Digital Humanities
English 498, Spring 2025
Singh

Adapting Melanie Walsh's *Cultural Analytics and Python* Textbook for Google Colab.

https://melaniewalsh.github.io/Intro-Cultural-Analytics/welcome.html

This document is open to comments.

Deep's note to self for 4/10: Open these tabs
–My Walsh Colab testing notebook is here.
–Google Drive folder with helpful files here.
–HathiTrust file cleaner code here.

Deep's note to self for 4/15:

–Melanie Walsh's chapter on Network diagrams
–Colab notebook with simple network diagram function at the bottom.
–Google Drive folder with helpful files here.
–Walsh's chapter on Sentiment Analysis in Python

Thursday 4/10

Today, let's explore some of the opening chapters of Melanie Walsh's 2021 online textbook teaching the basics of coding for literary studies scholars. This is an especially helpful guide since all of the examples and topics are essentially aligned with common DH sub-fields – text analysis (including sentiment analysis, topic modeling, word frequency, etc.), social network analysis, and mapping. The examples are often familiar (Charlotte Perkins Gilman's "The Yellow Wallpaper") or fun (the lyrics to Beyonce's "Lemonade" album).

For our brief crash course, you could in theory work through a big chunk of this textbook just by copying and pasting code examples – and not worry too much about learning the principles. Since we have very limited time, that might actually be good – it would allow us to see results without getting bogged down in concepts and vocabulary that would normally take a fair amount of time to learn. (Remember that we're taking a week to tap

into a syllabus that would normally span an entire semester!) The goal is partly just to demystify coding – it's not magic – and maybe also to give you the sense that if you wanted to try running something in Python you could be empowered to do that. (I would *not* suggest that after just a couple of sessions, any of us would be ready to write our own sophisticated scripts in Python.)

That said, if you think you might want to do more with this in the future, it might be worth your time to read the chapters on things like **Variables**, **Data Types**, and **Functions** to get a sense of what those things are and how they work. If you might want to tweak existing bits of code to do slightly different things than the examples allow, or write something from scratch, I would recommend taking the extra time.

(Note: another possible route forward from here might be to go to Google or even **generative AI** for help with snippets of code. There are also sites like StackOverflow that specialize in (human) coders answering questions on how to do something.

(It might also be worth mentioning that there are powerful software packages used in DH like **BookNLP** that only run in Python.)

Instead of using Jupyter notebooks (recommended by Walsh in the opening chapters of the textbook), we'll do the exercises in the textbook using Google Colab, a newer tool that allows you to work in Python without installing anything on your computer. Earlier, I gathered from a few of you that installing things on your personal laptops might not be ideal… Below, I'll tell you which sections from the opening chapters of the textbook you could theoretically skip or ignore.

(The Colab approach also helps us avoid the Mac/Windows problem – it's running on the cloud so all the commands are the same irrespective of what type of OS you're using.)

I have been testing it and it works pretty well. The upside is that your Colab notebooks will be shareable and saved in the cloud, so nothing will crash your computer – and if you're working on something and want someone else to see it that's easy to do. (I will share my own Colab notebooks with you all.)

The downside might be that while the code you write stays put and is automatically saved (like other Google Drive/Google cloud services), the *files* you upload to Colab in the cloud folder don't stay there – so if you run a Python program to create a certain file or transform a certain file, you'll want to download it again at the end of the session.

One other thing, admittedly some of the basic functions we'll learn will actually be things you can do in Voyant-Tools fairly easily without coding. However, if you start to do things in Python, you have much more power and control over the process. For instance, we have talked this spring about gendered language in fiction. If we code the queries ourselves, we can potentially have much more control over how we analyze gendered language than if we rely on someone else's choices with respect to things like stopwords.

Opening chapters:

Just read for now:
https://melaniewalsh.github.io/Intro-Cultural-Analytics/How-To-Interact-With-This-Book.html
(We're going to skip the part about installing Jupyter Notebooks)

https://melaniewalsh.github.io/Intro-Cultural-Analytics/course-schedule-2021.html
(This is the syllabus from Melanie Walsh's spring 2021 course at U-W. Note the readings – some really interesting materials in there alongside familiar texts like *Data Feminism.* There are also some coding vocabulary segments you might want to click on. I particularly liked this blog post by Aditya Mukerjee, and would probably include it on a future syllabus.)

