# Archiving the Web

Sofia Huang

due 10/11/2022

## 1. Collect URIs from Tweets

- **Write a Python program that collects English-language tweets that contain links.**
- **Write a Python program that extracts the links shared in tweets.**
- **Resolve all URIs to their final target URI (i.e., the one that responds with a 200).**
- **Save only unique final URIs (no repeats).**
- **if after this step, you don't have 1000 unique URIs, go back and gather more until you are able to get at least 1000 unique URIs**
- **Save this collection of 1000 unique links in a file and upload it to your repo in GitHub**

I used the function from get_tweets.py from a previous homework to collect 12,500 English tweets using the keywords: 'Queen', 'hurricane', 'Ukraine', 'NASA', and 'Biden'. Then, I read the JSONL file, one line at a time, to obtain any links from each tweet. I also used regex to check if the link led to a Twitter page or any video/audio only site and did not store those links. I also used the requests library to resolve links to their final URI and stored them all in a .txt file. I ended up with 16,004 resolved links.

```
import sys
import json
from twarc import Twarc2, expansions
from configparser import ConfigParser
import re
from subprocess import call
import requests
import os
import time
import csv

TWARC_CONFIG_FILE = "/Users/sofiahuang/Library/Application Support/
   twarc/config"
OUTPUT_FILE = "/Users/sofiahuang/Documents/WM/FALL2022/DATA440/tweets.
   jsonl"   # line-oriented JSON
MAX_TWEETS = 100

def get_tweets(input_search_term):
```

```
    # read Twitter API keys from twarc config file, setup twarc2 object
    config = ConfigParser(interpolation=None)
    with open(TWARC_CONFIG_FILE) as twarc_config:
        config.read_string("[TWARC]\n" + twarc_config.read())
    bearer_token = config['TWARC']['bearer_token'].strip('\'')
    t = Twarc2(bearer_token=bearer_token)

    search_term = input_search_term

    # limit search results to English language, with links, and no retweets
    query = search_term + " lang:en has:links -is:retweet"

    search_results = t.search_recent(query=query, max_results=MAX_TWEETS)
    # max_results is the number of results per page

    num_tweets = 0
    for page in search_results:
    # use expansions.flatten to get all the information in a single JSON
        result = expansions.flatten(page)

    # open the file and write one JSON object per new line (jsonl format)
        with open(OUTPUT_FILE, 'a+') as filehandle:   # if you want to append, change 'w' to 'a+'
            for tweet in result:
                print(tweet)
                filehandle.write('%s\n' % json.dumps(tweet))
                num_tweets = num_tweets + 1
                if num_tweets == MAX_TWEETS:
                # must include this to stop after a certain # of tweets
                    search_results.close()

    print (num_tweets, "tweets written to " + OUTPUT_FILE + " for query \"" + query + "\"\n");

def get_links():
    f = open(OUTPUT_FILE)
    # read in all the lines
    lines = f.readlines()
    links = []
    resolved_links = []
    # each line is a json
    for line in lines:
        tweet_data = json.loads(line)
```

```
        # collect links
        if 'urls' in tweet_data['entities']:
            for link in tweet_data['entities']['urls']:
                # check if link leads to a Twitter or video/audio only page using regex
                pattern = re.compile('(https:\/\/)(www\.|.*)(twitter|youtube|youtu|tiktok|twitch|soundcloud)(\.com|\.be)(.*)')
                m = pattern.match(link['expanded_url'])
                if (m == None):
                    # add to list if link leads to an acceptable page (not Twitter or video/audio)
                    links.append(link['expanded_url'])
                    #print('original: ' + link['expanded_url'])
                    try:
                        # resolve links to their final URI
                        resolved_link = requests.head(link['expanded_url'], allow_redirects=True, timeout=5).url
                        resolved_links.append(resolved_link)
                        #print('final: ' + resolved_link)
                    except:
                        continue
    print('Originally ' + str(len(links)) + ' links.')
    print('Resolved ' + str(len(resolved_links)) + ' links.')
    os.chdir("/Users/sofiahuang/Documents/WM/FALL2022/DATA440")
    # create txt file to store the resolved links
    resolved_uri_file = os.path.join(os.getcwd(), 'resolved_uris.txt')
    for link in resolved_links:
        try:
            with open(resolved_uri_file, 'a+') as f:
                f.write("\n%s" % link)
                print(link)
        except Exception as e:
            print(e)
    return resolved_links
```

