Week 7: Jupyter Notebook Assignment - Working with Data

Fill out the cell below with your information.

Student Name: Muhammad Rakibul Islam

- Date:
- Instructor: Lisa Rhody
- Assignment due:
- Methods of Text Analysis
- MA in DH at The Graduate Center, CUNY

Objectives

The purpose of this notebook is to get some hands-on experience putting what you've seen in tutorials about importing and working with text in Python into practice. You'll also be asked to put the reading you've been doing all semester into conversation with the process of importing, cleaning, and preparing data.

The object of the notebooks this week is:

- To practice several ways of importing text into your Python environment to study;
- To become more familiar with various pipelines for cleaning and preparing data for text analysis;
- To consider the challenges that the availability and scarcity of data presents to the literary scholar (and to consider how other kinds of research might also need to address similar issues);
- To connect examples of real-world text analysis projects with the practical process of cleaning and preparing data.

Getting Started

We're going to start by importing some important libraries for working with text data.

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

```
import urllib
import pprint

nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

Importing Data

So far, we have worked with data during the Datacamp exercises, but that was a much more controlled environment. When you are actually doing your own text analysis project, you will have a much messier process. During this week's reading, you will have read several pieces about what cleaning takes place and some of the challenges that data presents when working with text. In particular, we're looking at text analysis from a humanities / litereary perspective; however, one might argue that these challenges are more similar to the text analysis one might perform in the social sciences or with non-fiction work than might appear to be the case on the surface.

In this lesson, we'll practice importing data:

- from a file already on your computer (using a directory path);
- from a file on the web using a URL request
- from a file on the web using Beautiful Soup.

Loading data from a flat file on your local computer

Before you get started, be sure to download this file onto your local computer and save it as herland.txt.

Next, we're going to import `herland.txt` using an upload function that is part of the google.colab Python package. This function will open a button under the cell that you can use to "Choose Files" from your local computer. Choose the `herland.txt` file and then upload it. The for loop below will print out what the name of the file is that you are saving to the Google Colab content folder.

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print("User uploaded file '{name} with length {length} bytes".format(
      name=fn, length=len(uploaded[fn])))
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in t

```
Saving herland.txt to herland.txt
User uploaded file 'herland.txt with length 328410 bytes
```

To find the file you just uploaded, look to the left side of this browser window. Click on the icon of a file folder. A directory structure should open. Click on the arrow next to `content` and you should see your uploaded file appear inside.

Then we're going to use a Python function `open()`. We'll use a `for` loop, which simply means that we'll do an action that repeats until we tell it to stop. The following code says that we want to `open` the file `herland.txt` so we can read it (argument `mode='r'`). Then we're going to close the file. When we do this, we're going to assign a variable name to the resulting data, which is now a string called `file`.

```
filename = 'herland.txt'
herland = open(filename, mode='r')
hertext = herland.read()
herland.close()
```

Another way to read the text from a file into Python is to use a "context manager." The following tells python that with the `herland.txt` file open, read in the text and create a variable called `file` to store the data. Then, the next line tells Python to print the new variable `file`. When you run the next cell, it is going to print out the entire text of *Herland*. That's a lot of text, so once you've done it, you can clear the cell's output and move on to the next cell.

```
# Here is how you print a string from a file without having to close the file using a

with open('herland.txt','r') as file:
    print(file.read())
```

```
# If you don't want to save the text of the file, but just want to peek into it to see

with open('herland.txt') as file:
    print(file.readline())
    print(file.readline())
    print(file.readline())
```

```
The Project Gutenberg EBook of Herland, by Charlotte Perkins Stetson Gilman



This eBook is for the use of anyone anywhere at no cost and with
```

What happens when you import a flat file?

The python function `type()` will return to you output that explains the data type you are working with. When you pass the new text object `herland` through the `type()` function below, what response do you get? The response will look different from other data types that you've used before. In this case, it is read in as a "file object." Remember that Python won't know how to handle data unless it fits a particular data type that the computer expects when passing a function to it. In the next input, we ask Python for the length of the file. This will throw an error. Why do you think that is?