---

1. The Command Line chapter – via Colab.

https://melaniewalsh.github.io/Intro-Cultural-Analytics/01-Command-Line/01-The-Command-Line.html

We're going to skip a lot of this, though if you're curious to try installing a Python coding environment on your own computer, feel free to follow along.

Learning how to access the Command Line (via Terminal in MacOS and Shell/Powershell in Windows) is still valuable, as it helps you get past the limits we sometimes experience with the Graphical User Interface (GUI). Understanding **Filepaths** is a particularly important topic, which unfortunately is often overlooked in

computer literacy today, especially since we are so used to being able to easily search for files. (We talked about this stuff a little with the text editing / regular expressions unit)

(Within Filepaths, the distinction between Relative Path and Absolute Path is also worth understanding.)

If you're going to go along with me and use **Colab** (nothing to install) instead of Jupyter Notebooks, scroll down to "Working With Files and Texts" on this page of the textbook.

* * *

Getting started with Colab:

If you open **colab.google.com**, and then click on "+ New Notebook," you should be in Python.

Also, click on the file folder button on the left (under the key) and you'll see the files that are available in Colab right now. It should be mostly empty except for a folder called **sample_data.**

Based on Walsh's instructions in the Command Line chapter, try this command from the command line in Python:

```
!curl -O https://www.gutenberg.org/files/1952/1952-0.txt
```

(Push the "Play" button to execute. Hitting enter will not do anything)

(Also note there is an exclamation point at the beginning of the line. We'll need those whenever the command we're executing is not a Python command but a shell command. And don't worry right now if that doesn't make sense.)

If if worked, you should soon see a file on the left called 1952-0.txt. We just grabbed "The Yellow Wallpaper" from Project Gutenberg and saved it into a temporary folder on the cloud. At the end of the session, if we want to keep any files we create in that temporary folder, we should remember to download them to our computers.

[*Observation on Filepaths in Colab after teaching session on 4/10. Some students got confused about how to find the working default directory. You see quite a lot of unrelated directories if you click on the [..] button. However, if you don't see the default*

*folder on the left, you should be able to find your Colab cloud files in the **/content/** folder.*]

Let's also try grabbing Claude McKay's *Harlem Shadows* from Gutenberg the same way.

Since we're using a different approach to Python, we'll use a different way to rename files as well. We'll import a package called "os" that has a command called "rename."

```
!curl -O https://www.gutenberg.org/ebooks/64989.txt.utf-8
import os
os.rename("64989.txt.utf-8", "Harlem-Shadows.txt")
```

You can also of course rename using the GUI in the file folder on the left.

Trying out some commands:

Wordcount (wc):
Many of the other commands on the "Command Line" chapter of the textbook will work directly in Colab, though most of them will need the extra exclamation point at the beginning of the line.

```
!wc -w The-Yellow-Wallpaper.txt
```

Will give you a successful result (it shows 6103 words when I tried it)

The "head" and "tail" commands Walsh demonstrates in the Command Line chapter should also work just fine.

Grep:
The "grep" command is pretty cool. It should give you every line that has "yellow" in the story.

```
!grep "yellow" -n The-Yellow-Wallpaper.txt
```

```
133:yellow, strangely faded by the slow-turning sunlight.
556:had found yellow smooches on all my clothes and John's,
and she wished
```

```
582:There are always new shoots on the fungus, and new
shades of yellow all
586:It is the strangest yellow, that wallpaper! It makes me
think of all
587:the yellow things I ever saw—not beautiful ones like
buttercups, but
588:old foul, bad yellow things.
618:is the color of the paper! A yellow smell.
809:instead of yellow.
```

Try it with a different word to see what happens.

(Memorable line from 4/10: a student, after initially struggling to get this command to work, said, "I grepped!")

2. Next chapter in the Walsh – Programming in Python

https://melaniewalsh.github.io/Intro-Cultural-Analytics/02-Python/00-Python.html

A good and useful overview of why people use Python and what they use it for.
The Miriam Posner blog post about the gendered culture of coding is worth a read.

Walsh on why learn some programming:

- You can save time and energy by automating tasks
- You can collect and analyze cultural data from a different perspective, such as from a larger scale or a new angle
- You can gain more flexibility, customizability, and autonomy over your digital research projects
- You can better understand how data and programming languages shape contemporary society and culture

Walsh on why Python specifically:

- Python resembles the English language and is readable for those who know English
- Python is known to have a relatively smooth learning curve
- Python is a broadly used and popular language
- Python is increasingly good for data science

2a. Install Python.

