Assignment 2 **Hussam Hallak**
CS532, Web Science, Spring 2017 CS Master's Student
Old Dominion University, Computer Science Dept Prof: Dr. Nelson

Question 1:

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: Only Twitter API 1.1 is currently available; version 1 code will no longer work.

Note: 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:

Listing 1: Command:

```
$ 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:

The approach is divided into two steps:

1. Collect tweets with external links, parse them and save the unique external links in the file "links.txt". I wrote a python script to do that "getlinks.py". This python script must be run several times until 1000 links or more are collected (more duplicates are removed later in the second step). The script asks the user to enter twitter username(s) to pull their last tweets that contain links. Twitter username(s) are passed to the program as command line argument(s). The program loops over all usernames and pulls their tweets that contain links. It follows all redirects until 200 OK is returned. It removes links that lead to other tweets, not external to Twitter. The program removes duplicates in the captured links and writes them to the file "links.txt". Twitter API credentials are saved in a separate file, "config.py".

Listing 2: The content of config.py

```
#Get your Twitter API credentials and enter them here
consumer_key = "put_your_consumer_key_here"
consumer_secret = "put_your_consumer_secret_here"
access_key = "put_your_access_key_here"
access_secret = "put_your_access_secret_here"
```

Listing 3: The content of getlinks.py

```
#!/usr/bin/env python
import sys
import re
import requests
import urllib2
from urlparse import urlparse
import tweepy
#config.py contains your twitter's API consumer_key, consumer_secret, access_key,
    and access_secret
config = {}
execfile("config.py", config)
#method to get a user(s) last 200 tweets
def get_tweets(username):
   #http://tweepy.readthedocs.org/en/v3.1.0/getting_started.html#api
   auth = tweepy.OAuthHandler(config["consumer_key"], config["consumer_secret"])
   auth.set_access_token(config["access_key"], config["access_secret"])
   api = tweepy.API(auth)
   #set count to however many tweets you want; twitter only allows 200 at once
   number_of_tweets = 200
   #get tweets
   tweets = api.user_timeline(screen_name = username,count = number_of_tweets)
   final_urls = []
   for tweet in tweets:
      urls = re.findall('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\)
          ,]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', tweet.text)
      for url in urls:
         #keep external links and throw out links in tweets that lead to other
             tweets
         try:
            res = urllib2.urlopen(url)
            actual_url = res.geturl()
            parsed_uri = urlparse(actual_url)
            domain = '{uri.scheme}://{uri.netloc}/'.format(uri=parsed_uri)
            if (domain != 'https://twitter.com/' and domain != 'https://t.co/'):
               final_urls.append(actual_url)
         except:
            print "Discarded url: The url is not external: " + url
   fh = open ("links.txt", "a")
   #remove duplicates
   final_urls = list(set(final_urls))
   #save links
   for link in final_urls:
      fh.write(link)
      fh.write("\n")
   fh.close()
```

```
if __name__ == '__main__':
   if len(sys.argv) >= 2:
      i = 1
      #loop over all users entered as command line arguments and get their tweets
           (200 total allowed)
      for user in sys.argv:
           if (i < len(sys.argv)):
              get_tweets(sys.argv[i])
              i = i + 1
   else:
      print "Error: You did not enter twitter username(s)"
      print "Usage: Python get_links.py <twitter_username1> <twitter_username2> <
          twitter_username3> ...etc"
      print "e.g: Python get_links.py phonedude_mln HussamHallak1 weiglemc cnn
          Aljazeera"
      print "Note 1: Twitter limits this program to 200 tweets per request!"
      print "Note 2: Links not external to Twitter are not saved!"
```

Running the Program:

Listing 4: Running getlinks.py to collects external links from cnn's tweets

```
root@ima-app:/var/www/Hussam# python getlinks.py cnn
Discarded url: The url is not external: https://t.co/qE
Discarded url: The url is not external: https://t
root@ima-app:/var/www/Hussam# ls
build  config.py config.pyc getlinks.py links.txt make_unique.py
root@ima-app:/var/www/Hussam#
```

The program created the file "links.txt" and stored the links in it.

Note: The program discarded two of the captured links because they are not external. It shows what the discarded links are.

After running the program enough times to extract tweets with links from different users, I collected 1405 links.

Listing 5: The number of external links extracted from tweets after multiple runs

```
root@ima-app:/var/www/Hussam# cat links.txt | wc -l
1405
```

2. Ensuring that the extracted 1405 links are unique: After running the program multiple times to extract enough links to have 1000 unique links, now we need to make sure that the extracted links from multiple runs are unique.

The script "makeunique.py" is a simple script that takes 2 command line arguments, an input file that we want to remove duplicated links from, and an output file to store unique links in.

Listing 6: The content of makeunique.py

```
import sys

if len(sys.argv) < 3:
   print "Usage: Python makeunique.py <input_file> <output_file>"
   print "e.g: Python makeunique.py links.txt uniquelinks.txt"
else:
   seen_lines = set()
```

```
outputfile = open(sys.argv[2], "w")
for line in open(sys.argv[1], "r"):
      if line not in seen_lines:
         outputfile.write(line)
         seen_lines.add(line)
outputfile.close()
```

After running "makeunique.py" on "links.txt" and saving the output in "uniquelinks.txt", I got 1107 unique links out of the 1405 links in "links.txt". 298 links were duplicates.

Listing 7: The number of unique links in links.txt saved in uniquelinks.txt

```
root@ima-app:/var/www/Hussam# python makeunique.py links.txt uniquelinks.txt
root@ima-app:/var/www/Hussam# cat uniquelinks.txt | wc -l
1107
```

Included Files:

config.py getlinks.py, makeunique.py, links.txt, uniquelinks.txt, getlinks.run.txt

Question 2:

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

Listing 8: Command:

```
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:

The approach is divided into three steps:

1. Downloading the TimeMaps for each of the target URIs using the ODU Memento Aggregator. I wrote a python script to do that "downloadtimemapsjson.py". The script takes three command line arguments, an input file that has the links for which we want to download the TimeMaps "uniquelinks.txt", an output file to store the downloaded TimeMaps, "timemap.jsonx" where "x" is a number that identifies the timemap (e.g., 1,

2, 3, ...), and another output file, "uniquelinkswithmemes.txt", to store the unique links with mementos in.

