

OLD DOMINION UNIVERSITY

---

## Assignment one

---

Joshua Gahan

February 18, 2019

# 1 Tweet Collection

Using the keyword 'Trump' we collected tweets via Twitter's Streaming API in order. to extract unique URIs. These URIs were output to a file "unparsedURLs.txt" for further processing.

## 1.1 Retrieval of Tweets

---

```
class MyStreamListener(tweepy.StreamListener):
    def __init__(self, num_tweets):
        super().__init__()
        self.num_tweets = int(num_tweets)
        self.outFile = open("unparsedURLs.txt", 'a')
        self.errFile = open("error_log", 'a')
        self.foundURLs = 0 #used for output to console as a
            visual aid
        self.parsedURLCount = 0 #used for output to console as a
            visual aid.
        self.curious = 0
    def on_status(self, status):
        # regex to extract links from tweet
        parsed_tweet = re.findall('https*:\/\/\S+', status.text)

        self.curious += 1
        for url in parsed_tweet:
            self.outFile.write(url + '\n')
            self.num_tweets -= 1
            self.foundURLs += 1

        if self.foundURLs % 20 == 0:
            print("Found Urls: %d Tweets Processed: %d" %
                (self.foundURLs, self.curious))

        # False turns the stream off.
        if self.num_tweets < 1:
            return False

    def on_error(self, status_code):
        if status_code == 420:
            #returning False in on_error disconnects the stream
            return False
```

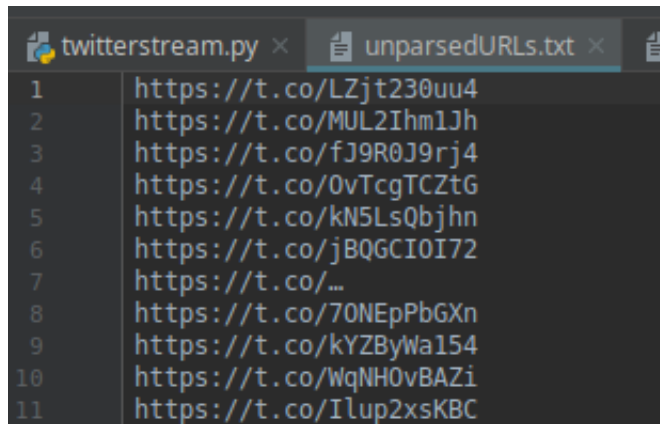
---

MyStreamListener functions as the 'on data' listener for the tweepy.Stream

object. When data is received, the `on_status` function is called. We use a regex search to extract the URIs within a tweet and store them within `parsed_tweet`. Finally we write the found URLs to `unparsedURLs.txt`. We utilize two variables: `foundURLs` and `curious` as helpers for giving meaningful output to the user as to the scripts current progress (printing our status out every 20 found URLs). Finally, we utilize a `num_tweets` variable as a control variable. When a tweepy stream object receives "False", the stream is disabled. We utilize this behavior to allow us to operate the stream until a desired amount of links have been found (provided as a command line argument).

```
(venv) joshua@joshua-MS-7917:~/source_code/pycharm_projects/webscience_a2$ python3 twitterstream.py sports 50
Found Urls: 20   Tweets Processed: 39
Found Urls: 40   Tweets Processed: 60
(venv) joshua@joshua-MS-7917:~/source_code/pycharm_projects/webscience_a2$
```

Figure 1: Stream Console output



```
twitterstream.py x unparsedURLs.txt x
1 https://t.co/LZjt230uu4
2 https://t.co/MUL2Ihm1Jh
3 https://t.co/fJ9R0J9rj4
4 https://t.co/0vTcgTCZtG
5 https://t.co/kN5LsQbjhn
6 https://t.co/jBQGCIOI72
7 https://t.co/...
8 https://t.co/70NEpPbGXn
9 https://t.co/kYZByWa154
10 https://t.co/WqNH0vBAZi
11 https://t.co/I1up2xsKBC
```

Figure 2: unparsedURLs.txt

## 2 Extracted Link Processing.

Utilizing multithreading we set ten threads to the task of resolving our extracted links to URLs. If these links are within the twitter domain they are discarded. Parsed URLs are then written to another intermediary file called "parsedURLs.txt"

