Assignment 2

CS 532: Introduction to Web Science
Spring 2017
Grant Atkins
Finished on February 9, 2017

1

Question

1. Write a Python program that extracts 1000 unique links from Twitter. You might want to take a look at:

http://adilmoujahid.com/posts/2014/07/twitter-analytics/

see also:

http://docs.tweepy.org/en/v3.5.0/index.html
https://github.com/bear/python-twitter
https://dev.twitter.com/rest/public

But there are many other similar resources available on the web. Note that only Twitter API 1.1 is currently available; version 1 code will no longer work.

Also note that you need to verify that the final target URI (i.e., the one that responds with a 200) is unique. You could have many different shortened URIs for www.cnn.com (t.co, bit.ly, goo.gl, etc.). For example:

```
$ curl -IL --silent https://t.co/DpO767Md1v | egrep -i "(HTTP/1.1|^location:)"
HTTP/1.1 301 Moved Permanently
location: https://goo.gl/40yQo2
HTTP/1.1 301 Moved Permanently
Location: https://soundcloud.com/roanoketimes
/ep-95-talking-hokies-recruiting-one-week-before-signing-day
HTTP/1.1 200 OK
```

You might want to use the search feature to find URIs, or you can pull them from the feed of someone famous (e.g., Tim O'Reilly). If you find something inappropriate for any reason you see fit, just discard it and get some more links. We just want 1000 links that were shared via Twitter.

Hold on to this collection and upload it to github -- we'll use it later throughout the semester.

Answer

To first approach this problem I started out with the the template code provided by Adil Moujahid in a blog post about twitter analytics [1]. I liked the fact that you could get tweets real time so I followed this approach of streaming tweets live, only later did I realize this took approximately 12 hours to get unique URIs. I wrote this code in python 3.6 and used the following dependencies:

- requests

- tweepy

You tweepy library allows the developer to stream tweets based on keywords. For my keywords I chose some of my favorite jazz and funk musicians and bands that were saved to my youtube playlist at the time, `https://www.youtube.com/playlist?list=PLYcZodEQpdPcZjZks1IV2V9nMPWNvzoxK`, to filter these streams.