**Listing 1:** Collecting tweets and extracting links

```
if __name__ == "__main__":
    search_terms = ['Queen', 'hurricane', 'Ukraine', 'NASA', 'Biden']
    for term in search_terms:
        for i in range(25):
            get_tweets(term)

    final_links = get_links()
```

**Listing 2:** Running get_tweets() and get$_links()$

Then, I used the Unix tools, sort and uniq to save only the unique URIs from resolved_uris.txt to a seperate .txt file. I ended up with 2,146 unique URIs stored.

```
(base) sofiahuang@Sofias-MacBook-Pro DATA440 % sort resolved_uris.txt |
    uniq > unique_uris.txt
(base) sofiahuang@Sofias-MacBook-Pro DATA440 % wc -l unique_uris.txt
    2146 unique_uris.txt
```

**Listing 3:** Keeping only unique URIs

# 2. Get TimeMaps for Each URI

**Obtain the TimeMaps for each of the unique URIs from Q1 using the ODU Memento Aggregator, MemGator.**

I read the .txt file with the unique URIs into a list and created a function to run MemGator, locally, on each of the URIs. I stored the timemaps I obtained in folder and ended up with 1,072 timemaps. I was unable to get timemaps for all of the unique URIs I had stored and I believe this has something to do with the MemGator server and maybe it didn't work for all of them.

```
def read_unique_links_file(txt_filename):
    # check if we are in the correct directory
    os.chdir("/Users/sofiahuang/Documents/WM/FALL2022/DATA440")
    summary_filename = os.path.join(os.getcwd(), txt_filename)
    # read the resolved links txt file into a list to be returned
    try:
        summary_file = open(summary_filename, 'r')
        url_data = summary_file.read()
        unique_list = url_data.split("\n")
        summary_file.close()
        print(str(len(unique_list)))
        return unique_list
    except Exception as e:
        print(e)
        return []

def get_timemaps(list, directory):
    # to keep track of how many URIs timemaps are obtained for
    # and to name the timemaps json file returned from MemGator
    link_index = 1
    for uri in list:
        filename = str(link_index) + ".json"
        # create a folder to store the timemaps
        if os.path.exists(os.path.join(directory, filename)):
            # file has already been created, just increment index and
   go to the next file
            print("Skipping url " + uri + ", file already exists")
            link_index += 1
            continue
```

```
29          else:
30              # run MemGator on each URI in the unique links list
31              print('running memgator on: ' + uri)
32              time.sleep(1)
33              os.system('/Users/sofiahuang/Downloads/memgator-darwin-
    amd64 -F 2 -f JSON ' + uri + ' > /Users/sofiahuang/Documents/WM/
    FALL2022/DATA440/timemaps/' + filename)
34              time.sleep(2)
```

**Listing 4:** MemGator to get timemaps for all unique URIs

```
1 if __name__ == "__main__":
2     # ...more function calls before, shown in previous question
3     links = read_unique_links_file('unique_uris.txt')
4     os.chdir("/Users/sofiahuang/Documents/WM/FALL2022/DATA440")
5     timemap_directory = os.path.join(os.getcwd(),'timemaps')
6     if not os.path.exists(timemap_directory):
7         os.mkdir(timemap_directory)
8     get_timemaps(links, timemap_directory)
```