(We're skipping all of this.)
https://melaniewalsh.github.io/Intro-Cultural-Analytics/02-Python/01-Install-Python.html

---

3. The Anatomy of a Python Script

https://melaniewalsh.github.io/Intro-Cultural-Analytics/02-Python/03-Anatomy-Python-Script.html

On this page, she starts to really get into it! Buckle up, we're going to do some coding.

One thing you'll notice is that there is a "Regular Expressions" function that works within Python called "re" – we're going to use it on this page.

Things should pretty much work ok for our Colab version of the code projects on this page, though the filenames are slightly different:

```
import re
from collections import Counter

def split_into_words(any_chunk_of_text):
    lowercase_text = any_chunk_of_text.lower()
    split_words = re.split("\W+", lowercase_text)
    return split_words

filepath_of_text =
"../texts/literature/The-Yellow-Wallpaper_Charlotte-Perkins-Gilm
an.txt"
number_of_desired_words = 40

stopwords = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours',
'ourselves', 'you', 'your', 'yours',
 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she',
'her', 'hers',
```

```
 'herself', 'it', 'its', 'itself', 'they', 'them', 'their',
'theirs', 'themselves',
 'what', 'which', 'who', 'whom', 'this', 'that', 'these',
'those', 'am', 'is', 'are',
 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
'having', 'do', 'does',
 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
'because', 'as', 'until',
 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against',
'between', 'into',
 'through', 'during', 'before', 'after', 'above', 'below', 'to',
'from', 'up', 'down',
 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',
'then', 'once', 'here',
 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more',
 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only',
'own', 'same', 'so',
 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don',
'should', 'now', 've', 'll', 'amp']

full_text = open(filepath_of_text, encoding="utf-8").read()

all_the_words = split_into_words(full_text)
meaningful_words = [word for word in all_the_words if word not
in stopwords]
meaningful_words_tally = Counter(meaningful_words)
most_frequent_meaningful_words =
meaningful_words_tally.most_common(number_of_desired_words)

most_frequent_meaningful_words
```

You can actually just paste all of the above into your Colab notebook then hit “Play” / Execute and see what happens. (**Note**: you should change the file path first – see my comment above next to filepath_of_text)

Down near the bottom of the chapter, you’ll see this variation:

```
with open("most-frequent-words-Yellow-Wallpaper.txt", "w")
as file_object:
```

```
file_object.write(str(most_frequent_meaningful_words))
```

This command takes the most-frequent-meaningful_words output and saves it as a file in the Colab Notebook. (Being able to do this would come in handy if we were working on lots of different text files.)

At the bottom of this page, Walsh tells us something important, which is that any sequence of code we put into the Colab notebook can also work as an independent "script."

In a text editor, you could take the same code and save it with the last name .py. That makes it a Python script. You could then run that script in any python environment from the command line. (We're not going to worry too much about that right now, but it's helpful to know in case you're ever looking for Python scripts online that do certain things – they'll come in the .py format.)

(One reason you might want to know that is that in many cases if you're looking for code online, it will come in the .py format. To translate that to Colab, you can just paste code from a .py file into Colab and it should run.)

---

4. Variables

https://melaniewalsh.github.io/Intro-Cultural-Analytics/02-Python/04-Variables.html

Not a lot of new stuff to do in this chapter, though it introduces a very important basic programming concept – the concept of variables.

This helps also to explain what the code we tried in the previous chapter actually does.

There are also some important tips about Variable names in this chapter.

5. Data Types

https://melaniewalsh.github.io/Intro-Cultural-Analytics/02-Python/05-Data-Types.html

Another chapter with a fundamental programming concept.

Reading / interpreting code.

Here is a simple script I put together – starting with a Google query: “python script to clean text files using regular expressions.”
Google’s Gemini AI gave me a sample script – which seemed better than the actual search results I looked at. I adapted it to fit some specific Regular Expressions issues I see when I clean files derived from HathiTrust.

