CS 432/532: Web Science

Spring 2017

Assignment 2

Michael Micros

Honor Pledge

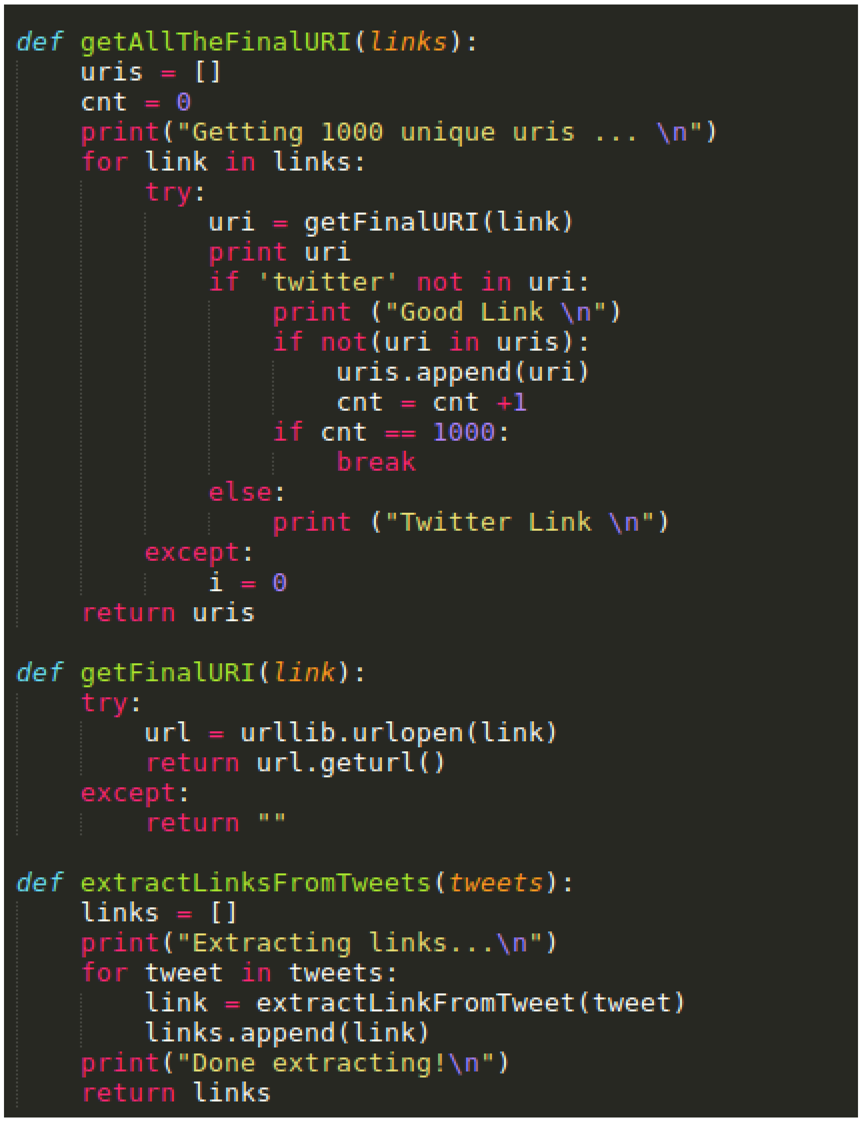
I pledge to support the Honor System of Old Dominion University. I will refrain from any form of academic dishonesty or deception, such as cheating or plagIiarism. I am aware that as a member of the academic community it is my responsibility to turn in all suspected violations of the Honor Code. I will report to a hearing if summoned.

Signed ________________________________

February 16, 2017

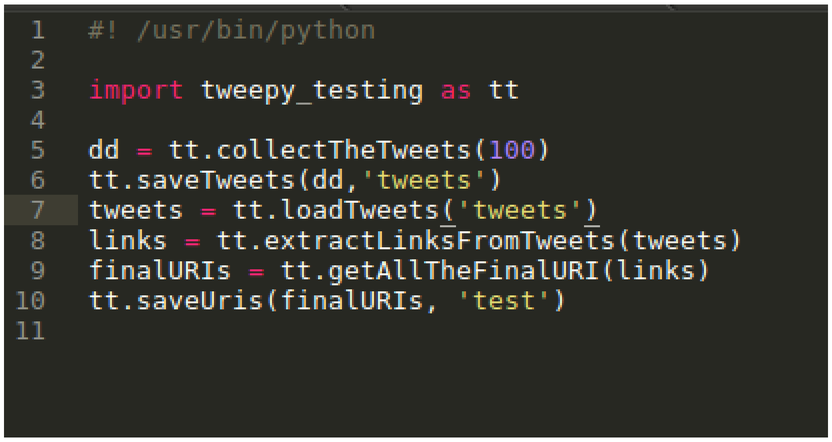
Problem 1: Extracting 1000 unique Links from twitter

