

CS896 Introduction to Web Science
Fall 2013
Report for Assignment 2

Corren G. McCoy

September 26, 2013

Contents

1	Question 1	4
1.1	Problem	4
1.2	Response	4
2	Question 2	4
2.1	Problem	4
2.2	Response	4
3	Question 3	5
3.1	Problem	5
3.2	Response	6
Appendices		8
A Python Source for extractLinks.py		9
B Python Source for getTimeMaps.py		12
C R Source for Histogram		15
D Python Source for estimateAge.py		16
E R Source for Scatterplot		18

List of Figures

1	URIs vs. Mementos	5
2	Carbon Date Sample Output	6
3	Age vs. Number of Mementos	7

List of Tables

1	Sample Twitter URLs	4
2	Sample TimeMap Output	5
3	Format of uri_age.txt	7

1 Question 1

1.1 Problem

Write a Python program that extracts 1000 unique links from Twitter.

1.2 Response

We used Twitter’s Search API¹ to search for recent tweets which contained any of the following keywords: “Putin”, “Syria”, “Assad”, “Obama,” or “chemical weapons”. Our Python program, *ExtractLinks*, as shown in Appendix A, stores the keywords in a list object with each element passed individually to the API as part of the query string. The text of each tweet returned from the search was parsed to remove URLs, if any, which were then written to a text file (i.e., *tweetFileCandidates.txt*). Using the keywords indicated, we built a collection of approximately 8400 candidate URLs from which we sought to obtain the desired 1000 unique links. The unique links were identified using line-by-line processing of the *tweetFileCandidates* list. Each shortened URL was opened using an HTTP GET operation in order to verify the reference to an existing web site. Any URL with an HTTP response status code of 200 (i.e., success) was checked against our program’s master domain list. The full URL was obtained from the response header. To ensure uniqueness only one URL from any top-level domain was allowed. Subsequent references to an existing domain were ignored. This constraint was facilitated by storing the domain as the key and the full URL as the value in a Python dictionary object which inherently ensures a unique domain for each key. Processing of the URLs in the *tweetFileCandidates* was terminated once 1000 domain URLs were obtained. This final list was then printed to a text (i.e., *tweetFile1000.txt*). A sample of the URLs harvested from Twitter is shown in Table 1.

http://investmentwatchblog.com/
http://whithererehwon.wordpress.com/
http://maxcouti.blogspot.com/
http://kevskewl.wordpress.com/
http://www.marketing-projects.biz/
http://mundonovovelho.blogspot.com.br/

Table 1: Sample Twitter URLs

2 Question 2

2.1 Problem

Download the TimeMaps for each of the target URIs. Create a histogram of URIs versus number of Mementos (as computed from the TimeMaps).

2.2 Response

Using our list of 1000 Twitter URIs, we used the ODU Memento aggregator to generate the associated TimeMaps via our Python program, *getTimeMaps*, which is described in Appendix B. A sample of the TimeMap output is shown in Table 2. The Rel-Tag² of each TimeMap was examined to determine whether the information presented was tagged as a memento. We specifically looked for valid combinations of either “first memento”, “last memento”, “memento first”, “memento last”, or “first last memento” using regular expressions in Python. Each occurrence of a memento tag increased the counter for each URI. The URI and the associated memento count were saved to a comma-delimited

text file (i.e., histogram.txt) that was used as the data source for our histogram. We used the graphing functions in R to produce the histogram shown in Figure 1. The labels on each bar represent the number of URIs in a particular bin. The histogram is representative of a long tail³ distribution with over 96% of the URIs having less than 2,500 mementos. The distribution is much sparser in the range of 2,500 to approximately 30,000 mementos. We noted that URIs in the higher range, such as <http://www.yahoo.com> with 23,388 mementos, tend to change quite frequently. The dynamic nature of this type of site would in turn require more frequent archiving to ensure continuity. The URIs in the lower range, such as <http://www.gary-weiss.com/> with 111 mementos, tend to change less frequently since many appear to represent personal websites or blogs with static content.

```
<http://http://mementoproxy.cs.odu.edu/aggr/timemap/link/http://
investmentwatchblog.com/>;rel="self";type="application/link-format",
<http://mementoproxy.cs.odu.edu/aggr/timegate/http://investmentwatchblog.com/>;
rel="timegate",<http://investmentwatchblog.com/>;rel="original",
<http://web.archive.org/web/20100212220649/http://investmentwatchblog.com/>;rel=
"firstmemento";datetime="Fri,12Feb201022:06:49GMT",
<http://web.archive.org/web/20100409065504/http://investmentwatchblog.com/>;rel=
"memento";datetime="Fri,09Apr201006:55:04GMT",
<http://web.archive.org/web/20100415152016/http://investmentwatchblog.com/?>;rel=
"memento";datetime="Thu,15Apr201015:20:16GMT",
```

