# The Spiderweb of Jane Addams

Olivia Luisi

2023-12-18

```
import geopandas as gpd
import pandas as pd
import numpy as np
import plotly as plt
import plotly.express as px

# This is my personal MapBox token, it should work on your computer!
px.set_mapbox_access_token("pk.eyJ1Ijoib2x1aXNpIiwiYSI6ImNsb2VsMG94eTA3eGUyanFydmR6c3c2MW0


def add_year(df):
    """
    This function cleans and creates a year dataset from a given df.
    I needed this as the database I pulled the geojson file from
    used hmtl style date data. I had to take the line of html
    and clean it into a year. Thankfully the title has
    a full date in it for the viewer of the map.
    """
    new_df = df.copy()
    result = []
    for index, row in new_df.iterrows():
        value = str(row['date'])
        first = value.split(">")
        second = first[1].split("-")
        third = second[0]
        third = third.replace("</a", "")
        # Special case for the two dates input incorrectly
        if len(third) == 5:
            third = third.replace("0", "")
        elif len(third) > 4:
```

```
            third = third.split(" ")
            third = third[2]
        third = str(third)
        result.append(third)
    return result


def count_per_years(df):
    """
    This function takes in the dataframe and sorts each row into
    groups based on each half decade it belongs to. For items before
    1900 or after 1925. I personally made this choice because
    as a transcriber for the project, I know that 1925+ isn't finished
    being entered yet.
    """
    new_df = df.copy()
    results = []
    for index, row in new_df.iterrows():
        if int(row['year']) >= 1900 and int(row['year']) <= 1905:
            results.append(1905)
        elif int(row['year']) >= 1906 and int(row['year']) <= 1910:
            results.append(1910)
        elif int(row['year']) >= 1911 and int(row['year']) <= 1915:
            results.append(1915)
        elif int(row['year']) >= 1916 and int(row['year']) <= 1920:
            results.append(1920)
        elif int(row['year']) >= 1921 and int(row['year']) <= 1924:
            results.append(1925)
        elif int(row['year']) >= 1925 and int(row['year']) <= 1930:
            results.append(np.NaN)
        else:
            results.append(np.NaN)
    return results


def ani_mapping(df):
    """
    This function creates the animated map shown below.
    It utilizes the information displayed in the title.
    """
    fig = px.scatter_mapbox(df, lat=df.geometry.y, lon=df.geometry.x,
```

```
                            color="year", hover_name="title",
                            animation_group="title", animation_frame="year",
                            title="Jane Addams Animation from 1900-1925",
                            zoom=1, labels={"year": "Years"})
    fig.show()


def text_mapping(df):
    """
    This function creates the combined map shown below.
    It utilizes the information displayed in the title.
    """
    fig = px.scatter_mapbox(df, lat=df.geometry.y, lon=df.geometry.x,
                            color="group", hover_name="title",
                            title="Jane Addams Correspondence from 1900-1925",
                            zoom=1, labels={"group": "Years"})
    fig.show()


# Reading in full DF
JAPP_df = gpd.read_file("JAPP.json")
JAPP_df.head(5)

# Seperating DFs into itemTypes
# This is how I chose what to focus on.
people_JAPP = JAPP_df[JAPP_df['itemtype'] == 'Person']
org_JAPP = JAPP_df[JAPP_df['itemtype'] == 'Organization']
nonyear_text_JAPP = JAPP_df[JAPP_df['itemtype'] == 'Text']

# Add 'year' column cleaned from the href string from 'date' column
nonyear_text_JAPP = nonyear_text_JAPP.dropna()
text_JAPP = nonyear_text_JAPP.copy()
text_JAPP['year'] = add_year(nonyear_text_JAPP)

# Seperate year groups into a group index
text_JAPP['group'] = count_per_years(text_JAPP)

text_JAPP = text_JAPP.dropna()
```

## Introduction

In my Senior year of my bachelor's degree I began working as a transcriber for the Jane Addams Papers Project. There I went through hundreds of documents, created profiles for countless people and entered them all into the Omeka database. I know the dataset better than most people, which has aided me greatly in this endeavor. Towards the beginning of the semester I met with the lead developer for the JAPP and asked what she did for the project and what she sought to improve. She walked me through the Omeka database, showed me how the project has gone through many iterations of mapping systems, and implored me to ask myself what an uninformed viewer would get out of our maps. While my job is asking me to work with the Mapbox API in order to make a functional and interactive map for the web page, myself and our main developer agree that the dataset would be perfect for this final project as well and I have been given permission to do so. I decided to focus on recreating a smaller scale of the map we use on the website. Upon realizing that Plotly had a MapBox collaboration, I knew I could use some of the functionality I was trying to learn. I set out to work on key functionality: I wanted to show how our database's documents grew outside of the United States during the first quarter of the 1900's.