The process of extracting links from twitter is straightforward once one understands the structure of the JSON file that tweets have. Most tweets contain links in the "text" field of the JSON, except if they happen to be retweets. That is why before extracting the links we have to check the "Retweeted Status" of the tweet and extract the link from the appropriate JSON field. Once we have collected a reasonable amount of tweets (in the case of this project 5000 tweets were collected), it is important to follow the link until the request response returns a 200. Finally, in order to avoid collecting URIs that point back to twitter we sift thrrough the 5000 collected URIs discarding the twitter links (which understandibly make up the majority of links collected) until we have a list of 1000 unique URIs. For part 1 a library was created in "tweepyTesting.py" that contains all the functions to be used to complete the question. A "main.py" function is used that runs the functions in the appropriate order. The 1000 unique URIs are located in the "links1000" file of part 1.



```
def getAllTheFinalURI(links):
    uris = []
    cnt = 0
    print("Getting 1000 unique uris ... \n")
    for link in links:
        try:
            uri = getFinalURI(link)
            print uri
            if 'twitter' not in uri:
                print ("Good Link \n")
                if not(uri in uris):
                    uris.append(uri)
                    cnt = cnt +1
                if cnt == 1000:
                    break
            else:
                print ("Twitter Link \n")
        except:
            i = 0
    return uris

def getFinalURI(link):
    try:
        url = urllib.urlopen(link)
        return url.geturl()
    except:
        return ""

def extractLinksFromTweets(tweets):
    links = []
    print("Extracting links...\n")
    for tweet in tweets:
        link = extractLinkFromTweet(tweet)
        links.append(link)
    print("Done extracting!\n")
    return links
```

Figure 1: Functions that extract 1000 unique links

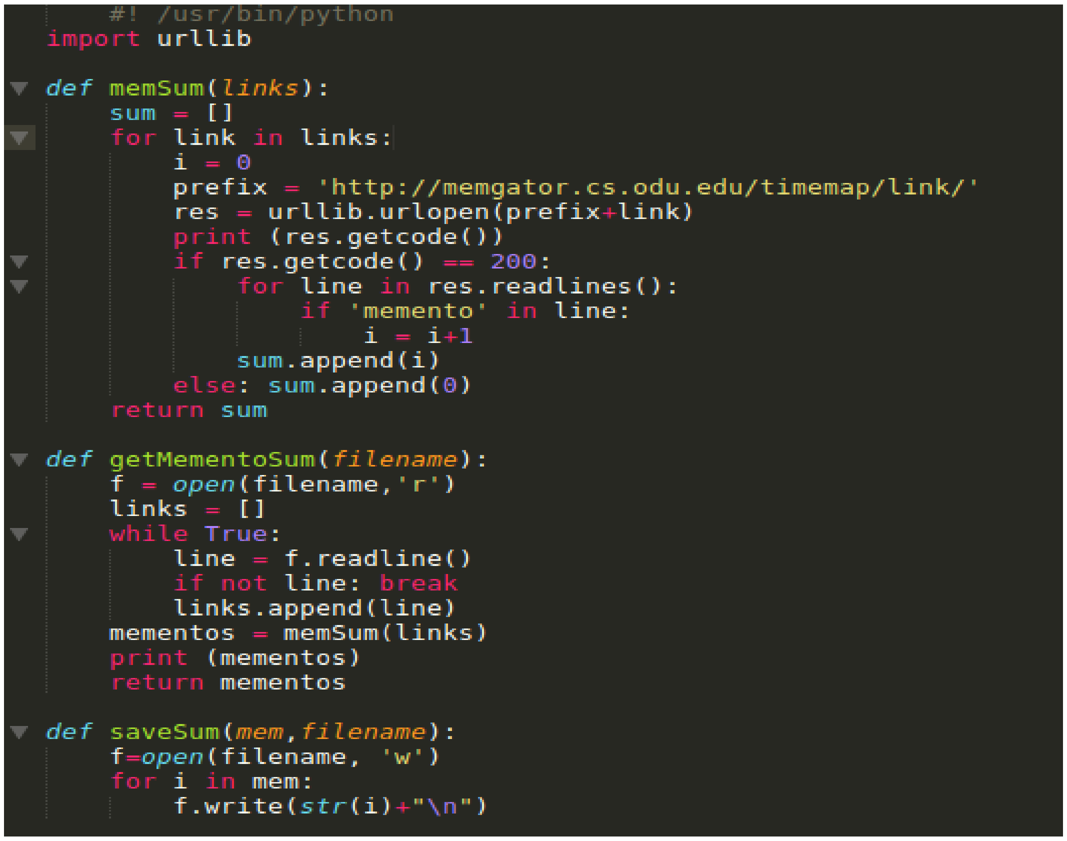


```python
#! /usr/bin/python

import tweepy_testing as tt

dd = tt.collectTheTweets(100)
tt.saveTweets(dd,'tweets')
tweets = tt.loadTweets('tweets')
links = tt.extractLinksFromTweets(tweets)
finalURIs = tt.getAllTheFinalURI(links)
tt.saveUris(finalURIs, 'test')