Table 2: Sample TimeMap Output

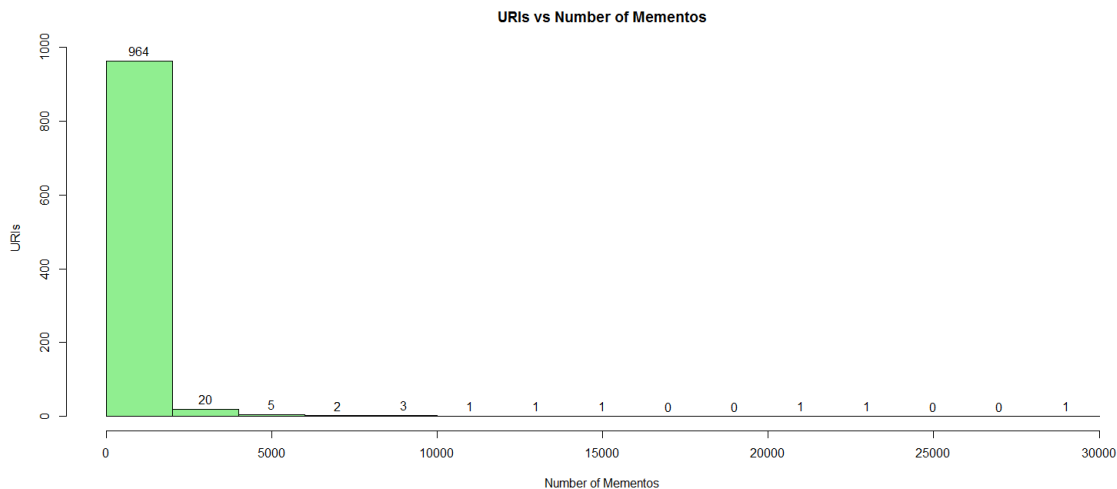


Figure 1: URIs vs. Mementos

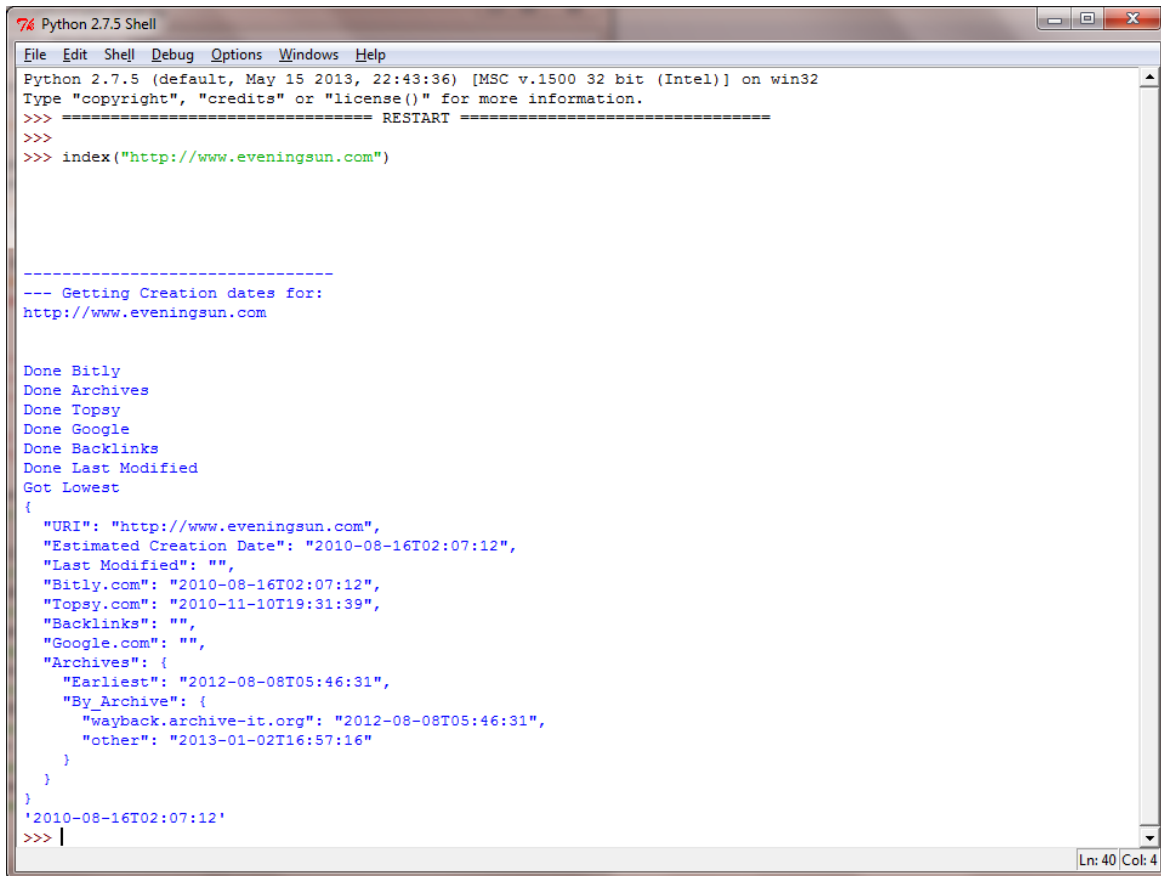
3 Question 3

3.1 Problem

Estimate the age of each of the 1000 URIs using the “Carbon Date” tool. For URIs that have at least one memento and an estimated creation date, create a graph with age (in days) on one axis and number of mementos on the other.

3.2 Response

We used the provided Carbon Date⁴ tool as the basis for our analysis. SalahEldeen and Nelson [1] describe the tool as “a simple web application that estimates the creation date for a URI by polling a number of sources of evidence and returning a machine-readable structure with their respective values”. These information sources include bitly, topsy, Google, the ODU Memento aggregator and the HTTP response header. The tool, originally designed as a web service, was modified to accept file-based input (i.e., histogram.txt). Our Python program, *EstimateAge*, described in Appendix D, performed line-by-line processing of the input file to identify the URLs with mementos. Once identified, the URL was passed to the main processes of Carbon Date to retrieve the estimated creation date. Sample output is shown in Figure 2. If an estimated creation date was returned, the date was recorded in a text file (i.e., uri_age.txt) along with the URI, number of mementos and calculated age in days. The format of the file is shown in Table 3. We used the graphing functions in R to create a