Note: The number of output files named , "timemap.jsonx" may not be the same as the number of links in the input file "uniquelinks.txt". The reason is because not all links have mementos.

Note: An interesting discovery is that, sometimes, links with certain query string variables, GET variables, will make our results inaccurate because they have no mementos for the link as it is, however, if these variables are removed, the page has mementos. Some websites add these arguments so they can log where the user came from to visit their page.

Example: this link is coming back with 404 error code, no mementos!

`http://memgator.cs.odu.edu/timemap/link/http://www.bbc.co.uk/news/business-37047416?utm_source=twitterfeed&utm_medium=twitter`

The original link comes with 200 OK:

`http://www.bbc.com/news/business-37047416?utm_source=twitterfeed&utm_medium=twitter`

Listing 9: This shows that the page has no mementos

```
root@ima-app:/var/www/Hussam# curl http://memgator.cs.odu.edu/timemap/link/http
    ://www.bbc.co.uk/news/business-37047416?utm_source=twitterfeed&utm_medium=
    twitter
[1] 21757
root@ima-app:/var/www/Hussam# 404 page not found
[1]+ Done                    curl http://memgator.cs.odu.edu/timemap/link/http://
    www.bbc.co.uk/news/business-37047416?utm_source=twitterfeed
root@ima-app:/var/www/Hussam# curl -s -o /dev/null -v http://www.bbc.com/news/
    business-37047416?utm_source=twitterfeed&utm_medium=twitter
[1] 21825
root@ima-app:/var/www/Hussam# * About to connect() to www.bbc.com port 80 (#0)
*   Trying 151.101.32.81... connected
* Connected to www.bbc.com (151.101.32.81) port 80 (#0)
> GET /news/business-37047416?utm_source=twitterfeed HTTP/1.1
> User-Agent: curl/7.21.3 (x86_64-pc-linux-gnu) libcurl/7.21.3 OpenSSL/0.9.8o
    zlib/1.2.3.4 libidn/1.18
> Host: www.bbc.com
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: Apache
< X-Cache-Action: MISS
< X-Cache-Age: 0
< Content-Type: text/html; charset=utf-8
< X-News-Data-Centre: cwwtf
< Content-Language: en
< X-PAL-Host: pal099.back.live.cwwtf.local:80
< X-News-Cache-Id: 94852
< Content-Length: 141821
< Accept-Ranges: bytes
< Date: Mon, 06 Feb 2017 22:22:29 GMT
< Via: 1.1 varnish
< Age: 0
< Connection: keep-alive
< X-LB-NoCache: true
< X-Fastly-Cache-Status: MISS-CLUSTER
```

```
< Set-Cookie: BBC-UID=
    cf5725fb665484f03ccf168d3099914bc1d880c0fb75b6f4f58139214b5afd470curl%2F7
    .21.3%20%28x86_64-pc-linux-gnu%29%20libcurl%2F7.21.3%20OpenSSL%2F0.9.8o%20
    zlib%2F1.2.3.4%20libidn%2F1.18; expires=Fri, 05 Feb 2021 22:22:29 GMT; path
    =/; domain=.bbc.com
< Cache-Control: private, max-age=60, stale-while-revalidate
< X-Served-By: cache-iad2620-IAD
< X-Cache: MISS
< X-Cache-Hits: 0
< X-Timer: S1486419748.297126,VS0,VE898
< Vary: Accept-Encoding
<
{ [data not shown]
* Connection #0 to host www.bbc.com left intact
* Closing connection #0

[1]+ Done                    curl -s -o /dev/null -v http://www.bbc.com/news/
    business-37047416?utm_source=twitterfeed
root@ima-app:/var/www/Hussam#
```

Let's remove the string that contains these arguments:

```
?utm_source=twitterfeed&utm_medium=twitter
```

Now it's time to see if there are any mementos for this link:
`http://www.bbc.co.uk/news/business-37047416`

Listing 10: The mementos for our BBC link without the arguments

```
root@ima-app:/var/www/Hussam# curl http://memgator.cs.odu.edu/timemap/link/http
    ://www.bbc.co.uk/news/business-37047416
<http://www.bbc.co.uk/news/business-37047416>; rel="original",
<http://memgator.cs.odu.edu/timemap/link/http://www.bbc.co.uk/news/business
    -37047416>; rel="self"; type="application/link-format",
<http://web.archive.org/web/20160811123949/http://www.bbc.co.uk/news/business
    -37047416>; rel="first memento"; datetime="Thu, 11 Aug 2016 12:39:49 GMT",
<http://archive.is/20160811130939/http://www.bbc.co.uk/news/business-37047416>;
    rel="memento"; datetime="Thu, 11 Aug 2016 13:09:39 GMT",
<http://web.archive.org/web/20160811135254/http://www.bbc.co.uk/news/business
    -37047416>; rel="memento"; datetime="Thu, 11 Aug 2016 13:52:54 GMT",
<http://web.archive.org/web/20160811181904/http://www.bbc.co.uk/news/business
    -37047416>; rel="memento"; datetime="Thu, 11 Aug 2016 18:19:04 GMT",
<http://web.archive.org/web/20160811204207/http://www.bbc.co.uk/news/business
    -37047416>; rel="memento"; datetime="Thu, 11 Aug 2016 20:42:07 GMT",
<http://web.archive.org/web/20160811214214/http://www.bbc.co.uk/news/business
    -37047416>; rel="memento"; datetime="Thu, 11 Aug 2016 21:42:14 GMT",
<http://web.archive.org/web/20160811220633/http://www.bbc.co.uk/news/business
    -37047416>; rel="memento"; datetime="Thu, 11 Aug 2016 22:06:33 GMT",
<http://web.archive.org/web/20160812001442/http://www.bbc.co.uk/news/business
    -37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 00:14:42 GMT",
<http://web.archive.org/web/20160812004911/http://www.bbc.co.uk/news/business
    -37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 00:49:11 GMT",
<http://web.archive.org/web/20160812020130/http://www.bbc.co.uk/news/business
    -37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 02:01:30 GMT",
<http://web.archive.org/web/20160812034118/http://www.bbc.co.uk/news/business
    -37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 03:41:18 GMT",
```