In terms of decoding what this text does, note that there are comments built into the code that actually explain most of the individual lines.

https://colab.research.google.com/drive/1YTH4JRF5TQGBZSeZUW35Sf0ddqPECFFx?usp=sharing

```
import re

def clean_text(input_file, output_file):
    """
    Cleans a text file using regular expressions and saves the output to a
new file.

    Args:
        input_file (str): Path to the input text file.
        output_file (str): Path to the output text file.
    """
    try:
        with open(input_file, 'r', encoding='utf-8') as infile,
open(output_file, 'w', encoding='utf-8') as outfile:
            for line in infile:
                # Remove lines that start with ## HathiTrust specific mess
                line = re.sub(r'##.*\n', '', line)
                # Remove spaces before punctuation, a common problem with
scanned poetry
                line = re.sub(r'(.*) ([:,.?!;])', '\1\2', line)
                # Remove end of line hyphen
                line = re.sub(r'¬\n', '', line)
```

```
                # Remove page numbers
                line = re.sub(r'\d+\n', '', line)
                # Remove page numbers in parentheses
                line = re.sub(r'\(\d+\)\n', '', line)
                # Remove large block of blank space five lines at a time
                line = re.sub(r'\n\n\n\n\n', '', line)

                outfile.write(line + '\n')
        print(f"Text cleaning complete. Output saved to {output_file}")
    except FileNotFoundError:
        print(f"Error: Input file '{input_file}' not found.")
    except Exception as e:
         print(f"An error occurred: {e}")

if __name__ == "__main__":
    input_file_path = 'George Marion McClellan Poems 1895.txt'
    output_file_path = 'output_cleaned.txt'
    clean_text(input_file_path, output_file_path)
```

What, roughly, does this Python script do? Some parts you can ignore for now. But the core parts – the specific actions we're performing on a text file using regular expressions – are pretty simple.

The code works pretty well – I uploaded a messy HathiTrust file, and it gave me back a much cleaner file (which, admittedly, still needs additional work by hand).

(If we have time, try running the code yourself! You should be able to see my Colab page at the link above and make a copy. Try uploading McClellan book of poems, run the cleaner, then download the file it creates. Examine both and see what the program actually did)

There is a file (George Marion McClellan's "Poems") in the following Google Drive folder:

https://drive.google.com/drive/folders/1QpgUIOGu80BiCR4TFaXJHbXIc3ukOLrN?usp=sharing

Try pasting the above code into Colab (maybe open a new notebook) and running it with the text file in the folder above.

In case you're interested, one of Melanie's examples for working with data (the "Pandas" sections) involves a dataset involving film dialogue. The intro blog post to the film dialogue dataset is below.

Film dialogue by Age and Gender. 2000 screenplays analyzed by gender representation. Not surprisingly, Hollywood movies are predominantly centered around male voices.

https://pudding.cool/2017/03/film-dialogue/

Their dataset:

https://github.com/matthewfdaniels/scripts/

Melanie's chapter on how to work with (and clean up) the data in this dataset in Python is here:

https://melaniewalsh.github.io/Intro-Cultural-Analytics/03-Data-Analysis/03-Pandas-Basics-Part3.html

I also put the Pudding dataset in this folder for convenience:

https://drive.google.com/drive/folders/1QpgUIOGu80BiCR4TFaXJHbXIc3ukOLrN?usp=sharing

Download it to your computer, and then upload it back to the Colab working directory before trying some of the functions with the data Melanie suggests.

DAY 2 – Tuesday 4/15

We moved through things pretty efficiently last time around. So I'm adding sentiment analysis as one of the chapters we can look at as well!

I discovered one piece of the Network Analysis piece was broken (the interactive network diagrams made using Bokeh), and happily when I wrote Melanie about it, she sent along a fixed version. Hopefully that won't make things too confusing.

Goals for today:

1. Network Analysis. Refresh on what it is and why we might want to be able to do it. Follow Walsh's chapters on Network analysis / network diagrams in her chapter (note that the second page needed some fixes / updates & we have those).

Replicate the diagrams she has in her textbook, then
A) Try the same code with other validated datasets (*The Crisis*, Dora Marsden feminist network)

B) Try the same code with non-validated / messy datasets created by Deep. (Nella Larsen's *Passing*, Bram Stoker's *Dracula*)

> *Passing* is clean enough to create something
> The *Dracula* dataset will need more work before it gives us anything useful.

2. Walsh's Sentiment Analysis chapter – doing Sentiment Analysis in Python with VADER.

Network analysis workflow:

–Get lists of names in a text, using NER or annotating the text by hand.

