Assignment 2

CS532-s16: Web Sciences

Spring 2016

John Berlin

Generated on February 11, 2016

1

Question 1

```
Write a Python program that extracts 1000 unique links from Twitter.
http://thomassileo.com/blog/2013/01/25/using-twitter-rest-api-v1-dot-1-with-python/

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.).

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).

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

Answer

Extracting a 1000 unique links from Twitter at first seemed like a daunting task. The first steps I took to solving this problem was to look at the libraries twitter themselves suggested for use Twitter Libraries. I also did search for blog posts on twitter mining and a tutorial that was most helpful.

The tutorial Mining Twitter Data with Python used the library `Tweepy` which I was quick to modify for use in this assignment.

To get the uri's use the file `twitter_stream.py`

1. `twitter_stream.py` <-q><the query>
    - Takes a query to look for in the twitter stream
    - Connect to the twitter stream and listen
    - On Tweet that matches query get the json data
    - Extract the uri using the expanded url field
    - Append the uri to the file
    - Code in listing 1 starting on page 3:

```
#!/usr/bin/env python3
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
import time
import argparse
import string
import config
import json
import re

# this code was adapted from the example found at
# https://gist.github.com/bonzanini/af0463b927433c73784d


# this regex was used to extract un-expanded urls before thinking smart was an idea
reg = re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+',re.VERBOSE | re.IGNORECASE)


def get_parser():
    """Get parser for command line arguments."""
    parser = argparse.ArgumentParser(description="Twitter Downloader")
    parser.add_argument("-q",
                        "--query",
                        dest="query",
                        help="Query/Filter",
                        default='-')
    return parser

# Subclass StreamListener so that I can handle the data
# Normally the StreamListener simply consumes and does nothing
# You must subclass for anything to happen
class MyListener(StreamListener):
    """Custom StreamListener for streaming data."""

    def __init__(self, query):
        query_fname = format_filename(query)
        self.outfile = "urls%s.dat" %  query_fname
        useragent = 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:44.0) Gecko/20100101 Firefox/44.01'
        # self.outfile = "%s/stream_%s.json" % (data_dir, query_fname)

    # when there is a tweet simply append the extracted urls to a file
    def on_data(self, data):
        try:
            with open(self.outfile, 'a') as f:
                map = json.loads(data)
                #I found in the json data that there is an expanded_url portion
                #Which unshortens one level
                #Even then some shortened uris get shortened
                for it in map['entities']['urls']:
```

```
                        f.write("%s\n"%it['expanded_url'])

                return True
            except BaseException as e:
                #if bad data just let me know and move on
                print("Error on_data: %s" % str(e))
                time.sleep(5)
            return True

        def on_error(self, status):
            print(status)
            return True


def format_filename(fname):
    """Convert file name into a safe string.
    Arguments:
        fname -- the file name to convert
    Return:
        String -- converted file name
    """
    return ''.join(convert_valid(one_char) for one_char in fname)


def convert_valid(one_char):
    """Convert a character into '_' if invalid.
    Arguments:
        one_char -- the char to convert
    Return:
        Character -- converted char
    """
    valid_chars = "-_.%s%s" % (string.ascii_letters, string.digits)
    if one_char in valid_chars:
        return one_char
    else:
        return '_'