<http://web.archive.org/web/20160812041952/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 04:19:52 GMT",
<http://web.archive.org/web/20160812051841/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 05:18:41 GMT",
<http://web.archive.org/web/20160812062056/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 06:20:56 GMT",
<http://web.archive.org/web/20160812062855/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 06:28:55 GMT",
<http://web.archive.org/web/20160812080938/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 08:09:38 GMT",
<http://web.archive.org/web/20160812104921/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 10:49:21 GMT",
<http://web.archive.org/web/20160812121019/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 12:10:19 GMT",
<http://web.archive.org/web/20160812143656/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 14:36:56 GMT",
<http://web.archive.org/web/20160812160028/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 16:00:28 GMT",
<http://web.archive.org/web/20160812184545/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 18:45:45 GMT",
<http://web.archive.org/web/20160812202225/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Fri, 12 Aug 2016 20:22:25 GMT",
<http://web.archive.org/web/20160813010047/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 01:00:47 GMT",
<http://web.archive.org/web/20160813020325/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 02:03:25 GMT",
<http://web.archive.org/web/20160813040810/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 04:08:10 GMT",
<http://web.archive.org/web/20160813061226/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 06:12:26 GMT",
<http://web.archive.org/web/20160813081111/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 08:11:11 GMT",
<http://web.archive.org/web/20160813100343/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 10:03:43 GMT",
<http://web.archive.org/web/20160813120030/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 12:00:30 GMT",
<http://web.archive.org/web/20160813142317/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 14:23:17 GMT",
<http://web.archive.org/web/20160813162341/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 16:23:41 GMT",
<http://web.archive.org/web/20160813180327/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 18:03:27 GMT",
<http://web.archive.org/web/20160813200908/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 20:09:08 GMT",
<http://web.archive.org/web/20160813221202/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sat, 13 Aug 2016 22:12:02 GMT",
<http://web.archive.org/web/20160814002539/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sun, 14 Aug 2016 00:25:39 GMT",
<http://web.archive.org/web/20160814022442/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sun, 14 Aug 2016 02:24:42 GMT",
<http://web.archive.org/web/20160814062518/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sun, 14 Aug 2016 06:25:18 GMT",
<http://web.archive.org/web/20160814081940/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sun, 14 Aug 2016 08:19:40 GMT",
<http://web.archive.org/web/20160814121621/http://www.bbc.co.uk/news/business
-37047416>; rel="memento"; datetime="Sun, 14 Aug 2016 12:16:21 GMT",

```
<http://web.archive.org/web/20160826123157/http://www.bbc.co.uk/news/business
    -37047416>; rel="memento"; datetime="Fri, 26 Aug 2016 12:31:57 GMT",
<http://web.archive.org/web/20161019000632/http://www.bbc.co.uk/news/business
    -37047416>; rel="memento"; datetime="Wed, 19 Oct 2016 00:06:32 GMT",
<http://web.archive.org/web/20161027193947/http://www.bbc.co.uk/news/business
    -37047416>; rel="last memento"; datetime="Thu, 27 Oct 2016 19:39:47 GMT",
<http://memgator.cs.odu.edu/timemap/link/http://www.bbc.co.uk/news/business
    -37047416>; rel="timemap"; type="application/link-format",
<http://memgator.cs.odu.edu/timemap/json/http://www.bbc.co.uk/news/business
    -37047416>; rel="timemap"; type="application/json",
<http://memgator.cs.odu.edu/timemap/cdxj/http://www.bbc.co.uk/news/business
    -37047416>; rel="timemap"; type="application/cdxj+ors",
<http://memgator.cs.odu.edu/timegate/http://www.bbc.co.uk/news/business
    -37047416>; rel="timegate"
root@ima-app:/var/www/Hussam#
```

Note: It is clear that the page without the arguments has many mementos, however there is no telling which arguments to remove in each and every extracted link to prevent this case, but since we discovered this one, I am going to quickly remove it from all extracted links in "uniquelinks.txt" using this find and replace command:

Listing 11: Command: Removing GET arguments

```
sed -i 's/utm_source=twitterfeed&utm_medium=twitter//g' uniquelinks.txt

root@ima-app:/var/www/Hussam# sed -i 's/utm_source=twitterfeed&utm_medium=twitter
    //g' uniquelinks.txt
root@ima-app:/var/www/Hussam# cat uniquelinks.txt | grep "utm_source=twitterfeed&
    utm_medium=twitter"
root@ima-app:/var/www/Hussam#
```

Syntax:
sed -i 's/old/new/g' file.txt
The command:
sed: Stream Editor
Options:
-i: in-place (Save the result in the same file)
Arguments:
s: substitute
old: old string
new: new string
g: global (Replace all occurrences)
file.txt: file name