```

Figure 2: "main.py" calling functions from "tweepyTesting.py"

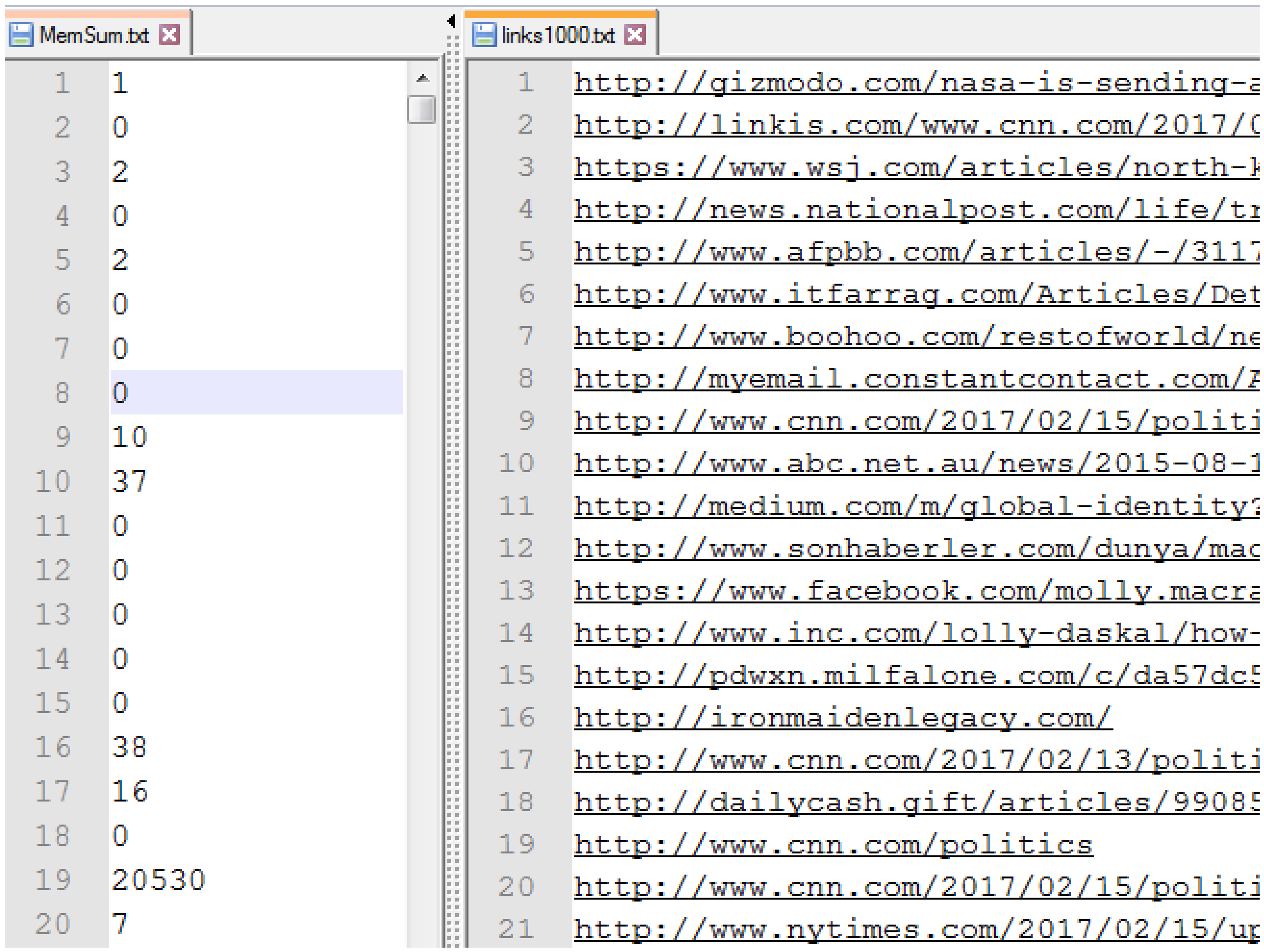
Problem 2: Finding number of mementos and plotting histogram