> (We used NER a little differently in our 'mapping' unit – NER toolkits can extract different kinds of information from texts, including place names as well as character names. The software aims to decode character names contextually. Different NER algorithms have been developed, and they might have different rates of success.)

–Define a system for identifying connections between characters based on our own (human / sociological) sense of how social connections work.

> One crude way of doing this algorithmically: *we could count it as a connection when two characters appear in the same sentence.* (Another, more inclusive approach might be: when two characters appear in the same paragraph.) That

algorithmic approach is limited in lots of ways, one of which being that it won't give us much in terms of multiplicity or triangles (for instance, if a paragraph shows Clarissa's connection to both Sally Seton and Peter Walsh, our approach might not work).

–Think about weighting. This is important because things like repetition of contact could make a connection between two characters stronger. Other things that might matter might be: familial relationships, business partnerships, etc. These are all choices you might make in thinking about weighting in a network diagram; there is no standard algorithm or automated way of describing all of this.

(If we want to automate it, the simplest way to do that might simply be to weight characters with repeated interactions numerically. Six interactions → weight of 6)

–Create a spreadsheet with the characters (nodes) and connections (edges). The standard format for an "Edges" spreadsheet has "Source," "Target," and "Weight" as the third column.

–The spreadsheet likely will need cleaning and editing. For really big networks, scholars use tools like OpenRefine to reduce duplications, standardize names/ spellings, etc.

For our purposes, we can do this by hand with search and replace (I prefer to do this using a text editor with Regular Expressions).

Characters that show up on the sheet by just their first name or a nickname should have their names standardized so they only show up once in the network diagram.

At least the code I have been using to create character and connections lists sometimes has glitches that need to be fixed.

–Using network analysis software to create a graph. In Python, there is a package called NetworkX that does this. You can also use a downloadable software package called Gephi to do this.

The file output for network analysis is ".graphml" Happily, this format is exportable. So if you create a file like that using this software, you could try plotting it with different graphing methods.

–Plot the diagram in Python, Gephi, or other visualization software.

–What do network diagrams show?

–They can confirm that characters we might think of as central actually are central. Sometimes this simply reaffirms what was obvious, but in big novels with multiple protagonists, the results can sometimes be a little surprising (in the *Game of Thrones* network, Tyrion Lannister comes out as more central than Jon Stark). They can accompany and represent visually what you can otherwise determine mathematically (i.e., Tyrion Lannister is the most centrally connected character in the network)

–They can show us minor characters who might be sneakily well-connected. (Lucrezia Smith in *Mrs. Dalloway* might be such a character)

(See this: https://modernismmodernity.org/forums/posts/social-network-analysis by Sam Alexander for an example of this with *Mrs. Dalloway*)

–They can show clustering we might not have expected. (This is definitely going to be visible in the Dora Marsden British feminist/modernist network we could look at today.) In advanced network diagrams, we can look at "communities" – groupings that emerge algorithmically. Hopefully those algorithmic communities align with other relationships we might have expected to see (in *GOT,* the various families/ "houses")

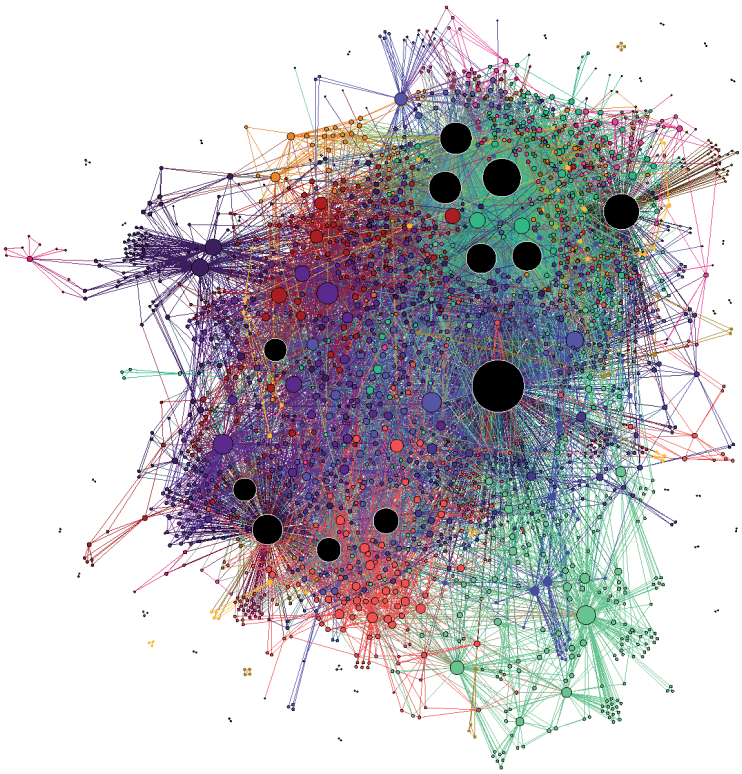
–More advanced network analysis looks at ideas of "degree centrality," "closeness centrality" and "betweenness centrality" and many other concepts that are beyond the scope of this class.