The main program for twitter streaming was **tweepyCrawler.py** shown in Listing 1. It first starts by listening to the stream of data from twitters streaming API. Then once a a piece of data is received in JSON format, I parsed the number of mementos found in the JSON key "urls" which provided the urls of a tweet. The JSON could also have a *"retweeted_status"* key which would contain more URIs provided from the tweet referenced, so I also included those URIs.

```
 1  #!/usr/bin/env python3
 2
 3  #Import the necessary methods from tweepy library
 4  import mmap
 5  import requests
 6  import json
 7  import os
 8  from datetime import datetime
 9  from tweepy.streaming import StreamListener
10  from tweepy import OAuthHandler
11  from tweepy import Stream
12
13  #Variables that contains the user credentials to access Twitter
        API
14  access_token = "821042028800802816 -
        E7SvwPXZKJRzazLctidudXhD0X0SgDZ"
15  access_token_secret = "
        hfEMDTkVBX6Kf7x8FddjBZi7joxKZIYYJztq1QFQcF8cp"
16  consumer_key = "RigRve4McsZdYXNpz2rwPRZfx"
```

```
17  consumer_secret = "
        EuFivjFeWCBmG205shXMjTPb0u56wTXJgRDRhqaWPRQU1CxYjW"
18  # Filter list for bad/inappropriate/repeating domains
19  blacklist = ['.xyz','.pw','http://artist-rack.com?','https://
        twitter.com/i/web/status/','paper.li']
20
21
22  def request(uri):
23      try:
24          resp = requests.get(uri,stream=True,timeout=5,
                allow_redirects=True,headers={'User-Agent':'Mozilla
                /5.0'})
25          if resp.status_code == 200:
26              print("Original URI:",uri)
27              uriFinal = resp.url
28              filteredURI = uriFilter(uriFinal)
29              if filteredURI is not None:
30                  uriFinal = filteredURI
31                  print("Final URI:",uriFinal)
32                  saveOutput(uri,uriFinal)
33      except KeyboardInterrupt:
34          print()
35          exit()
36      except:
37          pass
38
39
40  def uriFilter(uri):
41      for f in blacklist:
42          # for unique youtube URIs just get video id
43          if 'youtube.com' in uri and '&' in uri:
44              pos = uri.find('&')
45              finalURI = uri[:pos]
46              return finalURI
47          elif f in uri:
48              return
49      return uri
50
51
52  def saveOutput(origUri,finalUri):
53
54      if not os.path.exists("output"):
55          os.makedirs("output")
56
57      # final URIs to file
58      try:
59          with open('output/finalURIs.txt', 'a+b', 0) as file, \
60              mmap.mmap(file.fileno(), 0, access=mmap.ACCESS_READ)
                    as s:
```

```
61              if s.find(bytes(finalUri, encoding='utf-8')) != -1:
62                  return
63              else:
64                  file.write(bytes(finalUri+"\n", encoding='utf
                        -8'))
65      except (IOError, ValueError):
66          with open('output/finalURIs.txt', 'w') as file:
67              file.write(finalUri+"\n")
68
69      # write original URIs to separate file
70      try:
71          with open("output/originalURIs.txt", "a") as file:
72              file.write(origUri+"\n")
73      except (IOError, ValueError):
74          with open("output/originalURIs.txt", "w") as file:
75              file.write(origUri+"\n")
76
77
78  # This is a basic listener that just prints received tweets to
        stdout.
79  # Consider this the main class that calls the functions from
        before
80  class StdOutListener(StreamListener):
81
82      def on_data(self, data):
83          print (data)
84          jsonData = json.loads(data)
85          uriArr = jsonData['entities']['urls']
86          for item in uriArr:
87              uri = item['url']
88              request(uri)
89
90          # handle retweet json
91          if 'retweeted_status' in jsonData:
92              uriArr = jsonData['retweeted_status']['entities']['
                    urls']
93              for item in uriArr:
94                  uri = item['url']
95                  request(uri)
96
97          print("finished requests")
98          return True
99
100     def on_error(self, status):
101         print (status)
102
103
104 if __name__ == '__main__':
105
```

```
106        #This handles Twitter authetification and the connection to
               Twitter Streaming API
107        l = StdOutListener()
108        auth = OAuthHandler(consumer_key, consumer_secret)
109        auth.set_access_token(access_token, access_token_secret)
110        try:
111            stream = Stream(auth, l)
112            #This line filter Twitter Streams to capture data by the
                    keywords
113            stream.filter(track=['jazz music', 'marcus miller', '
                    victor wooten','bill evans','the seatbelts','james
                    brown','snarky puppy','jamiroquai','jazz messengers
                    '])
114        except KeyboardInterrupt:
115            print()
116            exit()
117        except:
118            print("Error occurred − SHUTTING DOWN\n", str(datetime.
                    now()))
119            exit()
```

Listing 1: Python script for twitter streaming

Since I was taking live tweets it was inevitable that I would get repeating
URIs, inappropriate URIs, or just javascript based HTML pages with no
content. Therefore, when receiving the JSON I would perform an HTTP
get request to the URI provided then check if the final URI was already in
my output file, if it wasn't I would add it. I also added a blacklist of URIs
and domain types that were in the bad category. Sometimes this searching
for unique URIs didn't work, especially when it came to youtube URIs
that had extra parameters inside them. To compensate for this I created
filterUnwanted.py shown Listing 2 in and **removeDuplicates.py** shown
in Listing 3. The first script would strip youtube of their extra parameters
except for their video ids and it would also remove bad URIs. The second
script would make sure all the URIs were still unique in the file, removing
duplicates if they were found.

```
1  #!/usr/bin/env python3
2
3  # unwanted domains
4  blacklist = ['.xyz','.pw','http://artist−rack.com?','https://
        twitter.com/i/web/status/','paper.li','twitter.com/']
5  # track lines removed from finalURIs for originalURIs
6  removedLines = []
7
8
9  # Remove unwanted from finalURIs
```

```
10   with open ("output/finalURIs.txt","r+") as file :
11       lines = file.readlines()
12       for num, line in enumerate(lines):
13           for f in blacklist:
14               if f in line:
15                   removedLines.append(num)
16                   print(num,"type found:",f,":",line)
17                   del lines[num]
18                   break
19               elif 'youtube.com' in line and '&' in line:
20                   pos = line.find('&')
21                   finalURI = line[:pos]
22                   print("FINALURI:",finalURI)
23                   lines[num] = finalURI+"\n"
24                   break
25       file.truncate(0)          # truncates the file
26       file.seek(0)             # moves the pointer to the start of
                 the file
27       file.writelines(lines)   # write the new data to the file
28
29
30   # remove from originalURIs as well
31   with open("output/originalURIs.txt","r+") as file :
32       lines = file.readlines()
33       for num, line in enumerate(lines):
34           for badLineNum in removedLines:
35               if num == badLineNum:
36                   print(badLineNum,"Line Removed:",line)
37                   lines.remove(line)
38                   break
39       file.truncate(0)          # truncates the file
40       file.seek(0)             # moves the pointer to the start of
                 the file
41       file.writelines(lines)   # write the new data to the file
```