Similarly to problem 1, a library named "part2.py" was created that contained all the functions necessary to download the timemaps of the 1000 URIs and calculate the number of memntos for that link. The main function that utilizes these functions is "main.py". Another action that needed to be taken, was to find and isolate URIs that more than 0 mementos (these URIs are used in Part3 of this assignment).



```python
    #! /usr/bin/python
import urllib

def memSum(links):
    sum = []
    for link in links:
        i = 0
        prefix = 'http://memgator.cs.odu.edu/timemap/link/'
        res = urllib.urlopen(prefix+link)
        print (res.getcode())
        if res.getcode() == 200:
            for line in res.readlines():
                if 'memento' in line:
                    i = i+1
            sum.append(i)
        else: sum.append(0)
    return sum

def getMementoSum(filename):
    f = open(filename,'r')
    links = []
    while True:
        line = f.readline()
        if not line: break
        links.append(line)
    mementos = memSum(links)
    print (mementos)
    return mementos

def saveSum(mem,filename):
    f=open(filename, 'w')
    for i in mem:
        f.write(str(i)+"\n")
```

Figure 3: Functions that get the timestamps and calculate the no. of mementos


MemSum.txt

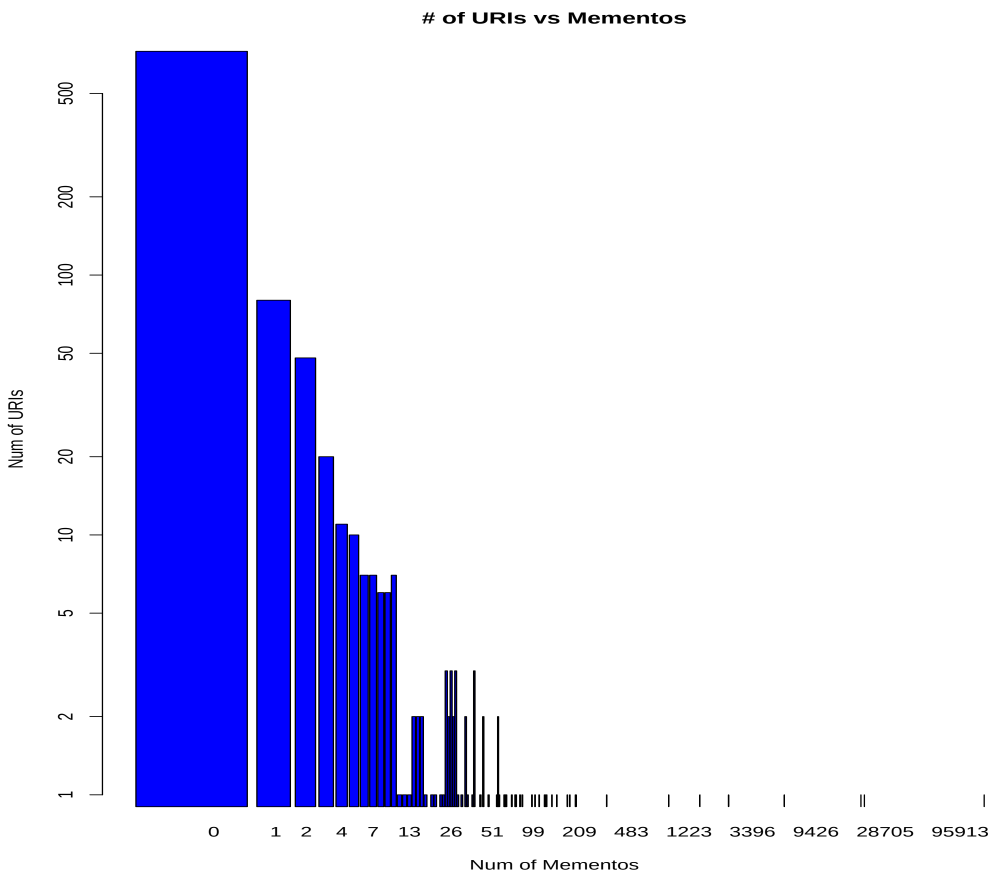
```
1
0
2
0
2
0
0
0
10
37
0
0
0
0
0
38
16
0
20530
7
```