**Listing 5:** Running functions to get timemaps for the URIs

# 3. Analyze Mementos Per URI-R

**Use the TimeMaps you saved in Q2 to analyze how well the URIs you collected in Q1 are archived. Create a table showing how many URI-Rs have certain number of mementos.**

To store the number of mementos for each URI, I created a dictionary. I iterated through all of the timemap json files in the directory and got the number of mementos archived for each URI. If the json file was zero bytes, it had 0 mementos and I stored that information in the dictionary but did not try to read the file as this would cause an error. I then saved this dictionary as a .csv file so I could easily see the data obtained.

```
1 def read_json():
2     # create dictionary and add entry for URI-Rs with 0 mementos
3     mem_dict = {0:0}
4     os.chdir("/Users/sofiahuang/Documents/WM/FALL2022/DATA440/timemaps"
    )
5     file_count = 0
6     # iterate over json files
7     for json_file in os.listdir("/Users/sofiahuang/Documents/WM/
    FALL2022/DATA440/timemaps"):
8         # open json file
9         f = open(json_file, 'r')
10        if (json_file == '.DS_Store'):
11            continue
```

```
        # if file is 0 bytes, don't read, just add data to dictionary
        if (os.stat(json_file).st_size == 0):
            current_val = mem_dict.get(0)
            mem_dict.update({0: current_val+1})
            file_count+=1
            continue
        # read json file
        data = json.loads(f.read())
        # get number of mementos and add to dictionary,
        # checking if key is already in the dictionary
        mementos = data['mementos']['list']
        mem_count = len(mementos)
        if mem_count in mem_dict:
            current_val = mem_dict.get(mem_count)
            mem_dict.update({mem_count: current_val+1})
        else:
            mem_dict.update({mem_count: 1})
        f.close()
        file_count+=1
        print(file_count)

    # open file for writing dictionary to csv file
    w = csv.writer(open("/Users/sofiahuang/Documents/WM/FALL2022/
DATA440/memento_counts.csv", "w"))
    # loop over dictionary keys and values
    for key, val in mem_dict.items():
        # write every key and value to file
        w.writerow([key, val])
```

**Listing 6:** Find how many mementos obtained from each URI and save to CSV file

I created a table to show how many URIs had a certain number of mementos. We can see that the majority had 100 or less mementos archived, with 449 URIs having 0. The number drastically decreases as the number of mementos increases. With less than 50 URIs having from 101-1000 mementos. I had 12 URIs that had over 1000 mementos archived, with memento counts ranging from 1,273 to 156,369. I checked to see what webpage had the highest number of mementos and it was a dailymail.co/uk webpage from 1998.

| Mementos | URI-Rs |
|---|---|
| 0 | 449 |
| 1-100 | 568 |
| 101-200 | 17 |
| 201-300 | 11 |
| 301-400 | 7 |
| 401-500 | 0 |
| 501-600 | 2 |
| 601-700 | 2 |
| 701-800 | 0 |
| 801-900 | 1 |
| 901-1000 | 3 |
| $>1000$ | 12 |

# 4. Analyze Datetimes of Mementos

**For each of the URI-Rs from Q3 that had $>0$ mementos, create a scatterplot with the age of each URI-R** `(today - earliest memento datetime)` **on the x-axis and number of mementos for that URI-R on the y-axis.**

Figure 1 shows the age and number of mementos for each URI-R.

