at the end of the URL when you search for a movie. For example, in this URL https://www.imdb.com/title/tt2906216/ we would use the item ID `tt2906216`. There are also additional arguments that can be used in the query string for OMDb to find the full plot description.

```
url = 'http://www.omdbapi.com/?apikey=2359b80d&i=tt2906216&plot=full'
dandd = requests.get(url)
json_data = dandd.json()
for key, value in json_data.items():
  print(key + ':', value)
```

```
Title: Dungeons & Dragons: Honor Among Thieves
Year: 2023
Rated: N/A
Released: 31 Mar 2023
Runtime: 134 min
Genre: Action, Adventure, Fantasy
Director: John Francis Daley, Jonathan Goldstein
Writer: Michael Gilio, John Francis Daley, Chris McKay
Actors: Chris Pine, Michelle Rodriguez, Regé-Jean Page
Plot: A charming thief and a band of unlikely adventurers embark on an epic ques
Language: English
Country: United States
Awards: N/A
Poster: https://m.media-amazon.com/images/M/MV5BZjAyMGMwYTEtNDk4ZS00YmY0LThhZjUt
Ratings: []
Metascore: N/A
imdbRating: N/A
imdbVotes: N/A
imdbID: tt2906216
```

```
Type: movie
DVD: N/A
BoxOffice: N/A
Production: Paramount Pictures
Website: N/A
Response: True
```

# Chronicling America

For this activity, we are going to use the Chronicling America API, which is created and maintained by the Library of Congress. Chronicling America is an archive of digitized newspapers from across the United States that are not under copyright protection by another digitization vendor. It is part of the National Digitial Newspaper project funded by the National Endowment for the Humanities. There are more than 140,000 newspaper titles included in the collection.

If you're interested in some of the critiques of the Chronicling America project, you might want to read Benjamin Fagen's article "Chronicling White America." (Fagan, Benjamin. "Chronicling White America." American Periodicals: A Journal of History & Criticism, vol. 26 no. 1, 2016, p. 10-13. Project MUSE https://muse.jhu.edu/article/613375.)

```
# Create a variable called 'api_search_url' and give it a value
api_search_url = 'https://chroniclingamerica.loc.gov/search/pages/results/'
```

```
# This creates a dictionary called 'params' and sets values for the API's mandatory pa
# The parameters are drawn from the API documentation which describes the fields in th
params = {
    'proxtext': 'poetry' # Search for this keyword -- feel free to change!

}
```

```
# This creates a dictionary called 'params' and sets values for the API's mandatory pa
# The parameters are drawn from the API documentation which describes the fields in th
params = {
    'proxtext': 'politics' # Search for this keyword -- feel free to change!

}
```

(Later on, you will be asked to return to the above cell and change the search parameters. You do this by replacing `poetry` with `yourterm`.)

```
# The following line adds a value for 'encoding' to our dictionary
params['format'] = 'json'
```

```
# Let's view the updated dictionary
params
```

```
{'proxtext': 'politics', 'format': 'json'}
```

```
# The next line uses the requests package that we imported above to pull data from the
# and stores the result in a variable called 'response'
response = requests.get(api_search_url, params=params)

# We use a print statement to show us the url that we are sending to the API
print('Here\'s the formatted url that gets sent to the ChronAmerca API:\n{}\n'.format(

# It's nice to have some feedback about the status of the API response
# to make sure there were no errors. The following checks to see that there are no err
if response.status_code == requests.codes.ok:
    print('All ok')
elif response.status_code == 403:
    print('There was an authentication error. Did you paste your API above?')
else:
    print('There was a problem. Error code: {}'.format(response.status_code))
    print('Try running this cell again.')
```

```
Here's the formatted url that gets sent to the ChronAmerca API:
https://chroniclingamerica.loc.gov/search/pages/results/?proxtext=politics&forma

All ok
```

```
# We are going to take the Chronicling America API's JSON results and turn them into a
data = response.json()
```