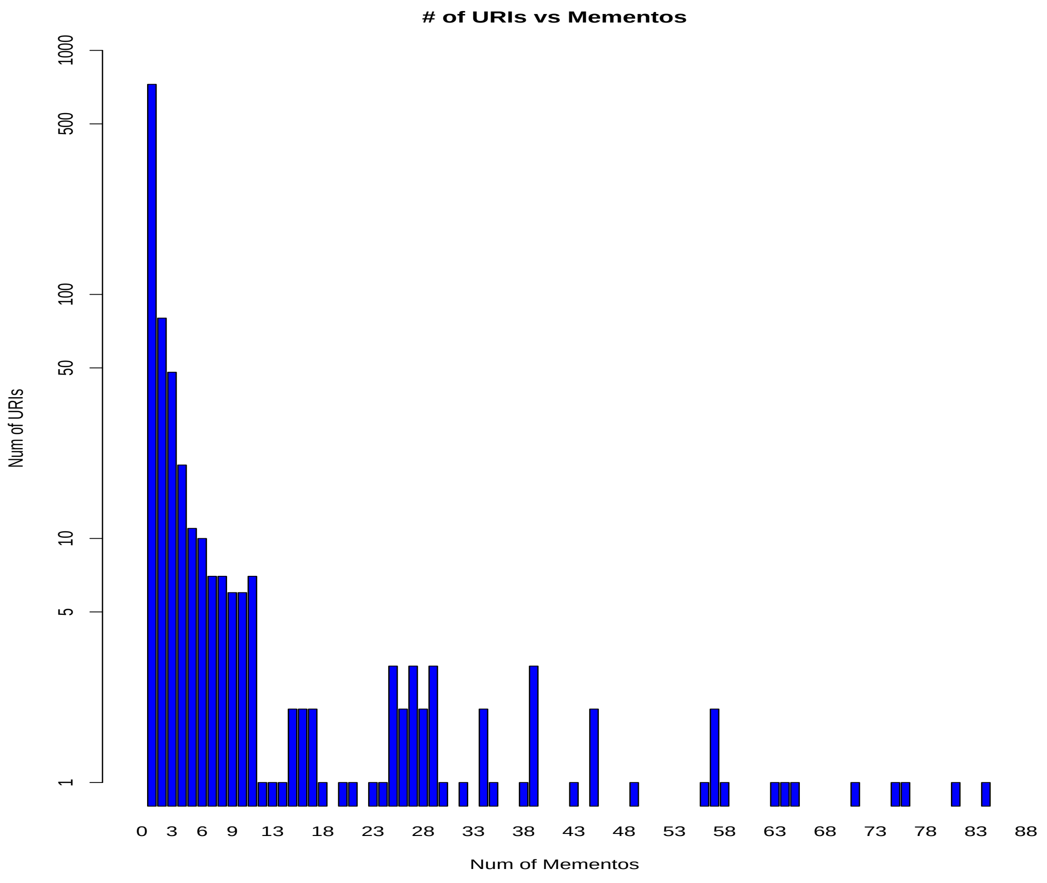
links1000.txt

```
http://gizmodo.com/nasa-is-sending-a
http://linkis.com/www.cnn.com/2017/0
https://www.wsj.com/articles/north-k
http://news.nationalpost.com/life/tr
http://www.afpbb.com/articles/-/3117
http://www.itfarrag.com/Articles/Det
http://www.boohoo.com/restofworld/ne
http://myemail.constantcontact.com/A
http://www.cnn.com/2017/02/15/politi
http://www.abc.net.au/news/2015-08-1
http://medium.com/m/global-identity?
http://www.sonhaberler.com/dunya/mac
https://www.facebook.com/molly.macra
http://www.inc.com/lolly-daskal/how-
http://pdwxn.milfalone.com/c/da57dc5
http://ironmaidenlegacy.com/
http://www.cnn.com/2017/02/13/politi
http://dailycash.gift/articles/99085
http://www.cnn.com/politics
http://www.cnn.com/2017/02/15/politi
http://www.nytimes.com/2017/02/15/up
```