scatter plot as shown in Figure 3 which presents the relationship between the age of the URI and the number of mementos. The positive slope of the blue regression line indicates a strong relationship between the age of the URI and the number of expected mementos. Basically, older URIs would have accumulated more mementos over time.



```
Python 2.7.5 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>> index("http://www.eveningsun.com")

-----
--- Getting Creation dates for:
http://www.eveningsun.com

Done Bitly
Done Archives
Done Topsy
Done Google
Done Backlinks
Done Last Modified
Got Lowest
{
  "URI": "http://www.eveningsun.com",
  "Estimated Creation Date": "2010-08-16T02:07:12",
  "Last Modified": "",
  "Bitly.com": "2010-08-16T02:07:12",
  "Topsy.com": "2010-11-10T19:31:39",
  "Backlinks": "",
  "Google.com": "",
  "Archives": {
    "Earliest": "2012-08-08T05:46:31",
    "By Archive": {
      "wayback.archive-it.org": "2012-08-08T05:46:31",
      "other": "2013-01-02T16:57:16"
    }
  }
}
'2010-08-16T02:07:12'
>>> |
```

Figure 2: Carbon Date Sample Output

¹<https://dev.twitter.com/docs/api/1.1>

²<http://www.microformats.org/wiki/Rel-Tag>

³http://en.wikipedia.org/wiki/The_Long_Tail

⁴<http://ws-dl.blogspot.com/2013/04/2013-04-19-carbon-dating-web.html>

URI	Mementos	Creation Date	Age
http://investmentwatchblog.com/	287	2010-02-12T22:06:49	1316
http://whithererehwon.wordpress.com/	2	2012-09-22T23:02:43	363
http://maxcouti.blogspot.com/	23	2008-05-18T15:07:07	1952