The request that we made was for data formatted in JSON, which means Javascript Object Notation. JSON is a structured way of organizing information and it can be converted into a Python Dataframe; however, it's not always easy for a human to read. We're going to use another package called Prettify to use indentation and color to help make the JSON a little more understandable to the human reader. We're also using a json library and a library called Pygments to add some colour to the output.

```
# Let's prettify the raw JSON data and then display it.

# We're using the Pygments library to add some colour to the output, so we need to imp
import json
from pygments import highlight, lexers, formatters
import pprint

# This uses Python's JSON module to output the results as nicely indented text
formatted_data = json.dumps(data, indent=2)
```

```
# This colours the text
highlighted_data = highlight(formatted_data, lexers.JsonLexer(), formatters.TerminalFo

# And now display the results
print(highlighted_data)
```


```
{
  "totalItems": 5663942,
  "endIndex": 20,
  "startIndex": 1,
  "itemsPerPage": 20,
  "items": [
    {
      "sequence": 50,
      "county": [
        "Hennepin"
      ],
      "edition": null,
      "frequency": "Daily",
      "id": "/lccn/sn83045366/1906-11-04/ed-1/seq-50/",
      "subject": [
        "Minneapolis (Minn.)--Newspapers.",
        "Minnesota--Minneapolis.--fast--(OCoLC)fst01204260"
      ],
      "city": [
        "Minneapolis"
      ],
      "date": "19061104",
      "title": "The Minneapolis journal. [volume]",
      "end_year": 1939,
      "note": [
        "Archived issues are available in digital format as part of the Library o
        "Available on microfilm from the Minnesota Historical Society, and the L
        "Merged with: Minneapolis star (Minneapolis, Minn. : 1928 : Home ed.), to
        "On Sundays published as: Sunday journal, <Nov. 3, 1918>.",
        "Weekly children's supplement called The journal junior published Jan. 1
      ],
      "state": [
        "Minnesota"
      ],
      "section_label": "Part VII, Comic Section",
      "type": "page",
      "place_of_publication": "Minneapolis, Minn.",
      "start_year": 1888,
      "edition_label": "",
      "publisher": "Journal Print. Co.",
      "language": [
        "English"
      ],
      "alt_title": [
        "Evening journal",
        "Journal junior",
        "Minneapolis evening journal",
        "Sunday journal"
      ],
      "lccn": "sn83045366",
```

```
  "country": "Minnesota",
  "ocr_eng": "mifslSIYENSENYANITOR*GETS INTO POLITICS\nWflmmfmmimmmfflmvMwmT
  "batch": "mnhi_dunbar_ver01",
  "title_normal": "minneapolis journal.",
  "url": "https://chroniclingamerica.loc.gov/lccn/sn83045366/1906-11-04/ed-1
  "place": [
    "Minnesota--Hennepin--Minneapolis"
  ],
```


```
type(json_data)
```

```
dict
```

## Reading text in a dataframe

Next, you will use what we learned about the data in the API using the keys. We're going to look into the "items" entry in the JSON file and create a dataframe using Pandas that pulls out the title, content, and year of publication for each of the items.

```
# Drill down into the API data to find the filds we want to pull out and work with
json_data = json.loads(formatted_data)
# print(json_data['items'])
cleaned_papers = []
for item in json_data['items']:
  # print(item['ocr_eng'])
  cleaned_papers.append({'title': item['title'], 'content': item['ocr_eng'], 'date': i
# print(cleaned_papers)
pp = pprint.PrettyPrinter(indent=4)

pp.pprint(cleaned_papers)
```

The output of the above cell will be quite long. Before turning in this assignment, please delete the cell above so the file you turn in is not difficult to read. Thank you!

```
type(cleaned_papers)
```

```
list
```

In the cell below, we will take the nested dictionary, which is also a json format, and we will convert it into a DataFrame.

```
df = pd.DataFrame.from_dict(json_data)
print(df.head(4))
```

```
   totalItems  endIndex  startIndex  itemsPerPage  \
0     5663942        20           1            20
1     5663942        20           1            20
2     5663942        20           1            20
3     5663942        20           1            20

                                               items
0  {'sequence': 50, 'county': ['Hennepin'], 'edit...
1  {'sequence': 29, 'county': ['Cook County'], 'e...
2  {'sequence': 92, 'county': [None], 'edition': ...
3  {'sequence': 23, 'county': ['Cook County'], 'e...
```

If we switch the layout of the dataframe, it becomes easier to see how the labels for the dataframe are different from the many items in the items observation. We can try to use the json method `normalize` to flatten out the file into columns.

```
df = pd.io.json.json_normalize(json_data)
df.columns
```

```
<ipython-input-41-c613c179d484>:1: FutureWarning: pandas.io.json.json_normalize
  df = pd.io.json.json_normalize(json_data)
Index(['totalItems', 'endIndex', 'startIndex', 'itemsPerPage', 'items'],
dtype='object')
```

```
dfpapers = pd.DataFrame.from_dict(json_data['items'])
dfpapers.head(5)
```

| | sequence | county | edition | frequency | id | subject |
|---|---|---|---|---|---|---|
| **0** | 50 | [Hennepin] | None | Daily | /lccn/sn83045366/1906-11-04/ed-1/seq-50/ | [Minneapolis (Minn.)--Newspapers., Minnesota--... |

```
for key in dfpapers:
    print(key)
```

```
sequence
county
edition
frequency
id
subject
city
date
title
end_year
note
state
section_label
type
place_of_publication
start_year
edition_label
publisher
language
alt_title
lccn
country
ocr_eng
batch
title_normal
url
place
page
```