```
# herland is a file object, not a string.
type(herland)
```

```
_io.TextIOWrapper
```

```
# since herland is a file object and not a string, you can't find the length of it.
len(herland)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-4105b81e8a20> in <cell line: 2>()
      1 # since herland is a file object and not a string, you can't find the le
----> 2 len(herland)

TypeError: object of type '_io.TextIOWrapper' has no len()
```

SEARCH STACK OVERFLOW

Observation:

Since herland is a file object and not a string, the len() function cannot determine the length which the code aims to do usually. That is why the error shows

Response here:

We had to go through a process to convert the file object to a string.

Looking at the cells below, which variable should return `type()` as a string? (The answer is in the cell below.)

```
# but hertext is a different datatype. How would you check?
type(hertext)
```

```
str
```

Response:

type() would return as string if the variable is a string which hertext is

Once you have a string, there are a number of functions that you can make use of. One of those is the `len()` command, which you can run below.

```
# How many characters are in the hertext string?
len(hertext)
```

```
315999
```

Once an object is recognized as a string, you can begin manipulating it. For example, you could count the number of times the sequence of characters "her" appear within the entire text of *Herland*.

```
hertext.count('her', 0, -1)
```

```
1244
```

The ability to count characters, words, n-grams, etc. means that we can also more easily target specific sections of the text. For example, when you print to your screen the opening of the herland file, you notice that it is accompanied with metadata. For the purposes of text analysis, what would be the advantages or disadvantages of removing the metadata associated with *Herland*?

Response:

When removing metadata, it can make the overall data cleaner as the metadata itself contains words which can hamper the analysis being done as those words can get counted. Also if some sort of a machine learning algorithm is run that can do stuff like sentiment analysis or other sorts of analysis, it can pickup the metadata text in that analysis which can give it an unacceptable result.

On the other hand, however, removing the metadata is removing otherwise relevant information that can provide context for analysis like publication information like publisher name, author name, publication date etc. that can be relevant to the analysis being done.

```
# What is happening at the beginning of the herland.txt file, though? We can check to
print(hertext[:660])
```

```
The Project Gutenberg EBook of Herland, by Charlotte Perkins Stetson Gilman

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: Herland

Author: Charlotte Perkins Stetson Gilman

Posting Date: June 25, 2008 [EBook #32]
Release Date: May 10, 1992
Last Updated: October 14, 2016

Language: English

Character set encoding: UTF-8

*** START OF THIS PROJECT GUTENBERG EBOOK HERLAND ***









HERLAND

by Charlotte Perkins Stetson Gilman



CHAPTER 1.
```

Working with a string is *more* helpful than simply working with a text object, but there are other things that we can do to the text to make it more easily manipulated in Python and NLTK. For example, when you're working with a string, it's not easy to count whole words. The NLTK word tokenizer function, however, will take a string and turn it into "tokens"--discrete segments of characters. Tokenized strings become a new data type--a list.

```
hertokens = nltk.word_tokenize(hertext)
type(hertokens)
```

```
list
```

A tokenized list can be called, acted upon, and manipulated differently than a string. If we call just the tokens that are in index positions 0-15, here is what you would get:

```
hertokens[:15]
```

```
['\ufeffThe',
 'Project',
 'Gutenberg',
 'EBook',
 'of',
 'Herland',
 ',',
 'by',
 'Charlotte',
 'Perkins',
 'Stetson',
 'Gilman',
 'This',
 'eBook',
 'is']
```

```
text1 = nltk.Text(hertokens)
```

```
type(text1)
```

```
nltk.text.Text
```

```
len(text1)
```

```
68514
```

```
text1[1000:1025]
```

```
['for',
 'men',
 '--',
 'of',
 'that',
 'they',
 'seemed',
 'sure',
 '.',
 'I',
 'told',
 'the',
 'boys',
 'about',
 'these',
 'stories',
 ',',
```