Table 3: Format of uri_age.txt

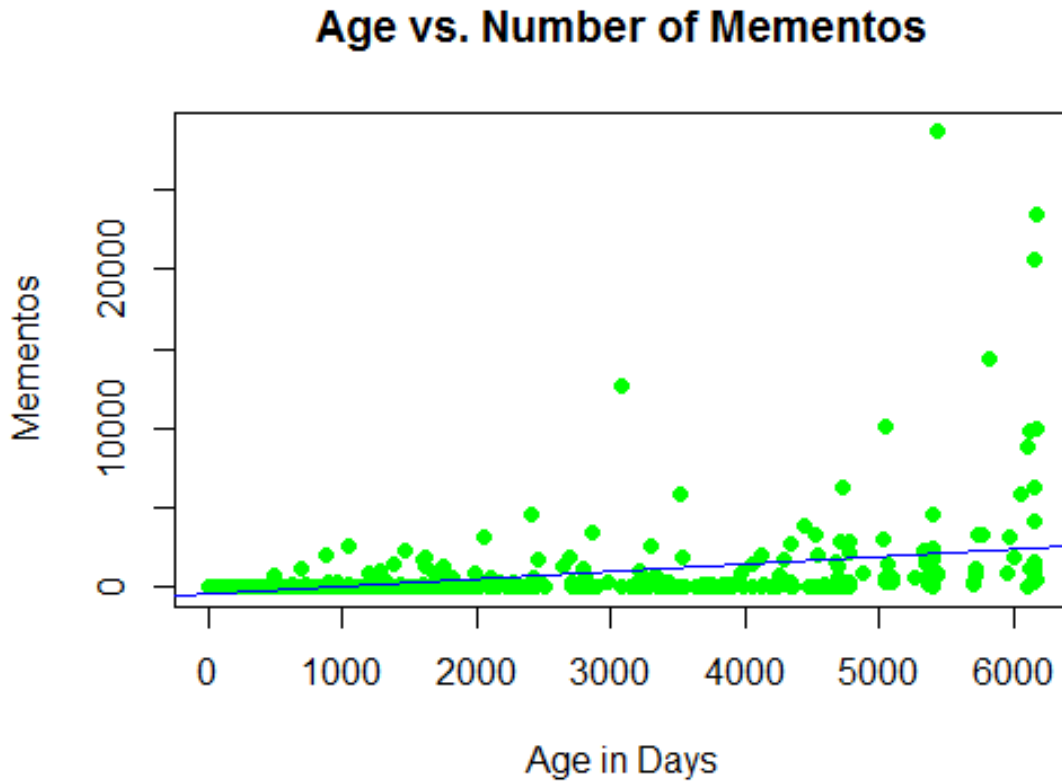


Figure 3: Age vs. Number of Mementos

Bibliography

- [1] H. M. SalahEldeen and M. L. Nelson. Carbon dating the web: estimating the age of web resources. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 1075–1082. International World Wide Web Conferences Steering Committee, 2013.