The `.tail()` method will print out just the last (in this case) 6 items in the dictionary.

```
dates = dfpapers["date"]
dates.head()
```

```
0    19061104
1    19160516
2    19420920
3    19150325
4    19150325
Name: date, dtype: object
```

```
dfpapers["date"] = pd.to_datetime(dfpapers["date"])
dfpapers["date"]
```

```
0    1906-11-04
1    1916-05-16
2    1942-09-20
3    1915-03-25
4    1915-03-25
5    1912-11-09
6    1922-01-12
7    1913-07-02
8    1913-10-09
9    1906-07-13
10   1913-11-25
11   1913-12-10
12   1916-03-07
13   1945-05-14
14   1917-02-14
15   1913-06-09
16   1932-11-09
17   1913-07-02
18   1916-06-28
19   1915-02-15
Name: date, dtype: datetime64[ns]
```

The `shape()` method will show how many rows and how many columns are in your dataframe.

```
dfpapers.shape
```

```
(20, 28)
```

```
dfpapers.describe()
```

| | sequence | end_year | start_year |
|---|---|---|---|
| **count** | 20.000000 | 20.000000 | 20.000000 |
| **mean** | 20.850000 | 2329.550000 | 1903.450000 |
| **std** | 22.264617 | 1805.286957 | 19.345814 |
| **min** | 1.000000 | 1912.000000 | 1854.000000 |
| **25%** | 4.000000 | 1917.000000 | 1901.750000 |
| **50%** | 19.500000 | 1917.000000 | 1911.000000 |
| **75%** | 27.000000 | 1933.000000 | 1911.000000 |
| **max** | 92.000000 | 9999.000000 | 1938.000000 |

## Reflection and Writing

In this exercise, you queried an API from Chronicling America and pulled in files that included the search term "poetry." Those files, then, reshaped and made slightly more tidy by highlighting the "keys" to the dictionary, and then taking one small section of the dictionary and turning it into a dataframe.

Look back over the notebook and do the following:

1. Return to the section "Reading text in a dataframe." Read through some of the entries. Create a new text cell and explain what kind of "data cleaning" you would recommend to prepare the text for analysis. How can you take into consideration Munoz and Rawson's article? What makes the data "messy"? What messiness should remain? What messiness should be repaired? What messiness should be removed?
2. Go to the top of the Chronicling America section. Make a copy of the search query and try replacing the term "poetry" (the parameter of the search argument) with another one. What were the results?
3. What changes when you rerun the activity besides the results? Do you need to make any changes to the next cells for them to run?

## Response:

1) I find it a little difficult to suggest exactly what sorts of "data cleaning" I would suggest particularly more so because I have not fully understood what sort of analysis we would be making from the dataset as the cleaning can rely a lot on what we are trying to find.

But some of the the things that came to mind are that when the database is of digitized newspapers, does it also not contain advertisements or other forms of paid marketing elements. If so, are they identifiable? Based on the analysis performed, it may or may not make sense to remove them if they can be identified.

Also the same piece can be published in different newspapers. In that case maybe we need to remove duplicates or on the other hand keep track of how many times a certain piece appears so as to keep track of its popularity or importance of the topic.

Finally I would suggest generic cleaning like removing stop words.

However, since I did not properly get what the research question is, I am not fully sure of the cleaning needed here.

As Munoz and Rawson describe, the idea of "cleaning" implies that a dataset begins as being "messy" which suggests that there is an underlying order that the data does not fit into. Like there

are columns and rows that the data should fit into or certain standards that the data should maintain and fails to do so.

What "messiness" should remain, repaired or removed will very much depend on what context the data will be used for and the research questions being asked. I do believe that is the point that Munoz and Rawson was trying to explain.

Data that will directly hamper or interfere with the analaysis process completely should be repaired or removed. Maybe an article was not properly read in and you can tell the language does not make sense at all. Such data can be removed altogether. But on the other hand, what if an article was deliberately written in a style or manner that needed creative expression and such would make more sense in a poetry context as poetry might not fit the exact style and conformity that let's say an investigative article follows. So in that context, a piece of poetry may not just be removed if it does not fit convention as it is a valid form of the topic. In that case such data needs to be kept.

I do reiterate again that since I have not understood the research question(s) we mean to ask, I do not suggest any concrete "cleaning" steps but a generic understanding of what could be done.

2 & 3) I chose "politics" instead of "poetry" as the keyword being searched for. The results as such changed to those containing politics as a keyword instead of poetry.

I did not have to make any changesto the next cells to make them run as they were generic commands instead of anything specific so it all ran properly but just changed to results containing politics instead of poetry as the keyword.