## 2.1 Link Processing

---

```
def processURL(thread_name, q, count):
    while not exitFlag:
        count += 1
        queueLock.acquire()

        # follow the url until the location header isn't present, then
        # check if its a twitter domain.
        # push onto threadsafe list, which will be fed into a set down
        # the line to check for duplicates.
        if not workQueue.empty():
            url = q.get()
            queueLock.release()
            push = False
            try:
                res = requests.get(url)
                url = res.url

                while res.status_code == 301:
                    try:
                        url = res.headers['location']
                        res = requests.get(url)
                    except Exception as e:
                        print("Error: ", str(e))

                if url.find('twitter') < 0 and res.status_code == 200:
                    push = True
                else:
                    push = False
            except Exception as e:
                print("some error " + str(e) + ": " + url)

            if push == True:
                notTwitter.append(url)
        else:
            queueLock.release()
            time.sleep(1)

    if len(notTwitter) % 25 == 0:
        print("%d candidate links" % len(notTwitter))
```

```

# below is the file write which is not part of the above
function. Note that goodURLs is a set.
for item in notTwitter:
    goodURLS.add(item + '\n')
with open("parsedURLs.txt", 'a') as outFile:
    for entry in goodURLS:
        outFile.write(entry)

```

---

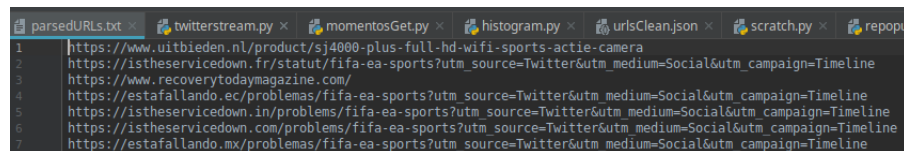
We start by pulling a url to test of the queue. We then attempt a 'GET' call on the url. If the response contains status code 301 (a redirect) we then follow the redirect (this is done via a while loop that stops once it encounters a non-301 status code). Following this we do a string search of the parsed url, pushing any found urls that don't contain 'twitter' and result in a status code 200 ('OK') to notTwitter (a list). URLs that have been pushed to this notTwitter are then passed into a set "goodURLs" to utilize the properties of a set object for removing duplicates. this goodURL set is then wrote to an intermediary file "parsedURLs.txt".

```

(venv) joshua@joshua-MS-7917:~/source_code/pycharm_projects/webscience_a2$ python3 urlCheck.py
5 candidate links
10 candidate links
10 candidate links
10 candidate links
10 candidate links
15 candidate links
15 candidate links
15 candidate links
15 candidate links
15 candidate links
15 candidate links
some error URL has an invalid label.: https://t.co...
some error URL has an invalid label.: https://t...
20 candidate links
Exiting Main Thread

```

Figure 3: URL Check output



```

1 https://www.uitbieden.nl/product/sj4000-plus-full-hd-wifi-sports-actie-camera
2 https://istheservicedown.fr/statut/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline
3 https://www.recoverytodaymagazine.com/
4 https://estafallando.ec/problemas/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline
5 https://istheservicedown.in/problems/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline
6 https://istheservicedown.com/problems/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline
7 https://estafallando.mx/problemas/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline

```

Figure 4: parsedURLs.txt

## 3 Further Extracted Link Processing

Having cleaned our data, we here push the URLs we wrote into 'parse-dURLs.txt' to urlsClean.json file which will serve as our database for all further operations on the dataset. Below code simply reads in the parsed urls and instantiates the ursClean.json per our desired format (shown below).

### 3.1 Further Link Processing

---

```
#initURL_json_db.py
def initializeURLJSON():
    # create backup of URL database file
    backup = json.load(open("urlsClean.json"))
    with open('urlsClean.bak', 'w') as json_file:
        json.dump(backup, json_file)
        json_file.close()

    # reopen file and get count of URLS present. Store data in
    # goodURL and count in goodURLCount
    goodURL = json.load(open("urlsClean.json"))

    with open('parsedURLs.txt', 'r') as urls:
        for line in urls:
            outUrl = line.strip()
            goodURL["%s" % outUrl] = {"momentos": 0,
                                      "timeMapFilename": " ", "carbonDate": " "}

    with open('urlsClean.json', 'w') as outFile:
        pretty_data = json.dumps(goodURL, indent=4)
        outFile.write(pretty_data)

initializeURLJSON()
```

---

## 4 Further Extracted Link Processing

Having finished collecting our data. We begin to process it in momentosGet.py. Here we read in our URL database and pass the URLs to Mengator to collect momento data.