Appendix A

Python Source for extractLinks.py

```
# TwitterSearch package obtained from https://github.com/ckoepp/TwitterSearch

from TwitterSearch import *
import codecs
import urllib
import urllib2
from urlparse import urlparse

# This function iterates over the twitter entity object to
# find a particular element
def posted_url(dictionary):
    # Recursively iterates over entities to find expanded URLs
    for key, value in dictionary.iteritems():
        if isinstance(value, dict):
            posted_url(value)
        else:
            return(value)

# Save the final list of 1000 unique links
def saveToFile(pLinks):
    linkFile = codecs.open('C:/Python27/myFiles/linkFile.txt','w','utf-8')
    for key in pLinks:
        linkFile.write(key+'\n');

# This function searches Twitter for a particular keyword
def searchTwitter(pThisTerm):
    try:
        # create a TwitterSearchOrder object
        tso = TwitterSearchOrder()
        # Define all the keywords which must be passed URL encoded.
        # Only return/filter tweets that contain links
        tso.setKeywords(["filter%3Alinks",pThisTerm])
        # we want to see English tweets only'
        tso.setLanguage('en')
        # maximum number of tweets to return
```

```

tso.setCount(100)
# include the entity information
tso.setIncludeEntities(True)
#tso.setResultType('recent')

# create a TwitterSearch object with my secret tokens (@CorrenMcCoy)
ts = TwitterSearch(
    consumer_key = 'LrA1DdH1QJ5cfS8gGaWp0A',
    consumer_secret = '9AX14EQBLjRjJM4ZHt2kNf0I4G77sKsYX1bEXQCW8',
    access_token = '1862092890-FrKbhd7ngeJtTZfZwf2SMjOPwgsCToq2A451iWi',
    access_token_secret = 'AdMQmyfaxollI596G82FBipfSMhagv6hjlNKoLYjeg8'
)

# Iterate over the tweet entities which are in a nested dictionary
tweetFile = codecs.open('C:/Python27/myFiles/tweetFile.txt','a','utf-8')
for tweet in ts.searchTweetsIterable(tso):
    #print( '%s tweeted: %s' % ( tweet['user']['screen_name'], tweet['text'] ) )
    if tweet['user']['url'] is not None:
        print (tweet['user']['url'])
        tweetFile.write(tweet['user']['url']+'\n');

tweetFile.close()

# Error handling. Close file and terminate
except TwitterSearchException as e:
    tweetFile.close()
    print(e)
    exit()

def findUniqueLinks():
    tweetFile = open('C:/Python27/myFiles/tweetFile.txt')
    # initialize the list to hold all the fully expanded URIs
    # that respond with 200
    tweetURLs=[]
    # create an empty dictionary
    domains={}
    for line in iter(tweetFile):
        uri = line.rstrip('\n')
        #package the request
        request=urllib2.Request(uri)
        request.add_header('User-agent','Mozilla 5.10')
        try:
            response=urllib2.urlopen(request)
            if response.code == 200:
                tweetURLs.append(response.url)
                # make sure there's only one URI with the same domain.
                # http://stackoverflow.com/questions/10362453/unique-by-domains-urls-list
                # Return the last URL for each domain the URL list. By definition, the key
                # in a dictionary is unique.

```

```

domains=dict((urlparse(u).netloc, u) for u in tweetURLs).values()

# Stop traversing the file when we reach the 1000th domain
if len(domains)== 1000:
    # Write out the URLS so we can use them for other tasks
    saveToFile(domains)
    return
print response.code,response.url, "Domains=", len(domains)
response.close()
except urllib2.HTTPError,e:
    print "Error", line, e
    continue
except urllib2.URLError, e:
    print "Error", line, e
    continue
except IOError, e:
    print "Error", line, e
    continue
# how many links do we have?
print "Unique links = ", len(domains)
tweetFile.close()

```

```

#####
# This is main procedure in this package
#####
def extractLinks():
    # Use these keywords to build a list of Tweets from which we will extract
    # the desired 1000 unique links.
    keywordList={"Putin","Syria","Assad","Obama","chemical%20weapons"}
    for thisTerm in keywordList:
        print "Tweets for keyword>>>",thisTerm
        searchTwitter(thisTerm)
    print "Twitter search complete>>>>>>"

    # Read the file of links.
    findUniqueLinks()
    print "1000 links saved to file>>>>>>"