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Twitter
        Downloader")
    parser.add_argument("-q",
                        "--query",
                        dest="query",
                        help="Query/Filter",
                        default='-')
    args = parser.parse_args()
    #set up oauth
    auth = OAuthHandler(config.consumer_key, config.consumer_secret
        )
    auth.set_access_token(config.access_token, config.access_secret
        )
    # open a stream up and give it my listener
    twitter_stream = Stream(auth, MyListener(args.query))
    # filter the open stream for the query
    # this looks for recent tweets only
    twitter_stream.filter(track=[args.query])
```

Listing 1: Python program to mine urls for a twitter query

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://mementoproxy.cs.odu.edu/aggr/timemap/link/1/http://www.cs.odu.edu/

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.

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

Answer

To answer this question and the last one I combined the steps to generate the required data into as few files as possible. The following steps give a break down to use the files with detail into what is going on is provided as comments in the code

1. run `local.py` from the carbon date library
    - use method `uniques` and `zeroNoneZero` from `util.py` to ensure the selected uris are unique, also to generate the nonzero uri file
    - copy the file nonzero.txt to the directory containing the Carbon Dating library with modified local.py
    - The file used to provide the timemap support was originally written by Scott Ainsworth which I modified to be compatible with python3. It is found in listing 3 starting on page 8. The number of mementos is the length of the keys to memento map in TimeMap class contained in this file
    - run and copy dumped json file called dated.json back to location where `carbonDating.py` is
    - Code in listing 5 starting on page 12
2. `carbonDating.py`
    - Count the number of Mementos per uri if there is an exception there are no mementos for the uri so associate a count of 0 with that uri
    - Parse the json file dated.json with the uri's in contained in the nonezero file produced by running the `zeroNoneZero` from `util.py`
    - the age of the memento is the python Date delta from today to age gotten from running the modified `local.py` in listing 5 starting on page 12 from the CarbonDating library

- Code in listing 6 starting on page 15

The R script to generate the figure 1 shows the histogram of the mementos counts is found is shown below.

```
# set the working directory
setwd(getwd())
# get dataframe site,count
mementoCount <- read.csv('counted.csv')
# make histogram
h <- hist(mementoCount$count,main="Histogram of Memento Counts",
    xlab = "Count")
# add count labels
text(h$mids,h$counts,labels=h$counts, adj=c(0.5, -0.5))
```

Listing 2: Memento Count HistoGram R script

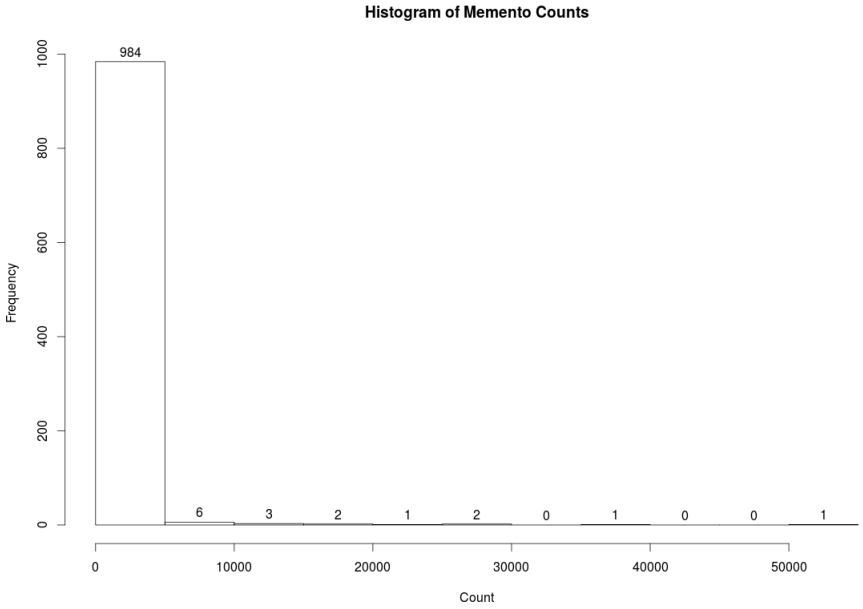

Figure 1: histogramCounts

As seen in the Histogram there were more uris with less than 1000 mementos

```
#!/usr/bin/python -B

import re
import urllib.request

import dateutil.parser
import requests

tokenizer = re.compile('(<[^>]+>|[a-zA-Z]+="[^"]*"|[;,])\\s*')


class TimeMap:
    def __init__(self, timemap_uri):
        self.original = None
        self.timebundle = None
        self.timegate = None
        self.timemap = None
        self.first_memento = None
        self.last_memento = None
        self.mementos = {}
        self.__tokens = TimeMapTokenizer("http://mementoproxy.cs.
            odu.edu/aggr/timemap/link/1/%s" % timemap_uri)
        link = self.get_next_link()
        while link is not None:
            if link[0] == 'memento':
                self.mementos[link[1]] = link[2]
            elif link[0] == 'original':
                self.original = link[2] if link is not None else
                    None
            elif link[0] == 'timebundle':
                self.timebundle = link[2] if link is not None else
                    None
            elif link[0] == 'timegate':
                self.timegate = link[2] if link is not None else
                    None
            elif link[0] == 'timemap':
                self.timemap = link[2] if link is not None else
                    None
            elif link[0] == 'first memento':
                self.mementos[link[1]] = link[2]
                self.first_memento = link[1] if link is not None
                    else None
            elif link[0] == 'last memento':
                self.mementos[link[1]] = link[2]
                self.last_memento = link[1] if link is not None
                    else None
            link = self.get_next_link()