Listing 12: The content of downloadtimemapsjson.py

```
import sys
import urllib2
import json
if len(sys.argv) != 4:
    print "Usage: Python downloadtimemapsjson.py <input_file> <output_file1> <
        output_file2>"
    print "e.g: Python downloadtimemapsjson.py uniquelinks.txt timemap.json
        uniquelinkswithmemes.txt"
else:
```

```
    i = 1
    fh_input = open(sys.argv[1], 'r')
    fh_output = open(sys.argv[3], 'w')
    for line in fh_input:
        try:
            link = "http://memgator.cs.odu.edu/timemap/json/" + line
            response = urllib2.urlopen(link)
            content = json.load(response)
            output_file_name = sys.argv[2] + str(i)
            fh_json_output = open(output_file_name, "w")
            json.dump(content, fh_json_output)
            fh_output.write(line)
       i = i + 1
            fh_json_output.close()
        except:
            print "This link came with an error code:"
            print "http://memgator.cs.odu.edu/timemap/json/" + line
    fh_input.close()
    fh_output.close()
```

Listing 13: Running downloadtimemapsjson.py

```
root@ima-app:/var/www/Hussam# python downloadtimemapsjson.py uniquelinks.txt
    timemap.json uniquelinkswithmemes.txt
```

Now I have 445 links with mementos out of the original 1107 unique links in the input file.

Listing 14: Links with mementos:

```
root@ima-app:/var/www/Hussam# cat uniquelinks.txt | wc -l
1107
root@ima-app:/var/www/Hussam# cat uniquelinkswithmemes.txt | wc -l
445
root@ima-app:/var/www/Hussam# ls timemap.json* | wc -l
445
```

2. Parse the downloaded TimeMaps, JSON files, to find the number of mementos for each URI that has a TimeMap. I wrote a Python script, "parsejsontimemap.py", to do that. The script loops over all TimeMaps' files "timemap.jsonx" and parses them to find the number of mementos for each link. The program saves the results in a file as output, "timemapreport.txt". This is the file that we will use to generate our histogram in R in the next step.

Listing 15: The content of parsejsontimemap.py

```
import sys
import json
import os
if len(sys.argv) != 3:
   print "Usage: Python parsejsontimemap.py <input_file> <output_file>"
   print "e.g: Python parsejsontimemap.py timemap.json timemapreport.txt"
else:
   fh_output = open(sys.argv[2] , 'w')
   for i in range(1,446):
```

```
    input_file_name = sys.argv[1] + str(i)
    data_file = open(input_file_name, 'r')
    data = json.load(data_file)
    N = len(data['mementos']['list'])
    N = str(N)
    fh_output.write(N)
    fh_output.write("\n")
fh_output.close()
```

Listing 16: Running parsejsontimemap.py

```
root@ima-app:/var/www/Hussam# python parsejsontimemap.py timemap.json
    timemapreport.txt
root@ima-app:/var/www/Hussam# cat timemapreport.txt | wc -l
445
```

3. Now we are ready to generate the histogram for URIs vs. the number of Mementos in R.

This line of code reads the data in the file and stores it in the vector timemaps_data.

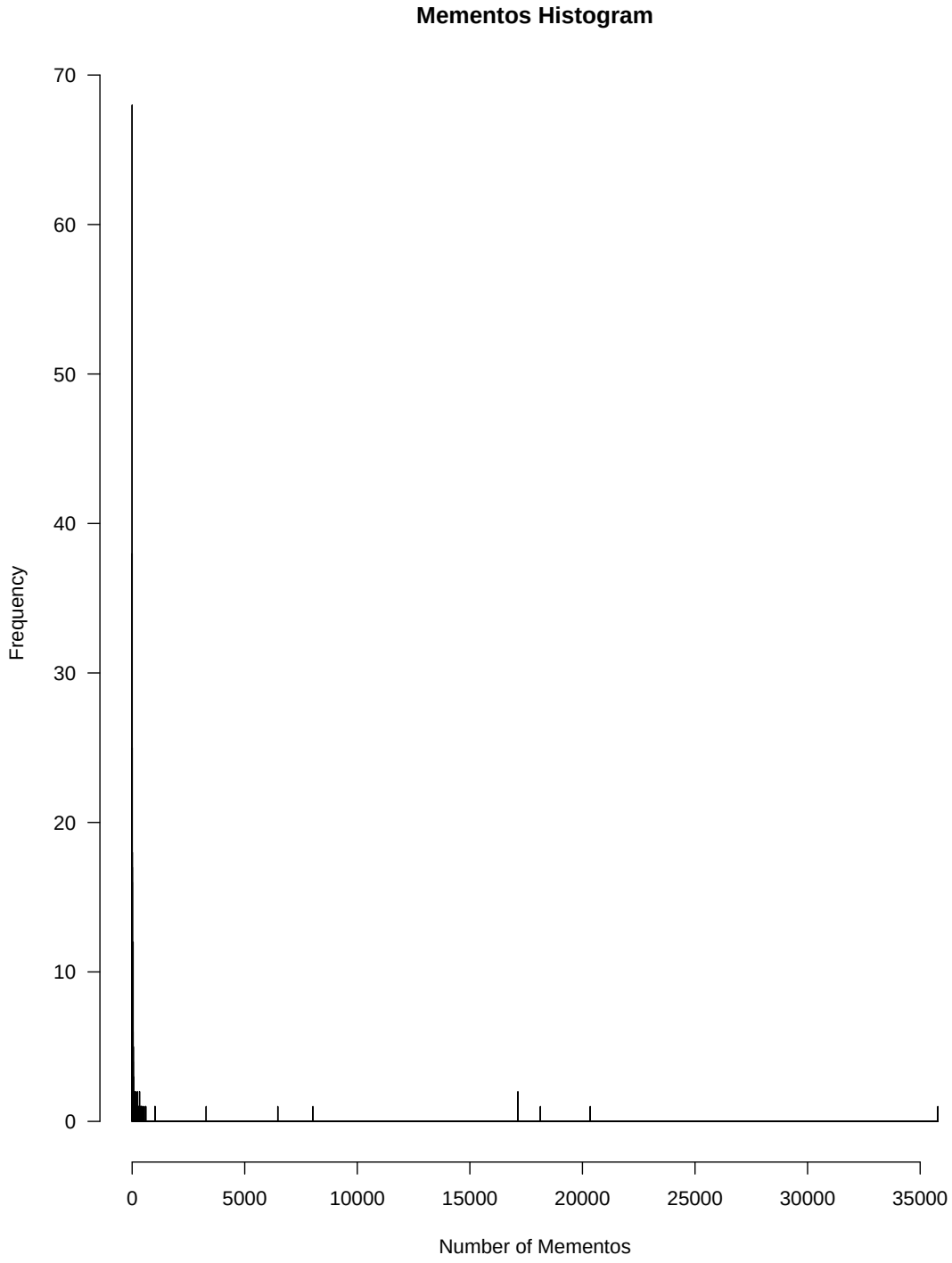
```
> timemaps_data <- read.table("C:/R/timemapreport.txt", header=F, sep="\t")
```