Downsides/Possible sources of error.

As with sentiment analysis, the results of working with network analysis can be variable. Just because we *can* do this doesn't mean the results will always be interesting. And small choices about how we design our networks make a big difference, so we have to thnk carefully.

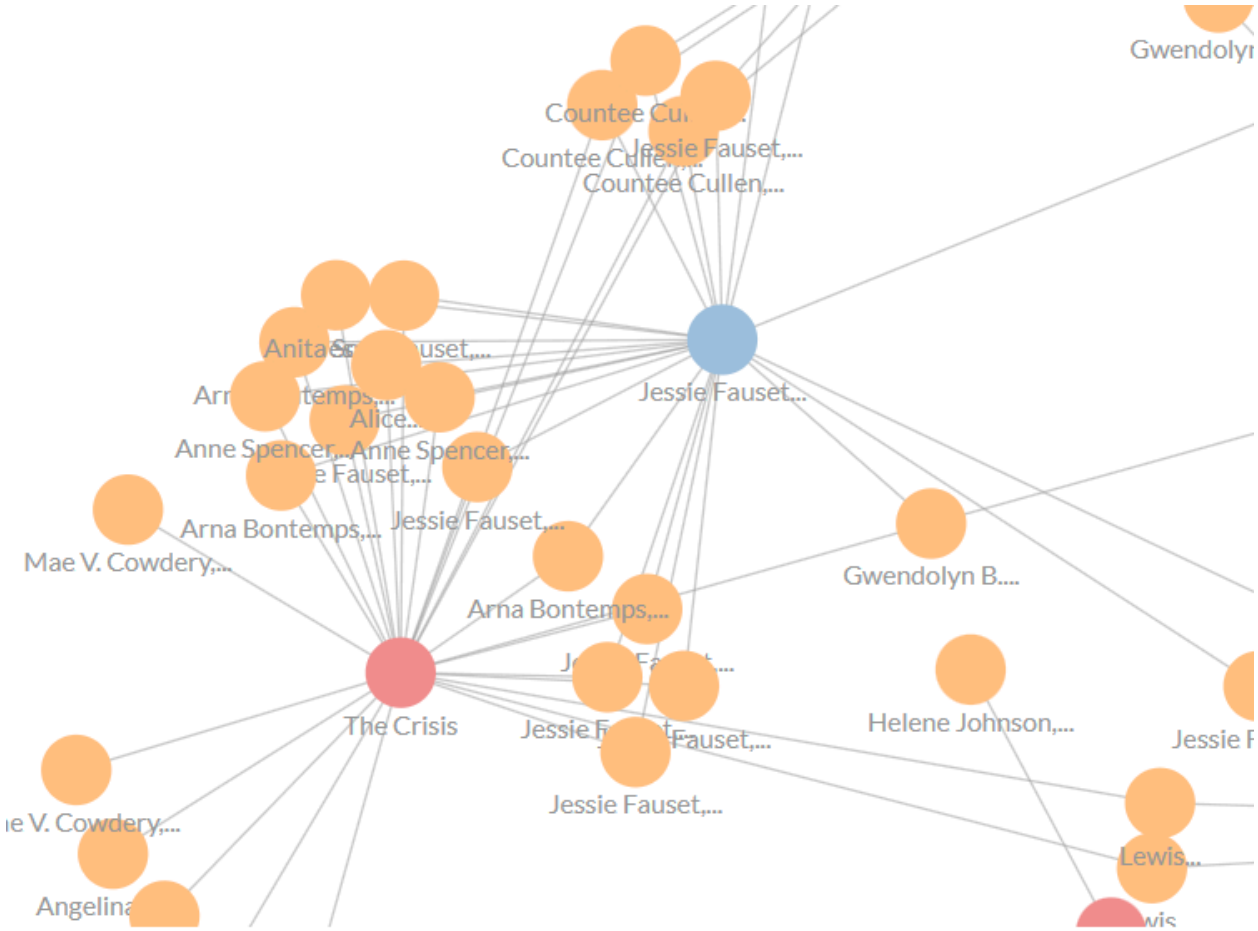
Producing pretty diagrams definitely has a "wowee" effect, and sometimes DH scholars get a little carried away with it. We have to be self-critical and ask whether the diagrams we might be making really show us anything / really are useful.