Upon receiving the geo json file I immediately started working on cleaning. The dataset details the Persons, Organizations, Texts, Publications, Events, Moving Images and Still Images associated with Jane Addams and Hull House. It has 10 columns and over 32,000 rows, a huge amount of data. Python and its Pandas library made short work of parsing through the information to create friendly dataframes specific to our desired categories. While I decided to limit myself to the 'text' item type, I still heavily explored the 'people' and 'organization' data for possible crossover. Other columns represented are: Address, Title of item, Image associated with the item, Item Type, Description, ID number, Marker-Size, Marker-Color, Geometry. These additional columns seemed helpful at first before I actually tried applying said data. Multiple columns were heavily dependent on the row's item type, meaning a combined map would end up being hard to connect to the dates of other items. After all, if a person's date is their birth how would that fit alongside a map based on going through time? They would be a blip largely before the start of the textual data's appearance. I came to the conclusion that sticking with the text data would be a more comprehensive experience for a viewer. It would be interesting to see all the people connected with Jane Addams but the average viewer will have no idea the difference between people like Emily Balch and Havelock Ellis and their impact on her work. Therefore to ensure that the viewer gets the information I want out of the graphic, I focused entirely on the text item type. This led into how I went about cleaning the data for my animated map.

## Methodology

The methodology around my data cleaning was entirely summed up by the moment I realized that this data was heavily meant for the website it was created by and for. The actual dates

[5]:

| | address | title | thumb | itemtype | date | desc | id | marker-color | marker-size | geometry |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 800 S Halsted St, Chicago, IL 60607 | Addams, Jane (1860-1935) | https://digital.janeaddams.ramapo.edu/files/fu... | Person | NaN | | 5 | #f00 | small | POINT (-87.64745 41.87168) |
| 1 | 20 Sagamore Hill Rd, Oyster Bay, NY 11771 | Roosevelt, Theodore (1858-1919) | https://digital.janeaddams.ramapo.edu/files/fu... | Person | NaN | Lived in | 6 | #f00 | small | POINT (-73.50538 40.88259) |
| 2 | London, England | Fawcett, Millicent Garrett (1847-1929) | https://digital.janeaddams.ramapo.edu/files/fu... | Person | NaN | Lived in | 7 | #f00 | small | POINT (-0.12549 51.50852) |
| 3 | California State Capitol Sacramento, CA 95814 | Johnson, Hiram W. (1866-1945) | https://digital.janeaddams.ramapo.edu/files/fu... | Person | NaN | Lived in | 9 | #f00 | small | POINT (-121.49368 38.57670) |
| 4 | London, England | Pankhurst, Emmeline Goulden (1858-1928) | https://digital.janeaddams.ramapo.edu/files/fu... | Person | NaN | | 10 | #f00 | small | POINT (-0.12549 51.50852) |

Figure 1: Original Dataframe

associated with each item type depended on the way the website organized itself. I knew how the database worked but was still very surprised when I realized that the date column was filled with lines of HTML or that the thumbnails were linked directly to the website. At first I tried to persist in using the thumbnail images but decided very quickly that it would be too much trouble to both grab and size the thumbnails. In a future iteration of this project I would like to include them as well as a clickable way to view a larger version of the image. Instead my main problem was with the date column. To put it simply, I found multiple items with incorrect data inputs during this part of the project and was able to quickly change them in the Omeka database. But before I realized the inconsistencies I wrote a function which parsed the date string and returned a clean version of the year. This took a lot of trial and error but I did accidentally create a script that checks if our database has incorrect syntax entries so I think it was a win! I used the python string split function to break down the entry into parts that I could decipher the year from. While this might seem strange to use the split function instead of replace, I was able to constantly check that the split lists were of the correct length. This vastly improved my search for broken entries. I used a similar concept to break down the geometry column into latitude and longitude values. Before I realized that I was able to use my MapBox token, I had been besides myself attempting to hard code the latitude and longitude values because Plotly was being extremely temperamental with the geo points. This was equally painful because of incorrectly entered data, but I do not know locations associated with latitude and longitude off the top of my head like I do know that the year '19010' is not a valid date. In this iteration of the project I had decided to drop those values from the table. Thankfully however, I realized Plotly and MapBox work rather well together on the geo json front.

Next came organizing what pieces of data I wanted to use. To someone who has not worked with the JAPP, it might seem fantastic that the data goes all the way to the 1930's! But as a transcriber I know that we actually have very few documents transcribed after 1926. Our transcribers work year by year, separating photocopies of documents by the hundred.