This line of code defines a variable max_num and assigns the maximum value in the vector timemaps_data to it.

```
> max_num <- max(timemaps_data)
```

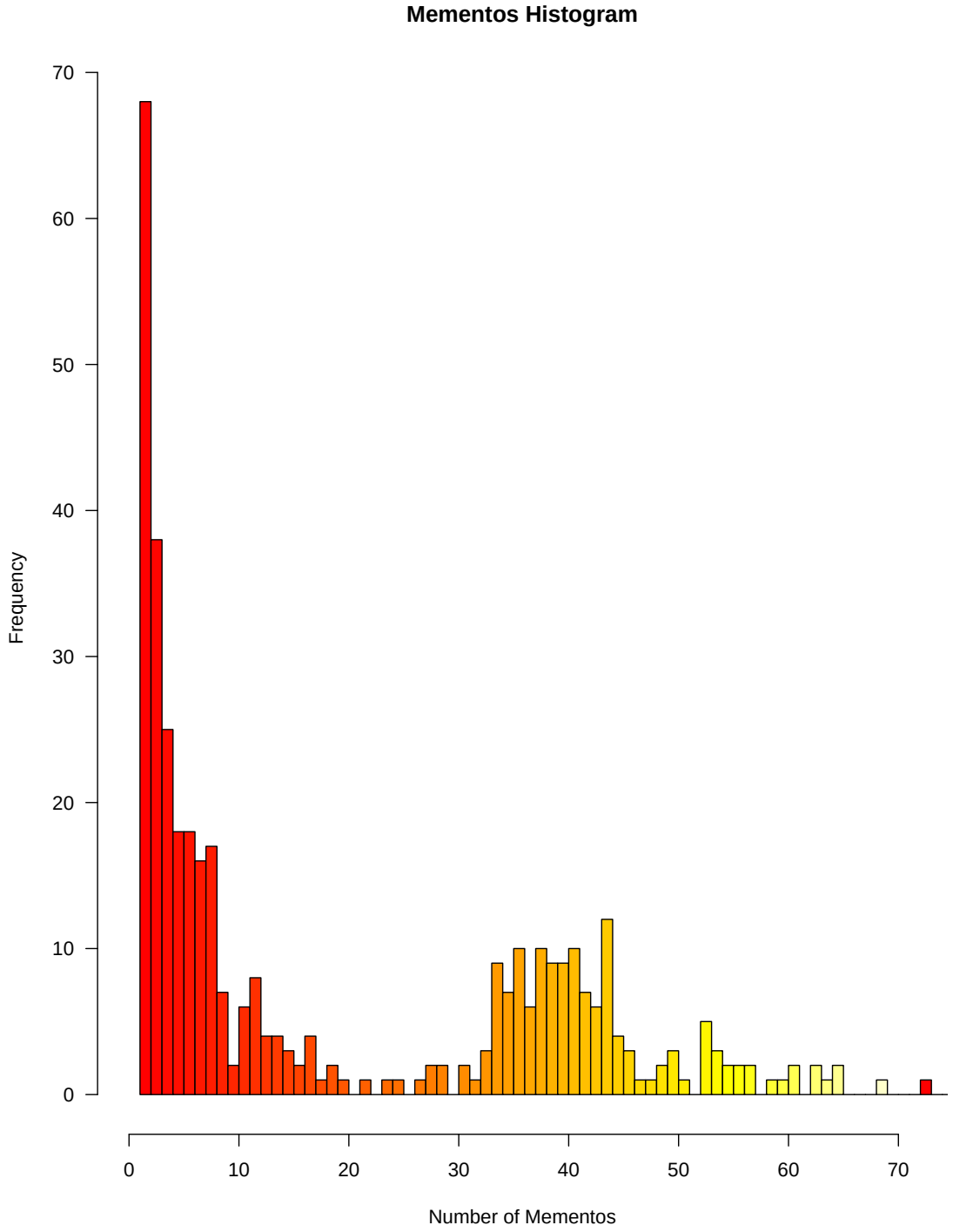
This line of code creates a histogram for timemaps_data with fire colors, sets bin = 1 so each number is in its own group, makes x axis range from 0 to max_num, disables right-closing of cell intervals, sets heading, changes the label of the x-axis, and makes y-axis labels horizontal

```
> hist(timemaps_data[,1], col=heat.colors(max_num), breaks=max_num, xlab="Number
    of Mementos", xlim=c(0,max_num), right=F, main="Mementos Histogram", las=1)
```


Mementos Histogram
Frequency
70
60
50
40
30
20
10
0
0
5000
10000
15000
20000
25000
30000
35000
Number of Mementos

We notice that our histogram did not provide a good representation of the results. I am going to zoom into the part where most of the data is and exclude the outliars in the data. Modifying the last line of code to this one will do just that.

```
> hist(timemaps_data[,1], col=heat.colors(max_num/500), breaks=max_num,
    xlab="Number of Mementos", xlim=c(0,max_num/500), right=F, main="Mementos
    Histogram", las=1)
```



The histogram shows that about 68 of the URIs have only one memento and about 38 URIs have two mementos, so on and so forth. It shows that almost half of the URIs, that actually have mementos, have less than 10 mementos.