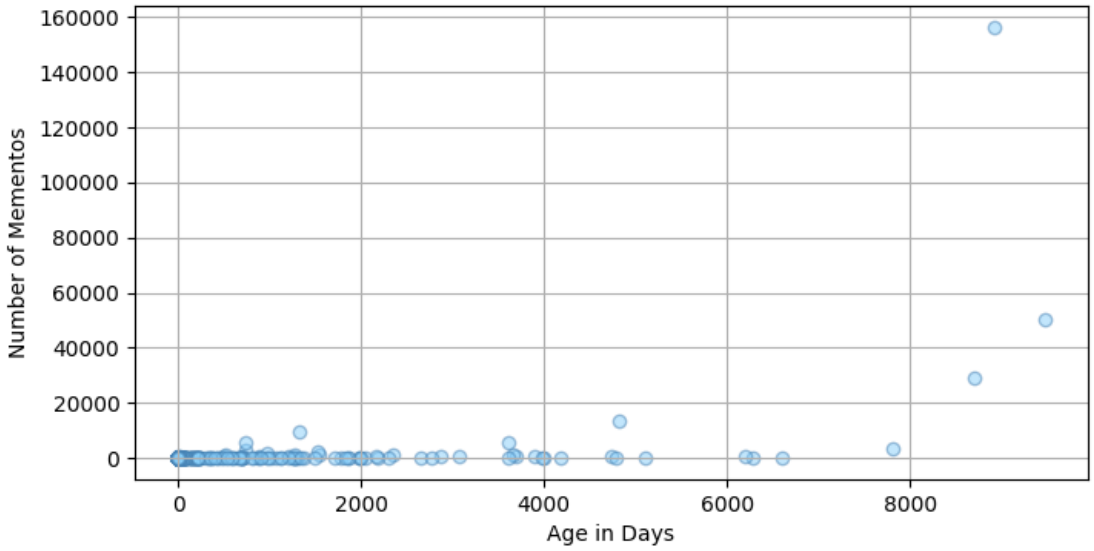


**Figure 1:** Age vs Memento Counts for URI-Rs

Figure 2 shows the age and number of mementos for each URI-R without outliers.

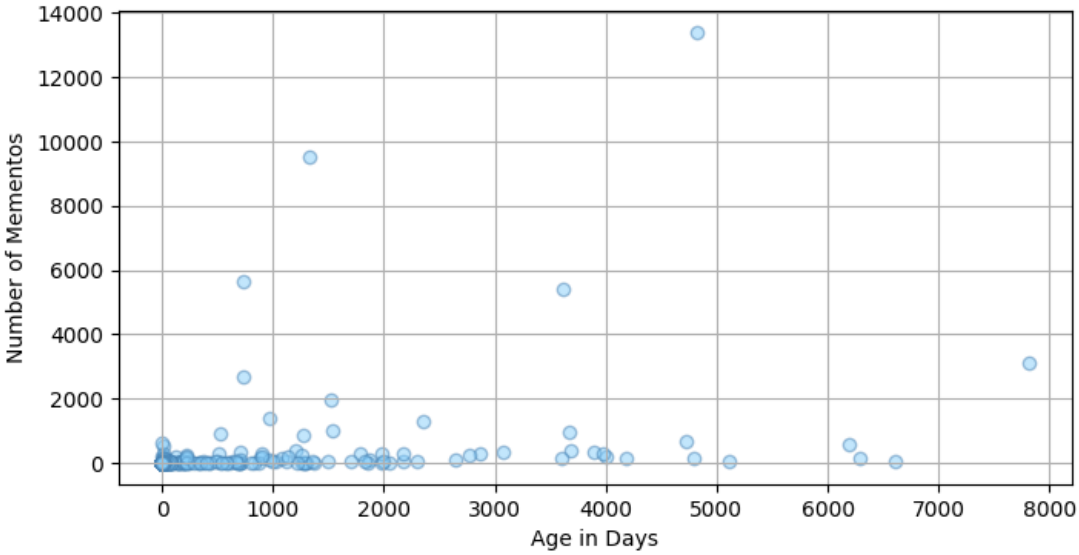


**Figure 2:** Age vs Memento Counts for URI-Rs *excluding URI-Rs with more than 20,000 mementos*

I iterated through the memento json files and stored the age (using the datetime library), memento count, and URI-R in lists for each URI-R. I created a dataframe out of the 3 lists and then stored the dataframe as a .csv file. Then, I wrote another function to create a scatterplot (using Matplotlib). After I looked at the resulting scatterplot, I decided to create another one without the outliers or the URI-Rs that had greater than 20,000 mementos so that the relationship between age and memento count could be seen more clearly for the lower values.