    def get_next_link(self):
        uri = None
        datetime = None
        rel = None
        resource_type = None
        for token in self.__tokens:
            if token[0] == '<':
                uri = token[1:-1]
            elif token[:9] == 'datetime=':
```

```
                datetime = token[10:-1]
            elif token[:4] == 'rel=':
                rel = token[5:-1]
            elif token[:5] == 'type=':
                resource_type = token[6:-1]
            elif token == ';':
                None
            elif token == ',':
                return (rel, dateutil.parser.parse(datetime)
                if datetime is not None else None,
                        uri, resource_type)
            else:
                raise Exception('Unexpected timemap token', token)
        if uri is None:
            return None
        else:
            return (rel, dateutil.parser.parse(datetime)
            if datetime is not None else None,
                    uri, resource_type)

    def __getitem__(self, key):
        return self.mementos[key]


class TimeMapTokenizer:
    def __init__(self, timemap_uri):
        self.http = urllib.request.urlopen(timemap_uri)
        # self._tmfile = requests.get(timemap_uri).iter_lines()
        self.r = requests.get(timemap_uri)
        self._tmfile = self.r.iter_lines()
        # print(self.r.text)
        self._tokens = []
        self.lines = []
        self.size = 0
        self.cur = 0
        self.doIt()

    def doIt(self):
        for line in self._tmfile:
            self.lines.append(line.decode("utf-8"))
        self.size = len(self.lines)

    def __next__(self):
        if len(self._tokens) == 0:
            if self.cur == self.size:
                raise StopIteration
            line = self.lines[self.cur]
            self.cur += 1
            if self.cur == self.size:
                raise StopIteration
            self._tokens = tokenizer.findall(line)
        return self._tokens.pop(0)

    def __iter__(self):
        return self
```

Listing 3: TimeMaps

```python
import requests
import csv

# get set of unique urls
def unique():
    s = set()
    with open("urls.dat","r") as o:
        for line in o:
            url = line.rstrip("\n")
            s.add(url)
    with open("urls.dat","w+") as write:
        for url in sorted(s):
            write.write("%s\n"%url)

# get the real urls for shortened ones
def uniqueShortened():
    useragent = 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:44.0)
        Gecko/20100101 Firefox/44.01'
    session = requests.Session()
    session.headers.update({'User-Agent': useragent})
    s = set()
    c = 0

    with open("shortenedURLS.txt", "r") as o:
        for line in o:
            url = line.rstrip("\n")
            try:
                r = session.get(url)
                """:type : requests.Response """
                # if status code is 200 then we have a good link
                if r.status_code == 200:
                    # add it our set of urls
                    s.add(r.url)
                    c += 1
                    print(c)
            except:
                continue

    print(len(s))
    print("
        +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ")

    session.close()
    with open("urls2.dat", "w+") as write:
        for url in sorted(s):
            print(url)
            write.write("%s\n" % url)


# method to write out the zero and non-zero memento count urls
def zeroNoneZero():
    # count()
    zero = []
    nonZero = []

    with open("counted.csv", "r") as csvFile:
```

```
        reader = csv.DictReader(csvFile)
        for row in reader:
            print(row)
            print(row["site"], row["count"])
            if int(row["count"]) > 0:
                nonZero.append(row["site"])
            else:
                zero.append(row["site"])

    with open("nonZero.txt", "w+") as onz:
        for site in nonZero:
            onz.write("%s\n" % site)

    with open("zero.txt", "w+") as zo:
        for site in zero:
            zo.write("%s\n" % site)
```

Listing 4: util