Figure 4: Sample of file containing no. of mementos

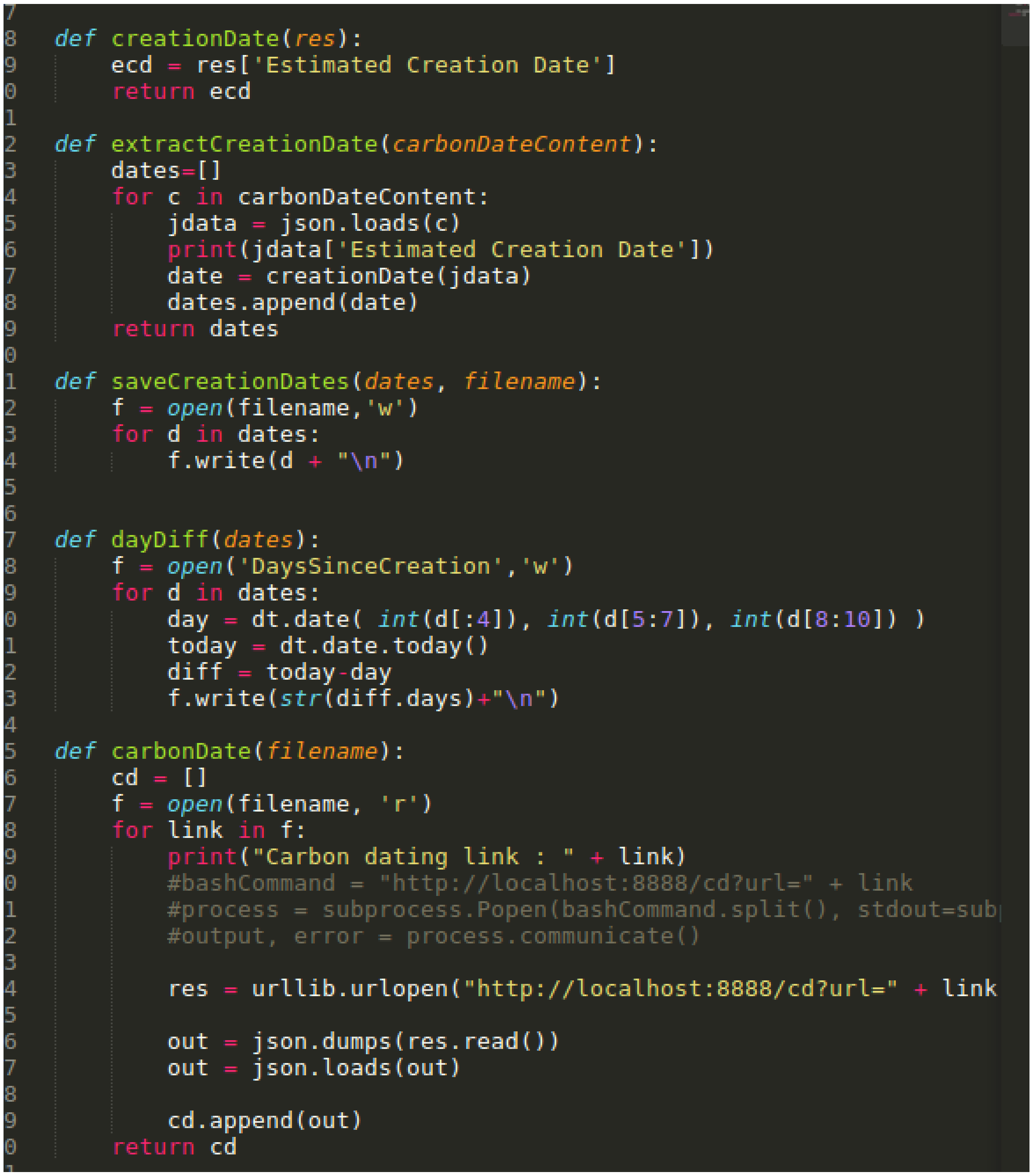
Finally, the mementos are plotted by loading the "memSum.txt" data to RStudio and running a simple script. The following figures are barplots of the same data, with the second one zoomed in to an area of interest.


# of URIs vs Mementos
Num of URIs
500
200
100
50
20
10
5
2
1
0
1
2
4
7
13
26
51
99
209
483
1223
3396
9426
28705
95913
Num of Mementos


# of URIs vs Mementos
Num of URIs
1000
500
100
50
10
5
1
0
3
6
9
13
18
23
28
33
38
43
48
53
58
63
68
73
78
83
88
Num of Mementos