Listing 2: Python script for filtering and cleaning data files

```
 1   #!/usr/bin/env python3
 2
 3   # unwanted domains
 4   blacklist = ['.xyz','.pw','http://artist-rack.com?','https://
         twitter.com/i/web/status/','paper.li','twitter.com/']
 5   # track lines removed from finalURIs for originalURIs
 6   removedLines = []
 7
 8
 9   # Remove unwanted from finalURIs
10   with open("output/finalURIs.txt","r+") as file :
11       lines = file.readlines()
```

```
12        for num, line in enumerate(lines):
13            for f in blacklist:
14                if f in line:
15                    removedLines.append(num)
16                    print(num,"type found:",f,":",line)
17                    del lines[num]
18                    break
19                elif 'youtube.com' in line and '&' in line:
20                    pos = line.find('&')
21                    finalURI = line[:pos]
22                    print("FINALURI:",finalURI)
23                    lines[num] = finalURI+"\n"
24                    break
25        file.truncate(0)              # truncates the file
26        file.seek(0)                  # moves the pointer to the start of
                 the file
27        file.writelines(lines)    # write the new data to the file
28
29
30   # remove from originalURIs as well
31   with open("output/originalURIs.txt","r+") as file:
32        lines = file.readlines()
33        for num, line in enumerate(lines):
34            for badLineNum in removedLines:
35                if num == badLineNum:
36                    print(badLineNum," Line Removed:",line)
37                    lines.remove(line)
38                    break
39        file.truncate(0)              # truncates the file
40        file.seek(0)                  # moves the pointer to the start of
                 the file
41        file.writelines(lines)    # write the new data to the file
```

Listing 3: Python script for removing duplicates in data files

2

Question

2. Download the TimeMaps for each of the target URIs. We'll use the ODU
Memento Aggregator, so for example:

URI-R = http://www.cs.odu.edu/

URI-T = http://memgator.cs.odu.edu/timemap/link/http://www.cs.odu.edu/

or:

URI-T = http://memgator.cs.odu.edu/timemap/json/http://www.cs.odu.edu/

(depending on which format you'd prefer to parse)

Create a histogram* of URIs vs. number of Mementos (as computed
from the TimeMaps). For example, 100 URIs with 0 Mementos, 300
URIs with 1 Memento, 400 URIs with 2 Mementos, etc. The x-axis
will have the number of mementos, and the y-axis will have the
frequency of occurence.

* = https://en.wikipedia.org/wiki/Histogram

What's a TimeMap?
See: http://www.mementoweb.org/guide/quick-intro/
And the week 4 lecture.