```
from checkForModules import checkForModules
import json
from ordereddict import OrderedDict
#import simplejson
import urlparse
import re

from getBitly import getBitlyCreationDate
from getArchives import getArchivesCreationDate
from getGoogle import getGoogleCreationDate
from getBacklinks import *
from getLowest import getLowest

from getLastModified import getLastModifiedDate
#Topsy service is no longer available
#from getTopsyScrapper import getTopsyCreationDate
from htmlMessages import *
from pprint import pprint

from threading import Thread
import Queue
import datetime

import os,sys, traceback




def cd(url, backlinksFlag = False):

    #print 'Getting Creation dates for: ' + url
    #scheme missing?
    parsedUrl = urlparse.urlparse(url)
    if( len(parsedUrl.scheme)<1 ):
        url = 'http://'+url
    threads = []
    outputArray =['','','','','','']
    now0 = datetime.datetime.now()


    lastmodifiedThread = Thread(target=getLastModifiedDate, args=(
        url, outputArray, 0))
    bitlyThread = Thread(target=getBitlyCreationDate, args=(url,
        outputArray, 1))
    googleThread = Thread(target=getGoogleCreationDate, args=(url,
        outputArray, 2))
    archivesThread = Thread(target=getArchivesCreationDate, args=(
        url, outputArray, 3))

    if( backlinksFlag ):
        backlinkThread = Thread(target=
            getBacklinksFirstAppearanceDates, args=(url,
            outputArray, 4))

    #topsyThread = Thread(target=getTopsyCreationDate, args=(url,
        outputArray, 5))

```

```
51
52      # Add threads to thread list
53      threads.append(lastmodifiedThread)
54      threads.append(bitlyThread)
55      threads.append(googleThread)
56      threads.append(archivesThread)
57
58      if( backlinksFlag ):
59          threads.append(backlinkThread)
60
61      #threads.append(topsyThread)
62
63
64      # Start new Threads
65      lastmodifiedThread.start()
66      bitlyThread.start()
67      googleThread.start()
68      archivesThread.start()
69
70      if( backlinksFlag ):
71          backlinkThread.start()
72
73      #topsyThread.start()
74
75
76      # Wait for all threads to complete
77      for t in threads:
78          t.join()
79
80      # For threads
81      lastmodified = outputArray[0]
82      bitly = outputArray[1]
83      google = outputArray[2]
84      archives = outputArray[3]
85
86      if( backlinksFlag ):
87          backlink = outputArray[4]
88      else:
89          backlink = ''
90
91      #topsy = outputArray[5]
92
93      #note that archives["Earliest"] = archives[0][1]
94      try:
95          #lowest = getLowest([lastmodified, bitly, google, archives
                [0][1], backlink, topsy]) #for thread
96          lowest = getLowest([lastmodified, bitly, google, archives
                [0][1], backlink]) #for thread
97      except:
98         print sys.exc_type, sys.exc_value , sys.exc_traceback
99
100
101
102     result = []
103
104     result.append(("URI", url))
105     result.append(("Estimated Creation Date", lowest))
```

```
106     values = OrderedDict(result)
107     r = json.dumps(values, sort_keys=False, indent=2, separators=('
            ,', ': '))
108
109     now1 = datetime.datetime.now() - now0
110
111     return r
112
113
114
115
116 if len(sys.argv) == 1:
117     print "Usage: ", sys.argv[0] + " url backlinksOnOffFlag ( e.g:
            " + sys.argv[0] + " http://www.cs.odu.edu  [--compute-
            backlinks] )"
118 elif len(sys.argv) == 2:
119
120     time.strptime("1995-01-01T12:00:00", '%Y-%m-%dT%H:%M:%S')
121     file = open("nonZero.txt","r")
122     r = []
123     for url in file:
124         print url.rstrip("\n")
125         r.append(cd(url.rstrip("\n")))
126     file.close()
127     f = open("dated.json","w+");
128     out = ",".join(r)
129     f.write(out)
130     print out
131     f.close()
132 elif len(sys.argv) == 3:
133     time.strptime("1995-01-01T12:00:00", '%Y-%m-%dT%H:%M:%S')
134     file = open("nonZero2.txt","r")
135
136     for url in file:
137         print url
138     file.close()
139     if(sys.argv[2] == '--compute-backlinks'):
140         cd(sys.argv[1], True)
141     else:
142         cd(sys.argv[1])
```

Listing 5: Modified local.py