4. Although this step is not necessary, but I wanted to count the occurrences of each value in "timemapreport.txt" and line every unique value with its number of occurrences in a table. From the file "timemapreport.txt", generate the data file "memesvslinks.txt" that contains our data, number of mementos vs number of links. I wrote a simple script, "generatelinksvsmemes.py", that loops over the lines in "timemapreport.txt" and generates "memesvslinks.txt" by counting the occurrences of each line in "timemapreport.txt" and saving the value in the line along with the number of its occurrences.

Listing 17: The content of generatelinksvsmemes.py

```
import sys
from collections import Counter

if len(sys.argv) != 3:
    print "Usage: python generatelinksvsmemes.py <input_file> <output_file>"
    print "e.g: python generatelinksvsmemes.py timemapreport.txt memesvslinks.txt
        "
else:
    fh_output = open(sys.argv[2] , 'w')
    number_of_memes = []
    fh_input = open(sys.argv[1], "r")

    for line in fh_input:
   line = line.strip('\n')
        number_of_memes.append(line)

    fh_output.write('memesCountX')
    fh_output.write('\t')
    fh_output.write('linksCountY')
    fh_output.write('\n')
    C = Counter(number_of_memes)
    for k,v in C.items():
        fh_output.write(str(k))
   fh_output.write('\t')
   fh_output.write(str(v))
   fh_output.write('\n')
    fh_input.close()
    fh_output.close()
```

Listing 18: Running generatelinksvsmemes.py

```
root@ima-app:/var/www/Hussam# python generatelinksvsmemes.py timemapreport.txt
    memesvslinks.txt
root@ima-app:/var/www/Hussam# cat memesvslinks.txt | wc -l
107
root@ima-app:/var/www/Hussam#
```

Included Files:

downloadtimemapsjson.py, uniquelinks.txt, uniquelinkswithmemes.txt, parsejsontimemap.py, timemapreport.txt, memesvslinks.txt, generatelinksvsmemes.py, bbcmemes.txt, bbcnomemes.txt, hist1.pdf, hist2.pdf
Folder: timemaps that has all timemap.json* files,

Question 3:

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 more than 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:

1. To download the estimated creation date for each link in "uniquelinks.txt", I wrote a python script, "downloadcreationdatejson.py", that does that using the "Carbon Date" tool. The script takes two command line arguments, an input file that has the links for which we want to download the estimated creation time "uniquelinks.txt", an output file to stores the age of each link (in days) based on the response "creationreport.txt".

Listing 19: The content of downloadcreationdatejson.py

```
import sys
import urllib2
import json
from datetime import *

current_datetime = datetime.now()

#d0 = date(2008, 8, 18)
#d1 = date(2008, 9, 26)
#delta = d0 - d1
#print delta.days

if len(sys.argv) != 3:
    print "Usage: Python downloadcreationdatejson.py <input_file> <output_file>"
    print "e.g: Python downloadcreationdatejson.py uniquelinks.txt creationreport
        .txt"
else:
    fh_input = open(sys.argv[1], 'r')
    fh_output = open(sys.argv[2], 'w')
    for line in fh_input:
   try:
```

```
        link = "http://cd.cs.odu.edu/cd?url=" + line
            response = urllib2.urlopen(link)
            data = json.load(response)
            creation_date = data['Estimated Creation Date']
        creation_date = str(creation_date)
        if creation_date:
               creation_date_clean = creation_date[0:10]
               year_C = creation_date_clean.split('-')[0]
               month_C = creation_date_clean.split('-')[1]
               day_C = creation_date_clean.split('-')[2]
               month_C = month_C.lstrip("0")
               day_C = day_C.lstrip("0")
          year_C = int(year_C)
          month_C = int(month_C)
          day_C = int(day_C)
               date1 = date(year_C, month_C, day_C)
               today = date(current_datetime.year, current_datetime.month,
                   current_datetime.day)
               old_in_days = (today - date1).days
          old_in_days = str(old_in_days)
               fh_output.write(old_in_days)
                fh_output.write("\n")
    except:
        print "This link generated an error:"
        print line
     fh_input.close()
     fh_output.close()
```

Listing 20: Running downloadcreationdatejson.py

```
root@ima-app:/var/www/Hussam# Python downloadcreationdatejson.py uniquelinks.txt
    creationreport.txt
```

2. From the first step in the second question, I found out which links has mementos and saved those links in a separate file and named it "uniquelinkswithmemes.txt". Now we need to create a program and pass this file as input to see which link has an estimated creation date so we can create a graph with age (in days) on the x-axis and number of mementos on the y-axis. I wrote a python script, "generatememesvsage.py" that will take "uniquelinkswithmemes.txt" as input and find the number of mementos as well as the age of each link in days and generate the data that necessary to create the graph in R and save it in "memesvsage.txt" as output.