Answer

For this problem I used `http://memgator.cs.odu.edu/` and wrote **timeMap.py** to retrieve a JSON response for the time maps of my 1000 URIs. If the response was "404 not found" it was assumed that there were no mementos for the URI. If there was a JSON response, I took the count of all archives provided in the JSON key "list". For each response I saved the URI and the count to a CSV file named **timeMaps.csv**. I then created a simple histogram, using R shown in Listing 4, of URIs vs. Number of Mementos as shown in Figure 1. An overwhelming majority of the URIs had either no mementos or a very low amount.

```
1  setwd(getwd())
2  # csv dataframe of URI, number of mementos
3  numMementos <- read.table(header = FALSE, sep = ",", 'output/
       timeMaps.csv')
4  # histogram
5  hgram <- hist(numMementos$V2, col = "gray", breaks = seq(from=0,
       to=20000, by=1000), main="URIs vs. Number of Mementos", xlab =
       "Number of Mementos")
6  # add count labels
7  text(hgram$mids, hgram$counts, adj=c(0.5, -0.5), labels=
       hgram$counts)
```
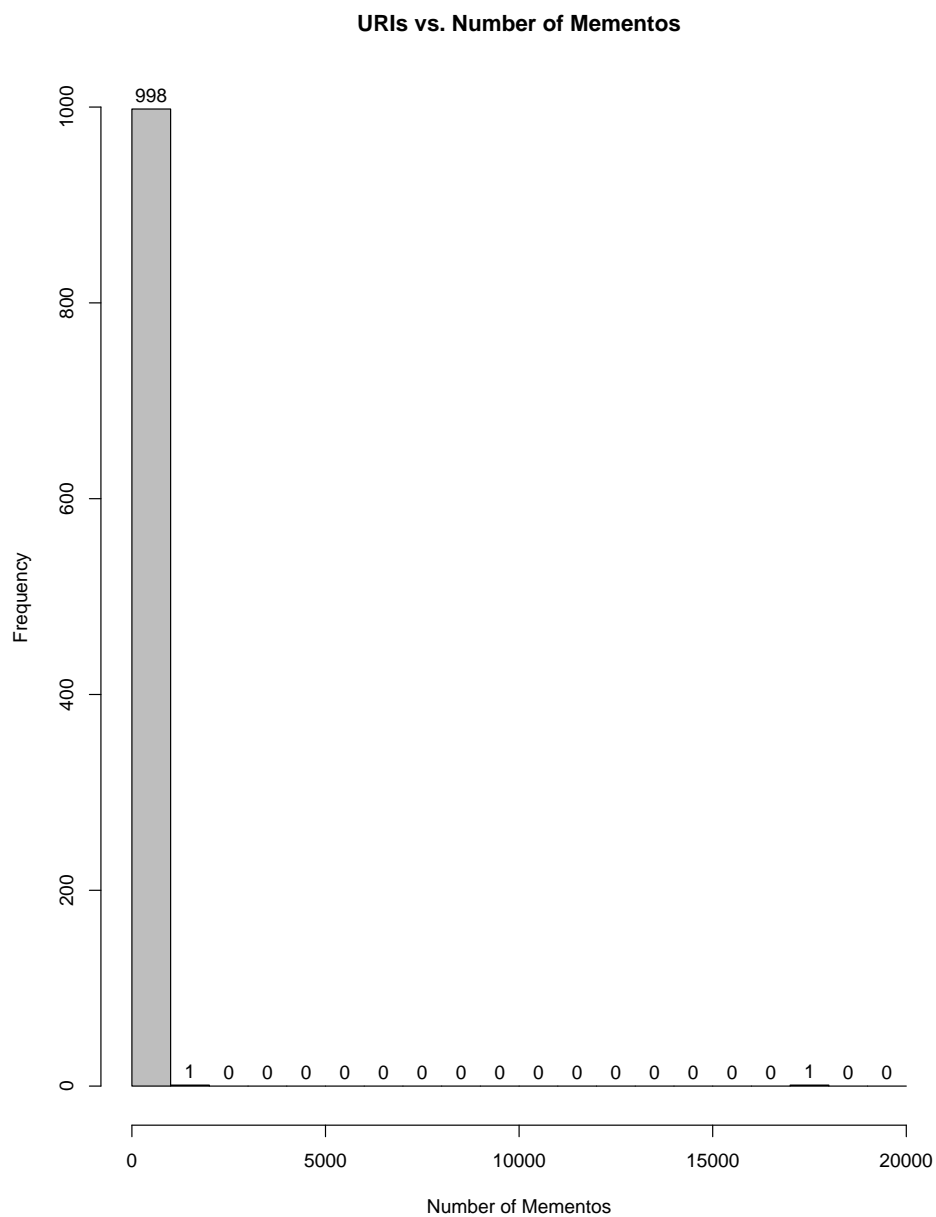
Listing 4: R script for creating Histogram

URIs vs. Number of Mementos

Figure 1: Histogram of Number of URIs vs. Number of Mementos

3

Question

3. Estimate the age of each of the 1000 URIs using the "Carbon Date" tool:

http://ws-dl.blogspot.com/2016/09/2016-09-20-carbon-dating-web-version-30.html

Note: you should use "docker" and install it locally. You can do it like this:

http://cd.cs.odu.edu/cd?url=http://www.cs.odu.edu/

But it will inevitably crash when everyone tries to use it at the last minute.

For URIs that have > 0 Mementos and an estimated creation date, create a graph with age (in days) on the x-axis and number of mementos on the y-axis.