Problem 3: Estimating the age of the 1000 URIs

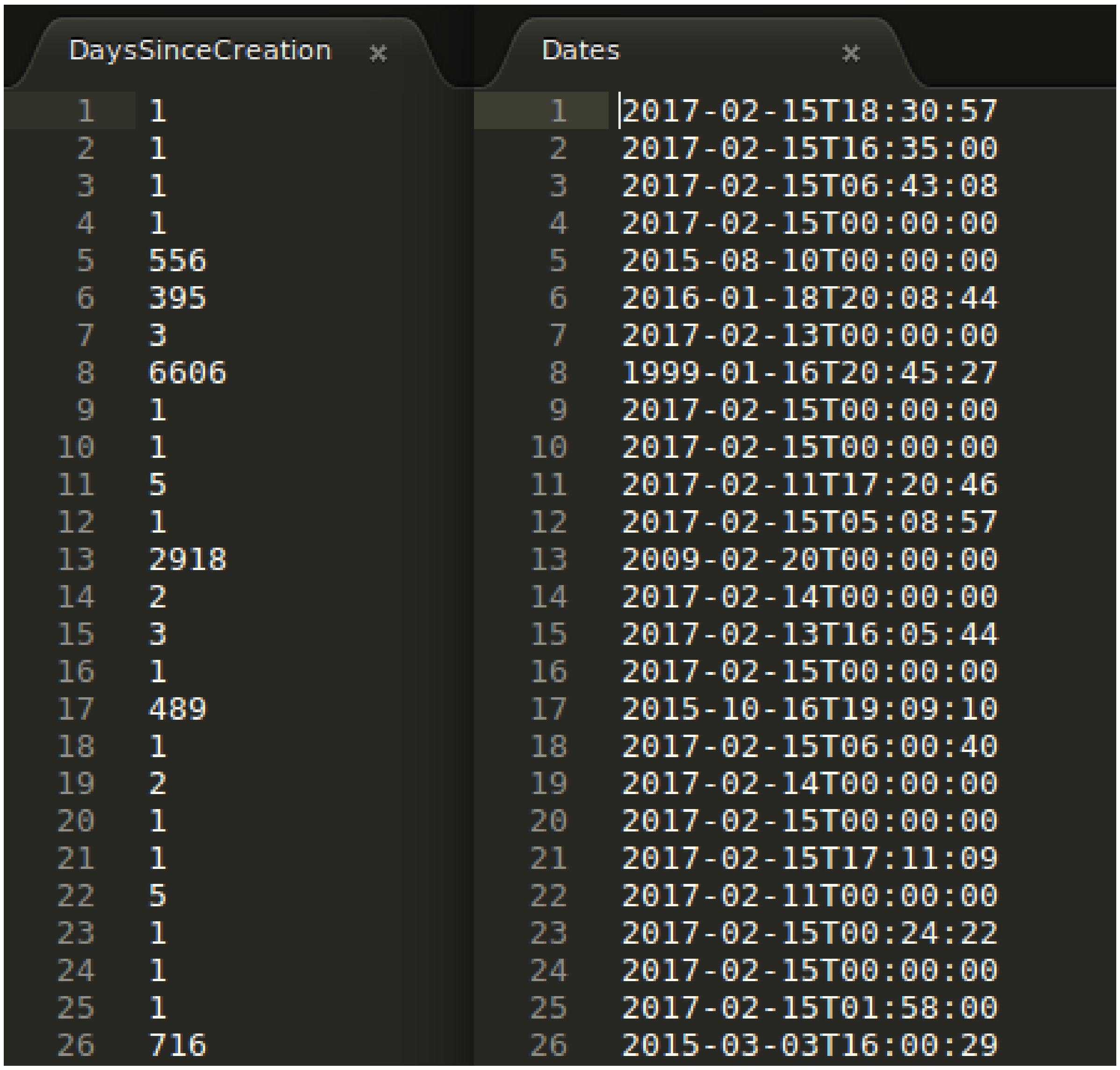
In order to complete this part of the assignment Docker was installed and operated as a server. Having created an image of the CarbonDate repository on the Docker, the server was able to recieve requests and not overload the CS departments CarbonDate server.



```python
def creationDate(res):
    ecd = res['Estimated Creation Date']
    return ecd

def extractCreationDate(carbonDateContent):
    dates=[]
    for c in carbonDateContent:
        jdata = json.loads(c)
        print(jdata['Estimated Creation Date'])
        date = creationDate(jdata)
        dates.append(date)
    return dates

def saveCreationDates(dates, filename):
    f = open(filename,'w')
    for d in dates:
        f.write(d + "\n")


def dayDiff(dates):
    f = open('DaysSinceCreation','w')
    for d in dates:
        day = dt.date( int(d[:4]), int(d[5:7]), int(d[8:10]) )
        today = dt.date.today()
        diff = today-day
        f.write(str(diff.days)+"\n")

def carbonDate(filename):
    cd = []
    f = open(filename, 'r')
    for link in f:
        print("Carbon dating link : " + link)
        #bashCommand = "http://localhost:8888/cd?url=" + link
        #process = subprocess.Popen(bashCommand.split(), stdout=sub
        #output, error = process.communicate()

        res = urllib.urlopen("http://localhost:8888/cd?url=" + link

        out = json.dumps(res.read())
        out = json.loads(out)

        cd.append(out)
    return cd
```