### 4.1 Memento Collection Algorithm

```

{
  "https://www.uitbieden.nl/product/sj4000-plus-full-hd-wifi-sports-actie-camera": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": " "
  },
  "https://istheservicedown.fr/statut/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": " "
  },
  "https://www.recoverytodaymagazine.com/": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": " "
  }
}

```

Figure 5: Example entries from urlsClean.json

---

```

urls_to_Momentize = json.load(url_db)
count = 0
momento_count = 0
for url in urls_to_Momentize:
    if urls_to_Momentize[url]["momentos"] <= 0:
        count += 1
        try:
            res = requests.get("%s/timemap/json/%s" %
                               (memgator_url, url))
            if(res.status_code == 200 or res.status_code == 302):
                try:
                    urls_to_Momentize[url]["momentos"] =
                        res.headers['X-Memento-Count']
                    momento_count +=
                        int(res.headers['X-Memento-Count'])
                    if urls_to_Momentize[url]["timeMapFilename"]
                        == " ":
                        timeMapFileName = "%d.txt" % abs(hash(url))
                    else:
                        timeMapFileName =
                            urls_to_Momentize[url]["timeMapFilename"]
                    with open("./timemaps/%s" % timeMapFileName,
                              'w') as outFile:
                        outFile.write(res.text)
                    urls_to_Momentize[url]["timeMapFilename"] =
                        timeMapFileName
                except Exception as e:
                    print("Error: ", str(e))

```

---

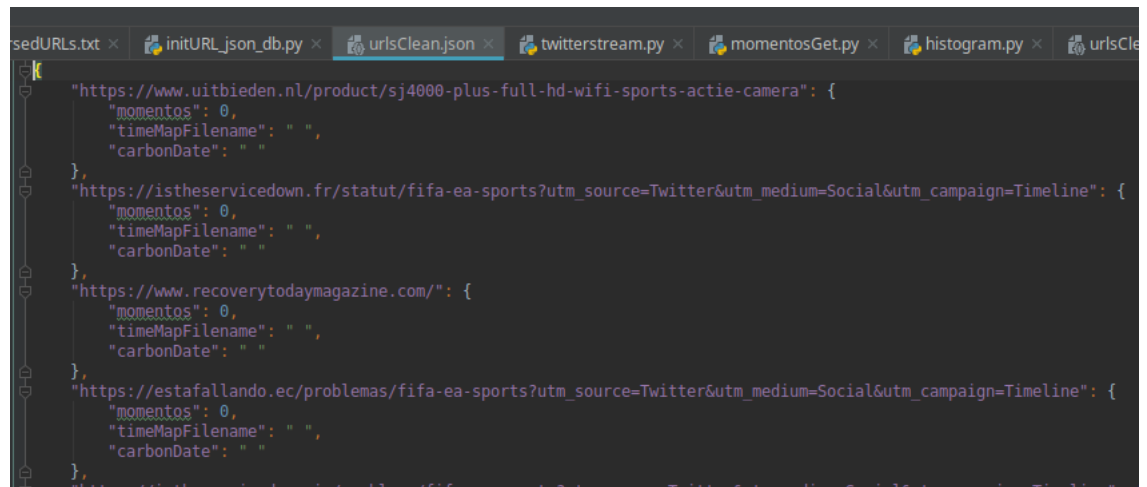
After loading our urls from urlsClean.json we check for each URL if

mementos have already been populated, if not we make a request to the provided Memgator URL. If Memgator returns either a 200 or 302, we extract the Memento count from the 'X-Memento-Count' header. We then create (or open if it exists) a text file containing the json output of the response (if status code 300 or 302) which is named based on the hash of the url. Finally. We take the name of this file we have just created and add it to our database for easy indexing later.