Not all URIs will have Mementos, and not all URIs will have an estimated creation date. Show how many fall into either categories. For example,

```
total URIs:          1000
no mementos:      137
no date estimate:  212
```

Answer

Using the memento counts retrieved from question 2, I used the carbon dating tool provided in the question to retrieve the estimated creation dates for each of these URIs [2]. For each response I saved the URI and the "Estimated Creation Date" JSON key to a CSV file named **carbonDate.csv**. The code used to complete this task is displayed in Listing 5.

```python
#!/usr/bin/env python3

import requests
import os
import json
import csv


def saveOutput(uri, estDate):

    if not os.path.exists("output"):
        os.makedirs("output")

    # Save json received/created to output
    fields = [uri, estDate]
    try:
        with open('output/carbonDate.csv', 'a', newline='') as
              file:
            writer = csv.writer(file, delimiter=',')
            writer.writerow(fields)
    except (IOError, ValueError):
        with open('output/carbonDate.csv', 'w', newline='') as
              file:
            writer = csv.writer(file, delimiter=',')
            writer.writerow(fields)


def getJson(jsonResp):
    data = json.loads(jsonResp)
    estDate = data["Estimated Creation Date"]
    return estDate


startFlag = False
with open("output/finalURIs.txt", "r") as file:
    lines = file.readlines()
    for num, line in enumerate(lines):
        try:
            carbonDateURI = "http://localhost:8888/cd?url=" +
                  line
```

```
38              print(carbonDateURI)
39              resp = requests.get(carbonDateURI, stream=True,
                    allow_redirects=True, headers={
40                                  'User-Agent': 'Mozilla/5.0'})
41
42              if resp.status_code == 200:
43                  # count through json arr
44                  print(resp.text)
45
46                  estDate = getJson(resp.text)
47                  saveOutput(line, estDate)
48              else:
49                  estDate = ""
50                  saveOutput(line, estDate)
51
52          except KeyboardInterrupt:
53              print()
54              exit()
55          except:
56              pass
```

Listing 5: Python script for retrieving estimated creation date

Once I retrieved all of the dates I merged both the **carbonDate.csv**
and **timeMaps.csv** together on their URIs using **mergeCSVs.py**, shown
in Listing 6. At the same time if the URI had an estimated creation date,
the number of days difference from the current date (02/09/2017) to the
date provided would be determined. Finally I created a scatter plot, built
using R shown in Listing 7, depicting the the Days vs. Number of Mementos
shown in Figure 2.

The following values were found using **carbonDateMementos.R**:

- total URIs: 1000

- no mementos: 949

- no date estimate: 266

```
1  import csv
2  import os
3  import datetime
4
5
6  def findDays(datePassed):
7      try:
8          now = datetime.datetime.today()
9          datePassed = datetime.datetime.strptime(datePassed, "%Y
                -%m-%dT%H:%M:%S")
```

13

```
10          days = (now − datePassed).days
11          return days
12      except:
13          return ""
14
15
16  def mergeCSVs():
17
18      if not os.path.exists("output"):
19          os.makedirs("output")
20
21      desiredRows = []
22
23      # Save json received/created to output
24      with open('output/timeMaps.csv', 'r', newline='') as timeMaps
            , open('output/carbonDate.csv', 'r', newline='') as
            carbonDates:
25
26          reader = csv.reader(timeMaps)
27          reader2 = csv.reader(carbonDates)
28
29          for row in reader:
30              temp = []
31              temp.append(row[0])
32              temp.append(row[1])
33              for row2 in reader2:
34                  if(row[0] == row2[0]):
35                      days = findDays(row2[1])
36                      temp.append(days)
37                      break
38
39              desiredRows.append(temp)
40
41      with open('output/carbonDateMerged.csv', 'w', newline='') as
            file:
42          for row in desiredRows:
43              writer = csv.writer(file, delimiter=',')
44              writer.writerow(row)
45
46
47  mergeCSVs()
```

Listing 6: Python script to merge CSVs and determine date

```
1  setwd(getwd())
2  # Merged Dataset with memento count and days
3  mementoDays <- read.table(header = FALSE,sep = ",",'output/
       carbonDateMerged.csv')
4  # filtered set w/ atleast 1 memento
```

14

```
5   withMementos <- subset(mementoDays, mementoDays$V2 > 0)
6   emptyDate <- subset(mementoDays, is.na(mementoDays$V3))
7   plot(withMementos$V3,withMementos$V2,ylab="Number of Mementos",
        xlab="Age in Days",main="Scatter Plot for Days vs. Mementos")
```