Listing 21: The content of generatememesvsage.py

```
import sys
import urllib2
import json
from datetime import *

current_datetime = datetime.now()

if len(sys.argv) != 3:
    print "Usage: Python generatememesvsage.py <input_file> <output_file>"
    print "e.g: Python generatememesvsage.py uniquelinkswithmemes.txt memesvsage.
        txt"
```

```
else:
    fh_input = open(sys.argv[1], 'r')
    fh_output = open(sys.argv[2], 'w')
    fh_output.write('ageX')
    fh_output.write('\t')
    fh_output.write('memesY')
    for line in fh_input:
        try:
            cr_link = "http://cd.cs.odu.edu/cd?url=" + line
            me_link = "http://memgator.cs.odu.edu/timemap/json/" + line
            cr_response = urllib2.urlopen(cr_link)
            me_response = urllib2.urlopen(me_link)
            cr_data = json.load(cr_response)
            me_data = json.load(me_response)
            memes = len(me_data['mementos']['list'])
            memes = str(memes)
            creation_date = cr_data['Estimated Creation Date']
            creation_date = str(creation_date)
            fh_output.write('\n')
            if creation_date:
                creation_date_clean = creation_date[0:10]
                year_C = creation_date_clean.split('-')[0]
                month_C = creation_date_clean.split('-')[1]
                day_C = creation_date_clean.split('-')[2]
                month_C = month_C.lstrip("0")
                day_C = day_C.lstrip("0")
                year_C = int(year_C)
                month_C = int(month_C)
                day_C = int(day_C)
                date1 = date(year_C, month_C, day_C)
                today = date(current_datetime.year, current_datetime.month,
                    current_datetime.day)
                old_in_days = (today - date1).days
                old_in_days = str(old_in_days)
                fh_output.write(old_in_days)
            else:
                fh_output.write('0')
            fh_output.write('\t')
            fh_output.write(memes)
            fh_output.write("\n")
        except:
            print "This link generated an error:"
            print line
    fh_input.close()
    fh_output.close()
```

Listing 22: Running generatememesvsage.py

```
root@ima-app:/var/www/Hussam# python generatememesvsage.py uniquelinkswithmemes.
    txt memesvsage.txt
```

The data in "memesvsage.txt" is ready to be graphed using R.

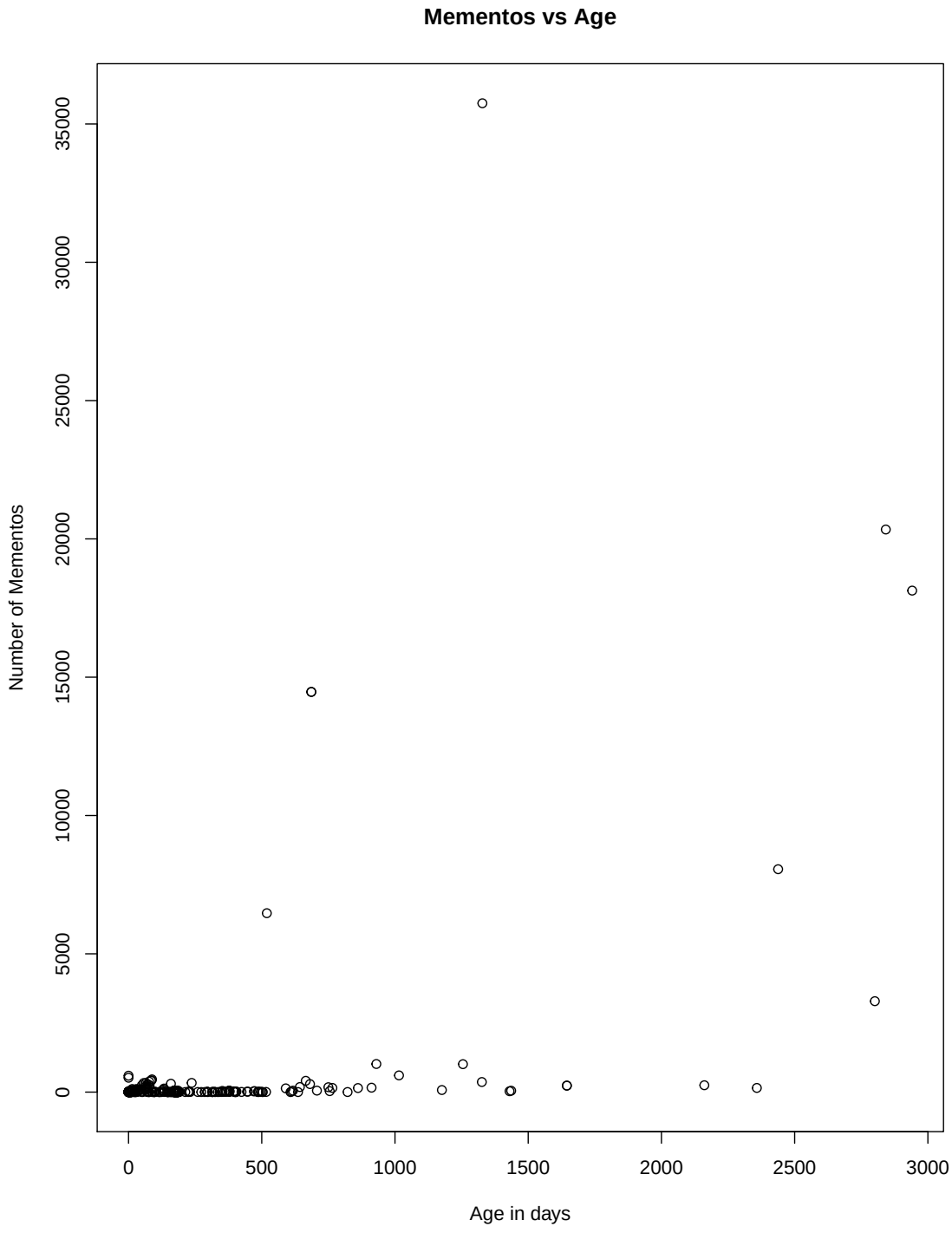
3. Now we are ready to generate the scatter plot for the number of mementos vs. the the age in days in R.