## 4.2 Memgator Processing

```
(venv) joshua@joshua-MS-7917:~/source_code/pycharm_projects/webscience_a2$ python3 momentosGet.py http://memgator.cs.odu.edu/
memgator url: http://memgator.cs.odu.edu/
5 links passed to memgator. 0 total momentos found
10 links passed to memgator. 0 total momentos found
15 links passed to memgator. 0 total momentos found
(venv) joshua@joshua-MS-7917:~/source_code/pycharm_projects/webscience_a2$
```

Figure 6: console log of using memgator



```
sedURLs.txt x initURL_json_db.py x urlsClean.json x twitterstream.py x momentosGet.py x histogram.py x urlsCle
{
  "https://www.uitbieden.nl/product/sj4000-plus-full-hd-wifi-sports-actie-camera": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": " "
  },
  "https://istheservicedown.fr/statut/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": " "
  },
  "https://www.recoverytodaymagazine.com/": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": " "
  },
  "https://estafallando.ec/problemas/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": " "
  },
}
```

Figure 7: our updated JSON. Note: for demo purpose, we are working with a limited dataset, hence no mementos.



## 5 CarbonDate

Utilizing the same urlsClean.json database, we then use the CarbonDate service to estimate the creation date of the URLs within urlsClean.json

### 5.1 CarbonDate Algorithm

---

```
def processURL(thread_name):
    while not exitFlag:
        global count
        global workQueue
        global goodURL
        count += 1
        queueLock.acquire()

        if not workQueue.empty():
            url = workQueue.get()
            queueLock.release()
            try:
                res = requests.get("%s/cd/%s" % (carbodate_url,
                    url))
                responseJSON = json.loads(res.text)
                creationDate =
                    responseJSON["estimated-creation-date"]
                goodURL[url]["carbonDate"] = creationDate
            except Exception as e:
                print("Error: ", str(e))

        if count % 25 == 0:
            print("%d links passed to carbodate. Executed by
                thread: %s" % (count, thread_name))
```