```
import csv
import json
from datetime import date

from dateutil.parser import parse

import timemaps


# method to count the number of timemaps for a uri
def count():
    counted = {}
    # for each url in the 1000 urls
    with open("urls.dat", "r") as o:
        for line in o:
            # remove the newline character
            url = line.rstrip("\n")
            try:
                # get the time map
                tm = timemaps.TimeMap(url)
                # the number of mementos is the number of keys
                counted[url] = len(tm.mementos.keys())
                print("%s, %i\n" % (url, len(tm.mementos.keys())))
            except:
                # if an exception is raised it means no mementos
                counted[url] = 0
    # write the counts to file
    with open("counted.csv", "w+") as oo:
        oo.write("site,count\n")
        for url, count in counted.items():
            print("%s, %i\n" % (url, count))
            oo.write("%s, %i\n" % (url, count))

# method to count and carbon date the uri mementos
def countAndDate():
    count()
    zero = []
    nonZero = []
    all = {}
    # get todays date
    now = date.today()

    with open("counted.csv", "r") as csvFile:
        # read the count file
        reader = csv.DictReader(csvFile)
        for row in reader:
            print(row)
            print(row["site"], row["count"])
            # get two collections the nonZero and the zero counts
            if int(row["count"]) > 0:
                nonZero.append(row["site"])
            else:
                zero.append(row["site"])
            all[row["site"]] = row["count"]

    # open the json data from carbondate
    jdata = open("dated.json", "r")
```

```
    data = json.load(jdata)
    jdata.close()
    cds = {}

    for jd in data:
        # create mapping for the uri to date
        cds[jd['URI']] = parse(jd['Estimated Creation Date'])
        print(jd, type(parse(jd['Estimated Creation Date'])))

    with open("dated.csv", "w+") as out:
        out.write("age,mementos\n")
        # for all nonZero memento write the estimated age in days
        for it in nonZero:
            # python dates have substraction for dates
            # to mean the days between two daes
            age = now - date(cds[it].year, cds[it].month, cds[it].
                day)
            """ :type datetime.timedelta """
            out.write("%s,%s\n" % (age.days, all[it]))
            print(it, age.days, all[it])