```
 'and',
 'they',
 'laughed',
 'at',
 'them',
 '.',
 'Naturally',
 'I']
```

Review

When you import text from a flat file that is saved on your local computer, what will you need to do in order to select parts of the text using an index?

Response:

As we see in the code above, when importing text from a flat file saved on the local computer, we will first have to tokenize the string which turns the string file into a new file type "list".

Once tokenized, we can select parts of the text using an index.

Ingesting data from a URL

Next, we're going to retrieve text directly from a URL with the `urlllib` package To do this, we're going to call the package `urllib` and specifically from that we're going to use `urlretrieve.` Next, we need to assign the text in the file to a variable. In this case, that variable is `url`. We're going to run `urlretrieve` with two parameters, the name of the URL you want to import (which you assigned to the variable `url` above, and the file name and extension. Here that is `203-0.txt.` If you pay attention to the output, you'll realize that you've imported the file as an object.

```
import urllib.request
```

```
from urllib.request import urlopen
from urllib.request import Request
url = 'https://www.gutenberg.org/files/203/203-0.txt'
uncletom = urlopen(url)
```

Question:

Using what you've learned so far, how would you figure out what data type the file `uncletom` is? Add a cell below and show how you would find the answer.

```
type(uncletom)
```

```
http.client.HTTPResponse
```

Response:

The code is already written but we get an HTTP Response type

Next, we're going to turn the text of Uncle Tom's Cabin into a list. A list is a mutable, ordered sequence of items. It can be indexed, sliced, and changed. Items in the list can be accessed through it's indexical placement.

```
dir(uncletom.read())
```

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'capitalize',
```

```
 'center',
 'count',
 'decode',
 'endswith',
 'expandtabs',
 'find',
 'fromhex',
 'hex',
 'index',
 'isalnum',
 'isalpha',
 'isascii',
 'isdigit',
 'islower',
 'isspace',
 'istitle',
 'isupper',
 'join',
 'ljust',
 'lower',
 'lstrip',
 'maketrans',
 'partition',
 'removeprefix',
```

```
words = uncletom.read().decode().split()
```

```
print(type(words))
```

```
<class 'list'>
```

Let's practice those steps again, but with a new file this time.

```
from urllib.request import urlopen

shakespeare = 'http://composingprograms.com/shakespeare.txt'

print( type(shakespeare) )
```

```
<class 'str'>
```

```
shakespeare = 'http://composingprograms.com/shakespeare.txt'
shakespeare = urlopen('http://composingprograms.com/shakespeare.txt')
print(type(shakespeare))
```

```
<class 'http.client.HTTPResponse'>
```

```
dir(shakespeare)
```

```
['__abstractmethods__',
 '__class__',
 '__del__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__enter__',
 '__eq__',
 '__exit__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__next__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '_abc_impl',
 '_checkClosed',
 '_checkReadable',
 '_checkSeekable',
 '_checkWritable',
 '_check_close',
 '_close_conn',
 '_get_chunk_left',
 '_method',
 '_peek_chunked',
 '_read1_chunked',
 '_read_and_discard_trailer',
 '_read_next_chunk_size',
 '_read_status',
 '_readall_chunked',
 '_readinto_chunked',
 '_safe_read',
 '_safe_readinto',
 'begin',
 'chunk_left',
 'chunked',
 'close',
 'closed',
 'code',
 'debuglevel',
```

```
       'detach',
       'fileno',
```

```
words = shakespeare.read().decode().split()
```

```
print(type(words))
```

```
<class 'list'>
```

```
title = words[0:3]
```

```
body = words[3:]
```

```
print(body[:10])
```

```
['Now', ',', 'fair', 'Hippolyta', ',', 'our', 'nuptial', 'hour', 'Draws', 'on']
```

Indexing Operator

Indexing operator ([ ]) selects one or more elements from a sequence. Each element of a sequence is assigned a number - its position or index. Index must be an integer value and is called inside a pair of square brackets.