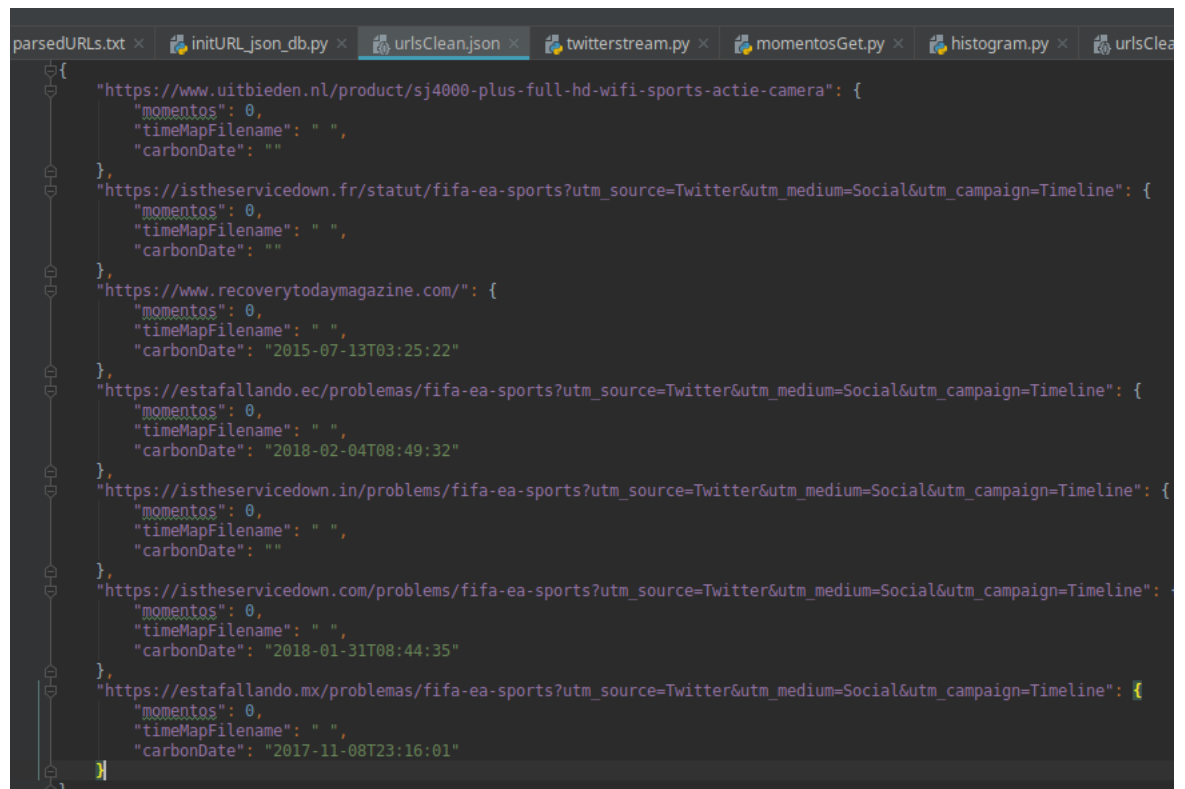
---

Utilizing the same multithreaded scheme used before, each thread pulls urls from a queue and executes the function above. We make a request to the carbodate service and store the response JSON in responseJSON. Within this responseJSON we extract the value from the "estimated-creation-date" key. we then push this found date to our JSON database (urls are

```
(venv) joshua@joshua-MS-7917:~/source_code/pycharm_projects/webscience_a2$ python3 carbondate.py http://localhost:8888
carbondate url: http://localhost:8888
Exiting Main Thread
(venv) joshua@joshua-MS-7917:~/source_code/pycharm_projects/webscience_a2$
```

Figure 8: console log of using carbondate.py

## 5.2 Carbondate processing



```
{
  "https://www.uitbieden.nl/product/sj4000-plus-full-hd-wifi-sports-actie-camera": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": ""
  },
  "https://istheservicedown.fr/statut/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": ""
  },
  "https://www.recoverytodaymagazine.com/": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": "2015-07-13T03:25:22"
  },
  "https://estafallando.ec/problemas/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": "2018-02-04T08:49:32"
  },
  "https://istheservicedown.in/problems/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": ""
  },
  "https://istheservicedown.com/problems/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": "2018-01-31T08:44:35"
  },
  "https://estafallando.mx/problemas/fifa-ea-sports?utm_source=Twitter&utm_medium=Social&utm_campaign=Timeline": {
    "momentos": 0,
    "timeMapFilename": " ",
    "carbonDate": "2017-11-08T23:16:01"
  }
}
```

Figure 9: our updated JSON following carbondating

used as keys for urlsClean.json)

## 6 Findings

We found by that the overwhelming majority of URLs did not possess memento representations. This could be that we used "Trump" as our

keyword and thus the vast majority of our URLs linked to recent news and events. As the number of mementos increased, the number of URLs possessing that many mementos decreased. We had some outliers, such as URLs with over 50,000 mementos, but just over 76% of our URLs had between 0 and 2 mementos. In looking at estimated CarbonDate reported