This line of code reads the data in the file and stores it in the vector mydf.

```
> mydf <- read.csv2('C:/R/memesvsage.txt', sep = '\t', header = T)
```

This line of code generates a basic scatter plot for mementos vs. age.

```
> plot(x = mydf$ageX, y = mydf$memesY, xlab="Age in days",ylab="Number of
    Mementos",main="Mementos vs Age")
```



Again, I want to zoom in where most of the data is and excluded the outliars, but first let me color code the points based on a simple criteria.

Let's make everything black first:

```
> mydf$Colour="black"
```

Points colored in red if the number of mementos is larger than the age in days.

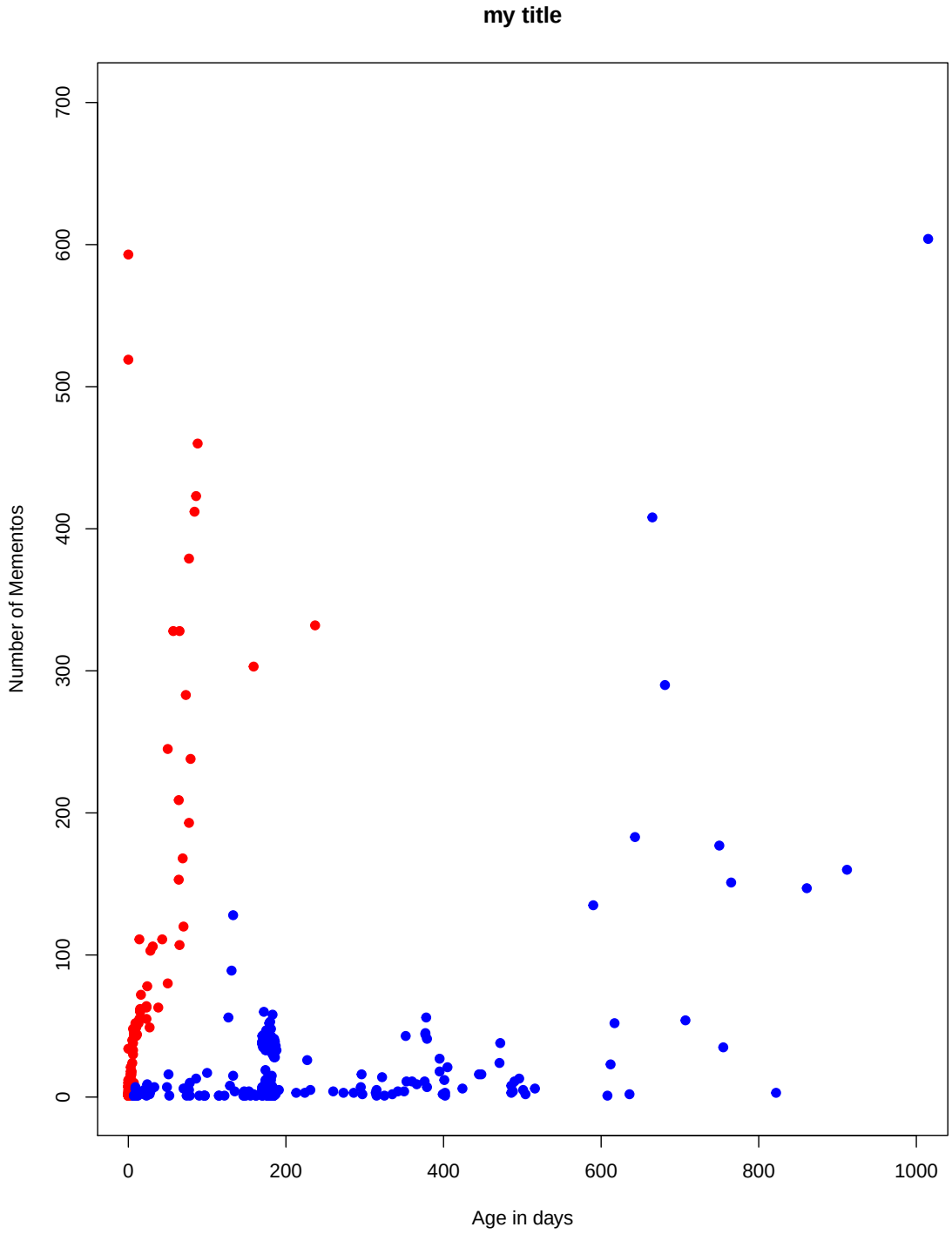
```
> mydf$Colour[mydf$memesY>mydf$ageX]="red"
```

Points colored in blue if the number of mementos is smaller than the age in days.

```
> mydf$Colour[mydf$memesY<mydf$ageX]="blue"
```

Now this line of code will generate the scatter plot we are looking for:

```
> plot(x = mydf$ageX, y =
    mydf$memesY,pch=19,xlim=range(1:1000),ylim=range(1:700),xlab="Age in
    days",ylab="Number of Mementos",main="my title", col=mydf$Colour)
```


my title
Number of Mementos
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
Age in days

Conclusion: Although the URIs I extracted from Twitter were randomly pulled from different tweets from accounts that belong to news channels, celebrities and so on, it is clear that the data can be represented by a single line using various mathematical algorithms like the least square, and this line is $y = x$. This means that as the age of the URI in days increases, the number of mementos increases, which makes perfect sense.

total URIs: 1107
no mementos: 662
no date estimate: 6

Included Files:

downloadcreationdatejson.py, generatememesvsage.py, memesvsage.txt, uniquelinks.txt, uniquelinksage.txt, uniquelinkswithmemes.txt, splot1.pdf, splot2.pdf

References

[1] Harding University: Producing Simple Graphs with R by Frank McCown
http://stackoverflow.com/questions/tagged/python

[2] Old Dominion University: Plotting in R by Martin Klein
http://www.cs.odu.edu/ mklein/cs796/lecture/plotting.html

[3] Stack Overflow: Python
http://stackoverflow.com/questions/tagged/python

[4] Stack Overflow: R
http://stackoverflow.com/questions/tagged/r