The operation that extracts a subsequence is called **slicing**. When selecting more than one element **": operator"** is used with integer before and after it to indicate where to start and where to stop the index, respectively.

Python indexing starts at 0 and ends at (n-1), where n refers to the number of items in the sequence. The function "len" can be used to get the number of items in a list.

Negative indexing is also supported by Python. It can be done by adding "-" operator before the integer value.

```
n_words = len(body)
print( n_words )
```

```
980634
```

```
print( body[980634])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-59-92d037a2deff> in <cell line: 1>()
```

```
print( body[980633])
```

```
.
```

SEARCH STACK OVERFLOW

```
sub_body = body[:10]
print( sub_body)
```

```
['Now', ',', 'fair', 'Hippolyta', ',', 'our', 'nuptial', 'hour', 'Draws', 'on']
```

```
print( sub_body[:-2])
```

```
['Now', ',', 'fair', 'Hippolyta', ',', 'our', 'nuptial', 'hour']
```

```
print( sub_body[::2])    # gives every 2nd element
```

```
['Now', 'fair', ',', 'nuptial', 'Draws']
```

Python Syntax

Syntax refers to the structure of the language.

The end of the statement does not require semicolon or other symbol. After a statement is complete, the code is considered completed. However, using semicolon can allow you to execute two separate codes from the same line.

Indentation i.e. the whitespace matters in Python. A block of code is a set of statements that should be treated as a unit even when written in a new line. A code block in python are denoted by indentation. For example, in compound statements such as loops and conditionals, after the colon we must enter into a new line and add exactly four spaces to continue further. Whitespaces **within** the same line does not matter however.

Comments about codes can be made using hashtag #. anything written after # is ignored by the interpreter. Python does not have any syntax for multi-line comments.

```
sub_body_lowercase = []
for word in sub_body:
  sub_body_lowercase.append(word.lower())
  #print(sub_body_lowercase)
#print(sub_body_lowercase)
sub_body_lowercase
```

```
['now', ',', 'fair', 'hippolyta', ',', 'our', 'nuptial', 'hour', 'draws', 'on']
```

Importing an HTML file using an http: request

The previous two files that we imported were *plain text* files. In other words, there is little to no descriptive encoding. However, we can also use another module from the URLLIB package that is designed to import an .html file directly from the web. We can actually do this with just a few lines of code. First, we import the URLLIB package, and specifically the `request` module. We assign the URL we want to manipulate by assigning the URL to a variable. Next, we pass the URL through the urlopen.request function from the URLLIB package, and also at the same time "read" the file. The output of that string becomes the variable `html`. When we print the variable html, we discover that all of the HTML from the page has been pulled into the variable name. Unfortuantely, it doesn't look very clean.

```
# Now import the bibliography page from Colored Conventions in HTML
import urllib.request
anotherurl='http://coloredconventions.org/exhibits/show/bishophmturner'
```

```
html = urllib.request.urlopen(anotherurl).read()
print(html)
```

```
b'<!DOCTYPE html>\n<!--[if IE 6]>\n<html id="ie6" lang="en-US" xmlns:fb="https:/
```

If you are interested in doing text analysis of a webpage, and the only way to ingest the web page is with HTML included, what are things you might need to learn to do to separate the HTML tags from the text? Look at the code above and write a short description of what might need to stay and what might need to be extracted. Should the extracted data be preserved or discarded?

Response:

I am not exactly sure how this is done but what is needed is to somehow find and remove the HTML tags (like < p > and < h1 > etc.) and other markup elements (like comments people might be making on the code) from the webpage. I am sure there is some library that already does this, just that I am not familiar with it.