One of the biggest mistakes people tend to make is to create networks that are so big that the result is just a big "hairball" – essentially a tangle of nodes and edges that just shows that everything is connected to everything else.



The network diagrams that you can make in Scalar using the built-in visualizer have the advantage of being clickable and functional – you can click on a poem or author in one of my diagrams, and go to the actual text / author. So even if the shape of the network isn't that exciting by itself, you could in theory use the network diagram to follow up on relationships and explore clusterings you're interested in.

https://scalar.lehigh.edu/aapada-periodical-poetry/index



Refresher from February (see my notes from 2/13)

Refresh from notes from earlier in the semester:

Helpful vocabulary term from this essay – Named Entity Recognition (NER)

A natural language processing (NLP) algorithm that can look through large text files and find words that appear to be names of people or places.

There are free online tools to do this without coding:
https://wordcount.com/entity-extractor
https://demos.explosion.ai/displacy-ent

These are new. Even just a few years ago, if you wanted to do NER, you needed to be able to run the packages that can use these algorithms in coding environments like Python.

More vocabulary / terminology: Network Analysis
Notes and Edges
Degree Centrality
Betweenness Centrality

There's a good guide to network analysis here:
https://visiblenetworklabs.com/guides/social-network-analysis-101/

Learning how to make Network Diagrams using Networkx in Python

https://melaniewalsh.github.io/Intro-Cultural-Analytics/06-Network-Analysis/01-Network-Analysis.html

Most of the code on this page works just fine in Colab. When you get to this part there is one error you'll probably see:

```
import networkx
import pandas as pd
pd.set_option('max_rows', 400)
import matplotlib.pyplot as plt
```

Apparently the attribute 'max_rows' has been replaced by a newer attribute, 'display.max.rows'.
So do it this way instead:

```
import networkx
import pandas as pd
pd.set_option('display.max_rows', 400)
import matplotlib.pyplot as plt
```

When you get to the part where you need the file got-edges.csv, you can get it from here:

https://github.com/melaniewalsh/sample-social-network-datasets/blob/master/sample-datasets/game-of-thrones/got-edges.csv

We should take a minute and examine this file. How is it structured? (The data needs to be structured in particular ways for it to lead to working network diagrams.)

Or from my Google Drive folder https://drive.google.com/drive/folders/1QpgUIOGu80BiCR4TFaXJHbXIc3ukOLrN?usp=drive_link

Click on the download symbol just above the table on the right.

In Colab, upload it to the local files on the server using the + button in the folder on the left.

If the file is uploaded correctly, you should see it on the left (should be small 5.76 KB). When you get to this step:

```
got_df = pd.read_csv('../data/got-edges.csv')
```

You'll want to change this slightly in Colab once the file has been uploaded to the temporary folder.

```
got_df = pd.read_csv('got-edges.csv')
```

The other steps should work just fine and you should be able to get the network diagrams in Melanie has in her document in Colab without any further alterations in the code.

Calculate Degree: This is a helpful figure to calculate in network analysis – who is the most connected? In fact, the answer here was a little surprising to me (as someone who read the first *Game of Thrones* novel and watched the show all the way through) – Tyrion Lannister.

Tyrion is followed closely by Jon Stark.

Calculate Weighted Degree: This is a related number, but here the number is calculated as the summary of the edge weights. We use weighted degree to show stronger connections between characters. In some visualization styles, stronger connections (weighted degree) is shown with thicker lines.