```

Appendix B

Python Source for getTimeMaps.py

```
#!/usr/bin/python -B
import urllib2
import urllib
import codecs
import re

# Save the final count to a comma delimited file
def saveToFile(pUri, pCount):
    # append the data
    histogram = codecs.open('C:/Python27/myFiles/histogram.txt','a','utf-8')
    histogram.write(pUri + ',' + str(pCount) + '\n')
    histogram.close()

def countMementos(pUri, pTimeMap):
    # Re-use from regular expressions from Scott's timeMap.py
    tokenizer = re.compile('(<[^\>]+>|[a-zA-Z]+="[^\"]*"|[,;])\\s*')
    mementoCount=0
    # The timeMap is passed as a string. Read/split each line
    for line in pTimeMap.splitlines():
        # Parse the line into tokens to find what's in the rel= tag
        tokens=tokenizer.findall(line)
        for x in tokens:
            # Find the token using Scott's logic from timeMap.py
            if x[:4] == 'rel=':
                rel=x[5:-1]
                # if "memento" is anywhere in the rel tag, let's count it
                if rel.find('memento') <> -1:
                    mementoCount = mementoCount + 1

    # This will be a number >= 0
    saveToFile(pUri, mementoCount)

#####
# This is main function in this package
#####
```

```

'''
Invoke the ODU Memento Aggregator for each of our 1000 unique
Twitter links. Search the resulting timeMaps, if any, for valid mementos
which are identified as any valid combination of:
    rel="memento"
    rel="first memento"
    rel="last memento"
    rel="memento first"
    rel="memento last"
    rel="first last memento"
'''

def findMementos():
    # Base setting for the aggregator
    uriBase="http://mementoproxy.cs.odu.edu/aggr/timemap/link/"
    notInArchive=0

    # Write the header of our histogram data file
    histogram = codecs.open('C:/Python27/myFiles/histogram.txt','w','utf-8')
    histogram.write("uri" + "," + "mementos" + "\n")
    histogram.close()

    # Open the file containing our 1000 links from Twitter
    tweetFile = open('C:/Python27/myFiles/tweetFile1000.txt')
    lineNo=0
    for line in iter(tweetFile):
        lineNo=lineNo+1 # for checking progress of iteration over URIs
        uri=line.rstrip('\n')
        # Package the request. Append the target uri to complete.
        uri_t=uriBase+uri
        print lineNo, uri_t
        # Search for any timeMaps
        try:
            request=urllib2.Request(uri_t)
            response=urllib2.urlopen(request)
            if response.code==200:
                timeMap=response.read()
                # Parse out the string we're looking for.
                # Record the URI and the count in a file.
                # The file will be used later to create a Histogram in R
                countMementos(uri, timeMap)
            response.close()
        except urllib2.HTTPError,e:
            print "Error", e
            # A 404 response from aggregator indicates "Resource not in archive"
            saveToFile(uri, notInArchive)
            continue
        except urllib2.URLError, e:
            print "Error", e
            continue

```

```
except IOError, e:
    print "Error", e
    continue
# Done processing. Close the file.
tweetFile.close()
print ">>> Session Complete. . ."
```

Appendix C

R Source for Histogram

```
w1 <- read.csv(file="histogram.txt",sep=" ",head=TRUE)
names(w1)
m<-mean(w1$mementos)
std<-sqrt(var(w1$mementos))
hist(w1$mementos,
col="lightgreen",
main="URIs vs Number of Mementos",
xlab="Number of Mementos",
ylab="URIs",
labels=TRUE)
curve(dnorm(x, mean=m, sd=std), add=TRUE)
```


Appendix D

Python Source for estimateAge.py

```
# Utilities from Hany's Carbon Date tool
import codecs
import datetime
from server import index

# Initialize the file for our final results
ageFile = codecs.open('C:/Python27/myFiles/Assignment 2/uri_age.txt','w','utf-8')

# Save the URIs, creation date, age
def saveToFile(pURI, pMementos, pCreationDate, pAge):
    ageFile.write(pURI + "," + pMementos + "," + pCreationDate + "," + pAge + '\n');

def estimateAge():

    # Today's date
    today=datetime.datetime.now()
    # Write the header line
    ageFile.write("URI" + "," + "Mementos" + "," + "Creation Date" + "," + "Age" + '\n');

    # The histogram file contains our 1000 URIs and number of mementos
    # skip the header row: URI, memento
    uriFile = open('C:/Python27/myFiles/Assignment 2/histogram.txt').readlines()[1:]

    fileCounter=0
    for line in uriFile:
        fileCounter=fileCounter+1
        data = line.rstrip('\n')
        # separate comma delimited file into fields
        words={}
        uri=""
        mementos=""
        age=None
        words=data.split(",")
        if len(words) >=2:
            uri=words[0]
```

```

mementos=words[1]
# for URIs with > 0 mementos, find the estimated creation date
if int(mementos) > 0:
    # send URI to Carbon Date tool
    est_creation_date=index(uri)
    if est_creation_date != "":
        delta= today - datetime.datetime.strptime(est_creation_date,
'%Y-%m-%dT%H:%M:%S')
        # convert to days
        age = str(delta.days)
        saveToFile(uri, mementos, est_creation_date, age)
    print fileCounter, mementos, uri,est_creation_date, age
ageFile.close()

```

Appendix E

R Source for Scatterplot

```
# Scatter plot
w1 <- read.csv(file="uri_age.txt",sep="," ,head=TRUE)
attach(w1)
age=w1$Age
mementos=w1$Mementos
plot(age, mementos, col="green",
main="Age vs. Number of Mementos",
xlab="Age in Days",
ylab="Mementos",
pch=19
)
# Add fit lines
abline(lm(mementos~age), col="blue") # regression line ( $y \sim x$ )
```