Now about whether the extracted data be preserved or discarded, I would say it depends on the context of the analysis. For example, let's say in our analysis it is relevant to know how the content is structure like we want to know what is the content in the heading, in the paragraph etc. In such a case, preserving data is relevant. Otherwise we can discard it.

Importing Data by Webscraping with BeautifulSoup

If you are interested in scraping data from the open web, BeautifulSoup is a Python pacakge worth exploring in detail. For our purposes here, though, we're going to consider how to use Beautiful Soup to turn "unstructured" data into "structured" data. As you read through this section, consider Muñoz and Rawson's argument about data cleaning. Is there a need for the data to stay unstructured? What is the value of cleaning?

Response:

The answer to the question on whether there is a need for data to stay unstructured it actually goes back to my previous response. Context and the end goal of an analysis matters. If there is a need to differentiate content by the HTML structure (like header, paragaph etc.) we need to keep the data unstructured because it becomes a relevant necessity. If such a context does not exist and we are purely looking at content, then by all means we can clean it up and make it structured.

The value of cleaning also goes back to the previous response and the general usage of data cleaning where if there is no context to keep data unstructured where preserving is necessary to draw relevant conclusions, it is better to clean it. This makes the data far easier to analyse especially when using tools like NLP which might not be able to pickup the HTML tags and other markups present.

```
import requests
from bs4 import BeautifulSoup
```

```
# Specify url: url
url4 = 'http://coloredconventions.org/press#scholarship'

# Package the request, send the request and catch the response: r
r = requests.get(url4)

# Extracts the response as html: html_doc
html_doc = r.text

# Create a BeautifulSoup object from the HTML: soup
soup = BeautifulSoup(html_doc)

# Prettify the BeautifulSoup object: pretty_soup
pretty_soup = soup.prettify()

# Print the response
print(pretty_soup)
```

```
<!DOCTYPE html>
<html lang="en" xmlns:addthis="https://www.addthis.com/help/api-spec" xmlns:fb="
 <head>
  <meta charset="utf-8"/>
  <meta content="IE=edge" http-equiv="X-UA-Compatible"/>
  <link href="https://coloredconventions.org/xmlrpc.php" rel="pingback"/>
  <script type="text/javascript">
   document.documentElement.className = 'js';
  </script>
  <link crossorigin="" href="https://fonts.gstatic.com" rel="preconnect"/>
  <style id="et-builder-googlefonts-cached-inline">
   /* Original: https://fonts.googleapis.com/css?family=Oswald:200,300,regular,5
  </style>
  <meta content="index, follow, max-image-preview:large, max-snippet:-1, max-vid
  <script type="text/javascript">
   let jqueryParams=[],jQuery=function(r){return jqueryParams=[...jqueryParams,r
  </script>
  <!-- This site is optimized with the Yoast SEO plugin v20.4 - https://yoast.co
  <title>
   Press &amp;amp; Notices - Colored Conventions Project
  </title>
  <link href="https://coloredconventions.org/about/press-notices/" rel="canonica
  <meta content="en_US" property="og:locale"/>
  <meta content="article" property="og:type"/>
  <meta content="Press &amp;amp; Notices - Colored Conventions Project" property="og
  <meta content="https://coloredconventions.org/about/press-notices/" property="
  <meta content="Colored Conventions Project" property="og:site_name"/>
  <meta content="2023-02-13T22:49:48+00:00" property="article:modified_time"/>
  <meta content="https://coloredconventions.org/wp-content/uploads/2018/03/CCP-P
  <meta content="summary_large_image" name="twitter:card"/>
  <meta content="Est. reading time" name="twitter:label1"/>
  <meta content="6 minutes" name="twitter:data1"/>
  <script class="yoast-schema-graph" type="application/ld+json">
   {"@context":"https://schema.org","@graph":[{"@type":"WebPage","@id":"https://
  </script>
  <!-- / Yoast SEO plugin. -->
  <link href="//s7.addthis.com" rel="dns-prefetch"/>
  <link href="//i0.wp.com" rel="dns-prefetch"/>
  <link href="https://coloredconventions.org/feed/" rel="alternate" title="Color
  <link href="https://coloredconventions.org/comments/feed/" rel="alternate" tit
  <meta content="Divi Child Mainsite v.1.0.0" name="generator"/>
  <link href="https://coloredconventions.org/wp-content/plugins/popup-by-supsyst
  <style id="wp-block-library-theme-inline-css" type="text/css">
   .wp-block-audio figcaption{color:#555;font-size:13px;text-align:center}.is-da
  </style>
  <link href="https://coloredconventions.org/wp-includes/js/mediaelement/mediael
  <link href="https://coloredconventions.org/wp-includes/js/mediaelement/wp-medi
  <style id="global-styles-inline-css" type="text/css">
   body{--wp--preset--color--black: #000000;--wp--preset--color--cyan-bluish-gra
.wp-block-navigation a:where(:not(.wp-element-button)){color: inherit;}
:where(.wp-block-columns.is-layout-flex){gap: 2em;}
.wp-block-pullquote{font-size: 1.5em;line-height: 1.6;}
  </style>
  <link href="https://coloredconventions.org/wp-content/plugins/aw-divi-image-ov
  <style id="divi-style-parent-inline-inline-css" type="text/css">
   /*!
```

```
Theme Name: Divi
Theme URI: http://www.elegantthemes.com/gallery/divi/
```