Currently most of us are working between the years of 1925 to 1928. Additionally, while we have documents from before 1900 there are not many and some years would be entirely empty. Therefore I cut the data down between 1900 to 1925. I did this by parsing which rows would stay in the copy of the data frame and adding a 'group' column dedicated to tracking which visual group I would have the entry in. This was not a good way to represent data by itself, but I did it so that there would be a static version of the map with colors representing the half decade I grouped it to. I wanted to show some kind of progression without any movement so that I could get an idea of the overall impact. One of the drawbacks that I will further discuss later is that between the bubble map and the heat map I could not get a middle ground of showing growing reach rather than movement of reach. This led me onto working with animation. The main goal of my project was to get an animated map. Logically, I know where Jane Addams correspondents move across the globe. But I wanted to show how she was slowly expanding: first across America, then deeper into Europe and finally into the Pacific Islands and coast of Asia. These big-little movements can spark curiosity to the viewer. Because of my work separating and cleaning the years, this was surprisingly easy. Most of the work that went into actually showing the maps was figuring out how to make things appear correctly. The 'year' needed to be a string or else it would put decimals in between them, the labels needed to be renamed and base maps needed to be rendered. Plotly was extremely sophisticated for all of this. I had originally written a function that was very long and complicated as I tried to figure out margins and movement. But scrapped almost all of it in favor of instead changing the default zoom. Again, I worried about the hover pop-up but it was easily built in and even color coordinated to the marker. Most of my code throughout this process has been: Attempt to hard code something, achieve what I want, realize that I can do it with less code in another way. This has summarized my entire time learning Python and I would like to think it helps me remember things better.

## Results

My first graphic was more of a proof concept. I wanted to visualize how crowded the map ought to be when all of the years were added together. This fullness is rather impactful and is something I did not feel like I achieved with the animated map. Regardless, I do like how the colors are close enough to blend together so that the viewer can see how the times bloomed over the world map. The darker colors are farther and few between but the yellow shows the main areas that Jane Addams was communicating with as time went on. To my eyes, you can clearly see how late into the quarter century it was when the WILPF was able to hold correspondence with a Chinese delegation. Or how it took until after the end of the first World War for the League to expand deeper into Europe.

```
```{python}
#| label: "Static Map"
#| echo: true
```
```

```
#| fig-cap: "Static Map w/ Mouse-over & Scroll"
text_mapping(text_JAPP)
```

Because these maps are fully interactable with hover information and map movement I could not directly call it into my document without it not functioning as intended. For both of these maps I had to import them in order to show their use.

```{python}
#| label: "Animated Map"
#| echo: true
#| fig-cap: "Animated Map w/ Mouse-over, Scroll & Time Data"
ani_mapping(text_JAPP)
```

This map is the entire point of the project. I wanted to illustrate how Jane Addams' reach slowly webbed through the United States and then into western Europe and Asia. It can be hard to verbalize just how slowly things had to happen in the Feminist movement Jane Addams was a part of. This animated graphic does a good job of showcasing it. It has the same functionality as the static map with mouse-over and pop-ups. I will discuss further just how I would like to improve the map but I must excitedly say that this serves as a fantastic proof of concept for what I would like to do with our data. The additional perk of the map relying on MapBox helps too. I personally tried out MapBox's studio web application and found it fairly overwhelming, but programming it myself has helped immensely. We use JavaScript MapBox on our website and as someone who does not know JavaScript very well, translating it into Python was perfect.

## Discussion

I have a few things I would like to discuss about the pitfalls of the project. One of which being that cleaning the data ended up being more work than the actual initialization of the map. The animated map was the entire point of this project. I wanted to explore geo json files and also figure out how the developer at my work went about the map she made. I am pleasantly surprised and vaguely worried that what I made is not actually as cool as I think it is. I certainly tried exploring other maps too. Heat maps end up losing a lot of information because of the lack of mouse-over. A bar graph could have shown the increase in documents over the years but that would be a very obvious piece of information. The work that went into cleaning the data let me flex my understanding of python strings and pandas dataframes. It was challenging to take data not meant for my purpose and force it to work. But I did not expect how straightforward the map generation would be once the data was cleaned. I could have used the other item types that I had started to clean in the beginning but that

would be perpendicular to the problem I set out to solve. I wanted to make a map that taught something about Jane Addams' reach through the years. Perhaps it would have been a more encompassing project if I reached more but I am not totally sure if I would have had a handle on the scope. As I said in the introduction, the different item types had different columns suited for them. There were a lot of NaN values to sort through and even more value inconsistencies. I am extremely motivated to write scripts to parse the data to be able to point out incorrect entries in the database. How I would connect the data on the same map however leads me back to the original reason for choosing this project. How do you teach an average viewer something from a graphic? While I can spout information about the changes in years on the map I am still very aware that no data entries can do the same. The next step then is simple. While I was working on this project I was already noting and fixing data errors, I have already proved that this is a great way of checking back on new loads of data before being injected into the website. I plan on expanding on these scripts first. Then exploring more map types after.