**Q: What can you say about the relationship between the age of a URI-R and the number of its mementos?**

Typically, the older URI-Rs have more mementos than newer ones, however the correlation isn't very strong and there are certainly a few outliers. Most of the URI-Rs were only a couple of years old and had less than 1,000 mementos. I included a scatterplot of the same data just excluding the URI-Rs that had over 20,000 mementos so you could see the majority of the data better.

**Q: What URI-R had the oldest memento? Did that surprise you?**

The URI-R with the oldest memento was 'http://archive.md/19961101130030/http://www.redcross.org/' and it was 9,471 days old or almost 26 years old. The Red Cross has been around for a long time and is a well-known organization so it doesn't surprise me that their webpage has been archived well.

**Q: How many URI-Rs had an age of $<$ 1 week, meaning that their first memento was captured the same week you collected the data?**

175 URI-Rs had an age of less than a week. I looked at the .csv file to figure this out.

```
def get_memento_age():
    # create dictionary and add entry for URI-Rs with 0 mementos
    mem_dict = {0:0}
    os.chdir("/Users/sofiahuang/Documents/WM/FALL2022/DATA440/timemaps")
    file_count = 0
    memento_counts = []
    memento_ages = []
    uris = []
    # iterate over json files
    for json_file in os.listdir("/Users/sofiahuang/Documents/WM/FALL2022/DATA440/timemaps"):
        # open json file
        f = open(json_file, 'r')
        # if file is 0 bytes, don't read
        if (os.stat(json_file).st_size == 0):
            continue
        # read json file
        data = json.loads(f.read())
        # get earliest date
        memento_earliest_date_str = data['mementos']['list'][0]['datetime']
        memento_earliest_date = datetime.strptime(memento_earliest_date_str[0:10], '%Y-%m-%d')
        memento_age = (datetime.today() - memento_earliest_date).days
        # add data to lists
        memento_ages.append(memento_age)
        memento_counts.append(len(data['mementos']['list']))
        uris.append(data['mementos']['list'][0]['uri'])
        f.close()
        file_count+=1
    # zip lists in dataframe and store as .csv file
    df = pd.DataFrame(list(zip(uris, memento_ages, memento_counts)))
    df = df.rename(columns={"0": "uri", "1": "age", "2": "num_memento"})
    df.to_csv("/Users/sofiahuang/Documents/WM/FALL2022/DATA440/memento_ages_counts.csv", index=False)

def memento_age_scatterplot(df, filename):
    # rename columns
    df.rename(columns={"0": "uri", "1": "age", "2": "num_memento"}, inplace=True)
    plt.figure(figsize=(8, 4))
    # create scatterplot, add grid, axis labels, and save plot
    plt.scatter(df['age'], df['num_memento'], c ="lightskyblue", alpha=0.5, edgecolors='steelblue')
    plt.grid()
```

```
40      plt.xlabel('Age in Days')
41      plt.ylabel('Number of Mementos')
42      plt.savefig(filename)
```

**Listing 7:** Create scatterplot to show age of mementos

# References


- StackOverflow - Pretty Print JSON `https://stackoverflow.com/questions/20265439/how-can-i-pretty-print-a-json-file-from-the-command-line`
- MemGator API `https://memgator.cs.odu.edu/api.html`
- Python parse JSON file `https://www.freecodecamp.org/news/python-parse-json-how-to-read-a-json-file/`
- Python Save Dictionary to File `https://pythonspot.com/save-a-dictionary-to-a-file/`
- Check if Key Exists in Dictionary `https://www.geeksforgeeks.org/python-check-whether-given-key-already-exists-in-a-dictionary/`