Compare the text imported using the "webscraping" method included with BeautifulSoup versus the option of importing the entire file using URLLIB.

Response:

The URLLIB method simply read in the HTML code without any form or structure as an endless text.

The BeautifulSoup method on the other hand read in the html code with the structure that HTML codes have which makes it easier to understand the overall code and how it was written.

Cleaning up Webscraped text

```
# Import packages
import requests
from bs4 import BeautifulSoup

# Specify url: url
url5 = 'http://coloredconventions.org/press#scholarship'

# Package the request, send the request and catch the response: r
r = requests.get(url5)

# Extract the response as html: html_doc
html_doc = r.text

# Create a BeautifulSoup object from the HTML: soup
soup = BeautifulSoup(html_doc)

# Get the title of Colored Conventions' webpage: ccc_title
ccc_title = soup.title

# Print the title of Colored Conventions' webpage to the shell
print(ccc_title)
```

```
<title>Press &amp;amp; Notices - Colored Conventions Project</title>
```

```
# Get Colored Conventions' text: ccc_text
ccc_text = soup.get_text()

# Print CCC's text
print(ccc_text)
```

```
Press & Notices – Colored Conventions Project
```

```
# Find all 'a' tags (which define hyperlinks): a_tags
a_tags = soup.find_all('a')

# Print the URLs to the shell
for link in a_tags:
    print(link.get('href'))
```

```
/about/book/
https://coloredconventions.org/
https://coloredconventions.org/
https://coloredconventions.org/about-conventions/
https://coloredconventions.org/about-conventions/
https://coloredconventions.org/about-records/
https://coloredconventions.org/about-conventions/submit-records/
https://coloredconventions.org/about-records/ccp-corpus/
https://coloredconventions.org/bibliography/
https://coloredconventions.org/exhibits/
https://coloredconventions.org/teaching/
https://coloredconventions.org/teaching/#teaching-partners
https://coloredconventions.org/curriculum/
https://coloredconventions.org/news/
https://douglassday.org/
https://douglassday.org/
https://coloredconventions.org/digblk/symposium-ccp-making-social-movement/
https://coloredconventions.org/news/mural-dedication-philadelphia/
https://coloredconventions.org/news/
https://coloredconventions.org/about/press-notices/
https://coloredconventions.org/about/videos/
https://coloredconventions.org/about/
https://coloredconventions.org/about/principles/
https://coloredconventions.org/about/team/
https://coloredconventions.org/about/team/committees/
https://coloredconventions.org/about/cv/
https://coloredconventions.org/about/speakers-agreement/
https://coloredconventions.org/digblk/
https://coloredconventions.org/about/using-two-sites/
https://coloredconventions.org/contact/
https://coloredconventions.org/donate/
http://muse.jhu.edu/journals/american_periodicals/v026/26.1.fagan.html
http://muse.jhu.edu/login?auth=0&type=summary&url=/journals/early_american_litera
https://muse.jhu.edu/login?auth=0&type=summary&url=/journals/american_periodicals
https://muse.jhu.edu/article/593047
http://common-place.org/article/column/colored-conventions-project/
http://common-place.org/book/the-colored-conventions-project-and-the-changing-sa
http://common-place.org/book/toward-meaning-making-in-the-digital-age-black-wome
http://common-place.org/book/the-colored-conventions-movement-in-print-and-beyon
http://common-place.org/book/convention-minutes-and-unconventional-proceedings/
http://common-place.org/book/liberating-history-reflections-on-rights-rituals-an
http://digitalhumanitiesnow.org/2016/03/digital-rudisell-of-the-colored-conventi
https://dmlcentral.net/digitally-improving-historical-knowledge/
https://www.forbes.com/sites/drsarahbond/2017/10/20/how-is-digital-mapping-chang
https://www.brown.edu/academics/public-humanities/news/2017-02/put-it-digital-wr
```