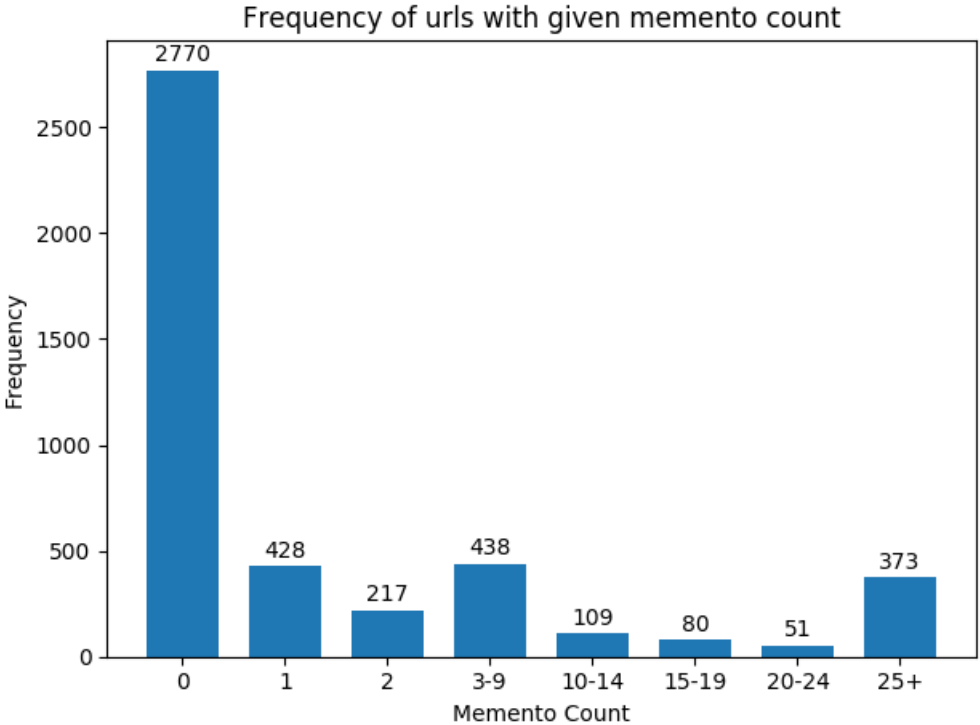


Figure 10: #of urls possessing x number of mementos

date vs memento count, one would hypothesize that as the age of a site increases, so to would the number of mementos for that sight in a more or less linear fashion. The graph below shows our findings (Note that URLs with memento counts greater than 50,000 have been excluded as data points below).

### 6.1 Suspicion of Data

We find ourselves suspicious of these data points however. The carbondate was only set when we received a date from the Carbondate server,



Figure 11: Estimated age of site vs Memento Count

and was not set otherwise, but it is the suspicion of the author that that we encountered throttling and CarbonDate was filling in the date with the current date. The graph attached as an appendix after the conclusion we believe shows this.

## 7 Conclusion

Despite our suspicions of the CarbonDate vs Memento count data, our collected data lines up well with the hypothesis that there is a linear relationship between the age of a site and the number of times it has been archived. A further investigation of the reported creation dates, might tease out if the data is incorrect due to the above hypothesis, correct but aberrant data, or if our hypothesis is incorrect.

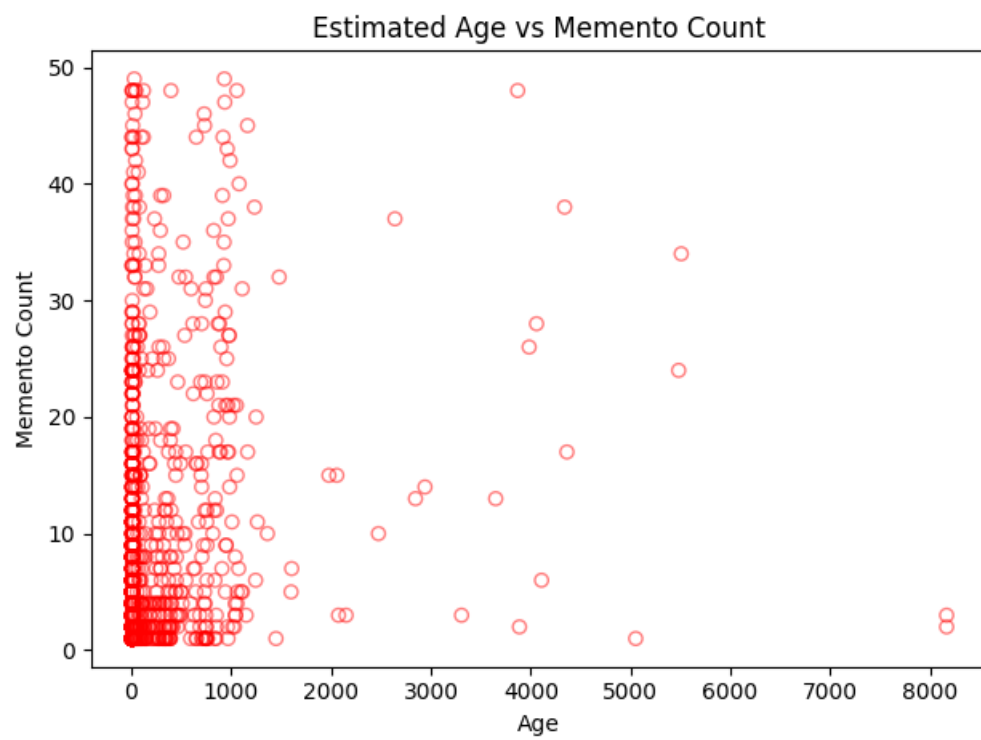


Figure 12: Estimated age of site vs Memento Count