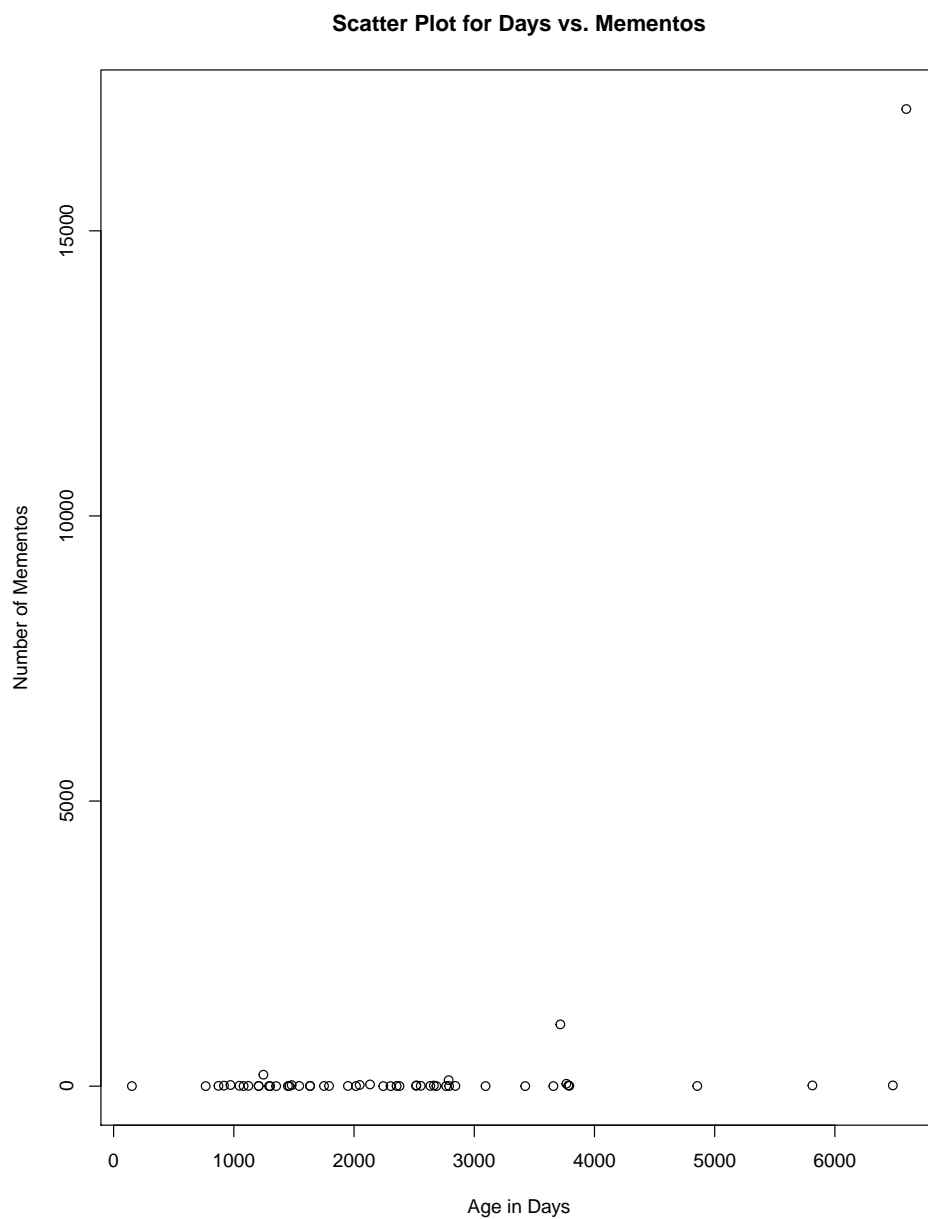Listing 7: R Script to find create scatterplot and determine values

Scatter Plot for Days vs. Mementos

Figure 2: Scatter plot of age in days and number of mementos

References

[1] Moujahid, Adil. "An Introduction to Text Mining Using Twitter Streaming API and Python." An Introduction to Text Mining Using Twitter Streaming API and Python // Adil Moujahid // Data Analytics and More. N.p., 21 July 2014. Web. 08 Feb. 2017. `http://adilmoujahid.com/posts/2014/07/twitter-analytics/`

[2] Zetan, Li. "2016-09-20: Carbon Dating the Web, Version 3.0." 2016-09-20: Carbon Dating the Web, Version 3.0. N.p., 20 Sept. 2016. Web. 08 Feb. 2017. `http://ws-dl.blogspot.com/2016/09/2016-09-20-carbon-dating-web-version-30.html`