http://www.delawareonline.com/story/news/education/2017/02/15/ud-group-celebrate
http://technical.ly/delaware/2016/12/13/colored-conventions-project-mla-award/
http://www1.udel.edu/udmessenger/vol24no3/digital/vol24no3/index.html#p=20
http://blog.historians.org/2016/12/uncovering-activism-engaging-students-colored
http://www.phillytrib.com/commentary/resurrect-philly-s-black-economic-educationa
http://www.nytimes.com/2016/08/05/arts/design/colored-conventions-a-rallying-poi
http://www.slate.com/blogs/the_vault/2015/12/21/some_neat_new_digital_history_pr
http://blogs.loc.gov/digitalpreservation/2015/09/cultural-institutions-embrace-c
http://www.electrostani.com/2015/09/the-archive-gap-race-canon-and-digital.html
http://delawarepublic.org/post/history-matters-colored-conventions
https://slis.wisc.edu/wp-content/uploads/2016/02/2015_spring_jottings.pdf
http://www.thefacultylounge.org/2015/03/black-originalism-part-3-the-syracuse-co
https://web.archive.org/web/20160428054608/http://infospace.ischool.syr.edu/2015

Questions for reflection

Explain what the value is of importing HTML files using BeautifulSoup. How does this relate to the concerns that Rawson and Muñoz raise in their article? Are there times when you might want to keep the HTML?

Response:

From what I can tell so far, BeautifulSoup enables the ability to import HTML files with the structure that its supposed to have which makes it east to read the HTML code and would hence make it easier to manuever the data, extract needed parts and portions and make the process of analysis simpler.

I for one don't think BeautifulSoup initially causes any concerns with the concerns Rawson and Munoz raise. Initially we are reading in the HTML as it is supposed to be with the structure and all.

Now once we start cleaning it up that is when issues can rise that cleaning out data that might otherwise provide value when the context of the analysis calls for it.

About wanting to keep the HTML. Again. Depends on context.

Consider Rob Kitchin's criteria of "good data." Would these datasets satisfy his definition of "good data"? Why or why not? What kinds of questions could one ask about the Colored Conventions Project using what you've learned here?

Response:

I did have a respones I had written when doing this notebook for this question but for some reason I might have mistakenly not saved it (or pasted in properly since I did so some of the responses first

in word and then copy-pasted here).

I apologize for this inconvenience but I cannot find that response.