Betweenness Centrality: I believe this is a way of showing who is the most connected overall. (Interestingly, in this measure, the most connected person is actually Jon Stark, not Tyrion Lannister)

Communities: This is exactly what it sounds like – it shows social clusterings within the data. In *Game of Thrones*, the communities line up largely the way you might expect – Starks, Lannisters, etc. (with some exceptions for characters who cross over in one way or another)

Once we've tried some of the code involving Game of Thrones, let's try the same with the file:

Marsden-edges.csv

(Get that file from this Google Drive folder)

What happens when we do the first graph with this file instead of *Game of Thrones* ? Since this is a bigger network, the graph using the same parameters we used for *Game of Thrones* just looks like a messy tangle.

It starts to look better if we increase the size of the graph. It tried changing "figsize" from 8,8 to 20,20, and the graph I got is still not that useful – but it at least shows a clear pattern of a split network.:

```
got_df = pd.read_csv('marsden-edges.csv')
G = networkx.from_pandas_edgelist(got_df, 'Source', 'Target',
'Weight')
networkx.write_graphml(G, 'Marsden-network.graphml')
plt.figure(figsize=(20,20))
networkx.draw(G, with_labels=True, node_color='skyblue',
width=.3, font_size=8)
```

Make an Interactive Network Visualization with Bokeh

This page has some commands that are a bit broken.

https://melaniewalsh.github.io/Intro-Cultural-Analytics/06-Network-Analysis/02-Making-Network-Viz-with-Bokeh.html

Fixed / updated version here:

https://colab.research.google.com/drive/1vV6Tv6Umo8Mr78FT8YP724Q5mkwfGbUW?usp=sharing

This page has more advanced visualizations which are interactive and somewhat more informative than the visualizations from the previous section. The first, main textbook page above has some code that no longer works, since the packages they work with have been revised since she first created them. However, there is an updated page Melanie created in Colab (the second link above), where the code works well (though we still might want to tweak the settings / format of the diagrams to make them more useful).

The most exciting one to my eye is the one that shows Communities (at the bottom).

On this page, if we've already done the basics from the previous page, we should be able to skip some of the steps. We need to import Numpy and also install Bokeh, which will be the visualization engine we'll use to make the more interactive visualizations in this unit.

```
import numpy as np
!pip install bokeh
```

You can skip the step involving output to a Jupyter notebook (since we're not using Jupyter notebooks here).

We already have the Data Frame (df) set up from the last unit, so you can skip that step.
We should also have the network (G = …)

Again, try the updated Colab link above to actually make the interactive diagrams she shows.

If there's time and interest…

Working on Sentiment Analysis in Python using VADER

https://melaniewalsh.github.io/Intro-Cultural-Analytics/05-Text-Analysis/04-Sentiment-Analysis.html

https://colab.research.google.com/drive/1e176BvulXDNzZ5mSz3ehQxaHK3Xq5Qyo?usp=sharing

There's a fair bit in here we haven't covered and that isn't entirely obvious (for instance, some of the steps involve dictionaries, which is an important concept in Python we haven't talked about). But if you follow her instructions here, you can get to the point where you can run sentiment analysis on your own texts and maybe compare them to the results you were getting with Syuzhet.

The Sentiment analysis package she is using is called VADER – we saw VADER before, with the Hoyeol Kim article we read (see my notes from March 4). VADER might be slightly better than Syuzhet because it has more sensitivity to negators and amplifiers:

> Table 8 demonstrates how Syuzhet simply reports accumulative sentiment scores based on the words in each sentence, as does SentimentAnalysis, while **VADER and sentimentr employ detectors for negators, adversative conjunctions, and amplifiers.**
>
> VADER and sentimentr provide functions for detecting **negators** (not, aren't, no), **amplifiers** (really, absolutely, very), **de-amplifiers** (hardly, barely, rarely), and adversative conjunctions/transitions (nonetheless, however, although).
>
> (source)

If you want to try these, I would create a new notebook in Colab.

I have copies of the files she uses here:

https://drive.google.com/drive/folders/1QpgUIOGu80BiCR4TFaXJHbXIc3ukOLrN?usp=drive_link

I won't write out all of my observations and tips as I worked through this material recently. However, here's my notebook of my attempts to work with Walsh's code and examples.

Almost all of it still works, though there were a couple of things that needed fixing along the way.

https://colab.research.google.com/drive/1e176BvulXDNzZ5mSz3ehQxaHK3Xq5Qyo?usp=sharing

You can make a copy of that if you like.