if __name__ == "__main__":
    countAndDate()
```

Listing 6: Python program to mine urls for a twitter query

Question 3

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

http://ws-dl.blogspot.com/2014/11/2014-11-14-carbon-dating-web-version-20.html

Note: you'll should download the library and run it locally; don't try to use the web service.

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

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

Answer

Out of the 1000 uri mementos 28 of them were found to have an age of 0 but none of the uri were found to have a memento count of 0.

Using the generated files from the python files I used R code found in listing 7 starting on page 17 to generate figure 2.

In the r script I found the median memento count and partitioned the data frame into two halves those that fell below the median and those the fall about it. This was to have two graphs that clearly show the entire graph.

The number of mementos that were above the median are fare more dispersed than those below it. As seen in the all age count graph it is difficult to see the distribution due to outliers.

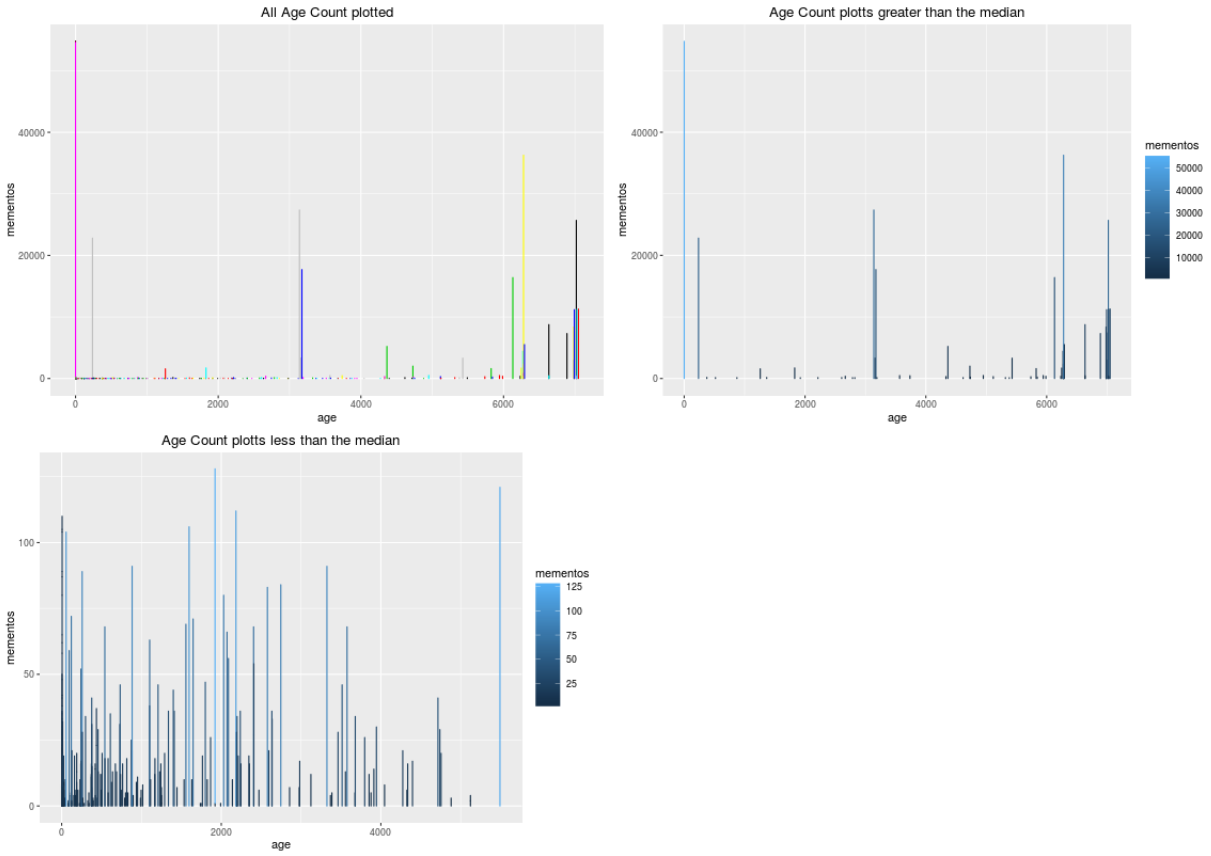

Figure 2: histogramCounts

```
1  library(ggplot2)
2  setwd(getwd())
3
4  #this function wonderfully borrowed from
5  #http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_%28
       ggplot2%29/
6  multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL
       ) {
```

```
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)
  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

  if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(
        layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain
          this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row
          ,
                                      layout.pos.col = matchidx$col
                                          ))
    }
  }
}

# get the age,mementos dataframe
dateNum <- read.csv('dated.csv')

# find the median value to split the plots
mcount <- sort(unique(dateNum$mementos))
midval <- median(mcount)

# find all values less than the midval
lt <- subset(dateNum, dateNum$mementos <= midval)
# find all values greater than the midval
gt <- subset(dateNum, dateNum$mementos >= midval)

# plot the entire dataframe
# use stat_identity to plot the actual data not a count
a<-ggplot(dateNum,   aes(y=mementos,x=age,color=mementos))+
  geom_bar(stat="identity")+scale_color_identity()+
  ggtitle("All Age Count plotted")

# plot the lower values, use color values based on memento count
```

```
b<-ggplot(lt,   aes(y=mementos,x=age,color=mementos))+
  geom_bar(stat="identity",position = "stack",aes(color=mementos))+
  ggtitle(as.character("Age Count plotts less than the median"))

# plot the upper values, use color values based on memento count
c<-ggplot(gt,   aes(y=mementos,x=age))+
  geom_bar(stat="identity",position = "stack",aes(color=mementos))+
  scale_fill_identity(breaks = lt$mementos,guide = "legend")+
  ggtitle(as.character("Age Count plotts greater than the median"))

#combine the three plots
multiplot(a,b,c,cols=2)
```

Listing 7: Memento Count Age R script