Figure 5: Functions used to obtain the "Estimated Creation Date" of the URIs

After succesfully retrieving the CarbonDate information for each link, the "Estimated Creation Date" was isolated and saved to a text file to be loaded into RStudio along with the corresponding number of memntos for those URIs.



```
1
1
1
1
556
395
3
6606
1
1
5
1
2918
2
3
1
489
1
2
1
1
5
1
1
1
716
```

```
2017-02-15T18:30:57
2017-02-15T16:35:00
2017-02-15T06:43:08
2017-02-15T00:00:00
2015-08-10T00:00:00
2016-01-18T20:08:44
2017-02-13T00:00:00
1999-01-16T20:45:27
2017-02-15T00:00:00
2017-02-15T00:00:00
2017-02-11T17:20:46
2017-02-15T05:08:57
2009-02-20T00:00:00
2017-02-14T00:00:00
2017-02-13T16:05:44
2017-02-15T00:00:00
2015-10-16T19:09:10
2017-02-15T06:00:40
2017-02-14T00:00:00
2017-02-15T00:00:00
2017-02-15T17:11:09
2017-02-11T00:00:00
2017-02-15T00:24:22
2017-02-15T00:00:00
2017-02-15T01:58:00
2015-03-03T16:00:29
```

Figure 6: Sample of CarbonDate results

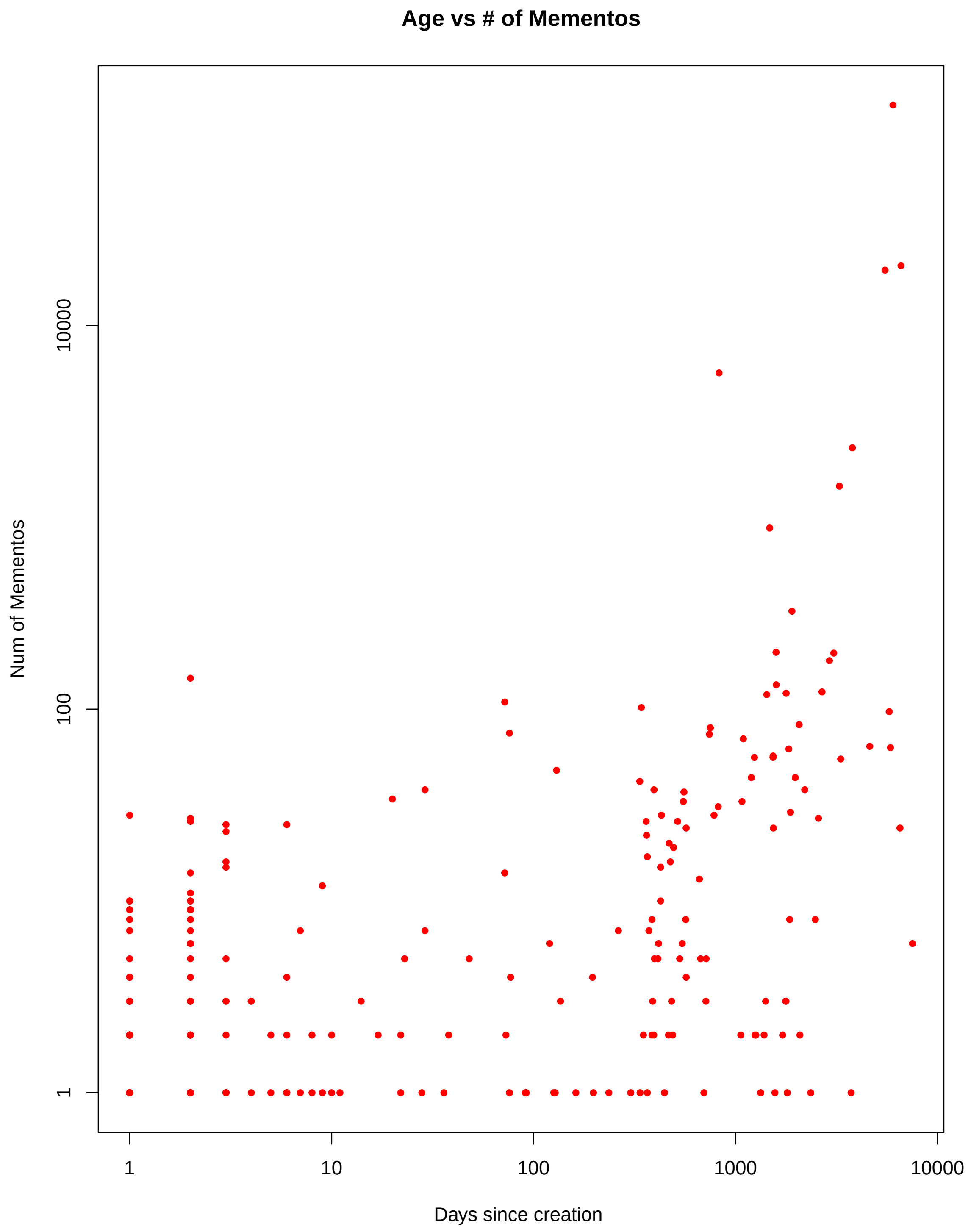


Figure 7: Sample of CarbonDate results